Burroughs

B 6800 System

REFERENCE MANUAL

PRICED ITEM

Printed in U.S. America

July 1977

5001290

Burroughs

B 6800 System

REFERENCE MANUAL

PRICED ITEM

Printed in U.S. America

July 1977

5001290

Correspondence regarding this document should be addressed directly to Burroughs Corporation, P.O. Box 4040, El Monte, California 91734, Attn: Publications Department, TIO–West.

LIST OF EFFECTIVE PAGES

Issue	Page	Issue	Page
Title	Original	7-1 thru 7-33	Original
ii	Original	7-34	Blank
iii	Original	8-1 thru 8-15	Original
iv	Blank	8-16	Blank
v thru xxii	Original	9-1 thru 9-4	Original
1-1 thru 1-31	Original	10-1 thru 10-8	Original
1-32	Blank	11-1 thru 11-23	Original
2-1 thru 2-39	Original	11-24	Blank
2-40	Blank	11-25 thru 11-34	Original
3-1 thru 3-18	Original	12-1 thru 12-8	Original
4-1 thru 4-55	Original	13-1 thru 13-7	Original
4-56	Blank	13-8	Blank
5-1 thru 5-3	Original	A-1 thru A-5	Original
5-4	Blank	A-6	Blank
5-5 thru 5-67	Original	B-1 thru B-6	Original
5-68	Blank	C-1 thru C-2	Original
5-69 thru 5-76	Original	D-1 thru D-2	Original
6-1 thru 6-8	Original	Index-1 thru Index-14	Original

TABLE OF CONTENTS

Section		Page
	INTRODUCTION	xxi
1	SYSTEM DESCRIPTION	1-1
	General	1-1
	Scope of This Manual	1-1
	B 6800 Hardware System Organization	1-1
	B 6800 System Hardware Module Organization	1-5
	B 6800 Module Interfaces	1-5
	B 6800 Central Processing Unit Cabinet	1-7
	Data Processor Module	1-8
	Multiplexor Module (IO Processor)	1-9
	Memory Control Module	1-10
	B 6800 Maintenance Display Processor	1-12
	Display Control Logic	1-17
	B 6800 Central Power Supply Cabinet	1-17
	B 6800 Peripheral Control Cabinet	1-18
	B 6800 System Peripheral Controls	1-18
	B 6800 Memory Cabinets	1-21
	B 6800 Operators Display Console	1-25
	B 6800 Optional Units	1-30
2	DATA REPRESENTATION	2-1
	General	2-1
	Internal Character Codes	2-3
	Number Bases	2-3
	Number Conversion	2-6
	Decimal to Nondecimal	2-6
	Nondecimal to Decimal	2-6
	Nondecimal to Nondecimal	2-7
	Word Types and Physical Word Layouts	2-10
	Character Type Words	2-10
	Operands	2-10
	Single Precision Operand	2-12
	Double Precision Operand	2-13
	Logical Operands	2-15
	Data Descriptors	2-15
	Step Index Words	2-18
	Software Words	2-19
	Indirect Reference Words	2-20
	Program Control Words	2-23
	Mark Stack Control Words	2-25
	Interrupt Parameter Words	2-26
	P1 Parameter	2-27
	P3 Parameter	2-28
	P2 Parameter	2-28
	Return Control Words	2-32
	Program Words (Code Words)	2-34
	Program Segments and the Segment Descriptor	2-36
	Top of Stack Control Words	2-37

TABLE OF CONTENTS (Cont)

Section		Page
3	STACK AND REVERSE POLISH NOTATION	3-1
	The Stack	3-1
	General	3-1
	Base and Limit of Stack	3-2
	Bi-Directional Data Flow in the Stack	3-2
	Stack Push Down	3-2
	Stack Push Up	3-2
	Double Precision Stack Operation	3-3
	Top-of-Stack Register Conditions	3-3
	Stack Adjustments	3-3
	Data Addressing	3-5
	Data Descriptor	3-5
	Presence Bit	3-5
	Index Bit	3-5
	Invalid Index	3-5
	Valid Index	3-5
	Read-Only Bit	3-6
	Copy Bit	3-6
	Reverse Polish Notation	3-6
	General	3-6
	Simplified Rules for Generation of Polish String	3-6
	Polish String	3-8
	Rules for Evaluating a Polish String	3-8
	Simple Stack Operation	3-8
	Program Structure in Local Memory	3-11
	Local Memory Area Allocation	3-12
	Stack-History and Addressing-Environment Lists	3-12
	Mark Stack Control Word Linkage	3-12
	Stack Deletion	3-12
	Relative-Addressing	3-12
	Base of Address Level Segment	3-13
	Absolute Address Conversion	3-13
	Multiple Variables with Common Address Couples	3-13
	Address Environment Defined	3-15
	Mark Stack Control Word Linkage	3-15
	Stack History Summary	3-17
	Multiple Stacks and Reentrant Code	3-17
	Level Definition	3-17
	Reentrance	3-17
	Job-Splitting	3-17
	Stack Descriptor	3-17
	Stack Vector Descriptor	3-18
	Presence Bit Interrupt	3-18
4	MAJOR REGISTERS AND CONTROL PANELS	4-1
	Display Panels	4-1
	Programmers Display Panel	4-1
	System Control Panel	4-1
	Status Display Panel	4-4

TABLE OF CONTENTS (Cont)

Section		Page
4	MAJOR REGISTERS AND CONTROL PANELS (Cont)	
	Register Display Panel	4-10
	Maintenance Display Panel	4-11
	Maintenance Display Registers	4-11
	Display Panel One, Page Zero Logic Signals	4-27
	Display Panel One, Page One Logic Signals	4-27
	Display Panel One, Page Two Logic Signals	4-28
	Display Panel One, Page Three Logic Signals	4-29
	Panel 1, Page 4 Logic Signals	4-29
	Panel 1, Page 5 Logic Signals	4-31
	Panel 2, Page 0 Logic Signals	4-33
	Display Panel Two, Page One Logic Signals	4-33
	Display Panel Two, Page Two Logic Signals	4-34
	Display Panel Two, Page Three Logic Signals	4-36
	Panel 2, Page 4 Logic Signals	4-37
	Panel 2, Page 5 Logic Signals	4-38
	Display Panel Three, Page Zero Logic Signals	4-39
	Display Panel Three, Page One Logic Signals	4-40
	Display Panel Three, Page Two Logic Signals	4-41
	Panel 3, Page 3 Logic Signals	4-42
	Panel 3, Page 4 Logic Signals	4-43
	Panel 3, Page 5 Logic Signals	4-44
	Panel 4, Page 0 Logic Signals	4-46
	Display Panel Four, Page One Logic Signals	4-47
	Panel 4, Page 2 Logic Signals	4-48
	Panel 4, Page 3 Logic Signals	4-48
	Panel 4, Page 4 Logic Signals	4-49
	Panel 4, Page 5 Logic Signals	4-49
	Maintenance Processor Control Panel and Display	4-49
	Maintenance Processor Programmers Display Keyboard	4-52
	Logic Indicator Lamps	4-55
5	SYSTEM CONCEPT	5-1
	General	5-1
	Data Processor	5-1
	Operator Families	5-1
	Program Controller	5-2
	Look Ahead Logic	5-5
	Integrated Circuit (IC) Memory	5-5
	Address Adder and Residue Test Logic	5-7
	Transfer Controller	5-7
	Stack Registers	5-7
	Internal Data Transfer Section	5-8
	Mask and Steering	5-8
	Mask and Steering Example	5-8
	Stack Controller	5-10

TABLE OF CONTENTS (Cont)

Section		Page
5	SYSTEM CONCEPT (Cont)	
	Arithmetic Controller	5-10
	Exponent and Mantissa Adders	5-10
	Interrupt Controller	5-10
	Interrupt Parameters	5-10
	Syllable Dependent Interrupts	5-14
	Memory Protect Interrupt	5-15
	Invalid Operand Interrupt	5-15
	Divide by Zero Interrupt	5-15
	Exponent Overflow and Underflow Interrupt	5-16
	Invalid Index Interrupt	5-16
	Integer Overflow Interrupt	5-17
	Bottom of Stack Interrupt	5-17
	Presence Bit Interrupt	5-17
	Special Consideration-Presence Bit Interrupts	5-18
	Data-Dependent Presence Bit Interrupt	5-18
	Procedure-Dependent Presence Bit Interrupt	5-18
	Program Restart	5-18
	Sequence Error Interrupt	5-18
	Segmented Array Interrupt	5-19
	Programmed Operator Interrupt	5-20
	Interval Timer Interrupt	5-20
	Stack Overflow Interrupt	5-21
	Confidence Error Interrupt	5-21
	Operator Independent Interrupts	5-21
	External Interrupts	5-21
	I/O Finish, Data Communications, and Status Change Interrupts	5-22
	Alarm Interrupts	5-22
	Loop Interrupts	5-23
	Memory Address Interrupts	5-23
	Scan Bus Parity Interrupts	5-23
	Invalid Address-Local Interrupts	5-24
	Stack Underflow Interrupts	5-24
	Invalid Program Word Interrupts	5-24
	Memory Address Residue Interrupts	5-24
	Read Data Multiple Error Interrupts	5-25
	Invalid Address-Global Interrupts	5-25
	Global Memory Not Ready Interrupts	5-25
	Scan-In Information Error Interrupts	5-25
	Scan-Out Error Interrupts	5-26
	General Control Interrupts	5-26
	Read Data Single Error Interrupts	5-26
	Read Data Retry Interrupts	5-27
	Read Data Check Bit Interrupts	5-27
	Address Retry Interrupts	5-27
	Hardware Interrupts	5-28

TABLE OF CONTENTS (Cont)

Section		Page
5	SYSTEM CONCEPT (Cont)	
	PROM Card Parity Interrupts	5-28
	RAM Card Parity Interrupts	5-28
	Bus Residue Interrupts	5-29
	Adder Residue Interrupts	5-29
	Compare Residue Interrupts	5-29
	String Operators	5-30
	Memory Controller	5-30
	Control State/Normal State	5-31
	Multiplexor Function	5-31
	Data Processor Scan-In Functions to the Multiplexor	5-31
	Data Processor Scan-Out Functions to the Multiplexor	5-33
	Data Processor Scan-Out Functions to External Subsystems	5-35
	Multiplexor Scan-In Functions	5-35
	Interrogate Peripheral Status Multiplexor Function	5-35
	Interrogate IO Path Multiplexor Function	5-35
	Read Time of Day Multiplexor Function	5-37
	Read Interrupt Register Multiplexor Function	5-38
	Interrogate Unit Type Multiplexor Function	5-39
	Interrogate IO Path Address Multiplexor Function	5-39
	Read Processor Time Counter Multiplexor Function	5-42
	Read Scratch Pad Word Multiplexor Function	5-43
	Interrogate IO Path Address Override Multiplexor Function	5-45
	Read Interrupt Literal Multiplexor Function	5-45
	Read Interrupt Mask Multiplexor Function	5-45
	Multiplexor Scan-Out Functions	5-45
	Set Time of Day Multiplexor Function	5-46
	Set Interrupt Mask Multiplexor Function	5-46
	Set Pseudo Busy Multiplexor Function	5-46
	Software Aspects of IO Operations in the B 6800 System	5-46
	Ready Status	5-50
	Status Change	5-52
	Input Output Operations	5-53
	IO Device Numbering System	5-53
	Initiate Input Output Operation	5-54
	IIOWD Format	5-56
	IOAD Format	5-56
	IOCW Format	5-56
	Scratch Pad Memory	5-58
	Data Buffer Logic	5-60
	OP Code and Variant Character Generator	5-61
	Status Vector Control Circuits	5-62
	Memory Organization	5-63
	System Memory Interface	5-64
	Channel A Memory Requestor	5-64
	Memory Error Detection and Correction	5-69

TABLE OF CONTENTS (Cont)

Section		Page
5	SYSTEM CONCEPT (Cont)	
	Memory Retry	5-71
	Global Memory	5-71
	Scan Bus Operations	5-71
	Channel B Memory Requestor	5-72
	Memory Storage Unit Port Interface	5-72
	Local Memory Port Interface Control Logic	5-74
	Scan Bus Port Interface Control Logic	5-75
	Memory Tester Logic	5-76
6	PROGRAM OPERATORS	6-1
	General	6-1
	Syllable Addressing and Syllable Identification	6-1
	Syllable Format and Addressing	6-1
	P and T Registers	6-1
	Operation Types	6-2
	Name Call	6-2
	Value Call	6-4
	Operators	6-7
7	PRIMARY MODE OPERATORS	7-1
	General	7-1
	Arithmetic Operators	7-1
	Add (ADD) 80	7-1
	Subtract (SUBT) 81	7-2
	Multiply (MULT) 82	7-2
	Extended Multiply (MULX) 8F	7-2
	Divide (DIVD) 83	7-2
	Integer Divide (IDIV) 84	7-3
	Remainder Divide (RDIV) 85	7-3
	Integerize, Truncated (NTIA) 86	7-3
	Integerize, Rounded (NTGR) 87	7-4
	Type-Transfer Operators	7-4
	Set to Single-Precision, Truncated (SNGT) CC	7-4
	Set to Single-Precision, Rounded (SNGL) CD	7-4
	Set to Double-Precision (XTND) CE	7-5
	Logical Operators	7-5
	Logical AND (LAND) 90	7-5
	Logical OR (LOR) 91	7-5
	Logical Negate (LNOT) 92	7-5
	Logical Equivalence (LEQV) 93	7-5
	Relational Operators	7-5
	Greater Than (GRTR) 8A	7-6
	Greater Than or Equal (GREQ) 89	7-7
	Equal (EQUL) 8C	7-7
	Less Than or Equal (LSEQ) 8B	7-7
	Less Than (LESS) 88	7-7
	Not Equal (NEQL) 8D	7-7

TABLE OF CONTENTS (Cont)

Section		Page
7	PRIMARY MODE OPERATORS (Cont)	
	Branch Operators	7-7
	Branch False (BRFL) AO	7-7
	Branch True (BRTR) A1	7-8
	Branch Unconditional (BRUN) A2	7-8
	Dynamic Branch False (DBFL) A8	7-8
	Dynamic Branch True (DBTR) A9	7-8
	Dynamic Branch Unconditional (DBUN) AA	7-8
	Step and Branch (STBR) A4	7-8
	Universal Operators	7-9
	No Operation (NOOP) FE	7-9
	Conditional Halt (HALT) DF	7-9
	Invalid Operator (NVLD) FF	7-9
	Store Operators	7-9
	Store Destructive (STOD) B8	7-9
	Store Non-Destructive (STON) B9	7-9
	Overwrite Destructive (OVRD) BA	7-9
	Overwrite Non-Destructive (OVRN) BB	7-10
	Stack Operators	7-10
	Exchange (EXCH) B6	7-10
	Delete Top of Stack (DLET) B5	7-10
	Duplicate Top of Stack (DUPL) B7	7-10
	Push Down Stack Registers (PUSH) B4	7-10
	Literal Call Operators	7-10
	Lit Call Zero (ZERO) B0	7-10
	Lit Call One (ONE) B1	7-10
	Lit Call 8 Bits (LT8) B2	7-10
	Lit Call 16 Bits (LT16) B3	7-10
	Lit Call 48 Bits (LT48) BE	7-10
	Make Program Control Word (MPCW) BF	7-11
	Index and Load Operators	7-11
	Index (INDX) A6	7-11
	Index and Load Name (NXLN) A5	7-11
	Index and Load Value (NXLV) AD	7-11
	Load (LOAD) BD	7-11
	Scale Operators	7-12
	Scale Left (SCLF) CO	7-12
	Dynamic Scale Left (DSLF) C1	7-12
	Scale Right Save (SCRS) C4	7-12
	Dynamic Scale Right Save (DSRS) C5	7-12
	Scale Right Truncate (SCRT) C2	7-12
	Dynamic Scale Right Truncate (DSRT) C3	7-13
	Scale Right Final (SCRF) C6	7-13
	Dynamic Scale Right Final (DSRF) C7	7-13
	Scale Right Rounded (SCRR) C8	7-13
	Dynamic Scale Right Round (DSRR) C9	7-13

TABLE OF CONTENTS (Cont)

Section		Page
7	PRIMARY MODE OPERATORS (Cont)	
	Bit Operators	7-13
	Bit Set (BSET) 96	7-13
	Dynamic Bit Set (DBST) 97	7-13
	Bit Reset (BRST) 9E	7-14
	Dynamic Bit Reset (DBRS) 9F	7-14
	Change Sign Bit (CHSN) 8E	7-14
	Transfer Operators	7-14
	Field Transfer (FLTR) 98	7-14
	Dynamic Field Transfer (DFTR) 99	7-14
	Field Isolate (ISOL) 9A	7-15
	Dynamic Field Isolate (DISO) 9B	7-15
	Field Insert (INSR) 9C	7-15
	Dynamic Field Insert (DINS) 9D	7-15
	String Transfer Operators	7-16
	Transfer Words, Destructive (TWSD) D3	7-16
	Transfer Words, Update (TWSU) DB	7-16
	Transfer Words, Overwrite Destructive (TWOD) D4	7-16
	Transfer Words, Overwrite Update (TWOU) DC	7-16
	Transfer While Greater, Destructive (TGTD) E2	7-16
	Transfer While Greater Update (TGTU) EA	7-17
	Transfer While Greater or Equal, Destructive (TGED) E1	7-17
	Transfer While Greater or Equal, Update (TGEU) E9	7-17
	Transfer While Equal, Destructive (TEQD) E4	7-17
	Transfer While Equal, Update (TEQU) EC	7-17
	Transfer While Less or Equal, Destructive (TLED) E3	7-17
	Transfer While Less or Equal, Update (TLEU) EB	7-17
	Transfer While Less, Destructive (TLSD) EO	7-17
	Transfer While Less, Update (TLSU) E8	7-17
	Transfer While Not Equal, Destructive (TNED) E5	7-17
	Transfer While Not Equal, Update (TNEU) ED	7-18
	Transfer Unconditional, Destructive (TUND) E6	7-18
	Transfer Unconditional, Update (TUNU) EE	7-18
	String Isolate (SISO) D5	7-18
	Compare Operators	7-18
	Compare Characters Greater, Destructive (CGTD) F2	7-18
	Compare Characters Greater, Update (CGTU) FA	7-19
	Compare Characters Greater or Equal, Destructive (CGED) F1	7-19
	Compare Characters Greater or Equal, Update (CGEU) F9	7-19
	Compare Characters Equal, Destructive (CEQD) F4	7-20
	Compare Characters Equal, Update (CEQU) FC	7-20
	Compare Characters Less or Equal, Destructive (CLED) F3	7-20
	Compare Characters Less or Equal, Update (CLEU) FB	7-20
	Compare Characters Less, Destructive (CLSD) F0	7-20
	Compare Characters Less, Update (CLSU) F8	7-20
	Compare Characters Not Equal, Destructive (CNED) F5	7-20
	Compare Characters Not Equal, Update (CNEU) FD	7-20

TABLE OF CONTENTS (Cont)

Section		Page
7	PRIMARY MODE OPERATORS (Cont)	
	Edit Operators	7-20
	Table Enter Edit, Destructive (TEED) D0	7-20
	Table Enter Edit, Update (TEEU) D8	7-21
	Execute Single Micro, Destructive (EXSD) D2	7-21
	Execute Single Micro, Update (EXSU) DA	7-21
	Execute Single Micro, Single Pointer Update (EXPU) DD	7-21
	Pack Operators	7-21
	Pack, Destructive (PACD) D1	7-21
	Pack, Update (PACU) D9	7-22
	Input Convert Operators	7-22
	Input Convert, Destructive (ICVD) CA	7-22
	Input Convert, Update (ICVU) CB	7-23
	Read True False Flip-Flop (RTFF) DE	7-23
	Set External Sign (SXSN) D6	7-23
	Read and Clear Overflow Flip-Flop (ROFF) D7	7-23
	Subroutine Operators	7-23
	Value Call (VALC) 00 ⇒3F	7-23
	Name Call (NAMC) 40 ⇒7F	7-26
	Exit Operator (EXIT) A3	7-26
	Return Operator (RETN) A7	7-26
	Enter Operator (ENTR) AB	7-26
	Evaluate (EVAL) AC	7-26
	Mark Stack Operator (MKST) AE	7-26
	Stuff Environment (STFF) AF	7-31
	Insert Mark Stack Operator (IMKS) CF	7-31
	Enter Vector Mode Operators	7-31
	Vector Mode Enter Multiple (VMEM) E7	7-31
	Vector Mode Enter Single (VMES) EF	7-31
8	VARIANT MODE OPERATION AND OPERATORS	8-1
	General	8-1
	Escape to 16-Bit Instruction (VARI) 95	8-1
	Variant Mode Operators	8-1
	Set Two Singles to Double (JOIN) 9542	8-1
	Set Double to Two Singles (SPLT) 9543	8-1
	Idle Until Interrupt (IDLE) 9544	8-1
	Set Interval Timer (SINT) 9545 (Control State Operator)	8-1
	Enable External Interrupts (EEXI) 9546	8-2
	Disable External Interrupts (DEXI) 9547	8-2
	Scan Operators	8-2
	Scan In (SCNI) 954A	8-2
	Read Time-Of-Day Clock	8-2
	Read Interrupt Mask	8-2
	Read Interrupt Register	8-2
	Read Interrupt Literal	8-3
	Interrogate Peripheral Status	8-3

TABLE OF CONTENTS (Cont)

Section		Page
8	VARIANT MODE OPERATION AND OPERATORS (Cont)	
	Interrogate Peripheral Unit Type	8-3
	Interrogate IO Path	8-3
	Interrogate IO Path Address	8-3
	Interrogate IO Path Address Override	8-3
	Read Scratch Pad Word	8-3
	Read Processor Time Counter	8-3
	Scan Out (SCNO) 954B	8-3
	Set Time of Day	8-4
	Set Interrupt Mask	8-4
	Set Pseudo Busy	8-4
	Initiate IO Device (Control State Only)	8-4
	Initiate IO Device Path Address	8-4
	Initiate IO Device Path Address Override	8-5
	Read Processor Identification (WHOI) 954E	8-5
	Occurs Index (OCRX) 9585	8-5
	Integerize, Rounded, Double-Precision (NTGD) 9587	8-5
	Leading One Test (LOG2) 958B	8-5
	Normalize (NORM) 958E	8-6
	Move to Stack (MVST) 95AF	8-6
	Read Compare Flip-Flop (RCMP) 95B3	8-7
	Set Tag Field (STAG) 95B4	8-7
	Read Tag Field (RTAG) 95B5	8-8
	Rotate Stack Up (RSUP) 95B6	8-8
	Rotate Stack Down (RSDN) 95B7	8-8
	Read Processor Register (RPRR) 95B8	8-8
	Set Processor Register (SPRR) 95B9	8-9
	Read With Lock (RDLK) 95BA	8-10
	Count Binary Ones (CBON) 95BB	8-10
	Load Transparent (LODT) 95BC	8-10
	Linked List Lookup (LLLU) 95BD	8-10
	Masked Search for Equal (SRCH) 95BE	8-11
	Unpack Absolute, Destructive (UABD) 95D1	8-11
	Unpack Absolute, Update (UABU) 95D9	8-11
	Unpack Signed, Destructive (USND) 95DO	8-11
	Unpack Signed, Update (USNU) 95D8	8-11
	Transfer While True, Destructive (TWTD) 95D3	8-12
	Transfer While True, Update (TWTU) 95DB	8-12
	Transfer While False, Destructive (TWFD) 95D2	8-12
	Transfer While False, Update (TWFU) 95DA	8-12
	Translate (TRNS) 95D7	8-12
	Scan While Greater, Destructive (SGTD) 95F2	8-13
	Scan While Greater, Update (SGTU) 95FA	8-13
	Scan While Greater or Equal, Destructive (SGED) 95F1	8-13
	Scan While Greater or Equal, Update (SGEU) 95F9	8-13
	Scan While Equal, Destructive (SEQD) 95F4	8-14
	Scan While Equal, Update (SEQU) 95FC	8-14

TABLE OF CONTENTS (Cont)

Section		Page
8	VARIANT MODE OPERATION AND OPERATORS (Cont)	
	Scan While Less or Equal, Destructive (SLED) 95F3	8-14
	Scan While Less or Equal, Update (SLEU) 95FB	8-14
	Scan While Less, Destructive (SLSD) 95FO	8-14
	Scan While Less, Update (SLSU) 95F8	8-14
	Scan While Not Equal, Destructive (SNED) 95F5	8-14
	Scan While Not Equal, Update (SNEU) 95FD	8-15
	Scan While True, Destructive (SWTD) 95D5	8-15
	Scan While True, Update (SWTU) 95DD	8-15
	Scan While False, Destructive (SWFD) 95D4	8-15
	Scan While False, Update (SWFU) 95DC	8-15
9	EDIT MODE OPERATION AND OPERATORS	9-1
	General	9-1
	Edit Mode Operators	9-1
	Move Characters (MCHR) D7	9-1
	Move Numeric Unconditional (MVNU) D6	9-1
	Move With Insert (MINS) DO	9-1
	Move With Float (MFLT) D1	9-2
	Skip Forward Source Characters (SFSC) D2	9-2
	Skip Reverse Source Characters (SRSC) D3	9-2
	Skip Forward Destination Characters (SFDC) DA	9-2
	Skip Reverse Destination Characters (SRDC) DB	9-3
	Reset Float (RSTF) D4	9-3
	End Float (ENDF) D5	9-3
	Insert Unconditional (INSU) DC	9-3
	Insert Conditional (INSC) DD	9-3
	Insert Display Sign (INSG) D9	9-3
	Insert Overpunch (INOP) D8	9-3
	End Edit (ENDE) DE	9-4
10	VECTOR MODE OPERATORS	10-1
	General	10-1
	Limitations of Vector Mode	10-1
	Hardware Functions	10-1
	Primary Mode Enter Vector Mode Operators	10-2
	Enter Vector Mode Operation	10-2
	Vector Stack Operators	10-4
	Vector Mode Operator Codes	10-5
	Vector Operators	10-6
	Vector Branch and Vector Exit Operators	10-8
11	PERIPHERAL DEVICES AND CONTROLS	11-1
	General	11-1
	Typical Input Output Device System Operation	11-1
	Interrupt Stack Parameters	11-3
	P1 Parameter	11-3

TABLE OF CONTENTS (Cont)

Section		Page
11	PERIPHERAL DEVICES AND CONTROLS (Cont)	
	P3 Parameter	11-4
	P2 Parameter	11-5
	Peripheral Controls	11-7
	Peripheral Control Bus	11-7
	Input Output Device Commands and Result Descriptors	11-10
12	DATA COMMUNICATIONS SUBSYSTEM	12-1
	General	12-1
	Data Communications Processor	12-3
	Terminal Devices	12-4
	Basic Control	12-4
	Broadband Control	12-4
	Data Comm to Disk Control	12-4
	Store-to-Store Control	12-5
	Adapter Cluster III	12-5
	Data Communications Subsystem Scan Bus Interface	12-5
	Data Communications Subsystem Memory Interface	12-7
13	B 6800 BUS INTERFACE CONTROL (READER/SORTER SUBSYSTEM)	13-1
	The BIC Module	13-1
	BUS Interface Control Scan BUS Interface	13-1
	Scan Out Functions	13-1
	Scan In Functions	13-4
	BUS Interface Control Memory Interface	13-6
APPENDIX A.	OPERATORS, ALPHABETICAL LIST	A-1
APPENDIX B.	OPERATORS, NUMERICAL LIST	B-1
APPENDIX C.	DATA REPRESENTATION	C-1
APPENDIX D.	B 6700 EBCDIC/HEX CARD CODE	D-1
INDEX		Index-1

LIST OF ILLUSTRATIONS

Figure		Page
1-1	B 6800 Cabinets Sizes	1-2
1-2	B 6800 System (Minimum Cabinets) Layout	1-3
1-3	B 6800 System (Maximum Cabinets) Layout	1-4
1-4	B 6800 System Module Block Diagram	1-6
1-5	B 6800 System Busing	1-7
1-6	Maintenance Display Processor Cabinet	1-13
1-7	Central Power Cabinet	1-19
1-8	B 6800 Power Subsystem Distribution Diagram	1-20
1-9	Peripheral Control Cabinet	1-21
1-10	Peripheral Control Interface	1-22

LIST OF ILLUSTRATIONS (Cont)

Figure		Page
1-11	B 6800 Memory Cabinet	1-23
1-12	Memory Control Interface Bus	1-24
1-13	Operators Display Console	1-27
1-14	Operators System Control Panel	1-28
1-15	Operators Keyboard	1-29
2-1	B 6800 Word Structure	2-1
2-2	Character and Digit Formats	2-4
2-3	B 6800 Word Formats	2-5
2-4	EBCDIC Character Word Format	2-11
2-5	BCL Character Word Format	2-11
2-6	Hexadecimal Character Word Format	2-11
2-7	Single Precision Operand Format	2-12
2-8	Double Precision Operand Format	2-14
2-9	Data Descriptor Format	2-16
2-10	Step Index Word Format	2-18
2-11	Software Control (LINKA) Word	2-19
2-12	Software Control (MASK) Word	2-20
2-13	Indirect Reference Word	2-21
2-14	Program Control Word	2-24
2-15	Mark Stack Control Word	2-25
2-16	B 6800 Interrupt Stack Organization	2-27
2-17	P3 Parameter Configurations	2-33
2-18	P3 Parameter Contents for IO Finished Interrupt	2-34
2-19	P2 Parameter Status Change Format	2-34
2-20	P2 Parameter Result Descriptor Foramt	2-35
2-21	P2 Parameter Scratch Pad Parity Format	2-35
2-22	Return Control Word	2-36
2-23	Segment Descriptor Word	2-38
2-24	TOSCW Word Layout	2-38
2-25	Program Word Format	2-39
3-1	Top-of-Stack and Stack Bounds Registers	3-1
3-2	Reverse Polish Notation Flow Chart	3-7
3-3	Stack Operation	3-10
3-4	Object Program in Memory	3-13
3-5	Stack History and Addressing Environment List	3-14
3-6	Stack Cut-Back Operation on Procedure Exit	3-14
3-7	ALGOL Program With Lexicographical Structure Indicated	3-15
3-8	D Registers Indicating Current Addressing Environment	3-16
3-9	Addressing Environment Tree of ALGOL Program	3-16
3-10	Multiple Linked Stacks	3-18
4-1	System Control and Display Registers	4-2
4-2	Programmers Display Panel	4-3
4-3	System Status Indicator Panel	4-5
4-4	Maintenance Display (Facing) Panel	4-12
4-5	Maintenance Display Panel (Cover)	4-13
4-6	Maintenance Display Processor Control Panel	4-50
4-7	Keyboard Pushbuttons and Indicators	4-53

LIST OF ILLUSTRATIONS (Cont)

Figure		Page
5-1	B 6800 CPU Organization	5-2
5-2	B 6800 CPU Block Diagram	5-3
5-3	Internal Data Transfer Section	5-9
5-4	Mask and Steering	5-11
5-5	Hardware Stack Adjustment	5-12
5-6	Arithmetic Control	5-13
5-7	Presence Bit Interrupt	5-19
5-8	B 6800 IO Function Block Diagram	5-32
5-9	B 6800 Scan-In Function Word	5-33
5-10	B 6800 Scan-Out Function Word	5-34
5-11	Interrogate Peripheral Status	5-36
5-12	Interrogate IO Path	5-37
5-13	Read Time of Day	5-38
5-14	Read Interrupt Register	5-40
5-15	Interrogate Unit Type	5-41
5-16	Interrogate IO Path Address	5-42
5-17	Read Processor Timer	5-43
5-18	Read Scratch Pad Word	5-44
5-19	Read Interrupt Literal	5-47
5-20	Read Interrupt Mask.	5-48
5-21	Set Time of Day	5-49
5-22	Set Interrupt Mask	5-50
5-23	Set Pseudo Busy	5-51
5-24	Multiplexer Initiate IO Words Format	5-55
5-25	Multiplexor Scratch Pad Memory	5-59
5-26	Memory Address Decoding	5-63
5-27	Memory Control Block Diagram	5-65
5-28	Data Processor to Memory Control Exchange Transfer Paths	5-66
5-29	Memory Exchange Channel A Functional Block Diagram	5-67
5-30	Channel B Functional Block Diagram	5-70
6-1	Program Word	6-2
6-2	Program Word, Syllable Addressing	6-3
6-3	Primary Mode Operator Syllable Decode Table	6-4
6-4	Name Call Operator Function	6-5
6-5	Value Call Operator Function	6-6
7-1	Flow of Value Call Operator	7-24
7-2	Value Call (Descriptor) Operator	7-25
7-3	Flow of Exit Operator	7-27
7-4	Flow of Return Operator	7-28
7-5	Flow of Enter Operator	7-29
7-6	Flow of Evaluate Operator	7-30
7-7	Flow of Stuff Environment Operator	7-32
8-1	WHOI Operator Returned Word	8-6
8-2	Index Control Word (ICW) and Index Word	8-7
8-3	Top-of-Stack Control Word (TSCW)	8-8
8-4	Rotate Stack Operations	8-9

LIST OF ILLUSTRATIONS (Cont)

Figure		Page
11-1	Input-Output Operation Cycles	11-2
11-2	Finished Interrupt Stack Parameters	11-4
11-3	B 6800 Peripheral Controls Organization	11-8
11-4	Supervisory Display Control II IOCW Format	11-11
11-5	Supervisory Display Control II Result Descriptor Format	11-12
11-6	Single Line Control IOCW Format	11-13
11-7	Single Line Control Result Descriptor Format	11-14
11-8	Card Punch IOCW Format	11-15
11-9	Card Punch Result Descriptor Format	11-16
11-10	Card Reader IOCW Format	11-17
11-11	Card Reader Result Descriptor Format	11-18
11-12	Line Printer IOCW Format	11-19
11-13	Line Printer Result Descriptor Format	11-20
11-14	Train Printer IOCW Format	11-21
11-15	Train Printer Result Descriptor Format	11-22
11-16	Magnetic Tape IOCW Format	11-23
11-17	Magnetic Tape Result Descriptor Format	11-25
11-18	Head Per Track Disk File IOCW Format	11-26
11-19	Head Per Track Disk File Result Descriptor Format	11-27
11-20	Flexible Disk IOCW Format	11-28
11-21	Flexible Disk Result Descriptor Format	11-29
11-22	Disk Pack IOCW Format	11-30
11-23	Disk Pack IOCW Unit Control Format	11-31
11-24	Disk Pack Result Descriptor Format	11-32
11-25	5N Disk File IOCW Format	11-33
11-26	5N Disk File Result Descriptor Format	11-34
12-1	B 6800 Data Communications Subsystem Block Diagram	12-2
13-1	BUS Interface Subsystem Modules	13-2
13-2	BIC Scan-Out Function Word	13-3
13-3	Set Bounds Registers Data Word	13-4
13-4	Clear-Load Data Word	13-5
13-5	BIC Scan-In Function Word	13-5
13-6	Read BIC Status Response	13-6

LIST OF TABLES

Table		Page
2-1	Decimal Place Values of Digits in Various Number Bases	2-8
2-2	Address Couple Value Fields	2-23
2-3	P1 Parameter Words	2-29
2-4	Interrupt Procedure Stack Parameter Contents	2-31
3-1	Evaluation of Polish String A 7 B C + x :=	3-9
3-2	Description of Stack Operation	3-11
4-1	B 6800 Maintenance Display Panel Register Selection Positions	4-14
4-2	Maintenance Display Register Logic Signals for Register 1, Pages 0 (Top), and 1 (Bottom)	4-15
4-3	Maintenance Display Register Logic Signals for Register 1, Pages 2 (Top), and 3 (Bottom)	4-16

LIST OF TABLES (Cont)

Table		Page
4-4	Maintenance Display Register Logic Signals for Register 1, Pages 4 (Top), and 5 (Bottom)	4-17
4-5	Maintenance Display Register Logic Signals for Register 2, Pages 0 (Top), and 1 (Bottom)	4-18
4-6	Maintenance Display Register Logic Signals for Register 2, Pages 2 (Top), and 3 (Bottom)	4-19
4-7	Maintenance Display Register Logic Signals for Register 2, Pages 4 (Top), and 5 (Bottom)	4-20
4-8	Maintenance Display Register Logic Signals for Register 3, Pages 0 (Top), and 1 (Bottom)	4-21
4-9	Maintenance Display Register Logic Signals for Register 3, Pages 2 (Top), and 3 (Bottom)	4-22
4-10	Maintenance Display Register Logic Signals for Register 3, Pages 4 (Top), and 5 (Bottom)	4-23
4-11	Maintenance Display Register Logic Signals for Register 4, Pages 0 (Top), and 1 (Bottom)	4-24
4-12	Maintenance Display Register Logic Signals for Register 4, Pages 2 (Top), and 3 (Bottom)	4-25
4-13	Maintenance Display Register Logic Signals for Register 4, Pages 4 (Top), and 5 (Bottom)	4-26
7-1	Relational Operator Indications	7-6
7-2	Compare Type Operator Results	7-19

INTRODUCTION

The B 6800 is a large scale, modular, high-speed data processing system. The B 6800 system consists of from 7, to 9 cabinets, which are joined together to form a single mainframe organization. The leading features of the B 6800 system are:

a. Monolithic circuits

b. System memory expandable in increments of 65,536 words

c. Memory cycle times of 1.2 microseconds

d. Automatic memory error detection and correction

e. Peripheral units expandable to 256 units

f. 20 peripheral channels for IO operation

g. Data communications processing through the use of optional standard equipment cabinets

h. Reader/sorter subsystem capability through the use of optional standard equipment cabinets

i. Centralized power supplies, with solid metalic bus-bar organization

A unique design concept, developed from years of experience with the B 5500 and B 6700 Information Processing Systems has resulted in the B 6800 hardware and software design. The hardware and the software were simultaneously designed in a parallel and coordinated process, such that these two parts of the System act to augment, and complement each other. This method assures that the hardware will contains the logic circuits necessary to implement the concepts of the software, and also that the software constructs will utilize the hardware circuits in an efficient manner.

The B 6800 system is designed to use the hardware stack concept which was successful in both the B 5500 and the B 6700 systems. However, the hardware used in the B 6800 system also represents recent state-of-the-arts improvements in data processing circuit components. This blending of proven design with modern material results in a more efficient, and powerful data processing system.

The B 6800 system utilizes the same dynamic storage allocation concept that was utilized in the B 6700 Information Processing System. This concept utilizes a descriptor method of segmentation which allows variable length segments of data to be used. This method is more efficient than "fixed-size" paging concepts.

A new "look ahead" logical circuit is used in the B 6800 system data processor to fetch program code words from memory. This circuit virtually eliminates the need to halt the flow of a user program to obtain the next word of program code. This new circuit represents an improvement in the way that user programs are executed, and results in more efficient operation of the hardware system resources.

The use of new, and more compact logical circuit components has allowed the B 6800 system to have a greater degree of packaging density than was available in the B 6700 system design. This greater packaging density of electronic components has resulted in the use of a central processor unit in the B 6800 system. The central processing unit, which is a single system cabinet, takes the place of 4 cabinets which were required in the B 6700 system. This improvement in packaging saves space, and reduces operating costs in the B 6800 system, without requiring a loss in data processing capability.

The B 6800 system utilizes a new centralized power supply cabinet. This new centralized power supply eliminates the need to mount an inverter module in each mainframe cabinet. It collects most power supplies for the B 6800 system within a single cabinet, and thus makes the power supply subsystem easier to maintain.

The B 6800 system cabinets have a fixed relative location within the mainframe cabinet layout. This fixed location scheme reduces the complexity of the system installation process, reduces interface cabling requirements, and allows more efficiency in site planning.

The B 6800 system contains the capability to be interfaced with, and to operate from Global (TM) memory applications.

SECTION 1

SYSTEM DESCRIPTION

GENERAL

This manual explains how the B 6800 Information Processing System achieves flexibility and efficiency through a comprehensive system approach to problem solving without considering the areas of computer logic or circuit design. The program-independent modular system design efficiently uses available units to process programs and also permits system configuration changes without the need to reprogram or recompile. This approach also offers the user the advantages of simplified programming, ease of operation, and a complete freedom of system expansion. The B 6800 is a compiler oriented system, designed to accept the high level problem-solving language compilers such as ALGOL, COBOL, FORTRAN, and PL/I.

The B 6800 system software operates under the control of a Master Control Program (MCP), which automatically handles memory assignments, program segmentation, and subroutine linkages. The use of the MCP eliminates many arduous programming tasks which are likely to produce errors. The compilers are operated under the control of the MCP, as are the object-programs that result from the use of the compilers. The programs are debugged and corrected in the source language.

SCOPE OF THIS MANUAL

This manual will describe the major hardware characteristics of the B 6800 system. Because of the strong interdependence of the system software and system hardware this manual will discuss both parts of the system design at times. Wherever a choice is available, to discuss a part of the system in terms of either the hardware or the software, the hardware discussion will be used. Both discussions will be used where insight can be developed by the use of this method.

B 6800 HARDWARE SYSTEM ORGANIZATION

The B 6800 system consists of a series of cabinet types, which are arranged in a specific order. The ordering of the cabinets within the system is classed as a minimum configuration B 6800 system, or as an expanded configuration B 6800 system. The arrangement of the cabinets within a B 6800 system is such that a minimum configuration B 6800 may be upgraded to an expanded configuration by adding additional cabinets. No reorganization of the cabinets within a B 6800 system is required to upgrade an existing system to the expanded configuration class.

There are three standard size cabinets used in the organization of a B 6800 system. Figure 1-1 shows these three cabinet sizes, and indicates the various dimensions of the cabinets. The cabinets in a B 6800 system are joined together to form a continuous mainframe appearance. This appearance is enhanced by the use of outer panels that give the illusion of a single mainframe structure.

Figure 1-2 shows the cabinets in a minimum configuration B 6800 system. The layout of the various cabinets within the B 6800 system mainframe structure is invariable, and thus the minimum area for the mainframe of a B 6800 system is also invariable. The minimum area required for a B 6800 system mainframe is 21 feet three inches wide, by 25 feet in length. This area will allow for the expansion of a minimum configuration B 6800 system into a fully expanded configuration B 6800 system. The area given in this paragraph does not include the area required to contain the peripheral devices that are connected to the B 6800 system.

Figure 1-3 shows the maximum configuration B 6800 system. This B 6800 system configuration contains two more cabinets than the minimum system layout. The additional cabinets are so located that they may be added to the system without causing reorganization of the cabinets in the original system layout.

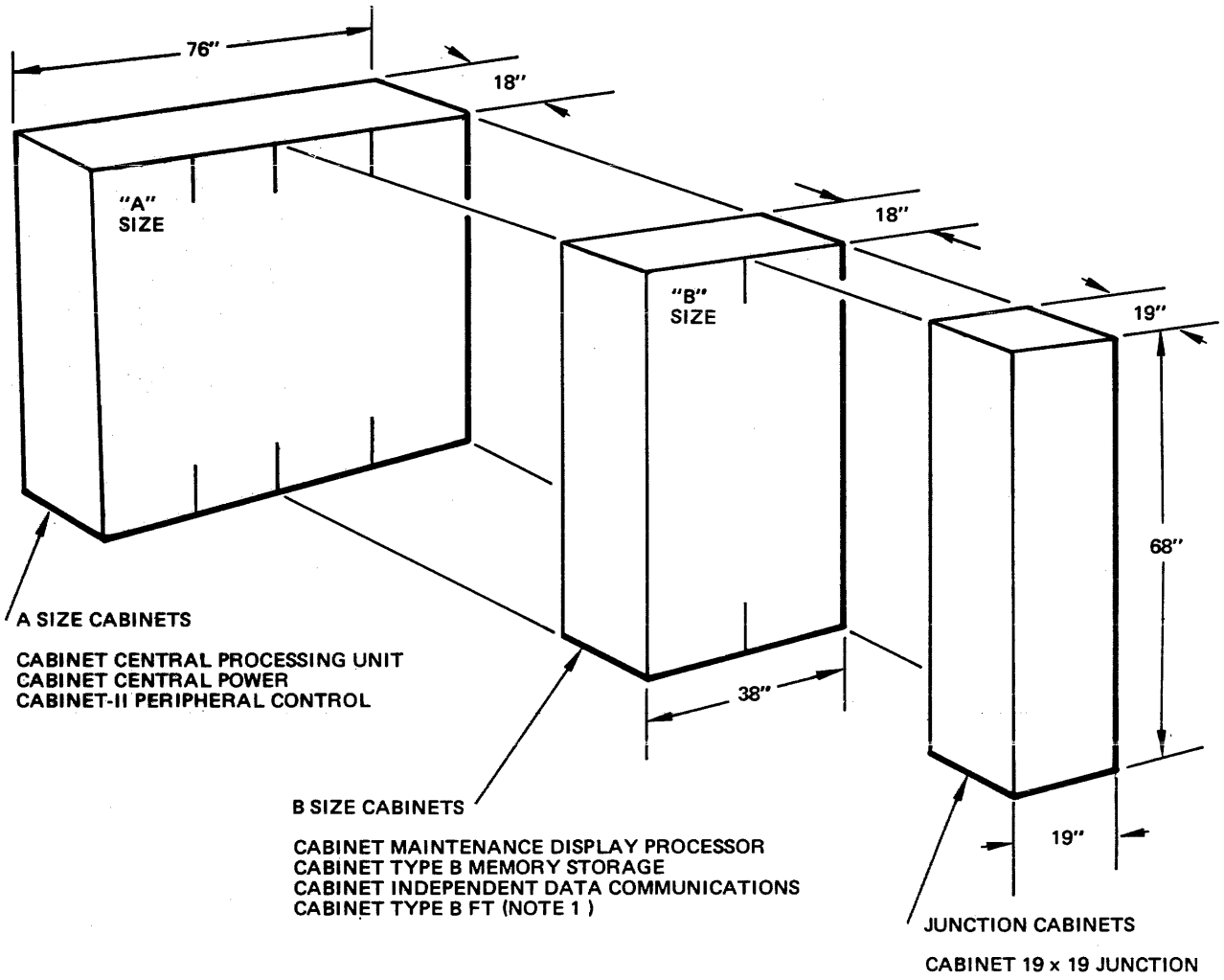


Figure 1-1. B 6800 Cabinets Sizes

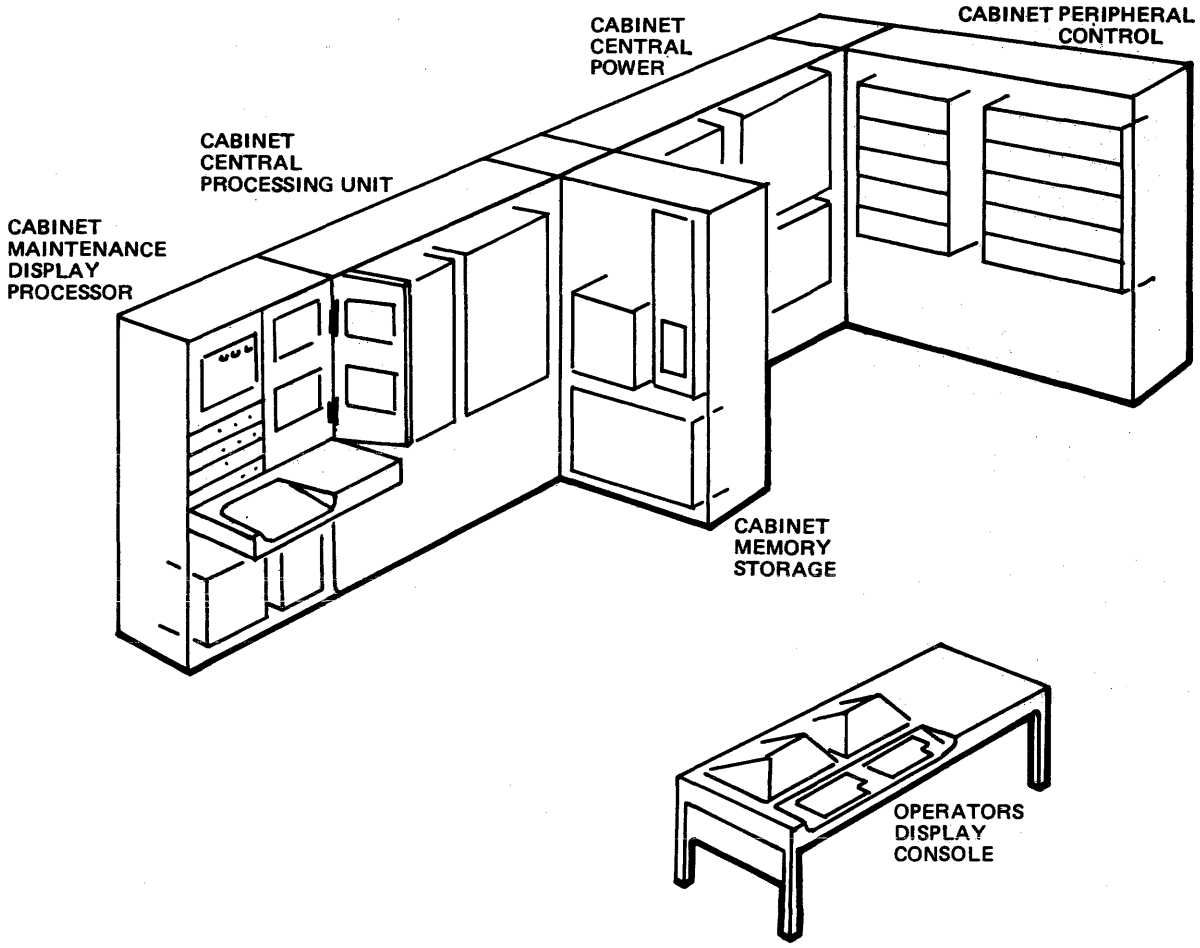


Figure 1-2. B 6800 System (Minimum Cabinets) Layout

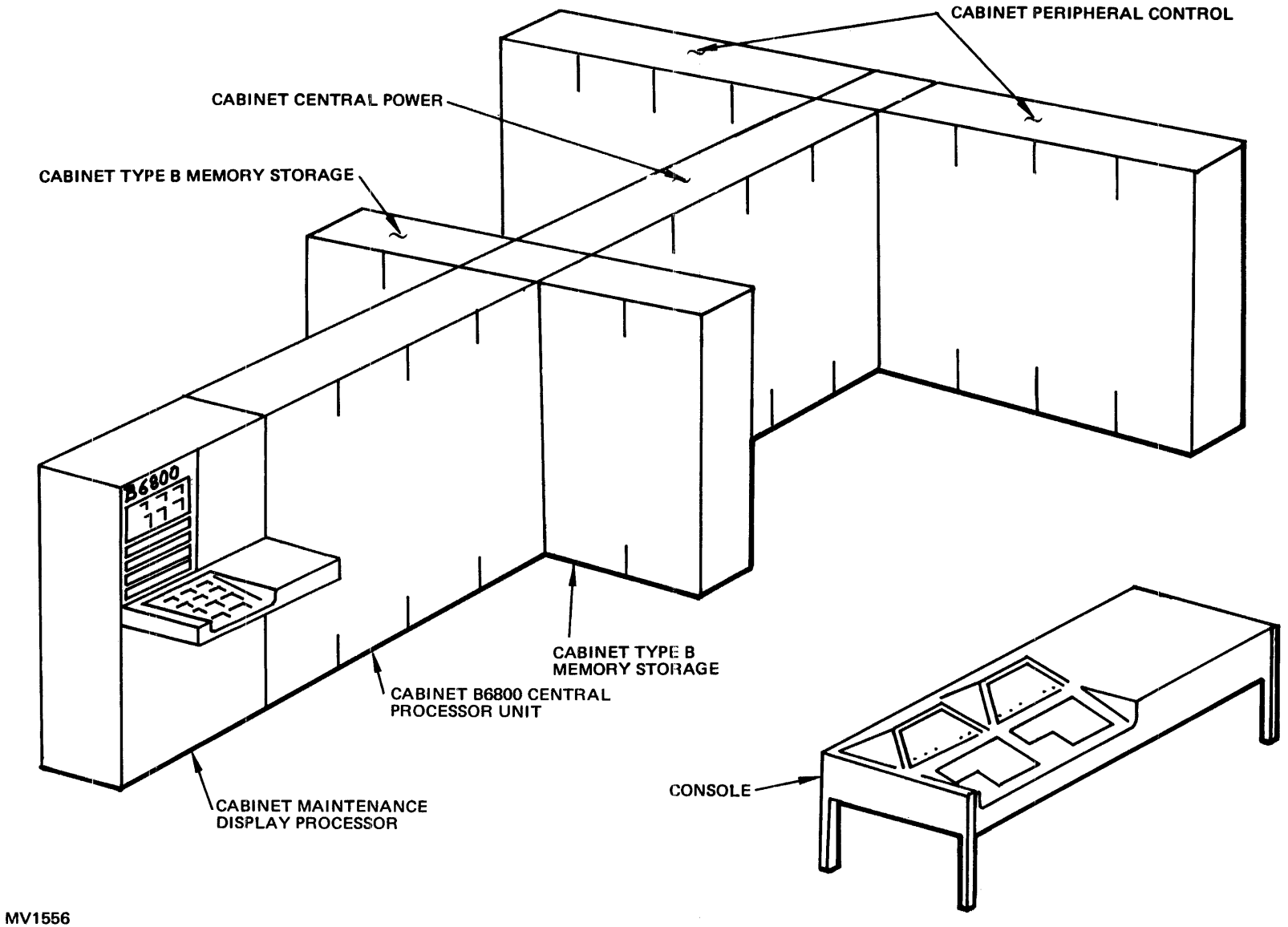


Figure 1-3. B 6800 System (Maximum Cabinets) Layout

B 6800 SYSTEM HARDWARE MODULE ORGANIZATION

The following paragraphs discuss the B 6800 system modules that are located within the system cabinets. A module in the B 6800 system is defined as a unit of hardware equipment that performs a specific function, or a set of specific functions. A module of hardware equipment in the B 6800 system is limited to a single system cabinet. Modules in separate cabinets that perform similar functions are separate modules.

A B 6800 system cabinet is not limited to a single module. The use of new types of logic circuit devices in the B 6800 have made it possible to mount more modules in a cabinet than was possible in the B 6700 system. Figure 1-4 is a block-diagram of the B 6800 system that shows the relationship of the modules in the B 6800 system.

B 6800 MODULE INTERFACES

Cabinets within the B 6800 system are connected together through a series of interface buses (see figure 1-5). These buses provide a method for the transfer of information and control data, and power between system modules. The major buses in the B 6800 system are as follows:

Memory bus	A B 6800 system has up to four modules of local memory and also can interface with one global memory. A separate interface is required for each memory module. The interface between a memory module and the central processor is called a memory port. The CPU contains provisions for a total of five ports.
Scan bus	The scan bus interfaces the CPU with subsystems that are expansions of the 6800 system. The scan bus is used to transfer control information and data between the CPU, and the subsystems. The B 6800 system CPU contains provision for a single scan bus interface.
Peripheral bus	The peripheral bus interfaces the CPU to the peripheral control cabinets. The peripheral bus is used to transfer control data, and information between the CPU and either one, or two peripheral control cabinets. At least one peripheral bus is required, and a second bus may be used to expand the B 6800 system by adding another peripheral cabinet to the system.
Power buses	The power buses are used to connect the cabinets of the B 6800 system to the centralized power supply cabinet. These buses are used to transfer control signals to the power supply cabinet, and to distribute the power from the central power supply cabinet to the other cabinets in the system.
PCIO	The peripheral control IO is used to interface the maintenance display cabinet to the CPU cabinet. The PCIO is used to provide a path from the maintenance display processor to up to four peripheral devices.
MFIO	The mainframe IO interfaces the maintenance display cabinet to the CPU cabinet. This interface is used to control and sample the state of flip-flops in the B 6800 system. The sampled state of a flip-flop is displayed by illuminating an indicator on the display panel. Control consists of setting, or resetting the state of flip-flops.

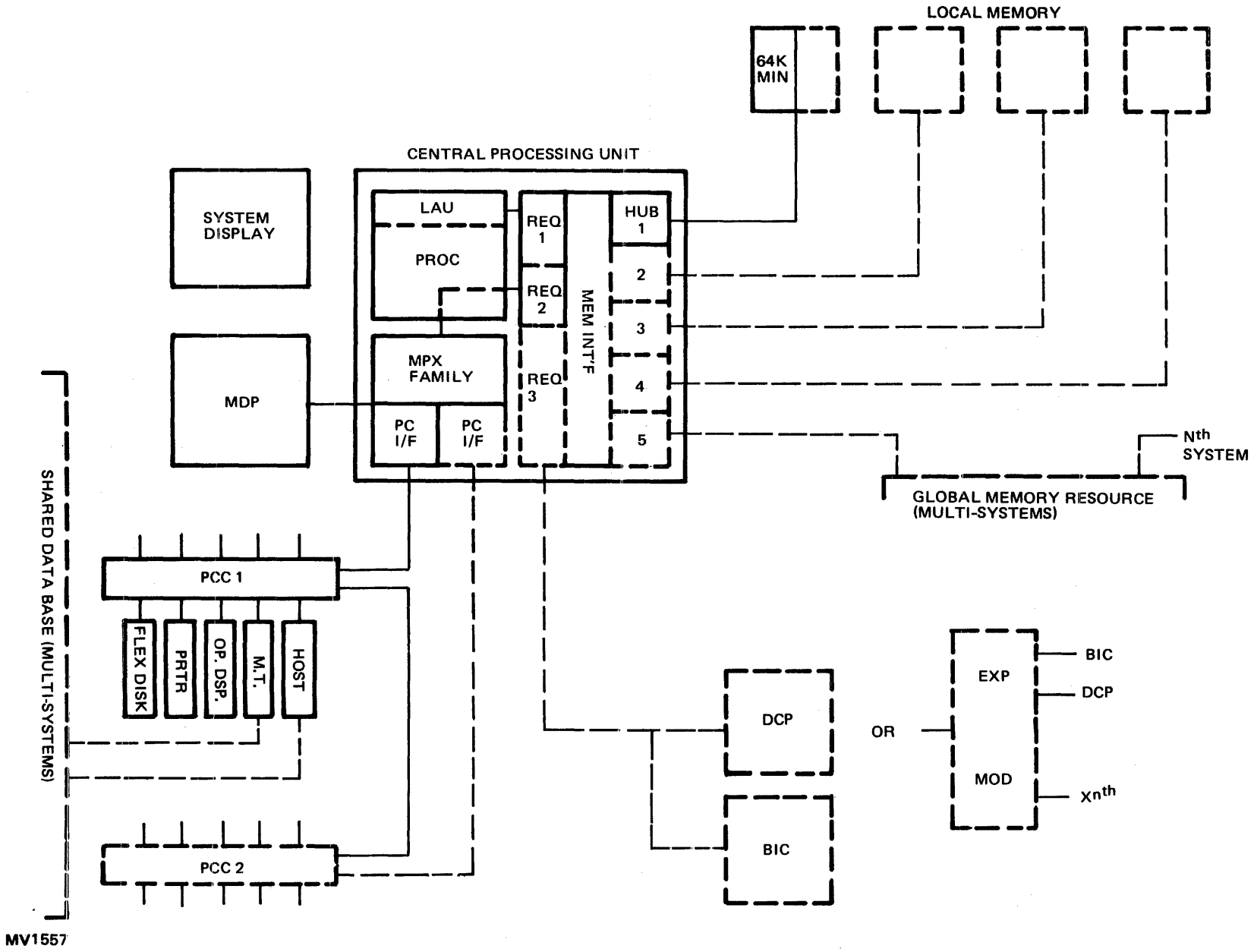


Figure 1-4. B 6800 System Module Block Diagram

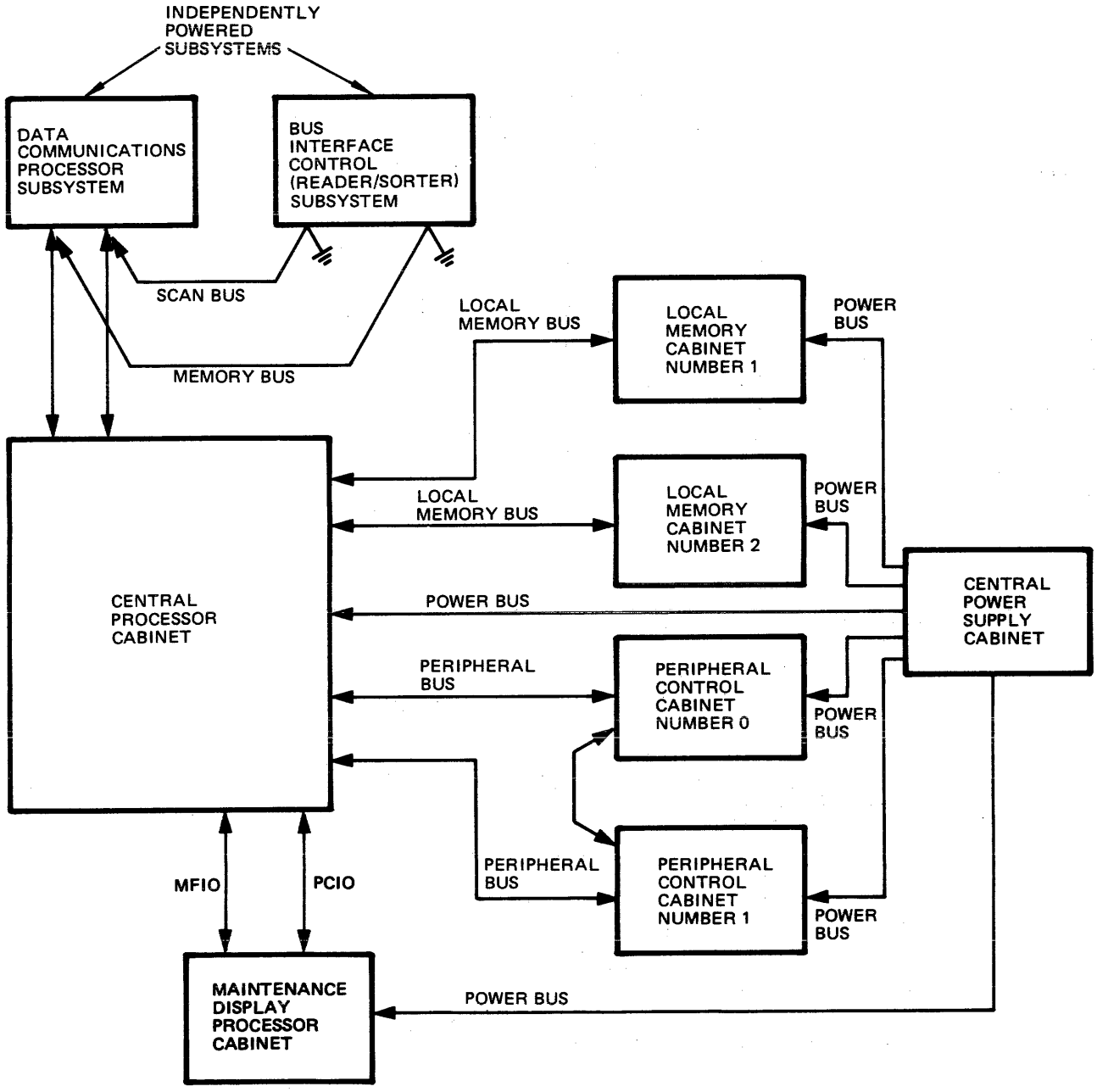


Figure 1-5. B 6800 System Busing

B 6800 CENTRAL PROCESSING UNIT CABINET

The Central Processing Unit (CPU) is the heart of the B 6800 system. The CPU contains four modules, which are the data processor module, the IO processor module (also referred to as a multiplexor, or MPX module), the memory exchange, and the memory tester. The CPU also generates system clock pulses that are distributed to other modules in the system. The CPU contains logic circuits that operate in conjunction with the maintenance display processor to perform memory testing. The following pargraphs will discuss the modules that are located within the CPU cabinet.

The CPU contains the master clock generation circuits for the clock pulses of the system. The master clock frequency of the B 6800 system is 6.67 megahertz. This clock frequency produces clock pulses that occur each 150 nanoseconds. These clock pulses are distributed throughout the logic circuits of the system. The clock pulses in the B 6800 system are used to synchronize the various circuits contained in the modules of the system. In this manner, each circuit operates in concert with other circuits in the system, in an harmonious, and efficient manner.

DATA PROCESSOR MODULE

The Data Processor (DP) is the key module through which the B 6800 software operating system directs and controls the resources of the B 6800 system. The DP initiates all operations performed by the other system modules, including the operation of all peripheral devices. The DP also performs data arithmetic operations, and manipulates data within the system. The DP contains logic circuits to sense interrupts from other modules, and also within itself. When the DP senses an interrupt the software operating system becomes aware of the interrupt, and handles the cause of the interrupt. The DP performs comparisons, and logical operations that allow the software operating system to evaluate conditions, represented as data, and to make decisions based on the results of the evaluation. Because the software makes decisions, it provides the capability for altering the future course of programmed operations both within the operating user programs, and within the MCP itself.

A new feature of the B 6800 system is the use of look-ahead logic in the DP. This new feature fetches words of program code before the DP is ready to execute the code, and thus virtually eliminates the need for halting a program to fetch words of program code. The memory accesses that are performed by the look-ahead logic are independent of other memory cycles performed for the DP, and do not cause delays in obtaining data for normal DP functions. When a new word of program code is required, the first resource is the buffer circuit of the look-ahead logic. A memory cycle will only be performed if the look-ahead logic has not already fetched the word of code that is needed, or a branch operator causes a change in the sequential program code addressing. If the next word of program code is the proper program word, and is present in the look-ahead logic buffer circuit, then that is the source from which the next word will be taken.

The DP of the B 6800 system also contains an improved adder circuit for performing arithmetic functions. The improved mantissa adder circuit is a double-precision, high-speed adder which is more efficient than the single-precision adder that was used in the B 6700 system. In addition to using the new adder circuits, the algorithms for double precision arithmetic operations have been improved to provide more accurate double precision answers to arithmetic problem-solving processes.

The B 6800 DP contains new logic circuits that provide for a retry of a DP operator that fails during its execution. This retry of failed operators is only applicable up to a predetermined point in the flow of an operator. If an operator fails, and a retry is possible, then a flag is set to indicate that the retry can occur. If an operator fails, and a retry is not possible, then the failure will result in the execution of the interrupt procedure for failed DP operators.

A failed operator retry operation is controlled by the system software. The system hardware indicates whether or not a retry may be attempted, but the decision to retry a particular operation is made by the software.

The B 6800 DP makes extensive use of RAM, and PROM memory integrated circuit components. Parity testing is performed on these component parts in the DP. When a failure of one of these component parts is detected, an entry is made in the Error register. The error register is decoded, and written in the system log. The log entry will provide such pertinent data as:

a. The location of the card package that failed

b. The J-count sequence number, and OP code of the DP operator that was being executed at the time of the failure

The B 6800 system DP performs recursive confidence testing when the DP is in an IDLE condition. The confidence tests check such circuits as:

a. The top of stack registers

b. Shift paths for data that is placed in the top of stack registers

c. The barrel shifter logic

d. The mantissa adder logic

e. The exponent adder logic

f. The address adder logic

g. The arithmetic operation algorithms

h. The DP control buses

If an interrupt occurs while the DP is performing a confidence test, the DP will immediately exit from the IDLE state. If an error caused the exit, the error will be reported in the SYSTEM SUM LOG disk file.

The DP performs residue testing of the contents of the integrated circuit memory address registers. Residue testing is also performed on literal values that are used as indices to the addresses that are contained in the integrated circuit address registers. The purpose of residue testing is to increase the integrity of the address adder circuits. Residue testing is an automatic function that detects addressing errors, and cause the software operating system to make log entries that identify the nature of the error.

MULTIPLEXOR MODULE (IO PROCESSOR)

The multiplexor module controls the operation of all of the peripheral subsystems except the data communications processor, and the bus interface control. These two subsystems are operated as separate subsystems, and are controlled directly from the data processor module.

The data processor module initiates IO operations in the B 6800 system by transmitting command instructions to the multiplexor via the interface bus. The command instruction information that is transmitted to the multiplexor contains such data as the unit number of the IO device that is to be initiated, the address of a buffer area in memory that is to be used for the operation, and the length of the buffer area. The multiplexor stores the command information which it receives from the DP in scratch pad memory.

From the time that the multiplexor receives data from the DP for an IO operation that is to be performed, until the IO operation is completed, or is interrupted, the multiplexor operates as an independent module from the DP. When an IO operation needs access to memory it has priority for the use of the single path to memory that is shared by the DP, and the multiplexor.

The multiplexor will proceed in an independent manner to control the IO operations until they are completed. When the operation is completed, the multiplexor uses an interrupt path to the DP to report that the operation has been terminated. If the peripheral operation was terminated because of an IO error, then the DP will interrogate the multiplexor to determine the cause of the error. In this way the operating software system is aware of what IO operations are in process, or have been completed.

The multiplexor contains provisions for 2 peripheral bus interfaces. Each peripheral bus interface conducts data and control information communications for up to 10 peripheral channels. Thus, the multiplexor contains provisions for either 10 channels, or 20 channels to the IO devices. A maximum of 256 peripheral devices may be controlled by the multiplexor. The minimum number of IO devices that are used in a B 6800 system is five devices, which are:

a. A TD830 Operators Display

b. A 225 Dual Disk Pack Drive

c. A model V Magnetic Tape Transport Unit

d. An 1100 LPM Train Printer

e. A Card Reader or a Flexible Disk Unit

Any substitutions and/or additions to the peripheral unit list above must be made from the following B 6800 compatible peripheral unit list.

a. 150/300 CPM 80 Column Card Punch

b. 800/1400 CPM 80 Column Card Reader

c. 300/600/800 CPM 80 Column Card Reader

d. 1C Disk File

e. 5N Disk File

f. Nine Track PE Magnetic Tape Transport Unit

g. Seven Track NRZ Upright Magnetic Tape Transport Unit

h. 400/700 Lines Per Minute Train Printer

i. 206 Disk Pack Subsystem

j. 235 Disk Pack Subsystem

The 15 IO device types listed above, in addition to the B 9137/B 9134 reader/sorter (BIC module) are the standard IO devices that are utilized in the B 6800 system. When the BIC module is utilized, the reader/sorter units are not considered as part of the 256 peripheral unit limitation. This limitation is only applicable to the peripheral devices that are under the control of the IO processor. The reader/sorter subsystem is not controlled by the multiplexor.

MEMORY CONTROL MODULE

The memory control module operates a memory interface exchange that allows three different system requestors to access one of five memory storage modules. The three requestors are as follows:

a. The look-ahead logic of the data processor module.

b. Either the data processor module, or the multiplexor module. These two modules share a common requestor path to the memory control exchange, as was defined in the sub-section on the multiplexor.

c. The external requestor is utilized for a data comm processor and/or a bus interface control module. This external requestor path to the memory control module may alternatively be connected to a memory control expansion module. If a memory control expansion module is utilized then up to four external requestors can share the single path to a memory storage device, through the memory control module.

The five memory storage modules that may operate as respondents to the three memory control requestors are defined as follows:

a. The first four modules are each either 64K, or 128K local memory modules (1K = 1024 words).

b. The fifth module is an interface to the global memory of the B 6800 system.

In addition to controlling the interface paths through the memory exchange, the memory control module also performs memory retries, and memory read data error corrections. A read memory retry consists of detecting an error in the data fetched from memory, and causing a second memory read strobe pulse to be generated. A read memory retry is not a second complete memory cycle. A read memory retry will only be performed for a requestor that is internal to the CPU module.

A memory retry is also performed when a memory module detects a parity error on the address data lines. The memory address error retry will repeat the complete memory cycle operation.

An error correction memory cycle will be performed for a read memory cycle that detects a single bit error in the data that was stored. If a memory cycle still produces and error in the data after a read memory cycle retry has been performed then the memory control module will perform an error correction cycle. An error correction cycle can only correct single bit errors.

External memory requestors operate asynchronously, and internal memory requestors operate synchronously when requesting access to the memory modules. This condition causes the memory access time for an external requestor to be greater than the access time for an internal requestor. The memory cycle times for internal requestors, and external requestors are as follows:

Device Type	Read/Restore Operation	Clear/Write Operation	Read/Modify/Write Operation
Internal Requestor	900 ns	900 ns	1200 ns
External Requestor	1200 ns	1200 ns	1500 ns

The memory cycle times listed above are based on the assumption that no retry (for an internal requestor), or error correction cycle is performed. At least two clock periods have been added to the cycle times for external requestors because of the asynchronous memory interface. If a retry for an internal requestor and/or an error correction cycle is performed, then one clock period (450 nanoseconds) must be added to the memory read times listed.

The look-ahead logic always yields to any other channel A requestor for priority to access memory. Contention for access to memory is allowed between the data processor and multiplexor interface (channel A), and the external requestor interface (channel B). Such contention is resolved on a first come first served basis. If both channel A and B memory requestors request accesses at the same time then channel A interface takes precedence over the channel B interface.

B 6800 MAINTENANCE DISPLAY PROCESSOR (figure 1-6)

The Maintenance Display Processor (MDP) Cabinet layout is shown in figures 1-3, and 1-4. The MDP performs display and control functions in the B 6800 system. The leading features and functions of the B 6800 maintenance display Processor are:

a. The MDP can display the states of up to 4096 logic devices

b. The MDP can write into and verify the code of a PROM device

c. The MDP contains logic card package testing capability, with static go/no go test cases for all non-discrete logic cards

d. The MDP can operate up to four selectable system IO devices

e. The MDP can be programmed to beam test (at single clock level) and to compare all flip-flops in the system, as well as any flip-flop that is under test

f. The MDP can be programmed to allow a system operator to test the logic circuits of the system at the single clock level

g. The MDP can be programmed to dynamically isolate most failures that occur in the hardware elements of the system

Figure 1-6 shows the major parts of the MDP cabinet.

The MDP cabinet can be divided into three main parts:

a. The upper half of the cabinet contains the displays

b. The lower half of the cabinet contains the micro-processor, the display control logic, five IO controllers, and a power supply for the micro-processor

c. The upper half of the cabinet is separated from the lower half of the cabinet by a keyboard

The keyboard consists of the necessary indicators and switches to perform the following functions:

a. Provides a means of selecting one of the four register displays for the purpose of manually changing the state of the data displayed

b. Provides a means of selecting a hexadecimal digit through cursor movement for the purpose of manually changing the bit pattern of that digit for the selected register display

c. Provides a means of entering a hexadecimal character to alter a selected digit within the selected register display

d. Provides a means of manually initiating a memory read/write cycle on either the mainframe memory or the IC memory within the data processor

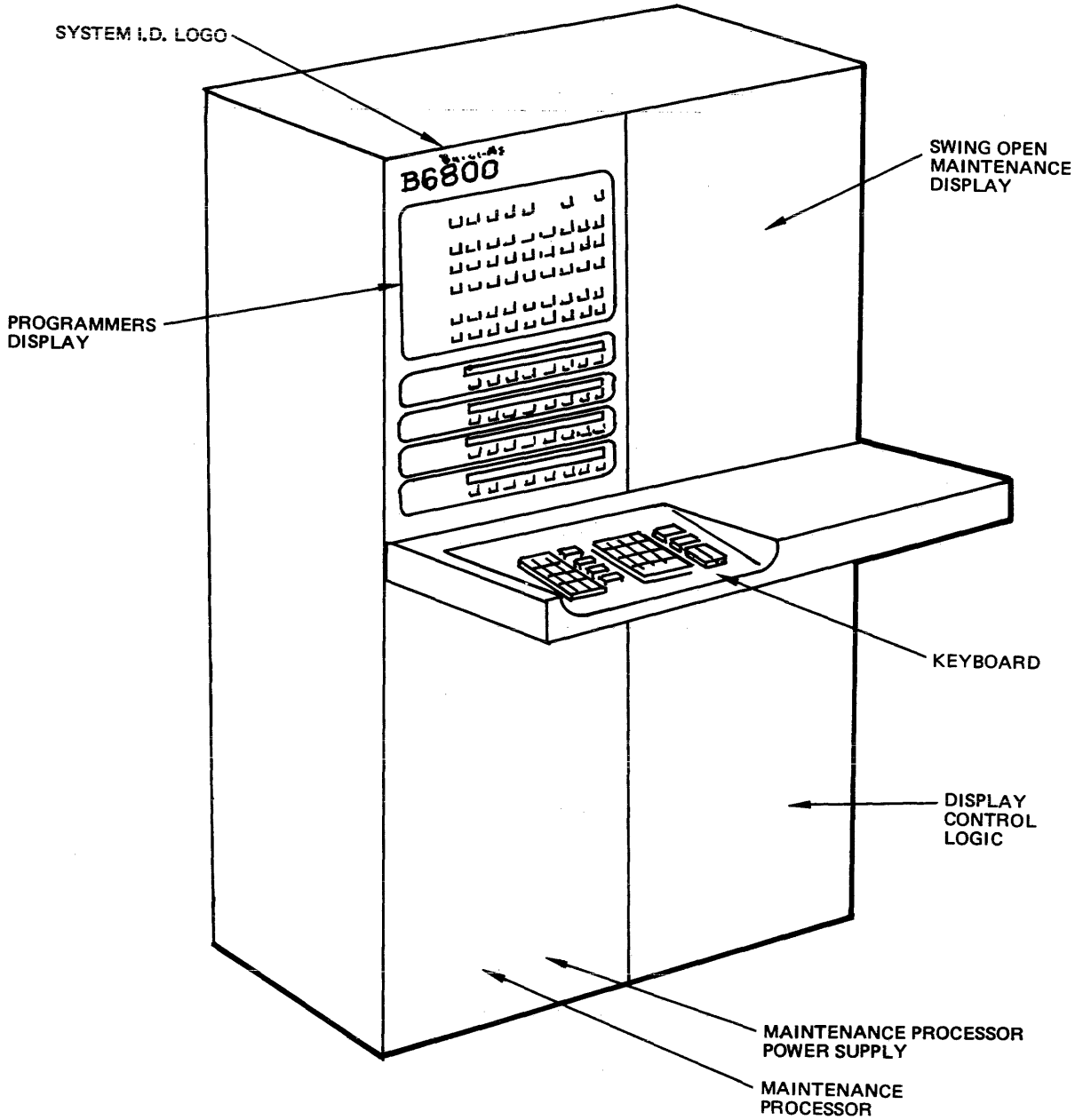


Figure 1-6. Maintenance Display Processor Cabinet (sheet 1 of 2)

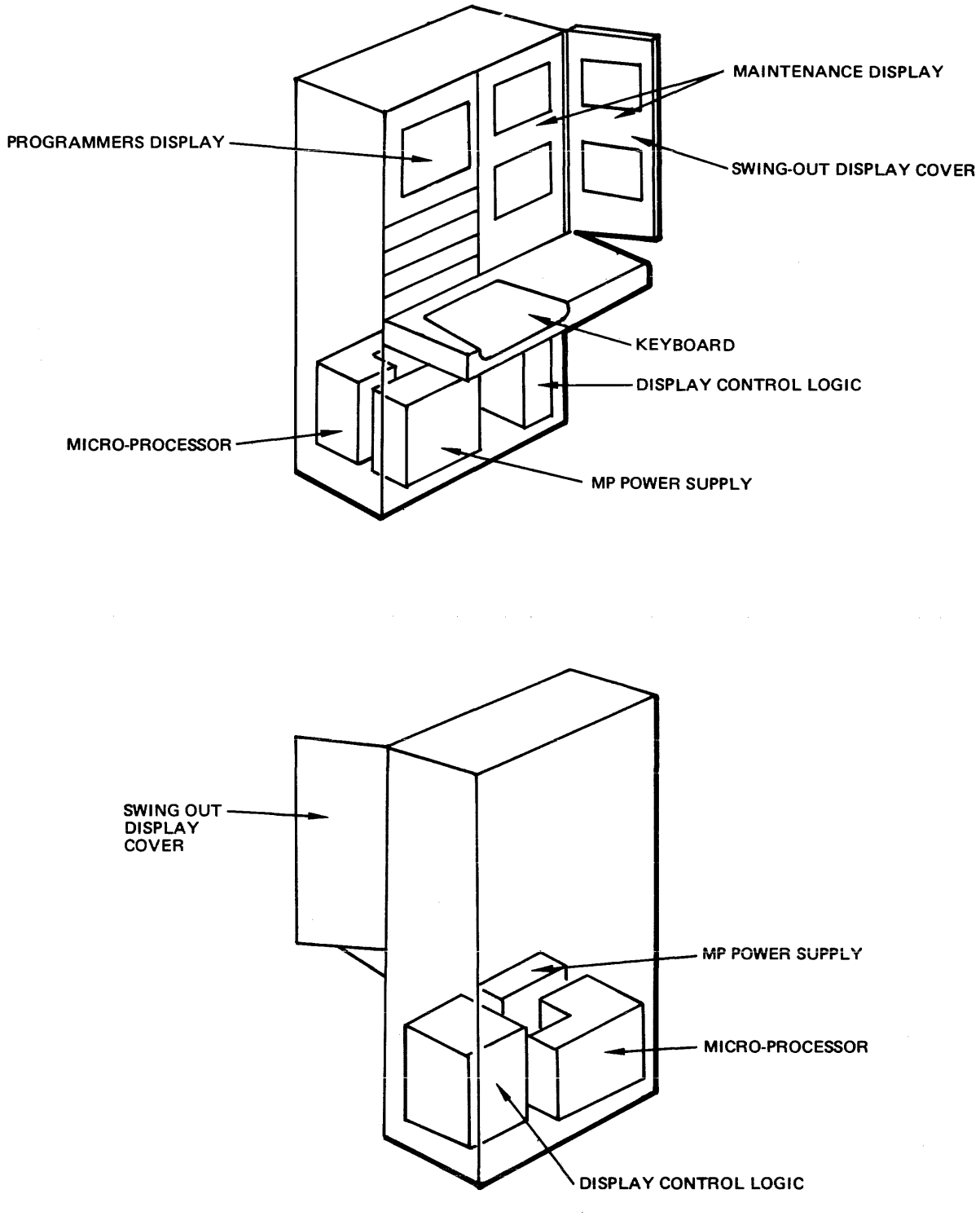


Figure 1-6. Maintenance Display Processor Cabinet (sheet 2 of 2)

e. Provides a means of manually adjusting the top of stack registers in the data processor

f. Provides a means of manually inhibiting the start of the next CPU instruction

g. Provides a means of manually initiating a conditional halt to the CPU

h. Provides a means of selecting either hexadecimal display or octal display for display registers one, two, or three

The displays are divided into the programmers display panel, and the maintenance display panels. The programmers display is on the upper left-hand side of the MDP cabinet, and is always visible. The maintenance display panels are on the upper right-hand side of the MDP, and are not always visible. To view the maintenance panels, a swing-out display cover must be extended. Four maintenance panels are exposed to view when the swing-out cover is extended. A switch panel is located at the bottom of the maintenance display panel. This switch panel is used to control the operation of the maintenance processor.

The maintenance processor is the principal operating unit in the MDP cabinet. The maintenance processor operates in either of two modes, which are; Maintenance Test Routine mode (MTR), and normal mode. These two modes will be discussed in the following paragraphs.

The PROC ENABLE switch (on the MDP switch panel) is used to place the maintenance processor in the MTR mode. The MTR mode provides a way of testing the maintenance processor through the use of test-routines that are stored in PROM memory. The PROM memory is an integral part of the maintenance processor. This PROM memory contains firmware to:

a. Test the maintenance processor circuits

b. Test the memory interface logic between the maintenance processor and RAM memory which is an integral part of the Micro-Processor.

c. Test the RAM memory up to a checkerboard test

d. Test the PCIO, CTIO, MFIO controller of the MDP

e. Perform an extensive (Galpat) test on the RAM memory

f. Load the MTR test-routine program from the selected system peripheral unit to the RAM memory of the MDP

g. Perform a program branch to the start of the MTR test-routine that was loaded into the RAM memory

h. Handles Interrupt procedures that occur during the operation of the maintenance processor in MTR mode

The same switch that was used to place the maintenance processor in MTR mode (the PROC ENABLE switch) is also used to select normal mode. The normal mode of operation provides a way to test the B 6800 system through the use of the MTR test routines that are loaded to the RAM memory. The maintenance processor uses

the PROM memory to initiate the loading of MTR test-routines into the RAM memory as follows:

a. Sets up peripheral tables for the peripheral units that are to be used

b. Provides a quick confidence check for the peripheral units that are to be used

c. Initializes the RAM memory to receive the data from the IO devices

d. Purges the RAM memory of all parity errors

e. Communicates with the system operator to determine which system IO devices are to be used to load the system MTR program

f. Loads the system MTR program from the selected IO devices, into the RAM memory

g. Performs a program branch to the start of the system MTR test-routine which is residing in the RAM memory

h. Handles interrupt procedures during system operation

The micro-processor logic contains two controllers which are the PROM writer controller (PWIO), and the Keyboard/ Switch/Indicator (KSI) controller.

The purpose of the KSI controller is to interface the micro-processor to the keyboard of the MDP. The keyboard is used as a source input device by a human operator, to direct that various functions of the MDP be performed. The KSI controller coordinates and synchronizes these human control demands with the normal logical operations of the micro-processor. The orderly responses of the micro-processor, to a keyboard demand, are returned to the keyboard for display, by the KSI controller.

The PROM Writer IO controller provides a method of creating a selected bit pattern in a PROM device. In addition, a PROM device can be verified to have the correct pattern inserted.

The MDP contains three controllers which are as follows:

a. The Mainframe Input Output (MFIO) controller

b. The Peripheral Control Input Output (PCIO) controller

c. The Card Test Input Output (CTIO) controller

The purpose and use of each of these three controllers is defined in the following paragraphs.

The purpose of the Mainframe IO (MFIO) controller is to allow either the micro-processor or the display logic to set and sample the state of the mainframe flip-flops. In addition, the micro-processor can monitor various conditions within the controller through the use of status and data transfers. The MFIO controller interfaces the logic of the MDP with one of two connectors that are identified as normal, and alternate interfaces. The PROC ENABLE switch selects either the micro-processor, or the display logic to control the data lines from the MDP to the CPU cabinet.

The micro-processor uses a set of command words, and fixed format status reports to control the operation of the MFIO controller. These controller directing commands, and status reports are passed between the micro-processor and the

MFIO logic over the DIN and DOUT lines of the MFIO interface bus. The format and use of the MFIO command words, and the status reports are covered in detail in the B 6800 Maintenance Display Processor Field Engineering Technical Manual, number 5001340.

When the PROC ENABLE switch is in the ENABLED position (UP) the micro-processor is permitted to control data that is sent to the CPU cabinet; and therefore, control the setting of mainframe flip-flops. When the PROC ENABLE switch is in the down position, the display logic controls the data sent to the CPU cabinet; and therefore, the setting of mainframe flip-flops.

The purpose of the Peripheral Control IO Controller (PCIO) is to provide the maintenance processor with a way to communicate with the peripheral units that are attached to the system. The peripheral control IO controller controls the PCIO bus between the CPU and the MDP. The PCIO controller contains a 1024 byte IC memory buffer that is used to hold the data that is received from an IO device. The PCIO controller can initiate four different IO devices, but only one IO operation is allowed to be in process at any one time. Configuration jumpers are used to select which four system IO devices the PCIO controller will be allowed to initiate. The maintenance processor uses a set of command words, and status reports to control the PCIO controller, and the PCIO interface bus to the CPU.

The Card Test IO Controller (CTIO) is used to test logical card-packages from the hardware of the B 6800 system . The CTIO logic can control the state of each pin of a card-package, both the foreplane, and the backplane pins. The logic of the CTIO controller can issue clock pulses to any of the six pins that normally receive clock pulse inputs. The logic can also issue bursts of up to 15 clock pulses. The CTIO controller contains a 120 bit pin state register that is used to contain the state of each pin of a card under test. The maintenance processor controls the operation of the CTIO controller through a set of command words. The maintenance processor samples the state of the card-package pins, performs comparisons against known good results, and isolates failures of the logic on the card-package. The maintenance processor controls the logic circuits on the cards that are tested because it controls the state of each pin on the card. The testing of a card-package by the maintenance processor is conducted as a series of test-cases. This method allows a logical failure on the card-package to be repeated in a recursive manner such that the card-test logic can be used for dynamic trouble-shooting by engineering personnel.

DISPLAY CONTROL LOGIC

The display control logic (see figure 1-6) is located in the bottom half of the MDP cabinet. The operation of the display control logic is controlled by the maintenance processor.

B 6800 CENTRAL POWER SUPPLY CABINET

The central power supply cabinet in the B 6800 system is an A size cabinet, which is located near the center of the system cabinet complex (refer back to figure 1-2). The Central Power Supply Cabinet (PSC) provides centralized power to all cabinets within the B 6800 system except for the independently powered cabinets.

Power buses route the power generated in the PSC to other cabinets in the B 6800 system. The source power to the B 6800 system PSC is discussed in the B 6800 System Installation Planning Manual, number 5001308.

The power supplies in the B 6800 system PSC are capable of supplying electrical power to the mainframe cabinets of the system. The power supplies in the PSC use constant voltage transformers, that provide sufficient pre-regulation conditions to ensure constant voltage outputs with a loss of input power of up to 30 percent of normal line supply. These design characteristics in the PSC provide for continuous system operation during "brown-out" operations. A "brown-out" is defined as a reduction of up to 15 percent of normal operating line voltage, for an unspecified period of time.

Figure 1-7 shows the major parts of the PSC, and the relative location of these parts within the cabinet. Figure 1-8 shows the power bus distribution between the PSC, and other cabinets within the B 6800 system mainframe.

B 6800 PERIPHERAL CONTROL CABINET

The B 6800 Peripheral Control Cabinet (PCC) is an A size cabinet that contains three separate logic backplane panels (refer to figure 1-9). Two of the three panels contain control modules, and the third panel contains control logic for the peripheral interface between the multiplexor, and the PCC. As shown in figure 1-9, one of the control mounting panels accommodates control modules for IO controls of up to 86 card locations. The other control mounting panel can accommodate IO controls of up to 36 card locations. A PCC can accommodate a maximum of ten IO controls. The card sizes of the two IO control mounting panels refers to the number of plug-in card-modules that can be physically plugged into the backplane of an IO control.

A PCC can accommodate up to ten IO controls. These 10 controls can be any combination of large and small controls; however, a maximum of five large controls may be included in a cabinet.

The peripheral control interface bus that connects a PCC to the multiplexor (see figure 1-10) is connected to the central control logic panel of the PCC. The B 6800 system utilizes this interface bus to transfer data between the main system modules, and one of 256 system IO devices. The interface bus is also used to transfer control information from the mainframe system, to the peripheral controls. Within the PCC, data, and control information is passed to a control module via interframe jumpers.

Figure 1-10 is a representation of the interface bus that passes between the IO processor, and the PCC. In figure 1-10 there are two cables that go to each of the PCCs. In addition, there are two cables that go to both of the PCCs. The cables that go to both PCCs are used to pass data, and control information between the IO processor, and a peripheral control. The two cables that go to a particular PCC are used to select a particular IO control position within the PCC with which the IO processor will communicate. The IO control positions within a PCC are designated as channels, and are further identified by a channel numbering system, to distinguish one IO control position from another.

The large control channel numbers within a PCC are zero through four. Channel zero is the lowest large IO control position in a PCC, and channel four is the highest large IO control position. The lowest small IO control position in a PCC is channel number five, and the highest small IO control position in a PCC is channel number nine.

The peripheral control bus cables contain interface signals that identify which channel within a PCC is to communicate with the IO processor on the peripheral bus, and also in which direction on the bus the information will be passed.

When a single PCC is used in a B 6800 system the channel numbers that may be used are zero through nine. When a second PCC is used, the channel numbers in the first PCC remain unchanged, but the channel numbers in the second PCC are channels ten through nineteen. Thus; the value of a channel designation defines the channel position within the PCC, and also in which of two PCCs the channel is located.

B 6800 System Peripheral Controls

The peripheral controls that may be mounted in a B 6800 PCC are limited to those controls that are compatible with the B 6800 mainframe system. The lists of those peripheral devices that are compatible with the B 6800 system were previously identified in the subsection that is titled MULTIPLEXOR MODULE, in this manual.

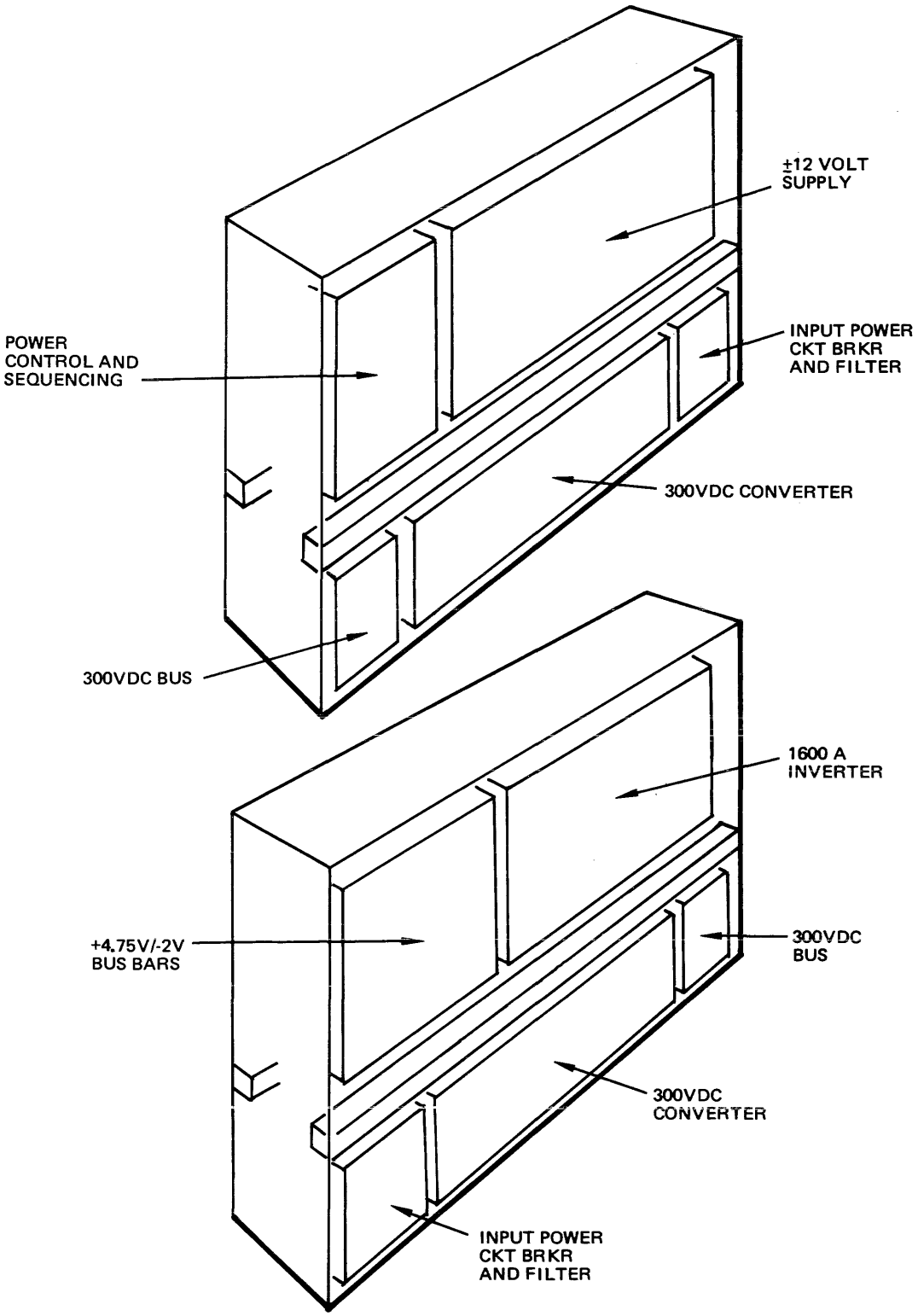


Figure 1-7. Central Power Cabinet

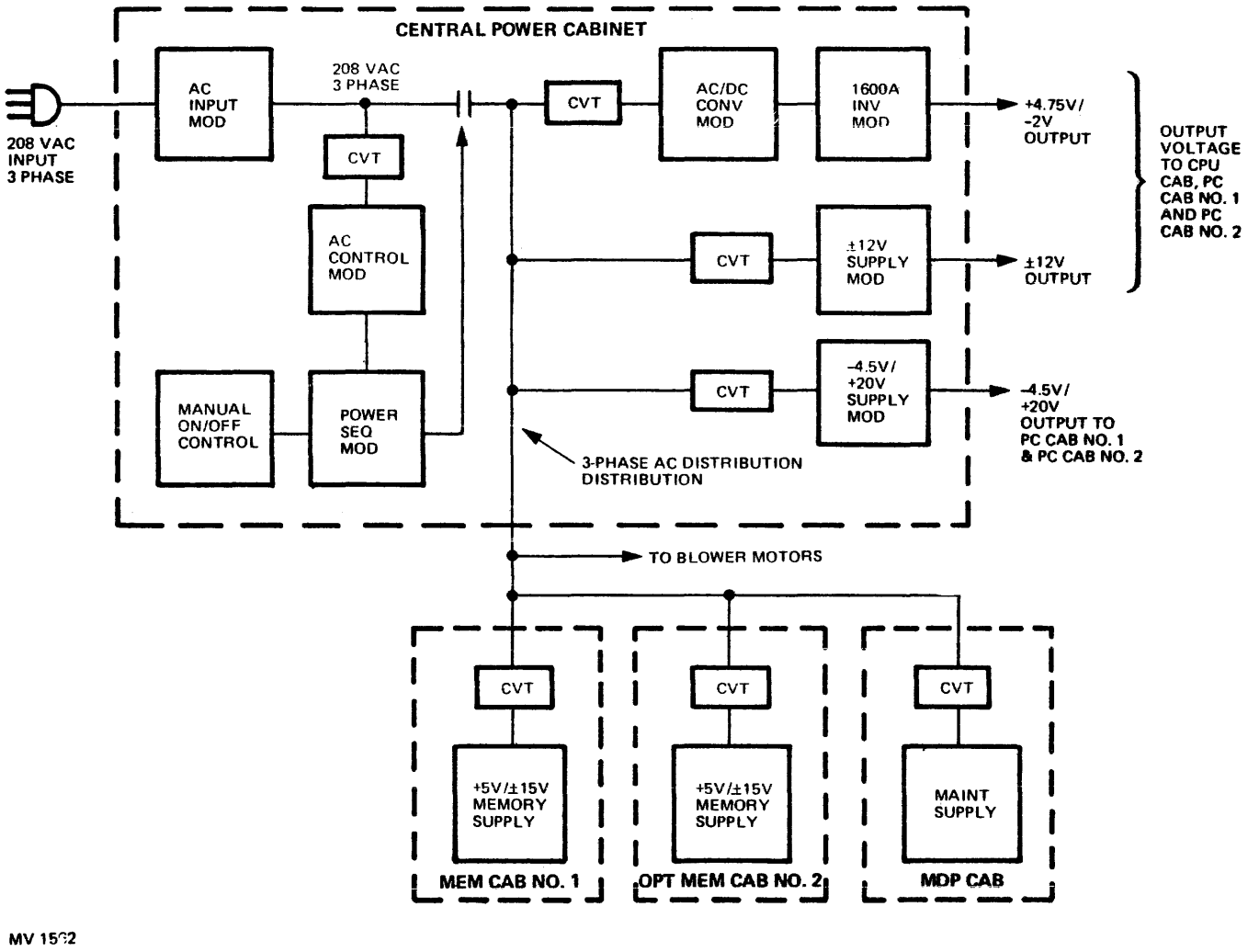


Figure 1-8. B 6800 Power Subsystem Distribution Diagram

There are 15 different types of peripheral devices that are completely compatible with the B 6800 system mainframe. Nine of these peripheral types require a small control, and the other five types require a large size control.

A peripheral control is associated with one or more system IO devices. The UNIT NUMBER that is used to identify a peripheral device, is also associated with the IO control through which the device is operated. UNIT NUMBERs of the peripheral devices that operate through a single IO control must follow the minterm numbering conventions for peripheral devices.

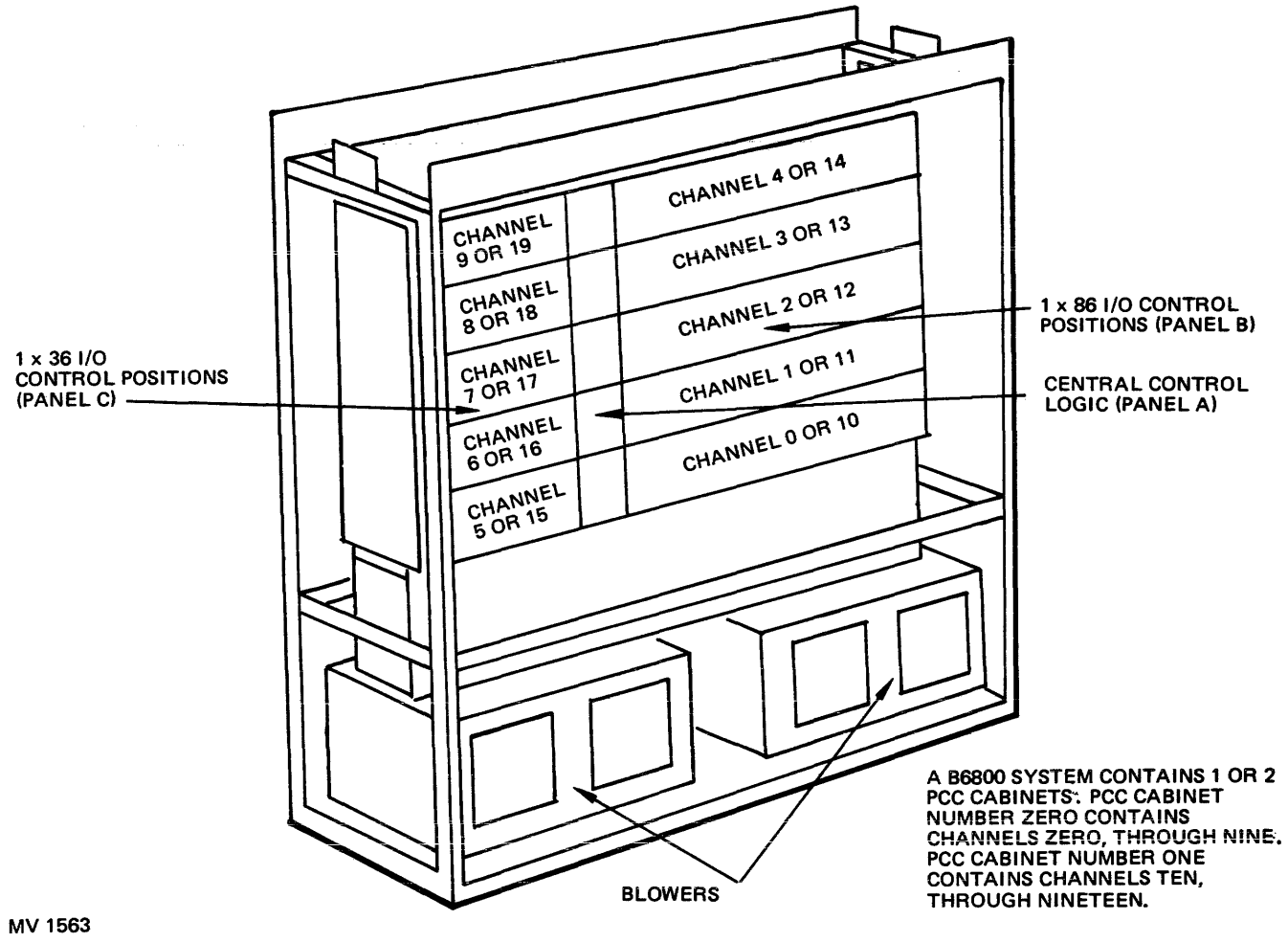


Figure 1-9. Peripheral Control Cabinet

The minterm numbering conventions for the B 6800 system are the same as the conventions that were used for the B 6700 system. A minterm group consists of a group of one, two, four, eight, ten, sixteen, or twenty peripheral devices. Within the minterm group, all of the unit numbners are in consecutive numeric sequence. Thus, all of the UNIT NUMBERs associated with a particular IO control are in consecutive numeric order. The Multiplexor module of the CPU cabinet contains logic circuits that define a particular IO control channel number according to the UNIT NUMBER.

The MCP software operating system constructs UNIT TABLES through which it associates a UNIT NUMBER with a particular IO device type. The MCP keeps the UNIT TABLES updated so that they contain the current status of each peripheral device, by unit type, and by UNIT NUMBER. In this manner, the software operating system is aware of the condition and extent of the IO device subsystems at all times.

B 6800 Memory Cabinets

The B 6800 local memory cabinet (refer to figure 1-11) is a B size cabinet that can contain a maximum of 256K words of local memory. With a maximum of two local memory cabinets in a B 6800 system, a maximum of 512K words of local memory is available to the system. Local memory is expandable from 64K words to 512K words, in increments of 64K words. In the common context, one K of memory is actually 1024 words in length.

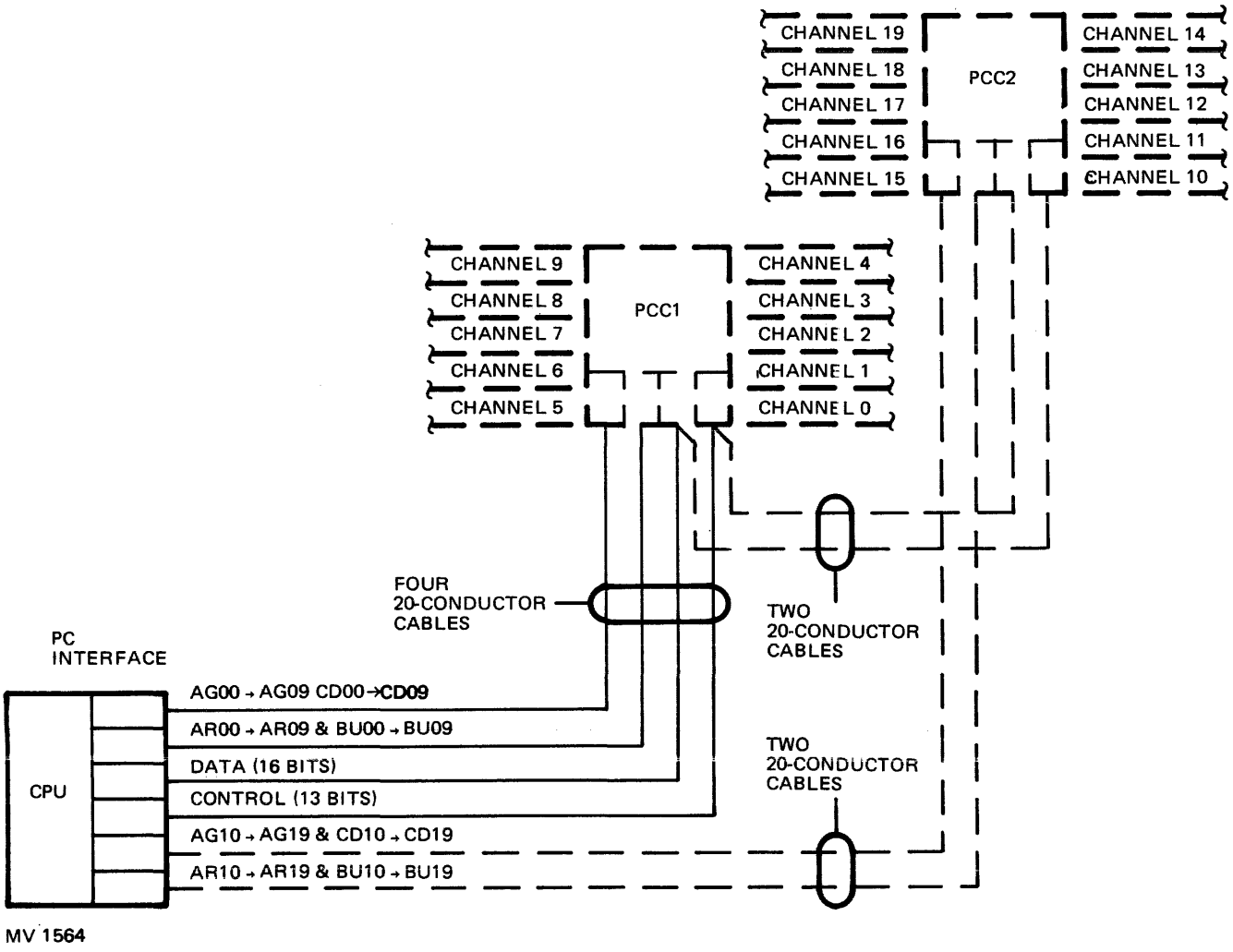


Figure 1-10. Peripheral Control Interface

Each word of memory consists of 60 bits. These 60 bits are divided to provide 51 bits of data, one parity bit, and eight bits which are utilized for error detection and correction.

A B 6800 memory interface consists of six cables. Figure 1-12 shows these six cables, and how they operate to provide the interface between the memory control module of the CPU cabinet, and a B 6800 memory module.

The B 6800 memory modules are capable of performing in any of three types of operations as follows:

a. Read/Restore operation

b. Clear/Write operation

c. Read/Modify/Write operation

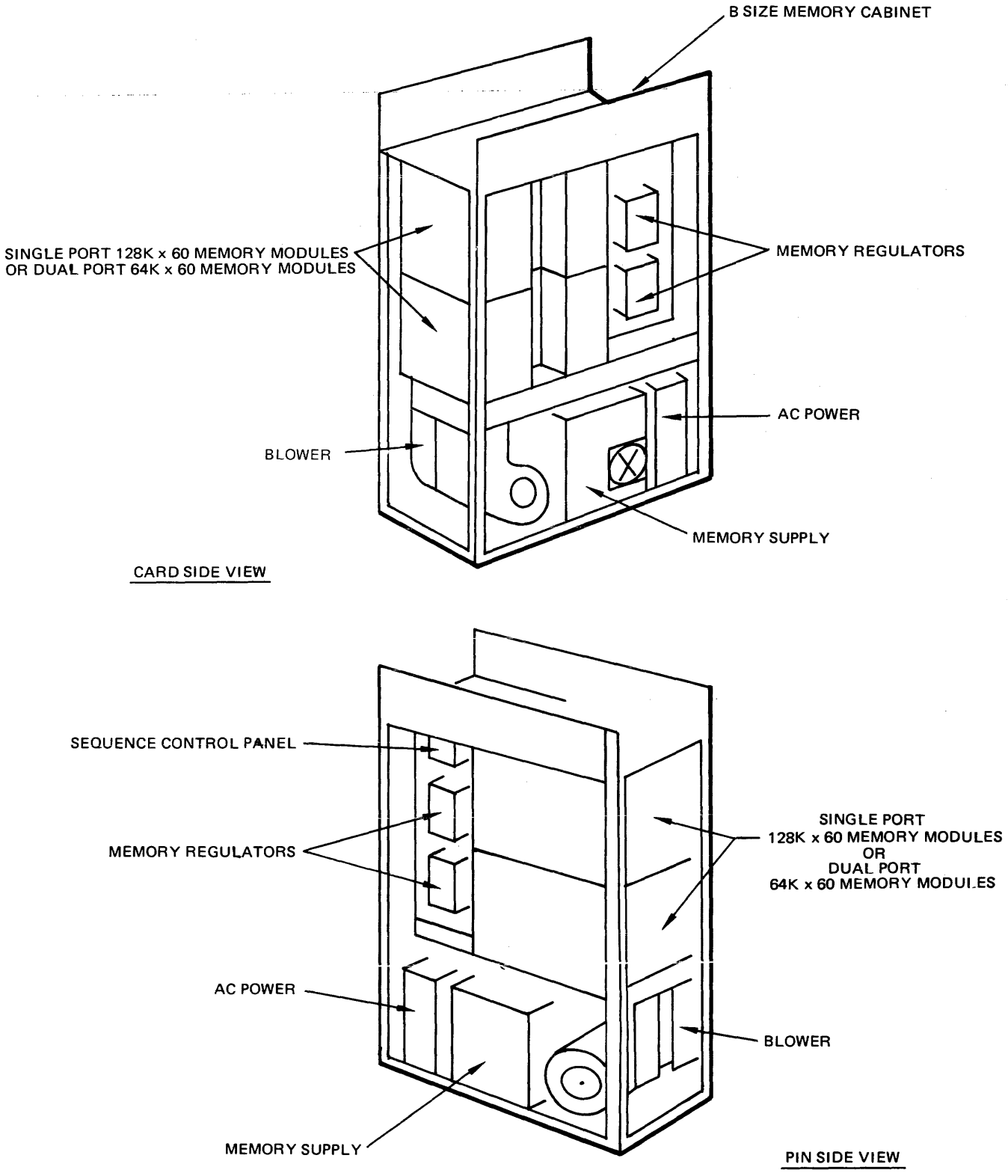


Figure 1-11. B 6800 Memory Cabinet

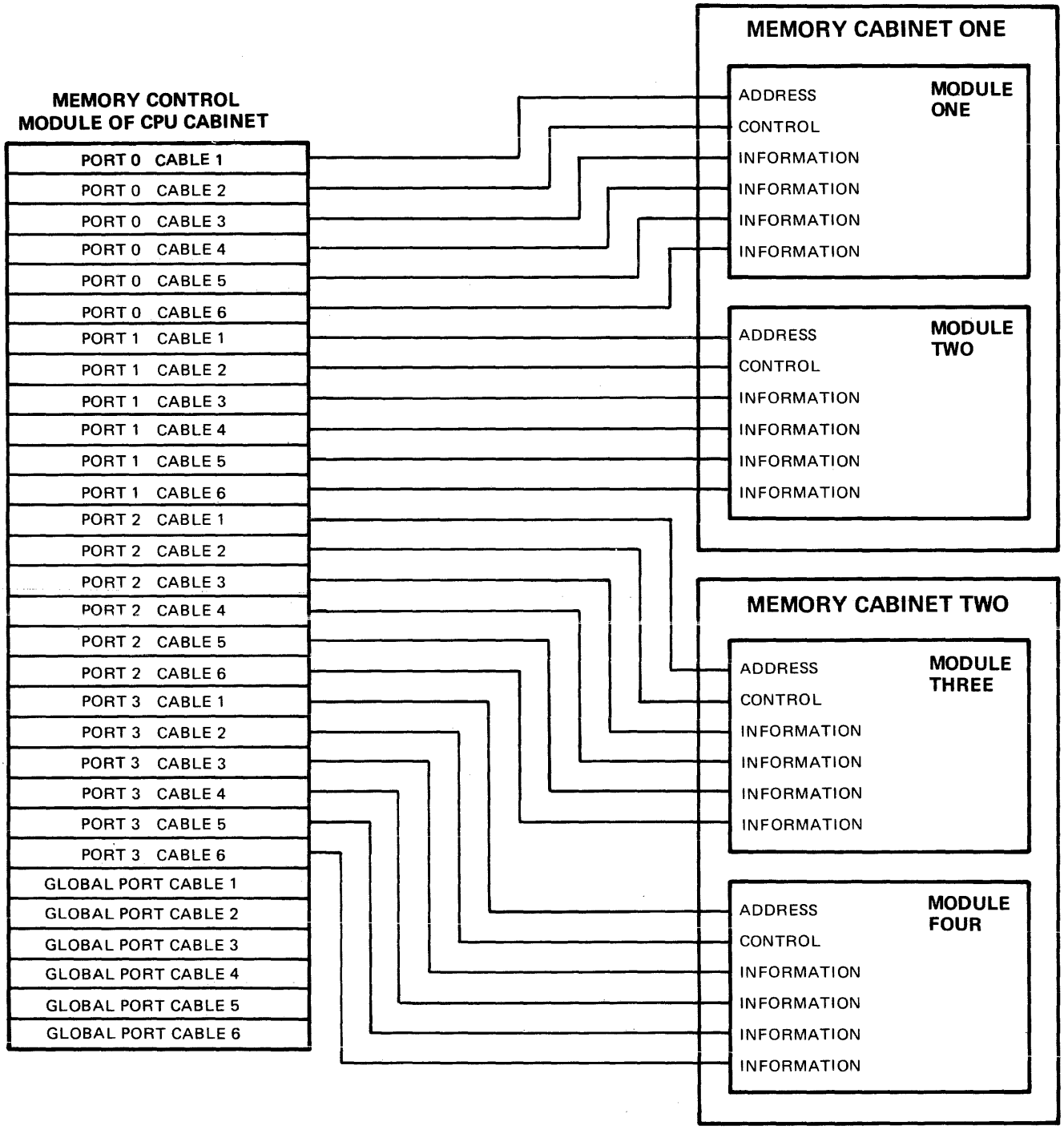


Figure 1-12. Memory Control Interface Bus

A memory read cycle is completed in 780 nanoseconds, as measured at the cable connection from the memory control interface. This cycle time is the minimum time that must occur between two consecutive Initiate Memory Cycle (IMC) pulses. A Read/Restore memory operation, or a Clear/Write memory operation may be performed in the time given for a memory read cycle. A Read/Modify/Write memory cycle requires 1180 nanoseconds memory cycle time because this operation requires that both a memory read, and a memory write function must be performed (two IMC pulses are required) to complete a memory cycle.

