iW-80186XL Technical Specification

Table of Contents

List of Figures

List of Tables

1 The iW-80186XL Processor

1.1 Overview

The iW-80186XL is a powerful 16-bit microprocessor core, executes instruction list compatible with 80186XL microprocessor. The design along with multiple peripherals can be fit into single FPGA.

1.2 Features

- **iW-80186XL CPU Core**
  - Multiplexed 20-bit address and 16-bit data bus
  - 1M-byte memory space divided into 4 segments
  - 64K-byte IO space
  - Non Maskable Interrupt support
  - Arithmetic-Logic Unit
    - 8,16,32-bit arithmetic operations
    - 8,16-bit logical operations
    - Boolean manipulations
    - 16 x 16 bit multiplication (signed or unsigned)
    - 32/16-bit division (signed or unsigned)
- **CPU On-Chip Peripherals**
  - Programmable Timer / Counter Unit
    - Three programmable independent 16-bit timers
    - TOUT0 to TOUT1 pin outputs
    - TIN0 & TIN1 used either as clock or control signals
    - Timer-2 can be used to clock other two timers
    - Internal / external input clock selectable
  - Direct Memory Access Unit
    - Two independent high-speed DMA channels.
    - Data can be transferred between any combination of memory & IO space.

- DMA transfer can be initiated by external, internal request or by direct programming.
- 20-bit length address register.
- 16-bit length transfer count register.
- Transfer address can be incrementing, decrementing or remained constant.
- Two kinds of channel priority order
  - Fixed priority
  - Rotating priority
- DMAU can be programmed to produce interrupt request when its transfer count reaches zero.
- Both byte & word transfer is possible in case of 80186XL;word transfer is illegal in case of 80186XL processor.

♦ Interrupt Control Unit

- Four external interrupt request inputs (INT0 to INT3).
- Timer0, Timer1, Timer2 and DMA0, DMA1 Interrupts (Internal Interrupts)
- Edge or level triggered interrupt request inputs.
- Individually Mask-able interrupts request
- Programmable interrupt request priority orders.
- Polling operation capability.
- Cascade with external 8259A interrupts (only on INT0 and INT1) operates in either Master mode or Slave mode.
- Special fully nested mode support

♦ Chip Select Unit

- Thirteen programmable chip-select outputs
- Six of the chip-selects map only into memory address space, while the remaining seven can map into either memory or I/O address space
- Programmable block size and start / end address
- Memory or I/O bus cycle decoder
- Programmable wait-state generator
- Provision to disable a chip-select
- Provision to override bus ready

♦ Clock Generator

1.3 Logic Symbol

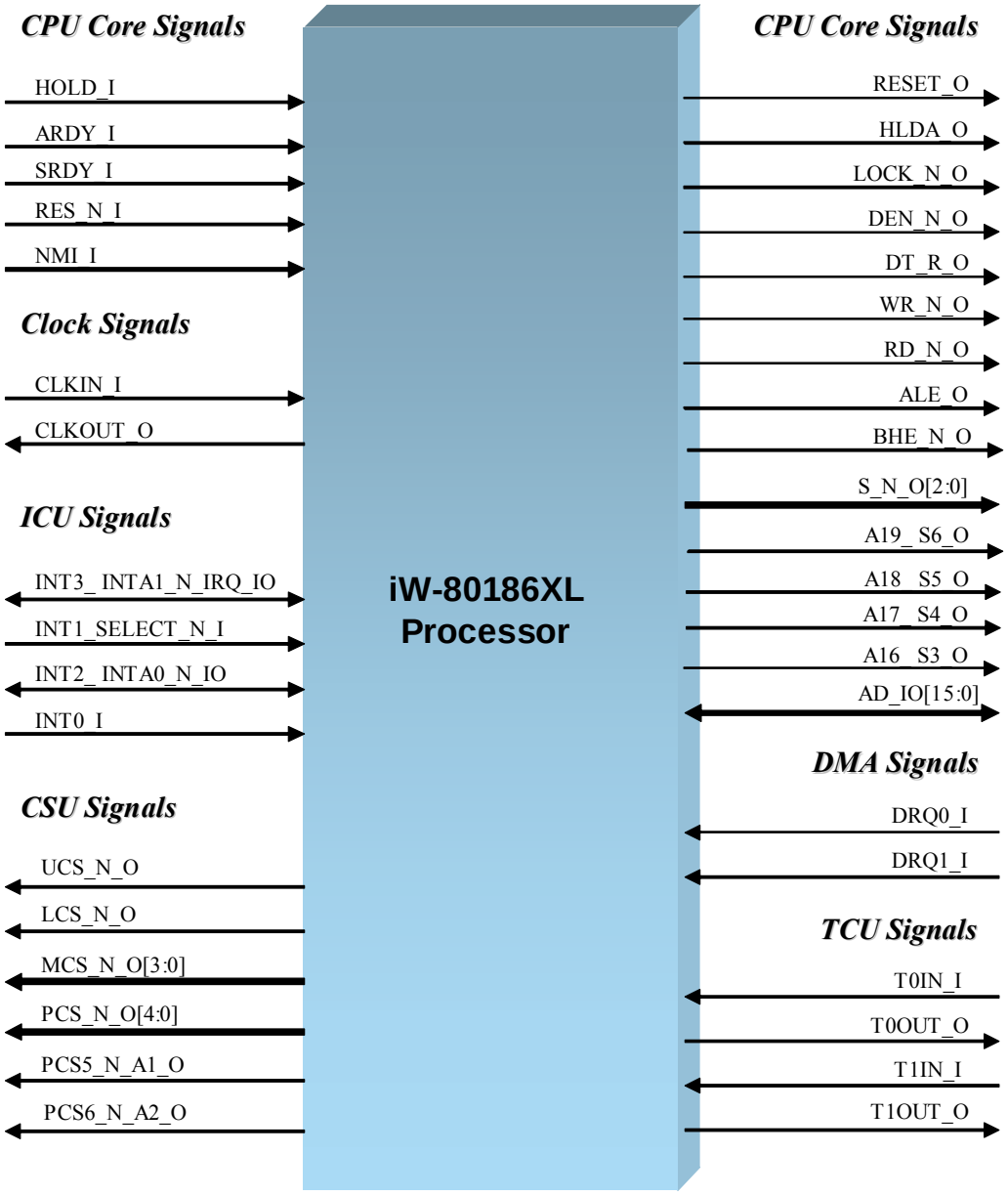

Figure 1: Logic Symbol

1.4 System Level Block Diagram

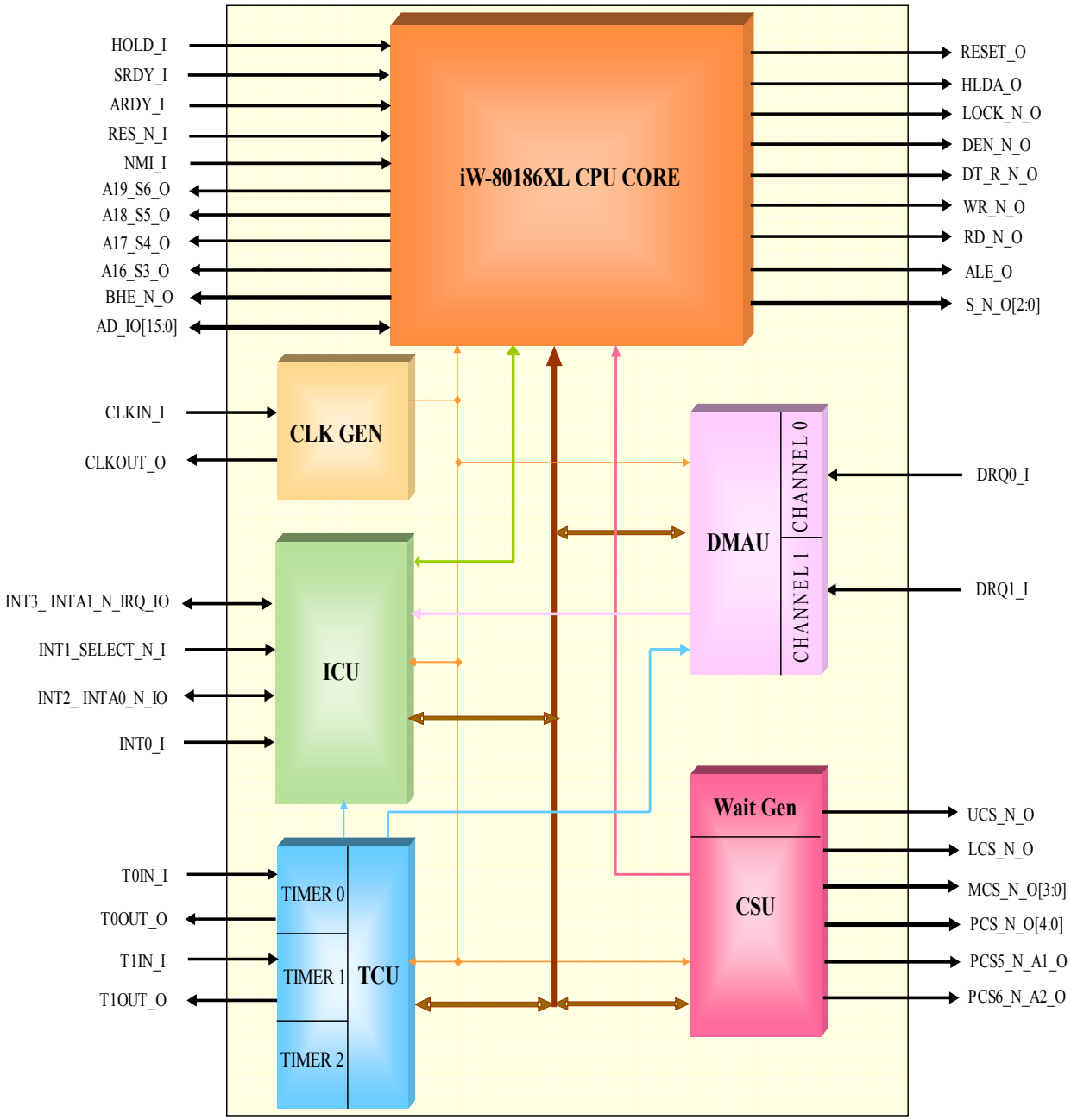

Figure 2: System Level Block Diagram

1.5 Functional Description

The **Central Processing Unit (CPU)** executes instructions, which include fetching, decoding instructions and generating appropriate requests to the Bus Interface Unit.

The **Timer / Counter Unit (TCU)** provides three programmable 16-bit Timers. Two of these are highly flexible and are connected to external pins for control or clocking. A third timer is not connected to any external pins and can only be clocked internally.

The **Direct Memory Access Unit (DMAU)** allows data to be transferred between memory and I/O devices without requiring the CPU's intervention. The DMA Unit has two channels. Each channel can accept DMA requests from one of three sources: an external request pin, the Timer/Counter Unit or direct programming. Data can be transferred between any combination of memory and I/O space.

The **Interrupt Control Unit (ICU)** processes three internal and four external interrupt requests by allocating a priority level to the each request. The Interrupt Control Unit can independently mask each interrupt source or the CPU can globally mask all interrupts. The Interrupt Control Unit operates in either of two modes: Master or Slave.

- **Master mode**: The ICU controls the mask-able interrupt input to the CPU. Interrupts can originate from the on-chip peripherals and from four external interrupt pins.
- **Slave mode**: An external 8259A module controls the mask-able interrupt input to the CPU and acts as the master interrupt controller. The ICU processes only those interrupts from the on-chip peripherals and acts as an interrupt input to the 8259A.

The **Chip-Select Unit (CSU)** integrates logic, which provides up to thirteen programmable chip selects to access memories and peripherals. Six of the chip-selects map only into memory address space, while the remaining seven can map into either memory or I/O address space. Besides selecting a specific device, each chip-select can be used to control the number of wait states inserted into the bus cycle.

