LH70108(V20) High-Performance 16-Bit Microprocessor

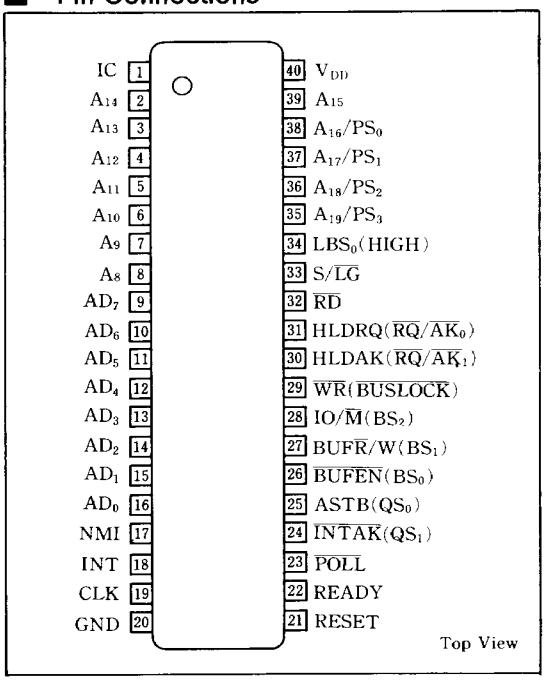
■ Description

The LH70108(V20) is a CMOS 16-bit microprocessor with internal 16-bit architecture and an 8-bit external data bus. The LH70108 additionally has a powerful instruction set including bit processing, packed BCD operations, and high-speed multiplication/division operations. The LH70108 can also execute the entire 8080 instruction set comes with a standby mode that significantly reduces power consumption. It is software-compatible with the LH70116 16-bit microprocessor.

■ Features

1. Minimum instruction execution time: 250ns (at 8MHz)
2. Maximum addressable memory: 1M byte
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)
10. High-speed block transfer instructions: 1M byte/s (at 8MHz)
11. Hign-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 8080 emulation mode
14. CMOS technology
15. Low-power consumption
16. Low-power standby mode
17. Single power supply
18. 5MHz or 8MHz clock
19. 40-pin DIP (DIP40-P-600)