The planar core memory utilized in the B 6800 system is destructive read-out memory. That is, when information is read out of a memory core, the core is magnetized to contain a given specific polarity. Therefore; to preserve data in core memory, the read data must be written back into the address.

A Read/Modify/Write memory cycle accepts input data, and a memory address from the memory requestor. A memory cycle is performed on the address specified, and the data that is present at the address is made available to the memory requestor.

A read/modify/write operation in the memory control may be changed into a read/restore operation under either of the following conditions:

a. A protected memory operation is in progress, and the data in the word addressed by the read part of the read/modify/write operation determines that the memory protect bit (bit 48) is true. If this condition exists, the data read out of the memory address is re-written back into the same address, and the memory protect interrupt is detected by the memory control.

b. A parity error occurs during the read part of the read/modify/write operation. If this condition exists after a memory retry has been attempted, then the data with the parity error is re-written into the same address, and the memory parity error interrupt is detected by the requesting function.

If the memory control does not detect a memory protect interrupt, or a parity error interrupt during the read part of a read/modify/write operation, then the operation continues as follows.

The data that was accepted by the memory module is written into the same address from which the memory read operation was performed and thus, the original data is destroyed. The B 6800 system uses the Read/Modify/Write mode of operation to perform normal memory write functions.

A Read/Restore memory cycle accepts an address from the memory requestor, a read memory cycle is performed on the address specified, and the data that is present at the address is made available to the memory requestor. The same data that was present in the specified address is written back into the specified address. The B 6800 system uses the Read/Restore mode of operation to perform normal memory read functions.

A Clear/Write memory cycle accepts an address from the memory requestor, and writes a requestor supplied data word into the address. The changing of the clear/write operation into a read/restore operation, (for a parity error) is analogous to that change defined for the read/modify/write operation previously.

B 6800 OPERATORS DISPLAY CONSOLE (ref. figure 1-2)

The purpose of this console is to provide a position where all of the necessary system operating controls are collected in one physical place. The collection of the normal operating controls into a single central location is efficient, and provides a logical place for the system operational staff to function.

There are three parts to the operators display console (see figure 1-13), in addition to the table-top work area. The three parts of the console are the TD830 video display, the system control panel, and the keyboard for the video display. The video display terminal is recessed into the table-top in such a way that the display is visible without distortion (due to paralax) when the user of the display is either sitting, or standing. The system control panel is mounted flush with the table-top, and is located immediately in front of the recessed video display. The keyboard for the video display terminal is mounted at an angle immediately in front of the system control panel. The angle at which the keyboard is mounted complements the recess angle of the display terminal screen, such that the lettering on the keys of the keyboard are visible regardless of whether the user is sitting, or standing.

The operators display console contains two separate operator stations. Full control of the system is possible from either station of the console. A locking device is installed for each operators station. The locking device is a security feature used for system integrity. When the device is locked, the keyboard is disconnected, and the operators station cannot communicate with the software operating system. The locking device is activated by the use of a hand key that must be inserted into the lock, and turned to either open, or lock the operators console station keyboard. The locking device has no effect on the system control panel, and the controls on the panel may be operated without regard to whether the keyboard is locked, or not.

Figure 1-14 shows the operators system control panel details. This panel contains the operators controls for the video display portion of the TD830, in addition to the controls for operating the B 6800 system. As shown in figure 1-14, the controls for the video display are at the top of the control panel, and the controls for the B 6800 system are at the bottom of the drawing.

The controls for the video display consist of a thumbwheel type adjustment, and an ON-OFF switch for the video display. The purpose, and use of the video display controls are as follows:

a. The ON-OFF switch. This switch controls the power utilized by the video display.

b. The BRIGHTNESS thumbwheel controls the lighting intensity of the video display.

The controls for the B 6800 system consist of seven indicator/switch pushbutton controls shown at the bottom of the control panel, in figure 1-14. The purpose and use of the B 6800 system controls is as follows:

a. The ENABLE pushbutton switch allows the use of the HALT, POWER ON, and POWER OFF pushbutton switches. If the ENABLE pushbutton is not depressed then the three other pushbuttons listed are inoperative, and have no effect on System operation. If the ENABLE pushbutton is depressed then the other three pushbuttons listed are enabled, and depressing any one of the pushbuttons will cause the circuit corresponding to the switch to be activated. The purpose of the ENABLE pushbutton is to prevent accidental system operation caused by inadvertently depressing one of the pushbutton controls listed.

b. The POWER OFF pushbutton is used to remove source power from the circuits of the system that are supplied power from the central power supply cabinet. The POWER OFF pushbutton does not remove power from circuits that receive their source power from some other source.

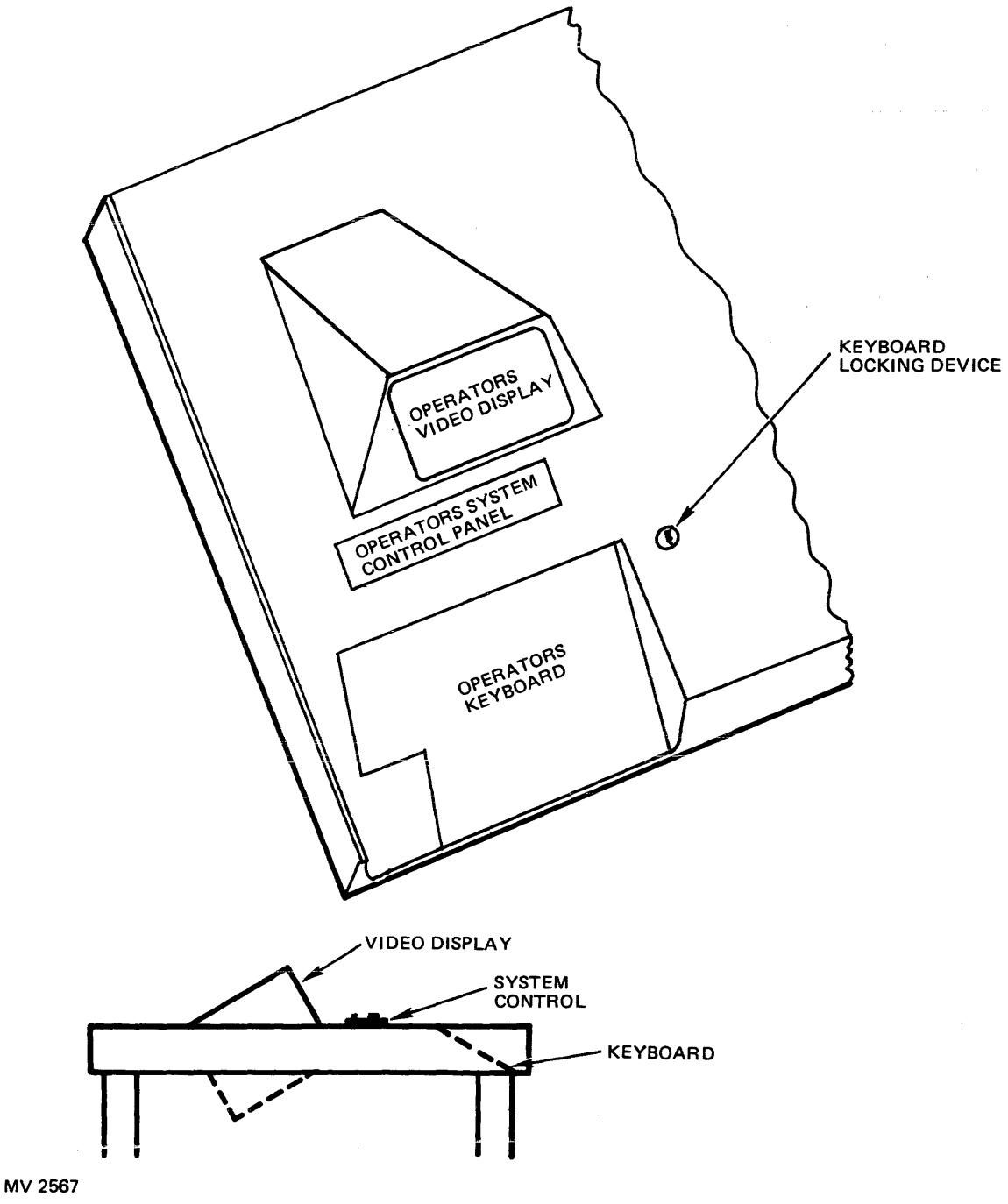
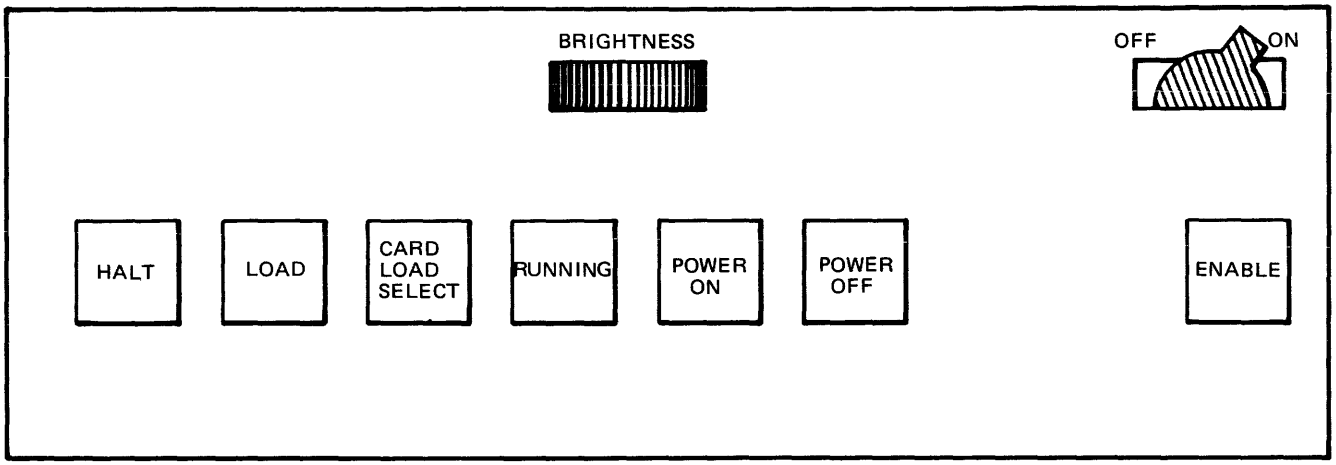


Figure 1-13. Operators Display Console



MV 2568

Figure 1-14. Operators System Control Panel

c. The POWER ON pushbutton is used to apply source power to the B 6800 system cabinets that derive their power input from the central power supply cabinet. The POWER ON pushbutton does not provide a method for applying source power to cabinets and peripheral units that do not derive their source power from the central power supply cabinet.

d. The HALT pushbutton is used to stop the B 6800 system at the end of the current machine language operator that is in process.

e. The LOAD pushbutton is used to cause the B 6800 system to initiate a Halt/Load sequence of operations. When the LOAD pushbutton is depressed the B 6800 system logic is general cleared (Set to the binary zero condition). When the pushbutton is released the Load operation is initiated. The Halt/Load sequence is a predetermined set of operations that results in the software operating system being placed in control of the system hardware.

f. The CARD LOAD SELECT pushbutton is used in conjunction with the LOAD pushbutton, to control the Halt/Load sequence of operations. If the CARD LOAD SELECT pushbutton is illuminated, and a system Halt/Load sequence is initiated (by depressing the LOAD pushbutton), then a Load operation proceeds from the card reader (or flex disk) peripheral device. If the CARD LOAD SELECT pushbutton is not illuminated when the LOAD pushbutton is depressed then the Load sequence proceeds to perform a load operation from the system disk (or pack) peripheral device. The selection of either a card reader device, or a system disk device from which to perform a system Load operation depends on whether the pushbutton is illuminated, or extinguished.

g. The RUNNING indicator lamp is illuminated when the system is operating. The purpose of the RUNNING indicator is to provide an indication of whether or not the system is capable of responding to certain stimuli during system operations. The reason why a RUNNING indication is necessary is that under certain conditions there is no other visible way to determine if the system is trapped in a perpetual operating loop.

Figure 1-15 shows the keyboard for the operators video display console. This keyboard is used by a system operator to input commands and data to the operating system. The operators display console and keyboard are commonly referred to as an Operators Display Terminal (ODT), and also as a Supervisory Printer Output (SPO).

When the security lock mechanism for system integrity is engaged, the keyboard is disabled, and has no effect on system operations. However, if the keyboard is disabled, but the video display switch (discussed previously in this subsection) is in the ON position, then the operating system will display status messages, and other pertinent data about current system operations.

The operators display video screen is used to pass communications between a human operator, and the operating software system of the B 6800 system. The display screen is similar to a home television receiver, except that the display screen can only display characters and numbers, and not pictures. The only sound that the display is capable of making is the bleep tone that is used to gain the operators attention when the software operating system needs a response from the operator.

When the operator needs to communicate with the operating system, the keyboard is used to write data which is displayed on the screen. The screen is capable of displaying 3200 characters, which are arranged in a matrix that consists of 40 rows of characters. Each row of characters contains 80 character positions. A cursor blinks at the position that the next character will occupy. If the next character position contains a valid character then the valid character blinks, but if the next character position is not occupied then the cursor illuminates the character position, and causes the illuminated position to blink. The cursor moves from left to right, and from top to bottom on the screen. The display screen has automatic line-feed, and carriage-return features so that the operator is not required to control these functions. When the operator writes data on the screen, the last character written is the End-Of-Text special character. This special character is used to indicate where the communication is to stop.

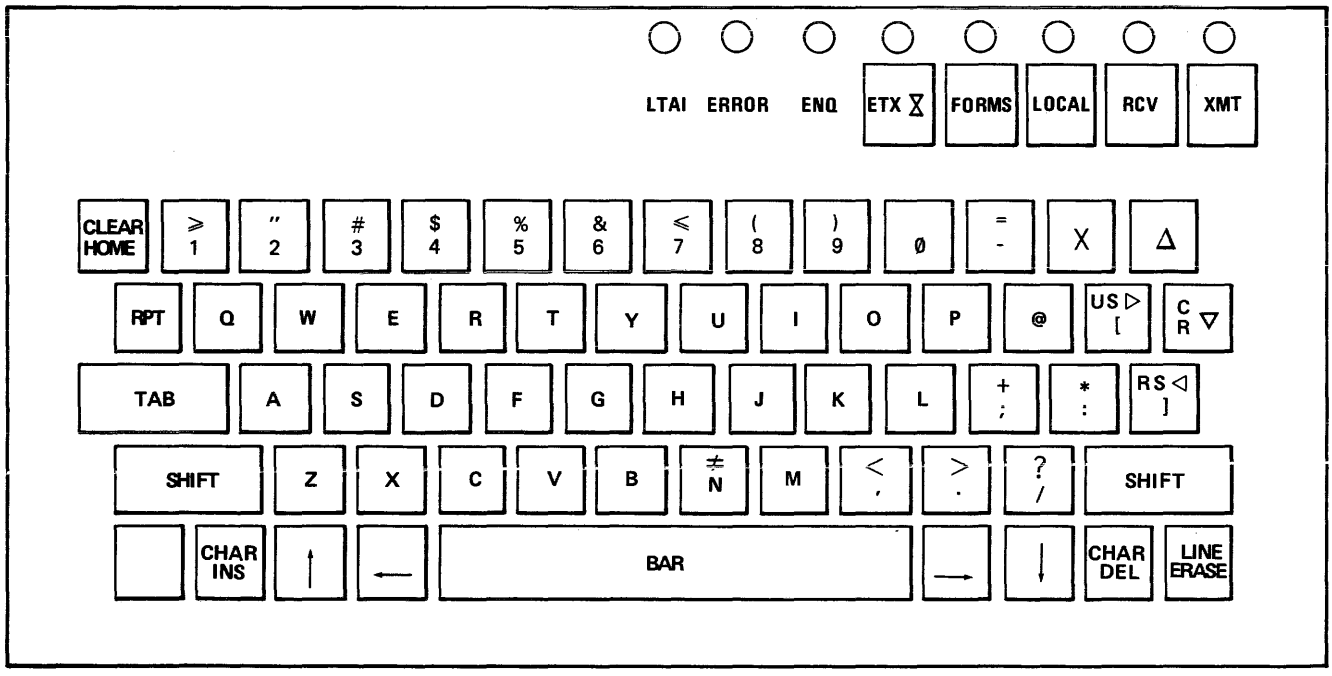


Figure 1-15. Operators Keyboard

B 6800 OPTIONAL UNITS

The B 6800 system may be expanded by adding optional subsystems to an existing B 6800 system. The optional subsystems that may be added to a B 6800 system are a data communications subsystem, and/or a bus interface control (Reader/Sorter control) subsystem. The following paragraphs will discuss these two optional subsystems, and will describe the manner in which these units are interfaced and controlled by the B 6800 system.

Figure 1-4 shows that the two optional subsystems are interfaced to the system through the use of the scan bus. In addition, figure 1-4 shows that the two subsystems, when used, are required to be independently powered cabinets.

As shown in figure 1-4, the CPU of the B 6800 system contains a scan bus interface capability, through which the optional units of the system communicate with the mainframe modules. This scan bus is essentially the same as the scan bus that is used in the B 6700 system, however; it is reduced in scope because the B 6800 system has a single interface port through which all of the units that use the scan bus must communicate.

The scan bus used in the B 6800 system consists of 80 lines which are used in the following ways:

a. 52 lines are used to transfer information between the mainframe of the B 6800 system, and the optional unit. One of these lines is used to transmit a longitudinal parity bit (odd parity) between the transmitting and receiving modules.

b. 20 lines are used to transfer a memory address field between the optional module, and the mainframe of the system.

c. Eight lines of the scan bus are used to control the operation, and direction of transfer of the data that passes through the scan bus. One of the control lines used for the scan bus is the scan transmission error line (STEX). The STEX signal is normally at a low (false) logic level.

   The STEX signal line is also used to transmit the Scan Address Parity Level (SAPL) signal on the scan bus. SAPL is used to cause the scan bus address to contain an odd parity. The STEX signal line is only used to transmit the SAPL signal during the first part of a scan bus operation (when the scan address is transmitted) and thereafter it is only used to transmit the STEX signal.

   During the transmission of the scan address the receiving module tests the parity of the address data received. If a parity error is detected the receiving module will cause the STEX signal to go from a false level to a true level. The memory controller logic of the CPU contains logic to detect a scan bus parity error condition (and therefore interrupt the scan bus operation) if this condition occurs.

   During the transmission of data through the scan bus the module that is receiving the data tests the parity of the data received. If a parity error is detected the receiving module will cause the STEX signal to go from a low (false) level to a high (true) level.

   The data processor samples the state of the STEX line for all communications on the scan bus. If the STEX line is a high (true) level the data processor will terminate the scan operation, and will initiate the interrupt controller to declare the scan error condition.

If the STEX line is true during data transmission on the scan bus, and the data processor is the transmitter module, the scan out alarm interrupt is sensed. The scan-in error is sensed if another module is the transmitter and the data processor is the receiver.

The system software is aware of scan bus failures through the initiation of the alarm type interrupt that is generated by the data processor.

All scan bus operations are initiated by the data processor module of the CPU. The data processor uses the scan bus to transmit command instructions to the units that are interfaced with the system through the scan bus interface.

The optional units that use the scan bus for system communications also have an interrupt line to the multiplexor module of the CPU. When the data processor has initiated some unit by the use of the scan bus the unit that was initiated proceeds to perform its function until the function is completed, or until it generates an interrupt. Upon finishing a commanded function, or upon encountering an interrupt, the unit that was initiated through the use of the scan bus will cause its interrupt line to the multiplexor to become a true level. The multiplexor will identify the optional unit that caused its interrupt line to be a true level, and will interrupt the data processor. In this way, the software operating system (through the scan bus, and the interrupt lines to the multiplexor) is aware of the operating conditions of the optional units in the system. The data processor, upon receiving an interrupt from the multiplexor, will interrogate the unit that caused the interrupt, through use of the scan bus. The unit that caused the interrupt will respond to the interrogation of the data processor by providing its status to the CPU through the scan bus. In this manner, the software operating system controls the operations of the optional units of the system.

All of the optional subsystems that are connected to the B 6800 system share a single memory bus requester path. If more than a single optional subsystem is connected to the B 6800 system, then the units that are connected must contend for access to the memory resources of the system. In addition, the optional units requestor path of the memory control exchange module has the lowest priority of the three requester paths. Both the data processor/multiplexor requester path, and the look ahead requester path have a higher priority to the memory resources of the system than does the optional subsystem path.

As was stated previously, the memory control exchange of the CPU will not perform memory retries for the optional subsystem requester port. However, in the event that a request from the optional subsystem requester port results in an error being detected in the read data that is fetched from memory, then the memory control exchange will perform an error correction cycle upon the data.

The memory bus, through which optional subsystems access memory resources of the system, is an 80 line bus. This bus is the same as the memory bus used in the B 6700 system, and the 80 lines are used in the same way that the scan bus lines are used. The scan bus lines were discussed previously in this subsection of this manual.

SECTION 2

DATA REPRESENTATION

GENERAL

All data in the B 6800 System is in binary form. The basic unit of data is the word, (see figure 2-1) which consists of 52 consecutive binary bits. All words of data in the B 6800 system have three distinct parts which are; a parity bit, a tag field, and the information field. The 52 bits in a word are numbered for identification.

Bit number 51 (the most significant bit in a word) is the parity bit. The parity bit is used to represent the odd parity of the word. If the number of binary ones present in the tag field, and the information field is an even number then the parity bit is a binary one value. If the number of binary ones present in the tag field, and the information field is an odd number, then the parity bit is a binary zero value. The B 6800 system uses the parity bit to monitor the quality of data in a word. Logic circuits in the B 6800 system count the number of bits in a word, and compare the count against the parity bit state. If the result of the comparison is not equal, then the B 6800 system recognizes that a parity error has occurred. The process of parity checking is an automatic feature of the B 6800 system. The parity bit for a word is not directly available to the user of the system because it is only used when words are transferred from one module to another. Data that is internal to a module has already been tested for parity.

Bits 50, 49, and 48 are the tag field. The tag field is used to identify the type of interpretation that is to be applied to the data that is present in the information field of the word. There are eight different values that may be present in the tag field, and each value specifies a different interpretation to be used. The meaning of the tag field values are as follows:

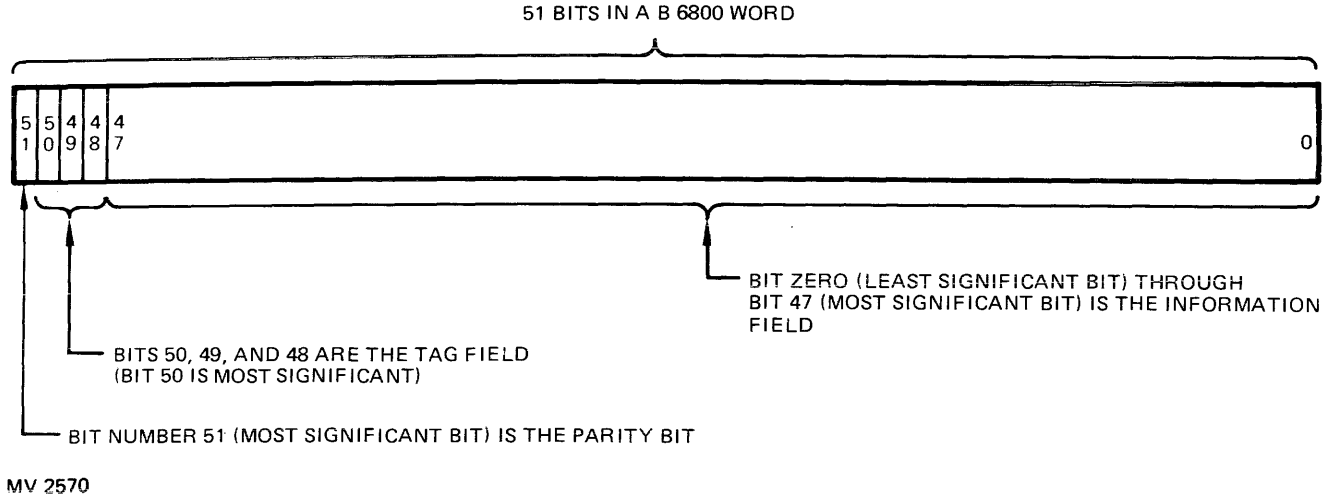


Figure 2-1. B 6800 Word Structure

TAG (50)	FIELD (49)	BITS (48)		MEANING
0	0	0	–	A tag field of zero indicates that single-precision data is present in the information field of the word.

TAG (50)	FIELD (49)	BITS (48)	MEANING
0	0	1	– A tag field of one indicates that the data in the information field is an indirect address, not data.
0	1	0	– A tag field of two indicates that double-precision data is present in the information field of the word.
0	1	1	– A tag field of three indicates that a control word is present in the information field of the word. There are several different types of control words used in the B 6800 system. These types of control words are discussed individually, later in this section of this manual.
1	0	0	– A tag field of four normally indicates that a step index word is present. The meaning and use of a step index word is discussed later in this section of this manual.

NOTE

A special use for a word that has a tag of four may be invoked by the MCP when a fault condition is to be handled by a user program.

The compiler will place a word with a tag of four in the stack as a flag word. This flag is used to indicate that the program using the stack is responsible for handling one or more of the interrupts that may occur when the program is executed.

This special use for a word with a tag field of four is only invoked when the programmer of the user program specifies that the user program is responsible for interrupt handling. The compilers that utilize this special case are the ALGOL, FORTRAN, ESPOL, and the PL/I compilers.

1	0	1	– A tag field of five indicates that a descriptor word is present. The meaning and use of a descriptor word is discussed later in this section of this manual.
1	1	0	– A tag of six indicates that a software control word is present. The meaning and use of a software control word is discussed later in this section of this manual.
1	1	1	– A tag of seven indicates that a program control word is present. The meaning and use of a program control word is discussed later in this section of this manual.

This manual uses a convention to refer to data bits in a word. The rules of this convention follow:

a. A data field within a word is represented by two numbers, separated by a colon character, and enclosed in brackets.

b. The meaning of the two numbers enclosed in the brackets is as follows:

   1. The first (left-most) number identifies the most significant bit in the field of data bits.

   2. The second (right-most) number identifies the number of bits that are contained in the field of data bits (including the most significant bit, which was identified in rule b1 above).

c. Bits in the tag field are not included in the field unless the most significant bit (rule b1 above) is one of the tag field bits.

d. All bits in the information field are considered to "wrap-around" the word in such a way that the next least significant bit after bit zero is bit 47.

Examples of this convention are as follows:

Bits [50:3] (the tag field)	–	Beginning with bit 50 for three bits, or bits 50, 49, and 48.
Bits [06:9] (a data field)	–	Beginning with bit 06 for 9 bits, or bits 06, 05, 04, 03, 02, 01, 00, 47, 46.
Bits [47:48] (a data field)	–	Beginning with bit 47 for 48 bits, or all of the information field.

The convention that was stated in the previous paragraph is used to further define the bits that make up the information field of the B 6800 system words. There are 48 bits in this field, of which bit 47 is the most significant bit, and bit zero is the least significant bit.

INTERNAL CHARACTER CODES

The B 6800 uses several different character codes (see figure 2-2). The primary internal code that is used is Extended Binary Coded Decimal Interchange Code (EBCDIC). EBCDIC is an 8-bit alphanumeric code containing four zone bits, followed by four numeric bits. Another important internal code that is used in the B 6800 system is the Burroughs Common Language code (BCL). BCL is a 6-bit alphanumeric code containing two zone bits, followed by four numeric bits. The primary character code used for Data Communications Subsystems is the American Standard Code for Information Interchange (ASCII). ASCII may be either a 6-bit, 7-bit, or 8-bit alphanumeric code. Within the B 6800 system, EBCDIC, or BCL codes may be compacted by deleting the zone bits, and retaining the numeric portion of the character. When data in the B 6800 system is compacted it is said to be packed.

Appendix C of this manual lists the character codes of the character sets that are used in the B 6800 system. Appendix D. gives the card codes that are required to produce an EBCDIC, or hexadecimal coded character representation.

NUMBER BASES

Number bases used in the B 6800 system are base 10 (decimal), base 16 (hexadecimal), base 2 (binary), and base 8 (octal) (see figure 2-2). Because the system utilizes various of these number bases in performing its functions, it is necessary that the user of the system be familiar with the number bases, and know how to convert a value from one number base to any of the other number bases. A brief discussion of the number systems used follows.

The decimal numbering system is based on the numeric digits zero through nine, and on the powers of ten. Similarly, the binary numbering system is based on the numeric digits zero and one, and on the powers of two. In the case of

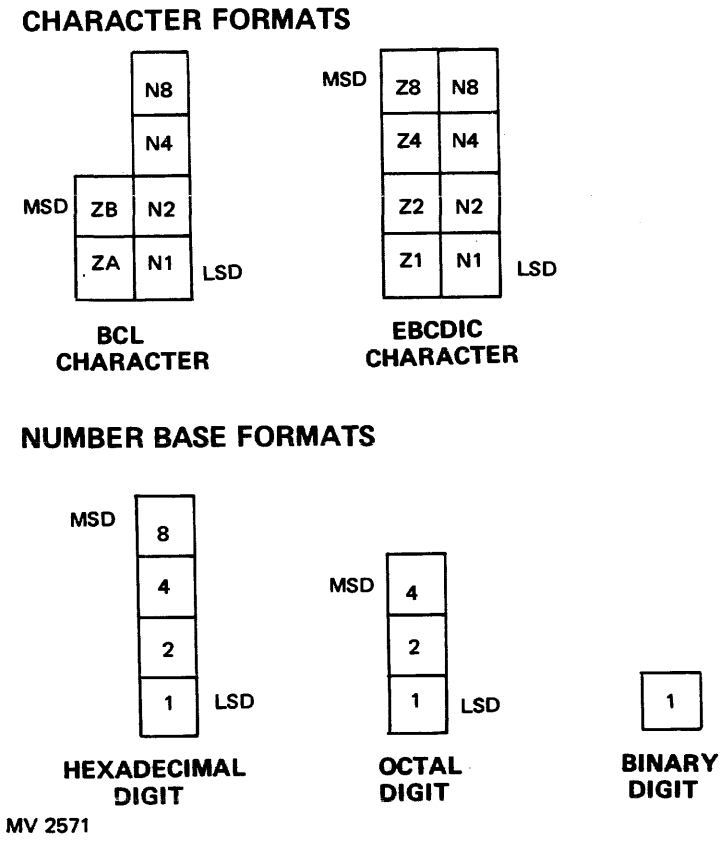


Figure 2-2. Character and Digit Formats

the numbering systems described above, it is apparent that a decimal digit may have any value from zero through nine, and that a binary digit may have either a value of zero, or one.

The octal numbering system is based on the numeric digits zero through seven, and on the powers of eight. An octal digit may have any value from zero through seven. Further, two raised to the third power is eight, the base of the octal numbering system. Therefore, because the octal numbering base is a multiple of the binary number base, an octal number can be converted to a binary number conveniently, and vice versa.

The hexadecimal numbering system is based on the numeric digits zero through nine, and A through F; where A equals decimal 10, B equals decimal 11, C equals decimal 12, D equals decimal 13, E equals decimal 14, and F equals decimal 15. Hexadecimal numbering is also based on the powers of sixteen. Two raised to the fourth power is sixteen, the base of the hexadecimal numbering system. Therefore, because the hexadecimal numbering base is a multiple of the binary numbering base, a hexadecimal number can be converted to a binary number conveniently, and vice versa.

A B 6800 word contains 48 bits in the value field of the word (refer to figure 2-3). These 48 bits can be converted into hexadecimal, octal, BCL, or EBCDIC values by arrangement of the 48 bits in the proper order. A hexadecimal digit is equivalent to four binary digits because 1111 binary is equal to hexadecimal F. Since a hexadecimal digit contains four binary digits, the value field of a B 6800 word contains 12 complete hexadecimal digits (48/4 = 12). The same value field can also be considered to contain 16 octal digits (48/3 = 16), or 6 EBCDIC characters (48/8 = 6). BCL character code (6-bit data) also converts into an equal number of binary digits, and a B 6800 word can contain 8 BCL characters (48/6 = 8).

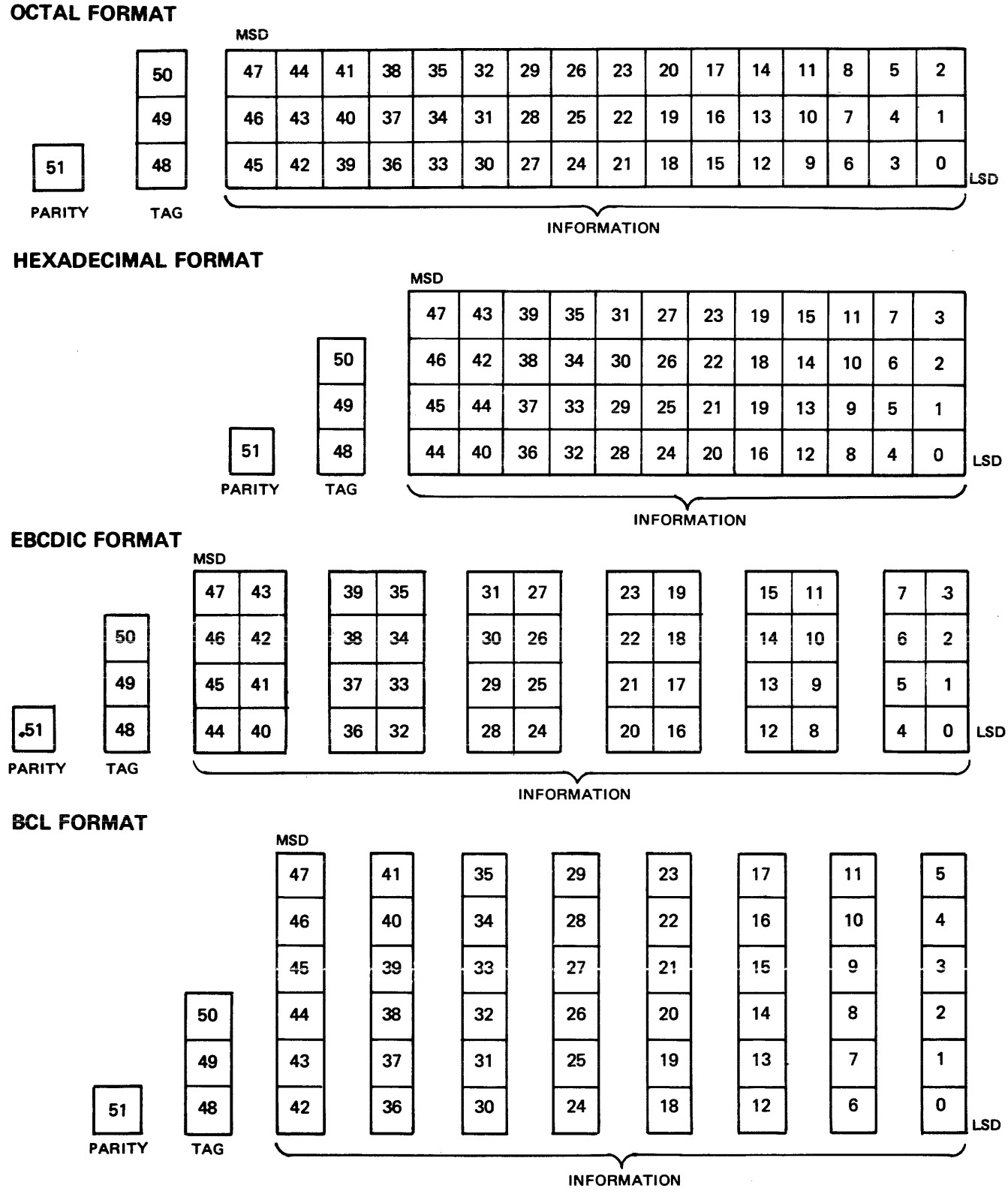


Figure 2-3. B 6800 Word Formats

From the foregoing discussion it is clear that the choice of 48 bits for the value field of a B 6800 word was not a random choice, but rather was chosen because that number is a multiple of the common character codes, and number bases used in the B 6800 System.

NUMBER CONVERSION

The B 6800 system normally converts decimal data that is input to the system from decimal notation to EBCDIC or BCL codes. An exception to this normal mode of operation may occur in the case of the data communications subsystem where input data may be in ASCII coded form. It is also possible to find that the input data has been packed, and is thus in hexadecimal notation in the System. The user of the system must be familiar with the forms in which the data can be stored. The user must be able to perform manual conversion of numeric data from one form to another so that the internal data conversion processes can be assessed for proper operation. The following paragraphs will present methods for performing manual conversion of numeric data from one form to other forms.

DECIMAL TO NONDECIMAL

Decimal numeric data is converted from base 10 to some other number base by repeatedly dividing the decimal value by the base number for the numbering system to which it is to be converted. Each time a division is performed, the remainder becomes the next most significant digit, or bit in the new number base. When no more whole numbers occur during the division the conversion is complete.

EXAMPLES:

a. Convert the decimal number 1776 to octal (base 10 converted to base 8).

1776/8 = 222 with a remainder of 0;

222/8 = 27 with a remainder of 6;

27/8 = 3 with a remainder of 3;

3/8 = 0 with a remainder of 3.

1776 decimal = 3360 octal.

b. Convert the decimal number 1776 to hexadecimal (base 10 converted to base 16).

1776/16 = 111 with a remainder of 0;

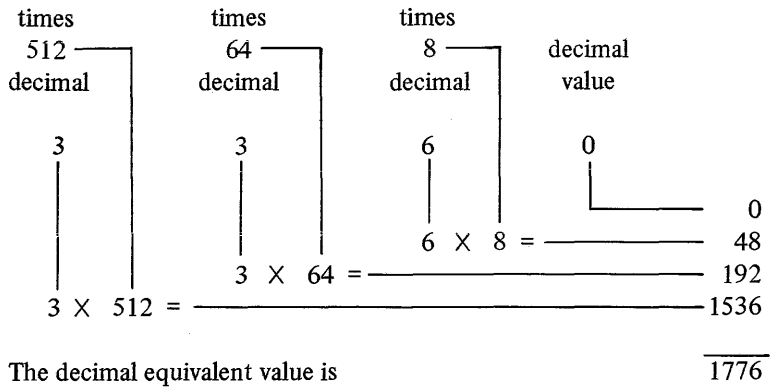
111/16 = 6 with a remainder of 15 F (15 decimal = F hex);

6/16 = 0 with a remainder of 6.

1776 decimal value = 6F0 hexadecimal.

NONDECIMAL TO DECIMAL

Nondecimal numeric data is converted to decimal data by multiplying each digit of the numeric value by the value of the digit position, in decimal values. For example, in the preceding subsection of this manual the decimal number 1776 was converted to octal, and hexadecimal notation. The successively more significant digits of the octal notation are as follows.



By the same logic, a hexadecimal number is converted to decimal as follows:

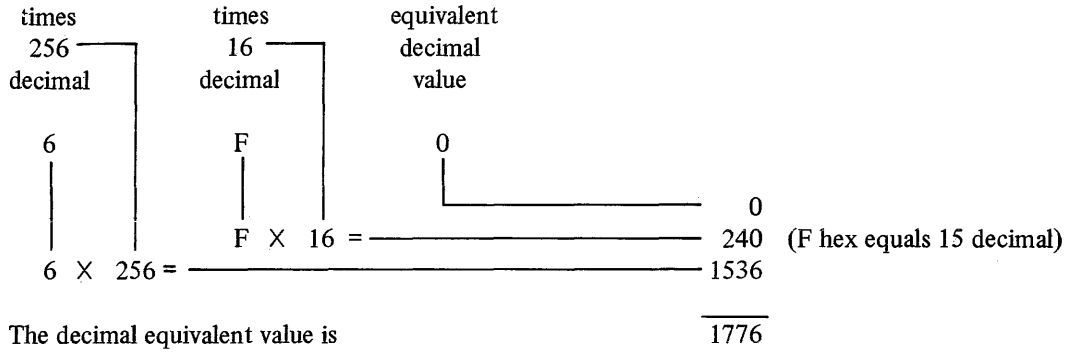


Table 2-1 gives the value of each succeeding digit in a number. These values are provided for binary, octal, and hexadecimal digit positions. The values in this table are expressed in decimal equivalents for the corresponding digit positions. There are 16 octal digits in a B 6800 word (refer to figure 2-3) and therefore, table 2-1 gives the place values for 16 octal digits. A B 6800 word contains 12 hexadecimal digits, and therefore table 2-1 gives the place values for 12 hexadecimal places.

Observing table 2-1 while again reading the examples of converting a nondecimal value to a decimal value will show the origin of the place values that were used to perform the multiplication portions of the examples. The sum of the multiplications provides the decimal values of the nondecimal numbers used in the examples.

NONDECIMAL TO NONDECIMAL

It is occasionally necessary to convert a hexidecimal number to an octal number, or vice versa. The easiest way to perform this conversion is to first convert this binary value to the final form.

EXAMPLE:

Convert the hexadecimal value ABCDE to octal notation.

a. Convert hexadecimal ABCDE to binary form as follows:

An A in the fifth position is 1010 in binary form
A B in the fourth position is 1011 in binary form

Table 2-1. Decimal Place Values of Digits in Various Number Bases

Digit Place	Binary Number Place Value	Octal Number Place Value	Hexadecimal Number Place Value
1	1	1	1
2	2	8	16
3	4	64	256
4	8	512	4096
5	16	4096	65536
6	32	32768	1048576
7	64	262144	16777216
8	128	2097152	268435456
9	256	16777216	4294967296
10	512	134217728	68719476736
11	1024	1073741824	1099511827776
12	2048	8589934592	17592189244416
13	4096	68719476736	
14	8192	549755813888	
15	16384	4398047311104	
16	32768	35184378488832	
17	65536		
18	131072		
19	262144		
20	524288		
21	1048576		
22	2097152		
23	4194304		
24	8388608		
25	16777216		
26	33554432		
27	67108864		
28	134217728		
29	268435456		
30	536870912		
31	1073741824		
32	2147483648		
33	4294967296		
34	8589934592		
35	17179869184		
36	34359738368		
37	68719476736		
38	137438953472		
39	274877906944		
40	549755813888		
41	1099511827776		
42	2199023655552		
43	4398047311104		
44	8796094622208		
45	17592189244416		
46	35184378488832		
47	70368756977664		
48	140737513955328		

A C in the third position is 1100 in binary form
A D in the second position is 1101 in binary form
An E in the first position is 1110 in binary form

The binary representation for the hexadecimal value is

1010 1011 1100 1101 1110.

b. Convert the binary value from step a above, to octal notation as follows:

10 101 011 110 011 011 110

2 5 3 6 3 3 6

Thus, the octal equivalent for the hexadecimal value ABCDE, is 2536336. Reversing the procedure of the preceding example converts the octal value to hexadecimal notation.

The example shown works well when the present form of the value to be converted to another form is relatively small. However, it can be seen that a five digit hexadecimal number converts into a twenty digit binary number (as in the preceding example), and from this it is evident that larger hexadecimal numbers will become long strings of binary digits. Extremely long strings of binary digits are cumbersome, and become awkward in performing the conversion. Another method that may be used to perform conversions in this case is as follows:

EXAMPLE:

Convert the hexadecimal value ABCDE to octal notation.

a. Using the values in table 2-1, convert the hexadecimal number to its equivalent decimal value, as follows:

(1) The value of the fifth position in a hexadecimal number (from table 2-1) is 65,536. The fifth position of the value to be converted is hexadecimal A (A hexadecimal is equal to 10 decimal). Therefore, the hexadecimal A in the fifth position is equal to 10 times 65, 536, or 655,360 decimal.

(2) The fourth position of a hexadecimal number has a value of 4,096 (from table 2-1). The fourth position of the hexadecimal number to be converted is B (hexadecimal B is equal to 11 decimal). Decimal 11 times 4,096 is equal to 45,056.

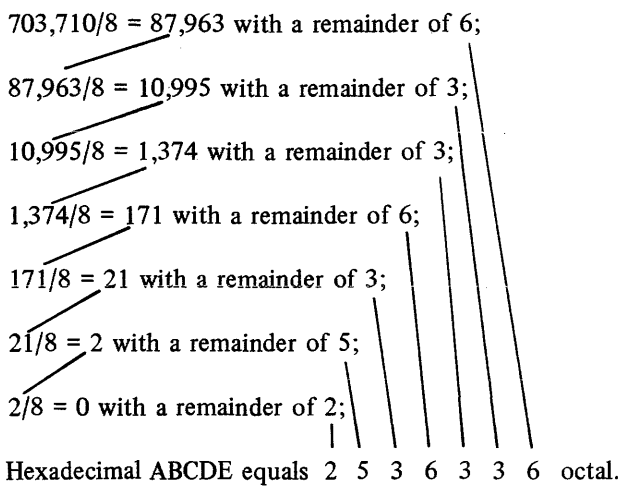
(3) Hexadecimal C times 256 decimal is equal to 3,072.

(4) Hexadecimal D times 16 decimal is equal to 208.

(5) Hexadecimal E is equal to 14 decimal.

655,360	hexadecimal Annnn
45,056	hexadecimal nBnnn
3,072	hexadecimal nnCnn
208	hexadecimal nnnDn
14	hexadecimal nnnnE
703,710	hexadecimal ABCDE equals 703,710 decimal

b. Convert the decimal number 703,710 (from step a above) to the equivalent octal value, as follows:



The procedure for converting nondecimal numbers to nondecimal numbers shown in the preceding example can also be used to convert an octal number to a hexadecimal equivalent. The only difference is that the place values from table 2-1 (used in step a of the procedure) must be taken from the octal column instead of from the hexadecimal column.

WORD TYPES AND PHYSICAL WORD LAYOUTS

It was stated previously in this section of this manual that a B 6800 system word consisted of a parity bit, a tag field, and an information field. The tag field defines an interpretation that is to be applied to the contents of the information field. This subsection of this manual will define the interpretations that are to be used for the data in the B 6800 system, and will present the format of data in the information field of each type of word used in the B 6800 system.

There are two types of data used in the B 6800 system, which are character strings, and operands. The following paragraphs will define character strings, and operands.

CHARACTER TYPE WORDS

Character type words are used to contain character strings. A character type word has a tag field of zero (a single precision word) and contains EBCDIC, BCL, or hexadecimal coded data. A string may occupy more than a single word of character data. However, a string must have at least one character type word.

The most significant character in a character string occupies the left-most character position, in the first character word of the string. Each word in a character string will contain 6 EBCDIC character, or 8 BCL character, or 12 hexadecimal character positions. The final word in a character string may contain less than a full word of characters if the number of characters in the string is not a multiple of the number of characters in a full word. Figure 2-4 through 2-6 show the various formats that are used for character type words.

OPERANDS

Operands are words of data that are used to contain numeric values, or logical information. An operand may be either a single precision word (tag field of zero), or a double precision word (tag field of two). Single, and double precision words are used for mathematical operations. Logical information is used for decision making processes, and operations. The following paragraphs will discuss the uses of operands in the B 6800 system.

		A	A	B	B	C	C	D	D	E	E	F	F
	0	A	A	B	B	C	C	D	D	E	E	F	F
	0	A	A	B	B	C	C	D	D	E	E	F	F
P	0	A	A	B	B	C	C	D	D	E	E	F	F

P = WORD PARITY VALUE
0 = BINARY ZERO VALUES (TAG FIELD)
A ⇒ F 6 EBCDIC CHARACTER FIELDS
A IS THE MOST SIGNIFICANT CHARACTER

MV 2573

Figure 2-4. EBCDIC Character Word Format

		A	A	B	C	C	D	E	E	F	G	G	H
	0	A	A	B	C	C	D	E	E	F	G	G	H
	0	A	B	B	C	D	D	E	F	F	G	H	H
P	0	A	B	B	C	D	D	E	F	F	G	H	H

P = WORD PARITY VALUE
0 = BINARY ZERO VALUES (TAG FIELD)
A ⇒ H 8 BCL CHARACTER FIELDS
A IS THE MOST SIGNIFICANT CHARACTER

MV 2574

Figure 2-5. BCL Character Word Format

		A	B	C	D	E	F	G	H	J	K	L	M
	0	A	B	C	D	E	F	G	H	J	K	L	M
	0	A	B	C	D	E	F	G	H	J	K	L	M
P	0	A	B	C	D	E	F	G	H	J	K	L	M

P = WORD PARITY VALUE
0 = BINARY ZERO VALUES (TAG FIELD)
A ⇒ M 12 HEXADECIMAL CHARACTERS
A IS THE MOST SIGNIFICANT CHARACTER

MV 2575

Figure 2-6. Hexadecimal Character Word Format

Single Precision Operand

A single precision operand is a numeric value that has an exponent part, and a mantissa part. Figure 2-7 shows the format for a single precision operand. The fields in a single precision operand are as follows:

bits [50:3] are the tag field, and are always equal to zero for a single precision operand.

bit 47 bit 47 is not used in a single precision operand.

bit 46 bit 46 is used as the sign of the mantissa field. If the sign bit is a binary one then the mantissa field contains a negative value, and if bit 46 is a binary zero then the mantissa contains a positive value.

bit 45 bit 45 is used as the sign of the exponent field. If the sign bit is a binary one then the exponent field contains a negative value, and if bit 45 is a binary zero then the exponent contains a positive value.

bits [44:6] are the exponent field. Bit 44 is the most significant bit in the exponent value. The value of the bits in this field are as follows:

bit 39 value is decimal one

bit 40 value is decimal two

bit 41 value is decimal four

bit 42 value is decimal eight

bit 43 value is decimal sixteen

bit 44 value is decimal thirty-two

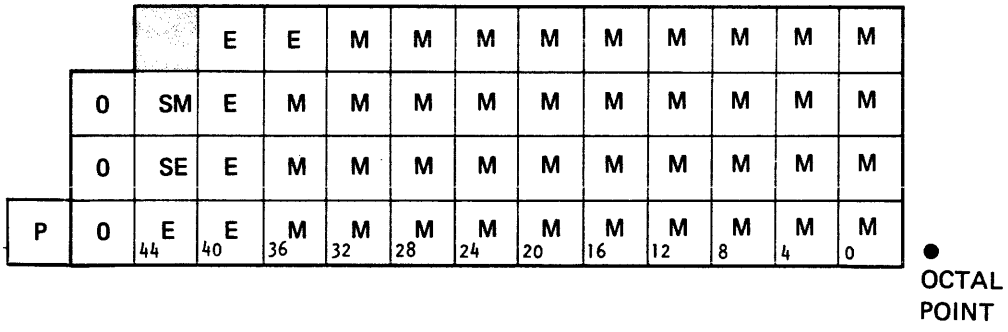


Figure 2-7. Single Precision Operand Format

The maximum value that the exponent field can contain is decimal 63. When the exponent is used in conjunction with the exponent sign bit (45), the range of the exponent value is from +63, to -63 decimal.

bits [38:39] are the mantissa field. Bit 38 is the most significant bit in the mantissa value. The mantissa is divided into thirteen octal fields of which bits [38:3] are the most significant octal digit, and bits [2:3] are the least significant digit.

An octal point (similar to a decimal point) is always located to the right of bit zero, in the mantissa field. This point is not displayed in any way and must be assumed to exist.

The software of the B 6800 system classes numeric data into two classes, namely, INTEGER, and REAL. An INTEGER number is a single precision or double precision numeric value with an exponent value of zero. The maximum value that an INTEGER may have in the B 6800 system is +7777777777777 octal, or 549,755,813,887 decimal. The minimum integer value is -7777777777777 octal. A REAL numeric value is any value that has an exponent that is not equal to zero, or any value that contains a part value (contains a decimal, or octal point prior to the least significant digit of the value). From the format given for a single precision operand it is evident that REAL numbers may not qualify to be expressed as single precision values. For this reason, REAL numbers are always expressed as double precision values (with floating point) in the B 6800 system.

Double Precision Operand

A double precision value is two consecutive words, with a tag field of two (010 binary). The two words are concatenated in such a way that they form a single numeric value, with an octal point located between the two words. The most significant part of the mantissa in a double precision operand is commonly referred to as the most significant part (MSP) and the least significant part of the mantissa is commonly referred to as the least significant part (LSP). The octal point that separates the MSP from the LSP is used to separate whole values from partial values, with whole values present in the MSP, and partial values present in the LSP. The format for the MSP of a double precision operand is identical with the format for a single precision operand, except for the tag field. The LSP of a double precision operand is an extension of the exponent field, and the mantissa field contained in the MSP of the word. Figure 2-8 shows the word format for a double precision operand.

The largest double precision value (type REAL) that can be contained in a B 6800 is 1.94882938205028079124469, with an exponent value of +29603. The smallest double precision value (type REAL) that can be contained in a B 6800 is 1.9385458571375858335564, with an exponent value of -29581. The value zero, and positve or negative values between the largest and smallest values given above may be represented in double precision numbers in the B 6800 system.

When a double precision value is used the exponent extension field is an extension of the high order end of the exponent field in the upper-half word. Bit 39 in the LSP word is the next bit in sequence after bit 44 of the upper-half, and has a binary value of 64. Bit 40 in the LSP word is the next bit in sequence after bit 39 of the word, and has a binary value of 128. This same order is used for all of the bits in the LSP exponent extension field, so that bit 47 of the LSP becomes the most significant bit in the exponent value. The whole exponent field in a double precision operand is as follows:

MSP bit 39 is the least significant bit of the exponent, and has a value of 1, decimal.

LSP bit 39 is the next most significant bit in the exponent, and has a value of 64, decimal.

bit 47 is the most significant bit in the exponent, and has a value of 16384, decimal.

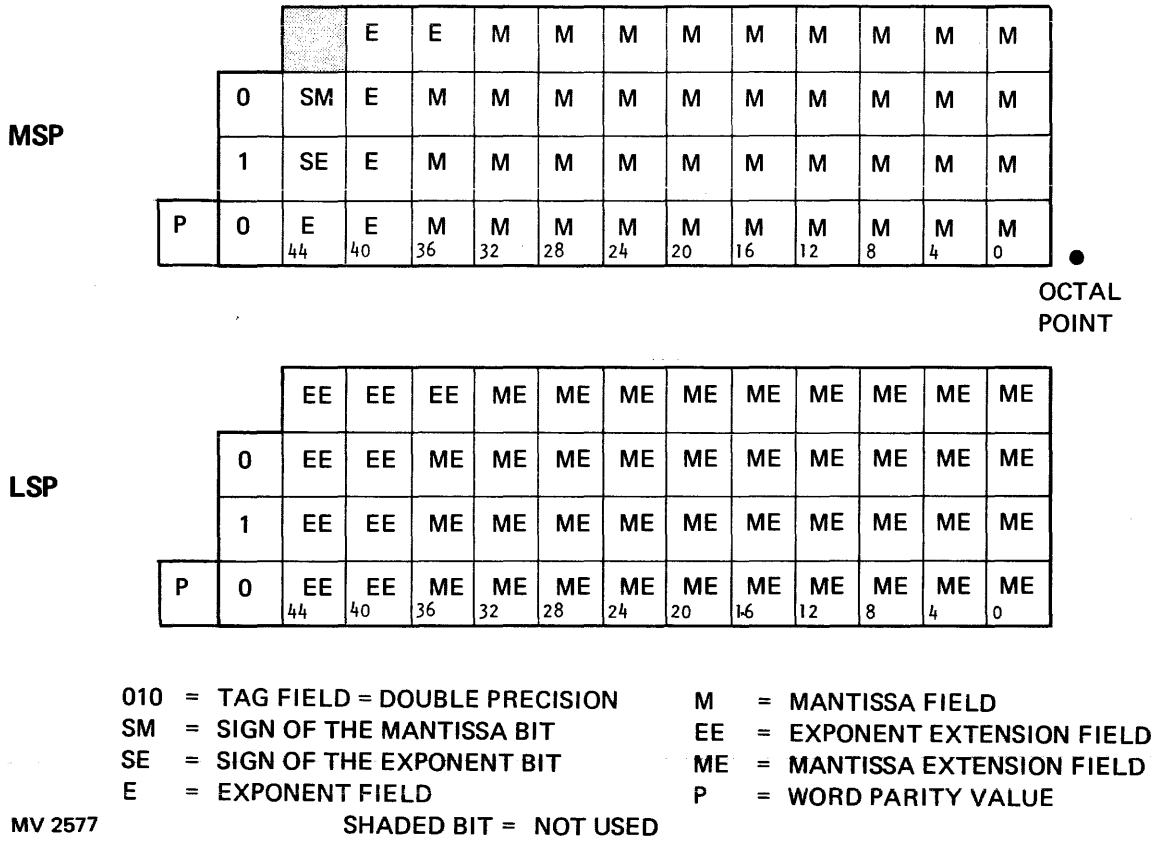


Figure 2-8. Double Precision Operand Format

The maximum size of an exponent in the B 6800 system is 32,767 decimal, and the range of the exponent field is from +32,767, to -32,767 decimal.

The mantissa extension field in the LSP of the double precision operand contains that portion of the mantissa that is less than unity. The mantissa extension field is divided into 13 octades, in the same manner as is the mantissa field in the MSP of the double precision operand. These octal digits are arranged in the same way as the octal digits in the MSP of the word. The least significant octade of the mantissa extension field is bits [2:3], and the most significant octade is bits [38:3].

The B 6800 system utilizes two processes known as integerization, and normalization, in performing mathematical operations. Normalization is a process that removes leading zeroes from a single or double precision word. This process is used to make the operation of the adder logic circuits more efficient. Integerization is a process that alters the value of a number such that it meets the requirements of an integer, as was defined previously in this section of this manual.

Normalization is accomplished by adjusting the value of the exponent field of a number in a positive direction until it is at the maximum value for an exponent, or until there are no leading zeroes in the mantissa of the number. Each time the exponent is incremented, the mantissa is shifted one octade to the left. There are no more leading zeroes in a mantissa when the most significant octade of the mantissa is located in bits [38:3] (of the upper-half word).

The process of integerization is a two step process. The first step is to adjust the exponent in either a positive, or a negative direction until the exponent field is equal to zero. Each time the exponent is incremented or decremented, the mantissa is shifted one octade in the corresponding direction. Octades that fall out of the low order digit of the mantissa during the adjustment of the exponent are saved until the exponent is equal to zero. After the exponent has been adjusted to zero, then that part of the mantissa that is less than unity (located to the right of the octal point) is either rounded upward to the next whole number, or it is truncated (deleted from the number). The process of rounding, or truncation is selective in the B 6800 system, and is the second step of the integerization process.

The mathematical operations that are performed in the B 6800 system can be completed regardless of the format of the operands used. If an arithmetic operation is performed using two single precision operands then the result of the operation will be in the single precision format. If, however, either operand is in the double precision format then the result of the operation will be in the double precision format.

Logical Operands

Logical operands are words that result from the performance of either a relational operation, or a logical (boolean) operation. A relational operation is one that determines the relative merits of two values by means of a comparison process. A logical operation is one that constructs a result based on the relative merit of each bit in the word when compared to the corresponding bits in another word.

A relational operation results in either a true, or a false answer. The answer is true if the result of an algebraic comparison of two arithmetic values is valid. The answer is false if the result of the algebraic comparison of the two arithmetic values is not valid. The B 6800 constructs a single precision logical operand (tag field equal to binary zero) each time that a relational operation is performed. If the answer is valid bit zero is a one in the logical operand, and if the answer is not valid then bit zero is a zero. All other bits in the answer word logical operand are not used, and are zeroes.

A logical (boolean) operation results in the construction of a different type of logical operand. The constructed logical operand may contain a number of bits. The reason for this is that a logical operation looks at each bit in two different words, and places a corresponding bit in the result operand if the conditions of the logical operation are satisfied.

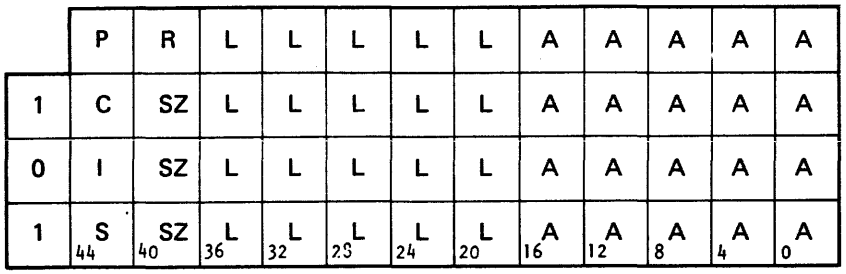
Logical operands are discussed later in this manual.

DATA DESCRIPTORS

Data descriptor words refer to data areas, including input/output buffer areas. The data descriptor defines an area of memory starting at the base address contained in the descriptor. The size of the memory area in words is contained in the length field of the descriptor. Data descriptors may directly reference any memory word address from word number zero through word number 1, 048, 576. The structure of the data descriptor word is illustrated in figure 2-9. The fields in the data descriptor are as follows:

bits 50:3 — Bits 50, 49, and 48 are the tag field, and are always equal to a binary value of 101.

bit 47 — Bit 47 is the presence bit. The presence bit is used to indicate whether or not the information described by the data descriptor is present in main memory. If the presence bit is equal to a binary one then the data is present in main memory. If the presence bit is equal to a binary zero then the data is not in main memory. Attempting to access data with a data descriptor that has its presence bit equal to a binary zero causes a presence bit interrupt. The B 6800 system uses the occurrence of a presence bit interrupt as the preliminary step to start an MCP process which will move the data described by the data descriptor from system disk, or system pack storage into the main memory.



[50:3] = THE TAG FIELD.
THE TAG FIELD FOR A DATA DESCRIPTOR IS ALWAYS 101 BINARY
47 = PRESENCE BIT
46 = COPY BIT
45 = INDEXED BIT
44 = SEGMENTED BIT
43 = READ ONLY BIT
[42:3] = THE SIZE FIELD
[39:20] = THE LENGTH FIELD
[19:20] = THE ADDRESS FIELD

MV 2578

Figure 2-9. Data Descriptor Format

bit 46 Bit 46 is the copy bit. The copy bit indicates whether the data descriptor is the original descriptor for the data, or is a copy of the original descriptor. If the copy bit is equal to a binary zero then the data descriptor is the original. If the copy bit is a binary one then the data descriptor is a copy of the original descriptor. An original data descriptor is commonly referred to as a mother (or MOM) descriptor and a copy of a mother descriptor is commonly referred to as a copy descriptor.

bit 45 Bit 45 is the indexed bit. The indexed bit is used to indicate whether or not an indexing operation has been performed on the data descriptor. If the index bit is equal to a binary one then the descriptor has been indexed previously, and the value of the previous index is located in the length field 39:20. If the index bit is equal to binary zero then the data descriptor has never been indexed before, and such an indexing operation must be performed before the data that is described by the descriptor can be accessed. The process that causes the indexing operation to be performed will set the indexed bit, and will store the value of the index in the field 39:20.

bit 44 Bit 44 is the segmented bit. The segmented bit is used to identify whether or not the data described by the data descriptor is segmented. If the segmented bit is equal to a binary zero then the data is not in segments, and this descriptor describes the entire field.

bit 43 Bit 43 is the read only bit. The read only bit is used to show whether the memory area described by the data descriptor can be written into or not. If the read only bit is equal to a binary one then the data descriptor describes a memory area that may be read, but may not be written into. If the read only bit is a binary zero then the data descriptor describes a memory area that may be written into, or read from. It is possible for a single area in memory to be

described by two different data descriptors, one where the read only bit is a binary one, and another descriptor which has the read only bit equal to a binary zero. The memory area may be written into by use of the data descriptor that has the read only bit equal to a binary zero, but may not be written into by use of the data descriptor that has the read only bit equal to a binary one.

bits 42:3 Bits 42, 41, and 40 are used to define the type of data that is contained in the memory area described by the data descriptor. If bits 42 and 41 are both equal to binary zeroes, then the data descriptor defines an area in memory in words. A data descriptor that describes a string of character data is commonly called a string descriptor. If either bit 42, or bit 41 is equal to a binary one then the descriptor is a string descriptor. Bits 42:3 may contains several different binary values and the meaning of the different values that are used have the following meanings:

bit 42	bit 41	bit 40	
0	0	0	Bits 42, and 41 being equal to zero indicate that the data descriptor is a word descriptor. Bit 40 being equal to binary zero indicates that the data described by the descriptor is in single precision operands.
0	0	1	Bits 42, and 41 being equal to zero indicate that the data descriptor is a word descriptor. Bit 40 being equal to binary one indicates that the data described by the descriptor is in double precision operands.
0	1	0	Bits 42, and 41 not being equal to zero indicates that the data descriptor is a string descriptor, and bit 41 being a binary one indicates that the data described contains hexadecimal (4-bit) data.
0	1	1	Bits 42, and 41 not being equal to zero indicates that the data descriptor is a string descriptor. Bits 41, and 40 both being equal to binary ones indicate that the data described contains BCL data.
1	0	0	Bits 42, and 41 not being equal to zero indicates that the data descriptor is a string descriptor. Bit 42 equal to binary one indicates that the data described contains EBCDIC (8-bit) data.

bits 39:20 Bits 39:20 contain either the length of the memory area (if bit 45 is a binary zero) or an index value (if bit 45 is a binary one). If bit 45 is equal to binary zero the descriptor has not been indexed. This field is used for size checking during the indexing operation. If bit 45 is equal to a binary one the descriptor has been indexed. If the data descriptor is a word descriptor, and also if bit 40 is a binary one (the word area contains double precision operands) then the index is doubled after the indexing operation and the size checking operation have been completed. The doubled index is stored in the index field.

bits 19:20 Bits 19:20 contain either a main memory or a disk file address. If the presence bit is equal to a binary one and the copy bit is also equal to a binary one then the address field contains a main memory address of the MOM descriptor, of which the current descriptor is a copy. If the presence bit is equal to a binary one and the copy bit is equal to a binary zero then the address

field contains the main memory address of the first word that contains the data described by the descriptor. If the presence bit is equal to a binary zero, and the copy bit is also equal to a binary zero then the address field contains a six-bit binary coded decimal disk file address where the data described by the data descriptor is located.

STEP INDEX WORDS

Step index words are words that are used in conjunction with the step and branch operator in the B 6800 system. The purpose of the step and branch operator in the B 6800 system is to perform a series of other machine language operators in a recursive manner, but with control over the number of times the series of operators are executed. The step index word is used to provide the control part of the function of the step and branch operator.

The step index word (see figure 2-10) contains a tag of four (100 binary), and four other fields, as follows:

47:12 the increment value

35:16 the final value

19:04 an unused, but value specified field which must be equal to zero

15:16 the current value

Each time the series of machine language operators are performed the value of the increment is added to the value of the current value field. The step and branch operator then compares the current value field to the final value field. If the current value field is greater than the final value field a branch is taken out of the recursive series of operators. If the current value field is not greater than the final value field then the recursive series of operators are executed.

The increment value, the final value, and the current value are binary values. To determine the number of times a recursive series of operations will occur binary mathematics must be used, and not decimal mathematics. The unused but value specified field (19:04) must be equal to zero in the step index word.

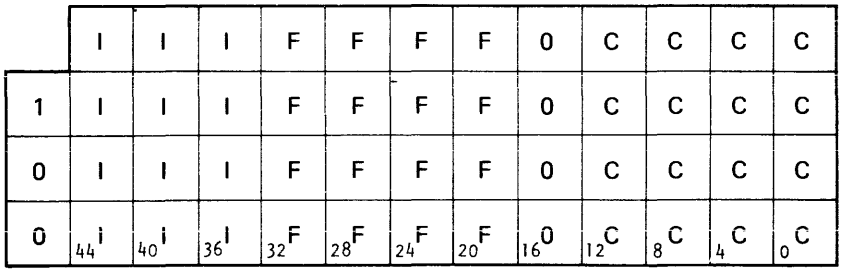


Figure 2-10. Step Index Word Format

SOFTWARE WORDS

A software word is a word with a tag field of six (110 binary) that is used by the MCP of the B 6800 system for software purposes. The MCP uses the software word for several different purposes, and the format of the word is different for each purpose. The software word is utilized as a linking word for memory allocation, as a software control word, as an un-initialized pointer word, and to contain system intrinsics data. Each of these uses for software words causes a different format to be used for the fields of data that are contained in the word.

The format of the software word when it is used for un-initialized pointers or for intrinsics information are not defined in this manual. These formats are specialized applications that are properly documented in manuals which deal with the specific application subjects.

The format of the software word when it is used for a memory link word and for a software control word is given in the following paragraphs. The specific use of the software word in either of these formats is not covered in this manual. Like the un-initialized pointer word and the intrinsics information word, these specific uses are specialized applications, and are more properly documented in manuals that deal with the software system as a specific subject.

The MCP maintains linking words in main memory to show which portions of the memory are in use, and which portions are not currently in use. A software word is used as the first link word for a portion of memory that is in use. This word is defined in the memory link system as the LINKA word, and each part of the main memory that is in use begins with a LINKA word. Memory link words are a mechanism for dynamic storage allocation which will be covered in more detail later in this manual. Figure 2-11 shows the format of a LINKA word.

Software control words are used by the software operating system to indicate the existance of memory areas that are related to the operating stack, but are physically located outside of the operating stack. When the memory area of an operating stack is deallocated (the stack is cut back), related memory areas must also be deallocated. The software control

	CF	S	S	S	S	S		A	A	A	A	A
1	CF	S	S	S	S	S	CS	A	A	A	A	A
1		S	S	S	S	S	AS	A	A	A	A	A
0		S	S	S	S	S	1	A	A	A	A	A
	44	40	36	32	28	24	20	16	12	8	4	0

TAG = 6 (110 BINARY) = SOFTWARE CONTROL WORD.
CF [47:2] = CONTROL FIELD FOR AREA DURING THE OVERLAY AREA MCP PROCESS.
S [43:20] = SIZE OF THE IN-USE AREA IN WORDS.
CS (BIT 22) = CONTROL SAVE FIELD – IF AREA IS TEMPORARILY SAVED CS=1.
AS (BIT 21) = AREA SAVED FIELD – IF AREA IS NON-OVERLAYABLE (SAVED) AS=1.
BIT 20 = IS BINARY 1 FOR A LINKA WORD.
A [19:20] = THE CORE MEMORY ADDRESS FOR THE MOM DATA DESCRIPTOR OF THE AREA CONTENTS

MV 2580

Figure 2-11. Software Control (LINKA) Word

word is a mask word that indicates the presence or absence of such related memory areas by the state of the bits in the mask word. At the time that the stack area is to be deallocated a related memory area is present for each bit that is a binary one value in the mask field of the software control word. Figure 2-12 shows the format of the software control word.

INDIRECT REFERENCE WORDS

Indirect reference words (IRW) are used in the B 6800 system to reference data that is located within the addressing environment of the current procedure. The addressing environment of the current procedure includes the current operating stack, and all stacks (that are a part of the current procedure) at a lower lexicographical level than the current operating stack level.

Stuffed indirect reference words (SIRW) are used in the B 6800 system to reference data that is located outside of the addressing environment of the current operating procedure.

	1						PLM SKF	ALM SKF	ALM SKF	ALM SKF	PC	PC
1	0						PLM SKF	ALM SKF	ALM SKF		PC	PC
1	GOTO ABO RTF						PLM SKF	ALM SKF	ALM SKF	PC	PC	PC
0						NOC PBT	PLM SKF	ALM SKF	ALM SKF	PC	PC	PC
	44	40	36	32	28	24	20	16	12	8	4	0

[50:3] = TAG FIELD = 110 = SOFTWARE CONTROL WORD
[47:2] = 2 = SOFTWARE CONTROL WORD (MASK WORD)
45 = 1 = GO TO ABORTE
24 = 1 = NOCPBIT
[23:4] = PL/I COMPILER BLOCKEXIT AND FAULT FIELD
[19:9] = MASK FIELD
- 19 = NOT USED
- 18 = FMT PSUEDO BUFFER FIB-LOCKED
- 17 = NON-LOCAL GOTO
- 16 = DIRECT ARRAY DECLARATION IN BLOCK
- 15 = FAULT IN BLOCK DECLARATION
- 14 = INTERRUPT IN BLOCK DECLARATION
- 13 = FILE IN BLOCK DECLARATION
- 12 = MULTI-DIMENSION ARRAY IN BLOCK DECLARATION
- 11 = SINGLE-DIMENSION ARRAY IN BLOCK DECLARATION

[9:10] = PROCESS COUNT

MV 2581

Figure 2-12. Software Control (MASK) Word

The fields of an indirect reference word or a stuffed indirect reference word do not contain data. Instead, the fields of an indirect reference word or a stuffed indirect reference word contain addressing information that is used to point at the location of data. The fields of an IRW, or a SIRW are both displayed in figure 2-13. The fields within the IRW and the SIRW are as follows:

bits 50:3 — Bits 50:3 are the tag field. The tag field for an IRW is always 001 binary, regardless of whether the IRW is stuffed, or normal.

bit 46 — Bit 46 is the environment bit. If bit 46 is a binary one the IRW is stuffed. If bit 46 is a binary zero the IRW is a normal IRW.

bits 45:10 — Bits 45:10 are the stack number field. The stack number is not used in a normal IRW and is equal to binary zero. If bit 46 is a binary one then the value of the stack number field is the identification number of the stack that is to be referenced.

bits 35:16 — Bits 35:16 are the displacement field. The displacement field is not used for a normal IRW and is equal to binary zero. If bit 46 is a binary one then the displacement field is added to the address of the base of the stack being referenced to locate a mark stack control word within the referenced stack area.

		SNR	SNR	D	D	D	D			A	A	A
0	E	SNR	SNR	D	D	D	D			A	A	A
0	SNR	SNR	SNR	D	D	D	D		A	A	A	A
1	SNR 44	SNR 40	SNR 36	D 32	D 28	D 24	D 20	 16	A 12	A 8	A 4	A 0

MV 2582

Figure 2-13. Indirect Reference Word

bits 12:13 Bits 12:13 are the index field. The index field is not used in a normal IRW, however the same bits are used for a different purpose. If bit 46 is a binary one then the index field is added to the address of the mark stack control word in the referenced stack. The sum of these values is the address of the data that is being addressed.

bits 13:14 Bits 13:14 are the address couple field. The address couple field is not used in the SIRW, however the same bits are used for a different purpose. The address couple field is used in an IRW to locate data in the addressing environment of the current procedure. The address couple consists of two separate values each of which are of variable bit length. The most significant part of the address couple contains the lexicographical level value. The least significant part of the address couple contains an index value that is added to the address of the mark stack control word that corresponds to the lexicographical control level. The sum of the address of the mark stack control word, and the index value is the address of the data referenced by the IRW.

The lexicographical level (program level) of a current procedure may have any value from zero, through thirty-one. The lexicographical level (LL) part of an address couple is represented by the most significant bits of the address couple. The LL requires five bits of the address couple to represent the binary value of thirty-one which is the highest LL value possible. When the LL contains a value of zero or one only one bit is required to represent the binary LL value. The actual number of binary bits that are used to contain the LL value in an address couple is defined by the level of the current operating procedure. Thus, if the current procedure is at lexicographical level seven then the number of bits in the address couple that are used to indicate LL is three, because three binary bits are required to represent the value of seven decimal.

The index part of an address couple consists of the bits that are not required to represent the LL value. Thus, if the lexicographical level of the current procedure is seven, then three binary bits (bits 13, 12, and 11) are required to represent the LL value, and the remaining bits (bits zero through ten) are used to represent the index part of the address couple.

The B 6800 system derives the absolute memory address referred to by an IRW in the following manner:

a. The LL part of the address couple defines the IC memory display register that contains the address of a mark stack control word in main memory.

b. The index part of the address couple is added to the address of the mark stack control address. This sum is the absolute address of the data referred to by the IRW.

Since the number of bits in the address couple that are required to contain the LL value is a variable number, the size of the index value is limited by the number of bits that comprise the index value. Thus, if three bits are required to contain the LL value, then the size of the index part is limited to an eleven bit binary value (or a maximum index value of 2047 decimal memory words). Table 2-2 shows the maximum number of memory words that may be contained in the index part of an address couple for any given LL value part of the address couple.

The B 6800 system determines the absolute address referred to by the SIRW in a different way than is used for determining the absolute address referred to by an IRW. The method used to determine the absolute address referred to by a SIRW is as follows:

a. The stack number field in the SIRW is an index into the segment descriptor index, which is maintained by the MCP. The segment descriptor index contains a list of data descriptors that give the absolute memory addresses of all stacks in main memory. The stack number field of the SIRW identifies the descriptor that contains the base address of the stack that is to be referenced.

Table 2-2. Address Couple Value Fields

Lexicographical Level Value	Number of Bits Required	Bits Available for Index Value	Maximum Index Value
0	1	13	8191
1	1	13	8191
2	2	12	4095
3	2	12	4095
4	3	11	2047
5	3	11	2047
6	3	11	2047
7	3	11	2047
8	4	10	1023
9	4	10	1023
10	4	10	1023
11	4	10	1023
12	4	10	1023
13	4	10	1023
14	4	10	1023
15	4	10	1023
16 through 31	5	9	511

b. The displacement field value of the SIRW is an index on the base address of the stack that is being referenced. The value of the base address of the stack, plus the value of the displacement field is the absolute memory address of a mark stack control word in the stack that is being referenced.

c. The index field value of the SIRW is an index on the address of the mark stack control word in the stack that is being referenced. The sum of the address of the mark stack control word plus the value of the index field is the address of the value that is being addressed by the SIRW.

PROGRAM CONTROL WORDS

The program control word (PCW) is used by the B 6800 system to point to the program code for a procedure or segment of a program. The PCW also contains program information about the system environment that is to be used during the execution of the segment or program.

The use of PCW's provides the flexibility that the software requires to utilize reentrant code techniques, and also dynamic storage allocation principals. The reentrant code techniques are used in the B 6800 system to provide the software capability to execute more than one job at a time while using the same machine language code.

Figure 2-14 shows the fields of data that are contained in a PCW. The fields of data in a PCW are used as follows:

bits 50:3 — The tag field. The tag field for a PCW is seven (111 binary).

bits 45:10 — The stack number. The stack number field is used to identify the stack that contains the PCW (not always the stack that is to be associated with the program code that is to be executed).

The MCP uses stack numbers to identify jobs that are currently being executed, or are scheduled to be executed. The MCP assigns stack numbers for program stacks on a first come first served basis. Therefore the stack number for a program stack is a dynamic variable that is assigned to a program at execution time.

		SNR	SNR	PSR	PIR	PIR	PIR	N	LL	SDI	SDI	SDI
1		SNR	SNR	PSR	PIR	PIR	PIR	LL	LL	SDI	SDI	SDI
1	SNR	SNR	SNR	PSR	PIR	PIR	PIR	LL	SDI	SDI	SDI	SDI
1	SNR 44	SNR 40	SNR 36	PIR 32	PIR 28	PIR 24	PIR 20 -	LL 16	SDI 12	SDI 8	SDI 4	SDI 0

50:3 = THE TAG FIELD.
7 IS A PCW TAG
45:10 = THE STACK NUMBER FIELD
35:3 = THE PROGRAM SYLLABLE REGISTER VALUE
32:13 = THE PROGRAM INDEX REGISTER VALUE
19 = THE NORMAL/CONTROL STATE BIT
18:5 = THE LEXICOGRAPHICAL LEVEL VALUE
13:14 = THE SEGMENT DESCRIPTOR INDEX VALUE

MV 1583

Figure 2-14. Program Control Word

bits 35:3 The program syllable register (PSR) field. The PSR field is used to indicate the first machine language operator in the first memory word of a machine language code string. A program code string is not required to begin at the first machine language operator in a memory word. There are 6 syllables in a machine language code word, and the PSR value indicates which of the 6 syllables the current string of code starts in.

bits 32:13 The program index register (PIR) value. The PIR field is used to indicate the first word of the program machine language code string. The combination of the PIR field and the PSR field combine to identify the specific first machine language operator in the program code string. The PIR value defines the first word address of the string, and the PSR value defines the first syllable within the first word of the string.

bit 19 The normal state/control state bit. The B 6800 system may operate in either of two states, and the proper state for the current code segment is defined by the normal state/control state bit. If the normal state/control state bit is a binary one then control state is specified, and normal state is specified otherwise.

bits 18:5 The lexicographical level (LL) field. The LL field is used to specify the lex level at which the program string is to be executed. The LL value defines one of the 32 IC memory display registers. The value in the selected IC memory display register is the base address in core memory of the program stack with which the program code string is associated.

bit 13:1 This bit is used to indicate that the D0 stack contains the segment descriptor (if 0), or the D1 stack (if 1).

bits 12:13 The segment descriptor index (SDI) field. The SDI is used to indicate the location of the segment descriptor for the program code string in core memory.

The 13 bits of the SDI field are a binary index value which are added to the base address from the display register (either D0 or D1) to define the absolute core memory address of the segment descriptor for the machine language code string.

MARK STACK CONTROL WORDS

The mark stack control word (MSCW) is used to define an area within the stack in main memory. The MSCW and the return control word (RCW) together provide a history of the stack linkage, and a record of the stack operating environment. The historical links of a stack, and the operating environment record of the stack are key data in the reconstruction and analysis of program operations.

Figure 2-15 shows the fields of data that are contained in the MSCW. The meaning of the fields of data in the MSCW are as follows:

bits 50:3 — The tag field. The tag for a MSCW is three (011 binary).

bit 47 — The different stack bit. The different stack bit indicates whether the stack number field refers to the same stack, or to a different stack. If the different stack bit is a binary zero then the stack number field refers to the same stack. If the different stack bit is a binary one then the stack number refers to a different stack.

	DS	SNR	SNR	DS	DS	DS	DS	V	LL	DF	DF	DF
0	E	SNR	SNR	DS	DS	DS	DS	LL	LL	DF	DF	DF
1	SNR	SNR	SNR	DS	DS	DS	DS	LL	DF	DF	DF	DF
1	SNR 44	SNR 40	SNR 36	DS 32	DS 28	DS 24	DS 20	LL 16	DF 12	DF 8	DF 4	DF 0

50:3 = TAG FIELD. MARK STACK TAG IS ALWAYS 3
47 = DIFFERENT STACK BIT
46 = ENVIRONMENT BIT
45:10 = STACK NUMBER FIELD
35:16 = DISPLACEMENT FIELD
19 = VALUE BIT
18:5 = LEXICOGRAPHICAL LEVEL FIELD
13:14 = DIFFERENCE FIELD

MV 1584

Figure 2-15. Mark Stack Control Word

bit 46 — The entered bit. The entered bit is used to indicate whether the stack is active or not. If the stack is currently in use (is active) then the bit will be set to a binary one. If the stack is not currently in use then the bit will be reset to a binary zero. If the entered bit is a binary one then it indicates that the MSCW is active and was entered into the stack by a procedure entry. If the entered bit is a binary zero it shows that the MSCW was entered into the stack by the mark stack machine language operator, and no procedure entry has been made in the stack. When a procedure entry is made into the stack the environment fields of the MSCW are completed from the PCW that caused entry, and the entered bit is set to a binary one.

bits 45:10 — The stack number. The stack number field is completed at procedure entry time, and contains the stack number value from the PCW that was entered. The stack number is the designation of the stack that contains the PCW, not the number of the current stack.

bits 35:16 — The displacement field. The displacement field is used to link a program together by its lexicographical levels. The value of the displacement field defines the MSCW that represents the last previous lexicographical level of the procedure. The location of the MSCW that corresponds to the preceding lexicographical level is determined by adding the value of the displacement field to the value of BOSR for the stack.

bit 19 — The value bit. The value bit is used to indicate whether or not the operator that caused entry to the current operator is to be restarted at the beginning of the operator in the procedure that caused entry. If the value bit is a binary zero then the previous operator must be restarted from the beginning. If the value bit is a binary one then the previous operator must be continued at the next operator in sequence.

bits 18:5 — The lexicographical level field. The value of the lexicographical level field defines the lexicographical level at which the program will run when the procedure is entered.

bits 13:14 — The difference field. The difference field is used to store the stack history. The value of the difference field is the number of words between the current MSCW and the previous MSCW in the stack. Subtracting the value of the difference field from the address of the current MSCW gives the address of the previous MSCW.

INTERRUPT PARAMETER WORDS

The interrupt controller of the B 6800 data processor recognizes certain types of system interrupts. The DP interrupt controller interrupts the program that is running, and causes an entry into the MCP interrupt handling procedures when a

system interrupt is sensed. The interrupt handling procedures of the MCP initiate system actions that are required because of the interrupt condition that exists. At the conclusion of the interrupt handling function, the MCP returns control of the DP to the program or process that was interrupted when the system interrupt was recognized.

The interrupt controller collects and formats data about the type of interrupt that occurred. This data is placed in a special stack (see figure 2-16) which the interrupt controller creates for the interrupt handling procedures of the MCP. After the interrupt controller has created and filled the interrupt handling stack, a program entry is made into the interrupt handling procedures of the MCP.

P1 Parameter

The format and content of the data that is placed in the interrupt handling stack depends on the type of interrupt that occurred. There are five types of interrupts that are recognized by the interrupt controller of the DP, which are: Alarm type, Hardware type, General Control type, External type, and Syllable Dependent type. The first word of data in the interrupt stack is the P1 parameter. The P1 parameter defines the type of interrupt that was sensed, and indicates the

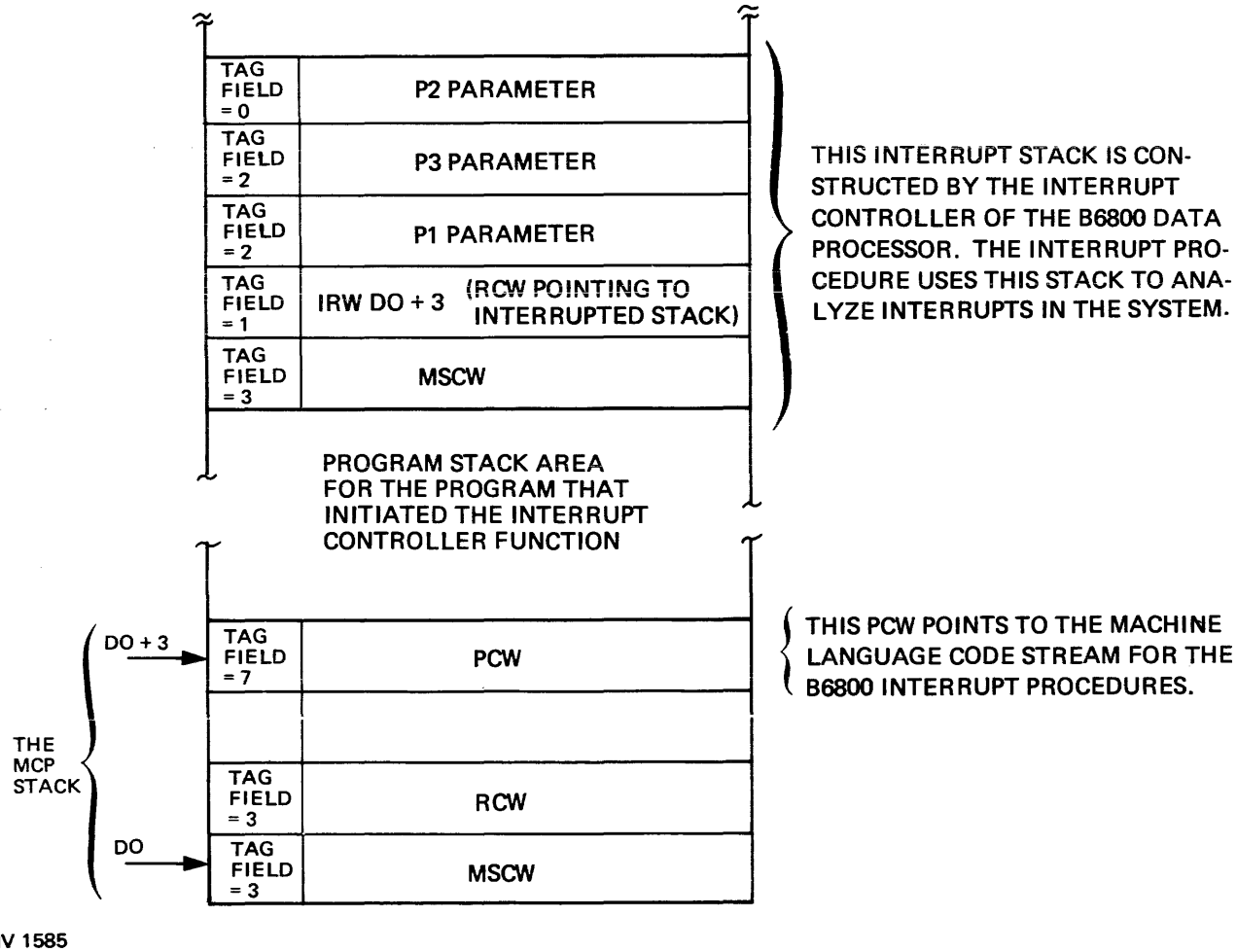


Figure 2-16. B 6800 Interrupt Stack Organization

cause of the interrupt. Table 2-3 shows the types of interrupts that are defined in the P1 parameter, and also shows the various causes of each type of interrupt. The P1 parameter is the first half (upper half) of a double precision word. The last half (lower half) of the double precision word is the P3 parameter. Table 2-4 shows what information about an interrupt is to be present in the P2, and P3 parameters of the interrupt handling procedure stack.

P3 Parameter

The P3 parameter is the second half of a double precision word in the interrupt handling procedure stack.

The purpose of the P3 parameter is to provide a place to record the hardware operating environment conditions at the time that an interrupt occurs. The B 6800 system uses the information contained in the P3 parameter to help in the analysis of the cause of the interrupt.

The information contained in the P3 parameter is also valuable in determining the cause of a hardware failure which results in an operating system interrupt. The information that is present in the P3 parameter is recorded in the SYSTEM SUMLOG file, and thus is available to help maintenance personnel in determining the cause of hardware failures.

The P3 parameter has a variable format that depends on the type of interrupt that has occurred. There are five different formats, but only one format is used for each type of interrupt. Figure 2-17 shows the formats that are used for Alarm type, Hardware type, Syllable Dependent type, and General Control type interrupts. Figure 2-18 shows the format that is used for the P3 parameter when an IO finished interrupt occurs. Table 2-4 shows what data is present in the P3 parameter for the specific cause of each of the five types of interrupts.

P2 Parameter

The P2 parameter for the B 6800 typically contains the contents of the top-of-stack register at the time the interrupt occurred. This context is true for alarm type interrupts with the single exception of the stack underflow interrupt. In the case of the stack underflow interrupt the value of the S-register will be placed in the P2 parameter word.

The B 6800 system P2 parameter for syllable dependent interrupts contains additional information. The additional information that is contained in the P2 parameter as follows:

a. For a sequence error that occurs during a family C operation the P2 parameter will contain the value of the word that caused the sequence error

b. For an invalid operation interrupt that occurs during a SPLT (9543) operator the word that caused the interrupt will be reported in the P2 parameter.

c. For an invalid operation interrupt that occurs during a JOIN (9542) operator function the word that caused the interrupt will be reported in the P2 parameter. If the information in both the A and B registers is bad then the the word in the A register becomes the P2 parameter data.

Table 2-3. P1 Parameter Words (Sheet 1 of 2)

Type	Cause	Parameter Bits																													
		46	45	44	39	27	26	25	24	23	22	21	20	19	18	17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Alarm	Loop Timer					1		1						Ø		Ø																1
Alarm	Memory Addr Parity					1		1				Ø		Ø	Ø	Ø															1	
Alarm	Scan Bus Parity					1		1						Ø		Ø														1		
Alarm	Inv Address-Local					1		1				Ø		Ø		Ø													1			
Alarm	Stack Underflow					1		1						Ø		Ø												1				
Alarm	Inv Program Word					1		1						Ø		Ø											1					
Alarm	Memory Address Residue					1		1				Ø		Ø	Ø	Ø										1						
Alarm	Read Data Mult. Error					1		1				Ø		Ø	Ø	Ø									1							
Alarm	Inv. Address Global					1		1				Ø		Ø	Ø	Ø								1								
Alarm	Global Memory Not Ready					1		1				Ø		Ø	Ø	Ø							1									
Alarm	Scan in Info Error					1		1						Ø								1										
Alarm	Scan Out Error					1		1						Ø							1											
Hardware	PROM Card Parity					1	1							Ø		Ø																1
Hardware	RAM Card Parity					1	1							Ø		Ø															1	
Hardware	Bus Residue					1	1							Ø		Ø														1		
Hardware	Adder Residue					1	1							Ø		Ø													1			
Hardware	Compare Residue					1	1							Ø		Ø												1				
Gen. Control	Read Data Single Error	Ø		Ø		1					1			X	Ø																	
Gen. Control	Read Data Retry	Ø		Ø		1					1			X	Ø														1			
Gen. Control	Read Data Check Bit	Ø		Ø		1					1			X	Ø													1				
Gen. Control	Address Retry	Ø		Ø		1					1			X	Ø												1					1

NOTES: 1. 1 = BIT is a binary one.
0 = BIT is a binary zero.
Ø = Bit may be either a binary one or a binary zero.
X = State of bit is immaterial.

2. Bit 18 indicates whether the operation was a scan or a memory operation. to the Global Memory:
If bit 18 = 0 it was a memory operation.
= 1 it was a scan operation.

3. If bit 17 is a binary one it indicates that the data in the P3 parameter is inconsistent.
4. Bit 27 is the B 6800 bit. This bit is true for B 6800 systems.
5. Bit 21 is the memory error during external device operation (Channel B of memory) bit.

Table 2-3. P1 Parameter Words (Sheet 2 of 2)

Type	Cause	Parameter Bits 46	45	44	39	27	26	25	24	23	22	21	20	19	18	17	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
External	Status Change					1							1	X											1	1	1	1				1
External	I/O Finished					1							1	X											1			1				1
External	DCP					1							1	X												(See Note 2)						1
External	Scratch Pad Parity					1							1	X											1							1
SDI	Programmed Operator			Ø		1			1					Ø																		
SDI	Memory Protected			Ø		1			1					Ø																		1
SDI	Invalid OP			Ø		1			1					Ø																	1	
SDI	Divide by Zero			Ø		1			1					Ø																1		
SDI	Exp. Overflow			Ø		1			1					Ø															1			
SDI	Exp. Underflow			Ø		1			1					Ø														1				
SDI	Invalid Index			Ø		1			1					Ø													1					
SDI	Integer Overflow			Ø		1			1					Ø												1						
SDI	Bottom of Stack			Ø		1				1				Ø											1							
SDI	Presence Bit	RT	RT	Ø	VS	1			1					Ø										1								
SDI	Seq. Error			Ø		1			Ø	Ø				Ø									1									
SDI	Segm. Array			Ø		1			1					Ø								1										
SDI	Interval Timer			Ø		1			1					Ø							1											
SDI	Stack Overflow			Ø		1			1					Ø						1												
SDI	Confidence Error			Ø		1			1					X					1													

NOTES: 1. 1 = Bit is a binary one.
0 = Bit is a binary zero.
Ø = Bit may be either a binary one or a binary zero.
X = State of the bit is immaterial.

2. External interrupt from DCP

Bits	Unit
6 5 4	
0 0 1	DCP 1
0 1 0	DCP 2
0 1 1	DCP 3
1 0 0	DCP 4
1 1 0	BIC 1
1 1 1	BIC 2

3. Bit 27 is the B 6800 bit. This bit is 1 for B 6800 systems.

Table 2-4. Interrupt Procedure Stack Parameter Contents

Kind of Error	Interrupt Type P1 Parameter	Contents of the P2 Parameter	Contents of the P3 Parameter
1. Loop Timer	Alarm		Strb, JC, Op
2. Memory Address Parity	Alarm		Addr, JC, Strb, Op
3. Scan Bus Parity Address	Alarm		Addr, JC, Strb, Op
4. Inv. Address, Local	Alarm		Addr, JC, Strb, Op
5. Stack Underflow	Alarm	S Register	Addr, JC, Strb, Op
6. Inv. Progr. Word	Alarm	Word	JC, Strb, Op
7. Memory Address Residue	Alarm		Addr, JC, Strb, Op
8. Read Data Multiple Error	Alarm	Word	Addr, JC, Strb, Op
9. Inv. Addr, Global	Alarm		Addr, Strb, JC, Op
10. Global Memory Not Ready	Alarm		Addr, Strb, JC, Op
11. Scan In Info Error	Alarm	Word	Addr, Strb, JC, Op
12. Scan Out Error	Alarm	Word	Addr, Strb, JC, Op
1. Prom Card Parity	Hardware		JC, Strb, Op, Card #
2. RAM Card Parity	Hardware		JC, Strb, Op, Card #
3. Bus Residue	Hardware		JC, Strb, Op
4. Adder Residue	Hardware		JC, Strb, Op
5. Compare Residue	Hardware		JC, Strb, Op
1. Read Data Single Error	Gen. Cntr.		Addr, Bit #
2. Read Data Retry	Gen. Cntr.		Addr
3. Read Data Check Bit	Gen. Cntr.		Addr, Bit #
4. Address Retry	Gen. Cntr.		Addr
1. Unit Status Change	External	Status Vector	
2. I/O Finished	External	Result Descriptor	Error Conditions
3. DCP	External		
4. Scratch Pad Parity	External	Card #, Channel #	
1. Programmed Operator	SDI	See the text under the	JC, Str, Op
2. Memory Protected	SDI	subheading titled P2	JC, Str, Op
3. Invalid Op	SDI	Parameter	JC, Str, Op
4. Divide by zero	SDI		JC, Str, Op
5. Exponent Overflow	SDI		JC, Str, Op
6. Exponent Underflow	SDI		JC, Str, Op
7. Invalid Index	SDI		JC, Str, Op
8. Integer Overflow	SDI		JC, Str, Op
9. Bottom of Stack	SDI		JC, Str, Op
10. Presence Bit	SDI		JC, Str, Op
11. Seq. Error	SDI		JC, Str, Op
12. Segm. Array	SDI		JC, Str, Op
13. Interval Timer	SDI		JC, Str, Op
14. Stack Overflow	SDI		JC, Str, Op
15. Confidence Error	SDI		JC, Str, Op

Footnotes:
Addr is the Memory or Scan address
Strb is the family strobe
JC is the family seq. counter count
OP is the Op code
Card # is the number of the failing card
Bit # is the number of the failing bit

The B 6800 system external type interrupts provide the following information:

a. Unit status change external interrupts provide the status vector word information in the P2 parameter word.

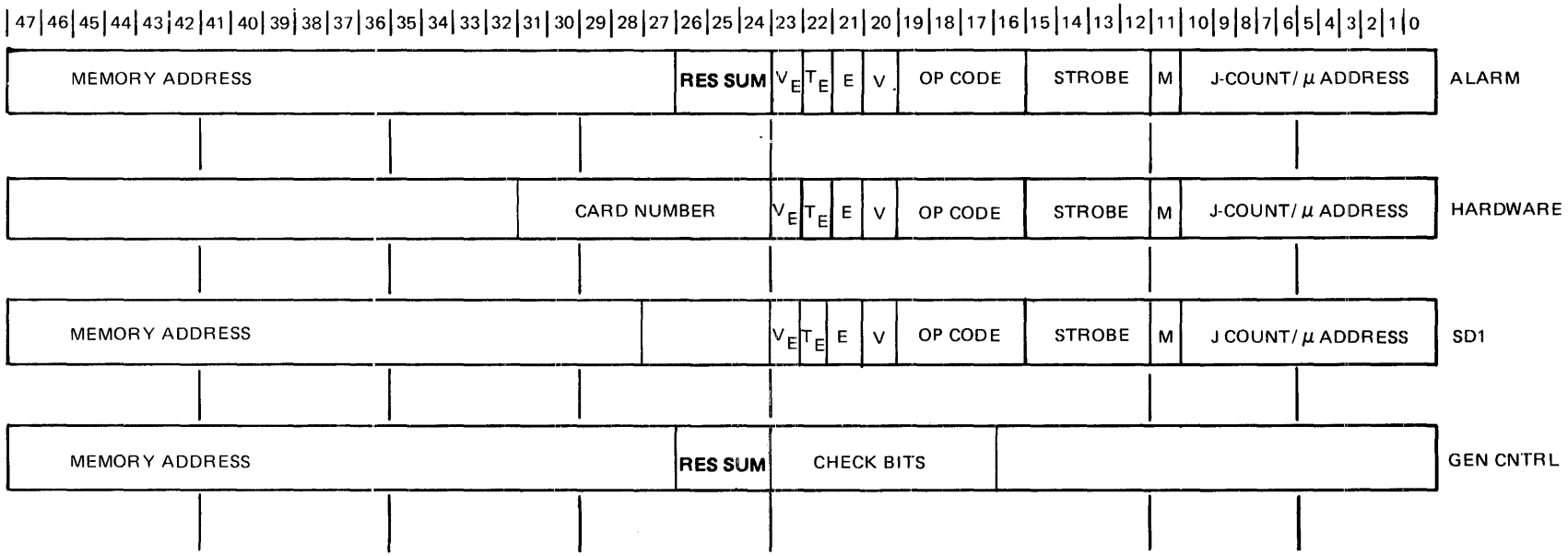
b. IO finished external interrupts provide the result descriptor data from the IO processor (multiplexer) in the P2 parameter word.

c. Scratch pad parity external interrupts report the card number, and the channel number of the multiplexer scratch pad memory word that caused the interrupt in the P2 parameter

Figures 2-19 through 2-21 show the P2 parameter word layouts for the three cases stated in a through c above. Table 2-4 indicates the data that is present in the P2 word for different types of interrupts in the B 6800 system.

RETURN CONTROL WORDS

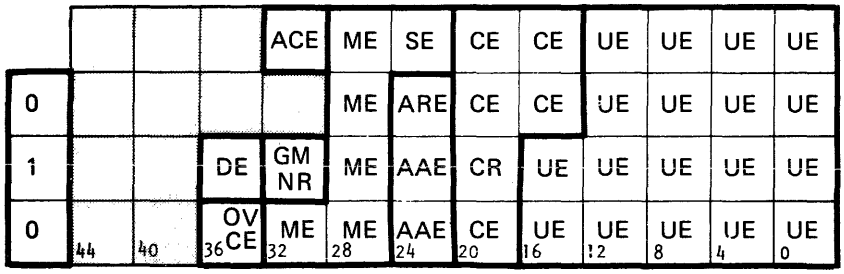
A return control word is used in the B 6800 system to provide a method for returning to a previous procedure. The second entry in an active job stack is always a return control word. The hardware of the B 6800 system automatically creates the return control word (RCW) for a previous procedure or program when an entry to the new procedure is made. Prior to the hardware inserting the return control word into the stack, the second word in the stack is either a PCW, or an IRW. The return control word is substituted for which ever type of word is the second word in the new procedure stack.



RES SUM = RESIDUE OF ADDRESS
V_E = VECTOR
T_E = TABLE
E = EDIT
V = VARIANT
M = MODE
M = MODE 0 MEANS J-COUNT IS ACTIVE.
1 MEANS μ ADDRESS IS ACTIVE

MV 1586 A

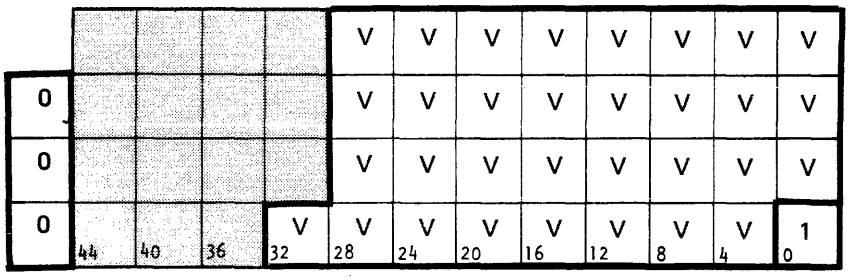
Figure 2-17. P3 Parameter Configurations



BITS 17:18 = UNIT ERROR FIELD
23:6 = CONTROL ERROR FIELD
26:3 = ADDRESS ADDER ERROR FIELD
32:6 = MEMORY ERROR FIELD
33 = GLOBAL MEMORY NOT READY BIT
35 = ADDR COMP ERROR
36 = OP CODE OR VARIANT CHARACTER GENERATOR ERROR
37 = DESCRIPTOR ERROR

MV 1587

Figure 2-18. P3 Parameter Contents for IO Finished Interrupt



50:3 = TAG FIELD, (ALWAYS = 0 FOR P2 PARAMETER)
32:32 = VECTOR WORD. EACH BIT IN THE VECTOR WORD REPRESENTS A UNIT THAT IS REPORTING A STATUS CHANGE

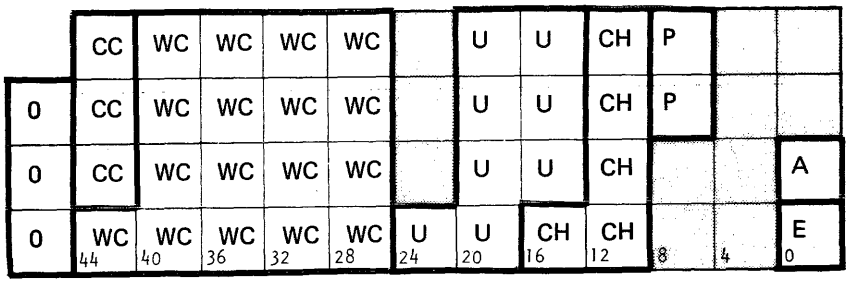
MV 1588 BIT 0 = IS ALWAYS A BINARY ONE

Figure 2-19. P2 Parameter Status Change Format

Figure 2-22 shows the fields of data that are present in the RCW, and defines the meaning of the data in each field. The combination of data fields that are stored in the RCW indicates what the hardware environment will be after the return to the previous procedure has been made.

PROGRAM WORDS (CODE WORDS)

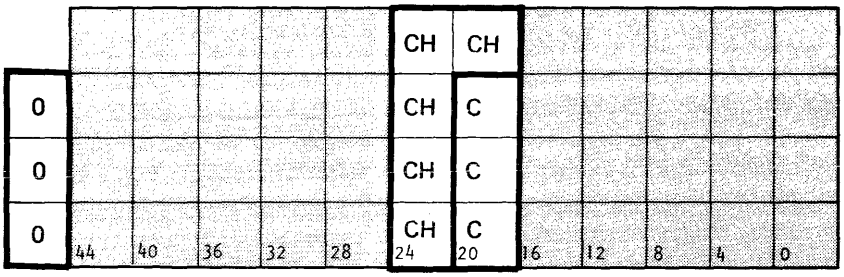
Program words are B 6800 words that contain the machine language instructions which the data processor executes. Program code words are grouped into units of words called segments. A segment consists of all the machine language code for a program or a segment of a program. A program segment may consist of from one program code word, to a



50:3 = TAG FIELD.
(ALWAYS = TO ZERO FOR P2 PARAMETER)
47:3 = CHARACTER COUNT
44:17 = WORD COUNT
24:8 = UNIT NUMBER FIELD
16:5 = CHANNEL NUMBER FIELD
11:2 = PATH NUMBER FIELD
BIT 1 = ATTENTION BIT
BIT 0 = ERROR BIT

MV 1589

Figure 2-20. P2 Parameter Result Descriptor Format

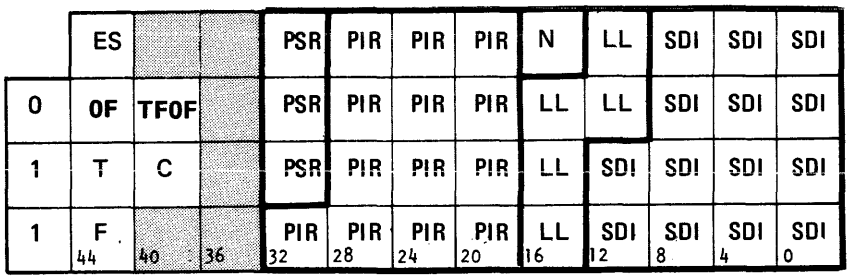


50:3 = TAG FIELD.
(ALWAYS = 0 FOR P2 PARAMETER)
27:5 = CHANNEL NUMBER FIELD
22:3 = CARD NUMBER FIELD

MV 1590

Figure 2-21. P2 Parameter Scratch Pad Parity Format

maximum of 16,384 words. It is unusual for a program segment to exceed several hundred words. Each segment of program code in a program is referenced (and located) through the segment descriptor index field in the PCW that calls the segment to be executed by the data processor. A segment of code may call upon the system to execute another



50:3 = TAG FIELD.
(ALWAYS A VALUE OF 3 FOR AN RCW)
BIT 47 = EXTERNAL SIGN BIT FLIP-FLOP STATE
BIT 46 = OVERFLOW FLIP-FLOP STATE
BIT 45 = TRUE/FALSE FLIP-FLOP STATE
BIT 44 = FLOAT FLIP-FLOP STATE
BIT 42 = TRUE/FALSE FLIP-FLOP OCCUPIED FLIP-FLOP STATE
BIT 41 = COMPARE FLIP-FLOP
35:3 = VALUE OF PROGRAM SYLLABLE REGISTER FIELD
32:13 = VALUE OF PROGRAM INDEX REGISTER FIELD
BIT 19 = NORMAL/CONTROL STATE FLIP-FLOP STATE;
BINARY ZERO = NORMAL STATE
BINARY ONE = CONTROL STATE
18:5 = VALUE OF LEXACOGRAPHICAL LEVEL REGISTER
13:14 = SEGMENT DESCRIPTOR INDEX VALUE

MV 1591

Figure 2-22. Return Control Word

segment of code. At the conclusion of such a called segment, the system will return to the calling segment. The location of the code for the calling segment is not lost during the execution of the called segment code because the RCW of the called segment contains the SDI value for the code of the calling procedure. Thus when returning to the calling procedure the code segment location is known.

PROGRAM SEGMENTS AND THE SEGMENT DESCRIPTOR

The program code that is executed when a program job or task is performed is contained in words of machine language operator codes. All of the operator codes that comprise the task are grouped together in groups called segments. A segment may contain all of the machine language operators, or a major group of the operator codes in a program task.