The **Clock Generator** generates both internal and external clock.

1.6 Pin Description

The Pinout of the iW-80186XL core has not been fixed to specific FPGA I/O, thereby allowing flexibility with a user's application. Signal names are described in Table-1.

Table 1: Pin Description

Signal	Width	Direction	Description
Clock and Reset Signals			
RES_N_I	1	Input	An active low signal causes the processor to immediately terminate any bus cycle in progress and assume an initialized state. All pins will be driven to a known state, and RESET will also be driven active.
RESET_O	1	Output	Indicates the processor is currently in the reset state. RESET will remain active as long as RES_N remains active.
CLKIN_I	1	Input	External clock input operating at two times the processor operating frequency.
CLKOUT_O	1	Output	Processor Clcok output. It is half of Clock input (CLKIN_I).
CPU Core Signals			
AD_IO [15:0]	16	Input/Output	Provide a multiplexed address and Data bus. During the address phase of the bus cycle, address bits AD [15:0] are presented on the bus and can be latched using ALE. 16-bit data information is transferred during the data phase of the bus cycle.
BHE_N_O	1	Output	Byte High Enable output indicates that the bus cycle in progress is transferring data over the upper half of the data bus. BHE and A0 have the following logical encoding scheme **A0 | BHE_N_O | Encoding** 0 | 0 | Word Transfer 0 | 1 | Even byte transfer 1 | 0 | Odd byte transfer 1 | 1 | NA

<table>
<tr><th>Signal</th><th>Width</th><th>Direction</th><th>Description</th></tr>
<tr><td>A19_S6_O,
A18_S5_O,
A17_S4_O,
A16_S3_O,</td><td>1</td><td>Output</td><td>Address Bus Outputs and Bus Cycle Status (3-6) indicate the four most significant address bits during T1. These signals are active HIGH.
During T2, T3, TWand T4, the S6 pin is LOW to indicate a CPU-initiated bus cycle or HIGH to indicate a DMA initiated.
During the same T-states, S3, S4 and S5 are always LOW.</td></tr>
<tr><td>ALE_O</td><td>1</td><td>Output</td><td>Address Latch Enable, an active high signal used to strobe address information into a transparent type latch during the address phase of the bus cycle.</td></tr>
<tr><td rowspan="10">S_N_O [2:0]</td><td rowspan="10">3</td><td rowspan="10">Output</td><td>Bus cycle Status is encoded on these pins to provide bus transaction information. S [2:0] are encoded as follows:</td></tr>
<tr><td><table>
<tr><th colspan="4">Bus Cycle Status Information</th></tr>
<tr><th>S2_N</th><th>S1_N</th><th>S0_N</th><th>BUS CYCLE</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>Interrupt</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>Read I/O</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>Write I/O</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>Processor HALT</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>Queue Instruction Fetch</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>Read Memory</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>Write Memory</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>Passive (no bus activity)</td></tr>
</table></td></tr>
<tr><td>ARDY_I</td><td>1</td><td>Input</td><td>Asynchronous Ready informs the processor that the addressed memory space or I/O device will complete a data transfer. The ARDY pin accepts a rising edge that is asynchronous to CLKOUT and is active HIGH. The falling edge of ARDY must be synchronized to the processor clock. Connecting ARDY HIGH will always assert the ready condition to the CPU. If this line is unused, it should be tied LOW to yield control to the SRDY pin.</td></tr>
</table>

Signal	Width	Direction	Description
SRDY_I	1	Input	Synchronous Ready informs the processor that the addressed memory space or I/O device will complete a data transfer. The SRDY pin accepts an active-HIGH input synchronized to CLKOUT. The use of SRDY allows relaxed system timing over ARDY. This is accomplished by elimination of the one-half clock cycle required to internally synchonize the ARDY input signal.
RD_N_O	1	Output	Active low Read output signals that the accessed memory or I/O device must drive data information onto the data bus.
WR_N_O	1	Output	Active low Write output signals that data available on the data bus are to be written into the accessed memory or I/O device.
LOCK_N_O	1	Output	The processor will not service other bus requests (such as HOLD) while LOCK is active.
DT_R_N_O	1	Output	Data Transmit/Receive controls the direction of data flow through an external data bus transceiver. When LOW, data is transferred to the procesor. When HIGH the processor places write data on the data bus.
DEN_N_O	1	Output	Data Enable is provided as a data bus transceiver output enable. DEN_N is active LOW during each memory and I/O access. DEN_N is HIGH whenever DT/R_N changes state. During RESET, DEN_N is driven HIGH for one clock, then floated.
HOLD_I	1	Input	HOLD request input to signal that an external bus master wishes to gain control of the local bus.
HLDA_O	1	Output	The processor generates HLDA in response to a HOLD indicating that bus is granted. It indicates that the processor has relinquished control of the local bus.
NMI_I	1	Input	Non-Maskable Interrupt input causes a TYPE-2 interrupt to be serviced by the CPU. NMI is latched internally.

Direct Memory Access Unit Signals			
DRQ0_I, DRQ1_I	1	Input	DMA Request is asserted HIGH by an external device when it is ready for DMA Channel 0 or Channel 1 to perform a transfer. These signals are level-triggered and internally synchronized.
Interrupt Control Unit Signals			
INT0_I, INT1_SELECT_N_I,	1	Input	Maskable Interrupt Requests can be requested by activating one of these pins. Interrupt Requests are synchronized internally. INT1 is multiplexed with Slave SELECT input to the ICU (Slave mode).
INT2_INTA0_N_IO, INT3_INTA1_N_IRQ_IO	1	Input/Output	Maskable Interrupt Requests can be requested by activating one of these pins. When configured as inputs, these pins are active HIGH. Interrupt Requests are synchronized internally. INT2 and INT3 may be configured to provide active-LOW interrupt acknowledge output signals. When Slave Mode is selected, Output is Interrupt Request (IRQ) from the Controller to the external 8259A.
Timer / Counter Unit Signals			
T0OUT0_O, T1OUT1_O	1	Output	Timer output pins can be programmed to provide a single clock or continuous waveform generation, depending on the timer mode selected.
T0IN0_I, T1IN1_I	1	Input	Timer input is used either as clock or control signals, depending on the timer mode selected.
Chip Select Unit Signals			
UCS_N_O	1	Output	Upper Memory Chip Select is an active LOW output whenever a memory reference is made to the defined upper portion (1K-256K block) of memory. The address range activating UCS_N is software programmable.
LCS_N_O	1	Output	Lower Memory Chip Select is active LOW whenever a memory reference is made to the defined lower portion (1K-256K) of memory. The address range activating LCS_N is software programmable.

MCS_N_O [3:0]	4	Output	Mid-Range Memory Chip Select signals are active LOW when a memory reference is made to the defined midrange portion of memory (8K-512K). The address ranges activating MCS_N [0-3] are software programmable.
PCS_N_O [4:0]	5	Output	Peripheral Chip Select signals are active LOW when a reference is made to the defined peripheral area (64 Kbyte I/O or 1 MByte memory space). The address ranges activating PCS_N [0-4] are software programmable.
PCS5_N_A1_O	1	Output	Peripheral Chip Select 5 or Latched A1 may be programmed to provide a sixth peripheral chip select, or to provide an internally latched A1 signal. The address range activating PCS5_N is software-programmable.
PCS56_N_A2_O	1	Output	Peripheral Chip Select 6 or Latched A2 may be programmed to provide a seventh peripheral chip select, or to provide an internally latched A2 signal. The address range activating PCS6_N is software-programmable.

1.7 Timing Diagram

1.7.1 CPU Write Cycle (BUS_RDY_DISABLE = 1 and Without wait state)

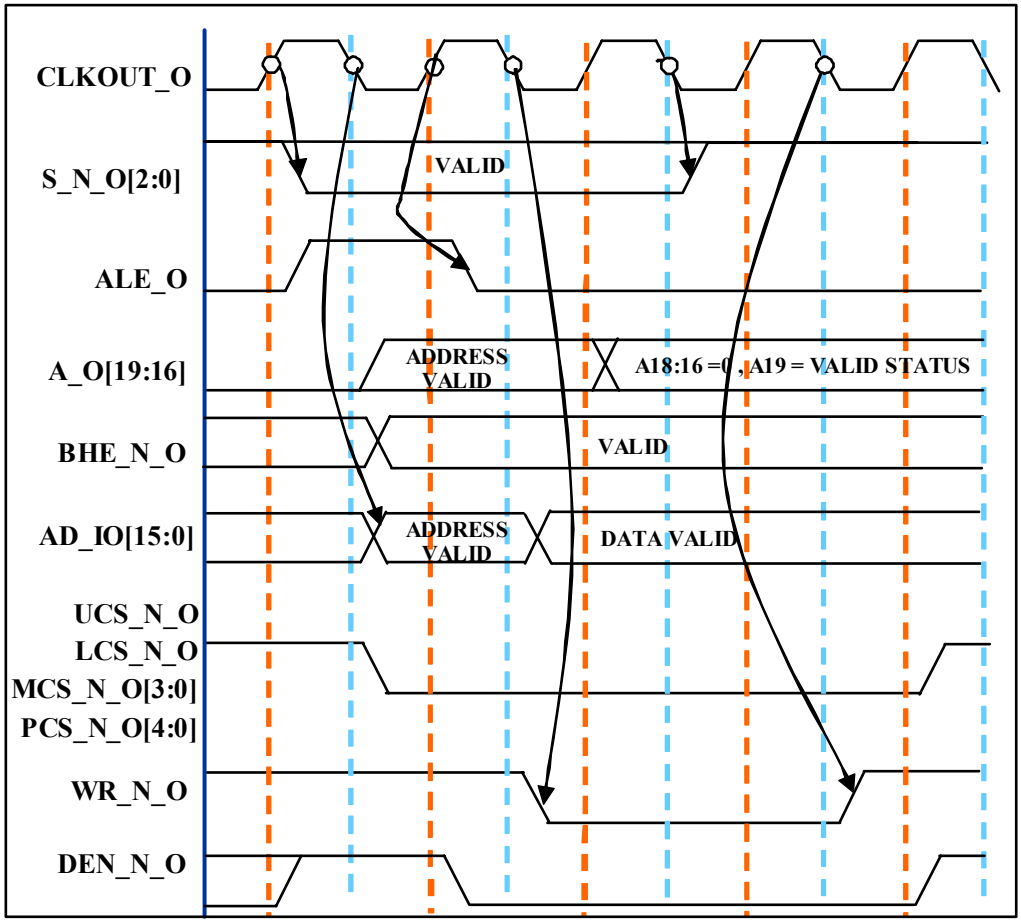

Figure 3: CPU Write Cycle (BUS_RDY_DISABLE = 1 and Without wait state)

