# 6. INSTRUCTION SET

This section summarizes the 6x86 CPU instruction set and provides detailed information on the instruction encodings. All instructions are listed in the CPU Instruction Set Summary Table (Table 6-20, Page 6-14), and the FPU Instruction Set Summary Table (Table 6-22, Page 6-30). These tables provide information on the instruction encoding, and the instruction clock counts for each instruction. The clock count values for both tables are based on the assumptions described in Section 6.3.

## 6.1 Instruction Set Summary

Depending on the instruction, the 6x86 CPU instructions follow the general instruction format shown in Figure 6-1. These instructions vary in length and can start at any byte address. An instruction consists of one or more bytes that can include: prefix byte(s), at least one opcode byte(s), mod r/m byte, s-i-b byte, address displacement byte(s) and immediate data byte(s). An instruction can be as short as one byte and as long as 15 bytes. If there are more than 15 bytes in the instruction a general protection fault (error code of 0) is generated.

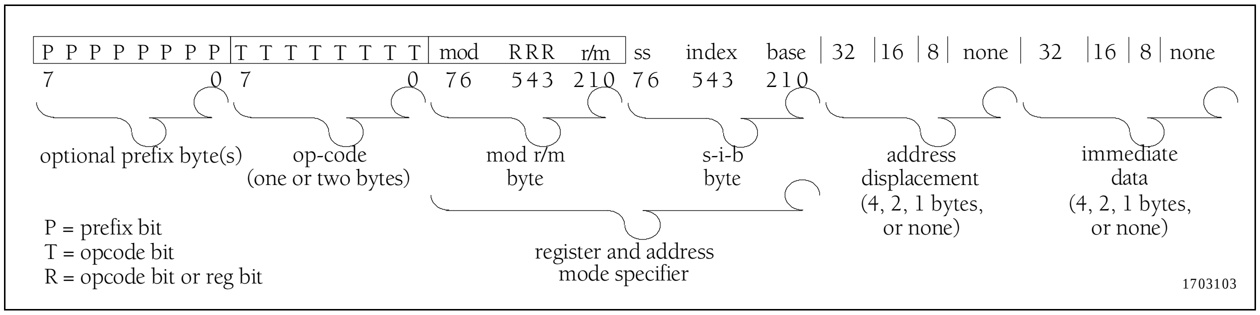

**Figure 6-1. Instruction Set Format**

## 6.2 General Instruction Fields

The fields in the general instruction format at the byte level are listed in Table 6-1.

**Table 6-1. Instruction Fields**

| FIELD NAME | DESCRIPTION | WIDTH |
|---|---|---|
| Optional Prefix Byte(s) | Specifies segment register override, address and operand size, repeat elements in string instruction, LOCK# assertion. | 1 or more bytes |
| Opcode Byte(s) | Identifies instruction operation. | 1 or 2 bytes |
| mod and r/m Byte | Address mode specifier. | 1 byte |
| s-i-b Byte | Scale factor, Index and Base fields. | 1 byte |
| Address Displacement | Address displacement operand. | 1, 2 or 4 bytes |
| Immediate data | Immediate data operand. | 1, 2 or 4 bytes |

### 6.2.1 Optional Prefix Bytes

Prefix bytes can be placed in front of any instruction. The prefix modifies the operation of the next instruction only. When more than one prefix is used, the order is not important. There are five type of prefixes as follows:

1. Segment Override explicitly specifies which segment register an instruction will use for effective address calculation.
2. Address Size switches between 16- and 32-bit addressing. Selects the inverse of the default.
3. Operand Size switches between 16- and 32-bit operand size. Selects the inverse of the default.
4. Repeat is used with a string instruction which causes the instruction to be repeated for each element of the string.
5. Lock is used to assert the hardware LOCK# signal during execution of the instruction.

Table 6-2 lists the encodings for each of the available prefix bytes.

**Table 6-2. Instruction Prefix Summary**

| PREFIX | ENCODING | DESCRIPTION |
|---|---|---|
| ES: | 26h | Override segment default, use ES for memory operand |
| CS: | 2Eh | Override segment default, use CS for memory operand |
| SS: | 36h | Override segment default, use SS for memory operand |
| DS: | 3Eh | Override segment default, use DS for memory operand |
| FS: | 64h | Override segment default, use FS for memory operand |
| GS: | 65h | Override segment default, use GS for memory operand |
| Operand Size | 66h | Make operand size attribute the inverse of the default |
| Address Size | 67h | Make address size attribute the inverse of the default |
| LOCK | F0h | Assert LOCK# hardware signal. |
| REPNE | F2h | Repeat the following string instruction. |
| REP/REPE | F3h | Repeat the following string instruction. |

## 6.2.2 Opcode Byte

The opcode field specifies the operation to be performed by the instruction. The opcode field is either one or two bytes in length and may be further defined by additional bits in the mod r/m byte. Some operations have more than one opcode, each specifying a different form of the operation. Some opcodes name instruction groups. For example, opcode 80h names a group of operations that have an immediate operand and a register or memory operand. The reg field may appear in the second opcode byte or in the mod r/m byte.

### 6.2.2.1 w Field

The 1-bit w field (Table 6-11) selects the operand size during 16 and 32 bit data operations.

**Table 6-3. w Field Encoding**

| w FIELD | OPERAND SIZE | |
|---|---|---|
| | 16-BIT DATA OPERATIONS | 32-BIT DATA OPERATIONS |
| 0 | 8 Bits | 8 Bits |
| 1 | 16 Bits | 32 Bits |

### 6.2.2.2 d Field

The d field (Table 6-4) determines which operand is taken as the source operand and which operand is taken as the destination.

**Table 6-4. d Field Encoding**

| d FIELD | DIRECTION OF OPERATON | SOURCE OPERAND | DESTINATION OPERAND |
|---|---|---|---|
| 0 | Register --> Register or<br>Register --> Memory | reg | mod r/m or<br>mod ss-index-base |
| 1 | Register --> Register or<br>Memory --> Register | mod r/m or<br>mod ss-index-base | reg |

### 6.2.2.3 s Field

The s field (Table 6-5) determines the size of the immediate data field. If the S bit is set, the immediate field of the OP code is 8-bits wide and is sign extened to match the operand size of the opcode.

**Table 6-5. s Field Encoding**

| s FIELD | Immediate Field Size | | |
|---|---|---|---|
| | 8-Bit Operand Size | 16-Bit Operand Size | 32-Bit Operand Size |
| 0 (or not present) | 8 bits | 16 bits | 32 bits |
| 1 | 8 bits | 8 bits (sign extended) | 8 bits (sign extended) |

### 6.2.2.4 eee Field

The eee field (Table 6-6) is used to select the control, debug and test registers in the MOV instructions. The type of register and base registers selected by the eee field are listed in Table 6-6. The values shown in Table 6-6 are the only valid encodings for the eee bits.

**Table 6-6. eee Field Encoding**

| eee FILED | REGISTER TYPE | BASE REGISTER |
|---|---|---|
| 000 | Control Register | CR0 |
| 010 | Control Register | CR2 |
| 011 | Control Register | CR3 |
| 000 | Debug Register | DR0 |
| 001 | Debug Register | DR1 |
| 010 | Debug Register | DR2 |
| 011 | Debug Register | DR3 |
| 110 | Debug Register | DR6 |
| 111 | Debug Register | DR7 |
| 011 | Test Register | TR3 |
| 100 | Test Register | TR4 |
| 101 | Test Register | TR5 |
| 110 | Test Register | TR6 |
| 111 | Test Register | TR7 |

### 6.2.3 mod and r/m Byte

The mod and r/m fields (Table 6-7), within the mod r/m byte, select the type of memory addressing to be used. Some instructions use a fixed addressing mode (e.g., PUSH or POP) and therefore, these fields are not present. Table 6-7 lists the addressing method when 16-bit addressing is used and a mod r/m byte is present. Some mod r/m field encodings are dependent on the w field and are shown in Table 6-8 (Page 6-7).

**Table 6-7. mod r/m Field Encoding**

| mod and r/m fields | 16-BIT ADDRESS MODE with mod r/m Byte | 32-BIT ADDRESS MODE with mod r/m Byte and No s-i-b Byte Present |
|---|---|---|
| 00 000 | DS:[BX+SI] | DS:[EAX] |
| 00 001 | DS:[BX+DI] | DS:[ECX] |
| 00 010 | DS:[BP+SI] | DS:[EDX] |
| 00 011 | DS:[BP+DI] | DS:[EBX] |
| 00 100 | DS:[SI] | s-i-b is present (See 6.2.4 (Page 6-9)) |
| 00 101 | DS:[DI] | DS:[d32] |
| 00 110 | DS:[d16] | DS:[ESI] |
| 00 111 | DS:[BX] | DS:[EDI] |
| | | |
| 01 000 | DS:[BX+SI+d8] | DS:[EAX+d8] |
| 01 001 | DS:[BX+DI+d8] | DS:[ECX+d8] |
| 01 010 | DS:[BP+SI+d8] | DS:[EDX+d8] |
| 01 011 | DS:[BP+DI+d8] | DS:[EBX+d8] |
| 01 100 | DS:[SI+d8] | s-i-b is present (See 6.2.4 (Page 6-9)) |
| 01 101 | DS:[DI+d8] | SS:[EBP+d8] |
| 01 110 | SS:[BP+d8] | DS:[ESI+d8] |
| 01 111 | DS:[BX+d8] | DS:[EDI+d8] |
| | | |
| 10 000 | DS:[BX+SI+d16] | DS:[EAX+d32] |
| 10 001 | DS:[BX+DI+d16] | DS:[ECX+d32] |
| 10 010 | DS:[BP+SI+d16] | DS:[EDX+d32] |
| 10 011 | DS:[BP+DI+d16] | DS:[EBX+d32] |
| 10 100 | DS:[SI+d16] | s-i-b is present (See 6.2.4 (Page 6-9)) |
| 10 101 | DS:[DI+d16] | SS:[EBP+d32] |
| 10 110 | SS:[BP+d16] | DS:[ESI+d32] |
| 10 111 | DS:[BX+d16] | DS:[EDI+d32] |
| | | |
| 11 000-11 111 | See Table 6-7 | See Table 6-7 |

**Table 6-8. mod r/m Field Encoding Dependent on w Field**

| mod r/m | 16-BIT OPERATION w = 0 | 16-BIT OPERATION w = 1 | 32-BIT OPERATION w = 0 | 32-BIT OPERATION w = 1 |
|---|---|---|---|---|
| 11 000 | AL | AX | AL | EAX |
| 11 001 | CL | CX | CL | ECX |
| 11 010 | DL | DX | DL | EDX |
| 11 011 | BL | BX | BL | EBX |
| 11 100 | AH | SP | AH | ESP |
| 11 101 | CH | BP | CH | EBP |
| 11 110 | DH | SI | DH | ESI |
| 11 111 | BH | DI | BH | EDI |

## 6.2.3.1 reg Field

The reg field (Table 6-9) determines which general registers are to be used. The selected register is dependent on whether a 16 or 32 bit operation is current and the status of the w bit.

**Table 6-9. reg Field**

| reg | 16-BIT OPERATION w Field Not Present | 32-BIT OPERATION w Field Not Present | 16-BIT OPERATION w = 0 | 16-BIT OPERATION w = 1 | 32-BIT OPERATION w = 0 | 32-BIT OPERATION w = 1 |
|---|---|---|---|---|---|---|
| 000 | AX | EAX | AL | AX | AL | EAX |
| 001 | CX | ECX | CL | CX | CL | ECX |
| 010 | DX | EDX | DL | DX | DL | EDX |
| 011 | BX | EBX | BL | BX | BL | EBX |
| 100 | SP | ESP | AH | SP | AH | ESP |
| 101 | BP | EBP | CH | BP | CH | EBP |
| 110 | SI | ESI | DH | SI | DH | ESI |
| 111 | DI | EDI | BH | DI | BH | EDI |

### 6.2.3.2 sreg3 Field

The sreg3 field (Table 6-10) is 3-bit field that is similar to the sreg2 field, but allows use of the FS and GS segment registers.

**Table 6-10. sreg3 Field Encoding**

| sreg3 FIELD | SEGMENT REGISTER SELECTED |
|---|---|
| 000 | ES |
| 001 | CS |
| 010 | SS |
| 011 | DS |
| 100 | FS |
| 101 | GS |
| 110 | undefined |
| 111 | undefined |

### 6.2.3.3 sreg2 Field

The sreg2 field (Table 6-11) is a 2-bit field that allows one of the four 286-type segment registers to be specified.

