

# LH70116 (V30) High-Performance 16-Bit Microprocessor

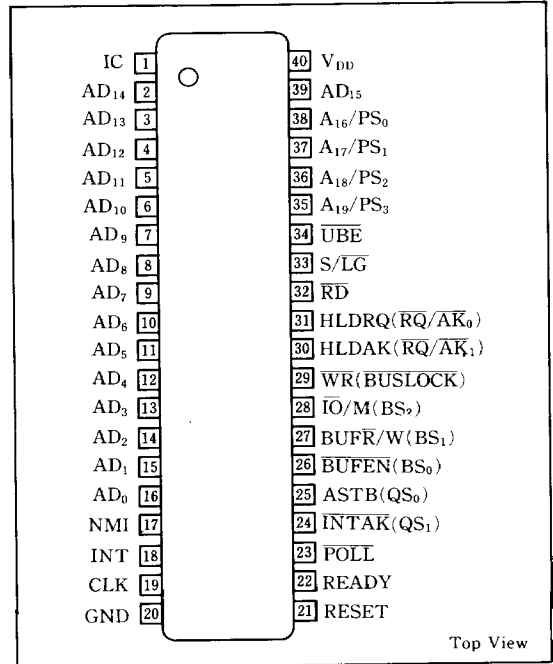
## ■ Description

The LH70116 (V30) is a CMOS 16-bit microprocessor with internal 16-bit architecture and a 16-bit external data bus. The LH70116 additionally has a powerful instruction set including bit processing, packed BCD operations, and high-speed multiplication/division operations. The LH70116 can also execute the entire 8080 instruction set and comes with a standby mode that significantly reduces power consumption. It is software-compatible with the LH70108 16-bit microprocessor.

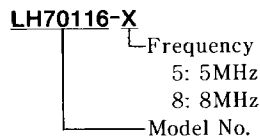
## ■ Features

1. Minimum instruction execution time: 250ns (at 8MHz)
2. Maximum addressable memory: 1 Mbyte
3. Abundant memory addressing modes
4. 14×16-bit register set
5. 101 instructions
6. Bit, byte, word, and block operations
7. Bit field operation instructions
8. Packed BCD instructions
9. Multiplication/division instruction execution time: 2.4 μs to 7.1 μs (at 8MHz), 3.8 μs to 11.4 μs (at 5MHz)
10. High-speed block transfer instructions: 2 Mbyte /s (at 8MHz)
11. High-speed calculation of effective addresses: 2 clock cycles in any addressing mode
12. Maskable (INT) and nonmaskable (NMI) interrupt inputs
13. IEEE-796 bus compatible interface
14. 8080 emulation mode
15. CMOS technology
16. Low-power consumption
17. Low-power standby mode
18. Single power supply
19. 5MHz or 8MHz clock
20. 40-pin DIP (DIP40-P-600)

## ■ Pin Connections

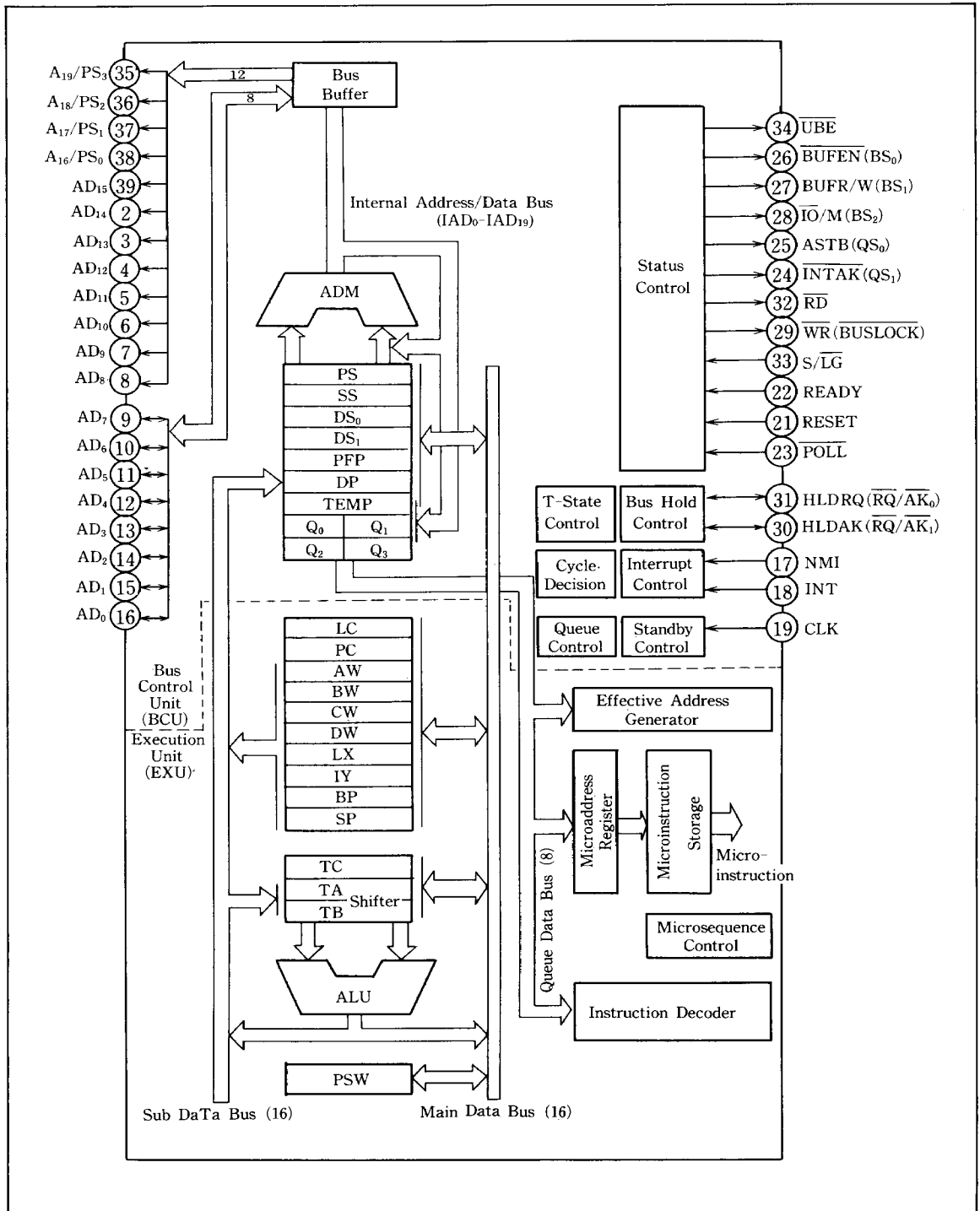


## ■ Ordering Information



\* V30 is a trademark of NEC corporation.

■ Block Diagram



## ■ Pin Identification

No.	Symbol	Direction	Function
1	IC*		Internally connected
2-8	AD <sub>14</sub> -AD <sub>8</sub>	Out	Address bus, middle bits
9-16	AD <sub>7</sub> -AD <sub>0</sub>	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 potential
21	RESET	In	Reset input
22	READY	In	Ready input
23	POLL	In	Poll input
24	INTAK (QS <sub>1</sub> )	Out	Interrupt acknowledge output (queue status bit 1 output)
25	ASTB (QS <sub>0</sub> )	Out	Address strobe output (queue status bit 0 output)
26	BUFEN (BS <sub>0</sub> )	Out	Buffer enable output (bus status bit 0 output)
27	BUFR/W (BS <sub>1</sub> )	Out	Buffer read/write output (bus status bit 1 output)

No.	Symbol	Direction	Function
28	IO/M (BS <sub>2</sub> )	Out	Access is I/O or memory (bus status bit 2 output)
29	WR (BUSLOCK)	Out	Write strobe output (bus lock output)
30	HLDK (RQ/AK <sub>1</sub> )	Out (In/Out)	Hold acknowledge output, (bus hold request input/acknowledge output 1)
31	HLDRQ (RQ/AK <sub>0</sub> )	In (In/Out)	Hold request input (bus hold request input/acknowledge output 0)
32	RD	Out	Read strobe output
33	S/LG	In	Small-scale/large-scale system input
34	UBE	Out	Latched bus status output 0 (always high in large-scale systems)
35-38	A <sub>19</sub> /PS <sub>3</sub> - A <sub>16</sub> /PS <sub>0</sub>	Out	Address bus, high bits or processor status output
39	AD <sub>15</sub>	Out	Address bus, bit 15
40	V <sub>DD</sub>		Power supply

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

**Absolute Maximum Ratings**

(Ta = +25°C)

Parameter	Symbol	Ratings	Units
Supply voltage	V <sub>DD</sub>	-0.5 to +7.0	V
Power dissipation	PD <sub>MAX</sub>	0.5	W
Input voltage	V <sub>I</sub>	-0.5 to V <sub>DD</sub> +0.3	V
CLK input voltage	V <sub>K</sub>	-0.5 to V <sub>DD</sub> +1.0	V
Output voltage	V <sub>O</sub>	-0.5 to V <sub>DD</sub> +0.3	V
Operating temperature	T <sub>opr</sub>	-40 to +85	°C
Storage temperature	T <sub>stg</sub>	-65 to +150	°C

**Capacitance**

(Ta = +25°C, V<sub>DD</sub> = 0V)

Parameter	Symbol	Conditions	MIN.	MAX.	Unit
Input capacitance	C <sub>I</sub>	fc = 1MHz Unmeasured		15	pF
I/O capacitance	C <sub>IO</sub>	pins returned to 0V		15	pF

**DC Characteristics**

(LH70116-5, Ta = -40°C to +85°C, V<sub>DD</sub> = +5V ± 10%)

(LH70116-8, Ta = -10°C to +70°C, V<sub>DD</sub> = +5V ± 5%)

Parameter	Symbol	Conditions	MIN.	TYP.	MAX.	Unit
Input HIGH voltage	V <sub>IH</sub>		2.2		V <sub>DD</sub> +0.3	V
Input LOW voltage	V <sub>IL</sub>		-0.5		0.8	V
CLK input HIGH voltage	V <sub>KH</sub>		3.9		V <sub>DD</sub> +1.0	V
CLK input LOW voltage	V <sub>KL</sub>		-0.5		0.6	V
Output HIGH voltage	V <sub>OH</sub>	I <sub>OH</sub> = -400 μA	0.7V <sub>DD</sub>			V
Output LOW voltage	V <sub>OL</sub>	I <sub>OL</sub> = 2.5mA			0.4	V
Input leakage HIGH current	I <sub>LIH</sub>	V <sub>I</sub> = V <sub>DD</sub>			10	μA
Input leakage LOW current	I <sub>LIL</sub>	V <sub>I</sub> = 0V			-10	μA
Output leakage HIGH current	I <sub>LOH</sub>	V <sub>O</sub> = V <sub>DD</sub>			10	μA
Output leakage LOW current	I <sub>LOL</sub>	V <sub>O</sub> = 0V			-10	μA
Supply current	I <sub>DD</sub>	Normal operation	70108-5	30	60	mA
		Standby mode		5	10	mA
		Normal operation	70108-8	45	80	mA
		Standby mode		8MHz	6	12