1.7.2 CPU Write Cycle (BUS_RDY_DISABLE = 1 and With wait state)

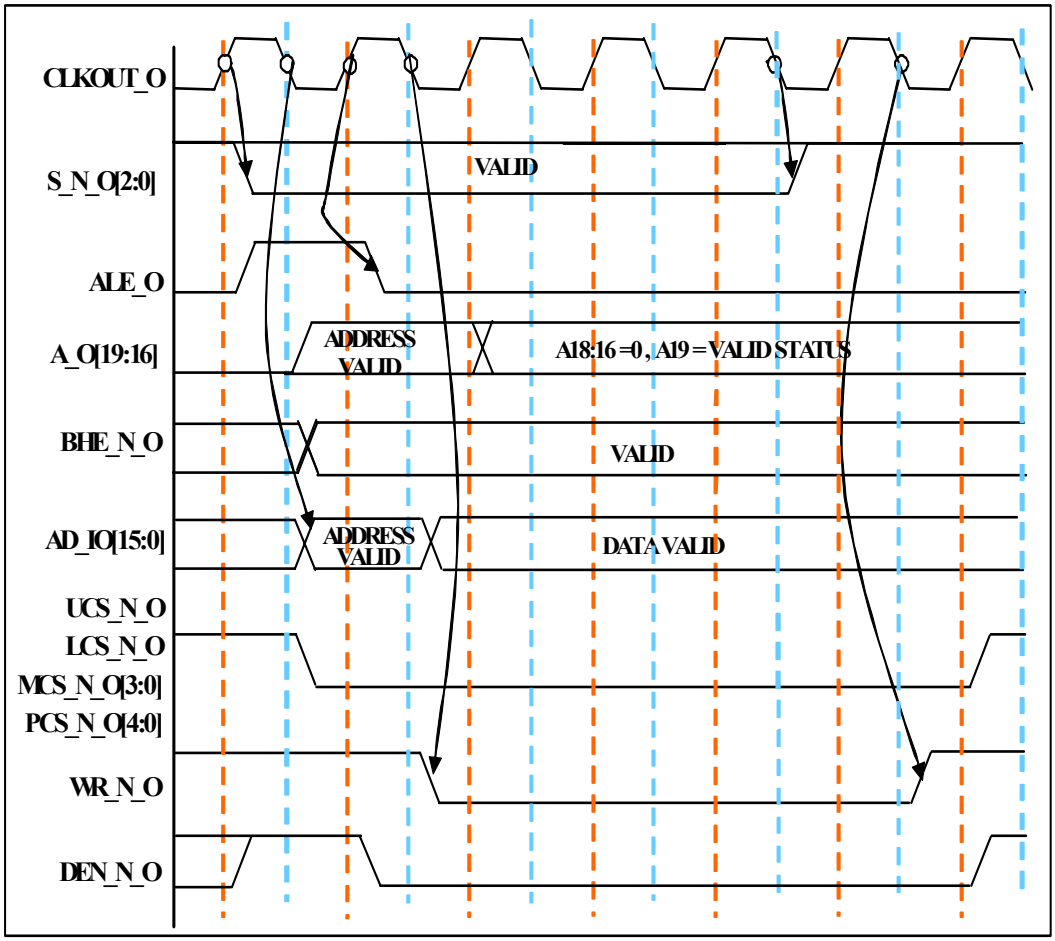

Figure 4: CPU Write Cycle (BUS_RDY_DISABLE = 1 and With wait state)

1.7.3 CPU Write Cycle (BUS_RDY_DISABLE = 0)

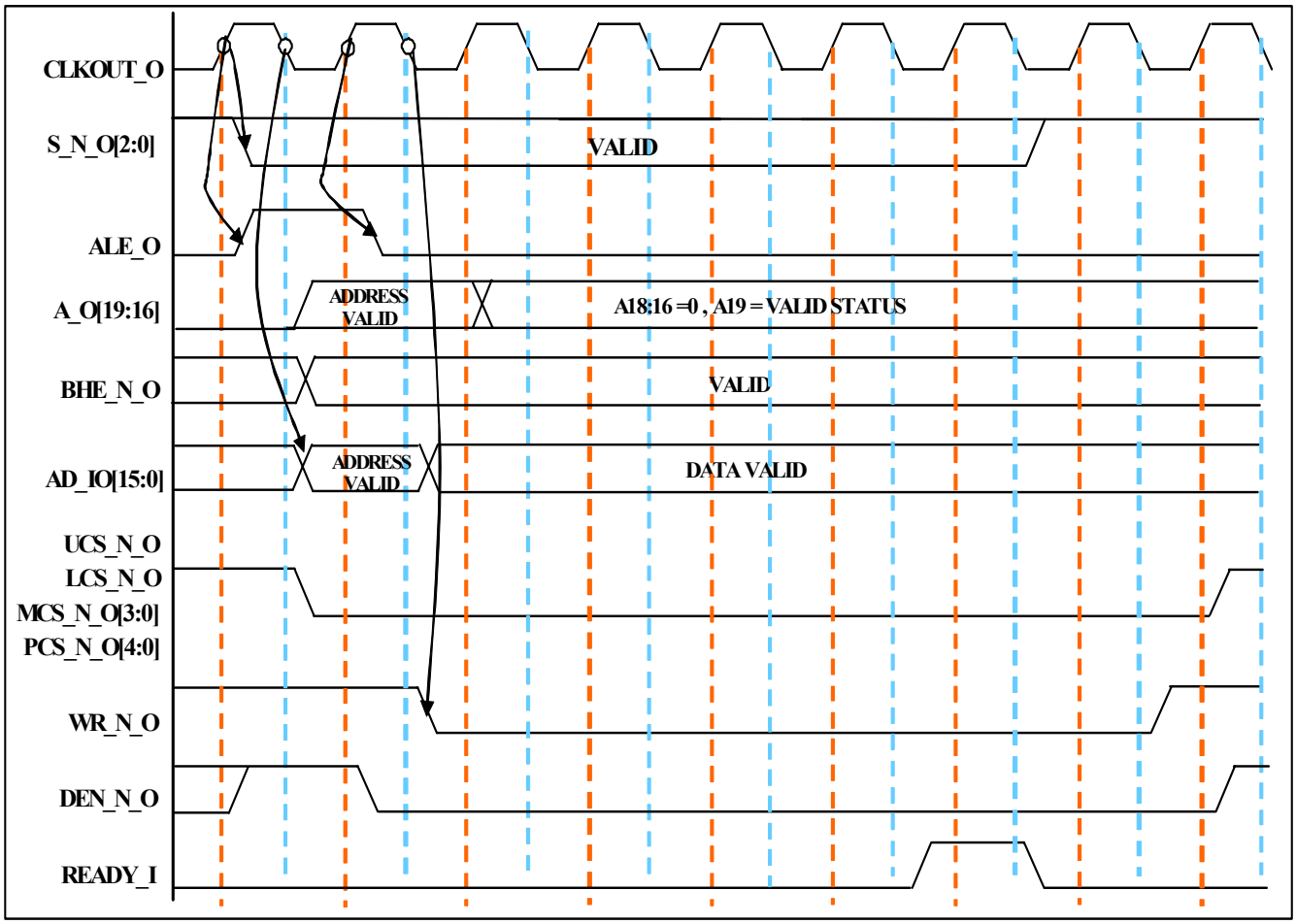

Figure 5: CPU Write Cycle (BUS_RDY_DISABLE = 0)