When a program task is to be executed, an ENTER operator causes the PCW for the task to be brought into the stack, and distributed to the various parts of the operating system. The SDI field of the PCW word (refer to figure 2-14) locates a segment descriptor (SD) for the program task. A description of the SD (figure 2-23) is as follows:

bits 50:3 — The tag field. The tag for a SD is always three (011 binary).

bit 47:1 — The presence bit. If this bit is binary one then the program code segment is present in local memory.

bit 46:1 — The copy bit. If this bit is a binary zero then the segment descriptor is the original segment descriptor. If this bit is a binary one then this descriptor is a copy of an original segment descriptor.

bits 45:6 — An unused field. These bits may be either binary ones or zeros because they have no effect upon the use of the word as a segment descriptor.

bits 39:20 — The length field. This field specifies the length of the code segment, in words, in binary notation.

bits 19:20 — The address field. If the presence bit is a binary one then this field contains the absolute address of the first word in the segment. If the presence bit is a binary zero and the copy bit is also a binary zero then this field contains a five digit binary coded decimal disk address for the code segment. If the presence bit is a binary zero and the copy bit is a binary one then this field contains the absolute memory address of the original segment descriptor.

A program code segment may call another program segment to be executed. Each of these program code segments (the calling segment, and the called segment) has a separate segment descriptor. The address (SDI) for the current code segment is saved in the data processor IC memory registers. The value of the called SDI is saved when the called segment is executed. However, the SDI for the calling segment is not lost, because this address is saved in the RCW (refer to figure 2-22). Thus, when a called segment is executed, and a return (or EXIT) to the calling segment is performed, the SDI is always available for the currently executing program segment.

The use of copy segment descriptors, and the mechanism for saving the SDI values for segments of program code are basic components used to provide for the concepts of reentrant code. Reentrant code techniques are defined in section 3 of this manual.

TOP OF STACK CONTROL WORDS

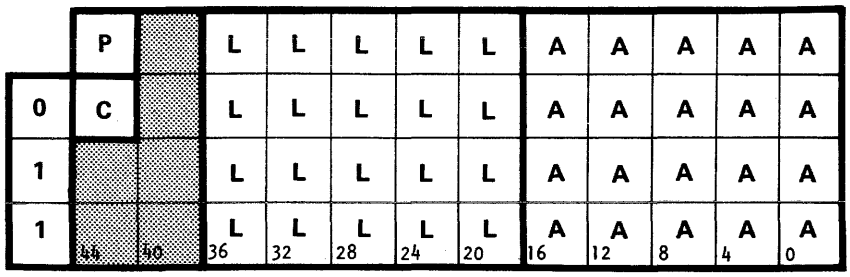
A top of stack control word (refer to figure 2-24) is originated when the data processor executes the move to stack operator. This word occupies the address in memory of the lower word boundary for a job or task area. A TOSCW contains the relative addressing and environment record for the program or task. The address of a TOSCW for an operating program or task is the same as the value of the BOSR address register. A TOSCW therefore also corresponds to the address of the first MSCW for a job or task.

The addressing environment for a program or task consists of the values of the BOSR, F, S, and lexicographical level registers. The values of these registers are stored in the TOSCW when another program or task is to be executed. Upon re-entry into the program or task procedures, the proper values from the TOSCW are used to restore the proper addressing environment for the program or task, in the memory address registers.

The operating environment of a job or task consists of the state of seven flip-flops. These flip-flops are the external sign, overflow, true/false, float, true/false occupied, compare, and normal/control state flip-flops. The state of these flip-flops is stored in the TOSCW when another job or task is to be executed. Upon re-entry into the original job or task, the proper values for operating environment flip-flops are restored from the TOSCW.

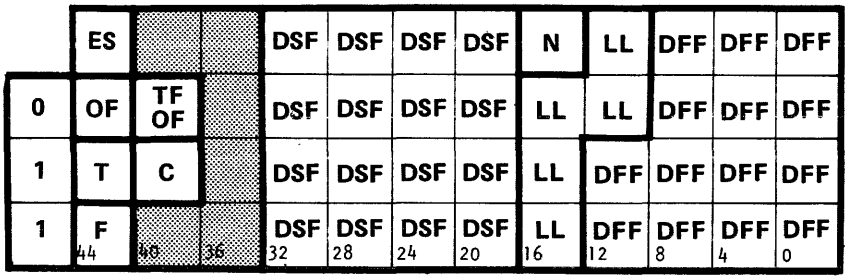
The TOSCW for the currently operating program or task does not contain the operating and addressing environment. Instead, the CPU data processor identity (001 for a B 6800 system) is stored in bits 2:3, and the rest of the bits (except the tag field) are zeros. The presence of a TOSCW which only contains the data processor identity field indicates the address of the lowest word in the current job or task stack. This word is addressed by the value of the BOSR register.

A program code word is composed of six syllables, and a tag field (see figure 2-25). The tag field for a program code word is always a value of three. The remaining 48 bits of the program word is divided into six 8-bit syllable fields. A machine language instruction consists of from one to seven syllables. An instruction is not limited to a single code word but may extend across the boundary of a code word, and into the next word of program code in sequence. For this reason the contents of a word of machine language code may be portions of two operators, plus from one to four complete operator codes.



50:3 = TAG FIELD.
(ALWAYS A VALUE OF 3 FOR A SEGMENT DESCRIPTOR)
47:1 = PRESENCE BIT, 1 = PRESENT IN MEMORY
0 = PRESENT IN LIBRARY
46:1 = COPY BIT, 1 = COPY OF ORIGINAL SEGMENT DESCRIPTOR
0 = ORIGINAL SEGMENT DESCRIPTOR
39:20 = LENGTH FIELD - THE NUMBER OF WORDS IN THE SEGMENT
19:20 = ADDRESS FIELD - THE BEGINNING MEMORY ADDRESS IF [47:1] = 1.
- THE DISK OR PACK ADDRESS IF [47:1] = 0, AND [46:1] = 0.
- THE MEMORY ADDRESS OF THE ORIGINAL SEGMENT DESCRIPTOR IF [46:1] = 1, AND [47:1] = 0.

Figure 2-23. Segment Descriptor Word



50:3 = TAG FIELD.
(ALWAYS A VALUE OF 3 FOR A TOSCW)
47:1 = EXTERNAL SIGN FLIP-FLOP
46:1 = OVERFLOW FLIP-FLOP
45:1 = TRUE FALSE FLIP-FLOP
44:1 = FLOAT FLIP-FLOP
42:1 = TRUE FALSE OCCUPIED FLIP-FLOP
41:1 = COMPARE FLIP-FLOP
55:16 = DELTA S-REGISTER FIELD (VALUE OF THE S-REGISTER DISPLACEMENT ABOVE BOSR)
19:1 = NORMAL/CONTROL STATE FLIP-FLOP;
0 = NORMAL STATE
1 = CONTROL STATE
18:5 = LEXICOGRAPHICAL LEVEL
13:14 = DELTA F REGISTER FIELD (VALUE OF THE F-REGISTER DISPLACEMENT, BELOW THE VALUE OF THE S-REGISTER)
2:3 = THE CPU PROCESSOR ID VALUE (001) WHEN THE TOSCW IS FOR AN ACTIVE PROCESS PROGRAM OR TASK

Figure 2-24. TOSCW Word Layout

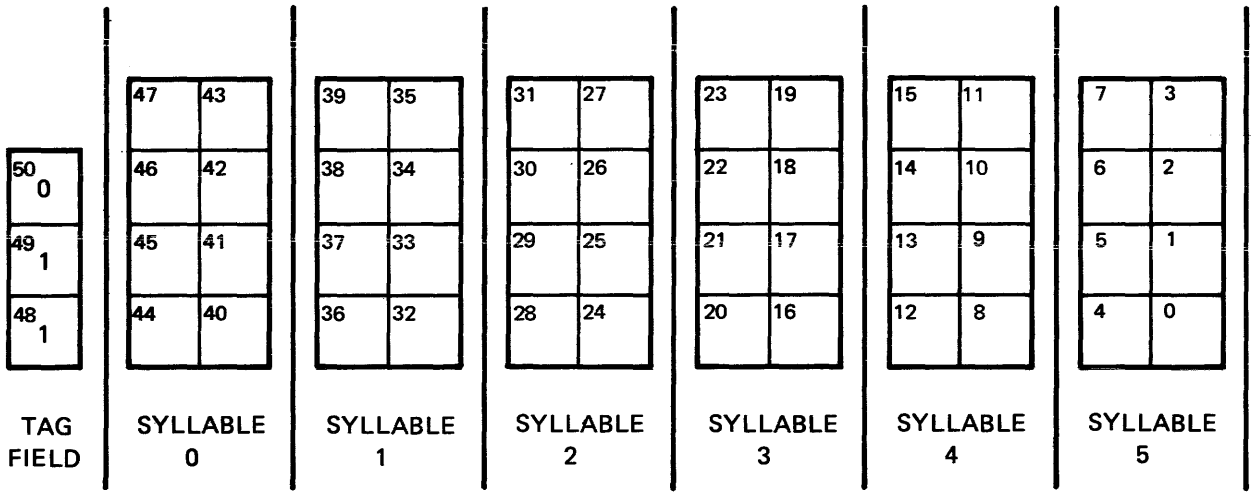


Figure 2-25. Program Word Format

SECTION 3

STACK AND REVERSE POLISH NOTATION

THE STACK

GENERAL

The stack is the memory storage area assigned to a job. The stack provides storage for the basic program and data references for the job. It also provides for temporary storage of data and job history. When a job is activated, four high-speed hardware registers (A, X, B, Y) are linked to the memory portion of the job's stack (see figure 3-1). This linkage is established by the stack pointer register (the S register), which contains the memory address of the last word placed in the stack. The four hardware top-of-stack registers (A, X, B, Y) extend the stack to provide quick access for data manipulation. Another stack pointer value (the F register) always points to the last valid MSCW in the stack.

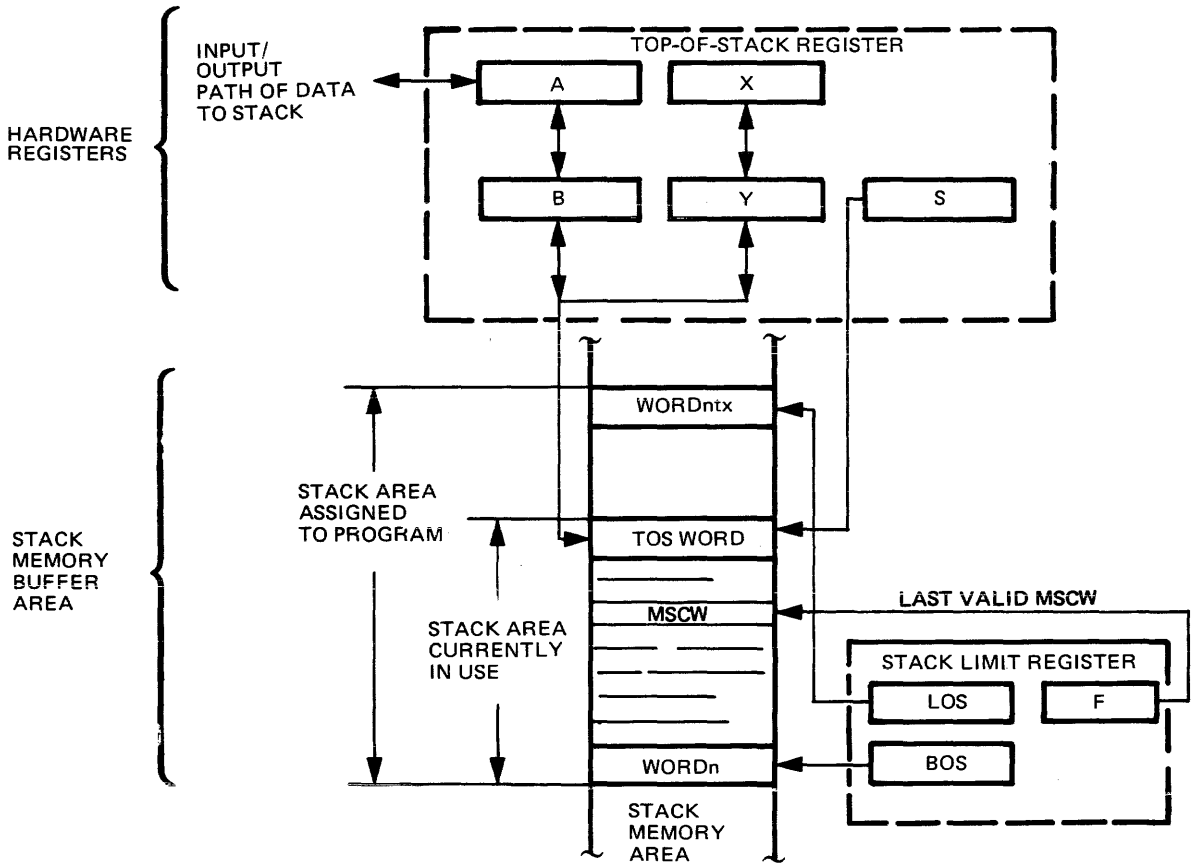


Figure 3-1. Top-of-Stack and Stack Bounds Registers

The number of words in the memory portion of the stack is equal to the difference between the values of the BOS register, and the S register (S minus BOS). Data are brought into the stack through the top-of-stack registers in such a manner that the last word placed in the stack (as indicated by the value of the S register) is the first word to be extracted from the stack (last in first out method). The total capacity of the top-of-stack registers is two words or two operands. Loading a third word or operand into the top-of-stack registers causes the first word or operand to be pushed from the top-of-stack registers into the memory portion of the stack. The stack pointer value in the S register is incremented by one as a word or operand is pushed into the memory portion of the stack, and is decremented by one when a word or operand is withdrawn from the stack area and placed in the hardware top-of-stack registers. As a result, the S register continually points to the last word or operand placed into the memory portion of the job stack.

BASE AND LIMIT OF STACK

A job's stack is bounded, for memory protection, by two registers: the base-of-stack register (BOSR) and the limit-of-stack register (LOSR). The contents of BOSR define the base of the memory portion of the stack, and the contents of LOSR define the upper limit of the memory portion of the stack. The job is interrupted if the S register is set to a value that is present in either the BOSR, or the LOSR register. If the S register equals or exceeds the value of the LOSR register value a stack overflow interrupt occurs.

BI-DIRECTIONAL DATA FLOW IN THE STACK

The contents of the top-of-stack registers are maintained automatically by the data processor to meet the requirements of the current machine language operator. If the current operator requires data transfer into the memory portion of the stack, the top-of-stack registers receive the incoming data, and surplus contents in the top-of-stack registers are pushed down into the memory portion of the stack. Pushing data into the memory portion of the stack means that the bottom word or operand in the top-of-stack register is transferred to the next word or operand in sequence, in the memory portion of the stack. Pushing data down into the memory portion of the stack makes room in the top-of-stack registers to contain the incoming data that is required by the current machine language operator.

Data are also automatically brought from the memory portion of the stack and placed in the top-of-stack registers when the machine language operator requires that the top-of-stack registers be filled. This automatic function is the opposite of the push function described in the previous paragraph, and is commonly called a push up function. A push up transfers the last operand or word in the memory portion of the stack into the second word position in the top-of-stack registers. The word or operand in the memory portion of the stack is then deleted by decrementing the S register. The automatic maintenance of the top-of-stack registers takes the form of "push down", and "push up" functions which are described in the following paragraphs.

Stack Push Down

A stack push down occurs when a third word or operand is loaded into the top-of-stack registers, and both the A register and B register already contain stack words or operands. A push down consists of moving data from the top-of-stack registers to the local memory portion of the stack. Moving data to the local memory portion of the stack makes room in the top-of-stack registers so that a third operand may be loaded into the top-of-stack registers.

Stack Push Up

A stack push up occurs when an operand or word is moved from the local memory portion of the stack, to the top-of-stack register portion of the stack. A push up can only occur when a machine language operator is executed by the data processor. The data processor operator that is to be performed must require that words or operands be present in the top-of-stack registers, and such words or operands must not be present in the proper top-of-stack registers.

DOUBLE PRECISION STACK OPERATION

The top-of-stack registers are operand oriented rather than word oriented. Calling a double precision operand into the top-of-stack registers causes two memory words to be loaded into the top-of-stack registers. The first word is loaded into the A register, where it's tag bits are checked. If the value indicates double precision the second word is loaded into the X register. The A and X registers are concatenated, or linked together, to form the double precision operand. The B and Y registers concatenate when a double precision operand reverts to single words as it is pushed from the B and Y registers into the memory portion of the stack. The concatenation is repeated when a double precision operand is returned from the memory portion of the stack to the top-of-stack registers.

TOP-OF-STACK REGISTER CONDITIONS

Two logical indicators are used to indicate the condition of the top-of-stack register portion of the stack. These two indicators are AROF (A register is occupied flip-flop), and BROF (B register is occupied flip-flop). The meaning of these two logical indicators is as follows:

AROF	BROF	MEANING
0	0	Neither the A, or the B register contains valid data. The top word in the stack is presently located in the local memory address specified by the contents of the S register.
0	1	The B register contains the top word in the stack, and the contents of the A register are not valid data. The second word in the stack is presently located in the local memory address specified by the contents of the S register.
1	0	The A register contains the top word in the stack, and the contents of the B register are not valid data. The second word in the stack is presently located in the local memory address specified by the contents of the S register.
1	1	The A register contains the top word in the stack, and the second word in the stack is presently in the B register. The third word in the stack is in the local memory address specified by the contents of the S register.

STACK ADJUSTMENTS

Each machine language operator that is executed by the data processor contains the requirement to adjust the top-of-stack registers so that their contents provide accomodation for the operation that is to be performed. A convention is used to show what stack adjustment is required, as follows:

CONVENTION NOTATION	MEANING
(ADJ 0,0)	Both the A and B registers are to be adjusted so that their contents are not valid. The top word in the stack is to be located in the local memory address pointed at by the contents of the S register. The data processor will use the state of the AROF and BROF flip-flops to determine if the stack must be pushed down to achieve the required adjustment. The 0,0 portion of the convention notation shows what the logical states of AROF and BROF must be to satisfy the requirements of the adjustment. The first 0 in the expression of the notation defines what the logical state of the AROF flip-flop must be at the conclusion of the stack adjustment. The second 0 in the expression defines what the logical state of the BROF flip-flop must be at the conclusion of the adjustment. The ADJ portion of the convention notation reads "adjust the stack until AROF and BROF meet the logical states".
(ADJ 0,1)	The A register is to be adjusted so that its contents are not valid. The top word or operand in the stack is to be present in the B register, and the second word or operand in the stack is to be located in the local memory address pointed at by the contents of the S register.
(ADJ 1,0)	The A register is to be adjusted so that its contents are the top word or operand in the stack. The B register must not contain valid data. The second word or operand in the stack is to be located in the local memory address pointed at by the contents of the S register.
(ADJ 1,1)	The A register is to be adjusted so that it contains the top word or operand in the stack. The B register is to be adjusted so that it contains the second word or operand in the stack. The third word or operand in the stack is to be in the local memory address pointed at by the contents of the S register.
(ADJ 0,2)	The A register is to be adjusted so that its contents are not valid. The B register condition is immaterial to the operation. The top word in the stack is present in the B register if BROF is set.
(ADJ 1,2)	The A register is to be adjusted so that it contains the top word in the stack. The B register condition is immaterial to the operation. The second word in the stack is located in the B register if BROF is set.
(ADJ 1,3)	The A register is adjusted so that it contains the top word in the stack if and only if the original stack condition is AROF/ and BROF/ (0,0). If any other condition than (0,0) is the original condition, then no stack adjustment occurs.

Some machine language operations require that several stack adjustments must be performed during the course of the operation. Such operations merely pause at the appropriate place until the adjustment is completed, and then continue the sequence.

Stack push down and/or stack push up (which were defined previously in this section) are intrinsic functions of the stack adjustments. That is, a push-up or a push-down may be implied because of the current state of the top of stack registers, and the required stack adjustment. Where a stack push-up or push-down is implied, such operation will be performed as an integral and automatic function of the stack adjustment procedure.

DATA ADDRESSING

The B 6800 data processor provides three methods for addressing data or program code:

a. Data descriptor (DD)/segment descriptor (SD)

b. Indirect reference word (IRW)

c. Stuffed indirect reference word (SIRW)

The data descriptor (DD) and segment descriptor (SD) provide for the addressing of data or program segments located outside of the job's stack area. The indirect reference word (IRW) and the stuffed indirect reference word (SIRW) address data located within (IRW), or outside (SIRW) the job's stack. The IRW and SIRW address components are both relative. The IRW addresses within the immediate environment of the job relative to a display register (described later in Non-local Addressing). The SIRW addresses beyond the immediate environment of the current procedure, the addressing being relative to the base of the job's stack. Addressing across stacks is accomplished with an SIRW.

Data Descriptor

In general, the descriptor describes and locates data associated with a given job. The data descriptor (DD) is used to fetch data to the stack or to store data from the stack into an array located outside the job's stack area. The formats of the data and segment descriptors were illustrated in section 2. The address field in each of these descriptors is 20 bits in length; this field contains the absolute address of an array in either local memory or in the disk file, as indicated by setting of the presence bit (P). The referenced data is in main memory when the presence bit is set.

Presence Bit

A presence bit interrupt occurs when the job references data by means of a descriptor in which the P-bit is equal to 0; i.e., the data is located in a disk file, rather than in local memory. The Master Control Program (MCP) recognizes the presence bit interrupt and transfers data from disk file storage to local memory. After the data transfer to local memory is completed, the MCP marks the descriptor present by setting the P-bit to 1, and places the new local memory address into the address field of the descriptor. The interrupted job is then reactivated.

Index Bit

A data descriptor describes either an entire array of data words, or a particular element within an array of data words. If the descriptor describes the entire array, the index bit (1-bit) in the descriptor is 0, indicating that the descriptor has not yet been indexed. The length field of the descriptor defines the length of the data array.

Invalid Index

A particular element of an array is described by indexing an array descriptor. Memory protection is ensured during indexing operations by performing a comparison between the length field of the descriptor and the index value. An invalid index interrupt results if the index value exceeds the length of the local memory area defined by the descriptor, or if the index is less than 0.

Valid Index

If the index value is valid, the length field of the descriptor is replaced by the index value, and the I-bit in the descriptor is set to 1 to indicate that indexing has taken place. The address and index fields are added together to generate the absolute machine address whenever an indexed data descriptor in which the P-bit is set is used to fetch or store data.

The double-precision bit (D) is used to identify the referenced data as single- or double-precision and directly affects the indexing operation. The D-bit equal to 1 signifies double-precision and causes the index value to be doubled before indexing.

Read-Only Bit

The read-only bit (R) specifies that the local memory area described by the data descriptor is read-only area. If the R-bit of a descriptor is set to 1, and the area referenced by that descriptor is used for storage purposes, an interrupt results.

Copy Bit

The copy bit (C) identifies a descriptor as a copy of a master descriptor and is related to the presence-bit action. The copy bit links multiple copies of an absent descriptor (i.e., the presence bit is off) to the one master descriptor. The copy bit mechanism is invoked when a copy is made in the stack. If it is a copy of the original, absent descriptor, the processor sets the copy bit to 1 and inserts the address of the master descriptor into the address field. Thus, multiple copies of absent data descriptors are all linked back to the master descriptor.

REVERSE POLISH NOTATION

GENERAL

Reverse polish notation is an arithmetical or logical notational system using only operands and operators arranged in sequence or strings, thus eliminating the necessity for defining the boundaries of any terms. Figure 3-2 presents a flow chart for conversion to reverse polish notation.

SIMPLIFIED RULES FOR GENERATION OF POLISH STRING

The source of expression is as follows:

Name	Action
Variable or constant	Place variable or constant in string being built and examine next symbol.
Operator-separator "(" or "["	Place in delimiter list and examine next symbol.
Arithmetic or Boolean operator and last-entered delimiter list symbol were as follows: 1. An operator of lower priority. 2. A left bracket "[" or parenthesis "(". 3. A separator. 4. Nothing (delimiter list empty).	Place operator in the delimiter list and examine next source symbol.
Arithmetic or Boolean operator and last-entered delimiter list symbol were as follows: an operator of priority equal to or greater than the symbol in the source.	Remove the operator from the delimiter list and place it in the string being built. Then compare the next symbol in the delimiter list against the source expression symbol.

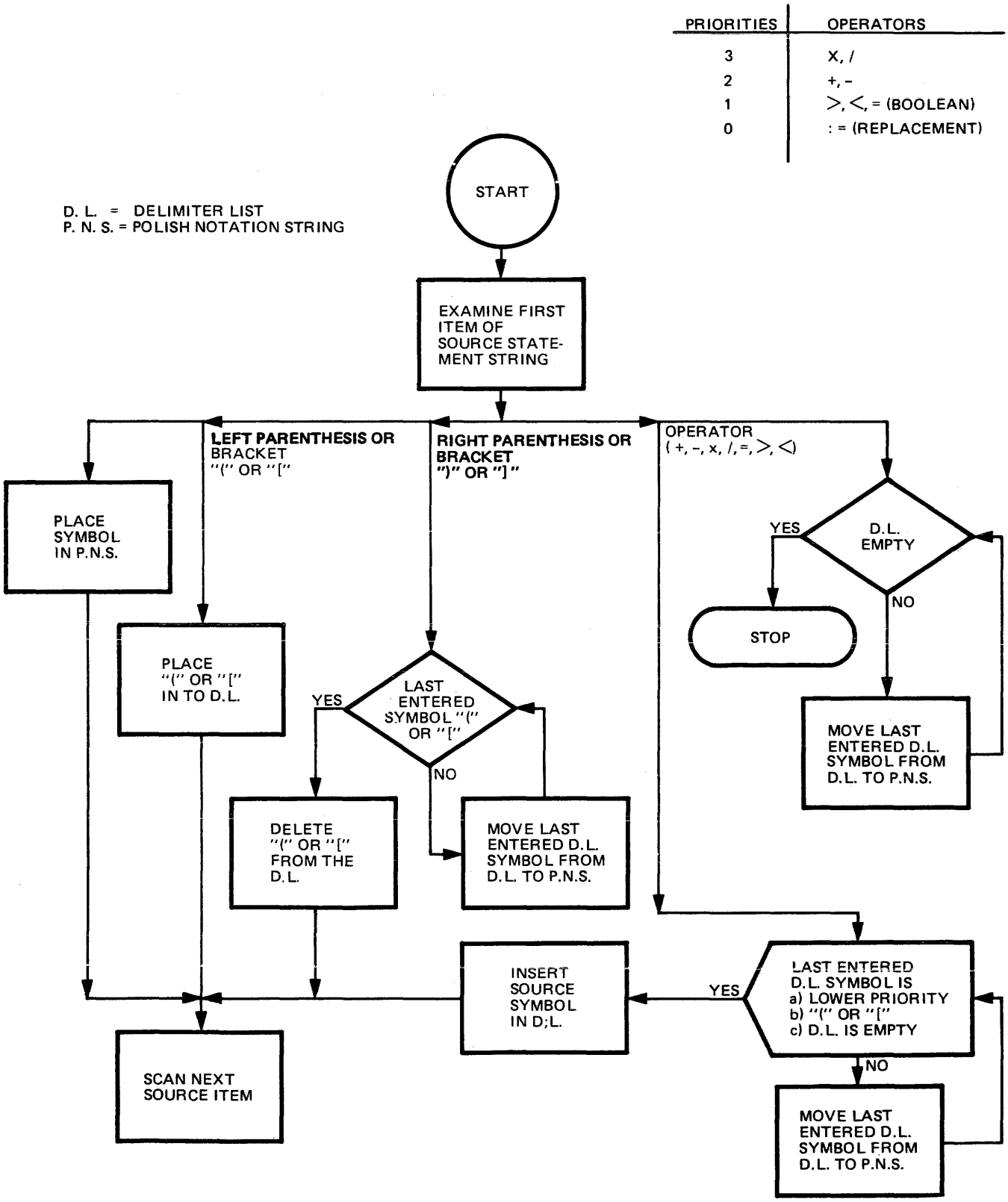


Figure 3-2. Reverse Polish Notation Flow Chart

Name	Action
A right bracket "]" or parenthesis ")".	Pull from delimiter list until corresponding left bracket or parenthesis.
End of expression.	Move last-entered delimiter list symbols to Polish notation string until empty.

POLISH STRING

The essential difference between reverse polish and conventional notation is that operators are written to the right of the operands instead of between them. For example, the conventional B + C is written B C + in reverse polish notation: A = 7 x (B + C) becomes A 7 B C + x :=.

Any expression written in reverse polish notation is called a polish string. In order to fully understand this concept, the user should know the rules for evaluating a polish string.

RULES FOR EVALUATING A POLISH STRING

The following is the procedure for evaluating a polish string:

a. Scan the string from left to right.

b. Remember the operands and the order in which they occur.

c. When an operator is encountered perform the following:

   1. Record the last two operands encountered.

   2. Execute the required operation.

   3. Disregard the two operands.

   4. Consider the result of (b) above as a single operand, the first of the next pair to be operated upon.

Following this rule, the reverse polish string A 7 B C + x := results in A assuming the value 7 x (B+C) (table 3-1).

NOTE

Because replacement operators vary depending upon the language used, ←, =, and := are equivalent for this discussion.

SIMPLE STACK OPERATION

All program information must be in the system before it can be used. Input areas are allocated for information entering the system and output areas are set aside for information exiting the system; array and table areas are also allocated to store certain types of data. Thus data is stored in several different areas: the input/output areas, data tables (arrays), and the stack. Since all work is done in the arithmetic registers, all information or data is transferred to the arithmetic registers and the stack.

Table 3-1. Evaluation of Polish String A 7 B C + x :=

Step No.	Symbol Being Examined	Symbol Type	Operands Being Remembered Order of Occurrence (1 or 2) Before Operation	Occurring Operation	Operation Results
1	B	Operand			
2	C	Operand	1 B		
3	+	Add Operator	2 C 1 B	B + C	(B + C)
4	7	Operand	1(B + C)		
5	x	Multiply Operator	2 7 1 x (B + C)	7 x (B + C)	7 x (B + C)
6	A	Name	1 7 x (B + C)		
7	=	Replace Operator	2 A 1 7 x (B + C)	A :=7x(B + C)	A=7x(B + C)

At this point, an ALGOL assignment statement and the reverse polish notation equivalent will be related to the stack concept of operation. The example is Z:=Y + 2x(W+V), where := means "is replaced by." In terms of a computer program, this assignment statement indicates that the value resulting from the evaluation of the arithmetic expression is to be stored in the location represented by the variable Z.

When Z:=Y + 2x(W+V) is translated to reverse polish notation, the result is ZY2WV+ x +:=. Each element of the example expression causes a certain type of syllable to be included in the machine language program when the source problem is compiled. The following is a detailed description of each element of the example, the type of syllable compiled, and the resulting operation (see figure 3-3 and table 3-2).

In the example statement, Z is to be the recipient of a value, the address of Z must be placed into the stack just prior to the store command. This is accomplished by a name call syllable which places an indirect reference word (IRW) in the stack. The IRW contains the address of Z in the form of an "address couple" that references the memory location reserved in the stack for the variable Z.

Since Y is to be added to a quantity, Y is brought into the top of the stack as an operand. This is accomplished with a value call (VALC) syllable that references Y. The value 2 is then brought to the stack, with an eight-bit literal syllable (LT8). Since W and V are to be added, the respective variables are brought to the stack with value call syllables. The ADD operator adds the two top operands and places the sum in the top of stack. This example assumes, for simplicity, single-precision operands not requiring use of the X and Y registers which are used in double-precision operations.

The multiply operator is the next symbol encountered in the reverse polish string; when executed, it places the product "2x(W+V)" in the top of the stack. The next symbol, ADD, when executed, leaves the final result "7+2x(W+V)" in the top of the stack.

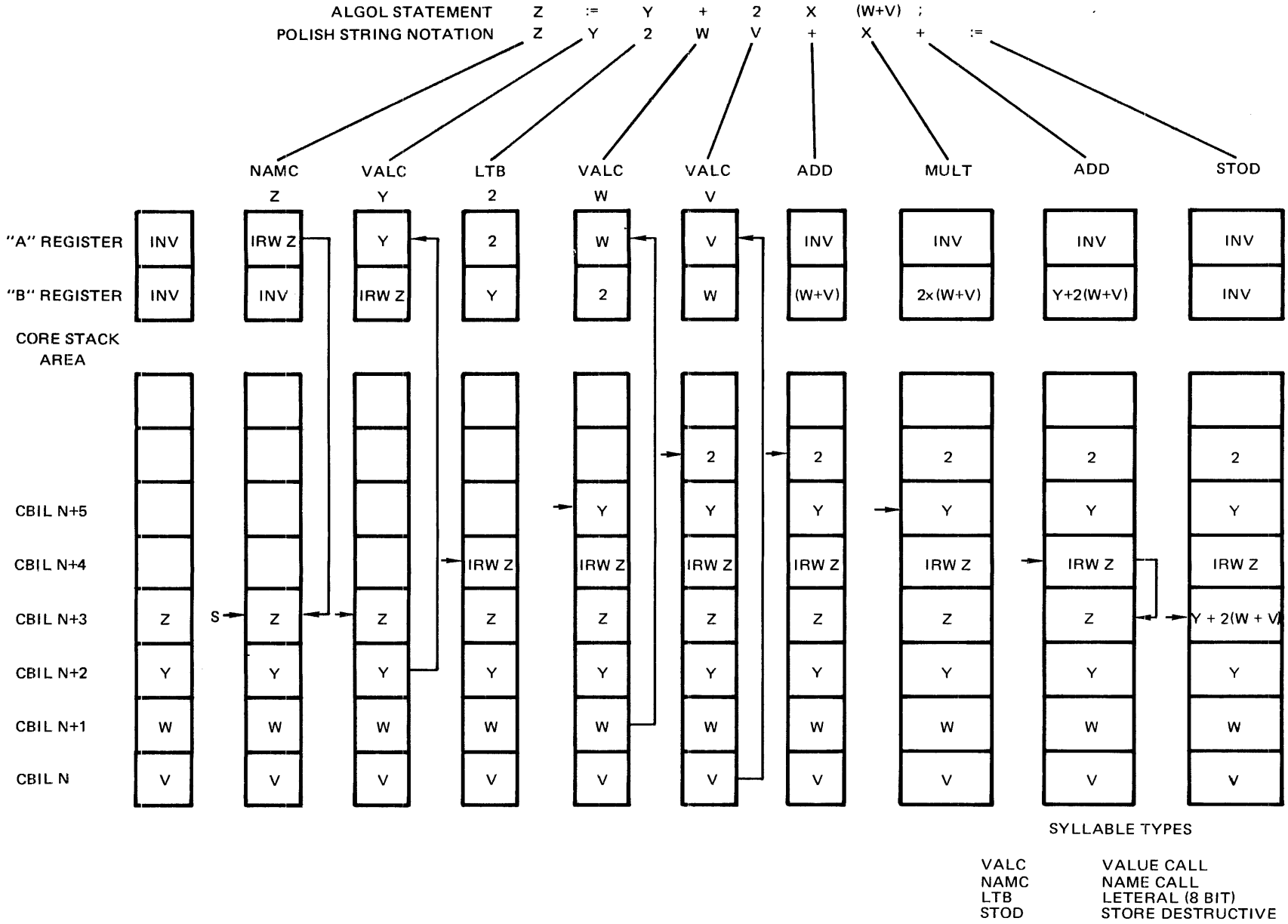


Figure 3-3. Stack Operation

Table 3-2. Description of Stack Operation

Execution Sequence	Reverse Polish Notation Element	Syllable Type Complied	Function of Syllable During Running of the Program
0			Stack location of program variables illustrated
1	Z	Name call for Z	Build an indirect reference word that contains the address of Z and place it in the top of the stack
2	Y	Value call for Y	Place the value of Y in the top of the stack
3	2	Literal 2	Place a 2 in the top of the stack
4	W	Value call for W	Place the value of W in the top of the stack
5	V	Value call for V	Place the value of V in the top of the stack
6	+	Operator add	Add the two top words in the stack and place the result in B register as the top of the stack
7	x	Operator multiply	Multiply the two top-of-the-stack operands. The product is left in the B register as the top of the stack
8	+	Operator add	Add the two top words in the stack and leave the result in the B register as the top of the stack
9	:=	Operator store destructive	Store an item into memory. The address in which to store is indicated by an indirect reference word or a data descriptor; the address can be above or below the item stored

The store syllable completes the execution of the statement Z:=Y + 2x(W+V). The store operation examines the two top-of-stack operands looking for an IRW or data descriptor. In this example, the IRW addresses the location where the computed value of Z is to be stored. The stack is empty at the completion of this statement.

PROGRAM STRUCTURE IN LOCAL MEMORY

When a problem is expressed in a source language, portions of the source language fall into one of two categories. One describes the constants and variables that will be used in the program, and the other the computations that will be executed (refer to figure 3-4). When the source program is compiled, variables are assigned locations within the stack whereas the constants are embedded within the code stream that forms the computational part. A program residing in memory occupies separately allocated areas. "Separately allocated" means that each part of the program may reside anywhere in memory, and the actual address is determined by the MCP. In particular, the various areas are not assigned to contiguous memory areas. Registers within the processor indicate the bases of the various areas during the execution of a program.

LOCAL MEMORY AREA ALLOCATION

The separately allocated areas of a program are as follows:

a. Program Segments: These are sequences of instructions (syllables) that are performed by the processor in executing the program. Note that there is a distinction between program segments and data areas. The program segments contain no data, and are not modified by the processor as it executes the program.

b. Segment Dictionary: This is a table containing one word for each program segment. This word tells whether the program segment is in local memory or on the disk, and gives the corresponding local memory or disk address of the program segment.

c. Stack Area: This is the pushdown stack storage, which contains all the variables associated with the program, including control words which indicate the dynamic status of the job as it is being executed.

STACK-HISTORY AND ADDRESSING-ENVIRONMENT LISTS

One very important aspect of the B 6800 is the retention of the dynamic history for the program being processed. Two lists of program history are maintained in the B 6800 stack, the stack-history list and the addressing-environment list. The stack-history list is dynamic, varying as the job proceeds along different program paths with changing sets of data. Both lists are generated and maintained by B 6800 hardware.

MARK STACK CONTROL WORD LINKAGE

The stack history is a list of Mark Stack Control Words (MSCW), linked together by their displacement fields (DF) (figure 3-5). An MSCW is inserted into the stack as a procedure is entered and is removed as that procedure is exited. Therefore, the stack history list grows and contracts with the procedural depth of the program. Mark stack control words identify the portion of the stack related to each procedure. When the procedure is entered, its parameters and local variables are entered in the stack following the MSCW. When the procedure is executed its parameters and local variables are referenced by addressing relative to the MSCW.

STACK DELETION

Each MSCW is linked to the prior MSCW through the contents of its DF field in order to identify the point in the stack where the prior procedure began. When a procedure is exited, its portion of the stack is discarded. This action is achieved by setting the stack-pointer register (S) to address the memory cell preceding the most recent MSCW (figure 3-6). This topmost MSCW, addressed by another register (F), is deleted from the stack-history list by changing F to address the prior MSCW, placing this MSCW at the head of the stack history.

This is an efficient and convenient means of subroutine entry and exit.

RELATIVE-ADDRESSING

Analyzing the structure of an ALGOL program results in a better understanding of the relative-addressing procedures used in the B 6800 stack. The addressing environment of an ALGOL procedure is established when the program is structured by the programmer and is referred to as the lexicographical ordering of the procedural blocks (figure 3-7). At compile time, the lexicographical ordering is used to form address couples. An address couple consists of two items:

1. The addressing level (ℓℓ) of the variable

2. An index value (S) used to locate the specific variable within its addressing level

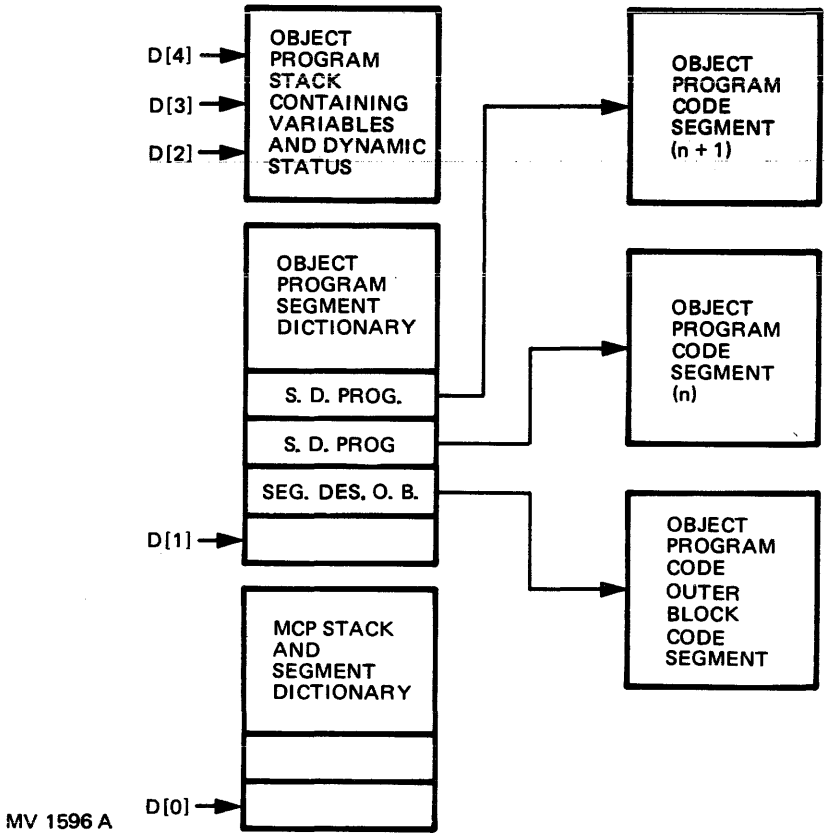


Figure 3-4. Object Program in Memory

The lexicographical ordering of the program remains static as the program is executed, thereby allowing variables to be referenced via address couples as the program is executed.

Base of Address Level Segment

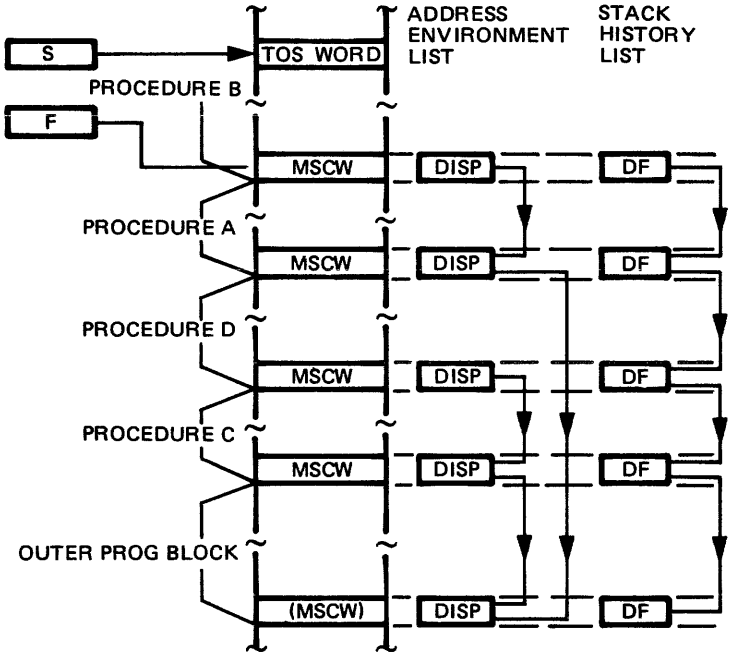
The B 6800 processor contains an array of D registers (D0 through D31). These registers address the base of each addressing-level segment (figure 3-8). The local variables of all procedures are addressed relative to the D registers.

Absolute Address Conversion

The address couple is converted into an absolute memory address when the variable is referenced. The addressing level portion of the address couple selects the D register which contains the absolute memory address of the MSCW for the environment (addressing level) in which the variable is located. The index value of the address couple is added to the contents of the D register to generate the absolute memory address.

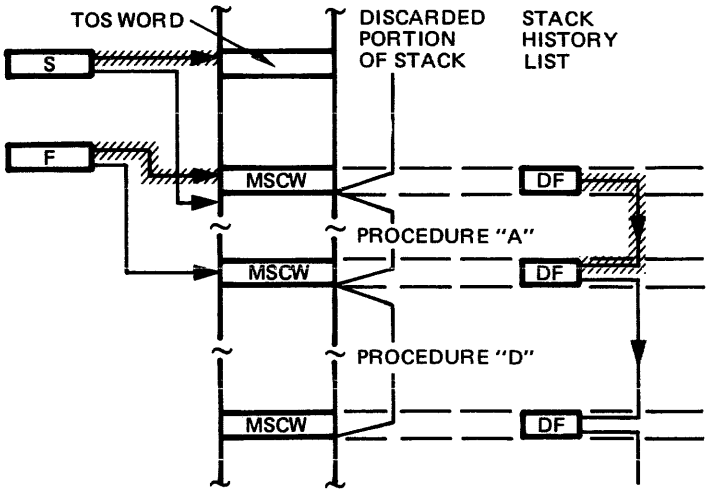
Multiple Variables With Common Address Couples

The address couples assigned to the variables in a program are not unique. This is true because of the ALGOL scope-of-definition rules, which imply that if there is no procedure which can address both of any two quantities, then these two quantities may unambiguously have the same address couple. This addressing system works because, whereas two variables may have the same address couples, there is never any doubt as to which variable is being referenced within any particular procedure.



MV 1597

Figure 3-5. Stack History and Addressing Environment List



MV 1598

Figure 3-6. Stack Cut-Back Operation on Procedure Exit

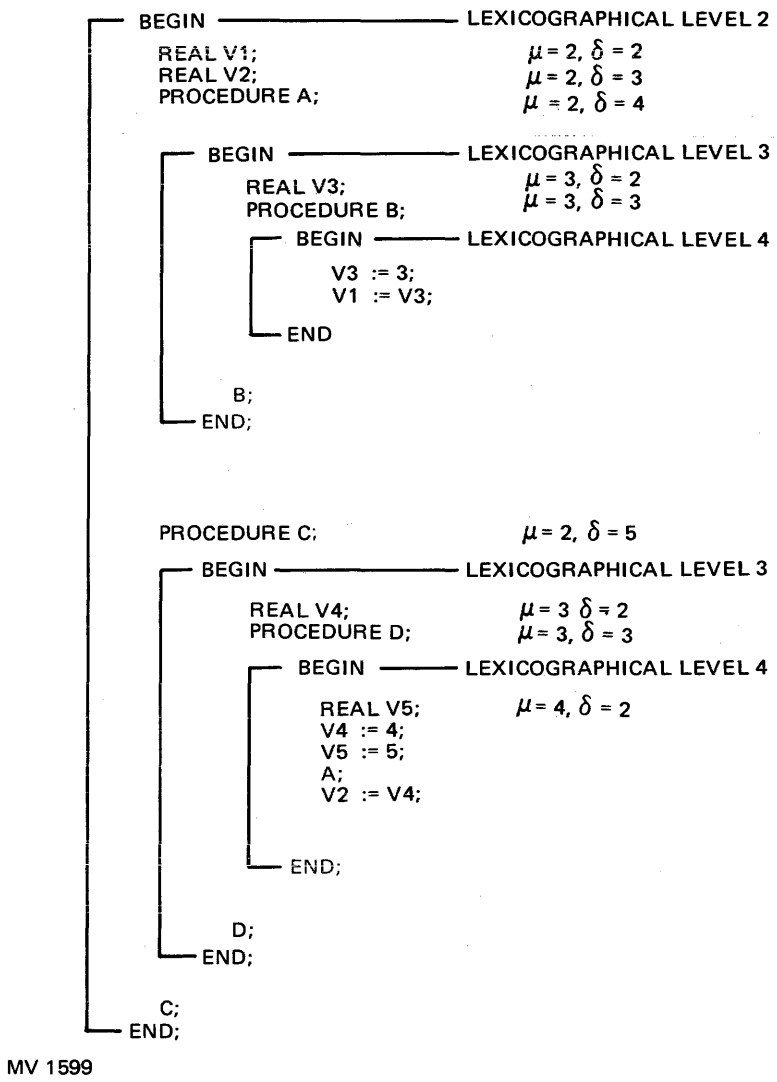


Figure 3-7. ALGOL Program With Lexicographical Structure Indicated

Address Environment Defined

There is a unique MSCW which each D register must address during the execution of any particular procedure. The D registers must be changed, upon procedure entry or exit, to address the correct MSCWs. The list of MSCWs which the D registers address is the addressing environment of the procedure.

Mark Stack Control Word Linkage

The addressing environment of the program is maintained automatically by linking the MSCWs together in accordance with the lexicographical structure of the program. This linkage is the stack number (Stack No.) and displacement (DISP) fields of the MSCW, and is inserted into the MSCW whenever the procedure is entered. The addressing environment list is formed by linking each MSCW to the MSCW immediately below the declaration for the procedure being entered. This forms a tree-structured list which indicates the addressing environment of each procedure (figures 3-8 and 3-9). This list is used to update the D registers whenever a procedure entry or exit occurs.

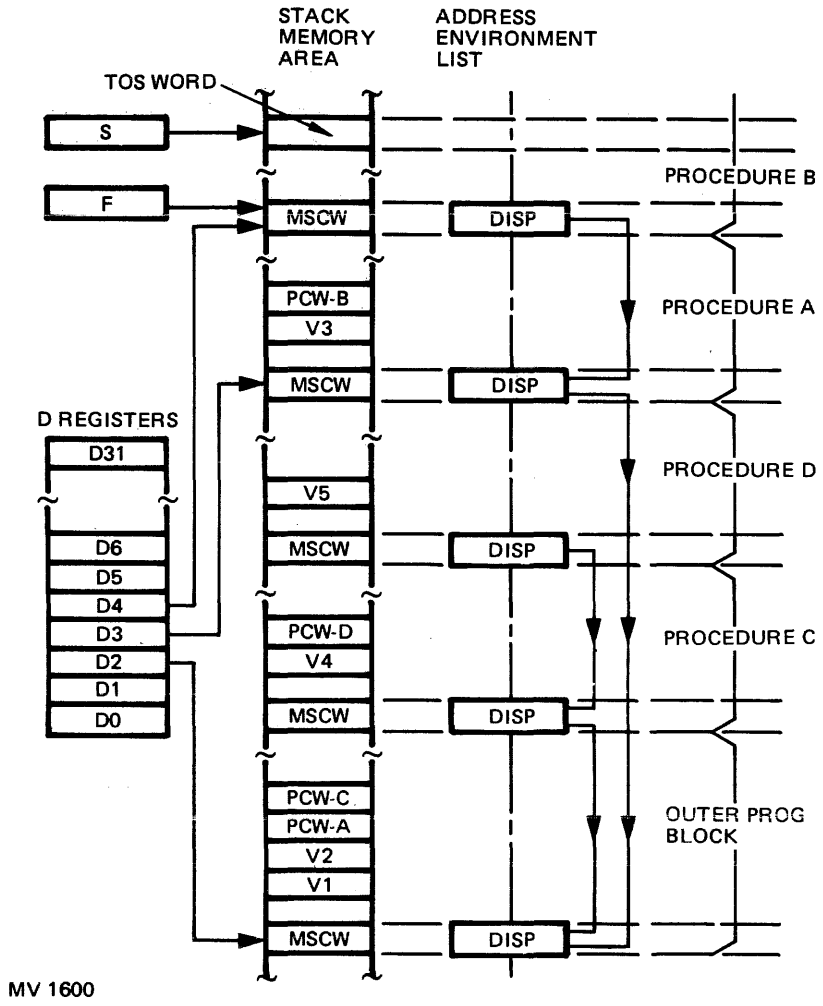


Figure 3-8. D Registers Indicating Current Addressing Environment

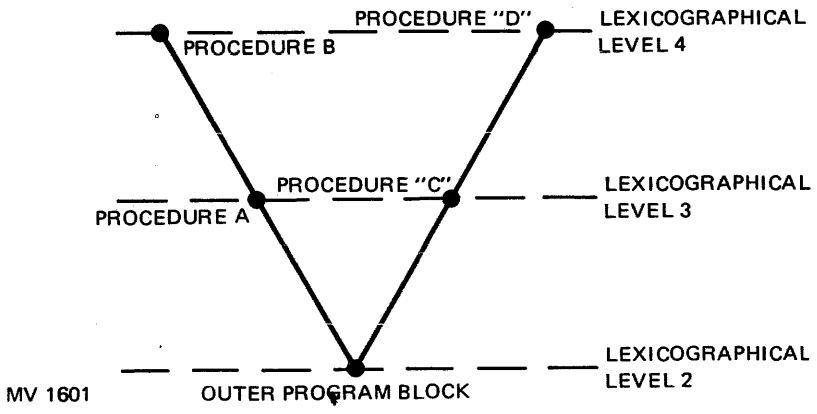


Figure 3-9. Addressing Environment Tree of ALGOL Program

STACK HISTORY SUMMARY

The entry and exit mechanism of the processor hardware automatically maintains both the stack history and address-environment lists to reflect the current status of the program. Interrupt response is a procedure entry. Therefore, the system is able to conveniently respond to, and return from, interrupts. Upon recognition of an interrupt condition, the processor creates a MSCW, inserts an indirect reference word into the stack to address the interrupt-handling procedure, inserts a literal constant to identify the interrupt condition and two other parameters, and initiates an MCP interrupt-handling procedure. The D registers are updated upon entry into the interrupt-handling procedure, to display all legitimate variables. Upon return from this procedure, the D registers are updated to display variables of the former procedure.

MULTIPLE STACKS AND REENTRANT CODE

The B 6800 stack mechanism provides a facility for handling several active stacks, which are organized in a tree structure. The trunk of this tree structure is a stack containing MCP global quantities.

LEVEL DEFINITION

A program is a set of executable instructions, and a job is a single execution of a program for a particular set of data. As the MCP is requested to run a job, a level-1 branch of the basic stack is created. This level-1 branch contains the descriptors pointing to the executable code and read-only data segments for the program. Emerging from this level-1 branch is a level-2 branch, containing the variables and data for this job. Starting from the job's stack and tracing downward through the tree structure, one finds first the stack containing the variables and data for the job (at level 2), the segment descriptor to be executed (at level 1), and the MCP's stack at the trunk (level 0).

REENTRANCE

A subsequent request to run another execution of an already-running program requires that only a level-2 branch be established. This level-2 stack branch emerges from the level-1 stack of the already-running program. Thus two jobs which are different executions of the same program have a common node, at level-1, describing the executable code. It is in this way that program code is re-entrant and shared. This results simply from the proper tree-structured organization of the various stacks within the machine. All programs within the system are re-entrant, including all user programs as well as the compilers and the MCP.

JOB-SPLITTING

The B 6800 stack mechanism also provides the facility for a single job to split itself into two independent jobs. A common use of this facility occurs when there is a point in a job where two relatively large independent processes must be performed. This splitting can be used to make full use of a multiprocessor configuration, or to reduce elapsed time by multiprogramming the independent processes.

A split of this type establishes a new limb of the tree-structured stack, with the two independent jobs sharing that part of the stack which was created before the split was requested. The process is recursively defined and can happen repeatedly at any level.

STACK DESCRIPTOR

Stack branches are located by an array of descriptors, the stack vector array (figure 3-10). There is a data descriptor in this array for every stack branch. This data descriptor, the stack descriptor, describes the length of the memory area assigned to a stack branch and its location in either local memory or disk.

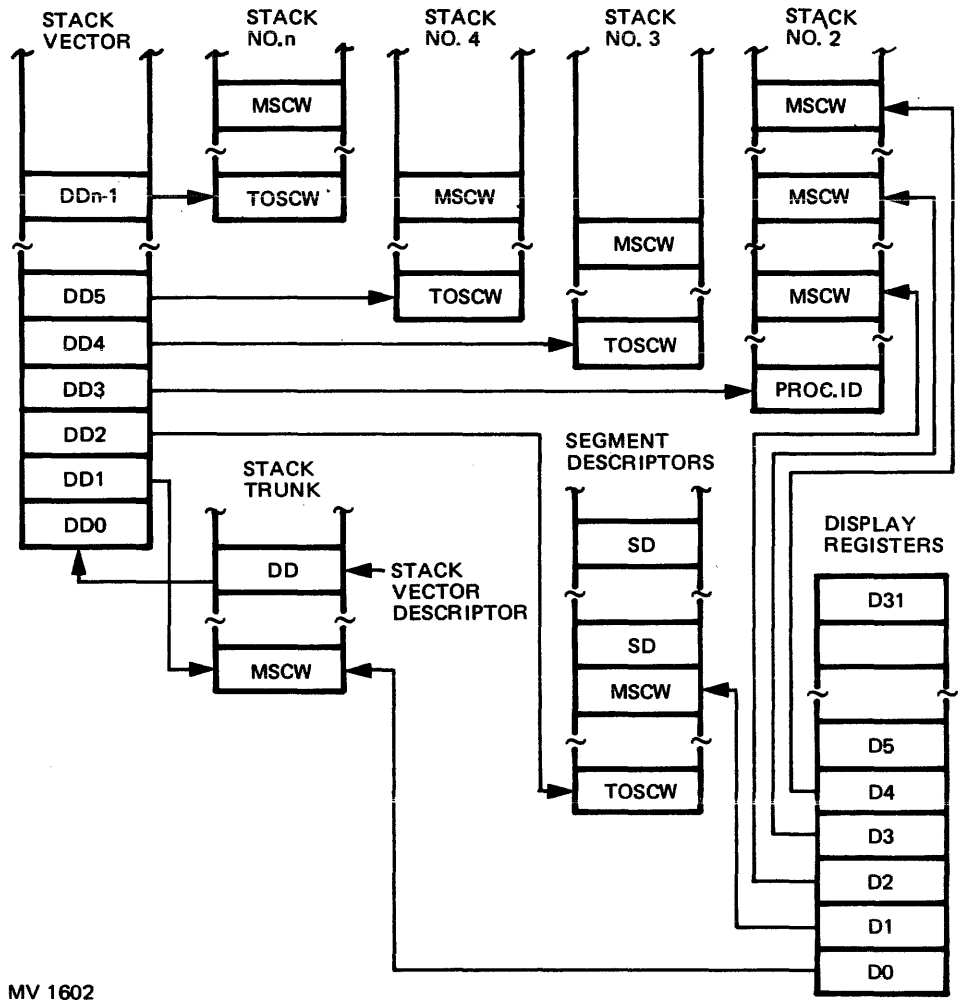


Figure 3-10. Multiple Linked Stacks

A stack number is assigned to each stack branch. The stack number is the index value of the stack descriptor in the stack vector array.

STACK VECTOR DESCRIPTOR

The array size of the stack vector and its location in memory is described by the stack vector descriptor, located in a reserved position of the trunk of the stack (figure 3-10). All references to stack branches are made through the stack vector descriptor, indexed by the stack number.

PRESENCE BIT INTERRUPT

A presence bit interrupt results when an addressed stack is not present in memory. This presence bit interrupt facility permits stack overlays and recalls under dynamic conditions. Idle or inactive stacks may be moved from main memory to disk as the need arises and, when a stack is subsequently referenced, a presence bit interrupt is generated to cause the MCP to recall the non-present stack from disk.

SECTION 4

MAJOR REGISTERS AND CONTROL PANELS

DISPLAY PANELS

The B 6800 system registers and flip flops are displayed on the display section of the MDP cabinet (see figure 4-1). The display section of the B 6800 MDP cabinet is divided into the programmers display panel and the maintenance display panel. These two panels occupy that portion of the MDP cabinet that is above the keyboard panel. The programmers display panel (the left front half of the MDP display section) contains those system controls that are relevant to the programmers display. The maintenance display panel (the right front half of the MDP display section) contains those system controls that are relevant to the maintenance display.

PROGRAMMERS DISPLAY PANEL

The programmers display panel (see figure 4-2) is divided into three sections which are the system control panel, the status display panel, and the register display panel.

System Control Panel

The top row of indicators on the programmers display panel are the system controls (refer to figure 4-2). The system controls consist of eight combination indicator lamps and/or push button switches. The use and meaning of each of these system controls is as follows:

HALT switch and indicator

When depressed, causes the system to halt at the end of the current instruction. When the system is halted, the HALT indicator is illuminated.

LOAD switch

When depressed, causes a general clear. When released, causes a LOAD signal to be sent to the mainframe.

CARD LOAD SELECT switch and indicator

When depressed, causes the card load select flip flop in the display control logic to change state. The output of the flip flop goes to the mainframe and the CARD LOAD SELECT indicator. When true, card load has been selected and the indicator is on. When false, disk has been selected and the indicator is off.

RUNNING indicator

This indicates that the system is running.

GENERAL CLEAR switch

When depressed, causes the systems flip flops to be reset.

POWER FAIL indicator

The power failure lamp illuminates if any over voltage condition is sensed by any of the B 6800 system power supplies except the MDP power supply. The power failure lamp will also illuminate if 2 regulators in each planar memory cabinet (one regulator for each of two memory modules) senses an under voltage condition. The B 6800

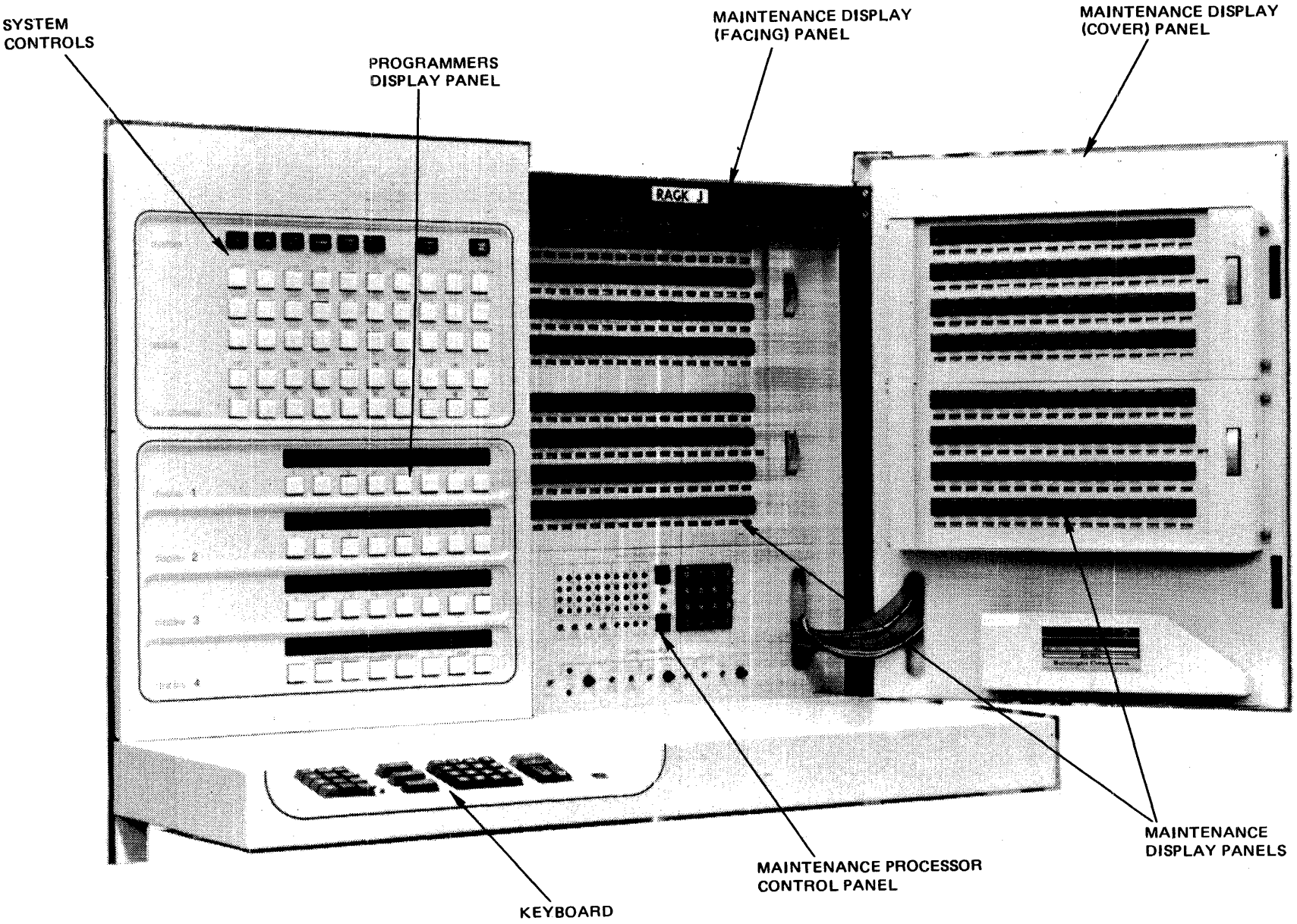
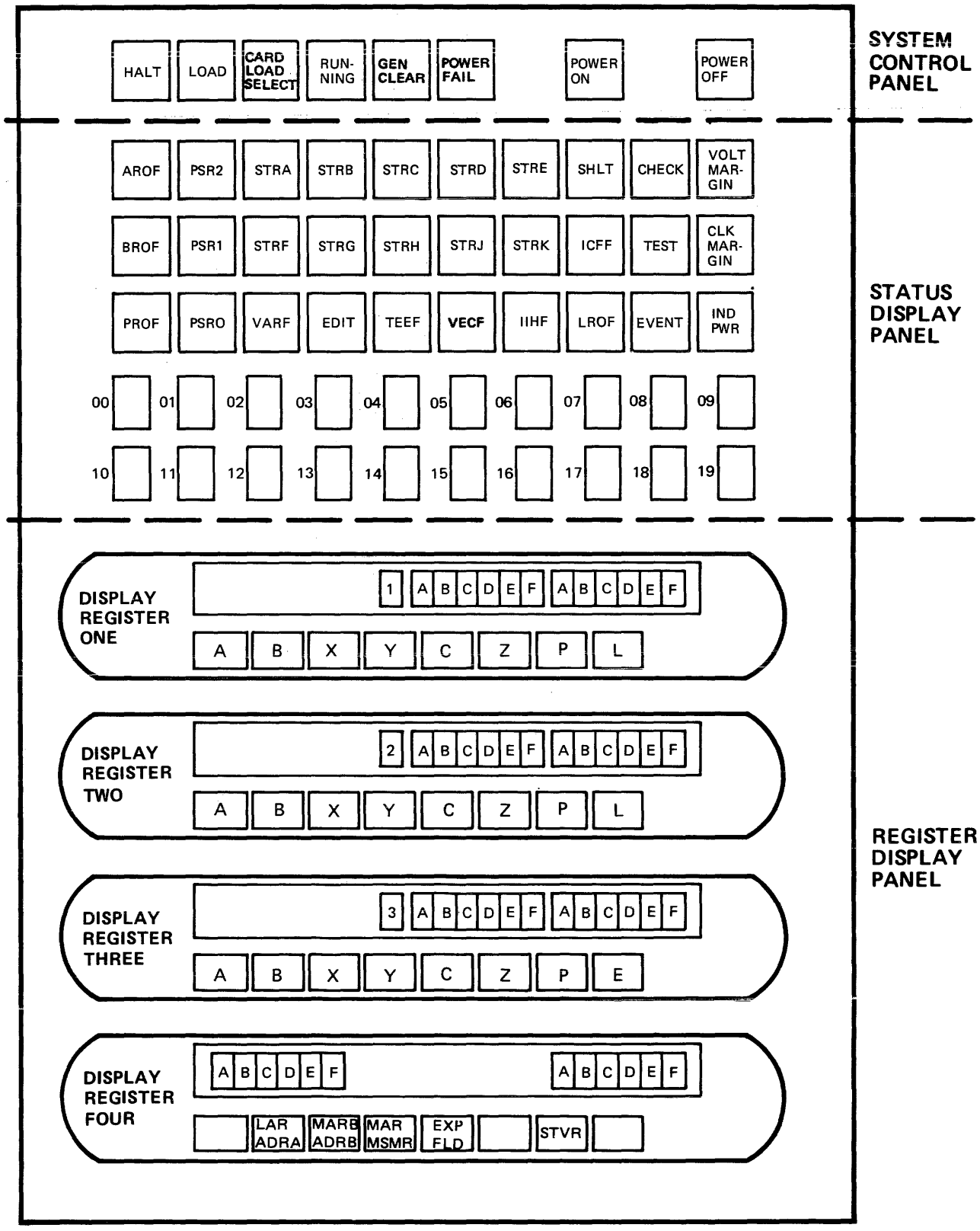


Figure 4-1. System Control and Display Registers



MV1604A

Figure 4-2. Programmers Display Panel

system will detect a system over voltage power fault with this condition, even though the actual cause of the fault is an under voltage condition.

POWER ON switch and indicator

When depressed, this switch initiates a power up sequence. The power on indicator is illuminated while power is applied to the system.

POWER OFF switch

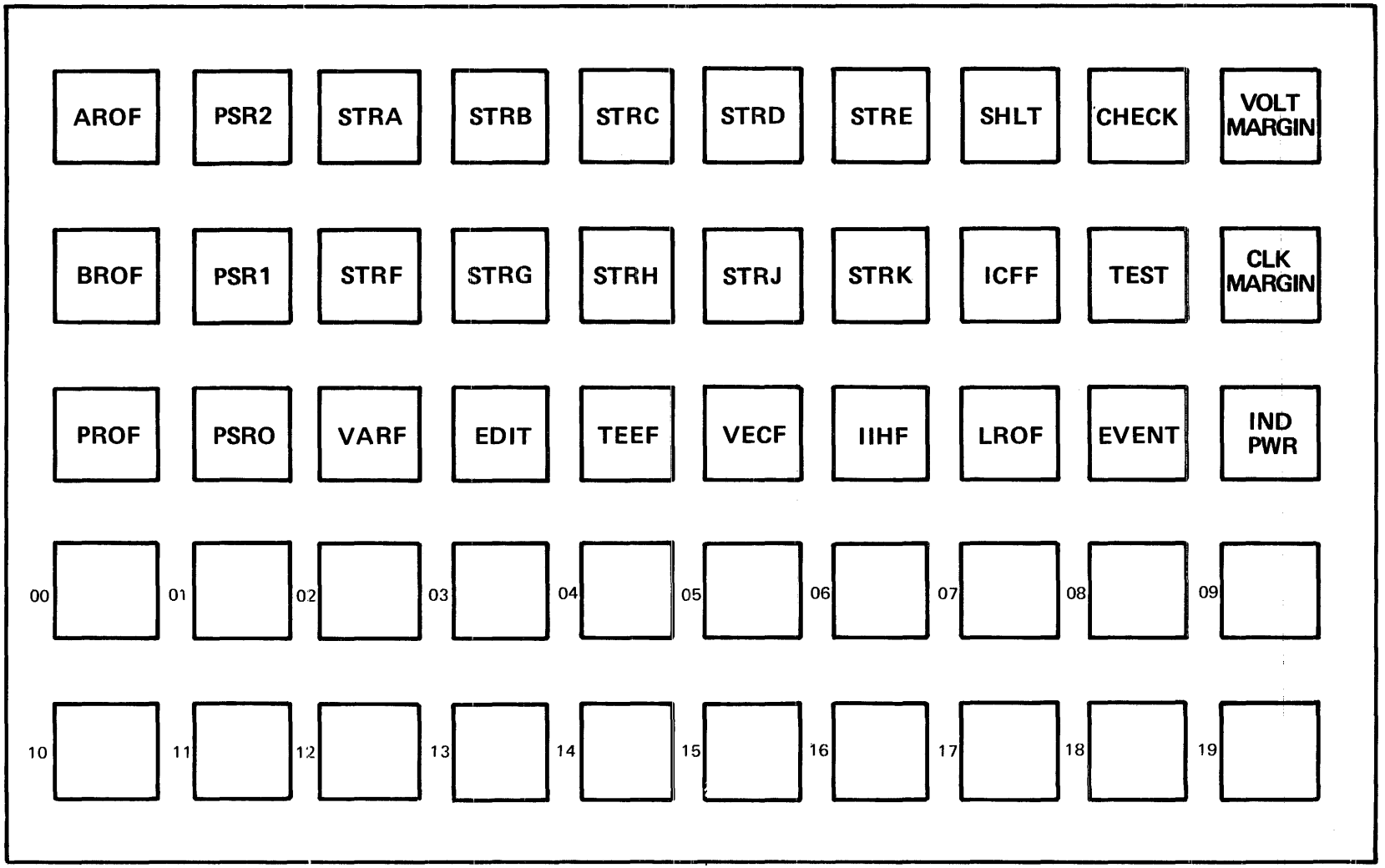
When depressed, initiates a power down sequence. This indicator is illuminated when power is off, the main system circuit-breaker is on, and +10 Vdc is available in the system.

Status Display Panel

The status display panel (see figure 4-3) consists of a series of 50 individual indicator lamp. These individual indicators are arranged in a matrix of five rows of ten indicators. Each indicator displays the state of a major control flip-flop within the B 6800 system. The logical state of the flip-flop is true when the lamp indicator is illuminated, and is false when the indicator is not illuminated.

The meaning of each indicator on the status display panel is as follows:

Mnemonic	Meaning
AROF	This indicator is illuminated when the A register contains a valid stack word.
BROF	This indicator is illuminated when the B register contains a valid stack word.
PROF	This indicator is illuminated when the P register contains a word of program code.
PSR 2, PSR 1, PSR 0	The three-bit program syllable register. This register is used as an index value into the contents of the P register. The P register contains machine language code operators. PSR points to the first syllable of the next machine language operator to be executed. When the next machine language operator is executed it will cause PSR to point at a new syllable in the P register. The initial value of the PSR register comes from a PCW for program entry, or from a RCW for program exit, or from the second syllable of a branching operator. The P register contains six syllables, numbered zero through five. The binary value of PSR selects a beginning syllable. The PSR = 5 condition, together with a SECL (Syllable Execute Complete Level), will cause a new word of program code to be placed in the P register, and PSR will be reset so that it points at the first syllable of the new program code in the P register.
STRA, STRB, STRC, STRD, STRE, STRF, STRG, STRH, STRJ, STRK	The ten data processor family strobe indicators. Every machine language operator that is executed by the data processor belongs to one of the ten DP operator families. When a machine language operator is executed by the data processor, the proper strobe lamp is illuminated to indicate which family of the data processor the current operator belongs to. The proper family (strobe) is determined by decoding the four most significant bits of the operator code (pointed at by PSR) in the P register.
VARF	The variant mode flip flop indicator lamp. Each family of primary operator codes in the DP can contain a maximum of sixteen specific operators. The VARF flip flop is used to extend the number of operator codes in a family. The use of variant mode causes an



MV 1605

Figure 4-3. System Status Indicator Panel

Mnemonic	Meaning
VARF (continued)	operator code to use a minimum of two syllables in the P register of the DP. The first syllable of a variant mode operator must be the VARI (95 hexidecimal) code. The execution of the VARI (95) primary mode operator causes the VARF flip flop to set, and the indicator lamp to illuminate. The state of the VARF flip flop identifies either a primary mode operator (VARF not), or a variant mode operator (VARF). A variant mode operator occupies two or more syllables. The first syllable of a variant mode operator code is the primary mode VARI operator. This operator causes the VARF flip flop to be set, and thereby specifies that the next syllable contains a variant mode operator code. The second syllable in a variant mode operator code specifies the family strobe for the operation to be performed and also specifies the specific variant mode operation that is to be performed. The VARF flip-flop is reset each time the data processor completes a variant mode operator. If the next operation to be performed is another variant mode operation, then another VARI operator must be performed to set the VARF flip-flop. The data processor detects two consecutive VARI (95) operator codes to be an invalid program code error condition. This error condition will cause the system to abort the program that contains the error condition.
EDIT	The EDIT indicator represents the state of the EDIT flip flop in the data processor. The EDIT flip flop is used to extend the number of operator codes that a family may contain, in a manner that is similar to the way that the VARF flip flop extends the number of operators in a family. The B 6800 system cannot be in edit mode, and in variant mode at the same time, except when a variant mode operation is performed by means of a pseudo-call from an edit mode operation. Edit mode is never pseudo-called by a variant mode operation. There are three data processor machine language operators that directly cause the EDIT flip flop to be set. These operator codes aer as follows: EXSD Execute Single Micro destructive (code D2). EXSU Execute Single Micro Update (code DA). ESPU Execute Single Micro, Single Pointer Update (code DD). Each of these three operator codes cause a single edit mode micro operator to be executed by the data processor, after which the data processor reverts to primary mode. The edit mode micro that will be executed is defined in the next syllable of program code following the Execute Single Micro operator code syllable. The EDIT flip flop is set at the beginning of the operator code, and is reset at the end of the operator code. Resetting the EDIT flip flop causes the data processor to revert to primary mode operations. The EDIT flip flop is also set during the execution of a table of edit mode operators. The use of a table of edit mode micro operators is indicated when the TEEF indicator is illuminated. When a table of edit mode micro operators are executed the last micro operator in the table must be the End Edit Micro Operator (EEND). This operator, when executed, will cause the EDIT flip flop to be reset, returning the data processor to primary mode operations.

Mnemonic	Meaning
TEEF	The TEEF indicator is illuminated when a table of edit mode micro operators is being executed. The table of micro operators is fetched from a location in memory in a manner that is similar to that used to fetch primary mode machine language code. The edit mode micro code is placed in the P register, and the syllables of micro code are executed the same as if they were primary mode machine language code. There are two primary mode machine language code operators that cause the B 6800 data processor to begin table edit mode operations, and they are: TEED Table Enter Edit Destructive operator (code D0) TEEU Table Enter Edit Update operator (code D8). Executing the TEED, or TEEU operator causes the TEEF, and the EDIT flip flops to be set, placing the data processor in the table edit mode of operations. The last micro operator in the table must be the ENDE micro operator. The execution of the ENDE micro code operator causes both the EDIT, and the TEEF flip flops to be reset, and these flip flops being reset returns the data processor to primary mode operations. The use of table edit mode operations causes a special set of base and index registers to be used to fetch the micro operators from memory. These IC memory registers are defined and discussed in section 5.
VECF	The VECF flip flop is set, and the indicator is lit when the DP is operating in vector mode. Two primary mode operators are used to place the B 6800 data processor in vector mode, and these operators are: VMES Vector Mode Enter Single (code EF). VMEM Vector Mode Enter Multiple (code E7). The type of vector mode entry operator used depends on the number of words of vector operators that are to be executed. If only one word of vector mode operators is to be executed the VMES operator is used to enter vector mode. If more than one word of vector mode code is to be executed then the VMEM operator is used to enter vector mode. The vector mode word may contain primary mode operators from families A, B, C, D, and E. If a primary operator is executed during vector mode operations, and the primary mode operator incurs an interrupt condition, the data processor will exit from vector mode, and resume processing in primary mode. The cause of the interrupt in the primary mode operator will not be remembered, and no indication that a failure occurred will be maintained. A word of vector mode machine language coded operators may be executed recursively, without regard to the type of vector mode entry that is used. The VECF indicator is illuminated when vector mode operations is entered, and stays illuminated during the time that the data processor is performing vector mode operations.

Mnemonic	Meaning
VECF (continued)	There are two vector mode operators that are used to exit vector mode and return to primary mode. These vector operators are: VEBR Vector Branch (code EE). VMEX Vector Exit (code E6). When either of these vector mode operators are executed the VECF flip flop is reset, the indicator is extinguished, and the data processor is returned to primary mode. The data processor will also exit from vector mode and return to primary mode if an internal interrupt occurs during vector mode operations. External interrupts are disabled during vector mode interrupts.
IIHF	The Inhibit External Interrupt flip flop (IIHF), when set, prevents an external interrupt from initiating the interrupt controller logic in the data processor. When reset, IIHF allows external interrupts to initiate the interrupt controller logic. The IIHF flip flop is set, and the indicator is illuminated when the Disable External Interrupt (DEXI, code 9547) variant mode operator is executed. The IIHF flip flop is reset, and the indicator is extinguished when the Enable External Interrupt (EEXI, code 9546) variant mode operator is executed. IIHF may also be set or reset upon entry into a new procedure. The state of bit number nineteen in a PCW that causes entry into a new procedure, conditions the state of IIHF. If bit nineteen (in the PCW) is true, IIHF will be set. IIHF is conditioned by the state of bit nineteen in an RCW (during the execution of an EXIT or RETURN operation) in a similar manner to that of the PCW (during an ENTER operation).
SHLT	The SHLT indicator shows the state of the Super Halt flip-flop. If the flip-flop is set, the indicator is illuminated. The super halt logic is used to prevent stack runaway due to repetitive errors in the interrupt handling procedure. Stack runaway causes the interrupt controller to destroy the contents of memory by inserting multiple interrupt handling stacks. The super halt flip flop is set if the interrupt controller is initiated four times without performing the normal EXIT operator at the end of the interrupt handling procedure. When the SHLT flip flop is set the system is halted such that stack runaway is stopped. The SHLT indicator is illuminated to indicate the reason for stopping the system.
ICFF	The ICFF indicator shows the state of the Interrupt Controller Run (ICFF) flip flop. The ICFF flip flop is used by the interrupt controller to enable the sequence counts of the controller flow. The ICFF flip flop is set while the interrupt controller is performing its normal functions, and is reset otherwise.
LROF	The LROF flip-flop (Look Ahead Register Occupied) indicates valid code is contained in the look ahead register. The LROF flip flop performs a function for the look ahead register that is analogous to the function the PROF flip flop performs for the P register. If the next word of program code

Mnemonic	Meaning
LROF (continued)	to be executed is present in the look ahead register then LROF is set. If LROF is reset then the next word of program code is present in memory at the address specified by the sum of the addresses in the PBR, and the PIR IC memory registers, or the LAR (look ahead memory address) register. If the P register becomes empty (PROF is reset) and LROF is set, then the contents of the look ahead register are transferred to the P register to be executed as program code. Transferring the contents of the look ahead register to the P register will cause the LROF flip flop to be reset. When LROF is reset the look ahead logic will attempt to perform a memory cycle and fill the look ahead register with the next word of program code in sequence.
CHECK	The CHECK indicator is used to determine when the Maintenance Processor (MP) module has detected a fault condition during the MP module internal testing.
TEST	The TEST indicator is used to indicate whether or not the MDP PROC ENABLE switch is in the ENABLED position. If the TEST indicator is illuminated the switch is in the ENABLED position. This indicator also illuminates if the SECL or CHLT pushbuttons on the keyboard are depressed, or if the CPU LOCAL/REMOTE switch on the maintenance control panel is in the LOCAL position.
EVENT	The EVENT indicator is used to indicate whether or not the MDP EVENT switch is in the ENABLED position. If the EVENT indicator is illuminated the switch is in the ENABLED position.
VOLT MARGIN	The VOLT MARGIN indicator circuit is used to indicate when a voltage margin has been selected in a power supply.
CLK MARGIN	The CLK MARGIN indicator circuit is used to indicate when a clock margin has been selected.
IND PWR	The IND PWR indicator circuit is used to indicate when a fault condition has occurred in the power supply of a planar memory cabinet. The IND PWR lamp is illuminated for either of the following conditions: a. Either the +5V or the ±15V regulator, or both regulators for one of the memory modules in a planar memory cabinet detects an UV condition. b. Either or both MODE switches on the auxiliary sequence control board in a planar memory cabinet is in the OFF position (that is, disabling the regulators for one or both of the memory modules).
00 through 19	The twenty indicators numbered 00 through 19 are used to indicate the busy status (or availability) of peripheral channels in the B 6800 system. Channel 00 through 09 are the ten channels located in peripheral control cabinet number zero. Channel 10 through 19 are located in PCC number one. A peripheral channel is available when its indicator is illuminated.

Register Display Panel

Figure 4-2 shows that there are four register displays on the register display panel. Each register can display up to eight different registers. There are eight pushbutton indicators that are used to select which register in the CPU is displayed in each of the four displays. When a pushbutton is pressed, the pushbutton illuminates, and remains illuminated, to indicate which display is selected.

The meaning of the eight register selection indicators for each of the four registers is as follows:

Register Number	Pushbutton	Data Displayed When Pushbutton Illuminated
1, 2, 3	A	The contents of the A register of the CPU
	B	The contents of the B register of the CPU
	X	The contents of the X register of the CPU
	Y	The contents of the Y register of the CPU
	C	The contents of the C register of the CPU
	Z	The contents of the Z register of the CPU
	P	The contents of the P register of the CPU
1, 2	L	The contents of the look ahead register of the CPU cabinet
3	E	The contents of the external input bus to channel B of the memory control
4	LAR/ADRA	The contents of the five hexadecimal digits (twenty bits) of the look ahead logic memory control address field on the left, and the contents of the five hexadecimal digits (twenty bits) of the Channel A save memory control address field on the right.
	MARB/ ADRB	The contents of the five hexadecimal digits (twenty bits) of the channel B memory control address field on the left, and the contents of the five hexadecimal digits (twenty bits) of the channel B save memory address on the right.
	MAR/MSMR	The contents of the five hexadecimal digits (twenty bits) of the memory control address register on the left, and the contents of the five hexadecimal digits (twenty bits) of the address adder sum register on the right.
	EXP FLD	See note one below.
	STVR	The contents of one of four vector status words in the CPU module

Note one: When a top-of-stack register is displayed in octal format, the register that is selected to display the register data contains the 13 octal digits of the mantissa field. The contents of the exponent field for the selected word are displayed in register number four if the EXP FLD position is selected.

MAINTENANCE DISPLAY PANEL

The maintenance display panel (refer to figure 4-1) is divided into two sections which are the Maintenance Processor (MP) control panel and display, and the maintenance display registers. The maintenance display panel is covered by a hinged panel, and during normal system operations is not visible. When the hinged panel is opened, two maintenance registers, and the MP control panel and display are visible on the facing panel (see figure 4-4). Two other maintenance display registers are mounted on the back of the hinged panel (see figure 4-5). The MP control panel and display are located at the bottom of the facing panel, and the two maintenance display panels are located above the MP control panel.

Maintenance Display Registers

A maintenance display register consists of 64 indicator lamps, 64 label windows, a thumbwheel selector switch, and a selector switch display window. The 64 indicator lamps are organized into four bars of 16 indicators. A label window is located immediately beneath each indicator lamp. A thumbwheel selector switch is located on the right hand side of the register, and the selector switch display window is located to the left of the thumbwheel selector switch, between the thumbwheel and the four bars of indicator lamps.

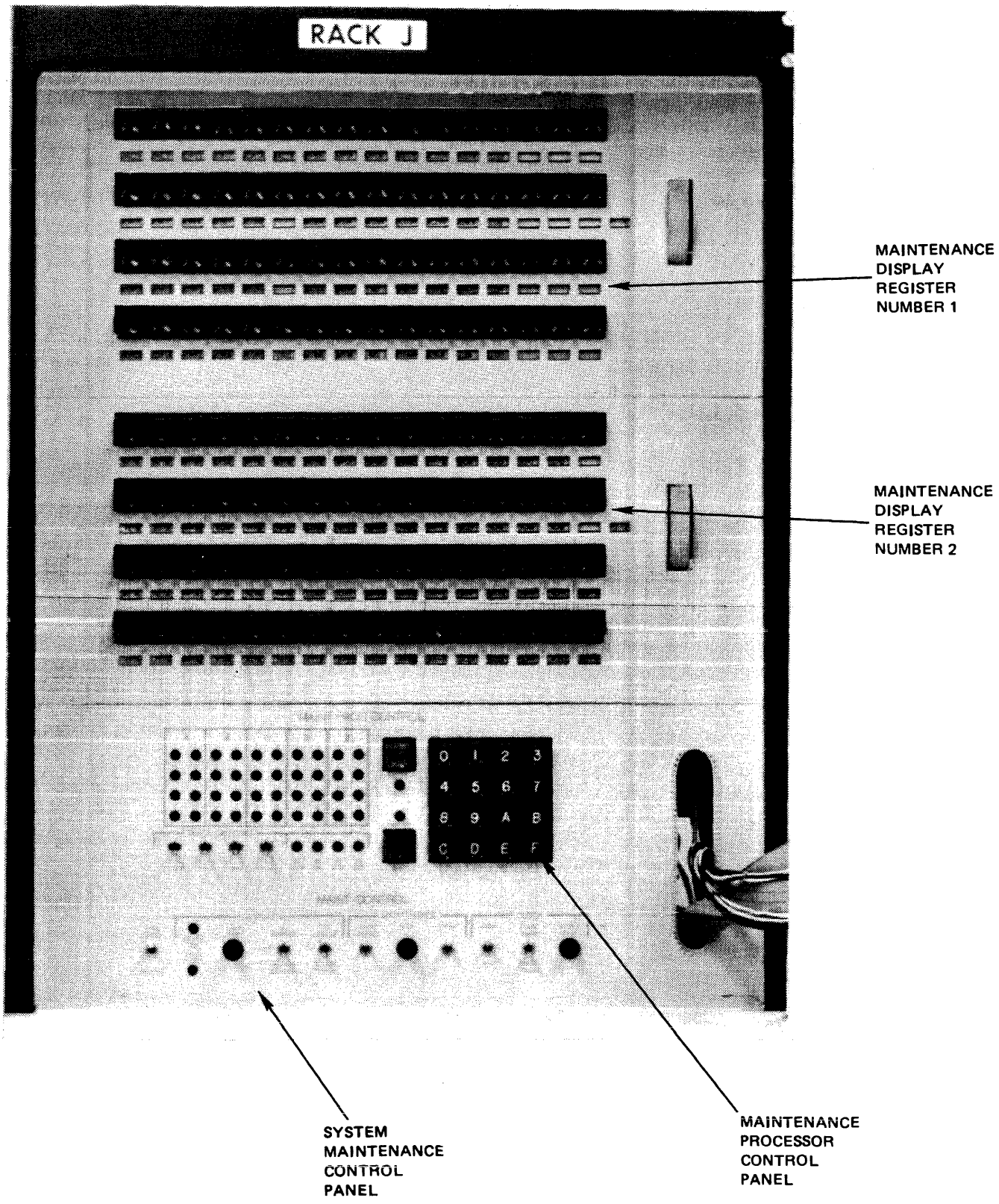


Figure 4-4. Maintenance Display (Facing) Panel

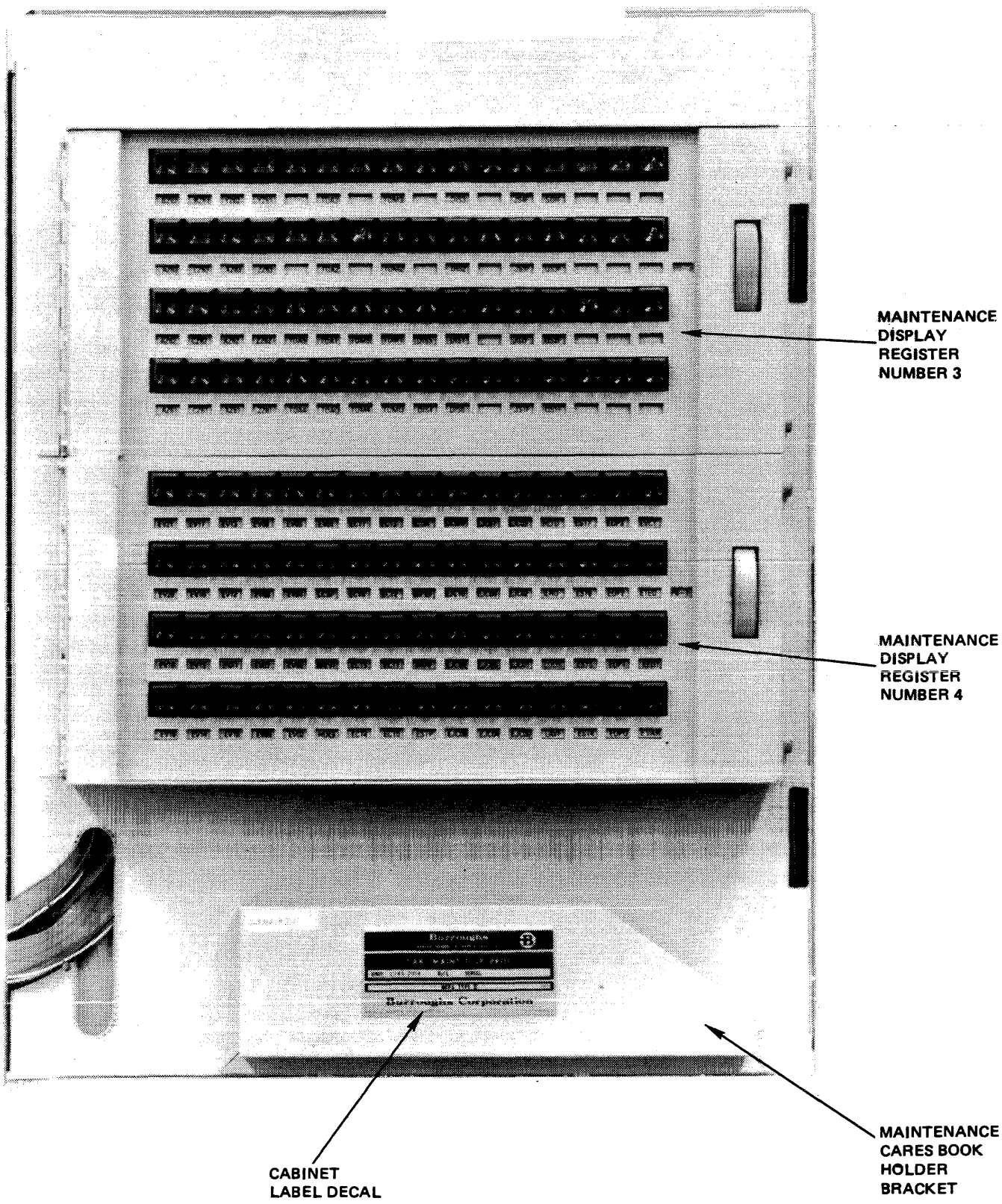


Figure 4-5. Maintenance Display Panel (Cover)

The thumbwheel selector switch selects one of six pages of 64 signals on a panel of the CPU cabinet to be displayed in a register. The selector switch display window indicates which page and panel are presently selected for display in the register. The page, panel, and display register correlation is shown in table 4-1. This table also indicates the major circuit or logical module that is displayed in each of the maintenance display registers, for all settings of the thumbwheel selector switches.

Table 4-1. B 6800 Maintenance Display Panel Register Selection Positions

Maintenance Display Register No.	Thumbwheel Selector Position	Panel Selected	Page Selected	Major Circuit or Module Displayed
1	P1-0	1	0	Multiplexer
	P1-1	1	1	Multiplexer
	P1-2	1	2	Data Processor
	P1-3	1	3	Data Processor
	P1-4	1	4	Data Processor
	P1-5	1	5	Data Processor
2	P2-0	2	0	Multiplexer
	P2-1	2	1	Multiplexer
	P2-2	2	2	Data Processor
	P2-3	2	3	Data Processor
	P2-4	2	4	Data Processor
	P2-5	2	5	Data Processor
3	P3-0	3	0	Multiplexer
	P3-1	3	1	Multiplexer
	P3-2	3	2	Data Processor
	P3-3	3	3	Data Processor
	P3-4	3	4	Data Processor
	P3-5	3	5	Data Processor
4	P4-0	4	0	Maintenance and Event logic
	P4-1	4	1	Maintenance and Event logic
	P4-2	4	2	Clock Control logic, and Time of Day logic
	P4-3	4	3	Memory Control
	P4-4	4	4	Micro Program Logic
	P4-5	4	5	Not used

Tables 4-2 through 4-13 show the flip flops or logic terms that are displayed in each bit position of the maintenance display registers. If the flip flop is set (the logic term is a true logic level) the indicator is illuminated.

Table 4-2. Maintenance Display Register Logic Signals For Register 1, Pages 0 (Top), and 1 (Bottom)

Maintenance Register Number 1 Switch Position (Page) Number P1-0

	Bits															
Bar	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	RDAF	DREN	RDA8	PDB8	PDC8	PDD8			10CF	SCH3	ITP3		IC3F		ICH3	
2	MINF	CDLF	PDA4	PDB4	PDC4	PDD4	SCF2	STCF	J512	SCH2	ITP2		IC2F		ICH2	
3	LSAF	DAGL	PDA2	RDB2	PDC2	PDD2	SCF1			SCH1	ITP1	IC5F	IC1F		ICH1	
4	ADPF	RDPR	PDA1	PDB1	RDC1	PDD1	SCF0		SCH4	SCH0	ITP0	IC4F	IC0F	ICH4	ICH0	

Maintenance Register Number 1 Switch Position (Page) Number P1-1

	Bits															
Bar	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1			BDA8	BDB8	BDC8	BDD8	OUTP	BCC3		BCF3		BCH3	LSTC		IOR3	
2			BDA4	BDB4	BDC4	BDD4	ZWPR	BCC2		BCF2		BCH2	MAHF		IOR2	
3			BDA2	BDB2	BDC2	BDD2	CRPR	BCC1	BCF5	BCF1		BCH1		PAVL	IOR1	
4		BDPR	BDA1	BDB1	BDC1	BDD1	BRQF	BCC0	BCF4	BCF0	BCH4	BCH0		IORG	IOR0	

Table 4-3. Maintenance Display Register Logic Signals For Register 1, Pages 2 (Top), and 3 (Bottom)

Maintenance Register Number 1 Switch Position (Page) Number P1-2

								Bits	Bits						
Bar	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	TA3F	SA3F	JA7F	JA3F	EXAI	QA7F	QA3F	SM03	SM04			NLZ3	HR15	HR11	HR07	HR03
2	TA2F	SA2F	JA6F	JA2F	KA2F	QA6F	QA2F	SM02				NLZ2	HR14	HR10	HR06	HR02
3	TA1F	SA1F	JA5F	JA1F	KA1F	QA5F	QA1F	SM01	PSCF			NLZ1	HR13	HR08	GR05	HR01
4	TA0F	SA0F	JA4F	JA0F	KA0F	QA4F	QA0F	SM00	CMPF		NLZF	NLZ0	HR12	HR08	HR04	HR00

Maintenance Register Number 1 Switch Position (Page) Number P1-3

								Bits							
Bar	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	TC3F	JC7F	JC3F	QC8F	QC4F	CRNCF		ACL7	ACL3		TD3F	JD7F	JD3F		QD8F	QD4F
2	TC2F	JC6F	JC2F	QC7F	QC3F	SASG		ACL6	ACL2		TD2F	JD6F	JD2F	QD6F	QD7F	QD3F
3	TC1F	JC5F	JC1F	QC6F	QC2F	QCZ2	ACM5	ACL5	ACL1		TD1F	JD5F	JD1F	QDAF	QD6F	QD2F
4	TC0F	JC4F	JC0F	QC5F	QC1F	QCZ1	ACM4	ACL4	ACL0		TD0F	JD4F	JD0F	QD9F	QD5F	QD1F

Table 4-4. Maintenance Display Register Logic Signals For Register 1, Pages 4 (Top), and 5 (Bottom)

Maintenance Register Number 1 Switch Position (Page) Number P1-4

	Bits															
Bar	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	MREZ9	MREZ8	MDRSS	MZ812	MDRSX	MACT8	MPBR	MPIR	MPDR	MB0S	MCOUT	MZ6T8	LL03			
2	MDR0C	MDR1C	MDRF4	MZ812	MSOR2		MSBR	MSIR	MSNR	MLOSR	MCZIN	MZ6L8	LL02			
3	MDR0B	MDR1B	MDRF3	MZ810	MSOR1		MDBR	MDIR	MF	MBUF	MSUBT	MZ6T9	LL01			
4	MDR0A	MDR1A	MDRF2	MZ809	MSOR0	CR1C	MTBR	MTIR	MS	MTEMP	MZ6L9	LL04	LL00			

Maintenance Register Number 1 Switch Position (Page) Number P1-5

	Bits															
Bar	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	RDCBA	RDREA	CKB4A	CBIN	CKB4B	ADREB	INAGB	CBM3	SC1D	STAR	SCNR	SCAN	STB2	ADDR	EREN7	EREN3
2	ADREA	RDSEA	CKB3A	CBAI	CKB3B	RDCBB	RDMEB	GBNTB	GNTR	INPW	STAP	RUN1	STB1	BURE	EREN6	EREN2
3	STOF	CKB6A	CKB2A	CKB6B	CKB2B	RDREB	INALB	CAM3	INAGA	STUF	LOPE	LOPT	STB0	RCPE	EREN5	EREN1
4	SDIS	CKB5A	CKB1A	CKB5B	CKB1B	RDSEB	MPARB	SCOR	RDMEA	INALA	LPEN	LOOP	CMPR	PCPE	EREN4	EREN0

Table 4-5. Maintenance Display Register Logic Signals For Register 2, Pages 0 (Top), and 1 (Bottom)

Maintenance Register Number 2 Switch Position (Page) Number P2-0

	Bits															
Bar	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	CC19	CC15	CC11	CC07	CC03		PF19	PF15	PF11	PBA7	PBA3	PC19	PC15	PC11	PTA7	PTA3
2	CC18	CC14	CC10	CC06	CC02		PF18	PF14	PF10	PBA6	PBA2	PC18	PC14	PC10	PTA6	PTA2
3	CC17	CC13	CC09	CC05	CC01		PF17	PF13	PIEN	PBA5	PBA1	PC17	PC13	PC09	PTA5	PTA1
4	CC16	CC12	CC08	CC04	CC00		PF16	PF12	PBA8	PBA4	PBA0	PC16	PC12	PTA8	PTA4	PTA0

Maintenance Register Number 3 Switch Position (Page) Number P2-1

	Bits															
Bar	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	AC19	AC15	AC11	AC07	AC03	PINH	MINH	BKWD	NEAR	BTA7	BTA3	BS19	BS15	BS11	BBA7	BBA3
2	AC18	AC14	AC10	AC06	AC02	ASCI	XLAT	TEST	BB=T	BTA6	BTA2	BS18	BS14	BS10	BBA6	BBA2
3	AC17	AC13	AC09	AC05	AC01	ATTN	FRAM	TGC1	BIEN	BTA5	BTA1	BS17	BS13	BOEN	BBA5	BBA1
4	AC16	AC12	AC08	AC04	AC00	BINP	MPRT	TGCO	BTA8	BTA4	BTA0	BS16	BS12	BBA8	BBA4	BBA0

Table 4-6. Maintenance Display Register Logic Signals For Register 2, Pages 2 (Top), and 3 (Bottom)

Maintenance Register Number 2 Switch Position (Page) Number P2-2

	Bits															
Bar	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	BETB	NZTB	HRTB1	EXSB	B1TB	B8TB	ADSB	SPCI	BX02	AX02	YR-3	SC3F	SCEF	ICR7	ICR3	
2	YETB	ZDTB	HRTB2	ECRI	Y1TB	Y8TB	CCNS	DPCI	BX01	AX01	YR-2	SC2F	ICRE	ICR6	ICR2	
3	AETA	NZTA	HRTA1	A1TA	A2TA	A4TA	CCR3	CI75	BX00	AX00	YR-1	SC1F	BXSE	ICR5	ICR1	BDPD
4	XETA	ZDTA	HRTA2	X1TA	X2TA	X4TA	CCL3	DP0V	YX00	XX00	XR-1	SC0F	DISX	ICR4	ICR0	ADPD

Maintenance Register Number 2 Switch Position (Page) Number P2-3

	Bits															
Bar	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	RQTB	RQT7	RQT3	RQRB	RQR7	RQR3	CSC4	CAPFE	CAPFD	CAPFC	CAPFB	CAPFA	ABRF		HAR3	
2	RQTA	RQT6	RQT2	RQRA	RQR6	RQR2	CSC3	SRM2	SRL2	ATEF	MAOF	WAIT	CARQ		HAR2	
3	RQT9	RQT5	RQT1	RQR9	RQR5	RQR1	CSC2	SRM1	SRL1	TRYF	RDEF	LOG2	MI51R		HAR1	
4	RQT8	RQT4	RQT0	RQR8	RQR4	RQR0	CSC1	SRM0	SRL0	CNGO	CINF	LOG1	MI48	PTGO	HAR0	

Table 4-7. Maintenance Display Register Logic Signals For Register 2, Pages 4 (Top), and 5 (Bottom)

Maintenance Register Number 2 Switch Position (Page) Number P2-4

	Bits															
Bar	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	TU8F	EEND	RETF		JU3F	SSZ2	SI08	DI08	QU4F	DGSF	SOPF			TFFF	EQVF	QUDF
2	TU4F	FINI	RNTF	JU6F	JU2F	SSZ1	SI04	DI04	QU3F	LHFF	UPDF			TF0F		QUCF
3	TU2F	EXSF	NVLF	JU5F	JU1F	DSZ2	SI02	DI02	QU2F	RPZF	SPRF			OFFF		QUBF
4	TU1F	EXPF	MPOP	JU4F	JU0F	DSZ1	SI01	DI01	QU1F	XROF	DPRF			FLTF	EXTF	QUAF

Maintenance Register Number 2 Switch Position (Page) Number P2-5

	Bits															
Bar	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	BYR19	BYR15	BYR11	BYR07	BYR03	MTST	JV1	CMPE	CBPW	CB4W	CBPR	CB4R	IT10	IT06	IT02	ECSF
2	BYR18	BYR14	BYR10	BYR06	BYR02	TV2	JV0	TABT	WEFW	CB3W	WEFR	CB3R	IT09	IT05	IT01	EXTI
3	BYR17	BYR13	BYR09	BYR05	BYR01	TV1	ONCK	ALTWC	CB6W	CB2W	CB6R	CB2R	IT08	IT04	IT00	INTV
4	BYR16	BYR12	BYR08	BYR04	BYR00	TV0		MI51W	CB5W	CB1W	CB5R	CB1R	IT07	IT03	IMTV	INTE

Table 4-8. Maintenance Display Register Logic Signals For Register 3, Pages 0 (Top), and 1 (Bottom)

Maintenance Register Number 2 Switch Position (Page) Number P3-0

	Bits															
Bar	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	SL23	SL21	SL19	SL17	SL15	SL13	SL11	SCL9	SCL7	SCL5	SCL3	SCL1		TOD3		
2	SE23	SE21	SE19	SE17	SE15	SE13	SE11	SCE9	SCE7	SCE5	SCE3	SCE1		TOD2		
3	SL22	SL20	SL18	SL16	SL14	SL12	SL10	SCL8	SCL6	SCL4	SCL2	SCL0		TOD1		
4	SE22	SE20	SE18	SE16	SE14	SE12	SE10	SCE8	SCE6	SCE4	SCE2	SCE0		TOD0		

Maintenance Register Number 3 Switch Position (Page) Number P3-1

	Bits															
Bar	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	BAFP4	VALV	PC3F	SPWF	XPCC	INM9	INM7	INM5	INM3	INM1			CSV1	SPW3	SPPE2	
2	BAFP3	BURV	PC2F	BFWF	XACC	INR9	INR7	INR5	INR3	INR1	MER2	CSV4	CSV0	SPW2	SPPE1	
3	BAFP2	RDEV	PC1F	XPBA	XBBA	INM8	INM6	INM4	INM2	INM0	MER1	CSV3	MREC	SPW1	SPRF2	
4	BAFP1	PBZV	PC0F	XRDR	MPXT	INR8	INR6	INR4	INR2	INR0	MER0	CSV2	SPW4	SPW0	SPRF1	

Table 4-9. Maintenance Display Register Logic Signals For Register 3, Pages 2 (Top), and 3 (Bottom)

Maintenance Register Number 3 Switch Position (Page) Number P3-2

	Bits															
Bar	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	TE3F	JBCF	JE3F	QE3F	SMVF	LC2F	SF3F	MP35	QE4F	TB3F	JB3F	QB4F				
2	TE2F	JE6F	JE2F	QE2F	MPYF	LC1F	SF2F	DBZF		TB2F	JB2F	QB3F				
3	TE1F	JE5F	JE1F	QE1F	SUBF	LC0F	SF1F	FNWF		TB1F	JB1F	QB2F				
4	TE0F	JE4F	JE0F	QE0F	LC3F	DPFF	SF0F	ZR0F		TB0F	JB0F	QB1F				

Maintenance Register Number 3 Switch Position (Page) Number P3-3

	Bits															
Bar	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	BZ62	BZ61	YZ62	YZ61		TCA3		TOM3		DIS3		JS4F	SOIF			
2	AZ63	CZ63	XZ63	ZZ63		TOA2		TOM2		DIS2		JS3F	QS3F			
3	AZ62	CZ62	XZ62	ZZ62	TOA5	TOA1	TOM5	TOM1	DIS5	DIS1		JS2F	QS2F			
4	AZ61	CZ61	XZ61	ZZ61	TOA4	TOA0	TOM4	TOM0	DIS4	DIS0		JS1F	QS1F			

Table 4-10. Maintenance Display Register Logic Signals For Register 3, Pages 4 (Top), and 5 (Bottom)

Maintenance Register Number 3 Switch Position (Page) Number P3-4

	Bits															
Bar	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	CPA8	ICRF	CPIR1	CTIR	SSR2	WPIR	QP8F	QP4F	STMC		MPRCB	MTEXB	MDY4F	MATEB	MIF51	MBSC3
2	CPA4	ICCF	CPIR0	CSR2	SSR1	SECF	QP7F	QP3F	JP02		MWRCB	MAPLB	MDY3F	MTRYB	MHARB	MBSC2
3	CPA2	FWFF	WPTF	CSR1	SSR0		QP6F	QP2F	JP01		MNRFB	MAOFB	MDY2F	MHOLD	MCBQF	MBSC1
4	CPA1	PRVA	WBCF	CSR0		VSJK	QP5F	QP1F	JP00		MREQB	MABXB	MDY1F	MDRYB	MRDBF	MBSC0

Maintenance Register Number 3 Switch Position (Page) Number P3-5

	Bits															
Bar	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	LRAP	LRIG	IML2	ABR1	INCT	INF+1	INFF	MPXI	DR31	DR27	DR23	DR19	DR15	DR11	DR07	DR03
2	LRIL	LRGN	IML1	ABEI	MEWT	SEC+2	ALSB	MPXB	DR30	DR26	DR22	DR18	DR14	DR10	DR06	DR02
3	LRAR	LAER	IML0	ILDM	BDST	SEC+1		MPXG	DR29	DR25	DR21	DR17	DR13	DR09	DR05	DR01
4	LRDM	OPTF	GCDS	SEIN	ABIT	AYER	RTRY		DR28	DR24	DR20	DR16	DR12	DR08	DR04	DR00

Table 4-11. Maintenance Display Register Logic Signals For Register 4, Pages 0 (Top), and 1 (Bottom)

Maintenance Register Number 4 Switch Position (Page) Number P4-0

	Bits															
Bar	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1		WMMF	AMMF						JCMP	JCS07	JCS03			SRS3	OSR3	VARS
2	PLK1	RMMF	AIMF						JCS10	JCS06	JCS02			SRS2	OSR2	VCTS
3	PLK0	WIMF	MEXI		HALT				JSC09	JCS05	JCS01			SRS1	OSR1	EDTS
4	PSOR	RIMF			ARPT				JCS08	JCS04	JCS00			SRS0	OSR0	TEDS

Maintenance Register Number 4 Switch Position (Page) Number P4-1

	Bits															
Bar	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	EV21	EV17	EV13	EV09	EV05	EV01	ECT7	ECT3	ICOR	EJCMP	EJC07	EJC03	HLTD	EST7	EOP3	EVCT
2	EV20	EV16	EV12	EV08	EV04	CCSF	ECT6	ECT2	MBPI	EJC10	EJC06	EJC02	ILHD	EST6	EOP2	ETED
3	EV19	EV15	EV11	EV07	EV03	MEVF	ECT5	ECT1	MIAI	EJC09	EJC05	EJC01	LODS	EST5	EOP1	EEDT
4	EV18	EV14	EV10	EV06	EV02	HOEF	ECT4	ECT0	ESTP	EJC08	EJC04	EJC00	LAVF	EST4	EOP0	EVAR

Table 4-12. Maintenance Display Register Logic Signals For Register 4, Pages 2 (Top), and 3 (Bottom)

Maintenance Register Number 4 Switch Position (Page) Number P4-2

	Bits															
Bar	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	CPC28	CPC24	CPC20	CPC16	CPC12	CPC08		TD35	TD31	TD27	TD23	TD19	TD15	TD11	TD07	TD03
2	CPC27	CPC23	CPC19	CPC15	CPC11	CPC07		TD34	TD30	TD26	TD22	TD18	TD14	TD10	TD06	TD02
3	CPC26	CPC22	CPC18	CPC14	CPC10	CPC06		TD33	TD29	TD25	TD21	TD17	TD13	TD09	TD05	TD01
4	CPC25	CPC21	CPC17	CPC13	CPC09	CPC05		TD32	TD28	TD24	TD20	TD16	TD12	TD08	TD04	TD00

Maintenance Register Number 4 Switch Position (Page) Number P4-3

	Bits															
Bar	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	IMCF3	CBOF3	IMCF2	CBOF2	IMCF1	CBOF1	IMCF0	CBOF0	MTRIP	WSTF0	WSTF1		GS0F	ICNF	GT2F	GOAF
2	PS2F3	CAOF3	PS2F2	CAOF2	PS2F1	CAOF1	PS2F0	CAOF0	MTMR2	WSTF2	WSTF3		GS1F	GRDF	GT1F	GOBF
3	PS1F3	WCCF3	PS1F2	WCCF2	PS1F1	WCCF1	PS1F0	WCCF0	MTMR1	MSPY			GS2F	GABF	GTOF	GAOF
4	PS0F3	PEDF3	PS0F2	PEDF2	PS0F1	PEDF1	PS0F0	PEDF0	MTMR0	MSPX				CRFF		EGTM

Table 4-13. Maintenance Display Register Logic Signals For Register 4, Pages 4 (Top), and 5 (Bottom)

Maintenance Register Number 4 Switch Position (Page) Number P4-4

	Bits															
Bar	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	MM1E	M17F	M13F		MM2E	M27F	M23F		MM3E	M37F	M33F		MMPD			
2		M16F	M12F			M26F	M22F			M36F	M32F					
3	M19F	M15F	M11F		M29F	M25F	M21F		M39F	M35F	M31F					
4	M18F	M14F	M10F		M28F	M24F	M20F		M38F	M34F	M30F					

Maintenance Register Number 4 Switch Position (Page) Number P4-5

	Bits															
Bar	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1																
2																
3																
4																

DISPLAY PANEL ONE, PAGE ZERO LOGIC SIGNALS

Page zero of register number one is used to display the multiplexor functions of the CPU cabinet. The logic signals and flip-flops that are displayed (refer to table 4-2) are as follows:

RDAF	The result descriptor available flip-flop.
MINF	The minus flip-flop.
LSAF	The least significant address flip-flop.
ADPF	The even character required flip-flop (used for card punch IO devices).
DREN	The driver enable flip-flop.
CDLF	The channel designate level flip-flop.
DAGL	The access granted level, delayed signal.
PDPR	The peripheral data register parity bit.
PDA1, PDA2, PDA4, PDA8 PDB1, PDB2, PDB4, PDB8 PDC1, PDC2, PDC4, PDC8 PDD1, PDD2, PDD4, PDD8	The peripheral data register bits.
SCF0, SCF1, SCF2	The service cycle sequence count flip-flops.
STCF	The start channel flip-flop.
IOCF	The IO complete flip-flop.
J512	The large buffer (512 bytes) indicator.
SCH0, SCH1, SCH2, SCH3 SCH4	The scratchpad channel address flip-flops.
ITP0, ITP1, ITP2, ITP3	The internal unit type flip-flops.
ICH0, ICH1, ICH2, ICH3 ICH4	The initiate channel number register.