■ AC Characteristics

(LH70116-5,  $T_a = -40^{\circ}\text{C}$  to  $+85^{\circ}\text{C}$ ,  $V_{DD} = +5\text{V} \pm 10\%$ )  
 (LH70116-8,  $T_a = -10^{\circ}\text{C}$  to  $+70^{\circ}\text{C}$ ,  $V_{DD} = +5\text{V} \pm 5\%$ )

Parameter	Symbol	Conditions	LH70116-5		LH70116-8		Unit	
			MIN.	MAX.	MIN.	MAX.		
Small/Larg Scale								
Clock cycle	$t_{CYK}$		200	500	125	500	ns	
Clock pulse HIGH width	$t_{KKH}$	$V_{KH} = 3.0\text{V}$	69		44		ns	
Clock pulse LOW width	$t_{KKL}$	$V_{KL} = 1.5\text{V}$	90		60		ns	
Clock rise time	$t_{KR}$	1.5V to 3.0V		10		10	ns	
Clock fall time	$t_{KF}$	3.0V to 1.5V		10		10	ns	
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	
Input rise time (except CLK)	$t_{IR}$	0.8V to 2.2V		20		20	ns	
Input fall time (except CLK)	$t_{IF}$	2.2V to 0.8V		12		12	ns	
Output rise time	$t_{OR}$	0.8V to 2.2V		20		20	ns	
Output fall time	$t_{OF}$	2.2V to 0.8V		12		12	ns	
Small Scale								
Address delay time from CLK ↓	$t_{DKA}$	$C_L = 100\text{pF}$	10	90	10	60	ns	
Address hold time 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 HIGH width	$t_{STST}$		$t_{KKL} - 20$		$t_{KKL} - 10$		ns	
Address hold time from ASTB ↓	$t_{HSTA}$		$t_{KKH} - 10$		$t_{KKH} - 10$		ns	
Control delay time from CLK	$t_{DKCT}$			10	110	10	65	ns
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 LOW width	$t_{RR}$		$2t_{CYK} - 75$		$2t_{CYK} - 50$		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	
WR LOW width	$t_{WW}$	$2t_{CYK} - 60$		$2t_{CYK} - 40$		ns		
HLDRQ setup time to CLK ↑	$t_{SHQK}$		35		20	ns		
HLDAK delay time from CLK ↓	$t_{DKHA}$		10	160	10	100	ns	

■ AC Characteristics (Cont)

(LH70116-5,  $T_a = -40^\circ\text{C}$  to  $+85^\circ\text{C}$ ,  $V_{DD} = +5V \pm 10\%$ )

(LH70116-8,  $T_a = -10^\circ\text{C}$  to  $+70^\circ\text{C}$ ,  $V_{DD} = +5V \pm 5\%$ )

Parameter	Symbol	Conditions	LH70116-5		LH70116-8		Unit
			MIN.	MAX.	MIN.	MAX.	
Large Scale							
Address delay time from CLK	$t_{DKA}$	$C_L = 100\text{pF}$	10	90	10	60	ns
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 ↓	$t_{FKA}$		$t_{HKA}$	80	$t_{HKA}$	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 LOW width	$t_{RR}$		$2t_{CYK} - 75$		$2t_{CYK} - 50$		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
RQ setup time to CLK ↑	$t_{SRQK}$		20		10		ns
RQ hold time after CLK ↓	$t_{HRRQ1}$		0		0		ns
RQ hold time after CLK ↑	$t_{HRRQ2}$	40		30		ns	



## ■ Pin Functions

Some pins of the LH70116 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<sub>15</sub>-AD<sub>0</sub> (Address/Data Bus)

For small and large scale systems

The AD<sub>15</sub>-AD<sub>0</sub> is a time-multiplexed Address/Data bus. This performs output of lower 16 bits of 20 bits address information and input/output of byte or word data. The LH70116 locates memory and I/O operand to a byte-data bank to be accessed with even address (AD<sub>0</sub>=0) and a byte-data bank to be accessed with odd address (AD<sub>0</sub>=1). The LSB (AD<sub>0</sub>) has no meaning as a word data address but used for selecting the odd or even address bank. The  $\overline{UBE}$  (Upper Byte Enable) signal is provided to access byte/word data besides AD<sub>0</sub>. This is used as the combination of the following table.

Operand	$\overline{UBE}$	AD <sub>0</sub>	No. of bus cycle
Word of Even Address	0	0	1
Word of Odd Address	0	1 *	2
	1	0 **	
Byte of Even Address	1	0	1
Byte of Odd Address	0	1	1

\*First time, \*\*Second time

A word operand in odd address is performed via two continuous access of odd-byte bank and even-byte bank. In this case, first AD<sub>0</sub>=1 showing odd bank is output, and second AD<sub>0</sub>=1 showing continuous even bank is output automatically. These outputs are held to high or low level in the standby mode. These are 3-state output and becomes high impedance in the hold acknowledge and interrupt acknowledge states.

### 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 must be held high for five clocks to guarantee recognition. 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 LH70116 to exit the standby mode.

### INT (Maskable Interrupt)

For small- and large-scale systems.

This pin is an interrupt request that can be masked by software.

INT is active high level and is sensed during the last clock of the instruction. The interrupt will be accepted 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. INT must be asserted until the interrupt acknowledge is returned.

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 LH70116 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 an active high level. Input of this signal has priority over all other operations. After the reset signal input returns to a low level, the CPU begins execution of the program starting at address FFFF0H.

In addition to causing normal CPU start, RESET input will cause the LH70116 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 (T<sub>w</sub>) by setting this signal to inactive (low level) and requesting a read/write cycle delay.

If the READY signal is active (high level) during either the T<sub>3</sub> or T<sub>w</sub> state, the CPU will not generate a wait state.

This signal must be input in synchronization with external clock signals to satisfy the setup/hold time for normal operation.

### $\overline{POLL}$ (Poll)

For small- and large-scale systems.

The CPU checks this input upon execution of the  $\overline{POLL}$  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  $\overline{\text{READY}}$  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  $\overline{\text{IO/M}}$  signal is used to select between I/O and memory.

The three-state output is held high during standby mode and enters the high-impedance state during hold acknowledge.

### **S/LG (Small/Large)**

For small- and large-scale systems.

This signal determines the operation mode of the CPU. This signal is fixed at either a high or low level. When this signal is a high level, the CPU will operate in small-scale system mode, and when low, in the large-scale system mode. Pins 24 to 31 and 34 function differently depending on the operating mode of the CPU. Separate nomenclature is adopted for these signals in the two operating modes.

Pin No.	Function	
	S/LG-high	S/LG-low
24	$\overline{\text{INTAK}}$	$\text{QS}_1$
25	$\overline{\text{ASTB}}$	$\text{QS}_0$
26	$\overline{\text{BUFEN}}$	$\text{BS}_0$
27	$\overline{\text{BUFR/W}}$	$\text{BS}_1$
28	$\overline{\text{IO/M}}$	$\text{BS}_2$
29	$\overline{\text{WR}}$	$\overline{\text{BUSLOCK}}$
30	$\overline{\text{HLDK}}$	$\overline{\text{RQ/AK}}_1$
31	$\overline{\text{HLDRQ}}$	$\overline{\text{RQ/AK}}_0$

### **$\overline{\text{INTAK}}$ (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 ( $\text{AD}_7\text{-AD}_0$ ).

### **$\overline{\text{ASTB}}$ (Address Strobe)**

For small-scale systems.

The CPU outputs this strobe signal to latch address information at an external latch.

$\overline{\text{ASTB}}$  is held at a low level after held at a high level for a half clock cycle during standby mode.

### **$\overline{\text{BUFEN}}$ (Buffer Enable)**

For small-scale systems

This 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.

This three-state output is held high during standby mode and enters the high-impedance state during hold acknowledge.

### **$\overline{\text{BUFR/W}}$ (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.

$\overline{\text{BUFR/W}}$  is a three-state output and becomes high impedance during hold acknowledge.

### **$\overline{\text{IO/M}}$ (IO/Memory)**

For small-scale systems.

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

$\overline{\text{IO/M}}$ 's output is three state and becomes high impedance during hold acknowledge.

### **$\overline{\text{WR}}$ (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  $\overline{\text{IO/M}}$  signal.

This three-state output is held high during standby mode and enters the high-impedance state during hold acknowledge.

### **$\overline{\text{HLDK}}$ (Hold Acknowledge)**

For small-scale systems.

The  $\overline{\text{HLDK}}$  signal is used to indicate that the CPU accepts the hold request signal ( $\overline{\text{HLDRQ}}$ ). When this signal is a high level, the address bus, address/data bus, and the control lines become high impedance.

### **$\overline{\text{HLDRQ}}$ (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.

This signal must be input in synchronization with external clock signals to satisfy the setup/hold time for normal operation.

**UBE (Upper Byte Enable)**

For small- and large-scale systems.

This output indicates the use of the upper 8 bits ( $AD_{15}$ - $AD_8$ ) of the Address/Data bus during T2-T4 of bus cycle. This signal is active low and output during T1-T4 of the bus cycle. Bus cycles in which the UBE is active are shown in the following table.

Operand	UBE	$AD_0$	No. of bus cycle
Word of Even Address	0	0	1
Word of Odd Address	0	1*	2
	1	0**	
Byte of Even Address	1	0	1
Byte of Odd Address	0	1	1

\*First time, \*\*Second time

The UBE signal goes low level continuously during interrupt acknowledge state (because of necessity of word access of even address for vector read).

This signal is held to high level in the standby mode. The UBE is 3-state output and becomes high impedance during hold acknowledge.

**A<sub>19</sub>/PS<sub>3</sub>-A<sub>16</sub>/PS<sub>0</sub> (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<sub>3</sub> is always 0 in the native mode and 1 in 8080 emulation mode. The interrupt enable flag (IE) is output to PS<sub>2</sub>. Pins PS<sub>1</sub> and PS<sub>0</sub> indicate which memory segment is being accessed.

A <sub>17</sub> /PS <sub>1</sub>	A <sub>16</sub> /PS <sub>0</sub>	Segment
0	0	Data segment 1
0	1	Stack segment
1	0	Program segment
1	1	Data segment 0

The output of these pins is three state and becomes high impedance during hold acknowledge.

**QS<sub>1</sub>, QS<sub>0</sub> (Queue Status)**

For large-scale systems.

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

QS <sub>1</sub>	QS <sub>0</sub>	Instruction Queue Status
0	0	NOP (Queue does not change)
0	1	First byte of instruction
1	0	Queue empty
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.