1.7.4 CPU Read Cycle

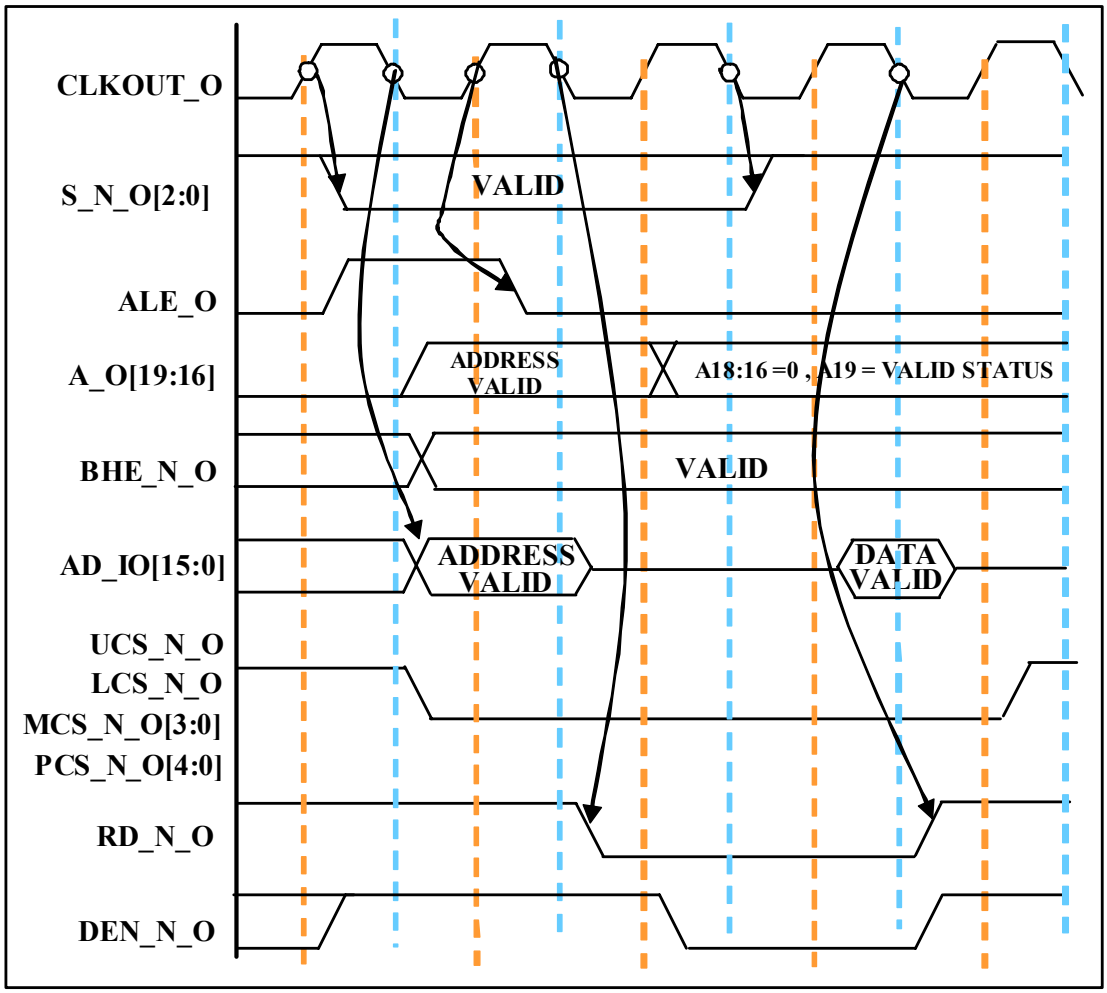

Figure 6: CPU Read Cycle

1.7.5 Interrupt Acknowledge Cycle

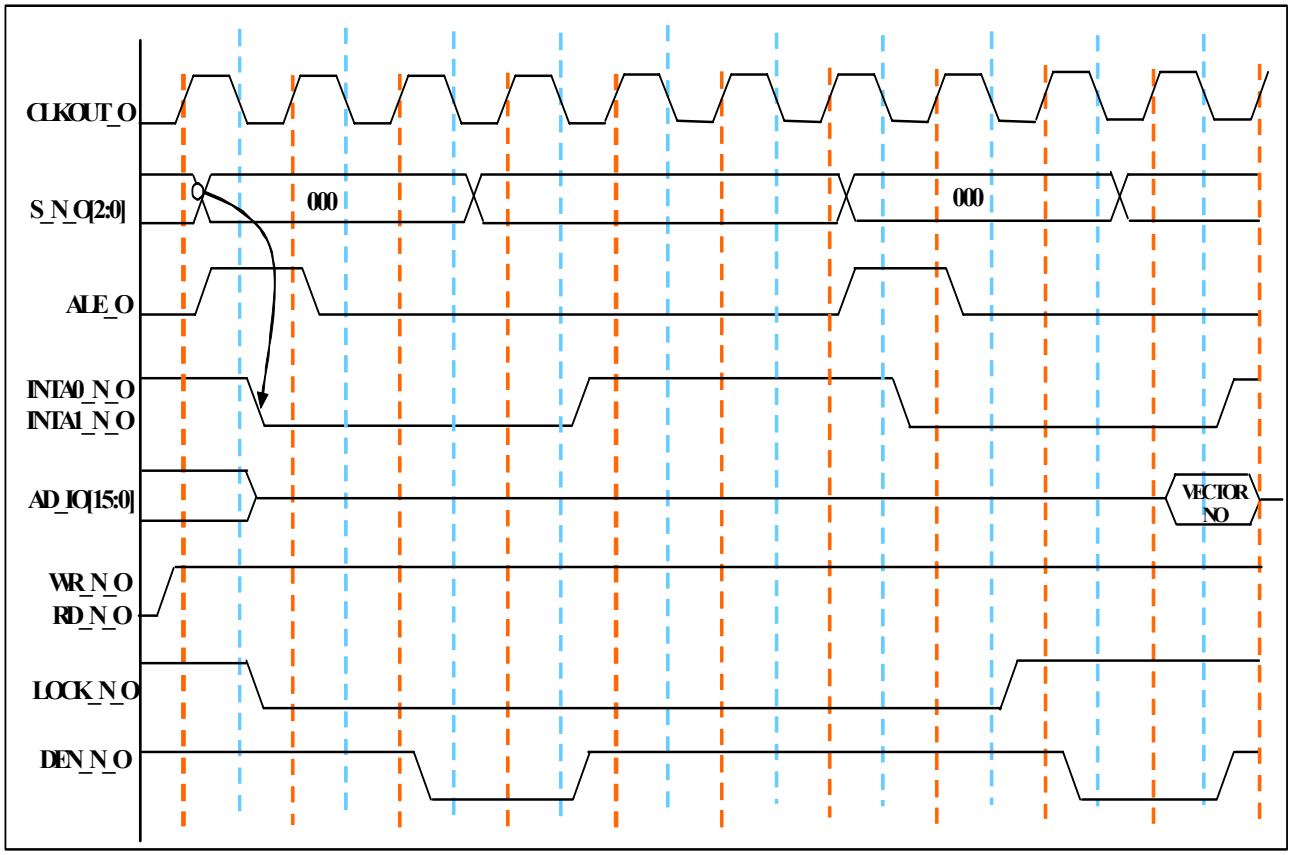

Figure 7: Interrupt Acknowledge Cycle

1.7.6 HOLD/ HLDA Cycle

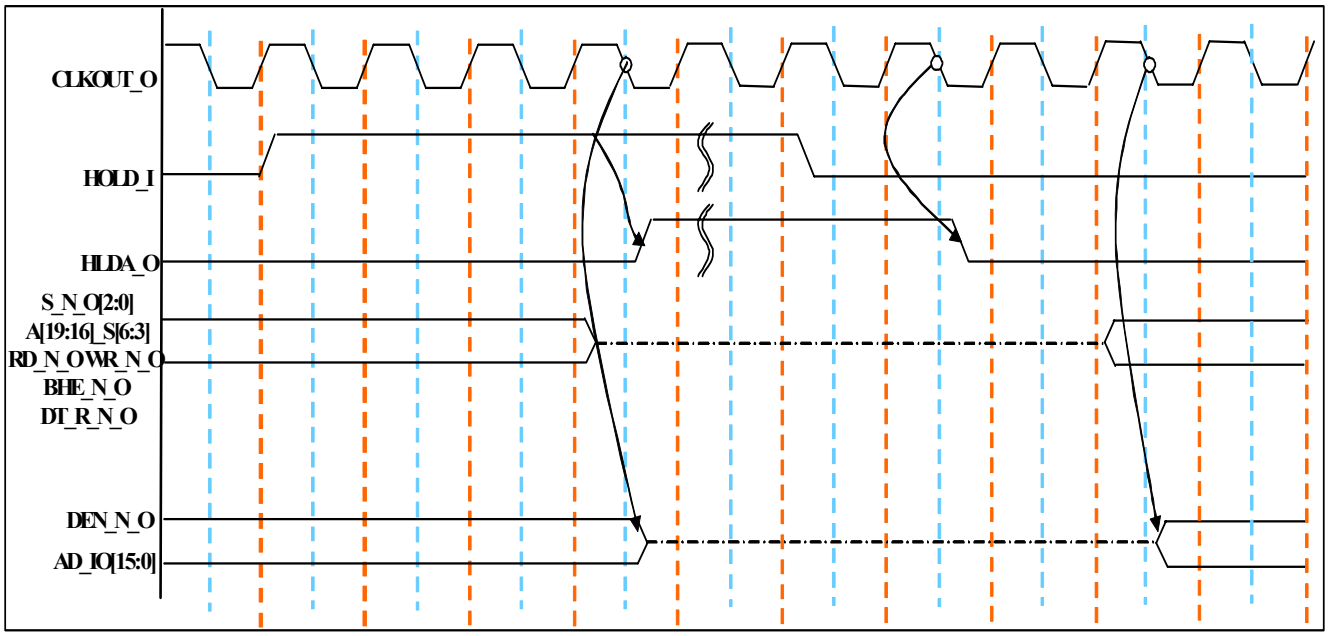

Figure 8: HOLD/ HLDA Cycle

1.7.7 HALT Cycle

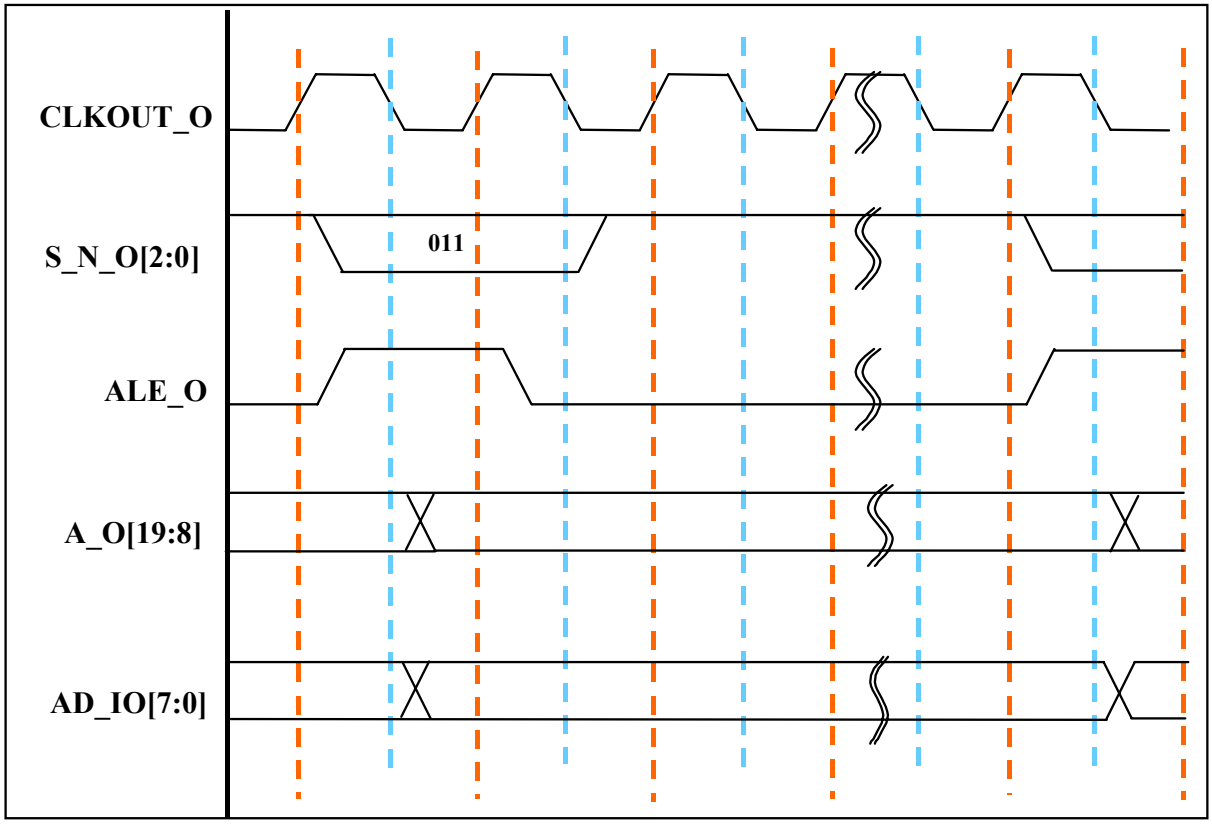

Figure 9: HALT Cycle

1.8 Instruction Set

1.8.1 Identifier and Notations

The tables below show various symbols and notations used to describe the Instruction set supported by the core.

Table 2: Operand Types Legend

Identifier	Description
reg, reg'	8/16-bit general register
reg8, reg8'	8-bit general register
regl6, regl6'	16-bit general register
dmem	8/16-bit memory location
mem	8/16-bit memory location
mem8	8-bit memory location
meml6	16-bit memory location
mem32	32-bit memory location
imm	Constant in range 0 to FFFFH
imm3	Constant in range 0 to 7
imm4	Constant in range 0 to FH
imm8	Constant in range 0 to FFH
imml6	Constant in range 0 to FFFFH
acc	AW or AL register
sreg	Segment register
src-table	Name of 256-byte conversion table
src-block	Name of block addressed by IX register
dst-block	Name of block addressed by IY register
near-proc	Procedure in current program segment
far-proc	Procedure in different program segment
near-label	Label in current program segment
short-label	Label in range -128 to +I27 bytes from end of instruction
far-label	Label in different program segment
memptrl6	Word containing offset of location in current program segment to which control is to be transferred
memptr32	Double word containing offset and segment base address of location in different program segment to which control is to be transferred
regptrl6	16-bit general register containing offset of location in different program segment to which control is to be transferred

pop-value	Number of bytes to be removed from stack (0 to 64K, normally an even number)
R	Register set
n	Shift number

Table 3: Operands Code Legend

Identifier	Description
W	Byte/word specify bit (0: byte; 1: word). However, when -1, the sign extension byte data becomes a 16-bit operand even when W=1
reg	Register field (000 to 111)
reg'	Register field (000 to 111) (Source-side register in instructions using two registers)
mem	Memory field (000 to 111)
mod	Mode field (00 to 10)
s	Sign extension specify bit (0: no sign extension, 1: sign extension)
X, XXX,YYY,ZZZ	Data for discrimination of external floating-point coprocessor, operation code

Table 4: Flag Operation Legend

Identifier	Description
(Blank)	Not Affected
0	Cleared to 0
1	Set to 1
x	Set or cleared depending upon result
U	Undefined
R	Previously saved value is restored

Table 5: 8/ 16 Bit General Register Selection

reg, reg'	W = 0	W = 1
000	AL	AW
001	CL	CW
010	DL	DW
011	BL	BW
100	AH	SP
101	CH	BP
110	DH	IX
111	BH	IY

Table 6: Segment Register Selection

sreg	Segment
00	DS1
01	PS
10	SS
11	DS0

1.8.2 Instructions

The tables below list the Instruction set supported by the iW-80186XL core. The instructions are classified and listed based on the operations they perform. The table also lists the Mnemonics, operands used, opcodes, opcode length, execution time, operations performed and the Flags affected for each of the instructions Clock Cycles field in the below table indicates the number of clock cycles utilized by the processor to execute the instructions. Pre-fetching and decoding the instructions are not included.

