<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA</td>
<td>6</td>
</tr>
<tr>
<td>AAD</td>
<td></td>
</tr>
<tr>
<td>AAM</td>
<td>8</td>
</tr>
<tr>
<td>AAS</td>
<td>8</td>
</tr>
<tr>
<td>ADC</td>
<td>6</td>
</tr>
<tr>
<td>ADD</td>
<td>5</td>
</tr>
<tr>
<td>AND</td>
<td>10</td>
</tr>
<tr>
<td>CALL</td>
<td>12</td>
</tr>
<tr>
<td>CBW</td>
<td>9</td>
</tr>
<tr>
<td>CLC</td>
<td>17</td>
</tr>
<tr>
<td>CLD</td>
<td>17</td>
</tr>
<tr>
<td>CLI</td>
<td>17</td>
</tr>
<tr>
<td>CMC</td>
<td>17</td>
</tr>
<tr>
<td>CMP</td>
<td>7</td>
</tr>
<tr>
<td>CMPS</td>
<td>12</td>
</tr>
<tr>
<td>CWD</td>
<td>9</td>
</tr>
<tr>
<td>DAA</td>
<td>6</td>
</tr>
<tr>
<td>DAS</td>
<td>8</td>
</tr>
<tr>
<td>DEC</td>
<td>7</td>
</tr>
<tr>
<td>DIV</td>
<td>8</td>
</tr>
<tr>
<td>ESC</td>
<td>17</td>
</tr>
<tr>
<td>HLT</td>
<td>17</td>
</tr>
<tr>
<td>IDIV</td>
<td>8</td>
</tr>
<tr>
<td>IMUL</td>
<td>8</td>
</tr>
<tr>
<td>IN</td>
<td>4</td>
</tr>
<tr>
<td>INC</td>
<td>6</td>
</tr>
<tr>
<td>INT</td>
<td>16</td>
</tr>
<tr>
<td>INTO</td>
<td>16</td>
</tr>
<tr>
<td>IRET</td>
<td>16</td>
</tr>
<tr>
<td>JA</td>
<td>15</td>
</tr>
<tr>
<td>JAE</td>
<td>15</td>
</tr>
<tr>
<td>JB</td>
<td>14</td>
</tr>
<tr>
<td>JBE</td>
<td>14</td>
</tr>
<tr>
<td>JCXZ</td>
<td>16</td>
</tr>
<tr>
<td>JE</td>
<td>13</td>
</tr>
<tr>
<td>JG</td>
<td>15</td>
</tr>
<tr>
<td>JGE</td>
<td>14</td>
</tr>
<tr>
<td>JL</td>
<td>14</td>
</tr>
<tr>
<td>JLE</td>
<td>14</td>
</tr>
<tr>
<td>JMP</td>
<td>13</td>
</tr>
<tr>
<td>JNA</td>
<td>14</td>
</tr>
<tr>
<td>JNAE</td>
<td>14</td>
</tr>
<tr>
<td>JNB</td>
<td>15</td>
</tr>
<tr>
<td>JNBE</td>
<td>15</td>
</tr>
<tr>
<td>JNE</td>
<td>14</td>
</tr>
<tr>
<td>JNG</td>
<td>14</td>
</tr>
<tr>
<td>JNGE</td>
<td>14</td>
</tr>
<tr>
<td>JNL</td>
<td>14</td>
</tr>
<tr>
<td>JNLE</td>
<td>16</td>
</tr>
<tr>
<td>JNO</td>
<td>15</td>
</tr>
<tr>
<td>JNP</td>
<td>15</td>
</tr>
<tr>
<td>JNS</td>
<td>15</td>
</tr>
<tr>
<td>JNZ</td>
<td>14</td>
</tr>
<tr>
<td>JO</td>
<td>14</td>
</tr>
<tr>
<td>JP</td>
<td>14</td>
</tr>
<tr>
<td>JPE</td>
<td>14</td>
</tr>
<tr>
<td>JPO</td>
<td>15</td>
</tr>
<tr>
<td>JS</td>
<td>14</td>
</tr>
<tr>
<td>JZ</td>
<td>13</td>
</tr>
<tr>
<td>LAHF</td>
<td>5</td>
</tr>
<tr>
<td>LDS</td>
<td>5</td>
</tr>
<tr>
<td>LEA</td>
<td>5</td>
</tr>
<tr>
<td>LES</td>
<td>5</td>
</tr>
<tr>
<td>LOCK</td>
<td>17</td>
</tr>
<tr>
<td>LODS</td>
<td>12</td>
</tr>
<tr>
<td>LOOP</td>
<td>15</td>
</tr>
<tr>
<td>LOOPE</td>
<td>15</td>
</tr>
<tr>
<td>LOOPNE</td>
<td>15</td>
</tr>
<tr>
<td>LOOPNZ</td>
<td>15</td>
</tr>
<tr>
<td>LOOPZ</td>
<td>15</td>
</tr>
<tr>
<td>MOV</td>
<td>3</td>
</tr>
<tr>
<td>MOVSB</td>
<td>12</td>
</tr>
<tr>
<td>MUL</td>
<td>8</td>
</tr>
<tr>
<td>NEG</td>
<td>7</td>
</tr>
<tr>
<td>NOP</td>
<td>17</td>
</tr>
<tr>
<td>NOT</td>
<td>9</td>
</tr>
<tr>
<td>OR</td>
<td>11</td>
</tr>
<tr>
<td>OUT</td>
<td>4</td>
</tr>
<tr>
<td>POP</td>
<td>4</td>
</tr>
<tr>
<td>POPF</td>
<td>5</td>
</tr>
<tr>
<td>PUSH</td>
<td>3</td>
</tr>
<tr>
<td>PUSHF</td>
<td>5</td>
</tr>
<tr>
<td>RCL</td>
<td>10</td>
</tr>
<tr>
<td>RCR</td>
<td>10</td>
</tr>
<tr>
<td>REP</td>
<td>11</td>
</tr>
<tr>
<td>RET</td>
<td>13</td>
</tr>
<tr>
<td>ROL</td>
<td>9</td>
</tr>
<tr>
<td>ROR</td>
<td>10</td>
</tr>
<tr>
<td>SAHF</td>
<td>5</td>
</tr>
<tr>
<td>SAL</td>
<td>9</td>
</tr>
<tr>
<td>SAR</td>
<td>9</td>
</tr>
<tr>
<td>SBB</td>
<td>7</td>
</tr>
<tr>
<td>SCAS</td>
<td>12</td>
</tr>
<tr>
<td>SHL</td>
<td>9</td>
</tr>
<tr>
<td>SHR</td>
<td>9</td>
</tr>
<tr>
<td>STC</td>
<td>17</td>
</tr>
<tr>
<td>STD</td>
<td>17</td>
</tr>
<tr>
<td>STI</td>
<td>17</td>
</tr>
<tr>
<td>STOS</td>
<td>12</td>
</tr>
<tr>
<td>SUB</td>
<td>6</td>
</tr>
<tr>
<td>TEST</td>
<td>10</td>
</tr>
<tr>
<td>WAIT</td>
<td>17</td>
</tr>
<tr>
<td>XCHG</td>
<td>4</td>
</tr>
<tr>
<td>XLAT</td>
<td>5</td>
</tr>
<tr>
<td>XOR</td>
<td>11</td>
</tr>
</tbody>
</table>
**8086 REGISTER MODEL**

