SONY **CXQ70116**

16-Bit Microprocessor

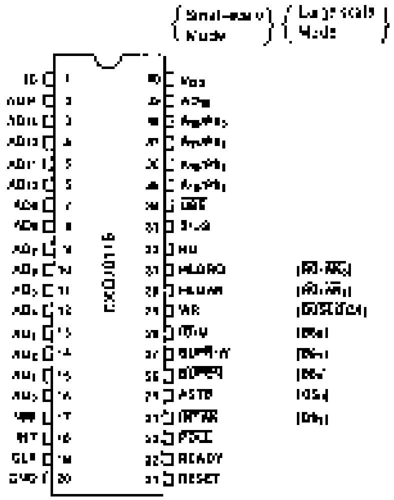
Description

The CXQ70116 is a CMOS 16-bit microprocessor with internal 16-bit architecture and a 16-bit external data bus. The CXQ70116 instruction set is a superset of the 8086/8088; however, mnemonics and execution times are different. The CXQ70116 additionally has a powerful instruction set including bit processing, packed BCD operations, and high-speed multiplication/division operations. The CXQ70116 can also emulate the functions of an 8080 and comes with a standby mode that significantly reduces power consumption. It is software-compatible with the CXQ70108 microprocessor.

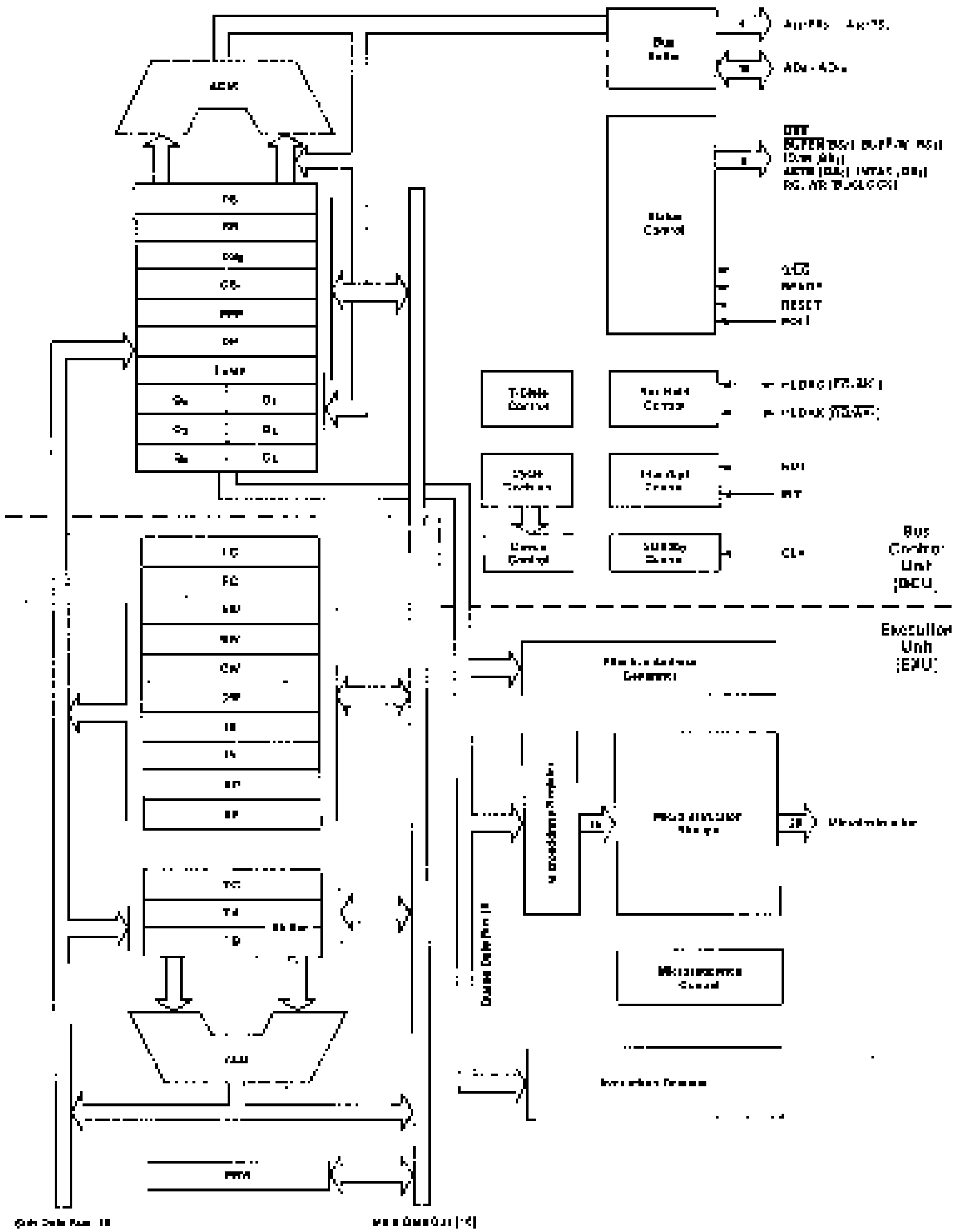
Features

- Minimum instruction execution time: 250 ns (at 8 MHz)
- Maximum addressable memory: 1 Mbytes
- Abundant memory addressing modes
- 14 × 16-bit register set
- 101 instructions
- Instruction set is a superset of 8086/8088 instruction set
- Bit, byte, word, and block operations
- Bit field operation instructions
- Packed BCD operation instructions
- Multiplication/division instructions execution time: 2.4 µs to 7.1 µs (at 8 MHz)
- High-speed block transfer instructions: 2 Mbytes/s (at 8 MHz)
- High-speed calculation of effective addresses: 2 clock cycles in any addressing mode
- Maskable (INT) and nonmaskable (NMI) interrupt inputs
- IEEE-796 bus compatible interface
- 8080 emulation functions
- CMOS technology
- Low power consumption
- Standby function
- Single power supply
- 5-MHz or 8-MHz clock
- 40-pin Plastic/Ceramic DIP (600 mil)
- NEC µPD70116 (V30) compatible

Pin Configuration (Top View)



Block Diagram



Pin Identification

No.	Symbol	Direction	Function
1	IC*		Internally connected
2—16	AD_{14}—AD_0	In/Out	Address/data bus
17	NMI	In	Nonmaskable interrupt input
18	INT	In	Maskable interrupt input
19	CLK	In	Clock input
20	GND		Ground
21	RESET	In	Reset input
22	READY	In	Ready input
23	$\overline{POLL}$	In	Poll input
24	$\overline{INTAK}$ (QS_1)	Out	Interrupt acknowledge output (queue status bit 1 output)
25	ASTB (QS_0)	Out	Address strobe output (queue status bit 0 output)
26	$\overline{BUFEN}$ ($\overline{BS_0}$)	Out	Buffer enable output (bus status bit 0 output)
27	BUFR/$\overline{W}$ ($\overline{BS_1}$)	Out	Buffer read/write output (bus status bit 1 output)
28	IO/$\overline{M}$ ($\overline{BS_2}$)	Out	Access is I/O or memory (bus status bit 2 output)
29	$\overline{WR}$ ($\overline{BUSLOCK}$)	Out	Write strobe output (bus lock output)
30	HLDAK ($\overline{RQ}/\overline{AK_1}$)	Out (In/Out)	Hold acknowledge output (bus hold request input/acknowledge output 1)
31	HLDRQ ($\overline{RQ}/\overline{AK_0}$)	In (In/Out)	Hold request input (bus hold request input/acknowledge output 0)
32	$\overline{RD}$	Out	Read strobe output
33	S/$\overline{LG}$	In	Small-scale/large-scale system input
34	$\overline{UBE}$	Out	Upper byte enable
35—38	A_{19}/PS_3—A_{16}/PS_0	Out	Address bus, high bits or processor status output
39	AD_{15}	In/Out	Address/data bus, bit 15
40	V_{DD}		Power supply

Notes: *IC should be connected to ground.
Where pins have different functions in small- and large-scale systems, the large-scale system pin symbol and function are in parentheses.
Unused input pins should be tied to ground or V_{DD} to minimize power dissipation and prevent the flow of potentially harmful currents.

Pin Functions

Some pins of the CXQ70116 have different functions according to whether the microprocessor is used in a small- or large-scale system. Other pins function the same way in either type of system.

AD_{15} — AD_0 [Address/Data Bus]

For small- and large-scale systems.

AD_{15} — AD_0 are the time-multiplexed address and data bus. They are active high. This bus contains the lower 16 bits of the 20-bit address during T1 of the bus cycle. It is used as a 16-bit data bus during T2, T3, and T4 of the bus cycle.

The address/data bus is a three-state bus and can be high or low during standby mode. The bus will float to the high impedance during hold and interrupt acknowledge.

NMI [Nonmaskable Interrupt]

For small- and large-scale systems.

This pin is used to input nonmaskable interrupt requests. NMI cannot be masked by software. This input is positive edge-triggered and can be sensed during any clock cycle. Actual interrupt processing begins, however, after completion of the instruction in progress.

The contents of interrupt vector 2 determine the starting address for the interrupt-servicing routine. Note that a hold request will be accepted even during NMI acknowledge.

This interrupt will cause the CXQ70116 to exit the standby mode.

INT [Maskable Interrupt]

For small- and large-scale systems.

This pin is a level-triggered interrupt request that can be masked by software.

INT is active high and is sensed during the last clock of the instruction. The interrupt will be accepted if the system is in interrupt enable state (if the interrupt enable flag IE is set). The CPU outputs the $\overline{INTAK}$ signal to inform external devices that the interrupt request has been granted.

If NMI and INT interrupts occur at the same time, NMI has higher priority than INT and INT cannot be accepted. A hold request will be accepted during INT acknowledge.

This interrupt causes the CXQ70116 to exit the standby mode.

CLK [Clock]

For small- and large-scale systems.

This pin is used for external clock input.

RESET [Reset]

For small- and large-scale systems.

This pin is used for the CPU reset signal. It is active high. Input of this signal has priority over all other operations. After the reset signal input returns low, the CPU begins execution of the program starting at address FFFF0H.

In addition to causing normal CPU start, RESET input will cause the CXQ70116 to exit the standby mode.

READY [Ready]

For small- and large-scale systems.

When the memory or I/O device being accessed cannot complete data read or write within the CPU basic access time, it can generate a CPU wait state (Tw) by setting this signal to inactive (low) and requesting a read/write cycle delay.

If the READY signal is active (high) during either T3 or Tw state, the CPU will not generate a wait state.

$\overline{\text{POLL}}$ [Poll]

For small- and large-scale systems.

The CPU checks this input upon execution of the POLL instruction. If the input is low, then execution continues. If the input is high, the CPU will check the $\overline{\text{POLL}}$ input every five clock cycles until the input becomes low again.

The $\overline{\text{POLL}}$ and READY functions are used to synchronize CPU program execution with the operation of external devices.

$\overline{\text{RD}}$ [Read Strobe]

For small- and large-scale systems.

The CPU outputs this strobe signal during data read from an I/O device or memory. The IO/$\overline{\text{M}}$ signal is used to select between I/O and memory. $\overline{\text{RD}}$ will be high during standby mode. It is three-state and floats to the high impedance during hold acknowledge.

S/$\overline{\text{LG}}$ [Small/Large]

For small- and large-scale systems.

This signal determines the operation mode of the CPU. This signal is fixed either high or low. When this signal is high, the CPU will operate in small-scale system mode, and when low, in the large-scale system mode. A small-scale system will have at most one bus master such as a DMA controller device on the bus. A large-scale system can have more than one bus master accessing the bus as well as the CPU.

Pins 24 to 31 function differently depending on the operating mode of the CPU. Separate nomenclature is adapted for these signals in the two operational modes.

Pin No.	Function	
	S/$\overline{\text{LG}}$-high	S/$\overline{\text{LG}}$-low
24	$\overline{\text{INTAK}}$	QS_1
25	ASTB	QS_0
26	$\overline{\text{BUFEN}}$	$\overline{BS_2}$
27	BUFR/$\overline{\text{W}}$	$\overline{BS_1}$
28	IO/$\overline{\text{M}}$	$\overline{BS_0}$
29	$\overline{\text{WR}}$	$\overline{\text{BUSLOCK}}$
30	HLDAK	$\overline{\text{RQ}}/\overline{AK_1}$
31	HLDRQ	$\overline{\text{RQ}}/\overline{AK_0}$

$\overline{\text{INTAK}}$ [Interrupt Acknowledge]

For small-scale systems.

The CPU generates the $\overline{\text{INTAK}}$ signal low when it accepts an INT signal.

The interrupting device synchronizes with this signal and outputs the interrupt vector to the CPU via the data bus (AD_7 — AD_0). $\overline{\text{INTAK}}$ will be high during standby mode.

ASTB [Address Strobe]

For small-scale systems.

The CPU outputs this strobe signal to latch address information at an external latch. ASTB will be low during standby mode.

$\overline{BUFEN}$ [Buffer Enable]

For small-scale systems.

It is used as the output enable signal for an external bidirectional buffer. The CPU generates this signal during data transfer operations with external memory or I/O devices or during input of an interrupt vector.

$\overline{BUFEN}$ will be high during standby mode. It is three-state and floats to the high impedance during hold acknowledge.

BUF$\overline{R}$/W [Buffer Read/Write]

For small-scale systems.

The output of this signal determines the direction of data transfer with an external bidirectional buffer. A high output causes transmission from the CPU to the external device; a low signal causes data transfer from the external device to the CPU.

BUF$\overline{R}$/W will be either high or low during standby mode. It is three-state and floats to the high impedance during hold acknowledge.

$\overline{IO}$/M [IO/Memory]

For small-scale systems.

The CPU generates this signal to specify either I/O access or memory access. A low-level output specifies I/O and a high-level specifies memory.

$\overline{IO}$/M will be either high or low during standby mode. It is three-state and floats to the high impedance during hold acknowledge.

$\overline{WR}$ [Write Strobe]

For small-scale systems.

The CPU generates this strobe signal during data write to an I/O device or memory. Selection of either I/O or memory is performed by the $\overline{IO}$/M signal.

$\overline{WR}$ will be high during standby mode. It is three-state and floats to the high impedance during hold acknowledge.

HLDAK [Hold Acknowledge]

For small-scale systems.

The HLDAK signal is used to indicate that the CPU accepts the hold request signal (HLDRQ). When this signal is high, the address bus, address/data bus, and the control lines become high impedance.

HLDRQ [Hold Request]

For small-scale systems.

This input signal is used by external devices to request the CPU to release the address bus, address/data bus, and the control bus.

$\overline{UBE}$ [Upper Byte Enable]

For small- and large-scale systems.

$\overline{UBE}$ indicates the use of the upper eight bits (AD_{15} — AD_8) of the address/data bus during a bus cycle. This signal is active low during T_1 for read, write, and interrupt acknowledge cycles when AD_{15} — AD_8 are to be used. Bus cycles in which $\overline{UBE}$ is active are shown in the following table.

Type of Bus Operation	UBE	A0	Number of Bus Cycle
Word at even address	0	0	1
Word at odd address	0 1	1* 0**	2
Byte at even address	1	0	1
Byte at odd address	0	1	1

Notes: *First bus cycle
**Second bus cycle

$\overline{UBE}$ is low continuously during the interrupt acknowledge state. It will be high during standby mode. It is three-state and floats to the high impedance during hold acknowledge.

A_{19}/PS_3 — A_{16}/PS_0 (Address Bus/Processor Status)

For small- and large-scale systems.

These pins are time-multiplexed to operate as an address bus and as processor status signals.

When used as the address bus, these pins are the high 4 bits of the 20-bit memory address. During I/O access, all 4 bits output data 0.