**BS<sub>2</sub>-BS<sub>0</sub> (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 <sub>2</sub>	BS <sub>1</sub>	BS <sub>0</sub>	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

The output of these signals is three state and becomes high impedance during hold acknowledge.

These signals will be high from the rising edge of clock immediately after RESET signal is active to the next clock rise.

**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, or during an interrupt acknowledge cycle. 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.

**SHARP**

$\overline{\text{BUSLOCK}}$  is high during standby mode except if the HALT instruction has a  $\overline{\text{BUSLOCK}}$  prefix.

### $\overline{\text{RQ}}/\overline{\text{AK}}_1$ , $\overline{\text{RQ}}/\overline{\text{AK}}_0$ (Hold Request/Acknowledge)

For large-scale systems.

These pins function as bus hold request inputs ( $\overline{\text{RQ}}$ ) and as bus hold acknowledge outputs ( $\overline{\text{AK}}$ ).  $\overline{\text{RQ}}/\overline{\text{AK}}_0$  has a higher priority than  $\overline{\text{RQ}}/\overline{\text{AK}}_1$ .

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

Bus Hold Request Input (RQ) must be input in synchronization with external clock signal to satisfy the setup/hold time for normal operation.

### $V_{DD}$ (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)

The LH70116 is used with this pin at ground potential.

## ■ 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 increments 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 (PEP).

### 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 byte 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 (PC, SS, DS<sub>0</sub>, and DS<sub>1</sub>)

The memory addresses accessed by the LH70116 are divided into 64K-byte logical segments. The starting (base) address of each segment is specified by a 16-bit 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)	PS, effective address
DS <sub>0</sub> (Data Segment 0)	IX, effective address
DS <sub>1</sub> (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, data conversion
- 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.

<b>Status Flags</b>	<b>Control Flags</b>
• V (Overflow)	• MD (Mode)
• S (Sign)	• DIR (Direction)
• Z (Zero)	• IE (Interrupt Enable)
• AC (Auxiliary Carry)	• BRK (Break)
• P (Parity)	
• CY (Carry)	

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	I	C
					D	I	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.

**Q<sub>0</sub>-Q<sub>5</sub> (Prefetch Queue)**

The LH70116 has 6 byte instruction queue (FIFO), and it can store up to 6 instruction byte prefetched by the BCU. The instruction codes stored in the queue are fetched and executed by the EXU. The queue is cleared and prefetched with branch, call, return, or break instruction has been executed and when external interrupt has been acknowledged. Normally, the LH70108 prefetches if the queue has one word (two bytes) or more space. If the time required to prefetch the instruction code from the external memory is less than the mean execution time of instructions which are executed sequentially, then the actual instructions cycle will be shortened by this amount of time i. e. the instruction code to be next executed by the EXU can be available in the queue immediately after the completion of one instruction. As the result, processing speed is highly upgraded compared with the conventional CPU which fetch and execute instructions one by one. Queuing effect is lowered if there were many instructions which clears queue like the branch instruction or in the case of continuous instructions with too short instruction time.

**DP (Data Pointer)**

The data pointer is a 16-bit register indicates read/write addresses of variables. Effective address made in the effective address generator and the register contents including memory address offsets are transferred to the DP.

**TEMP (Temporary Communication Register)**

This is a 16-bit temporary register used by communications between external data bus and the EXU. The TEMP can be read or written by upper byte or lower byte independently for byte access. Basically, the EXU completes write operation with transferring data to the TEMP and completes read operation with recognizing the data has been transferred to the TEMP from external data bus.

**EAG (Effective Address Generator)**

The Effective Address Generator (EAG) performs high-speed effective address calculation necessary for memory access. This completes all the calculations with 2 clocks for every addressing mode.

This fetches the instruction byte (2nd or 3rd byte) which has operand specifying field, if the instruction needs memory access. Then calculates effective address and transfers it to the DP (Data Pointer) and generates control signals relating to handling ALU and corresponding registers. In addition, if it is necessary, the EAG requests to the BCU for starting the bus cycle (memory read).

**Instruction Decoder**

The Instruction Decoder classifies 1st byte of an instruction code into some groups with specific function and holds them during micro-instruction execution.

**Microaddress Register**

The microaddress register specifies the address of a microinstruction ROM to be next executed. At starting of a microinstruction execution, the 1st byte of instruction bytes stored in the queue is fetched in this register and it specifies a start address of the corresponding microinstruction sequence.

**Microinstruction ROM**

The Microinstruction ROM has 1024 words by 29 bits of microinstructions.

**Microinstruction Sequencer**

The Microinstruction Sequencer controls the microaddress register operation, microinstruction

ROM output, and synchronizing the EXU with BCU.

#### **ADM (Address Modifier)**

Address Modifier performs the generation of physical address (adding segment register and PFP or DP) and increment of PFP (Prefetch Pointer).

#### **TA/TB (Temporary Register/Shifter A, B)**

The TA/TB are 16-bit temporary register/shifter used with execution of multiply/divided and shift/rotate (including BCD rotate) instructions. When executing multiply or divide instruction TA + TB operates as a 32-bit temporary register/shifter when executing shift/rotate instructions. Both the TA and TB can be read or written to and from the internal bus by upper byte or lower byte independently. The contents of the TA and TB are input to the ALU.

#### **TC (Temporary Register C)**

The TC is a 16-bit temporary register used with

internal processing like the multiply or divide operation, etc. The TC content is output to the ALU.

#### **ALU (Arithmetic & Logic Unit)**

The Arithmetic and Logic Unit consists of a full adder and logical operation circuit and performs these operations:

- 1) Arithmetic operation (Add, Subtract, Multiply, Divide, increment, decrement, and complement)
- 2) Logical operation (test, AND, OR, XOR and bit test, set, clear, and complement)

#### **LC (Loop Counter)**

The Loop Counter (LC) is a 16-bit register which counts below items.

- 1) Loop number of the primitive block transfer and input/output instructions (MOVBK, OUTM, etc.) controlled with repeat prefix instructions (REP, REPC, etc.).
- 2) Shift number of the multi-bit shift/rotate instructions.



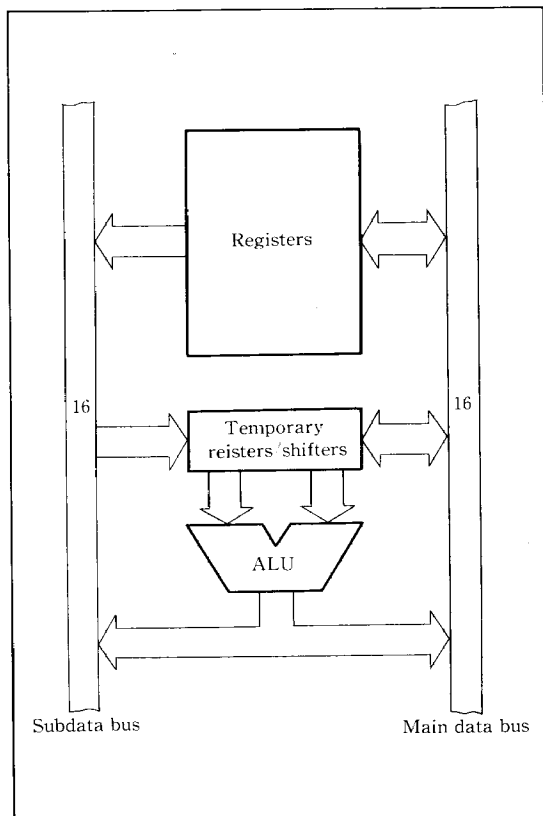
**High-Speed Execution of Instructions**

This section highlights the major architectural features that enhance the performance of the LH70116.

- 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 LH70116 (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**

**Example**

ADD AW, BW;  $AW \leftarrow AW + BW$

**Single Bus**

Step 1  $TA \leftarrow AW$

Step 2  $TB \leftarrow BW$

Step 3  $AW \leftarrow TA + TB$

**Dual Bus**

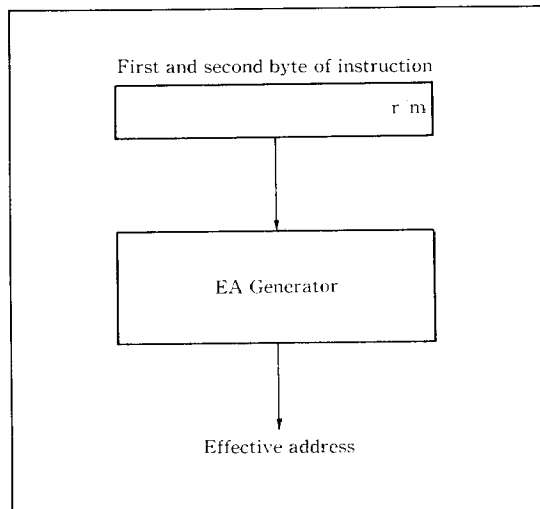
$TA \leftarrow AW, TB \leftarrow BW$

$AW \leftarrow TA + TB$

**Effective Address Generator**

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

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



**Fig. 2 Effective Address Generator**

**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

<b>Microprogram method</b>	<b>LC method</b>
8 + (4 × 5) = 28 clocks	7 + 5 = 12 clocks

**Program Counter and Prefetch Pointer (PC and PFP)**

The LH70116 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.

**Unique Instructions**

**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.

**(1) INS reg8, reg8/INS reg8, imm4**

This instruction (figure 3) 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 (DS<sub>1</sub> register) pulse 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.