DISPLAY PANEL ONE, PAGE ONE LOGIC SIGNALS

Page one of register number one is used to display the multiplexor functions of the CPU cabinet. The logic signals and flip-flops that are displayed (refer to table 4-2) are as follows:

BDPR	The burst data register parity bit.
BDA1, BDA2, BDA4, BDA8 BDB1, BDB2, BDB4, BDB8 BDC1, BDC2, BDC4, BDC8 BDD1, BDD2, BDD4, BDD8	The burst data register bits.

OUTP	The output parity bit.
ZWPR	The Z register word parity bit.
CRPR	The C register parity bit (even parity).
BRQF	The burst request flip-flop.
BCC0, BCC1, BCC2, BCC3 BCF0, BCF1, BCR2, BCF3	The burst cycle character count flip-flops.
BCF4, BCF5	The burst cycle sequence count flip-flops.
BCH0, BCH1, BCH2, BCH3 BCH4	The burst channel number register.
LSTC	The last character logic signal.
MAHF	The save memory access obtained logic signal.
PAVL	The path available logic signal.
IORG	The IO regulator logic signal.
IOR0, IOR1, IOR2, IOR3	The IO regulator count register.

DISPLAY PANEL ONE, PAGE TWO LOGIC SIGNALS

Page two of register number one is used to display the status of family A and the arithmetic controller functions of the data processor, in the CPU cabinet. The logic signals and flip-flops that are displayed (refer to table 4-3) are as follows:

TA0F, TA1F, TA2F, TA3F	The T register flip-flops for family A.
SA0F, SA1F, SA2F, SA3F	The T value save register for family A.
JA0F, JA1F, JA2F, JA3F JA4F, JA5F, JA6F, JA7F	The family A sequence count register.
EXAI	The exponent add initiate flip-flop.
KA0F, KA1F, KA2F	The K counter for family A.
QA0F, QA1F, QA2F, QA3F QA4F, QA5F, QA6F, QA7F	Logical flip-flops used within family A.
SM00, 01, 02, 03, 04	Steering and mask value register used for generating TOA, TOM, and DIS values from family A operations.
PSCF	Psuedo call on family A flip-flop.
CMPF	Compare flip-flop used for relational operators.

NLZF, NLZ0, 1, 2, 3	Number of leading zeros register for arithmetic controller.
HR00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15	The arithmetic controller holding register.

DISPLAY PANEL ONE, PAGE THREE LOGIC SIGNALS

Page three of register number one is used to display the status of families C and D of the data processor, in the CPU cabinet. The logic signals and flip-flops that are displayed (refer to table 4-3) are as follows:

TC0F, TC1F, TC2F, TC3F	The T register flip-flops for family C.
JC0F, JC1F, JC2F, JC3F, JC4F, JC5F, JC6F, JC7F	The family C sequence count register.
QC1F, QC2F, QC3F, QC4F, QC5F, QC6F, QC7F, QC8F	Logical flip-flops used within family C.
CRNCF	The PIR and PBR register values are not consistant flip-flop signal (used by the interrupt controller).
SASG	Save segmented bit flip-flop.
QCZ2	Save size two flip-flop.
QCZ1	Save size one flip-flop.
ACM4, ACM5	Most significant two bits of address couple for NAMC operators.
ACL0, 1, 2, 3, 4, 5, 6, 7	Least significant eight bits of address couple for NAMC operators.
TD0F, TD1F, TD2F, TD3F	The T register for family D.
JD0F, JD1F, JD2F, JD3F, JD4F, JD5F, JD6F, JD7F	The family D sequence count register.
QD1F, QD2F, QD3F, QD4F, QD5F, QD6F, QD7F, QD8F	Logical flip-flops used within family D.

PANEL 1, PAGE 4 LOGIC SIGNALS

Page four in maintenance display register one is used to show the status of the IC memory control, address adder, and sum register functions of the data processor in the CPU cabinet. The logic signals and flip-flops (refer to table 4-4) that are displayed are as follows:

MREZ9	Residue error on Z9 bus logic signal.
MDROC	Display address bit 0 (group C cards).
MDROB	Display address bit 0 (group B cards).
MDROA	Display address bit 0 (group A cards).

MREZ8	Residue error on Z8 bus logic signal.
MDR1C	Display address bit 1 (group C cards).
MDR1B	Display address bit 1 (group B cards).
MDR1A	Display address bit 1 (group A cards).
MDRSS	Display memory read enable logic signal.
MDRF2, 3, 4	Display address bits 2, 3, and 4.
MZ812	Bit 12 index portion of address couple value.
MZ811	Bit 11 index portion of address couple value.
MZ810	Bit 10 index portion of address couple value.
MZ809	Bit 09 index portion of address couple value.
MDRSX	Bit 08 index portion of address couple value.
MSOR2	The sum of residue bit 2 (address adder).
MSOR1	The sum of residue bit 1 (address adder).
MSOR0	The sum of residue bit 0 (address adder).
MACT8	Address couple to Z8 register.
CRIC	Clear IC memory flip-flop.
MPBR	Program base register read select flip-flop.
MSBR	Source base register read select flip-flop.
MDBR	Destination base register read select flip-flop.
MTBR	Table base register read select flip-flop.
MPIR	Program index register read select flip-flop.
MSIR	Source index register read select flip-flop.
MDIR	Destination index register read select flip-flop.
MTIR	Table index register read select flip-flop.
MPDR	Program dictionary register read select flip-flop.
MSNR	Stack number register read select flip-flop.
MF	F register read select flip-flop.

MS	S register read select flip-flop.
MBOS	Bottom of stack register read select flip-flop.
MLOSR	Limit of stack register read select flip-flop.
MBUF	Buf (temp) register read select flip-flop.
MTEMP	Temporary register read select flip-flop.
MCOUT	Address adder carry out flip-flop.
MCZIN	Address adder carry in flip-flop.
MSUBT	Address adder subtract function flip-flop.
MZ6L9	Z6 bus (35:16) to Z9 bus (15:16) enable flip-flop.
MZ6T8	Z6 bus (19:20) to Z8 bus (19:20) enable flip-flop.
MZ6L8	Z6 bus (13:14) to Z8 bus (13:14) enable flip-flop.
MZ6T9	Z6 bus (39:20) to Z9 bus (19:20) enable flip-flop.
LL00, 01, 02, 03, 04	Lexagographical level register.

PANEL 1, PAGE 5 LOGIC SIGNALS

Page five display in maintenance display register one is used to show the status of the interrupt controller, memory controller, stack controller, and scan bus control functions of the data processor in the CPU cabinet. The logic signals and flip-flops (refer to table 4-4) that are displayed are as follows:

RDCBA	The read data check bit for channel A.
ADREA	The address retry bit for channel A.
STOF	The stack overflow flip-flop.
SDIS	The syllable dependent interrupt flip-flop.
RDREA	The read data retry bit for channel A.
RDSEA	The read data single bit for channel A.
CKB1A, CKB2A, CKB3A, CKB4A, CKB5A, CKB6A	The check bits for channel A.
CBIN	Channel B interrupt being reported logic signal.
CBAI	Channel B alarm interrupt flip-flop.
CKB1B, CKB2B, CKB3B, CKB4B, CKB5B, CKB6B	The check bits for channel B.

ADREB	The address retry bit for channel B.
RDCBB	The read data check bit for channel B.
RDREB	The read data retry bit for channel B.
RDSEB	The read data single bit for channel B.
INAGB	Invalid address bit-global for channel B.
RDMEB	The read data multiple error bit for channel B.
INALB	Invalid address bit-local for channel B.
MPARB	Memory parity error bit for channel B.
CBM3	Memory Control error bit for channel B.
GBNTB	Global memory not ready to channel B logic signal.
CAM3	Memory control error bit for channel A.
SCOR	Scan out error flip-flop.
SC1D	Scan in data error flip-flop.
GNTR	Global memory not ready flip-flop.
INAGA	Invalid address bit-global for channel A.
RDMEA	Read data multiple error bit for channel A.
STAR	Store address residue for channel A.
INPW	Invalid program word flip-flop.
STUF	Stack underflow flip-flop.
INALA	Invalid address-local for channel A.
SCNR	Scan bus not ready.
STAP	Memory address parity for channel A.
LOPE	Loop timer error flip-flop.
LPEN	Loop timer enable signal.
SCAN	Scan command active flip-flop.
RUNI	Running timer signal.
LOPT	Loop timer trigger signal.

LOOP	Loop timer multi-timer signal.
STB0, 1, 2	Stack register (tells where a read data word was put in the stack).
CMPR	Compare residue flip-flop.
ADDR	Address adder residue error flip-flop.
BURE	Bus residue error flip-flop.
RCPE	RAM card parity error flip-flop.
PCPE	PROM card parity error flip-flop.
ERENO, 1, 2, 3, 4, 5, 6, 7	Parity error PROM card number register.

PANEL 2, PAGE 0 LOGIC SIGNALS

Page zero display in maintenance register two is used to show the status of the multiplexor function in the CPU cabinet. The logic signals and flip-flops (refer to table 4-5) that are displayed are as follows:

CC00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19	The peripheral character counter.
STVC	The status vector logic signal.
PF10, 11, 12, 13, 14, 15, 16, 17, 18, 19	The peripheral flag register.
PIEN	Peripheral input end flip-flop.
PBA0, 1, 2, 3, 4, 5, 6, 7, 8	Peripheral buffer address register.
PC09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19	Peripheral control register.
PTA0, 1, 2, 3, 4, 5, 6, 7, 8	Peripheral target address register.

DISPLAY PANEL TWO, PAGE ONE LOGIC SIGNALS

Page one of register number two is used to display the status of the multiplexor burst control function, in the CPU cabinet. The logic signals and flip-flops (refer to table 4-5) displayed are as follows:

AC00, AC01, AC02, AC03, AC04, AC05, AC06, AC07, AC08, AC09, AC10, AC11, AC12, AC13, AC14, AC15, AC16, AC17, AC18, AC19	The multiplexor accumulator register.

PINH	The peripheral inhibit logic signal.
ASCI	The ASCII translate logic signal.
ATTN	The software attention logic signal.
BINP	The burst input logic signal.
MINH	The memory inhibit logic signal.
XLAT	The translate logic signal.
FRAM	The 8-bit character (frame) size logic signal.
MPRT	The memory protect logic signal.
BKWD	The backward operation logic signal.
TEST	The test operation logic signal.
TGC0, TGC1	The tag field control logic signals.
NEAR	The buffer address near to burst target address logic signal.
BB=T	The burst buffer address equal to burst target address logic signal.
BIEN	The burst input end signal.
BTA0, BTA1, BTA2, BTA3, BTA4, BTA5, BTA6, BTA7 BTA8	The burst target address register.
BS10, BS11, BS12, BS13, BS14, BS15, BS16, BS17, BS18, BS19	The burst save register.
BOEN	The burst output end logic signal.
BBA0, BBA1, BBA2, BBA3, BBA4, BBA5, BBA6, BBA7, BBA8	The burst buffer address register.

DISPLAY PANEL TWO, PAGE TWO LOGIC SIGNALS

Page two of display register number two is used to display the status of the arithmetic controller and family A of the data processor, in the CPU cabinet. The logic signals and flip-flops that are displayed (refer to table 4-6) are as follows:

BETB	The add B exponent to B input of exponent adder signal.
YETB	The add Y exponent to B input of exponent adder signal.
AETA	The add A exponent to A input of exponent adder signal.

XETA	The add X exponent to A input of exponent adder signal.
NZTB	The add NLZ (number of leading zeros) register output to the B input of the exponent adder signal.
NDTB	The add 13 (decimal) to B input of exponent adder signal.
NZTA	The add NLZ (number of leading zeros) register output to the A input of the exponent adder signal.
NDTA	The add 13 (decimal) to A input of exponent adder signal.
MRTB1, MRTB2	The holding register for the B input to the exponent adder.
MRTA1, MRTA2	The holding register for the A input to the exponent adder.
EXSB	The exponent adder add/subtract control logic signal.
ECRI	The exponent adder-carry-in/borrow signal.
A1TA	The A mantissa to A input of mantissa adder signal.
X1TA	The X mantissa to A input of mantissa adder signal.
B1TB	The B mantissa to B input of mantissa adder signal.
Y1TB	The Y mantissa to B input of mantissa adder signal.
A2TA	The 2 times the value of the A mantissa to A input of mantissa adder signal.
X2TA	The 2 times the value of the X mantissa to A input of mantissa adder signal.
B8TB	The 8 times the value of the B mantissa to B input of mantissa adder signal.
Y8TB	The 8 times the value of the Y mantissa to B input of mantissa adder signal.
A4TA	The 4 times the value of the A mantissa to A input of mantissa adder signal.
X4TA	The 4 times the value of the X mantissa to A input of mantissa adder signal.
ADSB	The mantissa adder add/subtract flip-flop.
CCNS	The mantissa output unshifted logic signal.
CCR3	The mantissa output shifted right 3 bits logic signal.
CCL3	The mantissa output shifted left 3 bits logic signal.
SPCI	The single precision carry-in/borrow to mantissa adder logic signal.
DPCI	The double precision carry-in/borrow to mantissa adder logic signal.
CI75	The carry-in to bit 75 of the mantissa adder signal.

DPOV	The double precision mantissa adder gating override logic signal.
BX00, BX01, BX02	The B mantissa one octade extension register.
YX00	The Y exponent one-bit extension signal.
AX00, AX01, AX02	The A mantissa one octade extension register.
XX00	The X exponent one-bit extension signal.
YR-1, YR-2, YR-3	The Y mantissa low-order octade extension register.
XR-1	The X mantissa low-order bit (input conversion).
SC0F, SC1F, SC2F, SC3F	The scale count register.
SCEF	The scale count enable flip-flop.
ICRE	The input convert register enable flip-flop.
BXSE	The B side of mantissa adder output logic signal.
DISX	The disable extension register flip-flop.
ICR0, ICR1, ICR2, ICR3, ICR4, ICR5, ICR6, ICR7	The input convert operation register.
BDPD	The double precision operand in B register (tag = 010) logic signal.
ADPD	The double precision operand in A register (tag = 010) logic signal.

DISPLAY PANEL TWO, PAGE THREE LOGIC SIGNALS

Page three of display register number two is used to display the status of the memory control of the CPU cabinet. The logic signals and flip-flops (refer to table 4-6) that are displayed are as follows:

RQT0, RQT1, RQT2, RQT3, RQT4, RQT5, RQT6, RQT7, RQT8, RQT9, RQTA, RQTB	The request trap register.
RQR0, RQR1, RQR2, RQR3, RQR4, RQR5, RQR6, RQR7, RQR8, RQR9, RQRA, RQRB	The request register.
CSC1, CSC2, CSC3, CSC4	The channel A sequence count register.
CAPFA, CAPFB, CAPFC, CAPFD, CAPFE	The memory address register priority over look ahead address register logic signals.
SRL0, SRL1, SRL2	The sum of residue for the address in LAR signals.
SRM0, SRM1, SRM2	The sum of residue for the address in MAR signals.

ATEF	The address transmission error flip-flop.
TRYF	The address retry flip-flop.
CMGO	The channel go (to complete a memory cycle) flip-flop.
MAOF	The memory access obtained flip-flop.
RDFF	The read phase control flip-flop.
CINF	The clock inhibit control flip-flop.
WAIT	The general purpose delay flip-flop.
LOG1, LOG2	The error control flip-flops.
ABRF	The abort memory cycle flip-flop.
CARQ	The channel A request to ports flip-flop.
MI51R	Memory bit 51 (the multiplexor parity bit).
MI48	Memory bit 48 (the memory protect bit).
PTGO	The port go (to complete a memory cycle) signal.
HAR0, HAR1, HAR2, HAR3	The hold address for return (for each port) signals.

PANEL 2, PAGE 4 LOGIC SIGNALS

Page four in maintenance display register two is used to show the status of string operations (families F, G, H of the data processor) in the CPU cabinet. The logic signals and flip-flops (refer to table 4-7) that are displayed are as follows:

TU1F, TU2F, TU4F, TU8F	The T register for string operation decoding (1, 2, 4, 8 bits).
EEND	The end of edit cycle control flip-flop.
FINI	The end of the end edit cycle control flip-flop.
EXSF	Execute single micro operator (S and D pointers for enter edit). Control flip-flop.
EXPF	Execute single micro operator (single pointer for enter edit). Control flip-flop.
RETF	Return to using operation control flip-flop.
RNTF	Reentrant from interrupt control flip-flop.
NVLF	Not valid control flip-flop.
MPOP	Micro program control flip-flop.
JU0F, JU1F, JU2F, JU3F, JU4F, JU5F, JU6F	String operations sequence counter flip-flops.

SSZ1, SSZ2	Source size flip-flops.
DSZ1, DSZ2	Destination size flip-flops.
SI01, SI02, SI03, SI04	Source input buffer register flip-flops.
DI01, DI02, DI03, DI04	Destination input buffer register flip-flops.
QU1F, QU2F, QU3F, QU4F	Logical flip-flops used for string operations.
DGSF	Destination greater than source control flip-flop.
LHFF	Lower half control flip-flop.
RPZF	String operations control flip-flop.
XROF	X register occupied control flip-flop.
SOPF	Source pointer equals an operand control flip-flop.
UPDF	Update control flip-flop.
SPRF	Source pointer read only control flip-flop.
DPRF	Destination read only control flip-flop.
TFFF	True false (string operation comparison) control flip-flop.
TFOF	True false flip-flop occupied control flip-flop.
OFFF	Overflow control flip-flop.
FLTF	Float control flip-flop.
EQVF	Equivalent control flip-flop (sum equal to zero).
EXTF	External sign bit flag control flip-flop.
QUDF	Segmented array (QF04) control flip-flop.
QUCF	**Memory protect (QF03) control flip-flop.**
QUBF	**Presence bit (QF02) control flip-flop.**
QUAF	**Invalid operation (QF01) control flip-flop.**

PANEL 2, PAGE 5 LOGIC SIGNALS

Page five in maintenance display register two is used to show the status of the memory controller, memory tester logic, and interrupt controller of the data processor in the CPU cabinet. The logic signals and flip-flops (refer to table 4-7) that are displayed are as follows:

BYR00, 01, 02, 03, 04 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19	The memory tester bypass register.
MTST	The memory test mode control flip-flop.
TV0, 1, 2	The memory test type register.
JV0, JV1	The memory tester sequence counter.
ONCK	
CMPE	The memory tester compare error control flip-flop.
TABT	The memory tester test all bits control flip-flop.
ALTWC	The memory tester alternate worst case control flip-flop.
M151W	The write bit 51 memory tester parity bit flip-flop.
CBPW	The memory tester check bit write parity flip-flop.
WEFW	The memory tester word parity write bit flip-flop.
CB1W, CB2W, CB3W, CB4W CB5W, BC6W	The memory tester check bit memory write register.
IT00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 10	The interval timer register.
IMTV	
ECSF	The freeze parameters control flip-flop.
EXTI	The external interrupt flip-flop.
INTV	The interval timer error flip-flop.
INTE	The interval timer enable logic signal.

DISPLAY PANEL THREE, PAGE ZERO LOGIC SIGNALS

Page zero of register number three is used to display the status of the family C (scan operations) of the data processor, in the CPU cabinet. The logic signals and flip-flops (refer to table 4-8) that are displayed are as follows:

SL10, SL11, SL12, SL13, SL14, SL15, SL16, SL17, SL18, SL19, SL20, SL21, SL22, SL23	The status change levels for units 10, through 23.

SE10, SE11, SE12, SE13, SE14, SE15, SE16, SE17, SE18, SE19, SE20, SE21, SE22, SE23	The status change read levels for units 10 through 23.
SCL0, SCL1, SCL2, SCL3, SCL4, SCL5, SCL6, SCL7, SCL8, SCL9	The status change levels for units zero through nine.
SCE0, SCE1, SCE2, SCE3, SCE4, SCE5, SCE6, SCE7, SCE8, SCE9	The status change read levels for units zero through nine.
TOD0, TOD1, TOD2, TOD3	The four low-order bits of the time of day register.

DISPLAY PANEL THREE, PAGE ONE LOGIC SIGNALS

Page one of register number three is used to display the status of the multiplexor function, and the interrupt controller function, of the data processor, in the CPU cabinet. The logic signals and flip-flops (refer to table 4-8) that are displayed are as follows:

BAFP4	The peripheral buffer available for burst logic signals.
VALV	The valid vector logic signal.
BURV	The burst request vector logic signal.
RDEV	The result descriptor present vector logic signal.
PBZV	The pseudo busy vector logic signal.
PC0F, PC1F, PC2F, PC3F	The priority sequence count register.
SPWF	The scratch pad write logic signal.
BFWF	The buffer write logic signal.
XPBA	The transfer peripheral buffer address logic signal.
XPDR	The transfer peripheral data register logic signal.
XPCC	The transfer peripheral character count logic signal.
XACC	The transfer accumulator logic signal.
XBBA	The transfer burst buffer address logic signal.
MPXT	The multiplexor test logic signal.
INM0, INM1, INM2, INM3, INM4, INM5, INM6, INM7, INM8, INM9	The interrupt mask register.

INR0, INR1, INR2, INR3, INR4, INR5, INR6, INR7, INR8, INR9	The interrupt register.
MER0, MER1, MER2	The multiplexor error register.
CSV0, CSV1, CSV2, CSV3, CSV4	The channel save register.
MREC	The maintenance recycle logic signal (allows repetative IO operations).
SPW0, SPW1, SPW2, SPW3, SPW4	The scratchpad word address register.
SPPE1, SPPE2	The scratchpad parity error flip-flops.
SPRF1, SPRF2	The scratchpad read flip-flops.

DISPLAY PANEL THREE, PAGE TWO LOGIC SIGNALS

Page two of register number three is used to display the status of families B and E of the data processor, in the CPU cabinet. The logic signals and flip-flops that are displayed (refer to table 4-9) are as follows:

TE0F, TE1F, TE2F, TE3F	The T register for family E.
JBCF	The J count bus control flip-flop.
JE0F, JE1F, JE2F, JE3F, JE4F, JE5F, JE6F	The sequence count register for family E.
QE0F, QE1F, QE2F, QE3F	The logical flip-flops for family E operations.
SMVF	The enable scale right PROM (generates TOA, TOM, and DIS values).
MPYF	The scale right multiply (times ten), raised to the value of the scale factor enable logic signal.
SUBF	The last octade (of shift register multiplication) was a subtract logic signal.
LC0F, LC1F, LC2F, LC3F	The loop count register.
DPFF	The double precision scale right multiplier flip-flop.
SF0F, SF1F, SF2F, SF3F	The scale factor register.
MP35	The scale right multiplied by 3 or 5 octade logic signal.
DBZF	The destination bit zero flip-flop.
FNWF	The final word flip-flop.
ZROF	The Z register occupied flip-flop.

TB0F, TB1F, TB2F, TB3F	The T register for family B.
JB0F, JB1F, JB2F, JB3F	The sequence counter for family B operations.
QB1F, QB2F, QB3F, QB4F	The logical flip-flops for family B operations.

PANEL 3, PAGE 3 LOGIC SIGNALS

Page three in maintenance display register three is used to show the status of the stack controller and the transfer controller of the data processor in the CPU cabinet. The logic signals and flip-flops (refer to table 4-9) that are displayed are as follows:

BZ62	The gate logic signal to transfer bits 19:20 from the B register to the Z6 bus, in the transfer controller.
AZ63	The gate logic signal to transfer bits 19:20 from the A register to the Z6 bus, in the transfer controller.
AZ62	The gate logic signal to transfer bits 39:20 from the A register to the Z6 bus, in the transfer controller.
AZ61	The gate logic signal to transfer bits 50:11 from the A register to the Z6 bus, in the transfer controller.
BZ61	The gate logic signal to transfer bits 50:3 from the B register to the Z6 bus, in the transfer controller.
CZ63	The gate logic signal to transfer bits 19:20 from the C register to the Z6 bus, in the transfer controller.
CZ62	The gate logic signal to transfer bits 39:20 from the C register to the Z6 bus, in the transfer controller.
CZ61	The gate logic signal to transfer bits 50:11 from the C register to the Z6 bus, in the transfer controller.
YZ62	The gate logic signal to transfer bits 19:20 from the Y register to the Z6 bus, in the transfer controller.
XZ63	The gate logic signal to transfer bits 19:20 from the X register to the Z6 bus, in the transfer controller.
XZ62	The gate logic signal to transfer bits 39:20 from the X register to the Z6 bus, in the transfer controller.
XZ61	The gate logic signal to transfer bits 50:11 from the X register to the Z6 bus, in the transfer controller.
YZ61	The gate logic signal to transfer bits 50:11 from the Y register to the Z6 bus, in the transfer controller.
ZZ63	The gate logic signal to transfer bits 19:20 from the Z register to the Z6 bus, in the transfer controller.

ZZ62	The gate logic signal to transfer bits 39:20 from the Z register to the Z6 bus, in the transfer controller.
ZZ61	The gate logic signal to transfer bits 50:11 from the Z register to the Z6 bus, in the transfer controller.
TOA0, 1, 2, 3, 4, 5	The top of apperature register.
TOM0, 1, 2, 3, 4, 5	The top of mask register.
DIS0, 1, 2, 3, 4, 5	The displacement register.
JS1F, 2F, 3F, 4F	The stack controller sequence counter.
SOIF	The stack overflow interrupt flip-flop.
QS1F, 2F, 3F	Stack controller logical flip-flops.

PANEL 3, PAGE 4 LOGIC SIGNALS

Page four in maintenance display register three is used to show the status of the program controller, CPU clock control logic, and the port control logic for the memory exchange in the CPU cabinet. The logic signals and flip-flops (refer to table 4-10) that are displayed are as follows:

CPA1, 2, 4, 8	The CPU clock counter low order flip-flop bits.
ICRF	The increment CPIR and CTIR remember control flip-flop.
ICCF	The increment CPIR and CTIR normal control flip-flop.
FWFF	The first word fetch flip-flop.
PRVA	The PROF and VARF valid logic term.
CPIR0, 1	The PIR word boundary crossed counter.
WPTF	Write PIR or TIR flip-flop.
WBCF	Word boundary crossed flip-flop.
CTIR	TIR word boundary crossed flip-flop.
CSR0, 1, 2	Syllable counter for the syllable from which the present operator was strobed.
SSR0, 1, 2	Syllable counter for the syllable that initiated a table enter edit operator.
WPIR	Write PIR - return from table mode flip-flop.
SECF	Syllable execute complete level save flip-flop.
VSJK	Vector strobe fetch or store flip-flop.

QP1F, 2F, 3F, 4F, 5F, 6F, 7F, 8F	Program controller logic flip-flops.
STMC	Start memory cycle save flip-flop.
JP00, 01, 02	Program controller sequence control register.
MPRCB	Protected write logic signal from external subsystem device.
MWRCB	Memory write request signal from external subsystem device.
MNRFB	Memory not ready signal to an external subsystem device.
MREQB	Memory request signal from external subsystem device.
MTEXB	Memory transmission error signal to external subsystem device.
MAPLB	Memory address parity level from an external subsystem device.
MAOFB	Memory access obtained level to an external subsystem device.
MABXB	Memory access begun signal to an external subsystem device.
MDY1F, 2F, 3F, 4F	Memory clock delay register for an external subsystem device.
MATEB	Memory address transmission error signal for an external subsystem device.
MTRYB	Address retry signal to an external subsystem device.
MHOLD, MDRYB	Logical flip-flops used to send an initiate memory cycle (IMC) to a memory port from an external subsystem device.
MIF51	Information parity bit for an external subsystem device memory cycle.
MHARB	Hold address for return for external subsystem.
MCBQF	Channel B request signal to memory exchange port.
MRDBF	Read phase flip-flop for channel B.
MBSC0, 1, 2, 3	Sequence control counter for channel B.

PANEL 3, PAGE 5 LOGIC SIGNALS

Page five in maintenance display register three is used to show the status of the look ahead logic and other interrupt controller functions, and the display register validity logic of the data processor in the CPU cabinet. The logic signals and flip-flops (refer to table 4-10) that are displayed are as follows:

LRAP	The address parity signal for the lock ahead logic.
LRIL	The invalid address-local signal for the look ahead logic.

LRAR	The address residue signal for the look ahead logic.
LRDM	The read data multitimer signal for the look ahead logic.
LRIG	The invalid address-global signal for the look ahead logic.
LRGN	The global memory not ready signal for the look ahead logic.
LAER	The look ahead logic memory error signal.
OPTF	The optional adapter test flip-flop (used in MTCE mode).
IML0, 1, 2	The consecutive interrupt counter for detection of a superhalt condition.
GCDS	The general control disable logic signal.
ABR1	The abort clock save logic signal.
ABEI	The abort interrupt controller logic signal.
ILDM	The interrupt load micro-program logic signal.
SEIN	The syllable execute complete level interrupts enable signal.
INCT	The inconsistant P3 parameter logic signal.
MEWT	The families memory cycle wait logic signal.
BDST	The Maintenance Display Processor (MDP) test logic signal.
ABIT	The abort interrupt logic signal.
INF+1	The INFF flip-flop delayed by 1 clock pulse logic signal.
SEC+2	The syllable execute complete level delayed by 2 clock pulses logic signal.
SEC+1	The syllable execute complete level delayed by 1 clock pulse logic signal.
AYER	The any memory error for event logic signal.
INFF	The inhibit operator from P register logic signal.
ALSB	The allow strobe logic signal.
RTRY	The retry logic signal.
MPXI	The multiplexor initiate burst request remembered logic signal.
MPXB	The multiplexor burst logic signal.
MPXG	The multiplexor granted for burst logic signal.

DR00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31	The display register valid flip-flops for each of the 32 display registers.

PANEL 4, PAGE 0 LOGIC SIGNALS

Page zero in maintenance display register four is used to show the status of the maintenance and event logic in the CPU cabinet. The logic signals and flip-flops (refer to table 4-11) that are displayed are as follows:

PLK1	The phase lock two logic signal.
PLK0	The phase lock one logic signal.
PSOR	The pseudo OP flip-flop.
WMMF	The write main memory flip-flop.
RMMF	The read main memory flip-flop.
WIMF	The write IC memory flip-flop.
RIMF	The read IC memory flip-flop.
AMMF	The access main memory flip-flop.
AIMF	The access IC memory flip-flop.
MEXI	The mask external interrupt flip-flop.
HALT	The halt flip-flop.
ARPT	The Anti-repeat flip-flop.
JCMP	The sequence (J) count micro program select flip-flop.
JCS00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 10	The sequence (J) count save register.
SRS0, 1, 2, 3	The strobe save register.
OSR0, 1, 2, 3	The OP code save register.
VARS	The variant bit save flip-flop
VCTS	The vector save flip-flop.
EDTS	The edit save flip-flop.
TEDS	The table save flip-flop.

DISPLAY PANEL FOUR, PAGE ONE LOGIC SIGNALS

Page one of register number four is used to display the status of the maintenance and event logic, of the CPU cabinet. The logic signals and flip-flops (refer to table 4-11) that are displayed are as follows:

EV01, EV02, EV03, EV04, EV05, EV06, EV07, EV08, EV09, EV10, EV11, EV12, EV13, EV14, EV15, EV16, EV17, EV18, EV19, EV20, EV21	The event register.
CCSF	The count clock select flip-flop.
MEVF	The multiple event flip-flop.
HOEF	The halt on event flip-flop.
ECT0, ECT1, ECT2, ECT3, ECT4, ECT5, ECT6, ECT7	The event counter.
ICOR	The inhibit memory correction logic signal.
MPBI	The mask presence bit interrupt logic signal.
MIAI	The mask invalid address interrupt logic signal.
ESTP	The event stop logic signal.
EJCMP	The micro program J count select logic signal.
EJC00, EJC01, EJC02, EJC03, EJC04, EJC05, EJC06, EJC07, EJC08, EJC09, EJC10	The micro program J counter.
HLTD	The halted flip-flop.
ILHD	The inhibit look ahead logic flip-flop.
LODS	The load select flip-flop.
LAVF	The look ahead valid flip-flop.
EST4, EST5, EST6, EST7	The strobe event logic signals.
EOP0, EOP1, EOP2, EOP3	The operator code event logic signals.
EVCT	The vector event logic signal.
ETED	The table edit event logic signal.

EEDT	The edit event logic signal.
EVAR	The variant mode event logic signal.

PANEL 4, PAGE 2 LOGIC SIGNALS

Page two in maintenance display register four is used to show the status of the clock control logic, and the time of day register of the multiplexor function in the CPU cabinet. The logic signals and flip-flops (refer to table 4-12) that are displayed are as follows:

CPC05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28	The high order twenty-four bits of the CPU timer register.
TD00, 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35	The high order flip-flops of the time of day register.

PANEL 4, PAGE 3 LOGIC SIGNALS

Page three in maintenance display register four is used to show the status of the store control memory exchange logic, and the global memory control logic. The logic signals and flip-flops (refer to table 4-12) that are displayed are as follows:

PS0F0, 1, 2, 3 PS1F0, 1, 2, 3 PS2F0, 1, 2, 3	The port sequence flip-flops for the four ports of local memory.
WCCF0, 1, 2, 3	Clear write operation flip-flops for the four local memory ports.
PEDF0, 1, 2, 3	Parity error disable flip-flops for the four local memory ports.
IMCF0, 1, 2, 3	Initiate memory cycle flip-flops for the four local memory ports.
CAOF0, 1, 2, 3	Channel A occupying port priority flip-flops for four local memory ports.
CBOF0, 1, 2, 3	Channel B occupying port priority flip-flops for four local memory ports.
MTRIP	The enable scan timer flip-flop.
MTMR0, 1, 2	The scan not ready timer register.
WSTF0, 1, 2, 3	The memory write strobe for memory port 0, 1, 2, or 3.
MSPY, MSPX	Scan control flip-flops.
GS0F, 1F, 2F, INCF, GRDF, GABF, CRFF, GT0F, 1F, 2F, GOAF, GOBF, GAOF, EGTM	Global memory (or memory control III) control flip-flops.

PANEL 4, PAGE 4 LOGIC SIGNALS

Page four in maintenance display register four is used to show the status of the micro program module flip-flops in the CPU cabinet. The logic signals and flip-flops (refer to table 4-13) that are displayed are as follows:

MM1E	Micro module one enable flip-flop.
M10F, 11F, 12F, 13F, 14F, 15F, 16F, 17F, 18F, 19F	Micro module one address flip-flops.
MM2E	Micro module two enable flip-flop.
M20F, 21F, 22F, 23F, 24F, 25F, 26F, 27F, 28F, 29F	Micro module two address flip-flops.
MM3E	Micro module three enable flip-flop.
M30F, 31F, 32F, 33F, 34F, 35F, 36F, 37F, 38F, 39F	Micro module three address flip-flops.
MMPD	The micro module parity disable for first clock pulse logic signal.

PANEL 4, PAGE 5 LOGIC SIGNALS

Page five in maintenance display register four is not used, and therefore has no flip-flops specified. This display register page is reserved for future expansion of the display registers.

Maintenance Processor Control Panel and Display

The maintenance display processor control panel (see figure 4-6) is located below maintenance display register number two, on the maintenance display register panel. When the hinged maintenance display panel cover is closed the MDP processor control panel is not visible, and the controls for the processor cannot be exercised. Figure 4-6 shows the maintenance and display processor controls and indicators, and also shows the system maintenance control panel.

The maintenance display processor control panel consists of the following controls and indicators:

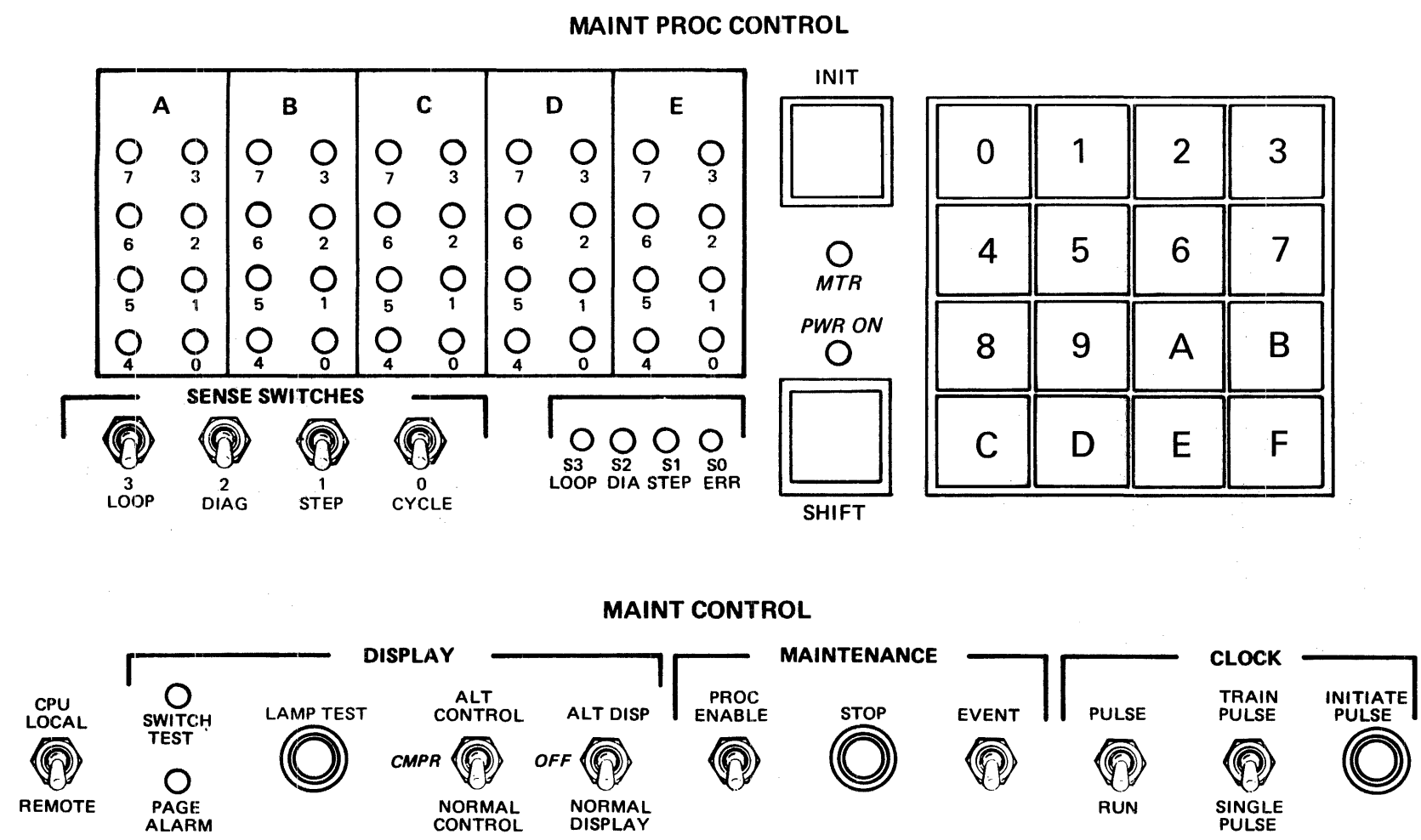
a. The A, B, C, D, and E display indicators. These indicators are driven by the KBI0 Controller in the micro-processor, and are used for operator interface.

b. The INIT (initiate) pushbutton switch. This switch is used to clear and reinitialize the microprocessor.

c. The MTR indicator lamp. This lamp is used to indicate when the microprocessor is running a self diagnostic test on itself, and/or the MDP. If such a test is being performed the lamp is illuminated. The MTR level is used to select one of two areas of micro-processor PROM storage.

d. The PWR ON (power on) indicator lamp. This lamp is used to indicate that power is applied to the micro-processor.

e. The SHIFT (shift) pushbutton switch. This pushbutton is used to cause an expansion of the sixteen key positions of the keyboard. If the SHIFT pushbutton is not depressed, the value of the keyboard selects one character position in the first set of sixteen characters. If the SHIFT pushbutton is depressed the value of the keyboard selects one character position in the second set of sixteen characters.



MV 1608

Figure 4-6. Maintenance Display Processor Control Panel

f. The sixteen key keyboard. The sixteen keys on the keyboard are labeled 0, 1, 2, 3, 4, 5, 6, 7, 8, A, B, C, D, E, and F. These sixteen keys are used to enter data into the microprocessor.

g. The four SENSE SWITCHES, numbered 0, 1, 2, and 3. These switches are read by the microprocessor, and provide major function control of the microprocessor.

h. The four indicator lamps, numbered S0, S1, S2, and S3. These four lamps are used to indicate error conditions, and other operating conditions while executing MDP programs.

The MAINT CONTROL panel consists of the following controls:

a. The CPU LOCAL/REMOTE selector switch. This switch has two positions which are LOCAL (up), and REMOTE (down). In the LOCAL position the CPU may only access local memory (global memory and scan bus operations are disabled). In the REMOTE position the CPU may access either local or global memory.

b. The SWITCH TEST, and PAGE ALARM indicator lamps. The SWITCH TEST lamp is illuminated if any push-button in the entire MDP is depressed. The PAGE ALARM lamp is illuminated if any of the maintenance display register thumbwheels are not in the detent (page selected) position.

c. The LAMP TEST pushbutton. This switch when depressed, causes all indicators on the display panel to illuminate.

d. The ALT/CMPR/NORMAL CONTROL switch. This switch has three positions which are:

ALT CONTROL (alternate control)	in the up position
CMPR (comparator)	in the middle position
NORMAL CONTROL	in the down position

This switch is used to route control levels from the system control panel and the keyboard to the selected CPU(s).

e. The ALT/OFF/NORMAL display control switch. This switch has three positions which are alternate display (up), normal display (down), or comparator (middle). This switch is used to select which CPU(s) will be displayed by the indicators on the MDP panels.

f. The MAINTENANCE switch group. This group has two switches and one pushbutton in it:

1) PROC ENABLE (processor enable) has two positions which are enable (up), and disable (down). This switch is used to select either the micro-processor or the display logic to control MF10. The micro processor is selected when the switch is up, and the display logic is selected when the switch is down.

2) STOP pushbutton. This switch, when depressed, unconditionally stops the running of the mainframe in event mode.

3) EVENT switch has two positions which are enabled (up), and disabled (down). This switch enables the event logic when the PROC ENABLE switch is in the disabled position.

g. The CLOCK switch group. This group has two switches and one pushbutton in it; the clock mode switch has two positions which are PULSE (up), and RUN (down). In the PULSE position system clock is controlled by the TRAIN PULSE/SINGLE PULSE switch. In the RUN position the system clock is free running.

The clock select switch has two positions which are TRAIN PULSE (up), and SINGLE PULSE (down).

In the TRAIN PULSE position a train of three clock pulses are emitted when the INITIATE PULSE pushbutton is depressed. In the SINGLE PULSE position a single clock pulse is emitted.

The clock INITIATE PULSE pushbutton.

This switch causes a pulse train or a single clock pulse to be emitted when the pushbutton is depressed.

Maintenance Processor Programmers Display Keyboard

The MDP programmers keyboard is located immediately beneath the programmers display panel (refer to figure 4-1). The pushbutton switches and indicators are shown in figure 4-7. The functions of the switches and indicators are as follows.

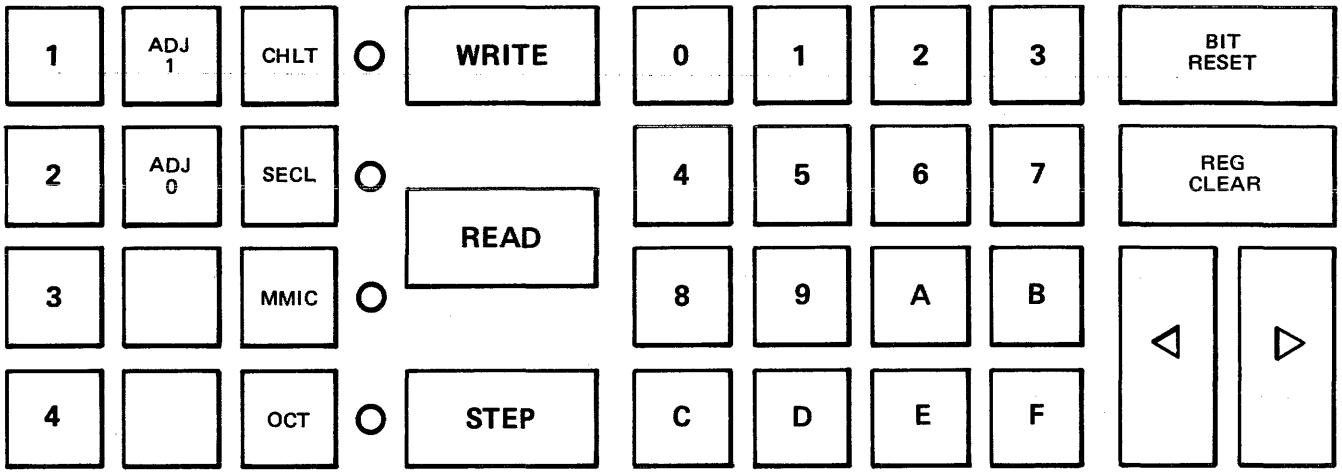


Figure 4-7. Keyboard Pushbuttons and Indicators

Register select switches 1, 2, 3, 4.

One of the four register displays are selected. Depressing one of the four switches will cause the selection to change to that associated register. When a register is selected, the cursor will blink at the leftmost digit of that register and the associated register select indicator will be on.

HEX DIGITS 0 through F switches

When a hex digit switch is depressed, the four flip-flops indicated by the cursor will take the state of that hex character. On releasing the switch, the cursor will move one position to the right.

CURSOR LEFT switch (◁)

Depressing this switch causes the cursor to move one position to the left in the selected display register, (R1, R2, R3, or R4).

CURSOR RIGHT switch (▷)

Depressing this switch causes the cursor to move one position to the right in the selected display register, (R1, R2, R3, or R4).

REG CLR switch

Depressing this switch causes the selected register to be cleared (reset), and the cursor to point to hexadecimal address number nine of the display register.

BIT RESET switch

Depressing the switch causes any flip-flop to be reset when its associated switch is depressed. The bit reset switch must be kept depressed for the entire time the flip-flop switch is depressed to ensure the flip-flop is left in a reset state.

SECL switch and indicator

Depressing the SECL pushbutton causes the state of the SECL flip-flop to change state in the display control logic. The output of the flip-flop is sent to the indicator and to the mainframe. When false, the indicator is off and it has no effect on the mainframe. When true, the indicator is on and the CPU finishes its current instruction and is inhibited from starting the next instruction.

CHLT switch and indicator

Depressing CHLT causes the state of the CHLT flip-flop in the display control logic to change state. The output of the flip flop is sent to the indicator and to the mainframe. When the output is false, the indicator is off and when a conditional halt operator is executed by the mainframe, it acts as a NOOP. When the output of the CHLT flip-flop is true, the indicator is on and when the conditional halt operator is strobed into family F, the operator continues to run without ever developing a SECL, thus stopping the system. A loop timeout interrupt is inhibited when the CHLT flip-flop is set.

STEP switch

The Step switch works in conjunction with the event switch. With the event switch off, the step switch being depressed causes the CPU to begin execution of the next instruction where the CPU has been stopped by the HALT or SECL switch. With the EVENT switch on, depressing STEP causes the mainframe flip flops to go into a normal run condition.

OCT switch and indicator

Depressing the OCT switch causes the OCT flip flop in the display control logic to change state. When the output of the OCT flip flop is false, the indicator is off and the stack registers are displayed in hexadecimal format. When the OCT flip flop output is true, the indicator is on and the stack register mantissa is displayed in octal format.

MMIC switch and indicator

When this switch is depressed, the MMIC flip flop in the Display Control logic changes state. When the flip flop is reset, the indicator is off and the MMIC interface line to the mainframe is false. In this state, any memory cycle initiated from the keyboard will be to IC memory. When the MMIC flip flop is set, the MMIC indicator is on and the MMIC interface line to the mainframe is true. In this state, any memory cycle initiated from the keyboard will be to local memory.

WRITE switch

When depressed, a memory write cycle is initiated. If IC is selected, the write will be from the top of the stack to the IC addressed by the second word in the stack. If local memory is selected, the contents of the A register is written to the address pointed to by the B registers, (bits 0 through 19).

READ switch

When depressed, a memory read cycle is initiated. If IC is selected, the IC addressed by the top of the stack will be read into the top of the stack. If memory is selected, the word addressed by the B registers (0 through 19), will be read into the A register.

ADJ1 switch

When depressed, initiates those local memory read cycles necessary to adjust the top of stack into the A and B registers, (A reg. does not need to be displayed).

ADJ0 switch

When depressed, initiates those memory write cycles necessary to push top of stack into memory.

Logic Indicator Lamps

Four lamps are mounted flush on the keyboard, adjacent to the CHLT, SECL, MMIC, and OCT pushbuttons. These lamps indicate the state of the corresponding logical flip flop. For example, when the CHLT lamp is illuminated, the CHLT flip flop is set, and when the lamp is extinguished, the CHLT flip flop is reset.

SECTION 5

SYSTEM CONCEPT

GENERAL

The B 6800 system consists of a central processing unit, local memory unit(s), a central power cabinet, a maintenance display processor cabinet, peripheral control cabinet(s), and the associated peripheral equipment for input/output. This section generally defines the overall system hardware operation.

The central processing unit (CPU) is the heart of system operations in the B 6800 system; and therefore, while other units of the system will be discussed in this section, the main thrust will be descriptive of the units that are parts of the CPU cabinet. The three main parts of the CPU cabinet are as follows:

a. The data processor

b. The multiplexor

c. The memory control

DATA PROCESSOR

The data processor part of the CPU produces the objective results of a program by performing the necessary arithmetic and logical functions of the program flow.

The data processor contains two major divisions: the functional resources and operator algorithms (figure 5-1). The functional resources are referred to as the "hardcore" of the processor.

The functional resources are the event logic, the micro-program modules, the top of stack registers, the address adder, the multiplexor, and six controllers. The operator algorithms are a group of six families of operators. The operator algorithms provide the logic required to control the functional flow of the program.

OPERATOR FAMILIES

The operator families and functional controllers are linked by 11 busses (bus Z1 through Z6, and Z8 through Z12). These busses provide for data movement and signal routing within the processor (see figure 5-2).

A bus is a group of wires used to transmit signals from one place to another. The busses within the transfer controller are etched on a single card connecting the same bit of all "hard registers" together, i.e., Bit 1 of registers A, B, C, X, Y and Z are all on the same physical card.

The operators are grouped into six groups called the operator families (figure 5-1). The grouping of related operators into families minimizes the logic required in the processor. The six families of operators with a brief purpose for each are:

a. Family A OPS – Arithmetic Operators

b. Family B OPS – Logical Operators

c. Family C OPS – Subroutine Operators

d. Family D OPS – B 6800 Word Oriented Operators

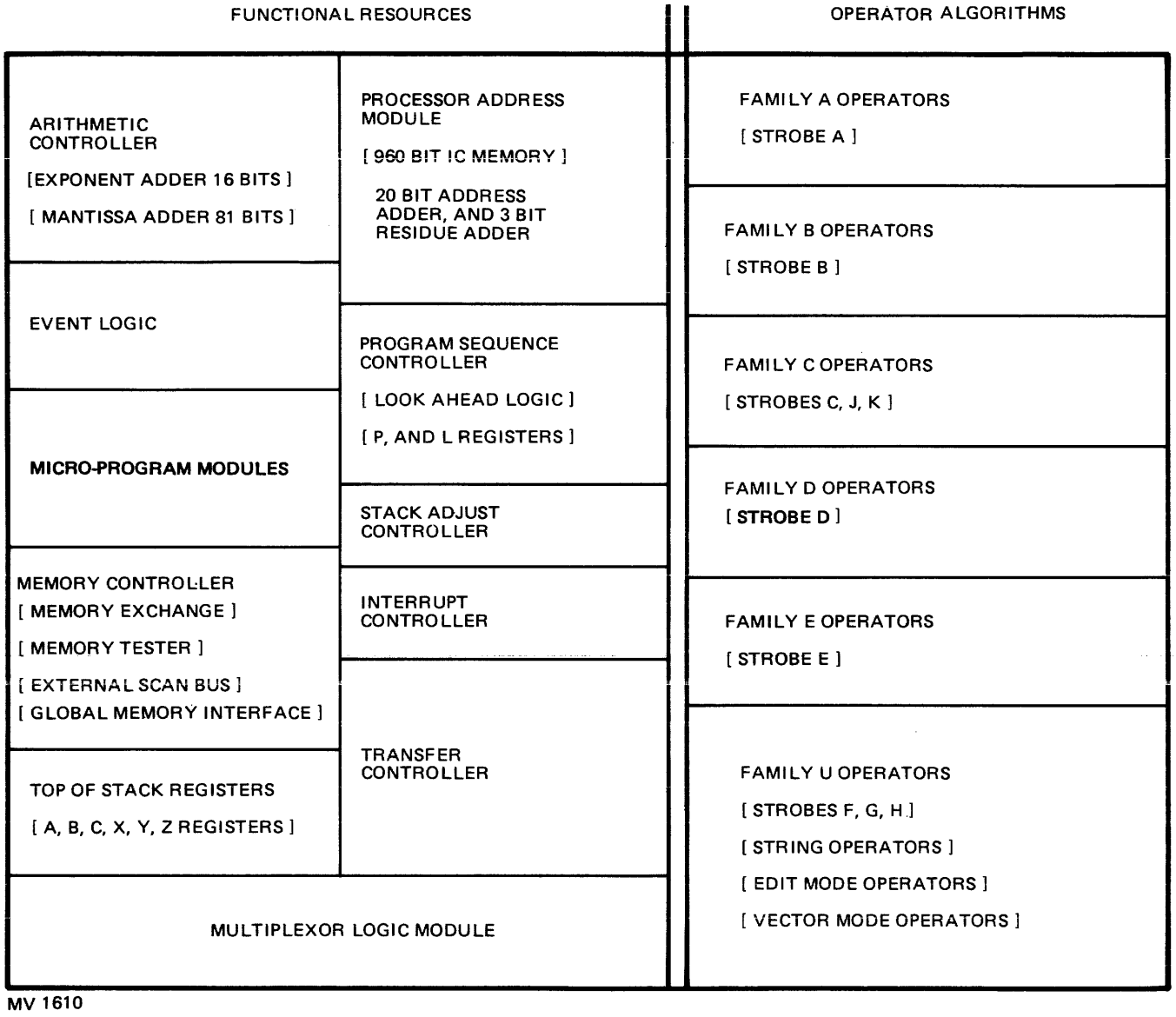


Figure 5-1. B 6800 CPU Organization

e. Family E OPS – Scaling Operators

f. Families F, G, H, OPS – String Operators

PROGRAM CONTROLLER (SEE FIGURE 5-2)

The program controller controls the program flow in the data processor. The program controller determines when the P register contains machine language operators to be executed, which syllable of code is to be executed next, when to

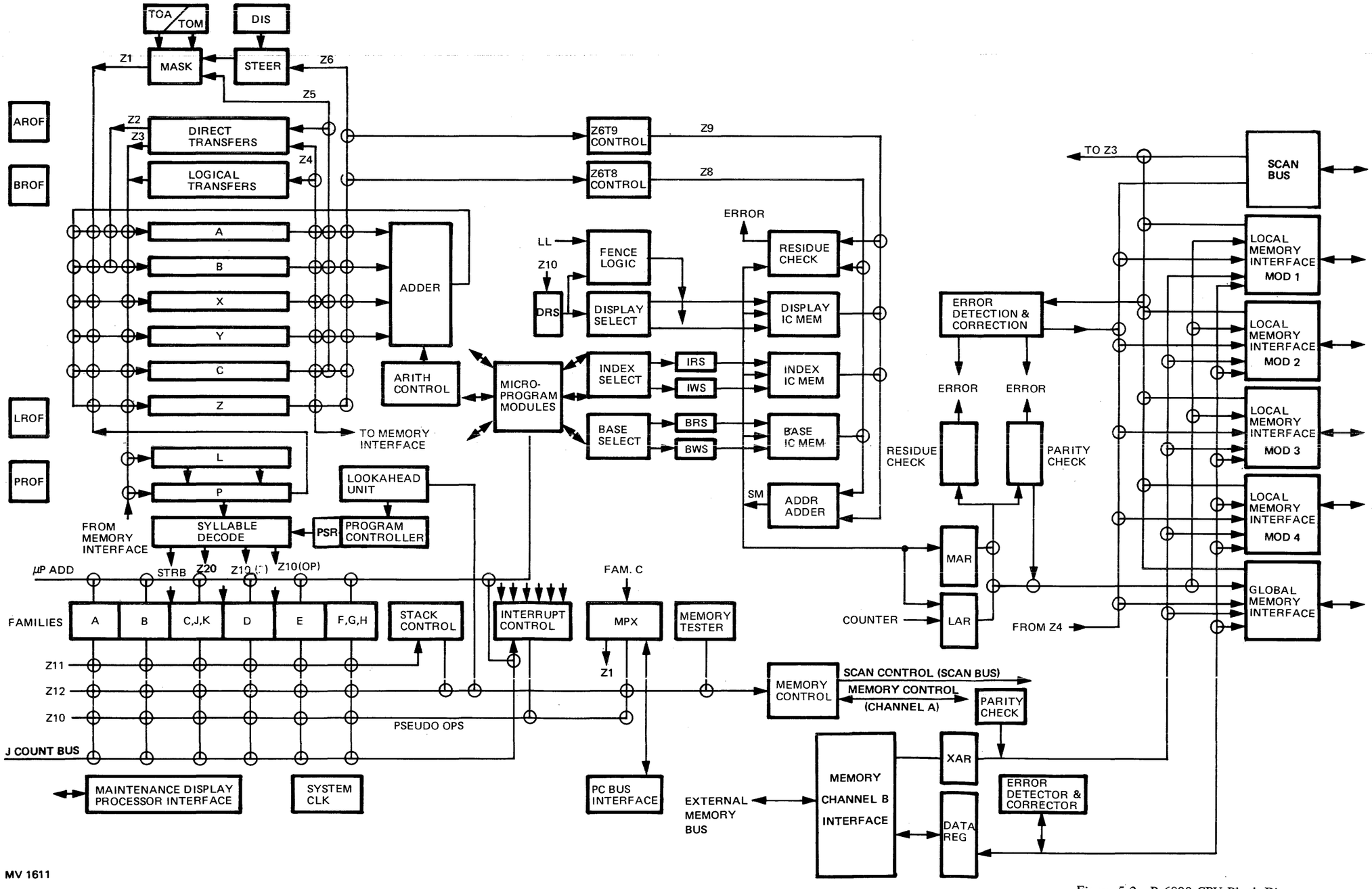


Figure 5-2. B 6800 CPU Block Diagram

replace the contents of the P register and L register, and the source location of the data that is used to replace the contents of the P register and L register. The P register is considered to contain valid program code only if the Program Register Occupied Flip-Flop (PROF) is set.

The Program Syllable Register (PSR) serves as a pointer to the next syllable to be executed from the P register.

Look Ahead Logic

A look ahead function is implemented by provision of the L register, and its associated L Register Occupied Flip-flop (LROF). The function of the look ahead logic is to overlap as far as possible, when fetching code from main memory. In look ahead mode, L acts as a buffer against the P register, such that code is executed from P while L gets the next code word. Code addresses are initially formed by adding the value of the Program Base Register (PBR), to the value of the Program Index Register (PIR). Code addresses are maintained in the look ahead logic, in the Look ahead Address Register (LAR).

In certain modes, the normal sequential code execution, as affected by the look ahead logic, is undesirable, and therefore inhibited. Such cases are branch instructions, subroutine entries and exits (or returns), and table edit mode operations. In the first two cases, new values of PBR and PIR are presented to the program controller, and used as described. In table edit mode, look ahead logic is totally inhibited, and the program controller uses the Table Base Register (TBR) and the Table Index Register (TIR), to form the table mode edit operator code address. Only the P register is used to contain edit mode table operator code (and not the L register). In table edit mode operations, the TIR address register is updated by the program controller, as required.

Integrated Circuit (IC) Memory

The B 6800 system data processor maintains the procedure addresses of the program that is being executed currently in the data processor. These procedure addresses are maintained in a group of address registers commonly identified as IC memory address registers (see figure 5-2). The IC memory address registers are classified as display address, base address, and index registers.

There are 32 display address registers (labeled D0 through D 31) in the data processor. A display register number corresponds to a lexicographical programming level, and locates the absolute local memory base address of the process stack (the MSCW of the stack) for all current programming levels. The maximum number of programming levels (lexacographical levels) in a procedure is fixed by the number of display address registers available in the data processor. The number of programming levels in a procedure is limited to 30, because programming level zero is required for the MCP, and programming level one is required for the segment descriptor index. The bottom of a stack is identified by the address located in the BOSR register, which was identified earlier in this manual. The top of a stack is identified by the address located in the S register which was also identified earlier in this manual (refer to section three in this manual).

There are eight base address registers in the data processor. The eight base address registers are identified as follows:

Base Register Number	Base Register Name	Register Usage
0	PBR	The base address of the program code segment.
1	SBR	The base address of string source data.
2	DBR	The base address of string destination data.

Base Register Number	Base Register Name	Register Usage
3	TBR/BUF2	The base address of table program code, or alternatively a temporary buffer for storing an address value.
4	S	The address of the top word in the current stack.
5	SNR	The stack number register. The stack number is used to contain a vector value for locating the current stack descriptor. The vector value is an index on the address of the stack vector descriptor, for locating the stack descriptor.
6	PDR	The program dictionary register. This register is used to contain the address of the base of the current program code segment descriptor in memory.
7	TEMP	The temporary register. This register is a general purpose register used to store addresses temporarily

There are eight index address registers in the data processor. The eight index address registers are as follows:

Index Register Number	Index Register Name	Register Usage
0	PIR	The program index register. The program index value is an index on the base address that is contained in the PBR register. The sum of PBR and PIR is the absolute address of the word of program code that is presently in the P register.
1	SIR	The source index register. The source index value is an index on the base address that is contained in the SBR register. The sum of SBR plus SIR defines the address of a word of source data for string operations.
2	DIR	The destination index register. The destination index value is an index on the base destination register. The sum of DBR plus DIR defines the address of a word of destination data for string operations.
3	TIR/BUF3	The table index register. The table index value is an index on the address that is contained in the TBR register. The sum of TBR and TIR defines the address of the word containing the micro operators, in the table code. When this address register is not being used for table type operations it is alternatively used (as BUF3) for temporary storage of other address values.
4	LOSR	The limit of stack register. This register contains the upper stack boundary address for the current procedure. This register limits the size of the stack.

Index Register Number	Index Register Name	Register Usage
5	BOSR	The bottom of stack register. This register contains the lower boundary address for the current stack.
6	F	The F register. This register contains the address of the last MSCW for the current process stack in memory. The F register, and the display register that corresponds to the present lexagographical level, contain the identical address value.
7	BUF	The buffer address register. The buffer is used to temporarily store addresses.

Address Adder and Residue Test Logic (Refer to Figure 5-2)

The address adder is a shared mechanism through which all addresses used within the B 6800 system are manipulated. Figure 5-2 shows this mechanism, together with associated data paths, and data integrity residue generation and check blocks.

All traffic to and from the IC memory is conducted through the address adder, or the Z8 and Z9 busses to the address adder. Data integrity within all of these blocks is maintained by modulo three residue checking. This guarantees to detect any single bit error, and some multiple bit errors that occur in IC memory, or the address adder. An error in the modulo three residue generation circuit, or in the residue check circuit is also detected.

Any addressing error in the address adder, or in the residue check circuit is a fatal condition, and results in an "abort" type interrupt condition.

TRANSFER CONTROLLER (REFER TO FIGURE 5-2)

The transfer controller has two major sections (see figure 5-3): a hard register section, referred to as stack registers, for data and program information, and an internal data transfer section. Six busses, Z1 through Z6, are used for the normal data movement to and from the hard registers. Z1, Z2, and Z3 are input busses to these registers, and Z4, Z5, and Z6 are output busses. The capacity of each bus is 51 bits.

Three special busses are used for arithmetic operations (see figures 5-3 and 5-6).

Stack Registers (Refer to Figure 5-3).

Each information register has 51 bit positions. Registers A, B, C, X Y and Z are for information handling during program flow. Registers P and L contain B 6800 program words. The P and L registers contents are never written into Memory.

The Z3 and Z4 busses provide for bi-directional data flow between the hard registers and memory or the multiplexor.

The A and B registers are the top of stack registers, and X and Y are normally second-word information registers for double-precision operands. Registers C, and Z are general purpose registers which provide temporary storage during operator execution.

Internal Data Transfer Section (Refer to Figure 5-3)

The internal transfer section permits the following data transfers between stack registers:

a. A direct, full-word transfer path using the Z5 and Z2 busses.

b. A logical transfer path to create the results of the Family B (logical) operators, using the Z4 and Z3 busses. The logical transfer path also provides one additional full word transfer path between registers.

c. A steering Network and Mask network providing a field displacement between stack registers using the Z6 and Z1 busses.

d. A transfer path to the address adder via the Z6 to Z8 or Z9 busses. This path extracts one of four fields, (39:20), (35:16), (19:20) or (13:14), from a stack register during execution of operator syllables.

e. A data movement path to and from the high speed adder via the AA, BB, and SL busses.

Mask and Steering

The mask and steering network moves bit fields from register to register, via the Z6 and Z1 busses. All bits are transferred to and from the busses in parallel. Two pointers (TOA/TOM) set up a "window" defining the upper and lower limit of the bits being transferred to the accepting data register. A displacement register (DIS) shifts the bits to the right, 0 to 47 bits from the position previously held in the sending data register. The three controls used to steer and mask are as follows:

1. TOA – the highest bit position of the accepting field (highest bit of the window).

2. TOM – the highest bit position to be inhibited on the transfer (lowest bit of the window).

3. DIS – a right shift of the bits through the steering matrix.

Registers TOA, TOM, and DIS are set by the operator families or other controllers.

Mask and Steering Example

Assume the C register contains a stuffed indirect reference word (SIRW) and it is necessary to extract the STKNR (stack number) field (bits 45:10) and place these bits into the index field of the C register. The logic sets the window TOA := 29, TOM := 19, as shown in figure 5-4. The displacement register is set to 16: DIS := 16. The actual starting bit of the field is calculated as: TOA + DIS = 29 + 16 = 45.

All Bits in the C register are gated to the Z6 bus. The bits (except tag) are then shifted 16 places to the right with only the bits that align with the window appearing on the Z1 bus. The Z1 bus is then gated to the C register, with the masked fields destroyed or retained; if the masked field is to be retained, the C register must be gated onto the Z5 bus as "prior content".

If no register is gated on the Z5 bus during a Z1 bus to Z6 bus transfer, the masked field is cleared.

In the example shown in figure 5-4, a field of ten bits is transferred from one field location in the C register, to another field location in this same register. Because the STKNR field of the C register lies outside of the receiving field range, bits 45:10 is cleared, and bits 29:10 will contain the STKNR value at the conclusion of the example operation. Bit fields 47:18, and 19:20 of the C register are cleared and only 50:03 remain unchanged.

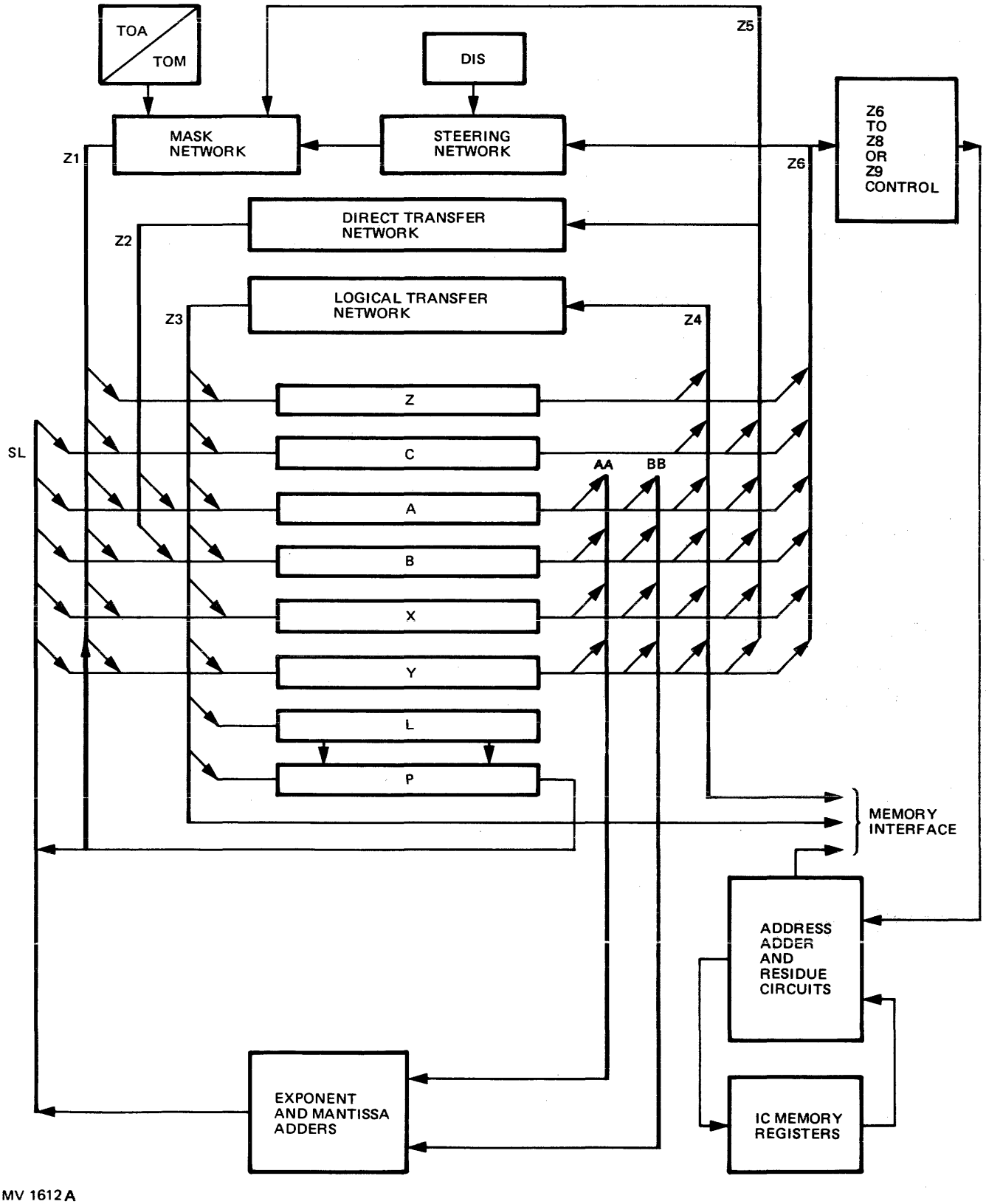


Figure 5-3. Internal Data Transfer Section

Stack Controller

The B 6800 provides automatic stack adjustment as required by the operators. These requirements are supplied to the stack controller on the Z11 bus from the operator families and other functional controllers.

The stack controller manipulates data between main memory and the A and B registers during both the pop-up and push-down cycles. The X and Y registers are included in the adjustment cycles when double-precision operands are involved.

A typical program stack is shown in figure 5-5. The stack controller determines whether a push-up or push-down cycle will be initiated. All other Controllers remain idle until an adjust complete signal is sent to the controller that initiated the adjustment.

ARITHMETIC CONTROLLER (REFER TO FIGURE 5-6)

The arithmetic controller is a functional controller between the stack registers (A, B, C, X, Y and Z) and the exponent and Mantissa Adders. This controller is enabled by the family A operators and other operator families that require the use of these facilities.

Exponent and Mantissa Adders

Figure 5-6 shows the logical path of data flow to and from the exponent and mantissa adders. The exponent adder is composed of a sixteen bit full adder/subtractor circuit, and the mantissa adder is composed of an 81 bit full adder/subtractor circuit. The inputs to the two adder circuits, and the outputs from the adder circuits are directed from and to the stack hardware registers by the arithmetic controller.

The arithmetic controller and the two adder circuits are capable of performing complete double precision mathematics in one continuous synchronized operation. The arithmetic controller gates both the exponent and mantissa portions of both halfs of a double precision operand to the two adder circuits in a single operational step. Exponent adder operations are only performed during multiply or divide functions, and for mantissa alignments.

Each of the two adder circuits consist of an A input (AA), a B input (BB), and a C (SL) resultant output. During a double precision ADD (80) operation the A input to the mantissa adder consists of the 78 bits of the mantissa field from the double precision operand in the A and X registers. The B inputs to the two adders for a double precision ADD operation is the same as the A inputs but is derived from the B and Y registers. After the inputs to the two adders have been routed to the adder inputs, by the arithmetic controller, the ADD operation is performed in one step. After the ADD algorithm is completed, the resultant sum of the two numbers is routed by the arithmetic controller back to the proper stack register(s).

INTERRUPT CONTROLLER (FIGURE 5-2)

The interrupt controller provides a method of temporarily interrupting the program flow when a predetermined interrupt condition arises.

Interrupt Parameters

The controller sets up the necessary control words in the stack for entry into the interrupt handling procedure of the MCP. Three words are placed in the stack by the interrupt controller or the operator that caused entry to the interrupt controller. These three words were described in section 2 under the subheadings INTERRUPT PARAMETER WORDS, P1 PARAMETER, P3 PARAMETER, and P2 PARAMETER.

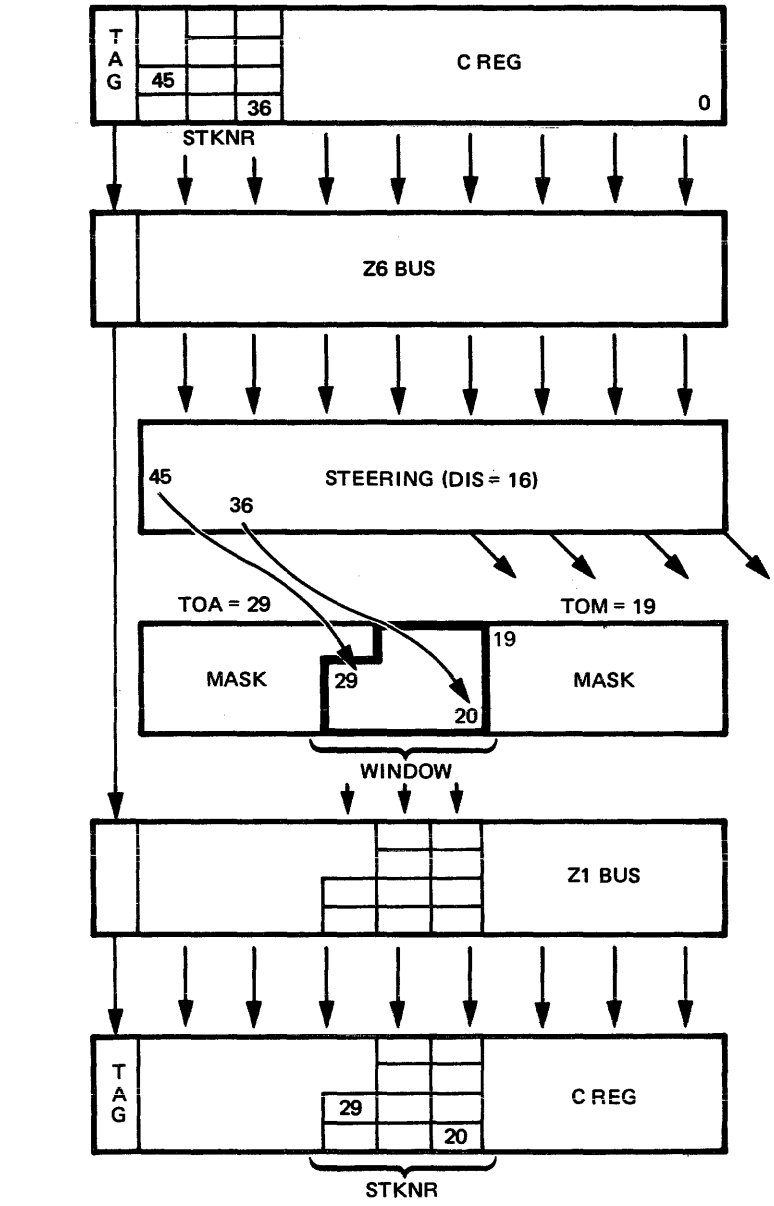
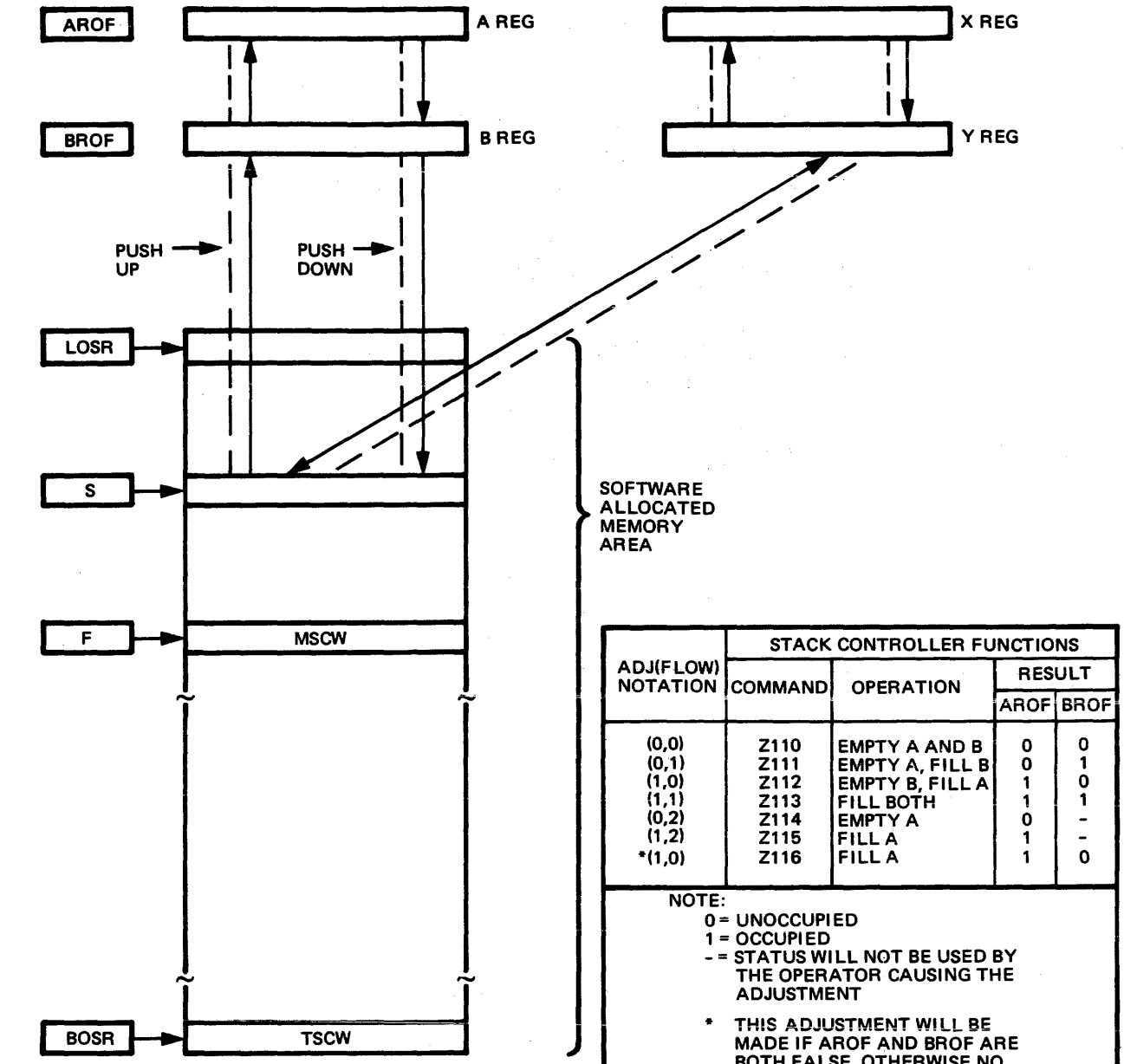


Figure 5-4. Mask and Steering

There are five different types of interrupts detected by the hardware of the interrupt controller. These five types are:

a. Syllable dependent interrupts

b. External interrupts



ADJ(FLOW) NOTATION	STACK CONTROLLER FUNCTIONS			
	COMMAND	OPERATION	RESULT	
			AROF	BROF
(0,0)	Z110	EMPTY A AND B	0	0
(0,1)	Z111	EMPTY A, FILL B	0	1
(1,0)	Z112	EMPTY B, FILL A	1	0
(1,1)	Z113	FILL BOTH	1	1
(0,2)	Z114	EMPTY A	0	-
(1,2)	Z115	FILL A	1	-
*(1,0)	Z116	FILL A	1	0

NOTE:
0 = UNOCCUPIED
1 = OCCUPIED
- = STATUS WILL NOT BE USED BY THE OPERATOR CAUSING THE ADJUSTMENT

* THIS ADJUSTMENT WILL BE MADE IF AROF AND BROF ARE BOTH FALSE, OTHERWISE NO ADJUSTMENT WILL BE MADE.

Figure 5-5. Hardware Stack Adjustment

c. Alarm interrupts

d. General control interrupts

e. Hardware interrupts

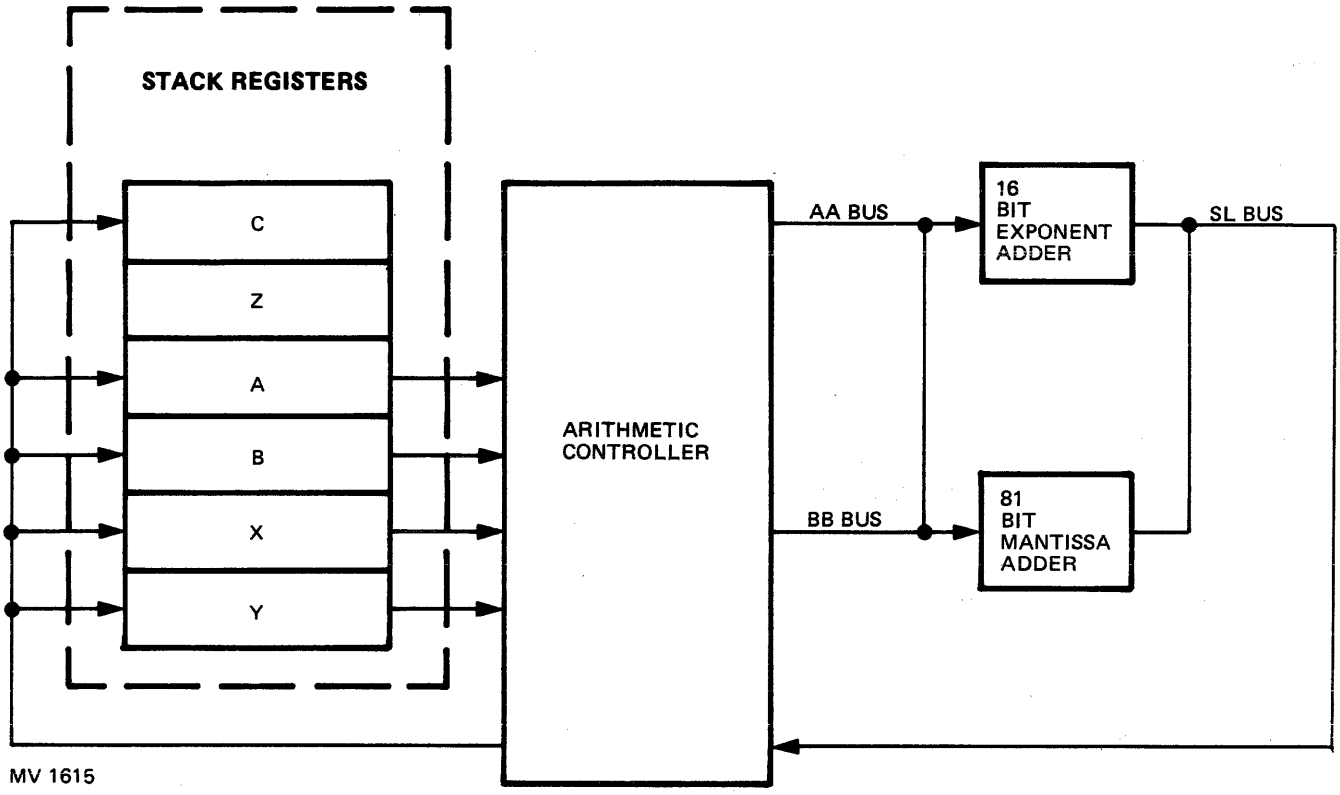


Figure 5-6. Arithmetic Control

The P1 parameter is the portion of an interrupt stack that identifies the type of interrupt that is being processed. There are five different types of interrupts, and a bit in the P1 parameter identifies which of the five types of interrupt is present. The following definitions identify the bits in the P1 parameter that show interrupt type, and system type:

Interrupt Parameter Bits

27	26	25	24	23	22	21	20	
λ								Identifies a B 6800 system
	X							Identifies a hardware interrupt
		X						Identifies an alarm interrupt
			X					Identifies a syllable dependent interrupt
					X			Identifies a general control interrupt
							X	Identifies an external interrupt

The following paragraphs define the five types of interrupts and identify the major causes of the interrupts.

Syllable Dependent Interrupts

Syllable dependent interrupts are sensed by an operator and normally result in a premature termination of the operator under control of the logic for the operator. The operator inserts data for the P1, P2, and P3 parameter into the top of stack registers, and activates the interrupt controller. The values of the PIR and PSR registers are reset to the beginning of the current operator address, and the interrupted operator is restarted, upon a return from the interrupt handling procedure of the MCP.

The syllable dependent interrupts are:

a. Memory protect interrupt

b. Invalid operand interrupt

c. Divide by zero interrupt

d. Exponent overflow interrupt

e. Exponent underflow interrupt

f. Invalid index interrupt

g. Integer overflow interrupt

h. Bottom of stack interrupt

i. Presence bit interrupt

j. Sequence error interrupt

k. Segmented array interrupt

l. Programmed operator interrupt

m. Interval timer interrupt

n. Stack overflow interrupt

o. Confidence error interrupt

NOTE

Although the interval timer interrupt and the stack overflow interrupts are classed here as syllable dependent interrupts, it should be pointed out that these two types of interrupts are not truly syllable dependent. These two interrupts would be more clearly defined as asynchronous interrupts because they do not depend on the operator that is in process at the time that the

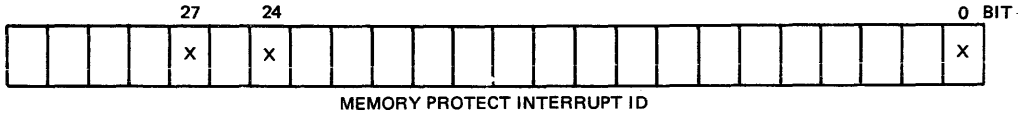
interrupt is raised. However, the handling of these two interrupts, with respect to the formation of the P2 parameter, and the handling of the syllable address, are the same as other syllable dependent interrupts, and are therefore classed as syllable dependent.

Memory Protect Interrupt

This interrupt occurs under the following conditions:

a. A store, overwrite, or read/lock or string transfer operation is attempted using a data descriptor that has the read only bit set (bit 43). The operation is terminated prior to the memory access, leaving the descriptor word in the A register.

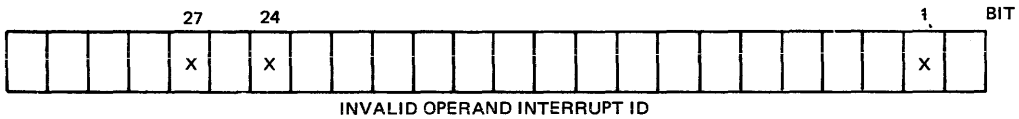
b. A store is attempted into a word in memory that has a tag field representing program code, RCW, MSCW, or segment descriptor. The memory write is aborted when bit 48 is detected in the "flasback" word. The operation is terminated leaving the original addressing word in the A register.



MEMORY PROTECT INTERRUPT ID

Invalid Operand Interrupt

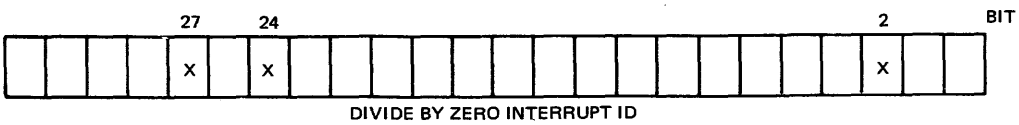
This interrupt occurs when operators attempt to use the wrong types of control words or data. When control words and data are accessed, they are checked to ensure that they meet the necessary requirements of the operator being executed. When the interrupt occurs, the operator is terminated prematurely.



INVALID OPERAND INTERRUPT ID

Divide by Zero Interrupt

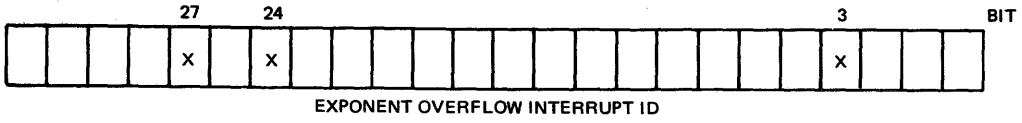
This interrupt results when a division operator is attempted with the divisor equal to zero. This interrupt terminates the operation prematurely, leaves the A register cleared, the interrupt ID in the B register, and PSR and PIR backed up to point to the initiating operator.



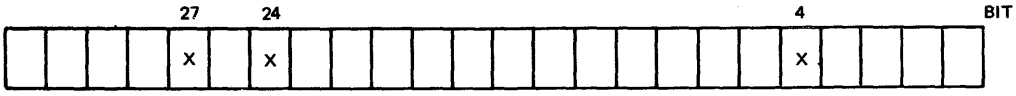
DIVIDE BY ZERO INTERRUPT ID

Exponent Overflow and Underflow Interrupt

These interrupts occur when the capacity of the exponent field is exceeded for either single- or double-precision arithmetic results. The interrupt ID is dependent on the exponent sign, and both interrupts clear the A register.



EXPONENT OVERFLOW INTERRUPT ID

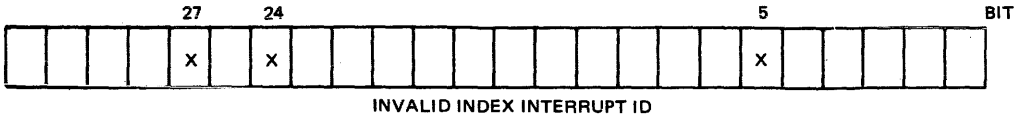


EXPONENT UNDERFLOW INTERRUPT ID

Invalid Index Interrupt

This interrupt is caused by an attempt to index by less than zero or not less than the upper bound (length) in the operations:

		Family
a.	Occurs Index	(A)
b.	Link List Lookup	(B)
c.	Index	(C)
d.	Move Stack	(C)
e.	Display Update	(C)
f.	Dynamic Branch	(C)
g.	Stuffed IRW (pseudo)	(C)
h.	Index and Load Name	(C)
i.	Index and Load Value	(C)



INVALID INDEX INTERRUPT ID

Integer Overflow Interrupt

This interrupt occurs when an attempt is made to integerize operands which have a value greater than maximum integer. In general, the checking is performed before the operand is converted into an integer by reducing the exponent field. The following are some of the operators that may invoke this interrupt.

a. Integer Divide (both single and double precision)

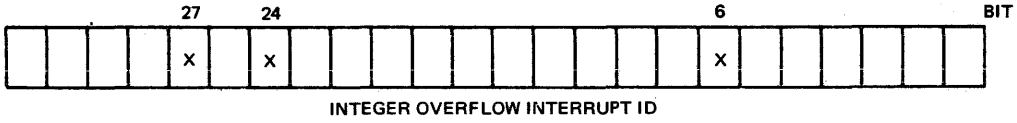
b. Integerize Truncated

c. Integerize Rounded

d. Occurs Index

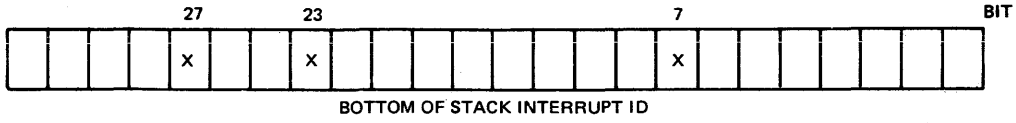
e. Integerize rounded, double precision

If the interrupt is invoked, the operator is terminated.



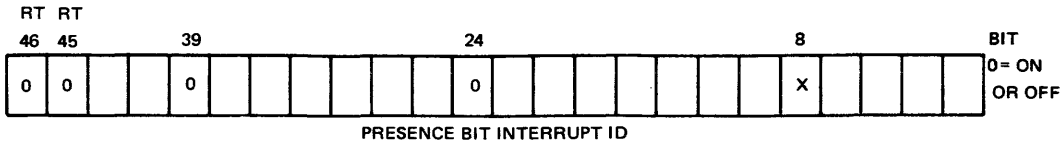
Bottom of Stack Interrupt

This interrupt is used to inform the operating system that a return or exit operator has caused the program stack to be returned to its base. If this condition arises, the operator will terminate with the last accessed RCW (Return Control Word) left in the A register.



Presence Bit Interrupt

This interrupt is used to inform the system that an attempt has been made to access a quantity not present in main memory. All operators that access memory with descriptors have the ability to set this interrupt. Special consideration is given to this type of an interrupt for data or procedure-dependent descriptors.



Special Consideration-Presence Bit Interrupts

There are two classes of presence bit interrupt conditions:

a. Data-Dependent

b. Procedure-Dependent

Each class requires that the PIR and PSR value for the RCW be manipulated differently.

Data-Dependent Presence Bit Interrupt

Data-Dependent Presence Bit. The data-dependent presence bit interrupts are incurred while the processor is seeking data from within its current procedural environment. Recovery is achieved by re-executing the operator upon return from the presence bit interrupt-handling procedure.

The presence bit procedure makes the non-present reference present prior to returning to the interrupted program. The PIR and PSR setting for the current operator are saved in the RCW for data-dependent presence-bit interrupts.

Procedure-Dependent Presence Bit Interrupt

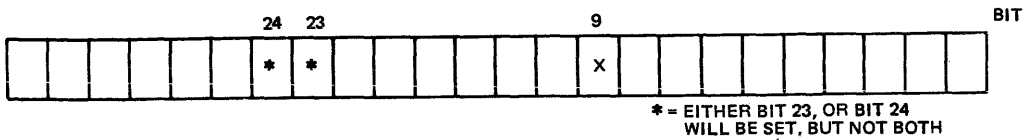
Procedure-Dependent Present Bit. The procedure-dependent presence bit interrupts are incurred when the processor attempts to enter a new procedural environment or to return to an old procedure. These interrupts occur during display update and when the processor is trying to access a non-present segment descriptor. Recovery is achieved by the exit operator mechanism after the presence bit procedure has made the referenced area present. The processor has not yet fetched the first operator of the new procedure when this presence bit interrupt occurs; therefore, the PIR and PSR settings from the PCW or RCW, depending on whether an entry or exit was being performed, are saved when fabricating the RCW upon entry into the presence bit interrupt procedure.

Program Restart

Program Restart. In order to restart some operators after a presence bit interrupt, it is necessary for the presence bit procedure to return either an IRW or Data Descriptor. The "RT-bit" in the presence bit ID (P1) indicates to the presence bit procedure whether to perform an exit or return operator when returning to the interrupt program. The "RT-bit" is manipulated by the hardware prior to honoring the presence bit interrupt. Figure 5-7 (Presence Bit Interrupt Table) illustrates the (PSR and PIR), exit/return and "RT-bit" relationship to the various presence bit interrupt conditions.

Sequence Error Interrupt

This interrupt is used to inform the system that while attempting to access a Mark Stack Control Word (MSCW), a word with a tag field value of three was not found. This error implies that the stack linkage or stack history (of the stack that was being accessed) is in error. A sequence error may occur at different places in an operator sequence, and may occur before, or after the time in the sequence where PIR, PSR, PBR and PDR are adjusted. If the sequence error occurs after the required adjustment has been made, then bit 23 of the interrupt parameter will be set to indicate a class two syllable dependent interrupt. If the sequence error occurs prior to the adjustment then bit 24 of the interrupt parameter will be set to indicate a class one syllable dependent interrupt. The interrupt parameter for a sequence error is as follows:



SEQUENCE ERROR INTERRUPT PARAMETER

PRESENCE BIT INTERRUPT CONDITION				RT BIT (3) (BIT 46)	RETURNING OPERATOR	PIR, PSR NEW RCW	SOFTWARE FUNCTION
DATA DEPENDENT	STACK VECTOR STACK VECTOR D.D. DURING DATA REFERENCE	(1) IRW (STUFFED)	INT. I.D.	0	EXIT	S_n (4)	MAKE STACK OR STACK VECTOR PRESENT.
		(2) IRW	INT.	1	RETURN	S_n (4)	
	DATA DESCRIPTOR DURING DATA REFERENCE	(1) D.D. (COPY)	INT. I.D.	0	EXIT	S_n (4)	SEARCH STACK FOR COPIES OF NOT PRESENT D.D., MAKE MOM AND COPIES PRESENT, RETURN D.D. WHERE NOTED.
		(2) D.D. (COPY)	INT. I.D.	1	RETURN	S_n (4)	
PROCEDURE DEPENDENT	STACK VECTOR STACK VECTOR D.D. DURING DISPLAY UPDATE	– D.D. (COPY)	INT. I.D.	0	EXIT	FROM RCW/PCW	
	SEGMENT DESCRIPTOR	– S.D. (COPY)	INT. I.D.	0	EXIT	FROM RCW/PCW	LOCATE S.D. (MOM) VIA COPY IN P_2, AD FIELD OF COPY POINTS TO MOM

(1) VALUE CALL OR ENTER
(2) ALL OPERATORS EXCEPT VALUE CALL, ENTER, OR MOVE STACK
(3) RT BIT IS PACKED IN THE INT. I.D. (P_1)
(4) S_n INDICATES THE PIR AND PSR POINT TO CURRENT OPERATOR SYLLABLE
(5) MOVE STACK OPERATORS

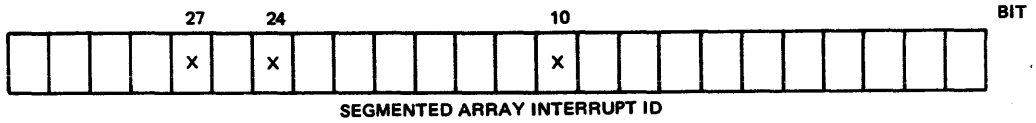
MV 1616

Figure 5-7. Presence Bit Interrupt

Segmented Array Interrupt

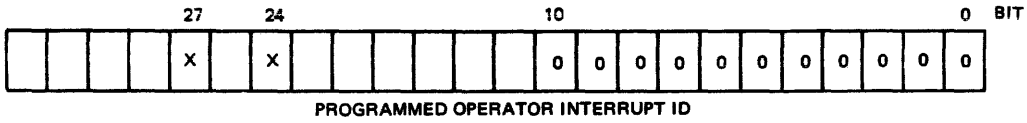
This interrupt is used by the string operators as an upper limit boundary detection. Arrays in main memory may be segmented into groups of 256 words each, bounded on both ends by memory link words. Each word read from memory during string operator executions is checked for the presence of bit 48 (memory protect). If the bit is on, the segmented-array interrupt is set. String operator interrupts leave a special parameter in the A register. This parameter indicates how

many words in the stack, below the parameter, will be needed to restart the operation after the new segment of data has been brought to main memory.



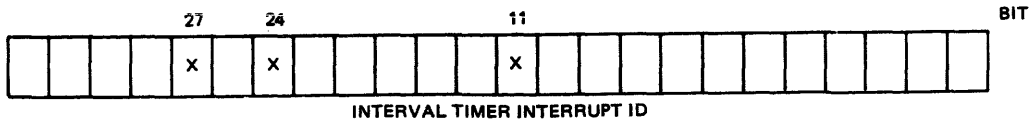
Programed Operator Interrupt

This interrupt is used for the detection of an invalid operator code. Primary code FF is detected and causes this interrupt. An invalid code not detectable will result in a loop timer interrupt. The programed operator interrupt is used as a communicate operator to the system.



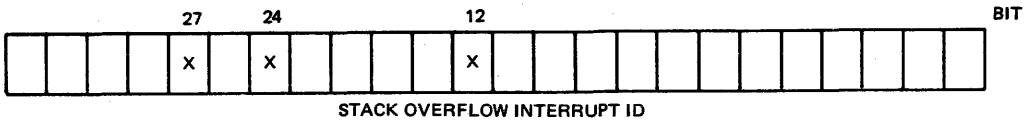
Interval Timer Interrupt

The interval timer interrupt occurs when the interval timer times out. This timer is used by the MCP for time slicing. The SINT operator is used to establish an initial value in the interval timer register. The data processor then proceeds to count 512 micro-second intervals until the number of intervals that have occurred since the SINT operator was executed is equal to the value that was set in the interval timer register. At this point an interrupt is generated that forces the data processor to enter the interrupt handling procedure. The presence of the interval timer P1 parameter indicates to the MCP that it is time to perform some other time sliced procedure.



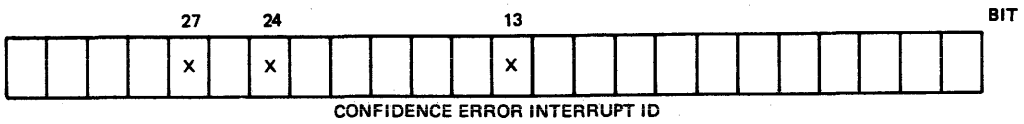
Stack Overflow Interrupt

The stack overflow interrupt occurs when the S register in the IC memory is equal to the LOSR register. This interrupt occurs because the procedure that is being executed has attempted to utilize more word space in the stack than was originally allocated for the memory portion of the stack.



Confidence Error Interrupt

The confidence error interrupt occurs when the confidence test routine is being executed by the data processor, and an error is detected as a result of a test failure. The confidence test is automatically initiated when the data processor is not performing other software procedures. A failure in the confidence test is placed in the system/SUMLOG for maintenance analysis. The test places data about the nature of the error in the interrupt parameters and the MCP upon recognizing a confidence error causes the data to be written in the SUMLOG.



OPERATOR INDEPENDENT INTERRUPTS

These interrupts are induced by conditions outside the operator or processor logic. They are divided into two groups, external interrupts and alarm interrupts.

EXTERNAL INTERRUPTS

These interrupt conditions are anticipated and inform the system of some change in the external environment. They normally result in a momentary interruption of a program process which will be continued after handling or recording of the interrupt condition. The program sequence controller senses the interrupt condition, inhibits activation of the next operator. The interrupt controller then processes the interrupt. PIR and PSR fields of the RCW address the next operator syllable so that the program will be restarted with the execution of the next syllable upon continuation. The external interrupts are as follows:

a. I/O finish interrupt

b. Data Communications interrupt

c. General Control Adapter

d. Change of Peripheral Status interrupt

e. Scratch Pad Parity interrupt

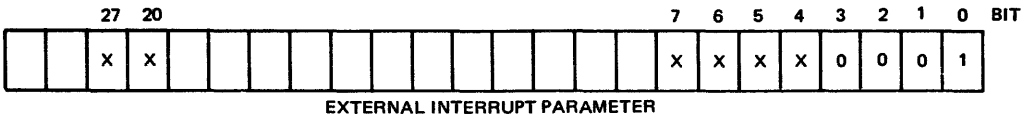
I/O Finish, Data Communications, and Status Change Interrupts

I/O finish, data communications and status change interrupts are handled by the interrupt controller as follows:

1. A hardware branch is made to the multiplexer interrupt routine, in the micro-logic module. The micro-module contains the necessary logic to place the correct type of interrupt parameters in the interrupt stack (see Interrupt Parameter Words, in section two).

   After the micro-module has assembled the proper interrupt stack parameters in the data processor top of stack registers, the interrupt controller resumes the automatic interrupt handling process.

2. The normal operation of entry to the MCP interrupt handling procedure is then executed.



NOTE

Bits 3:4=0001
Bits 7:4 identify type of interrupt.
1001=I/O finished
0001=DCP #1
0010=DCP #2
0011=DCP #3
0100=DCP #4
0110=BIC #1
0111=BIC #2
1111=change of status
1000=scratch pad parity error

Alarm Interrupts

These interrupt conditions are not anticipated and inform the system of some detrimental change in environment. They normally result from either a programing error or hardware failure. The alarm interrupt conditions are recognized upon occurrence by the interrupt controller. The interrupt controller assumes control of the machine, clears the activated operator family, and marks the TOS registers full. In either case (programming error or hardware failure) the current operator is terminated prematurely. The alarm interrupts are:

a. Loop timer interrupt

b. Memory address parity interrupt

c. Scan bus parity interrupt

d. Invalid address-local interrupt

e. Stack underflow interrupt

f. Invalid program word interrupt

g. Memory address residue interrupt

h. Read data multiple error interrupt

i. Invalid address-global interrupt

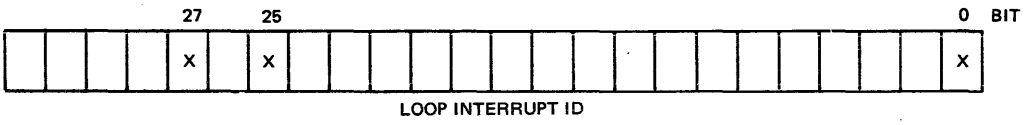
j. Global memory not ready interrupt

k. Scan-in information error interrupt

l. Scan-out error interrupt

Loop Interrupts

This interrupt is invoked if the Processor hardware fails to provide a SECL (Syllable execute complete level) at least every 2 seconds. This could occur if an attempt is made to execute an invalid operator. If the interrupt occurs, the ID remains in the B register, the A register is cleared and PIR is backed up.

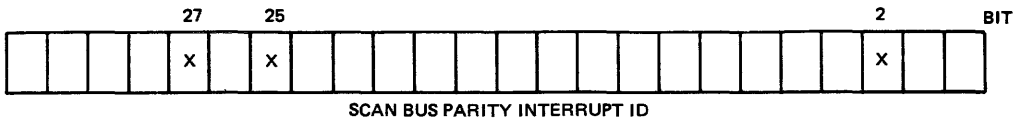


Memory Address Interrupts

This interrupt is invoked if the Memory Controller detects an even number of address and control bits being transmitted between the Processor and Memory. Should the interrupt occur, the ID is left in the B register, the A register is cleared and PIR is backed up.

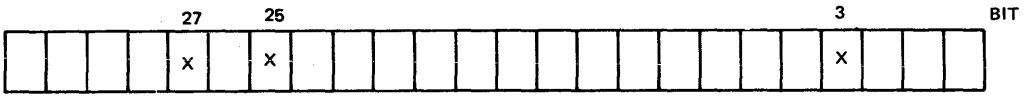
Scan Bus Parity Interrupts

This interrupt is the same as Memory Address Parity above, except that it is used for transfer of information on the scan bus interface.



Invalid Address-Local Interrupts

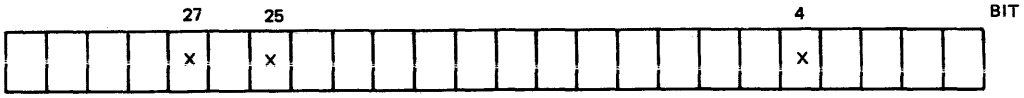
This interrupt is set by the Memory Controller when it fails to obtain an acknowledgement to a local memory request within eight clock periods. This indicates that an attempt has been made to access a non-existent memory module. The memory controller initiates the interrupt and the interrupt controller leaves the ID in the B register with the A register clear and PIR backed up.



INVALID ADDRESS LOCAL INTERRUPT ID

Stack Underflow Interrupts

This interrupt is invoked if the stack controller detects an attempt to move the S register to an address less than F during stack adjustment. If this interrupt occurs, the ID remains in the B register, the A register is cleared and PIR backed up.



STACK UNDERFLOW INTERRUPT ID

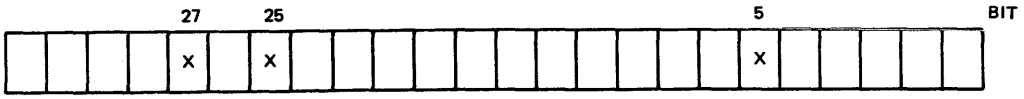
Invalid Program Word Interrupts

This interrupt is invoked if one of the following conditions is encountered:

a. A word with a tag not equal to 3 is placed in the P register for execution (except in Table edit mode).

b. The variant operator syllable (95) is followed by another variant operator syllable (95).

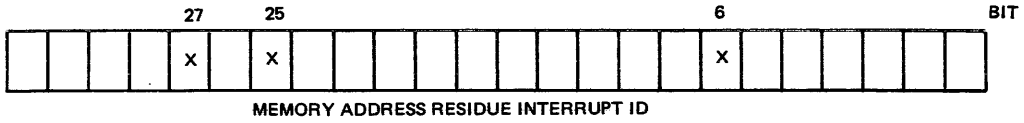
c. The processor is in edit mode and a family strobe is emitted for another operator family. Should the interrupt occur, the ID is left in the B register, the A register is cleared and PIR is backed up.



INVALID PROGRAM WORD INTERRUPT ID

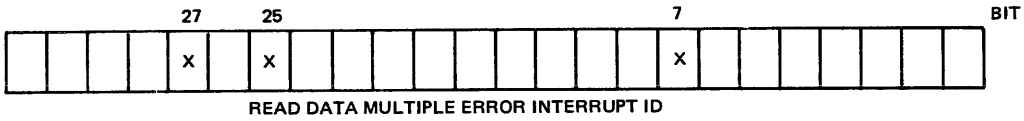
Memory Address Residue Interrupts

This interrupt is set when the memory controller detects an error in the MAR, or LAR address registers. Residue checking is a method for detecting abnormalities in the address adder and/or the IC memory registers. Any activity of the address adder that results in the setting of a residue interrupt prevents a memory cycle from occuring.



Read Data Multiple Error Interrupts

This interrupt is set when the memory controller detects more than a single bit in error during a memory read operation. Multiple bits in error are not correctable, and thus when such errors are detected the memory controller causes an alarm interrupt to occur.

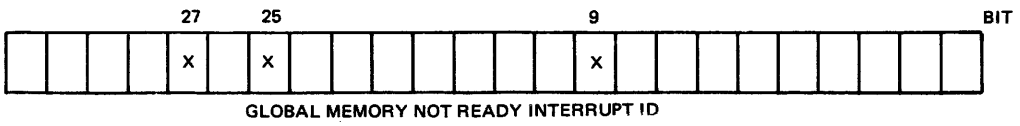


Invalid Address-Global Interrupts

This interrupt is set in the same manner as is the invalid address-local alarm interrupt, except that the invalid address is for a global memory address instead of for a local address.

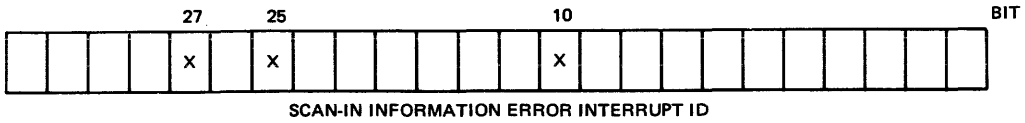
Global Memory Not Ready Interrupts

This interrupt is set when a memory access is initiated on a memory address in global memory, and the global memory does not respond properly to the memory controller.



Scan-In Information Error Interrupts

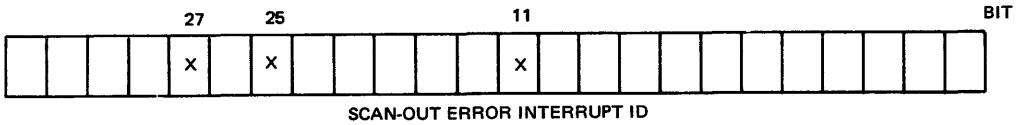
This interrupt is set when an external sub-system attempts to communicate with the CPU cabinet through the scan bus interface, and the memory control detects an even number of data bits on the scan bus information lines.



SCAN-IN INFORMATION ERROR INTERRUPT ID

Scan-Out Error Interrupts

This interrupt is the same as the scan-in information error interrupt except that the detection of an error on the scan bus information lines was made by one of the sub-systems that are connected to the scan bus. The direction of data flow was from the B 6800 system to the sub-system interface. The sub-system that detected the error responded by making the scan bus control level STEX a true (high) signal. The CPU initiates the alarm interrupt when this level is a true signal.