Data Transfer Instructions:

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
MOV	reg, reg'	1 0 0 0 1 0 1 W 1 1 reg reg'	2	2	reg ← reg'	
	mem, reg	1 0 0 0 1 0 0 W mod reg mem	2-4	4	(mem) ← reg	
	reg, mem	1 0 0 0 1 0 1 W mod reg mem	2-4	3	reg ← (mem)	
	mem, imm	1 1 0 0 0 1 1 W mod 0 0 0 mem	3-6	4	(mem) ← imm	
	reg, imm	1 0 1 1 W reg	2-3	2	reg ← imm	
	acc, dmem	1 0 1 0 0 0 0 W	3	3	If W = 0: AL ← (dmem) If W=l: AH ← (dmem + 1). AL ← (dmem)	
	dmem, acc	1 0 1 0 0 0 1 W	3	4	If W=0: (dmem) ← AL If W=l: (dmem + 1) ← AH (dmem) ← AL	

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
MOV	sreg, mem16	1 0 0 0 1 1 1 0 mod 0sreg mem	2-4	3	sreg ← (mem16) sreg:SS, DS0, DS1	
	reg16, sreg	1 0 0 0 1 1 0 0 1 1 0sreg reg	2	3	reg16← sreg	
	mem16, sreg	1 0 0 0 1 1 0 0 mod 0sreg mem	2-4	3	(mem16)← sreg	
LDS	regl6, mem32	1 1 0 0 0 1 0 1 mod reg mem	2-4	5	reg16← (mem32)	
LES	reg16, mem32	1 1 0 0 0 1 0 0 mod reg mem	2-4	5	reg16 ← (mem32)	
LAHF	AH, PSW	1 0 0 1 1 1 1 1	1	2	AH←S, Z, x, AC, x, P, x, CY	
SAHF	PSW, AH	1 0 0 1 1 1 1 0	1	2	S, Z, x, AC, x, P, x, CY←AH	x x x x x
LEA	regl6, meml6	1 0 0 0 1 1 0 1 mod reg mem	2-4	3	reg16 ←meml6	
XLAT	src-table	1 1 0 1 0 1 1 1	1	3	AL ← (BW +AL)	
XCHG	reg, reg'	1 0 0 0 0 1 1 W 1 1 reg reg'	2	4	reg ↔ reg'	
	mem, reg reg, mem	1 0 0 0 0 1 1 W mod reg mem	2-4	6	(mem) ↔ reg	
	AW, reg16 reg16, AW	1 0 0 1 0 reg	1	4	AW ↔ regl6	

Repeat Prefixes:

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
REPC		0 1 1 0 0 1 0 1	1	2	While CW≠ 0, the following byte primitive block transfer instruction is executed and CW is decremented (-1). If there is a pending interrupt, it is serviced. If CY ≠ 1 the loop is exited.	
REPNC		0 1 1 0 0 1 0 0	1	2	Same as above If CY ≠ 0 the loop is exited.	
REP, REPE, REPZ		1 1 1 1 0 0 1 1	1	2	While CW ≠ 0, the following byte primitive block transfer instruction is executed and CW is decremented (-1). If there is a pending interrupt, it is serviced. If the primitive block transfer instruction is CMPBK or CMPM and Z ≠ 1 the loop is exited.	

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
REPNE, REPNZ		1 1 1 1 0 0 1 0	1	2	Same as above If Z ≠ 0 the loop is exited.	

Primitive Block Transfer Instructions:

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
MOVS	dst-block, src-block	1 0 1 0 0 1 0 W	1	7*rep (7)	If W = 0: (IY) ← (IX) DIR = 0: IX ←IX + l, IY ← IY + l DIR = l: IX ←IX - 1, IY ← IY - 1	
					I f W = l: (IY+l, IY)←(IX+l, IX) DIR = 0: IX ← IX + 2, IY ← IY + 2 DIR = 1: IX ← IX - 2, IY ← IY - 2	
CMPS	src-block, dst-block	1 0 1 0 0 1 1 w	1	9*rep (9)	If W = 0: (IX) - (IY) DIR = 0: IX ← IX + 1, IY ← IY + 1 DIR = l: IX ← IX - 1, IY ← IY - 1	x x x x x x

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
					If W = 1: (IX+1,IX) - (IY+1,IY) DIR = 0: IX ← IX + 2, IY ← IY + 2 DIR = 1: IX ←IX - 2, IY ←IY - 2	
SCAS	dst-block	1 0 1 0 1 1 1 W		8*rep (8)	If W = 0: AL - (IY) DIR = 0: IY ← IY + 1; DIR = 1: IY ← IY – 1; If W = 1: AW - (IY + 1, IY) DIR = 0: IY ← IY + 2 DIR = 1: IY ← IY - 2	x x x x x x
LODS	src-block	1 0 1 0 1 1 0 W	1	6*rep (5)	If W = 0: AL ← (IX) DIR = 0: IX ← IX + 1 DIR = 1: IX ← IX - 1 If W = 1: AW ← (IX + 1. IX) DIR = 0: IX ← IX + 2 DIR = 1: IX ← IX - 2	

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
STOS	dst-block	1 0 1 0 1 0 1 W	1	5*rep (5)	If W = 0: (IY) ← AL DIR = 0: IY ← IY + 1 DIR = 1: IY ← IY - 1	

Note:"rep" indicates the number of repetitions.One execution is equivalent to one repetition Values in brackets gives no.of clock cycles when used without repeat prefix.

Input/Output Instructions:

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
IN	acc, imm8	1 1 1 0 0 1 0 W	2	3	If W = 0: AL ← (imm8) If W= 1: AH ← (imm8+ 1), AL ← (imm8)	
	acc, DW	1 1 1 0 1 1 0 W	1	3	If W= 0: AL ← (DW) If W = 1: AH ← (DW + I), AL ← (DW)	
OUT	imm8, acc	1 1 1 0 0 1 1 W	2	3	If W = 0: (imm8) ← AL If W= 1: (imm8+ 1) ← AH, (imm8) ← AL	

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
	DW, acc	1 1 1 0 1 1 1 W	1	3	If W = 0: (DW)← AL If W = 1: (DW + 1) ← AH, (DW) ← AL	

Primitive Input/ Output Instructions:

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
INS	dst-block, DW	0 1 1 0 1 1 0 W	1	8*rep (8)	If W = 0: (IY) ←(DW) DIR = 0: IY ←IY + 1 DIR = 1: IY ←IY - 1	
					IfW=1: (IY+l,IY) ← (DW+l, DW) DIR = 0: IY ←IY + 2 DIR = l : IY ←IY - 2	
OUTS	DW, src-block	0 1 1 0 1 1 1 W	1	6*rep (6)	If W = 0: (DW) ← (IX) DIR = 0: IX ← IX + 1 DIR = l: IX ←IX – 1	
					If W= 1: (DW+1, DW) ← (IX+1, IX) DIR = 0: IX← IX + 2 DIR = l: IX ← IX - 2	

Note: "rep" indicates the number of repetitions.One execution is equivalent to one repetition
Values in brackets gives no.of clock cycles when used without repeat prefix.

Addition/ Subtraction Instructions:

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
ADD	reg, reg'	0 0 0 0 0 0 1 W 1 1 reg reg'	2	5	reg ←reg + reg'	x x x x x x
	mem, reg	0 0 0 0 0 0 0 W mod reg mem	2-4	7	(mem) ←(mem) + reg	x x x x x x
	reg, mem	0 0 0 0 0 0 1 W mod reg mem	2-4	6	reg ←reg + (mem)	x x x x x x
	reg, imm	1 0 0 0 0 0 s W 1 1 0 0 0 reg	3-4	5	reg ←reg + imm	x x x x x x
	mem, imm	1 0 0 0 0 0 s W mod 0 0 0 mem	3-6	7	(mem) ←(mem) + imm	x x x x x x
	acc, imm	0 0 0 1 0 1 0 W	2-3	5	If W=0: AL← AL+imm +CY If W=l: AW←AW+imm +CY	x x x x x x
ADDC	reg, reg'	0 0 0 1 0 0 1 W 1 1 reg reg'	2	5	reg ←reg + reg'+ CY	x x x x x x
	mem, reg	0 0 0 1 0 0 0 W mod reg mem	2-4	7	(mem)←(mem)+ reg + CY	x x x x x x
	reg, mem	0 0 0 1 0 0 1 W mod reg mem	2-4	6	reg←reg+(mem) +CY	x x x x x x
	reg, imm	1 0 0 0 0 0 s W 1 1 0 1 0 reg	3-4	5	reg← reg+imm+CY	x x x x x x
	mem, imm	1 0 0 0 0 0 s W mod 0 l 0 mem	3-6	7	(mem)← (mem) + imm + CY	x x x x x x
	acc, imm	0 0 0 1 0 1 0 W	2-3	5	If W=0: AL← AL+imm+CY If W=l: AW←AW+imm +CY	x x x x x x
SUB	reg, reg'	0 0 1 0 1 0 1 W 1 1 reg reg'	2	5	reg ← reg - reg'	x x x x x x

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
	mem, reg	0 0 1 0 1 0 0 W mod reg mem	2-4	7	(mem) ← (mem) - reg	x x x x x x
	reg, mem	0 0 1 0 1 0 1 W mod reg mem	2-4	6	reg ← reg - (mem)	x x x x x x
	reg, imm	1 0 0 0 0 0 s W 1 1 1 0 1 reg	3-4	5	reg ← reg -imm	x x x x x x
	mem, imm	1 0 0 0 0 0 s W mod 1 0 1 mem	3-6	7	(mem) ← (mem) - imm	x x x x x x
	acc, imm	0 0 1 0 1 1 0 W	2-3	5	IfW = 0: AL ← AL - imm IfW = l: AW ← AW-imm	x x x x x x
SUBC	reg, reg'	0 0 0 1 1 0 1 W 1 1 reg reg'	2	5	reg ←reg -reg'-CY	x x x x x x
	mem, reg	0 0 0 1 1 0 0 W mod reg mem	2-4	7	(mem) ← (mem) – reg - CY	x x x x x x
	reg, mem	0 0 0 1 1 0 1 W mod reg mem	2-4	6	reg ← reg-(mem)-CY	x x x x x x
	reg, imm	1 0 0 0 0 0 s W 1 1 0 1 1 reg	3-4	5	reg ← reg - imm -CY	x x x x x x
	mem, imm	1 0 0 0 0 0 s W mod 0 1 l mem	3-6	7	(mem) ← (mem) – imm - CY	x x x x x x
	acc, imm	0 0 0 1 1 1 0 W	2-3	5	If W = 0: AL ←AL-imm-CY If W = 1: AW ←AW-imm-CY	x x x x x x

Increment/ Decrement Instructions:

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
INC	reg8	1 1 1 1 1 1 1 0 1 1 0 0 0 reg	2	5	Reg8 ← reg8 + 1	x x x x x
	mem	1 1 1 1 1 1 1 W mod 0 0 0 mem	2-4	7	(mem) ← (mem) + 1	x x x x x
	reg16	0 1 0 0 0 reg	1	5	regl6 ← regl6 + 1	x x x x x
DEC	reg8	1 1 1 1 1 1 1 0 1 1 0 0 1 reg	2	5	reg8 ← reg8 - 1	x x x x x
	mem	1 1 1 1 1 1 1 W mod 0 0 1 mem	2-4	7	(mem) ← (mem) - 1	x x x x x
	reg16	0 1 0 0 1 reg	1	5	regl6 ← regl6 + 1	x x x x x

Multiplication Instructions:

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
MUL	reg8	1 1 1 1 0 1 1 0 1 1 1 0 0 reg	2	7	AW ← AL × reg8 AH = 0: CY← 0, V ← 0 AH ≠ 0: CY← 1, V ← 1	U x x U U U
	mem8	1 1 1 1 0 1 1 0 mod1 0 0 mem	2-4	8	AW← AL× (mem8) AH = 0: CY← 0, V ← 0 AH ≠ 0: CY← l, V ← 1	U x x U U U
	reg16	1 1 1 1 0 1 1 1 1 1 1 0 0 reg	2	8	DW,AW← AW×(regl6) DW = 0: CY←0, V←0 DW ≠ l: CY ← l, V ←l	U x x U U U

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
	mem16	1 1 1 1 0 1 1 1 mod1 0 0 mem	2-4	9	DW, AW←AW×(meml6) DW = 0: CY←0, V←0 DW≠ l: CY ← l, V ← l	U x x U U U
IMUL	reg8	1 1 1 1 0 1 1 0 1 1 1 0 1 reg	2	7	AW ← A L × reg8 AH = AL sign extension: CY ← 0, V ← 0 AH ≠AL sign extension: CY ←1, V ←1	U x x U U U
	mem8	1 1 1 1 0 1 1 0 mod1 0 1 mem	2-4	8	AW←AL×(mem8) AH = AL sign extension: CY ← 0. V ← 0 AH ≠ AL sign extension: CY ←1, V ←1	U x x U U U
	reg16	1 1 1 1 0 1 1 1 1 1 1 0 1 reg	2	8	DW, AW ← AW × reg16; DW = AW sign extension: CY ← 0, V ← 0 DW ≠ AW sign extension: CY ← 1. V ←1	U x x U U U
	mem16	1 1 1 1 0 1 1 1 mod1 0 1 mem	2-4	9	DW, AW ← AW × (meml6); DW = AW sign extension: CY ← 0, V ← 0 DW ≠ AW sign extension: CY ← 1, V ← 1	U x x U U U