**Table 6-11. sreg2 Field Encoding**

| sreg2 FIELD | SEGMENT REGISTER SELECTED |
|---|---|
| 00 | ES |
| 01 | CS |
| 10 | SS |
| 11 | DS |

### 6.2.4 s-i-b Byte

The s-i-b fields provide scale factor, indexing and a base field for address selection.

#### 6.2.4.1 ss Field

The ss field (Table 6-12) specifies the scale factor used in the offset mechanism for address calculation. The scale factor multiplies the index value to provide one of the components used to calculate the offset address.

**Table 6-12. ss Field Encoding**

| ss FIELD | SCALE FACTOR |
|---|---|
| 00 | x1 |
| 01 | x2 |
| 01 | x4 |
| 11 | x8 |

#### 6.2.4.2 index Field

The index field (Table 6-13) specifies the index register used by the offset mechanism for offset address calculation. When no index register is used (index field = 100), the ss value must be 00 or the effective address is undefined.

**Table 6-13. index Field Encoding**

| Index FIELD | INDEX REGISTER |
|---|---|
| 000 | EAX |
| 001 | ECX |
| 010 | EDX |
| 011 | EBX |
| 100 | none |
| 101 | EBP |
| 110 | ESI |
| 111 | EDI |

#### 6.2.4.3 Base Field

In Table 6-7 (Page 6-6), the note “s-i-b present” for certain entries forces the use of the mod and base field as listed in Table 6-14. The first two digits in the first column of Table 6-14 identifies the mod bits in the mod r/m byte. The last three digits in the first column of this table identifies the base fields in the s-i-b byte.

**Table 6-14. mod base Field Encoding**

| mod FIELD WITHIN mode/rm BYTE | base FIELD WITHIN s-i-b BYTE | 32-BIT ADDRESS MODE with mod r/m and s-i-b Bytes Present |
|---|---|---|
| 00 | 000 | DS:[EAX+(scaled index)] |
| 00 | 001 | DS:[ECX+(scaled index)] |
| 00 | 010 | DS:[EDX+(scaled index)] |
| 00 | 011 | DS:[EBX+(scaled index)] |
| 00 | 100 | SS:[ESP+(scaled index)] |
| 00 | 101 | DS:[d32+(scaled index)] |
| 00 | 110 | DS:[ESI+(scaled index)] |
| 00 | 111 | DS:[EDI+(scaled index)] |
| | | |
| 01 | 000 | DS:[EAX+(scaled index)+d8] |
| 01 | 001 | DS:[ECX+(scaled index)+d8] |
| 01 | 010 | DS:[EDX+(scaled index)+d8] |
| 01 | 011 | DS:[EBX+(scaled index)+d8] |
| 01 | 100 | SS:[ESP+(scaled index)+d8] |
| 01 | 101 | SS:[EBP+(scaled index)+d8] |
| 01 | 110 | DS:[ESI+(scaled index)+d8] |
| 01 | 111 | DS:[EDI+(scaled index)+d8] |
| | | |
| 10 | 000 | DS:[EAX+(scaled index)+d32] |
| 10 | 001 | DS:[ECX+(scaled index)+d32] |
| 10 | 010 | DS:[EDX+(scaled index)+d32] |
| 10 | 011 | DS:[EBX+(scaled index)+d32] |
| 10 | 100 | SS:[ESP+(scaled index)+d32] |
| 10 | 101 | SS:[EBP+(scaled index)+d32] |
| 10 | 110 | DS:[ESI+(scaled index)+d32] |
| 10 | 111 | DS:[EDI+(scaled index)+d32] |

## 6.3 CPUID Instruction

The 6x86 CPU executes the CPUID instruction (opcode 0FA2) as documented in this section only if the CPUID bit in the CCR4 configuration register is set. The CPUID instruction may be used by software to determine the vendor and type of CPU.

When the CPUID instruction is executed with EAX = 0, the ASCII characters “CyrixInstead” are placed in the EBX, EDX, and ECX registers as shown in Table 6-15:

**Table 6-15. CPUID Data Returned When EAX = 0**

| REGISTER | CONTENTS (D31 - D0) |
|---|---|
| EBX | 69 72 79 43<br>i r y C* |
| EDX | 73 6E 49 78<br>s n I x* |
| ECX | 64 61 65 74<br>d a e t* |

*ASCII equivalent

When the CPUID instruction is executed with EAX = 1, EAX and EDX contain the values shown in Table 6-16.

**Table 6-16. CPUID Data Returned When EAX = 1**

| REGISTER | CONTENTS |
|---|---|
| EAX(3-0) | 0 |
| EAX(7-4) | 3 |
| EAX(11-8) | 5 |
| EAX(13-12) | 0 |
| EAX(31-14) | reserved |
| EDX | If EDX = 00, FPU not on-chip.<br>If EDX = 01, FPU on-chip. |

## 6.4 Instruction Set Tables

The 6x86 CPU instruction set is presented in two tables: Table 6-20. “6x86 CPU Instruction Set Clock Count Summary” on page 6-14 and Table 6-22. “6x86 FPU Instruction Set Summary” on page 6-30. Additional information concerning the FPU Instruction Set is presented on page 6-29.

### 6.4.1 Assumptions Made in Determining Instruction Clock Count

The assumptions made in determining instruction clock counts are listed below:

1. All clock counts refer to the internal CPU internal clock frequency. For example, the clock counts for a clock-doubled 6x86 CPU-100 refer to 100 MHz clocks while the external clock is 50 MHz.
2. The instruction has been prefetched, decoded and is ready for execution.
3. Bus cycles do not require wait states.
4. There are no local bus HOLD requests delaying processor access to the bus.
5. No exceptions are detected during instruction execution.
6. If an effective address is calculated, it does not use two general register components. One register, scaling and displacement can be used within the clock count shown. However, if the effective address calculation uses two general register components, add 1 clock to the clock count shown.
7. All clock counts assume aligned 32-bit memory/IO operands.
8. If instructions access a 32-bit operand that crosses a 64-bit boundary, add 1 clock for read or write and add 2 clocks for read and write.
9. For non-cached memory accesses, add two clocks (6x86 CPU with 2x clock) or four clocks (6x86 CPU with 3x clock). (Assumes zero wait state memory accesses).
10. Locked cycles are not cacheable. Therefore, using the LOCK prefix with an instruction adds additional clocks as specified in paragraph 9 above.
11. No parallel execution of instructions.

### 6.4.2 CPU Instruction Set Summary Table Abbreviations

The clock counts listed in the CPU Instruction Set Summary Table are grouped by operating mode and whether there is a register/cache hit or a cache miss. In some cases, more than one clock count is shown in a column for a given instruction, or a variable is used in the clock count. The abbreviations used for these conditions are listed in Table 6-17.

**Table 6-17. CPU Clock Count Abbreviations**

| CLOCK COUNT SYMBOL | EXPLANATION |
|---|---|
| / | Register operand/memory operand. |
| n | Number of times operation is repeated. |
| L | Level of the stack frame. |
| \| | Conditional jump taken \| Conditional jump not taken. (e.g. "4\|1" = 4 clocks if jump taken, 1 clock if jump not taken) |
| \ | CPL ≤ IOPL \ CPL > IOPL (where CPL = Current Privilege Level, IOPL = I/O Privilege Level) |
| m | Number of parameters passed on the stack. |

## 6.4.3 CPU Instruction Set Summary Table Flags Table

The CPU Instruction Set Summary Table lists nine flags that are affected by the execution of instructions. The conventions shown in Table 6-18 are used to identify the different flags. Table 6-19 lists the conventions used to indicate what action the instruction has on the particular flag.

**Table 6-18. Flag Abbreviations**

| ABBREVIATION | NAME OF FLAG |
|---|---|
| OF | Overflow Flag |
| DF | Direction Flag |
| IF | Interrupt Enable Flag |
| TF | Trap Flag |
| SF | Sign Flag |
| ZF | Zero Flag |
| AF | Auxiliary Flag |
| PF | Parity Flag |
| CF | Carry Flag |

**Table 6-19. Action of Instruction on Flag**

| INSTRUCTION TABLE SYMBOL | ACTION |
|---|---|
| x | Flag is modified by the instruction. |
| - | Flag is not changed by the instruction. |
| 0 | Flag is reset to "0". |
| 1 | Flag is set to "1". |
| u | Flag is undefined following execution of the instruction. |

## Table 6-20. 6x86 CPU Instruction Set Clock Count Summary

| INSTRUCTION | OPCODE | FLAGS | REAL MODE CLOCK COUNT | PROTECTED MODE CLOCK COUNT | NOTES | |
|---|---|---|---|---|---|---|
| | | OF DF IF TF SF ZF AF PF CF | Reg/ Cache Hit | Reg/ Cache Hit | Real Mode | Protected Mode |
| **AAA** *ASCII Adjust AL after Add* | 37 | u - - - u u x u x | 7 | 7 | | |
| **AAD** *ASCII Adjust AX before Divide* | D5 0A | u - - - x x u x u | 7 | 7 | | |
| **AAM** ASCII *Adjust AX after Multiply* | D4 0A | u - - - x x u x u | 13-21 | 13-21 | | |
| **AAS** ASCII *Adjust AL after Subtract* | 3F | u - - - u u x u x | 7 | 7 | | |
| **ADC** *Add with Carry*<br>Register to Register<br>Register to Memory<br>Memory to Register<br>Immediate to Register/Memory<br>Immediate to Accumulator | <br>1 [00dw] [11 reg r/m]<br>1 [000w] [mod reg r/m]<br>1 [001w] [mod reg r/m]<br>8 [00sw] [mod 010 r/m]###<br>1 [010w] ### | x - - - x x x x x | <br>1<br>1<br>1<br>1<br>1 | <br>1<br>1<br>1<br>1<br>1 | b | h |
| **ADD** *Integer Add*<br>Register to Register<br>Register to Memory<br>Memory to Register<br>Immediate to Register/Memory<br>Immediate to Accumulator | <br>0 [00dw] [11 reg r/m]<br>0 [000w] [mod reg r/m]<br>0 [001w] [mod reg r/m]<br>8 [00sw] [mod 000 r/m]###<br>0 [010w] ### | x - - - x x x x x | <br>1<br>1<br>1<br>1<br>1 | <br>1<br>1<br>1<br>1<br>1 | b | h |
| **AND** *Boolean AND*<br>Register to Register<br>Register to Memory<br>Memory to Register<br>Immediate to Register/Memory<br>Immediate to Accumulator | <br>2 [00dw] [11 reg r/m]<br>2 [000w] [mod reg r/m]<br>2 [001w] [mod reg r/m]<br>8 [00sw] [mod 100 r/m]###<br>2 [010w] ### | 0 - - - x x u x 0 | <br>1<br>1<br>1<br>1<br>1 | <br>1<br>1<br>1<br>1<br>1 | b | h |
| **ARPL** *Adjust Requested Privilege Level*<br>From Register/Memory | <br>63 [mod reg r/m] | - - - - - x - - - | | 9 | a | h |
| **BOUND** *Check Array Boundaries*<br>If Out of Range (Int 5)<br>If In Range | 62 [mod reg r/m] | - - - - - - - - - | <br>20<br>11 | <br>20+INT<br>11 | b, e | g,h,j,k,r |
| **BSF** *Scan Bit Forward*<br>Register, Register/Memory | <br>0F BC [mod reg r/m] | - - - - - x - - - | 3 | 3 | b | h |
| **BSR** *Scan Bit Reverse*<br>Register, Register/Memory | <br>0F BD [mod reg r/m] | - - - - - x - - - | 3 | 3 | b | h |
| **BSWAP** *Byte Swap* | 0F C[1 reg] | - - - - - - - - - | 4 | 4 | | |
| **BT** *Test Bit*<br>Register/Memory, Immediate<br>Register/Memory, Register | <br>0F BA [mod 100 r/m]#<br>0F A3 [mod reg r/m] | - - - - - - - - x | <br>2<br>5/6 | <br>2<br>5/6 | b | h |
| **BTC** *Test Bit and Complement*<br>Register/Memory, Immediate<br>Register/Memory, Register | <br>0F BA [mod 111 r/m]#<br>0F BB [mod reg r/m] | - - - - - - - - x | <br>3<br>5/6 | <br>3<br>5/6 | b | h |