- **AX:** AH, AL
  - **ACCUMULATOR**
- **BX:** BH, BL
  - **BASE**
- **CX:** CH, CL
  - **COUNT**
- **DX:** DH, DL
  - **DATA**
- **SP**
  - **STACK POINTER**
- **BP**
  - **BASE POINTER**
- **SI**
  - **SOURCE INDEX**
- **DI**
  - **DESTINATION INDEX**

- **IP**
  - **INSTRUCTION POINTER**
- **FLAGS**
  - **STATUS FLAGS**

- **CS**
  - **CODE SEGMENT**
- **DS**
  - **DATA SEGMENT**
- **SS**
  - **STACK SEGMENT**
- **ES**
  - **EXTRA SEGMENT**

Instructions which reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file:

\[
\begin{array}{cccccccc}
\text{X} & \text{X} & \text{X} & \text{X} & \text{OF} & \text{DF} & \text{IF} & \text{TF} & \text{SF} & \text{ZF} & \text{X} & \text{AF} & \text{X} & \text{PF} & \text{X} & \text{CF} \\
\end{array}
\]

\( X = \text{Don't Care} \)

- **AF:** AUXILIARY CARRY — BCD
- **CF:** CARRY FLAG
- **PF:** PARITY FLAG
- **SF:** SIGN FLAG
- **ZF:** ZERO FLAG

**OPERAND SUMMARY**

"reg" field Bit Assignments:

<table>
<thead>
<tr>
<th>16-Bit (w = 1)</th>
<th>8-Bit (w = 0)</th>
<th>Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>000 AX</td>
<td>000 AL</td>
<td>00 ES</td>
</tr>
<tr>
<td>001 CX</td>
<td>001 CL</td>
<td>01 CS</td>
</tr>
<tr>
<td>010 DX</td>
<td>010 DL</td>
<td>10 SS</td>
</tr>
<tr>
<td>011 BX</td>
<td>011 BL</td>
<td>11 DS</td>
</tr>
<tr>
<td>100 SP</td>
<td>100 AH</td>
<td></td>
</tr>
<tr>
<td>101 BP</td>
<td>101 CH</td>
<td></td>
</tr>
<tr>
<td>110 SI</td>
<td>110 DH</td>
<td></td>
</tr>
<tr>
<td>111 DI</td>
<td>111 BH</td>
<td></td>
</tr>
</tbody>
</table>