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
	reg16, (reg16') imm8	0 1 1 0 1 0 1 1 1 1 reg reg'	3	7	reg16 ← reg16' × imm8; Product ≤ 16 bits: CY ← 0, V ← 0 Product > 16 bits: CY ← 1, V ← 1	U x x U U U
	reg16, meml6, imm8	0 1 1 0 1 0 1 1 mod reg mem	3-5	8	reg16 ← (meml6) × imm8; Products ≤ 16 bits: CY ← 0, V ← 0 Product > 16 bits: CY ← 1, V ← 1	U x x U U U
	regl6, (reg16') imml6	0 1 1 0 1 0 0 1 1 1 reg reg'	4	7	reg16 ← regl6' × imml6; Product ≤ 16 bits: CY ←0, V ← 0 Product> 16 bits: CY ←1, V ←1	U x x U U U
	regl6, meml6, imml6	0 1 1 0 1 0 0 1 mod reg mem	4-6	8	regl6 ← (meml6) × imml6; Product ≤ 16 bits: CY ← 0, V ← 0 Product > 16 bits: CY ← 1, V ← 1	U x x U U U

Unsigned Division Instructions:

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
DIV	reg8	1 1 1 1 0 1 1 0 1 1 1 1 0 reg	2	25	temp ← AW If temp + reg8 ≤FFH AH ← temp % reg8, AL ← temp + reg8 If temp + reg8 > FFH TA ← (001H. 000H), TC ← (003H, 002H) SP ← SP-2, (SP+l, SP)←PSW, IE←0. BRK←0 SP ← SP - 2, (SP+l,SP) ← PS, PS ←TC S P ← S P - 2 , (SP+l, SP) ← PC, PC ← TA	U UU U U U

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
	mem8	1 1 1 1 0 1 1 0 mod1 1 0 mem	2-4	26	temp← AW If temp + (mem8) ≤ FFH AH ←temp%(mem8), AL ←temp + (mem8) If temp + (mem8) > FFH TA ← (001H, 000H), TC ← (003H, 002H) S P ←S P - 2, (SP+l, SP) ← PSW, IE←0, BRK←0, SP ← SP - 2 , (SP+l, SP) ← PS, PS←TC SP ← SP - 2, (SP + 1. SP) ←PC, PC ←TA	U UU U U U

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
	reg16	1 1 1 1 0 1 1 1 1 1 1 1 0 reg	2	41	temp ← DW, AW If temp + reg16 ≤FFFFH DW ← temp%regl6, AW ← temp + reg16 If temp + reg16 > FFFFH TA ← (001 H, 000H), TC ← (003H, 002H) S P ← S P - 2, (SP+l, SP) ← PSW, IE←0, BRK←0 SP ←SP - 2, (SP+l, SP) ← PS, PS←TC SP← SP - 2, (SP+1, SP) ← PC, PC←TA	U UU U U U

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
	mem16	1 1 1 1 0 1 1 1 mod1 1 0 mem	2-4	42	temp ← DW, AW If temp + (meml6) ≤FFFFH DW←temp%(meml6) AW←temp +(meml6) If temp + (meml6) > FFFFH TA ← (001 H, 000H), TC ← (003H, 002H) SP ← SP-2, (SP+l, SP) ← PSW, IE←0, BRK←0 SP ← SP - 2, (SP+l, SP) ← PS, PS ←TC SP ← SP-2, (SP+1, SP) ←PC, PC ←TA	U UU U U U

Signed Division Instructions:

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
IDIV	reg8	1 1 1 1 0 1 1 0 1 1 1 1 1 reg	2	25	temp ← AW If temp + reg8 >0 and temp + reg8 ≤ 7FH or temp + reg8 < 0 and temp + reg8 > 0-7F-1 AH ← temp % reg8, AL ← temp + reg8 If temp + reg8 > 0 and temp + reg8 > 7FH or temp + reg8 < 0 and temp + reg8 ≤ 0-7F-1 TA ← (001H. 000H), TC ← (003H, 002H) SP ← SP-2, (SP+l, SP) ← PSW, IE←0. BRK←0 SP ← SP - 2, (SP+l, SP) ← PS, PS ← TC SP ← SP - 2, (SP+l, SP) ← PC, PC←TA	U UU U U U

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
	mem8	1 1 1 1 0 1 1 0 mod 1 1 1mem	2-4	26	temp ← AW If temp + (mem8) >0 and temp + (mem8) ≤7FH or temp + (mem8) < 0 and temp + (mem8) > 0-7F-1 AH←temp%(mem8), AL← temp + (mem8) If temp + (mem8) > 0 and temp + (mem8) > 7FH or temp + (mem8) < 0 and temp + (mem8) ≤ 0-7FH-1 TA ← (001H. 000H), TC ← (003H, 002H) SP ← SP-2, (SP+l, SP) ← PSW, IE←0. BRK← 0 SP ← SP - 2, (SP+l, SP) ← PS, PS ← TC SP ← SP - 2, (SP+l, SP) ← PC, PC ← TA	U UU U U U

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
	reg16	1 1 1 1 0 1 1 1 1 1 1 1 1 reg	2	41	temp ← DW, AW If temp + reg16 >0 and temp + reg16 ≤7FFFH or temp + reg16 < 0 and temp + reg16 > 0-7FFFH-1 DW← temp % reg16, AW← temp + reg16 If temp + reg16 > 0 and temp + reg16 > 7FFFH or temp + reg16 < 0 and temp + reg16 ≤ 0-7FFFH-1 TA ← (001H. 000H), TC ← (003H, 002H) SP ← SP-2, (SP+l,SP) ← PSW, IE←0. BRK←0 SP ← SP - 2, (SP+l, SP) ← PS, PS ← TC SP ← SP - 2, (SP+l, SP) ← PC, PC ← TA	U UU U U U

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
	mem16	1 1 1 1 0 1 1 1 mod 1 1 1 mem	2-4	42	temp ← DW, AW If temp + (mem16) >0 and temp + (mem16) ≤7FFFH or temp + (mem16) < 0 and temp + (mem16) > 0-7FFFH-1 DW ← temp % (mem16), AW ← temp + (mem16) If temp + (mem16) > 0 and temp + (mem16) > 7FFFH or temp + (mem16) < 0 and temp + (mem16) ≤ 0-7FFFH-1 TA ← (001H. 000H), TC ← (003H, 002H) SP ← SP-2, (SP+l, SP) ← PSW, IE←0. BRK←0 SP ← SP - 2, (SP+l, SP) ← PS, PS ← TC SP ← SP - 2, (SP+l, SP) ← PC, PC ← TA	U UU U U U

BCD Adjustment Instructions:

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
AAA		0 0 1 1 0 1 1 1	1	4	If AL & 0FH > 9 or AC = 1: AL ←AL+6, AH ← AH+1, AC ← 1, CY ← AC, AL ← AL & 0FH	x x U U U U
DAA		0 0 1 0 0 1 1 1	1	4	If AL & 0FH > 9 or AC = 1: AL ←AL+6, If AL > 9F or CY=1 AL ← AL + 60H, CY ← 1	x x U x x x
AAS		0 0 1 1 1 1 1 1	1	4	If AL & 0FH > 9 or AC = 1: AL ←AL+6, AH ← AH -1 AC ←1 CY ← AC, AL ← AL & 0FH	x x U U U U
DAS		0 0 1 0 1 1 1 1	1	4	If AL & 0FH > 9 or AC = 1: AL ←AL-6, AC ←1 If AL > 9F or CY =1 AL ← AL – 60H, CY ← 1	x x U x x x

Data Conversion Instructions:

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
AAM		1 1 0 1 0 1 0 0 0 0 0 0 1 0 1 0	2	25	AH ← AL+ 0AH, AL ← AL % 0AH	U U U x x x
AAD		1 1 0 1 0 1 0 1 0 0 0 0 1 0 1 0	2	12	AL ← 0 AL ← AH×0A + AL	U U U x x x
CBW		1 0 0 1 1 0 0 0	1	2	If AL < 80H: AH ← 0 Otherwise: AH← FFH	
CWD		1 0 0 1 1 0 0 1	1	2	If AW < 8000H: DW ← 0 Otherwise: AH← FFFFH	

Comparison Instructions:

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
CMP	reg, reg’	0 0 1 1 1 0 1W 1 1 reg reg’	2	4	reg – reg’	x x x x x x
	mem, reg	0 0 1 1 1 0 0W mod reg mem	2-4	5	(mem) – reg	x x x x x x
	reg, mem	0 0 1 1 1 0 1W mod 1 1 1 mem	2-4	5	reg- (mem)	x x x x x x
	reg, imm	1 0 0 0 0 0 s W 1 1 1 1 1 reg	3-4	4	reg – imm	x x x x x x
	mem, imm	1 0 0 0 0 0 s W mod 1 1 1 mem	3-6	5	(mem)-imm	x x x x x x
	acc, imm	0 0 1 1 1 1 0 W	2-3	4	If W = 0: AL – imm If W = 1: AW - imm	x x x x x x

Complement Operation Instructions:

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
NOT	reg	1 1 1 1 0 1 1 W 1 1 0 1 0 reg	2	5	reg ← ~reg	
	mem	1 1 1 1 0 1 1 W 1 1 0 1 0 reg	2-4	7	(mem) ← ~ (mem)	
NEG	reg	1 1 1 1 0 1 1 W 1 1 0 1 1 reg	2	5	reg ← ~reg + 1	x x x x x x
	mem	1 1 1 1 0 1 1 W mod 0 1 1 mem	2-4	7	(mem) ← ~(mem)+ 1	x x x x x x

Logical Operation Instructions:

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
TEST	reg, reg’	1 0 0 0 0 1 0 W mod reg mem	2-4	4	reg & reg’	U 0 0 x x x
	mem, reg reg, mem	1 0 0 0 0 1 0 W mod reg mem	2-4	5	(mem)& reg	U 0 0 x x x
	reg, imm	1 1 1 1 0 1 1 W 1 1 0 0 0 reg	3-4	4	reg & imm	U 0 0 x x x
	mem, imm	1 1 1 1 0 1 1 W mod reg mem	3-6	5	(mem) & imm	U 0 0 x x x
	acc, imm	0 0 1 0 0 1 0 W	2-3	4	If W =0:AL&imm8 If W=1:AL&imm16	U 0 0 x x x
AND	reg, reg’	0 0 1 0 0 0 1 W 1 1 reg reg’	2	5	reg ← reg & reg’	U 0 0 x x x
	mem, reg	0 0 1 0 0 0 0 W mod reg mem	2-4	7	(mem) ← (mem) & reg	U 0 0 x x x
	reg, mem	0 0 1 0 0 0 1 W mod reg mem	2-4	6	reg ← reg & (mem)	U 0 0 x x x
	reg, imm	1 0 0 0 0 0 0 W 1 1 1 0 0 reg	3-4	5	reg ← reg & imm	U 0 0 x x x