# = immediate 8-bit data
## = immediate 16-bit data
### = full immediate 32-bit data (8, 16, 32 bits)

+ = 8-bit signed displacement
+++ = full signed displacement (16, 32 bits)

x = modified
- = unchanged
u = undefined

## Table 6-20. 6x86 CPU Instruction Set Clock Count Summary (Continued)

| INSTRUCTION | OPCODE | FLAGS | REAL MODE CLOCK COUNT | PROTECTED MODE CLOCK COUNT | NOTES | |
|---|---|---|---|---|---|---|
| | | OF DF IF TF SF ZF AF PF CF | Reg/ Cache Hit | Reg/ Cache Hit | Real Mode | Protected Mode |
| **BTR** *Test Bit and Reset* | | - - - - - - - - - - x | | | b | h |
| Register/Memory, Immediate | 0F BA [mod 110 r/m]# | | 3 | 3 | | |
| Register/Memory, Register | 0F B3 [mod reg r/m] | | 5/6 | 5/6 | | |
| **BTS** *Test Bit and Set* | | - - - - - - - - - - x | | | b | h |
| Register/Memory | 0F BA [mod 101 r/m] | | 3 | 3 | | |
| Register (short form) | 0F AB [mod reg r/m] | | 5/6 | 5/6 | | |
| **CALL** *Subroutine Call* | | - - - - - - - - - - - | | | b | h,j,k,r |
| Direct Within Segment | E8 +++ | | 1 | 1 | | |
| Register/Memory Indirect Within Segment | FF [mod 010 r/m] | | 1/3 | 1/3 | | |
| Direct Intersegment | 9A [unsigned full offset, | | 3 | 4 | | |
| Call Gate to Same Privilege | selector] | | | 15 | | |
| Call Gate to Different Privilege No Parameters | | | | 26 | | |
| Call Gate to Different Privilege m Par's | | | | 35+2m | | |
| 16-bit Task to 16-bit TSS | | | | 110 | | |
| 16-bit Task to 32-bit TSS | | | | 118 | | |
| 16-bit Task to V86 Task | | | | 96 | | |
| 32-bit Task to 16-bit TSS | | | | 112 | | |
| 32-bit Task to 32-bit TSS | | | | 120 | | |
| 32-bit Task to V86 Task | | | | 98 | | |
| Indirect Intersegment | FF [mod 011 r/m] | | 5 | 8 | | |
| Call Gate to Same Privilege | | | | 20 | | |
| Call Gate to Different Privilege No Parameters | | | | 31 | | |
| Call Gate to Different Privilege Level m Par's | | | | 40+2m | | |
| 16-bit Task to 16-bit TSS | | | | 114 | | |
| 16-bit Task to 32-bit TSS | | | | 122 | | |
| 16-bit Task to V86 Task | | | | 100 | | |
| 32-bit Task to 16-bit TSS | | | | 116 | | |
| 32-bit Task to 32-bit TSS | | | | 124 | | |
| 32-bit Task to V86 Task | | | | 102 | | |
| **CBW** *Convert Byte to Word* | 98 | - - - - - - - - - - - | 3 | 3 | | |
| **CDQ** *Convert Doubleword to Quadword* | 99 | - - - - - - - - - - - | 2 | 2 | | |
| **CLC** *Clear Carry Flag* | F8 | - - - - - - - - - - 0 | 1 | 1 | | |
| **CLD** *Clear Direction Flag* | FC | - 0 - - - - - - - - - | 7 | 7 | | |
| **CLI** *Clear Interrupt Flag* | FA | - - 0 - - - - - - - - | 7 | 7 | | m |
| **CLTS** *Clear Task Switched Flag* | 0F 06 | - - - - - - - - - - - | 10 | 10 | c | l |
| **CMC** *Complement the Carry Flag* | F5 | - - - - - - - - - - x | 2 | 2 | | |

| | | |
|---|---|---|
| # = immediate 8-bit data | + = 8-bit signed displacement | x = modified |
| ## = immediate 16-bit data | +++ = full signed displacement (16, 32 bits) | - = unchanged |
| ### = full immediate 32-bit data (8, 16, 32 bits) | | u = undefined |

## Table 6-20. 6x86 CPU Instruction Set Clock Count Summary (Continued)

| INSTRUCTION | OPCODE | FLAGS OF DF IF TF SF ZF AF PF CF | REAL MODE CLOCK COUNT Reg/ Cache Hit | PROTECTED MODE CLOCK COUNT Reg/ Cache Hit | NOTES Real Mode | NOTES Protected Mode |
|---|---|---|---|---|---|---|
| **CMP** *Compare Integers* |  | x - - - x x x x x |  |  | b | h |
| Register to Register | 3 [10dw] [11 reg r/m] |  | 1 | 1 |  |  |
| Register to Memory | 3 [101w] [mod reg r/m] |  | 1 | 1 |  |  |
| Memory to Register | 3 [100w] [mod reg r/m] |  | 1 | 1 |  |  |
| Immediate to Register/Memory | 8 [00sw] [mod 111 r/m] ### |  | 1 | 1 |  |  |
| Immediate to Accumulator | 3 [110w] ### |  | 1 | 1 |  |  |
| **CMPS** *Compare String* | A [011w] | x - - - x x x x x | 5 | 5 | b | h |
| **CMPXCHG** *Compare and Exchange* |  | x - - - x x x x x |  |  |  |  |
| Register1, Register2 | 0F B [000w] [11 reg2 reg1] |  | 11 | 11 |  |  |
| Memory, Register | 0F B [000w] [mod reg r/m] |  | 11 | 11 |  |  |
| **CPUID** *CPU Identification* | 0F A2 | - - - - - - - - - | 12 | 12 |  |  |
| **CWD** *Convert Word to Doubleword* | 99 | - - - - - - - - - | 2 | 2 |  |  |
| **CWDE** *Convert Word to Doubleword Extended* | 98 | - - - - - - - - - | 2 | 2 |  |  |
| **DAA** *Decimal Adjust AL after Add* | 27 | - - - - x x x x x | 9 | 9 |  |  |
| **DAS** *Decimal Adjust AL after Subtract* | 2F | - - - - x x x x x | 9 | 9 |  |  |
| **DEC** *Decrement by 1* |  | x - - - x x x x - |  |  | b | h |
| Register/Memory | F [111w] [mod 001 r/m] |  | 1 | 1 |  |  |
| Register (short form) | 4 [1 reg] |  | 1 | 1 |  |  |
| **DIV** *Unsigned Divide* | F [011w] [mod 110 r/m] | - - - - x x u u - |  |  | b,e | e,h |
| Accumulator by Register/Memory |  |  |  |  |  |  |
| Divisor: Byte |  |  | 13-17 | 13-17 |  |  |
| Word |  |  | 13-25 | 13-25 |  |  |
| Doubleword |  |  | 13-41 | 13-41 |  |  |
| **ENTER** *Enter New Stack Frame* | C8 ##,# | - - - - - - - - - |  |  | b | h |
| Level = 0 |  |  | 10 | 10 |  |  |
| Level = 1 |  |  | 13 | 13 |  |  |
| Level (L) > 1 |  |  | 10+L*3 | 10+L*3 |  |  |
| **HLT** Halt | F4 | - - - - - - - - - | 5 | 5 |  | l |
| **IDIV** *Integer (Signed) Divide* |  | - - - - x x u u - |  |  | b,e | e,h |
| Accumulator by Register/Memory | F [011w] [mod 111 r/m] |  |  |  |  |  |
| Divisor: Byte |  |  | 16-20 | 16-20 |  |  |
| Word |  |  | 16-28 | 16-28 |  |  |
| Doubleword |  |  | 17-45 | 17-45 |  |  |

| | | |
|---|---|---|
| # = immediate 8-bit data | + = 8-bit signed displacement | x = modified |
| ## = immediate 16-bit data | +++ = full signed displacement (16, 32 bits) | - = unchanged |
| ### = full immediate 32-bit data (8, 16, 32 bits) | | u = undefined |

**Table 6-20. 6x86 CPU Instruction Set Clock Count Summary (Continued)**

| INSTRUCTION | OPCODE | FLAGS | REAL MODE CLOCK COUNT | PROTECTED MODE CLOCK COUNT | NOTES | |
|---|---|---|---|---|---|---|
| | | OF DF IF TF SF ZF AF PF CF | Reg/ Cache Hit | Reg/ Cache Hit | Real Mode | Protected Mode |
| **IMUL** *Integer (Signed) Multiply* | | x - - - x x u u x | | | b | h |
| Accumulator by Register/Memory | F [011w] [mod 101 r/m] | | | | | |
| Multiplier: Byte | | | 4 | 4 | | |
| Word | | | 4 | 4 | | |
| Doubleword | | | 10 | 10 | | |
| Register with Register/Memory | 0F AF [mod reg r/m] | | | | | |
| Multiplier: Word | | | 4 | 4 | | |
| Doubleword | | | 10 | 10 | | |
| Register/Memory with Immediate to Register2 | | | | | | |
| Multiplier: Word | 6 [10s1] [mod reg r/m] ### | | 5 | 5 | | |
| Doubleword | | | 11 | 11 | | |
| **IN** *Input from I/O Port* | | - - - - - - - - - | | | | m |
| Fixed Port | E [010w] [#] | | 14 | 14/28 | | |
| Variable Port | E [110w] | | 14 | 14/28 | | |
| **INC** *Increment by 1* | | x - - - x x x x - | | | b | h |
| Register/Memory | F [111w] [mod 000 r/m] | | 1 | 1 | | |
| Register (short form) | 4 [0 reg] | | 1 | 1 | | |
| **INS** *Input String from I/O Port* | 6 [110w] | - - - - - - - - - | 14 | 14/28 | b | h,m |
| **INT** *Software Interrupt* | | - - x 0 - - - - - | | | b,e | g,j,k,r |
| INT i | CD # | | 9 | | | |
| Protected Mode: | | | | | | |
| Interrupt or Trap to Same Privilege | | | | 21 | | |
| Interrupt or Trap to Different Privilege | | | | 32 | | |
| 16-bit Task to 16-bit TSS by Task Gate | | | | 114 | | |
| 16-bit Task to 32-bit TSS by Task Gate | | | | 122 | | |
| 16-bit Task to V86 by Task Gate | | | | 100 | | |
| 16-bit Task to 16-bit TSS by Task Gate | | | | 116 | | |
| 32-bit Task to 32-bit TSS by Task Gate | | | | 124 | | |
| 32-bit Task to V86 by Task Gate | | | | 102 | | |
| V86 to 16-bit TSS by Task Gate | | | | 124 | | |
| V86 to 32-bit TSS by Task Gate | | | | 102 | | |
| V86 to Privilege 0 by Trap Gate/Int Gate | | | | 46 | | |
| **Continued on the next page...** | | | | | | |

| | | |
|---|---|---|
| # = immediate 8-bit data | + = 8-bit signed displacement | x = modified |
| ## = immediate 16-bit data | +++ = full signed displacement (16, 32 bits) | - = unchanged |
| ### = full immediate 32-bit data (8, 16, 32 bits) | | u = undefined |

**Table 6-20. 6x86 CPU Instruction Set Clock Count Summary (Continued)**

| INSTRUCTION | OPCODE | FLAGS | REAL MODE CLOCK COUNT | PROTECTED MODE CLOCK COUNT | NOTES | |
|---|---|---|---|---|---|---|
| | | OF DF IF TF SF ZF AF PF CF | Reg/ Cache Hit | Reg/ Cache Hit | Real Mode | Protected Mode |
| **INT** *Software Interrupt* **(Continued)** | | - - x 0 - - - - - | | | b,e | g,j,k,r |
| INT 3 | CC | | INT | INT | | |
| INTO | CE | | | | | |
| If OF==0 | | | 6 | 6 | | |
| If OF==1 (INT 4) | | | | 15+INT | | |
| **INVD** *Invalidate Cache* | 0F 08 | - - - - - - - - - | 12 | 12 | t | t |
| **INVLPG** *Invalidate TLB Entry* | 0F 01 [mod 111 r/m] | - - - - - - - - - | 13 | 13 | | |
| **IRET** *Interrupt Return* | CF | x x x x x x x x x | | | | g,h,j,k,r |
| Real Mode | | | 7 | | | |
| Protected Mode: | | | | | | |
| Within Task to Same Privilege | | | | 10 | | |
| Within Task to Different Privilege | | | | 26 | | |
| 16-bit Task to 16-bit Task | | | | 117 | | |
| 16-bit Task to 32-bit TSS | | | | 125 | | |
| 16-bit Task to V86 Task | | | | 103 | | |
| 32-bit Task to 16-bit TSS | | | | 119 | | |
| 32-bit Task to 32-bit TSS | | | | 127 | | |
| 32-bit Task to V86 Task | | | | 105 | | |
| **JB/JNAE/JC** *Jump on Below/Not Above or Equal/ Carry* | | - - - - - - - - - | | | | r |
| 8-bit Displacement | 72 + | | 1 | 1 | | |
| Full Displacement | 0F 82 +++ | | 1 | 1 | | |
| **JBE/JNA** *Jump on Below or Equal/Not Above* | | - - - - - - - - - | | | | r |
| 8-bit Displacement | 76 + | | 1 | 1 | | |
| Full Displacement | 0F 86 +++ | | 1 | 1 | | |
| **JCXZ/JECXZ** *Jump on CX/ECX Zero* | E3 + | - - - - - - - - - | 1 | 1 | | r |
| **JE/JZ** *Jump on Equal/Zero* | | - - - - - - - - - | | | | r |
| 8-bit Displacement | 74 + | | 1 | 1 | | |
| Full Displacement | 0F 84 +++ | | 1 | 1 | | |
| **JL/JNGE** *Jump on Less/Not Greater or Equal* | | - - - - - - - - - | | | | r |
| 8-bit Displacement | 7C + | | 1 | 1 | | |
| Full Displacement | 0F 8C +++ | | 1 | 1 | | |
| **JLE/JNG** *Jump on Less or Equal/Not Greater* | | - - - - - - - - - | | | | r |
| 8-bit Displacement | 7E + | | 1 | 1 | | |
| Full Displacement | 0F 8E +++ | | 1 | 1 | | |

| | | |
|---|---|---|
| # = immediate 8-bit data | + = 8-bit signed displacement | x = modified |
| ## = immediate 16-bit data | +++ = full signed displacement (16, 32 bits) | - = unchanged |
| ### = full immediate 32-bit data (8, 16, 32 bits) | | u = undefined |