■ Pin Connections

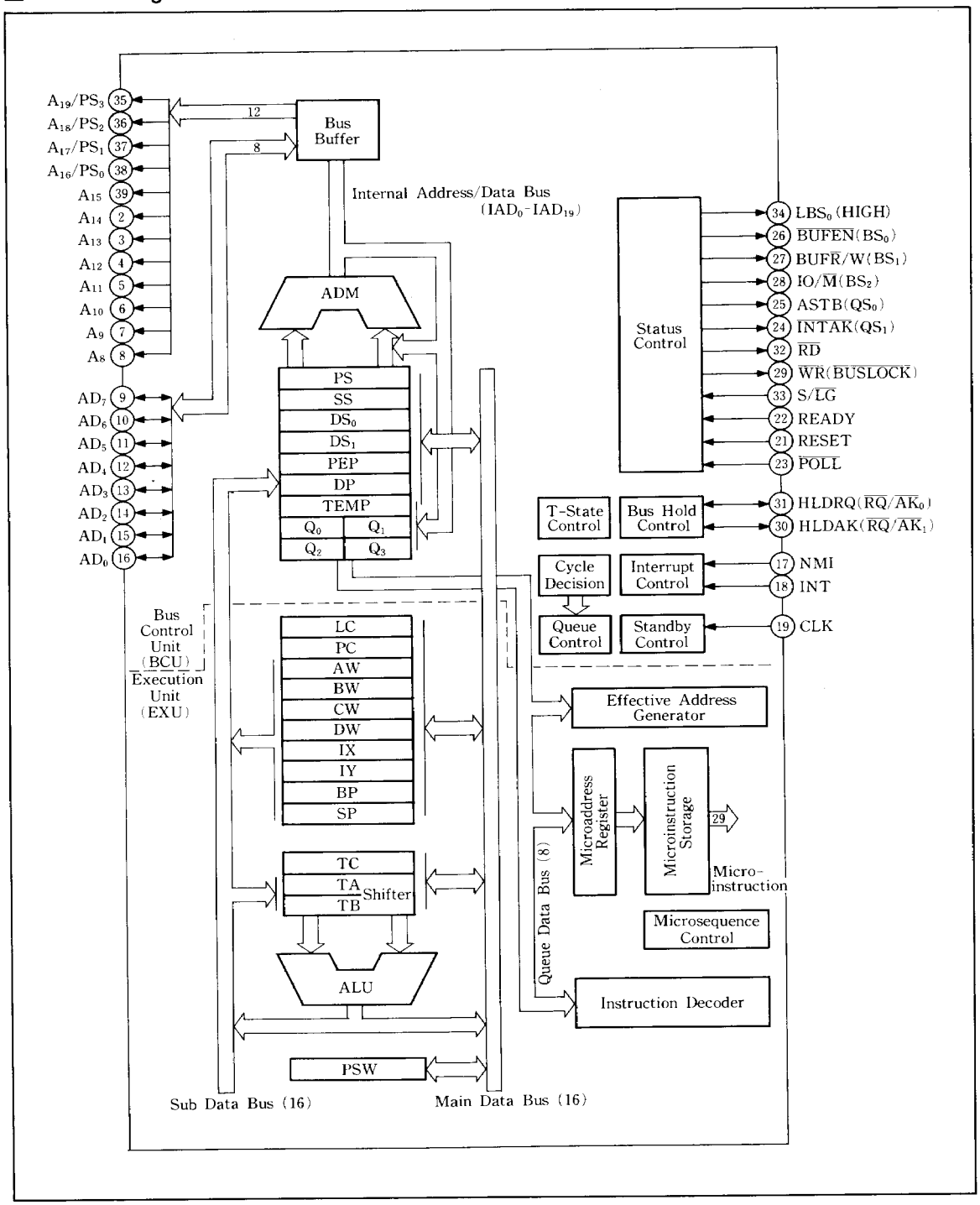


■ Ordering Information

LH70108-X
- Frequency
  - 5: 5MHz
  - 8: 8MHz
- Model No.

*V20 is a trademark of NEC corporation.

Block Diagram



Pin Identification

No.	Symbol	Direction	Function
1	IC*		Internally connected
2-8	A_{14}-A_8	Out	Address but, middle bits
9-16	AD_7-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 potential
21	RESET	In	Reset input
22	READY	In	Ready input
23	$\overline{\text{POLL}}$	In	Poll input
24	$\overline{\text{INTAK}}$ (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	$\overline{\text{BUFEN}}$ (BS_0)	Out	Buffer enable output (bus status bit 0 output)
27	BUFR/$\overline{\text{W}}$ (BS_1)	Out	Buffer read/write output (bus status bit 1 output)
28	IO/$\overline{\text{M}}$ (BS_2)	Out	Access is I/O or memory (bus status bit 2 output)
29	$\overline{\text{WR}}$ ($\overline{\text{BUSLOCK}}$)	Out	Write strobe output (bus lock output)
30	$\overline{\text{HLDAK}}$ ($\overline{\text{RQ}}/\overline{\text{AK}}_1$)	Out (In/Out)	Hold acknowledge output, (bus hold request input/ acknowledge output 1)
31	$\overline{\text{HLDRQ}}$ ($\overline{\text{RQ}}/\overline{\text{AK}}_0$)	In (In/Out)	Hold request input (bus hold request input/ acknowledge output 0)
32	$\overline{\text{RD}}$	Out	Read strobe output
33	S/$\overline{\text{LG}}$	In	Small-scale/large-scale system input
34	LBS_0 (HIGH)	Out	Latched bus status output 0 (always high in large-scale systems)
35-38	A_{19}/PS_3-A_{16}/PS_0	Out	Address bus, high bits or processor status output
39	A_{15}	Out	Address 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.

