Conversion Specifications for 6809 to 68000

Bruce North

07/17/84

PROPRIETARY

Following is a description for conversion from 6809 assembly language to 68000 assembly language. Special care should be taken with the concatenation of ACCA + ACCB to form ACCD. It is possible that some sequences will not operate properly after conversion.

Register definition

6809	68000	
CC	CC	(L.S. BYTE)
DP	--	
A	D0.B	(L.S. BYTE)
B	D1.B	(L.S. BYTE)
D	D0.W	(L.S. WORD)
X	A0.W	(L.S. WORD)
Y	A1.W	(L.S. WORD)
U	A6.W	(L.S. WORD)
S	A7.W	(L.S. WORD)
PC	PC	

Specifications and assumptions

1. Word operations done on byte values are assumed to have all 0's in the unused portion of the register.

2. Register D7 is reserved for conversion use only.

3. The following 6809 instructions can not be converted directly to a single 68000 instruction.

ADCA	ext.,imm.,ind.
ADCB	" " "
ASL	ext.,ind.
ASR	" "
BITA	ext.,ind.
BITB	" "
EORA	ext.,ind.
EORB	" "
LSL	ext.,ind.
LSR	" "
MUL	inh.
ROL	ext.,ind.
ROR	" "
SBCA	ext.,imm.,ind.
SBCB	" " "
SEX	inh.

6809 INST.	ADDR. MODE	68000 INST.(S)		COMMENTS
ABX	INH.	ADD.W	D1,A0	
ADCA	EXT.	MOVE.B	<MEM>,D7	<EA>,Dn not allowed
		ADDX.B	D7,D0	for ADDX
	IMM.	MOVE.B	#<DATA>,D7	
		ADDX.B	D7,D0	
	IND.	MOVE.B	<EA>,D7	
		ADDX.B	D7,D0	
ADCB	same as ADCA using D1			
ADDA	EXT.	ADD.B	<MEM>,D0	A + M -> A
	IMM.	ADDI.B	#<DATA>,D0	A + M -> A
	IND.	ADD.B	<EA>,D0	(note 1)
ADDB	same as ADDA using D1			
ADDD	EXT.	ADD.W	<MEM>,D0	D0 must be set up
				as word
	IMM.	ADDI.W	#<DATA>,D0	
	IND.	ADD.W	<EA>,D0	(note 1)
ANDA	EXT.	AND.B	<MEM>,D0	A ^ M -> A
	IMM.	ANDI.B	#<DATA>,D0	A ^ IMM -> A
	IND.	AND.B	<EA>,D0	(note 1)
ANDB	same as ANDA using D1			
ANDCC	IMM.	ANDI.B	#<DATA>,CCR	CC ^ IMM -> CC
ASLA	INH.	ASL.B	#1,D0	
ASLB	INH.	ASL.B	#1,D1	
ASL	EXT.	MOVE.B	<MEM>,D7	can not be done with
		ASL.B	#1,D7	a single instruction
		MOVE.B	D7,<MEM>	(limited to word)
	IND.	MOVE.B	<EA>,D7	
		ASL.B	#1,D7	
		MOVE.B	D7,<EA>	(note 1)
ASR	same as ASL, other direction			

6809 INST.	ADDR. MODE	68000 INST.(S)		COMMENTS
BCC	REL.	BCC	<LABEL>	Branch if C = 0
BCS	REL.	BCS	<LABEL>	Branch if C = 1
BEQ	REL.	BEQ	<LABEL>	Branch if Z = 1
BGE	REL.	BGE	<LABEL>	Branch if >= zero
BGT	REL.	BGT	<LABEL>	Branch if > zero
BHI	REL.	BHI	<LABEL>	Branch if higher
BHS	REL.	BHS	<LABEL>	branch higher or same Motorola Assembler
		BCC	<LABEL>	BSO, Motorola Assembler
BITA	EXT.	MOVE.B	<MEM>,D7	
		BTST	D7,D0	
	IMM.	BTST	#<DATA>,D0	numbering is mod 32
	IND.	MOVE.B	<EA>,D7	(note 1)
		BTST	D7,D0	
BITB	same as BITA using D1			
BLE	REL.	BLE	<LABEL>	branch if <= zero
BLO	REL.	BLO	<LABEL>	branch if lower Motorola Assembler
	REL.	BCC	<LABEL>	BSO, Motorola Assembler
BLS	REL.	BLS	<LABEL>	branch lower or same
BLT	REL.	BLT	<LABEL>	branch if less
BMI	REL.	BMI	<LABEL>	branch if minus
BNE	REL.	BNE	<LABEL>	branch not = zero
BPL	REL.	BPL	<LABEL>	branch if plus
BRA	REL.	BRA	<LABEL>	branch always
BRN	REL.	NOP		branch never
BSR	REL.	BSR	<LABEL>	branch subroutine
BVC	REL.	BVC	<LABEL>	branch if no overflow
BVS	REL.	BVS	<LABEL>	branch if overflow
LBXX	REL.	BXX	<LABEL>	(note 2)