**Table 6-20. 6x86 CPU Instruction Set Clock Count Summary (Continued)**

| INSTRUCTION | OPCODE | FLAGS | REAL MODE CLOCK COUNT | PROTECTED MODE CLOCK COUNT | NOTES | |
|---|---|---|---|---|---|---|
| | | OF DF IF TF SF ZF AF PF CF | Reg/ Cache Hit | Reg/ Cache Hit | Real Mode | Protected Mode |
| **JMP** *Unconditional Jump* | | - - - - - - - - - | | | b | h,j,k,r |
| 8-bit Displacement | EB + | | 1 | 1 | | |
| Full Displacement | E9 +++ | | 1 | 1 | | |
| Register/Memory Indirect Within Segment | FF [mod 100 r/m] | | 1/3 | 1/3 | | |
| Direct Intersegment | EA [unsigned full offset, selector] | | 1 | 4 | | |
| Call Gate Same Privilege Level | | | | 14 | | |
| 16-bit Task to 16-bit TSS | | | | 110 | | |
| 16-bit Task to 32-bit TSS | | | | 118 | | |
| 16-bit Task to V86 Task | | | | 96 | | |
| 32-bit Task to 16-bit TSS | | | | 112 | | |
| 32-bit Task to 32-bit TSS | | | | 120 | | |
| 32-bit Task to V86 Task | | | | 98 | | |
| Indirect Intersegment | FF [mod 101 r/m] | | 5 | 7 | | |
| Call Gate Same Privilege Level | | | | 17 | | |
| 16-bit Task to 16-bit TSS | | | | 113 | | |
| 16-bit Task to 32-bit TSS | | | | 121 | | |
| 16-bit Task to V86 Task | | | | 99 | | |
| 32-bit Task to 16-bit TSS | | | | 115 | | |
| 32-bit Task to 32-bit TSS | | | | 123 | | |
| 32-bit Task to V86 Task | | | | 101 | | |
| **JNB/JAE/JNC** *Jump on Not Below/Above or Equal/ Not Carry* | | - - - - - - - - - | | | | r |
| 8-bit Displacement | 73 + | | 1 | 1 | | |
| Full Displacement | 0F 83 +++ | | 1 | 1 | | |
| **JNBE/JA** *Jump on Not Below or Equal/Above* | | - - - - - - - - - | | | | r |
| 8-bit Displacement | 77 + | | 1 | 1 | | |
| Full Displacement | 0F 87 +++ | | 1 | 1 | | |
| **JNE/JNZ** *Jump on Not Equal/Not Zero* | | - - - - - - - - - | | | | r |
| 8-bit Displacement | 75 + | | 1 | 1 | | |
| Full Displacement | 0F 85 +++ | | 1 | 1 | | |
| **JNL/JGE** *Jump on Not Less/Greater or Equal* | | - - - - - - - - - | | | | r |
| 8-bit Displacement | 7D + | | 1 | 1 | | |
| Full Displacement | 0F 8D +++ | | 1 | 1 | | |
| **JNLE/JG** *Jump on Not Less or Equal/Greater* | | - - - - - - - - - | | | | r |
| 8-bit Displacement | 7F + | | 1 | 1 | | |
| Full Displacement | 0F 8F +++ | | 1 | 1 | | |

| | | |
|---|---|---|
| # = immediate 8-bit data | + = 8-bit signed displacement | x = modified |
| ## = immediate 16-bit data | +++ = full signed displacement (16, 32 bits) | – = unchanged |
| ### = full immediate 32-bit data (8, 16, 32 bits) | | u = undefined |

**Table 6-20. 6x86 CPU Instruction Set Clock Count Summary (Continued)**

| INSTRUCTION | OPCODE | FLAGS | REAL MODE CLOCK COUNT | PROTECTED MODE CLOCK COUNT | NOTES | |
|---|---|---|---|---|---|---|
| | | OF DF IF TF SF ZF AF PF CF | Reg/ Cache Hit | Reg/ Cache Hit | Real Mode | Protected Mode |
| **JNO** *Jump on Not Overflow* | | - - - - - - - - - | | | | r |
| 8-bit Displacement | 71 + | | 1 | 1 | | |
| Full Displacement | 0F 81 +++ | | 1 | 1 | | |
| **JNP/JPO** *Jump on Not Parity/Parity Odd* | | - - - - - - - - - | | | | r |
| 8-bit Displacement | 7B + | | 1 | 1 | | |
| Full Displacement | 0F 8B +++ | | 1 | 1 | | |
| **JNS** *Jump on Not Sign* | | - - - - - - - - - | | | | r |
| 8-bit Displacement | 79 + | | 1 | 1 | | |
| Full Displacement | 0F 89 +++ | | 1 | 1 | | |
| **JO** *Jump on Overflow* | | - - - - - - - - - | | | | r |
| 8-bit Displacement | 70 + | | 1 | 1 | | |
| Full Displacement | 0F 80 +++ | | 1 | 1 | | |
| **JP/JPE** *Jump on Parity/Parity Even* | | - - - - - - - - - | | | | r |
| 8-bit Displacement | 7A + | | 1 | 1 | | |
| Full Displacement | 0F 8A +++ | | 1 | 1 | | |
| **JS** *Jump on Sign* | | - - - - - - - - - | | | | r |
| 8-bit Displacement | 78 + | | 1 | 1 | | |
| Full Displacement | 0F 88 +++ | | 1 | 1 | | |
| **LAHF** *Load AH with Flags* | 9F | - - - - - - - - - | 2 | 2 | | |
| **LAR** *Load Access Rights* | | - - - - - x - - - | | | a | g,h,j,p |
| From Register/Memory | 0F 02 [mod reg r/m] | | | 8 | | |
| **LDS** *Load Pointer to DS* | C5 [mod reg r/m] | - - - - - - - - - | 2 | 4 | b | h,i,j |
| **LEA** *Load Effective Address* | 8D [mod reg r/m] | - - - - - - - - - | | | | |
| No Index Register | | | 1 | 1 | | |
| With Index Register | | | 1 | 1 | | |
| **LEAVE** *Leave Current Stack Frame* | C9 | - - - - - - - - - | 4 | 4 | b | h |
| **LES** *Load Pointer to ES* | C4 [mod reg r/m] | - - - - - - - - - | 2 | 4 | b | h,i,j |
| **LFS** *Load Pointer to FS* | 0F B4 [mod reg r/m] | - - - - - - - - - | 2 | 4 | b | h,i,j |
| **LGDT** *Load GDT Register* | 0F 01 [mod 010 r/m] | - - - - - - - - - | 8 | 8 | b,c | h,l |
| **LGS** *Load Pointer to GS* | 0F B5 [mod reg r/m] | - - - - - - - - - | 2 | 4 | b | h,i,j |
| **LIDT** *Load IDT Register* | 0F 01 [mod 011 r/m] | - - - - - - - - - | 8 | 8 | b,c | h,l |
| **LLDT** *Load LDT Register* | | - - - - - - - - - | | | a | g,h,j,l |
| From Register/Memory | 0F 00 [mod 010 r/m] | | 5 | 5 | | |
| **LMSW** *Load Machine Status Word* | | - - - - - - - - - | | | b,c | h,l |
| From Register/Memory | 0F 01 [mod 110 r/m] | | 13 | 13 | | |
| **LODS** *Load String* | A [110 w] | - - - - - - - - - | 3 | 3 | b | h |
| **LOOP** *Offset Loop/No Loop* | E2 + | - - - - - - - - - | 1 | 1 | | r |

| | | |
|---|---|---|
| # = immediate 8-bit data | + = 8-bit signed displacement | x = modified |
| ## = immediate 16-bit data | +++ = full signed displacement (16, 32 bits) | – = unchanged |
| ### = full immediate 32-bit data (8, 16, 32 bits) | | u = undefined |

## Table 6-20. 6x86 CPU Instruction Set Clock Count Summary (Continued)

| INSTRUCTION | OPCODE | FLAGS | REAL MODE CLOCK COUNT | PROTECTED MODE CLOCK COUNT | NOTES | |
|---|---|---|---|---|---|---|
| | | OF DF IF TF SF ZF AF PF CF | Reg/ Cache Hit | Reg/ Cache Hit | Real Mode | Protected Mode |
| **LOOPNZ/LOOPNE** *Offset* | E0 + | - - - - - - - - - - - | 1 | 1 | | r |
| **LOOPZ/LOOPE** *Offset* | E1 + | - - - - - - - - - - - | 1 | 1 | | r |
| **LSL** *Load Segment Limit* | | - - - - - - x - - - | | | a | g,h,j,p |
| From Register/Memory | 0F 03 [mod reg r/m] | | | 8 | | |
| **LSS** *Load Pointer to SS* | 0F B2 [mod reg r/m] | - - - - - - - - - - - | 2 | 4 | a | h,i,j |
| **LTR** *Load Task Register* | | - - - - - - - - - - - | | | a | g,h,j,l |
| From Register/Memory | 0F 00 [mod 011 r/m] | | | 7 | | |
| **MOV** *Move Data* | | - - - - - - - - - - - | | | b | h,i,j |
| Register to Register | 8 [10dw] [11 reg r/m] | | 1 | 1 | | |
| Register to Memory | 8 [100w] [mod reg r/m] | | 1 | 1 | | |
| Register/Memory to Register | 8 [101w] [mod reg r/m] | | 1 | 1 | | |
| Immediate to Register/Memory | C [011w] [mod 000 r/m] ### | | 1 | 1 | | |
| Immediate to Register (short form) | B [w reg] ### | | 1 | 1 | | |
| Memory to Accumulator (short form) | A [000w] +++ | | 1 | 1 | | |
| Accumulator to Memory (short form) | A [001w] +++ | | 1 | 1 | | |
| Register/Memory to Segment Register | 8E [mod sreg3 r/m] | | 1 | 1/3 | | |
| Segment Register to Register/Memory | 8C [mod sreg3 r/m] | | 1 | 1 | | |
| **MOV** *Move to/from Control/Debug/Test Regs* | | - - - - - - - - - - - | | | | l |
| Register to CR0/CR2/CR3 | 0F 22 [11 eee reg] | | 20/5/5 | 20/5/5 | | |
| CR0/CR2/CR3 to Register | 0F 20 [11 eee reg] | | 6 | 6 | | |
| Register to DR0-DR3 | 0F 23 [11 eee reg] | | 16 | 16 | | |
| DR0-DR3 to Register | 0F 21 [11 eee reg] | | 14 | 14 | | |
| Register to DR6-DR7 | 0F 23 [11 eee reg] | | 16 | 16 | | |
| DR6-DR7 to Register | 0F 21 [11 eee reg] | | 14 | 14 | | |
| Register to TR3-5 | 0F 26 [11 eee reg] | | 10 | 10 | | |
| TR3-5 to Register | 0F 24 [11 eee reg] | | 5 | 5 | | |
| Register to TR6-TR7 | 0F 26 [11 eee reg] | | 10 | 10 | | |
| TR6-TR7 to Register | 0F 24 [11 eee reg] | | 6 | 6 | | |
| **MOVS** *Move String* | A [010w] | - - - - - - - - - - - | 4 | 4 | b | h |
| **MOVSX** *Move with Sign Extension* | | - - - - - - - - - - - | | | b | h |
| Register from Register/Memory | 0F B[111w] [mod reg r/m] | | 1 | 1 | | |
| **MOVZX** *Move with Zero Extension* | | - - - - - - - - - - - | | | b | h |
| Register from Register/Memory | 0F B[011w] [mod reg r/m] | | 1 | 1 | | |

| | | |
|---|---|---|
| # = immediate 8-bit data | + = 8-bit signed displacement | x = modified |
| ## = immediate 16-bit data | +++ = full signed displacement (16, 32 bits) | – = unchanged |
| ### = full immediate 32-bit data (8, 16, 32 bits) | | u = undefined |