The processor status signals are provided for both memory and I/O use. PS_3 is always 0 in the native mode and 1 in 8080 emulation mode. The interrupt enable flag (IE) is output on pin PS_2. Pins PS_1 and PS_0 indicate which memory segment is being accessed.

A_{17}/PS_1	A_{16}/PS_0	Segment
0	0	Data segment 1
0	1	Stack segment
1	0	Program segment
1	1	Data segment 0

A_{19}/PS_3 — A_{16}/PS_0 will be either high or low during standby mode. They are three-state and float to the high impedance during hold acknowledge.

QS_1, QS_0 (Queue Status)

For large-scale systems.

The CPU uses these signals to allow external devices, such as the floating-point arithmetic processor chip, to monitor the status of the internal CPU instruction queue.

QS_1	QS_0	Instruction Queue Status
0	0	NOP (queue does not change)
0	1	First byte of instruction
1	0	Flush queue
1	1	Subsequent bytes of instruction

The instruction queue status indicated by these signals is the status when the execution unit (EXU) accesses the instruction queue. The data output from these pins is therefore valid only for one clock cycle immediately following queue access. These status signals are provided so that the floating-point processor chip can monitor the CPU's program execution status and synchronize its operation with the CPU when control is passed to it by the FPO (Floating Point Operation) instructions.

QS_1, QS_0 will be low during standby mode.

$\overline{BS_2}$ — $\overline{BS_0}$ [Bus Status]

For large-scale systems.

The CPU uses these status signals to allow an external bus controller to monitor what the current bus cycle is.

The external bus controller decodes these signals and generates the control signals required to perform access of the memory or I/O device.

BS_2	BS_1	BS_0	Bus Cycle
0	0	0	Interrupt acknowledge
0	0	1	I/O read
0	1	0	I/O write
0	1	1	Halt
1	0	0	Program fetch
1	0	1	Memory read
1	1	0	Memory write
1	1	1	Passive state

$\overline{BS_2}$ — $\overline{BS_0}$ will be High during standby mode. They are three-state and float to the high impedance during hold acknowledge.

$\overline{BUSLOCK}$ [Bus Lock]

For large-scale systems.

The CPU uses this signal to secure the bus while executing the instruction immediately following the BUSLOCK prefix instruction. It is a status signal to the other bus masters in a multiprocessor system inhibiting them from using the system bus during this time.

The output of this signal is three-state and becomes high impedance during hold acknowledge. $\overline{BUSLOCK}$ is high during standby mode except if the HALT instruction has a BUSLOCK prefix.

$\overline{RQ}/\overline{AK_1}$, $\overline{RQ}/\overline{AK_0}$ [Hold Request/Acknowledge]

For large-scale systems.

These pins function as bus hold request inputs ($\overline{RQ}$) and as bus hold acknowledge outputs ($\overline{AK}$). $\overline{RQ}/\overline{AK_0}$ has a higher priority than $\overline{RQ}/\overline{AK_1}$.

These pins have three-state outputs with on-chip pull-up resistors which keep the pin at high level when the output is high impedance.

V_{DD} [Power Supply]

For small- and large-scale systems.

This pin is used for the +5V power supply.

GND [Ground]

For small- and large-scale systems.

This pin is used for ground.

IC [Internally Connected]

This pin is used for tests performed at the factory by SONY. The CXQ70116 is used with this pin at ground potential.

Absolute Maximum Ratings

(Ta=+25°C)

Parameter	Symbol	Rating Value	Unit
Power supply voltage	V_{CC}	−0.5 to +7.0	V
Input voltage	V_I	−0.5 to V_{CC} +0.3	V
CLK input voltage	V_K	−0.5 to V_{CC} +1.0	V
Output voltage	V_O	−0.5 to V_{CC} +0.3	V
Power dissipation	P_{DMAX}	+0.5	W
Operating temperature	T_{OPR}	−40 to +85	°C
Storage temperature	T_{STG}	−65 to +150	°C

Comment Exposing the device to stresses above those listed in Absolute Maximum Ratings could cause permanent damage. The device is not meant to be operated under conditions outside the limits described in the operational sections of this specification.
Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

DC Characteristics

CXQ70116-5, Ta=−40°C to +85°C, V_{DD}=+5V±10%
CXQ70116-8, Ta=−10°C to +70°C, V_{DD}=+5V±5%

Parameter	Symbol		Limits Min.	Typ.	Max.	Unit	Test Conditions
Input voltage high	V_{IH}		2.2		V_{DD}+0.3	V	
Input voltage low	V_{IL}		−0.5		0.8	V	
CLK input voltage high	V_{KH}		3.9		V_{DD}+1.0	V	
CLK input voltage low	V_{KL}		−0.5		0.6	V	
Output voltage high	V_{OH}		0.7× V_{DD}			V	I_{OH}=−400 μA
Output voltage low	V_{OL}				0.4	V	I_{OL}=2.5 mA
Input leakage current high	I_{LIH}				10	μA	V_I=V_{DD}
Input leakage current low	I_{LIL}				−10	μA	V_I=0V
Output leakage current high	I_{LOH}				10	μA	V_O=V_{DD}
Output leakage current low	I_{LOL}				−10	μA	V_O=0V
Supply current	I_{DD}	70116-5 5 MHz		30	60	mA	Normal operation
				5	10	mA	Standby mode
		70116-8 8 MHz		45	80	mA	Normal Operation
				6	12	mA	Standby mode

Capacitance

(Ta=+25°C, V_{DD}=0V)

Parameter	Symbol	Limits Min.	Max.	Unit	Test Conditions
Input capacitance	C_I		15	pF	fc=1 MHz Unmeasured pins returned to 0V
I/O capacitance	C_{IO}		15	pF	

AC Characteristics

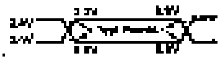
CXQ70116-5, Ta=−40°C to +85°C, Vcc=+5V±10%
CXQ70116-8, Ta=−10°C to +70°C, Vcc=+5V±5%

Parameter	Symbol	CXQ70116-5 Min.	CXQ70116-5 Max.	CXQ70116-8 Min.	CXQ70116-8 Max.	Unit	Test Conditions
Small/Large Scale							
Clock cycle	t_{CYK}	200	500	125	500	ns	
Clock pulse width high	t_{KKH}	69		50		ns	V_{KH}=3.0V
Clock pulse width low	t_{KKL}	90		60		ns	V_{KL}=1.5V
Clock rise time	t_{KR}		10		8	ns	1.5V to 3.0V
Clock fall time	t_{KF}		10		7	ns	3.0V to 1.5V
READY inactive setup to CLK ↓	t_{SRYLK}	8		8		ns	
READY inactive hold after CLK ↑	t_{HKRYH}	30		20		ns	
READY active setup to CLK ↑	t_{SRYHK}	t_{KKL}−8		t_{KKL}−8		ns	
READY active hold after CLK ↑	t_{HKRYL}	30		20		ns	
Data setup time to CLK ↓	t_{SDK}	30		20		ns	
Data hold time after CLK ↓	t_{HKD}	10		10		ns	
NMI, INT, POLL setup time to CLK ↑	t_{SIK}	30		15		ns	
RESET setup time to CLK ↑	t_{SRST}	30		20		ns	
RESET hold time to CLK ↑	t_{HRST}	10		10		ns	
Input rise time (except CLK)	t_{IR}		20		20	ns	0.8V to 2.2V
Input fall time (except CLK)	t_{IF}		12		12	ns	2.2V to 0.8V
Output rise time	t_{OR}		20		20	ns	0.8V to 2.2V
Output fall time	t_{OF}		12		12	ns	2.2V to 0.8V
Small Scale							
Address delay time from CLK	t_{DKA}	10	90	10	60	ns	C_L=100 pF
Address hold time from CLK	t_{HKA}	10		10		ns	
PS delay time from CLK ↓	t_{DKP}	10	90	10	60	ns	
PS float delay time from CLK ↑	t_{FKP}	10	80	10	60	ns	
Address setup time to ASTB ↓	t_{SAST}	t_{KKL}−60		t_{KKL}−30		ns	
Address float delay time from CLK ↓	t_{FKA}	t_{HKA}	80	t_{HKA}	60	ns	
ASTB ↑ delay time from CLK ↓	t_{DKSTH}		80		50	ns	
ASTB ↓ delay time from CLK ↑	t_{DKSTL}		85		55	ns	
ASTB width high	t_{STST}	t_{KKL}−20		t_{KKL}−10		ns	
Address hold time from ASTB ↓	t_{HSTA}	t_{KKL}−10		t_{KKL}−10		ns	

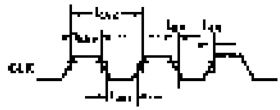
Parameter	Symbol	CXQ70116-5 Min.	CXQ70116-5 Max.	CXQ70116-8 Min.	CXQ70116-8 Max.	Unit	Test Conditions
Control delay time from CLK	t_{DKCT}	10	110	10	65	ns	$C_L=100$ pF
Address float to RD ↓	t_{AFRL}	0		0		ns	
RD ↓ delay time from CLK ↓	t_{DKRL}	10	165	10	80	ns	
RD ↑ delay time from CLK ↓	t_{DKRH}	10	150	10	80	ns	
Address delay time from RD ↑	t_{DRHA}	$t_{CYK}-45$		$t_{CYK}-40$		ns	
RD width low	t_{RR}	$2t_{CYK}-75$		$2t_{CYK}-50$		ns	
Data output delay time from CLK ↓	t_{DKD}	10	90	10	60	ns	
Data float delay time from CLK ↓	t_{FKD}	10	60	10	60	ns	
WR width low	t_{WW}	$2t_{CYK}-60$		$2t_{CYK}-40$		ns	
HLDRQ setup time to CLK ↑	t_{SHQK}	35		20		ns	
HLDAK delay time from CLK ↑	t_{DKHA}	10	100	10	100	ns	
Large Scale							
Address delay time from CLK	t_{DKA}	10	90	10	60	ns	$C_L=100$ pF
Address hold time from CLK	t_{HKA}	10		10		ns	
PS delay time from CLK ↓	t_{DKP}	10	90	10	60	ns	
PS float delay time from CLK ↑	t_{FKP}	10	80	10	60	ns	
Address float delay time from CLK ↑	t_{FKA}	t_{HKA}	60	t_{HKA}	60	ns	
Address delay time from RD ↑	t_{DRHA}	$t_{CYK}-45$		$t_{CYK}-40$		ns	
ASTB ↓ delay time from BS ↓	t_{DBST}		15		15	ns	
BS ↓ delay time from CLK ↑	t_{DKBL}	10	110	10	60	ns	
BS ↑ delay time from CLK ↓	t_{DKBH}	10	130	10	65	ns	
RD ↓ delay time from address float	t_{DAFRL}	0		0		ns	
RD ↓ delay time from CLK ↓	t_{DKRL}	10	165	10	80	ns	
RD ↑ delay time from CLK ↓	t_{DKRH}	10	150	10	80	ns	
RD width low	t_{RR}	$2t_{CYK}-75$		$2t_{CYK}-50$		ns	
Data output delay time from CLK ↓	t_{DKD}	10	90	10	60	ns	
Data float delay time from CLK ↓	t_{FKD}	10	80	10	60	ns	
AK delay time from CLK ↓	t_{DKAK}		70		50	ns	
RB setup time to CLK ↑	t_{SRBK}	20		10		ns	
RB hold time after CLK ↑	t_{HKRB}	40		30		ns	

Timing Waveforms

AC Test Input Waveform [Except CLK]

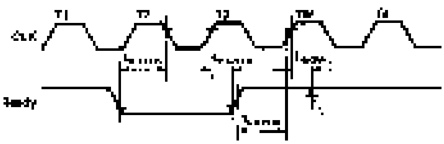


Clock Timing

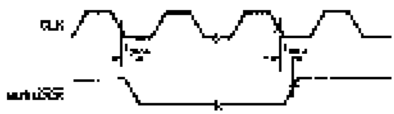


AC Output Test Points

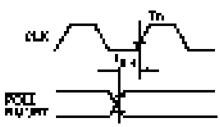
Wait [Ready] Timing



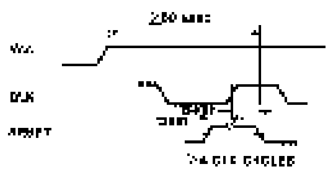
$\overline{\text{BUSLOCK}}$ Output Timing



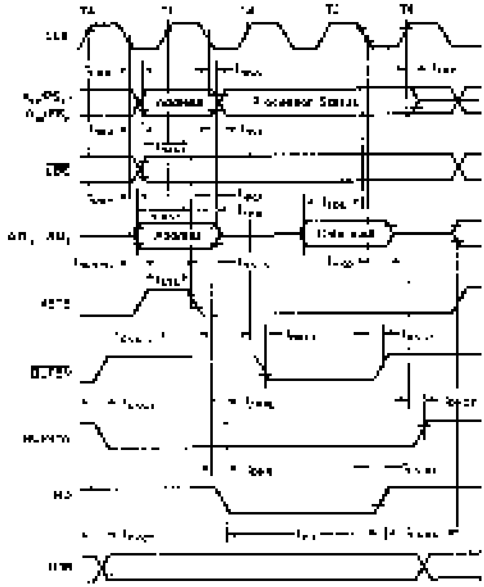
$\overline{\text{POLL}}$ NMI, INT input Timing



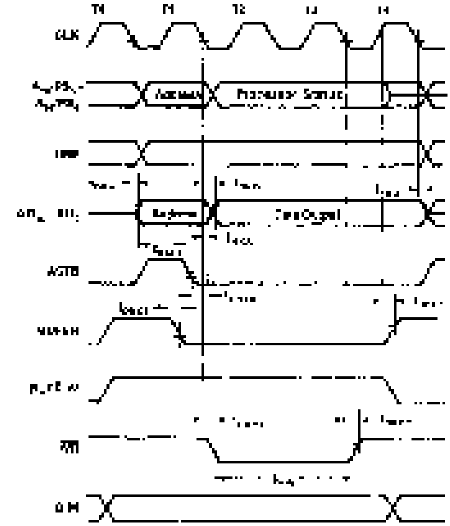
RESET Timing



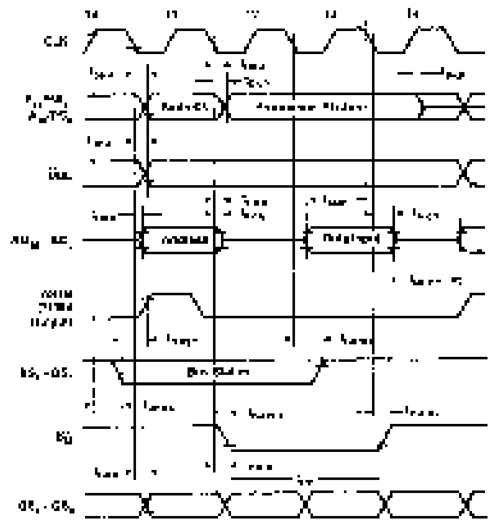
Read Timing [Small Scale]



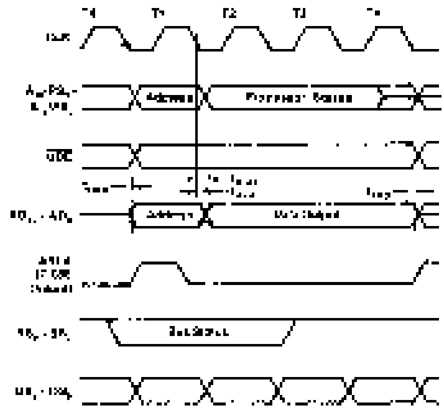
Write Timing [Small Scale]