SCAN-OUT ERROR INTERRUPT ID

General Control Interrupts

General control interrupts are used to cause information to be inserted into the System/SUMLOG, about correctable error conditions during normal system operations. This type of error does not result in an operator or a procedure failure, but the implication of the error is that further deterioration of the hardware may lead to an operator or procedure failure. A subsequent analysis of the information in the System/SUMLOG may be used to identify the nature of the error condition. By definition such errors are intermittant and/or random, and usually cannot be duplicated for troubleshooting and maintenance purposes. However, in some cases, knowledge about the frequency of occurance, and simularity of operating conditions may lead to the solution of otherwise non-solvable problems. The purpose of the general control interrupt is to produce the data upon which such analysis can be made.

There are four different kinds of general control interrupts as follow:

a. Read data single error interrupt.

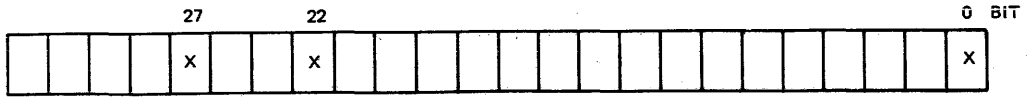
b. Read data retry interrupt.

c. Read data check bit interrupt.

d. Address retry interrupt.

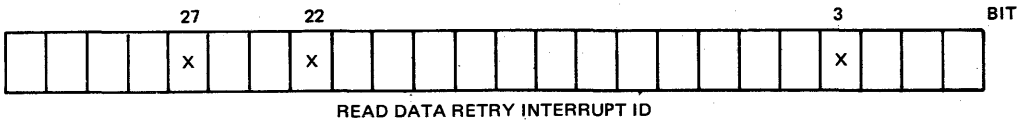
Read Data Single Error Interrupts

This interrupt is set when the memory interface error correction circuit detects a single bit in error during a read from memory operation. The single bit in error is corrected, and the procedure being executed is not aware that a bit failed on the memory bus. The memory controller is aware that a single bit failed and causes the read data single error interrupt to occur.



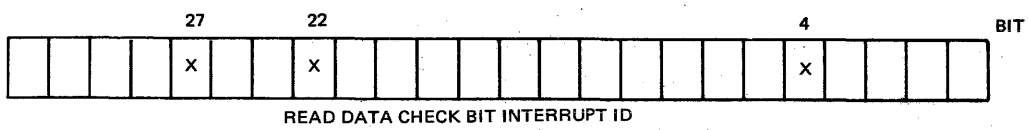
Read Data Retry Interrupts

The memory controller contains logic that can cause a read strobe to be regenerated if a parity error is detected on the read data memory interface bus. This retry of a memory read strobe can prevent the occurrence of a memory interface parity error if the cause of the parity error is a problem in the data signal on the interface bus. A general control interrupt is initiated by the memory controller when a memory retry is performed. The purpose of this interrupt is to cause the System/SUMLOG to record information about the retry.



Read Data Check Bit Interrupts

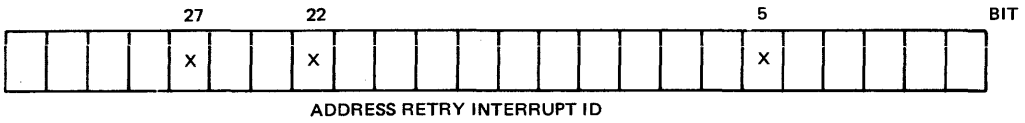
The memory controller error detection/correction circuit receives 60 bits of read data for a memory read operation. These 60 bits include 52 bits of data, and eight check bits. The check bits are used to detect errors in the 52 data bits. The correction circuit corrects any single bit in error in the read data (see the subsection titled READ DATA SINGLE ERROR). Multiple data bit errors are detectable, but are not correctable (see the subsection titled READ DATA MULTIPLE ERROR). A check bit in error is detectable, and, providing that a data bit error does not occur at the same time as the check bit error, does not result in an alarm interrupt condition. The memory controller reports check bit errors, and the resulting interrupt causes an entry in the System/SUMLOG to be written which identifies the nature of the error.



Address Retry Interrupts

The memory controller holds a memory address for an access on the memory until the memory reports back to the controller that the address received was correct, or that the address was incorrect. If the memory reports back that the address was incorrect the controller will strobe the address to the memory a second time, for a retry. If the retry of the address is correct the memory access is completed for the requesting unit. If the retry of the address was incorrect an invalid address error is reported to the memory control by the exchange (see alarm type interrupts).

The memory exchange reports retries to the memory controller, and the interrupt that is generated as a result of this report causes the information about the retry to be written in the System/SUMLOG.



Hardware Interrupts

Hardware interrupts are abort type interrupts that cause operator and procedure errors. These errors are related to a hardware device or circuit that can be identified for maintenance analysis. The purpose of the hardware interrupts is to identify the device and/or cause SYSTEM/SUMLOG entries of other significant data to be written into the System/ SUMLOG, for a subsequent analysis. There are five hardware interrupts as follows:

a. Prom card parity interrupt

b. Ram card parity interrupt

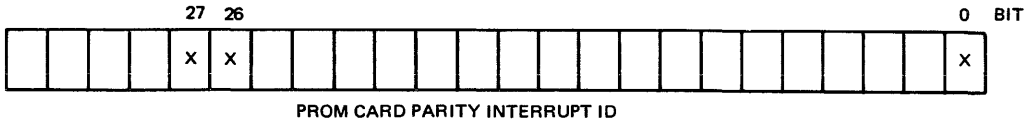
c. Bus residue interrupt

d. Adder residue interrupt

e. Compare residue interrupt

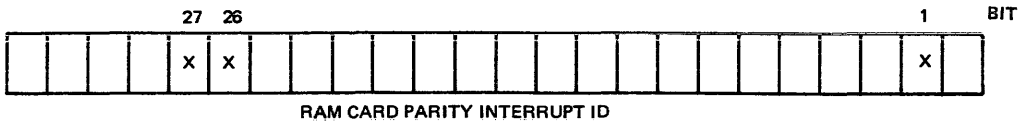
PROM Card Parity Interrupts

The data processor contains many PROM chip devices that are used to hold pre-selected micro codes and addresses. Each time that one of the PROM devices are addressed, the output code is tested for parity. If a PROM parity error is detected, a hardware interrupt is initiated, and the interrupt parameters contain the address of the PROM device that caused the error. Other information in the interrupt parameters is defined in table 2-4 (section 2).



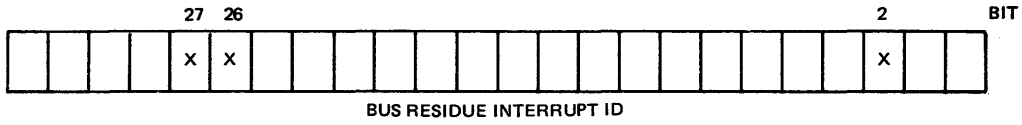
RAM Card Parity Interrupts

The multiplexor contains RAM chip devices in the multiplexor buffer. Each time that an address in the buffer is accessed a parity check of the contents of the buffer address is made. If a parity error in the buffer is detected, a RAM parity error is reported to the data processor, and this error initiates the interrupt controller. The address of the RAM device that caused the error is transferred to the data processor interrupt controller and is stored in the interrupt parameters (see table 2-4 in section 2).



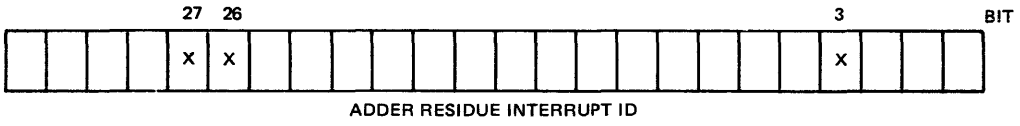
Bus Residue Interrupts

The residue generator cards in the data processor tests the residue bits from the Z8 and Z9 busses. If an error is detected in the residue bits from these two busses a bus residue interrupt is initiated, and the residue bits are passed to the interrupt controller for inclusion in the interrupt parameters (see table 2-4 in section 2).



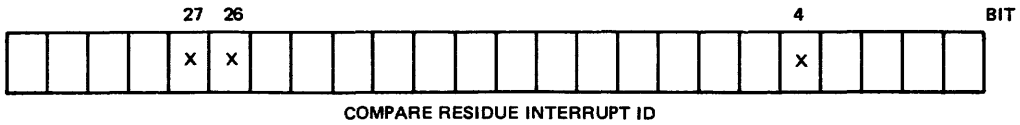
Adder Residue Interrupts

Each time that the residue adder sums two addresses (from the Z8 and Z9 busses) the bits in the two residue values are tested. An error causes the interrupt controller to be initiated, and the residue bits are passed to the interrupt controller for inclusion in the interrupt parameters (see table 2-4 in section 2).



Compare Residue Interrupts

Each time that the address adder sums two address values a new residue value is generated as a result of the sum of the two addresses. At the same time that the address adder is summing the two addresses, the residue adder sums the residue bits from the two addresses. A comparator circuit compares the two sums (one sum from the residue adder and the other sum from the address adder). If the two sums are not identical, there is a high probability that an IC memory circuit is at fault. The interrupt controller senses a compare residue error, and passes the value of the residue to the interrupt handling procedure in the interrupt parameters (see table 2-4 in section 2).



STRING OPERATORS

String operators control the character formatting capability of the B 6800 system while the system is operating in primary mode. The same string operators may also be performed as edit operators while the system is operating in edit mode. The string operators are comprised of the normal mode operators in the F, G, and H families, which are grouped in a "super-family", and designated as family U. Family U operators share a common "T" register (operator code register), a common logical sequence counter, and a common group of logical flip flops.

The most significant advantage from collecting all string operators into a single super-family is that the common logical functions that all string operators share are not duplicated in each family controller. For instance, all string operators require a method for accessing local memory, and for addressing the characters of data within a memory word. A typical string operator must be capable of addressing a number of different words in memory, in order to perform an editing operation on a string of data characters. Moreover, once the editing has been performed, the word must be stored in memory, so that the same editing can be performed on other words of data. The logic circuits and operator functions that are required to perform this type of operation are common, and are thus collected into the single super-family U in the B 6800 system.

MEMORY CONTROLLER

The memory controller in the CPU (refer to figure 5-2) services requests for access to memory resources of the system from the data processor, the look ahead logic, and the multiplexor. These three modules are all located within the CPU cabinet, and share a common path to/from memory. Internal logic circuits of the memory controller establish when each of these three modules has priority for accessing system memory resources.

When the multiplexor is processing an IO operation, and a need for a burst cycle exists, the multiplexor has first priority for a memory access request. This condition causes the data processor to suspend its operation while the multiplexor obtains access to memory. The data processor will suspend its operation until the multiplexor completes its memory access. At the conclusion of the multiplexor memory access operation the data processor will continue its operations at the place where the suspension occured.

The order of priority in accessing memory is multiplexor, processor, and look ahead logic, in that order.

The memory controller logic has the capability to store two requests for access to memory. The storing of access requests consists of remembering what requests were received over the Z12 memory control bus. The memory controller examines the contents of the two requests to determine which request has the higher priority for the next access to memory.

The logic mechanism used by the memory controller to remember what memory requestor units require an access to memory consists of two request registers that are located in the channel A input logic to the memory control. When a request for a memory access is transmitted to the memory control, the request (bits D:14 on the Z12 bus) is stored in the RQT register (13:14). Each time that a memory request is to be processed for the CPU cabinet, the memory controller will examine both the RQT, and the RQR registers, to determine which of two possible requests for access to memory has the higher priority. As one of the two possible memory requests are performed, the stored request information in the RQT register (or alternatively the RQR register) is reset to binary zeroes. This removes a request that is presently being executed from further contention for an access to memory, and frees the register that was reset to accept a new access request.

The memory controller also monitors all memory and scan bus requests for errors. If an error condition is detected during a memory bus or scan bus operation, the memory controller will cause an interrupt to be present in the data

processor interrupt controller. The memory controller passes parameters that describe the type of interrupt that occured to the interrupt controller. The interrupt handling procedure of the MCP will cause the interrupt parameters from the memory controller to be written in the SYSTEM/SUMLOG, thus preserving a record of memory and/or scan bus errors. The interrupt handling procedure for logging memory errors is also used for memory accesses that originate outside of the CPU (in a subsystem memory interface, from an external subsystem). The memory controller detects memory errors that originate in the external subsystem interface to memory, and initiates the interrupt controller to log all such errors.

CONTROL STATE/NORMAL STATE

A B 6800 data processor has the ability to perform in either normal or control state. In control state, all external interrupts are inhibited and a few privileged operators are enabled. The Inhibit Interrupt Flip Flop (IIHF) must be set for processing to occur in control state.

The data processor switches to control state upon entering a procedure via a control state program control word (PCW).

MULTIPLEXOR FUNCTION

The multiplexor function in the B 6800 system represents the collection of system IO functions into a semi-independent functional group. This multiplexor grouping of functions operates at times like a part of family C of the data processor. At other times it operates in an autonomous manner, independently of the data processor. The multiplexor cannot initiate an IO function except upon command of the data processor.

The "semi-independent" characteristic of the multiplexor is achieved by the manner in which the data processor and the multiplexor communicate with each other (see figure 5-8). These two component modules of the CPU cabinet are linked to each other in such a way that they share the Z1 and Z5 busses. Through sharing these two busses the data processor and the multiplexor pass control information and data to each other. In addition to the linkage on the common busses, the multiplexor controls an interrupt signal line which allows the multiplexor to directly and independently invoke the interrupt controller function of the data processor.

Two variant mode operator codes (954A, SCAN-IN operator, and 954B, SCAN-OUT operator) are used by the data processor to cause the multiplexor to perform one of its functions.

DATA PROCESSOR SCAN-IN FUNCTIONS TO THE MULTIPLEXOR

The functions that the multiplexor performs in response to a data processor SCAN-IN operation are defined by the contents of the function code field in the multiplexor function word (see figure 5-9). Each of these functions will be covered in detail later in this section. None of the SCAN-IN operations performed by the multiplexor require an interface to the memory control, because the data generated in the multiplexor is returned to the data processor as a part of the SCAN-IN operation sequence. If the information that is returned to the data processor is to be subsequently stored in memory, the data processor will cause the memory storage to occur.

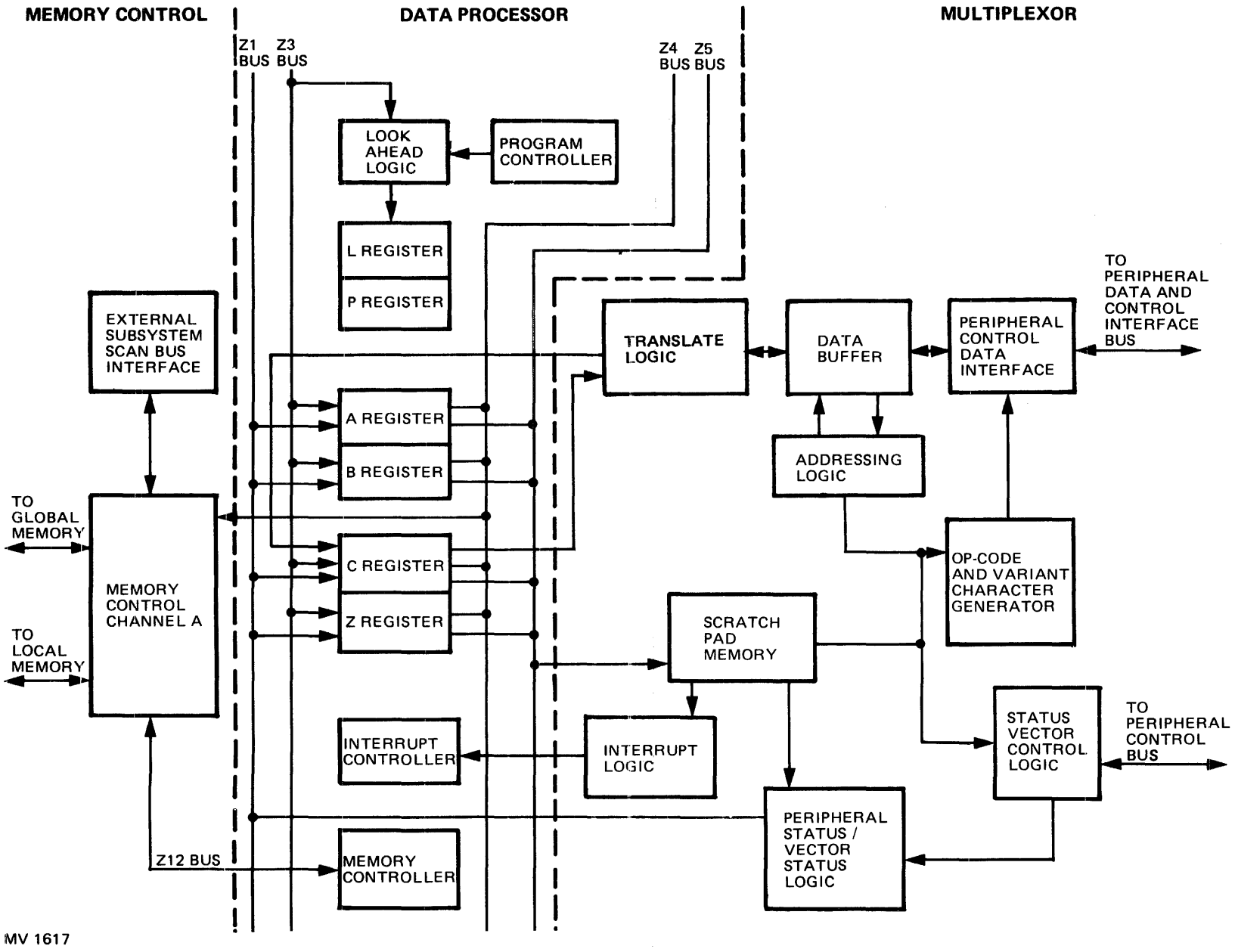
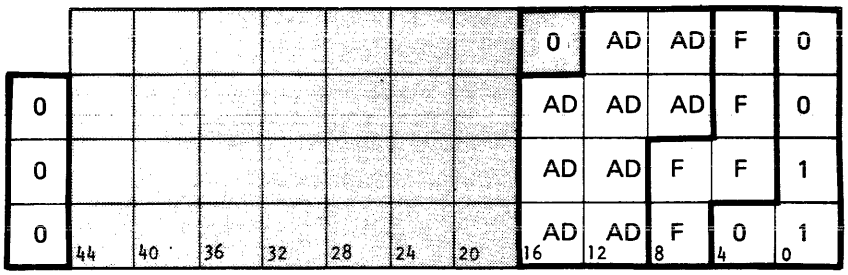


Figure 5-8. B 6800 IO Function Block Diagram



BIT FIELD — MEANING

50:3 = TAG FIELD.
THE TAG FIELD IS ALWAYS EQUAL TO ZERO

47:29 = UNUSED FIELD. THE UNUSED FIELD IS ALWAYS EQUAL TO ZERO

18:9 = THE ADDITIONAL DATA FIELD. THIS FIELD IS USED TO COMMUNICATE ADDITIONAL DATA ABOUT THE OPERATION THAT IS TO BE PERFORMED

9:5 = THE FUNCTION CODE FIELD. THIS FIELD SPECIFIES WHICH OF THE VARIOUS MULTIPLEXOR SCAN-IN FUNCTIONS IS TO BE PERFORMED

CODE		FUNCTION
00000	=	INTERROGATE IO PATH
00001	=	INTERROGATE PERIPHERAL STATUS
00011	=	READ TIME OF DAY REGISTER
00100	=	READ INTERRUPT REGISTER
00110	=	INTERROGATE UNIT TYPE
01000	=	INTERROGATE IO PATH ADDRESS
01010	=	READ PROCESSOR TIME COUNTER
01011	=	READ SCRATCH PAD WORD
01100	=	INTERROGATE IO PATH ADDRESS OVERRIDE
01111	=	READ INTERRUPT LITERAL
10100	=	READ INTERRUPT MASK

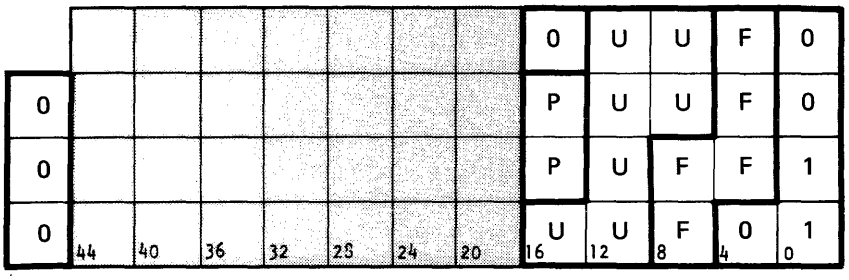
4:5 CODE REQUIRED FOR B 6800 SYSTEM IS 00011 BINARY

MV 1618

Figure 5-9. B 6800 Scan-in Function Word

DATA PROCESSOR SCAN-OUT FUNCTIONS TO THE MULTIPLEXOR

The functions that are performed by the multiplexor in response to a SCAN-OUT command from the data processor are different from responses to a SCAN-IN command. The data processor uses two words to provide a function word and the data that is required to perform the function. The function word (or alternatively the initiate IO word) is passed to the multiplexor in the same way that the SCAN-IN function word was passed. The data word (or alternatively the IO address word) that is required to perform the function is initially in the second word of the stack (the B register).



50:3 = TAG FIELD.
THIS FIELD CONTAINS A TAG VALUE OF ZERO FOR SCAN-OUT OPERATIONS

19:1 = AN UNUSED BIT. REQUIRED TO BE A BINARY ZERO IN THE MULTIPLEXOR FUNCTION WORD

18:2 = THE PATH FIELD. THIS FIELD IS USED TO INDICATE ONE OF FOUR PATHS TO BE USED FOR AN IO OPERATION

16:7 = THE UNIT DESIGNATE FIELD. THIS FIELD IS USED TO DESIGNATE ONE OF 256 PERIPHERAL UNITS TO BE USED FOR AN IO OPERATION

9:5 = THE FUNCTION CODE FIELD. THIS FIELD SPECIFIES WHICH OF THE VARIOUS MULTIPLEXOR SCAN-OUT FUNCTIONS IS TO BE PERFORMED

- 00000 = INTIATE IO DEVICE
- 00011 = SET TIME OF DAY
- 00100 = SET INTERRUPT MASK
- 01110 = SET PSUEDO BUSY
- 01000 = INITIATE IO DEVICE WITH PATH ADDRESS
- 01100 = INITIATE IO DEVICE WITH PATH ADDRESS OVERRIDE

4:5 = CODE REQUIRED FOR B 6800 SYSTEM IS 00011 BINARY

MV 1619

Figure 5-10. B 6800 Scan-Out Function Word

The format of the data word that is present in the B register at the beginning of the SCAN-OUT operation is variable, depending on the type of function that the multiplexor is to perform. Each of the various functions will be discussed in detail later in this section, and the format of the data word that is used will be given with the discussion.

The function codes that are used for all SCAN-OUT function words except the initiate IO type operations are shown in figure 5-10. The function word for the initiate IO type of operations is called the initiate IO word (IIOWD), and this format will be discussed later in this section, as a part of the IO operations topic.

DATA PROCESSOR SCAN-OUT FUNCTIONS TO EXTERNAL SUBSYSTEMS

The data processor also uses the SCAN-IN/OUT operators to communicate with the subsystems that may be attached to the B 6800 system (the data communications processor and/or the bus interface control, which includes the reader sorter subsystem). The scan function word and scan data word (IOAD) for external subsystems scan bus operations are explained in sections 12, and 13 of this manual.

An external scan bus operation in the B 6800 system uses the Z4 bus, and is thus similar to a memory operation. The IOAD word that is present in the top of stack registers is routed via the Z4 bus to the memory control. The scan-in (SCNI), and scan-out (SCNO) operators cause bit C of the Z12 bus to be a true level when an external scan bus operation is performed. Bit C of the Z12 bus is used to identify an external scan bus operation in the memory control. When bit C of the Z12 bus is true, the control links the Z4 bus to the external scan bus, instead of to the local or global memory interface buses.

Figure 5-8 shows that the path from the data processor to the external subsystem interface scan bus is through channel A of the memory control (through the Z3 and Z4 busses).

MULTIPLEXOR SCAN-IN FUNCTIONS

The B 6800 system multiplexor responds to eleven different SCAN-IN function words. The functions are defined by the value of the function code, as shown in figure 5-9. The following paragraphs will define the specific information that is passed to the multiplexor during the execution of a SCAN-IN operation. They also will define the information that is returned to the data processor as a result of the SCAN-IN operation.

Interrogate Peripheral Status Multiplexor Function

When a SCAN-IN operator passes a function word that contains the interrogate peripheral status function code, the multiplexor responds by returning a peripheral status word to the data processor. The function word and the "returned" word are shown in figure 5-11.

The status word that is returned to the data processor represents the status vector bits from 32 peripheral devices, out of the 256 peripheral devices that may be operated by the multiplexor. The 256 peripheral units are arranged into groups of 32 units, or eight groups. Each group is numbered, with peripheral device number zero, through device number 31 reported in status vector word number zero. The ninth status vector word (word number eight) is used for those peripheral units that require system action as a result of the status change in the peripheral device, such as system console displays. In the ninth status word the 32 bits do not represent 32 consecutive unit numbers, but rather those units that require system response, in ascending order, according to all such units in the system.

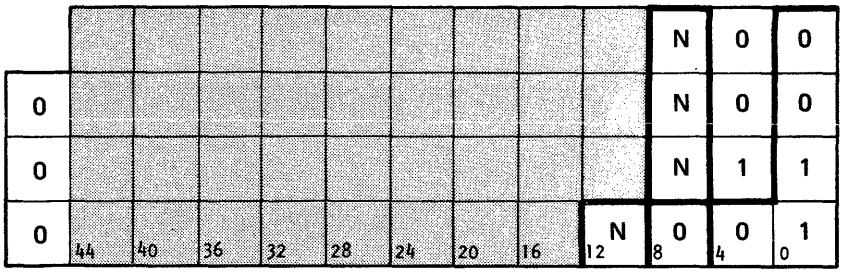
The status vector field in the function word defines which group of 32 units for which the status word is to be returned.

Bit zero in the returned word is used to indicate that the word is present.

Interrogate IO Path Multiplexor Function

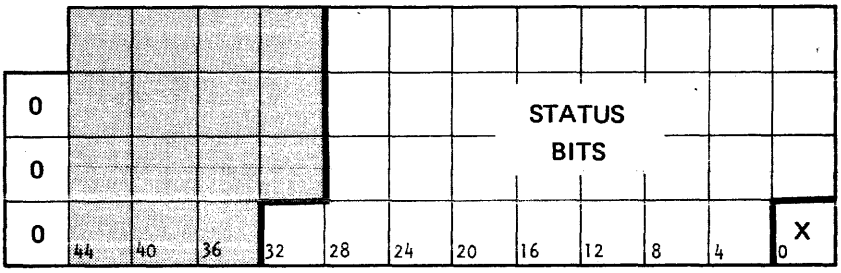
When a SCAN-IN operator passes a function word that contains the interrogate IO path function code, the multiplexor responds by returning a word to the data processor. The "returned" word identifies whether or not a path is available to the IO device. The function word, which is transmitted to the multiplexor as a result of the data processor scan-in operation, is shown in figure 5-12. The Unit number field, of the function word contains the binary unit number of one of the 256 IO devices that may be connected to the system.

FUNCTION WORD



N = NUMBER OF STATUS VECTOR WORD TO BE RETURNED

RETURNED WORD



[32:32] STATUS VECTOR BITS
X=0 STATUS WORD NOT PRESENT
X=1 STATUS WORD PRESENT

MV 1620

Figure 5-11. Interrogate Peripheral Status

Figure 5-12 also shows the format of the returned word, which is the answer from the multiplexor to the data processor as to whether or not a path is available to the particular IO device. The returned word is placed in the B register of the data processor, the B register is marked to contain valid data (BROF is set), and the A Register is marked not valid (AROF is reset).

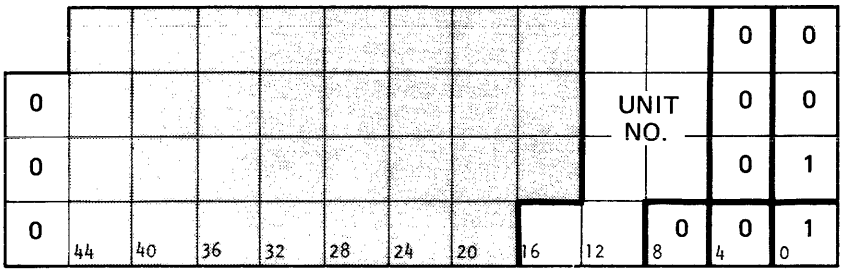
A path is available to an IO device if:

a. The IO control is in remote

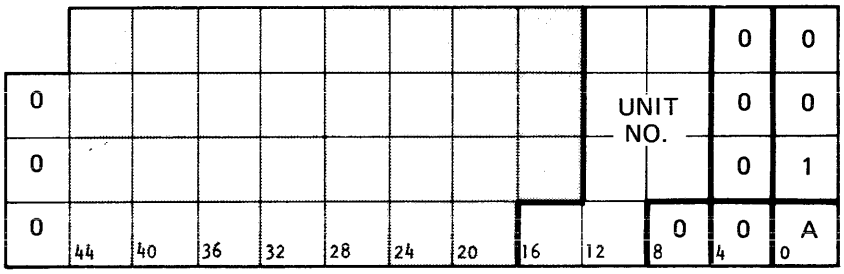
b. The IO control is at sequence count zero

c. The multiplexor channel Psuedo-busy flip-flop is reset

FUNCTION WORD



RETURNED WORD



A=1 PATH AVAILABLE
A=0 NO PATH AVAILABLE

MV 1621

Figure 5-12. Interrogate IO Path

d. The multiplexor IC memory channel does not contain a result descriptor from a previous operation of the IO device

e. The multiplexor will not exceed the pre-determined traffic counter value due to initiating the particular IO device for which path information is requested (high-speed IO devices only)

An available path is not dependent upon the state of an external exchange device, through which the IO control communicates with the peripheral device. If two IO controls have the capability to communicate with the particular device for which path information is required, and one of the two paths is available, the multiplexor will report that a path is available.

Read Time of Day Multiplexor Function

When a SCAN-IN operator passes a function word that contains the read time of day function code, the multiplexor responds by returning a word that contains the binary value of the time of day register. The "returned" word contains 36 bits of time of day information. The value of bit zero in the returned word is 2.4 microseconds.

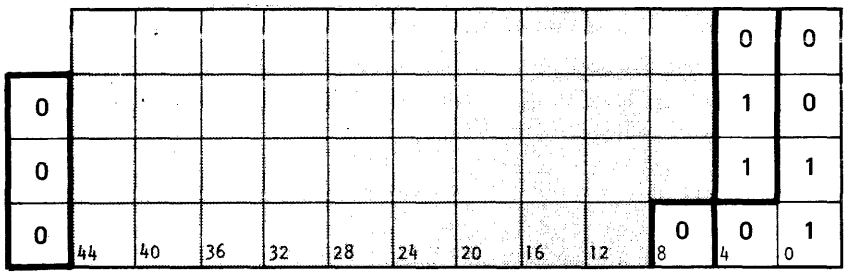
Figure 5-13 shows the format of the SCAN-IN read time of day function word, and also the format of the returned word.

Read Interrupt Register Multiplexor Function

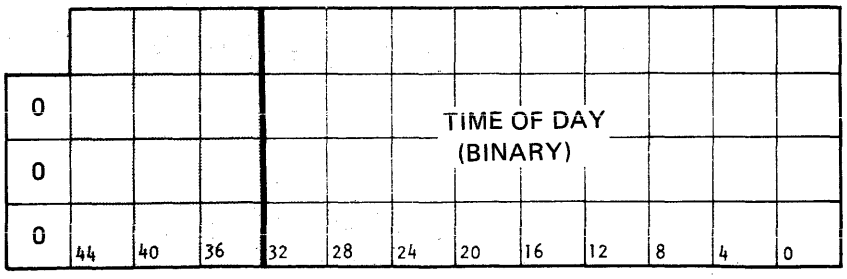
When a SCAN-IN operator passes a function word that contains the read interrupt function code, the multiplexor responds by returning a word that contains the value of the interrupt register. The "returned" word contains ten bits, and each bit represents a different interrupt. The returned value is not treated as a binary value, but rather as a group of individual values. The meaning of each bit in the returned word is as follows:

Bit	Meaning
0	A bit has been set in the status change vector word.
1	The number one data communications processor has caused its interrupt HEYU line to be a true level.
2	This bit is the same as bit number 1, but originates from data communications processor number two.

FUNCTION WORD



RETURNED WORD



MV 1622

Figure 5-13. Read Time of Day

Bit	Meaning
3	This bit is the same as bit number one, but originates from data communications processor number three.
4	This bit is the same as bit number one, but originates from data communications processor number four.
5	Not used.
6	This bit is the same as bit number one, but originates from bus interface control number one.
7	This bit is the same as bit number one, but originates from bus interface control number two.
8	Multiplexor error.
9	This bit indicates that one of the IO devices which are connected to the multiplexor through the peripheral control bus has completed an operation. This bit implies that at least one result descriptor describing a particular IO operation is presently located in the corresponding channel of scratch pad memory (there may be more than one result descriptor present).

Figure 5-14 shows the formats of the function word, and the returned word.

Interrogate Unit Type Multiplexor Function

When a SCAN-IN operator passes a function word that contains the interrogate unit type function code, the multiplexor responds by returning a word that contains a unit type code field. The "returned" word type field contains six bits that identify the type of peripheral device that is assigned to the unit number specified in the function word.

Figure 5-15 shows the format of the function word that is passed to the multiplexor, and also the format of the returned word. The multiplexor field that is part of the returned word, is 010 (bit 46 on) for the B 6800 multiplexor. The various unit type codes, for the corresponding unit types are shown represented in hexadecimal value, as they appear in the data processor top of stack register.

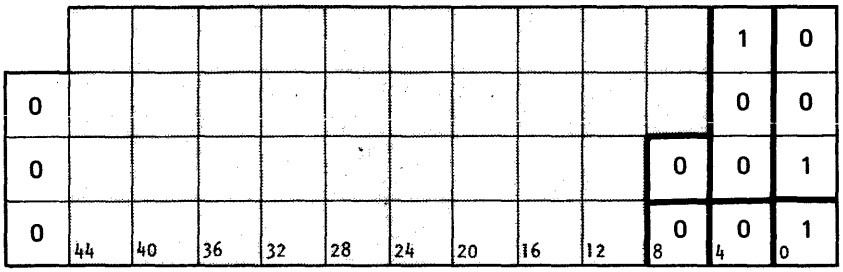
Interrogate IO Path Address Multiplexor Function

When a SCAN-IN operator passes a function word that contains the interrogate IO path address function code, the multiplexor responds by returning a word to the data processor. The "returned" word indicates whether or not a particular path to a peripheral device is available. This multiplexor function is the same as the interrogate IO path function except that where the interrogate IO path function is not specific about which path is to be interrogated, the interrogate IO path address function is specific. The path field of the function word defines which specific path is to be interrogated, and the path field of the "returned" word indicates whether or not the specific path is available.

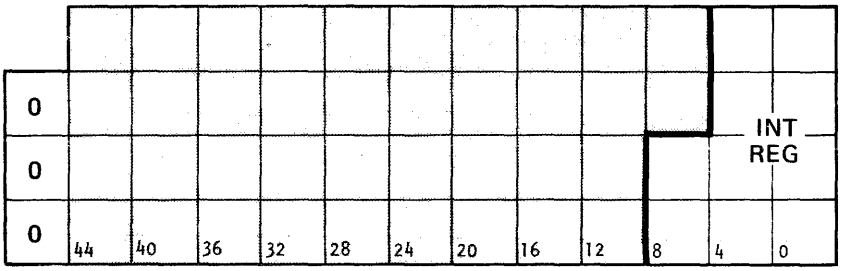
Figure 5-16 shows the format of the function word, and also the format of the returned word. The path (P) field in both words is the same as the two low-order bits of the unit modifier code. The binary value of the P field selects one of four specific channels in a minterm group to be interrogated as follows:

Bit 18	Bit 17	
0	0	Channel number four is selected for interrogation.
0	1	Channel number three is selected for interrogation.

FUNCTION WORD



RETURNED WORD



INT REG BIT	UNIT
0	MPX STATUS CHANGE
1	DCP 1
2	DCP 2
3	DCP 3
4	DCP 4
6	BIC 1
7	BIC 2
8	MPX ERROR
9	MPX I/O FINISH

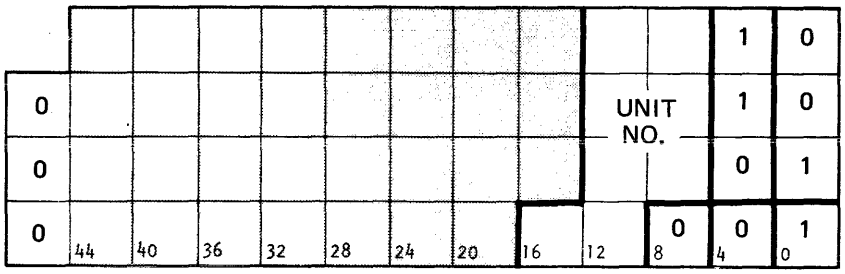
MV 1623

Figure 5-14. Read Interrupt Register

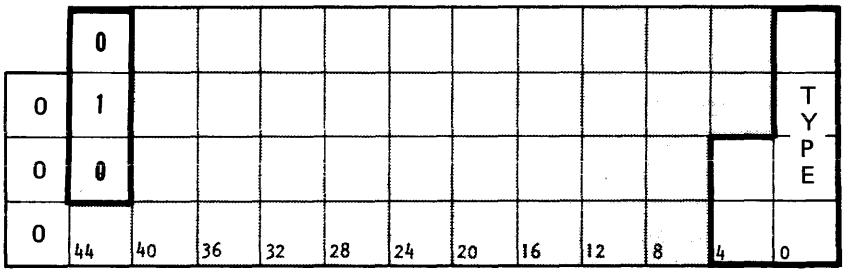
Bit 18	Bit 17	
1	0	Channel number two is selected for interrogation.
1	1	Channel number one is selected for interrogation.

The criteria for determining if the path selected is available is the same as that specified for the multiplexor interrogate IO path function. This criteria was listed previously in this section.

FUNCTION WORD



RETURNED WORD



MULTIPLEXOR MODEL FIELD

BITS 45 46 47

0 1 0 = B6800 MULTIPLEXOR

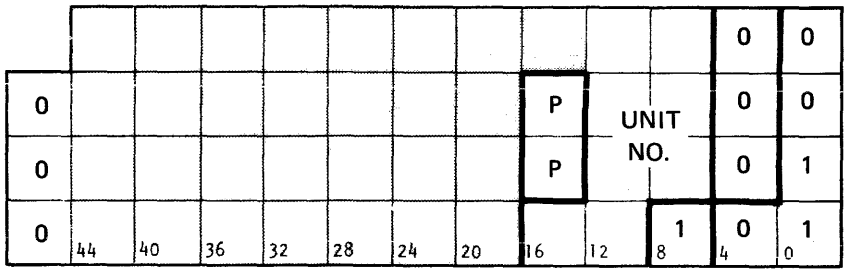
UNIT TYPE FIELD

TYPE	TYPE CODE HEX	TYPE	TYPE CODE HEX
NO UNIT	00	TRAIN PRINTER	07
DISK FILE CONTROL IVA	01	CARD READER	09
SINGLE LINE CONTROL	02	CARD PUNCH II	0B
BUFFERED LINE PRINTER		MT 7-TRACK NRZ	0D
BCL	06	MT 9-TRACK NRZ	0E
EBCDIC	26	MT 9-TRACK PE	0F
CONSOLE II	02	DISK PACK	
DISK FILE 5N	19	B 6383	31

MV 1624

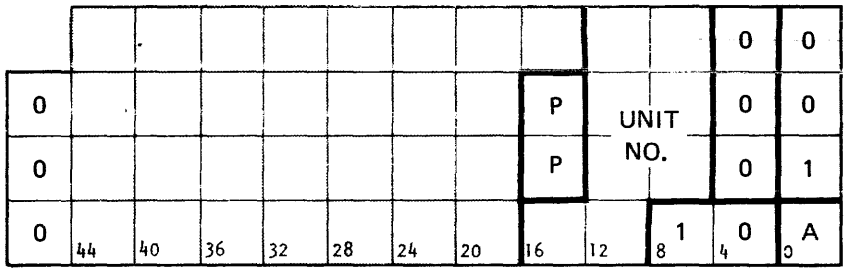
Figure 5-15. Interrogate Unit Type

FUNCTION WORD



BITS [18:2] = PATH = 2 LOW-ORDER UNIT MODIFIER BITS

RETURNED WORD



A=1 PATH AVAILABLE
A=0 NO PATH AVAILABLE

MV 1625

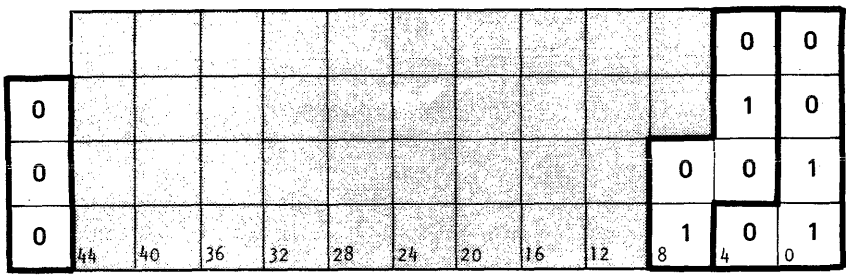
Figure 5-16. Interrogate IO Path Address

Read Processor Time Counter Multiplexor Function

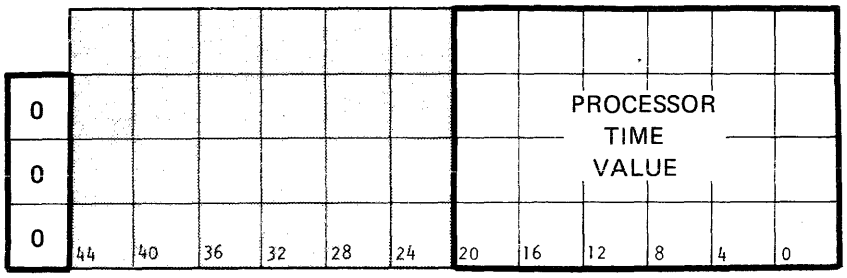
When a SCAN-IN operator passes a function word that contains the read processor timer function code, the multiplexor responds by returning a word to the data processor. The "returned" word contains the value of the processor timer register.

Figure 5-17 shows the format of the function word and the returned word for the read processor timer function. The purpose of this function is to provide the B 6800 system with a method for counting billing time other than real time in the data processor. The processor timer is a twenty-four bit register that counts at the rate of 2.4 microseconds per increment, up to a maximum time count of about 40 seconds. The value that is returned to the data processor is the state of the processor timer register. Each time that the read processor timer function is executed the processor timer is reset to a value of zero.

FUNCTION WORD



RETURNED WORD



MV 1626

Figure 5-17. Read Processor Timer

The processor timer is inhibited from counting any time that either the data processor or the multiplexor is performing a memory cycle.

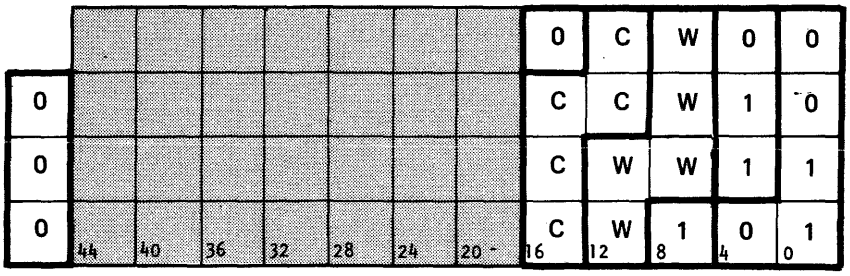
Read Scratch Pad Word Multiplexor Function

When a SCAN-IN operator passes a function word that contains the read scratch pad word function code, the multiplexor responds by returning a word to the data processor that contains the values from the scratch pad word specified.

Figure 5-18 shows the format of the function word and the returned word. The channel of scratch pad memory is specified by the value of the channel field and the particular word is specified by the value of the word field.

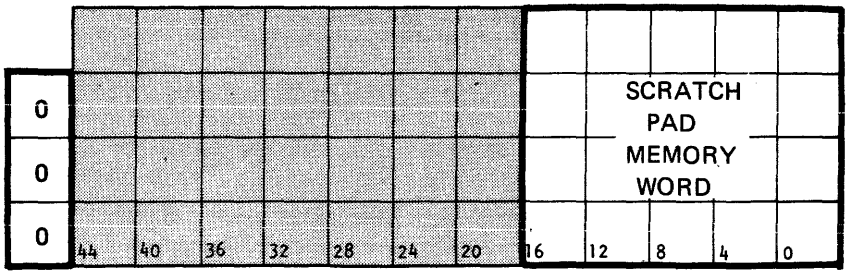
The values that are present in each word of scratch pad memory are defined later in this section under the subject heading of SCRATCH PAD MEMORY. The first word in a channel of scratch pad memory is word zero, and the last word in a channel is word fifteen.

FUNCTION WORD



BIT FIELD	FIELD NAME	MEANING
50:3	TAG	THIS FIELD CONTAINS A TAG VALUE OF ZERO FOR SCAN-IN OPERATIONS.
19		THIS BIT MUST BE ZERO TO ADDRESS THE MULTIPLEXOR
18:5	CHANNEL	THE MULTIPLEXOR SCRATCH PAD CHANNEL ID FIELD. THIS FIELD IS ONLY USED FOR READ SCRATCH PAD WORD TYPE FUNCTIONS. THIS FIELD SPECIFIES ONE OF TWENTY SCRATCH PAD CHANNELS IN THE MULTIPLEXOR WHICH IS TO BE READ.
13:5	WORD	THE SCRATCH PAD CHANNEL WORD FIELD. THIS FIELD IS ONLY USED FOR READ SCRATCH PAD WORD TYPE FUNCTIONS. THIS FIELD SPECIFIES ONE OF SIXTEEN WORDS IN A MULTIPLEXOR SCRATCH PAD CHANNEL WHICH IS TO BE READ.
8:4	FUNCTION	THE FUNCTION CODE.
4:5		REQUIRED CODE FOR B 6800 SYSTEM.

RETURNED WORD



MV 1627

Figure 5-18. Read Scratch Pad Word

Interrogate IO Path Address Override Multiplexor Function

When a SCAN-IN operator passes a function word that contains the interrogate IO path address override function code the multiplexor responds by returning a word that identifies whether or not an override path to a peripheral unit is available. The format of the returned word is identical with that shown in figure 5-16.

The difference between the interrogate IO path address and the interrogate IO path address override functions is that a path will be available for the override scan operation even if the pseudo-busy flip flop is set. This is not the case for the interrogate IO path address scan operation.

The use of the override path address method of IO operations allows the multiplexor to exercise IO devices when the pseudo-busy flip flop is set. The purpose of this form of IO operations is to allow an IO device to be initiated after an error has caused the pseudo-busy flip flop to be set. Some types of IO devices retain information about why the pseudo-busy flip flop was set (the cause of the error) but this information is lost if the IO device is initiated in the normal manner. Therefore, the use of override path operations allows this error data to be recovered.

Read Interrupt Literal Multiplexor Function

When a SCAN-IN operator passes a function word which contains the read interrupt literal function code, the multiplexor returns a word indicating the highest priority external interrupt that is not masked by the interrupt mask register. Figure 5-19 shows the coded value for each interrupt.

Read Interrupt Mask Multiplexor Function

When a SCAN-IN operator passes a function word that contains the read interrupt mask function code, the multiplexor responds by returning an interrupt mask word to the data processor. Figure 5-20 shows the format of the function word and the interrupt mask value word.

The multiplexor has the capability to mask interrupts and prevent them from interrupting the data processor. The operating system controls the value of the bits in the interrupt mask register, and the capability of reading the value of the interrupt mask register is part of the method used to provide control over this function. The other part of the control function is through use of a SCAN-OUT function, to set a value in the interrupt register.

MULTIPLEXOR SCAN-OUT FUNCTIONS

The B 6800 system multiplexor performs four functions as a result of the SCAN-OUT operation. The different SCAN-OUT functions are defined by the value of the function code in the function word. The following paragraphs will define the specific information that is passed to the multiplexor when three of the SCAN-OUT operations are executed. These three definitions will also define the specific information that is returned to the data processor as a result of the SCAN-OUT operation. The fourth SCAN-OUT function is the initiate IO operation, and will be discussed as a separate subject later in this section.

The top word in the stack is a function word that defines one of the multiplexor functions which is to be performed. The second word in the stack contains information or data that is required by the multiplexor in performing the required function.

Set Time of Day Multiplexor Function

The top word in the stack registers is a function word that defines the set time of day function. The second word in the data processor top of stack registers is the value that is to be set into the time of day register. Figure 5-21 shows the format of the two top words in the data processor stack registers when the SCAN-OUT operation is performed. After the scan-out operation, the top of stack registers in the data processor (the A and B registers) are marked not valid (AROF and BROF are reset).

Set Interrupt Mask Multiplexor Function

The top word in the data processor stack registers is a function word that defines the set interrupt mask functions. The second word in the stack is the value that is to be set into the interrupt mask register. The format of the function word and the mask register information word are shown in Figure 5-22. After the scan-out operation, the top of stack registers are marked not valid.

Set Pseudo Busy Multiplexor Function

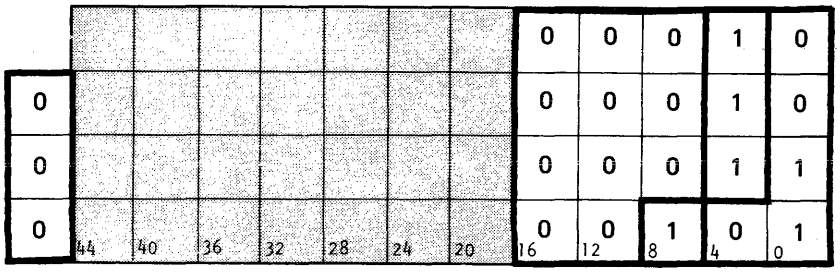
The top word in the data processor stack registers is a function word that defines the set pseudo busy function. The second word in the stack is the value that is to be set into the pseudo busy flip-flop, specified by the function word. The format of the function word and the pseudo busy data word are shown in Figure 5-23. After the scan-out operation, the top of stack registers are marked not valid.

There is a pseudo busy flip-flop for each multiplexor channel. The unit number field from the function word is used by the multiplexor in conjunction with the P field to determine which of the twenty pseudo busy flip-flop is to be set (or reset). For more information about the P field of the function word refer to the discussion of the SCAN-IN interrogate IO path address operation, which was covered earlier in this section.

SOFTWARE ASPECTS OF IO OPERATIONS IN THE B 6800 SYSTEM

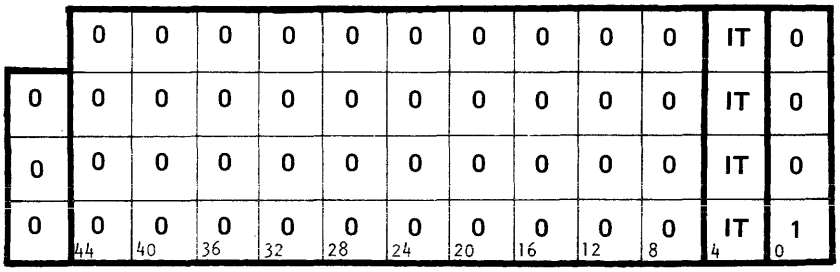
One of the major functions of the MCP in the B 6800 system is to provide control over input/output operations. The use of the software operating system to perform this function is efficient because the management of peripheral device operations is a major time consuming consideration of computer system operations. The use of software procedures to control input/output operations relieves the programming and system operations staffs of this burden.

FUNCTION WORD



50:05 = TAG FIELD, EQUAL TO ZERO
47:28 = UNUSED
19:11 = UNUSED FIELD THAT MUST CONTAIN ZEROS
08:04 = FUNCTION CODE, EQUAL TO F (HEX)
04:05 = UNUSED CODE, MUST EQUAL 03 (BINARY) FOR B 6800 SYSTEM

WORD IN "B" REGISTER



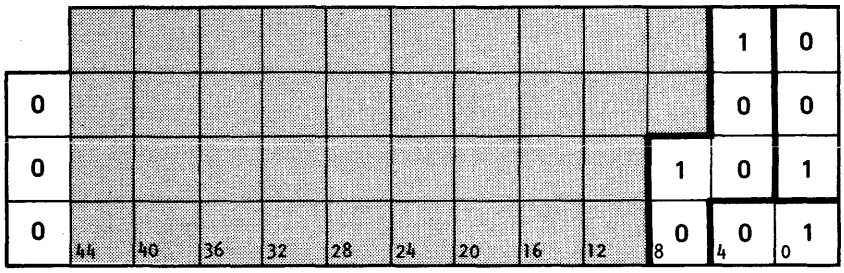
50:03 = TAG, DOUBLE PRECISION
47:40 = UNUSED FIELD, EQUAL TO ZERO
03:04 = REQUIRED BINARY VALUE

07:04 = INTERRUPT TYPE
= 0001 = DCP 1
= 0010 = DCP 2
= 0011 = DCP 3
= 0100 = DCP 4
= 0110 = BIC 1
= 0111 = BIC 2
= 1000 = MPX ERROR
= 1001 = IO FINISH
= 1111 = STATUS CHANGE

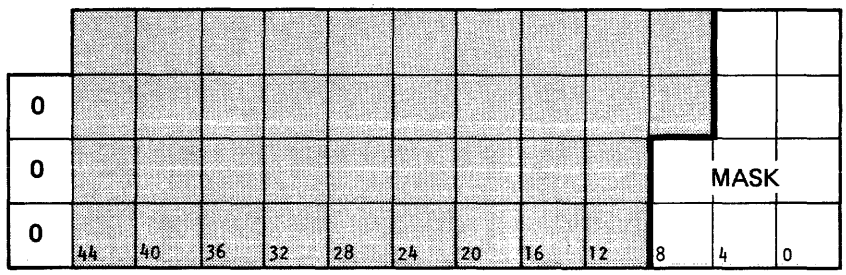
MV 1628A

Figure 5-19. Read Interrupt Literal

FUNCTION WORD



RETURNED WORD



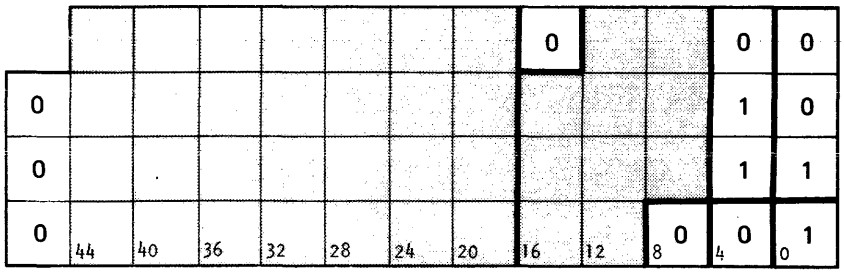
RETURNED WORD CODING:

MASK BIT	UNIT
0	MPX STATUS CHANGE
1	DCP 1
2	DCP 2
3	DCP 3
4	DCP 4
6	BIC 1
7	BIC 2
8	MPX ERROR
9	MPX I/O FINISH

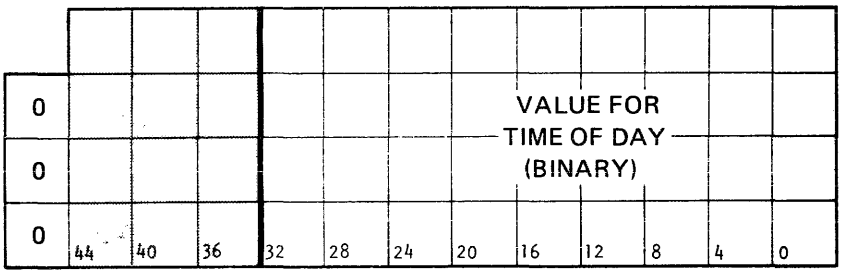
MV 1629

Figure 5-20. Read Interrupt Mask

FUNCTION WORD (DATA PROCESSOR A REGISTER)



INFORMATION WORD (DATA PROCESSOR B REGISTER)



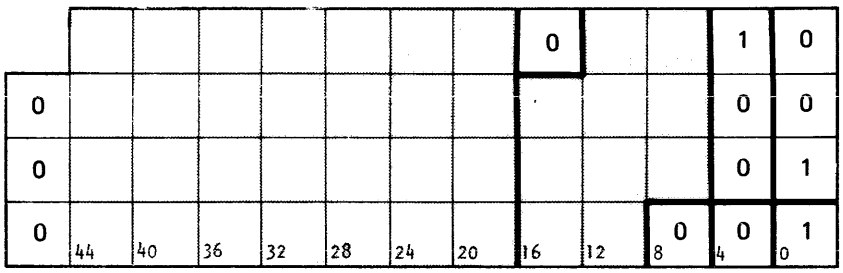
MV 1630

Figure 5-21. Set Time of Day

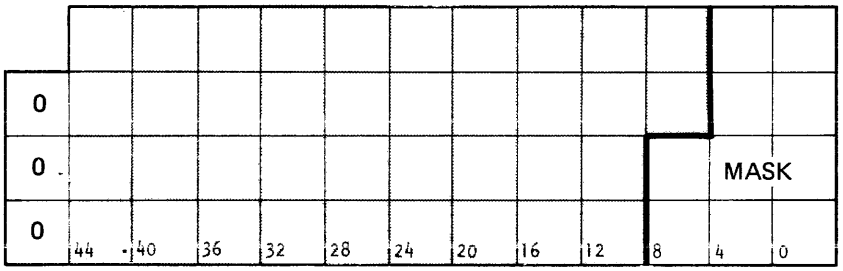
To make the operation and control of peripheral devices functional in the B 6800 system, the MCP creates and maintains peripheral unit tables and file label tables in local memory. The unit tables in memory contain such information as unit numbers, unit types, unit status, and unit assignments. The file label tables contain the file labels of files associated with a peripheral device. All of the information about a particular peripheral device or file label is cross referenced such that given a unit number, the MCP can determine all of the unit table or file label data. Based on the information that is maintained in the unit tables, and file label tables, the MCP monitors the operation of system peripheral devices in such a way that human intervention is kept to a minimum, and efficient input/output operations are maintained at the maximum.

The control of the input/output subsystems of the B 6800, by the MCP, is through use of the SCAN-IN/OUT operators of the data processor. However, these operators can only direct that an IO operation be initiated, or that some information that is maintained in the multiplexor about a peripheral device is to be "returned" to the MCP (through the data processor). The MCP also requires information about any change in the status of a peripheral device, when a directed IO operation is completed, or when the operation is terminated because of an error.

FUNCTION WORD (DATA PROCESSOR A REGISTER)



INFORMATION WORD (DATA PROCESSOR B REGISTER)



B REGISTER CODING:
BITS [9:10] = 1 = MASKED:

BIT	INFORMATION
0	STATUS CHANGE
1	DCP1
2	DCP2
3	DCP3
4	DCP4
6	BIC1
7	BIC2
9	I/O FINISH

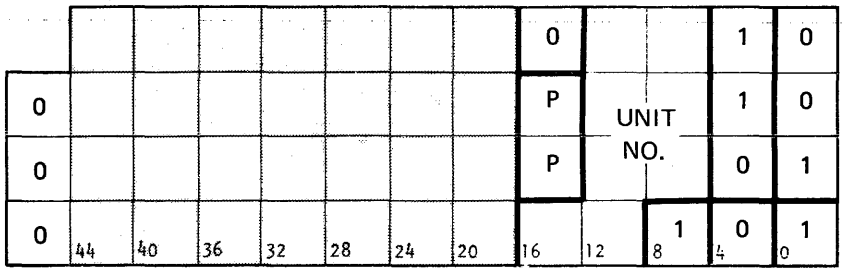
MV 1631

Figure 5-22. Set Interrupt Mask

READY STATUS

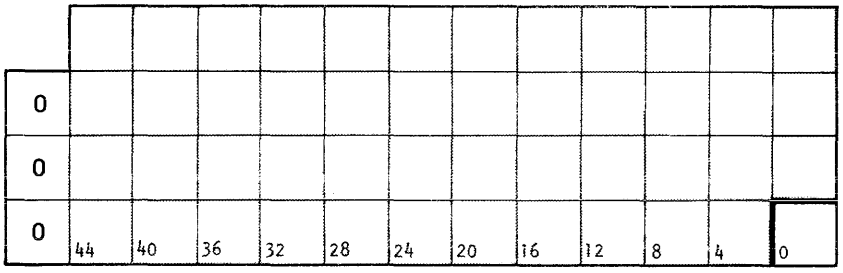
A change in the status of certain peripheral devices, or the completion/termination of a directed operation on a peripheral device causes the multiplexor to interrupt the data processor. The interrupt in the data processor causes the MCP to interrogate the cause of the multiplexor interrupt, and thus the MCP is made aware of changes in peripheral units.

FUNCTION WORD (DATA PROCESSOR A REGISTER)



BITS [18:2] = PATH = 2 LOW-ORDER UNIT MODIFIER BITS

INFORMATION WORD (DATA PROCESSOR B REGISTER)



BIT 0=1 – SET BIT 0=0 – RESET

MV 1632

Figure 5-23. Set Pseudo Busy

If the peripheral control or unit goes from NOT READY to READY, the MCP will perform different functions depending on the other data in the unit tables, as follows:

a. If the unit is assigned to a TASK, the unit tables will be updated to show that the unit is READY, and the data file operation will be resumed from the point where the unit went NOT READY.

b. If the unit is not assigned to a TASK, the MCP will read the label record for the data file (rewinding a tape file if necessary, to get to the label record of the file). The MCP will then initiate a search by all tasks that are currently waiting for a file, to see if the file is needed currently, or not. If the file is needed by a TASK that is in progress, the MCP will cause the peripheral unit to be dedicated to the particular JOB (in the unit tables), and will write the file label data into the unit tables.

   If no TASK is in process that requires a file with the data file label, then it will enter the label of the file in the label tables, for later reference, when a TASK may require the file of data. This label table contains a reference to the IO device that contains the file, so that the device can be properly dedicated to a TASK, when the TASK is executed.

The multiplexor maintains information about the status of the system peripheral controls for the IO devices. The MCP causes the status of the peripheral controls, as it is reported by the multiplexor, to be compared against the data in its unit tables from time to time. If the status of a peripheral unit, as reported by the multiplexor, is not the same as the data in the MCP unit tables, then the MCP will perform additional operations, as required, to update the data in the unit tables.

The status of a peripheral control or device that is maintained in the multiplexor only indicates whether or not the peripheral control or device is available for operation, and whether or not the device is not ready. The comparison of the current status with the last status that was recorded in the unit tables indicates whether or not there has been a change in the ready status of the control or device.

A change in the ready status of an IO control or device can occur if:

a. The control or device goes from READY to NOT READY.

b. The control or device goes from NOT READY to READY.

c. The IO device reports a STATUS CHANGE.

If the comparison of the unit tables to the status vector data from the multiplexor shows that a peripheral control or device has gone from READY to NOT READY, and the unit tables show that the unit is assigned to a software TASK, the MCP will update the unit tables to show that the unit went NOT READY. The MCP may also inform the operator of the system that the program (TASK) is waiting for the IO device to be made READY.

STATUS CHANGE

The preceding discussion defined the actions that the MCP takes when there has been a change in the ready status of a peripheral control or peripheral device in the system. There are two types of peripheral units that report a STATUS CHANGE, in addition to reporting a change in ready status. These two unit types are disk packs, and operator display terminals (ODT's). These two units need this additional reporting mechanism because they perform operations that are asynchronous to the normal directed IO operations.

The disk pack device requires that a write/read head must be positioned properly before the write or read operation can be performed. At times, the positioning of the write/read head takes so much time that another IO device can be serviced while the write/read head is being positioned. The positioning of a disk pack write/read head is called a SEEK function. The disk pack device uses the status change reporting mechanism to let the multiplexor (and the MCP) know that a seek operation is completed, and the write or read operation to the pack can be performed without further SEEKing being required.

The ODT device is used for two-way communications between the system (MCP), and the human operator of the system. The MCP writes messages upon the CRT screen of the ODT for the operator to observe. This part of the ODT unit operation is the same as any other IO device that serves as an output for the MCP. However, the human operator must have some way to cause the MCP to read a message that has been typed upon the screen of the CRT. This capability is provided by the use of the TRANSMIT key of the ODT device. The TRANSMIT key of an ODT device causes a STATUS CHANGE condition to be sensed in the multiplexor, and consequently, be known to the MCP. The MCP knows that the particular device type is an ODT, and will read the message on the face of the ODT screen, when the STATUS CHANGE condition occurs.

INPUT OUTPUT OPERATIONS

The multiplexor controls the flow of information for the peripheral devices that are connected to the B 6800 system. The software procedures of the MCP direct that an IO operation is to be performed, through use of the SCAN-OUT operator in the data processor. Once the multiplexor has been initiated to perform an IO operation, it becomes an autonomous control unit in the system. When the IO operation is complete the multiplexor causes an interrupt in the data processor.

The multiplexor is capable of performing up to twenty simultaneous IO operations. Each channel in the multiplexor has 16 words of scratch pad memory (twenty bits per word), which is used for storing control and status information about the IO operations for the IO control channel. The multiplexor also has a data buffer for each of the twenty IO control channels. Twelve of the IO control data buffers are 256 or 512 bytes in length, and the other eight are 128 or 256 bytes in length.

Figure 5-8 shows the path of information flow between the memory control interface bus and the peripheral control interface bus. This path includes the use of two of the hardware stack registers (the C and Z registers) in the data processor, the translate logic, and data buffer in the multiplexor.

IO Device Numbering System

The B 6800 system can interface with up to 256 different peripheral devices. Each peripheral device is assigned a unit number, and the data processor uses this unit number to identify which peripheral device the multiplexor is to initiate (in the SCAN-OUT function word). The multiplexor groups common peripheral device types into groups in which the unit numbers of the group are numerically consecutive. A common group of peripheral numbers are assigned to a channel or channels of the multiplexor scratch pad memory, and to common data buffer(s). Therefore all of the peripheral devices that are operated through a channel of scratch pad memory in the multiplexor have common operating characteristics, they have consecutive unit numbers, and they all use common data buffers in the multiplexor.

Initiate Input Output Operation

The control of an IO device by the multiplexor is divided into two types of multiplexor operations. These two types of IO operations are identified as the initiate cycle, and the service cycle. An initiate cycle is performed to prepare the scratch pad memory channel, the data buffer, and the IO control for the operation of the peripheral device. A service cycle is performed to handle data flow through the data buffer. A modified type of service cycle is performed at the conclusion of an IO operation to accumulate and format the result descriptor information about how the peripheral device functioned during the operation.

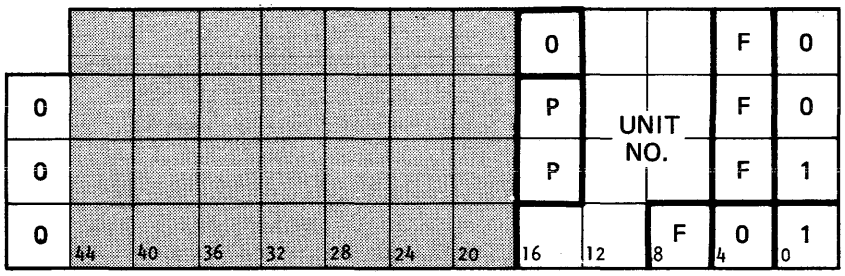
The multiplexor performs Burst cycles to transfer data between memory and the data buffer.

At the conclusion of the result descriptor service cycle, the multiplexor causes the interrupt line to the data processor interrupt controller to go to a true level (refer to figure 5-10). This interrupt is the end of the IO operation in the multiplexor. The result descriptor data that was collected and stored in scratch pad memory is retained until the data processor answers the interrupt. Removing the result descriptor data from scratch pad memory clears the channel, and providing that there are no faults in the IO subsystem, prepares the IO device for the next system operation.

The data processor utilizes the SCAN-OUT operator to start an initiate IO cycle in the multiplexor. The format of the initiate IO word (IIOWD), the IO area descriptor (IOAD), as well as the IO control word (IOCW) are described in the following paragraphs. These three words are required by the multiplexor to initiate an IO device.

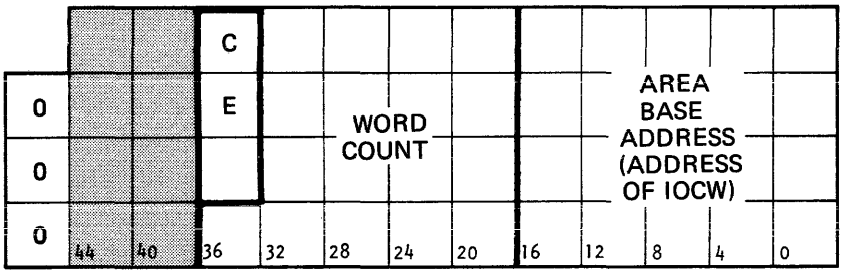
Figure 5-24 shows the formats of the IIOWD, IOAD, and the IOCW. The fields in these words are described in the following paragraphs.

IIOWD



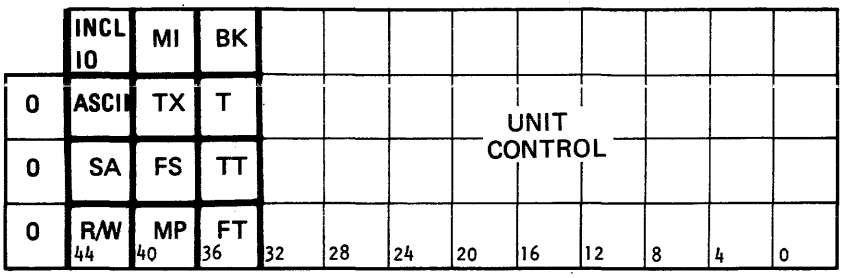
LOCATED IN TOP OF STACK "A" REGISTER

IOAD



LOCATED IN TOP OF STACK "B" REGISTER

IOCW



LOCATED IN SYSTEM MEMORY IN THE WORD ADDRESSED BY THE BASE ADDRESS FROM THE IOAD WORD

MV 1633

Figure 5-24. Multiplexer Initiate IO Words Format

IIOWD Format

The IIOWD contains five significant fields as follows:

Bit Field	Significance
4:5=00011	A fixed value required for B 6800 operations.
8:4	The function code field. This field contains a code that identifies which of three initiate IO functions the multiplexor is to perform.
16:8	The unit ID field. The unit ID field is used to specify which IO device is to be initiated.
18:2	The path address field. The path address field is used to specify which of several different paths to an IO device is to be used.

IOAD Format

The IOAD contains three significant fields as follows:

Bit Field	Significance
19:20	The base address field. The base address field is used to specify the beginning address of a buffer in system memory which is used to pass data from/to an IO device. The first word in the buffer contains the IOCW that describes the type of operation the IO device is to perform. The rest of the buffer is used to store data that is input or output from/to the multiplexor data buffer.
36:17	The word count field. The word count field is used to specify the number of words in the buffer. The address of the first word of the buffer in system memory is specified by the contents of the base address field.
39:3	The count extension field. The count extension field is used to specify the number of characters of data that are present in the last word of the buffer. If the count extension field is equal to zero the number of characters in the buffer is a multiple of the number of words in the buffer, (the word count). If the count extension field is not equal to zero the word count of the buffer is extended by one word and the number of character positions used in the additional word is equal to the value of the count extension field.

IOCW Format

The IOCW word contains data that is used by the multiplexor to determine what kind of operation the IO device is to perform, and what optional features of the device are to be used during the operation. The IOCW also specifies what form the data is in, and any translation characteristics that are to be applied to the data as it passes through the multiplexor. The data fields in the IOCW are as follows:

Bit Field	Significance
BIT 47	Bit 47 is the incomplete IO operation bit. This bit is used as part of the maintenance functions of the MDP. If this bit is true (a binary one), information will be moved between memory and the multiplexor data buffer, but no IO device will be initiated. The IO buffer, that will have information loaded into it, is the same one that would be used for a completed initiate IO operation. It is selected by the same process of the multiplexor that is used to

Bit Field	Significance
	select a channel or scratch pad memory for a normal completed IO operation. If this bit is false a completed IO operation will be performed, and the information that is loaded into the buffer will be handled as specified by bit 44 of the IOCW.
BIT 46	The ASCII translate bit. If this bit is true, data loaded into the data buffer of the multiplexor will be translated to the ASCII code form as it is passed between the data buffer in the multiplexor, and memory. (See the steering and mask translate logic block in figure 5-8.) If this bit is false no ASCII translation will occur, and the other translation forms that may be used will be as specified by the states of bits 41 and 42 of the IOCW.
BIT 45	Bit 45 is the software attention bit. The MCP uses this bit in the IOCW for software retry capability. This bit is returned in the result descriptor, and when true, forces the result descriptor exception bit to be true.
BIT 44	The read or write bit. If bit 44 is true, a peripheral read operation is to be performed. If bit 44 is false, a peripheral write operation is to be performed.
BIT 43	The memory inhibit bit. If this bit is true the multiplexor will initiate the IO device, and will accept data input from the IO device, but no information is transferred from the multiplexor to system memory. If this bit is false the multiplexor will transfer input data from or to the IO device, to or from system memory.
BIT 42	The translate bit. If this bit is true the multiplexor performs a translation of the data as it is transferred between system memory and the data buffer. If this bit is false no translation is performed. If translation is specified (bit 42=1) **then bit 41 will define what type** of translation is to be performed.
BIT 41	The eight-bit frame size bit. If this bit is true the translation performed is to eight-bit characters. If this bit is zero the translation is to six-bit characters.
BIT 40	The memory protect bit. If this bit is true and bit 44 is also true the multiplexor will perform a protected memory write operation. If the memory protect bit (bit 48 of the word in memory that is being written into) is true a memory protect interrupt is detected, and no data is written into memory.
BIT 39	The backward bit. If this bit is true the IO device will be operated in the backward direction. The data flowing between the data buffer in the multiplexor and the IO control will be in the reverse order, proceeding from the last address in the buffer to the first address in the buffer. If this bit is false the data flow between the data buffer in the multiplexor and the IO control will be in the forward direction.
BIT 38	The test operation bit. If this bit is true the multiplexor will cause the IO control to perform a test operation upon the device specified. If this bit is false no test operation will be performed. When this bit is true it overrides the other bits in the unit control field of the IOCW such that only a test operation will be performed, regardless of the state of the other bits in the unit control field.

Bit Field	Significance
37:2	Bit 37 is the transfer tags bit, and bit 36 is the force tags bit. These two bits operate as a control field to define the type of tag fields that the multiplexor is to use when words of data from the data buffer are transferred to system memory. The tags that will be used are as follows:

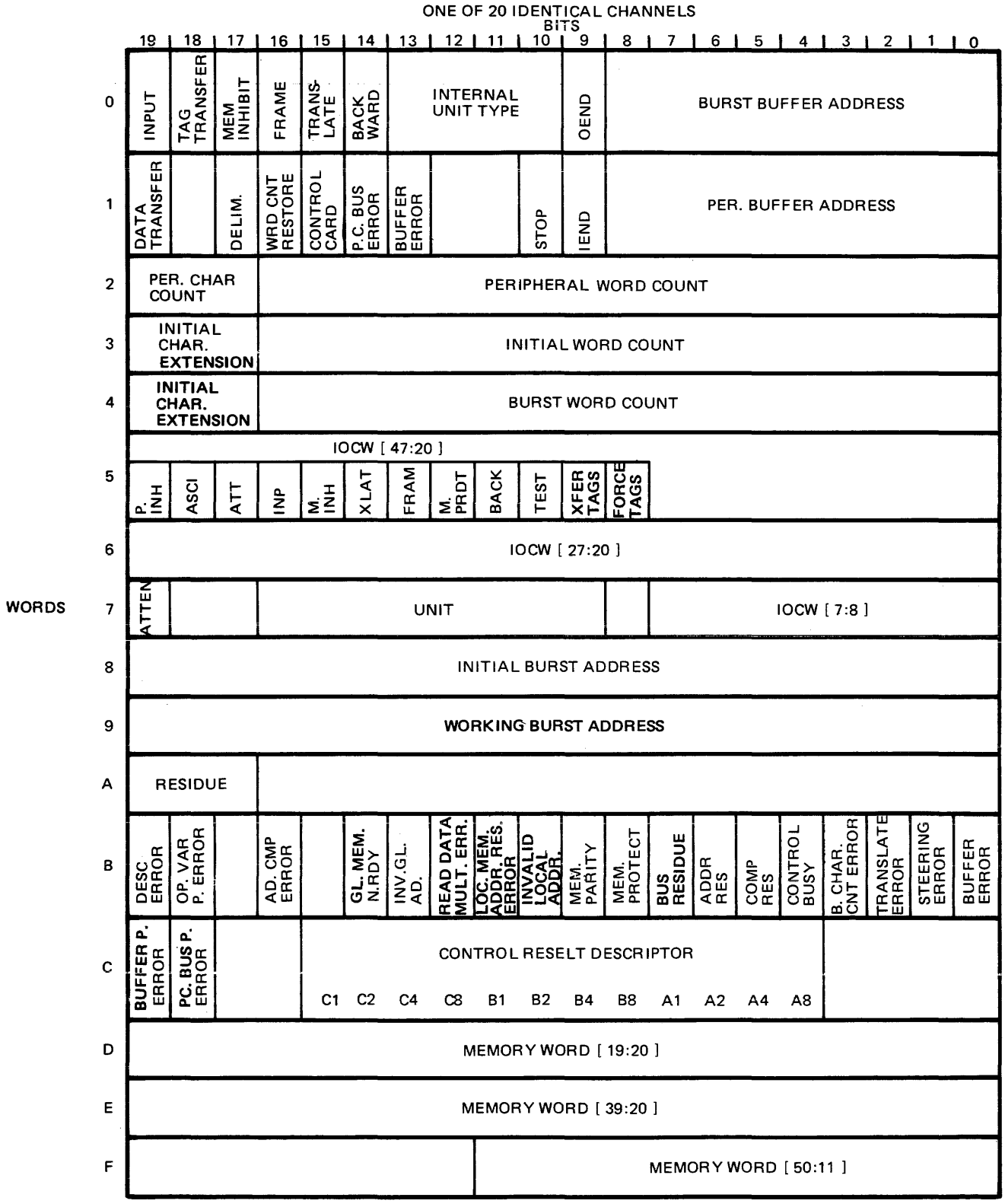
Bit 37	Bit 36	Tag Field
0	0	Data words will be transferred with single precision word tags (000).
0	1	Data words will be transferred with program code word tags (011).
1	0	Data words will be transferred with tag fields as specified in the input data. If eight-bit data is being received every seventh character position of the input data, beginning with the first character, will be used to make the tag field that is written into memory. The tag field will be equal to the value of the low-order three bits of the character. If six-bit data is being received every ninth character of the input data, beginning with the first character, will be used to make the tag field.
1	1	Data words will be transferred with double precision word tags (010).

35:36	The unit control field. The unit control field is used to specify the optional characteristics that may be used for a peripheral device type. Examples of unit control options are spacing, skipping, cyclic redundancy characteristics (CRC), code table loading, etc. These characteristics are variable between different types of IO devices, and thus the format of the unit control field is also variable. The formats of unit control fields are defined in section 11 of this manual.

Scratch Pad Memory

Scratch pad memory in the multiplexor (refer to figure 5-8) is used to store control information about IO operations that are in process, and result descriptor information for IO operations that are completed. This memory is composed of twenty channels, each with 16 words. The data that is stored in each word of a channel of scratch pad memory is shown in figure 5-25. Each channel of scratch pad memory corresponds to one of the twenty IO controls that may be mounted in one of the two possible peripheral control cabinets.

A unit number for an initiate IO cycle is present in the IIOWD. This unit number is decoded to select a channel of the scratch pad memory. The data fields that were present in the IIOWD and the IOAD are placed in word three and word seven of the proper IO channel. The multiplexor performs a memory cycle on the base address of the system memory



MV 1634

Figure 5-25. Multiplexor Scratch Pad Memory

buffer to fetch the IOCW from memory. The data fields in the IOCW are distributed into words five, six, and seven of the proper scratch pad channel. The information contained in the IIOWD, IOAD, and IOCW are sufficient to allow the multiplexor to start the IO operation that is required. If the type of IO operation that is to be performed is an output operation, then the multiplexor burst logic causes the data buffer in the multiplexor to be filled with output data from the system memory buffer. The multiplexor then causes the IO control in the peripheral control cabinet to start the peripheral device. The IO control will pass the data input from the peripheral device into the data buffer, through the peripheral control interface bus.

Data Buffer Logic

The data buffer in the multiplexor (refer to figure 5-8) consists of 20 channels of static random access memory (RAM). These 20 buffers are used to accumulate data in the multiplexor that is being passed between the system memory, and a peripheral unit. The data buffers accumulate and distribute data regardless of the direction of data flow through the multiplexor.

A data buffer is assigned to be used for each channel of scratch pad memory in the multiplexor. This assignment is accomplished by hardware configuration of the multiplexor. When a scratch pad memory channel is selected, the use of a specific data buffer is implied.

Data buffers are either 256 bytes in length, or they are 512 bytes in length. The multiplexor contains 12 data buffers that are 512 bytes in length, and eight buffers that are 256 bytes in length. Through the configuration of the multiplexor, a peripheral device may be assigned to operate with either a 256 bytes data buffer, or a 512 byte data buffer.

The number of bits that may be written into (or read from) an address of a data buffer is 16 bits, or two bytes. When six-bit data (BCL characters) are written into an address of the data buffer, the high-order two bits of each byte are not used, thus, each buffer address will contain two characters regardless of the size of the characters used.

Peripheral devices that are interfaced with a data buffer are classed as single or double character transfer devices. High speed IO devices are of the double character transfer type, while most low speed IO devices are of the single character transfer type.

OP Code and Variant Character Generator

The multiplexor receives control information about the type of peripheral operation that is to be performed from the IOCW. The control field information bits for an IOCW word were previously defined in this section of this manual.

During a multiplexor initiate cycle the multiplexor fetches the IOCW from system memory, and places the IO control information in a channel of scratch pad memory. Subsequently, the multiplexor uses the control information in scratch pad memory to format command instructions for the type of IO device that is to be initiated. The OP code and variant character generator logic (see figure 5-8) is used to format IO control commands which are transferred to the IO control via the peripheral bus. An IO command format is as follows:

COMMAND CODE, VARIANT CHARACTER ONE, VARIANT CHARACTER TWO, FILE ADDRESS:

Where:

COMMAND CODE	consists of two 4-bit characters that define the particular type of IO operation that is to be performed (such as a read or write operation) by the IO device.
VARIANT DIGIT ONE AND TWO	consist of two 4-bit code fields that define optional characteristics of the particular IO device that are to be used/not used for the duration of the current command execution. These two digits contain the unit number of the peripheral device.
VARIANT DIGIT THREE AND FOUR	consist of two 4-bit code fields that are used as an extension of the first two variant digit fields, to define optional characteristics of the IO device for the current command execution.
FILE ADDRESS	consists of a 24-bit field which contains six binary coded decimal numeric digits. The value of this 6-bit number represents the location of the first (starting) address to be used by the peripheral device. This number is only used for disk file or disk pack IO operations on the B 6800 system.

All IO devices do not require all parts of the IO command format. Some devices do not require variant characters to define optional characteristics because the only variations in the type of operation that may be performed are defined by the OP (COMMAND) CODE. Some IO devices only require a single variant character because the number of optional operating characteristics that may be defined is small. Other IO devices do not require a FILE ADDRESS field because they are not disk type devices.

Status Vector Control Circuits

Status vectors are information about the current status of multiplexor channels. Vectors are stored in four 20-bit words of IC memory in the multiplexor. The four words of IC memory are addressable, and the status information in a word of the status vector memory may be displayed in register four of the programmers display by operation of pushbuttons on the keyboard. Each bit in a vector status word represents one bit of status for one of twenty IO paths in the multiplexor.

There are four vector status bits maintained in the status vectors for each PCC channel. The four bits are as follows:

Control Flip-Flop	Vector Word Bits	Meaning of Terms
VALV	Valid	The valid bit is true when the data buffer contains valid data for a peripheral unit (during output operations from the system) and when the buffer contains space in which data may be stored (during input operations to the system). The valid bit is false at all other times.
BURV	Burst request	The burst request bit is true when the multiplexor requires access to main memory. This bit is reset when memory burst is completed, and remains reset until another memory burst is needed.
RDEV	Result. Descriptor	The result descriptor bit is set during data input operations when the last data is written from the buffer (to the peripheral device, and the result descriptor has been returned), and during output operations when the IO unit result descriptor is written into scratchpad memory. The result descriptor bit is reset when the interrupt handling procedures clear the peripheral channel by reading the result descriptor data from scratch pad memory.
PBZF	Pseudo-busy	The pseudo-busy bit is set when the result descriptor from the control contains any error bit.

MEMORY ORGANIZATION

The memory resources of the B 6800 system (refer back to figure 5-2) are organized so that two storage modules of memory may be accessed at any one time. The memory resources of the system consist of up to 512 K words of local memory, and up to 512 K words of global memory. One K is equal to 1024 (decimal) words of memory storage capacity.

A memory word consists of 60 parallel bits of data that are present at the memory exchange port interface. These 60 bits are divided into a parity bit, 51 bits of information, and eight bits of error detection/correction code.

A memory storage module contains 64K (or 128 K) words of continuous memory storage addresses. A 20 bit binary address field is used to select a memory module and a specific word address within the module (refer to figure 5-26). The low order 16 bits (17 bits for 128 K word modules) of the 20 bit address field select one word of the 64 K (or 128K) words within a memory module. The high order four bits (3 bits for 128 K word modules) of the 20 bit memory address field are used to select one of eight local memory modules, or global memory. Any memory address value that does not select a local memory module, selects global memory by default. A local memory storage module is synonomous with one of the local memory ports of the memory exchange.

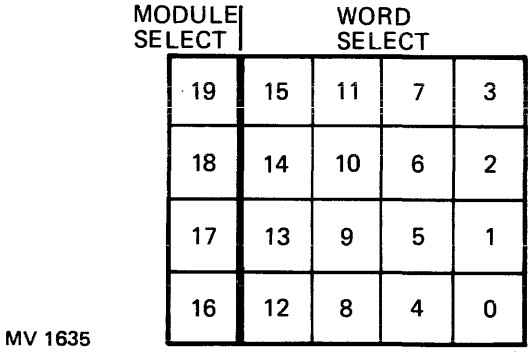


Figure 5-26. Memory Address Decoding

In addition to address and information data, the memory interface bus also transmits control information between the memory control and the memory module. This control information directs the memory operation that will be performed by the memory module, such as write or read functions. For local memory modules, the control signals include the Initiate Memory Cycle (IMC) timing signal, and a three bit memory function code that is comprised of the Read Modify Write (RMW), and Write Cycle Conditional (WCC), and the Parity Error Disable (PED) control signals. The significance of these control signals is discussed in the subsection of this manual titled LOCAL MEMORY PORT INTERFACE CONTROL LOGIC. The control signals that are present at the global memory interface port are discussed in the subsection of this manual titled GLOBAL MEMORY PORT INTERFACE CONTROL LOGIC.

SYSTEM MEMORY INTERFACE

The system memory interface consists of a two-by-five exchange that is used to interface the B 6800 CPU, and the external subsystem(s) associated with the B 6800 system, to the memory resources of the B 6800 system. The two requestor inputs are designated as channel A, and channel B. The five memory storage module interfaces are designated as ports number zero through three (local memory), and the global memory port. Figure 5-27 shows the organization of the requestor interfaces and the port interfaces to the system memory control.

Control of channel A of the system memory control is via the Z12 (14 bit) bus. Control of channel B to the system memory is the responsibility of the external subsystem(s) that are interfaced to memory through that channel. Channel A has priority over channel B for access to the memory resources of the system. However, a channel B access to memory will not be interrupted to service a request from channel A. The priority is limited to determining which of two simultaneous requests for the same memory port will be serviced first. Simultaneous requests to different memory module ports are allowed.

CHANNEL A MEMORY REQUESTOR

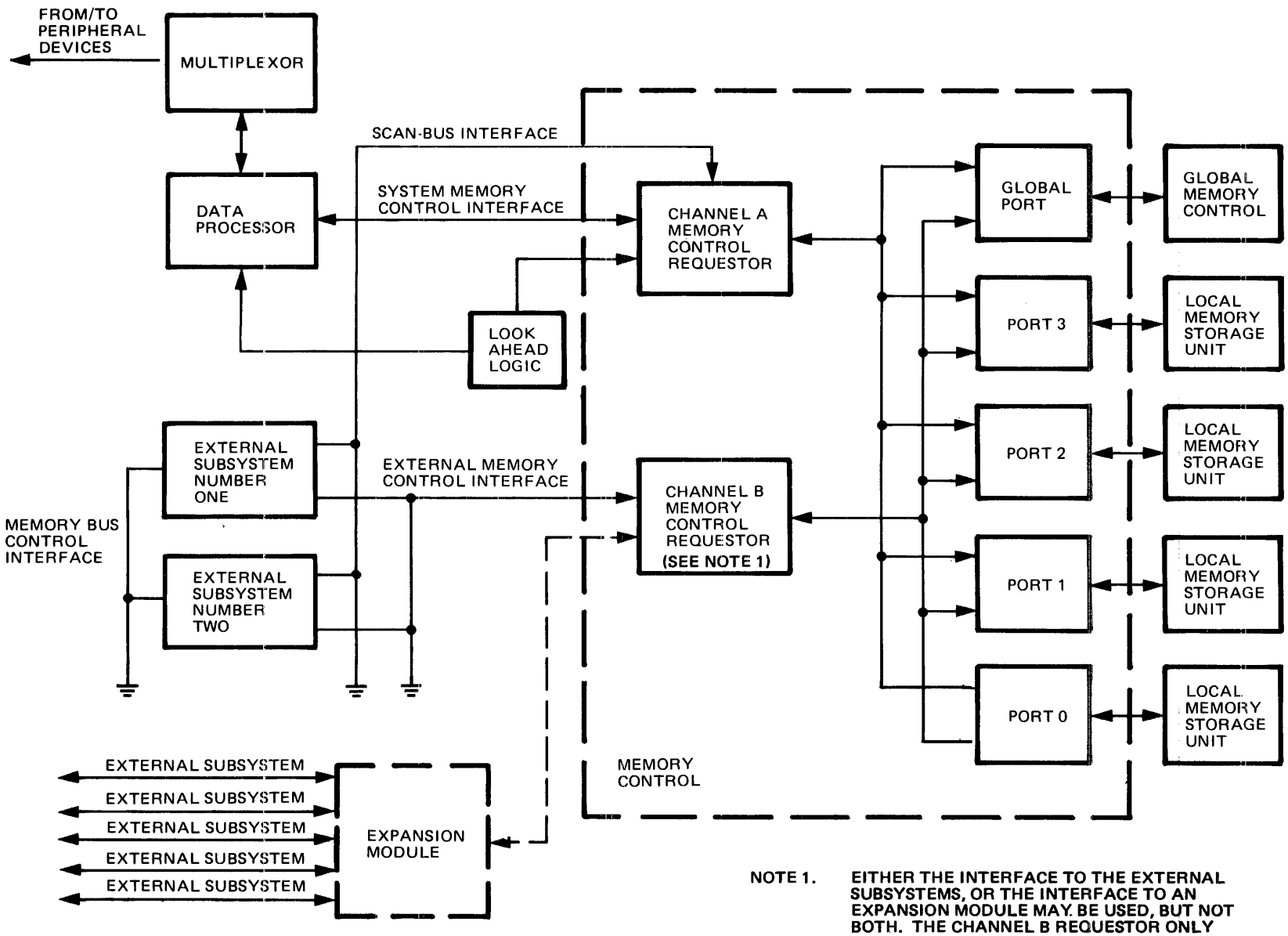
Figure 5-28 shows the paths used in the data processor to access channel A of the system memory control. These paths are controlled by the memory controller, through use of the Z12 bus. All data that is written into memory from the data processor or multiplexor is routed to the system memory interface exchange via the Z4 bus. All data that is read into the data processor, multiplexor, or look ahead logic is routed from the system memory port interface to the Z3 bus. Address information is routed from the memory address register or look ahead address register via an internal memory address bus.

Figure 5-29 shows how information, address, and control data are routed internally within channel A of the memory exchange. This figure also shows how port selection is made within the exchange module, by means of the port select logic.

Figure 5-29 shows the PACK (port A acknowledge) control bus. This bus has a true level if a local memory port interface is selected by the port select logic. If a local memory port is not selected and a valid request is present in the channel A requestor logic, then the global memory port is selected by default.

The 14 bits of the memory control Z12 bus are identified as follows:

Bit Field	Meaning and Usage
5:6	The register select field. This field identifies the data processor register that is to receive the data for a memory read operation, or the data processor register from which data is to be written into memory for a memory write operation. Bit zero is used to select register Z Bit one is used to select register Y Bit two is used to select register X Bit three is used to select register C Bit four is used to select register B Bit five is used to select register A



NOTE 1. EITHER THE INTERFACE TO THE EXTERNAL SUBSYSTEMS, OR THE INTERFACE TO AN EXPANSION MODULE MAY BE USED, BUT NOT BOTH. THE CHANNEL B REQUESTOR ONLY CONTAINS A SINGLE INTERFACE PATH TO THESE UNITS.

Figure 5-27. Memory Control Block Diagram

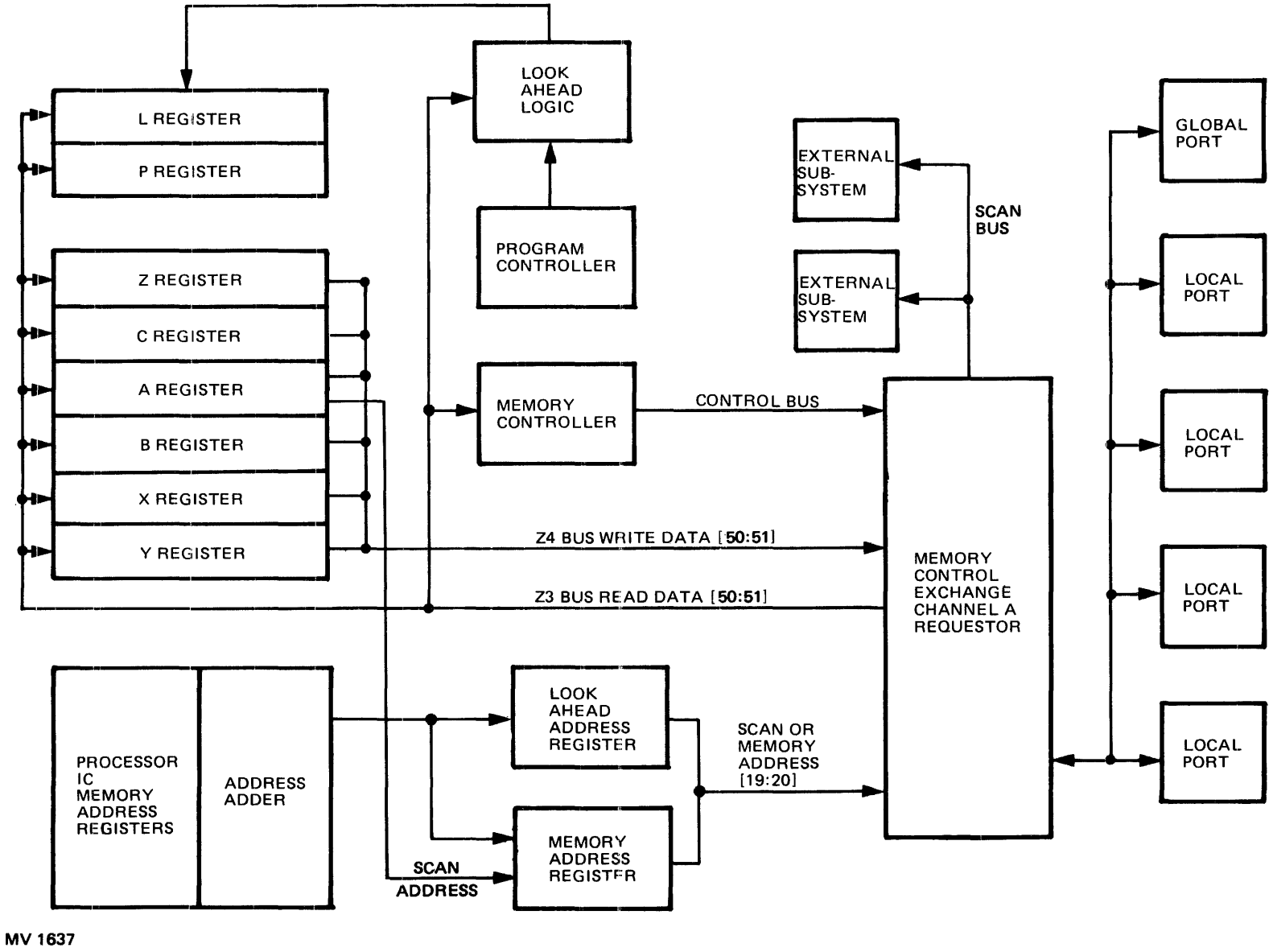


Figure 5-28. Data Processor to Memory Control Exchange Transfer Paths

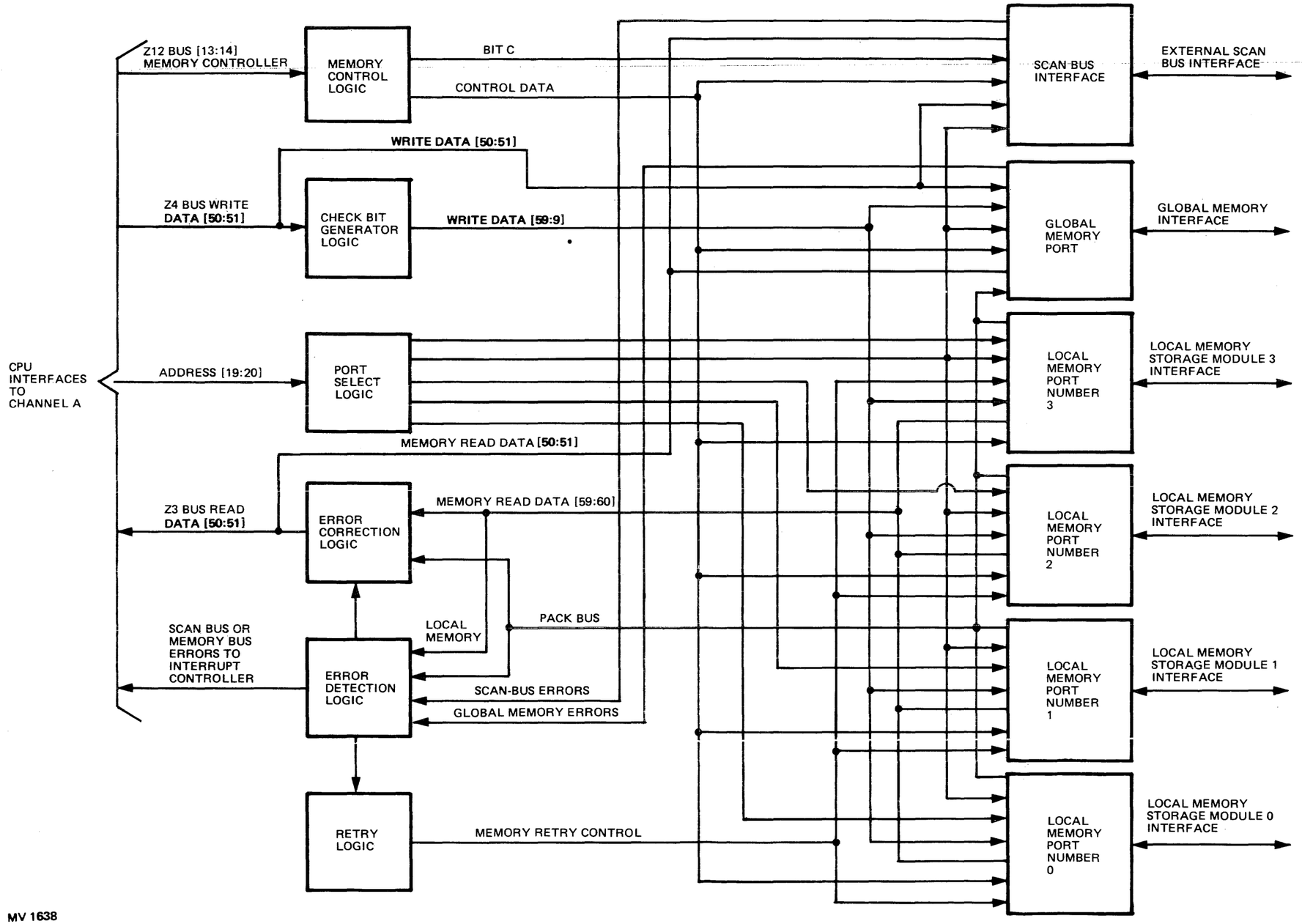


Figure 5-29. Memory Exchange Channel A Functional Block Diagram

Bit Field	Meaning and Usage

8:3 The request field. This field identifies the type of memory operation that is to be performed.

BIT: 9	7	6	Operation to be Performed
0	0	1	Protected write with flashback to C register
0	1	0	Clear write
1	1	1	Overwrite with flashback to C register
0	0	0	Read
0	0	1	Protected write with no flashback

A, B The look ahead request field. When bit A is true the request originates in the look ahead logic. If bit A is false the request originates in the data processor/multiplexor.

Bit B is used to specify which register in the data processor is to receive the data input from memory when a look ahead memory cycle is completed. If bit B is true, the data is to be placed in the L register of the data processor. If bit B is false, the data is to be placed in the P register of the data processor.

C The scan bit. If bit C is true, the data on the channel A information lines is scan data, and will be routed by channel A to the external subsystem(s) via the 80 wire scan bus interface, instead of to the memory storage modules. If bit C is false the operation specified is not a scan operation.

D The global scan bit. If bit D is true the operation to be performed through the global memory port is a global scan operation instead of a global memory operation. If bit D is false the operation to be performed is a memory operation instead of a global scan operation.

Memory Error Detection and Correction

Channel A contains error detection/correction logic circuits (refer to figure 5-30). Each time that a local memory write operation is performed, eight bits of error detection check code are generated by the error detection circuits and appended to the memory write data. The total number of bits that are written in memory during a local memory write operation is 60 bits, of which 52 data bits are write data from the CPU, and the other eight bits are the error detection check code.

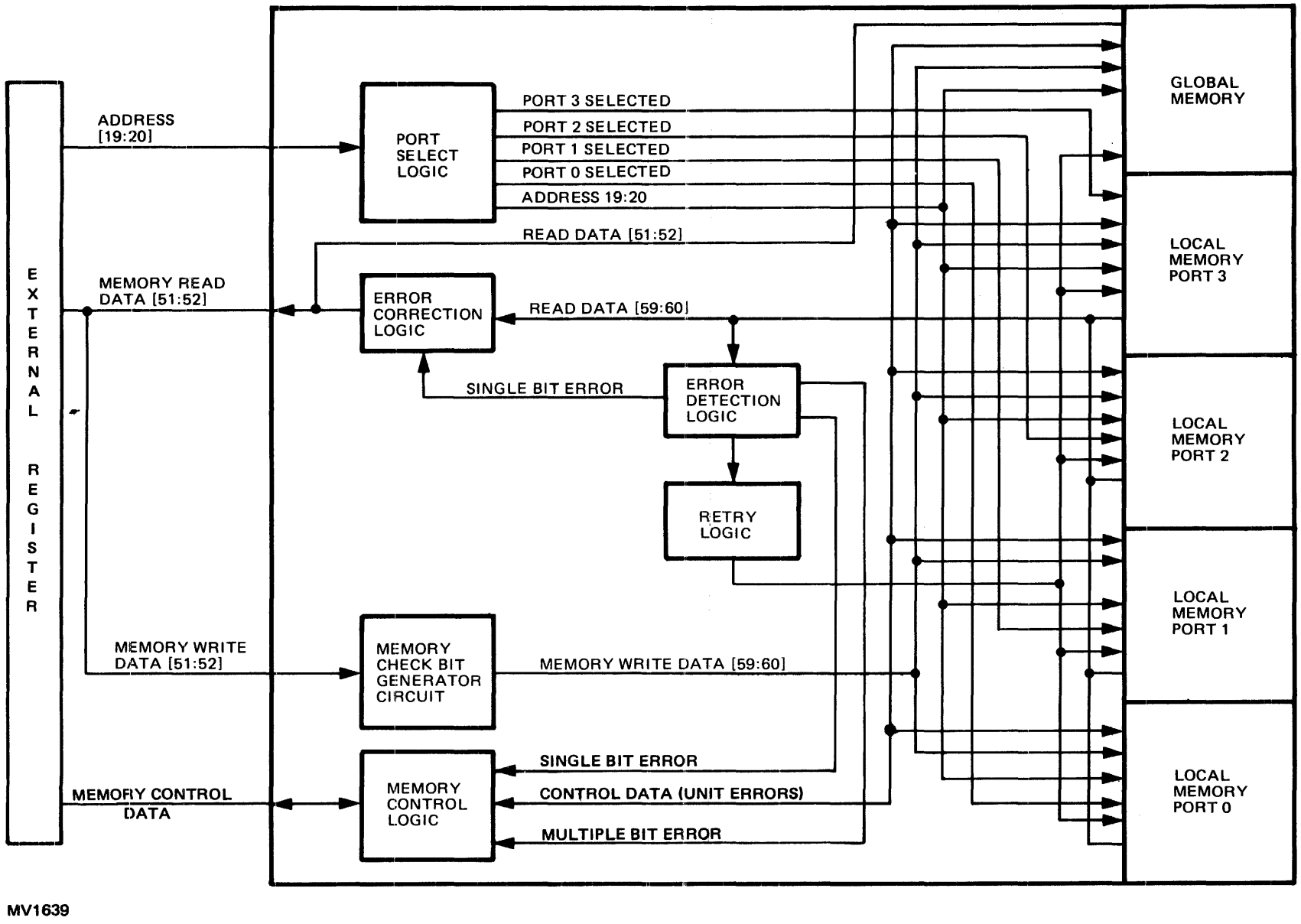


Figure 5-30. Channel B Functional Block Diagram

During memory read operations, the error detection check bits (which were written into memory during the memory write operation) are tested for bit errors in the data word received from the memory storage unit. If a single bit of a memory read data word is in error, the error correction circuit will correct the bit in error. If more than a single bit in the memory read data word is in error, the error is not correctable, but the error detection circuit will detect a multiple bit data error. All single bit and multiple bit data errors are reported to the data processor interrupt handling procedure, and are logged in the SYSTEM/SUMLOG.

Memory Retry

The memory control performs memory retry operations under certain conditions.

The memory control will perform a memory retry operation if the memory module detects a parity error in the address and control data that is transmitted from the CPU cabinet to the memory module cabinet over the port interface. This retry consists of performing the entire memory cycle over again. If the retry of the memory cycle is successful then the memory controller will cause the interrupt controller to make an entry in the SYSTEM/SUMLOG that indicates a retry operation occured, and the memory operation proceeds in a normal manner. If the retry operation is not successful (a second parity error is detected in the memory address and control data) then the memory cycle is aborted, and the memory controller causes an alarm interrupt to be recorded in the SYSTEM/SUMLOG. The procedure that caused the memory cycle which was aborted is terminated because of the memory parity error.

The memory control will also perform a retry operation if the memory control senses a parity error in the read data that is transmitted from the memory module cabinet to the CPU cabinet. This retry operation consists of causing the read data in the storage module read latches to be transmitted to the CPU cabinet a second time. A second memory cycle is not performed by the storage module. The results of successful retry operation is reported in the same way that a successful address and control retry is reported.

If the retry operation for a parity error in the read data is not successful, then an error correction memory cycle is initiated. The entries that are made in the SYSTEM/SUMLOG as a result of an error correction memory cycle were described previously in this section of this manual.

The memory control does not perform retry operations for parity errors in the write data transmitted from the CPU cabinet to the memory module cabinet.

Only one retry operation will be attempted for each memory operation.

Global Memory

The global memory application is undefined at the present time. This subject will be documented when available.

Scan Bus Operations

Figure 5-27 shows that external subsystems, such as the B 6800 data communications subsystem, are interfaced with the CPU through a separate scan bus interface. These same external subsystems are interfaced to memory through channel B of the memory control.

All external subsystems that are controlled by the B 6800 system, such as data communications subsystems and reader/sorter (BIC) subsystems, share the separate scan bus interface to the CPU. Thus, only one of the subsystems may receive or transmit information over the scan bus at a time. The identification of a subsystem for scan bus communications is indicated by the contents of the address lines on the scan bus interface (refer to figure 5-29). Only the subsystem addressed will respond to data on the scan bus, even though all subsystems connected to the scan bus receive the same information and address inputs. The subsystems connected to the scan bus do not initiate communications, they only respond to scan bus inputs from the data processor. Thus, there is never a conflict between the subsystems for use of the scan bus interface.

A scan bus operation is performed by the data processor when the Z12 bus bit C is true. When this bit is true the channel A logic routes the address and data to the external scan bus interface instead of to the memory port interface. The error detection/correction logic, and the retry logic of channel A are bypassed for scan bus operations. However, scan bus address parity errors, and scan bus data parity errors are reported to channel A of the memory control. These two types of scan bus errors are subsequently reported to the interrupt handling procedure, and logged in the SYSTEM/SUMLOG, in the same way that memory bus errors are reported and logged.

CHANNEL B MEMORY REQUESTOR

The channel B requestor of the memory control is used to interface external subsystems of the B 6800 system to the memory resources of the B 6800 system. A memory request from channel B is entirely separate from a request from channel A.

Several different subsystems may share the channel B requestor input to the memory exchange. Channel B does not contain logic circuits to queue requests from external subsystems, therefore, only one of the several possible external subsystems may use channel B requestor at any one time. Priority for the use of the channel B memory requestor, among the various external subsystems that share the requestor, is a requirement of the external subsystems, and not of the channel B requestor logic.

If more than two subsystems are connected to the channel B requestor, then an expansion module (shown in dotted lines in figure 5-27) must be used. An expansion module is essentially a 1 x 5 exchange, that allows five separate subsystems to be interfaced with the single channel B requestor interface port.

If an expansion module is used to interface external subsystems to the B 6800 system, then the expansion module must be mounted in an independently powered cabinet. This independently powered cabinet is not part of the B 6800 system.

MEMORY STORAGE UNIT PORT INTERFACE

There are two different types of port interfaces used to connect the memory control to the units that are remote from the CPU cabinet. The units that are remote from the CPU cabinet, the type of interface connection used for each type of unit, and the information that is transmitted on each cable of the interface are as follows:

Unit(s)	Type of Interface	Cables and Signals
DCP and/or BIC	Scan Bus 80 wire, four cable.	Each cable of the interface contains 20 wires, and each wire may be used to transmit one logic signal between the memory exchange and the external unit. Some of the wires are used to transmit signals in a single direction on the bus, and other wires are used to transmit signals bidirectionally on the bus.
	Cable Name	Signals on the Cable
	Address 19:20	A twenty bit address field which is transmitted in a single direction. The field provides for one million (binary) addresses to be available for use.
	Bus control signals, and information signals 51:12	Twelve bits of the word information (51:12), and eight bus control bits. The twelve wires for word information are bidirectional.

Unit(s)	Type of Interface	Cables and Signals
	Cable Name	Signals on the Cable
	Information signals 39:20	Twenty bits of the word information (39:20). The twenty information lines are used bidirectionally.
	Information signals 19:20	Twenty bits of the word information (19:20). The twenty information lines are used bidirectionally.

Unit(s)	Type of Interface	Cables and Signals
Local memory unit	Local Memory 264 wire, six cable.	Six cables are used to interface each of four possible memory storage units to the memory exchange. Each cable contains 44 wires which may be used to pass information, control, and address data between the storage unit, and the memory control port. All signal lines of the local memory interface bus are single direction lines, and no cable lines are used to pass data in both directions.
	Cable Name	Signals on the Cable
	1	This cable is used to pass a 16 bit address to the memory storage unit, and is also used to pass a three bit address check value from the storage unit back to the memory control. The other lines on this cable are not used.
	2	This cable is used to pass 12 control signals from the storage unit to the exchange port, or vice versa. The other wires of this cable are not used.
	3	This cable is used to pass 15 write data signals (14:15), and 15 read data signals (14:15) between the storage module and the exchange port. The other wires of this cable are not used.
	4	This cable is the same as cable number three, except that it passes write data bits (29:15) and read data bits (29:15).
	5	This cable is the same as cable number three, except that it passes write data bits (44:15), and read data bits (44:15).
	6	This cable is the same as cable number three, except that it passes write data bits (59:15), and read data bits (59:15).

Local Memory Port Interface Control Logic

An access request to one of the four local memory storage units may originate in channel A, or channel B of the memory control. Regardless of which channel originates an access request, the logic and control signals of the memory control port interface are the same. The logical control signals of the port interface (on cable number 2) are as follows:

Signal Name	Signal Usage
RMW, WCC, PED	These three signals; RMW (read modify write), WCC (write cycle control), PED (parity error disable), form a three bit code that is used to define the type of operation that is to be performed by the memory storage unit. The types of operations that are performed by the storage unit are as follows:

RMW	WCC	PED	Function
0	1	0	Clear write operation.
0	0	1	Memory read restore operation.
1	1	0	Read/Modify/Write.