## Table 6-20. 6x86 CPU Instruction Set Clock Count Summary (Continued)

| INSTRUCTION | OPCODE | FLAGS | | | | | | | | | REAL MODE CLOCK COUNT | PROTECTED MODE CLOCK COUNT | NOTES | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | OF | DF | IF | TF | SF | ZF | AF | PF | CF | Reg/ Cache Hit | Reg/ Cache Hit | Real Mode | Protected Mode |
| **MUL** *Unsigned Multiply* | F [011w] [mod 100 r/m] | x | - | - | - | x | x | u | u | x | | | b | h |
| Accumulator with Register/Memory | | | | | | | | | | | | | | |
| Multiplier: Byte | | | | | | | | | | | 4 | 4 | | |
| Word | | | | | | | | | | | 4 | 4 | | |
| Doubleword | | | | | | | | | | | 10 | 10 | | |
| **NEG** *Negate Integer* | F [011w] [mod 011 r/m] | x | - | - | - | x | x | x | x | x | 1 | 1 | b | h |
| **NOP** *No Operation* | 90 | - | - | - | - | - | - | - | - | - | 1 | 1 | | |
| **NOT** *Boolean Complement* | F [011w] [mod 010 r/m] | - | - | - | - | - | - | - | - | - | 1 | 1 | b | h |
| **OIO** *Official Invalid OpCode* | 0F FF | - | - | x | 0 | - | - | - | - | - | 1 | 8 - 125 | | |
| **OR** *Boolean OR* | | 0 | - | - | - | x | x | u | x | 0 | | | b | h |
| Register to Register | 0 [10dw] [11 reg r/m] | | | | | | | | | | 1 | 1 | | |
| Register to Memory | 0 [100w] [mod reg r/m] | | | | | | | | | | 1 | 1 | | |
| Memory to Register | 0 [101w] [mod reg r/m] | | | | | | | | | | 1 | 1 | | |
| Immediate to Register/Memory | 8 [00sw] [mod 001 r/m] ### | | | | | | | | | | 1 | 1 | | |
| Immediate to Accumulator | 0 [110w] ### | | | | | | | | | | 1 | 1 | | |
| **OUT** *Output to Port* | | - | - | - | - | - | - | - | - | - | | | | m |
| Fixed Port | E [011w] # | | | | | | | | | | 14 | 14/28 | | |
| Variable Port | E [111w] | | | | | | | | | | 14 | 14/28 | | |
| **OUTS** *Output String* | 6 [111w] | - | - | - | - | - | - | - | - | - | 14 | 14/28 | b | h,m |
| **POP** *Pop Value off Stack* | | - | - | - | - | - | - | - | - | - | | | b | h,i,j |
| Register/Memory | 8F [mod 000 r/m] | | | | | | | | | | 1 | 1 | | |
| Register (short form) | 5 [1 reg] | | | | | | | | | | 1 | 1 | | |
| Segment Register (ES, SS, DS) | [000 sreg2 111] | | | | | | | | | | 1 | 3 | | |
| Segment Register (FS, GS) | 0F [10 sreg3 001] | | | | | | | | | | 1 | 3 | | |
| **POPA** *Pop All General Registers* | 61 | - | - | - | - | - | - | - | - | - | 6 | 6 | b | h |
| **POPF** *Pop Stack into FLAGS* | 9D | x | x | x | x | x | x | x | x | x | 9 | 9 | b | h,n |
| **PREFIX BYTES** | | - | - | - | - | - | - | - | - | - | | | | m |
| Assert Hardware LOCK Prefix | F0 | | | | | | | | | | | | | |
| Address Size Prefix | 67 | | | | | | | | | | | | | |
| Operand Size Prefix | 66 | | | | | | | | | | | | | |
| Segment Override Prefix | | | | | | | | | | | | | | |
| CS | 2E | | | | | | | | | | | | | |
| DS | 3E | | | | | | | | | | | | | |
| ES | 26 | | | | | | | | | | | | | |
| FS | 64 | | | | | | | | | | | | | |
| GS | 65 | | | | | | | | | | | | | |
| SS | 36 | | | | | | | | | | | | | |

| | | |
|---|---|---|
| # = immediate 8-bit data | + = 8-bit signed displacement | x = modified |
| ## = immediate 16-bit data | +++ = full signed displacement (16, 32 bits) | – = unchanged |
| ### = full immediate 32-bit data (8, 16, 32 bits) | | u = undefined |

## Table 6-20. 6x86 CPU Instruction Set Clock Count Summary (Continued)

| INSTRUCTION | OPCODE | FLAGS | | | | | | | | | REAL MODE CLOCK COUNT | PROTECTED MODE CLOCK COUNT | NOTES | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | OF | DF | IF | TF | SF | ZF | AF | PF | CF | Reg/ Cache Hit | Reg/ Cache Hit | Real Mode | Protected Mode |
| **PUSH** *Push Value onto Stack* | | - | - | - | - | - | - | - | - | - | | | b | h |
| Register/Memory | FF [mod 110 r/m] | | | | | | | | | | 1 | 1 | | |
| Register (short form) | 5 [0 reg] | | | | | | | | | | 1 | 1 | | |
| Segment Register (ES, CS, SS, DS) | [000 sreg2 110] | | | | | | | | | | 1 | 1 | | |
| Segment Register (FS, GS) | 0F [10 sreg3 000] | | | | | | | | | | 1 | 1 | | |
| Immediate | 6 [10s0] ### | | | | | | | | | | 1 | 1 | | |
| **PUSHA** *Push All General Registers* | 60 | - | - | - | - | - | - | - | - | - | 6 | 6 | b | h |
| **PUSHF** *Push FLAGS Register* | 9C | - | - | - | - | - | - | - | - | - | 2 | 2 | b | h |
| **RCL** *Rotate Through Carry Left* | | | | | | | | | | | | | b | h |
| Register/Memory by 1 | D [000w] [mod 010 r/m] | x | - | - | - | - | - | - | - | x | 3 | 3 | | |
| Register/Memory by CL | D [001w] [mod 010 r/m] | u | - | - | - | - | - | - | - | x | 8 | 8 | | |
| Register/Memory by Immediate | C [000w] [mod 010 r/m] # | u | - | - | - | - | - | - | - | x | 8 | 8 | | |
| **RCR** *Rotate Through Carry Right* | | | | | | | | | | | | | b | h |
| Register/Memory by 1 | D [000w] [mod 011 r/m] | x | - | - | - | - | - | - | - | x | 4 | 4 | | |
| Register/Memory by CL | D [001w] [mod 011 r/m] | u | - | - | - | - | - | - | - | x | 9 | 9 | | |
| Register/Memory by Immediate | C [000w] [mod 011 r/m] # | u | - | - | - | - | - | - | - | x | 9 | 9 | | |
| **REP INS** *Input String* | F3 6[110w] | - | - | - | - | - | - | - | - | - | 12+5n | 12+5n\ 28+5n | b | h,m |
| **REP LODS** *Load String* | F3 A[110w] | - | - | - | - | - | - | - | - | - | 10+n | 10+n | b | h |
| **REP MOVS** *Move String* | F3 A[010w] | - | - | - | - | - | - | - | - | - | 9+n | 9+n | b | h |
| **REP OUTS** *Output String* | F3 6[111w] | - | - | - | - | - | - | - | - | - | 12+5n | 12+5n\ 28+5n | b | h,m |
| **REP STOS** *Store String* | F3 A[101w] | - | - | - | - | - | - | - | - | - | 10+n | 10+n | b | h |
| **REPE CMPS** *Compare String* | F3 A[011w] | x | - | - | - | x | x | x | x | x | | | b | h |
| *(Find non-match)* | | | | | | | | | | | 10+2n | 10+2n | | |
| **REPE SCAS** *Scan String* *(Find non-AL/AX/EAX)* | F3 A[111w] | x | - | - | - | x | x | x | x | x | 10+2n | 10+2n | b | h |
| **REPNE CMPS** *Compare String* *(Find match)* | F2 A[011w] | x | - | - | - | x | x | x | x | x | 10+2n | 10+2n | b | h |
| **REPNE SCAS** *Scan String* *(Find AL/AX/EAX)* | F2 A[111w] | x | - | - | - | x | x | x | x | x | 10+2n | 10+2n | b | h |
| **RET** *Return from Subroutine* | | - | - | - | - | - | - | - | - | - | | | b | g,h,j,k,r |
| Within Segment | C3 | | | | | | | | | | 3 | 3 | | |
| Within Segment Adding Immediate to SP | C2 ## | | | | | | | | | | 4 | 4 | | |
| Intersegment | CB | | | | | | | | | | 4 | 7 | | |
| Intersegment Adding Immediate to SP | CA ## | | | | | | | | | | 4 | 7 | | |
| Protected Mode: Different Privilege Level | | | | | | | | | | | | | | |
| Intersegment | | | | | | | | | | | | 23 | | |
| Intersegment Adding Immediate to SP | | | | | | | | | | | | 23 | | |

| | | |
|---|---|---|
| # = immediate 8-bit data | + = 8-bit signed displacement | x = modified |
| ## = immediate 16-bit data | +++ = full signed displacement (16, 32 bits) | - = unchanged |
| ### = full immediate 32-bit data (8, 16, 32 bits) | | u = undefined |

## Table 6-20. 6x86 CPU Instruction Set Clock Count Summary (Continued)

| INSTRUCTION | OPCODE | FLAGS | | | | | | | | | REAL MODE CLOCK COUNT | PROTECTED MODE CLOCK COUNT | NOTES | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | OF | DF | IF | TF | SF | ZF | AF | PF | CF | Reg/ Cache Hit | Reg/ Cache Hit | Real Mode | Protected Mode |
| **ROL** *Rotate Left* | | | | | | | | | | | | | b | h |
| Register/Memory by 1 | D[000w] [mod 000 r/m] | x | - | - | - | - | - | - | - | x | 1 | 1 | | |
| Register/Memory by CL | D[001w] [mod 000 r/m] | u | - | - | - | - | - | - | - | x | 2 | 2 | | |
| Register/Memory by Immediate | C[000w] [mod 000 r/m] # | u | - | - | - | - | - | - | - | x | 1 | 1 | | |
| **ROR** *Rotate Right* | | | | | | | | | | | | | b | h |
| Register/Memory by 1 | D[000w] [mod 001 r/m] | x | - | - | - | - | - | - | - | x | 1 | 1 | | |
| Register/Memory by CL | D[001w] [mod 001 r/m] | u | - | - | - | - | - | - | - | x | 2 | 2 | | |
| Register/Memory by Immediate | C[000w] [mod 001 r/m] # | u | - | - | - | - | - | - | - | x | 1 | 1 | | |
| **RSDC** *Restore Segment Register and Descriptor* | 0F 79 [mod sreg3 r/m] | - | - | - | - | - | - | - | - | - | 6 | 6 | s | s |
| **RSLDT** *Restore LDTR and Descriptor* | 0F 7B [mod 000 r/m] | - | - | - | - | - | - | - | - | - | 6 | 6 | s | s |
| **RSM** *Resume from SMM Mode* | 0F AA | x | x | x | x | x | x | x | x | x | 40 | 40 | s | s |
| **RSTS** *Restore TSR and Descriptor* | 0F 7D [mod 000 r/m] | - | - | - | - | - | - | - | - | - | 6 | 6 | s | s |
| **SAHF** *Store AH in FLAGS* | 9E | - | - | - | - | x | x | x | x | x | 1 | 1 | | |
| **SAL** *Shift Left Arithmetic* | | | | | | | | | | | | | b | h |
| Register/Memory by 1 | D[000w] [mod 100 r/m] | x | - | - | - | x | x | u | x | x | 1 | 1 | | |
| Register/Memory by CL | D[001w] [mod 100 r/m] | u | - | - | - | x | x | u | x | x | 2 | 2 | | |
| Register/Memory by Immediate | C[000w] [mod 100 r/m] # | u | - | - | - | x | x | u | x | x | 1 | 1 | | |
| **SAR** *Shift Right Arithmetic* | | | | | | | | | | | | | b | h |
| Register/Memory by 1 | D[000w] [mod 111 r/m] | x | - | - | - | x | x | u | x | x | 1 | 1 | | |
| Register/Memory by CL | D[001w] [mod 111 r/m] | u | - | - | - | x | x | u | x | x | 2 | 2 | | |
| Register/Memory by Immediate | C[000w] [mod 111 r/m] # | u | - | - | - | x | x | u | x | x | 1 | 1 | | |
| **SBB** *Integer Subtract with Borrow* | | x | - | - | - | x | x | x | x | x | | | b | h |
| Register to Register | 1[10dw] [11 reg r/m] | | | | | | | | | | 1 | 1 | | |
| Register to Memory | 1[100w] [mod reg r/m] | | | | | | | | | | 1 | 1 | | |
| Memory to Register | 1[101w] [mod reg r/m] | | | | | | | | | | 1 | 1 | | |
| Immediate to Register/Memory | 8[00sw] [mod 011 r/m] ### | | | | | | | | | | 1 | 1 | | |
| Immediate to Accumulator (short form) | 1[110w] ### | | | | | | | | | | 1 | 1 | | |
| **SCAS** *Scan String* | A [111w] | x | - | - | - | x | x | x | x | x | 2 | 2 | b | h |
| **SETB/SETNAE/SETC** *Set Byte on Below/Not Above or Equal/Carry* | | - | - | - | - | - | - | - | - | - | | | | h |
| To Register/Memory | 0F 92 [mod 000 r/m] | | | | | | | | | | 1 | 1 | | |
| **SETBE/SETNA** *Set Byte on Below or Equal/Not Above* | | - | - | - | - | - | - | - | - | - | | | | h |
| To Register/Memory | 0F 96 [mod 000 r/m] | | | | | | | | | | 1 | 1 | | |
| **SETE/SETZ** *Set Byte on Equal/Zero* | | - | - | - | - | - | - | - | - | - | | | | h |
| To Register/Memory | 0F 94 [mod 000 r/m] | | | | | | | | | | 1 | 1 | | |
| **SETL/SETNGE** *Set Byte on Less/Not Greater or Equal* | | - | - | - | - | - | - | - | - | - | | | | h |
| To Register/Memory | 0F 9C [mod 000 r/m] | | | | | | | | | | 1 | 1 | | |