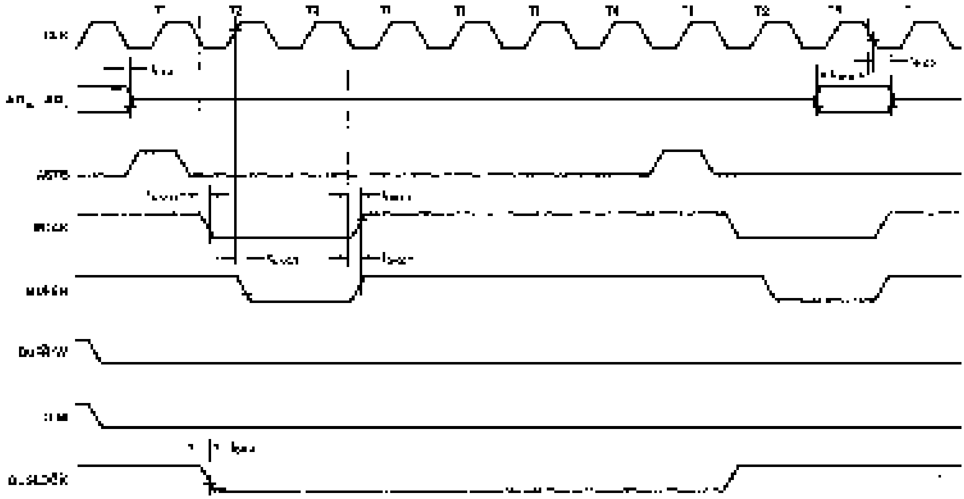
Read Timing [Large Scale]



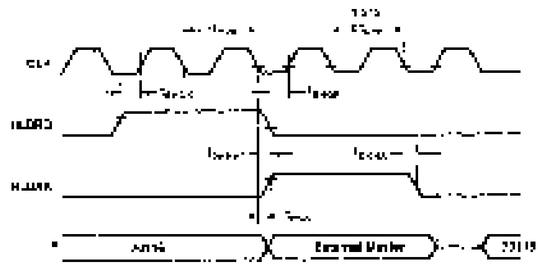
Write Timing [Large Scale]



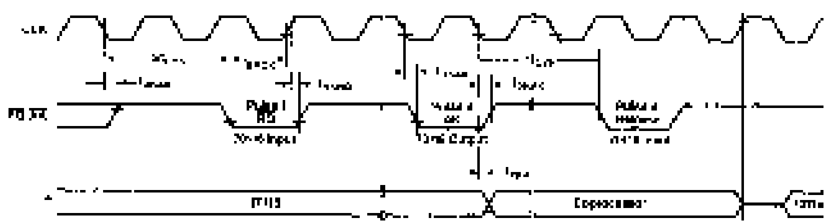
Interrupt Acknowledge Timing



Hold Request/Acknowledge Timing (Small Scale)



Bus Request/Acknowledge Timing (Large Scale)



Register Configuration

Program Counter [PC]

The program counter is a 16-bit binary counter that contains the segment offset address of the next instruction which the EXU is to execute.

The PC is incremented each time the microprogram fetches an instruction from the instruction queue. A new location value is loaded into the PC each time a branch, call, return, or break instruction is executed. At this time, the contents of the PC are the same as the Prefetch Pointer (PFP).

Prefetch Pointer [PFP]

The prefetch pointer (PFP) is a 16-bit binary counter which contains a segment offset which is used to calculate a program memory address that the bus control unit (BCU) uses to prefetch the next word for the instruction queue. The contents of PFP are an offset from the PS (Program Segment) register.

The PFP is incremented each time the BCU prefetches an instruction from the program memory. A new location will be loaded into the PFP whenever a branch, call, return, or break instruction is executed. At that time the contents of the PFP will be the same as those of the PC (Program Counter).

Segment Registers [PS, SS, DS0, and DS1]

The memory addresses accessed by the CXQ70116 are divided into 64K-byte logical segments. The starting (base) address of each segment is specified by a segment register, and the offset from this starting address is specified by the contents of another register or by the effective address.

These are the four types of segment registers used.

Segment Register	Default Offset
PS (Program Segment)	PFP
SS (Stack Segment)	SP, effective address
DS0 (Data Segment 0)	IX, effective address
DS1 (Data Segment 1)	IY

General Purpose Registers [AW, BW, CW, and DW]

There are four 16-bit general-purpose registers. Each one can be used as one 16-bit register or as two 8-bit registers by dividing them into their high and low bytes (AH, AL, BH, BL, CH, CL, DH, DL).

Each register is also used as a default register for processing specific instructions. The default assignments are:

AW: Word multiplication/division, word I/O, BCD rotation, data conversion, translation
AL: Byte multiplication/division, byte I/O, BCD rotation, data conversion, translation
AH: Byte multiplication/division
BW: Translation
CW: Loop control branch, repeat prefix
CL: Shift instructions, rotation instructions, BCD operations
DW: Word multiplication/division, indirect addressing I/O

Pointers [SP, BP] and Index Registers [IX, IY]

These registers serve as base pointers or index registers when accessing the memory using based addressing, indexed addressing, or based indexed addressing.

These registers can also be used for data transfer and arithmetic and logical operations in the same manner as the general-purpose registers. They cannot be used as 8-bit registers.

Also, each of these registers acts as a default register for specific operations. The default assignments are:

SP: Stack operations
IX: Block transfer (source), BCD string operations
IY: Block transfer (destination), BCD string operations

Program Status Word (PSW)

The program status word consists of the following six status and four control flags.

Status Flags
- V (Overflow)
- S (Sign)
- Z (Zero)
- AC (Auxiliary Carry)
- P (Parity)
- CY (Carry)

Control Flags
- MD (Mode)
- DIR (Direction)
- IE (Interrupt Enable)
- BRK (Break)

When the PSW is pushed on the stack, the word images of the various flags are as shown here.

PSW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M	1	1	1	V	D	I	B	S	Z	0	A	0	P	1	C
D					I	E	R				C				Y
					R		K								

The status flags are set and reset depending upon the result of each type of instruction executed. Instructions are provided to set, reset, and complement the CY flag directly.
Other instructions set and reset the control flags and control the operation of the CPU.

High-Speed Execution of Instructions

This section highlights the major architectural features that enhance the performance of the CXQ70116.
- Dual data bus in EXU
- Effective address generator
- 16/32-bit temporary registers/shifters (TA, TB)
- 16-bit loop counter
- PC and PFP

Dual Data Bus Method

To reduce the number of processing steps for instruction execution, the dual data bus method has been adopted for the CXQ70116 (figure 1). The two data buses (the main data bus and the subdata bus) are both 16 bits wide. For addition/subtraction and logical and comparison operations, processing time has been speeded up some 30% over single-bus systems.

Fig. 1. Dual Data Buses

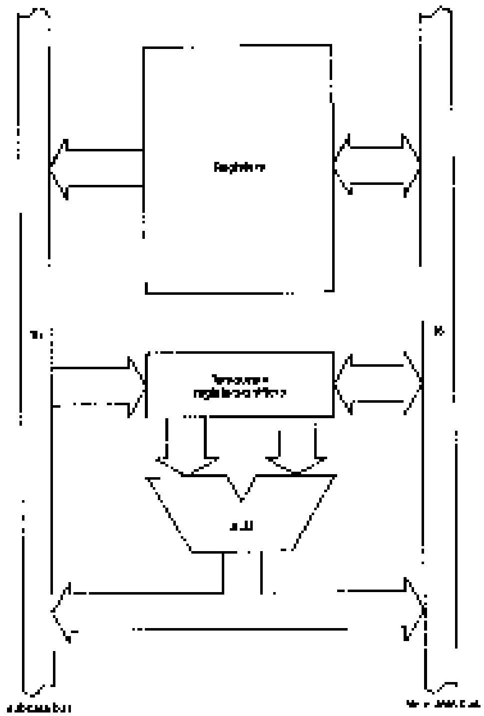
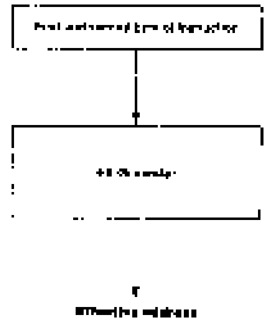


Fig. 2. Effective Address Generator



Example

ADD AW, BW ;AW ← AW + BW

	Single Bus	Dual Bus
Step 1	TA ← AW	TA ← AW, TB ← BW
Step 2	TB ← BW	AW ← TA + TB
Step 3	AW ← TA + TB	

Effective Address Generator

This circuit (figure 2) performs high-speed processing to calculate effective addresses for accessing memory.

Calculating an effective address by the microprogramming method normally requires 5 to 12 clock cycles. This circuit requires only two clock cycles for addresses to be generated for any addressing mode. Thus, processing is several times faster.

16/32-Bit Temporary Registers/Shifters (TA, TB)

These 16-bit temporary registers/shifters (TA, TB) are provided for multiplication/division and shift/rotation instructions.

These circuits have decreased the execution time of multiplication/division instructions. In fact, these instructions can be executed about four times faster than with the microprogramming method.

TA + TB: 32-bit temporary register/shifter for multiplication and division instructions.

TB: 16-bit temporary register/shifter for shift/rotation instructions.

Loop Counter (LC)

This counter is used to count the number of loops for a primitive block transfer instruction controlled by a repeat prefix instruction and the number of shifts that will be performed for a multiple bit shift/rotation instruction.

The processing performed for a multiple bit rotation of a register is shown below. The average speed is approximately doubled over the microprogram method.

Example

RORC AW,CL ; CL = 5

Microprogram method	**LC method**
8 + (4 × 5) = 28 clocks	7 + 5 = 12 clocks

Program Counter and Prefetch Pointer (PC and PFP)

The CXQ70116 microprocessor has a program counter (PC), which addresses the program memory location of the instruction to be executed next, and a prefetch pointer (PFP), which addresses the program memory location to be accessed next. Both functions are provided in hardware. A time saving of several clocks is realized for branch, call, return, and break instruction execution, compared with microprocessors that have only one instruction pointer.

Enhanced Instructions

In addition to the 8086/88 instructions, the CXQ70116 has the following enhanced instructions.

Instruction	Function
PUSH imm	Pushes immediate data onto stack
PUSH R	Pushes 8 general registers onto stack
POP imm	Pops immediate data onto stack
POP R	Pops 8 general registers from stack
MUL imm	Executes 16-bit multiply of register or memory contents by immediate data
SHL imm8 SHR imm8 SHRA imm8 ROL imm8 ROR imm8 ROLC imm8 RORC imm8	Shifts/rotates register or memory by immediate value
CHKIND	Checks array index against designated boundaries
INM	Moves a string from an I/O port to memory
OUTM	Moves a string from memory to an I/O port
PREPARE	Allocates an area for a stack frame and copies previous frame pointers
DISPOSE	Frees the current stack frame on a procedure exit

Enhanced Stack Operation Instructions

PUSH imm/POP imm

These instructions allow immediate data to be pushed onto or popped from the stack.

PUSH R/POP R

These instructions allow the contents of the eight general registers to be pushed onto or popped from the stack with a single instruction.

Enhanced Multiplication Instructions

MUL reg16, imm16/MUL mem16, imm16

These instructions allow the contents of a register or memory location to be 16-bit multiplied by immediate data.

Enhanced Shift and Rotate Instructions

SHL reg, imm8/SHR reg, imm8/SHRA reg, imm8

These instructions allow the contents of a register to be shifted by the number of bits defined by the immediate data.

ROL reg, imm8/ROR reg, imm8/ROLC reg, imm8/RORC reg, imm8

These instructions allow the contents of a register to be rotated by the number of bits defined by the immediate data.

Check Array Boundary Instruction

CHKIND reg16, mem32

This instruction is used to verify that index values pointing to the elements of an array data structure are within the defined range. The lower limit of the array should be in memory location mem32, the upper limit in mem32 + 2. If the index value in reg16 is not between these limits when CHKIND is executed, a BRK 5 will occur. This causes a jump to the location in interrupt vector 5.

Block I/O Instructions

OUTM DW, src-block/INM dst-block, DW

These instructions are used to output or input a string to or from memory, when preceded by a repeat prefix.

Stack Frame Instructions

PREPARE imm16, imm8

This instruction is used to generate the stack frames required by block-structured languages, such as PASCAL and Ada. The stack frame consists of two area. One area has a pointer that points to another frame which has variables that the current frame can access. The other is a local variable area for the current procedure.

DISPOSE

This instruction releases the last stack frame generated by the PREPARE instruction. It returns the stack and base pointers to the values they had before the PREPARE instruction was used to call a procedure.

Unique Instructions

In addition to the 8086/88 instructions and the enhanced instructions, the CXQ70116 has the following unique instructions.

Instruction	Function
INS	Insert bit field
EXT	Extract bit field
ADD4S	Adds packed decimal strings
SUB4S	Subtracts one packed decimal string from another
CMP4S	Compares two packed decimal strings
ROL4	Rotates one BCD digit left through AL lower 4 bits
ROR4	Rotates one BCD digit right through AL lower 4 bits
TEST1	Tests a specified bit and sets/resets Z flag
NOT1	Inverts a specified bit
CLR1	Clears a specified bit
SET1	Sets a specified bit
REPC	Repeats next instruction until CY flag is cleared
REPNC	Repeats next instruction until CY flag is set
FPO2	Additional floating point processor call

Variable Length Bit Field Operation Instructions

This category has two instructions: INS (Insert Bit Field) and EXT (Extract Bit Field). These instructions are highly effective for computer graphics and high-level languages. They can, for example, be used for data structures such as packed arrays and record type data used in PASCAL.

INS reg8, reg8/INS reg8, imm4

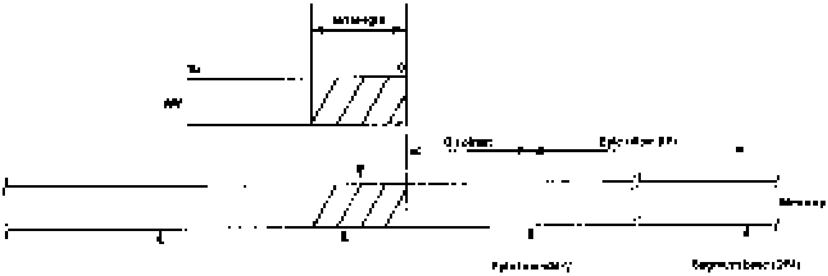
This instruction (figure 5) transfers low bits from the 16-bit AW register (the number of bits is specified by the second operand) to the memory location specified by the segment base (DS1 register) plus the byte offset (IY register). The starting bit position within this byte is specified as an offset by the lower 4-bits of the first operand.

After each complete data transfer, the IY register and the register specified by the first operand are automatically updated to point to the next bit field.

Either immediate data or a register may specify the number of bits transferred (second operand). Because the maximum transferable bit length is 16-bits, only the lower 4-bits of the specified register (00H to 0FH) will be valid.

Bit field data may overlap the byte boundary of memory.

Fig. 3. Bit Field Insertion



EXT reg8, reg8/EXT reg8, imm4

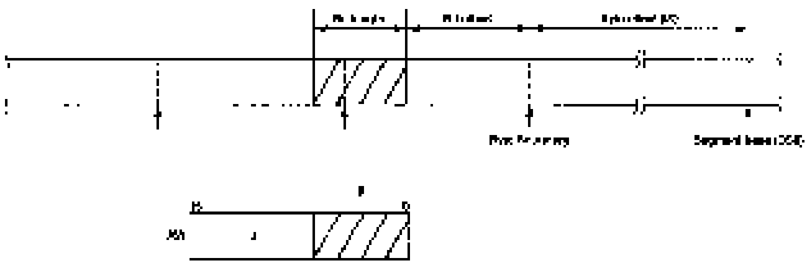
This instruction (figure 4) loads to the AW register the bit field data whose bit length is specified by the second operand of the instruction from the memory location that is specified by the DS0 segment register (segment base), the IX index register (byte offset), and the lower 4-bits of the first operand (bit offset).