IMC	The initiate memory cycle signal. Two IMC signals are required to perform read modify write memory operations. The memory control generates both of the IMC signals, (one for the read portion of the operation, followed by another one for the write portion of the operation) and transmits them both on the interface IMC wire. The timing of these two IMC signals is a function of the memory control.
PAR	The memory address parity bit. This signal is sent from the memory control to the memory storage unit, to cause the 17 bit address field, plus the RMW, WCC, and PED signals to have odd parity. If the number of binary one bits in the address field is even, the PAR signal will be true, thus making an odd number. If the number of binary one bits in the address field is odd, the PAR signal will be false, thus maintaining the odd parity. This signal is only transmitted during the clear write operation. For all other types of memory operations this signal is forced false.
MPE.	The memory parity even signal. This signal is returned from the memory storage unit to the memory control, to indicate whether or not memory address even parity error was detected at the storage unit interface.
WST	The write strobe signal. This signal is the write strobe signal for a memory write operation. The memory control generates this signal and transmits it to the memory storage unit which is to perform the write portion of a memory cycle. The system memory control must generate this signal instead of the memory storage unit because the write portion of a memory cycle is performed after a possible retry of the read portion is completed.
MSW	The memory select write signal. This signal is used to define whether the read register or the write register is to be used as the source of data for the write portion of a Read/Modify/Write operation. If the MSW signal is a true level the write register is the source, otherwise the read register is the source.

PCS (general clear) — The memory storage unit clear signal. This signal is generated in the memory control and is used to clear the logic circuits of the memory storage unit.

HAR — The hold address for return control signal. This signal is generated in the memory control, and transmitted to the memory storage unit to cause the storage unit to hold the memory address by using its address latch circuits. This signal is required in order to make it possible to single pulse a memory storage unit operation.

MAV. . — The memory available control level. This signal is generated in the memory storage unit, and a true level is transmitted to the memory control when the storage unit is powered up.

Scan Bus Port Interface Control Logic

Of the two requestor inputs, only channel A can cause a scan bus operation to be performed. A scan bus operation will be performed for the input from channel A when bit C of the Z12 bus is a binary one. The control signals that are generated for a scan bus operation, by the memory control and/or the external subsystem(s) are as follows:

SREQ — The scan bus request logic signal. The memory control causes the SREQ logic level to be true when bit C is true on the Z12 bus. This signal indicates to the subsystems connected to the scan bus that the CPU is processing a scan bus request. The subsystems that are connected to the scan bus are asynchronous with respect to the CPU, and the SREQ signal is used to cause a subsystem to synchronize with the CPU for the duration of the scan bus interface operation.

RQCW — The scan-out signal. The memory control causes the RQCW logic level to be true if the scan bus request from the CPU is the result of executing a SCAN-OUT operator in the data processor.

SAPL/STEX — The scan address parity level. During the initiation of a scan bus operation the SREQ and SWRC signals, plus a twenty bit address field is transmitted to the subsystem(s) connected to the scan bus interface. The number of binary one bits in these 22 signals must be an odd number. The SAPL signal is used to make the number of binary ones odd when the number of binary one bits is an even number. The subsystem that is addressed on the scan bus tests the number of binary ones, and if an even number of binary ones is detected, an address parity error is declared to exist.

The data word on the scan bus must also have an odd number of binary one bits. Control level SI51 is used to make the scan data word have an odd number of bits, in the same way that the SAPL level is used for the address field. If an even number of bits are detected in the scan data word, a transmission error is declared to exist.

The STEX control level is true if a scan address parity error, or a scan data transmission error exists. The SAPL/STEX control signals share a common interface wire on the scan bus interface.

SAOX — The scan access obtained logic signal. When the subsystem has accessed the area within the subsystem that is to be used for the scan bus request, and the area is interfaced to the scan bus in the subsystem, the SAOX control signal is made a true level. This signal indicates that the data to be transferred on the scan bus is available on the scan bus interface.

SRDY — The scan ready signal. When the CPU initiates a scan bus request the subsystem that is addressed makes its SRDY level true, if the subsystem is powered up, is not inhibited from performing a scan bus function, and is not performing an internal operation. The CPU of the B 6800 system does not allow the scan bus operation to delay other channel A functions in the memory exchange unnecessarily. When a scan bus operation is initiated, a counter in the CPU counts system clock periods until the SRDY level goes true. If seven clock periods pass, and the SRDY level does not go true, a "time out" occurs, and the scan bus operation is aborted. If SRDY goes true, the clock period counter in the CPU is inhibited from counting. Thus, a subsystem must respond within seven clock periods, or the scan bus operation will be terminated.

SI51 — The scan bus word parity bit. This control signal is used to cause the scan bus data word to have odd parity, in the same way that the SAPL signal is used to cause the address field to have odd parity.

MEMORY TESTER LOGIC

The B 6800 has memory test logic designed into the hardware circuits of the CPU cabinet. A separate memory tester, with access to local memory, is not provided. Therefore, when memory tests are to be performed, their execution will preempt any other system operation. The memory test logic does not use, and cannot test, the channel B logic of the memory control input requesters.

The memory tester logic is designed to be used with memory test routines that are resident on magnetic tape. The MDP controls the magnetic tape peripheral device. Memory tests are executed on the B 6800 system through messages on the system operators console (SPO), under control of the MDP Executive routine. Thus, memory testing is only performed by system operators, who must mount the memory test tape upon the peripheral device, and then direct the system to perform memory tests.

SECTION 6

PROGRAM OPERATORS

GENERAL

The machine language operators are composed of syllables in a program string. The operators are divided into four major classes, which are primary mode, variant mode, edit mode, and vector mode operators.

SYLLABLE ADDRESSING AND SYLLABLE IDENTIFICATION

SYLLABLE FORMAT AND ADDRESSING

A machine language program is a string of syllables which are normally executed sequentially. Each program word in memory contains six eight-bit syllables. The first syllable of a program word is labeled syllable zero, and is formed by bits 47 through 40 (see figure 6-1).

P AND T REGISTERS

The P register contains the currently active program word. The T registers are the control (instruction) registers. There is one four-bit T register for each operator family. The T register contains the code for the specific type of operator that is to be executed by the family, and is usually derived from the four low-order bits of the operator syllable code. The four high-order bits of the operator syllable code are used to select a family strobe. This family strobe is used to define which family is to receive the strobe pulse (execute pulse). Figure 6-2 shows how a program operator code in the P register is decoded to select a family strobe, and a T register value. In the example shown in figure 6-2, a divide operator (OP code 83 hexadecimal) is in the process of being executed, and this operator caused the family A strobe (STRA) to be selected. The family A T register contains a value of three (hexadecimal) which is derived from the four low-order bits of the operator code.

Figure 6-2 also shows an example of how a word of program code is selected to be executed. The addressing mechanism for program code words, and the way that the controllers of the B 6800 data processor function to provide automatic program code handling operation is also shown in this example.

In the program code handling example shown in figure 6-2, the program base register (PBR) points at the first word of program code in the current program code segment. The value of the PBR register is initially established from the segment descriptor for the current program segment, when the procedure is initiated.

The current word of program code in a program segment that is presently being executed is indicated by the value of the program index register (PIR). The initial value of the PIR register for a program segment is established from the PCW word that caused the segment to be executed. The initial value of PIR may also be established from an RCW, if the program segment is executed as the result of an exit or return from another code segment in the same program.

The first syllable that is to be executed in a program code segment is derived from the PCW (or alternatively the RCW) that caused entry into the current program segment. In the example shown in figure 6-2, the PSR register is pointing at syllable four of the P register because the divide operator (in syllable three) is being executed, and the PSR plus one logic has advanced the value of the PSR register to point at the next syllable that will be executed.

Program code words, in the B 6800 system, are normally fetched from system memory by the look ahead logic. The look ahead logic fetches the next word of program code while the current word of program code is being executed, and places it in the L register. When the PSR register indicates by its content value that all of the syllables of program code

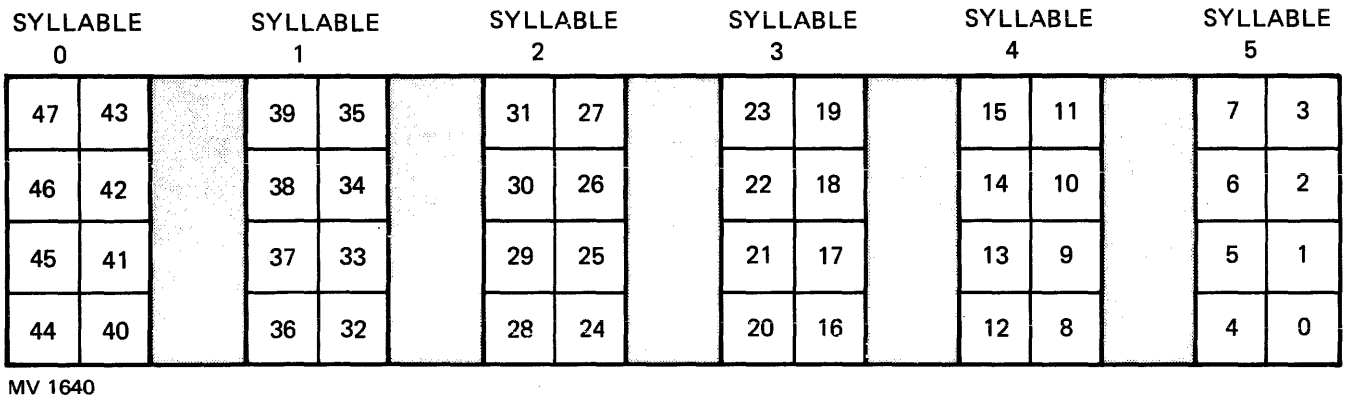


Figure 6-1. Program Word

in the P register have been executed, the program controller causes the next word of program code to be transferred from the L register, to the P register. The PSR register will point at the first syllable in the new program word.

When the next word of program code is transferred from the look ahead logic L register to the P register, the look ahead module causes the next word of program code to be fetched from memory, and placed in the emptied L register. The program controller will cause the value of the PIR register to be incremented by one, as the operators are strobed from the P register. Thus, the PIR register will always point at the code word that the present operator started in. The look ahead logic uses the look ahead address register (LAR) to address the next word of program code. The LAR register has an automatic plus one incrementation feature, that causes the LAR register to always point at the memory address of the next program word (following the program word that is present in the L register).

The dotted lines in figure 6-2 are used to show the origin of a word of program code in the P and L registers, and also to show what word of the program segment is pointed at by an address register. A dotted line is also used to show that the value of the PSR register temporarily points at syllable four, when syllable three is being executed by the data processor.

OPERATION TYPES

Operations are grouped into three classes: name call, value call, and operators. The two high-order bits (bits 7 and 6) determine whether a syllable begins a value call, name call or operator (figure 6-3).

Name Call

Name call builds an indirect reference word in the stack (see figure 6-4). Stack adjustment takes place so that the A register is empty. The six low-order bits of the first syllable of this operator are concatenated with the eight bits of the following syllable to form a 14-bit address couple. The address couple is placed, right-justified, into the A register, with the remainder of the A register filled with 0's. The TAG field of the A register is set to 001 and the register is marked full.

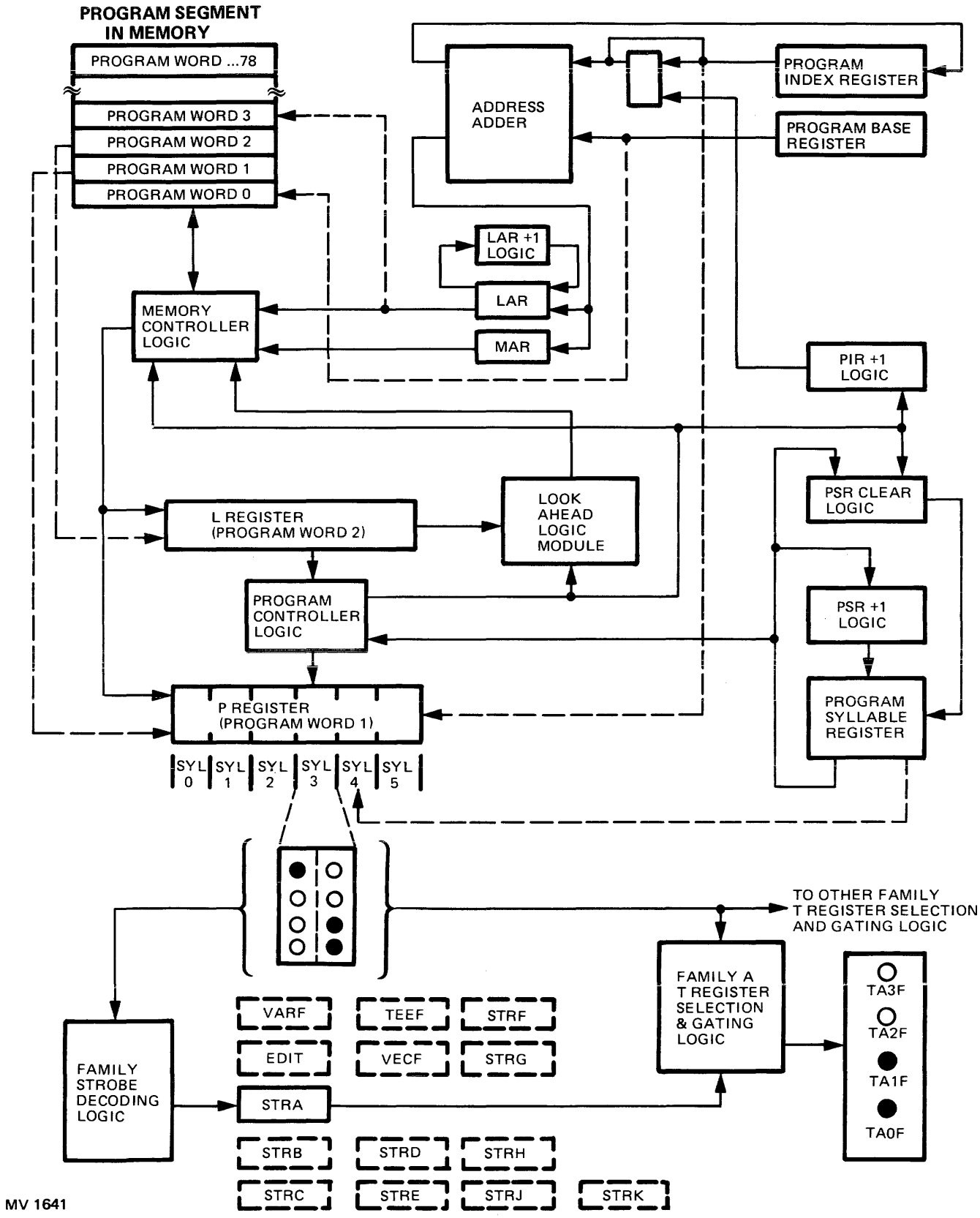


Figure 6-2. Program Word, Syllable Addressing

(BITS 7 AND 6) IDENT	SYLLABLE TYPE	NO. OF SYLLABLES	FUNCTION
00	VALUE CALL	2	BRINGS AN OPERAND INTO THE STACK
01	NAME CALL	2	BUILDS AN IRW IN THE STACK
OTHER THAN ABOVE	OTHER OPERATORS	1→7	PERFORMS THE SPECIFIED OPERATION

MV 1642

Figure 6-3. Primary Mode Operator Syllable Decode Table

Value Call

Value call loads into the top of the stack the operand referenced by the address couple. The operator is formed in the same manner as in the name call operator. If the referenced memory location is an indirect reference word or a data descriptor, additional memory accesses are made until the operand is located. The operand is then placed in the top of stack registers. The operand may be either single- or double-precision, causing either one or two words to be loaded into the top of the stack.

Figure 6-5 is an example of how a value call operator (VALC) is used to cause a word of data located at memory address D2 plus 4 to be fetched and placed in the top of the D3 stack. The current stack is known to begin at the MSCW pointed at by the D3 display IC memory register, because the lexicographical level register contains a value of 3 (LL00, LL01, LL02/, LL03/, LL04/).

The fence decoding logic defines the number of bits in the address couple that select a display register to provide the base address portion of the value call operation. The fence decoding logic uses the current programming level of the program segment to determine which IC memory display register is selected. The highest order bit of the lexicographical level register that is true in the example is bit LL01, which has a value or two. The fence decoding logic will therefore use the two high order bits of the address couple to select an IC memory display register as the source of the base address. The bits that are not used by the fence decoding logic, to select a display register, form the index portion of the value call operation.

Bits 29:5 are used by the fence decoding logic to select a display register. The value of the bits in this field are opposite to the word bit number order. That is, bit 29 of the address couple in the example has a binary value of one, and bit 25 has a binary value of sixteen. The following table equates bits 29:5 to a decimal value, and to the display registers which they will select.

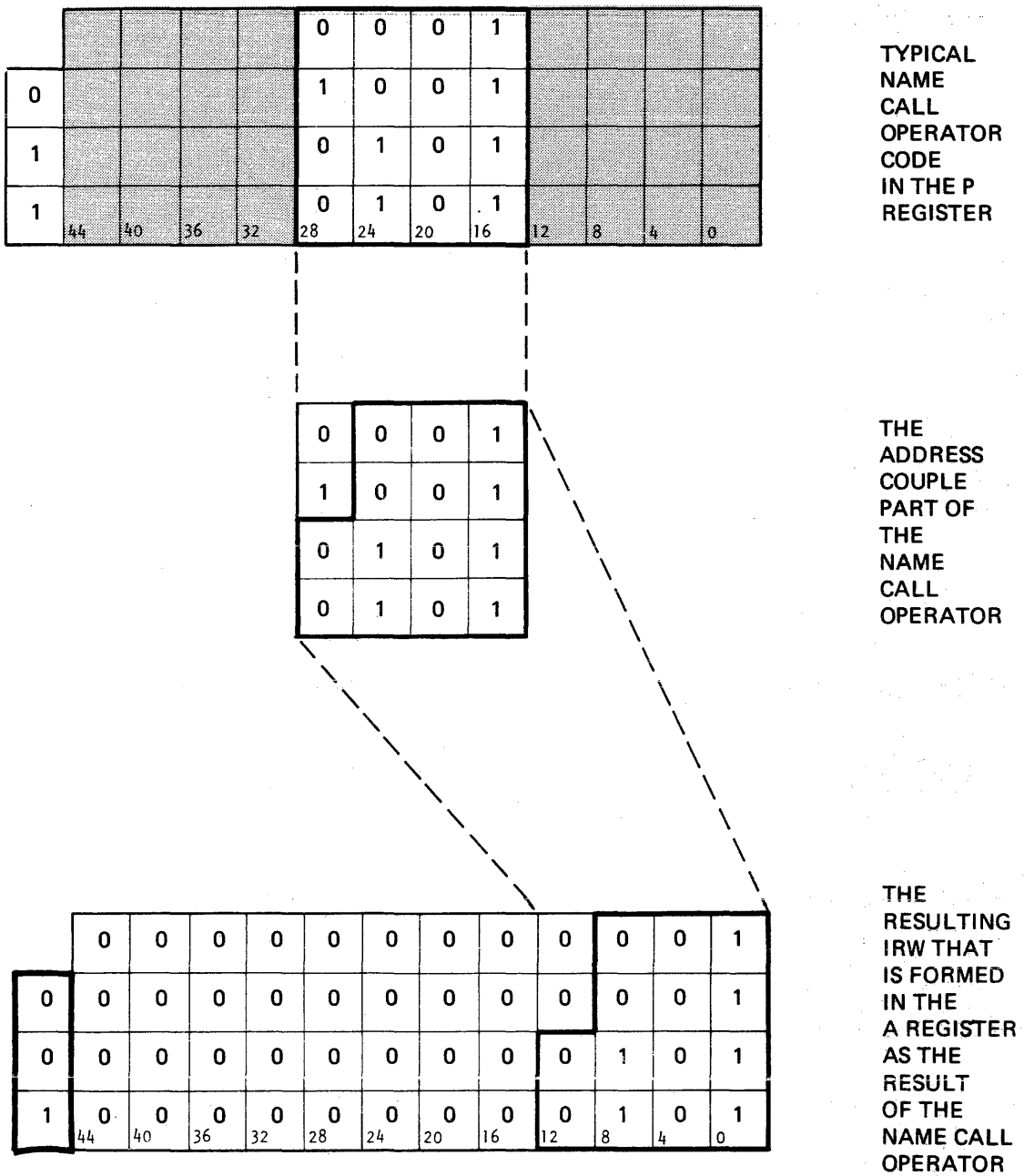


Figure 6-4. Name Call Operator Function

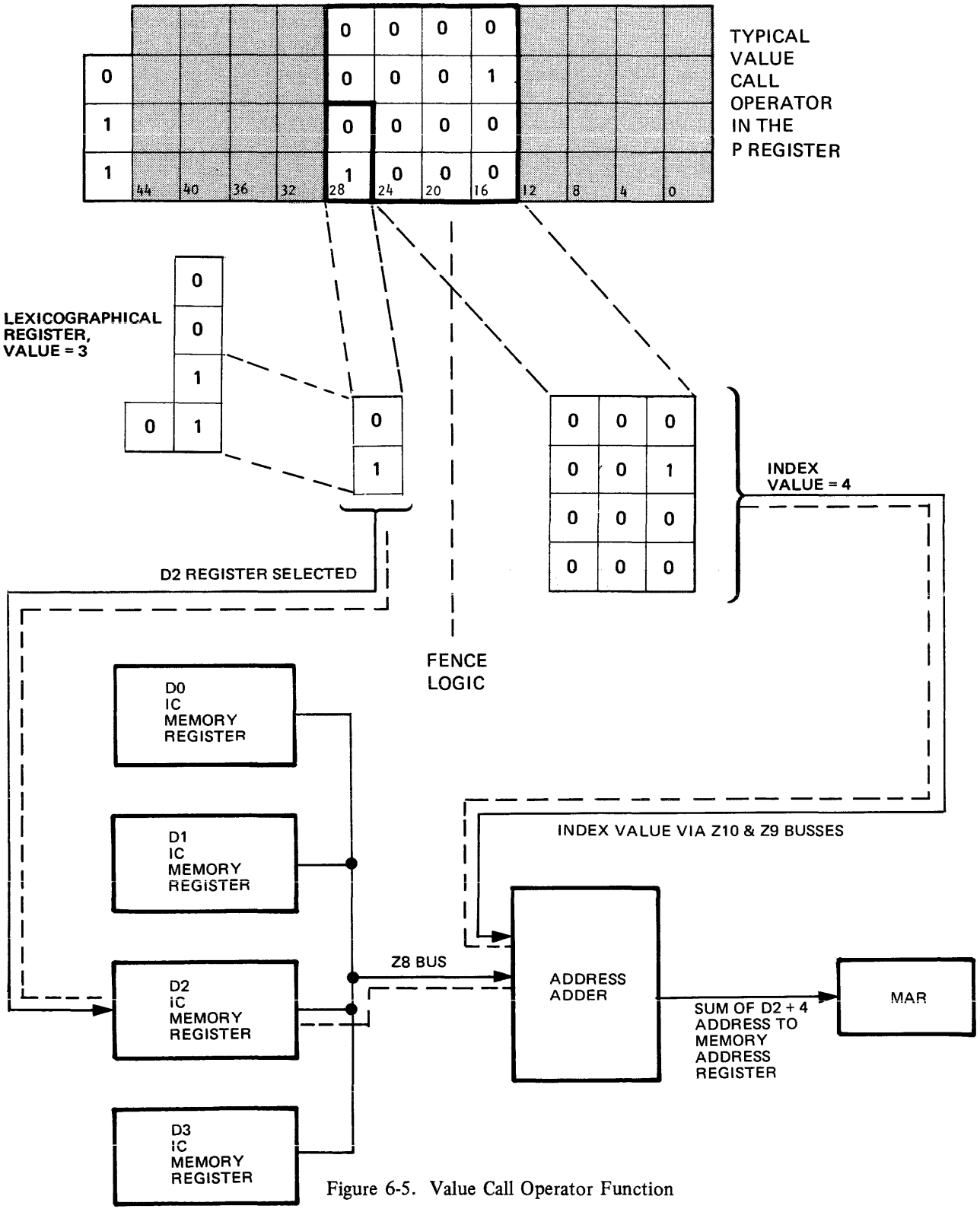


Figure 6-5. Value Call Operator Function

MV 1644

Bit Number	Decimal Value	Display Register Selected
29	1	1
28	2	2
27	4	4
26	8	8
25	16	16

There are 32 IC memory display registers that may be selected by the fence decoding logic.

In the following example, it is possible to see how bit 28 is used to select display register two and thus to provide the base portion of the value call address.

The index portion of the address couple is treated in the conventional manner, as a binary value. In the example shown in figure 6-5, bits 16, 17, and 18 have a binary value of 100, which is four decimal.

The absolute memory address that is placed in the memory address register, in the example in figure 6-5, is the sum of the address from display register two, and the index, which has a value of four (that is, D2 + 4). The word of data in memory at the absolute memory address will be fetched, and placed in the top of stack register. If the word at D2+4 is an IRW, or a data descriptor, then additional fetches from memory will be made. This process will continue until an operand, or a data word is placed in the top of stack register. Placing an operand or a data word in the top of stack register completes the value call operation.

The value call operator detects an invalid operand error condition if a word with a tag code of three, four, or six is referenced. If a word with a tag of seven is referenced by a value call operator, an accidental program entry into the procedure described by the PCW is performed. The final value that is placed in the stack by a value call operator must have a tag field of zero, or two.

An accidental program entry that is caused by a value call operator being executed is treated like a sub-routine of the procedure that executed the value call operator. The stack of the procedure is marked by an MSCW and an RCW. Then the sub-routine referenced by the PCW is executed, and terminates by means of a return operator. The return operator passes a parameter from the sub-routine to the procedure that executed the original value call operator. The program flow of the procedure is resumed at the next operator in sequence following the original value call operator.

Operators

Operators vary from one to seven syllables in length. The first syllable of each operator determines the number of additional syllables forming the operator. Upon completion of each operator, the PSR register addresses the first syllable beyond all of the syllables comprising the operator.

Operators work on data as either full words (48 data bits plus tag bits), or as strings of data characters. Word operators work with operands (single- or double-precision) in the top of the stack.

String operators are used for transferring, comparing, scanning, and translating strings of digits, characters, or bytes. In addition, a set of micro-operators provide a means of formatting data for input/output.

The string operators use source and destination pointers which are located in the stack. These pointers are set into the following hardware registers:

1. Source Base Register – (SBR).
2. Source Index Register – (SIR).
3. Source Index Byte Register – (SIB).
4. Source Size Register – (SSZ).
5. Destination Base Register – (DBR).
6. Destination Index Register – (DIR).
7. Destination Index Byte Register – (DIB).
8. Destination Size Register – (DSZ).

In some of the string operators the source pointer may not be used. In this case, an operand may be in the stack; its characters are circulated as it is being used.

String operators have an optional update function, that is, producing updated source and destination pointers, and count. At completion of an operation the source and destination pointers are updated as follows:

1. If the source is an operand it remains in the stack.
2. If the pointer is a descriptor, the word index fields and byte index fields are updated from SIR/DIR and SIB/DIB. The string size fields are updated from SSZ/DSZ.
3. If the pointer is a data descriptor or a non-indexed string descriptor, it is converted to an indexed string descriptor and updated.

If both the source and destination descriptors have size fields equal to 0, the size registers indicate 8-bit character size. When both a source and destination are required and the size field of one is equal to 0 and the other is not, then the size field of the non-zero descriptor is used.

If neither size field is equal to 0 and the size fields are not equal and the operator is not translate, the invalid operand interrupt is set and the operator is terminated. The size field is considered equal to 0 when the source is an operand.

SECTION 7

PRIMARY MODE OPERATORS

GENERAL

This section defines the functions of the primary operators. In each case, the name of the operator, corresponding mnemonic, and hexadecimal code are shown. Appendix A of this manual lists the operators in alphabetic order, and appendix B lists the operators in numeric order, by mode.

The universal operators are also included in this section.

ARITHMETIC OPERATORS

The arithmetic operators usually require two operands in the top of stack registers. These operands are combined by the arithmetic process specified with the result placed in the top of the stack. The operands may be either single-precision, double-precision, or intermixed. The specified arithmetic process adapts automatically to the data environment, with the single-precision process invoked if both operands are of the single-precision type and the double-precision process invoked if either operand is of the double-precision type.

Each double-precision operand occupies two words. The second word of the operand is an extension of the first word of the operand. The mantissa of the first word of the operand contains unit values, and the mantissa of the second word contains a fractional unit value. An implied octal point separates the mantissa of the first word from the mantissa of the second word. When the top of stack registers are full, the first word of the first operand is in the A register; the second word of the first operand occupies the X register. The first word of the second operand resides in the B register; the second word of the second operand occupies the Y register. Therefore, double-precision arithmetic processes operate on four words in the stack, instead of two as in single-precision operations. Double-precision arithmetic leaves a two-word result in the top of the stack.

Add, subtract, and multiply operations which use two integer operands yield an integer result if no overflow occurs. If one or both operands are non-integer, or if the result generates an overflow, the result is non-integer.

When an operator has been entered, the hardware stack-adjust function fills or empties the top of stack register as required by the operator. If either register contains an incorrect word, the operator is terminated by an invalid operand interrupt.

ADD (ADD) 80

The operands in the A register and the B register are added algebraically, with the sum left in the B register. At the end of the operation, the A register is marked empty, and the B register is marked full.

If only one of the operands is double-precision, the register (X or Y) associated with the register that contains the single-precision operand is set to all 0's. The B register is marked as a double-precision operand at completion of the operation.

If the mantissa signs and the exponents are equal, the mantissas are added and the sum placed in the B register. If the sum exceeds 13 single precision (26 double precision) octal digits, the mantissa of the sum is shifted right one octade, rounded, and the exponent is algebraically increased by one. The meaning of exponents and mantissas were given in section 2 of this manual.

If the exponents are equal but the mantissa signs are unequal, the difference of the mantissas plus the appropriate sign are placed in the B register.

If the exponents are unequal, the operands are first aligned. If the alignment causes the smaller operand to be shifted right 14 single precision (27 double precision) octal places, the larger operand is the result.

If the alignment causes the smaller operand to be shifted right, but less than 14 single precision (27 double precision) octal places, the digits of the smaller operand shifted out of the register are saved and used to obtain the rounded result.

If the signs of the operands are equal, the mantissas are added and the sum placed in the B register. If the sum does not exceed 13 single precision (26 double precision) octal digits, the last digit shifted out of the register is used to round the result. If the sum is 14 single precision (27 double precision) octades, the mantissa in B (Y) is rounded to 13 single precision (26 double precision) digits.

If the signs of the operands are unequal, an internal subtraction takes place, with the rounded result placed in the B register.

If the result has an exponent greater than +63 (+32,767), the exponent overflow interrupt is set. If the result has an exponent less than -63 (-32,767), the exponent underflow interrupt is set.

SUBTRACT (SUBT) 81

The operand in the A register is algebraically subtracted from the operand in the B register with the difference left in the B register. The operation is the same as for the Add operator except for initial sign comparisons.

MULTIPLY (MULT) 82

The operand in the A register is algebraically multiplied by the operand in the B register. The rounded product is left in the B register.

If the mantissa of either operand is 0, the B register is set to all 0's.

If both mantissas are non-zero, the product of the mantissa is computed. If the product contains more than 13 single precision (or 26 double precision) digits, it is normalized and rounded to 13 single precision (or 26 double precision) digits. A mantissa of all sevens is not rounded. Normalization was explained in section 2 of this manual.

If the result has an exponent greater than +63 (+32,767), an exponent overflow interrupt is set. If the result has an exponent less than -63 (-32,767), an exponent underflow interrupt is set.

EXTENDED MULTIPLY (MULX) 8F

The operands in the A and B registers are algebraically multiplied and a double-precision product is placed in the B and Y registers. The A register is marked empty and the B register marked full.

The actions outlined for Multiply operations also apply to this operator.

If either or both operands are double-precision, then a normal double-precision operation occurs.

DIVIDE (DIVD) 83

The operand in the B register is algebraically divided by the operand in the A register, with the quotient left in the B register. After the operation the A register is marked empty, and the B register is marked full.

If the mantissa of the B register is 0, the B register is set to all 0's. If the A register mantissa is equal to 0, the divide by zero interrupt is set. In either case the operation is terminated.

If the mantissa of both operands are non-zero, they are normalized and the operand in the B register is divided by the operand in the A register. The quotient is developed to 14 single precision (or 27 double precision) digits, rounded to 13 single precision (or 26 double precision) digits, and remains in the B register.

If the result has an exponent greater than +63 (32,767) the exponent overflow interrupt is set. If the result has an exponent less than -63 (-32,767) the exponent underflow interrupt is set.

INTEGER DIVIDE (IDIV) 84

The operand in the B register is algebraically divided by the operand in the A register and the integer part of the quotient is left in the B register. After the operation the A register is marked empty and the B register is marked full.

If the mantissa of the B register is 0, the B register is set to all 0's. If the mantissa of the A register is 0, the divide-by-zero interrupt is set. The operation is terminated in either case.

If the mantissas of both operands are non-zero, they are normalized. If the exponent of the B register is algebraically less than the exponent of the A register after both operands have been normalized, the B register is set to all 0's. If the exponent of the B register is algebraically equal to or greater than the exponent of the A register, the divide operation proceeds until an integer quotient or a quotient of 13 single precision (or 26 double precision) significant digits is calculated.

If an integer quotient is developed, the quotient is left in the B register with a 0 exponent for single precision and the exponent set to 13 for double precision. If a non-integer quotient is developed, the integer overflow interrupt is set.

REMAINDER DIVIDE (RDIV) 85

The operand in the B register is algebraically divided by the operand in the A register to develop an integer quotient. The remainder of this Division stays in the B register.

If the mantissa of the B register is 0, the B register is set to all 0's. If the mantissa of the A register is 0, the divide by zero interrupt is set. In either case the operation is terminated.

If both mantissas are non-zero, both operands are normalized. If the exponent of the B register is algebraically less than the exponent of the A register after both operands have been normalized, the operand in the B register is the result. If the exponent of the B register is algebraically equal to or greater than the exponent in the A register, the divide operation proceeds until an integer quotient is developed; the remainder is then placed in the B register.

If a non-integer quotient is developed, the integer overflow interrupt is set and the operation is terminated.

INTEGERIZE, TRUNCATED (NTIA) 86

The operand in the B register is converted to integer form without rounding and remains in the B register.

If the operand in the B register cannot be integerized, i.e., the exponent is greater than the number of leading zeros in the operand, the integer overflow interrupt is set and the operation is terminated.

INTEGERIZE, ROUNDED (NTGR) 87

The operand in the B register is converted to integer form. Rounding takes place if the absolute value of the fraction is greater than four. The rounded result is left in the B register.

If the operand in the B register cannot be integerized, i.e., the exponent is greater than the number of the leading zeros in the operand, the integer overflow interrupt is set and the operation is terminated.

The operand is rounded, if necessary, by adding one to the mantissa. If a non-integer results from this operation, the integer overflow interrupt is set.

TYPE-TRANSFER OPERATORS

SET TO SINGLE-PRECISION, TRUNCATED (SNGT) CC

The operand in the top-of-stack register is normalized and set to a single-precision operand; or in the case of a data descriptor, the double-precision bit is set to 0.

If the word in the top-of-stack register is a non-indexed, double-precision data descriptor, the double-precision bit is cleared to 0 and the length field multiplied by two.

If the double-precision operand in the top-of-stack register has an exponent greater than +63 after normalization, the exponent overflow interrupt is set. If the exponent is less than -63 after normalization, the exponent underflow interrupt is set, and the operation is terminated.

If the operand in the top-of-stack register is a double-precision operand with an exponent less than +63 or greater than -63; the operand is normalized, and the tag field in the top-of-stack register is set to single-precision.

If the word in the top-of-stack register is neither an operand nor a Data Descriptor, the invalid operand interrupt is set and the operation terminated.

If the operand is single-precision, it is normalized and the operation is terminated.

SET TO SINGLE-PRECISION, ROUNDED (SNGL) CD

The operand in the top-of-stack register is changed to a rounded, single-precision operand.

If the double precision operand in the top-of-stack register has an exponent greater than +63 the exponent overflow interrupt is set. If the exponent is less than -63, the exponent underflow interrupt is set. In either case, the operation is terminated.

If the operand in the top-of-stack register is a double-precision operand with an exponent less than +63 or greater than -63, the operand is normalized; the tag field in the top-of-stack register is set to single-precision, the operand in the top-of-stack register is rounded from the Y register, and the Y register is set to all 0's.

If a carry is developed during the rounding operation, the operand is adjusted and the new exponent is checked in the manner discussed in the preceding paragraph.

If the operand is a single-precision operand, it is normalized and no rounding occurs.

SET TO DOUBLE-PRECISION (XTND) CE

The word in the top-of-stack register is set to a double-precision operand and the Y register is set to all 0's. If a single precision data descriptor is present in the top-of-stack register, the double precision bit is set to one.

If the word in the top-of-stack register is a data descriptor with both the index bit and double-precision bit 0, the double-precision bit is set to one and the length field is divided by two.

If the operand in the top-of-stack register is a double-precision operand, the operation is complete. If it is a single-precision operand, the tag field in the top-of-stack register is set to double-precision and the Y is set to all 0's.

If the word in the top-of-stack register is neither an operand nor a Data Descriptor, the invalid operand interrupt is set and the operation terminated.

LOGICAL OPERATORS

For LAND, LOR, or LEQV, if only one of the operands is in double-precision form, the other operand is treated as double-precision with the least significant 13 octades equal to all 0's.

LOGICAL AND (LAND) 90

Each bit of the B operand result, except for the tag bits, is set to one where a one appears in the corresponding bit positions in both the A operand and the B operand. The other information bits of the B operand result are set to 0. If the tags of the two operands are identical the tag in the result is that of the B register. If the tags are different, the resultant tag is double precision.

LOGICAL OR (LOR) 91

Each bit position of the B operand except for the tag bits, is set to one if the corresponding bit position in either the A operand or the B operand is one, otherwise, the bit is set to 0. The tag bits are set to the value of the second item in the stack except when the A operand is double-precision; in which case, the B register tag is set to double-precision.

LOGICAL NEGATE (LNOT) 92

Each bit in the top word in the stack is complemented except for the tag bits, which remain unchanged. The result is always stored in the A register.

LOGICAL EQUIVALENCE (LEQV) 93

Each bit of the B operand is set to one, except for the tag bits, when the corresponding bits of the A operand and the B operand are equal. Each bit of the B operand is set to 0 except for the tag bits, when the corresponding bits of the A and B operands are not equal. The tag field is normally set to the value of the second item in the stack except when the A operand is double-precision; in that case, the B-register tag is set to double-precision.

RELATIONAL OPERATORS

The relational operators perform an algebraic comparison on the operands in the A register and the B register. The single precision result is left in the B register and the B register is marked full. The result is an operand in integer form with the value one if the relationship has been met or an operand with all information bits set to 0 if the relationship was not met. All relational operations compare the B operand to the A operand.

For all relational operators except equal (EQUL) and not equal (NEQL) the compare flip-flop is set when the relation is equal. For the equal or not equal operators, the compare flip-flop is set when the relationship is greater than equal.

The CMPF flip-flop is used in conjunction with the low order bit of the B register (BR[0:1]) to analyze the result of a relational operation. The following table shows the states of the CMPF flip-flop and BR[0:1] for various relational operations and possible results of relational operations.

Table 7-1. Relational Operator Indications

Relational	BR[0:1]	CMPF	Comparison Result
EQUAL	0	0	Less than
(8C) (EQUL)	0	1	Greater than
	1	0	Equal
	1	1	Not applicable
GREATER THAN	0	0	Less than
(8A) (GRTR)	0	1	Equal
	1	0	Greater than
	1	1	Not applicable
GREATER THAN	0	0	Less than
OR EQUAL	0	1	Not applicable
(89) (GREQ)	1	0	Greater than
	1	1	Equal
LESS THAN	0	0	Greater than
(88) (LESS)	0	1	Equal
	1	0	Less than
	1	1	Not applicable
LESS THAN	0	0	Greater than
OR EQUAL	0	1	Not applicable
(8B) (LESQ)	1	0	Less than
	1	1	Equal
NOT EQUAL	0	0	Equal
(8D) (NEQL)	0	1	Not applicable
	1	0	Less than
	1	1	Greater than

GREATER THAN (GRTR) 8A

If the B operand is algebraically greater than the A operand, the B register is set to one, otherwise, the B register is set to 0. AROF is reset, and BROF is set.

If the result of the algebraic comparison is "equal", the CMPF flip flop is set.

GREATER THAN OR EQUAL (GREQ) 89

If the B operand is algebraically greater than or equal to the A operand, the B register is set to one, otherwise, the B register is set to 0.

If the result of the algebraic comparison is "equal" the CMPF flip-flop is set. AROF is reset, and BROF is set.

EQUAL (EQUL) 8C

If the operands in the B and A registers are algebraically equal, the B register is set to one, otherwise, the B register is set to 0.

If the result of the algebraic comparison is "greater" the CMPF flip-flop is set. AROF is reset, and BROF is set.

LESS THAN OR EQUAL (LSEQ) 8B

If the B operand is algebraically less than or equal to the operand in the A register, the B register is set to one, otherwise, the B register is set to 0.

If the result of the algebraic comparison is "equal" the CMPF flip-flop is set. AROF is reset, and BROF is set.

LESS THAN (LESS) 88

If the operand in the B register is algebraically less than the operand in the A register, the B register is set to one, otherwise, the B register is set to 0.

If the result of the algebraic comparison is "equal" the CMPF flip-flop is set. AROF is reset, and BROF is set.

NOT EQUAL (NEQL) 8D

If the operand in the B register is not algebraically equal to the operand in the A register, the B register is set to one, otherwise, the B register is cleared.

If the result of the algebraic comparison is "greater than" the CMPF flip-flop is set. AROF is reset, and BROF is set.

BRANCH OPERATORS

Branch instructions break the normal sequence of serial instruction fetches. Branching may be either relative to the base address of the current program segment or to a location in another program segment. Branch operators may be conditional or unconditional.

BRANCH FALSE (BRFL) AO

If the low order bit of the A register is 0, the Program Index Register (PIR) and Program Syllable Register (PSR) are set from the next two syllables in the program string. Otherwise, PSR is advanced two syllable positions, and PIR is incremented if necessary.

The two syllables following the actual operator syllable form the new PIR and PSR settings as follows. The three high order bits are placed into PSR and the next 13 low order bits are placed in the PIR. The Program Register (P) is marked empty to cause an access to the new program word.

BRANCH TRUE (BRTR) A1

If the low order bit of the A register is one, the PIR and PSR are set from the next two syllables in the program string. Otherwise, PSR is advanced two syllable positions and PIR is incremented if necessary. The Branch True Operator uses the two syllables as described for the Branch False operator (BRFL), above.

BRANCH UNCONDITIONAL (BRUN) A2

The PIR and PSR are set from the next two syllables of the program string. The Branch Unconditional operator uses the two syllables as described for the Branch False operator (BRFL).

DYNAMIC BRANCH FALSE (DBFL) A8

If the low order bit of the B register is 0 and the word in the A register is a Program Control Word (PCW) or an indirect reference to one, a branch is made to the specified syllable of that program segment.

If the low order bit of the B register is 0 and the word in the A register is an operand, PIR and PSR are set from this operand.

If the word in the A register is an operand, it is used in the following manner. The operand is made into an integer. If it is negative or is greater than 16,384, the invalid index interrupt is set and the operation is terminated. If bit zero of the operand is 0, PSR is set to 0, otherwise PSR is set to 011. The next higher order 20 bits are placed in the PIR. The Program Register is then marked empty to cause access to the new program word.

DYNAMIC BRANCH TRUE (DBTR) A9

If the low order bit of the B register is one and the word in the A register is a PCW, or an indirect reference to one, a branch is made to the specified syllable of the program segment.

If the low order bit of the B register is one and the word in the A register is an operand, PIR and PSR are set from this operand.

The operand in the A register is used in this operator in the manner described for the Dynamic Branch False operator (DBFL).

DYNAMIC BRANCH UNCONDITIONAL (DBUN) AA

If the word in the A register is a PCW or an indirect reference to one, a branch is made to the specified syllable of the program segment.

If the word in the A register is an operand, PIR and PSR are set from this operand.

The operand in the A register is used in this operator in the same manner described for the Dynamic Branch False operator (DBFL).

STEP AND BRANCH (STBR) A4

The increment field of the step-index word (SIW) addressed by the contents of the A register is added to its current-value field. If the current-value field is then greater than the final-value field, the PIR and PSR are set from the next two syllables in the program string. Otherwise, the PIR and the PSR are advanced three syllables. The SIW is replaced in memory.

If no SIW is in memory, and if an operand is found, it is left in the stack. The A register is set to all 0's, the PIR and PSR are advanced and the next operator is executed. If no operand is encountered, the invalid operand interrupt is set.

UNIVERSAL OPERATORS

NO OPERATION (NOOP) FE

No operation takes place when this syllable is encountered. PIR and PSR are advanced to the next operator. This operator is also valid in the Variant and Edit modes.

CONDITIONAL HALT (HALT) DF

This operator halts the processor if the CHLT pushbutton on the MDP keyboard is illuminated. If the CHLT pushbutton is extinguished the operator is treated as a NOOP. This operator is also valid in the Variant and Edit modes.

INVALID OPERATOR (NVLD) FF

This operator sets the invalid operand interrupt. This operator is also valid in Variant and Edit modes.

STORE OPERATORS

The store operators use the words in the A register and B register. The operand in the B register is stored in memory at the location addressed by an Indirect Reference Word (IRW) or a Data Descriptor. If the A register contains an operand, a hardware interchange takes place so that the operand is transferred to the B register.

STORE DESTRUCTIVE (STOD) B8

If the word in the A register is an operand, the A and B operands are interchanged. The Data Descriptor or IRW in the A register is the address in memory where the operand in the B register (B, Y registers for double-precision) is stored. After the operand is stored, the A register and B register are marked empty and the operation is complete.

If the word addressed by the IRW is a Program Control Word, accidental procedure entry occurs. The spontaneously created Return Control Word (RCW) causes the Store Destructive (STOD) operator to be re-executed upon return from the procedure.

If the word addressed by the Data Descriptor has the memory protect bit on (bit 48), the memory protect interrupt is set and the operation is terminated.

If the presence bit in the Data Descriptor is 0, the presence bit interrupt is set. After the information has been made present, the operation is restarted.

STORE NON-DESTRUCTIVE (STON) B9

This operator functions in virtually the same way as the STOD operator, however, at the completion of this operator, the BROF remains set, and the operand is retained in the B register.

OVERWRITE DESTRUCTIVE (OVRD) BA

This operator functions in the same way as the STOD operator, except that the OVRD operator overrides memory protection checks.

OVERWRITE NON-DESTRUCTIVE (OVRN) BB

This operator functions in the same way as the STON operator, except that the OVRN operator overrides memory protection checks.

STACK OPERATORS

EXCHANGE (EXCH) B6

The operands in the A register and the B register are exchanged. The A and B registers may contain either operands or control words. The control words are treated as operands by this operator.

DELETE TOP OF STACK (DLET) B5

This operator marks the Top-of-Stack register empty.

DUPLICATE TOP OF STACK (DUPL) B7

The operand in the B register is copied into the A register, or the operand in the A register is copied into the B register. At the conclusion of the operation the register that received the copy is marked full.

PUSH DOWN STACK REGISTERS (PUSH) B4

This operator stores the valid word(s) from the A register and/or B register into the memory portion of the stack. The A and B registers are marked empty.

LITERAL CALL OPERATORS

LIT CALL ZERO (ZERO) B0

This operator sets the A register to all 0's and marks the register full. The result is a single-precision operand.

LIT CALL ONE (ONE) B1

This operator sets the A register low order bit (bit 0) to one, leaving all other bits set to 0. The A register is marked full. The result is a single-precision operand.

LIT CALL 8 BITS (LT8) B2

The syllable following the operator is the literal value to be placed in bits 7:8 of the A register. The rest of the A register is set to all 0's. The A register is marked as full and the PSR is set to the syllable following the literal.

LIT CALL 16 BITS (LT16) B3

The next two syllables following the operator are a 16-bit literal value that is placed in bits 15:16 of the A register. The rest of the register is set to all 0's. The A register is marked full and PSR is advanced past the 16-bit literal.

LIT CALL 48 BITS (LT48) BE

The next program word is placed in the A register, and the A register tag is set to all 0's. The A register is marked full, and the PIR and PSR are advanced to the program syllable following the 48-bit literal value. This operator requires that the 48 bit literal in the program string be word synchronized. If the operator syllable is in any syllable position other than syllable five, the intervening syllables are not executed.

The 48 bit literal word must contain a tag field value of 3 (program word), otherwise an invalid program word interrupt will be sensed when the literal word is present in the P (program) register.

MAKE PROGRAM CONTROL WORD (MPCW) BF

This operator performs a "Lit Call 48 Bits" (LT48) as described above; however, the tag is set to a PCW (111) and the Stack Number Register is placed in bits 45:10. The A register is marked full.

INDEX AND LOAD OPERATORS

INDEX (INDX) A6

The Index operator places the integerized value of the B register into the 20-bit length/index field of the Descriptor in the A register. The Descriptor is marked indexed (bit 45 is set to one), and the copy bit is set (bit 46 is set to one).

If the word in the A register is an operand, the A operand is exchanged with the B operand. If the word in the A register is neither a Descriptor nor an IRW pointing to a Descriptor, the invalid operand interrupt is set and the operation is terminated.

If the indexing value is negative or greater than or equal to the length field of the descriptor, the invalid index interrupt is set and the operation is terminated.

If the descriptor represents an array which is segmented, the index is partitioned into two portions by an approximation algorithm which is determined by the type of data referenced by the descriptor, double-precision word-128, single-precision word 256, four-bit digit-3072, six-bit character-2048, or eight-bit byte-1536). The product of the approximatior algorithm is used as an index to the given descriptor to fetch the array-row descriptor. The remainder is used to index the row descriptor.

If the double-precision bit (bit 45) in the descriptor is 1, the index value in the B register is doubled. The balance of the operation is as described in the first paragraph of the description of this operator (INDX).

INDEX AND LOAD NAME (NXLN) A5

This operator performs an index operation; after the word in the A register has been indexed, the Data Descriptor pointed to by this word is brought into the A register. The copy bit (bit 46) of the Data Descriptor is set to one and the A register is marked full. If the presence bit (bit 47) is off, the address of the original descriptor is placed in the address field of the stack copy. If the word accessed by the indexed word in the A register is not a Data Descriptor, the invalid operand interrupt is set and the operation is terminated.

If the Data Descriptor accessed by the indexed word in the A register has the Index bit (bit 45) set to one, the invalid operand interrupt is set and the operation is terminated.

INDEX AND LOAD VALUE (NXLV) AD

This operator performs an index operation; after the word in the A register has been indexed the operand pointed to by this descriptor is brought to the A register. The A register is marked full.

If the word accessed is other than an operand, the invalid operand interrupt is set and the operator is terminated.

LOAD (LOAD) BD

The Load operator places the word addressed by an IRW or Indexed Data Descriptor in the A register.

If at the start of this operator the A register contains other than a Data Descriptor or an IRW pointing at a Data Descriptor, the invalid operand interrupt is set and the operation is terminated.

If the word pointed at by the Data Descriptor is another Data Descriptor, the latter is marked as a copy (copy bit [bit 46] is set to one), and if the presence bit (bit 47) is off, the address of the original is placed in bits 19:20 of the copy in the stack.

SCALE OPERATORS

Higher-level languages such as COBOL require decimal arithmetic. The Scale Operators provide the means of aligning decimal points prior to the time that the arithmetic operations are performed. In addition, the Scale Right operators provide for binary-to-decimal conversions.

SCALE LEFT (SCLF) C0

This operator uses the second syllable as the scale factor. The operand to be scaled is placed in the B register and integerized. The resulting integer is then multiplied by 10 raised to the power specified by the scale factor.

If scaling of a single-precision operand results in overflow, the single-precision operand is converted to a double-precision integer. A double-precision integer is defined as a double-precision operand with an exponent equal to 13.

If scaling of the operand results in an exponent greater than 13, (double-precision operand), the overflow flip flop is set to one.

DYNAMIC SCALE LEFT (DSLF) C1

This operator performs virtually the same operation as the Scale Left (SCLF) operator; however, the scale factor is taken from the A register rather than from the program syllable following the operation syllable. The operand in the A register is integerized before scaling takes place.

SCALE RIGHT SAVE (SCRS) C4

This operator uses its second syllable as the scale factor. The operand to be scaled is placed in the B register and is then integerized. The resultant integer is divided by 10 raised to the power specified by the scale factor.

The quotient resulting from the division is left in the A register. The operand in the B register is the remainder which is converted to decimal (four-bit digits) and is left-justified. The A and B registers are both marked full.

If the scale factor is greater than 12, the invalid operand interrupt is set and the operation is terminated.

DYNAMIC SCALE RIGHT SAVE (DSRS) C5

This operator performs virtually the same operation as the Scale Right Save (SCRS) operator; however, the scale factor is obtained from the A register rather than from the program syllable following the operation syllable. The operand in the A register is integerized before being used.

SCALE RIGHT TRUNCATE (SCRT) C2

This operator performs a Scale Right function using its second syllable as the scale factor. The B register is marked as empty at the conclusion of this operator.

DYNAMIC SCALE RIGHT TRUNCATE (DSRT) C3

This operator performs the same operation as the Scale Right Truncate except that the scale factor is found in the A register and is first integerized by the operator.

SCALE RIGHT FINAL (SCRF) C6

This operator performs a Scale Right operation except that the quotient in the A register is deleted by marking the A register empty. The sign of the quotient is placed in the external sign flip flop.

If the quotient was non-zero at the conclusion of the operation, the overflow flip flop is set.

DYNAMIC SCALE RIGHT FINAL (DSRF) C7

This operator performs a Scale Right Final operation with the scale factor found in the A register which is integerized by the operator before use.

SCALE RIGHT ROUNDED (SCRR) C8

This operator performs a Scale Right operation and the quotient is rounded by adding one to it if the most-significant digit of the remainder is equal to or greater than five. The remainder is deleted from the stack by marking the B register empty.

DYNAMIC SCALE RIGHT ROUND (DSRR) C9

This operator performs a Scale Right Rounded operation using the scale factor found in the A register.

BIT OPERATORS

The Bit operators are concerned with a specified bit in the A register and/or B register.

BIT SET (BSET) 96

This operator sets a bit in the top of stack register. The bit that is set is specified by the program syllable following the operation syllable.

If the program syllable defining the bit to be set has a value greater than 47, the invalid-operand interrupt is set and the operation is terminated.

DYNAMIC BIT SET (DBST) 97

This operator performs a Bit Set Operation upon the bit specified by the operand in the top of stack register. This word is integerized before it is used as a bit number.

If the word in the top of stack register is not an operand, an invalid operand interrupt is set and the operation is terminated.

If after being integerized the operand is less than zero or greater than 47, an invalid operand interrupt is set and the operation is terminated.

BIT RESET (BRST) 9E

This operator resets a bit in the top of stack register. The bit that is reset is specified by the syllable following the operation syllable.

If the program syllable defining the bit to be reset has a value greater than 47, an invalid-operand interrupt is set and the operation is terminated.

DYNAMIC BIT RESET (DBRS) 9F

This operator performs a Bit Reset operation upon the bit specified by the operand in the top-of-stack register.

If the word in the top-of-the-stack register is not an operand, an invalid operand interrupt is set and the operation is terminated.

If after being integerized the operand is less than zero or greater than 47, an invalid operand interrupt is set and the operand is terminated.

CHANGE SIGN BIT (CHSN) 8E

The sign bit (bit 46) of the top-of-stack operand is complemented, i.e., if it is a one, it is set to 0; if it is a 0, the bit is set to one.

TRANSFER OPERATORS

The Transfer Operators transfer any field of bits from one word in the stack to any field of another word in the stack.

FIELD TRANSFER (FLTR) 98

This operator uses its following three syllables to establish the pointers used in the field transfer. This is done in the following manner. The second syllable of the operator is K. The third syllable of the operator is G. The fourth syllable of the operator is L.

The field in the A register, starting at the bit position addressed by G, is transferred into the B register, starting at the bit position addressed by K. The length of the field in the A and B registers is defined by L. When the specified number of bits have been transferred, the A register is set to empty, the B register is marked full and the operation is complete.

If the second or third syllables of the operator are found to be greater than 47, or the fourth syllable is greater than 48, the invalid operand interrupt is set and the operation is terminated.

DYNAMIC FIELD TRANSFER (DFTR) 99

This operator performs a Field Transfer operation with the exception that the B register operand is L. The B register is then reloaded from the stack and this operand is G. The B register is again loaded from the stack and this operand is K.

If any of the three operands is a non-integer, it is first integerized. Each is checked for a value less than equal to zero or greater than equal to 48, or less than 48, as specified in Field Transfer above. If either of these conditions exists in any one of the three operands, an invalid operand interrupt is set and the operation is terminated.

FIELD ISOLATE (ISOL) 9A

This operator isolates a field of the word in the A register, placing it right-justified in the top of stack register. The balance of the top of stack register is cleared to 0's. The top of stack register is marked full.

This operator uses its second and third syllables as the BIT pointers. The second syllable of the operator addresses the starting bit of the field in the A register. The third syllable of the operator specifies the length of the field to be isolated.

If the value of the second syllable is greater than 47 or the value of the third syllable is greater than 48, an invalid operand interrupt is set and the operation is terminated.

DYNAMIC FIELD ISOLATE (DISO) 9B

This operator performs a Field Isolate operation except that the first item in the stack specifies the length of the field to be isolated. The second operand in the stack addresses the bit in the word of the third item in the stack that is to be isolated.

If after being integerized the value of the first item in the stack is less than 0 or greater than 47, an invalid operand interrupt is set and the operation is terminated.

If after being integerized the value of the second item in the stack is less than 0 or greater than 48, an invalid interrupt is set and the operation is terminated.

FIELD INSERT (INSR) 9C

This operator inserts a field from the A register into the B register word. The field in the A register is right justified with the length of the field specified by the third syllable of the operator. The second syllable of the operand addresses the starting bit of the field in the B register. At completion the A register is marked empty and the B register is marked full.

If the value of the second syllable of the operator is greater than 47, an invalid operand interrupt is set and the operation is terminated.

If the value of the third syllable of the operator is greater than 48 an invalid operand interrupt is set and the operation is terminated.

DYNAMIC FIELD INSERT (DINS) 9D

This operator performs a Field Insert operation except the first item in the stack is used as the insert field data. The second item in the stack is used to specify the length of the field. The third item in the stack is used to address the starting bit in the receiving field in the B register. When the operation is complete, the A register is marked empty and the B register is marked full.

If after being integerized the value of the second item in the stack is less than 0 or greater than 48, an invalid operand interrupt is set and the operation is terminated.

If after being integerized the value of the third item in the stack is less than 0 or greater than 47, an invalid operand interrupt is set and the operation is terminated.

STRING TRANSFER OPERATORS

String Transfer operators give the system the ability to transfer characters or words from one location in memory to another location in memory. The source and destination pointers are set from String Descriptors in the stack.

TRANSFER WORDS, DESTRUCTIVE (TWSD) D3

This operator requires three items in the top-of-the-stack: an operand, a String Descriptor or operand, and a String Descriptor. The first operand is integerized and used as the count or repeat field. The second item is either the source data or a descriptor which points at the source string and the third item is used to address the destination string. The number of words specified by the repeat field are transferred from the source to the destination. At completion of the operation, the A and the B registers are marked empty.

If the memory protect bit is found on during the execution of the Transfer Words operator, the segmented array interrupt is set and the operation is terminated.

TRANSFER WORDS, UPDATE (TWSU) DB

This operator performs the Transfer Words operator except that at the completion of the transfer of data, the source and destination pointers are updated to point to the location in memory where the transfer ended. The A and B registers are both marked full.

TRANSFER WORDS, OVERWRITE DESTRUCTIVE (TWOD) D4

This operator performs a Transfer Words, Destructive operation, except that it overrides the memory protection checks.

TRANSFER WORDS, OVERWRITE UPDATE (TWOU) DC

This operator performs a Transfer Words, Update operation, except that it overrides the memory protection checks.

TRANSFER WHILE GREATER, DESTRUCTIVE (TGTD) E2

This operator transfers characters from a location in memory pointed to by the source pointer, to a location in memory pointed to by the destination pointer, until the number of characters specified has been transferred or the comparison fails. The TFFF flip-flop is used to indicate the results of the comparison. TFFF is set at the beginning of the operator.

The first item in the stack is used as the delimiter. The second item in the stack, bits 19:20, is the maximum number of characters to be transferred. The third item in the stack is the source data or a source pointer, and the fourth item in the stack is the destination pointer.

The source and destination strings are checked for memory protection. The source character is compared to the delimiter. After each comparison, a decision is made whether the condition has been met. If the condition is met, TFFF remains set to one, if it is not met it is set to 0. If the result of the comparison is equal then the CMPF flip-flop is set, and otherwise CMPF is reset.

If the number of characters transferred was equal to the repeat field the TFOF flip-flop is set to one. The A and B registers are marked empty and the operation is complete.

If the first operand in the stack is not a single-precision operand, an invalid operand interrupt is set and the operation is terminated.

If either the source or destination word has a memory protect bit on (bit 48=1), the segmented array interrupt is set and the operation is terminated.

If the second item in the stack is a descriptor, it is used as the source pointer and the length field or repeat field is set to 1,048,575. All comparisons are binary (EBCDIC collating sequence).

TRANSFER WHILE GREATER UPDATE (TGTU) EA

This operator performs a Transfer While Greater operation and updates the source pointer and destination pointer to point at the next characters in the source and destination strings. The repeat count is updated to give the number of characters not transferred. If the operation is terminated because the relationship is not met, the source pointer points at the character that failed the comparison. If the result of the comparison is equal, then the CMPF flip-flop is set, otherwise CMPF is reset.

TRANSFER WHILE GREATER OR EQUAL, DESTRUCTIVE (TGED) E1

This operator performs a Transfer While operation using the relation greater than or equal to for comparison.

TRANSFER WHILE GREATER OR EQUAL, UPDATE (TGEU) E9

This operator performs a Transfer While Greater or Equal operation. The source pointer, destination pointers, and count are updated at the conclusion of the operation.

TRANSFER WHILE EQUAL, DESTRUCTIVE (TEQD) E4

This operator performs a Transfer While operation with the relation used in the comparison being equal. If the result of the comparison is greater, then the CMPF flip-flop is set, otherwise CMPF is reset.

TRANSFER WHILE EQUAL, UPDATE (TEQU) EC

This operator performs a Transfer While Equal operation. The source pointer, the destination pointer and count are updated at the conclusion of the operation. CMPF is set if the result of the comparison is greater, and CMPF is reset otherwise.

TRANSFER WHILE LESS OR EQUAL, DESTRUCTIVE (TLED) E3

This operator performs a Transfer While operation, using the Less than or Equal comparison.

TRANSFER WHILE LESS OR EQUAL, UPDATE (TLEU) EB

This operator performs a Transfer While Less or Equal operation. The source pointer, destination pointer and count are updated at the conclusion of the operation.

TRANSFER WHILE LESS, DESTRUCTIVE (TLSD) EO

This operator performs a Transfer While operation using the Less than comparison. If the result of the comparison is equal then the CMPF flip-flop is set, otherwise CMPF is reset.

TRANSFER WHILE LESS, UPDATE (TLSU) E8

This operator performs a Transfer While Less operation. The source pointer, destination pointer and count are updated at the conclusion of the operation.

TRANSFER WHILE NOT EQUAL, DESTRUCTIVE (TNED) E5

This operator performs a Transfer While operation, using the not equal comparison. CMPF is not used.

TRANSFER WHILE NOT EQUAL, UPDATE (TNEU) ED

This operator performs a Transfer While Not Equal operation. The source pointer, the destination pointer and count are updated at the conclusion of the operation.

TRANSFER UNCONDITIONAL, DESTRUCTIVE (TUND) E6

This operator performs a Transfer Characters until the length is equal to zero. No comparisons are made.

TRANSFER UNCONDITIONAL, UPDATE (TUNU) EE

This operator performs a Transfer Unconditional operation. The source pointer and the destination pointer are updated at the conclusion of the operation.

STRING ISOLATE (SISO) D5

This operator places in the top-of-the-stack, right justified, the number of source characters specified by the repeat field. The first item in the stack is the number of characters in the repeat field. The second item in the stack is either an operand or a descriptor used as the source pointer.

If the number of bits to be transferred is greater than 48, the item is double-precision.

If the number of bits is greater than 96, an invalid operand interrupt is set and the operation is terminated.

If the source data has the memory protect bit (bit 48) set to one, the segmented array interrupt is set and the operation is terminated.

COMPARE OPERATORS

The compare operators perform the specified comparison of two strings of data. The True False flip flop (TFFF) and the Compare flip flop (CMPF) are used to indicate the result of the comparison, at the conclusion of the operation. Table 7-2 shows the significance of the state of TFFF and CMPF at the conclusion of a compare type operator.

COMPARE CHARACTERS GREATER, DESTRUCTIVE (CGTD) F2

This operator compares the value of two character strings, one character at a time. The operator compares characters until it encounters a pair which are unequal. If the B string character is greater than the A string character, the TFFF is set, otherwise it is reset. If the length is depleted and the character strings are equal, the CMPF flip-flop is set. If the characters in the B string are greater than the characters in the A string, the TFFF is set to one. If not, the TFFF is set to zero.

The first item in the stack is an operand which contains the length of the fields being compared. The second item in the stack is an operand or a descriptor pointing at the character string to be compared against. The third item in the stack is a descriptor pointing at the character string to be compared.

If the repeat count is depleted the TFFF is reset.

If either of the data strings has the memory protect bit on (bit 48=1), the segmented array interrupt is set and the operation is terminated.

All comparisons are by the binary character position in the collating sequence.

Table 7-2. Compare Type Operator Results

Compare	TFFF	CMPF	Comparison Result
=	0	0	Less than equal
	0	1	Greater than equal
	1	0	Equal
	1	1	Not applicable
≠	0	0	Equal
	0	1	Not applicable
	1	0	Less than equal
	1	1	Greater than equal
>	0	0	Less than equal
	0	1	Equal
	1	0	Greater than equal
	1	1	Not applicable
<	0	0	Greater than equal
	0	1	Equal
	1	0	Less than equal
	1	1	Not applicable
⩾	0	0	Less than equal
	0	1	Not applicable
	1	0	Greater than equal
	1	1	Equal
⩽	0	0	Greater than equal
	0	1	Not applicable
	1	0	Less than equal
	1	1	Equal

COMPARE CHARACTERS GREATER, UPDATE (CGTU) FA

This operator performs a Compare Characters Greater operation. The source pointer and destination pointer are updated at the conclusion of the operation.

COMPARE CHARACTERS GREATER OR EQUAL, DESTRUCTIVE (CGED) F1

This operator performs the Compare Characters operation with the comparison being greater than or equal. If the repeat count ⩽ 0, the TFFF is set to 1.

COMPARE CHARACTERS GREATER OR EQUAL, UPDATE (CGEU) F9

This operator performs a Compare Character Greater or Equal operation. The source pointer and destination pointer are updated at the conclusion of the operation.

COMPARE CHARACTERS EQUAL, DESTRUCTIVE (CEQD) F4

This operator performs the Compare Characters operation using the Equal comparison. If the repeat count ≤ 0, then TFFF is set to 1.

COMPARE CHARACTERS EQUAL, UPDATE (CEQU) FC

This operator performs a Compare Characters Equal operation. The source pointer and destination pointer are updated at the conclusion of the operation.

COMPARE CHARACTERS LESS OR EQUAL, DESTRUCTIVE (CLED) F3

This operator performs the Compare Characters operation with the Less than or Equal comparison. If the repeat count ≤ 0, then TFFF is set to 1.

COMPARE CHARACTERS LESS OR EQUAL, UPDATE (CLEU) FB

This operator performs a Compare Characters Less or Equal operation. The source pointer and destination pointers are updated at the conclusion of the operation.

COMPARE CHARACTERS LESS, DESTRUCTIVE (CLSD) F0

This operator performs the Compare Characters operation using the Less than comparison. If the repeat count ≤ 0, the TFFF is set to 0.

COMPARE CHARACTERS LESS, UPDATE (CLSU) F8

This operator performs a Compare Characters Less operation. The source pointer and the destination pointer are updated at the conclusion of the operation.

COMPARE CHARACTERS NOT EQUAL, DESTRUCTIVE (CNED) F5

This operator performs the Compare Characters operation using the Not equal relation. If the repeat count ≤ 0, then TFFF is set to 0.

COMPARE CHARACTERS NOT EQUAL, UPDATE (CNEU) FD

This operator performs a Compare Characters Not Equal operation. The source pointer and the destination pointer are updated at the conclusion of the operation.

EDIT OPERATORS

TABLE ENTER EDIT, DESTRUCTIVE (TEED) D0

This operator is used to prepare for edit micro-instructions. These edit micro-instructions are contained in memory as a table and not as part of the normal program string. When this operator is entered, program execution is transferred to a table of micro-instructions. The last micro-instruction in this table must be the End Edit operator (see section 9). The table contains Edit Mode operators.

The first item in the stack is a descriptor pointing to the table of Edit Micro-Instructions. The second item in the stack is a single-precision operand or a descriptor pointing at the source string. The third item in the stack is descriptor pointing at the destination.

If the first item in the stack is not a descriptor, the invalid operand interrupt is set and the operation is terminated.

If the second item in the stack is a single-precision operand, it is the source string.

If the third item in the stack is not a descriptor, the invalid operand interrupt is set and the operation is terminated.

TABLE ENTER EDIT, UPDATE (TEEU) D8

This operator performs a Table Enter Edit operation and updates the source pointer and destination pointer at the completion of the operation.

EXECUTE SINGLE MICRO, DESTRUCTIVE (EXSD) D2

This operator performs the same function as the Table Enter Edit operator with the following exceptions: There is only one micro-operator and it follows this syllable. The first item in the stack is a single-precision operand that defines the length field.

An end edit operation is performed as an implicit part of the EXSD operator, thus an explicit END EDIT operator (in program line code) is not required.

EXECUTE SINGLE MICRO, UPDATE (EXSU) DA

This operator performs the same functions as an Execute Single Micro operator, except that it updates the source pointer and destination pointer at the completion of the edit operator operation.

EXECUTE SINGLE MICRO, SINGLE POINTER UPDATE (EXPU) DD

This operator performs the same functions as an Execute Single Micro Update operator, except that one pointer is used as both source and destination pointer. The destination pointer is updated at the completion of the operation.

PACK OPERATORS

PACK, DESTRUCTIVE (PACD) D1

This operator packs data, addressed by the source pointer, into the top of the stack in four-bit (digit) format. The TFFF is set to one if the source data is negative. A negative number for an eight-bit (byte) format has a zone bit configuration of 1101 in the least significant byte. The six-bit (BCL) format for a negative number has a configuration of 10 in the least significant character position. The four-bit (digit) format has a 1101 configuration in the most-significant digit position. Data is right-justified as it is placed in the top-of-stack.

The operand in the top-of-the-stack is used as the length field. The second item is the source pointer. The operation then continues until the number of digits specified by the length/repeat field have been packed.

If the length is less than 13, the operand in the top-of-the-stack is a single-precision operand. If the operand is 13 or greater, the result is a double-precision operand.

If the length is not less than 25, an invalid operand interrupt is set and the operation terminated.

If initial length is zero, TOS is filled with zeros.

If the second item in the stack is an operand, it is the source string and is comprised of eight-bit bytes.

If the source data has the memory protect bit (bit 48) set to one, the segmented array interrupt is set and the operation is terminated.

PACK, UPDATE (PACU) D9

This operator performs a Pack operation, updating the source pointer at the completion of the operation.

INPUT CONVERT OPERATORS

INPUT CONVERT, DESTRUCTIVE (ICVD) CA

This operator converts either six-bit BCL code, eight-bit EBCDIC or four-bit digit code to an operand for internal arithmetic operations.

The first item in the stack is an operand that is integerized to form the repeat field. The second item in the stack is a descriptor used as a source pointer.

The input convert operator converts a string of input EBCDIC or BCL character data into a numeric operand. The resultant operand may be either single precision or double precision.

The manner in which the conversion of character data into numeric data is performed is as follows:

The four high-order zone bits of the input EBCDIC character (two high-order bits of a BCL input character) are discarded. The remaining four low-order digit bits from the input character form a hexadecimal character, which is placed in the top of stack register receiving field.

Each time that a source input character is converted, the repeat field is decremented by one. When the repeat field is equal to zero, all input characters have been converted.

If the repeat field value is 13 (decimal) or less the resultant operand in the top of stack register is a single precision operand. If the repeat field value is between 13, and 24 (decimal) the resultant operand in the top of stack register is a double precision operand. If the repeat field is greater than 24, an invalid operator interrupt is set, and the operation is terminated.

The sign of the converted resultant operand is determined from the zone bits of the least significant character in the input character string. For EBCDIC input characters the sign is negative if the least significant character zone bits are equal to 1101 binary, and is positive otherwise. For BCL input characters the sign is negative if the two zone bits equal 10 binary, and is positive otherwise. The detected sign bit for the resultant operand is saved in the TFFF flip flop.

The sign of the converted operand is then set from the TFFF. If the converted operand is a single-precision operand, the TFFF is then set to 1. If the converted operand is a double-precision operand, the TFFF is set to 0.

At the completion of the operation the B register is marked full. The tag field is set to indicate either a single- or a double-precision operand.

If after being integerized, the item in the top-of-stack is greater than 23, the invalid operand interrupt is set and the operation is terminated.

INPUT CONVERT, UPDATE (ICVU) CB

This operator performs an Input Convert operation. The source pointer is updated at the completion of the operation.

READ TRUE FALSE FLIP-FLOP (RTFF) DE

This operator places the status of the TFFF into the low-order bit position of the A register. The rest of the A register is set to all 0's. The A register is marked full at completion of this operation.

SET EXTERNAL SIGN (SXSN) D6

This operator places the mantissa sign of the top word of the stack in the External Sign flip flop. This operand is not deleted from the stack at the end of the operation.

READ AND CLEAR OVERFLOW FLIP-FLOP (ROFF) D7

This operation places the status of the Overflow flip flop in the least-significant bit of the A register, sets the rest of the A register to all 0's, marks the register full, and sets the Overflow flip flop to 0.

SUBROUTINE OPERATORS

VALUE CALL (VALC) 00 ⇒ 3F

This operator loads into the A register the operand addressed by the address couple formed by the concatenation of the six low order bits of the first syllable and the eight bits of the following syllable. The A register is marked full. Figures 7-1 and 7-2 are simplified flow charts of the Value Call operator.

This operator makes multiple memory accesses if the word accessed is either an indexed descriptor, PCW, or an IRW.

If the word accessed is an indexed data descriptor, the word addressed by the data descriptor is brought to the top-of-the-stack. If the double-precision bit (bit 50) in the Data Descriptor is equal to one, the other half of the double-precision operand is brought to the X register.

If the word accessed is a non-indexed word data descriptor, the word is indexed using the second word in the stack for the index value. The word addressed by the non-indexed Data Descriptor is brought to the top-of-the-stack. If the double-precision bit (40) in the Data Descriptor is equal to one, the other half of the double-processor operand is brought to the X register.

If the word accessed by the Data Desciptor is another indexed Data Descriptor, the word addressed by the Data Descriptor is brought to the top-of-the-stack, and one of the two above paragraphs is repeated.

If a Data Descriptor does not address an operand, SIRW, or a word descriptor or indexed string descriptor, an invalid operand interrupt is set and the operation is terminated.

If the word accessed by the value call is an Indirect Reference Word (IRW) the word addressed by the IRW is accessed and evaluated. If the word is an operand, it is placed in the top-of-the-stack.

If the word accessed by the IRW is another IRW, the operation continues as described above.

If the word accessed by the IRW is an indexed or non-indexed Data Descriptor, the operator proceeds as described above for Data Descriptors.

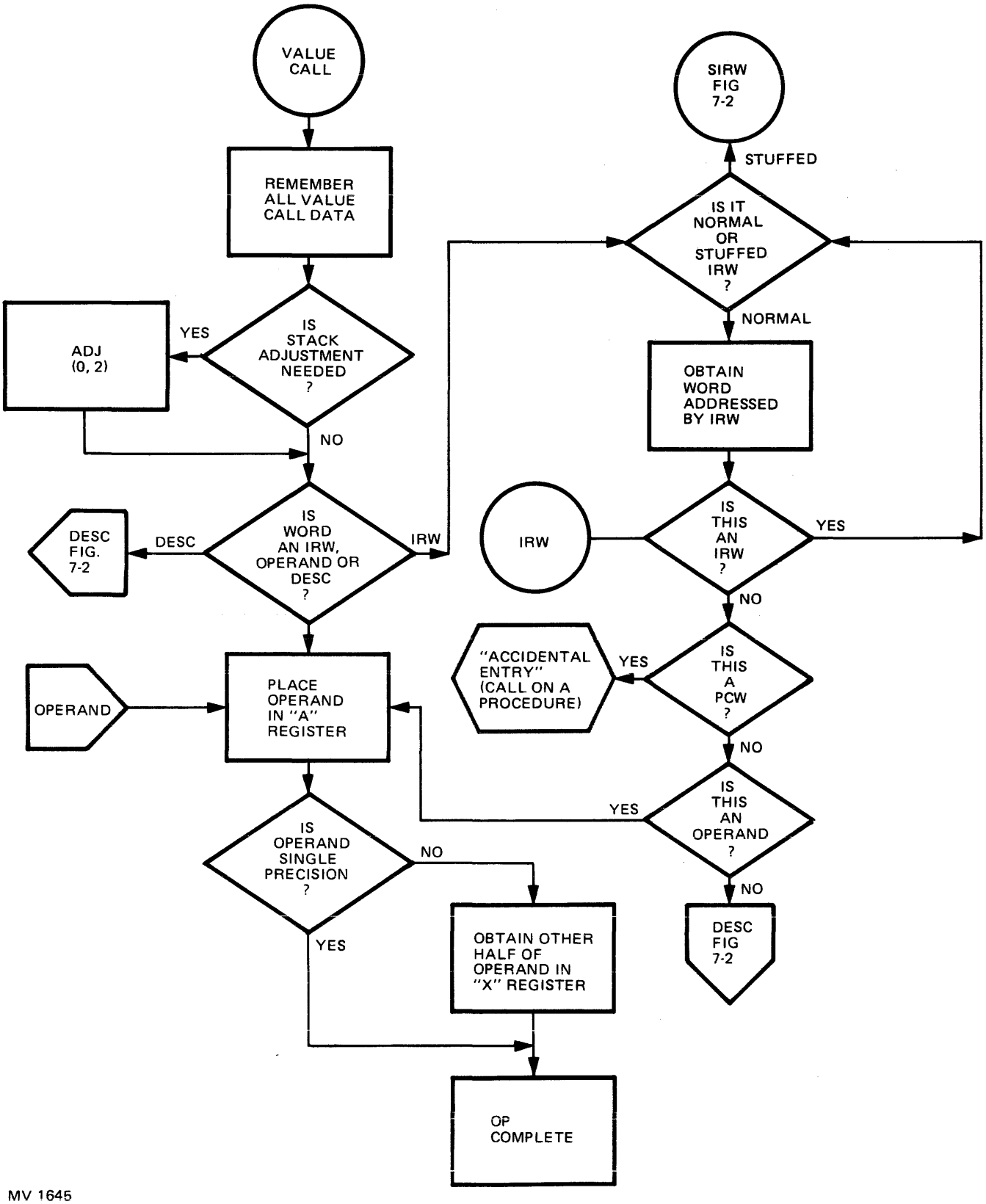


Figure 7-1. Flow of Value Call Operator

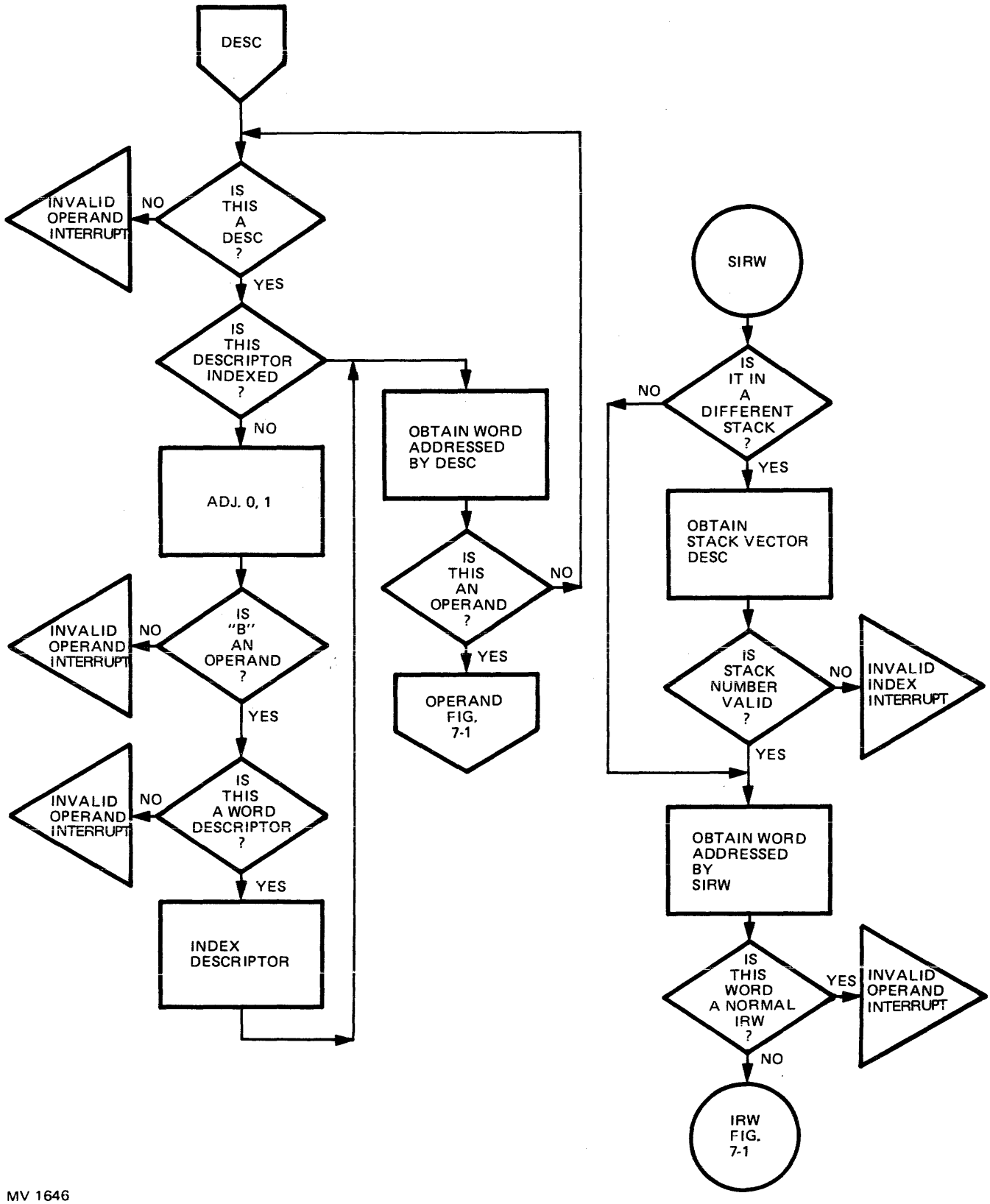


Figure 7-2. Value Call (Descriptor) Operator

If the word accessed by the IRW is a Program Control Word (PCW), an accidental entry into the subroutine addressed by the PCW is initiated. A Mark Stack Control Word (MSCW) and a Return Control Word (RCW) are placed in the stack and an entry is made into the program. Upon completion of the program, a return operator will re-enter the flow value call at the label IRW, (figure 7-1).

NAME CALL (NAMC) 40 ⇒ 7F

This operator builds an IRW in the A register. The address couple is formed by concatenating the six low-order bits of the first syllable and the eight bits of the following syllable. The A register is marked full and the operation is complete.

EXIT OPERATOR (EXIT) A3

This operator returns to a calling procedure from a called procedure resetting all control registers from the RCW and the MSCW. The Exit operator does not return a value to the calling routine. Figure 7-3 shows a simplified flow chart of the Exit operator.

RETURN OPERATOR (RETN) A7

This operator performs the same functions as an Exit operator with the exception that an operand or name in the B register is returned to the calling procedure. If a name is returned, and the V bit (bit 19) in the MSCW is on, the name is evaluated to yield an operand as described in the VALC operator. Figure 7-4 shows a simplified flow chart of the Return operator.

ENTER OPERATOR (ENTR) AB

This operator is used to cause an entry into a procedure from a calling procedure. Entry is to the program segment and syllable addressed by the PCW. Figure 7-5 shows a simplified flow chart of the Enter operator.

The Enter operator accesses the IRW at F + 1, which points to the PCW (or to the PCW directly, without the use of an IRW). The operator then builds a RCW into the stack at F + 1.

EVALUATE (EVAL) AC

This operator loads the A register with an indexed Data Descriptor or an IRW that addresses A "target," which may be an SIW, and Un-Indexed Data Descriptor, a String Descriptor, or an operand. The "target" may be referenced through a chain of accidental entries, or IRW. In any case memory accesses will continue to be made until the target is located. The A register is left containing the Data Descriptor or the IRW which addresses the target. Figure 7-6 is a simplified flow chart of the Evaluate operator.

An indexed Data Descriptor is left in the A register when the target is referenced by an indexed Data Descriptor; a stuffed IRW is left in the A register when the target is referenced by IRW's.

If the A register does not contain a Data Descriptor or an IRW at the start of this operator, an invalid operand interrupt is set and the operation is terminated.

MARK STACK OPERATOR (MKST) AE

This operator places a Mark Stack Control Word in the B register which contains a pointer to the previous MSCW in the stack. The F register is updated to point at the address of the MSCW.

This operator is used to mark the stack when entry into a procedure is anticipated.

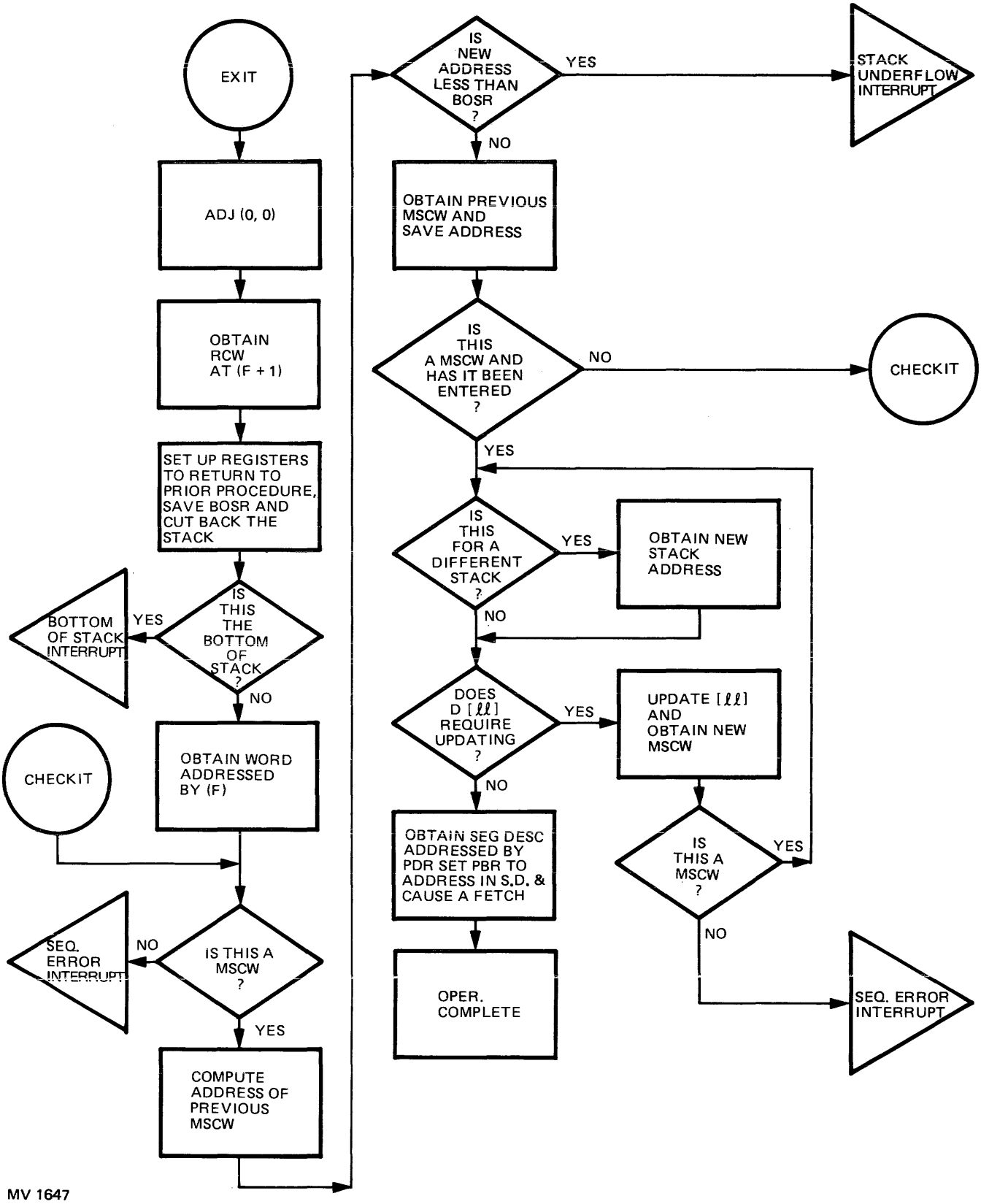


Figure 7-3. Flow of Exit Operator

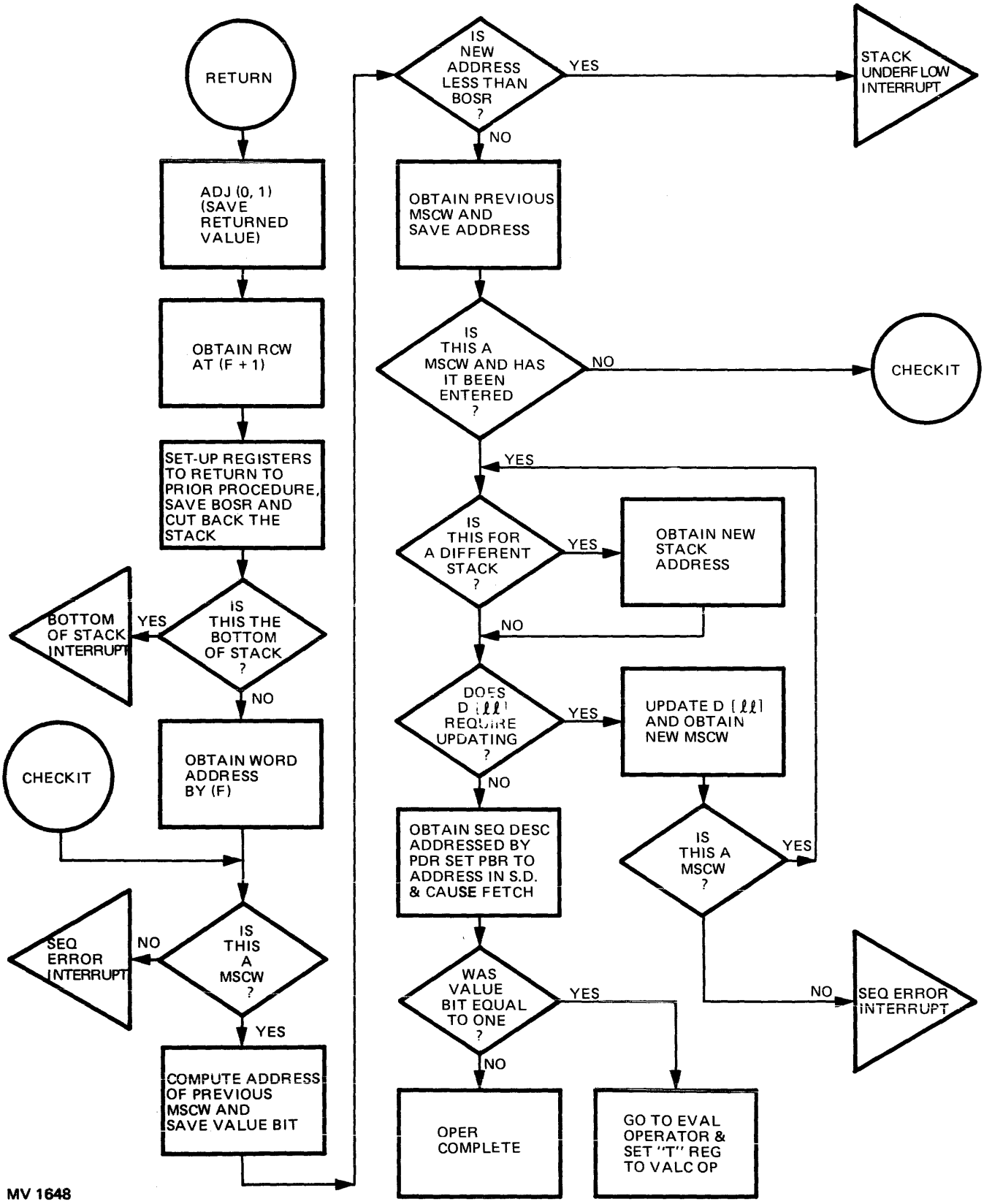


Figure 7-4. Flow of Return Operator

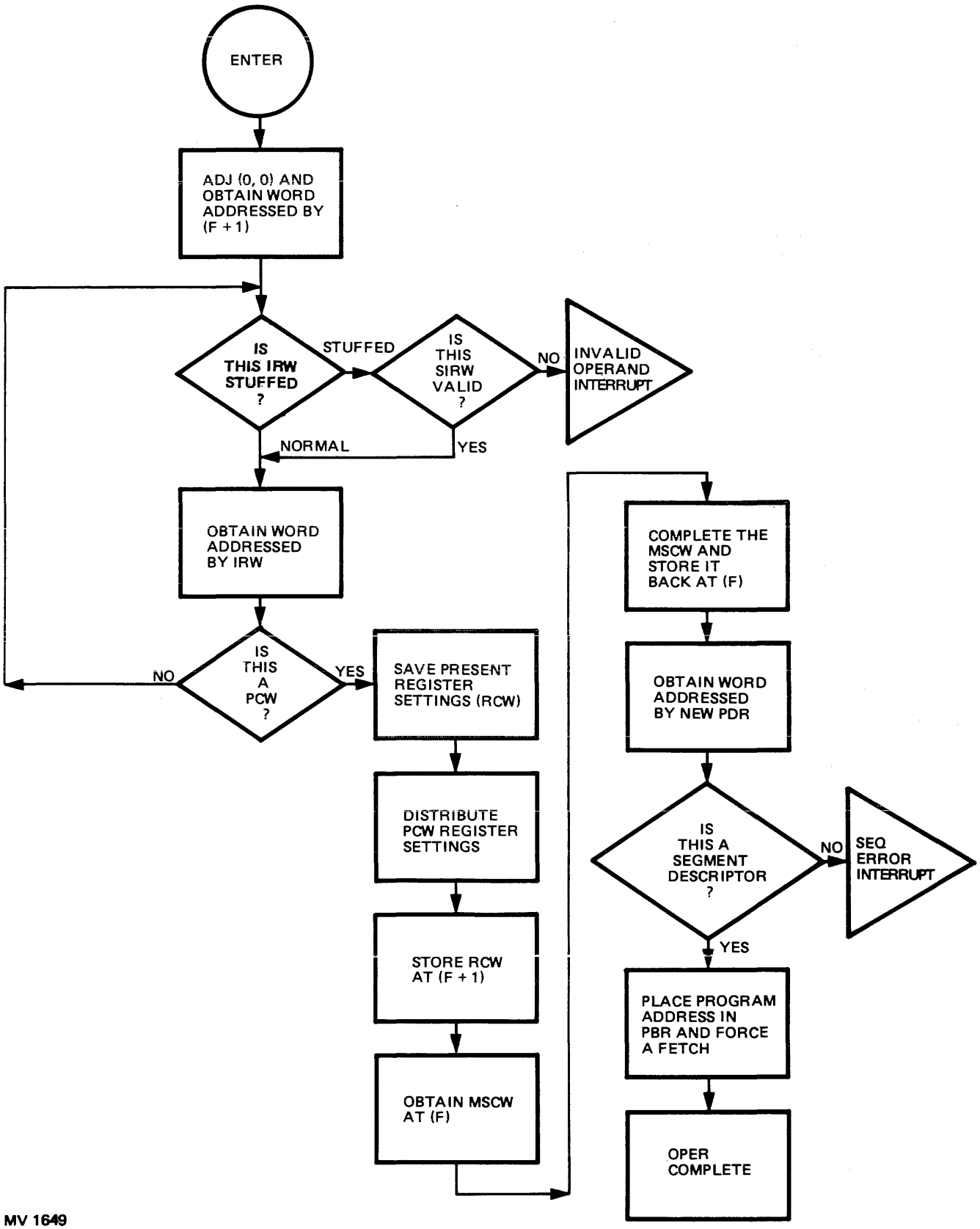


Figure 7-5. Flow of Enter Operator

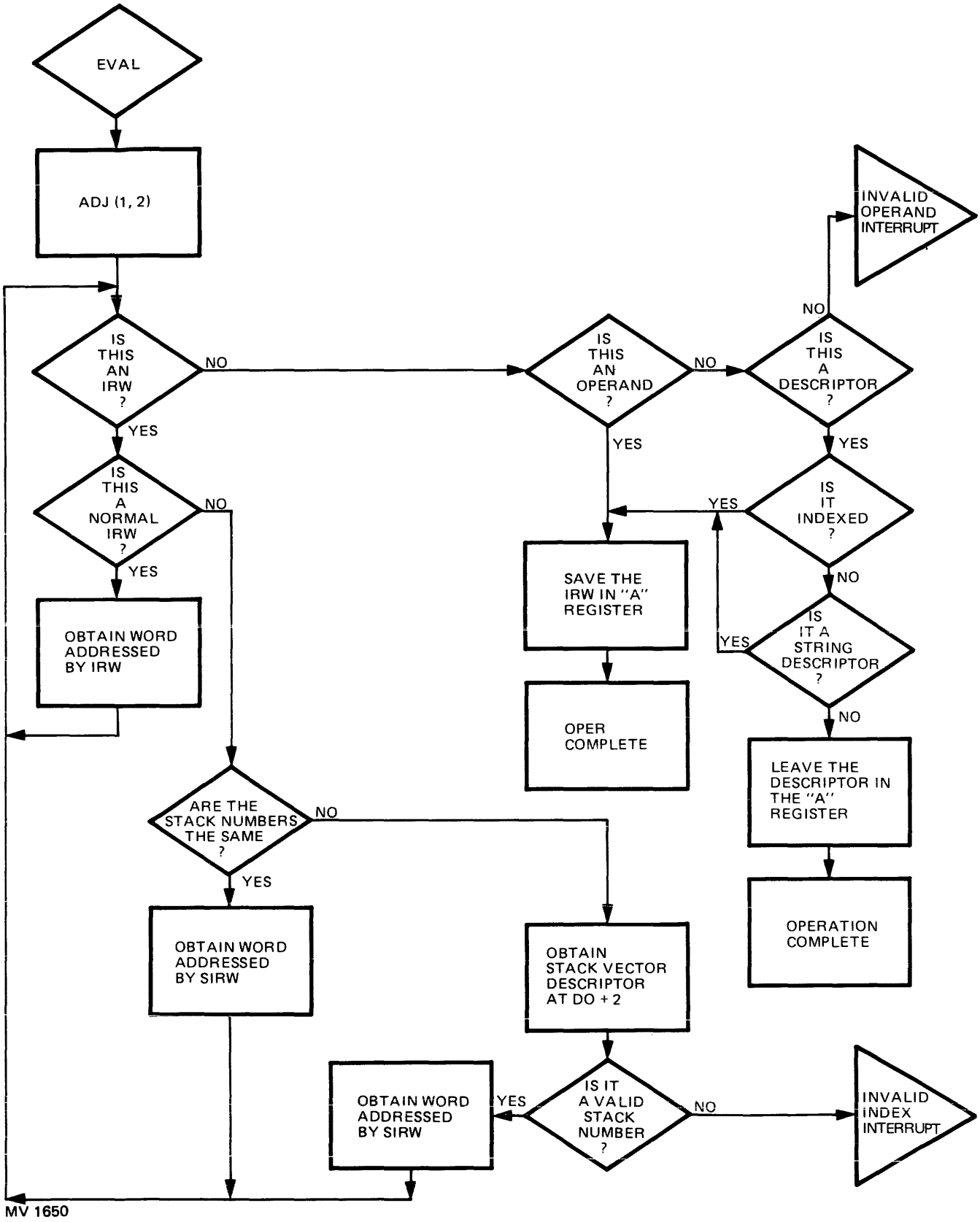


Figure 7-6. Flow of Evaluate Operator

STUFF ENVIRONMENT (STFF) AF

This operator changes a normal IRW to a stuffed IRW so that a quantity may be referenced from a different addressing environment. The displacement field locates the MSCW below the quantity and the index field locates the quantity relative to the MSCW. Figure 7-7 shows a simplified flow chart of the Stuff Environment operator.

If the word in the A register at the start of the operation is not an IRW, an invalid operand interrupt is set and the operation is terminated.

If, when creating this stuffed IRW, other than an MSCW is accessed, a sequence error interrupt is set and the operation is terminated.

INSERT MARK STACK OPERATOR (IMKS) CF

This operator builds an MSCW and places it below the two top-of-stack quantities.

ENTER VECTOR MODE OPERATORS

There are two different operators used to cause the B 6800 system to enter into the vector mode of operation. The Vector Mode Enter Single (VMES) operator is used to enter the vector mode of operation when a single word of program code contains all of the vector mode operators that are to be executed. The Vector Mode Enter Multiple (VMEM) operator is used to enter into the vector mode of operation when the number of vector mode operators that are to be executed uses more than a single word of program code.

A description of the two methods for entering the vector mode of operation is as follows:

VECTOR MODE ENTER MULTIPLE (VMEM) E7

This operator is used to cause entry into the vector mode of operation in the same way that the VMES operator performs. The only difference between the operation of the VMES and the VMEM operators is the number of words of vector mode machine language code that may be used.

If an interrupt occurs while entry into vector mode is in process, the entry process is terminated, and processing resumes with the next normal mode machine language operator in sequence. Since multiple words of vector mode machine language operators are used when the VMEM operator causes entry to vector mode, the first word of normal mode operators may be greatly removed from the VMEM operator code word.

The use of the VMEM operator causes the data processor to retain the address of the next normal mode operator word. This address is required in the event that the entry into vector mode is terminated. The retention of the next normal mode operator word address (in IC memory) is the only difference between the VMES and VMEM operators.

VECTOR MODE ENTER SINGLE (VMES) EF

This operator is used to cause entry into the vector mode of operation. Vector mode operations are performed in control state (IIHF flip flop is set). The VMES operator uses a subset of the table enter edit logic to distribute vector mode parameters in the IC memory address registers of the data processor. The vector mode operator parameters must be on the top of the data processor stack at the beginning of the VMES operator.

The VMES operator expects to find three data descriptors, and three incrementation parameters present on the top of the data processor stack. The VMES operator optionally expects that a LENGTH parameter may be present on the top of the data processor stack.

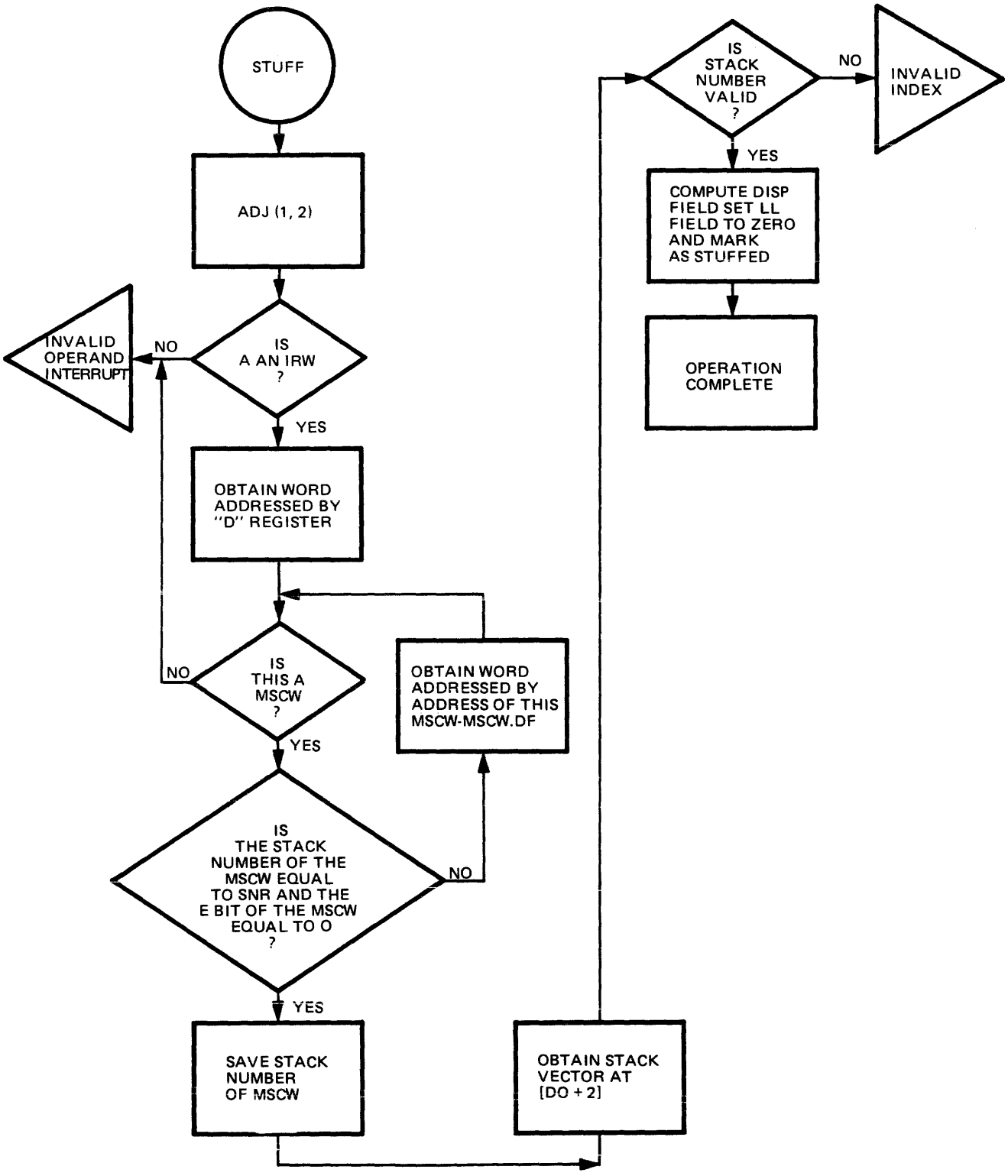


Figure 7-7. Flow of Stuff Environment Operator

If the VMES operator does not find the three data descriptors on the top of the data processor stack an invalid operand interrupt is detected, and the VMES operator releases control to the interrupt controller.

The VMES operator expects to find that bit 47 (the presence bit) is true in each of the three data descriptors. If any of the three data descriptors do not have the presence bit true, a presence bit interrupt is detected, and the VMES operator releases control to the interrupt controller.

The order of occurrence of the three data descriptors and the three increment parameters (and optionally, the LENGTH parameter) is as follows:

Parameter	Word Type	Word Usage
Pointer C	Data descriptor	The top word in the data processor stack.
LENGTH	SP operand	When a LENGTH parameter is present, it is the second word in the data processor stack, and its presence is indicated by bit 44 of pointer C being set. If a LENGTH parameter is not present in the stack a default length value of FFFFF – 1 (HEX) is used
Pointer A	Data descriptor	If a LENGTH parameter is not present in the data processor stack, pointer A is the second word in the data processor stack. If a LENGTH parameter is present in the stack, then pointer A is the third word in the stack.
Pointer B	Data descriptor	If a LENGTH parameter is not present in the stack, pointer B is the third word in the stack. If a LENGTH parameter is present in the stack, then pointer B is the fourth word in the stack.
Increment C	SP operand	The incrementation value that will be used as the incrementation unit to access data elements of the array pointed at by pointer C.
Increment A	SP operand	The incrementation value used for accessing data elements in the array pointed at by pointer A.
Increment B	SP operand	The incrementation value used for accessing data elements in the array pointed at by pointer B.

If bit 44 (the segmented bit) is true in pointer A or B, an invalid operator interrupt is detected, and the VMES operator releases control to the interrupt controller.

If pointer A has the read only bit (bit 43) true, a memory protect interrupt is detected, and the VMES operator releases control to the interrupt controller.

If any of the three types of interrupts described in the preceding paragraphs are detected, the entry into vector mode is terminated, and the program will be resumed (in normal state) at the next code word following the vector operator code word. The use of the VMES operator implies that only one word of vector mode operators is to be used, and the first vector mode operator to be executed is present in syllable zero of the next program code word in sequence. Therefore, the next word of program code (the vector mode code word) is fetched by the program controller and placed in the P register. If an interrupt occurs during the VMES operator, the interrupt controller will fetch another new word of program code (the word following the vector mode code word). Thus, the VMES operator releases control to the interrupt controller, and the interrupt controller fetches the next word of normal state program code that is to be executed.

SECTION 8

VARIANT MODE OPERATION AND OPERATORS

GENERAL

ESCAPE TO 16-BIT INSTRUCTION (VARI) 95

The variant mode of operation extends the number of operation codes. These operators are not used as often and require two syllables; the first is the "Escape to 16-Bit Instruction" (VARI) operator. When the VARI operator is encountered, the following syllable is the actual operation and the syllable pointer is positioned beyond the two syllables. The VARI operator is valid only for the syllables covered in this section.

Variant codes EO through EF are detected and cause a programed operator interrupt. All other unassigned variant codes cause no action and result in a loop timer interrupt.

Variant mode operations are both word- and string-oriented operators.

Appendix A of this manual lists the operators in alphabetic order, and appendix B lists the operators in numeric order, by mode.

VARIANT MODE OPERATORS

SET TWO SINGLES TO DOUBLE (JOIN) 9542

The operands in the A and B registers are combined to form a double-precision operand that is left in the B and Y registers.

The operand in the A register is placed in the Y register. The A register is marked empty and the B register tag field is set to double-precision.

SET DOUBLE TO TWO SINGLES (SPLT) 9543

The SP(DP) operand in the B register is changed to two single-precision operands which are placed in the A and the B registers; both registers are marked full.

If the operand in the B register is a single-precision operand, the A register is set to all 0's and the A and B registers are marked full. Both the A and the B register tag fields are set to single-precision.

If the operand in the B register is a double-precision operand, the Y register operand is placed in the A register and the tag fields of both the A and B registers are set to single-precision.

IDLE UNTIL INTERRUPT (IDLE) 9544

This operator suspends processor program execution until the program is restarted by an external interrupt. Inhibit Interrupt flip flop (IIFF) is unconditionally reset to allow external interrupts.

SET INTERVAL TIMER (SINT) 9545 (CONTROL STATE OPERATOR)

This operator places the 11 low-order bits of the B register into the Interval Timer register, and arms the timer. The Interval Timer decrements each 512 microseconds. The processor is interrupted when the timer reaches 0 and is still armed. The Interval Timer is disarmed when the processor is interrupted by an external interrupt.

The operand used to set the Interval Timer is integerized before the 11 low-order bits are used. If the operand can not be integerized, an integer overflow interrupt is set and the operation is terminated.

ENABLE EXTERNAL INTERRUPTS (EEXI) 9546

This operator causes the processor to enter normal state, allowing it to respond to external interrupts. This is accomplished by setting the IIHF flip flop to 0.

DISABLE EXTERNAL INTERRUPTS (DEXI) 9547

This operator causes the processor to ignore external interrupts. This is accomplished by setting the IIHF to 1 and entering control state.

SCAN OPERATORS

The scan operators communicate between the B 6800 data processor and the multiplexor, and between the data processor and external subsystems of the B 6800 system.

The scan in functions read information from the multiplexor or external subsystem to the top of stack registers in the data processor. The scan out functions write information from the top of stack registers in the data processor to the multiplexor or to the external subsystems.

Parity is checked during transmission of both address and information and a scan-bus parity error interrupt is generated if the check fails.

Scan In (SCNI) 954A

Scan In uses the A register to specify the type of input required. The input data is placed in the B register. The A register is empty and the B register full at the completion of the operation. Refer to section 5 of this manual for the format of the function and data words for scan-in operations.

Read Time-Of-Day Clock

The read time of day scan-in operation is used to transfer the current value of the time of day register in the multiplexor, to the data processor. The current value of the multiplexor time of day register is not altered in any way and proceeds to count upward in the normal manner.

Read Interrupt Mask

This operation is used to transfer the current value of the interrupt mask register to the data processor top-of-stack register. The current value of the interrupt mask register in the multiplexor register is not changed by this operation.

Read Interrupt Register

This operation is used to transfer the current value of the interrupt register to the data processor top-of-stack register. The process of transferring the value of the interrupt register from the multiplexor to the data processor resets the highest priority interrupt.

Read Interrupt Literal

This operation is used to transfer a word indicating the highest priority interrupt, from the multiplexor to the data processor.

Interrogate Peripheral Status

This operation is used to transfer the current value of one of nine status vector words from the multiplexor to the top-of-stack register in the data processor.