| | | |
|---|---|---|
| # = immediate 8-bit data | + = 8-bit signed displacement | x = modified |
| ## = immediate 16-bit data | +++ = full signed displacement (16, 32 bits) | – = unchanged |
| ### = full immediate 32-bit data (8, 16, 32 bits) | | u = undefined |

## Table 6-20. 6x86 CPU Instruction Set Clock Count Summary (Continued)

| INSTRUCTION | OPCODE | FLAGS | REAL MODE CLOCK COUNT | PROTECTED MODE CLOCK COUNT | NOTES | |
|---|---|---|---|---|---|---|
| | | OF DF IF TF SF ZF AF PF CF | Reg/ Cache Hit | Reg/ Cache Hit | Real Mode | Protected Mode |
| **SETLE/SETNG** *Set Byte on Less or Equal/Not Greater* | | - - - - - - - - - - | | | | h |
| To Register/Memory | 0F 9E [mod 000 r/m] | | 1 | 1 | | |
| **SETNB/SETAE/SETNC** *Set Byte on Not Below/ Above or Equal/Not Carry* | | - - - - - - - - - - | | | | h |
| To Register/Memory | 0F 93 [mod 000 r/m] | | 1 | 1 | | |
| **SETNBE/SETA** *Set Byte on Not Below or Equal/Above* | | - - - - - - - - - - | | | | h |
| To Register/Memory | 0F 97 [mod 000 r/m] | | 1 | 1 | | |
| **SETNE/SETNZ** *Set Byte on Not Equal/Not Zero* | | - - - - - - - - - - | | | | h |
| To Register/Memory | 0F 95 [mod 000 r/m] | | 1 | 1 | | |
| **SETNL/SETGE** *Set Byte on Not Less/Greater or Equal* | | - - - - - - - - - - | | | | h |
| To Register/Memory | 0F 9D [mod 000 r/m] | | 1 | 1 | | |
| **SETNLE/SETG** *Set Byte on Not Less or Equal/Greater* | | - - - - - - - - - - | | | | h |
| To Register/Memory | 0F 9F [mod 000 r/m] | | 1 | 1 | | |
| **SETNO** *Set Byte on Not Overflow* | | - - - - - - - - - - | | | | h |
| To Register/Memory | 0F 91 [mod 000 r/m] | | 1 | 1 | | |
| **SETNP/SETPO** *Set Byte on Not Parity/Parity Odd* | | - - - - - - - - - - | | | | h |
| To Register/Memory | 0F 9B [mod 000 r/m] | | 1 | 1 | | |
| **SETNS** *Set Byte on Not Sign* | | - - - - - - - - - - | | | | h |
| To Register/Memory | 0F 99 [mod 000 r/m] | | 1 | 1 | | |
| **SETO** *Set Byte on Overflow* | | - - - - - - - - - - | | | | h |
| To Register/Memory | 0F 90 [mod 000 r/m] | | 1 | 1 | | |
| **SETP/SETPE** *Set Byte on Parity/Parity Even* | | - - - - - - - - - - | | | | h |
| To Register/Memory | 0F 9A [mod 000 r/m] | | 1 | 1 | | |
| **SETS** *Set Byte on Sign* | | - - - - - - - - - - | | | | h |
| To Register/Memory | 0F 98 [mod 000 r/m] | | 1 | 1 | | |
| **SGDT** *Store GDT Register* | | - - - - - - - - - - | 4 | 4 | b,c | h |
| To Register/Memory | 0F 01 [mod 000 r/m] | | | | | |
| **SHL** *Shift Left Logical* | | | | | b | h |
| Register/Memory by 1 | D [000w] [mod 100 r/m] | x - - - x x u x x | 1 | 1 | | |
| Register/Memory by CL | D [001w] [mod 100 r/m] | u - - - x x u x x | 2 | 2 | | |
| Register/Memory by Immediate | C [000w] [mod 100 r/m] # | u - - - x x u x x | 1 | 1 | | |
| **SHLD** *Shift Left Double* | | u - - - x x u x x | | | b | h |
| Register/Memory by Immediate | 0F A4 [mod reg r/m] # | | 4 | 4 | | |
| Register/Memory by CL | 0F A5 [mod reg r/m] | | 5 | 5 | | |

| | | |
|---|---|---|
| # = immediate 8-bit data | + = 8-bit signed displacement | x = modified |
| ## = immediate 16-bit data | +++ = full signed displacement (16, 32 bits) | - = unchanged |
| ### = full immediate 32-bit data (8, 16, 32 bits) | | u = undefined |

## Table 6-20. 6x86 CPU Instruction Set Clock Count Summary (Continued)

| INSTRUCTION | OPCODE | FLAGS | REAL MODE CLOCK COUNT | PROTECTED MODE CLOCK COUNT | NOTES | |
|---|---|---|---|---|---|---|
| | | OF DF IF TF SF ZF AF PF CF | Reg/ Cache Hit | Reg/ Cache Hit | Real Mode | Protected Mode |
| **SHR** *Shift Right Logical* | | | | | b | h |
| Register/Memory by 1 | D [000w] [mod 101 r/m] | x - - - x x u x x | 1 | 1 | | |
| Register/Memory by CL | D [001w] [mod 101 r/m] | u - - - x x u x x | 2 | 2 | | |
| Register/Memory by Immediate | C [000w] [mod 101 r/m] # | u - - - x x u x x | 1 | 1 | | |
| **SHRD** *Shift Right Double* | | u - - - x x u x x | | | b | h |
| Register/Memory by Immediate | 0F AC [mod reg r/m] # | | 4 | 4 | | |
| Register/Memory by CL | 0F AD [mod reg r/m] | | 5 | 5 | | |
| **SIDT** *Store IDT Register* | | - - - - - - - - - | | | b,c | h |
| To Register/Memory | 0F 01 [mod 001 r/m] | | 4 | 4 | | |
| **SLDT** *Store LDT Register* | | - - - - - - - - - | | | a | h |
| To Register/Memory | 0F 00 [mod 000 r/m] | | | 1 | | |
| **SMINT** *Software SMM Entry* | 0F 7E | - - - - - - - - - | 55 | 55 | s | s |
| **SMSW** *Store Machine Status Word* | 0F 01 [mod 100 r/m] | - - - - - - - - - | 6 | 6 | b,c | h |
| **STC** *Set Carry Flag* | F9 | - - - - - - - - 1 | 1 | 1 | | |
| **STD** *Set Direction Flag* | FD | - 1 - - - - - - - | 7 | 7 | | |
| **STI** *Set Interrupt Flag* | FB | - - 1 - - - - - - | 7 | 7 | | m |
| **STOS** *Store String* | A [101w] | - - - - - - - - - | 2 | 2 | b | h |
| **STR** *Store Task Register* | | - - - - - - - - - | | | a | h |
| To Register/Memory | 0F 00 [mod 001 r/m] | | | 4 | | |
| **SUB** *Integer Subtract* | | x - - - x x x x x | | | b | h |
| Register to Register | 2 [10dw] [11 reg r/m] | | 1 | 1 | | |
| Register to Memory | 2 [100w] [mod reg r/m] | | 1 | 1 | | |
| Memory to Register | 2 [101w] [mod reg r/m] | | 1 | 1 | | |
| Immediate to Register/Memory | 8 [00sw] [mod 101 r/m] ### | | 1 | 1 | | |
| Immediate to Accumulator (short form) | 2 [110w] ### | | 1 | 1 | | |
| **SVDC** *Save Segment Register and Descriptor* | 0F 78 [mod sreg3 r/m] | - - - - - - - - - | 12 | 12 | s | s |
| **SVLDT** *Save LDTR and Descriptor* | 0F 7A [mod 000 r/m] | - - - - - - - - - | 12 | 12 | s | s |
| **SVTS** *Save TSR and Descriptor* | 0F 7C [mod 000 r/m] | - - - - - - - - - | 14 | 14 | s | s |
| **TEST** *Test Bits* | | 0 - - - x x u x 0 | | | b | h |
| Register/Memory and Register | 8 [010w] [mod reg r/m] | | 1 | 1 | | |
| Immediate Data and Register/Memory | F [011w] [mod 000 r/m] ### | | 1 | 1 | | |
| Immediate Data and Accumulator | A [100w] ### | | 1 | 1 | | |
| **VERR** *Verify Read Access* | | - - - - - x - - - | | | a | g,h,j,p |
| To Register/Memory | 0F 00 [mod 100 r/m] | | | 7 | | |
| **VERW** *Verify Write Access* | | - - - - - x - - - | | | a | g,h,j,p |
| To Register/Memory | 0F 00 [mod 101 r/m] | | | 7 | | |
| **WAIT** *Wait Until FPU Not Busy* | 9B | - - - - - - - - - | 5 | 5 | | |

| | | |
|---|---|---|
| # = immediate 8-bit data | + = 8-bit signed displacement | x = modified |
| ## = immediate 16-bit data | +++ = full signed displacement (16, 32 bits) | – = unchanged |
| ### = full immediate 32-bit data (8, 16, 32 bits) | | u = undefined |

**Table 6-20. 6x86 CPU Instruction Set Clock Count Summary (Continued)**

| INSTRUCTION | OPCODE | FLAGS | | | | | | | | | REAL MODE CLOCK COUNT | PROTECTED MODE CLOCK COUNT | NOTES | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | OF | DF | IF | TF | SF | ZF | AF | PF | CF | Reg/ Cache Hit | Reg/ Cache Hit | Real Mode | Protected Mode |
| **WBINVD** *Write-Back and Invalidate Cache* | 0F 09 | - | - | - | - | - | - | - | - | - | 15 | 15 | t | t |
| **XADD** *Exchange and Add* | | x | - | - | - | x | x | x | x | x | | | | |
| Register1, Register2 | 0F C[000w] [11 reg2 reg1] | | | | | | | | | | 2 | 2 | | |
| Memory, Register | 0F C[000w] [mod reg r/m] | | | | | | | | | | 2 | 2 | | |
| **XCHG** *Exchange* | | - | - | - | - | - | - | - | - | - | | | b,f | f,h |
| Register/Memory with Register | 8[011w] [mod reg r/m] | | | | | | | | | | 2 | 2 | | |
| Register with Accumulator | 9[0 reg] | | | | | | | | | | 2 | 2 | | |
| **XLAT** *Translate Byte* | D7 | - | - | - | - | - | - | - | - | - | 4 | 4 | | h |
| **XOR** *Boolean Exclusive OR* | | 0 | - | - | - | x | x | u | x | 0 | | | b | h |
| Register to Register | 3 [00dw] [11 reg r/m] | | | | | | | | | | 1 | 1 | | |
| Register to Memory | 3 [000w] [mod reg r/m] | | | | | | | | | | 1 | 1 | | |
| Memory to Register | 3 [001w] [mod reg r/m] | | | | | | | | | | 1 | 1 | | |
| Immediate to Register/Memory | 8 [00sw] [mod 110 r/m] ### | | | | | | | | | | 1 | 1 | | |
| Immediate to Accumulator (short form) | 3 [010w] ### | | | | | | | | | | 1 | 1 | | |

| | | |
|---|---|---|
| # = immediate 8-bit data | + = 8-bit signed displacement | x = modified |
| ## = immediate 16-bit data | +++ = full signed displacement (16, 32 bits) | – = unchanged |
| ### = full immediate 32-bit data (8, 16, 32 bits) | | u = undefined |

**Instruction Notes for Instruction Set Summary**

**Notes a through c apply to Real Address Mode only:**

a. This is a Protected Mode instruction. Attempted execution in Real Mode will result in exception 6 (invalid op-code).

b. Exception 13 fault (general protection) will occur in Real Mode if an operand reference is made that partially or fully extends beyond the maximum CS, DS, ES, FS, or GS segment limit (FFFFH). Exception 12 fault (stack segment limit violation or not present) will occur in Real Mode if an operand reference is made that partially or fully extends beyond the maximum SS limit.

c. This instruction may be executed in Real Mode. In Real Mode, its purpose is primarily to initialize the CPU for Protected Mode.

d. -

**Notes e through g apply to Real Address Mode and Protected Virtual Address Mode:**

e. An exception may occur, depending on the value of the operand.

f. LOCK# is automatically asserted, regardless of the presence or absence of the LOCK prefix.

g. LOCK# is asserted during descriptor table accesses.