After the transfer is complete, the IX register and the lower 4-bits of the first operand are automatically updated to point to the next bit field.

Either immediate data or a register may be specified for the second operand. Because the maximum transferable bit length is 16 bits, however, only the lower 4-bits of the specified register (0H to 0FH) will be valid.

Bit field data may overlap the byte boundary of memory.

Fig. 4. Bit Field Extraction



Packed BCD Operation Instructions

The instructions described here process packed BCD data either as strings (ADD4S, SUB4S, CMP4S) or byte-format operands (ROR4, ROL4). Packed BCD strings may be from 1 to 255 digits in length.

When the number of digits is even, the zero and carry flags will be set according to the result of the operation. When the number of digits is odd, the zero and carry flags may not be set correctly. In this case, (CL = odd), the zero flag will not be set unless the upper 4 bits of the highest byte are all zero. The carry flag will not be set unless there is a carry out of the upper 4 bits of the highest byte. When CL is odd, the contents of the upper 4 bits of the highest byte of the result are undefined.

ADD4S

This instruction adds the packed BCD string addressed by the IX index register to the packed BCD string addressed by the IY index register, and stores the result in the string addressed by the IY register. The length of the string (number of BCD digits) is specified by the CL register, and the result of the operation will affect the carry flag (CY) and zero flag (Z).

BCD string (IY, CL) ← BCD string (IY, CL) + BCD string (IX, CL)

SUB4S

This instruction subtracts the packed BCD string addressed by the IX index register from the packed BCD string addressed by the IY index register, and stores the result in the string addressed by the IY register. The length of the string (number of BCD digits) is specified by the CL register, and the result of the operation will affect the carry flag (CY) and zero flag (Z).

BCD string (IY, CL) ← BCD string (IY, CL) − BCD String (IX, CL)

CMP4S

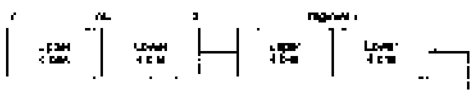
This instruction performs the same operation as SUB4S except that the result is not stored and only carry flag (CY) and zero flag (Z) are affected.

BCD string (IY, CL) − BCD string (IX, CL)

ROL4

This instruction (figure 5) treats the byte data of the register or memory directly specified by the instruction byte as BCD data and uses the lower 4 bits of the AL register (AL_L) to rotate that data one BCD digit to the left.

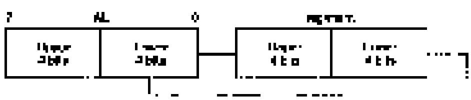
Fig. 5 BCD Rotate Left (ROL4)



ROR4

This instruction (figure 6) treats the byte data of the register or memory directly specified by the instruction byte as BCD data and uses the lower 4 bits of the AL register (AL_L) to rotate that data one BCD digit to the right.

Fig. 6. BCD Rotate Right (ROR4)



Bit Manipulation Instructions

TEST1

This instruction tests a specific bit in a register or memory location. If the bit is 1, the Z flag is reset to 0. If the bit is 0, the Z flag is set to 1.

NOT1

This instruction inverts a specific bit in a register or memory location.

CLR1

This instruction clears a specific bit in a register or memory location.

SET1

This instruction sets a specific bit in a register or memory location.

Repeat Prefix Instructions

REPC

This instruction causes the CXQ70116 to repeat the following primitive block transfer instruction until the CY flag becomes cleared or the CW register becomes zero.

REPNC

This instruction causes the CXQ70116 to repeat the following primitive block transfer instruction until the CY flag becomes set or the CW register becomes zero.

Floating Point Instruction

FPO2

This instruction is in addition to the 8088/86 floating point instruction, FPO1. These instructions are covered in a later section.

Mode Operation Instructions

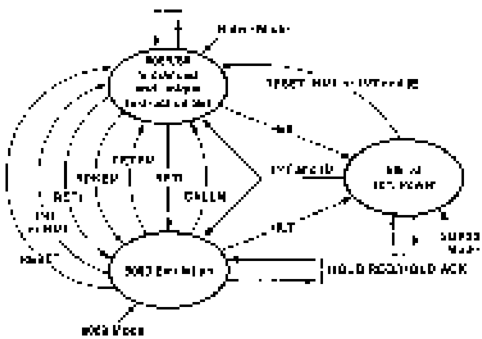
The CXQ70116 has two operating modes (Figure 7). One is the native mode which executes 8088/86, enhanced and unique instructions. The other is the 8080 emulation mode in which the instruction set of the 8080 is emulated. A mode flag (MD) is provided to select between these two modes. Native mode is selected when MD is 1 and emulation mode when MD is 0. MD is set and reset, directly and indirectly, by executing the mode manipulation instructions.

Two instructions are provided to switch operation from the native mode to the emulation mode and back: BRKEM (Break for Emulation), and RETEM (Return from Emulation).

Two instructions are used to switch from the emulation mode to the native mode and back: CALLN (Call Native Routine), and RETI (Return from Interrupt).

The system will return from the 8080 emulation mode to the native mode when the RESET signal is present, or when an external interrupt (NMI or INT) is present.

Fig. 7. Operating Modes



BRKEM imm8

This is the basic instruction used to start the 8080 emulation mode. This instruction operates exactly the same as the BRK instruction, except that BRKEM resets the mode flag (MD) to 0. PSW, PS and PC are saved to the stack. MD is then reset and the interrupt vector specified by the operand imm8 of this command is loaded into PS and PC.

The instruction codes of the interrupt processing routine jumped to are then fetched. Then the CPU executes these codes as 8080 instructions.

In 8080 emulation mode, registers and flags of the 8080 are performed by the following registers and flags of the CXQ70116.

	8080	CXQ70116
Registers:	A	AL
	B	CH
	C	CL
	D	DH
	E	DL
	H	BH
	L	BL
	SP	BP
	PC	PC
Flags:	C	CY
	Z	Z
	S	S
	P	P
	AC	AC

In the native mode, SP is used for the stack pointer. In the 8080 emulation mode this function is performed by BP.

This use of independent stack pointers allows independent stack areas to be secured for each mode and keeps the stack of one of the modes from being destroyed by an erroneous stack operation in the other mode.

The SP, IX, IY and AH registers and the four segment registers (PS, SS, DS0, and DS1) used in the native mode are not affected by operations in 8080 emulation mode.

In the 8080 emulation mode, the segment register for instructions is determined by the PS register (set automatically by the interrupt vector) and the segment register for data is the DS0 register (set by the programmer immediately before the 8080 emulation mode is entered).

RETEM [no operand]

When RETEM is executed in 8080 emulation mode (interpreted by the CPU as 8080 instruction), the CPU restores PS, PC, and PSW (as it would when returning from an interrupt processing routine), and returns to the native mode. At the same time, the contents of the mode flag (MD) which was saved to the stack by the BRKEM instruction, is restored to MD = 1. The CPU is set to the native mode.

CALLN imm8

This instruction makes it possible to call the native mode subroutines from the 8080 emulation mode. To return from subroutine to the 8080 emulation mode, the RETI instruction is used.

The processing performed when this instruction is executed in the 8080 emulation mode (it is interpreted by the CPU as 8080 instruction), is similar to that performed when a BRK instruction is executed in the native mode. The imm8 operand specifies an interrupt vector type. The contents of PS, PC and PSW are pushed on the stack and an MD flag value of 0 is saved. The mode flag is set to 1 and the interrupt vector specified by the operand is loaded into PS and PC.

RETI [no operand]

This is a general-purpose instruction used to return from interrupt routines entered by the BRK instruction or by an external interrupt in the native mode. When this instruction is executed at the end of a subroutine entered by the execution of the CALLN instruction, the operation that restores PS, PC and PSW is exactly the same as the native mode execution. When PSW is restored, however, the 8080 emulation mode value of the mode flag (MD) is restored, the CPU is set in emulation mode, and all subsequent instructions are interpreted and executed as 8080 instructions.

RETI is also used to return from an interrupt procedure initiated by an NMI or INT interrupt in the emulation mode.

Floating Point Operation Chip Instructions

FPO1 fp-op, mem/FPO2 fp-op, mem

These instructions are used for the external floating point processor. The floating point operation is passed to the floating point processor when the CPU fetches one of these instructions. From this point the CPU performs only the necessary auxiliary processing (effective address calculation, generation of physical addresses, and start-up of the memory read cycle).

The floating point processor always monitors the instructions fetched by the CPU. When it interprets one as an instruction to itself, it performs the appropriate processing. At this time, the floating point processor chip uses either the address alone or both the address and read data of the memory read cycle executed by the CPU. This difference in the data used depends on which of these instructions is executed.

Note: During the memory read cycle initiated by the CPU for FPO1 or FPO2 execution, the CPU does not accept any read data on the data bus from memory. Although the CPU generates the memory address, the data is used by the floating point processor.

Interrupt Operation

The interrupts used in the CXQ70116 can be divided into two types: interrupts generated by external interrupt requests and interrupts generated by software processing. These are the classifications.

External interrupts

(a) NMI input (nonmaskable)
(b) INT input (maskable)

Software processing

As the result of instruction execution
— When a divide error occurs during execution of the DIV or DIVU instruction
— When a memory-boundary-over error is detected by the CHKIND instruction
Conditional break instruction
— When V = 1 during execution of the BRKV instruction
Unconditional break instructions
— 1-byte break instruction: BRK3
— 2-byte break instruction: BRK imm8
Flag processing
— When stack operations are used to set the BRK flag
8080 Emulation mode instructions
— BRKEM imm8
— CALLN imm8

Interrupt vectors

Starting addresses for interrupt processing routines are either determined automatically by a single location of the interrupt vector table or selected each time interrupt processing is entered.

The interrupt vector table is shown in figure 8. The table uses 1K bytes of memory addresses 000H to 3FFH and can store starting address data for a maximum of 256 vectors (4 bytes per vector).

The corresponding interrupt sources for vectors 0 to 5 are predetermined and vectors 6 to 31 are reserved. These vectors consequently cannot be used for general applications.

The BRKEM instruction and CALLN instruction (in the emulation mode) and the INT input are available for general applications for vectors 32 to 255.

A single interrupt vector is made up of 4 bytes (figure 9). The 2 bytes in the low addresses of memory are loaded into PC as the offset, and the high 2 bytes are loaded into PS as the base address. The bytes are combined in reverse order. The lower-order bytes in the vector become the most significant bytes in the PC and PS, and the higher-order bytes become the least significant bytes.

Fig. 8. Interrupt Vector Table

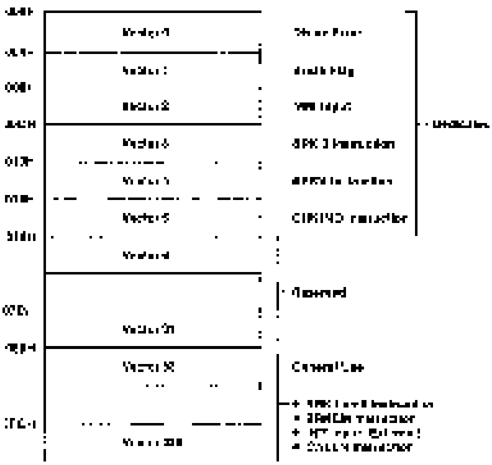
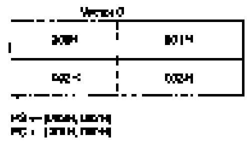


Fig. 9. Interrupt Vector 0



Based on this format, the contents of each vector should be initialized at the beginning of the program. The basic steps to jump to an interrupt processing routine are now shown.

(SP −1, SP −2) ← PSW
(SP −3, SP −4) ← PS
(SP −5, SP −6) ← PC
SP ← SP −6
IE ← 0, BRK ← 0, MD ← 1
PS ← vector high bytes
PC ← vector low bytes

Standby Function

The CXQ70116 has a standby mode to reduce power consumption during program wait states. This mode is set by the HALT instruction in both the native and the emulation mode.

In the standby mode, the internal clock is supplied only to those circuits related to functions required to release this mode and bus hold control functions. As a result, power consumption can be reduced to 1/10 the level of normal operation in either native or emulation mode.

The standby mode is released by inputting a RESET signal or an external interrupt (NMI, INT).

The bus hold function is effective during standby mode. The CPU returns to standby mode when the bus hold request is removed.

During standby mode, all control outputs are disabled and the address/data bus will be either high or low.

Instruction Set

The following tables briefly describe the CXQ70116's instruction set.

- Operation and Operand Types — defines abbreviations used in the Instruction Set table.
- Flag Operations — defines the symbols used to describe flag operations.
- Memory Addressing — shows how mem and mod combinations specify memory addressing modes.
- Selection of 8- and 16-Bit Registers — shows how reg and W select a register when mod = 111.
- Selection of Segment Registers — shows how sreg selects a segment register.
- Instruction Set — shows the instruction mnemonics, their effect, their operation codes, the number of bytes in the instruction, the number of clocks required for execution, and the effect on the CXQ70116 flags.

Operation and Operand Types

Identifier	Description
reg	8- or 16-bit general-purpose register
reg8	8-bit general-purpose register
reg16	16-bit general-purpose register
dmem	8- or 16-bit direct memory location
mem	8- or 16-bit memory location
mem8	8-bit memory location
mem16	16-bit memory location
mem32	32-bit memory location
imm	Constant (0 to FFFFH)
imm16	Constant (0 to FFFFH)
imm8	Constant (0 to FFH)
imm4	Constant (0 to FH)
imm3	Constant (0 to 7)
acc	AW or AL register
sreg	Segment register
src-table	Name of 256-byte translation table

Identifier	Description
src-block	Name of block addressed by the IX register
dst-block	Name of block addressed by the IY register
near-proc	Procedure within the current program segment
far-proc	Procedure located in another program segment
near-label	Label in the current program segment
short-label	Label between −128 and +127 bytes from the end of instruction
far-label	Label in another program segment
memptr16	Word containing the offset of the memory location within the current program segment to which control is to be transferred
memptr32	Double word containing the offset and segment base address of the memory location to which control is to be transferred
regptr16	16-bit register containing the offset of the memory location within the program segment to which control is to be transferred
pop-value	Number of bytes of the stack to be discarded (0 to 64K bytes, usually even addresses)
fp-op	Immediate data to identify the instruction code of the external floating point operation
R	Register set
W	Word/byte field (0 to 1)
reg	Register field (000 to 111)
mem	Memory field (000 to 111)
mod	Mode field (00 to 10)
S:W	When S:W=01 or 11, data=16 bits. At all other times, data=8 bits.
X, XXX, YYY, ZZZ	Data to identify the instruction code of the external floating point arithmetic chip
AW	Accumulator (16 bits)
AH	Accumulator (high byte)
AL	Accumulator (low byte)
BW	BW register (16 bits)
CW	CW register (16 bits)
CL	CW register (low byte)
DW	DW register (16 bits)
SP	Stack pointer (16 bits)
PC	Program counter (16 bits)
PSW	Program status word (16 bits)
IX	Index register (source) (16 bits)
IY	Index register (destination) (16 bits)

Identifier	Description
PS	Program segment register (16 bits)
SS	Stack segment register (16 bits)
DS_0	Data segment 0 register (16 bits)
DS_1	Data segment 1 register (16 bits)
AC	Auxiliary carry flag
CY	Carry flag
P	Parity flag
S	Sign flag
Z	Zero flag
DIR	Direction flag
IE	Interrupt enable flag
V	Overflow flag
BRK	Break flag
MD	Mode flag
(...)	Values in parentheses are memory contents
disp	Displacement (8 or 16 bits)
ext-disp8	16-bit displacement (sign-extension byte + 8-bit displacement)
temp	Temporary register (8/16/32 bits)
tmpcy	Temporary carry flag (1 bit)
seg	Immediate segment data (16 bits)
offset	Immediate offset data (16 bits)
←	Transfer direction
+	Addition
−	Subtraction
×	Multiplication
÷	Division
%	Modulo
AND	Logical product
OR	Logical sum
XOR	Exclusive logical sum
XXH	Two-digit hexadecimal value
XXXXH	Four-digit hexadecimal value

Flag Operations

Identifier	Description
(blank)	No change
0	Cleared to 0
1	Set to 1
X	Set or cleared according to the result
U	Undefined
R	Value saved earlier is restored

Memory Addressing

mem	mod 00	mod 01	mod 10
000	BW + IX	BW + IX + disp8	BW + IX + disp16
001	BW + IY	BW + IY + disp8	BW + IY + disp16
010	BP + IX	BP + IX + disp8	BP + IX + disp16
011	BP + IY	BP + IY + disp8	BP + IY + disp16
100	IX	IX + disp8	IX + disp16
101	IY	IY + disp8	IY + disp16
110	Direct address	BP + disp8	BP + disp16
111	BW	BW + disp8	BW + disp16

Selection of 8-and 16-Bit Registers (mod 11)

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

Selection of Segment Registers

sreg	
00	DS1
01	PS
10	SS
11	DS0

The table on the following pages shows the instruction set.