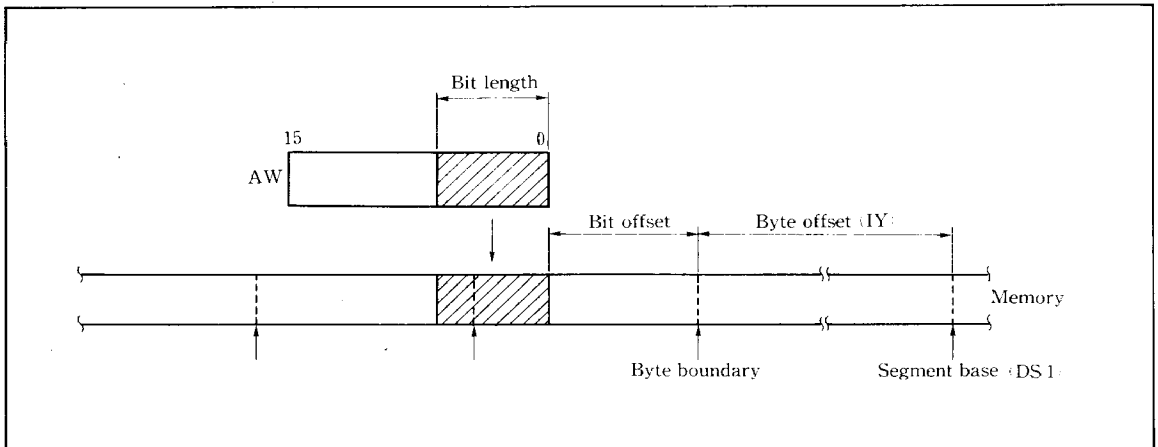


Fig. 3 Bit Field Insertion



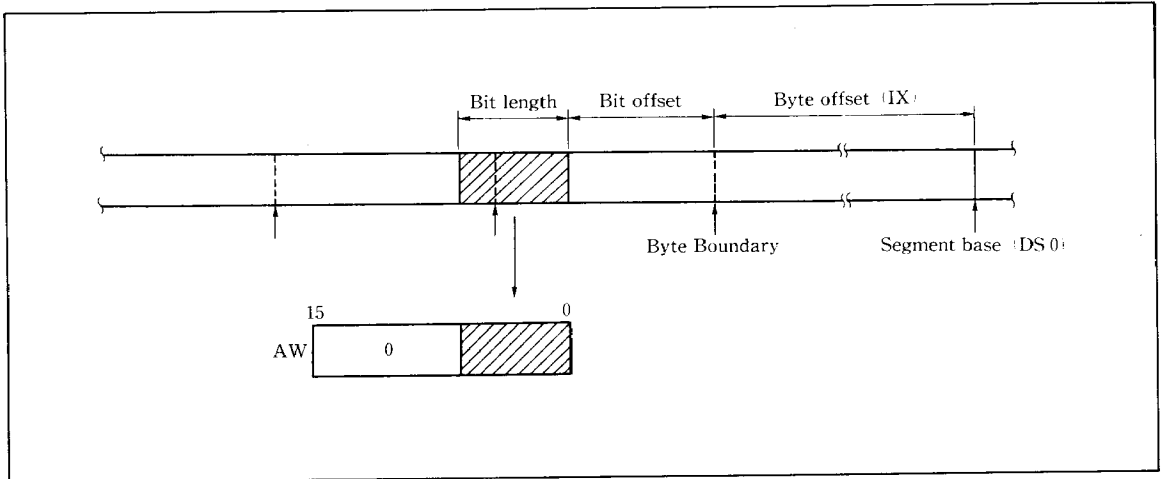


Fig. 4 Bit Field Extraction

**(2) 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 DS<sub>0</sub> segment register (segment base), the IX index register (byte offset), and the lower 4 bits of the first operand (bit offset).

**Packed BCD Operation Instructions**

The instructions described here process packed BCD data either as strings (ADD4S, SUB4S, CMP4S) or byte-format operands (ROR4, ROL4). 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 transferrable 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.

**(1) 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 overflow flag (V), the carry flag (CY), and zero flag (Z).

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

**(2) SUB4S**

This instruction subtracts the packed BCD string addressed by the IX index register from the packed BCD string addressed by the IY 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 overflow flag (V), the carry flag (CY), and zero flag (Z).

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

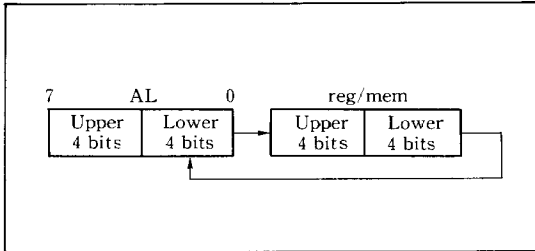
**(3) CMP4S**

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

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

**(4) ROL4**

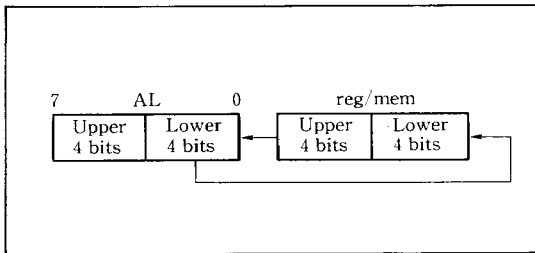
This instruction (Fig. 5) treats the byte data of the register or memory directly specified by the instruction byte as BCD data and uses the lower 4-bits of the AL register (AL<sub>L</sub>) to rotate that data one BCD digit to the left.



**Fig. 5 BCD Rotate Left (ROL4)**

**(5) ROR4**

This instruction (Fig. 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<sub>L</sub>) to rotate the data one BCD digit to the right



**Fig. 6 BCD Rotate Right (ROR4)**

**Stack Operation Instruction**

**(1) PREPARE imm16, imm8**

This instruction is used to generate "stack frames" required by the block structures of high-level languages such as Pascal and Ada. The stack frame includes a local variable area as well as pointers. These frame pointers point to the frame containing the variables that can be referenced from the current procedure.

The program example based upon Pascal language is shown below.

```

program EXAMPLE ;
  procedure P ;
    var a, b, c ;
  procedure Q ;
    var d, e ;

```

```

procedure R ;
  var f, g ;
  begin
    d := a+f+g ;
  end ;
begin
  R ;
  b := d+e ;
end ;
begin
  a := b+c ;
  Q ;
end ;
(*main program*)
begin
  P ;
end.

```

Note: The variables are defined as the words.

This program is an example of a procedure block with a triple nesting.

Procedure	Variables
P	a, b, c
Q	d, e
R	f, g

Accordingly, the global variables of a, b and c are referenced from the procedure Q, and a, b, c, d and e from the procedure R.

This instruction copies frame-pointers to reserve the local variable area and to enable global variable references. The first operand (16-bit immediate data) specifies (in bytes) the size of the local variable area. The second operation (8-bit immediate data) specifies the depth (or lexical level) of the procedure block. The frame base address generated by this instruction is set in the BP base pointer.

To compile the EXAMPLE program follows the assembler program shown next. (The DISPOSE instruction in the assembler program is used to return the stack pointer SP and the base pointer BP to the state just before the PREPARE instruction is executed. See DISPOSE section mentioned later.)

```

START : MOV     SP, SPTOP
        MOV     BP, SP           : ①
        CALL   P                 : ②
        BR     SYSTEM
P :     PREPARE 6, 1             : ③
        MOV     AW, [BP][B+BLEVEL×2]
        ADD    AW, [BP][C+CLEVEL×2]
        MOV     [BP][A+ALEVEL×2], AW
        CALL   Q
        DISPOSE
        RET

```

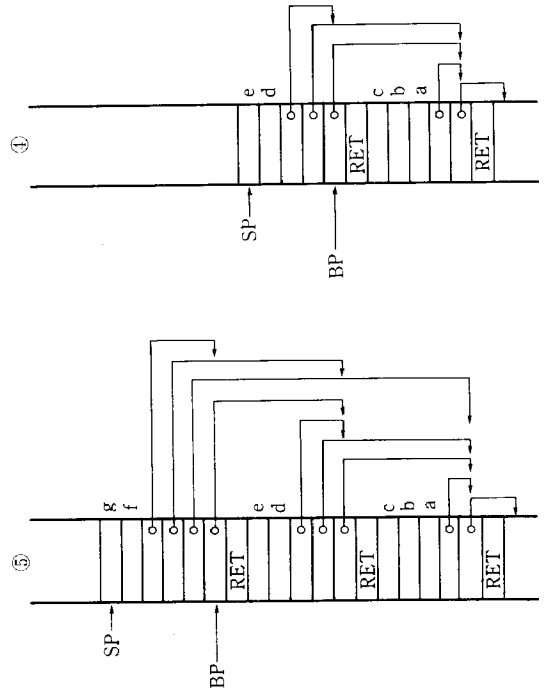


```

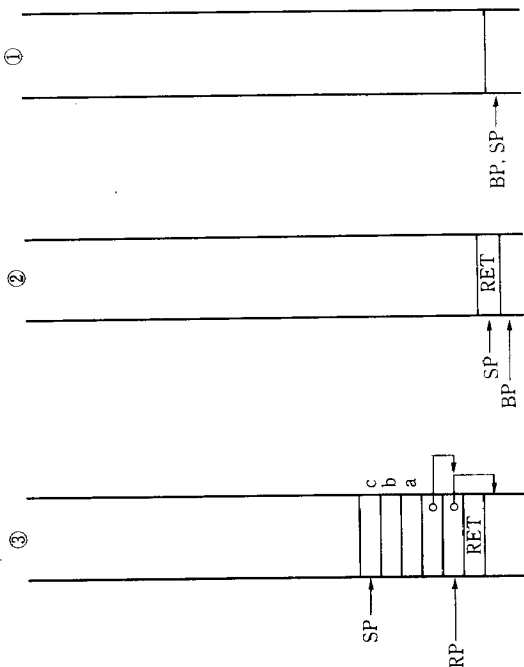
Q :   PREPARE  4, 2           : ④
      CALL    R
      MOV     AW, [BP][D+DLEVEL×2]
      ADD     AW, [BP][E+ELEVEL×2]
      MOV     IY, [BP][BLEVEL×2]
      MOV     SS : [IY][B+BLEVEL×2], AW
      DISPOSE
      RET

R :   PREPARE  4, 3           : ⑤
      MOV     AW, [BP][F+FLEVEL×2]
      ADD     AW, [BP][G+GLEVEL×2]
      MOV     IY, [BP][ALEVEL×2]
      ADD     AW, SS : [IY][A+ALEVEL×2]
      MOV     IY, [BP][DLEVEL×2]
      MOV     SS : [IY][D+DLEVEL×2], AW
      DISPOSE
      RET

; A = -2      ALEVEL = -1
; B = -4      BLEVEL = -1
; C = -6      CLEVEL = -1
; D = -2      DLEVEL = -2
; E = -4      ELEVEL = -2
; F = -2      FLEVEL = -3
; G = -4      GLEVEL = -3
    
```



The process of the generation of the stack frame according to the program is shown next. The numbers are referred to that in the program.



First the old BP value is saved to the stack. This is done so that BP of the calling procedure can be restored when the called procedure terminates. The frame pointer (BP value saved to the stack) that indicates the range of variables that can be referenced by the called procedure is placed on the stack. This range is always a value one less than the stack. This range is always a value one less than the lexical level of the procedure.

If the lexical level of a procedure is greater than 1, the pointers of that procedure will also be saved on the stack. This is so that the frame pointer of the calling procedure can also be copied when frame pointer copy is performed within the called procedure.

Next the new frame pointer value is set in BP and the area for local variables used by the procedure is reserved in the stack. In other words, SP is decremented only for the amount of stack memory required by the local variables.

```

display = 2nd operand
dynamics = 1st operand
SP = SP - 2 ;
(SP) = BP ;
temp = SP ;
if display > 0 then begin
    repeat display - 1 times
    
```

```

begin
  SP=SP-2 ;
  SP=BP-2 ;
  (SP)=(BP) ;
end ;
SP=SP-2 ;
(SP)=temp ;
end ;
BP=temp ;
SP=SP-dynamics
    
```

**Mode Operating Instructions**

The LH70116 has two operating modes (Fig. 7). One is the native mode, the other is the emulation mode in which the instruction set of the 8080A 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.