**SECOND INSTRUCTION BYTE SUMMARY**

<table>
<thead>
<tr>
<th>mod</th>
<th>Displacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>DISP = 0*, disp-low and disp-high are absent</td>
</tr>
<tr>
<td>01</td>
<td>DISP = disp-low sign-extended to 16-bits, disp-high is absent</td>
</tr>
<tr>
<td>10</td>
<td>DISP = disp-high: disp-low</td>
</tr>
<tr>
<td>11</td>
<td>r/m is treated as a &quot;reg&quot; field</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>r/m</th>
<th>Operand Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>(BX) + (SI) + DISP</td>
</tr>
<tr>
<td>001</td>
<td>(BX) + (DI) + DISP</td>
</tr>
<tr>
<td>010</td>
<td>(BP) + (SI) + DISP</td>
</tr>
<tr>
<td>011</td>
<td>(BP) + (DI) + DISP</td>
</tr>
<tr>
<td>100</td>
<td>(SI) + DISP</td>
</tr>
<tr>
<td>101</td>
<td>(DI) + DISP</td>
</tr>
<tr>
<td>110</td>
<td>(BP) + DISP*</td>
</tr>
<tr>
<td>111</td>
<td>(BX) + DISP</td>
</tr>
</tbody>
</table>

DISP follows 2nd byte of instruction (before data if required).

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low.

**Operand Address (EA) Timing (clocks):**

Add 4 clocks for word operands at ODD ADDRESSES.

Immediate Offset = 6
Base (BX, BP, SI, DI) = 5
Base + DISP = 9
Base + Index (BP + DI, BX + SI) = 7
Base + Index (BP + SI, BX + DI) = 8
Base + Index (BP + DI, BX + SI) + DISP = 11
Base + Index (BP + SI, BX + DI) + DISP = 12
ASSEMBLER DIRECTIVES

Symbol Definition:
  EQU
  LABEL
  PURGE

Memory Reservation and
Data Definition:
  DB
  DW
  DD
  RECORD

Location Counter and
Segmentation Control:
  SEGMENT/ENDS
  ORG
  GROUP
  ASSUME
  PROC/ENDP
  CODEMACRO/ENDM

Program Linkage:
  NAME
  PUBLIC
  EXTRN
  END

PROCESSOR RESET
REGISTER INITIALIZATION

Flags = 0000H  (to disable interrupts and single-stepping)

CS = FFFFFH  (to begin execution at FFFFFH)
IP = 0000H

DS = 0000H
SS = 0000H
ES = 0000H

No other registers are acted upon during reset.

MCS-86™ RESERVED LOCATIONS

Reserved Memory Locations
Intel Corporation reserves the use of memory locations FFFFOH through FFFFFFH (with the exception of FFFFOH - FFFE5H for JMP instr.) for Intel hardware and software products. If you use these locations for some other purpose, you may preclude compatibility of your system with certain of these products.

Reserved Input/Output Locations
Intel Corporation reserves the use of input/output locations F8H through FFH for Intel hardware and software products. Users who wish to maintain compatibility with present and future Intel products should not use these locations.

Reserved Interrupt Locations
Intel Corporation reserves the use of interrupts 0-31 (locations 00H through 7FH) for Intel hardware and software products. Users who wish to maintain compatibility with present and future Intel products should not use these locations.

Interrupts 0 through 4 (00H-13H) currently have dedicated hardware functions as defined below.

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>Location</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00H-03H</td>
<td>Divide by zero</td>
</tr>
<tr>
<td>1</td>
<td>04H-07H</td>
<td>Single step</td>
</tr>
<tr>
<td>2</td>
<td>08H-0BH</td>
<td>Non-maskable interrupt</td>
</tr>
<tr>
<td>3</td>
<td>0CH-0FH</td>
<td>One-byte interrupt instr.</td>
</tr>
<tr>
<td>4</td>
<td>10H-13H</td>
<td>Interrupt on overflow</td>
</tr>
</tbody>
</table>

INTERRUPT POINTER TABLE

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>3FFH</td>
<td>IP255</td>
<td>CS255</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1FH</td>
<td>IP1</td>
<td>CS1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0FH</td>
<td>IP0</td>
<td>CS0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