At "No. of Clocks," for instructions referencing memory operands, the left side of the slash (/) is the number of clocks for byte operands or word operands at an even address, and the right side is for word operands at an odd address. For conditional control transfer instructions, the left side of the slash (/) is the number of clocks if a control transfer takes place. The right side is the number of clocks when no control transfer or branch occurs. Some instructions show a range of clock times, separated by a hyphen. The execution time of these instructions varies from the minimum value to the maximum, depending on the operands involved.

Note: Add four clocks to these times for each word transfer made to an odd address.

"No. of Clocks" includes these times:

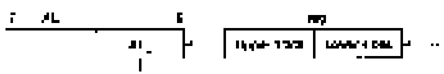
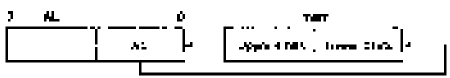
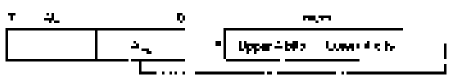
- Decoding
- Effective address generation
- Operand fetch
- Execution

It assumes that the instruction bytes have been prefetched.

Mnemonic	Operand	Operation	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	No. of Clocks	No. of Bytes	AC	CY	V	P	S	Z
Data Transfer Instructions																										
MOV	reg, reg	reg ← reg	1	0	0	0	1	0	1	W	1	1	reg			reg			2	2						
	mem, reg	(mem) ← reg	1	0	0	0	1	0	0	W	mod		reg			mem			9/13	2-4						
	reg, mem	reg ← (mem)	1	0	0	0	1	0	1	W	mod		reg			mem			11/15	2-4						
	mem, imm	(mem) ← imm	1	1	0	0	0	1	1	W	mod		0	0	0	mem			11/15	3-6						
	reg, imm	reg ← imm	1	0	1	1	W	reg											4	2-3						
	acc, dmem	When W = 0 AL ← (dmem) When W = 1 AH ← (dmem + 1), AL ← (dmem)	1	0	1	0	0	0	0	W									10/14	3						
	dmem, acc	When W = 0 (dmem) ← AL When W = 1 (dmem + 1) ← AH, (dmem) ← AL	1	0	1	0	0	0	1	W									9/13	3						
	sreg, reg16	sreg ← reg16 sreg : SS, DS0, DS1	1	0	0	0	1	1	1	0	1	1	0	sreg		reg			2	2						
	sreg, mem16	sreg ← (mem16) sreg : SS, DS0, DS1	1	0	0	0	1	1	1	0	mod		0	sreg		mem			11/15	2-4						
	reg16, sreg	reg16 ← sreg	1	0	0	0	1	1	0	0	1	1	0	sreg		reg			2	2						
	mem16, sreg	(mem16) ← sreg	1	0	0	0	1	1	0	0	mod		0	sreg		mem			10/14	2-4						
	DS0, reg16, mem32	reg16 ← (mem32) DS0 ← (mem32 + 2)	1	1	0	0	0	1	0	1	mod		reg			mem			18/26	2-4						
	DS1, reg16, mem32	reg16 ← (mem32) DS1 ← (mem32 + 2)	1	1	0	0	0	1	0	0	mod		reg			mem			18/26	2-4						
	AH, PSW	AH ← S, Z, x, AC, x, P, x, CY	1	0	0	1	1	1	1	1									2	1	x	x		x	x	x
	PSW, AH	S, Z, x, AC, x, P, x, CY ← AH	1	0	0	1	1	1	1	0									3	1	x	x		x	x	x
LDEA	reg16, mem16	reg16 ← mem16	1	0	0	0	1	1	0	1	mod		reg			mem			4	2-4						
TRANS	src-table	AL ← (BW + AL)	1	1	0	1	0	1	1	1									9	1						
XCH	reg, reg	reg ←→ reg	1	0	0	0	0	1	1	W	1	1	reg			reg			3	2						
	mem, reg or reg, mem	(mem) ←→ reg	1	0	0	0	0	1	1	W	mod		reg			mem			16/24	2-4						
	AW, reg16 or reg16, AW	AW ←→ reg16	1	0	0	1	0	reg											3	1						
Repeat Prefixes																										
REPC		While CW ≠ 0, the following primitive block transfer instruction is executed and CW is decremented (−1). If there is a waiting interrupt, it is processed. When CY ≠ 1, exit the loop.	0	1	1	0	0	1	0	1									2	1						
REPNC		While CW ≠ 0, the following primitive block transfer instruction is executed and CW is decremented (−1). If there is a waiting interrupt, it is processed. When CY ≠ 0, exit the loop.	0	1	1	0	0	1	0	0									2	1						

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	Operation Code 7 6 5 4 3 2 1 0	No. of Clocks	No. of Bytes	AC	CY	V	P	S	Z
Repeat Prefixed (cont)												
REP REPE REPZ		While CW ≠ 0, the following primitive block transfer instruction is executed and CW is decremented (−1). If there is a waiting interrupt, it is processed. If the primitive block transfer instruction is CMPBK or CMPM and Z ≠ 1, exit the loop.	1 1 1 1 0 0 1 1		2	1						
REPNE REPNZ		While CW ≠ 0, the following primitive block transfer instruction is executed and CW is decremented (−1). If there is a waiting interrupt, it is processed. If the primitive block transfer instruction is CMPBK or CMPM and Z ≠ 0, exit the loop.	1 1 1 1 0 0 1 0		2	1						
Primitive Block Transfer Instructions												
MOVBK	dst-block, src-block	When W = 0: (IY) ← (IX) DIR = 0: IX ← IX + 1, IY ← IY + 1 DIR = 1: IX ← IX − 1, IY ← IY − 1 When W = 1: (IY + 1, IY) ← (IX + 1, IX) DIR = 0: IX ← IX + 2, IY ← IY + 2 DIR = 1: IX ← IX − 2, IY ← IY − 2	1 0 1 0 0 1 0 W		11 + 8n	1						
CMPBK	dst-block, src-block	When W = 0: (IX) − (IY) DIR = 0: IX ← IX + 1, IY ← IY + 1 DIR = 1: IX ← IX − 1, IY ← IY − 1 When W = 1: (IX + 1, IX) − (IY + 1, IY) DIR = 0: IX ← IX + 2, IY ← IY + 2 DIR = 1: IX ← IX − 2, IY ← IY − 2	1 0 1 0 0 1 1 W		7 + 14n	1	x	x	x	x	x	x
CMPM	dst-block	When W = 0: AL − (IY) DIR = 0: IY ← IY + 1; DIR = 1: IY ← IY − 1 When W = 1: AW − (IY + 1, IY) DIR = 0: IY ← IY + 2; DIR = 1: IY ← IY − 2	1 0 1 0 1 1 1 W		7 + 10n	1	x	x	x	x	x	x
LDM	src-block	When W = 0: AL ← (IX) DIR = 0: IX ← IX + 1; DIR = 1: IX ← IX − 1 When W = 1: AW ← (IX + 1, IX) DIR = 0: IX ← IX + 2; DIR = 1: IX ← IX − 2	1 0 1 0 1 1 0 W		7 + 9n	1						
STM	dst-block	When W = 0: (IY) ← AL DIR = 0: IY ← IY + 1; DIR = 1: IY ← IY − 1 When W = 1: (IY + 1, IY) ← AW DIR = 0: IY ← IY + 2; DIR = 1: IY ← IY − 2	1 0 1 0 1 0 1 W	n: number of transfers	7 + 4n	1						
Bit Field Transfer Instructions												
INS	reg8, reg8	16 Bit field ← AW	0 0 0 0 1 1 1 1 1 1 reg reg	0 0 1 1 0 0 0 1	31 − 117 (35 − 133)	3						
	reg8, imm4	16 Bit field ← AW	0 0 0 0 1 1 1 1 1 1 0 0 0 reg	0 0 1 1 1 0 0 1	67 − 87 (75 − 103)	4						

Mnemonic	Operand	Operation	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	No. of Clocks	No. of Bytes	AC	CY	V	P	S	Z
Bit Field Transfer Instructions (cont)																										
EXT	reg8, reg8	AW ← 16-Bit field	0 1	0 1	0 reg	0	1	1 reg	1	1	0	0	1	1	0	0	1	1	26-55 (34-59)	3						
	reg8, imm4	AW ← 16-Bit field	0 1	0 1	0 0	0 0	1 0	1 reg	1	1	0	0	1	1	1	0	1	1	21-45 (25-52)	4						
I/O Instructions																										
IN	acc, imm8	When W = 0 AL ← (imm8) When W = 1 AH ← (imm8 + 1), AL ← (imm8)	1	1	1	0	0	1	0	W									9/13	2						
	acc, DW	When W = 0 AL ← (DW) When W = 1 AH ← (DW + 1), AL ← (DW)	1	1	1	0	1	1	0	W									8/12	1						
OUT	imm8, acc	When W = 0 (imm8) ← AL When W = 1 (imm8 + 1) ← AH, (imm8) ← AL	1	1	1	0	0	1	1	W									8/12	2						
	DW, acc	When W = 0 (DW) ← AL When W = 1 (DW + 1) ← AH, (DW) ← AL	1	1	1	0	1	1	1	W									8/12	1						
Primitive I/O Instructions																										
INM	dst-block, DW	When W = 0 (IY) ← (DW) DIR = 0: IY ← IY + 1; DIR = 1: IY ← IY − 1 When W = 1 (IY + 1, IY) ← (DW + 1, DW) DIR = 0: IY ← IY + 2; DIR = 1: IY ← IY − 2	0	1	1	0	1	1	0	W									9 + 8n	1						
OUTM	DW, src-block	When W = 0 (DW) ← (IX) DIR = 0: IX ← IX + 1; DIR = 1: IX ← IX − 1 When W = 1 (DW + 1, DW) ← (IX + 1, IX) DIR = 0: IX ← IX + 2; DIR = 1: IX ← IX − 2	0	1	1	0	1	1	1	W	n: number of transfers								9 + 8n	1						
Addition/Subtraction Instructions																										
ADD	reg, reg	reg ← reg + reg	0	0	0	0	0	0	1	W	1	1	reg			reg			2	2	x	x	x	x	x	x
	mem, reg	(mem) ← (mem) + reg	0	0	0	0	0	0	0	W	mod		reg			mem			16/24	2-4	x	x	x	x	x	x
	reg, mem	reg ← reg + (mem)	0	0	0	0	0	0	1	W	mod		reg			mem			11/15	2-4	x	x	x	x	x	x
	reg, imm	reg ← reg + imm	1	0	0	0	0	0	S	W	1	1	0	0	0	reg			4	3-4	x	x	x	x	x	x
	mem, imm	(mem) ← (mem) + imm	1	0	0	0	0	0	S	W	mod		0	0	0	mem			18/26	3-6	x	x	x	x	x	x
	acc, imm	When W = 0 AL ← AL + imm When W = 1 AW ← AW + imm	0	0	0	0	0	1	0	W									4	2-3	x	x	x	x	x	x
ADDC	reg, reg	reg ← reg + reg + CY	0	0	0	1	0	0	1	W	1	1	reg			reg			2	2	x	x	x	x	x	x
	mem, reg	(mem) ← (mem) + reg + CY	0	0	0	1	0	0	0	W	mod		reg			mem			16/24	2-4	x	x	x	x	x	x
	reg, mem	reg ← reg + (mem) + CY	0	0	0	1	0	0	1	W	mod		reg			mem			11/15	2-4	x	x	x	x	x	x
	reg, imm	reg ← reg + imm + CY	1	0	0	0	0	0	S	W	1	1	0	1	0	reg			4	3-4	x	x	x	x	x	x
	mem, imm	(mem) ← (mem) + imm + CY	1	0	0	0	0	0	S	W	mod		0	1	0	mem			18/26	3-6	x	x	x	x	x	x

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Clocks	No. of Bytes	AC	CY	V	P	S	Z
Addition/Subtraction Instructions (cont.)												
ADDC	acc, imm	When W = 0 AL ← AL + imm + CY When W = 1 AW ← AW + imm + CY	0 0 0 1 0 1 0 W		4	2-3	x	x	x	x	x	x
SUB	reg, reg	reg ← reg − reg	0 0 1 0 1 0 1 W	1 1 reg reg	2	2	x	x	x	x	x	x
	mem, reg	(mem) ← (mem) − reg	0 0 1 0 1 0 0 W	mod reg mem	16/24	2-4	x	x	x	x	x	x
	reg, mem	reg ← reg − (mem)	0 0 1 0 1 0 1 W	mod reg mem	11/15	2-4	x	x	x	x	x	x
	reg, imm	reg ← reg − imm	1 0 0 0 0 0 S W	1 1 1 0 1 reg	4	3-4	x	x	x	x	x	x
	mem, imm	(mem) ← (mem) − imm	1 0 0 0 0 0 S W	mod 1 0 1 mem	18/26	3-6	x	x	x	x	x	x
	acc, imm	When W = 0 AL ← AL − imm When W = 1 AW ← AW − imm	0 0 1 0 1 1 0 W		4	2-3	x	x	x	x	x	x
SUBC	reg, reg	reg ← reg − reg − CY	0 0 0 1 1 0 1 W	1 1 reg reg	2	2	x	x	x	x	x	x
	mem, reg	(mem) ← (mem) − reg − CY	0 0 0 1 1 0 0 W	mod reg mem	16/24	2-4	x	x	x	x	x	x
	reg, mem	reg ← reg − (mem) − CY	0 0 0 1 1 0 1 W	mod reg mem	11/15	2-4	x	x	x	x	x	x
	reg, imm	reg ← reg − imm − CY	1 0 0 0 0 0 S W	1 1 0 1 1 reg	4	3-4	x	x	x	x	x	x
	mem, imm	(mem) ← (mem) − imm − CY	1 0 0 0 0 0 S W	mod 0 1 1 mem	18/26	3-6	x	x	x	x	x	x
	acc, imm	When W = 0 AL ← AL − imm − CY When W = 1 AW ← AW − imm − CY	0 0 0 1 1 1 0 W		4	2-3	x	x	x	x	x	x
BCD Operation Instructions												
ADD4S		dst BCD string ← dst BCD string + src BCD string	0 0 0 0 1 1 1 1	0 0 1 0 0 0 0 0	7 + 19n	2	u	x	u	u	u	x
SUB4S		dst BCD string ← dst BCD string − src BCD string	0 0 0 0 1 1 1 1	0 0 1 0 0 0 1 0	7 + 19n	2	u	x	u	u	u	x
CMP4S		dst BCD string − src BCD string	0 0 0 0 1 1 1 1	0 0 1 0 0 1 1 0	7 + 19n	2	u	x	u	u	u	x
			n: number of BCD numerals divided by 2									
ROL4	reg8		0 0 0 0 1 1 1 1 1 1 0 0 0 reg	0 0 1 0 1 0 0 0	25	3						
	mem8		0 0 0 0 1 1 1 1 mod 0 0 0 mem	0 0 1 0 1 0 0 0	28	3-5						
ROR4	reg8		0 0 0 0 1 1 1 1 1 1 0 0 0 reg	0 0 1 0 1 0 1 0	29	3						
	mem8		0 0 0 0 1 1 1 1 mod 0 0 0 mem	0 0 1 0 1 0 1 0	33	3-5						