Interrogate Peripheral Unit Type

This operation transfers a peripheral unit type word from the multiplexor to the top-of-stack register in the data processor.

Interrogate IO Path

This operation transfers a word of path availability data from the multiplexor to the top-of-stack register in the data processor.

Interrogate IO Path Address

This operation transfers a word of path address data from the multiplexor to the top-of-stack register in the data processor.

Interrogate IO Path Address Override

This operation transfers a word of path address override data from the multiplexor to the top-of-stack register in the data processor.

Read Scratch Pad Word

This operation transfers the contents of one word of scratch pad memory to the top word in the data processor top of stack register.

Read Processor Time Counter

This operation transfers a word containing the current value of the processor time counter to the data processor top-of-stack register. The value of the processor time counter in the multiplexor is reset to zero.

Scan Out (SCNO) 954B

The scan out operation causes the multiplexor to sense a function code in the top-of-stack register of the data processor. The micro code of the multiplexor causes other data from the top 2 words in the stack to be transferred to the multiplexor logic circuits, through use of the Z5 bus. At the conclusion of the scan-out operator the top 2 words of the stack are deleted from the stack.

Set Time of Day

This operation is used to set a value into the time of day register in the multiplexor.

Set Interrupt Mask

This operation is used to set a value into the interrupt mask register of the multiplexor.

Set Pseudo Busy

This operation is used to set/reset the state of one of twenty peripheral control path pseudo busy flip flops.

Initiate IO Device (Control State Only)

This operation causes the multiplexor to initiate an IO device.

Refer to section five of this manual for the format of the IIOWD and IOAD words.

Initiate IO Device Path Address

This scan out operation is similar to the INITIATE IO DEVICE operation which was previously defined in this section of this manual.

The difference between the INITIATE IO DEVICE, and the INITIATE IO DEVICE PATH ADDRESS is that the multiplexor will only initiate the IO device through the specified path.

Initiate IO Device Path Address Override

This operation is similar to the INITIATE IO DEVICE, and INITIATE IO DEVICE PATH ADDRESS scan out operations that were defined previously in this section of this manual.

The difference between the INITIATE IO PATH ADDRESS, and the INITIATE IO PATH ADDRESS OVERRIDE operations is that the initiate IO device with override will disregard the state of the pseudo busy flip flop.

READ PROCESSOR IDENTIFICATION (WHOI) 954E

This operator places a word containing the value of the processor ID register in the A register of the data processor.

The format of the word that is placed in the A register of the data processor is shown in figure 8-1. At the conclusion of the WHOI operator the A register is marked full.

OCCURS INDEX (OCRX) 9585

This operator places the following in the B register: a new index value calculated from the Index Control Word (ICW) in the A register and the operand in the B register (figure 8-2).

The index word in the B register is integerized. If the index is greater than the maximum integer value (549,755,813,887), the integer overflow interrupt is set and the operation terminated.

The LENGTH field of the ICW [47:16] is multiplied by the index value [15:16] minus 1, and that value is added to the OFFSET field of the ICW. This result is the new index. The A register is marked empty and the B register is marked full.

If either the ICW or the operand has a value of 0, the invalid index interrupt is set and the operation is terminated.

If the index value is less than 0 or greater than the SIZE field [31:16] of the ICW, the invalid index interrupt is set and the operation is terminated.

INTEGERIZE, ROUNDED, DOUBLE-PRECISION (NTGD) 9587

This operator creates from the operand in the B register a double-precision, rounded integer in the B register. The B register is marked full. If the word in the B register at the start of this operator is not an operand, the invalid operand interrupt is set and the operation is terminated.

If the operand in the B register is larger than 8 ↑ 26-1 in absolute value, the integer overflow interrupt is set and the operation is terminated.

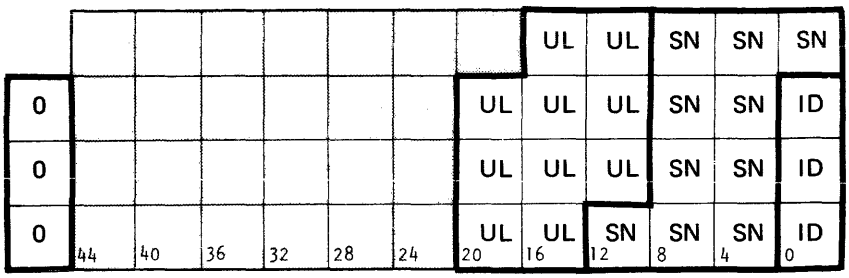
The B register is marked as a double-precision operand (tag bits set to 010) and the exponent is set to 13.

LEADING ONE TEST (LOG2) 958B

This operator locates the most significant one-bit of the word in the B register and places the location of that bit into the B register (bit number + 1).

If a one-bit is not sensed, the B register is set to all 0's.

The B register is marked full.



50:3 = TAG FIELD
47:25 = NOT USED
22:10 = THE UNIT DESIGN (ERL) LEVEL OF THE CPU.
THIS FIELD IS A BINARY NUMBER WHICH IS DERIVED FROM
A FOREPLANE CONFIGURATION PLUG-ON JUMPER.
ADAPTER OF THE CPU
12:10 = THE SERIAL NUMBER OF THE CPU.
THIS FIELD IS A BINARY NUMBER WHICH IS DERIVED FROM A
FOREPLANE CONFIGURATION PLUG-ON JUMPER ADAPTER
OF THE CPU
2:3 = THE PROCESSOR ID NUMBER OF THE CPU.
THIS FIELD IS A BINARY NUMBER WHICH IS DERIVED FROM A
FOREPLANE CONFIGURATION PLUG-ON JUMPER ADAPTER
OF THE CPU.

MV 1652

Figure 8-1. WHOI Operator Returned Word

NORMALIZE (NORM) 958E

This operator performs normalization of the operand in the top of stack. The normalized operand is left in the B register at the conclusion of the NORM operator, and the B register is marked full. Normalization is defined in section 2 of this manual.

MOVE TO STACK (MVST) 95AF

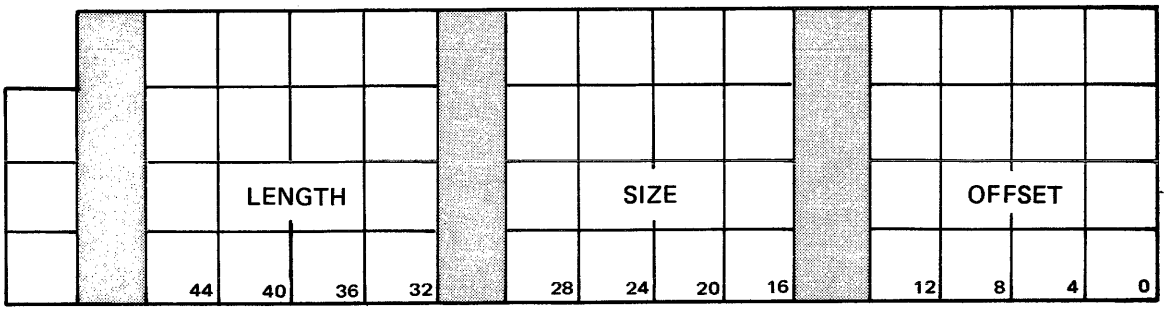
This operator causes the environment of the processor (or addressing space) to be moved from the current stack to the program stack specified by the operand in the B register.

The operator builds a Top-of-Stack Control Word (TSCW) (Figure 8-3) and places it at the base of the current stack as addressed by the Base-of-Stack Register.

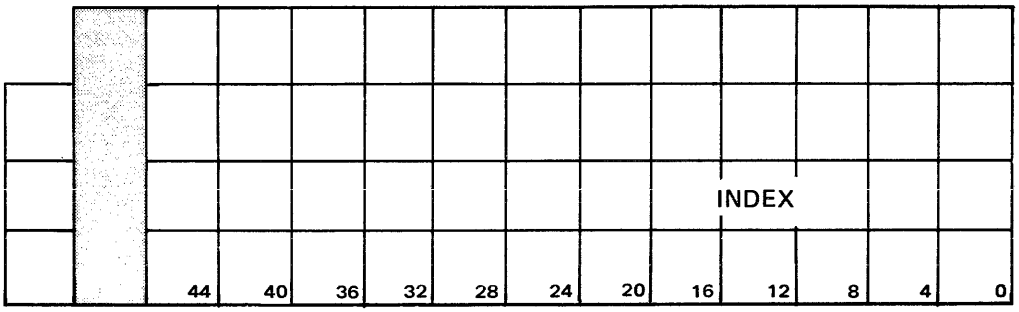
The operand in the B register is integerized and checked against the stack vector for invalid index. The value in the B register is added to the address field of the stack vector Descriptor (at D[0]+2), to address the descriptor for the new stack.

The Data Descriptor for the requested stack is accessed. If the presence bit is "on," the address field is placed into the Base-of-Stack Register. The TSCW is brought up and the stack is marked "active" by storing the processor ID at the base of the stack. The TSCW is distributed and the D registers are updated.

INDEX CONTROL WORD (ICX)



INDEX WORD



MV 1653

Figure 8-2. Index Control Word (ICW) and Index Word

If during the integerization the operand in the B register is too large, the integer overflow interrupt is set and the operation is terminated.

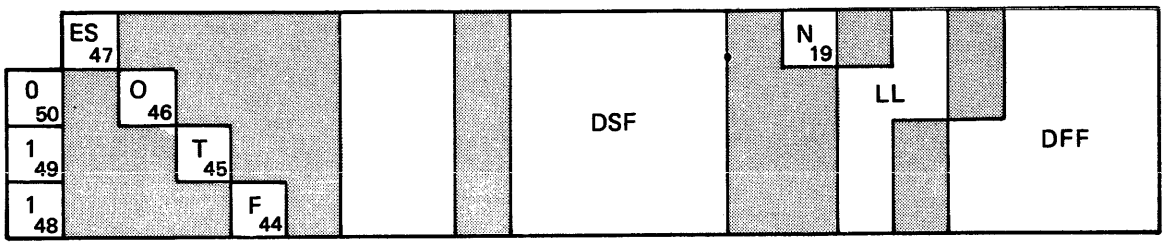
If the index value is less than 0 or greater than the length field of the Data Descriptor for the stack vector array, an invalid index interrupt is set and the operation is terminated.

READ COMPARE FLIP-FLOP (RCMP) 95B3

This operator reads the state of the CMPF flip flop, and creates a single-precision word in the data processor A register. If the CMPF flip flop is in the binary one state, the low order bit (bit zero) of the single precision word in the A register is set. If the CMPF flip flop is in the binary zero state, the low-order bit of the A register is reset. The A register is marked full at the conclusion of the operation.

SET TAG FIELD (STAG) 95B4

This operator sets the tag field (bits 50:3) in the B register to the value of bits 2:3 of the operand in the A register. At the completion of the operation, the A register is marked empty and the B register is left full.



ES – EXTERNAL SIGN FLIP FLOP
O – OVERFLOW FLIP FLOP
T – TOGGLE, TRUE-FALSE FLIP FLOP
F – FLOAT FLIP FLOP
DSF – DELTA S-REGISTER FIELD; VALUE OF rS RELATIVE TO BOSR
N – NORMAL-CONTROL STATE FLIP FLOP
LL – ADDRESSING LEVEL
DFF – DELTA F-REGISTER FIELD; VALUE OF rF RELATIVE TO rS

MV 1654

Figure 8-3. Top-of-Stack Control Word (TSCW)

READ TAG FIELD (RTAG) 95B5

This operator replaces the word in the A register with a single-precision operand equal to the tag field of that word. The tag bits are placed in bits 2:3. The A register is marked full.

ROTATE STACK UP (RSUP) 95B6

This operator permutes the top three operands of the stack so that the first operand has become the second, the second has become the third, and the third has become the first (see figure 8-4).

ROTATE STACK DOWN (RSDN) 95B7

This operator permutes the top three operands of the stack so that the first has become the third, the second has become the first, and the third has become the second (see figure 8-4).

READ PROCESSOR REGISTER (RPRR) 95B8

This operator reads the contents of one of the eight Base registers, eight Index registers or one of the 32 D registers into the A register.

The six low order bits of the A register selects the processor register to be read.

The decoding of these six bits is as follows:

Bits 5:2	= 10	= Index register
Bits 2:3	= 0,	= PIR
	= 1,	= SIR
	= 2,	= DIR

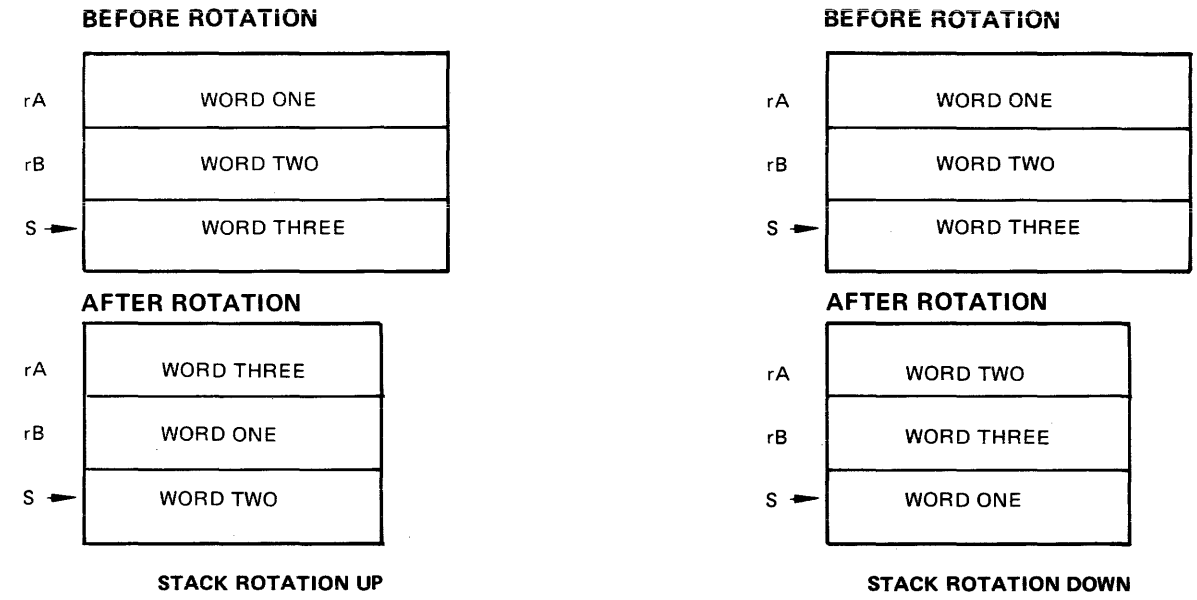


Figure 8-4. Rotate Stack Operations

	= 3,	= TIR, BUF 3
	= 4,	= LOSR
	= 5,	= BOSR
	= 6,	= F
	= 7,	= BUF
Bits 5:2	= 11	= Base register
Bits 2:3	= 0,	= PBR
	= 1,	= IBR
	= 2,	= DBR
	= 3,	= TBR, BUF 2
	= 4,	= S
	= 5,	= SNR
	= 6,	= PDR
	= 7,	= TEMP

If bit 5 is 0, bits 4:5 select the D register equal to the binary value of the bits; i.e., bits 4:5 = 00101 select D register 5.

At the completion of this operation the A register contains the contents of the selected register, and is marked full.

SET PROCESSOR REGISTER (SPRR) 95B9 .

This operator places the contents of the address field of the A register into one of the eight Base registers, eight Index registers or 32 D registers selected by the six low-order bits of the word in the B register.

The decoding of the six low-order bits is the same as in the Read Processor Register operator (RPRR) discussed under the previous heading.

The A and B registers are marked empty.

READ WITH LOCK (RDLK) 95BA

This operator performs the same operation as the Overwrite operator (see section 7), with the exception that the word which was in memory before the overwriting is left in the A register.

COUNT BINARY ONES (CBON) 95BB

This operator counts the number of one-bits in the single-precision (double-precision) operand in the A register. At the completion of the operation, the total count is left in the A register with the register marked full.

LOAD TRANSPARENT (LODT) 95BC

This operator performs a Load operator (see section 7) if the word in the A register is a Data Descriptor or an Indirect Reference Word. If it is neither of these, bits 19:20 of the A register are used as the address to bring an operand to the A register. Copy bit action does not occur.

LINKED LIST LOOKUP (LLLU) 95BD

This operator searches a linked list of words.

The operator starts with an operand in the top of the stack as the index pointer. The second word in the stack is a non-indexed Data Descriptor to the array containing the linked list. The third word in the stack is an operand that is the argument.

The base address of the linked list, the length of the list and the argument value are saved throughout the entire operator process.

The word addressed by the base address plus the index value are read and checked for a value of 0 in the address (Link) portion of the word (0 denotes the end of the linked list). If the link is non-zero, bits 47:28 are compared to the argument value. If the argument of the linked-list word is less than the argument value, the actions described in this paragraph are repeated using the link as the new index.

When the value of the argument field of the linked-list word is equal to or greater than the argument value, the operation is complete. The index pointing to the word whose link points to the argument which satisfies the test is left in the A register and is marked full.

If the value of the link portion of the linked-list word is equal to 0, the A register is set to minus one (-1), and marked full as the operation is completed.

If the index value in the linked list word is greater than the length value from the descriptor, an invalid index interrupt is set and the operation is terminated.

When the first word in the stack at the start of this operator is not an operand an invalid-operand interrupt is set and the operation is terminated.

If the Data Descriptor has been indexed, the invalid-operand interrupt is set and the operation is terminated.

MASKED SEARCH FOR EQUAL (SRCH) 95BE

At the start of this operator, the word in the A register must be a Data Descriptor. The operand in the B register is a 51-bit mask. The Data Descriptor in the A register and the mask in the B register are saved, and the 51-bit argument word is placed into the B register. If the descriptor is indexable (bit 45 equal to 0), 1 is subtracted from the length field. If bit 45 is equal to 1, the data descriptor is already indexed; therefore, that index is the starting value.

The word addressed by the descriptor is placed in the A register and ANDed with the mask word. The result of this AND function is tested to determine if it is identical to the argument word.

If the comparison is not equal, the index field of the descriptor is decreased by 1 and the operation is repeated. If the index field is equal to 0, the A register is set to a minus one value and marked full. The B register is marked empty.

If an equal comparison is made, the A register contains the index pointing at the last word compared and is marked full. The B register is marked empty.

UNPACK ABSOLUTE, DESTRUCTIVE (UABD) 95D1

This operator unpacks a string of four-bit digits into six-bit characters or eight-bit bytes. At the start of the operator, the word in the A register defines the length of the operand in the B register; i.e., the string of digits to be unpacked.

The third word in the stack is a string descriptor addressing the destination of the string.

As the specified number of digits are transferred to the destination (most significant bit first) zone fill is as follows:

1. If the destination size is six-bit (BCL) format, the characters are transferred to the destination with the two zone bits set to 0.

2. If the destination size is eight-bit (EBCDIC) format, the bytes are transferred to the destination string with the four zone bits set to 1111.

3. If the destination size is 0, it is set to eight-bit format and handled as in 2 above.

UNPACK ABSOLUTE, UPDATE (UABU) 95D9

This operator performs an Unpack Absolute operation; at the completion of the operation, the destination pointer is updated and left in the stack.

UNPACK SIGNED, DESTRUCTIVE (USND) 95DO

This operator performs an Unpack Absolute operation, plus an added function if the External Sign flip-flop is set, then a zone of 10 is set in the last character for six-bit or a zone of 1101 is set in the last byte for eight-bit.

If the destination size is four-bit, the first digit position of the destination string is set to 1101 provided the External Sign flip flop is set. If the External Sign flip flop is 0, the first digit is set to 1100.

UNPACK SIGNED, UPDATE (USNU) 95D8

This operator performs an Unpack Signed operation; at the completion of the operation, the destination pointer is updated.

TRANSFER WHILE TRUE, DESTRUCTIVE (TWTD) 95D3

This operator transfers characters from the source string to the destination string for the number of characters specified by the length operand while the stated relationship is met. If the relationship is not met, the transfer is terminated at that point. The relationship is determined by using the source character to index a table. If the bit indexed is a 1, the relationship is true.

The operator uses the top four words in the stack as follows. The top word addresses the table; the second word is the length of the string to be transferred. The third word in the stack is an operand or a descriptor addressing the source string or a single-precision operand which is the source string; and the fourth word in the stack is a descriptor pointing at the destination string.

The table is indexed as follows to obtain the decision bit. The source character is expanded to eight bits, if necessary, by appending two or four leading 0 bits. The three high-order bits of these eight select a word from the table, indexing the table pointer. The remaining five bits of the expanded source character select a bit from this word by their value.

TRANSFER WHILE TRUE, UPDATE (TWTU) 95DB

This operator performs a Transfer While True operation, but updates the source pointer, the destination pointer and repeat count.

If all the characters specified by the length field are transferred, the True False flip flop (TFFF) is set to 1 (true); otherwise it is set to 0 (false).

TRANSFER WHILE FALSE, DESTRUCTIVE (TWFD) 95D2

This operator performs a Transfer While operation and tests for a zero bit in the table.

TRANSFER WHILE FALSE, UPDATE (TWFU) 95DA

This operator performs a Transfer While False operation, but updates the source pointer, the destination pointer, and the repeat count.

If all the characters specified by the length field are transferred, the True/False flip flop (TFFF) is set to 1 (true); otherwise, it is set to 0 (false).

TRANSLATE (TRNS) 95D7

This operator translates the number of characters specified as they are transferred from the source string to the destination string.

The translation uses a table containing the translated characters. The word in the top of the stack is a descriptor that addresses the translation table. The second operand in the stack specifies the length of the string. The third word in

the stack is a descriptor addressing the source string (or an operand which is the source string), and the fourth word in the stack is a descriptor addressing the destination string. The source and destination are updated at the end of the operation.

The translation occurs as follows. The specified string character is used as an index into the table to locate a character. The located character is transferred to the destination string.

The least significant 32 bits of each table word provide four eight-bit characters. The table sizes are as follows:

1. Four-bit digits provide a 4-word table length.

2. Six-bit characters provide a 16-word table length.

3. Eight-bit bytes provide a 64-word table length.

SCAN WHILE GREATER, DESTRUCTIVE (SGTD) 95F2

This operator scans a string while the characters in the source string are greater than a delimiter character or until the number of characters specified have been scanned.

If all the characters have been scanned at the completion of this operation, TFFF is set to 1. If the scan was stopped by the delimiter test before the end of the string, the TFFF is set to 0.

If the delimiter against which the string is compared is equal to the character from the string then the compare flip-flop (CMPF) is set. If the character in the string is less than the delimiter then CMPF flip-flop is reset.

At the start of this operator the delimiter character is right justified in the top word of the stack. The length of the string to be scanned is the second word of the stack. The source pointer is the third word in the stack.

If the second word in the stack is a descriptor, it is the source pointer and the length of the character string is set to 1,048,575 (length field is all ones).

SCAN WHILE GREATER, UPDATE (SGTU) 95FA

This operator performs a Scan While Greater operation and also updates the count and the source pointer. The updated source pointer locates the character that stopped the scan. The number of characters not scanned is placed in the A register, and the register is marked full.

SCAN WHILE GREATER OR EQUAL, DESTRUCTIVE (SGED) 95F1

The operator performs a Scan While operation while the characters in the source string are equal to or greater than the delimiter character. If all the characters have been scanned at the completion of the operation then the TFFF flip-flop is set.

SCAN WHILE GREATER OR EQUAL, UPDATE (SGEU) 95F9

This operator performs a Scan While Greater or Equal operation, but also updates the count and the source pointer.

SCAN WHILE EQUAL, DESTRUCTIVE (SEQD) 95F4

This operator performs a Scan While operation while the characters in the source string are equal to the delimiter character. If all characters are compared then the TFFF flip-flop is set.

If the delimiter against which the string is compared is less than the character from the string then the compare flip-flop (CMPF) is set.

SCAN WHILE EQUAL, UPDATE (SEQU) 95FC

This operator performs a Scan While Equal operation, but also updates the count and the source pointer.

SCAN WHILE LESS OR EQUAL, DESTRUCTIVE (SLED) 95F3

This operator performs a Scan While operation while the characters in the source string are equal to or less than the delimiter character. If all characters are compared then the TFFF flip-flop is set.

SCAN WHILE LESS OR EQUAL, UPDATE (SLEU) 95FB

This operator performs a Scan While Less or Equal operation, but also updates the count and source pointer.

SCAN WHILE LESS, DESTRUCTIVE (SLSD) 95FO

This operator performs a Scan While operation while the characters in the source string are less than the delimiter character.

SCAN WHILE LESS, UPDATE (SLSU) 95F8

This operator performs a Scan While Less operation, but also updates the count and the source pointer.

If the character from the table, against which the string is compared, is equal to the character from the string then the compare flip flop (CMPF) is set.

SCAN WHILE NOT EQUAL, DESTRUCTIVE (SNED) 95F5

This operator performs a Scan While operation while the characters in the source string are not equal to the delimiter character. If all characters are compared then the TFFF flip-flop is set.

SCAN WHILE NOT EQUAL, UPDATE (SNEU) 95FD

This operator performs a Scan While not Equal operation, but also updates the count and the source pointer.

SCAN WHILE TRUE, DESTRUCTIVE (SWTD) 95D5

This operator uses each source character as an index into a table to locate a bit in the same fashion as the transfer while True operators. If the bit located contains the value of one, the relationship is true and the scan continues.

The first word in the stack is a descriptor addressing the table. The second and third words in the stack are the same as for all Scan While operators.

SCAN WHILE TRUE, UPDATE (SWTU) 95DD

This operator performs a Scan While True operation, but also updates the count and the source pointer. The number of characters not scanned is placed in the A register.

SCAN WHILE FALSE, DESTRUCTIVE (SWFD) 95D4

This operator performs a Scan While False operation, except the relation is true if the bit found by indexing into the table contains the value of zero.

SCAN WHILE FALSE, UPDATE (SWFU) 95DC

This operator performs a Scan While False operation, but also updates the count and the source pointer.

SECTION 9

EDIT MODE OPERATION AND OPERATORS

GENERAL

The purpose of the edit mode operators is to perform editing functions on strings of data. The editing functions are those which are normally involved in preparing information for output. They include such operators as move, insert, and skip, in the form of micro-operators in either the program string or in a separate table. In the program string, they are single micro-operators and are entered by use of the execute single micro or single pointer operators (see section 7). If the micro-operators are in a table, the table becomes the program string that is to be executed. This table is entered by means of the table enter edit operators (see section 7), and is exited through the end edit micro-operator as defined later in this section.

If the source or destination data has the memory protect bit (bit 48) equal to one, the segmented-array interrupt is set and the current micro-operator is terminated.

Appendix A of this manual lists the operators in alphabetic order, and appendix B lists the operators in numeric order, by mode.

EDIT MODE OPERATORS

The edit mode operators are described in the following paragraphs of this section.

MOVE CHARACTERS (MCHR) D7

This micro-operator transfers characters from the source string to the destination string.

If this micro-operator is entered by the table enter edit operator (see section 7), the number of characters to be transferred is specified by the syllable following the operator syllable.

If this micro-operator is entered by the execute single micro operator (see section 7), the number of characters to be transferred is specified by the operand in the top of the stack.

MOVE NUMERIC UNCONDITIONAL (MVNU) D6

This micro-operator transfers the four low-order bits of the characters of the source string to the destination string. If the destination string character size is 6 bits (BCL) the zone bits are set to 00. If the destination string character size is 8 bits (EBCDIC), the zone bits are set to 1111.

If this micro-operator was entered by use of the table enter edit operator (see section 7), the number of characters to be transferred is specified by the syllable following the micro-operator syllable.

If this micro-operator is entered by executing the execute single micro operator (see section 7), the number of characters to be transferred is specified by the operand in the top of the stack.

MOVE WITH INSERT (MINS) DO

This micro-operator performs a move numeric unconditional or an insert operation under the control of the Float flip flop.

In table edit mode the second syllable is the repeat value and the third syllable is the character to be inserted under control of the Float flip flop.

In execute single micro mode the repeat field value is the top word of the stack and the insert character is in the syllable following the micro-operator syllable.

If the Float flip flop equals 0 and the numeric portion of the source characters equals zero, the insert character is moved to the destination string.

If the Float flip flop is reset, and the numeric portion of the source character is not equal to zero, then set the Float flip flop, and perform a move numeric unconditional operation.

The number of characters transferred from the source string to the destination string is defined by the repeat value.

MOVE WITH FLOAT (MFLT) D1

In table edit mode the second syllable is the repeat value (the number of characters to transfer). The third, fourth, and fifth syllables are the three insert characters. In single-micro mode, the three insert characters are in the second, third, and fourth syllables.

If the Float flip flop equals 0 and the numeric portion of the character in the source string equals 0, the first-insert character is transferred to the destination string.

If the Float flip flop equals 0 and the numeric portion of the character in the source string is not 0 the Float flip flop is set. If the External Sign flip flop equals 1, the second-insert character is transferred to the destination string. If the External Sign flip flop equals 0, the third-insert character is transferred to the destination string. The numeric version of the source character is then transferred.

If the Float flip flop equals 1, the numeric equivalent of the source character is transferred to the destination.

This operation continues for the number of characters defined by the repeat field value.

This operator can be entered by the Execute Single Micro operator, with the repeat field value in the top word of the stack.

SKIP FORWARD SOURCE CHARACTERS (SFSC) D2

This micro-operator increments the source pointer registers.

If this micro-operator or any of the following Skip micro-operators is entered by the execution of the Execute Single Micro operator, the number of characters to be skipped is specified by the operand in the top of the stack. If entry is by the execution of the Table Enter Edit operators, the number of characters to be skipped is specified by the syllable following the micro-operator syllable.

SKIP REVERSE SOURCE CHARACTERS (SRSC) D3

This micro-operator decrements the source pointer registers.

Also see Skip Forward Source Characters micro-operator, second paragraph.

SKIP FORWARD DESTINATION CHARACTERS (SFDC) DA

This micro-operator increments the destination pointer registers.

SKIP REVERSE DESTINATION CHARACTERS (SRDC) DB

This micro-operator decrements the destination pointer registers.

RESET FLOAT (RSTF) D4

This micro-operator sets the Float flip flop to 0.

END FLOAT (ENDF) D5

This micro-operator transfers the character in the second syllable of this operator to the destination string if the Float flip flop contains a 0 and the External Sign flip flop is 1.

If the Float flip flop contains a 0 and the External Sign flip flop also equals 0, then the character in the third syllable of this operator is transferred.

If the Float flip flop contains a 1, then it is reset and no characters are transferred.

INSERT UNCONDITIONAL (INSU) DC

This micro-operator places an insert character into the destination string for the number of times specified by the repeat value. When entered by a Table Enter Edit operator, the repeat value is in the syllable following the micro-operator syllable, and the insert character is in the next syllable.

If this micro-operator is entered by an Execute Single Micro operator, the character to be inserted is in the second syllable and the repeat value is specified by the operand in the top of the stack.

INSERT CONDITIONAL (INSC) DD

This micro-operator inserts a string consisting of one of two characters into the destination string. The length of the string is given by the repeat value from the table or the stack.

If the Float flip flop contains a 0, the first insert character is inserted into the destination string.

If the Float flip flop contains a 1, the second insert character is inserted into the destination string.

The insert characters follow the repeat value syllable in Table Enter Edit operation or the micro-operator syllable in Execute Single Micro operations.

INSERT DISPLAY SIGN (INSG) D9

This micro-operator places in the destination string the character defined by the syllable following the micro-operator syllable, if the External Sign flip flop is equal to 1.

If the External Sign flip flop is equal to 0, this operator places in the destination string the character defined by the third syllable of this operator.

INSERT OVERPUNCH (INOP) D8

If the External Sign flip flop is equal to 1, this micro-operator places a sign overpunch in the destination string character of either 0 for BCL or 1101 for EBCDIC.

If the External Sign flip flop is equal to 0, the operator leaves the destination string character unaltered.

END EDIT (ENDE) DE

This operator terminates a string of Edit micro-operators in Table Enter Edit operation mode.

The micro program string in the table must end with the End Edit operator.

SECTION 10

VECTOR MODE OPERATORS

GENERAL

The use of Vector Mode provides for an increase in efficiency in the manipulation of arrays. The increase in efficiency is not an automatic feature that applies to all data processor operations. Vector mode makes it possible for certain software compilers, such as ALGOL or FORTRAN, to specify that vector mode rules will apply under controlled conditions.

LIMITATIONS OF VECTOR MODE

Vector mode operations require that the system be operated in control state. This requirement means that a processor performing vector mode operations cannot be interrupted to service external interrupts.

Vector mode operations do not permit segmentation of the arrays. This occurs because presence bit interrupts are disallowed. This limitation requires that the entire extent of the array/arrays must be present in memory while performing vector operations.

Vector mode operation allows the use of other modes and operators in the B 6800 operator set, subject to the following limitations:

a. String operators and edit mode operators are not allowed.

b. No family C operators, except the branching operators (BRTR, BRFL, and so forth) are allowed while operating in vector mode.

c. No operator that pseudo calls a family C operator is allowed while operating in vector mode.

d. The LIT 48, and branch operators are not used while performing in single program word vector mode (VMES) because of the size of the operator codes, in syllables.

Appendix A of this manual lists the operators in alphabetic order, and appendix B lists the operators in numeric order, by mode.

HARDWARE FUNCTIONS

The Vector Mode hardware does the following:

a. Utilizes registers to hold the actual addresses of array elements that are referenced.

b. Uses additional registers to contain the increment values used for altering the addresses (indexing) to reference successive array elements.

c. Uses one register to contain a "count" or length that controls the number of iterations.

d. Provides for cycling through one (single-word mode) or more (multiple-word mode) words of code for each iteration.

e. Introduces new operators for use while in Vector Mode to load and store the top-of-stack, and to control iterating and exiting from Vector Mode.

f. Provides two primary mode operators used to enter Vector Mode.

Seven IC memory locations are used as the registers mentioned above to hold the three absolute addresses, the three corresponding increment values, and the length.

The three addresses are referred to as A, B, and C, respectively.

These registers are loaded automatically from the stack upon execution of either of two Enter Vector Mode operators.

PRIMARY MODE ENTER VECTOR MODE OPERATORS

There are two primary mode operators used for vector operations in the B 6800 Data Processor. These operators are as follows:

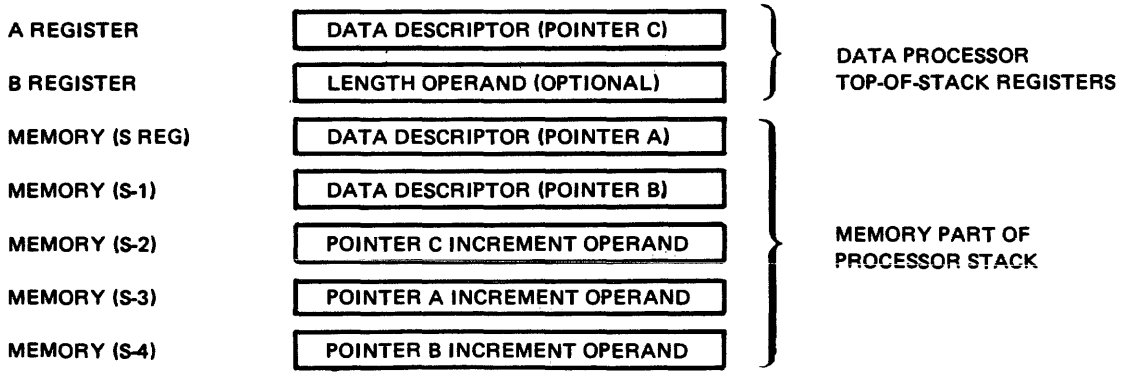
Family	Mnemonic	Hexadecimal Code	Operator Description
G	VMES	EF	Vector Mode Enter Single
G	VMEM	E7	Vector Mode Enter Multiple

Both of these operators perform a similar function. They use the string-operator enter edit mode logic procedure to initialize the data processor for Vector Mode operations. The difference between these two operators is the number of words of Vector Mode program code that are required. If the Vector Mode operators in the program code-stream consist of one word of program code, then the "VMES" operator is used. If more than one word of program code is required while in Vector Mode, then the "VMEM" operator is used. The compilers that allow Vector Mode operations contain code to determine which of these two "Enter Vector Mode" operators is to be used.

ENTER VECTOR MODE OPERATION

An entry into Vector Mode operations occurs when the VMES (EF), or VMEM (E7) operator is executed from the processor P register. Prior to entering Vector Mode, the processor stack must be properly configured to perform Vector operators.

The processor registers and the operating stack must have the following format:



Before entering Vector Mode, the values to be stored in IC memory must be placed in the stack. LENGTH specifies the number of iterations through the code to be executed while in Vector Mode, usually the number of elements in the arrays being manipulated. The presence of a LENGTH value in the stack is indicated by bit 44=1 in Pointer C. Should bit 44=0, a default LENGTH of 2^{20-1} is stored in the LENGTH register. Bit 44 (segmented bit) must be OFF in Pointer A and Pointer B. The software ascertains that bit 44 is ON in Pointer C before using it to indicate the presence of a LENGTH value.

The seven parameters are inserted in IC memory as follows:

Register	Vector Mode Contents of Register
BRS3	Pointer C [19:20] (or Pointer C [39:20] plus [19:20] if I* = 1)
BRS7	LENGTH [19:20] (or 2^{20}-1)
BRS1	Pointer A [19:20] (or Pointer A [39:20] plus [19:20] if I* = 1)
BRS2	Pointer B [19:20] (or Pointer B [39:20] ** plus [19:20] if I* = 1)
IRS3	Pointer C increment [19:20]
IRS1	Pointer A increment [19:20]
IRS2	Pointer B increment [19:20]

*I is the indexed bit, bit 45 in the descriptor.
**Use [35:16] if the size field is not equal to zero.

The Enter Vector Mode operator may be terminated by one of the following interrupts:

Type of Interrupt	Cause of the Interrupt
a. Invalid Op:	Pointer A, B or C not tagged as a data descriptor or Pointer A or B has bit 44=1.
b. Memory Protect:	Pointer A is read only (bit 43=1).
c. Presence Bit:	Pointer A, B or C has bit 47=0.

At the conclusion of the enter Vector Mode flow, the IC memory is configurated as follows:

Register Name	Contents of the Register
SIR	The value of the "A" increment
DIR	The value of the "B" increment
TIR	The value of the "C" increment
SBR	The base address of pointer "A"
DBR	The base address of pointer "B"
TBR	The base address of pointer "C"
TEMP	The value of the length operand

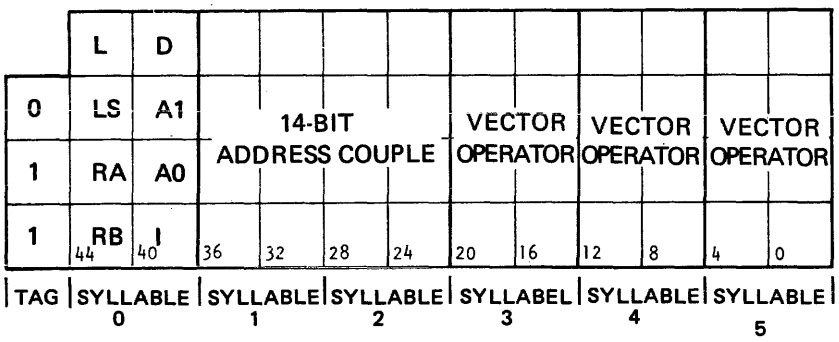
The word in the P register at the end of the Enter Vector Mode flow contains the Vector Operators that are to be executed. The PSR register is equal to zero, and thus specifies that the first Vector Mode operator commences in syllable zero.

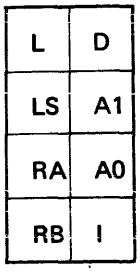
If the entry to Vector Mode is the single-word mode entry VMES operator, the single word of code following that entry is held in the P Register (program word fetching is inhibited) and executed a number of times equal to the LENGTH parameter. Each time the word is executed, LENGTH is decremented by one until it becomes zero. Then Vector Mode is exited and normal operation continues with the next word of code in sequence.

VECTOR STACK OPERATORS

Vector Stack operators are a group of twenty-eight operators with a common syllable format. Variations of this syllable provide the capabilities of storing or loading the top-of-stack with a single- or double-precision operand and choosing whether or not to increment the pointer.

P REGISTER





A VECTOR OPERATOR OCCUPIES ONE THROUGH THREE SYLLABLES OF THE P REGISTER. THE VECTOR BRANCH OPERATOR (VEBR, HEX CODE EE), AND THE FTCH/STOR OPERATORS USE THREE SYLLABLES. ALL OTHER VECTOR OPERATORS USE A SINGLE SYLLABLE

The format of the Vector Operator syllable is as follows:

Bit	Description
L	The most significant bit in the Vector Operator, equals one if a length factor is passed to the Vector Stack upon entering Vector Mode; otherwise, L equals zero.
LS	Bit is OFF (0) for a Top-of-Stack Load operator and ON (1) for a Top-of-Stack Store operator.
RA	If a Memory Protect Interrupt is sensed and no LENGTH is passed to the Vector Mode and RA=0, the top-of-stack word is deleted. If RA=1, the top-of-stack word is not deleted.
RB	Same as the RA bit except that it governs the action taken on the second word of the stack.
D	Double-precision bit. If D=0, load or store single-precision operand (Fam G). If D=1, load or store double-precision operand (Fam H).
A1, A0	Selects the IC Memory Address Register.

A1	A0	
0	0	Load from Pointer A (BRS1)
0	1	Load from Pointer B (BRS2)
1	0	Load from Pointer C (BRS3)

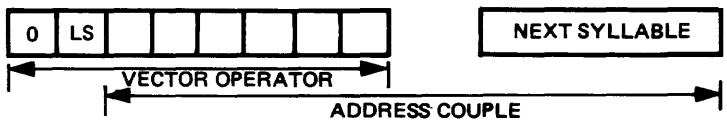
I	When I equals 1, the pointer used for the memory address is increased by its corresponding pointer increment following the load or store operator. When I equals 0, the pointer increment is inhibited.

VECTOR MODE OPERATOR CODES

The twenty-seven Vector Mode operators are identified as follows:

FAMILY		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
G	E	LDA	LDAI	LDB	LDBI	LDC	LDCI	VMEX		DLA	DLAI	DLB	DLBI	DLC	DLCI	VEBR	
H	F	STA	STAI	STB	STBI	STC	STCI			DSA	DSAI		DSBI	DSC	DSCI	NOOP	NVLD

Two other operators are used to load/store the top-of-stack from/to an address couple. They are enabled only when a LENGTH is passed by the Vector Mode entry. Their format is as follows:



The address couple is formed from the low-order six-bits of the vector-operator, and the next operator-syllable, which are concatenated to form a fourteen-bit address-couple.

Where; LS=0 then load (FTCH operator), or when
LS=1 then store (STOR operator).

The A Register is loaded from (or stored into) the memory location determined by the normal address-couple decoding convention (same as Value Call).

VECTOR OPERATORS

The following is a list of Vector Stack operators.

Operator	Hex OP-Code	Description
Load A	E0	The stack is adjusted (0,2) and the single-precision word selected by Pointer A (BRS1) is loaded into the top-of-stack.
Load B	E2	The stack is adjusted (0,2) and the single-precision word selected by Pointer B (BRS2) is loaded into the top-of-stack.
Load C	E4	The stack is adjusted (0,2) and the single-precision word selected by Pointer C (BRS3) is loaded into the top-of-stack.
Load A – Increment	E1	The stack is adjusted (0,2) and the single-precision word selected by Pointer A (BRS1) is loaded into the top-of-stack. Pointer A is increased by its increment (IRS1) following the transfer.
Load B – Increment	E3	The stack is adjusted (0,2) and the single-precision word selected by Pointer B (BRS2) is loaded into the top-of-stack. Pointer B is increased by its increment (IRS2) following the transfer.
Load C – Increment	E5	The stack is adjusted (0,2) and the single-precision word selected by Pointer C (BRS3) is loaded into the top-of-stack. Pointer C is increased by its increment (IRS3) following the transfer.
Store A	F0	The stack is adjusted (1,2) and the single-precision word in the top-of-stack is stored in the location given by Pointer A (BRS1).
Store B	F2	The stack is adjusted (1,2) and the single-precision word in the top-of-stack is stored in the location given by Pointer B (BRS2).
Store C	F4	The stack is adjusted (1,2) and the single-precision word in the top-of-stack is stored in the location given by Pointer C (BRS3).
Store A – Increment	F1	The stack is adjusted (1,2) and the single-precision word in the top-of-stack is stored in the location given by Pointer A (BRS1). Pointer A is increased by its increment (IRS1) following the transfer.

Operator	Hex OP-Code	Description
Store B – Increment	F3	The stack is adjusted (1,2) and the single-precision word in the top-of-stack is stored in the location given by Pointer B (BRS2). Pointer B is increased by its increment (IRS2) following the transfer.
Store C – Increment	F5	The stack is adjusted (1,2) and the single-precision word in the top-of-stack is stored in the location given by Pointer C (BRS3). Pointer C is increased by its increment (IRS3) following the transfer.
Double Load A	E8	The stack is adjusted (0,2) and the double-precision word selected by Pointer A (BRS1) is loaded into the top-of-stack.
Double Load B	EA	The stack is adjusted (0,2) and the double-precision word selected by Pointer B (BRS2) is loaded into the top-of-stack.
Double Load C	EC	The stack is adjusted (0,2) and the double-precision word selected by Pointer C (BRS3) is loaded into the top-of-stack.
Double Load A – Increment	E9	The stack is adjusted (0,2) and the double-precision word selected by Pointer A (BRS1) is loaded into the top-of-stack. Pointer A is increased by its increment (IRS1) following the transfer.
Double Load B – Increment	EB	The stack is adjusted (0,2) and the double-precision word selected by Pointer B (BRS2) is loaded into the top-of-stack. Pointer B is increased by its increment (IRS2) following the transfer.
Double Load C – Increment	ED	The stack is adjusted (0,2) and the double-precision word selected by Pointer C (BRS3) is loaded into the top-of-stack. Pointer C is increased by its increment (IRS3) following the transfer.
Double Store A	F8	The stack is adjusted (1,2) and the double-precision word in the top-of-stack is stored in the location given by Pointer A (BRS1).
Double Store B	FA	The stack is adjusted (1,2) and the double-precision word in the top-of-stack is stored in the location given by Pointer B (BRS2).
Double Store C	FC	The stack is adjusted (1,2) and the double-precision word in the top-of-stack is stored in the location given by Pointer C (BRS3).
Double Store A – Increment	F9	The stack is adjusted (1,2) and the double-precision word in the top-of-stack is stored in the location given by Pointer A (BRS1). Pointer A is increased by its increment (IRS1) following the transfer.
Double Store B – Increment	FB	The stack is adjusted (1,2) and the double-precision word in the top-of-stack is stored in the location given by Pointer B (BRS2). Pointer B is increased by its increment (IRS2) following the transfer.
Double Store C – Increment	FD	The stack is adjusted (1,2) and the double-precision word in the top-of-stack is stored in the location given by Pointer C (BRS3). Pointer C is increased by its increment (IRS3) following the transfer.

Operator	Hex OP-Code	Description
Vector Branch	EE	A three-syllable operator where the two syllables following the operator contain a branch address. If the length count is > 0 the length count is decremented by one, and the program continues at the next syllable following the address. If the length is equal to zero, Vector Mode is exited by fetching the program word specified by the branch address.
Vector Mode Exit	E6	Allows the program to exit from Vector Mode, to Primary Mode.

VECTOR BRANCH AND VECTOR EXIT OPERATORS

When the entry to Vector Mode is the multiple-word type (VMEM operator), whatever code that follows it is executed under Vector Mode rules. The two Vector Mode operators explained below are used only in conjunction with the VMEM operator.

a. Vector Mode Exit operator (VMEX) causes the program to exit from vector mode, and return to normal mode operations.

b. Vector Branch (VEBR) is a three syllable operator. The two syllables following the operator code contain the branch address. The Vector Branch operator examines LENGTH. If it is greater than zero, LENGTH is decremented by one, the next two program syllables containing the branch address are skipped, and the program is resumed at the following syllable. If the examined LENGTH is zero, Vector Mode is exited, and normal mode operation commences with the program word located by the branch address.

SECTION 11

PERIPHERAL DEVICES AND CONTROLS

GENERAL

Section one of this manual defines the types of peripheral devices that are operated as input or output devices of the B 6800 system. Section five defines the functions of the multiplexor in controlling the operations of the B 6800 system peripheral devices.

This section will define the unit control field of the IOCW, and the IO descriptor formats that are used by the multiplexor to initiate and control IO devices in the B 6800 system. This section will define the IO result descriptor that is returned to the multiplexor at the conclusion of an IO operation. This section will also indicate how the multiplexor uses the information from the IO descriptor and the IO result descriptor to format interrupt parameters (P1, P2, and P3) which are present in the interrupt stack at the end of an IO operation.

TYPICAL INPUT OUTPUT DEVICE SYSTEM OPERATION

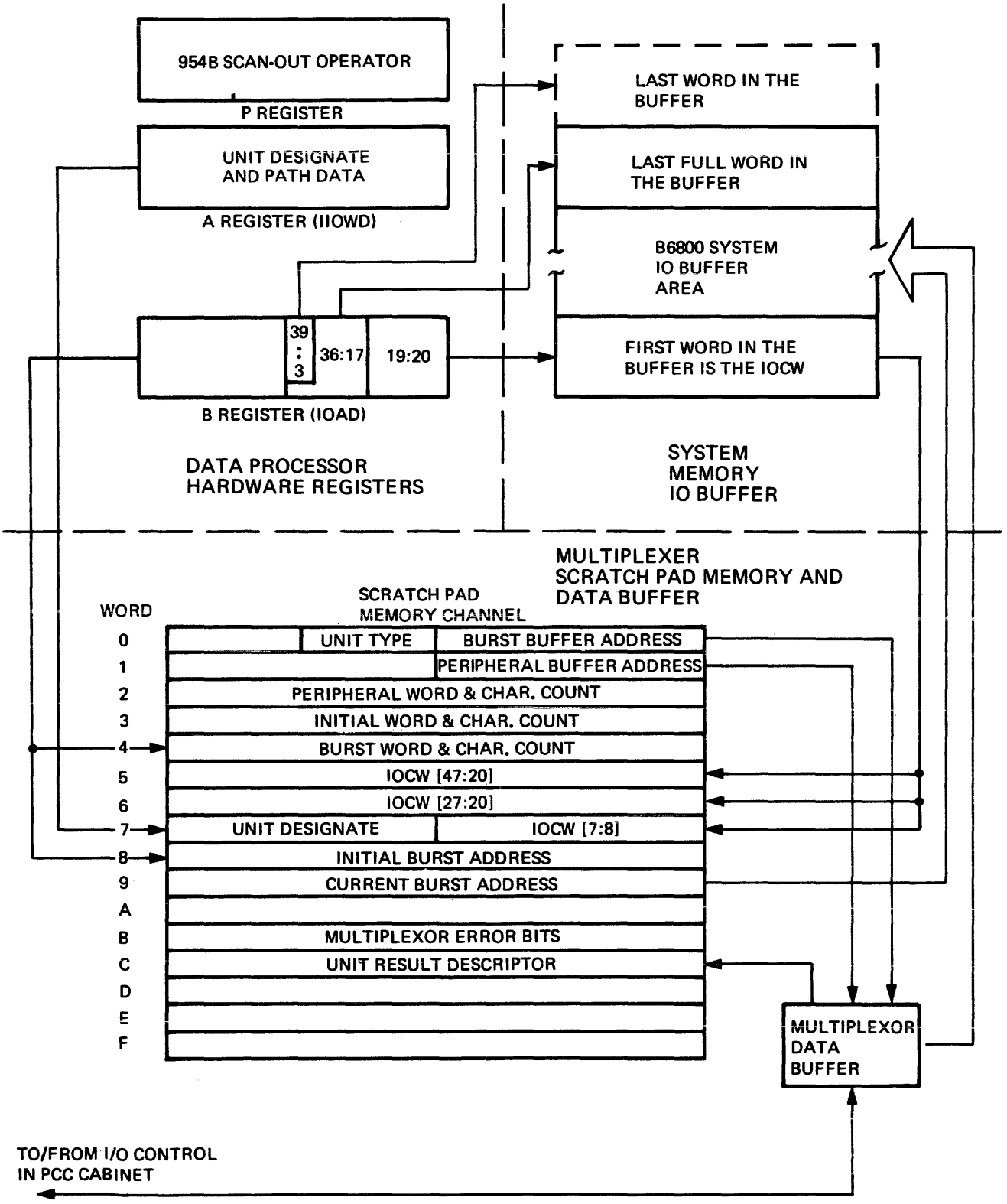
A typical IO operation (refer to figure 11-1) in the B 6800 system is initiated by a SCAN-OUT (initiate IO device) operation. The scan-out operation passes data to the multiplexor that defines what IO device is to be initiated, (IIOWD), and the address of a buffer area in system memory that is to be used for the IO operation (IOAD). The multiplexor performs a read memory operation, using the address of the first word in the IO buffer area. This word contains the IOCW. The data contained in the IIOWD, IOAD, and the IOCW, is stored in the multiplexor scratch pad memory, for use during the remainder of the IO operation.

During an initiate IO cycle an OP code, variant characters and a beginning address, if applicable, are generated by the multiplexor and passed to a peripheral control unit. The OP code, variant characters, and beginning address define the operation that is to be performed, the optional characteristics of the IO device that are to be used for the operation, and, in the case of a disk or pack storage device, a beginning file address. After the IO descriptor has been passed to the IO control, the multiplexor issues a STCB (Start Channel Bus) signal to the IO control, and the initiate IO cycle is completed.

After the IO control has received the STCB signal, it will request service cycles from the multiplexor when a transfer of data is required. During a service cycle, the multiplexor will transfer the data to/from its internal data buffer.

Multiplexor burst cycles are performed as needed to transfer data between main memory and the multiplexor data buffer.

The transfer of data to/from the IO control can be terminated by either the multiplexor, or the IO control.



MV 1656

Figure 11-1. Input-Output Operation Cycles

Regardless of how an IO operation is terminated, the IO control always passes an IO result descriptor to the multiplexor at the end of the IO data transfer. This result descriptor is saved in scratch pad memory. At the conclusion of the IO operation, the multiplexor raises the external interrupt signal to the interrupt controller.

The interrupt controller initiates an interrupt procedure and the multiplexor places the three interrupt parameters (P1, P2, and P3) in the interrupt procedure stack. After the three parameters are placed in the interrupt stack, the interrupt controller causes the interrupt procedure of the MCP to be executed. The interrupt procedure analyzes the three parameters in the stack, and determines what, if any, action is to be taken. After the interrupt has been handled properly, the processor returns to performing the procedure that was in progress when the interrupt occurred.

INTERRUPT STACK PARAMETERS

Figure 11-2 shows the order of the three parameters that are left on the top of the data processor stack when an IO finished interrupt is present. The data in these three parameters is inserted into the data processor hardware registers by the micro-program logic of the multiplexor. The IO finished interrupt from the multiplexor causes the data processor to interrupt any procedure that is in process, providing that the data processor is operating in normal state. If the data processor is not operating in normal state, the interrupt controller will hold the interrupt until the data processor returns to normal state, at which time the interrupt will be handled.

P1 PARAMETER

Bit 27 of the P1 parameter is true if the operating system is a B 6800. Bit 20, and bit zero of the P1 parameter are true for an external interrupt.

Bits 7:4 are the interrupt literal value. These bits have the following significance:

Value	Meaning
1111	status change interrupt.
0001	interrupt from data communications.
0010	interrupt from data communications processor number two.
0011	interrupt from data communications processor number three.
0100	interrupt from data communications processor number four.
0110	interrupt from bus interface control number one.
0111	interrupt from bus interface control number two.
1000	multiplexor error.
1001	IO finished interrupt.

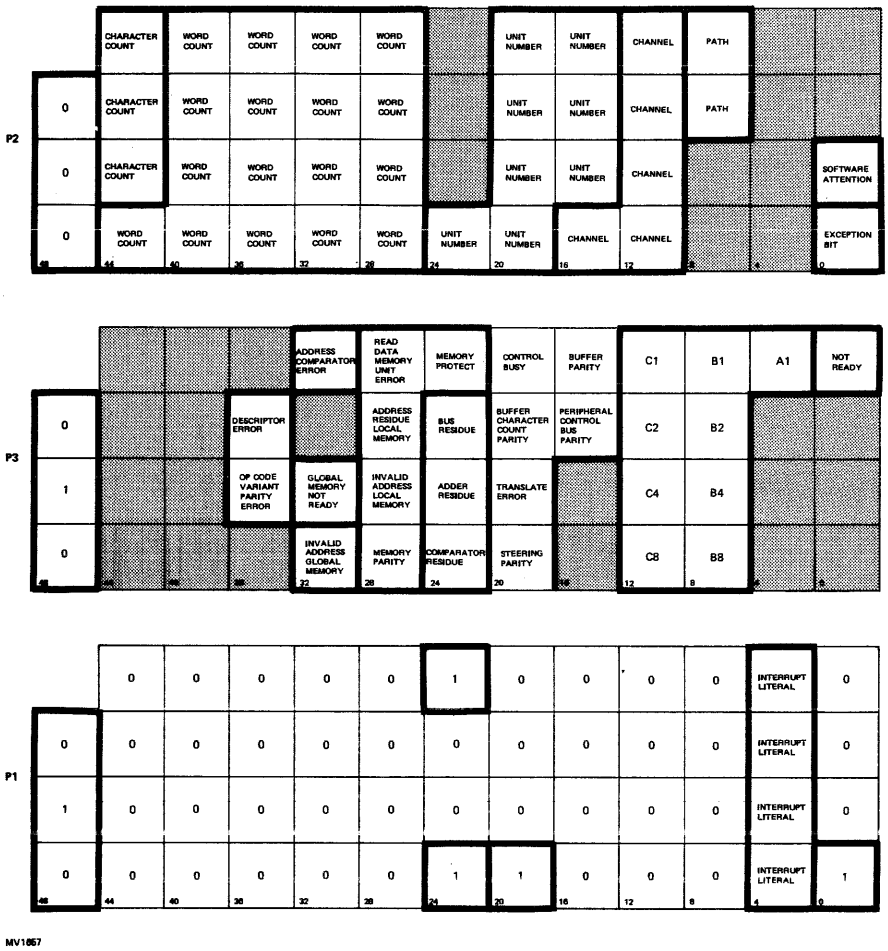


Figure 11-2. Finished Interrupt Stack Parameters

The P1 parameter is the first word of the two words of a double-precision operand in the data processor stack. The second word of the two words of the same double-precision word is the P3 parameter. At the end of the interrupt controller flow, the P1 parameter is present in the data processor B register, and the P3 parameter is present in the data processor Y register.

The data processor interrupt controller always releases control by performing a pseudo-call on the enter (ENTR, AB) operator. The ENTR operator causes the interrupt parameters that are present in the top of stack hardware registers to be pushed down into the memory portion of the stack. After the ENTR operator has been executed, the P1 parameter is located in the stack, at the address indicated by the value of the S register, minus two.

P3 PARAMETER

The P3 parameter is used to indicate any errors that may have occurred in the multiplexor, peripheral control, or on the peripheral control bus during the peripheral device operation. The P3 parameter also will indicate any error that was encountered on the memory bus or in the memory address adder circuits, during a burst cycle.

The significance of the bits in the P3 parameter are as follows:

Bits	Significance
[17:11] and [3:1]	The unit error field ([17:11]) and the not ready bit [3:1]. Various bit configurations in this field are used by different IO control types to indicate an error condition in the IO device, or in the IO control. This field does not mean the same thing for all types of peripheral devices, and therefore must be interpreted according to the type of device that is connected to the multiplexor channel.
23:6	The control error field. Various bits in this field are used to indicate errors in the control logic of the multiplexor. Each of the bits in this field is used to represent an error condition in a major functional group of the multiplexor.
26:3	The address adder error field. These three bits are used to indicate an address adder, residue adder, or IC memory register error. Any one of these three bits being true may also indicate that an error exists in the Z8 or Z9 bus, or in the special address bus that passes between the multiplexor and the data processor.
32:6	The memory error field. These six bits are used to indicate errors that are detected by the memory control logic of the CPU.
33	The global store not ready bit. This bit is used to show that a global memory operation was attempted, and the global memory that was addressed was not ready.
35	The address compare error. This bit is an extension of the address adder error field in bits 26:3.
38:2	The OP CODE or VARIANT CHARACTER PROM parity error bit, and the descriptor error bit. These bits are an extension of the control error field in bits 23:6.

P2 PARAMETER

The P2 parameter is used to indicate the unit number of the peripheral device that was utilized. This parameter also indicates the multiplexor path that was used to access the peripheral device, the length of the IO operation, whether a software attention bit was set, and whether an exception occurred during the IO operation. The fields of the P2 parameter are as follows:

Bits	Significance
0	The exception bit. This bit is set if an exception was detected during the IO operation. The placement of this bit in

Bits	Significance
	the interrupt stack configuration allows the system software interrupt handling procedure to determine if an error occurred during the IO operation.
1	The software attention bit. If bit number 45 of the IOCW that was used to initiate the IO device operation was set, then the software attention bit is set in the P2 parameter. The software attention bit is used to flag certain IO operations for the system software. Examples of the types of IO operations that the system software may flag are spacing over tape records, or changing from one train code to another train code on train printer devices.
11:2	IO path field.
16:5	The channel number field. The value of the five bit channel number (one of twenty multiplexor channels) is identified in the P2 parameter.
24:8	The unit number field. The unit number (one of 256 possible IO device identification numbers) is identified in the P2 parameter.
47:20	The character count and word count field. This field indicates the amount transferred to/from the peripheral control.

PERIPHERAL CONTROLS

All peripheral devices require a unique peripheral control. These controls are located in the peripheral control cabinet, and perform an interface function for the multiplexor.

PERIPHERAL CONTROL BUS

Control and information signals are transferred between the multiplexer and the peripheral control cabinet through a peripheral control bus.

The peripheral control bus contains sixteen data lines. These sixteen lines transmit two eight-bit characters of data bidirectionally, between the multiplexor, and the peripheral control cabinet. All peripheral controls in both of the peripheral control cabinets share the sixteen data lines, and thus, only one peripheral control can communicate with the multiplexor at any one time. Certain peripheral bus control signals that are common to all peripheral controls (such as the peripheral bus parity signal, and the Start Channel Signal) are also routed on the information bus lines.

Control signals that are unique to each peripheral control (the access request and access granted signals, the control busy signal, and the channel designate level signal) do not share common lines on the peripheral bus. These signals have specific lines on the peripheral bus assigned to them.

The multiplexor initiates an IO by passing an OP code, variant characters, a unit number, and in the case of a disk or pack device a file address value, to the particular peripheral control that is associated with the IO device. This control information is generated in the multiplexor by the OP code and variant character generator circuits, and is passed to the peripheral control cabinet through the peripheral bus information lines. The multiplexor can direct the control information to a particular IO control because of the four unique signals for each channel (ARL, AGL, BUSY, and CDL).

The transfer of control information from the multiplexor to an IO control is synchronized during the initiate IO cycle by the multiplexor controlling signals to each IO control. Each IO control contains a sequence counter that steps through its initial sequence counts in response to control signals from the multiplexor. After the initiation cycle is completed, the multiplexor will send a Start Channel Bus (STCB) signal to the IO control, and thereafter the IO control will proceed through its sequence counts in a self-initiated manner.

When an IO control requires information exchange with the multiplexor, it initiates a service cycle in the multiplexor by raising its ARL (Access Request Level) level. The multiplexor will respond to the ARL level by raising the AGL (Access Granted Level) signal when the IO control is to use the peripheral bus. With the communication linkage established, the IO control and multiplexor can begin data transfers. The multiplexor knows which IO control raised its ARL line, and therefore which data buffer to interface with the peripheral control bus.

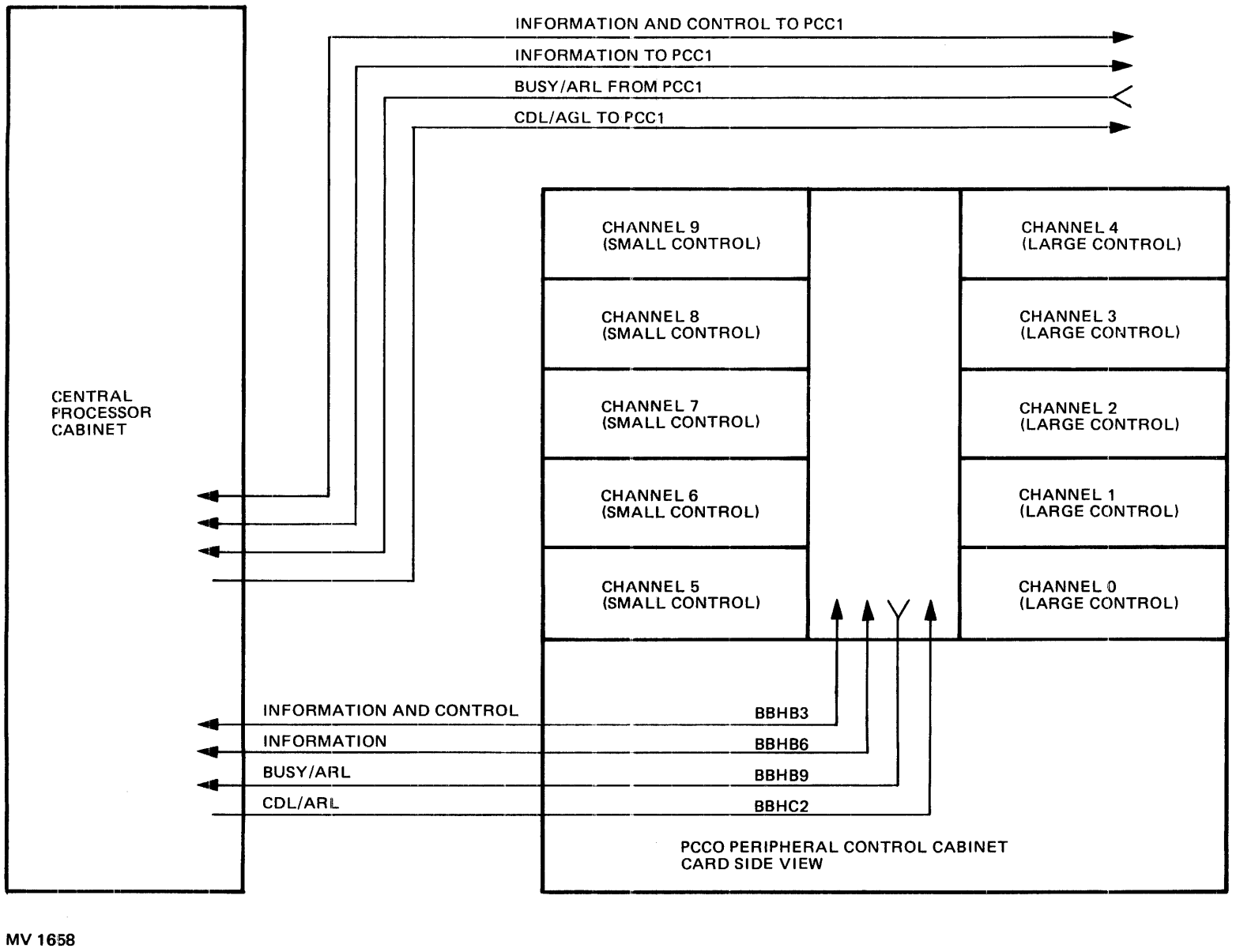


Figure 11-3. B 6800 Peripheral Controls Organization

At the conclusion of an IO operation, the IO control assembles an IO result descriptor that describes the IO operation, and any errors that were encountered. This result descriptor is returned to the multiplexor, along with the proper control signals to let the multiplexor know that the IO operation is complete.

The multiplexor uses the result descriptor to form a part of the P3 parameter. Bit zero of the P2 parameter will be set when any error condition occurs during the IO operation.

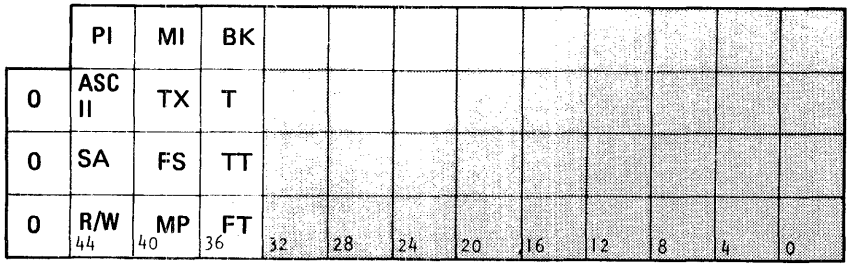
Signal Name	Meaning and Usage
CDLnn	The CDL logic signal is sent from the multiplexor to the IO control and is used by a peripheral control device to step to the next sequence count in the flow of sequences within the unit control, during the initialization sequence. For example; the multiplexor sends an OP code to a peripheral control at the beginning of each IO operation that is initiated. The OP code is strobed onto the peripheral control bus, and the multiplexor raises the CDL control signal level for the peripheral control channel. The peripheral control will accept the OP code that is present on the peripheral control bus, and will step to the next sequence count in the operational control flow.
BUnn/	The busy not logic is used by the multiplexor to determine whether or not a path is available to the IO control. If busy not is true, a path exists to the IO control, and if busy not is false the IO control is in use. The busy not signal is used in conjunction with the pseudo busy flip-flop, which is entirely local to the multiplexor logic, and has no direct connection to the peripheral bus signals. The pseudo busy flip-flop may be set by either of two different methods. The flip-flop can be set by means of a SCAN-OUT operator, as described previously in this section, to deny the use of the path except by overriding the pseudo busy flip-flop. Pseudo busy is also set by the multiplexor, when an IO error is detected in the peripheral control device. This latter method of setting the pseudo busy flip-flop is used to preserve extended result descriptor status about the error that occured. If a subsequent IO operation is attempted before the extended status is read from the IO control, the data in the extended status is lost, and cannot be recovered by the system. If an IO control is in the process of an IO command, the BUnn/ logic signal will be false, regardless of the state of the pseudo busy flip-flop. The pseudo busy flip-flop may be reset by means of a SCAN-OUT set pseudo busy operator. This operator was described previously, in this section of this manual.

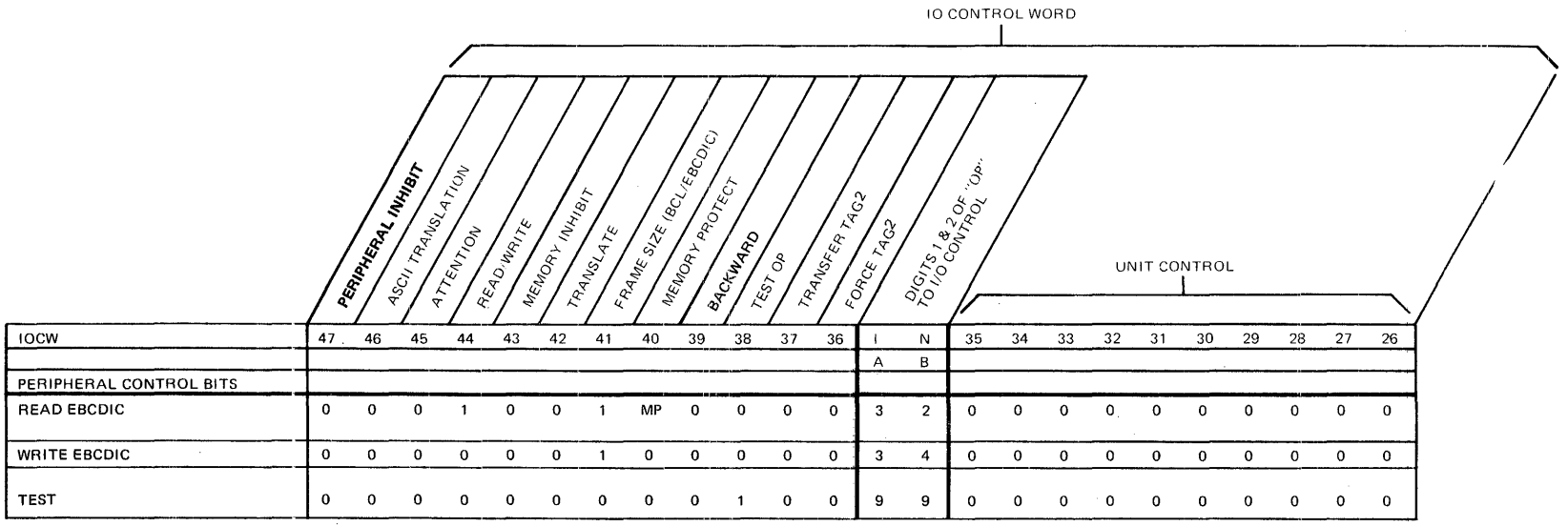
Signal Name	Meaning and Usage
ARLnn	The access request control logic level is used by the peripheral control device to notify the multiplexor that an access to the data buffer is required. During output operations, this level means that the IO control has processed the last character(s) received from the data buffer, and consequently requires the next character(s) to be passed through the peripheral bus. During input operations, this level means that the IO control has placed the next input character(s) in position to be transmitted to the data buffer, and consequently requires access to the data bus so that they can be passed to the multiplexor. At the conclusion of an IO operation, both input and output types, the IO control forms a unit result descriptor which is returned to the multiplexor. The IO control uses the ARL logic level to notify the multiplexor when the result descriptor is completed and ready to be transmitted to the data buffer.
AGLnn	The AGLnn control signal is used by the multiplexor to respond to an ARLnn control signal from a peripheral control unit. The multiplexor makes a path from the peripheral data bus to the data buffer available, and then raises the AGL level to notify the IO control that data may be passed to or from the buffer through the peripheral data bus. The multiplexor may be processing several simultaneous IO operations. If an IO control raises its ARL control level to obtain an access to the data buffer, the multiplexor must determine which of several possible requests will be granted first access to a data buffer. The multiplexor will determine this priority between several IO channels on the basis of priority configuration within the multiplexor. When a new IO control is installed in a B 6800 system, or an old control is removed from a B 6800 system, maintenance personnel may alter the priority configuration wires to account for the change in system IO resources.

INPUT OUTPUT DEVICE COMMANDS AND RESULT DESCRIPTORS

The following data will present the types of operations that an IO device can perform, and the "OP" codes that cause one of these operations to be performed. This data will also show the information that is contained in the P3 interrupt parameter. Although the P2 parameter is not presented, it is implied that the error bit in the P2 parameter will be set if any error bit is present in the P3 parameter.

SUPERVISORY DISPLAY CONTROL II





IOCW	47	46	45	44	43	42	41	40	39	38	37	36	I A	N B	35	34	33	32	31	30	29	28	27	26
PERIPHERAL CONTROL BITS																								
READ EBCDIC	0	0	0	1	0	0	1	MP	0	0	0	0	3	2	0	0	0	0	0	0	0	0	0	0
WRITE EBCDIC	0	0	0	0	0	0	1	0	0	0	0	0	3	4	0	0	0	0	0	0	0	0	0	0
TEST	0	0	0	0	0	0	0	0	0	1	0	0	9	9	0	0	0	0	0	0	0	0	0	0

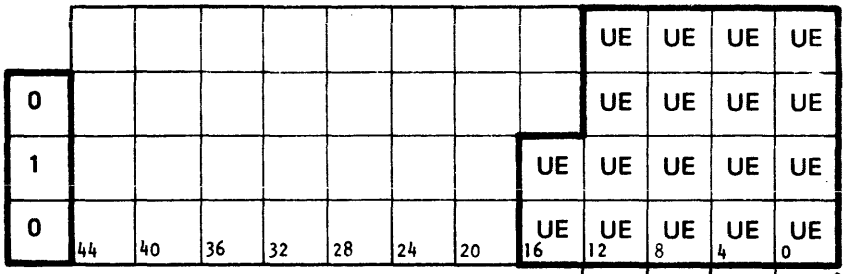
NOTES:

BIT CONFIGURATION:

BITS 37	36	
0	0	SET SINGLE TAG
0	1	SET CODE TAG
1	0	SET TRANSFER TAG
1	1	SET DOUBLE TAG

Figure 11-4. Supervisory Display Control II IOCW Format

Figure 11-4. Supervisory Display Control II IOCW Format

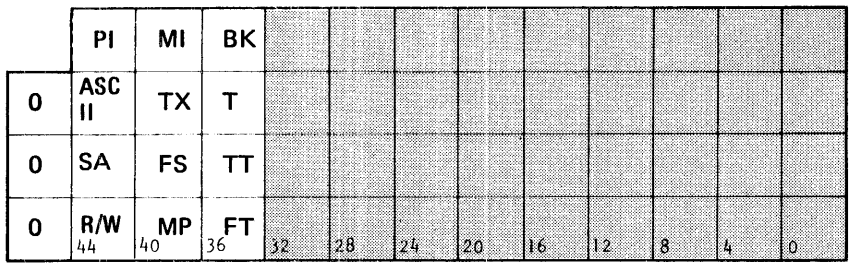


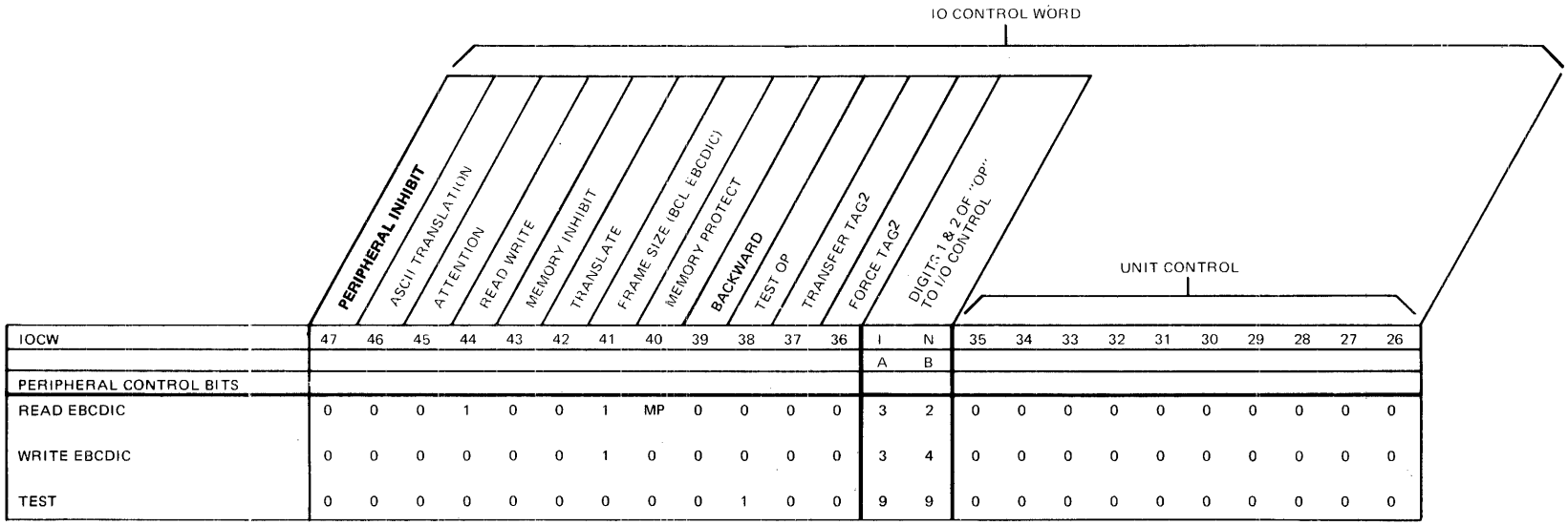
ERROR CODE MEANING		HEXIDECIMAL UNIT ERROR CODES			
UNIT NOT READY	0	0	0	0	8
IO PARITY ERROR	0	0	A	8	0
MEMORY ACCESS ERROR	0	0	2	8	0
DATA PARITY ERROR	0	0	2	0	0
CONTROL CHARACTER	0	0	4	0	0
READ OVERFLOW	0	0	8	0	0
TIMEOUT (INVALID CHARACTER)	0	8	0	0	0
INTERNAL PARITY ERROR	0	0	3	0	0
UNIT ID (TD804/TD830)	0	2	0	0	0
UNIT ID (B9352)	0	4	0	0	0
UNIT ID (B9348-34)	0	6	0	0	0

MV 1660

Figure 11-5. Supervisory Display Control II Result Descriptor Format

SINGLE LINE CONTROL





IOCW	47	46	45	44	43	42	41	40	39	38	37	36	I A	N B	35	34	33	32	31	30	29	28	27	26
PERIPHERAL CONTROL BITS																								
READ EBCDIC	0	0	0	1	0	0	1	MP	0	0	0	0	3	2	0	0	0	0	0	0	0	0	0	0
WRITE EBCDIC	0	0	0	0	0	0	1	0	0	0	0	0	3	4	0	0	0	0	0	0	0	0	0	0
TEST	0	0	0	0	0	0	0	0	0	1	0	0	9	9	0	0	0	0	0	0	0	0	0	0

NOTES:

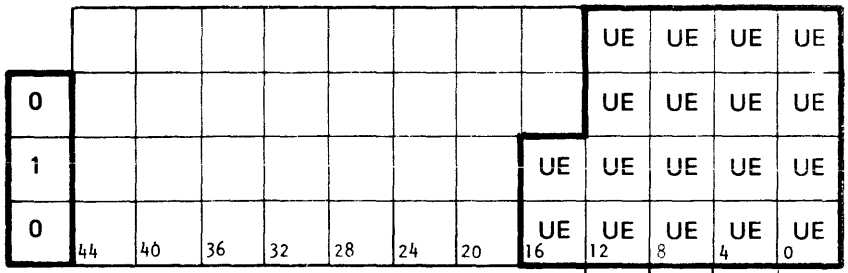
BIT CONFIGURATION:

BITS 37	36	
0	0	SET SINGLE TAG
0	1	SET CODE TAG
1	0	SET TRANSFER TAG
1	1	SET DOUBLE TAG

MV 1661

Figure 11-6. Single Line Control IOCW Format

Figure 11-6. Single Line Control IOCW Format



ERROR CODE MEANING	HEXIDECIMAL UNIT ERROR CODES				
MEMORY ACCESS ERROR	0	0	0	8	0
PARITY ERROR	0	0	2	8	0
CONTROL MESSAGE	0	0	4	0	0
OVERFLOW	0	0	8	0	0
TIMEOUT	0	8	0	0	0
UNIT ID (BIDS)	0	4	0	0	0
UNIT ID (TC500)	0	2	0	0	0

MV 1662

Figure 11-7. Single Line Control Result Descriptor Format

CARD PUNCH

	PI	MI	BK	0										
0	ASC II	TX	T	0										
0	SA	FS	TT	0										
0	R/W 44	MP 40	FT 36	S 32	28	24	20	16	12	8	4	0		

STANDARD CONTROL FIELD

UNIT CONTROL FIELD

IO CONTROL WORD

PERIPHERAL INHIBIT
ASCII TRANSLATION
ATTENTION
READ/WRITE
MEMORY INHIBIT
TRANSLATE
FRAME SIZE (BCL/EBCDIC)
MEMORY PROTECT
BACKWARD
TEST OP
TRANSFER TAG3
FORCE TAG3
DIGITS 1 & 2 OF "OP" TO I/O CONTROL

UNIT CONTROL

IOCW	47	46	45	44	43	42	41	40	39	38	37	36	I	N	35	34	33	32	31	30	29	28	27	26
													A	B										
PERIPHERAL CONTROL BITS																								
PUNCH BINARY (6-BIT FROM 6-BIT)	0	0	0	0	0	0	0	0	0	0	0	0	2	4	0	0	0	S	0	0	0	0	0	0
PUNCH EBCDIC (FROM EBCDIC)	0	0	0	0	0	0	1	0	0	0	0	0	2	5	0	0	0	S	0	0	0	0	0	0
PUNCH BCL (FROM INT BCL)	0	0	0	0	0	1	0	0	0	0	0	0	2	3	0	0	0	S	0	0	0	0	0	0
PUNCH BCL (FROM EBCDIC)	0	0	0	0	0	1	1	0	0	0	0	0	2	3	0	0	0	S	0	0	0	0	0	0
TEST	0	0	0	0	0	0	0	0	0	1	0	0	9	9	0	0	0	0	0	0	0	0	0	0

NOTES:

BIT CONFIGURATION:

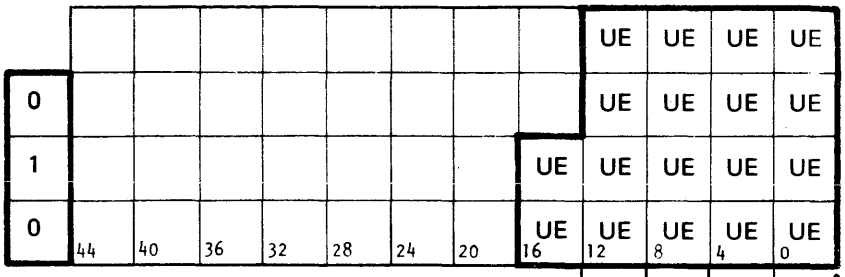
BITS 37	36	
0	0	SET SINGLE TAG
0	1	SET CODE TAG
1	0	SET TRANSFER TAG
1	1	SET DOUBLE TAG

S: STACKER
0 = PRIMARY
1 = AUXILIARY

STACKER = [32:1]

MV 1663

Figure 11-8. Card Punch IOCW Format

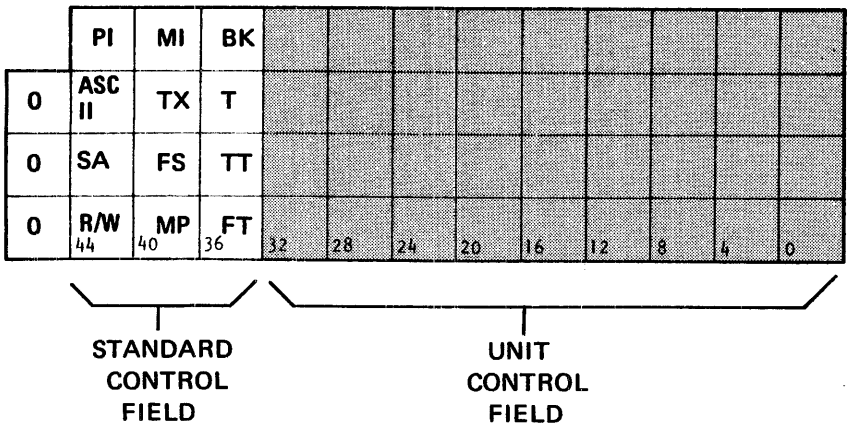


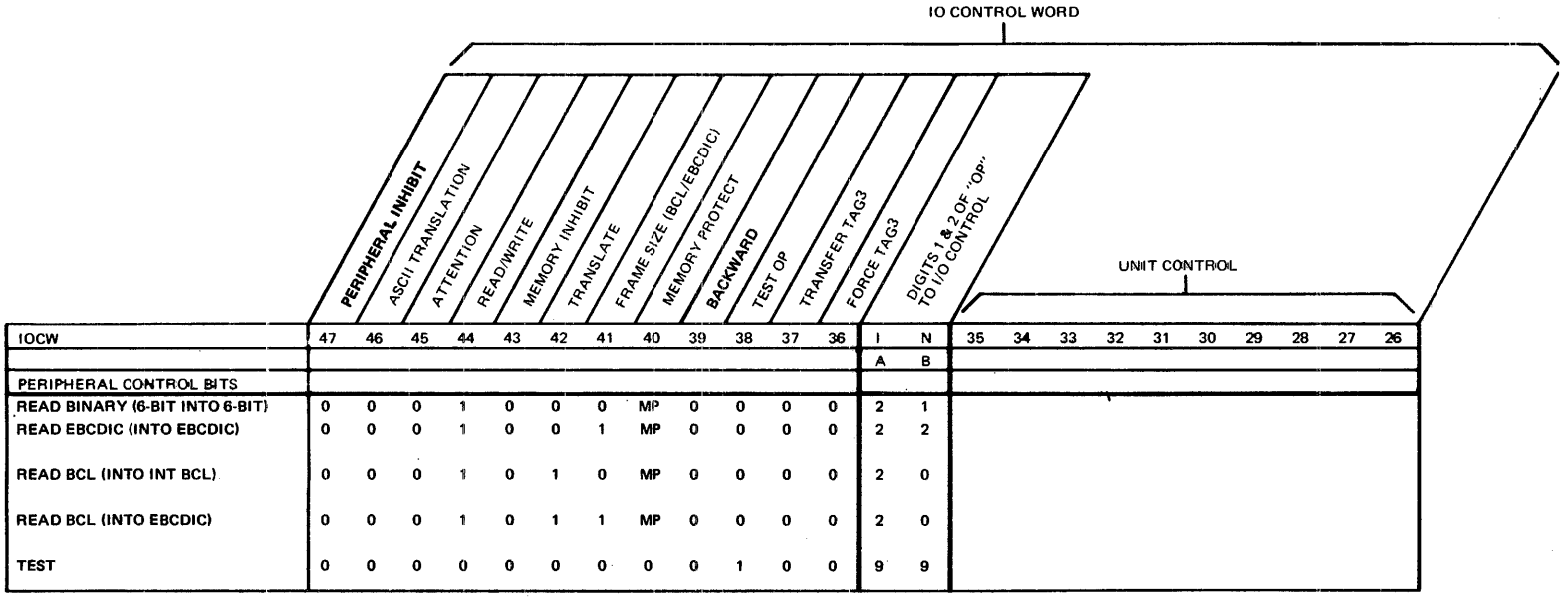
ERROR CODE MEANING	HEXIDECIMAL UNIT ERROR CODES				
NOT READY	0	0	0	0	8
PUNCH CHECK	0	0	0	8	0
MEMORY PARITY ERROR	0	0	2	8	0
MEMORY ACCESS ERROR	0	0	4	8	0

MV 1664

Figure 11-9. Card Punch Result Descriptor Format

CARD READER





IOCW	47	46	45	44	43	42	41	40	39	38	37	36	I	N	35	34	33	32	31	30	29	28	27	26
													A	B										
PERIPHERAL CONTROL BITS																								
READ BINARY (6-BIT INTO 6-BIT)	0	0	0	1	0	0	0	MP	0	0	0	0	2	1										
READ EBCDIC (INTO EBCDIC)	0	0	0	1	0	0	1	MP	0	0	0	0	2	2										
READ BCL (INTO INT BCL)	0	0	0	1	0	1	0	MP	0	0	0	0	2	0										
READ BCL (INTO EBCDIC)	0	0	0	1	0	1	1	MP	0	0	0	0	2	0										
TEST	0	0	0	0	0	0	0	0	0	1	0	0	9	9										

NOTES:

BIT CONFIGURATION:

BITS 37	36	
0	0	SET SINGLE TAG
0	1	SET CODE TAG
1	0	SET TRANSFER TAG
1	1	SET DOUBLE TAG

Figure 11-10. Card Reader IOCW Format

									UE	UE	UE	UE
0									UE	UE	UE	UE
1								UE	UE	UE	UE	UE
0								UE	UE	UE	UE	UE
	44	40	36	32	28	24	20	16	12	8	4	0

ERROR CODE MEANING	HEXIDECIMAL UNIT ERROR CODES				
NOT READY	0	0	0	0	8
MEMORY ACCESS ERROR	0	0	0	8	0
VALIDITY CHECK	0	0	2	8	0
READ CHECK	0	0	1	0	0
READ & VALIDITY CHECK	0	0	3	8	0
CONTROL CARD	0	0	4	0	0

MV 1666

Figure 11-11. Card Reader Result Descriptor Format

LINE PRINTER

	PI	MI	BK	SK	SP								
0	ASC II	TX	T	SK	SP								
0	SA	FS	TT	SK									
0	R/W	MP	FT	SK									
	44	40	36	32	28	24	20	16	12	8	4	0	

STANDARD CONTROL FIELD

UNIT CONTROL FIELD

IO CONTROL WORD

UNIT CONTROL

IOCW	47	46	45	44	43	42	41	40	39	38	37	36	I A	N B	35	34	33	32	31	30	29	28	27	26
PERIPHERAL CONTROL BITS	PERIPHERAL INHIBIT	ASCII TRANSLATION	ATTENTION	READ/WRITE	MEMORY INHIBIT	TRANSLATE	FRAME SIZE (BCL/EBCDIC)	MEMORY PROTECT	BACKWARD	TEST OP	TRANSFER TAG 1	FORCE TAG 3	DIGITS 1 & 2 OF "OP" TO I/O CONTROL											
PRINT BCL (FROM INT BCL)	0	0	0	0	0	1	0	0	0	0	0	0	1	0	SK	SK	SK	SK	SP	SP	0	0	0	0
PRINT BCL (FROM EBCDIC)	0	0	0	0	0	1	1	0	0	0	0	0	1	0	SK	SK	SK	SK	SP	SP	0	0	0	0
SKIP	0	0	0	0	1	0	0	0	0	0	0	0	1	1	SK	SK	SK	SK	0	0	0	0	0	0
SPACE	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	SP	SP	0	0	0	0
TEST	0	0	0	0	0	0	0	0	0	1	0	0	9	9	0	0	0	0	0	0	0	0	0	0

NOTES:

BIT CONFIGURATION:

BITS 37	BITS 36	
0	0	SET SINGLE TAG
0	1	SET CODE TAG
1	0	SET TRANSFER TAG
1	1	SET DOUBLE TAG

CHANNEL = [35:4]
SPACING = [35:6]
SINGLE SPACE = [35:6] = 1
DOUBLE SPACE = [35:6] = 2

SK = SKIP
SP = SPACE

Figure 11-12. Line Printer IOCW Format

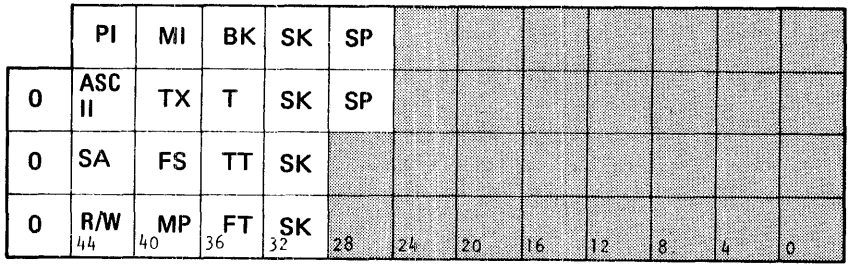
									UE	UE	UE	UE
0									UE	UE	UE	UE
1								UE	UE	UE	UE	UE
0								UE	UE	UE	UE	UE
	44	40	36	32	28	24	20	16	12	8	4	0

ERROR CODE MEANING		HEXIDECIMAL UNIT ERROR CODES			
BIT TRANSFER ERROR	0	0	1	8	0
BUFFER PARITY ERROR	0	0	2	8	0
PRINT CHECK	0	0	4	8	0
LOW PAPER	0	0	8	0	0
END OF PAGE	0	1	0	0	0

MV 1668

Figure 11-13. Line Printer Result Descriptor Format

TRAIN PRINTER



STANDARD CONTROL FIELD

UNIT CONTROL FIELD

IO CONTROL WORD

UNIT CONTROL

PERIPHERAL INHIBIT
ASCII TRANSLATION
ATTENTION
READ/WRITE
MEMORY INHIBIT
TRANSLATE
FRAME SIZE (BCL/EBCDIC)
MEMORY PROTECT
BACKWARD
TEST OP
TRANSFER TAG 1
FORCE TAG 1
DIGITS 1 & 2 OF "OP" TO I/O CONTROL

IOCW	47	46	45	44	43	42	41	40	39	38	37	36	I A	N B	35	34	33	32	31	30	29	28	27	26
PERIPHERAL CONTROL BITS																								
PRINT BINARY	0	0	0	0	0	0	0	0	0	0	0	0	1	0	SK	SK	SK	SK	SP	SP	0	0	0	0
PRINT BCL← 6-BIT BINARY	0	0	0	0	0	1	0	0	0	0	0	0	1	0	SK	SK	SK	SK	SP	SP	0	0	0	0
PRINT BCL← EBCDIC	0	0	0	0	0	1	1	0	0	0	0	0	1	0	SK	SK	SK	SK	SP	SP	0	0	0	0
PRINT EBCDIC	0	0	0	0	0	0	1	0	0	0	0	0	1	0	SK	SK	SK	SK	SP	SP	0	0	0	0
SKIP	0	0	0	0	1	0	0	0	0	0	0	0	1	1	SK	SK	SK	SK	0	0	0	0	0	0
SPACE	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	SP	SP	0	0	0	0
LOAD TRANSLATE TABLE2	0	0	0	0	0	0	0	0	1	0	0	0	2	9	0	0	0	0	0	0	0	0	0	0
TEST	0	0	0	0	0	0	0	0	0	1	0	0	9	9	0	0	0	0	0	0	0	0	0	0

NOTES:

BIT CONFIGURATION:

BITS 37	36	
0	0	SET SINGLE TAG
0	1	SET CODE TAG
1	0	SET TRANSFER TAG
1	1	SET DOUBLE TAG

LTT BIT CONFIGURATION

BITS 42	41	
0	0	LOAD LTT WITH 6-BIT BINARY CHARACTERS
0	1	LOAD LTT WITH 8-BIT EBCDIC CHARACTERS
1	0	LOAD LTT WITH 6-BIT BCL CHARACTERS

SK = SKIP
SP = SPACE

MV 1669

Figure 11-14. Train Printer IOCW Format

									UE	UE	UE	UE
0									UE	UE	UE	UE
1								UE	UE	UE	UE	UE
0	44	40	36	32	28	24	20	UE 16	UE 12	UE 8	UE 4	UE 0

ERROR CODE MEANING	HEXIDECIMAL UNIT ERROR CODES				
PRINT CHECK (SYNC)	0	0	1	8	0
PRINT CHECK (INTERNAL)	0	0	4	8	0
IO BUS PARITY	0	0	8	8	0
IO BUS PARITY (INITIATE)	0	0	8	8	0
END OF PAPER	0	1	0	0	0
TRAIN IMAGE BUFFER NOT LOADED	0	4	0	0	0
INCORRECT TRAIN LOADED	0	8	0	0	0

MV 1670

Figure 11-15. Train Printer Result Descriptor Format

MAGNETIC TAPE

	PI	MI	BK	TM	D	T	B	B				
0	ASC II	TX	T	REP	P	T	B	B				
0	SA	FS	TT	D	C		B	B				
0	R/W	MP	FT	D	T		B	B				
	44	40	36	32	28	24	20	16	12	8	4	0

STANDARD CONTROL FIELD

UNIT CONTROL FIELD

IO CONTROL WORD

UNIT CONTROL

	IOCW	47	46	45	44	43	42	41	40	39	38	37	36	I A	N B	35	34	33	32	31	30	29	28	27	26
	PERIPHERAL CONTROL BITS																								
7-TRACK	READ BINARY (6-BIT INTO 6-BIT)	0	0	0	1	0	0	0	MP	0	0	0	0	0	2	0	0	D	D	D	P	0	0	0	0
	READ BCL (INTO INT BCL)	0	0	0	1	0	1	0	MP	0	0	0	0	0	2	0	0	D	D	D	P	0	0	0	0
	READ BCL (INTO EBCDIC)	0	0	0	1	0	1	1	MP	0	0	0	0	0	2	0	0	D	D	D	0	0	0	0	0
	READ EBCDIC (INTO EBCDIC)	0	0	0	1	0	1	1	MP	0	0	0	0	0	2	0	0	D	D	D	P	0	0	0	0
	WRITE BINARY (6-BIT FROM 6-BIT)	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	D	D	D	P	0	0	0	0
	WRITE BCL (FROM INT BCL)	0	0	0	0	0	1	0	0	0	0	0	0	0	6	0	0	D	D	D	P	0	0	0	0
	WRITE BCL (FROM EBCDIC)	0	0	0	0	0	1	1	0	0	0	0	0	0	6	0	0	D	D	D	0	0	0	0	0
	WRITE EBCDIC (FROM EBCDIC)	0	0	0	0	0	1	1	0	0	0	0	0	0	6	0	0	D	D	D	P	0	0	0	0
	ERASE	0	0	0	0	1	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0
9-TRACK	READ (8-BIT INTO 8-BIT)	0	0	0	1	0	0	1	MP	0	0	0	0	0	2	0	0	D	D	D	1	C	T	T	T
	WRITE (8-BIT FROM 8-BIT)	0	0	0	0	0	0	1	0	0	0	0	0	0	6	0	0	D	D	D	1	0	0	0	0
	ERASE	0	0	0	0	1	0	1	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0	0
EITHER	REWIND	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
	SPACE	0	0	0	1	1	0	0	0	0	0	0	0	0	8	0	0	D	D	D	P	0	0	0	0
	WRITE TAPEMARK	0	0	0	0	0	0	0	0	0	0	0	0	0	6	1	0	0	0	0	0	0	0	0	0
	TEST	0	0	0	0	0	0	0	0	0	1	0	0	9	9	0	0	0	0	0	0	0	0	0	0
	REPOSITION	0	0	0	0	1	0	0	0	0	0	0	0	0	4	0	1	D	D	D	0	0	0	0	0

PERIPHERAL INHIBIT
ASCII TRANSLATION
ATTENTION
READ/WRITE
MEMORY INHIBIT
TRANSLATE
FRAME SIZE
MEMORY PROTECT
BACKWARD
TEST
TRANSFER TAG
FORCE TAG
DIGITS 1 & 2 OF "OP" TO I/O CONTROL

NOTES:

33	32	31	30		
0	0	0	0	=	DENSITY, PARITY (PERIPHERAL CONTROLS)
0	0	0	0	=	UNIT SELECTED, EVEN
0	0	0	1	=	UNIT SELECTED, ODD
1	0	0	0	=	800 BPI, EVEN (7-TRACK ONLY; II)
1	0	0	1	=	800 BPI, ODD (7- OR 9-TRACK; II OR IV)
1	0	1	0	=	556 BPI, EVEN
1	0	1	1	=	556 BPI, ODD (7-TRACK ONLY; II)
1	1	0	0	=	200 BPI, EVEN
1	1	0	1	=	200 BPI, ODD (7- OR 9-TRACK; II OR IV)
1	1	1	0	=	NONE
1	1	1	1	=	1600 BPI, ODD (9-TRACK P.E. ONLY; V OR VI)

IF c = 1 THEN 28 ➔ 26 = CRC: TRACK IN ERROR TO BE CORRECTED, OR TRACK IN ERROR FROM RESULT DESCRIPTOR

23 ➔ 16 = BLOCK TO SPACE (IN BCD)

MV 1671

Figure 11-16. Magnetic Tape IOCW Format

									UE	UE	UE	UE
0									UE	UE	UE	UE
1								UE	UE	UE	UE	UE
0								UE	UE	UE	UE	UE
	44	40	36	32	28	24	20	16	12	8	4	0

ERROR CODE MEANING		HEXIDECIMAL UNIT ERROR CODES			
MEMORY PARITY ERROR	0	0	2	8	0
MEMORY ACCESS ERROR	0	0	0	8	0
TAPE PARITY ERROR	0	0	0	8	0
WRITE LOCKOUT (OR TAPE MARK)	0	0	2	0	0
END OF TAPE/BEGINNING OF TAPE	0	0	1	0	0
SHORT RECORD (1600 BPI TOP)	0	0	4	0	0
LONG RECORD (1600 BPI TOP)	0	0	8	0	0
BLANK TAPE	0	8	0	0	0
MEC NOT READY AFTER TAPE MOTION	0	0	1	0	8
SYSTEM INTERFACE PARITY ERROR	0	0	8	8	0
SYSTEM INTERFACE PARITY ERROR BEFORE TAPE MOTION	0	0	8	8	0
PERIPHERAL INTERFACE PARITY ERROR	0	0	4	8	0
PERIPHERAL INTERFACE PARITY ERROR BEFORE TAPE MOTION	0	0	4	8	0
NONPRESENT OPTION	0	2	0	0	0

MV 1672

Figure 11-17. Magnetic Tape Result Descriptor Format

HEAD-PER-TRACK DISK FILE

	PI	MI	BK			0	DA	DA	DA	DA	DA	DA
0	ASC II	TX	T			0	DA	DA	DA	DA	DA	DA
0	SA	FS	TT		0	DA	DA	DA	DA	DA	DA	DA
0	R/W 44	MP 40	FT 36	32	0 28	DA 24	DA 20	DA 16	DA 12	DA 8	DA 4	DA 0

STANDARD CONTROL FIELD

UNIT CONTROL FIELD

IO CONTROL WORD

PERIPHERAL INHIBIT
ASCII TRANSLATION
ATTENTION
READ/WRITE
MEMORY INHIBIT
TRANSLATE
FRAME SIZE (BCL/EBCDIC)
MEMORY PROTECT
MAINTENANCE SEGMENT
TEST OP
TRANSFER TAG3
FORCE TAG3
DIGITS 1 & 2 OF "OP" TO I/O CONTROL

UNIT CONTROL

IOCW	47	46	45	44	43	42	41	40	39	38	37	36	I A	N B	35	34	33	32	31	30	29	28	27	26
PERIPHERAL CONTROL BITS																								
READ	0	0	0	1	0	0	1	MP	M	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0
READ CHECK	0	0	0	1	1	0	1	0	M	0	0	0	5	1	0	0	0	0	0	0	0	0	0	0
WRITE	0	0	0	0	0	0	1	0	M	0	0	0	5	2	0	0	0	0	0	0	0	0	0	0
TEST	0	0	0	0	0	0	0	0	0	1	0	0	9	9	0	0	0	0	0	0	0	0	0	0

NOTES:

BIT CONFIGURATION:

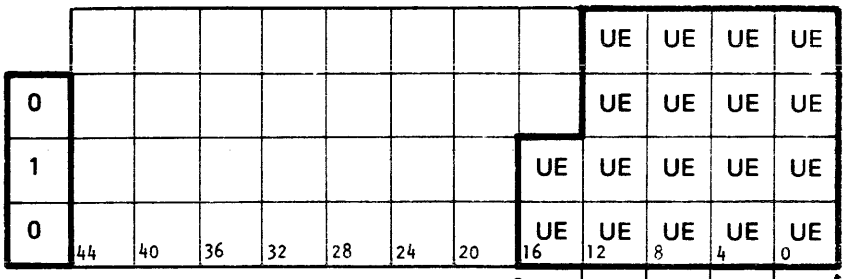
BITS 37	36	
0	0	SET SINGLE TAG
0	1	SET CODE TAG
1	0	SET TRANSFER TAG
1	1	SET DOUBLE TAG

M = MAINTENANCE SEGMENT

BITS 25:26 IS DISK ADDRESS, AND REPRESENTS A SIX DIGIT BINARY CODED DECIMAL DISK FILE SEGMENT ADDRESS

MV1673

Figure 11-18. Head Per Track Disk File IOCW Format

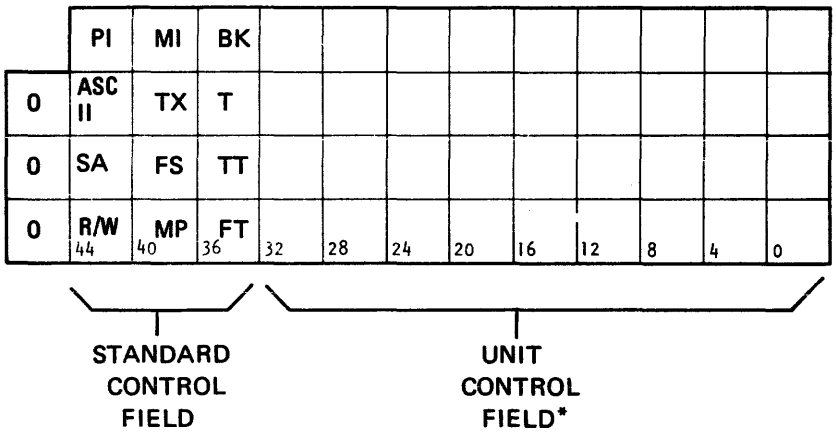


ERROR CODE MEANING	HEXIDECIMAL UNIT ERROR CODES				
NOT READY	0	0	0	0	8
MEMORY ACCESS ERROR	0	0	0	8	0
MEM PARITY ERROR (READ ERROR)	0	0	2	8	0
BUSY	0	0	1	0	0
WRITE LOCKOUT	0	0	2	0	0
NOT READY DURING OPERATION	0	0	8	0	8
TIMEOUT	0	8	0	0	0

MV 1674

Figure 11-19. Head Per Track Disk File Result Descriptor Format

FLEXIBLE DISK



NOTE
*DATA TO BE SUPPLIED WHEN AVAILABLE

MV 1675

Figure 11-20. Flexible Disk IOCW Format

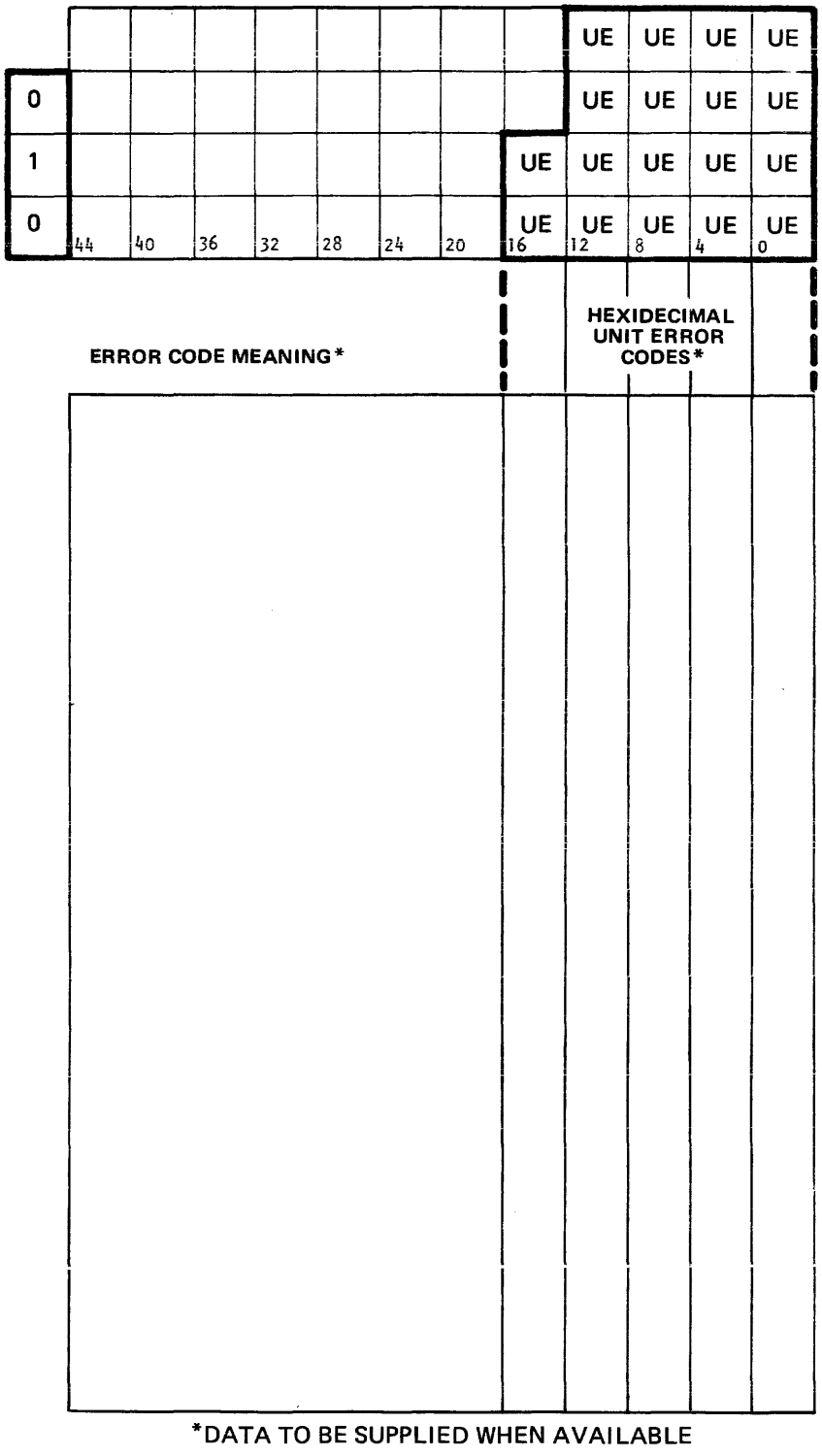


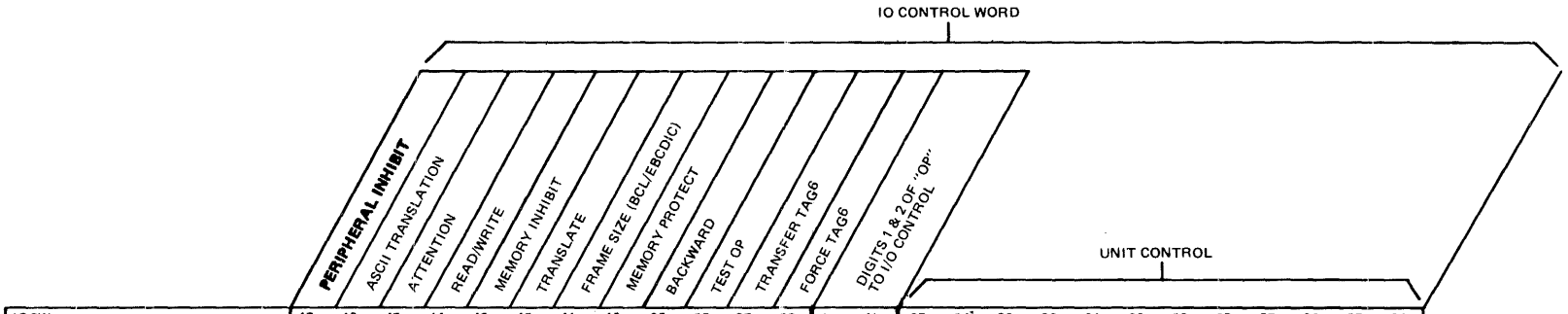
Figure 11-21. Flexible Disk Result Descriptor Format

DISK PACK

	PI	MI	BK	1	V2	S8	PA	PA	PA	PA	PA	PA
0	ASC II	TX	T	F1	V1	S4	PA	PA	PA	PA	PA	PA
0	SA	FS	TT	V8	0	S2	PA	PA	PA	PA	PA	PA
0	R/W	MP	FT	V4	0	S1	PA	PA	PA	PA	PA	PA
	44	40	36	32	28	24	20	16	12	8	4	0

STANDARD CONTROL FIELD

UNIT CONTROL FIELD



IOCW	47	46	45	44	43	42	41	40	39	38	37	36	I	N	35	34	33	32	31	30	29	28	27	26	25	24
													A	B												
PERIPHERAL CONTROL BITS																										
WRITE			0	0	0	0	1		0	0	0		5	0		F1	V8	0	0	0	0	0	S8	S4	S2	S1
READ			0	1	0	0	1	0	0	0	0	0	5	1		F1	V8	V4	V2	V1	0	0	S8	S4	S2	S1
INITIALIZE			0	0	0	0	1		0	0			5	6	1	F1	0	V4	V2	V1	0	0	0	0	S2	S1
VERIFY			0	1	0	0	1	0	0	0			5	7	1	F1	V8	V4	V2	V1	0	0	0	S4	0	S1
RELOCATE			0	0	1	0	1		0	0			5	8	0	F1	0	0	0	0	0	0	0	0	S2	S1
TEST			0							1			9	9	0					V1						

MV1677

Figure 11-22. Disk Pack IOCW Format

OPERATION	F1 34	V8 33	V4 32	V2 31	V1 30	29	28	S8 27	S4 26	S2 25	S1 24	DESCRIPTION
WRITE 41=1	0	0	0	0	0	0	0	0	1	0	0	READ AFTER WRITE OPERATION
37=0	0	0	0	0	0	0	0	0	0	0	0	NORMAL WRITE OPERATION
READ 44=1 41=1	0	0	0	0	0	0	0	0	1	0	0	READ 1 BINARY ADDRESS AS INDICATED BY THE FILE ADDRESS
	0	0	0	0	0	0	0	0	0	0	0	NORMAL READ OPERATION
INITIALIZE 41=1 35=1	0	0	0	0	1	0	0	0	0	0	0	INITIALIZE DESIGNATED CYLINDER
	0	0	0	0	0	0	0	0	0	0	0	INITIALIZE ENTIRE PACK
	0	0	0	1	0	0	0	0	0	0	0	INITIALIZE DESIGNATED TRACK
	0	0	1	0	0	0	0	0	0	0	0	WRITE TEST DATA PATTERN INTO EACH SECTOR
VERIFY 44=1 41=1 35=1	0	0	0	0	1	0	0	0	0	0	0	VERIFY DESIGNATED CYLINDER AND REPORT ALL ERRORS
	0	0	0	0	0	0	0	0	0	0	0	VERIFY ENTIRE PACK AND TERMINATE ON FIRST ERROR
	0	0	0	1	0	0	0	0	0	0	0	VERIFY DESIGNATED TRACK AND REPORT ALL ERRORS
	0	0	0	0	0	0	0	0	0	0	0	CHECK PARITY ON DATA, ERROR DETECTION, AND COUNT CHECK FIELDS
	0	1	0	0	0	0	0	0	0	0	0	VERIFY DATA PATTERN WITH PATTERN NORMALLY WRITTEN DURING INITIALIZATION
RELOCATE 43=1 41=1	0	0	0	0	0	0	0	0	0	0	0	BINARY COUNT INDICATING SPARE SECTOR ADDRESS 28 THROUGH 32
TEST 38=1	0	0	0	0	0	0	0	0	0	0	1	PROGRAMMATIC POWER-DOWN

BIT CONFIGURATION:

BITS 37	36	
0	0	SET SINGLE TAG
0	1	SET CODE TAG
1	0	SET TRANSFER TAG
1	1	SET DOUBLE TAG

NOTES:

23:24 IS PACK ADDRESS, AND REPRESENTS A SIX-DIGIT BINARY CODED DECIMAL DISK PACK SEGMENT ADDRESS

MV1678

Figure 11-23. Disk Pack IOCW Unit Control Format

									UE	UE	UE	UE
0									UE	UE	UE	UE
1								UE	UE	UE	UE	UE
0	44	40	36	32	28	24	20	UE 16	UE 12	UE 8	UE 4	UE 0

ERROR CODE MEANING		HEXIDECIMAL UNIT ERROR CODES			
MEMORY ACCESS ERROR	0	0	0	8	0
ADDRESS PARITY ERROR	0	0	1	0	0
CONTROLLER IN LOCAL	0	0	1	0	0
DRIVE SEEKING	0	0	2	0	0
FIRST ACTION WITH DRIVE	0	0	3	0	0
DRIVE BUSY	0	0	4	0	0
SPEED ERROR	0	0	5	8	0
WRITE LOCKOUT	0	0	5	0	0
DATA ERROR	0	0	5	8	0
SEEK ERROR	0	0	8	0	0
CONTROLLER MALFUNCTION	0	0	8	0	0
DATA PARITY IO	0	0	8	8	0
COMMAND PARITY	0	0	9	8	0
SECTOR TIMEOUT	0	0	9	0	0
LINK PARITY (DPDC-TO-HOST)	0	0	9	8	0
SEEK INITIATED	0	0	A	0	0
DATA ERROR CORRECTION	0	0	C	0	0
ADDRESS POSITION ERROR	0	4	1	0	0
SEEK TIMEOUT	0	8	8	0	0
DATA PARITY (HTC-TO-DPDC)	0	8	8	8	0
COMMAND PARITY (HTC-TO-DPDC)	0	8	8	8	0
DATA ERROR RETRY	0	C	8	0	0

MV 1679

Figure 11-24. Disk Pack Result Descriptor Format

5N DISK FILE

	PI	MI	BK		V3-2		SA	SA	SA	SA	SA	SA
0	ASC II	TX	T		V3-1		SA	SA	SA	SA	SA	SA
0	SA	FS	TT	V3-8		V2-2	SA	SA	SA	SA	SA	SA
0	R/W 44	MP 40	FT 36	V3-4 32	28	V2-1 24	SA 20	SA 16	SA 12	SA 8	SA 4	SA 0

STANDARD CONTROL FIELD

UNIT CONTROL FIELD

IO CONTROL WORD

PERIPHERAL INHIBIT
ASCII TRANSLATION
ATTENTION
READ/WRITE
MEMORY INHIBIT
TRANSLATE
FRAME SIZE (BCL/EBCDIC)
MEMORY PROTECT
BACKWARD
TEST OP
TRANSFER TAG 1
FORCE TAG 1
DIGITS 1 & 2 OF "OP" TO I/O CONTROL

UNIT CONTROL

IOCW	47	46	45	44	43	42	41	40	39	38	37	36	I A	N B	35	34	33 V3	32 V3	31 V3	30 V3	29	28	27	26	25 V2	24 V2
PERIPHERAL CONTROL BITS																										
WRITE	0	0	0	0	0	0	1	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0
READ	0	0	0	1	0	0	1	0	0	0	0	0	5	1	0	0	0	0	0	0	0	0	0	0	0	0
CHECK	0	0	0	0	0	0	1	0	0	0	0	0	5	2	0	0	0	0	0	0	0	0	0	0	0	0
TEST	0	0	0	0	0	0	0	0	0	1	0	0	9	9	0	0	0	0	0	0	0	0	0	0	0	0

NOTES:

BIT CONFIGURATION:

BITS 37	BITS 36	
0	0	SET SINGLE TAG
0	1	SET CODE TAG
1	0	SET TRANSFER TAG
1	1	SET DOUBLE TAG

BITS 25 24
- 0 0 = STORAGE UNIT 0
- 0 1 = STORAGE UNIT 1
- 1 0 = STORAGE UNIT 2
- 1 1 = STORAGE UNIT 3

35 ⇒ 24 VARIANT CONTROL FIELD
- 33 = V3-8 READ EXTENDED STATUS
- 32 = V3-4 DISABLE EXTRA REVOLUTION
- 31 = V3-2 INHIBIT PERIPHERAL DEVICE
- 30 = V3-1 INHIBIT ERROR CORRECTION
- 25 = V2-2
- 24 = V2-1

} FILE ADDRESS EXTENSION

23 ⇒ 0 = SEGMENT ADDRESS (6 BINARY CODED DECIMAL DIGITS, WITH THE MSD IN BITS 23⇒20)

MV 1680A

Figure 11-25. 5N Disk File IOCW Format

									UE	UE	UE	UE
0									UE	UE	UE	UE
1								UE	UE	UE	UE	UE
0								UE	UE	UE	UE	UE
	44	40	36	32	28	24	20	16	12	8	4	0

ERROR CODE MEANING	HEXIDECIMAL UNIT ERROR CODES				
NOT READY	0	0	0	0	8
WARNING	0	0	1	0	8
WRITE OPERATION–LOCKOUT READ OPERATION– DATA CORRECTED	0	0	2	0	0
DISK EXCHANGE BUSY	0	0	1	0	0
EXTRA REVOLUTION TEST OPERATION – DISK EXCHANGE ID	0	0	8	0	0
ADDRESS ERROR FROM DISK EXCHANGE	0	0	1	8	0
PERIPHERAL BUS PARITY ERROR (COMMAND)	0	0	2	8	0
TRANSMISSION ERROR	0	0	4	8	0
CONTROLLER PARITY	0	4	0	8	0
WRITE OPERATION – PERIPHERAL BUS DATA PARITY ERROR READ OPERATION – READ ERROR	0	8	0	8	0

MV 1681

Figure 11-26. 5N Disk File Result Descriptor Format

SECTION 12

DATA COMMUNICATIONS SUBSYSTEM

GENERAL

The Data Communications Subsystem (DCS) consists of one or more independently powered B or C size cabinets. The DCS is not an integral part of the B 6800 operating system, but rather is an optional extension of the B 6800 system operating capabilities. Through the use of a data communications subsystem, a B 6800 system can service a network with a maximum of 2048 remote terminals, or data communications lines.