INTERRUPT TYPE VECTOR
x 4 IS LOCATION FOR ADDRESS OF INTERRUPT SERVICE ROUTINE
**8086 INSTRUCTION**

<table>
<thead>
<tr>
<th>Hi</th>
<th>Lo</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
<td>ADD</td>
<td>PUSH</td>
<td>ES</td>
<td>POP</td>
</tr>
<tr>
<td>1</td>
<td>ADC</td>
<td>ADC</td>
<td>ADC</td>
<td>ADC</td>
<td>ADC</td>
<td>ADD</td>
<td>PUSH</td>
<td>ES</td>
<td>POP</td>
</tr>
<tr>
<td>2</td>
<td>AND</td>
<td>AND</td>
<td>AND</td>
<td>AND</td>
<td>AND</td>
<td>SEG</td>
<td>=ES</td>
<td>DAA</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>XOR</td>
<td>SEG</td>
<td>=SS</td>
<td>AAA</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>INC</td>
<td>INC</td>
<td>INC</td>
<td>INC</td>
<td>INC</td>
<td>INC</td>
<td>INC</td>
<td>INC</td>
<td>INC</td>
</tr>
<tr>
<td>5</td>
<td>PUSH</td>
<td>PUSH</td>
<td>PUSH</td>
<td>PUSH</td>
<td>PUSH</td>
<td>PUSH</td>
<td>PUSH</td>
<td>PUSH</td>
<td>PUSH</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hi</th>
<th>Lo</th>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OR</td>
<td>OR</td>
<td>OR</td>
<td>OR</td>
<td>OR</td>
<td>OR</td>
<td>POP</td>
<td>ES</td>
<td>CS</td>
</tr>
<tr>
<td>1</td>
<td>SBB</td>
<td>SBB</td>
<td>SBB</td>
<td>SBB</td>
<td>SBB</td>
<td>SBB</td>
<td>SBB</td>
<td>SBB</td>
<td>SBB</td>
</tr>
<tr>
<td>2</td>
<td>SUB</td>
<td>SUB</td>
<td>SUB</td>
<td>SUB</td>
<td>SUB</td>
<td>SUB</td>
<td>SUB</td>
<td>SUB</td>
<td>SUB</td>
</tr>
<tr>
<td>3</td>
<td>CMP</td>
<td>CMP</td>
<td>CMP</td>
<td>CMP</td>
<td>CMP</td>
<td>CMP</td>
<td>CMP</td>
<td>CMP</td>
<td>CMP</td>
</tr>
<tr>
<td>4</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
<td>DEC</td>
</tr>
<tr>
<td>5</td>
<td>POP</td>
<td>POP</td>
<td>POP</td>
<td>POP</td>
<td>POP</td>
<td>POP</td>
<td>POP</td>
<td>POP</td>
<td>POP</td>
</tr>
</tbody>
</table>

---

**SET MATRIX**

<table>
<thead>
<tr>
<th>Hi</th>
<th>Lo</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Immed</td>
<td>Immed</td>
<td>Immed</td>
<td>Immed</td>
<td>TEST</td>
<td>TEST</td>
<td>XCHG</td>
<td>XCHG</td>
<td>XCHG</td>
</tr>
<tr>
<td>1</td>
<td>NOP</td>
<td>XCHG</td>
<td>CX</td>
<td>XCHG</td>
<td>DX</td>
<td>XCHG</td>
<td>SP</td>
<td>XCHG</td>
<td>XCHG</td>
</tr>
<tr>
<td>2</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>b</td>
<td>MOV</td>
<td>MOV</td>
</tr>
<tr>
<td>3</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>w</td>
<td>MOV</td>
<td>MOV</td>
</tr>
<tr>
<td>4</td>
<td>TEST</td>
<td>TEST</td>
<td>TEST</td>
<td>TEST</td>
<td>TEST</td>
<td>TEST</td>
<td>b</td>
<td>TEST</td>
<td>TEST</td>
</tr>
<tr>
<td>5</td>
<td>B</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
</tr>
<tr>
<td>6</td>
<td>B</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
<td>MOV</td>
</tr>
<tr>
<td>7</td>
<td>Shift</td>
<td>Shift</td>
<td>Shift</td>
<td>Shift</td>
<td>Shift</td>
<td>Shift</td>
<td>Shift</td>
<td>Shift</td>
<td>Shift</td>
</tr>
<tr>
<td>8</td>
<td>E</td>
<td>LOCK</td>
<td>REP</td>
<td>REP</td>
<td>REP</td>
<td>REP</td>
<td>REP</td>
<td>REP</td>
<td>REP</td>
</tr>
</tbody>
</table>

---

**Legend**

- `b` = byte operation
- `d` = direct
- `f` = from CPU reg
- `i` = immediate
- `ia` = imm. to accum.
- `id` = indirect
- `is` = imm. byte, sign ext.
- `l` = long ie. intersegment
- `m` = memory
- `r/m` = EA is second byte
- `si` = short intrasegment
- `sr` = segment register
- `t` = to CPU reg
- `v` = variable
- `w` = word operation
- `z` = zero