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
	mem, imm	1 0 0 0 0 0 0 W mod1 0 0 mem	3-6	7	(mem) ← (mem) & imm	U 0 0 x x x
	acc, imm	0 0 1 0 0 1 0 W	2-3	5	If W = 0, AL ← AL & imm8 If W=1, AW ← AW & imm16	U 0 0 x x x
OR	reg, reg'	0 0 0 0 1 0 1 W 1 1 reg reg'	2	5	reg ← reg | reg'	U 0 0 x x x
	mem, reg	0 0 0 0 1 0 0 W mod reg mem	2-4	7	(mem) ← (mem)| reg	U 0 0 x x x
	reg, mem	0 0 0 0 1 0 1 W mod reg mem	2-4	6	reg ← reg | (mem)	U 0 0 x x x
	reg, imm	1 0 0 0 0 0 0 W 1 1 0 0 1 reg	3-4	5	reg ← reg | imm	U 0 0 x x x
	mem, imm	1 0 0 0 0 0 0 W mod 0 0 1 mem	3-6	7	(mem) ← (mem)| imm	U 0 0 x x x
	acc, imm	0 0 0 0 1 1 0 W	2-3	5	If W = 0 AL ← AL| imm8 If W=1 AW ← AW | imm16	U 0 0 x x x
XOR	reg, reg'	0 0 1 1 0 0 1 W 1 1 reg reg'	2	5	reg ← reg xor reg'	U 0 0 x x x
	mem, reg	0 0 1 1 0 0 0 W mod reg mem	2-4	7	(mem) ← (mem) xor reg	U 0 0 x x x
	reg, mem	0 0 1 1 0 0 1 W mod reg mem	2-4	6	reg ← reg xor (mem)	U 0 0 x x x
	reg, imm	1 0 0 0 0 0 0 W 1 1 1 1 0 reg	3-4	5	reg ← reg xor imm	U 0 0 x x x
	mem, imm	1 0 0 0 0 0 0 W mod 1 1 0 mem	3-6	7	(mem) ← (mem)xor imm	U 0 0 x x x
	acc, imm	0 0 1 1 0 1 0 W	2-3	5	If W = 0 AL ← AL xor imm8 If W=1 AW← AW xor imm16	U 0 0 x x x

Bit Manipulation Instructions:

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
CMC	CY	1 1 1 1 0 1 0 1	1	6	CY← CY	x
CLC	CY	1 1 1 1 1 0 0 0	1	6	CY ← 0	0
CLD	DIR	1 1 1 1 1 1 0 0	1	6	DIR ← 0	
STC	CY	1 1 1 1 1 0 0 1	1	6	CY ← 1	1
STD	DIR	1 1 1 1 1 1 0 1	1	6	DIR ← 1	

Shift Instructions:

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
SHL	reg, 1	1 1 0 1 0 0 0 W 1 1 1 0 0 reg	2	6	CY ← reg MSB, reg ← reg × 2 If reg MSB ≠ CY: V ← 1 If reg MSB = CY: V ←0	U x x x x x

	mem.1	1 1 0 1 0 0 0 W
		mod 1 0 0 mem
			2-4	8	CY←(mem) MSB,
					(mem) ←(mem)×2
					If (mem) MSB
					≠CY: V ← 1
					If (mem) MSB =
					CY: V ← 0
						U x x x x x

	reg, CL	1 1 0 1 0 0 1 W 1 1 1 0 0 reg	2	5+n	temp ← CL, while temp ≠ 0 the following operation are repeated: CY ← reg MSB, reg ← reg × 2 temp ← temp - 1	U x U x x x

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
	mem, CL	1 1 0 1 0 0 1 W mod 1 0 0 mem	2-4	7+n	temp ← CL, while temp ≠O the following operation are repeated: CY ← (mem) MSB, (mem) ← (mem) × 2 temp ← temp - 1	U x U x x x
	reg, imm8	1 1 0 0 0 0 0 W 1 1 1 0 0 reg	3	5+n	temp ←imm8, while temp ≠0 the following operations are repeated: CY ←reg MSB, reg ← reg × 2 temp ← temp - 1	U x U x x x
	mem, imm8	1 1 0 0 0 0 0 W mod 1 0 0 mem	3-5	7+n	temp ←imm8, while temp ≠0 the following operations are repeated: CY ← (mem) MSB, (mem) ← (mem) ×2 temp ←temp - 1	U x U x x x
SHR	reg, 1	1 1 0 1 0 0 0 W 1 1 1 0 1 reg	2	6	CY ← reg LSB, reg ← reg + 2 If reg MSB ≠ bit after reg MSB: V ←1 If reg MSB ≠ bit after reg MSB: V ←0	U x x x x x

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
	mem, 1	1 1 0 1 0 0 0 W mod 1 0 1 mem	2-4	8	CY ← (mem) LSB, (mem) ←(mem) + 2 If (mem) MSB ≠ bit after (mem) MSB: V ←1 If (mem) MSB ≠ bit after (mem) MSB: V ←0	U x x x x x
	reg, CL	1 1 0 1 0 0 1 W 1 1 1 0 1 reg	2	5+n	temp ←CL, while temp ≠0 the following operations are repeated: CY ←reg LSB, reg ←reg + 2, temp ←temp -1	U x U x x x
	mem, CL	1 1 0 1 0 0 1 W mod 1 0 1 mem	2-4	7+n	temp ←CL, while temp ≠0 the following operations are repeated: CY ←(mem) LSB, (mem) ←(mem) + 2, temp ←temp -1	U x U x x x
	reg, imm8	1 1 0 0 0 0 0 W 1 1 1 0 1 reg	3	5+n	temp ←imm8, while temp ≠ 0the following operations are repeated: CY ←reg LSB, reg ←reg + 2, temp ←temp -1	U x U x x x

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
	mem, imm8	1 1 0 0 0 0 0 W mod 1 0 1 mem	3-5	7+n	temp ←imm8, while temp ≠ 0the following operations are repeated: CY ←(mem) LSB, (mem) ←(mem) + 2, temp ←temp -1	U x U x x x
SHRA	reg, 1	1 1 0 1 0 0 0 W 1 1 1 1 1 reg	2	6	CY ← reg LSB, reg ← reg + 2, V ← 0 MSB of the operand is unchanged	U x 0 x x x
	mem, 1	1 1 0 1 0 0 0 W mod 1 1 1 mem	2-4	8	CY ← (mem) LSB, (mem) ← (mem) + 2, V ← 0 MSB of the operand is unchanged	U x 0 x x x
	reg, CL	1 1 0 1 0 0 1 W 1 1 1 1 1 reg	2	5+n	temp ← CL, while temp ≠ 0 the following operations are repeated: CY ← reg LSB, reg ← reg+2, temp ← temp – 1, MSB of the operand is unchanged	U x U x x x

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
	mem, CL	1 1 0 1 0 0 1 W mod 1 1 1 mem	2-4	7+n	temp ← CL, while temp ≠ 0 the following operations are repeated: CY ← (mem) LSB, (mem) ← (mem)+2, temp ← temp – 1, MSB of the operand is unchanged	U x U x x x
	reg, imm8	1 1 0 0 0 0 0 W 1 1 1 1 1 reg	3	5+n	temp ← imm8, while temp ≠ 0 the following operations are repeated: CY ← reg LSB, reg ← reg+2, temp ← temp – 1, MSB of the operand is unchanged	U x U x x x
	mem, imm8	1 1 0 0 0 0 0 W mod 1 1 1 mem	3-5	7+n	temp ← imm8, while temp ≠ 0 the following operations are repeated: CY ← (mem) LSB, (mem) ← (mem)+2, temp ← temp – 1, MSB of the operand is unchanged	

Rotate Instructions:

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
ROL	reg, 1	1 1 0 1 0 0 0 W 1 1 0 0 0 reg	2	6	CY ←reg MSB, reg ←reg × 2 + CY reg MSB ≠ CY: V←1 reg MSB = CY: V←0	x x
	mem, 1	1 1 0 1 0 0 0 W mod 0 0 0 mem	2-4	8	CY ←(mem) MSB, (mem) ←(mem) × 2 + CY (mem) MSB ≠ CY: V←1 (mem) MSB = CY: V←0	x x
	reg, CL	1 1 0 1 0 0 1 W 1 1 0 0 0 reg	2	5+n	temp ←CL, while temp ≠0 the following operations are repeated: CY ←reg MSB, reg ←reg ×2 + CY temp ← temp - 1	x U
	mem, CL	1 1 0 1 0 0 1 W mod 0 0 0 reg	2-4	7+n	temp ←CL, while temp ≠0 the following operations are repeated: CY ←(mem) MSB, (mem) ←(mem) ×2 + CY temp ← temp - 1	x U

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
	reg, imm8	1 1 0 0 0 0 0 W 1 1 0 0 0 reg	3	5+n	temp ←imm8, while temp ≠0 the following operations are repeated: CY ←reg MSB, reg ←reg ×2 + CY temp ← temp - 1	x U
	mem, imm8	1 1 0 0 0 0 0 W mod 0 0 0 mem	3-5	7+n	temp ←imm8, while temp ≠0 the following operations are repeated: CY ←(mem) MSB, (mem) ←(mem)×2+CY temp ← temp - 1	x U
ROR	reg, 1	1 1 0 1 0 0 0 W 1 1 0 0 1 reg	2	6	CY ←reg LSB, reg ←reg + 2 reg MSB ← CY reg MSB ≠ bit after reg MSB: V←1 reg MSB = bit after reg MSB: V←0	x x
	mem, 1	1 1 0 1 0 0 0 W mod 0 0 1 mem	2-4	8	CY ←(mem) LSB, (mem) ←(mem) + 2 (mem) MSB ← CY (mem) MSB ≠ bit after (mem) MSB: V←1 (mem) MSB = bit after (mem) MSB: V←0	x x

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
	reg, CL	1 1 0 1 0 0 1 W 1 1 0 0 1 reg	2	5+n	temp ←CL, while temp ≠0 the following operations are repeated: CY ←reg LSB, reg ←reg +2 reg MSB ← CY temp ← temp - 1	x U
	mem, CL	1 1 0 1 0 0 1 W mod 0 0 1 reg	2-4	7+n	temp ←CL, while temp ≠0 the following operations are repeated: CY ←(mem) LSB, (mem) ←(mem) +2 (mem) MSB ← CY temp ← temp - 1	x U
	reg, imm8	1 1 0 0 0 0 0 W 1 1 0 0 1 reg	3	5+n	temp ← imm8, while temp ≠0 the following operations are repeated: CY ← reg LSB, reg ←reg + 2 reg MSB ← CY temp ← temp -1	x U

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
	mem, imm8	1 1 0 0 0 0 0 W mod 0 0 1 mem	3-5	7+n	temp ← imm8, while temp ≠0 the following operations are repeated: CY ← (mem) LSB, (mem) ←(mem) + 2 (mem) MSB ← CY temp ← temp -1	x U
ROLC	reg, 1	1 1 0 1 0 0 0 W 1 1 0 1 0 reg	2	6	tmpcy ←CY, CY ←reg MSB reg ←reg ×2 + tmpcy reg MSB ≠ CY : V ←1 reg MSB = CY: V←0	x x
	mem, 1	1 1 0 I 0 0 0 W mod 0 1 0 mem	2-4	8	tmpcy ←CY, CY ← (mem) MSB (mem) ← (mem) ×2 + tmpcy (mem) MSB ≠CY : V ←1 (mem) MSB = CY : V ←0	x x

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
	reg, CL	1 1 0 1 0 0 1 W 1 1 0 1 0 reg	2	5+n	temp ← CL, while temp ≠0 the following operations are repeated: tmpcy ← CY, CY ←reg MSB reg ←reg ×2 + tmpcy temp ←temp - 1	x U
	mem, CL	1 1 0 1 0 0 1 W mod 0 1 0 mem	2-4	7+n	temp ←CL, while temp ≠0 the following operations are repeated: tmpcy ←CY, CY ← (mem) MSB (mem) ← (mem) ×2 + tmpcy temp ←temp - 1	x U
	reg, imm8	1 1 0 0 0 0 0 W 1 1 0 1 0 reg	3	5+n	temp ←imm8, while temp ≠0 the following operations are repeated: tmpcy ←CY, CY ←reg MSB reg ←reg × 2 + tmpcy temp ← temp - 1	x U