Figure 12-1 shows the various types and the main relationship between the logic modules of the DCS. The logical modules of the DCS are as follows:

a.	Data Communications Processor (DCP)	A DCS must contain at least one DCP, and may contain up to two DCP modules in a basic B 6800 system, or four DCP's in an expanded B 6800 system (including a memory port (channel B) expansion module).
b.	Adapter Cluster II (ACM II)	The ACM II module is used to interface up to 16 line adpaters to a DCP module. A DCP may interface with up to sixteen ACM II modules.
c.	Line Adapter Module (LA) (AC II type)	A DCS requires at least one LA module for each line that is interfaced to the DCP module. If the remote terminals or lines that are interfaced to the DCP are full duplex, then two LA's (two lines) are required. If the remote lines that are interfaced to the DCP are half-duplex, or simplex, then a single line adapter is used for each line.
d.	Local Memory Module(s) (LM)	A DCS may share the use of B 6800 system memory, or it may contain memory that is internal to the DCS (not used by the B6800 system). The DCS may contain up to four modules of local memory (96KB), with each local memory module containing 24KB of memory.
e.	Planar Core Local Memory	A DCS may contain a cabinet of planar core local memory instead of local memory modules (LM's). A planar core local memory cabinet contains 393 KB of storage capacity (minimum).

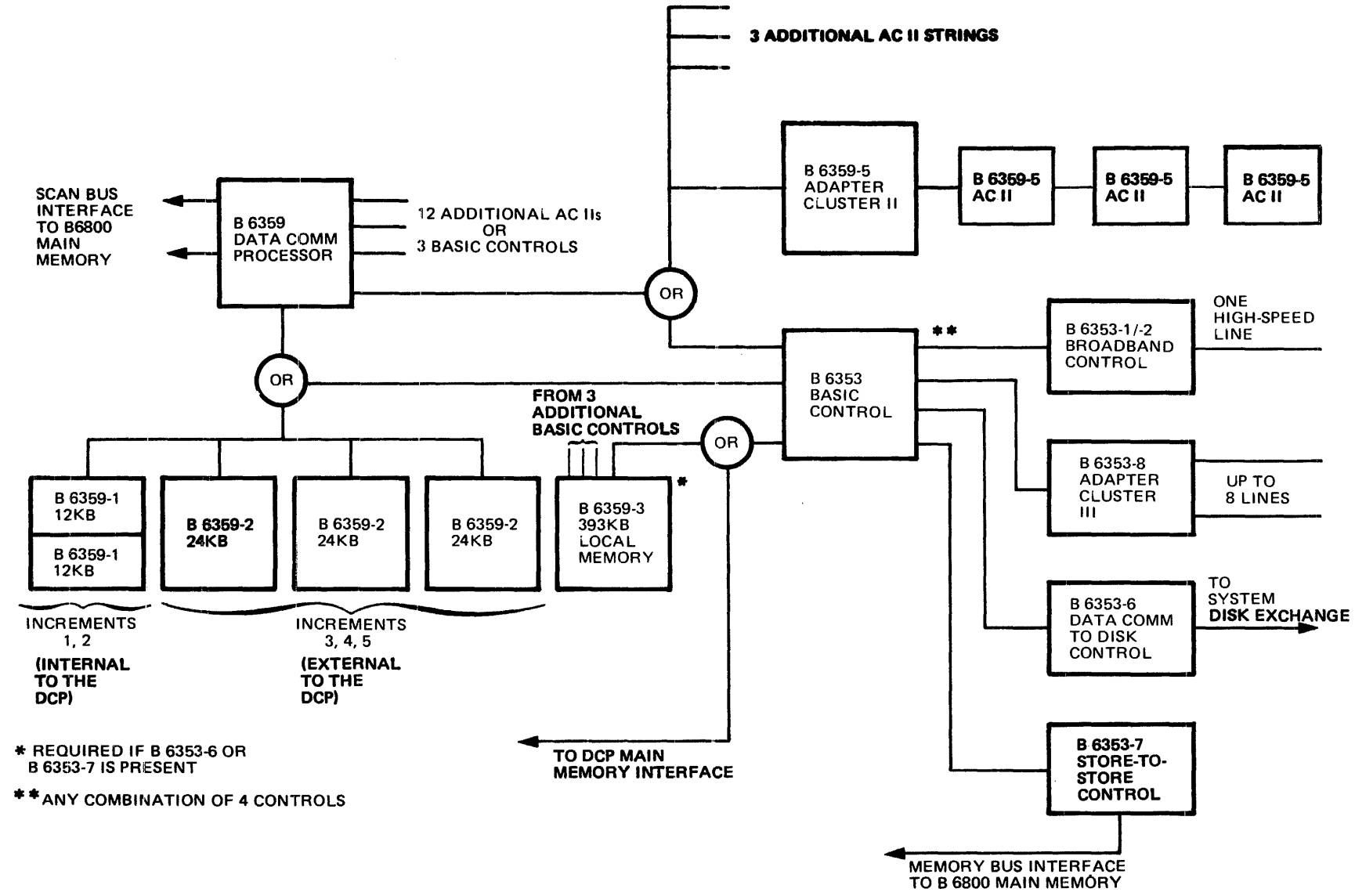


Figure 12-1. B 6800 Data Communications Subsystem Block Diagram

f. Basic Control

The basic control provides an interface between DCP and the Front End Controls (FEC). The five available FEC's are:

a. Broadband Binary Syncronous Control (BBSC).

b. Broadband Data Link Control (BDLC).

c. Adapter Cluster III (ACIII).

d. Data Comm to Disk Control (DCDC).

e. Store-to-Store Control (SSC).

The Basic Control can interface up to 4 FECs to the DCP. There is however a limitation in the combination of FECs that may be used. The Basic Control can only handle one DCDC and one SSC.

The Basic Control also provides the path to Planar Core for the DCP local memory interface.

DATA COMMUNICATIONS PROCESSOR

When the Data Communication Processor (DCP) is connected to Adapter Cluster II Modules (B6359-5) the DCP must handle all facets of both the transmission and receiving of data.

The DCP formats the communications that are transmitted to, or received from the lines, and it also formats the communications that are passed between the main system and the DCP. The DCP makes determinations of priority for the communications lines, conducts line polling operations, detects errors in line data or line discipline, and maintains status information about the communications lines connected to the ACMII(s).

When the DCP is connected to the basic control (B 6353) the DCP, under the direction of system software, sends control information to the front end controls. Once a front end control has received the control information it performs all controlling operations (disk write or memory write operations, line polling, line selecting, and line procedures), independently of the DCP. All priority resolution between requesting units is resolved by the basic control.

The DCP executes special machine language operator codes to perform its functions. The functions described above are encoded into groups of these machine language operators, and are stored in the local memory of the DCP. The encoded machine language functions are performed by the DCP on an "as required basis", and are essentially driven into execution by the detection of a pre-defined set of conditions.

The DCP is semi-autonomous in performing its functions. That is, the main system must initiate the operation of the DCP, or stop the DCP operation. Once the main system has initiated the DCP, the DCP will perform its normal functions until it receives a stop operation input from the main system, or until the DCP detects a non-recoverable operating condition. The detection of a non-recoverable condition will cause the DCP to branch to an error handling procedure located in main system memory. The DCP will cause a HEYU interrupt in the multiplexor and the interrupt level in the multiplexor will cause the main system to interrogate the DCP condition. In this manner, the main system will handle non-recoverable operating conditions in the data communications subsystem.

TERMINAL DEVICES

A terminal device is a hardware input/output device that is used to conduct two-way communications between itself, and a central system, or another terminal device. Terminals adhere to a preselected line discipline while performing their two-way communication. A line discipline is a set of hardware and software rules that establish the conditions under which communications are conducted between a terminal and the central system, or another terminal.

The line discipline for a communications line includes the format of the information and control signals that are transmitted through the line, and also the method for transmitting data over the line. The physical characteristics of the line, which determines the method that is used to transmit data over the line, are embodied in the hardware construction of the line adapter modules, and the adapter clusters. Each line adapter module in the DCS must be selected on the basis of the type of discipline that is to be practiced in exercising communications. There are five different types of line adapters used in an ACM II module, as follows:

a. Adapter-Data Set Connect.

b. Adapter-Direct Connect.

c. Adapter-Auto Call (dial in/out).

d. Adapter Connector Full-Duplex/Reverse Channel.

There are three different line adapters that can be used in the AC III module, as follows:

a. Adapter-Data Set Connect.

b. Adapter-Direct Connect.

c. Adapter-Auto Call (dial in/out).

The selection of a line discipline to be practiced for a data line is a user option. The user of a DCS must specify the characteristics of the data line adapters to be used, for each line in the DCS network.

BASIC CONTROL

The basic control provides an interface between the front end controls and the data communications processor. It connects directly to the DCP for the transfer of control information and has a path to the DCP local memory for message handling. Each basic control utilizes a cluster interface position within the DCP and will service any combination of four of the following new controls:

BROADBAND CONTROL

Each Broadband Control (BBC) operates asynchronously with the DCP and services one data communications line ranging from 19,200/BPS to 1,344,000/BPS. The BBC will support either binary synchronous or Burroughs Data Link Control line procedures and communicates with the DCP at the message level.

DATA COMM TO DISK CONTROL

This control provides the data comm subsystem with the capability to support network continuation, audit and recovery through disk tanking. This allows network message auditing and the reception and accumulation of messages in the event of a system failure and is intended to provide this function for a limited time until the central system becomes operational again. The Data Comm to Disk Control interfaces to a 23 millisecond head-per-track disk file subsystem shared by the main system, thereby providing a common storage area for subsequent system access.

STORE-TO-STORE CONTROL

This control provides high-speed block transfer of messages between the DCP Planar Core local memory dedicated to the DCS, and the main system memory independent of DCP operation. This further distribution of message handling increases the throughput of the data communications subsystem.

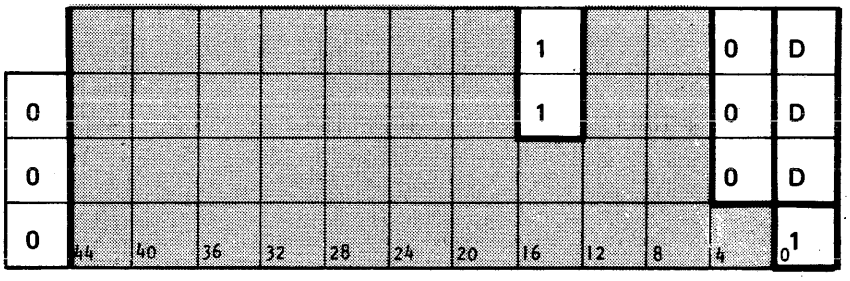
ADAPTER CLUSTER III

The Model III adapter cluster provides a facility for the asynchronous control of up to eight half-/full-duplex lines at speeds of up to 9600 BPS. AC III optimizes line procedures for Burroughs terminal equipment by supporting data transmission according to standard Burroughs line protocols. The AC III communicates with the DCP at the message level.

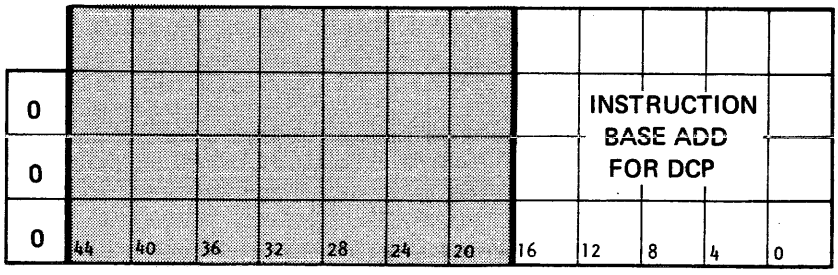
DATA COMMUNICATIONS SUBSYSTEM SCAN BUS INTERFACE

The DCS is a semi-autonomous subsystem. After the B 6800 CPU has initialized the DCS, the subsystem conducts data communications operations in an autonomous and independent manner, until the CPU directs that such operations are to be terminated. Thus, except for controlling the start and stop operations, the B 6800 CPU does not exercise control over the DCS.

The interface that is used by the B 6800 CPU to communicate control data to the DCS is the external subsystem scan bus. This external scan bus is defined and discussed in section five of this manual. The scan bus instructions that are passed to a DCS from the B 6800 CPU are as follows:



19:2 = DCP (EXTERNAL) SCAN BUS ADDRESS.
7:3 = DCP SCAN FUNCTION CODE.
4:3 = DCP UNIT ID NUMBER.



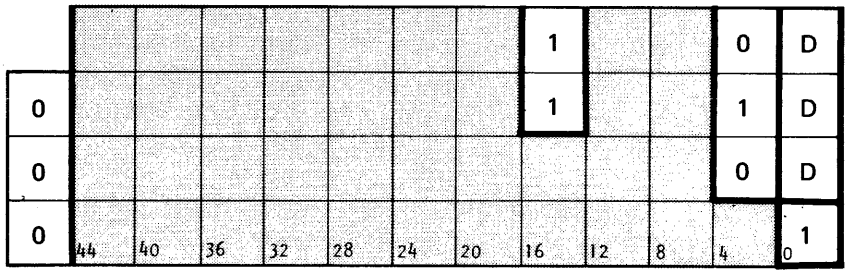
MV1691

SCAN OUT INITIALIZE DCP

The CPU executes the scan out initialize DCP operation to cause the DCS to begin data communications operations. The scan function word is present in the top of stack register of the data processor when the SCNO (954B) is executed. The data word, which is the second word in the data processor stack registers, specifies the beginning address of the DCS instructions, in the B 6800 system main memory. If the DCS contains local DCS memory, the data communications

operator code will be transferred from the main memory of the B 6800 system to the DCS local mamory, where the DCS will proceed to execute the programmed code. If the DCS does not contain local memory, the DCS will proceed to fetch its program code from the main B 6800 system memory, a word at a time, and to execute the program code.

Once the B 6800 system has caused the DCS to be initialized, the DCS will continue to operate in an autonomous manner until an interrupt occurs, or until the CPU directs that the DCS is to stop operations. The CPU executes a scan out halt command to the DCS, as follows:



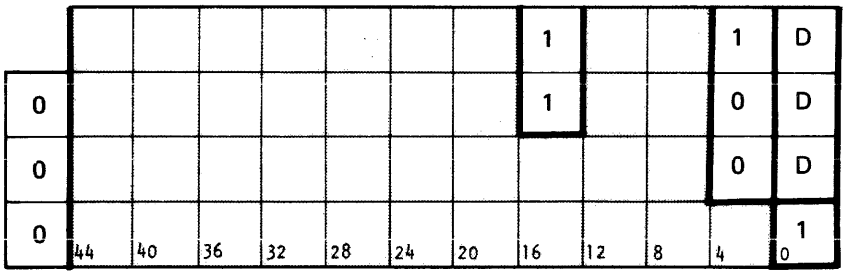
19:2 = DCP (EXTERNAL) SCAN BUS ADDRESS.
7:3 = DCP SCAN FUNCTION CODE.
4:3 = DCP UNIT ID NUMBER.

MV1692

SCAN-OUT HALT DCP

The DCP halt function word is present in the data processor when the SCNO (954B) instruction is executed by the CPU. No data word is used for this DCS control operation. This command causes the DCP to stop operations at the end of the present DCS command, and to respond by causing an interrupt to be present in the external subsystem interrupt collector logic of the multiplexor. The DCS will not resume operations until the CPU scans out an initialize instruction.

The B 6800 system has a way to pass information to the DCS without causing the DCS to stop operating, through the use of the scan out attention needed instruction. This instruction is executed in the same way as the halt scan out command is executed, with the following function word present in the data processor top of stack register.



19:2 = DCP (EXTERNAL) SCAN BUS ADDRESS.
7:3 = DCP SCAN FUNCTION CODE.
4:3 = DCP UNIT ID NUMBER.

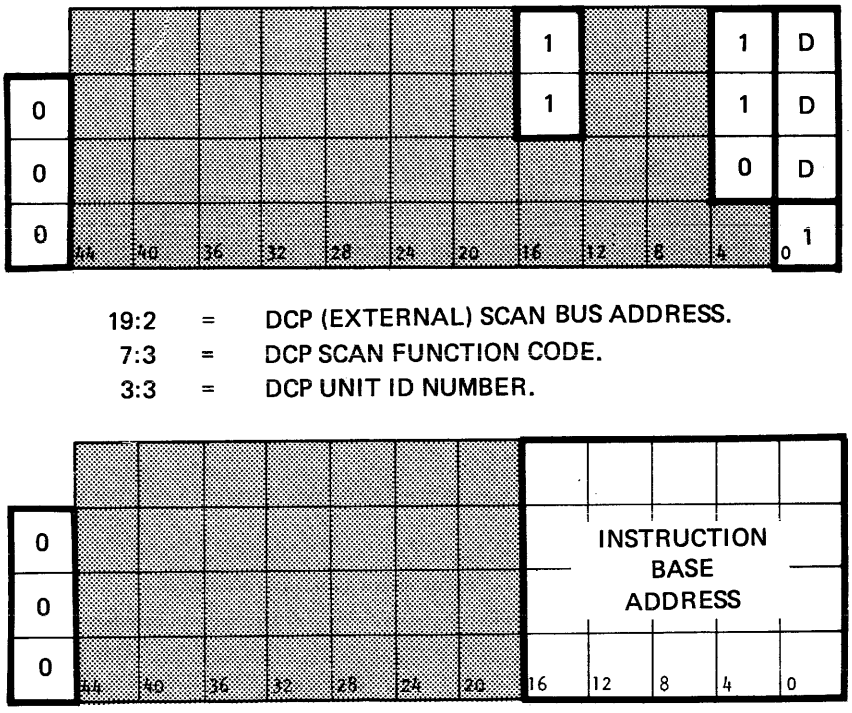
MV1693

SCAN-OUT SYSTEM ATTENTION NEEDED

When the DCS receives the attention needed scan out instruction, it looks in a memory queue (a predetermined location in local memory) for the text of the system information. The DCS will handle the information that it finds in the memory queue, in the manner specified.

The Set IBA command is only used within the Enhanced Data Comm environment and is a variation of the initialize command. The CPU executes the "Set IBA" only after the B 6800 has been reinitialized after an operational interruption (HALT/LOAD). When the DCP detects the B 6800 is not operational, the DCS goes into a special running operation so that incoming information is not lost.

The "Set IBA" command passes a 20-bit Instruction Base Address to the DCP, to indicate where the DCP instruction code is located. Unlike the initialize command, the "Set IBA" command does not cause the DCP to fetch instruction words from the system memory. If the DCP is running when the "Set IBA" occurs then the CPU must also issue a system attention needed, so that the DCP will utilize the new IBA.



19:2 = DCP (EXTERNAL) SCAN BUS ADDRESS.
7:3 = DCP SCAN FUNCTION CODE.
3:3 = DCP UNIT ID NUMBER.

MV1694

SCAN-OUT SET IBA (ENHANCED DATA COMM ONLY)

DATA COMMUNICATIONS SUBSYSTEM MEMORY INTERFACE

In addition to the external scan bus interface, the DCS shares a memory interface to channel B of the memory control, in the CPU cabinet. If the number of external subsystem units that are connected to channel B of the memory control is one or two subsystems, then the subsystems are interfaced directly to the memory control, in parallel. These subsystems contend with each other for access to the main system memory resources. If the number of subsystems that are interfaced to the memory control is three or more, then a memory port expansion module must be used to control memory access contention between the external subsystems.

If two external subsystems share the channel B interface to main system memory, the two subsystems must control the use of the memory bus such that only one subsystem may access memory at any one time. A special memory bus control interface is connected between the subsystems that share the memory bus for access to main system memory. Whenever one of the two subsystems is accessing memory, a signal on the special memory control bus causes the other subsystem to be denied access to memory until the access to memory is completed. If both subsystems attempt to access memory at the same time, then one subsystem will have priority, and will be granted use of the bus, while the other subsystem will be forced to wait until the first subsystem has completed its access of memory.

SECTION 13

B 6800 BUS INTERFACE CONTROL (READER/SORTER SUBSYSTEM)

THE BIC MODULE

The Bus Interface Control (BIC) is a component of the B 6800 that is used to interface a Reader/Sorter network to the main B 6800 system. The BIC is housed in a single independently powered cabinet. A BIC cabinet is not an integral part of the B 6800 system, but interfaces an operating subsystem of the B 6800. Through the use of a BIC, the B 6800 system controls the operations of up to three Reader Sorter Processors (RSP), and indirectly controls the operations of up to 12 Reader/Sorter input/output devices that are attached to the RSPs. The RSP has access to the memory resources of the B 6800 system through the memory interface of the BIC module.

The B 6800 system communicates control instructions to a BIC module through use of the external scan bus interface of the CPU. A maximum of four external subsystems may share the use of the external scan bus interface to the B 6800. These four external subsystems may be a combination of data communications subsystems, and BIC subsystems. However, multiple BIC's may not be used in the same configuration with multiple data communication subsystems.

Figure 13-1 shows a single BIC module, and the maximum units (RSP's and Reader/Sorter IO devices) that may be interfaced with the B 6800 system through the use of a single BIC module subsystem. Although this figure shows that up to three RSP's may be interfaced to a single BIC, typical current installations only interface one or two RSP's to a BIC module.

BUS INTERFACE CONTROL SCAN BUS INTERFACE

The B 6800 system uses the external scan bus interface of the CPU to pass control information and instructions to the BIC subsystem. The control information and instructions that are passed to the BIC on the external scan bus interface originate in the data processor top of stack registers. The data processor executes a SCNI (954A), or a SCNO (954B) operator to cause the external scan bus address lines to assume the value of the top word in the data processor stack registers (which is the scan function word). If the operator code that was performed in the data processor was a SCNO operator, the second word in the top of stack registers (the scan data word) is also passed to the external scan bus. If the operator code that was performed in the data processor was a SCNI operator, the BIC responds by placing a word of data in the second word of the top of stack registers in the data processor.

SCAN OUT FUNCTIONS

Figure 13-2 shows the format of the scan out function word located in the top of stack register of the data processor at the start of a data processor scan out operation. There are six different function codes that may be contained in the scan out function word.

a. Scan Address Required (set scan address) function.

b. Set RSP Interrupt function.

c. Set Bounds Registers function.

d. Halt RSP function.

e. Clear-load RSP function.

f. RSP (Port) Lockout function.

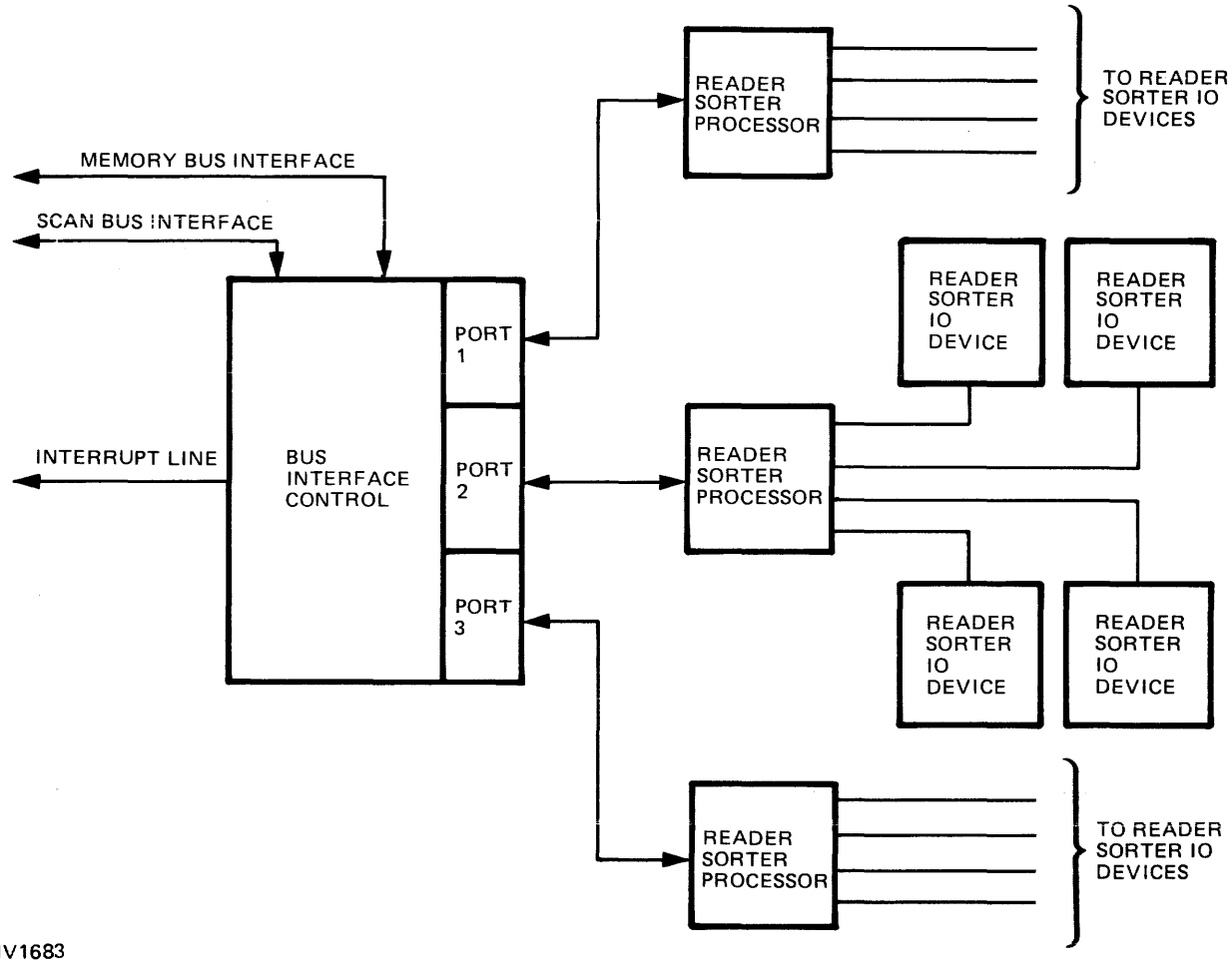
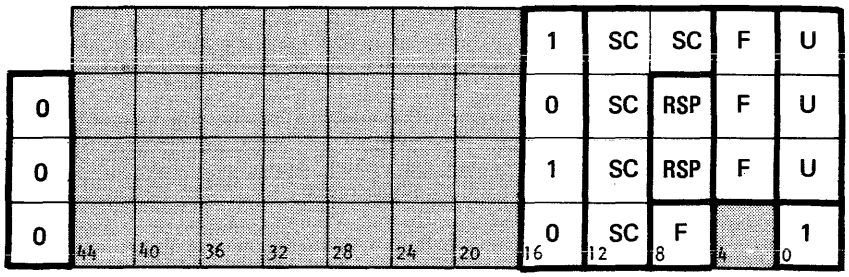


Figure 13-1. BUS Interface Subsystem Modules

The Set Scan Address function is used by the B 6800 system as part of the subsystem-to-main-system synchronization process (hand shaking). The B 6800 system performs extensive synchronization (hand shaking) operations during the initialization process because the RSP is a small system processor which operates asynchronously, with respect to the B 6800 system. The RSP executes micro-operator code command instructions which are contained in an M memory that is internal to the RSP, and cannot be accessed directly by the B 6800 system. The data word that is passed to the BIC by this scan out command is controlled by the system software and is not defined by a fixed hardware word format.

The set RSP interrupt function is used by the B 6800 system software control procedures to pass software control information to the RSP during normal subsystem operations. The software control information is contained in the scan out function word, and no data word is passed to the BIC.

The set bounds registers function is used to establish the boundries of the area in B 6800 main system memory that may be accessed by the BIC. The bounds data word (see figure 13-3) includes both a lower, and an upper address boundry. The BIC compares a memory address with the two bounds values, and detects an error condition if the memory address is not within the range of addresses specified by the bounds. The bounds function is used to protect the B 6800 system against inadvertent accessing of reserved memory addresses by the BIC.



50:3 = TAG FIELD.
THIS FIELD IS ALWAYS EQUAL TO ZERO.

19:4 = BIC SCAN BUS IDENTIFICATION CODE. THIS FIELD IS ALWAYS EQUAL TO A(HEX) FOR BIC SCAN BUS OPERATIONS

15:5 = SOFTWARE CONTROL FIELD.
- 00000 = WAIT B 6800 MEMORY DUMP IN PROGRESS
- 00001 = MEMORY DUMP FINISHED
- 00010 = NORMAL INTERRUPT
- 00011 = PAUSE. (INSTRUCTS RSP TO WAIT)
- 00100 = CLEAR
- 00101 = RSP MEMORY DUMP IN PROGRESS

10:2 = RSP IDENTIFICATION CODE
- 00 = INVALID CODE
- 01 = RSP NO. 1 (PORT 1)
- 10 = RSP NO. 2 (PORT 2)
- 11 = RSP NO. 3 (PORT 3)

8:4 = BIC FUNCTION CODE FIELD.
- 0000 = SCAN ADDRESS REQUIRED FUNCTION
- 0010 = SET RSP INTERRUPT
- 0100 = SET BOUNDS REGISTERS
- 1000 = HALT RSP
- 1010 = CLEAR-LOAD RSP
- 1110 = RSP (PORT) LOCKOUT

3:3 = BIC UNIT IDENTIFICATION FIELD. A B 6800 SYSTEM MAY HAVE FROM 1 THROUGH 7 BIC UNITS ATTACHED
- 000 = INVALID CODE
- 001 = BIC UNIT NO. 1
- 010 = BIC UNIT NO. 2
- 011 = BIC UNIT NO. 3
- 100 = BIC UNIT NO. 4
- 101 = BIC UNIT NO. 5
- 110 = BIC UNIT NO. 6
- 111 = BIC UNIT NO. 7

0:1 = THIS BIT IS REQUIRED TO BE A BINARY 1 VALUE

MV 1684

Figure 13-2. BIC Scan-Out Function Word

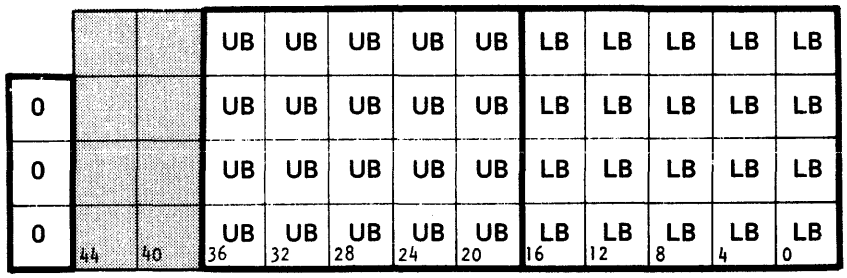


Figure 13-3. Set Bounds Registers Data Word

The halt RSP function is used to cause the RSP to stop processing information from the reader/sorter devices. This function is not normally performed because the RSP lockout function accomplishes the same thing, and is utilized instead of the halt function.

The clear-load RSP function is used by the B 6800 system to cause an RSP to begin operations. This function passes a data word (see figure 13-4) to the BIC, which contains an address, and a length (word count) field. The address field defines the base address of an area in main system memory that contains the RSP program data. The word count field specifies how many words of main system memory are used to contain the RSP program data. The BIC accesses the program data in main system memory, and loads a copy of the program into the local (S) memory of the RSP, beginning in address zero. When the BIC has loaded the control program data into the (S) memory of the RSP, the RSP branches to address zero of its S memory and proceeds to execute the instructions which it contains. Thus, the clear-load function is used to start the RSP into execution of a B 6800 main system directed sequence of operations.

The RSP lockout function is used to cause the RSP to stop operating. Figure 13-1 shows that the BIC has three ports through which an RSP is interfaced to the B 6800 system. When a port lockout function is performed, the port of the BIC that is specified (in the RSP field of the function word) is locked out, and the interface path between the main B 6800 system and the RSP that is attached to the port is broken. The RSP has no provision for causing the interface through a locked out port of the BIC to be resumed, therefore the RSP that is associated with the port is effectively eliminated from the subsystem, and has no effect upon the operation of other RSP's (ports) in the subsystem. The data word that is passed to the BIC by this scan out command is not specified.

SCAN IN FUNCTIONS

Figure 13-5 shows the format of the scan in function word that is located in the top of stack register of the data processor at the start of a data processor scan out operation. There are two different function codes that may be contained in the scan in function word, as follow:

a. Scan Address Required (set scan address) function.

b. Read BIC Status function.

The Scan Address function is used by the B 6800 system as a part of the hand shaking procedures, for synchronization of the subsystem to the main system, during the initialization process of the start up procedures. When the scan address

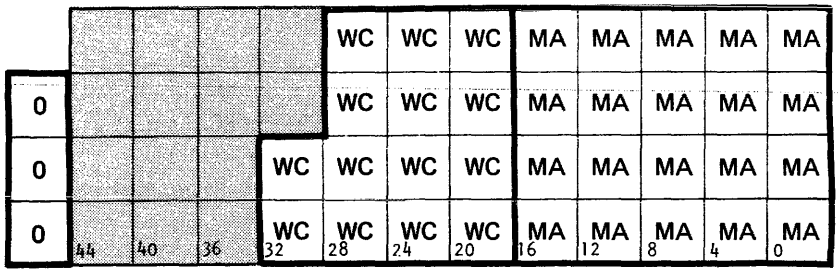
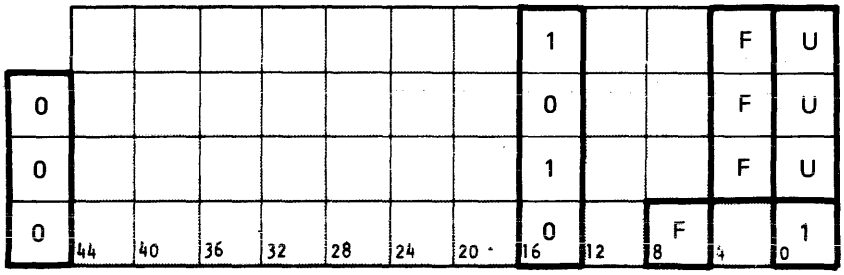


Figure 13-4. Clear-Load Data Word



50:3 = TAG FIELD.
ALWAYS EQUAL TO ZERO

19:4 = BIC SCAN BUS IDENTIFICATION CODE. THIS FIELD IS ALWAYS EQUAL TO A (HEX) FOR BIC SCAN BUS OPERATIONS

8:4 = BIC FUNCTION CODE FIELD
- 0000 = SCAN ADDRESS
- 1100 = READ BIC STATUS

3:3 = BIC UNIT IDENTIFICATION FIELD
- 000 = INVALID UNIT IDENTIFICATION
- 001 = BIC UNIT NO. 1
- 010 = BIC UNIT NO. 2
- 011 = BIC UNIT NO. 3
- 100 = BIC UNIT NO. 4
- 101 = BIC UNIT NO. 5
- 110 = BIC UNIT NO. 6
- 111 = BIC UNIT NO. 7

MV 1687

Figure 13-5. BIC Scan-In Function Word

function is performed, the RSP responds by transmitting a data word to the B 6800 system. The form of the data word is not specified, because the information contained in the word is only significant to the B 6800 software operating system.

The scan in Read BIC Status function causes the BIC to respond to the B 6800 system by placing a BIC status word in the second word of the data processor top of stack registers. The format of the BIC status word is shown in figure 13-6. There are three fields of information that give status information about one or more of the BIC ports. Each port status field consists of five bits of data that have the following meaning:

P1, P2, P3	The Pn (RSP powered off) bit is set if the corresponding RSP is powered off.
PL1, PL2, PL3	The PLn (Port n Locked Out) bit is set if the corresponding port is locked out, or the RSP has been halted. (If an RSP is powered off then the corresponding PLn bit will be set.)
T1, T2, T3	The Tn (RSP Memory Access Timeout) bit is set if a timeout has occurred for the corresponding RSP during a memory access.
I1, I2, I3	The In (Interrupt from RSP number n) bit is set if the corresponding RSP has sent an interrupt to the B 6800 system.
E1, E2, E3	The En (Exception) bit is set if a corresponding PLn, Tn, or In bit is set in this status word for a corresponding port status report.

If any one of the three port adapter cards of the BIC are not installed in the BIC then the corresponding port status field in the BIC status response word is not used, and all bits for that port status are reset.

BUS INTERFACE CONTROL MEMORY INTERFACE

In addition to the external scan bus interface, the BIC shares a memory interface to channel B of the memory control, in the CPU cabinet. If the number of external subsystems that are interfaced to channel B of the memory control is one or two subsystems, then the subsystems are interfaced directly to the memory control, in parallel. These subsystems contend with each other for access to the main system memory resources. If the number of subsystems that are interfaced to the memory control is three or four, then a memory port expansion module is required to control memory access contention between the external subsystems.

The BIC module contains bounds checking logic circuits to protect the B 6800 system against inadvertent overwriting of a memory address. This inadvertent overwriting could otherwise occur because the memory addresses that are accessed

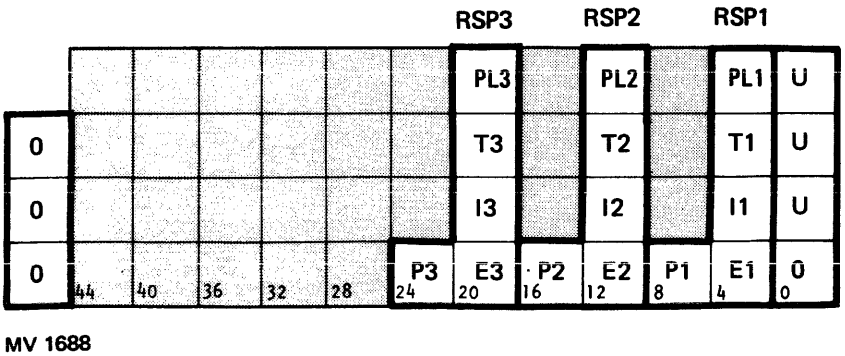


Figure 13-6. Read BIC Status Response

by the BIC are automatically incremented when the memory transfer is initiated. The bounds checking logic of the BIC prevent this condition from occurring. The bounds limits are assigned to a BIC by the B 6800 system MCP, and are dynamic in nature. The MCP will determine how much memory a BIC will be able to access, and will set the bounds registers of the BIC accordingly. The process of setting the memory bounds within a BIC subsystem were defined previously in this section of this manual, in the subsection titled Scan Out Functions.

If two external subsystems share the channel B interface to main memory, in parallel, then the two subsystems must control the use of the memory bus such that only one of the two subsystems may access memory at any one time. A special memory control interface bus is connected between the two subsystems that are connected in parallel. Whenever one of the two subsystems is accessing memory, a signal on this special memory control bus causes the other subsystem to be denied access to memory. If both of the subsystems try to access memory at the same time, one of the subsystems will have priority for the use of the memory bus, and the other subsystem will have to wait until the one that has priority has completed its access, before being allowed to access memory.

APPENDIX A

OPERATORS, ALPHABETICAL LIST

Name	Mnemonic	Hexa-Decimal Code
ADD	ADD	80
BIT RESET	BRST	9E
BIT SET	BSET	96
BRANCH FALSE	BRFL	A0
BRANCH TRUE	BRTR	A1
BRANCH UNCONDITIONAL	BURN	A2
CHANGE SIGN BIT	CHSN	8E
COMPARE CHARACTERS EQUAL DESTRUCTIVE	CEQD	F4
COMPARE CHARACTERS EQUAL, UPDATE	CEQU	FC
COMPARE CHARACTERS GREATER OR EQUAL, DESTRUCTIVE	CGED	F1
COMPARE CHARACTERS GREATER OR EQUAL, UPDATE	CGEU	F9
COMPARE CHARACTERS GREATER, DESTRUCTIVE	CGTD	F2
COMPARE CHARACTERS GREATER, UPDATE	CGTU	FA
COMPARE CHARACTERS LESS OR EQUAL, DESTRUCTIVE	CLED	F3
COMPARE CHARACTERS LESS OR EQUAL, UPDATE	CLEU	FB
COMPARE CHARACTERS LESS, DESTRUCTIVE	CLSD	F0
COMPARE CHARACTERS LESS, UPDATE	CLSU	F8
COMPARE CHARACTERS NOT EQUAL, DESTRUCTIVE	CNED	F5
COMPARE CHARACTERS NOT EQUAL, UPDATE	CNEU	FD
CONDITIONAL HALT (all modes)	HALT	DF
COUNT BINARY ONES	CBON	95BB
DELETE TOP OF STACK	DLET	B5
DISABLE EXTERNAL INTERRUPT	DEXI	9547
DIVIDE	DIVD	83
DOUBLE LOAD A	DLA	E0
DOUBLE LOAD A INCREMENT	DLAI	E9
DOUBLE LOAD B	DLB	E2
DOUBLE LOAD B INCREMENT	DLBI	EB
DOUBLE LOAD C	DLC	E4
DOUBLE LOAD C INCREMENT	DLCI	ED
DOUBLE STORE A	DSA	F8
DOUBLE STORE A INCREMENT	DSAI	F9
DOUBLE STORE B	DSB	FA
DOUBLE STORE B INCREMENT	DSBI	FB
DOUBLE STORE C	DSC	FC
DOUBLE STORE C INCREMENT	DSCI	FD
DUPLICATE TOP OF STACK	DUPL	B7
DYNAMIC BIT RESET	DBRS	9F
DYNAMIC BIT SET	DBST	97
DYNAMIC BRANCH FALSE	DBFL	A8
DYNAMIC BRANCH TRUE	DBTR	A9

Name	Mnemonic	Hexa-Decimal Code
DYNAMIC BRANCH UNCONDITIONAL	DBUN	AA
DYNAMIC FIELD INSERT	DINS	9D
DYNAMIC FIELD ISOLATE	DISO	9B
DYNAMIC FIELD TRANSFER	DFTR	99
DYNAMIC SCALE LEFT	DSLF	C1
DYNAMIC SCALE RIGHT FINAL	DSRF	C7
DYNAMIC SCALE RIGHT ROUND	DSRR	C9
DYNAMIC SCALE RIGHT SAVE	DSRS	C5
DYNAMIC SCALE RIGHT TRUNCATE	DSRT	C3
ENABLE EXTERNAL INTERRUPTS	EEXI	9546
END EDIT (edit mode)	ENDE	DE
END FLOAT (edit mode)	ENDF	D5
ENTER	ENTR	AB
EQUAL	EQUL	8C
ESCAPE TO 16-BIT INSTRUCTION	VARI	95
EVALUATE	EVAL	AC
EXCHANGE	EXCH	B6
EXECUTE SINGLE MICRO, SINGLE POINTER UPDATE	EXPU	DD
EXECUTE SINGLE MICRO, DESTRUCTIVE	EXSD	D2
EXECUTE SINGLE MICRO, UPDATE	EXSU	DA
EXIT	EXIT	A3
EXTENDED MULTIPLY	MULX	8F
FIELD INSERT	INSR	9C
FIELD ISOLATE	ISOL	9A
FIELD TRANSFER	FLTR	98
GREATER THAN	GRTR	8A
GREATER THAN OR EQUAL	GREQ	89
IDLE UNTIL INTERRUPT	IDLE	9544
INDEX	INDX	A6
INDEX AND LOAD NAME	NXLN	A5
INDEX AND LOAD VALUE	NXLV	AD
INPUT CONVERT, DESTRUCTIVE	ICVD	CA
INPUT CONVERT UPDATE	ICVU	CB
INSERT CONDITIONAL (edit mode)	INSC	DD
INSERT DISPLAY SIGN (edit mode)	INSG	D9
INSERT MARK STACK	IMKS	CF
INSERT OVERPUNCH (edit mode)	INOP	D8
INSERT UNCONDITIONAL (edit mode)	INSU	DC
INTEGER DIVIDE	IDIV	84
INTEGERIZE, ROUNDED	NTGR	87
INTEGERIZE, TRUNCATED	NTIA	86
INTEGERIZE, ROUNDED DOUBLE-PRECISION	NTGD	9587
INVALID OPERATOR (all modes)	NVLD	FF
LEADING ONE TEST	LOG2	958B
LINKED LIST LOOKUP	LLLU	95BD
LESS THAN	LESS	88
LESS THAN OR EQUAL	LSEQ	8B
LIT CALL ONE	ONE	B1

Name	Mnemonic	Hexa-Decimal Code
LIT CALL ZERO	ZERO	B0
LIT CALL 8 BITS	LT8	B2
LIT CALL 16 BITS	LT16	B3
LIT CALL 48 BITS	LT48	BE
LOAD	LOAD	BD
LOAD A	LDA	E0
LOAD A INCREMENT	LDAI	E1
LOAD B	LDB	E2
LOAD B INCREMENT	LDBI	E3
LOAD C	LDC	E4
LOAD C INCREMENT	LDCI	E5
LOAD TRANSPARENT	LODT	95BC
LOGICAL AND	LAND	90
LOGICAL EQUAL	SAME	94
LOGICAL EQUIVALENCE	LEQV	93
LOGICAL NEGATE	LNOT	92
LOGICAL OR	LOR	91
MAKE PROGRAM CONTROL WORD	MPCW	BF
MARK STACK	MKST	AE
MASKED SEARCH FOR EQUAL	SRCH	95BE
MOVE CHARACTERS (edit mode)	MCHR	D7
MOVE NUMERIC UNCONDITIONAL (edit mode)	MVNU	D6
MOVE TO STACK	MVST	95AF
MOVE WITH FLOAT (edit mode)	MFLT	D1
MOVE WITH INSERT (edit mode)	MINS	D0
MULTIPLY	MULT	82
NAME CALL	NAMC	40⇒7F
NO OPERATION (all modes)	NOOP	FE
NORMALIZE	NORM	958E
NOT EQUAL	NEQL	8D
OCCURS INDEX	OCRX	9585
OVERWRITE DESTRUCTIVE	OVRD	BA
OVERWRITE NON-DESTRUCTIVE	OVRN	BB
PACK DESTRUCTIVE	PACD	D1
PACK UPDATE	PACU	D9
PUSH DOWN STACK REGISTERS	PUSH	B4
READ AND CLEAR OVERFLOW FLIP FLOP	ROFF	D7
READ COMPARE FLIP FLOP	RCMP	95B3
READ PROCESSOR IDENTIFICATION	WHOI	954E
READ PROCESSOR REGISTER	RPRR	95B8
READ TAG FIELD	RTAG	95B5
READ TRUE/FALSE FLIP FLOP	RTFF	DE
READ WITH LOCK	RDLK	95BA
REMAINDER DIVIDE	RDIV	85
RESET FLOAT (edit mode)	RSTF	D4
RETURN	RETN	A7
ROTATE STACK DOWN	RSDN	95B7
ROTATE STACK UP	RSUP	95B6

Name	Mnemonic	Hexa-Decimal Code
SCALE LEFT	SCLF	C0
SCALE RIGHT FINAL	SCRF	C6
SCALE RIGHT ROUNDED	SCRR	C8
SCALE RIGHT SAVE	SCRS	C4
SCALE RIGHT TRUNCATE	SCRT	C2
SCAN IN	SCNI	954A
SCAN OUT	SCNO	954B
SCAN WHILE EQUAL, DESTRUCTIVE	SEQD	95F4
SCAN WHILE EQUAL, UPDATE	SEQU	95FC
SCAN WHILE FALSE, DESTRUCTIVE	SWFD	95D4
SCAN WHILE FALSE, UPDATE	SWFU	95DC
SCAN WHILE GREATER OR EQUAL, DESTRUCTIVE	SGED	95F1
SCAN WHILE GREATER OR EQUAL, UPDATE	SGEU	95F9
SCAN WHILE GREATER, DESTRUCTIVE	SGTD	95F2
SCAN WHILE GREATER, UPDATE	SGTU	95FA
SCAN WHILE LESS OR EQUAL, DESTRUCTIVE	SLED	95F3
SCAN WHILE LESS OR EQUAL, UPDATE	SLEU	95FB
SCAN WHILE LESS, DESTRUCTIVE	SLSD	95F0
SCAN WHILE LESS, UPDATE	SLSU	95F8
SCAN WHILE NOT EQUAL, DESTRUCTIVE	SNED	95F5
SCAN WHILE NOT EQUAL, UPDATE	SNEU	95FD
SCAN WHILE TRUE, DESTRUCTIVE	SWTD	95D5
SCAN WHILE TRUE, UPDATE	SWTU	95DD
SET DOUBLE TO TWO SINGLES	SPLT	9543
SET EXTERNAL SIGN	SXSN	D6
SET INTERVAL TIMER	SINT	9545
SET PROCESSOR REGISTER	SPRR	95B9
SET TAG FIELD	STAG	95B4
SET TO DOUBLE-PRECISION	XTND	CE
SET TO SINGLE-PRECISION, ROUNDED	SNGL	CD
SET TO SINGLE-PRECISION, TRUNCATED	SNGT	CC
SET TWO SINGLES TO DOUBLE	JOIN	9542
SKIP FORWARD DESTINATION CHARACTERS (edit mode)	SFDC	DA
SKIP FORWARD SOURCE CHARACTERS (edit mode)	SFSC	D2
SKIP REVERSE DESTINATION CHARACTERS (edit mode)	SRDC	DB
SKIP REVERSE SOURCE CHARACTERS (edit mode)	SRSC	D3
STEP AND BRANCH	STBR	A4
STORE A	STA	F0
STORE A INCREMENT	STAI	F1
STORE B	STB	F2
STORE B INCREMENT	STBI	F3
STORE C	STC	F4
STORE C INCREMENT	STCI	F5
STORE DESTRUCTIVE	STOD	B8
STORE NON-DESTRUCTIVE	STON	B9
STRING ISOLATE	SISO	D5

Name	Mnemonic	Hexa-Decimal Code
STUFF ENVIRONMENT	STFF	AF
SUBTRACT	SUBT	81
TABLE ENTER EDIT, DESTRUCTIVE	TEED	D0
TABLE ENTER EDIT, UPDATE	TEEU	D8
TRANSFER UNCONDITIONAL, DESTRUCTIVE	TUND	E6
TRANSFER UNCONDITIONAL, UPDATE	TUNU	EE
TRANSFER WHILE EQUAL, DESTRUCTIVE	TEQD	E4
TRANSFER WHILE EQUAL, UPDATE	TEQU	EC
TRANSFER WHILE GREATER OR EQUAL, DESTRUCTIVE	TGED	E1
TRANSFER WHILE GREATER OR EQUAL, UPDATE	TGEU	E9
TRANSFER WHILE GREATER, DESTRUCTIVE	TGTD	E2
TRANSFER WHILE GREATER, UPDATE	TGTU	EA
TRANSFER WHILE LESS OR EQUAL, DESTRUCTIVE	TLED	E3
TRANSFER WHILE FALSE, DESTRUCTIVE	TWFD	95D2
TRANSFER WHILE FALSE, UPDATE	TWFU	95DA
TRANSFER WHILE TRUE, DESTRUCTIVE	TWTD	95D3
TRANSFER WHILE TRUE, UPDATE	TWTU	95DB
TRANSFER WHILE LESS OR EQUAL, UPDATE	TLEU	EB
TRANSFER WHILE LESS, DESTRUCTIVE	TLSD	E0
TRANSFER WHILE LESS, UPDATE	TLSU	E8
TRANSFER WHILE NOT EQUAL, DESTRUCTIVE	TNED	E5
TRANSFER WHILE NOT EQUAL, UPDATE	TNEU	ED
TRANSFER WORDS OVERWRITE DESTRUCTIVE	TWOD	D4
TRANSFER WORDS OVERWRITE UPDATE	TWOU	DC
TRANSFER WORDS, DESTRUCTIVE	TWSD	D3
TRANSFER WORDS, UPDATE	TWSU	DB
TRANSLATE	TRNS	95D7
UNPACK ABSOLUTE, DESTRUCTIVE	UABD	95D1
UNPACK ABSOLUTE, UPDATE	UABU	95D9
UNPACK SIGNED, DESTRUCTIVE	USND	95D0
UNPACK SIGNED, UPDATE	USNU	95D8
VALUE CALL	VALC	00 ⇒ 3F
VECTOR BRANCH	VEBR	EE
VECTOR MODE ENTER MULTIPLE	VMEM	EF
VECTOR MODE ENTER SINGLE	VMES	E7
VECTOR MODE EXIT	VMEX	E6

APPENDIX B

OPERATORS, NUMERICAL LIST

Hexa-Decimal Code	Name	Mnemonic
PRIMARY MODE		
00⇒3F	VALUE CALL	VALC
40⇒7F	NAME CALL	NAMC
80	ADD	ADD
81	SUBTRACT	SUBT
82	MULTIPLY	MULT
83	DIVIDE	DIVD
84	INTEGER DIVIDE	IDIV
85	REMAINDER DIVIDE	RDIV
86	INTEGERIZE, TRUNCATED	NTIA
87	INTEGERIZE, ROUNDED	NTGR
88	LESS THAN	LESS
89	GREATER THAN OR EQUAL	GREQ
8A	GREATER THAN	GRTR
8B	LESS THAN OR EQUAL	LSEQ
8C	EQUAL	EQUL
8D	NOT EQUAL	NEQL
8E	CHANGE SIGN BIT	CHSN
8F	EXTENDED MULTIPLY	MULX
90	LOGICAL AND	LAND
91	LOGICAL OR	LOR
92	LOGICAL NEGATE	LNOT
93	LOGICAL EQUIVALENCE	LEQV
94	LOGICAL EQUAL	SAME
95	ESCAPE TO 16-BIT INSTRUCTION	VARI
96	BIT SET	BSET
97	DYNAMIC BIT SET	DBST
98	FIELD TRANSFER	FLTR
99	DYNAMIC FIELD TRANSFER	DFTR
9A	FIELD ISOLATE	ISOL
9B	DYNAMIC FIELD ISOLATE	DISO
9C	FIELD INSERT	INSR
9D	DYNAMIC FIELD INSERT	DINS
9E	BIT RESET	BRST
9F	DYNAMIC BIT RESET	DBRS
A0	BRANCH FALSE	BRFL
A1	BRANCH TRUE	BRTR
A2	BRANCH UNCONDITIONAL	BRUN
A3	EXIT	EXIT
A4	STEP AND BRANCH	STBR
A5	INDEX AND LOAD NAME	NXLN
A6	INDEX	INDX
A7	RETURN	RETN
A8	DYNAMIC BRANCH FALSE	DBFL

Hexa-Decimal Code	Name	Mnemonic
A9	DYNAMIC BRANCH TRUE	DBTR
AA	DYNAMIC BRANCH UNCONDITIONAL	DBUN
AB	ENTER	ENTR
AC	EVALUATE DESCRIPTOR	EVAL
AD	INDEX AND LOAD VALUE	NXLV
AE	MARK STACK	MKST
AF	STUFF ENVIRONMENT	STFF
B0	LIT CALL ZERO	ZERO
B1	LIT CALL ONE	ONE
B2	LIT CALL 8 BITS	LT8
B3	LIT CALL 16 BITS	LT16
B4	PUSH DOWN STACK REGISTERS	PUSH
B5	DELETE TOP OF STACK	DLET
B6	EXCHANGE	EXCH
B7	DUPLICATE TOP OF STACK	DUPL
B8	STORE DESTRUCTIVE	STOD
B9	STORE NON-DESTRUCTIVE	STON
BA	OVERWRITE DESTRUCTIVE	OVRD
BB	OVERWRITE NON-DESTRUCTIVE	OVRN
BD	LOAD	LOAD
BE	LIT CALL 48 BITS	LT48
BF	MAKE PROGRAM CONTROL WORD	MPCW
C0	SCALE LEFT	SCLF
C1	DYNAMIC SCALE LEFT	DSLF
C2	SCALE RIGHT TRUNCATE	SCRT
C3	DYNAMIC SCALE RIGHT RUNCATE	DSRT
C4	SCALE RIGHT SAVE	SCRS
C5	DYNAMIC SCALE RIGHT SAVE	DSRS
C6	SCALE RIGHT FINAL	SCRF
C7	DYNAMIC SCALE RIGHT FINAL	DSRF
C8	SCALE RIGHT ROUNDED	SCRR
C9	DYNAMIC SCALE RIGHT ROUND	DSRR
CA	INPUT CONVERT, DESTRUCTIVE	ICVD
CB	INPUT CONVERT, UPDATE	ICVU
CC	SET TO SINGLE-PRECISION, TRUNCATED	SNGT
CD	SET TO SINGLE-PRECISION, ROUNDED	SNGL
CE	SET TO DOUBLE-PRECISION	XTND
CF	INSERT MARK STACK	IMKS
D0	TABLE ENTER EDIT, DESTRUCTIVE	TEED
D1	PACK DESTRUCTIVE	PACD
D2	EXECUTE SINGLE MICRO, DESTRUCTIVE	EXSD
D3	TRANSFER WORDS, DESTRUCTIVE	TWSD
D4	TRANSFER WORDS OVERWRITE DESTRUCTIVE	TWOD
D5	STRING ISOLATE	SISO
D6	SET EXTERNAL SIGN	SXSN
D7	READ AND CLEAR OVERFLOW FLIP FLOP	ROFF
D8	TABLE ENTER EDIT, UPDATE	TEEU
D9	PACK UPDATE	PACU

Hexa-Decimal Code	Name	Mnemonic
DA	EXECUTE SINGLE MICRO, UPDATE	EXSU
DB	TRANSFER WORDS, UPDATE	TWSU
DC	TRANSFER WORDS OVERWRITE UPDATE	TWOU
DD	EXECUTE SINGLE MICRO, SINGLE POINTER UPDATE	EXPU
DE	READ TRUE/FALSE FLIP FLOP	TRFF
DF	CONDITIONAL HALT	HALT
E0	TRANSFER WHILE LESS, DESTRUCTIVE	TLSD
E1	TRANSFER WHILE GREATER OR EQUAL, DESTRUCTIVE	TGED
E2	TRANSFER WHILE GREATER, DESTRUCTIVE	TGTD
E3	TRANSFER WHILE LESS OR EQUAL, DESTRUCTIVE	TLED
E4	TRANSFER WHILE EQUAL, DESTRUCTIVE	TEQD
E5	TRANSFER WHILE NOT EQUAL, DESTRUCTIVE	TNED
E6	TRANSFER UNCONDITIONAL, DESTRUCTIVE	TUND
E7	VECTOR MODE ENTER SINGLE	VMES
E8	TRANSFER WHILE LESS, UPDATE	TLSU
E9	TRANSFER WHILE GREATER OR EQUAL, UPDATE	TGEU
EA	TRANSFER WHILE GREATER, UPDATE	TGTU
EB	TRANSFER WHILE LESS OR EQUAL, UPDATE	TLEU
EC	TRANSFER WHILE EQUAL, UPDATE	TEQU
ED	TRANSFER WHILE NOT EQUAL, UPDATE	TNEU
EE	TRANSFER UNCONDITIONAL, UPDATE	TUNU
EF	VECTOR MODE ENTER MULTIPLE	VMEM
F0	COMPARE CHARACTERS LESS, DESTRUCTIVE	CLSD
F1	COMPARE CHARACTERS GREATER OR EQUAL, DESTRUCTIVE	CGED
F2	COMPARE CHARACTERS GREATER, DESTRUCTIVE	CGTD
F3	COMPARE CHARACTERS LESS OR EQUAL, DESTRUCTIVE	CLED
F4	COMPARE CHARACTERS EQUAL, DESTRUCTIVE	CEQD
F5	COMPARE CHARACTERS NOT EQUAL, DESTRUCTIVE	CNED
F8	COMPARE CHARACTERS LESS, UPDATE	CLSU
F9	COMPARE CHARACTERS GREATER OR EQUAL, UPDATE	CGEU
FA	COMPARE CHARACTERS GREATER, UPDATE	CGTU
FB	COMPARE CHARACTERS LESS OR EQUAL, UPDATE	CLEU
FC	COMPARE CHARACTERS EQUAL, UPDATE	CEQU
FD	COMPARE CHARACTERS NOT EQUAL, UPDATE	CNEU
FE	NO OPERATION	NOOP
FF	INVALID OPERATOR	NVLD

VARIANT MODE

9542	SET TWO SINGLES TO DOUBLE	JOIN
9543	SET DOUBLE TO TWO SINGLES	SPLT
9544	IDLE UNTIL INTERRUPT	IDLE
9545	SET INTERVAL TIMER	SINT
9546	ENABLE EXTERNAL INTERRUPTS	EEXI

Hexa-Decimal Code	Name	Mnemonic
9547	DISABLE EXTERNAL INTERRUPTS	DEXI
954A	SCAN IN	SCNI
954B	SCAN OUT	SCNO
954E	READ PROCESSOR IDENTIFICATION	WHOI
9585	OCCURS INDEX	OCRX
9587	INTEGERIZE, ROUNDED, DOUBLE-PRECISION	NTGD
958B	LEADING ONE TEST	LOG2
958E	NORMALIZE	NORM
95AF	MOVE TO STACK	MVST
95B3	READ COMPARE FLIP FLOP	RCMP
95B4	SET TAG FIELD	STAG
95B5	READ TAG FIELD	RTAG
95B6	ROTATE STACK UP	RSUP
95B7	ROTATE STACK DOWN	RSDN
95B8	READ PROCESSOR REGISTER	RPRR
95B9	SET PROCESSOR REGISTER	SPRR
95BA	READ WITH LOCK	RDLK
95BB	COUNT BINARY ONES	CBON
95BC	LOAD TRANSPARENT	LODT
95BD	LINKED LIST LOOKUP	LLLU
95BE	MASKED SEARCH FOR EQUAL	SRCH
95D0	UNPACK SIGNED, DESTRUCTIVE	USND
95D1	UNPACK ABSOLUTE, DESTRUCTIVE	UABD
95D2	TRANSFER WHILE FALSE, DESTRUCTIVE	TWFD
95D3	TRANSFER WHILE TRUE, DESTRUCTIVE	TWTD
95D4	SCAN WHILE FALSE, DESTRUCTIVE	SWFD
95D5	SCAN WHILE TRUE, DESTRUCTIVE	SWTD
95D7	TRANSLATE	TRNS
95D8	UNPACK SIGNED, UPDATE	USNU
95D9	UNPACK ABSOLUTE, UPDATE	UABU
95DA	TRANSFER WHILE FALSE, UPDATE	TWFU
95DB	TRANSFER WHILE TRUE, UPDATE	TWTU
95DC	SCAN WHILE FALSE, UPDATE	SWFU
95DD	SCAN WHILE TRUE, UPDATE	SWTU
95DF	CONDITIONAL HALT	HALT
95F0	SCAN WHILE LESS, DESTRUCTIVE	SLSD
95F1	SCAN WHILE GREATER OR EQUAL, DESTRUCTIVE	SGED
95F2	SCAN WHILE GREATER, DESTRUCTIVE	SGTD
95F3	SCAN WHILE LESS OR EQUAL, DESTRUCTIVE	SLED
95F4	SCAN WHILE EQUAL, DESTRUCTIVE	SEQD
95F5	SCAN WHILE NOT EQUAL, DESTRUCTIVE	SNED
95F8	SCAN WHILE LESS, UPDATE	SLSU
95F9	SCAN WHILE GREATER OR EQUAL, UPDATE	SGEU
95FA	SCAN WHILE GREATER, UPDATE	SGTU
95FB	SCAN WHILE LESS OR EQUAL, UPDATE	SLEU
95FC	SCAN WHILE EQUAL, UPDATE	SEQU
95FD	SCAN WHILE NOT EQUAL, UPDATE	SNEU

Hexa-Decimal Code	Name	Mnemonic
95FE	NO OPERATION	NOOP
95FF	INVALID	NVLD
EDIT MODE		
D0	MOVE WITH INSERT	MINS
D1	MOVE WITH FLOAT	MFLT
D2	SKIP FORWARD SOURCE CHARACTERS	SFSC
D3	SKIP REVERSE SOURCE CHARACTERS	SRSC
D4	RESET FLOAT	RSTF
D5	END FLOAT	ENDF
D6	MOVE NUMERIC UNCONDITIONAL	MVNU
D7	MOVE CHARACTERS	MCHR
D8	INSERT OVERPUNCH	INOP
D9	INSERT DISPLAY SIGN	INSG
DA	SKIP FORWARD DESTINATION CHARACTERS	SFDC
DB	SKIP REVERSE DESTINATION CHARACTERS	SRDC
DC	INSERT UNCONDITIONAL	INSU
DD	INSERT CONDITIONAL	INSC
DE	END EDIT	ENDE
DF	CONDITIONAL HALT	HALT
FE	NO OPERATION	NOOP
FF	INVALID	NVLD
VECTOR MODE		
E0	LOAD A	LDA
E1	LOAD A INCREMENT	LDAI
E2	LOAD B	LDB
E3	LOAD B INCREMENT	LDBI
E4	LOAD C	LDC
E5	LOAD C INCREMENT	LDCI
E6	VECTOR MODE EXIT	VMEX
E8	DOUBLE LOAD A	DLA
E9	DOUBLE LOAD A INCREMENT	DLAI
EA	DOUBLE LOAD B	DLB
EB	DOUBLE LOAD B INCREMENT	DLBI
EC	DOUBLE LOAD C	DLC
ED	DOUBLE LOAD C INCREMENT	DLCI
EE	VECTOR BRANCH	VEBR
F0	STORE A	STA
F1	STORE A INCREMENT	STAI
F2	STORE B	STB
F3	STORE B INCREMENT	STBI
F4	STORE C	STC
F5	STORE C INCREMENT	STCI
F8	DOUBLE STORE A	DSA
F9	DOUBLE STORE A INCREMENT	DSAI

Hexa-Decimal Code	Name	Mnemonic
FA	DOUBLE STORE B	DSB
FB	DOUBLE STORE B INCREMENT	DSBI
FC	DOUBLE STORE C	DSC
FD	DOUBLE STORE C INCREMENT	DSCI

APPENDIX C

DATA REPRESENTATION

EBCDIC Graphic	BCL	Decimal Value	EBCDIC Internal	Hex. Graphic	EBCDIC Card Code	BCL Card Code	Octal	BCL Internal	BCL External
BLANK		64	0100 0000	40	No Punches	No Punches	60	11 0000	01 0000
[		74	0100 1010	4A	12 8 2	12 8 4	33	01 1011	11 1100
.		75	0100 1011	4B	12 8 3	12 8 3	32	01 1010	11 1011
<		76	0100 1100	4C	12 8 4	12 8 6	36	01 1110	11 1110
(		77	0100 1101	4D	12 8 5	12 8 5	35	01 1101	11 1101
+		78	0100 1110	4E	12 8 6				11 1010
|	←	79	0100 1111	4F	12 8 7	12 8 7	37	01 1111	11 1111
&		80	0101 0000	50	12	12	34	01 1100	11 0000
]		90	0101 1010	5A	11 8 2	0 8 6	76	11 1110	01 1110
$		91	0101 1011	5B	11 8 3	11 8 3	52	10 1010	10 1011
*		92	0101 1100	5C	11 8 4	11 8 4	53	10 1011	10 1100
)		93	0101 1101	5D	11 8 5	11 8 5	55	10 1101	10 1101
;		94	0101 1110	5E	11 8 6	11 8 6	56	10 1110	10 1110
	≤	95	0101 1111	5F	11 8 7	11 8 7	57	10 1111	10 1111
-		96	0110 0000	60	11	11	54	10 1100	10 0000
/		97	0110 0001	61	0 1	0 1	61	11 0001	01 0001
,		107	0110 1011	6B	0 8 3	0 8 3	72	11 1010	01 1011
%		108	0110 1100	6C	0 8 4	0 8 4	73	11 1011	01 1100
–	≠	109	0110 1101	6D	0 8 5	0 8 2	74	11 1100	01 1010
>		110	0110 1110	6E	0 8 6	8 6	16	00 1110	00 1110
?		111	0110 1111	6F	0 8 7	*	14	00 1100	00 0000
:		122	0111 1010	7A	8 2	8 5	15	00 1101	00 1101
#		123	0111 1011	7B	8 3	8 3	12	00 1010	00 1011
@		124	0111 1100	7C	8 4	8 4	13	00 1011	00 1100
'	≥	125	0111 1101	7D	8 5	8 7	17	00 1111	00 1111
=		126	0111 1110	7E	8 6	0 8 5	75	11 1101	01 1101
"		127	0111 1111	7F	8 7	0 8 7	77	11 1111	01 1111
(+)PZ	+	192	1100 0000	C0	12 0	12 0	20	01 0000	11 1010
A		193	1100 0001	C1	12 1	12 1	21	01 0001	11 0001
B		194	1100 0010	C2	12 2	12 2	22	01 0010	11 0010
C		195	1100 0011	C3	12 3	12 3	23	01 0011	11 0011
D		196	1100 0100	C4	12 4	12 4	24	01 0100	11 0100
E		197	1100 0101	C5	12 5	12 5	25	01 0101	11 0101
F		198	1100 0110	C6	12 6	12 6	26	01 0110	11 0110
G		199	1100 0111	C7	12 7	12 7	27	01 0111	11 0111
H		200	1100 1000	C8	12 8	12 8	30	01 1000	11 1000
I		201	1100 1001	C9	12 9	12 9	31	01 1001	11 1001
	MULT								
(!)MZ	x	208	1101 0000	D0	11 0	11 0	40	10 0000	10 1010
J		209	1101 0001	D1	11 1	11 1	41	10 0001	10 0001

*All other codes

EBCDIC Graphic	BCL	Decimal Value	EBCDIC Internal	Hex. Graphic	EBCDIC Card Code	BCL Card Code	Octal	BCL Internal	BCL External
K		210	1101 0010	D2	11 2	11 2	42	10 0010	10 0010
L		211	1101 0011	D3	11 3	11 3	43	10 0011	10 0011
M		212	1101 0100	D4	11 4	11 4	44	10 0100	10 0100
N		213	1101 0101	D5	11 5	11 5	45	10 0101	10 0101
O		214	1101 0110	D6	11 6	11 6	46	10 0110	10 0110
P		215	1101 0111	D7	11 7	11 7	47	10 0111	10 0111
Q		216	1101 1000	D8	11 8	11 8	50	10 1000	10 1000
R		217	1101 1001	D9	11 9	11 9	51	10 1001	10 1001
¢		224	1110 0000	E0	0 8 2				00 0000
S		226	1110 0010	E2	0 2	0 2	62	11 0010	01 0010
T		227	1110 0011	E3	0 3	0 3	63	11 0011	01 0011
U		228	1110 0100	E4	0 4	0 4	64	11 0100	01 0100
V		229	1110 0101	E5	0 5	0 5	65	11 0101	01 0101
W		230	1110 0110	E6	0 6	0 6	66	11 0110	01 0110
X		231	1110 0111	E7	0 7	0 7	67	11 0111	01 0111
Y		232	1110 1000	E8	0 8	0 8	70	11 1000	01 1000
Z		233	1110 1001	E9	0 9	0 9	71	11 1001	01 1001
0		240	1111 0000	F0	0	0	00	00 0000	00 1010
1		241	1111 0001	F1	1	1	01	00 0001	00 0001
2		242	1111 0010	F2	2	2	02	00 0010	00 0010
3		243	1111 0011	F3	3	3	03	00 0011	00 0011
4		244	1111 0100	F4	4	4	04	00 0100	00 0100
5		245	1111 0101	F5	5	5	05	00 0101	00 0101
6		246	1111 0110	F6	6	6	06	00 0110	00 0110
7		247	1111 0111	F7	7	7	07	00 0111	00 0111
8		248	1111 1000	F8	8	8	10	00 1000	00 1000
9		249	1111 1001	F9	9	9	11	00 1001	00 1001

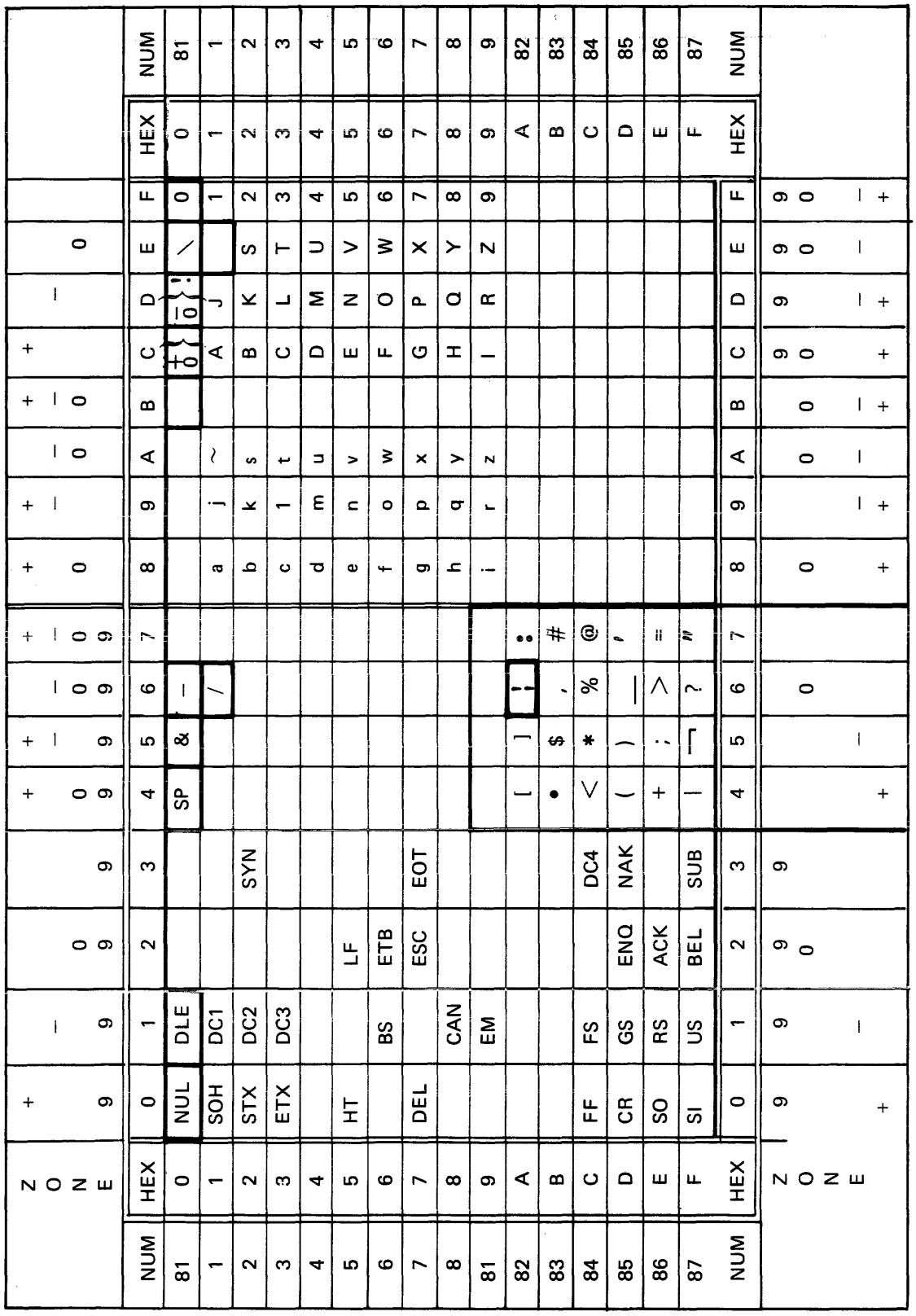
NOTES

1. EBCDIC 0100 1110 also translates to BCL 11 1010.
2. EBCDIC 1100 1111 is translated to BCL 00 0000 with an additional flag bit on the most significant bit line (8th bit). This function is used by the unbuffered printer to stop scanning.
3. EBCDIC 1110 0000 is translated to BCL 00 0000 with an additional flag bit on the next to most significant bit line (7th bit). As the print drums have 64 graphics and space this signal can be used to print the 64th graphic. The 64th graphic is a "CR" for BCL drums and a "¢" for EBCDIC drums.
4. The remaining 189 EBCDIC codes are translated to BCL 00 0000 (? code).
5. The EBCDIC graphics and BCL graphics are the same except as follows:

BCL		EBCDIC	
≥		'	(single quote)
x	(multiply)	!	
≤		¬	(not)
≠		¯	(underscore)
←			

APPENDIX D

B 6800 EBCDIC/HEX CARD CODE



ZONE	+ 9	– 9	+ – 0 9	+ – 0 9 9	+ – 0 0 9 9	+ 0 0	+ – 0	+ – 0 0	+ –	+	–	+ 0	NUM
HEX	0	1	2	3	4	5	6	7	8	9	A	B	81
0	NUL	DLE			SP	&	–						1
1	SOH	DC1					/			j	~		2
2	STX	DC2		SYN						k	s		3
3	ETX	DC3								l	t		4
4										m	u		5
5	HT		LF							n	v		6
6		BS	ETB							o	w		7
7	DEL		ESC	EOT						p	x		8
8		CAN								q	y		9
9		EM								r	z		82
A					[	]	¢	:					83
B					.	$	,	#					84
C	FF	FS		DC4	<	*	%	@					85
D	CR	GS	ENQ	NAK	(	)	_	'					86
E	SO	RS	ACK		+	;	>	=					87
F	SI	US	BEL	SUB	|	¬	?	"					NUM
ZONE	9	9	9 0	9			0						
	+	–			+	–							
HEX	0	1	2	3	4	5	6	7	8	9	A	B	HEX
NUM	81	1	2	3	4	5	6	7	8	9	82	83	NUM

Use of the B 6800 EBCDIC/HEX Card Code Chart.

a. Locate the desired EBCDIC graphic code within the table.

b. The two-part Hexadecimal Code is read as follows:

1. The first part is found in the vertical column above or below the desired EBCDIC code.

2. The second part is found in the horizontal row either to the right or left of the desired EBCDIC code.

(a) Examples:

SYN = 32
F = C6

c. The two-part Card Code is found in the same manner as HEX (2) except the zone and numeric bits are read from the very outer portion of the table.

1. Examples:

SYN = 9 2
F = + 6

2. The card code exceptions to the above procedures are enclosed in heavy lines on the chart and are defined below:

(1) 00 = + 0981 (NUL)
(2) 10 = + -981 (DLE)
(3) 20 = -0981
(4) 30 = + -0981
(5) 40 = BLANK
(6) 50 = + (&)
(7) 60 = - (-)
(8) 70 = + -0
(9) CO = + 0 ({) ($\overset{+}{o}$)
(10) DO = - 0 (}) ($\bar{o}$)
(11) EO = 0 82 (\)
(12) FO = 0 (0)
(13) 61 = 0 1 (/)
(14) E1 = -09 1
(15) 6A = + - (¦)

INDEX

Absolute Address Conversion . . . 2-22, 3-13
Address Couple . . . 2-22
Adapter Cluster . . . 12-1
Add . . . 7-1
Adder, Address . . . 5-7
Adder, Exponent . . . 5-10
Adder, Mantissa . . . 5-10
Adder, Residue Interrupt . . . 5-29
Address Adder . . . 5-7
Address Environment Defined . . . 3-15
Address Retry Interrupt . . . 5-27
Alarm Interrupts . . . 5-22
American Standard Code for Information Interchange . . . 2-3
Area Descriptor . . . 5-54, 5-56, 11-1
A Register . . . 4-10
ASCII . . . 2-3
Arithmetic Controller . . . 5-10
Arithmetic Operators . . . 7-1
AROF . . . 3-3, 4-4

Base and Limit of Stack . . . 3-2
Base of Address Level Segment . . . 3-13
BCL . . . 2-3
BIC . . . 1-30, 13-1
BIC Memory Bus Interface . . . 13-6
BIC Module . . . 13-1
BIC Scan Bus Interface . . . 13-1
BIC Scan-In Function . . . 13-4
BIC Scan-Out Function . . . 13-1
Bit Operators . . . 7-13
Bit Reset . . . 7-14
Bit Reset Dynamic . . . 7-14
Bit Set . . . 7-13
Bit Set Dynamic . . . 7-13
Bit Sign Change . . . 7-14
Bottom of Stack Interrupt . . . 5-17
Branch False . . . 7-7
Branch False Dynamic . . . 7-8
Branch Operators . . . 7-7
Branch True . . . 7-8
Branch True Dynamic . . . 7-8
Branch Unconditional . . . 7-8
Branch Unconditional Dynamic . . . 7-8
BROF . . . 3-3, 4-4
Brownout . . . 1-17
Buffer, Data (MPX) . . . 11-1
Burroughs Common Language . . . 2-3
Bus Interface Control (BIC) . . . 1-31, 13-1
Bus Interface Control Memory Bus . . . 13-6
Bus Interface Control Module . . . 1-31, 13-1

INDEX (Cont)

Bus Interface Control Scan Bus 13-1
Bus Interface Control Scan-In Function 13-4
Bus Interface Control Scan-Out Function 13-1
Bus Residue Interrupts 5-29

Central Power Cabinet 1-17
Central Processor Unit Cabinet 1-7
Change Sign 7-14
Channel A 5-64
Channel B 5-72
Character Codes, Internal 2-3
Character Translator 5-53
Character Type Data 2-3, 2-10
Clock Controls 4-52
Clocks 1-8
Compare Characters Equal Destructive 7-20
Compare Characters Equal Update 7-20
Compare Characters Greater, Destructive 7-18
Compare Characters Greater or Equal, Destructive 7-19
Compare Characters Greater or Equal Update 7-19
Compare Characters Greater, Update 7-19
Compare Characters Less Destructive 7-20
Compare Characters Less or Equal Destructive 7-20
Compare Characters Less or Equal Update 7-20
Compare Characters Less Update 7-20
Compare Characters Not Equal Destructive 7-20
Compare Characters Not Equal Update 7-20
Compare Operators 7-18
Compare Residue Interrupt 5-29
Conditional Halt 7-9
Confidence Error Interrupt 5-21
Controller, Arithmetic 5-10
Controller, Interrupt 5-10
Controller, Memory 5-30
Controller, Program 5-2
Controller, Stack 5-10
Controller, String Operator 5-30
Controller, Transfer 5-7
Control State/Normal State 5-31
Copy Bit 3-6
Count Binary Ones 8-10
C Register 4-10
CTIO 1-17

Data Addressing 3-5
Data Buffer (MPX) 5-60
Data Comm Adapter Cluster II 12-1
Data Comm Adapter Cluster III 12-5
Data Comm Basic Control 12-4
Data Comm Broadband Control 12-4
Data-Comm-To-Disk Control 12-4

INDEX (Cont)

Data Comm Store-To-Store Control . . . 12-5
Data Comm Memory Bus Interface . . . 12-7
Data Comm Scan Bus Interface . . . 12-5
Data Communications Adapters . . . 12-1
Data Communications Interface . . . 12-1
Data Communications Interrupt . . . 5-22
Data Communications Processor . . . 12-3
Data Communications Subsystem . . . 1-30, 12-1
Data-Dependent Presence Bit . . . 5-18
Data Descriptor . . . 2-15
Data Field Convention . . . 2-2
Data Processor . . . 1-8, 5-1
Data Processor, Scan-In Functions to Multiplexor . . . 5-31
Data Processor Scan-Out Functions to Multiplexor . . . 5-33
Data Processor Scan-Out Functions to External Subsystems . . . 5-35
Data Representation . . . 2-1
Data Types and Physical Layout . . . 2-1
Decimal to Coded Number Conversion . . . 2-6
Decimal and Hexadecimal Table Conversion . . . 2-8
Delete Top of Stack . . . 7-10
Descriptor Formats, IO . . . 5-61
Disable External Interrupts . . . 8-2
Display Panels . . . 1-12, 4-1
Display Registers . . . 4-1
Divide . . . 7-2
Divide by Zero Interrupt . . . 5-15
Double Load A . . . 10-7
Double Load A Increment . . . 10-7
Double Load B . . . 10-7
Double Load B Increment . . . 10-7
Double Load C . . . 10-7
Double Load C, Increment . . . 10-7
Double Store A . . . 10-7
Double Store A Increment . . . 10-7
Double Store B . . . 10-7
Double Store B Increment . . . 10-7
Double Store C . . . 10-7
Double Store C Increment . . . 10-7
Double Precision Operands . . . 2-13
Double Precision Stack OP . . . 3-3
Duplicate Top of Stack . . . 7-10
Dynamic Branch False . . . 7-8
Dynamic Branch True . . . 7-8
Dynamic Branch Unconditional . . . 7-8

EBCDIC . . . 2-3
Edit Mode Operation . . . 9-1
Edit Mode Operators . . . 9-1
Enable External Interrupts . . . 8-2
End Edit . . . 9-4
End Float . . . 9-3

INDEX (Cont)

Enter Operators 7-20, 7-26, 7-31
Enter Vector Mode 7-31
Equal 7-7
Escape to 16-bit Instruction 8-1
Evaluate 7-26
Exchange 7-10
Execute Single Micro Desctructive 7-21
Execute Single Micro Single Pointer Update 7-21
Execute Single Micro Update 7-21
Executing I/O Descriptors 11-1
Exit Operator 7-26
Exponent Adder 5-10
Exponent Overflow and Underflow Interrupt 5-16
Extended Binary Coded Decimal Interchange Code 2-3
External Interrupts 5-21

Family A 5-1
Family B 5-1
Family C 5-1
Family D 5-1
Family E 5-2
Family U (F, G, H) 5-2
Field Insert 7-15
Field Insert Dynamic 7-15
Field Isolate 7-15
Field Isolate Dynamic 7-15
Field Transfer 7-14
Field Transfer Dynamic 7-14
Function Word 5-33

General Control Interrupt 5-26
Global Memory 1-5, 5-71
Global Memory Not Ready Interrupt 5-25
Greater Than 7-6
Greater Than or Equal 7-7

Hardware Interrupts 5-28
Hexadecimal and Octal Notation 2-3
Hexadecimal to Decimal Table Conversion 2-8

Idle Confidence Testing 1-9
Idle Until Interrupt 8-1
Index 7-11
Index and Load Name 7-11
Index and Load Operators 7-11
Index and Load Value 7-11
Index Bit 3-5
Index, Invalid 3-5
Index, Valid 3-5
Indirect Reference Word 2-20
Initiate I/O 5-54, 8-4, 11-1

INDEX (Cont)

Initiate I/O Word Format 5-56
Input Convert Destructive 7-22
Input Convert Operators 7-22
Input Convert Update 7-23
Input/Output Area Description (IOAD) Word Format 5-56
Input/Output Control Word (IOCW) Format 5-56
Input/Output Device Numbering 5-53
Input/Output Operations 5-53, 11-1
Input/Output Processor (Multiplexor) 5-31
Input/Output Processor Interrupts 5-22
Insert Conditional 9-3
Insert Display Sign 9-3
Insert Mark Stack Operator 7-31
Insert Overpunch 9-3
Insert Unconditional 9-3
Integer Divide 7-3
Integerized Rounded, D.P. 8-5
Integerize Rounded 7-4
Intergerize Truncated 7-3
Interger Overflow Interrupt 5-17
Integrated Circuit (IC) Memory 5-5
Internal Character Codes 2-3
Internal Data Transfer Section 5-8
Interrogate I/O Path, Function 5-35, 5-39, 5-45, 8-3
Interrogate Peripheral Status 8-3
Interrogate Peripheral Unit Type 5-39, 8-3
Interrupt Controller 5-10
Interrupt Handling 2-26
Interrupt Parameters 2-26, 5-10
Interrupt System 2-26
Interrupts, Alarm 5-22
Interrupts, External 5-21
Interrupts, Operator Dependent 5-14
Interrupts, Operator Independent 5-21
Interval Timer Interrupt 5-20
Invalid Address Interrupt 5-24
Invalid Address Residue Interrupt 5-24
Invalid Address Local Interrupt 5-24
Invalid Address Global Interrupt 5-25
Invalid Index Interrupt 3-5, 5-16
Invalid Operand Interrupt 5-15
Invalid Operator 7-9
Invalid Program Word Interrupt 5-24
I/O Control Word 5-56, 11-1, 11-11, 11-13, 11-15, 11-17, 11-19, 11-21, 11-23, 11-26, 11-28, 11-20, 11-33
I/O Finish Interrupts 11-3
I/O Operations, Processor Initiated 5-49, 11-1
I/O Processor Parity (MPX Parity) 5-28

INDEX (Cont)

Job-Splitting 3-17

Keyboard Control Keys 1-29, 4-53
Keyboard, Maintenance Processors 4-49
KSI 1-16

Leading One Test 8-5
Less Than 7-7
Less Than or Equal 7-7
Level Definition 2-22, 3-17
Lexacographical Level 2-22
Linked List Lookup 8-10
Lit Call Zero 7-10
Lit Call One 7-10
Lit Call 8 Bits 7-10
Lit Call 16 Bits 7-10
Lit Call 48 Bits 7-10
Literal Call Operators 7-10
Load 7-11
Load A 10-6
Load A Increment 10-6
Load B 10-6
Load B Increment 10-6
Load C 10-6
Load C Increment 10-6
Load Transparent 8-10
Local Memory Allocation 3-12
Local Memory Interface 5-74
Logical And 7-5
Logical Equal 7-7
Logical Equivalence 7-5
Logical Negate 7-5
Logical Operands 2-15
Logical Operators 7-5
Logical Or 7-5
Logic Card Testing 1-17
Look Ahead Logic 5-5
Look Ahead Register 4-10
LROF 4-8
Loop Interrupt 5-23

Maintenance Display Panel 4-10
Maintenance Display Processor 1-12
Maintenance Display Registers 4-11
Maintenance Processor 1-15, 4-49
Make PCW 7-11
Mantissa Field 2-12
Mark Stack Control Word 2-25

INDEX (Cont)

Mark Stack Control Word Linkage . . . 3-15
Mark Stack Operator . . . 7-26
Mask and Steering . . . 5-8
Mask and Steering Example . . . 5-8
Masked Search for Equal . . . 8-11
Master Control Program . . . 1-1
Memory Address . . . 5-63
Memory Address Interrupt . . . 5-23
Memory and Multiplexor Controller . . . 11-1
Memory Area Allocation . . . 3-12
Memory Bus . . . 5-64, 5-74
Memory Cabinet Configuration . . . 1-21
Memory Control . . . 1-10
Memory Controller . . . 5-30
Memory Cycle Times . . . 1-11
Memory Error Detection/Correction . . . 1-11, 5-69
Memory Interface . . . 5-64
Memory Module . . . 1-10
Memory Organization . . . 5-63
Memory Parity Interrupt . . . 5-23
Memory Port Interface . . . 5-72
Memory Priority . . . 5-72
Memory Protect Interrupt . . . 5-15
Memory Protection . . . 5-15
Memory Retry . . . 1-11, 5-71
Memory Stack Controller . . . 5-10
Memory Tester . . . 5-76
Memory Testing . . . 5-76
Memory Words . . . 5-63
MFIO Bus . . . 1-5, 1-16
Micro Processor . . . 1-16
Module Definition . . . 1-5
Move Characters . . . 9-1
Move Numeric Unconditional . . . 9-1
Move to Stack . . . 8-6
Move With Float . . . 9-2
Move With Insert . . . 9-1
Multiple Stacks and Re-Entrant Code . . . 3-17
Multiple Variables (Common Address Couples) . . . 3-13
Multiplexor Function . . . 5-31
Multiplexor Scan-In Function . . . 5-35
Multiplexor Scan-Out Function . . . 5-45
Multiply . . . 7-2
Multiply (Extended) . . . 7-2

Name Call . . . 6-2, 7-26
No Operation . . . 7-9
Normalize . . . 8-6

INDEX (Cont)

Normal State . . . 5-31
Not Equal . . . 7-7
Number Bases . . . 2-3
Number Conversion . . . 2-6

Occurs Index . . . 8-5
Octal Notation . . . 2-3
OP Code and Variant Characters . . . 5-61
Operands . . . 2-10
Operation Types . . . 6-2
Operators Control Console . . . 1-25
Operator Dependent Interrupts . . . 5-14
Operator Families . . . 5-1
Operator Independent Interrupts . . . 5-21
Operator Panel . . . 1-26
Operators . . . 6-7
Overflow FF, Read and Clear . . . 7-23
Overwrite Destructive . . . 7-9
Overwrite Non-Destructive . . . 7-10

Pack Destructive . . . 7-21
Pack Operators . . . 7-21
Pack Update . . . 7-22
PCIO Bus . . . 1-5, 1-15, 1-17
Peripheral Control Bus . . . 11-7
Peripheral Control Cabinet . . . 11-7
Peripheral Control Interface . . . 1-18, 11-7
Peripheral Controls . . . 1-10, 1-18, 11-7
Peripheral Result Descriptor . . . 11-9, 11-12, 11-14, 11-16, 11-18, 11-20, 11-22, 11-25, 11-27, 11-29, 11-32, 11-34
Peripheral Result Descriptor, Supervisory Display II . . . 11-12
Peripheral Result Descriptor, Single Line Control . . . 11-14
Peripheral Result Descriptor, Card Punch . . . 11-16
Peripheral Result Descriptor, Card Reader . . . 11-18
Peripheral Result Descriptor, Line Printer . . . 11-20
Peripheral Result Descriptor, Train Printer . . . 11-22
Peripheral Result Descriptor, Magnetic Tape . . . 11-25
Peripheral Result Descriptor, Head-Per-Track Disk . . . 11-27
Peripheral Result Descriptor, Flexable Disk . . . 11-29
Peripheral Result Descriptor, Disk Pack . . . 11-32
Peripheral Result Descriptor, 5N Disk . . . 11-34
Peripheral Units . . . 1-10
Polish Notation . . . 3-6
Polish String . . . 3-8
Polish String, Rules for Evaluating . . . 3-8
Polish String, Rules for Generating . . . 3-6
Power Busses . . . 1-7
Power Cabinet . . . 1-17
Power Controls . . . 4-1

INDEX (Cont)

Power Off Switch 4-4
Power On Switch 4-4
Power, System 1-17
P Register 6-1
P1 Parameter 2-27, 11-3
P2 Parameter 2-28, 11-5
P3 Parameter 2-28, 11-4
Presence Bit 3-5
Presence Bit Interrupt 3-5, 3-18, 5-17
Primary Mode Operators 7-1
Procedure-Dependent Presence Bit 5-18
Processor 1-7, 5-1
Processor Initiated I/O Operations 11-1
Processor States 5-31
Processor System Concept 5-1
PROF 4-4
Program Control 6-1
Program Controller 5-2
Program Control Word 2-23
Program Index Register 2-24
Programmed Operator 5-2
Programmers Display Panel 4-1
Program Operators 6-1
Program (P) Register 4-10
Program Structure in Memory 3-11
Program Segment 3-12
Program Syllable Register 2-24, 4-4
Program Words 2-34
PROM Card Parity Interrupts 5-28
PROM Writer 1-16
PWIO 1-16
Push Down Stack Registers 3-2, 7-10

Ram Card Parity Interrupts 5-28
Read and Clear Overflow FF 7-23
Read Compare Flip-Flop 8-7
Read Data Check Bit Interrupt 5-27
Read Data Multiple Interrupt 5-25
Read Data Retry Interrupt 5-27
Read Data Single Error, Interrupt 5-26
Read IC Operation 4-54
Read Interrupt Literal 5-45, 8-3
Read Interrupt Mask 5-45, 8-2
Read Interrupt Register 5-38, 8-2
Read Main Memory 4-54
Read Only Bit 3-6
Read Processor Identification 8-5
Read Processor Register 8-8
Read Processor Time Counter 5-42, 8-3

INDEX (Cont)

Read Scratch Pad Word 5-43, 8-3
Read Tag Field 8-8
Read Time of Day Clock 5-37, 8-2
Read True False FF 7-23
Read With Lock 8-10
Ready Status 5-50
Re-Entrace 3-17
Register, A 4-10
Register, B 4-10
Register, C 4-10
Register, P 4-10, 6-1
Register, X 4-10
Register, Y 4-10
Register, Z 4-10
Registers, Display 4-10
Registers, Maintenance 4-11
Relational Operators 7-5
Relative-Addressing 3-12
Remainder Divide 7-3
Reset Float 9-3
Residue Adder Testing 5-7
Residue Testing 5-7
Result Descriptor 11-9
Result Descriptor, Card Punch Device 11-16
Result Descriptor, Card Reader Device 11-18
Result Descriptor, Disk-Flexable Device 11-29
Result Descriptor, Disk-Head Per Track Device 11-27
Result Descriptor, Disk-Pack Device 11-32
Result Descriptor, Disk-5N Device 11-34
Result Descriptor, Magnetic Tape Device 11-25
Result Descriptor, Printer-Line Device 11-20
Result Descriptor, Printer-Train Device 11-22
Result Descriptor, Single Line Control Devices 11-14
Result Descriptor, Supervisory Display II Device 11-12
Return Control Word 2-32
Return Operator 7-26
Reverse Polish Notation 3-6
Rotate Stack Down 8-8
Rotate Stack Up 8-8
Rules for Generating Polish String, Simplified 3-6
Running Indicator 4-1

Scale Left 7-12
Scale Left Dynamic 7-12
Scale Operators 7-12
Scale Right Dynamic Final 7-13
Scale Right Dynamic Save 7-12
Scale Right Dynamic Truncate 7-13
Scale Right Final 7-13

INDEX (Cont)

Scale Right Round Dynamic . . . 7-13
Scale Right Rounded . . . 7-13
Scale Right Save . . . 7-12
Scale Right Truncate . . . 7-12
Scan Bus . . . 1-5, 12-5, 13-1
Scan Bus Operations . . . 5-71
Scan Bus Interface . . . 5-75
Scan Bus Parity Error Interrupt . . . 5-23
Scan In . . . 8-2
Scan In Information Error Interrupt . . . 5-25
Scan Operators . . . 8-2
Scan Out . . . 8-3
Scan Out Error Interrupt . . . 5-26
Scan While Equal, Destructive . . . 8-14
Scan While Equal, Update . . . 8-14
Scan While False, Destructive . . . 8-15
Scan While False, Update . . . 8-15
Scan While Greater, Destructive . . . 8-13
Scan While Greater, Update . . . 8-13
Scan While Greater or Equal, Destructive . . . 8-13
Scan While Greater or Equal, Update . . . 8-13
Scan While Less, Destructive . . . 8-14
Scan While Less or Equal, Desctructive . . . 8-14
Scan While Less or Equal, Update . . . 8-14
Scan While Less, Update . . . 8-14
Scan While Not Equal, Destructive . . . 8-14
Scan While Not Equal, Update . . . 8-15
Scan While True, Destructive . . . 8-15
Scan While True, Update . . . 8-15
Scratchpad Memory . . . 5-58
Scratchpad Memory Channel . . . 5-58
Scratchpad Word Layout . . . 5-59
Segmented Array, Interrupt . . . 5-19
Segment Descriptor . . . 2-36
Segment Dictionary . . . 3-12
Sequence Error Interrupt . . . 5-18
Set Double to Two Singles . . . 8-1
Set External Sign . . . 7-23
Set Interrupt Mask . . . 5-46, 8-4
Set Interval Timer . . . 8-1
Set Processor Register . . . 8-9
Set PSUDO BUSY . . . 5-46, 8-4
Set Tag Field . . . 8-7
Set Time of Day Clock . . . 5-46, 8-4
Set to Double-Precision . . . 7-5
Set to Single Precision Rounded . . . 7-4
Set to Single-Precision Truncated . . . 7-4
Set Two Singles to Double . . . 8-1
Single Precision Operands . . . 2-12

INDEX (Cont)

Skip Forward Destination Characters . . . 9-2
Skip Forward Source Characters . . . 9-2
Skip Reverse Destination Characters . . . 9-3
Skip Reverse Source Characters . . . 9-2
Software Aspects of IO Operations . . . 5-46
Software Words . . . 2-19
Stack . . . 3-1
Stack Adjustment . . . 3-3
Stack Area . . . 3-12
Stack, Base and Limit . . . 3-2
Stack, Bi-Directional Data Flow . . . 3-2
Stack Controller . . . 5-10
Stack Deletion . . . 3-12
Stack Descriptor . . . 3-17
Stack, Double-Precision Operation . . . 3-3
Stack-History and Addressing-Environment Lists . . . 3-12
Stack History, Summary . . . 3-17
Stack Operation . . . 3-1
Stack Operators . . . 7-10
Stack Overflow Interrupt . . . 5-21
Stack Pushdown . . . 3-2
Stack Pushup . . . 3-2
Stack Registers . . . 5-7
Stack, Simple Operation . . . 3-8
Stack Underflow Interrupt . . . 5-24
Stack Vector Descriptor . . . 3-18
States, Processor . . . 5-31
Status Change . . . 5-52
Status Change Interrupt . . . 5-22
Status Display Panel . . . 4-4
Status Vectors . . . 5-62
Step and Branch . . . 7-8
Step Index Word . . . 2-18
Store A . . . 10-6
Store A Increment . . . 10-6
Store B . . . 10-6
Store B Increment . . . 10-7
Store C . . . 10-6
Store C, Increment . . . 10-7
Store Destructive . . . 7-9
Store, Non-Destructive . . . 7-9
Store Operators . . . 7-9
String Descriptor . . . 2-17
String Isolate . . . 7-18
String Operators . . . 5-30, 6-8
String Operator Controller . . . 5-30
String Transfer Operators . . . 7-16
Stuff Environment . . . 7-31
Stuffed Indirect Reference Word . . . 2-20

INDEX (Cont)

Subroutine Operators 7-23
Subtract 7-2
Syllable Addressing 2-37, 6-1
Syllable Dependent Interrupts 5-14
Syllable Format 2-37, 6-1
Syllable Identification 2-37, 6-1
System Clock 1-8
System Concept 5-1
System Controls 1-26, 4-1
System Description 1-1
System Expansion 1-1
System Memory Interface 5-72
System Options and Requirements 1-1
System Organization 1-1
System Power 1-17

Table Enter Edit Destructive 7-20
Table Enter Edit Update 7-21
TD 830 1-26
Terminal Device 12-4
Thumbwheel 4-11
Time of Day Register 4-48
Top-Of-Stack Register 3-1, 3-3
Transfer Controller 5-7
Transfer Operators 7-14
Transfer Unconditional Destructive 7-18
Transfer Unconditional, Update 7-18
Transfer While Equal, Destructive 7-17
Transfer While Equal, Update 7-17
Transfer While False, Destructive 8-12
Transfer While False, Update 8-12
Transfer While Greater Destructive 7-16
Transfer While Greater or Equal, Update 7-17
Transfer While Greater Update 7-17
Transfer While Less, Destructive 7-17
Transfer While Less, Update 7-17
Transfer While Less or Equal, Destructive 7-17
Transfer While Less or Equal, Update 7-17
Transfer While Not Equal, Destructive 7-17
Transfer While Not Equal, Update 7-18
Transfer While True, Destructive 8-12
Transfer While True, Update 8-12
Transfer Words Destructive 7-16
Transfer Words, Overwrite Destructive 7-16
Transfer Words, Overwrite Update 7-16
Transfer Words, Update 7-16
Translate 8-12
T Register 6-1
True False FF, Read 7-23
Type Transfer Operators 7-4

INDEX (Cont)

Unit Number 1-21
Unit of Memory 1-21
Unit Tables 1-21
Universal Operators 7-9
Unpack Absolute Destructive 8-11
Unpack Absolute Update 8-11
Unpack Signed Destructive 8-11

Valid Index 3-5
Value Bit 2-26
Value Call 6-4, 7-23
Variant Mode Operation and Operators 4-4, 8-1
Vector Mode Branch 10-8
Vector Mode Exit 10-8
Vector Mode Hardware Functions 10-1
Vector Mode Limitations 10-1
Vector Mode Enter Multiple 7-31
Vector Mode Enter Single 7-31
Vector Mode Operator Codes 10-1, 10-5, 10-6

Word Data Descriptor 2-15
Word Definition 2-1
Word Parity 2-1
Word Tag Field 2-1
Word Wraparound 2-3
Word Data Formats 2-1
Wrap Around 2-3
Write IC Operation 4-54
Write Main Memory 4-54

X Register 3-1, 4-10, 5-7

Y Register 3-1, 4-10, 5-7

Z Register 4-10, 5-7

Printed in U.S. America

July 1977

5001290