---

**Modifications**

- `mod` = `f/r/m`
- `ADD` = `OR`
- `ADC` = `AND`
- `SBB` = `SUB`
- `AND` = `XOR`
- `OR` = `CMP`

---

**Shifts**

- `ROL` = `ROR`
- `RCL` = `RCR`
- `SAL` = `SAR`
- `SHL` = `SHR`

---

**Group 1**

- `TEST` = `NOT`
- `NEG` = `MUL`
- `MUL` = `IMUL`
- `DIV` = `IDIV`

---

**Group 2**

- `INC` = `DEC`
- `CALL` = `CALL`
- `JMP` = `JMP`
- `PUSH` = `PUSH`

---

**Notes**

- `ESC` = `ESC`
- `OUT` = `OUT`

---

**Mnemonics © Intel, 1978.**
MEMORY SEGMENTATION MODEL

LOGICAL MEMORY SPACE

CODE SEGMENT
FFFFFH

64 KB

STACK SEGMENT

XXXXOH

OFFSET ADDRESS

DISPLACEMENT

15

WORD

15

SELECTED SEGMENT REGISTER
CS, SS, DS, ES

OR NONE FOR I/O, INT

ADDRES

0000H

EXTRA DATA SEGMENT

DATA SEGMENT

MSB

BYTE

SEGMENT OVERRIDE PREFIX

0 0 1 reg 1 1 0

Timing: 2 clocks

USE OF SEGMENT OVERRIDE

<table>
<thead>
<tr>
<th>Operand Register</th>
<th>Default</th>
<th>With Override Prefix</th>
<th>With Override Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP (code address)</td>
<td>CS</td>
<td>Never</td>
<td>Bay</td>
</tr>
<tr>
<td>SP (stack address)</td>
<td>SS</td>
<td>Never</td>
<td>Bay</td>
</tr>
<tr>
<td>BP (stack address or stack marker)</td>
<td>SS</td>
<td>BP + DS or ES, or CS</td>
<td>Bay</td>
</tr>
<tr>
<td>SI or DI (not incl. strings)</td>
<td>DS</td>
<td>ES, SS, or CS</td>
<td>Bay</td>
</tr>
<tr>
<td>SI (implicit source addr for strings)</td>
<td>DS</td>
<td>ES, SS, or CS</td>
<td>Bay</td>
</tr>
<tr>
<td>DI (implicit dest addr for strings)</td>
<td>DS</td>
<td>ES, SS, or CS</td>
<td>Bay</td>
</tr>
<tr>
<td></td>
<td>ES</td>
<td>Never</td>
<td>Bay</td>
</tr>
</tbody>
</table>

DATA TRANSFER

MOV = Move
Register/memory to/from register

1 0 0 0 1 0 d w mod reg r/m

Timing (clocks):

register to register 2
memory to register 8+EA
register to memory 9+EA

Immediate to register/memory

1 1 0 0 0 1 1 w mod 0 0 0 r/m data data if w=1

Timing: 10+EA clocks

Immediate to register

1 0 1 1 w reg data data if w=1

Timing: 4 clocks

Memory to accumulator

1 0 1 0 0 0 0 w addr-low addr-high

Timing: 10 clocks

Accumulator to memory

1 0 1 0 0 0 1 w addr-low addr-high

Timing: 10 clocks

Register/memory to segment register

1 0 0 0 1 1 1 0 mod 0 reg r/m

Timing (clocks):

register to register 2
memory to register 8+EA

Segment register to register/memory

1 0 0 0 1 1 0 mod 0 reg r/m

Timing (clocks):

register to register 2
register to memory 9+EA

PUSH = Push
Register/memory

1 1 1 1 1 1 1 1 mod 1 1 0 r/m

Timing (clocks):

register memory 10

Register

0 1 0 1 0 reg

Timing: 10 clocks

(Continued on following page)

Segment register
0 0 0 reg 1 1 0
Timing: 10 clocks

POP = Pop
Register/memory
1 0 0 0 1 1 1 1 mod 0 0 0 r/m
Timing (clocks): 
  register 8
  memory 17+EA

Register
0 1 0 1 1 reg
Timing: 8 clocks

Segment register
0 0 0 reg 1 1 1
Timing: 8 clocks

XCHG = Exchange
Register/memory with register
1 0 0 0 0 1 1 w mod reg r/m
Timing (clocks): 
  register with register 4
  memory with register 17+EA

Register with accumulator
1 0 0 1 0 reg
Timing: 3 clocks

IN = Input to AL/Ax from
Fixed port
1 1 1 0 0 1 0 w port
Timing: 10 clocks

Variable port (DX)
1 1 1 0 1 1 0 w
Timing: 8 clocks

OUT = Output from AL/Ax to
Fixed port
1 1 1 0 0 1 1 w port
Timing: 10 clocks

Variable port (DX)
1 1 1 0 1 1 1 w
Timing: 8 clocks

XLAT = Translate byte to AL
1 1 0 1 0 1 1 1
Timing: 11 clocks

LEA = Load EA to register
1 0 0 0 1 1 0 1 mod reg r/m
Timing: 2+EA clocks

LDS = Load pointer to DS
1 1 0 0 0 1 0 1 mod reg r/m
Timing: 16+EA clocks

LES = Load pointer to ES
1 1 0 0 0 1 0 0 mod reg r/m
Timing: 16+EA clocks

LAHF = Load AH with flags
1 0 0 1 1 1 1 1
Timing: 4 clocks

SAHF = Store AH into flags
1 0 0 1 1 1 1 0
Timing: 4 clocks

PUSHF = Push flags
1 0 0 1 1 1 0 0
Timing: 10 clocks

POPF = Pop flags
1 0 0 1 1 1 0 1
Timing: 8 clocks

ARITHMETIC

ADD = Add
Reg./memory with register to either
0 0 0 0 0 0 d w mod reg r/m
Timing (clocks): 
  register to register 3
  memory to register 9+EA
  register to memory 16+EA

Immediate to register/memory
1 0 0 0 0 0 s w mod 0 0 0 r/m data data if s:w=01
Timing (clocks): 
  immediate to register 4
  immediate to memory 17+EA

(Continued on following page)  Mnemonics © Intel, 1978.
### Immediate to accumulator

**0000010 w**
- **Data**: data if `w=1`

**Timing**: 4 clocks

### ADC = Add with carry
Reg./memory with register to either

**0001000 d w mod reg r/m**
- **Timing (clocks)**:
  - Register to register: 3
  - Memory to register: 9+EA
  - Register to memory: 16+EA