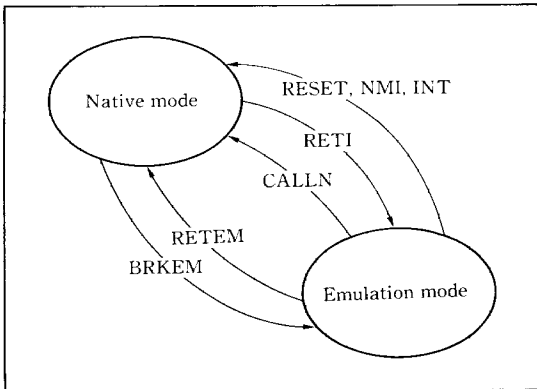


Fig. 7 V30 Modes

**Data Access**

**(1) Local variable access**

The local variables are assigned in the frame of the procedure. The effective address EA. L of the local variables is defined by the formula :

$$EA. L = SS : (BP + \text{offset})$$

The offset value is defined as the sum of the frame size (referenced frame base) and the variable from the base of the local variable area.

**(2) Global variable access**

The global variable is located at the address added by the offset of variables which are referenced to the accessed value of the base pointer of the old one saved on the stack frame.

The effective address EA. G is defined as below.

$$EA. G = SS : ((SS : (BP + \text{offset } 1)) + \text{offset } 2)$$

The offset 1 is defined by the offset value from the frame base (BP) to the address stored by the base address of the frame including the global variables.

The offset 2 is defined by the offset value from the frame base including variables to be referenced to the variables.

**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.

```

SP=BP ;
BP=(SP) ;
SP=SP+2
    
```

**Check Array Boundary Instruction**

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.

```
CHKIND reg16, mem32
```

When (mem32) > reg16 or (mem32+2) < reg16

```

TA ← (015H, 014H)
TC ← (017H, 016H)
SP ← SP - 2, (SP + 1, SP) ← PSW
IE ← 0, BREK ← 0
SP ← SP - 2, (SP + 1, SP) ← PS
PS ← TC
SP ← SP - 2, (SP + 1, SP) ← PC
PC ← TA
    } = BRK 5
    
```



**(1) BRKEN 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 8080A instructions.

In 8080 emulation mode, registers and flags of the 8080A are performed by the following registers and flags of the LH70116

	8080A	LH70116
Registers:	A	AL
	B	CH
	C	CL
	D	DH
	E	DL
	H	BH
	L	BL
	SP	BP
Flags:	PC	PC
	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.

The 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, DS<sub>0</sub>, and DS<sub>1</sub>) 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 DS<sub>0</sub> register (set by the programmer immediately before the 8080 emulation mode is entered).

It is prohibited to nest BRKEM instructions.

**(2) RETEM (no operand)**

When RETEM is executed in 8080 emulation mode (interpreted by the CPU as an 8080A 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.

**(3) CALLN imm8**

This instruction makes it possible to call the native mode subroutines from the 8080 emulation mode. To return from subroutine to the 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 8080A 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.

**(4) 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 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 8080A 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 LH70116 can be divided into two types: interrupts generated by external interrupt requests and interrupts generated by software processing. These are the classifications.

**(1) External Interrupts**

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

**(2) 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 (Fig. 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.

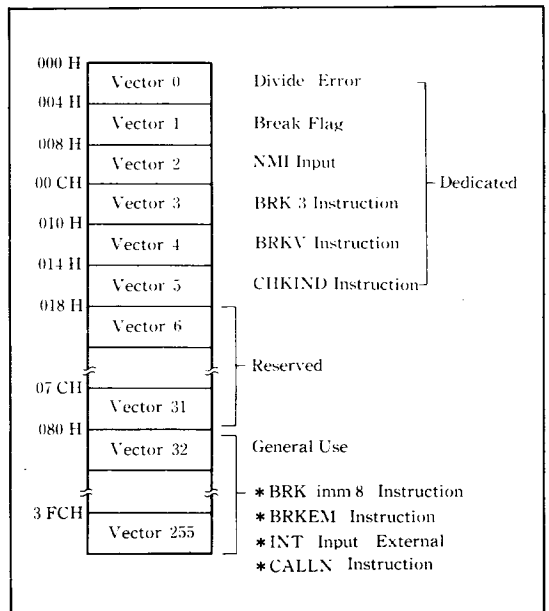


Fig. 8 Interrupt Vector Table

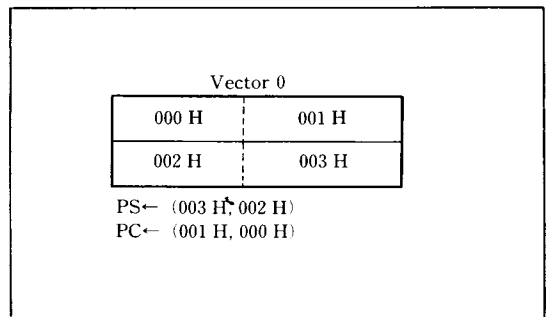


Fig. 9 Interrupt Vector 0



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.

$TA \leftarrow$  vector low bytes (offset)  
 $TC \leftarrow$  vector high bytes (segment base)  
 $SP \leftarrow SP - 2, (SP + 1, SP) \leftarrow PSW$   
 $IE \leftarrow 0, BRK \leftarrow 0, MD \leftarrow 0$   
 $SP \leftarrow SP - 2, (SP + 1, SP) \leftarrow PS$   
 $PS \leftarrow TC$   
 $SP \leftarrow SP - 2, (SP + 1, SP) \leftarrow PC$   
 $PC \leftarrow TA$

### ■ Standby Function

The LH70116 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 at either high or low levels.

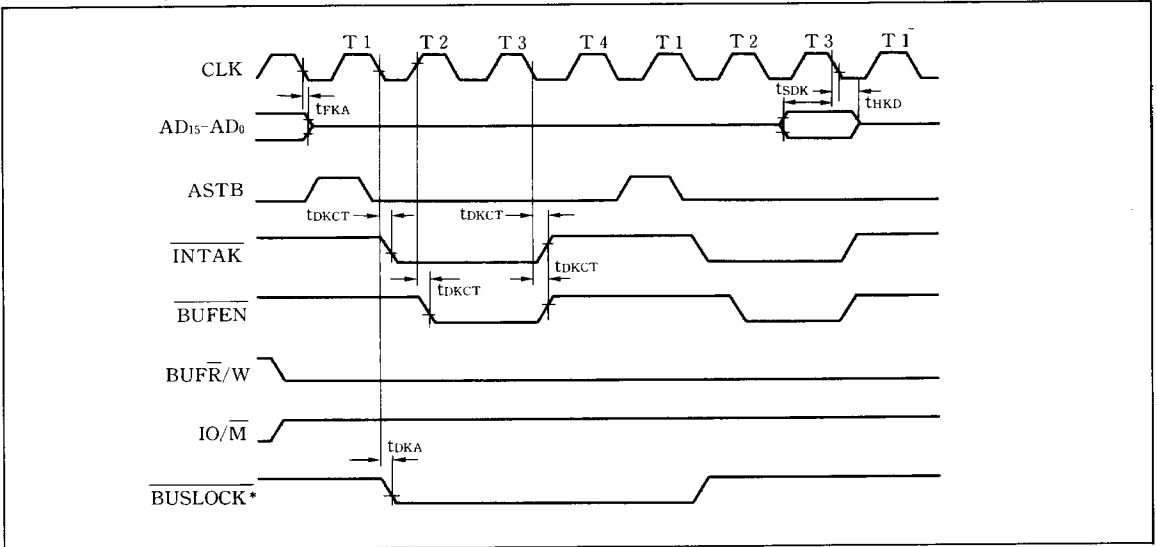
### ■ I/O Address Reservation

Reserve upper 256 bytes of I/O address (FF00H-FFFFH) in case it may be used in future.

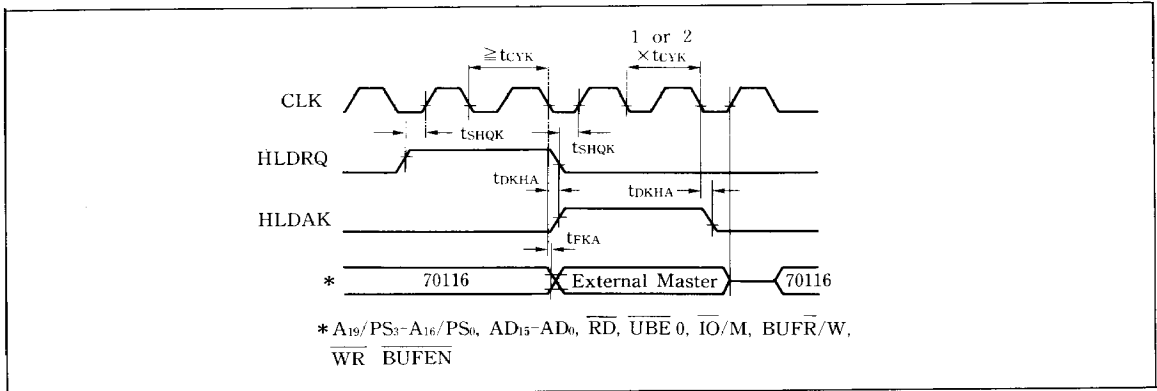




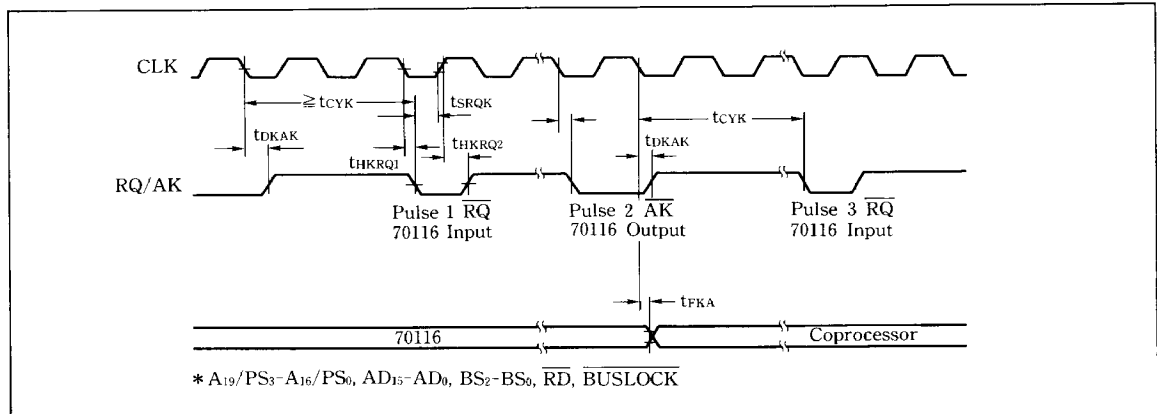
(11) Interrupt Acknowledge Timing



(12) Hold Request/Acknowledge Timing (Small Scale)



(13) Bus Request/Acknowledge Timing (Large Scale)



6

Table 1 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
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

Table 2 Operation Code Types

Identifier	Description
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

Table 3 Operational Description Types

Identifier	Description
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)
PS	Program segment register (16 bits)
SS	Stack segment register (16 bits)
DS <sub>0</sub>	Data segment 0 register (16 bits)
DS <sub>1</sub>	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) TA: Temporary register A (16 bits) TB: Temporary register B (16 bits) TC: Temporary register C (16 bits)

**Operational Description Types (cont)**

Identifier	Description
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

**Table 4 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

**Table 5 Memory Address**

mem	mod		
	00	01	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

**Table 6 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

**Table 7 Selection of Segment Registers**

sreg	
00	DS <sub>1</sub>
01	PS
10	SS
11	DS <sub>0</sub>

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 and the right side is for word operands. 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 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.