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
	mem, imm8	1 1 0 0 0 0 0 W mod 0 1 0 mem	3-5	7+n	temp ← imm8, while temp ≠0 the following operations are repeated: tmpcy ← CY, CY ← (mem) MSB (mem) ← (mem) ×2 + tmpcy temp ← temp - 1	x U
RORC	reg, 1	1 1 0 1 0 0 0 W 1 1 0 1 1 reg	2	6	tmpcy ← CY. CY ← reg LSB reg ← reg + 2 reg MSB ← tmpcy reg MSB ≠ bit after reg MSB :V ← 1 reg MSB = bit after reg MSB: V ← 0	x x
	mem, 1	1 1 0 1 0 0 0 W mod 0 1 1 mem	2-4	8	tmpcy ← CY. CY ← (mem) LSB (mem) ← (mem) + 2 (mem) MSB ← tmpcy (mem) MSB ≠ bit after (mem) MSB : V ← 1 (mem) MSB = bit after reg MSB: V ← 0	x x

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
	reg, CL	1 1 0 1 0 0 1 W 1 1 0 1 1 reg	2	5+n	temp ←CL, while temp ≠ 0 the following operations are repeated: tmpcy ←CY, CY ←reg LSB reg ←reg + 2 reg MSB ←tmpcy temp ← temp-1	x U
	mem, CL	1 1 0 1 0 0 1 W mod 0 1 1 mem	2-4	7+n	temp ←CL, while temp ≠ 0 the following operations are repeated: tmpcy ←CY, CY ←(mem) LSB (mem) ←(mem) + 2 (mem) MSB ←tmpcy temp ← temp-1	x U
	reg, imm8	1 1 0 0 0 0 0 W 1 1 0 1 1 reg	3	5+n	temp ←imm8, while temp ≠ 0 the following operations are repeated: tmpcy ←CY, CY ←reg LSB reg ←reg + 2 reg MSB ←tmpcy temp ← temp-1	x U

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
	mem, imm8	1 1 0 0 0 0 0 W mod 0 1 1 mem	3-5	7+n	temp ←imm8, while temp ≠ 0 the following operations are repeated: tmpcy ←CY, CY ←(mem) LSB (mem) ←(mem) + 2 (mem) MSB ←tmpcy temp ← temp-1	x U

Subroutine Control Instructions:

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
CALL	near-proc	1 1 1 0 1 0 0 0	3	6	SP ← SP -2, (SP + 1,SP) ← PC PC ← PC + disp	
	regptr 16	1 1 1 1 1 1 1 1 1 1 0 1 0 reg	2	3	SP ← SP -2, (SP + 1, SP) ← PC PC ← regptrl6	
	memptrl6	1 1 1 1 1 1 1 1 mod 0 0 mem	2-4	5	TA ← (memptrl6) SP← SP-2, (SP+l,SP)← PC, PC ← TA	
	far-proc	1 0 0 1 1 0 1 0	5	6	SP←SP-2, (SP+l,SP)←PS, PS←seg SP←SP-2, (SP+l,SP)←PC, PC←offset	

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
	memptr32	1 1 1 1 1 1 1 1 mod 0 1 1 mem	2-4	8	TA ←(memptr32), TB ←(memptr32+2) SP ←SP-2, (SP+l,SP) ←PS, PS←TB, SP ←SP-2, (SP + 1, SP) ←PC, PC ←TA	
RET		1 1 0 0 0 0 1 1	1	2	PC ←(SP+1, SP) SP ←SP+2	
	pop- value	1 1 0 0 0 0 1 0	3	7	PC ← (SP+1, SP) SP ← SP + 2, SP ← SP + pop-value	
		1 1 0 0 1 0 1 1	1	2	PC ← (SP+1, SP) PS ← (SP+3, SP+2) PS ←SP+4	
	pop- value	1 1 0 0 1 0 1 0	3	7	PC ←(SP+ 1,SP) PS ←(SP+3,SP+2) SP ←SP + 4, SP ←SP + pop-value	

Stack Manipulation Instructions:

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
PUSH	meml6	1 1 1 1 1 1 1 1 mod 1 1 0 mem	2-4	3	SP ←SP - 2 (SP+ 1, SP) ← (meml6)	
	reg16	0 1 0 1 0 reg	1	3	SP ←SP - 2 (SP + 1, SP) ←reg16	
	sreg	0 0 0sreg 1 1 0	1	3	SP ←SP - 2 (SP + 1, SP) ←sreg	
PUSHF		1 0 0 1 1 1 0 0	1	3	SP ←SP-2 (SP+ 1,SP) ←PSW	
PUSHA		0 1 1 0 0 0 0 0	1	11	Push registers on the stack	
PUSH	imm8	0 1 1 0 1 0 1 0	2	3	(SP - 1, SP - 2) ←imm8 sign extension SP ←SP - 2	
	imml6	0 1 1 0 1 0 0 0	3	3	(SP- 1, SP - 2) ←imml6 SP ←S P - 2	
POP	meml6	1 0 0 0 1 1 1 1 mod 0 0 0 mem	2-4	3	SP ←SP + 2 (meml6) ← (SP - 1, SP - 2)	
	reg16	0 1 0 1 1 reg	1	3	SP ←SP + 2 regl6← (SP-1, SP-2)	
	sreg	0 0 0sreg 1 1 1	1	3	SP ← SP + 2 sreg: SS, DS0, DS1 sreg ← (SP-1, SP-2)	
POPF		1 0 0 1 1 1 0 1	1	3	SP ← SP + 2 PSW← (SP-1, SP-2)	R R R R R R

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
POPA		0 1 1 0 0 0 0 1	1	8	Pop registers from the stack	
ENTER	imml6, imm8	1 1 0 0 1 0 0 0	4	10+imm8	Prepare New Stack Frame	
LEAVE		1 1 0 0 1 0 0 1	1	4	Dispose of Stack Frame	

Branch Instructions:

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
JMP	near-label	1 1 1 0 1 0 0 1	3	5	PC ← PC + disp	
	short-label	1 1 1 0 1 0 1 1	2	5	PC ← PC + ext-disp8	
	regptr l6	1 1 1 1 1 1 1 1 1 1 1 0 0 reg	2	5	PC ← regptrl6	
	memptrl6	1 1 1 1 1 1 1 1 mod 1 0 0 mem	2-4	5	PC ← (memptrl6)	
	far-label	1 1 1 0 1 0 1 0	5	5	PS ← seg PC ← offset	
	memptr32	1 1 1 1 1 1 1 1 mod 1 0 1 mem	2-4	10	PS ← (memptr32+2) PC ← (memptr32)	

Conditional Branch Instructions:

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
JO	short-label	0 1 1 1 0 0 0 0	2	5/2	If V = 1: PC ← PC + ext-disp8	
JNO	short-label	0 0 0 1	2	5/2	If V = 0: PC ← PC + ext-disp8	
JC, JB,JNAE	short-label	0 0 1 0	2	5/2	If CY = 1: PC ← PC + ext-disp8	
JNC, JNB,JAE	short-label	0 0 1 1	2	5/2	If CY = 0 PC ← PC + ext-disp8	
JE, JZ	short-label	0 1 0 0	2	5/2	If Z = 1 PC ← PC + ext-disp8	
JNE, JNZ	short-label	0 1 0 1	2	5/2	If Z = 0; PC ← PC + ext-disp8	
JBE/JNA	short-label	0 1 1 0	2	5/2	If CY OR Z = 1: PC ← PC + ext-disp8	
JNBE/JA	short-label	0 1 1 1	2	5/2	If CY OR Z = 0: PC ← PC + ext-disp8	
JS	short-label	1 0 0 0	2	5/2	If S = 1: PC ← PC + ext-disp8	
JNS	short-label	1 0 0 1	2	5/2	If S = 0 PC ← PC + ext-disp8	
JP/JPE	short-label	1 0 1 0	2	5/2	If P = 1: PC ← PC + ext-disp8	
JNP/JNO	short-label	1 0 1 1	2	5/2	If P = 0 PC ← PC + ext-	
JL/JNGE	short-label	1 1 0 0	2	5/2	If S xor V = 1 PC ← PC + ext-disp8	

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
JNL/JGE	short-label	1 1 0 1	2	5/2	If S xor V = 0 PC ← PC + ext-disp8	
JLE/JNG	short-label	1 1 1 0	2	5/2	If (S xor V) OR Z = 1 PC ← PC + ext-disp8	
JNLE/JG	short-label	1 1 1 1	2	5/2	If (S xor V) OR Z = 0 PC ← PC + ext-disp8	
LOOPNZ / LOOPNE	short-label	1 1 1 0 0 0 0 0	2	5/2	CW= CW-1 PC ← PC+ext - disp8 If Z=0 and CW≠0	
LOOPZ / LOOPE	short-label	0 0 0 1	2	5/2	CW = CW-1, PC ← PC+ ext-disp8, if Z= 1 and CW ≠0	
LOOP	short-label	0 0 1 0	2	5/2	CW = CW-1, PC ← PC +ext-disp8, If CW ≠ 0	
JCXZ	short-label	0 0 1 1	2	5/2	If CW = 0 PC ← PC +ext-disp8	

Interrupt Instructions:

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
INT	3	1 1 0 0 1 1 0 0	1	46	TA ←(00D, 00C), TC ← (00F, 00E) SP ← SP - 2 , (SP+l, SP) ← PSW IE ← 0 , BRK ← 0 SP ← SP - 2, (SP+1, SP) ←PS, PS ←TC SP ← SP - 2, (SP + 1, SP) ← PC, PC ← TA	
	imm8 (≠ 3)	1 1 0 0 1 1 0 1	2	46	TA ← (4n+1,4n), TC ← (4n+3,4n+2) n = imm8 SP ← SP- 2, (SP+l, SP) ← PSW, IE ← 0, BRK ← 0 SP ← SP - 2, (SP+l, SP) ←PS, PS ←TC SP ← SP - 2, (SP+1, SP) ← PC, PC ← TA	

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
INTO		1 1 0 0 1 1 1 0	1	43/2	If V = 1: TA ← (011, 010), TC ← (013, 012) SP ← SP - 2, (SP+l, SP) ← PSW, IE ← 0, BRK ← 0 SP ← SP - 2, (SP+l, SP) ← PS, PS ←TC SP ← SP - 2, (SP+1, SP) ← PC, PC ← TA	
IRET		1 1 0 0 1 1 1 1	1	7	PC ← (SP+1, SP), PS ← (SP+3, SP+2), PSW← (SP+5, SP+4), SP ← SP+6	
BOUND	regl6, mem32	0 1 1 0 0 0 1 0 mod reg mem	2-4	6	If (mem32) > reg16 or (mem32+2) < reg16 TA ← (015. 014), TC ← (017, 016) SP ← SP - 2, (SP+l, SP) ←PSW IE ← 0, BRK ← 0 SP ← SP - 2, (SP+l, SP) ← PS, PS ← TC SP ← SP - 2, (SP+1, SP) ← PC, PC ← TA	

CPU Control Instructions:

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
HLT		1 1 1 1 0 1 0 0	1	2	CPU Halt	
CLI		1 1 1 1 1 0 1 0	1	3	IE ← 0	
STI		1 1 1 1 1 0 1 1	1	2	IE ← 1	
LOCK		1 1 1 1 0 0 0 0	1	1	Bus Lock Prefix	
NOP		1 0 0 1 0 0 0 0	1	4	No Operation	

Segment Operation Instructions:

Mnemonic	Operand(s)	Operation Code 7 6 5 4 3 2 1 0	Byte	Clock Cycles	Operation	Flags A C V P S Z
DS0: DS1: PS: SS:		0 0 1 1 1 1 1 0 0 0 1 0 0 1 1 0 0 0 1 0 1 1 1 0 0 0 1 1 0 1 1 0	1	2 2 2 2	Segment override prefix.	