### Immediate to register/memory

**100000 s w mod 010 r/m**
- **Data**: data
- **Data if s.w=01**: data if `w=1`

**Timing (clocks)**:
- Immediate to register: 4
- Immediate to memory: 17+EA

### Immediate from register/memory

**100000 s w mod 101 r/m**
- **Data**: data if `s.w=01`

**Timing (clocks)**:
- Immediate from register: 4
- Immediate from memory: 17+EA

### SBB = Subtract with borrow
Reg./memory and register to either

**0001110 d w mod reg r/m**
- **Timing (clocks)**:
  - Register from register: 3
  - Memory from register: 9+EA
  - Register from memory: 16+EA

### Immediate from accumulator

**0001110 w**
- **Data**: data if `w=1`

**Timing**: 4 clocks

### DEC = Decrement
Register/memory

**1111111 w mod 000 r/m**
- **Timing (clocks)**:
  - Memory: 15+EA

### DEC = Decrement
Register/memory

**1111111 w mod 001 r/m**
- **Timing (clocks)**:
  - Register: 2
  - Memory: 15+EA

### Register

**01000 reg**
- **Timing**: 2 clocks

### AAA = ASCII adjust for add

**00110111**
- **Timing**: 4 clocks

### DAA = Decimal adjust for add

**00100111**
- **Timing**: 4 clocks

### SUB = Subtract
Reg./memory and register to either

**0011010 d w mod reg r/m**
- **Timing (clocks)**:
  - Register from register: 3
  - Memory from register: 9+EA
  - Register from memory: 16+EA

### CMP = Compare
Register/memory and register

**001110 d w mod reg r/m**
- **Timing (clocks)**:
  - Register with register: 3
  - Memory with register: 9+EA
  - Register with memory: 9+EA

---

(Continued from the previous page)
Immediate with register/memory

| 0 0 0 0 0 s w | mod 1 1 1 r/m | data | data if s:w=01 |

Timing (clocks):
- immediate with register: 4
- immediate with memory: 17+EA

Immediate with accumulator

| 0 0 1 1 1 1 0 w | data | data if w=1 |

Timing: 4 clocks

**AAS** = ASCII adjust for subtract

| 0 0 1 1 1 1 1 1 |

Timing: 4 clocks

**DAS** = Decimal adjust for subtract

| 0 0 1 0 1 1 1 1 |

Timing: 4 clocks

**MUL** = Multiply (unsigned)

| 1 1 1 1 0 1 1 w | mod 1 0 0 r/m |

Timing (clocks):
- 8-bit: 71+EA
- 16-bit: 124+EA

**IMUL** = Integer multiply (signed)

| 1 1 1 1 0 1 1 w | mod 1 0 1 r/m |

Timing (clocks):
- 8-bit: 90+EA
- 16-bit: 144+EA

**AAM** = ASCII adjust for multiply

| 1 1 0 1 0 1 0 | 0 0 0 0 1 0 1 0 |

Timing: 83 clocks

**DIV** = Divide (unsigned)

| 1 1 1 1 0 1 1 w | mod 1 1 0 r/m |

Timing (clocks):
- 8-bit: 90+EA
- 16-bit: 155+EA

**IDIV** = Integer divide (signed)

| 1 1 1 1 0 1 1 w | mod 1 1 1 r/m |

Timing (clocks):
- 8-bit: 112+EA
- 16-bit: 177+EA

**AAD** = ASCII adjust for divide

| 1 1 0 1 0 1 0 | 0 0 0 0 1 0 1 0 |

Timing: 60 clocks

**CBW** = Convert byte to word

| 1 0 0 1 1 0 0 |

Timing: 2 clocks

**CWD** = Convert word to double word

| 1 0 0 1 1 0 1 |

Timing: 5 clocks

**LOGIC**

**NOT** = Invert

| 1 1 1 1 0 1 1 w | mod 0 1 0 r/m |

Timing (clocks):
- register: 3
- memory: 16+EA

**SHL/SAL** = Shift logical/arithmetic left

| 1 1 0 1 0 0 v w | mod 1 0 0 r/m |

Timing (clocks):
- single-bit register: 2
- single-bit memory: 15+EA
- variable-bit register: 8+4/bit
- variable-bit memory: 20+EA+4/bit

**SHR** = Shift logical right

| 1 1 0 1 0 0 v w | mod 1 0 1 r/m |

Timing (clocks):
- single-bit register: 2
- single-bit memory: 15+EA
- variable-bit register: 8+4/bit
- variable-bit memory: 20+EA+4/bit

**SAR** = Shift arithmetic right

| 1 1 0 1 0 0 v w | mod 1 1 1 r/m |

Timing (clocks):
- single-bit register: 2
- single-bit memory: 15+EA
- variable-bit register: 8+4/bit
- variable-bit memory: 20+EA+4/bit

**ROL** = Rotate left

| 1 1 0 1 0 0 v w | mod 0 0 0 r/m |

Timing (clocks):
- single-bit register: 2
- single-bit memory: 15+EA
- variable-bit register: 8+4/bit
- variable-bit memory: 20+EA+4/bit