**Table 8 Primitive Block Transfer Instructions**

(n: number of transfers)

Mnemonic	Byte clocks (W=0)	Word clock (W=1)		
		Odd, Odd address	Odd, Even address	Even, Even address
MOVBK	11 + 8n (11)	11 + 16n (19)	11 + 12n (15)	11 + 8n (11)
CMPBK	7 + 14n (13)	7 + 22n (21)	7 + 18n (17)	7 + 14n (13)

Note: Values in parentheses apply to the case of single processing.

**Table 9 Primitive I/O Instructions**

(n: number of transfers)

Mnemonic	Byte clocks (W=0)	Word clock (W=1)	
		Odd address	Even address
CMPM	7 + 10n (7)	7 + 14n (11)	7 + 10n (7)
LDM	7 + 9n (7)	7 + 13n (11)	7 + 9n (7)
STM	7 + 4n (7)	7 + 8n (11)	7 + 4n (7)
INM	9 + 8n (10)	9 + 16n (18)	9 + 8n (10)
OUTM	9 + 8n (10)	9 + 16 (18)	9 + 8 (10)

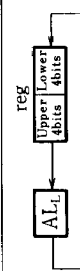
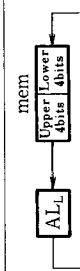
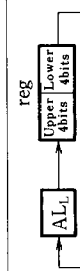
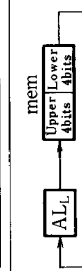
Note: Values in parentheses apply to the case of single processing.

Mnemonic	Operand	Operation	Operation Code										No. of Clocks	No. of Bytes	Flags																	
			7	6	5	4	3	2	1	0	7	6			5	4	3	2	1	0	ACC	V	P	S	Z							
<b>Data Transfer Instructions</b>																																
MOV	reg, reg	reg ← reg	1	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	2													
	mem, reg	(mem) ← reg	1	0	0	0	1	0	0	1	0	0	1	0	0	1	0	0	9/13	2-4												
	reg, mem	reg ← (mem)	1	0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	11/15	2-4												
	mem, imm	(mem) ← imm	1	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	11/15	3-6												
	reg, imm	reg ← imm	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	1	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	0	0	1	0	0	0	0	1	0	10/14	3												
	dmem, acc	When W = 0 (dmem) ← AL When W = 1 (dmem + 1) ← AH, (dmem) ← AL	1	0	1	0	0	0	0	1	1	0	1	0	1	0	1	0	9/13	3												
	sreg, reg16	sreg ← reg16 sreg: SS, DS0, DS1	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	0	2	2												
	sreg, mem16	sreg ← (mem16) sreg: SS, DS0, DS1	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	0	11/15	2-4												
	reg16, sreg	reg16 ← sreg	1	0	0	0	1	1	0	0	1	1	0	1	1	0	1	0	2	2												
	mem16, sreg	(mem16) ← sreg	1	0	0	0	1	1	0	0	1	1	0	1	1	0	1	0	10/14	2-4												
	DS0, reg16, mem32	reg16 ← (mem32) DS0 ← (mem32 + 2)	1	1	0	0	0	1	1	0	1	0	1	0	1	0	1	0	18/26	2-4												
DS1, reg16, mem32	reg16 ← (mem32) DS1 ← (mem32 + 2)	1	1	0	0	0	1	1	0	1	0	1	0	1	0	1	0	18/26	2-4													
AH, PSW	AH ← S, Z, x, AC, x, P, x, CY	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1													
PSW, AH	S, Z, x, AC, x, P, x, CY ← AH	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	3	1													
LDEA	reg16, mem16	reg16 ← mem16	1	0	0	0	1	1	0	1	1	0	1	1	0	1	0	4	2-4													
TRANS	src-table	AL ← (BW + AL)	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	9	1													
XCH	reg, reg	reg ↔ reg	1	0	0	0	0	1	1	1	0	1	1	1	0	1	1	3	2													
	mem, reg or reg, mem AW, reg16 or reg16, AW	(mem) ↔ reg AW ↔ reg16	1	0	0	0	1	1	1	0	1	1	0	1	1	0	1	16/24	2-4													
<b>Repeat Prefixes</b>																																
REPC		While CW ≠ 0, the next byte of the 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	0	1	0	1	0	1	0	2	1													
REPNC		While CW ≠ 0, the next byte of the 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	1	0	0	1	0	0	1	0	2	1													
REP		While CW ≠ 0, the next byte of the 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	1	1	1	1	0	0	1	0	1	1	1	0	0	1	1	2	1													
REPZ		CMPM and Z ≠ 1, exit the loop.																														







Mnemonic	Operand	Operation	Operation Code										No. of Clocks	No. of Bytes	Flags																
			7	6	5	4	3	2	1	0	7	6			5	4	3	2	1	0	ACC	CY	V	P	S	Z					
Addition/Subtraction Instructions (cont)																															
SUBC	reg, reg	reg ← reg - reg - CY	0	0	0	1	1	0	1	0	1	1	1	1	reg	2	2	x	x	x	x	x	x								
	mem, reg	(mem) ← (mem) - reg - CY	0	0	0	1	1	0	0	0	1	0	1	0	mem	16/24	2-4	x	x	x	x	x	x								
	reg, mem	reg ← reg - (mem) - CY	0	0	0	1	1	0	1	0	1	0	1	0	mem	11/15	2-4	x	x	x	x	x	x								
	reg, imm	reg ← reg - imm - CY	1	0	0	0	0	0	0	0	1	1	0	1	reg	4	3-4	x	x	x	x	x	x								
	mem, imm	(mem) ← (mem) - imm - CY	1	0	0	0	0	0	0	0	1	1	0	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	0	1	1	0	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	1	1	1	1	0	0	0	0	0	0	0	0	7+19n	2	u	x	u	u	u	x	
		dst BCD string ← dst BCD string - src BCD string	0	0	0	0	1	1	1	1	1	1	1	1	0	0	1	0	0	1	0	0	7+19n	2	u	x	u	u	u	x	
		dst BCD string ← ser BCD string	0	0	0	0	1	1	1	1	1	1	1	1	0	0	1	0	0	1	0	0	7+19n	2	u	x	u	u	u	x	
ROL4	reg8		0	0	0	0	1	1	1	1	1	1	1	1	0	0	1	0	0	0	0	0	13	3							
	mem8		0	0	0	0	1	1	1	1	1	1	1	1	0	0	1	0	0	0	0	0	28	3-5							
ROR4	reg8		0	0	0	0	1	1	1	1	1	1	1	1	0	0	1	0	1	0	1	0	17	3							
	mem8		0	0	0	0	1	1	1	1	1	1	1	1	0	0	1	0	1	0	1	0	32	3-5							
Increment/Decrement Instructions																															
INC	reg8	reg8 ← reg8 + 1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	0	0	0	0	0	2	2	x	x	x	x	x	x	
	mem	(mem) ← (mem) + 1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	0	0	0	0	0	16/24	2-4	x	x	x	x	x	x	
DEC	reg16	reg16 ← reg16 + 1	0	1	0	0	0	0	0	0	0	0	0	0	reg	2	1	x	x	x	x	x	x								
	reg8	reg8 ← reg8 - 1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	0	1	0	1	0	2	2	x	x	x	x	x	x	
	mem	(mem) ← (mem) - 1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	0	0	0	1	0	1	16/24	2-4	x	x	x	x	x	x	
reg16	reg16 ← reg16 - 1	0	1	0	0	1	1	1	1	1	1	1	1	0	1	1	0	0	0	1	0	1	2	1	x	x	x	x	x	x	

Mnemonic	Operand	Operation	Operation Code																No. of Clocks		No. of Bytes		Flags									
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0					A	C	Y	V	P	S	Z			
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	0	1	1	0	0	reg	21-22	2												
	mem8	AW←AL×(mem8) AH=0: CY←0, V←0 AH≠0: CY←1, V←1	1	1	1	1	0	1	1	0	1	1	0	mod	mem	27-28	2-4															
	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	0	0	reg	29-30	2															
	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	1	mod	mem	35-36 /39-40	2-4																	
	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	1	0	1	1	0	1	reg	33-39	2														
	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	1	0	mod	mem	39-45	2-4																	
MUL	reg16	DW, AW←AL×(reg8) 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	0	1	reg	41-47	2															
	mem16	DW, AW←AL×(mem8) 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	mod	mem	47-53 /51-57	2-4																	
	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	1	1	reg	28-34	3															
	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	1	mod	mem	34-40 /38-44	3-5																	
	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	0	1	1	1	1	1	reg	36-42	4															
	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	0	1	1	mod	mem	42-48 /46-52	4-6																	