■ Absolute Maximum Ratings (Ta=+25℃)

Parameter	Symbol	Ratings	Units
Supply voltage	V_{DD}	−0.5 to +7.0	V
Input voltage	V_I	−0.5 to V_{DD}+0.3	V
CLK input voltage	V_K	−0.5 to V_{DD}+1.0	V
Output voltage	V_O	−0.5 to V_{DD}+0.3	V
Operating temperature	Topr	−40 to +85	℃
Storage temperature	Tstg	−65 to +150	℃

■ Capacitance (Ta=+25℃, V_{DD}=0V)

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

■ DC Characteristics

(LH70108-5, Ta=−40℃ to+85℃, V_{pp}=+5V±10%)
(LH70108-8, Ta=−10℃ to+70℃, V_{pp}=+5V±5%)

Parameter	Symbol	Conditions		MIN.	TYP.	MAX.	Unit
Input HIGH voltage	V_{IH}			2.2		V_{DD}+0.3	V
Input LOW voltage	V_{IL}			−0.5		0.8	V
CLK input HIGH voltage	V_{KH}			3.9		V_{DD}+1.0	V
CLK input LOW voltage	V_{KL}			−0.5		0.6	V
Output HIGH voltage	V_{OH}	I_{OH}=−400 μA		0.7V_{DD}			V
Output LOW voltage	V_{OL}	I_{OL}=2.5mA				0.4	V
Input leakage HIGH current	I_{LIH}	V_I=V_{DD}				10	μA
Input leakage LOW current	I_{LIL}	V_I=0V				−10	μA
Output leakage HIGH current	I_{LOH}	V_O=V_{DD}				10	μA
Output leakage LOW current	I_{LOL}	V_O=0V				−10	μA
HLDRQ input HIGH current	I_{HQH}	V_I=V_{DD}				10	μA
HLDRQ input LOW current	I_{HQL}	V_I=0V				−0.5	mA
Supply current	I_{DD}	Normal operation	70108-5		30	60	mA
		Standby mode	5MHz		5	10	mA
		Normal operation	70108-8		45	80	mA
		Standby mode	8MHz		6	12	mA

■ AC Characteristics

(LH70108-5, Ta = −40℃ to +85℃, V_{DD} = +5V ± 10%)
(LH70108-8, Ta = −10℃ to +70℃, V_{DD} = +5V ± 5%)

Parameter	Symbol	Conditions	LH70108-5		LH70108-8		Unit
			MIN.	MAX.	MIN.	MAX.	
Small/Large Scale							
Clock cycle	t_{CYK}		200	500	125	500	ns
Clock pulse HIGH width	t_{KKH}	V_{KH} = 3.0V	69		44		ns
Clock pulse LOW width	t_{KKL}	V_{KL} = 1.5V	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, $\overline{POLL}$ 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 = 100pF	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 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 $\overline{RD}$ ↓	t_{AFRL}		0		0		ns
$\overline{RD}$ ↓ delay time from CLK ↓	t_{DKRL}		10	165	10	80	ns
$\overline{RD}$ ↑ delay time from CLK ↓	t_{DKRH}		10	150	10	80	ns
Address delay time from $\overline{RD}$ ↑	t_{DRHA}		t_{CYK} − 45		t_{CYK} − 40		ns
$\overline{RD}$ 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
$\overline{WR}$ 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
$\overline{BUFEN}$ ↑ from $\overline{WR}$ ↑	t_{WCT}		t_{KKL} − 20		t_{KKL} − 20		ns

■ AC Characteristics (cont)

(LH70108-5, Ta = −40℃ to +85℃, V_{DD} = −5V ± 10%)
(LH70108-8, Ta = −10℃ to 70℃, V_{DD} = +5V ± 5%)

Parameter	Symbol	Conditions	LH70108-5		LH70108-8		Unit
			MIN.	MAX.	MIN.	MAX.	
Large Scale							
Address delay time from CLK ↓	t_{DKA}	C_L = 100pF	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 $\overline{RD}$ ↑	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
$\overline{RD}$ ↓ delay time from address float	t_{DAFRL}		0		0		ns
$\overline{RD}$ ↓ delay time from CLK ↓	t_{DKRL}		10	165	10	80	ns
$\overline{RD}$ ↑ delay time from CLK ↓	t_{DKRH}		10	150	10	80	ns
$\overline{RD}$ 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
$\overline{AK}$ delay time from CLK ↓	t_{DKAK}			70		50	ns
$\overline{RQ}$ setup time to CLK ↑	t_{SRQK}		20		10		ns
$\overline{RQ}$ hold time after CLK ↓	t_{HKRQ1}		0		0		ns
$\overline{RQ}$ hold time after CLK ↑	t_{HKRQ2}		40		30		ns

■ Pin Functions

Some pins of the LH70108 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.