**ROR** = Rotate right

```
110100 w mod 001 r/m
```

Timing (clocks):
- single-bit register: 2
- single-bit memory: 15+EA
- variable-bit register: 8+4/bit
- variable-bit memory: 20+EA+4/bit

**RCL** = Rotate through carry left

```
110100 w mod 010 r/m
```

Timing (clocks):
- single-bit register: 2
- single-bit memory: 15+EA
- variable-bit register: 8+4/bit
- variable-bit memory: 20+EA+4/bit

**RCR** = Rotate through carry right

```
110100 w mod 011 r/m
```

Timing (clocks):
- single-bit register: 2
- single-bit memory: 15+EA
- variable-bit register: 8+4/bit
- variable-bit memory: 20+EA+4/bit

**AND** = And
Reg./memory and register to either

```
001000 d w mod reg r/m
```

Timing (clocks):
- register to register: 3
- memory to register: 9+EA
- register to memory: 16+EA

**Immediate to register/memory**

```
1000000 w mod 001 r/m data data if w=1
```

Timing (clocks):
- immediate to register: 4
- immediate to memory: 17+EA

**Immediate to accumulator**

```
0010010 w data data if w=1
```

Timing: 4 clocks

**TEST** = And function to flags, no result
Register/memory and register

```
1000010 w mod reg r/m
```

Timing (clocks):
- register to register: 3
- register with memory: 9+EA

(Continued on following page)
MOVS = Move String
1010010 w
Timing: 17 clocks
CMPS = Compare String
1010011 w
Timing: 22 clocks
SCAS = Scan String
1010111 w
Timing: 15 clocks
LODS = Load String
1010110 w
Timing: 12 clocks
STOS = Store String
1010101 w
Timing: 10 clocks

CONTROL TRANSFER

NOTE: Queue reinitialization is not included in the timing information for transfer operations. To account for instruction loading, add 8 clocks to timing numbers.

CALL = Call
Direct within segment
11101000 disp-low disp-high
Timing: 11 clocks
Indirect within segment
11111111 mod 010 r/m
Timing: 13+EA clocks
Direct intersegment
10011010 offset-low offset-high
Timing: 20 clocks
Indirect intersegment
111111111 mod 011 r/m
Timing: 29+EA clocks

JMP = Unconditional Jump
Direct within segment
11101001 disp-low disp-high
Timing: 7 clocks
Direct within segment-short
11101011 disp
Timing: 7 clocks
Indirect within segment
11111111 mod 100 r/m
Timing: 7+EA clocks
Direct intersegment
11101010 offset-low offset-high
Timing: 7 clocks
Indirect intersegment
11111111 mod 101 r/m
Timing: 16+EA clocks

RET = Return from CALL
Within segment
110000011
Timing: 8 clocks
Within seg. adding immed to SP
110001010 data-low data-high
Timing: 12 clocks
Intersegment
11001011
Timing: 18 clocks
Intersegment, adding immediate to SP
11001010 data-low data-high
Timing: 17 clocks

JE/JZ = Jump on equal/zero
01110100 disp

Timing (clocks): Jump is taken 8
Jump is not taken 4
JL/JNGE = Jump on less/not greater or equal
   0 1 1 1 1 1 0 0 disp
Timing (clocks): Jump is taken 8
                Jump is not taken 4

JLE/JNG = Jump on less or equal/not greater
   0 1 1 1 1 1 1 0 disp
Timing (clocks): Jump is taken 8
                Jump is not taken 4

JB/JNAE = Jump on below/ not above or equal
   0 1 1 1 0 0 0 1 0 disp
Timing (clocks): Jump is taken 8
                Jump is not taken 4

JBE/JNA = Jump on below or equal/not above
   0 1 1 1 0 1 1 1 0 disp
Timing (clocks): Jump is taken 8
                Jump is not taken 4

JP/JPE = Jump on parity/parity even
   0 1 1 1 1 0 1 0 0 disp
Timing (clocks): Jump is taken 8
                Jump is not taken 4

JO = Jump on overflow
   0 1 1 1 0 0 0 0 0 disp
Timing (clocks): Jump is taken 8
                Jump is not taken 4

JS = Jump on sign
   0 1 1 1 1 0 0 0 disp
Timing (clocks): Jump is taken 8
                Jump is not taken 4

JNE/JNZ = Jump on not equal/not zero
   0 1 1 1 0 1 0 1 1 disp
Timing (clocks): Jump is taken 8
                Jump is not taken 4

JNL/JGE = Jump on not less/greater or equal
   0 1 1 1 1 1 0 1 disp
Timing (clocks): Jump is taken 8
                Jump is not taken 4

JNLE/JG = Jump on not less or equal/greater
   0 1 1 1 1 1 1 1 disp
Timing (clocks): Jump is taken 8
                Jump is not taken 4

JNB/JAE = Jump on not below/above or equal
   0 1 1 1 0 0 1 1 disp
Timing (clocks): Jump is taken 8
                Jump is not taken 4

JNB/JA = Jump on not below or equal/above
   0 1 1 1 0 1 1 1 disp
Timing (clocks): Jump is taken 8
                Jump is not taken 4