Mnemonic	Operand	Operation	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	No. of Clocks	No. of Bytes	AC	CY	V	P	S	Z
Increment/Decrement Instructions (cont)																										
INC	reg8	reg8 ← reg8 + 1	1	1	1	1	1	1	1	0	1	1	0	0	0	reg			2	2	x		x	x	x	x
	mem	(mem) ← (mem) + 1	1	1	1	1	1	1	1	W	mod		0	0	0	mem			15/21	2-4	x		x	x	x	x
	reg16	reg16 ← reg16 + 1	0	1	0	0	0	reg											2	1	x		x	x	x	x
DEC	reg8	reg8 ← reg8 − 1	1	1	1	1	1	1	1	0	1	1	0	0	1	reg			2	2	x		x	x	x	x
	mem	(mem) ← (mem) − 1	1	1	1	1	1	1	1	W	mod		0	0	1	mem			16/21	2-4	x		x	x	x	x
	reg16	reg16 ← reg16 − 1	0	1	0	0	1	reg											2	1	x		x	x	x	x
Multiplication Instructions																										
MULU	reg8	AW ← AL × reg8 AH = 0: CY ← 0, V ← 0 AH ≠ 0: CY ← 1, V ← 1	1	1	1	1	0	1	1	0	1	1	1	0	0	reg			21-22	2	u	x	x	u	u	u
	mem8	AW ← AL × (mem8) AH = 0: CY ← 0, V ← 0 AH ≠ 0: CY ← 1, V ← 1	1	1	1	1	0	1	1	0	mod		1	0	0	mem			27-28	2-4	u	x	x	u	u	u
	reg16	DW, AW ← AW × reg16 DW = 0: CY ← 0, V ← 0 DW ≠ 0: CY ← 1, V ← 1	1	1	1	1	0	1	1	1	1	1	1	0	0	reg			29-30	2	u	x	x	u	u	u
	mem16	DW, AW ← AW × (mem16) DW = 0: CY ← 0, V ← 0 DW ≠ 0: CY ← 1, V ← 1	1	1	1	1	0	1	1	1	mod		1	0	0	mem			35-36 (39-40)	2-4	u	x	x	u	u	u
MUL	reg8	AW ← AL × reg8 AH = AL sign expansion: CY ← 0, V ← 0 AH ≠ AL sign expansion: CY ← 1, V ← 1	1	1	1	1	0	1	1	0	1	1	1	0	1	reg			33-39	2	u	x	x	u	u	u
	mem8	AW ← AL × (mem8) AH = AL sign expansion: CY ← 0, V ← 0 AH ≠ AL sign expansion: CY ← 1, V ← 1	1	1	1	1	0	1	1	0	mod		1	0	1	mem			39-45	2-4	u	x	x	u	u	u
	reg16	DW, AW ← AW × reg16 DW = AW sign expansion: CY ← 0, V ← 0 DW ≠ AW sign expansion: CY ← 1, V ← 1	1	1	1	1	0	1	1	1	1	1	1	0	1	reg			41-47	2	u	x	x	u	u	u
	mem16	DW, AW ← AW × (mem16) DW = AW sign expansion: CY ← 0, V ← 0 DW ≠ AW sign expansion: CY ← 1, V ← 1	1	1	1	1	0	1	1	1	mod		1	0	1	mem			47-53 (51-57)	2-4	u	x	x	u	u	u
	reg16, (reg16,) imm8	reg16 ← reg16 × imm8 Product ≤ 16 bits: CY ← 0, V ← 0 Product > 16 bits: CY ← 1, V ← 1	0	1	1	0	1	0	1	1	1	1	reg			reg			28-34	3	u	x	x	u	u	u
	reg16, mem16, imm8	reg16 ← (mem16) × imm8 Product ≤ 16 bits: CY ← 0, V ← 0 Product > 16 bits: CY ← 1, V ← 1	0	1	1	0	1	0	1	1	mod		reg			mem			34-40 (38-44)	3-5	u	x	x	u	u	u

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Clocks	No. of Bytes	AC	CY	V	P	S	Z
		Multiplication Instructions (cont)										
MUL	reg16, (reg16), imm16	reg16 ← (reg16) × imm16 Product ≤ 16 bits: CY ← 0, V ← 0 Product > 16 bits: CY ← 1, V ← 1	0 1 1 0 1 0 1 1	1 1 reg reg	36-37	4	u	x	x	u	u	u
	reg16, mem16, imm16	reg16 ← (mem16) × imm16 Product ≤ 16 bits: CY ← 0, V ← 0 Product > 16 bits: CY ← 1, V ← 1	0 1 1 0 1 0 1 1	mod reg mem	42-45 / 48-52	4-6	u	x	x	u	u	u
		Unsigned Division Instructions										
DIVU	reg8	temp ← AW When temp ÷ reg8 > FFH (SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0, PS ← (3, 2), PC ← (1, 0) All other times AH ← temp % reg8, AL ← temp ÷ reg8	1 1 1 1 0 1 1 0	1 1 1 1 0 reg	19	2	u	u	u	u	u	u
	mem8	temp ← AW When temp ÷ (mem8) > FFH (SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0, PS ← (3, 2), PC ← (1, 0) All other times AH ← temp % (mem8), AL ← temp ÷ (mem8)	1 1 1 1 0 1 1 0	mod 1 1 0 mem	25	2-4	u	u	u	u	u	u
	reg16	temp ← AW When temp ÷ reg16 > FFFFH (SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0, PS ← (3, 2), PC ← (1, 0) All other times AH ← temp % reg16, AL ← temp ÷ reg16	1 1 1 1 0 1 1 1	1 1 1 1 0 reg	25	2	u	u	u	u	u	u
	mem16	temp ← AW When temp ÷ (mem16) > FFFFH (SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0, PS ← (3, 2), PC ← (1, 0) All other times AH ← temp % (mem16), AL ← temp ÷ (mem16)	1 1 1 1 0 1 1 1	mod 1 1 0 mem	31-35	2-4	u	u	u	u	u	u
		Signed Division Instructions										
DIV	reg8	temp ← AW When temp ÷ reg8 > 0 and temp ÷ reg8 > 7FH or temp ÷ reg8 < 0 and temp ÷ reg8 < 0 − 7FH − 1 (SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0, PS ← (3, 2), PC ← (1, 0) All other times AH ← temp % reg8, AL ← temp ÷ reg8	1 1 1 1 0 1 1 0	1 1 1 1 1 reg	29-34	2	u	u	u	u	u	u

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0	No. of Clocks	No. of Bytes	Flags AC CY V P S Z
		Signed Division Instructions (cont)				
DIV	mem8	temp ← AW When temp ÷ (mem8) > 0 and temp ÷ (mem8) > 7FH or temp ÷ (mem8) < 0 and temp ÷ (mem8) < 0 − 7FH − 1 (SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0, PS ← (3, 2), PC ← (1, 0) All other times AH ← temp % (mem8), AL ← temp ÷ (mem8)	1 1 1 1 0 1 1 0 mod 1 1 1 mem	35–40	2-4	u u u u u u
	reg16	temp ← AW When temp ÷ reg16 > 0 and temp ÷ reg16 > 7FFFH or temp ÷ reg16 < 0 and temp ÷ reg16 < 0 − 7FFFH − 1 (SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0, PS ← (3, 2), PC ← (1, 0) All other times AH ← temp % reg16, AL ← temp ÷ reg16	1 1 1 1 0 1 1 1 1 1 1 1 1 reg	38-43	2	u u u u u u
	mem16	temp ← AW When temp ÷ (mem16) > 0 and temp ÷ (mem16) > 7FFFH or temp ÷ (mem16) < 0 and temp ÷ (mem16) < 0 − 7FFFH − 1 (SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0, PS ← (3, 2), PC ← (1, 0) All other times AH ← temp % (mem16), AL ← temp ÷ (mem16)	1 1 1 1 0 1 1 1 mod 1 1 1 mem	44-49 /48-53	2-4	u u u u u u
		BCD Conversion Instructions				
ADJBA		When (AL AND 0FH) > 9 or AC = 1, AL ← AL + 6, AH ← AH + 1, AC ← 1, CY ← AC, AL ← AL AND 0FH	0 0 1 1 0 1 1 1	3	1	x x u u u u
ADJ4A		When (AL AND 0FH) > 9 or AC = 1, AL ← AL + 6, CY ← CY OR AC, AC ← 1, When AL > 9FH, or CY = 1 AL ← AL + 60H, CY ← 1	0 0 1 0 0 1 1 1	3	1	x x u x x x
ADJBS		When (AL AND 0FH) > 9 or AC = 1, AL ← AL − 6, AH ← AH − 1, AC ← 1, CY ← AC, AL ← AL AND 0FH	0 0 1 1 1 1 1 1	7	1	x x u u u u
ADJ4S		When (AL AND 0FH) > 9 or AC = 1, AL ← AL − 6, CY ← CY OR AC, AC ← 1 When AL > 9FH or CY = 1 AL ← AL − 60H, CY ← 1	0 0 1 0 1 1 1 1	7	1	x x u x x x

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Clocks	No. of Bytes	AC	CY	V	P	S	Z
Data Conversion Instructions												
CVTBD		AH ← AL ÷ 0AH, AL ← AL % 0AH	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0	15	2	u	u	u	x	x	x
CVTDB		AH ← 0, AL ← AH × 0AH + AL	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0	7	2	u	u	u	x	x	x
CVTBW		When AL < 80H, AH ← 0, all other times AH ← FFH	1 0 0 1 1 0 0 0		2	1						
CVTWL		When AL < 8000H, DW ← 0, all other times DW ← FFFFH	1 0 0 1 1 0 0 1		4-5	1						
Comparison Instructions												
CMP	reg, reg	reg – reg	0 0 1 1 1 0 1 W	1 1 reg reg	2	2	x	x	x	x	x	x
	mem, reg	(mem) – reg	0 0 1 1 1 0 0 W	mod reg mem	11/15	2-4	x	x	x	x	x	x
	reg, mem	reg – (mem)	0 0 1 1 1 0 1 W	mod reg mem	11/15	2-4	x	x	x	x	x	x
	reg, imm	reg – imm	1 0 0 0 0 0 S W	1 1 1 1 1 reg	4	3-4	x	x	x	x	x	x
	mem, imm	(mem) – imm	1 0 0 0 0 0 S W	mod 1 1 1 mem	13/17	3-6	x	x	x	x	x	x
	acc, imm	When W = 0, AL – imm When W = 1, AW – imm	0 0 1 1 1 1 0 W		4	2-3	x	x	x	x	x	x
Complement Instructions												
NOT	reg	reg ← $\overline{reg}$	1 1 1 1 0 1 1 W	1 1 0 1 0 reg	2	2						
	mem	(mem) ← $\overline{(mem)}$	1 1 1 1 0 1 1 W	mod 0 1 0 mem	16/24	2-4						
NEG	reg	reg ← $\overline{reg}$ + 1	1 1 1 1 0 1 1 W	1 1 0 1 1 reg	2	2	x	x	x	x	x	x
	mem	(mem) ← $\overline{(mem)}$ + 1	1 1 1 1 0 1 1 W	mod 0 1 1 mem	16/24	2-4	x	x	x	x	x	x
Logical Operation Instructions												
TEST	reg, reg	reg AND reg	1 0 0 0 0 1 0 W	1 1 reg reg	2	2	u	0	0	x	x	x
	mem, reg or reg, mem	(mem) AND reg	1 0 0 0 0 1 0 W	mod reg mem	10/14	2-4	u	0	0	x	x	x
	reg, imm	reg AND imm	1 1 1 1 0 1 1 W	1 1 0 0 0 reg	4	3-4	u	0	0	x	x	x
	mem, imm	(mem) AND imm	1 1 1 1 0 1 1 W	mod 0 0 0 mem	11/15	3-6	u	0	0	x	x	x
	acc, imm	When W = 0, AL AND imm8 When W = 1, AW AND imm16	1 0 1 0 1 0 0 W		4	2-3	u	0	0	x	x	x
AND	reg, reg	reg ← reg AND reg	0 0 1 0 0 0 1 W	1 1 reg reg	2	2	u	0	0	x	x	x
	mem, reg	(mem) ← (mem) AND reg	0 0 1 0 0 0 0 W	mod reg mem	15/24	2-4	u	0	0	x	x	x
	reg, mem	reg ← reg AND (mem)	0 0 1 0 0 0 1 W	mod reg mem	11/15	2-4	u	0	0	x	x	x
	reg, imm	reg ← reg AND imm	1 0 0 0 0 0 0 W	1 1 1 0 0 reg	4	3-4	u	0	0	x	x	x
	mem, imm	(mem) ← (mem) AND imm	1 0 0 0 0 0 0 W	mod 1 0 0 mem	18/26	3-6	u	0	0	x	x	x
	acc, imm	When W = 0, AL ← AL AND imm8 When W = 1, AW ← AW AND imm16	0 0 1 0 0 1 0 W		4	2-3	u	0	0	x	x	x

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of clocks	No. of bytes	AC	CY	V	P	S	Z
Logical Operation Instructions (cont)												
OR	reg, reg	reg ← reg OR reg	0 0 0 0 1 0 1 W	1 1 reg reg	2	2	u	0	0	x	x	x
	mem, reg	(mem) ← (mem) OR reg	0 0 0 0 1 0 0 W	mod reg mem	16/24	2-4	u	0	0	x	x	x
	reg, mem	reg ← reg OR (mem)	0 0 0 0 1 0 1 W	mod reg mem	11/15	2-4	u	0	0	x	x	x
	reg, imm	reg ← reg OR imm	1 0 0 0 0 0 0 W	1 1 0 0 1 reg	4	3-4	u	0	0	x	x	x
	mem, imm	(mem) ← (mem) OR imm	1 0 0 0 0 0 0 W	mod 0 0 1 mem	18/26	3-6	u	0	0	x	x	x
	acc, imm	When W = 0 AL ← AL OR imm8 When W = 1 AW ← AW OR imm16	0 0 0 0 1 1 0 W		4	2-3	u	0	0	x	x	x
XOR	reg, reg	reg ← reg XOR reg	0 0 1 1 0 0 1 W	1 1 reg reg	2	2	u	0	0	x	x	x
	mem, reg	(mem) ← (mem) XOR reg	0 0 1 1 0 0 0 W	mod reg mem	16/24	2-4	u	0	0	x	x	x
	reg, mem	reg ← reg XOR (mem)	0 0 1 1 0 0 1 W	mod reg mem	11/15	2-4	u	0	0	x	x	x
	reg, imm	reg ← reg XOR imm	1 0 0 0 0 0 0 W	1 1 1 1 0 reg	4	3-4	u	0	0	x	x	x
	mem, imm	(mem) ← (mem) XOR imm	1 0 0 0 0 0 0 W	mod 1 1 0 mem	18/26	3-6	u	0	0	x	x	x
	acc, imm	When W = 0 AL ← AL XOR imm8 When W = 1 AW ← AW XOR imm16	0 0 1 1 0 1 0 W		4	2-3	u	0	0	x	x	x
Bit Operation Instructions			2nd byte*	3rd byte*								
TEST1	reg8, CL	reg8 bit no. CL = 0: Z ← 1 reg8 bit no. CL = 1: Z ← 0	0 0 0 1 0 0 0 0	1 1 0 0 0 reg	3	3	u	0	0	u	u	x
	mem8, CL	(mem8) bit no. CL = 0: Z ← 1 (mem8) bit no. CL = 1: Z ← 0	0 0 0 1 0 0 0 0	mod 0 0 0 mem	12	3-5	u	0	0	u	u	x
	reg16, CL	reg16 bit no. CL = 0: Z ← 1 reg16 bit no. CL = 1: Z ← 0	0 0 0 1 0 0 0 1	1 1 0 0 0 reg	3	3	u	0	0	u	u	x
	mem16, CL	(mem16) bit no. CL = 0: Z ← 1 (mem16) bit no. CL = 1: Z ← 0	0 0 0 1 0 0 0 1	mod 0 0 0 mem	12/16	3-5	u	0	0	u	u	x
	reg8, imm3	reg8 bit no. imm3 = 0: Z ← 1 reg8 bit no. imm3 = 1: Z ← 0	0 0 0 1 1 0 0 0	1 1 0 0 0 reg	4	4	u	0	0	u	u	x
	mem8, imm3	(mem8) bit no. imm3 = 0: Z ← 1 (mem8) bit no. imm3 = 1: Z ← 0	0 0 0 1 1 0 0 0	mod 0 0 0 mem	13	4-6	u	0	0	u	u	x
	reg16, imm4	reg16 bit no. imm4 = 0: Z ← 1 reg16 bit no. imm4 = 1: Z ← 0	0 0 0 1 1 0 0 1	1 1 0 0 0 reg	4	4	u	0	0	u	u	x
	mem16, imm4	(mem16) bit no. imm4 = 0: Z ← 1 (mem16) bit no. imm4 = 1: Z ← 0	0 0 0 1 1 0 0 1	mod 0 0 0 mem	13/17	4-6	u	0	0	u	u	x
			2nd byte*	3rd byte*								