**Notes h through r apply to Protected Virtual Address Mode only:**

h. Exception 13 fault will occur if the memory operand in CS, DS, ES, FS, or GS cannot be used due to either a segment limit violation or an access rights violation. If a stack limit is violated, an exception 12 occurs.

i. For segment load operations, the CPL, RPL, and DPL must agree with the privilege rules to avoid an exception 13 fault. The segment's descriptor must indicate "present" or exception 11 (CS, DS, ES, FS, GS not present). If the SS register is loaded and a stack segment not present is detected, an exception 12 occurs.

**Continued on next page...**

j. All segment descriptor accesses in the GDT or LDT made by this instruction will automatically assert LOCK# to maintain descriptor integrity in multiprocessor systems.
k. JMP, CALL, INT, RET, and IRET instructions referring to another code segment will cause an exception 13, if an applicable privilege rule is violated.
l. An exception 13 fault occurs if CPL is greater than 0 (0 is the most privileged level).
m. An exception 13 fault occurs if CPL is greater than IOPL.
n. The IF bit of the flag register is not updated if CPL is greater than IOPL. The IOPL and VM fields of the flag register are updated only if CPL = 0.
o. The PE bit of the MSW (CR0) cannot be reset by this instruction. Use MOV into CRO if desiring to reset the PE bit.
p. Any violation of privilege rules as apply to the selector operand does not cause a Protection exception, rather, the zero flag is cleared.
q. If the coprocessor's memory operand violates a segment limit or segment access rights, an exception 13 fault will occur before the ESC instruction is executed. An exception 12 fault will occur if the stack limit is violated by the operand's starting address.
r. The destination of a JMP, CALL, INT, RET, or IRET must be in the defined limit of a code segment or an exception 13 fault will occur.

**Note s applies to Cyrix specific SMM instructions:**

s. All memory accesses to SMM space are non-cacheable. An invalid opcode exception 6 occurs unless SMI is enabled and ARR3 size > 0, and CPL = 0 and [SMAC is set or if in an SMI handler].

**Note t applies to cache invalidation instructions with the cache operating in write-back mode:**

t. The total clock count is the clock count shown plus the number of clocks required to write all “modified” cache lines to external memory.

## 6.5 FPU Clock Counts

The CPU is functionally divided into the FPU, and the integer unit. The FPU processes floating point instructions only and does so in parallel with the integer unit.

For example, when the integer unit detects a floating point instruction without memory operands, after two clock cycles the instruction passes to the FPU for execution. The integer unit continues to execute instructions while the FPU executes the floating point instruction. If another FPU instruction is encountered, the second FPU instruction is placed in the FPU queue. Up to four FPU instructions can be queued. In the event of an FPU exception, while other FPU instructions are queued, the state of the CPU is saved to ensure recovery.

### 6.5.1 FPU Clock Count Table

The clock counts for the FPU instructions are listed in Table 6-19 (Page 13). The abbreviations used in this table are listed in Table 6-21.

**Table 6-21. FPU Clock Count Table Abbreviations**

| ABBREVIATION | MEANING |
|---|---|
| n | Stack register number |
| TOS | Top of stack register pointed to by SSS in the status register. |
| ST(1) | FPU register next to TOS |
| ST(n) | A specific FPU register, relative to TOS |
| M.WI | 16-bit integer operand from memory |
| M.SI | 32-bit integer operand from memory |
| M.LI | 64-bit integer operand from memory |
| M.SR | 32-bit real operand from memory |
| M.DR | 64-bit real operand from memory |
| M.XR | 80-bit real operand from memory |
| M.BCD | 18-digit BCD integer operand from memory |
| CC | FPU condition code |
| Env Regs | Status, Mode Control and Tag Registers, Instruction Pointer and Operand Pointer |

## Table 6-22. 6x86 FPU Instruction Set Summary

| FPU INSTRUCTION | OP CODE | OPERATION | CLOCK COUNT | NOTES |
|---|---|---|---|---|
| **F2XM1** *Function Evaluation $2^x$-1* | D9 F0 | TOS ⟵ $2^{TOS}$-1 | 92 - 108 | See Note 2 |
| **FABS** *Floating Absolute Value* | D9 E1 | TOS ⟵ \| TOS \| | 2 | |
| **FADD** *Floating Point Add* | | | | |
| Top of Stack | DC [1100 0 n] | ST(n) ⟵ ST(n) + TOS | 4 - 9 | |
| 80-bit Register | D8 [1100 0 n] | TOS ⟵ TOS + ST(n) | 4 - 9 | |
| 64-bit Real | DC [mod 000 r/m] | TOS ⟵ TOS + M.DR | 4 - 9 | |
| 32-bit Real | D8 [mod 000 r/m] | TOS ⟵ TOS + M.SR | 4 - 9 | |
| **FADDP** *Floating Point Add, Pop* | DE [1100 0 n] | ST(n) ⟵ ST(n) + TOS; then pop TOS | 4 - 9 | |
| **FIADD** *Floating Point Integer Add* | | | | |
| 32-bit integer | DA [mod 000 r/m] | TOS ⟵ TOS + M.SI | 8 - 14 | |
| 16-bit integer | DE [mod 000 r/m] | TOS ⟵ TOS + M.WI | 8 - 14 | |
| **FCHS** *Floating Change Sign* | D9 E0 | TOS ⟵ - TOS | 2 | |
| **FCLEX** *Clear Exceptions* | (9B)DB E2 | Wait then Clear Exceptions | 5 | |
| **FNCLEX** *Clear Exceptions* | DB E2 | Clear Exceptions | 3 | |
| **FCOM** *Floating Point Compare* | | | | |
| 80-bit Register | D8 [1101 0 n] | CC set by TOS - ST(n) | 4 | |
| 64-bit Real | DC [mod 010 r/m] | CC set by TOS - M.DR | 4 | |
| 32-bit Real | D8 [mod 010 r/m] | CC set by TOS - M.SR | 4 | |
| **FCOMP** *Floating Point Compare, Pop* | | | | |
| 80-bit Register | D8 [1101 1 n] | CC set by TOS - ST(n); then pop TOS | 4 | |
| 64-bit Real | DC [mod 011 r/m] | CC set by TOS - M.DR; then pop TOS | 4 | |
| 32-bit Real | D8 [mod 011 r/m] | CC set by TOS - M.SR; then pop TOS | 4 | |
| **FCOMPP** *Floating Point Compare, Pop Two Stack Elements* | DE D9 | CC set by TOS - ST(1); then pop TOS and ST(1) | 4 | |
| **FICOM** *Floating Point Compare* | | | | |
| 32-bit integer | DA [mod 010 r/m] | CC set by TOS - M.WI | 9 - 10 | |
| 16-bit integer | DE [mod 010 r/m] | CC set by TOS - M.SI | 9 - 10 | |
| **FICOMP** *Floating Point Compare* | | | | |
| 32-bit integer | DA [mod 011 r/m] | CC set by TOS - M.WI; then pop TOS | 9 - 10 | |
| 16-bit integer | DE [mod 011 r/m] | CC set by TOS - M.SI; then pop TOS | 9 - 10 | |
| **FCOS** *Function Evaluation: Cos(x)* | D9 FF | TOS ⟵ COS(TOS) | 92 - 141 | See Note 1 |
| **FDECSTP** *Decrement Stack Pointer* | D9 F6 | Decrement top of stack pointer | 4 | |
| **FDIV** *Floating Point Divide* | | | | |
| Top of Stack | DC [1111 1 n] | ST(n) ⟵ ST(n) / TOS | 24 - 34 | |
| 80-bit Register | D8 [1111 0 n] | TOS ⟵ TOS / ST(n) | 24 - 34 | |
| 64-bit Real | DC [mod 110 r/m] | TOS ⟵ TOS / M.DR | 24 - 34 | |
| 32-bit Real | D8 [mod 110 r/m] | TOS ⟵ TOS / M.SR | 24 - 34 | |
| **FDIVP** *Floating Point Divide, Pop* | DE [1111 1 n] | ST(n) ⟵ ST(n) / TOS; then pop TOS | 24 - 34 | |
| **FDIVR** *Floating Point Divide Reversed* | | | | |
| Top of Stack | DC [1111 0 n] | TOS ⟵ ST(n) / TOS | 24 - 34 | |
| 80-bit Register | D8 [1111 1 n] | ST(n) ⟵ TOS / ST(n) | 24 - 34 | |
| 64-bit Real | DC [mod 111 r/m] | TOS ⟵ M.DR / TOS | 24 - 34 | |
| 32-bit Real | D8 [mod 111 r/m] | TOS ⟵ M.SR / TOS | 24 - 34 | |