Mnemonic	Operand	Operation	Operation Code																No. of Bytes	No. of Clocks	Flags							
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0			ACC	CY	V	P	S	Z		
Signed Division Instructions (cont)																												
DIV	mem8	temp←AW When temp÷(mem8)>0, temp÷(mem8)≤7FH or temp÷(mem8)<0, temp÷(mem8)>0-7FH-1 AH←temp%(mem8), AL←temp÷(mem8) When temp÷(mem8)>0, temp÷(mem8)>7FH or temp÷(mem8)<0, temp÷(mem8)>0-7FH-1 TA←(001H, 000H), TC←(003H, 002H) SP←SP-2, (SP+1, SP)←PSW, IE←0, BRK←0 SP←SP-2, (SP+1, SP)←PS, PS←TC SP←SP-2, (SP+1, SP)←PC, PC←TA	1	1	1	1	0	1	1	1	0	1	1	0	1	1	1	0	mem	34-39	2-4	u	u	u	u	u	u	
	reg16	temp←DW, AW When temp÷reg16>0, temp÷reg16≤7FFFH or temp÷reg16<0, temp÷reg16>0-7FFFH-1 DW←temp%reg16, AW←temp÷reg16 When temp÷reg16>0, temp÷reg16>7FFFH or temp÷reg16<0, temp÷reg16<0-7FFFH-1 TA←(001H, 000H), TC←(003H, 002H) SP←SP-2, (SP+1, SP)←PSW, IE←0, BRK←0 SP←SP-2, (SP+1, SP)←PS, PS←TC SP←SP-2, (SP+1, SP)←PC, PC←TA	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	reg	38-43	2	u	u	u	u	u	u	
	mem16	temp←DW, AW When temp÷(mem16)>0, temp÷(mem16)≤7FFFH or temp÷(mem16)<0, temp÷(mem16)>0-7FFFH-1 DW←temp%(mem16), AW←temp÷(mem16) When temp÷(mem16)>0, temp÷(mem16)>7FFFH or temp÷(mem16)<0, temp÷(mem16)<0-7FFFH-1 TA←(001H, 000H), TC←(003H, 002H) SP←SP-2, (SP+1, SP)←PSW, IE←0, BRK←0 SP←SP-2, (SP+1, SP)←PS, PS←TC SP←SP-2, (SP+1, SP)←PC, PC←TA	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	mem	43-48 147-52	2-4	u	u	u	u	u	u	
BCD Adjust 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	1	1	1	1	1	1	1		7	1	x	x	x	x	x	x		
ADJAA		When (AL AND 0FH)>9 or AC=1, AL←AL+6, AC←1, When AL>9FH, or CY=1 AL←AL+60H, CY←1	0	0	1	0	0	1	1	1	1	1	1	1	1	1	1		3	1	x	x	x	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	1	1	1	1	1	1	1		7	1	x	x	x	x	x	x		
ADJAS		When (AL AND 0FH)>9 or AC=1, AL←AL-6, AC←1 When AL>9FH or CY=1 AL←AL-60H, CY←1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1		3	1	x	x	x	x	x	x		



Mnemonic	Operand	Operation	Operation Code																No. of Bytes	Flags	
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0			
<b>Rotation Instructions</b>																					
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	0	reg	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	0	0	mem	16	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	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	0	0	reg	19+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	0	reg	7+n	3	x	u
ROR	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	mod	0	0	0	0	0	mem	19+n	3-5	x	u
	reg, 1	CY←LSB of 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	0	1	reg	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	0	1	1	mem	16	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	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	mod	0	0	0	1	1	mem	19+n	2-4	x	u

Mnemonic	Operand	Operation	Operation Code										No. of Clocks	No. of Bytes	Flags										
			7	6	5	4	3	2	1	0	7	6			5	4	3	2	1	0	ACC	V	P	S	Z
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	0	0	0	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	0	0	0	1	1	0	0	1	mem	19+n	3-5				x	u
	reg, 1	temp ← temp - 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	0	0	0	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	0	0	0	1	1	0	1	0	mem	16	2-4				x	x
ROL	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	0	0	1	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	0	0	1	1	1	0	1	0	mem	19+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	0	0	0	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	0	0	0	1	1	0	1	0	mem	19+n	3-5				x	u

SHARP



Mnemonic	Operand	Operation	Operation Code										No. of Clocks	No. of Bytes	Flags						
			7	6	5	4	3	2	1	0	ACC	Y			V	P	S	Z			
Rotation Instructions (cont)																					
RORC	reg, 1	temp $\leftarrow$ CY, CY $\leftarrow$ LSB of reg reg $\leftarrow$ reg $\div$ 2, MSB of reg $\leftarrow$ temp MSB of reg $\neq$ bit following MSB of reg: V $\leftarrow$ 1 MSB of reg=bit following MSB of reg: V $\leftarrow$ 0	1	1	0	1	0	0	0	W	1	1	1	0	1	reg	2	2		x	x
	mem, 1	temp $\leftarrow$ CY, CY $\leftarrow$ LSB of (mem) (mem) $\leftarrow$ (mem) $\div$ 2, MSB of (mem) $\leftarrow$ temp MSB of (mem) $\neq$ bit following MSB of (mem): V $\leftarrow$ 1 MSB of (mem)=bit following MSB of (mem): V $\leftarrow$ 0	1	1	0	1	0	0	0	W	mod	0	1	1	mem	16	2-4		x	x	
	reg, CL	temp $\leftarrow$ CL, while temp $\neq$ 0, repeat this operation, temp $\leftarrow$ CY, CY $\leftarrow$ LSB of reg, reg $\leftarrow$ reg $\div$ 2, MSB of reg $\leftarrow$ temp, temp $\leftarrow$ temp-1	1	1	0	1	0	0	1	W	1	1	0	1	1	reg	7+n	2		x	u
	mem, CL	temp $\leftarrow$ CL, while temp $\neq$ 0, repeat this operation, temp $\leftarrow$ CY, CY $\leftarrow$ LSB of (mem), (mem) $\leftarrow$ (mem) $\div$ 2 MSB of (mem) $\leftarrow$ temp, temp $\leftarrow$ temp-1	1	1	0	1	0	0	1	W	mod	0	1	1	mem	19+n	2-4		x	u	
	reg, imm8	temp $\leftarrow$ imm8, while temp $\neq$ 0 repeat this operation, temp $\leftarrow$ CY, CY $\leftarrow$ LSB of reg, reg $\leftarrow$ reg $\div$ 2 MSB of reg $\leftarrow$ temp, temp $\leftarrow$ temp-1	1	1	0	0	0	0	0	W	1	1	0	1	1	reg	7+n	3		x	u
	mem, imm8	temp $\leftarrow$ imm8, while temp $\neq$ 0, repeat this operation, temp $\leftarrow$ CY, CY $\leftarrow$ LSB of (mem), (mem) $\leftarrow$ (mem) $\div$ 2 MSB of (mem) $\leftarrow$ temp, temp $\leftarrow$ temp-1	1	1	0	0	0	0	0	W	mod	0	1	1	mem	19+n	3-5		x	u	
Subroutine Control Instructions																					
CALL	near-proc	(SP-1, SP-2) $\leftarrow$ PC, SP $\leftarrow$ SP-2 PC $\leftarrow$ PC+disp	1	1	1	0	1	0	0	0						16	3				
	regptr16	(SP-1, SP-2) $\leftarrow$ PC, SP $\leftarrow$ SP-2 PC $\leftarrow$ regptr16	1	1	1	1	1	1	1	1	1	1	0	1	0	reg	14	2			
	memptr16	(SP-1, SP-2) $\leftarrow$ PC, SP $\leftarrow$ SP-2 PC $\leftarrow$ (memptr16)	1	1	1	1	1	1	1	1	mod	0	1	0	mem	23	2-4				
	far-proc	(SP-1, SP-2) $\leftarrow$ PS, (SP-3, SP-4) $\leftarrow$ PC SP $\leftarrow$ SP-4, PS $\leftarrow$ seg, PC $\leftarrow$ offset	1	0	0	1	1	0	1	0						21	5				
	memptr32	(SP-1, SP-2) $\leftarrow$ PS, (SP-3, SP-4) $\leftarrow$ PC SP $\leftarrow$ SP-4, PS $\leftarrow$ (memptr32+2), PC $\leftarrow$ (memptr32)	1	1	1	1	1	1	1	1	mod	0	1	1	mem	31	2-4				



Mnemonic	Operand	Operation	Operation Code										No. of Clocks	No. of Bytes	Flags		
			7	6	5	4	3	2	1	0	ACCY	V			P	S	Z
<b>Subroutine Control Instructions (cont)</b>																	
RET	pop-value	PC←(SP+1, SP), SP←SP+2	1	1	0	0	0	0	1	1							1
		PC←(SP+1, SP), SP←SP+2	1	1	0	0	0	0	1	0							3
		SP←SP+2, SP←pop-value	1	1	0	0	1	0	1	0							1
	pop-value	PC←(SP+1, SP), PS←(SP+3, SP+2)	1	1	0	0	1	0	1	1							3
		SP←SP+4	1	1	0	0	1	0	1	0							3
<b>Stack Manipulation Instructions</b>																	
PUSH	mem16	SP←SP-2 (SP+1, SP)←(mem16)	1	1	1	1	1	1	1	1	mod	1	1	0	mem		2-4
	reg16	SP←SP-2 (SP+1, SP)←reg16	0	1	0	1	0	0	reg							1	
	sreg	SP←SP-2 (SP+1, SP)←sreg	0	0	0	sreg	1	1	0							1	
	PSW	SP←SP-2 (SP+1, SP)←PSW	1	0	0	1	1	0	0							1	
	R	Push registers on the stack	0	1	1	0	0	0	0							1	
	imm	(SP-1, SP-2)←imm	0	1	1	0	1	0	S							2-3	
		SP←SP-2, When S=1, sign extension	1	0	0	0	1	1	1	mod	0	0	0	mem		2-4	
POP	mem16	(mem16)←(SP+1, SP), SP←SP+2	0	1	0	1	1	reg							1		
	reg16	reg16←(SP+1, SP), SP←SP+2	0	1	0	1	1	reg							1		
	sreg	sreg←(SP+1, SP) sreg: SS, DS0, DS1	0	0	0	sreg	1	1	1							1	
	PSW	PSW←(SP+1, SP), SP←SP+2	1	0	0	1	1	0	1							1	
	R	Pop registers from the stack	0	1	1	0	0	0	1							4	
PREPARE	imm16, imm8	Prepare new stack frame	1	1	0	0	1	0	0	★: imm8<0: 16						4	
			imm8<1: 23+16 (imm8-1)														
DISPOSE		Dispose of stack frame	1	1	0	0	1	0	0	1							1
<b>Branch Instruction</b>																	
BR	near-label	PC←PC+disp	1	1	1	0	1	0	0	1							13
	short-label	PC←PC+ext-disp8	1	1	1	0	1	0	1							2	
	regptr16	PC←regptr16	1	1	1	1	1	1	1	1	1	0	0	reg		2	
	memptr16	PC←(memptr16)	1	1	1	1	1	1	1	mod	1	0	0	mem		2-4	
	far-label	PC←seg, PC←offset	1	1	1	0	1	0	1	0							5
	memptr32	PS←(memptr32+2), PS←(memptr32)	1	1	1	1	1	1	1	mod	1	0	1	mem		2-4	
<b>Conditional Branch Instructions</b>																	
BV	short-label	if V=1, PC←PC+ext-disp8	0	1	1	1	0	0	0	0							14/4
BNV	short-label	if V=0, PC←PC+ext-disp8	0	1	1	1	0	0	1							2	
BC, BL	short-label	if CY=1, PC←PC+ext-disp8	0	1	1	1	0	0	1							2	
BNC, BNL	short-label	if CY=0, PC←PC+ext-disp8	0	1	1	1	0	0	1							2	
BE, BZ	short-label	if Z=1, PC←PC+ext-disp8	0	1	1	1	0	1	0							2	
BNE, BNZ	short-label	if Z=0, PC←PC+ext-disp8	0	1	1	1	0	1	0							2	
BNH	short-label	if CY OR Z=1, PC←PC+ext-disp8	0	1	1	1	0	1	1							2	
BH	short-label	if CY OR Z=0, PC←PC+ext-disp8	0	1	1	1	0	1	1							2	
BN	short-label	if S=1, PC←PC+ext-disp8	0	1	1	1	1	0	0							2	
BNP	short-label	if S=0, PC←PC+ext-disp8	0	1	1	1	1	0	1							2	
BPE	short-label	if P=1, PC←PC+ext-disp8	0	1	1	1	1	0	1							2	