6809 INST.	ADDR. MODE	68000 INST.(S)		COMMENTS
CLRA	INH.	CLR.B	D0	0 -> A
CLRB	INH.	CLR.B	D1	0 -> B
CLR	EXT.	CLR.B	<MEM>	0 -> M
	IND.	CLR.B	<EA>	(note 1)
CMPA	EXT.	CMP.B	<MEM>,D0	
	IMM.	CMPI.B	#<DATA>,D0	
	IND.	CMP.B	<EA>,D0	(note 1)
CMPB	same as CMPA using D1			
CMPD	EXT.	CMP.W	<MEM>,D0	
	IMM.	CMPI.W	#<DATA>,D0	
	IND.	CMP.W	<EA>,D0	(note 1)
CMPS	EXT.	CMPA	<MEM>,A7	
	IMM.	CMPA	#<DATA>,A7	
	IND.	CMPA	<EA>,A7	(note 1)
CMPU	same as CMPS using A6			
CMPX	same as CMPS using A1			
CMPY	same as CMPS using A2			
COMA	INH.	NOT.B	D0	1's complement
COMB	INH.	NOT.B	D1	" "
COM	EXT.	NOT	<MEM>	" "
	IND.	NOT	<EA>	(note 1)
CWAI	INH.	STOP	#<DATA>	privileged (data is word)
DAA	not required do to available BCD instructions			
DECA	INH.	SUBQ.B	#1,D0	A - 1 -> A
DECB	INH.	SUBQ.B	#1,D1	B - 1 -> B
DEC	EXT.	SUBQ.B	#1,<MEM>	M - 1 -> M
	IND.	SUBQ.B	#1,<EA>	(note 1)
EORA	EXT.	MOVE.B	<MEM>,D7	EOR can not be done
		EOR.B	D7,D0	as <EA>,Dn
	IMM.	EORI.B	#<DATA>,D0	M XOR A -> A
	IND.	MOVE.B	<EA>,D7	(note 1)
		EOR.B	D7,D0	
EORB	same as EORA using D1			
EXG	INH.	EXG	Rx,Ry	always 32 bit

6809 INST.	ADDR. MODE	68000 INST.(S)		COMMENTS
INCA	INH.	ADDQ.B	#1,D0	
INCB	INH.	ADDQ.B	#1,D1	
INC	EXT.	ADDQ.B	#1,<MEM>	
	IND.	ADDQ.B	#1,<EA>	(note 1)
JMP	EXT.	JMP	<MEM>	
	IND.	JMP	<EA>	(note 1)
JSR	EXT.	JSR	<MEM>	
	IND.	JSR	<EA>	(note 1)
LDA	EXT.	MOVE.B	<MEM>,D0	
	IMM.	MOVE.B	#<DATA>,D0	
	IND.	MOVE.B	<EA>,D0	(note 1)
LDB	same as LDA using D1			
LDD	EXT.	MOVE.W	<MEM>,D0	
	IMM.	MOVE.W	#<DATA>,D0	
	IND.	MOVE.W	<EA>,D0	(note 1)
LDS	EXT.	MOVEA.W	<MEM>,A7	
	IMM.	MOVEA.W	#<DATA>,A7	
	IND.	MOVEA.W	<EA>,A7	(note 1)
LDU	same as LDS using A6			
LDX	same as LDS using A1			
LDY	same as LDS using A2			
LEAS	IND.	LEA	<EA>,A7	(note 1) limitations
LEAU	same as LEAS using A6			
LEAX	same as LEAS using A1			
LEAY	same as LEAS using A2			
LSLA	INH.	LSL.B	#1,D0	
LSLB	INH.	LSL.B	#1,D1	
LSL	EXT.	MOVE.B	<MEM>,D7	memory shifts are
		LSL.B	#1,D7	limited to word
		MOVE.B	D7,<MEM>	
	IND.	MOVE.B	<EA>,D7	(note 1)
		LSL.B	#1,D7	
		MOVE.B	D7,<EA>	
LSRA	INH.	LSR.B	#1,D0	
LSRB	INH.	LSR.B	#1,D1	
LSR	same as LSL, other direction			