*Note: First byte = 0FH

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Clocks	No. of Bytes	AC	CY	V	P	S	Z
Bit Operation Instructions (cont'd)												
			2nd byte*	3rd byte*								
NOT1	reg8, CL	reg8 bit no. CL ← $\overline{\text{reg8 bit no. CL}}$	0 0 0 1 0 1 1 0	1 1 0 0 0 reg	4	3						
	mem8, CL	(mem8) bit no. CL ← $\overline{\text{(mem8) bit no. CL}}$	0 0 0 1 0 1 1 0	mod 0 0 0 mem	10	3-5						
	reg16, CL	reg16 bit no. CL ← $\overline{\text{reg16 bit no. CL}}$	0 0 0 1 0 1 1 1	1 1 0 0 0 reg	4	3						
	mem16, CL	(mem16) bit no. CL ← $\overline{\text{(mem16) bit no. CL}}$	0 0 0 1 0 1 1 1	mod 0 0 0 mem	18/26	3-5						
	reg8, imm3	reg8 bit no. imm3 ← $\overline{\text{reg8 bit no. imm3}}$	0 0 0 1 1 1 1 0	1 1 0 0 0 reg	5	4						
	mem8, imm3	(mem8) bit no. imm3 ← $\overline{\text{(mem8) bit no. imm3}}$	0 0 0 1 1 1 1 0	mod 0 0 0 mem	11	4-6						
	reg16, imm4	reg16 bit no. imm4 ← $\overline{\text{reg16 bit no. imm4}}$	0 0 0 1 1 1 1 1	1 1 0 0 0 reg	5	4						
	mem16, imm4	(mem16) bit no. imm4 ← $\overline{\text{(mem16) bit no. imm4}}$	0 0 0 1 1 1 1 1	mod 0 0 0 mem	19/27	4-6						
			2nd byte*	3rd byte*								
			*Note: First byte = 0FH									
	CY	CY ← $\overline{\text{CY}}$	1 1 1 1 0 1 0 1		2	1		x				
			2nd byte*	3rd byte*								
CLR1	reg8, CL	reg8 bit no. CL ← 0	0 0 0 1 0 0 1 0	1 1 0 0 0 reg	5	3						
	mem8, CL	(mem8) bit no. CL ← 0	0 0 0 1 0 0 1 0	mod 0 0 0 mem	14	3-5						
	reg16, CL	reg16 bit no. CL ← 0	0 0 0 1 0 0 1 1	1 1 0 0 0 reg	5	3						
	mem16, CL	(mem16) bit no. CL ← 0	0 0 0 1 0 0 1 1	mod 0 0 0 mem	14/22	3-5						
	reg8, imm3	reg8 bit no. imm3 ← 0	0 0 0 1 1 0 1 0	1 1 0 0 0 reg	6	4						
	mem8, imm3	(mem8) bit no. imm3 ← 0	0 0 0 1 1 0 1 0	mod 0 0 0 mem	15	4-6						
	reg16, imm4	reg16 bit no. imm4 ← 0	0 0 0 1 1 0 1 1	1 1 0 0 0 reg	6	4						
	mem16, imm4	(mem16) bit no. imm4 ← 0	0 0 0 1 1 0 1 1	mod 0 0 0 mem	15/23	4-6						
			2nd byte*	3rd byte*								
			*Note: First byte = 0FH									
	CY	CY ← 0	1 1 1 1 1 0 0 0		2	1		0				
	DIR	DIR ← 0	1 1 1 1 1 1 0 0		2	1						

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0	No. of Clocks	No. of Bytes	AC	CY	V	P	S	Z
Bit Operation Instructions (cont)											
			2nd byte* / 3rd byte*								
SET1	reg8, CL	reg8 bit no. CL ← 1	0 0 0 1 0 1 0 0 1 1 0 0 0 reg	4	3						
	mem8, CL	(mem8) bit no. CL ← 1	0 0 0 1 0 1 0 0 mod 0 0 0 mem	13	3-5						
	reg16, CL	reg16 bit no. CL ← 1	0 0 0 1 0 1 0 1 1 1 0 0 0 reg	4	3						
	mem16, CL	(mem16) bit no. CL ← 1	0 0 0 1 0 1 0 1 mod 0 0 0 mem	13/21	3-5						
	reg8, imm3	reg8 bit no. imm3 ← 1	0 0 0 1 1 1 0 0 1 1 0 0 0 reg	5	4						
	mem8, imm3	(mem8) bit no. imm3 ← 1	0 0 0 1 1 1 0 0 mod 0 0 0 mem	14	4-6						
	reg16, imm4	reg16 bit no. imm4 ← 1	0 0 0 1 1 1 0 1 1 1 0 0 0 reg	5	4						
	mem16, imm4	(mem16) bit no. imm4 ← 1	0 0 0 1 1 1 0 1 mod 0 0 0 mem	14/22	4-6						
			2nd byte* / 3rd byte* *Note: First byte = 0FH								
	CY	CY ← 1	1 1 1 1 1 0 0 1	2	1		1				
	DIR	DIR ← 1	1 1 1 1 1 1 0 1	2	1						
Shift Instructions											
SHL	reg, 1	CY ← MSB of reg, reg ← reg × 2 When MSB of reg ≠ CY, V ← 1 When MSB of reg = CY, V ← 0	1 1 0 1 0 0 0 W 1 1 1 0 0 reg	2	2	u	x	x	x	x	x
	mem, 1	CY ← MSB of (mem), (mem) ← (mem) × 2 When MSB of (mem) ≠ CY, V ← 1 When MSB of (mem) = CY, V ← 0	1 1 0 1 0 0 0 W mod 1 0 0 mem	16/24	2-4	u	x	x	x	x	x
	reg, CL	temp ← CL, while temp ≠ 0, repeat this operation: CY ← MSB of reg, reg ← reg × 2, temp ← temp − 1	1 1 0 1 0 0 1 W 1 1 1 0 0 reg	7 + n	2	u	x	u	x	x	x
	mem, CL	temp ← CL, while temp ≠ 0, repeat this operation: CY ← MSB of (mem), (mem) ← (mem) × 2, temp ← temp − 1	1 1 0 1 0 0 1 W mod 1 0 0 mem	19/27 + n	2-4	u	x	u	x	x	x
	reg, imm8	temp ← imm8, while temp ≠ 0, repeat this operation: CY ← MSB of reg, reg ← reg × 2, temp ← temp − 1	1 1 0 0 0 0 0 W 1 1 1 0 0 reg	7 + n	3	u	x	u	x	x	x
	mem, imm8	temp ← imm8, while temp ≠ 0, repeat this operation: CY ← MSB of (mem), (mem) ← (mem) × 2, temp ← temp − 1	1 1 0 0 0 0 0 W mod 1 0 0 mem n: number of shifts	19/27 + n	3-5	u	x	u	x	x	x
SHR	reg, 1	CY ← LSB of reg, reg ← reg ÷ 2 When MSB of reg ≠ bit following MSB of reg: V ← 1 When MSB of reg = bit following MSB of reg: V ← 0	1 1 0 1 0 0 0 W 1 1 1 0 1 reg	2	2	u	x	x	x	x	x

SONY

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	Operation Code 7 6 5 4 3 2 1 0	No. of Clocks	No. of Bytes	AC	CY	V	P	S	Z
Shift Instructions (cont)												
SHR	mem, 1	CY ← LSB of (mem), (mem) ← (mem) ÷ 2 When MSB of (mem) ≠ bit following MSB of (mem): V ← 1 When MSB of (mem) = bit following MSB of (mem): V ← 0	1 1 0 1 0 0 0 W	mod 1 0 1 mem	16/24	2-4		x	x	x	x	x
	reg, CL	temp ← CL, while temp ≠ 0, repeat this operation: CY ← LSB of reg, reg ← reg ÷ 2, temp ← temp − 1	1 1 0 1 0 0 1 W	1 1 1 0 1 reg	7 + n	2	u	x	u	x	x	x
	mem, CL	temp ← CL, while temp ≠ 0, repeat this operation: CY ← LSB of (mem), (mem) ← (mem) ÷ 2, temp ← temp − 1	1 1 0 1 0 0 1 W	mod 1 0 1 mem	19/27 + n	2-4	u	x	u	x	x	x
	reg, imm8	temp ← imm8, while temp ≠ 0, repeat this operation: CY ← LSB of reg, reg ← reg ÷ 2, temp ← temp − 1	1 1 0 0 0 0 0 W	1 1 1 0 1 reg	7 + n	3	u	x	u	x	x	x
	mem, imm8	temp ← imm8, while temp ≠ 0, repeat this operation: CY ← LSB of (mem), (mem) ← (mem) ÷ 2, temp ← temp − 1	1 1 0 0 0 0 0 W	mod 1 0 1 mem n: number of shifts	19/27 + n	3-5	u	x	u	x	x	x
SHRA	reg, 1	CY ← LSB of reg, reg ← reg ÷ 2, V ← 0 MSB of operand does not change	1 1 0 1 0 0 0 W	1 1 1 1 1 reg	2	2	u	x	0	x	x	x
	mem, 1	CY ← LSB of (mem), (mem) ← (mem) ÷ 2 V ← 0, MSB of operand does not change	1 1 0 1 0 0 0 W	mod 1 1 1 mem	16/24	2-4	u	x	0	x	x	x
	reg, CL	temp ← CL, while temp ≠ 0, repeat this operation: CY ← LSB of reg, reg ← reg ÷ 2, temp ← temp − 1 MSB of operand does not change	1 1 0 1 0 0 1 W	1 1 1 1 1 reg	7 + n	2	u	x	u	x	x	x
	mem, CL	temp ← CL, while temp ≠ 0, repeat this operation: CY ← LSB of (mem), (mem) ← (mem) ÷ 2, temp ← temp − 1 MSB of operand does not change	1 1 0 1 0 0 1 W	mod 1 1 1 mem	19/27 + n	2-4	u	x	u	x	x	x
	reg, imm8	temp ← imm8, while temp ≠ 0, repeat this operation: CY ← LSB of reg, reg ← reg ÷ 2, temp ← temp − 1 MSB of operand does not change	1 1 0 0 0 0 0 W	1 1 1 1 1 reg	7 + n	3	u	x	u	x	x	x
	mem, imm8	temp ← imm8, while temp ≠ 0, repeat this operation: CY ← LSB of (mem), (mem) ← (mem) ÷ 2, temp ← temp − 1 MSB of operand does not change	1 1 0 0 0 0 0 W	mod 1 1 1 mem n: number of shifts	19/27 + n	3-5	u	x	u	x	x	x

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	No. of Clocks	No. of Bytes	AC	CY	V	P	S	Z
		Rotate Instructions										
ROL	reg, 1	CY ← MSB of reg, reg ← reg × 2 + CY MSB of reg ≠ CY: V ← 1 MSB of reg = CY: V ← 0	1 1 0 1 0 0 0 W	1 1 0 0 0 reg	2	2		x	x			
	mem, 1	CY ← MSB of (mem), (mem) ← (mem) × 2 + CY MSB of (mem) ≠ CY: V ← 1 MSB of (mem) = CY: V ← 0	1 1 0 1 0 0 0 W	mod 0 0 0 mem	16/24	2-4		x	x			
	reg, CL	temp ← CL, while temp ≠ 0, repeat this operation: CY ← MSB of reg, reg ← reg × 2 + CY temp ← temp − 1	1 1 0 1 0 0 1 W	1 1 0 0 0 reg	7 + n	2		x	u			
	mem, CL	temp ← CL, while temp ≠ 0, repeat this operation: CY ← MSB of (mem), (mem) ← (mem) × 2 + CY temp ← temp − 1	1 1 0 1 0 0 1 W	mod 0 0 0 mem	19/27 + n	2-4		x	u			
	reg, imm8	temp ← imm8, while temp ≠ 0, repeat this operation: CY ← MSB of reg, reg ← reg × 2 + CY temp ← temp − 1	1 1 0 0 0 0 0 W	1 1 0 0 0 reg	7 + n	3		x	u			
	mem, imm8	temp ← imm8, while temp ≠ 0, repeat this operation: CY ← MSB of (mem), (mem) ← (mem) × 2 + CY temp ← temp − 1	1 1 0 0 0 0 0 W n: number of shifts	mod 0 0 0 mem	19/27 + n	3-5		x	u			
ROR	reg, 1	CY ← LSB of reg, reg ← reg ÷ 2 MSB of reg ← CY MSB of reg ≠ bit following MSB of reg: V ← 1 MSB of reg = bit following MSB of reg: V ← 0	1 1 0 1 0 0 0 W	1 1 0 0 1 reg	2	2		x	x			
	mem, 1	CY ← LSB of (mem), (mem) ← (mem) ÷ 2 MSB of (mem) ← CY MSB of (mem) ≠ bit following MSB of (mem): V ← 1 MSB of (mem) = bit following MSB of (mem): V ← 0	1 1 0 1 0 0 0 W	mod 0 0 1 mem	16/24	2-4		x	x			
	reg, CL	temp ← CL, while temp ≠ 0, repeat this operation: CY ← LSB of reg, reg ← reg ÷ 2, MSB of reg ← CY temp ← temp − 1	1 1 0 1 0 0 1 W	1 1 0 0 1 reg	7 + n	2		x	u			
	mem, CL	temp ← CL, while temp ≠ 0, repeat this operation: CY ← LSB of (mem), (mem) ← (mem) ÷ 2, MSB of (mem) ← CY temp ← temp − 1	1 1 0 1 0 0 1 W n: number of shifts	mod 0 0 1 mem	19/27 + n	2-4		x	u			