## Table 6-22. 6x86 FPU Instruction Set Summary (Continued)

| FPU INSTRUCTION | OP CODE | OPERATION | CLOCK COUNT | NOTES |
|---|---|---|---|---|
| **FDIVRP** *Floating Point Divide Reversed*, Pop | DE [1111 0 n] | ST(n) ⟵ TOS / ST(n); then pop TOS | 24 - 34 | |
| **FIDIV** *Floating Point Integer Divide* | | | | |
| 32-bit Integer | DA [mod 110 r/m] | TOS ⟵ TOS / M.SI | 34 - 38 | |
| 16-bit Integer | DE [mod 110 r/m] | TOS ⟵ TOS / M.WI | 33 - 38 | |
| **FIDIVR** *Floating Point Integer Divide Reversed* | | | | |
| 32-bit Integer | DA [mod 111 r/m] | TOS ⟵ M.SI / TOS | 34 - 38 | |
| 16-bit Integer | DE [mod 111 r/m] | TOS ⟵ M.WI / TOS | 33 - 38 | |
| **FFREE** *Free Floating Point Register* | DD [1100 0 n] | TAG(n) ⟵ Empty | 3 | |
| **FINCSTP** *Increment Stack Pointer* | D9 F7 | Increment top of stack pointer | 2 | |
| **FINIT** *Initialize FPU* | (9B)DB E3 | Wait then initialize | 8 | |
| **FNINIT** *Initialize FPU* | DB E3 | Initialize | 6 | |
| **FLD** *Load Data to FPU Reg.* | | | | |
| Top of Stack | D9 [1100 0 n] | Push ST(n) onto stack | 2 | |
| 64-bit Real | DD [mod 000 r/m] | Push M.DR onto stack | 2 | |
| 32-bit Real | D9 [mod 000 r/m] | Push M.SR onto stack | 2 | |
| **FBLD** *Load Packed BCD Data to FPU Reg.* | DF [mod 100 r/m] | Push M.BCD onto stack | 41 - 45 | |
| **FILD** *Load Integer Data to FPU Reg.* | | | | |
| 64-bit Integer | DF [mod 101 r/m] | Push M.LI onto stack | 4 - 8 | |
| 32-bit Integer | DB [mod 000 r/m] | Push M.SI onto stack | 4 - 6 | |
| 16-bit Integer | DF [mod 000 r/m] | Push M.WI onto stack | 3 - 6 | |
| **FLD1** *Load Floating Const.= 1.0* | D9 E8 | Push 1.0 onto stack | 4 | |
| **FLDCW** *Load FPU Mode Control Register* | D9 [mod 101 r/m] | Ctl Word ⟵ Memory | 4 | |
| **FLDENV** *Load FPU Environment* | D9 [mod 100 r/m] | Env Regs ⟵ Memory | 30 | |
| **FLDL2E** *Load Floating Const.= Log$_2$(e)* | D9 EA | Push Log$_2$(e) onto stack | 4 | |
| **FLDL2T** *Load Floating Const.= Log$_2$(10)* | D9 E9 | Push Log$_2$(10) onto stack | 4 | |
| **FLDLG2** *Load Floating Const.= Log$_{10}$(2)* | D9 EC | Push Log$_{10}$(2) onto stack | 4 | |
| **FLDLN2** *Load Floating Const.= Ln(2)* | D9 ED | Push Log$_e$(2) onto stack | 4 | |
| **FLDPI** *Load Floating Const.= π* | D9 EB | Push π onto stack | 4 | |
| **FLDZ** *Load Floating Const.= 0.0* | D9 EE | Push 0.0 onto stack | 4 | |
| **FMUL** *Floating Point Multiply* | | | | |
| Top of Stack | DC [1100 1 n] | ST(n) ⟵ ST(n) × TOS | 4 - 9 | |
| 80-bit Register | D8 [1100 1 n] | TOS ⟵ TOS × ST(n) | 4 - 9 | |
| 64-bit Real | DC [mod 001 r/m] | TOS ⟵ TOS × M.DR | 4 - 8 | |
| 32-bit Real | D8 [mod 001 r/m] | TOS ⟵ TOS × M.SR | 4 - 6 | |
| **FMULP** *Floating Point Multi*ply & Pop | DE [1100 1 n] | ST(n) ⟵ ST(n) × TOS; then pop TOS | 4 - 9 | |
| **FIMUL** *Floating Point Integer Multiply* | | | | |
| 32-bit Integer | DA [mod 001 r/m] | TOS ⟵ TOS × M.SI | 9 - 11 | |
| 16-bit Integer | DE [mod 001 r/m] | TOS ⟵ TOS × M.WI | 8 - 10 | |
| **FNOP** *No Operation* | D9 D0 | No Operation | 2 | |

## Table 6-22. 6x86 FPU Instruction Set Summary (Continued)

| FPU INSTRUCTION | OP CODE | OPERATION | CLOCK COUNT | NOTES |
|---|---|---|---|---|
| **FPATAN** *Function Eval: Tan$^{-1}$(y/x)* | D9 F3 | ST(1) ⟵ ATAN[ST(1) / TOS]; then pop TOS | 97 - 161 | See Note 3 |
| **FPREM** *Floating Point Remainder* | D9 F8 | TOS ⟵ Rem[TOS / ST(1)] | 82 - 91 | |
| **FPREM1** *Floating Point Remainder IEEE* | D9 F5 | TOS ⟵ Rem[TOS / ST(1)] | 82 - 91 | |
| **FPTAN** *Function Eval: Tan(x)* | D9 F2 | TOS ⟵ TAN(TOS); then push 1.0 onto stack | 117 - 129 | See Note 1 |
| **FRNDINT** *Round to Integer* | D9 FC | TOS ⟵ Round(TOS) | 10 - 20 | |
| **FRSTOR** *Load FPU Environment and Reg.* | DD [mod 100 r/m] | Restore state. | 56 - 72 | |
| **FSAVE** *Save FPU Environment and Reg* | (9B)DD[mod 110 r/m] | Wait then save state. | 57 - 67 | |
| **FNSAVE** *Save FPU Environment and Reg* | DD [mod 110 r/m] | Save state. | 55 - 65 | |
| **FSCALE** *Floating Multiply by 2$^n$* | D9 FD | TOS ⟵ TOS $\times 2^{(ST(1))}$ | 7 - 14 | |
| **FSIN** *Function Evaluation: Sin(x)* | D9 FE | TOS ⟵ SIN(TOS) | 76 - 140 | See Note 1 |
| **FSINCOS** *Function Eval.: Sin(x)& Cos(x)* | D9 FB | temp ⟵ TOS; TOS ⟵ SIN(temp); then push COS(temp) onto stack | 145 - 161 | See Note 1 |
| **FSQRT** *Floating Point Square Root* | D9 FA | TOS ⟵ Square Root of TOS | 59 - 60 | |
| **FST** *Store FPU Register* | | | | |
| Top of Stack | DD [1101 0 n] | ST(n) ⟵ TOS | 2 | |
| 80-bit Real | DB [mod 111 r/m] | M.XR ⟵ TOS | 2 | |
| 64-bit Real | DD [mod 010 r/m] | M.DR ⟵ TOS | 2 | |
| 32-bit Real | D9 [mod 010 r/m] | M.SR ⟵ TOS | 2 | |
| **FSTP** *Store FPU Register, Pop* | | | | |
| Top of Stack | DB [1101 1 n] | ST(n) ⟵ TOS; then pop TOS | 2 | |
| 80-bit Real | DB [mod 111 r/m] | M.XR ⟵ TOS; then pop TOS | 2 | |
| 64-bit Real | DD [mod 011 r/m] | M.DR ⟵ TOS; then pop TOS | 2 | |
| 32-bit Real | D9 [mod 011 r/m] | M.SR ⟵ TOS; then pop TOS | 2 | |
| **FBSTP** *Store BCD Data, Pop* | DF [mod 110 r/m] | M.BCD ⟵ TOS; then pop TOS | 57 - 63 | |
| **FIST** *Store Integer FPU Register* | | | | |
| 32-bit Integer | DB [mod 010 r/m] | M.SI ⟵ TOS | 8 - 13 | |
| 16-bit Integer | DF [mod 010 r/m] | M.WI ⟵ TOS | 7 - 10 | |
| **FISTP** *Store Integer FPU Register, Pop* | | | | |
| 64-bit Integer | DF [mod 111 r/m] | M.LI ⟵ TOS; then pop TOS | 10 - 13 | |
| 32-bit Integer | DB [mod 011 r/m] | M.SI ⟵ TOS; then pop TOS | 8 - 13 | |
| 16-bit Integer | DF [mod 011 r/m] | M.WI ⟵ TOS; then pop TOS | 7 - 10 | |
| **FSTCW** *Store FPU Mode Control Register* | (9B)D9[mod 111 r/m] | Wait Memory ⟵ Control Mode Register | 5 | |
| **FNSTCW** *Store FPU Mode Control Register* | D9 [mod 111 r/m] | Memory ⟵ Control Mode Register | 3 | |
| **FSTENV** *Store FPU Environment* | (9B)D9[mod 110 r/m] | Wait Memory ⟵ Env. Registers | 14 - 24 | |
| **FNSTENV** *Store FPU Environment* | D9 [mod 110 r/m] | Memory ⟵ Env. Registers | 12 - 22 | |
| **FSTSW** *Store FPU Status Register* | (9B)DD[mod 111 r/m] | Wait Memory ⟵ Status Register | 6 | |
| **FNSTSW** *Store FPU Status Register* | DD [mod 111 r/m] | Memory ⟵ Status Register | 4 | |
| **FSTSW AX** *Store FPU Status Register to AX* | (9B)DF E0 | Wait AX ⟵ Status Register | 4 | |
| **FNSTSW AX** *Store FPU Status Register to AX* | DF E0 | AX ⟵ Status Register | 2 | |

## Table 6-22. 6x86 FPU Instruction Set Summary (Continued)

| FPU INSTRUCTION | OP CODE | OPERATION | CLOCK COUNT | NOTES |
|---|---|---|---|---|
| **FSUB** *Floating Point Subtract* | | | | |
| Top of Stack | DC [1110 1 n] | ST(n) ⟵ ST(n) - TOS | 4 - 9 | |
| 80-bit Register | D8 [1110 0 n] | TOS ⟵ TOS - ST(n) | 4 - 9 | |
| 64-bit Real | DC [mod 100 r/m] | TOS ⟵ TOS - M.DR | 4 - 9 | |
| 32-bit Real | D8 [mod 100 r/m] | TOS ⟵ TOS - M.SR | 4 - 9 | |
| **FSUBP** *Floating Point Subtract, Pop* | DE [1110 1 n] | ST(n) ⟵ ST(n) - TOS; then pop TOS | 4 - 9 | |
| **FSUBR** *Floating Point Subtract Reverse* | | | | |
| Top of Stack | DC [1110 0 n] | TOS ⟵ ST(n) - TOS | 4 - 9 | |
| 80-bit Register | D8 [1110 1 n] | ST(n) ⟵ TOS - ST(n) | 4 - 9 | |
| 64-bit Real | DC [mod 101 r/m] | TOS ⟵ M.DR - TOS | 4 - 9 | |
| 32-bit Real | D8 [mod 101 r/m] | TOS ⟵ M.SR - TOS | 4 - 9 | |
| **FSUBRP** *Floating Point Subtract Reverse, Pop* | DE [1110 0 n] | ST(n) ⟵ TOS - ST(n); then pop TOS | 4 - 9 | |
| **FISUB** *Floating Point Integer Subtract* | | | | |
| 32-bit Integer | DA [mod 100 r/m] | TOS ⟵ TOS - M.SI | 14 - 29 | |
| 16-bit Integer | DE [mod 100 r/m] | TOS ⟵ TOS - M.WI | 14 - 27 | |
| **FISUBR** *Floating Point Integer Subtract Reverse* | | | | |
| 32-bit Integer Reversed | DA [mod 101 r/m] | TOS ⟵ M.SI - TOS | 14 - 29 | |
| 16-bit Integer Reversed | DE [mod 101 r/m] | TOS ⟵ M.WI - TOS | 14 - 27 | |
| **FTST** *Test Top of Stack* | D9 E4 | CC set by TOS - 0.0 | 4 | |
| **FUCOM** *Unordered Compare* | DD [1110 0 n] | CC set by TOS - ST(n) | 4 | |
| **FUCOMP** *Unordered Compare, Pop* | DD [1110 1 n] | CC set by TOS - ST(n); then pop TOS | 4 | |
| **FUCOMPP** *Unordered Compare, Pop two elements* | DA E9 | CC set by TOS - ST(I); then pop TOS and ST(1) | 4 | |
| **FWAIT** *Wait* | 9B | Wait for FPU not busy | 2 | |
| **FXAM** *Report Class of Operand* | D9 E5 | CC ⟵ Class of TOS | 4 | |
| **FXCH** *Exchange Register with TOS* | D9 [1100 1 n] | TOS ⟷ ST(n) Exchange | 3 | |
| **FXTRACT** *Extract Exponent* | D9 F4 | temp ⟵ TOS; TOS ⟵ exponent (temp); then push significant (temp) onto stack | 11 - 16 | |
| **FLY2X** *Function Eval. y × Log2(x)* | D9 F1 | ST(1) ⟵ ST(1) × $\text{Log}_2$(TOS); then pop TOS | 145 - 154 | |
| **FLY2XP1** *Function Eval. y × Log2(x+1)* | D9 F9 | ST(1) ⟵ ST(1) × $\text{Log}_2$(1+TOS); then pop TOS | 131 - 133 | See Note 4 |

**FPU Instruction Summary Notes**

All references to TOS and ST(n) refer to stack layout prior to execution.

Values popped off the stack are discarded.

A pop from the stack increments the top of stack pointer.

A push to the stack decrements the top of stack pointer.

**Note 1:**
For FCOS, FSIN, FSINCOS and FPTAN, time shown is for absolute value of TOS $< 3\pi/4$.
Add 90 clock counts for argument reduction if outside this range.

For FCOS, clock count is 141 if TOS $< \pi/4$ and clock count is 92 if $\pi/4 <$ TOS $> \pi/2$.

For FSIN, clock count is 81 to 82 if absolute value of TOS $< \pi/4$.

**Note 2:**
For F2XM1, clock count is 92 if absolute value of TOS $< 0.5$.

**Note 3:**
For FPATAN, clock count is 97 if ST(1)/TOS $< \pi/32$.

**Note 4:**
For FYL2XP1, clock count is 170 if TOS is out of range and regular FYL2X is called.

**Note 5:**
The following opcodes are reserved by Cyrix:
D9D7, D9E2, D9E7, DDFC, DED8, DEDA, DEDC, DEDD, DEDE, DFFC.
If a reserved opcode is executed, and unpredictable results may occur (exceptions are not generated).