6809 INST.	ADDR. MODE	68000 INST.(S)		COMMENTS
MUL	INH.	ANDI	#$00FF,D0	mask off unused
		MULU	D1,D0	upper byte
NEGA	INH.	NEG.B	D0	
NEGB	INH.	NEG.B	D1	
NEG	EXT.	NEG.B	<MEM>	
	IND.	NEG.B	<EA>	(note 1)
NOP	INH.	NOP		
ORA	EXT.	OR.B	<MEM>,D0	M .OR. A -> A
	IMM.	ORI.B	#<DATA>,D0	
	IND.	OR.B	<EA>,D0	
ORB	same as ORA using D1			
ORCC	IMM.	ORI	#<DATA>,CCR	
PSHS	IMM.	MOVEM	<REGLIST>,-(A7)	
PSHU	same as PSHS using A6			
PULS	IMM.	MOVEM	(A7)+,<REGLIST>	
PULU	same as PULS using A6			
ROLA	INH.	ROXL.B	#1,D0	
ROLB	INH.	ROXL.B	#1,D1	
ROL	EXT.	MOVE.B	<MEM>,D7	memory rotate is
		ROL.B	#1,D7	limited to word
		MOVE.B	D7,<MEM>	
	IND.	MOVE.B	<EA>,D7	(note 1)
		ROL.B	#1,D7	
		MOVE.B	D7,<MEM>	
RORA	INH.	ROXR.B	#1,D0	
RORB	INH.	ROXR.B	#1,D1	
ROR	same as ROL, other direction			
RTI	INH.	RTE		
RTS	INH.	RTS		
SBCA	EXT.	MOVE.B	<MEM>,D7	
		SUBX.B	D7,D0	A - M - C -> A
	IMM.	MOVE.B	#<DATA>,D7	
		SUBX.B	D7,D0	
	IND.	MOVE.B	<EA>,D7	(note 1)
		SUBX.B	D7,D0	
SBCB	same as SBCA using D1			B - M - C -> B

6809 INST.	ADDR. MODE	68000 INST.(S)		COMMENTS
SEX	INH.	EXT.W	D1	sign extend B
		MOVE.w	D1,D0	into A
STA	EXT.	MOVE.B	D0,<MEM>	A -> M
	IND.	MOVE.B	D0,<EA>	A -> M (note 1)
STB	same as STA using D1			
STD	EXT.	MOVE.W	D0,<MEM>	D -> M
	IND.	MOVE.W	D0,<EA>	D -> M (note 1)
STS	EXT.	MOVEA	A7,<MEM>	S -> M
	IND.	MOVEA	A7,<EA>	S -> M (note 1)
STU	same as STS using A6			
STX	same as STS using A1			
STY	same as STS using A2			
SUBA	EXT.	SUB.B	<MEM>,D0	A - M -> A
	IMM.	SUBI.B	#<DATA>,D0	
	IND.	SUB.B	<EA>,D0	(note 1)
SUBB	same as SUBA using D1			
SUBD	EXT.	SUB.W	<MEM>,D0	D - M -> D
	IMM.	SUBI.W	#<DATA>,D0	
	IND.	SUB.W	<EA>,D0	(note 1)
SWI	INH.	TRAP	#<VECTOR X>	stacks only PC + CC
SWI2	INH.	TRAP	#<VECTOR Y>	" "
SWI3	INH.	TRAP	#<VECTOR Z>	" "
SYNC	INH.	STOP	#xxx	
TFR	INH.	MOVE.W	R1,R2	
TSTA	INH.	TST.B	D0	test A
TSTB	INH.	TST.B	D1	test B
TST	EXT.	TST.B	<MEM>	test M
	IND.	TST.B	<EA>	(note 1)

****

note 1: EA = d(An,Xi)
(An)
(An)+
-(An)
d(An)

note 2: Assembler will allow 16 bit offsets (automatically) if displacement will not fit into a single byte.