Mnemonic	Operand	Operation	Operation Code										No. of Bytes	Flags							
			7	6	5	4	3	2	1	0	1	0		1	0	ACCY	V	P	S	Z	
<b>Logical Operation Instructions</b>																					
TEST	reg, reg	reg AND reg	1	0	0	0	1	0	W	1	1	reg	reg	2	u	0	0	x	x		
	mem, reg or reg, mem	(mem) AND reg	1	0	0	0	1	0	W	mod	mem	mem	10/14	u	0	0	x	x			
	reg, imm	reg AND imm	1	1	1	0	1	1	W	1	1	0	0	reg	3-4	u	0	0	x	x	
	mem, imm	(mem) AND imm	1	1	1	0	1	1	W	mod	0	0	mem	11/15	u	0	0	x	x		
	acc, imm	When W=0, AL AND imm8 When W=1, AW AND imm8	1	0	1	0	1	0	W	1	1	0	0	mem	2-3	u	0	0	x	x	
AND	reg, reg	reg←reg AND reg	0	0	1	0	0	1	W	1	1	reg	reg	2	u	0	0	x	x		
	mem, reg	(mem)←(mem) AND reg	0	0	1	0	0	1	W	mod	reg	mem	16/24	u	0	0	x	x			
	reg, mem	reg←reg AND (mem)	0	0	1	0	0	1	W	mod	reg	mem	11/15	u	0	0	x	x			
	reg, imm	reg←reg AND imm	1	0	0	0	0	0	W	1	1	0	0	reg	3-4	u	0	0	x	x	
	mem, imm	(mem)←(mem) AND imm When W=0, AL AND imm8 When W=1, AW←AW AND imm16	1	0	0	0	0	0	W	mod	1	0	mem	18/26	u	0	0	x	x		
acc, imm	When W=1, AW←AW AND imm16	0	0	1	0	0	1	0	W				4	u	0	0	x	x			
<b>Logical Operation Instructions (cont)</b>																					
OR	reg, reg	reg←reg OR reg	0	0	0	1	0	1	W	1	1	reg	reg	2	u	0	0	x	x		
	mem, reg	(mem)←(mem) OR reg	0	0	0	1	0	1	W	mod	reg	mem	16/24	u	0	0	x	x			
	reg, mem	reg←reg OR (mem)	0	0	0	1	0	1	W	mod	reg	mem	11/15	u	0	0	x	x			
	reg, imm	reg←reg OR imm	1	0	0	0	0	0	W	1	1	0	1	reg	3-4	u	0	0	x	x	
	mem, imm	(mem)←(mem) OR imm When W=0, AL←AL OR imm8 When W=1, AW←AW OR imm16	1	0	0	0	0	0	W	mod	0	1	mem	18/26	u	0	0	x	x		
acc, imm	When W=1, AW←AW OR imm16	0	0	0	1	1	1	W				4	u	0	0	x	x				
XOR	reg, reg	reg←reg XOR reg	0	0	1	1	0	0	1	W	1	1	reg	reg	2	u	0	0	x	x	
	mem, reg	(mem)←(mem) XOR reg	0	0	1	1	0	0	1	W	mod	reg	mem	16/24	u	0	0	x	x		
	reg, mem	reg←reg XOR (mem)	0	0	1	1	0	0	1	W	mod	reg	mem	11/15	u	0	0	x	x		
	reg, imm	reg←reg XOR imm	1	0	0	0	0	0	W	1	1	1	0	reg	3-4	u	0	0	x	x	
	mem, imm	(mem)←(mem) XOR imm When W=0, AL←AL XOR imm8 When W=1, AW←AW XOR imm16	1	0	0	0	0	0	W	mod	1	1	0	mem	18/26	u	0	0	x	x	
acc, imm	When W=1, AW←AW XOR imm16	0	0	1	1	0	1	0	W				4	u	0	0	x	x			
<b>Bit Operation Instructions</b>																					
TEST1	reg8, CL	reg8 bit no, CL=0: Z←1 reg8 bit no, CL=1: Z←0	2nd byte★										3rd byte★	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 0 1										1 1 0 0 0 0	reg	8	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 0 1										1 1 0 0 0 0	reg	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 0 0										mod 0 0 0 0	mem	8/12	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 0										1 1 0 0 0 0	reg	4	u	0	0	u	u	x



Mnemonic	Operand	Operation	Operation Code																No. of Clocks	No. of Bytes	Flags										
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0			ACC	V	P	S	Z						
Bit Operation Instructions(cont)																															
TEST1	mem8, imm3	(mem8) bit no. imm3=0: Z←1 (mem8) bit no. imm3=1: Z←0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	mod	0	0	0	0	mem	9	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	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	1	1	0	0	0	0	0	mem	9/13	4-6	u	0	0	u	u	x					
			2nd byte★ ★Note: First byte=0FH	3rd byte★																											
NOT1	reg8, CL	reg8 bit no. CL←reg8 bit no. CL	0	0	0	1	0	1	1	0	1	1	0	0	0	0	0	reg	4	3											
	mem8, CL	(mem8) bit no. CL←(mem8) bit no. CL	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	mem	13	3-5											
	reg16, CL	reg16 bit no. CL←reg16 bit no. CL	0	0	0	1	0	1	1	1	1	0	0	0	0	0	0	reg	4	3											
	mem16, CL	(mem16) bit no. LC←(mem16) bit no. CL	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	mem	13/21	3-5											
	reg8, imm3	reg8 bit no. imm3←reg8 bit no. imm3	0	0	0	1	1	1	1	0	1	0	0	0	0	0	0	reg	5	4											
	mem8, imm3	(mem8) bit no. imm3←(mem8) bit no. imm3	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	mem	14	4-6											
	reg16, imm4	reg16 bit no. imm4←(reg16) bit no. imm4	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	reg	5	4											
	mem16, imm4	(mem16) bit no. imm4←(mem16) bit no. imm4	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	mem	14/22	4-6											
			2nd byte★ ★Note: First byte=0FH	3rd byte★																											
		CY	CY←CY	1	1	1	1	0	1	0	1	0	1	0	1	0	1	0		2	1										
CLR1	reg8, CL	reg8 bit no. CL←0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	reg	5	3											
	mem8, CL	(mem8) bit no. CL←0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	mem	14	3-5											
	reg16, CL	reg16 bit no. CL←0	0	0	0	1	0	0	1	1	1	0	0	0	0	0	0	reg	5	3											
	mem16, CL	(mem16) bit no. CL←0	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	mem	14/22	3-5											
	reg8, imm3	reg8 bit no. imm3←0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	0	reg	6	4											
	mem8, imm3	(mem8) bit no. imm3←0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	mem	15	4-6											
	reg16, imm4	reg16 bit no. imm4←0	0	0	0	1	1	0	1	1	1	0	0	0	0	0	0	reg	6	4											
	mem16, imm4	(mem16) bit no. imm4←0	0	0	0	1	1	0	1	1	1	0	0	0	0	0	0	mem	15/23	4-6											
		2nd byte★ ★Note: First byte=0FH	3rd byte★																												
	CY	CY←0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0		2	1											
	DIR	DIR←0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0		2	1											



Mnemonic	Operand	Operation	Operation Code										No. of Clocks	No. of Bytes	Flags		
			7	6	5	4	3	2	1	0	ACC	V			P	S	Z
<b>Bit Operation Instructions (cont)</b>																	
SETI	reg8, CL mem8, CL reg16, CL mem16, CL reg8, imm3 mem8, imm3 reg16, imm4 mem16, imm4	reg8 bit no. CL←1 (mem8) bit no. CL←1 reg16 bit no. CL←1 (mem16) bit no. CL←1 reg8 bit no. imm3←1 (mem8) bit no. imm3←1 reg16 bit no. imm4←1 (mem16) bit no. imm4←1	0 0 0 1 0 1 0 0 1 1 0 0 0	reg	4	3											
			0 0 0 1 0 1 0 0 0 0 0 0 0	mem	13	3-5											
			0 0 0 1 0 1 0 1 1 0 0 0 0	reg	4	3											
			0 0 0 1 0 1 0 1 1 0 0 0 0	mem	13/21	3-5											
			0 0 0 1 1 1 0 0 1 1 0 0 0	reg	5	4											
			0 0 0 1 1 1 0 0 0 0 0 0 0	mem	14	4-6											
			0 0 0 1 1 1 0 1 1 0 0 0 0	reg	5	4											
			0 0 0 1 1 1 0 1 1 0 0 0 0	mem	14	4-6											
			2nd byte★ 3rd byte★														
			★Note: First byte=0FH														
	CY←1		1 1 1 1 1 0 0 1		2	1										1	
	DIR←1		1 1 1 1 1 1 0 1		2	1										1	
<b>Shift Instructions</b>																	
SHL	reg, 1	CY←MSB or reg, reg←X2 When MSB of reg≠CY, V←1 When MSB of reg=CY, V←0	1 1 0 1 0 0 0 W	reg	2	2										u x x x x x	
	mem, 1	CY←MSB or (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	mem	16/24	2-4										u x x x x x	
	reg, CL	temp←CL, while temp≠0, repeat this operation, CY←MSB or reg, reg←reg×2, temp←temp-1	1 1 0 1 0 0 1 W	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	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	reg	7+n	3										u x u x x x	
	mem, imm8	temp←imm8, while temp≠0, repeat this operation, CY←MSB or (mem), (mem)←(mem)×2, temp←temp-1	1 1 0 0 0 0 0 W	mem	19/27+n	3-5										u x u x x x	
SHR	reg, 1	CY←LSB of 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 n: number of shifts	reg	2	2										u x x x x x	



Mnemonic	Operand	Operation	Operation Code																No. of Bytes	Flags							
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0		Clocks	ACC	Y	V	P	S	Z	
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	0	0	0	0	0	0	0	0	0	16/24	2-4	u	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	0	0	0	0	0	0	0	0	0	0	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	0	0	0	0	0	0	0	0	0	0	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	0	0	0	0	0	0	0	0	0	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	0	0	0	0	0	0	0	0	0	19/27+n	3-5	u	x	u	x	x	x	
			n: number of shifts	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	2	2	u	x	0	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	0	0	0	0	0	0	0	0	0	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	0	0	0	0	0	0	0	0	0	16	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	0	0	0	0	0	0	0	0	0	0	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	0	0	0	0	0	0	0	0	0	0	19+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	0	0	0	0	0	0	0	0	0	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	0	0	0	0	0	0	0	0	0	19+n	3-5	u	x	u	x	x	x	
		n: number of shifts	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0										