A_{15}-A_8 (Address Bus)

For small- and large-scale systems.

The CPU uses these pins to output the middle 8 bits of the 20-bit address data. They are three-state outputs and become high impedance during hold acknowledge.

AD_7-AD_0 (Address/Data Bus)

For small- and large-scale systems.

The CPU uses these pins as the time-multiplexed address and data bus. When high, and AD bit is a one; when low, an AD bit is a zero. This bus contains the lower 8 bits of the 20-bit address during T1 of the bus cycle and is used as an 8-bit data bus during T2, T3, and T4 of the bus cycle.

Sixteen-bit data I/O performed in two steps. The low byte is sent first, followed by the high byte. The address/data bus is a three-state bus and can be at a high or low level during standby mode. The bus will be 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 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 LH70108 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 $\overline{\text{INTAK}}$ 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 LH70108 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 LH70108 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 level) and requesting a read/write cycle delay.

If the READY signal is active (high level) during either the T3 or Tw 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.

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

For small- and large-scale systems.

The CPU checks this input upon execution of the $\overline{\text{POLL}}$ 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/$\overline{\text{M}}$ sig-

nal 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/$\overline{LG}$ (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 pin 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/$\overline{LG}$-high	S/$\overline{LG}$-low
24	$\overline{INTAK}$	QS_1
25	ASTB	QS_0
26	$\overline{BUFEN}$	BS_0
27	BUF$\overline{R}$/W	BS_1
28	IO/$\overline{M}$	BS_2
29	$\overline{WR}$	$\overline{BUSLOCK}$
30	HLDAK	$\overline{RQ}/AK_1$
31	HLDRQ	$\overline{RQ}/AK_0$
34	LBS_0	Always high

$\overline{INTAK}$ (Interrupt Acknowledge)

For small-scale systems.

The CPU generates the $\overline{INTAK}$ 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).

ASTB (Address Strobe)

For small-scale systems.

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

ASTB is held at a low level during standby mode however, goes high at one time for a half clock cycle to latch L_{BS0} output.

$\overline{BUFEN}$ (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.

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 is a three-state output and becomes impedance during hold acknowledge.

IO/$\overline{M}$ (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 I/O and a low-level signal specifies memory.

IO/M's output is three state and becomes 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 IO/M signal.

This three-state output is held high during standby mode and enters the high-impedance state 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 a high level, 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.

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

LBS_0 (Latched Bus Status 0)

For small-scale systems.

The CPU uses the signal along with the IO/$\overline{M}$ and BUF$\overline{R}$/W signals to inform an external device what the current bus cycle is.

IO/$\overline{M}$	BUFR/$\overline{W}$	LBS_0	Bus Cycle
0	0	0	Program fetch
0	0	1	Memory read
0	1	0	Memory write
0	1	1	Passive state
1	0	0	Interrupt acknowledge
1	0	1	I/O read
1	1	0	I/O write
1	1	1	Halt

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 signal 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 to 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

The output of these pins is three state and becomes 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, about 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	Queue empty
1	1	Subsequent bytes of instruction

The insturction 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_2-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

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.

$\overline{BUSLOCK}$ (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, 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. $\overline{BUSLOCK}$ 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 a high level when the output is high impedance.