JNP/JPO = Jump on not parity/parity odd
   0 1 1 1 1 0 1 1 disp
Timing (clocks): Jump is taken 8
                Jump is not taken 4

JNO = Jump on not overflow
   0 1 1 1 0 0 0 1 disp
Timing (clocks): Jump is taken 8
                Jump is not taken 4

JNS = Jump on not sign
   0 1 1 1 1 0 0 1 disp
Timing (clocks): Jump is taken 8
                Jump is not taken 4

LOOP = Loop CX times
   1 1 1 0 0 0 1 0 disp
Timing (clocks): Jump is taken 9
                Jump is not taken 5

LOOPZ/LOope = Loop while zero/equal
   1 1 1 0 0 0 0 1 disp
Timing (clocks): Jump is taken 11
                Jump is not taken 5

LOOPNZ/LOope = Loop while not zero/ not equal
   1 1 1 0 0 0 0 0 disp
Timing (clocks): Jump is taken 11
                Jump is not taken 5
**JcXZ = Jump on CX zero**

111 000 11

Timing (clocks):  
Jump is taken 9  
Jump is not taken 5

---

**8086 Conditional Transfer Operations**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Condition</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>JE or JZ</td>
<td>ZF = 1</td>
<td>“equal” or “zero”</td>
</tr>
<tr>
<td>JL or JNGE</td>
<td>(SF xor OF) = 1</td>
<td>“less” or “not greater or equal”</td>
</tr>
<tr>
<td>JLE or JNG</td>
<td>(SF xor OF) or ZF = 1</td>
<td>“less or equal” or “not greater”</td>
</tr>
<tr>
<td>JB or JNAE</td>
<td>CF = 1</td>
<td>“below” or “not above or equal”</td>
</tr>
<tr>
<td>JBE or JNA</td>
<td>(CF or ZF) = 1</td>
<td>“below or equal” or “not above”</td>
</tr>
<tr>
<td>JP or JPE</td>
<td>PF = 1</td>
<td>“parity” or “parity even”</td>
</tr>
<tr>
<td>JG</td>
<td>OF = 1</td>
<td>“overflow”</td>
</tr>
<tr>
<td>JS</td>
<td>SF = 1</td>
<td>“sign:”</td>
</tr>
<tr>
<td>JNE or JNZ</td>
<td>ZF = 0</td>
<td>“not equal” or “not zero”</td>
</tr>
<tr>
<td>JNL or JGE</td>
<td>(SF xor OF) = 0</td>
<td>“not less” or “greater or equal”</td>
</tr>
<tr>
<td>JNLE or JG</td>
<td>(SF xor OF) or ZF = 0</td>
<td>“not less or equal” or “greater”</td>
</tr>
<tr>
<td>JNB or JAE</td>
<td>CF = 0</td>
<td>“not below” or “above or equal”</td>
</tr>
<tr>
<td>JNBE or JA</td>
<td>(CF or ZF) = 0</td>
<td>“not below or equal” or “above”</td>
</tr>
<tr>
<td>JNP or JPO</td>
<td>PF = 0</td>
<td>“not parity” or “parity odd”</td>
</tr>
<tr>
<td>JNO</td>
<td>OF = 0</td>
<td>“not overflow”</td>
</tr>
<tr>
<td>JNS</td>
<td>SF = 0</td>
<td>“not sign”</td>
</tr>
</tbody>
</table>

---

**Processor Control**

- **CLC = Clear carry**
  
  111 110 00
  
  Timing: 2 clocks

- **STC = Set carry**
  
  111 110 01
  
  Timing: 2 clocks

- **CMC = Complement carry**
  
  111 101 01
  
  Timing: 2 clocks

- **NOP = No operation**
  
  111 110 00
  
  Timing: 3 clocks

- **STD = Set direction**
  
  111 111 100
  
  Timing: 2 clocks

- **CLI = Clear interrupt**
  
  111 110 10
  
  Timing: 2 clocks

- **STI = Set interrupt**
  
  111 111 11
  
  Timing: 2 clocks

- **HLT = Halt**
  
  111 110 100
  
  Timing: 2 clocks

- **WAIT = Wait**
  
  111 110 11
  
  Timing: 3 clocks

- **LOCK = Bus lock prefix**
  
  111 100 00
  
  Timing: 2 clocks

- **ESC = Escape (to external device)**
  
  110 111 111
  
  Timing: 7+EA clocks

---

Footnotes:

if d = 1 then “to”; if d = 0 then “from”
if w = 1 then word instruction; if w = 0 then byte instruction
if s:w = 01 then 16 bits of immediate data form the operand
if s:w = 11 then an immediate data byte is sign extended to form the
16-bit operand
if v = 0 then “count” = 1; if v = 1 then “count” in (CL)
x = don’t care
z is used for some string primitives to compare with ZF FLAG
AL = 8-bit accumulator
AX = 16-bit accumulator
CX = Count register
DS = Data segment
DX = Variable port register
ES = Extra segment
Above/below refers to unsigned value
Greater = more positive;
Less = less positive (more negative) signed values
See page 1 for Operand Summary.
See page 2 for Segment Override Summary.