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0	Operation Code 7 6 5 4 3 2 1 0	No. of Clocks	No. of Bytes	AC	CY	V	P	S	Z
Rotation Instructions (cont)												
ROR	reg, imm8	temp ← imm8, while temp ≠ 0, repeat this operation: CY ← LSB of reg, reg ← reg ÷ 2, MSB of reg ← CY, temp ← temp − 1	1 1 0 0 0 0 0 W	1 1 0 0 1 reg	7 + n	3		x	u			
	mem, imm8	temp ← imm8, while temp ≠ 0, repeat this operation: CY ← LSB of (mem), (mem) ← (mem) ÷ 2, temp ← temp − 1	1 1 0 0 0 0 0 W	mod 0 0 1 mem	19/27 + n	3-5		x	u			
			n: number of shifts									
Rotate Instruction												
ROLC	reg, 1	tmpcy ← CY, CY ← MSB of reg, reg ← reg × 2 + tmpcy, MSB of reg = CY: V ← 0, MSB of reg ≠ CY: V ← 1	1 1 0 1 0 0 0 W	1 1 0 1 0 reg	2	2		x	x			
	mem, 1	tmpcy ← CY, CY ← MSB of (mem), (mem) ← (mem) × 2 + tmpcy, MSB of (mem) = CY: V ← 0, MSB of (mem) ≠ CY: V ← 1	1 1 0 1 0 0 0 W	mod 0 1 0 mem	16/24	2-4		x	x			
	reg, CL	temp ← CL, while temp ≠ 0, repeat this operation: tmpcy ← CY, CY ← MSB of reg, reg ← reg × 2 + tmpcy, temp ← temp − 1	1 1 0 1 0 0 1 W	1 1 0 1 0 reg	7 + n	2		x	u			
	mem, CL	temp ← CL, while temp ≠ 0, repeat this operation: tmpcy ← CY, CY ← MSB of (mem), (mem) ← (mem) × 2 + tmpcy, temp ← temp − 1	1 1 0 1 0 0 1 W	mod 0 1 0 mem	19/27 + n	2-4		x	u			
	reg, imm8	temp ← imm8, while temp ≠ 0, repeat this operation: tmpcy ← CY, CY ← MSB of reg, reg ← reg × 2 + tmpcy, temp ← temp − 1	1 1 0 0 0 0 0 W	1 1 0 1 0 reg	7 + n	3		x	u			
	mem, imm8	temp ← imm8, while temp ≠ 0, repeat this operation: tmpcy ← CY, CY ← MSB of (mem), (mem) ← (mem) × 2 + tmpcy, temp ← temp − 1	1 1 0 0 0 0 0 W	mod 0 1 0 mem	19/27 + n	3-5		x	u			
			n: number of shifts									

Mnemonic	Operand	Operation	Operation Code (7 6 5 4 3 2 1 0 / 7 6 5 4 3 2 1 0)	No. of Clocks	No. of Bytes	AC	CY	V	P	S	Z
Rotate Instructions (cont)											
RORC	reg, 1	tmpcy ← CY, CY ← LSB of reg reg ← reg ÷ 2, MSB of reg ← tmpcy MSB of reg ≠ bit following MSB of reg: V ← 1 MSB of reg = bit following MSB of reg: V ← 0	1 1 0 1 0 0 0 W / 1 1 0 1 1 reg	2	2		x	x			
	mem, 1	tmpcy ← CY, CY ← LSB of (mem) (mem) ← (mem) ÷ 2, MSB of (mem) ← tmpcy MSB of (mem) ≠ bit following MSB of (mem): V ← 1 MSB of (mem) = bit following MSB of (mem): V ← 0	1 1 0 1 0 0 0 W / mod 0 1 1 mem	16/24	2-4		x	x			
	reg, CL	temp ← CL, while temp ≠ 0, repeat this operation: tmpcy ← CY, CY ← LSB of reg, reg ← reg ÷ 2, MSB of reg ← tmpcy, temp ← temp − 1	1 1 0 1 0 0 1 W / 1 1 0 1 1 reg	7 + n	2		x	u			
	mem, CL	temp ← CL, while temp ≠ 0, repeat this operation: tmpcy ← CY, CY ← LSB of (mem), (mem) ← (mem) ÷ 2 MSB of (mem) ← tmpcy, temp ← temp − 1	1 1 0 1 0 0 1 W / mod 0 1 1 mem	19/27	2-4		x	u			
	reg, imm8	temp ← imm8, while temp ≠ 0, repeat this operation: tmpcy ← CY, CY ← LSB of reg, reg ← reg ÷ 2 MSB of reg ← tmpcy, temp ← temp − 1	1 1 0 0 0 0 0 W / 1 1 0 1 1 reg	7 + n	3		x	u			
	mem, imm8	temp ← imm8, while temp ≠ 0, repeat this operation: tmpcy ← CY, CY ← LSB of (mem), (mem) ← (mem) ÷ 2 MSB of (mem) ← tmpcy, temp ← temp − 1	1 1 0 0 0 0 0 W / mod 0 1 1 mem n: number of shifts	19/27	3-5		x	u			
Subroutine Control Instructions											
CALL	near-proc	(SP − 1, SP − 2) ← PC, SP ← SP − 2 PC ← PC + disp	1 1 1 0 1 0 0 0	16/20	3						
	regptr16	(SP − 1, SP − 2) ← PC, SP ← SP − 2 PC ← regptr16	1 1 1 1 1 1 1 1 / 1 1 0 1 0 reg	14/18	2						
	memptr16	(SP − 1, SP − 2) ← PC, SP ← SP − 2 PC ← (memptr16)	1 1 1 1 1 1 1 1 / mod 0 1 0 mem	23/31	2-4						
	far-proc	(SP − 1, SP − 2) ← PS, (SP − 3, SP − 4) ← PC SP ← SP − 4, PS ← seg, PC ← offset	1 0 0 1 1 0 1 0	21/29	5						
	memptr32	(SP − 1, SP − 2) ← PS, (SP − 3, SP − 4) ← PC SP ← SP − 4, PS ← (memptr32 + 2), PC ← (memptr32)	1 1 1 1 1 1 1 1 / mod 0 1 1 mem	31/47	2-4						

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0	No. of Clocks	No. of Bytes	Flags AC CY V P S Z
		Subroutine Control Instructions (cont'd)				
RET		PC ← (SP + 1, SP), SP ← SP + 2	1 1 0 0 0 0 1 1	15/19	1	
	pop-value	PC ← (SP + 1, SP), SP ← SP + 2, SP ← SP + pop-value	1 1 0 0 0 0 1 0	20/24	3	
		PC ← (SP + 1, SP), PS ← (SP + 3, SP + 2), SP ← SP + 4	1 1 0 0 1 0 1 1	21/29	1	
	pop-value	PC ← (SP + 1, SP), PS ← (SP + 3, SP + 2), SP ← SP + 4, SP ← SP + pop-value	1 1 0 0 1 0 1 0	24/32	3	
		Stack Manipulation Instructions				
PUSH	mem16	(SP − 1, SP − 2) ← (mem16), SP ← SP − 2	1 1 1 1 1 1 1 1 mod 1 1 0 mem	18/26	2-4	
	reg16	(SP − 1, SP − 2) ← reg16, SP ← SP − 2	0 1 0 1 0 reg	8/12	1	
	sreg	(SP − 1, SP − 2) ← sreg, SP ← SP − 2	0 0 0 sreg 1 1 0	8/12	1	
	PSW	(SP − 1, SP − 2) ← PSW, SP ← SP − 2	1 0 0 1 1 1 0 0	8/12	1	
	R	Push registers on the stack	0 1 1 0 0 0 0 0	35/67	1	
	imm	(SP − 1, SP − 2) ← imm, SP ← SP − 2, When S = 1 sign extension	0 1 1 0 1 0 S 0	7/11 or 8/12	2-3	
POP	mem16	(mem16) ← (SP + 1, SP), SP ← SP + 2	1 0 0 0 1 1 1 1 mod 0 0 0 mem	17/25	2-4	
	reg16	reg16 ← (SP + 1, SP), SP ← SP + 2	0 1 0 1 1 reg	8/12	1	
	sreg	sreg ← (SP + 1, SP) sreg: SS, DS0, DS1, SP ← SP + 2	0 0 0 sreg 1 1 1	8/12	1	
	PSW	PSW ← (SP + 1, SP), SP ← SP + 2	1 0 0 1 1 1 0 1	8/12	1	R R R R R R
	R	Pop registers from the stack	0 1 1 0 0 0 0 1	43/75	1	
PREPARE	imm16, imm8	Prepare new stack frame	1 1 0 0 1 0 0 0 *: imm8 = 0: 12/16 imm8 ≥ 1: 23 + 20 (imm8 − 1): Odd Address 19 + 12 (imm8 − 1): Even Address	*	4	
DISPOSE		Dispose of stack frame	1 1 0 0 1 0 0 1	6/10	1	
		Branch Instruction				
BR	near-label	PC ← PC + disp	1 1 1 0 1 0 0 1	12	3	
	short-label	PC ← PC + ext-disp8	1 1 1 0 1 0 1 1	12	2	
	regptr16	PC ← regptr16	1 1 1 1 1 1 1 1 1 1 1 0 0 reg	11	2	
	memptr16	PC ← (memptr16)	1 1 1 1 1 1 1 1 mod 1 0 0 mem	20/24	2-4	
	far-label	PS ← seg, PC ← offset	1 1 1 0 1 0 1 0	15	5	
	memptr32	PS ← (memptr32 + 2), PC ← (memptr32)	1 1 1 1 1 1 1 1 mod 1 0 1 mem	27/35	2-4	

Mnemonic	Operand	Operation	7	6	5	4	3	2	1	0	7 6 5 4 3 2 1 0	No. of Clocks	No. of Bytes	AC	CY	V	P	S	Z
Conditional Branch Instructions																			
BV	short-label	if V = 1, PC ← PC + ext-disp8	0	1	1	1	0	0	0	0		14/4	2						
BNV	short-label	if V = 0, PC ← PC + ext-disp8	0	1	1	1	0	0	0	1		14/4	2						
BC, BL	short-label	if CY = 1, PC ← PC + ext-disp8	0	1	1	1	0	0	1	0		14/4	2						
BNC, BNL	short-label	if CY = 0, PC ← PC + ext-disp8	0	1	1	1	0	0	1	1		14/4	2						
BE, BZ	short-label	if Z = 1, PC ← PC + ext-disp8	0	1	1	1	0	1	0	0		14/4	2						
BNE, BNZ	short-label	if Z = 0, PC ← PC + ext-disp8	0	1	1	1	0	1	0	1		14/4	2						
BNH	short-label	if CY OR Z = 1, PC ← PC + ext-disp8	0	1	1	1	0	1	1	0		14/4	2						
BH	short-label	if CY OR Z = 0, PC ← PC + ext-disp8	0	1	1	1	0	1	1	1		14/4	2						
BN	short-label	if S = 1, PC ← PC + ext-disp8	0	1	1	1	1	0	0	0		14/4	2						
BP	short-label	if S = 0, PC ← PC + ext-disp8	0	1	1	1	1	0	0	1		14/4	2						
BPE	short-label	if P = 1, PC ← PC + ext-disp8	0	1	1	1	1	0	1	0		14/4	2						
BPO	short-label	if P = 0, PC ← PC + ext-disp8	0	1	1	1	1	0	1	1		14/4	2						
BLT	short-label	if S XOR V = 1, PC ← PC + ext-disp8	0	1	1	1	1	1	0	0		14/4	2						
BGE	short-label	if S XOR V = 0, PC ← PC + ext-disp8	0	1	1	1	1	1	0	1		14/4	2						
BLE	short-label	if (S XOR V) OR Z = 1, PC ← PC + ext-disp8	0	1	1	1	1	1	1	0		14/4	2						
BGT	short-label	if (S XOR V) OR Z = 0, PC ← PC + ext-disp8	0	1	1	1	1	1	1	1		14/4	2						
DBNZNE	short-label	CW ← CW − 1 if Z = 0 and CW ≠ 0, PC ← PC + ext-disp8	1	1	1	0	0	0	0	0		14/5	2						
DBNZE	short-label	CW ← CW − 1 if Z = 1 and CW ≠ 0, PC ← PC + ext-disp8	1	1	1	0	0	0	0	1		14/5	2						
DBNZ	short-label	CW ← CW − 1 if CW ≠ 0, PC ← PC + ext-disp8	1	1	1	0	0	0	1	0		13/5	2						
BCWZ	short-label	if CW = 0, PC ← PC + ext-disp8	1	1	1	0	0	0	1	1		13/5	2						
Interrupt Instructions																			
BRK	3	(SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS, (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0 PS ← (15, 14), PC ← (13, 12)	1	1	0	0	1	1	0	0		38/50	1						
	imm8 (≠ 3)	(SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS, (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0 PC ← (n × 4 + 1, n × 4) PS ← (n × 4 + 3, n × 4 + 2) n = imm8	1	1	0	0	1	1	0	1		38/50	2						

SONY

Mnemonic	Operand	Operation	Operation Code 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0	No. of Clocks	No. of Bytes	AC	CY	V	P	S	Z
Interrupt Instructions (cont)											
BRKV		When V = 1 (SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0 PS ← (19, 18), PC ← (17, 16)	1 1 0 0 1 1 1 0	40/52 3	1						
RETI		PC ← (SP + 1, SP), PS ← (SP + 3, SP + 2) PSW ← (SP + 5, SP + 4), SP ← SP + 6	1 1 0 0 1 1 1 1	27/39	1	R	R	R	R	R	R
CHKIND	reg16, mem32	When (mem32) > reg16 or (mem32 + 2) < reg16 (SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS, (SP − 5, SP − 6) ← PC, SP ← SP − 6 IE ← 0, BRK ← 0, PS ← (23, 22), PC ← (21, 20)	0 1 1 0 0 0 1 0 mod reg mem	53-56 (73-76) 18/26	2-4						
BRKEM	imm8	(SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS, (SP − 5, SP − 6) ← PC, SP ← SP − 6 MD ← 0, PC ← (n × 4 + 1, n × 4) PS ← (n × 4 + 3, n × 4 + 2), n = imm8	0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1	38/50	3						
CPU Control Instructions											
HALT		CPU Halt	1 1 1 1 0 1 0 0	2	1						
BUSLOCK		Bus Lock Prefix	1 1 1 1 0 0 0 0	2	1						
FPO1	fp-op	No Operation	1 1 0 1 1 X X X 1 1 Y Y Y Z Z Z	2	2						
	fp-op, mem	data bus ← (mem)	1 1 0 1 1 X X X mod Y Y Y mem	11/15	2-4						
FPO2	fp-op	No Operation	0 1 1 0 0 1 1 X 1 1 Y Y Y Z Z Z	2	2						
	fp-op, mem	data bus ← (mem)	0 1 1 0 0 1 1 X mod Y Y Y mem	11/15	2-4						
POLL		Poll and wait	1 0 0 1 1 0 1 1 n: number of times POLL pin is sampled	2 + 5n	1						
NOP		No Operation	1 0 0 1 0 0 0 0	3	1						
DI		IE ← 0	1 1 1 1 1 0 1 0	2	1						
EI		IE ← 1	1 1 1 1 1 0 1 1	2	1						
8080 Mode Instructions											
RETEM		PC ← (SP + 1, SP), PS ← (SP + 3, SP + 2), PSW ← (SP + 5, SP + 4), SP ← SP + 6	1 1 1 0 1 1 0 1 1 1 1 1 1 1 0 1	27/39	2	R	R	R	R	R	R
CALLN	imm8	(SP − 1, SP − 2) ← PSW, (SP − 3, SP − 4) ← PS, (SP − 5, SP − 6) ← PC, SP ← SP − 6 MD ← 1, PC ← (n × 4 + 1, n × 4) PS ← (n × 4 + 3, n × 4 + 2), n = imm8	1 1 1 0 1 1 0 1 1 1 1 0 1 1 0 1	38/58	3						

Package Outline

Unit: mm