Bus Hold Request Input (RQ) must be input in

synchronization with external clock signals 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)

This pin is used for tests performed at the factor by SHARP. The LH70108 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 (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 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_0, and DS_1)

The memory addresses accessed by the LH70108 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_0 (Data Segment 0)	IX, effective address
DS_1 (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: Work 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)

Control Flags
- MD (Mode)
- DIR (Direction)
- 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
MD	1	1	1	V	DIR	IE	BRK	S	Z	0	AC	0	P	1	CY

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_0-Q_3 (Prefetch Queue)

The LH70108 has 4 byte instruction queue (FIFO), and it can store up to 4 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 byte 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 lewer 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 micro-instructions.

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/shif-

ter 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 highlinghts the major architectural features that enhance the performance of the LH70108.

- 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 LH70108 (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.

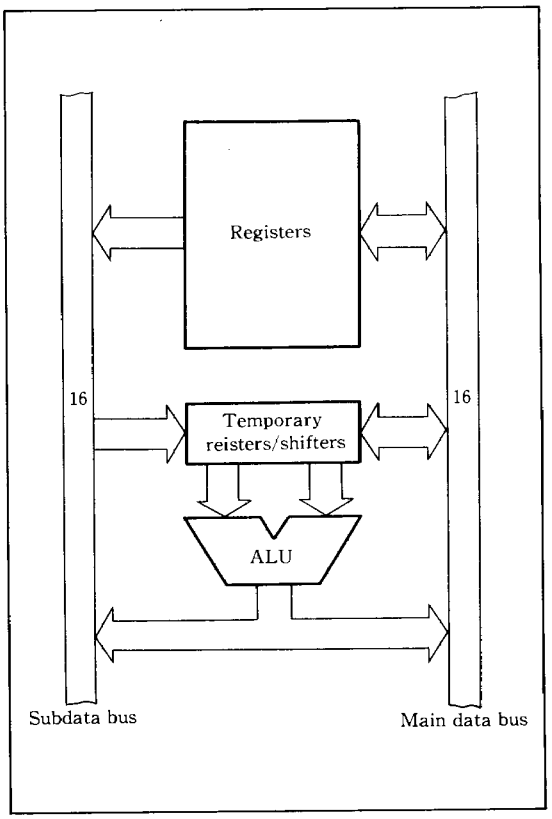


Fig. 1 Dual Data Buses

Example

ADD AW, BW; AW←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.

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.

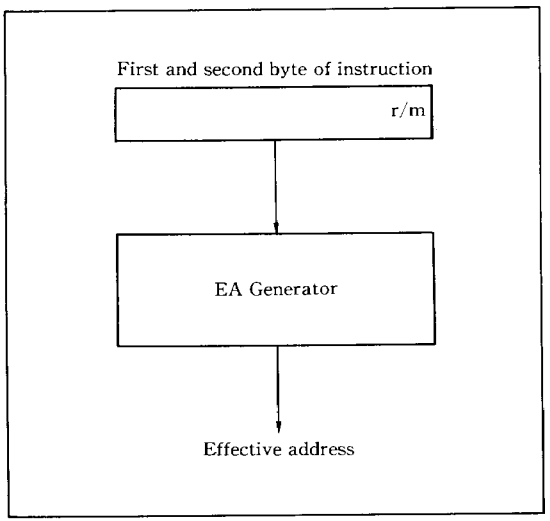


Fig. 2 Effective Address Generator

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 divison 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 LH70108 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_1 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.

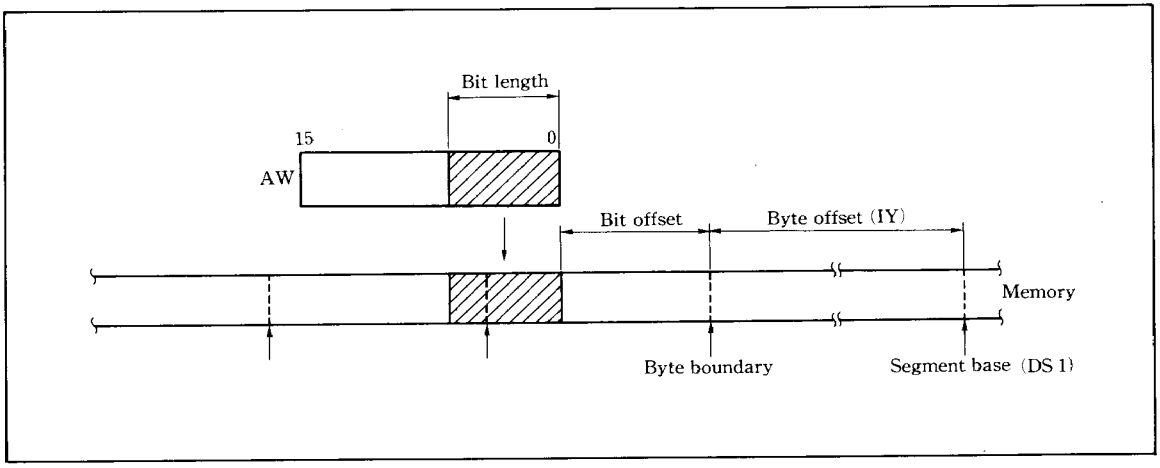


Fig. 3 Bit Field Insertion

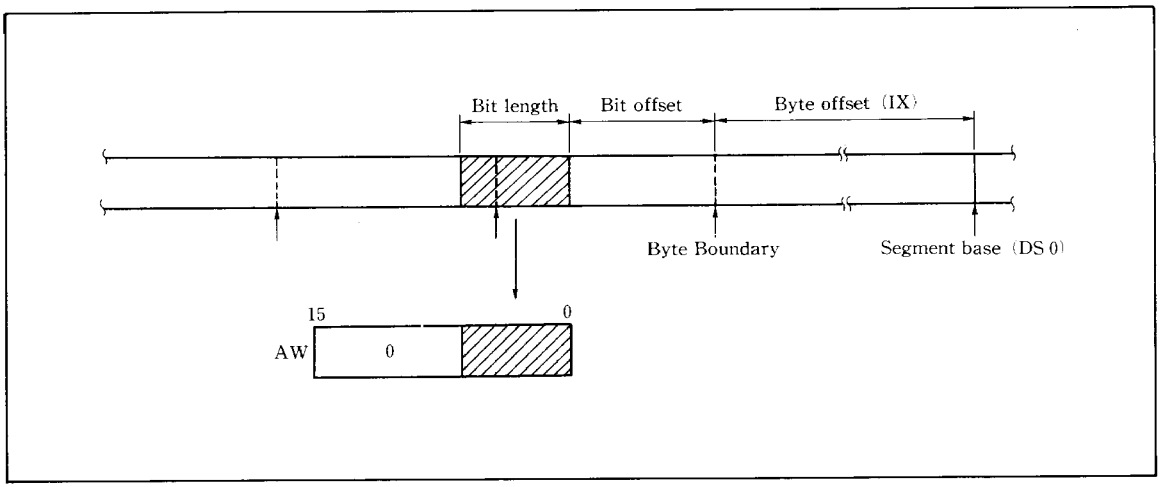


Fig. 4 Bit Field Extraction

(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_0 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 (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.

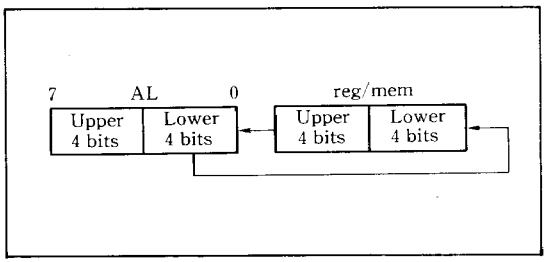


Fig. 5 BCD Rotate Left (ROL4)

(5) 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 the data one BCD digit to the right

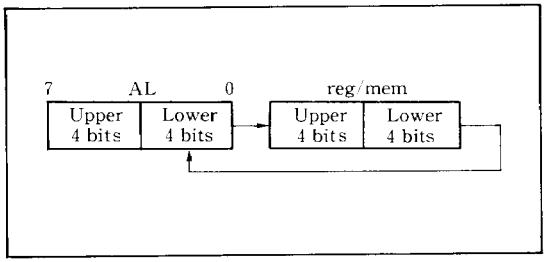


Fig. 6 BCD Rotate Right (ROR4)

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

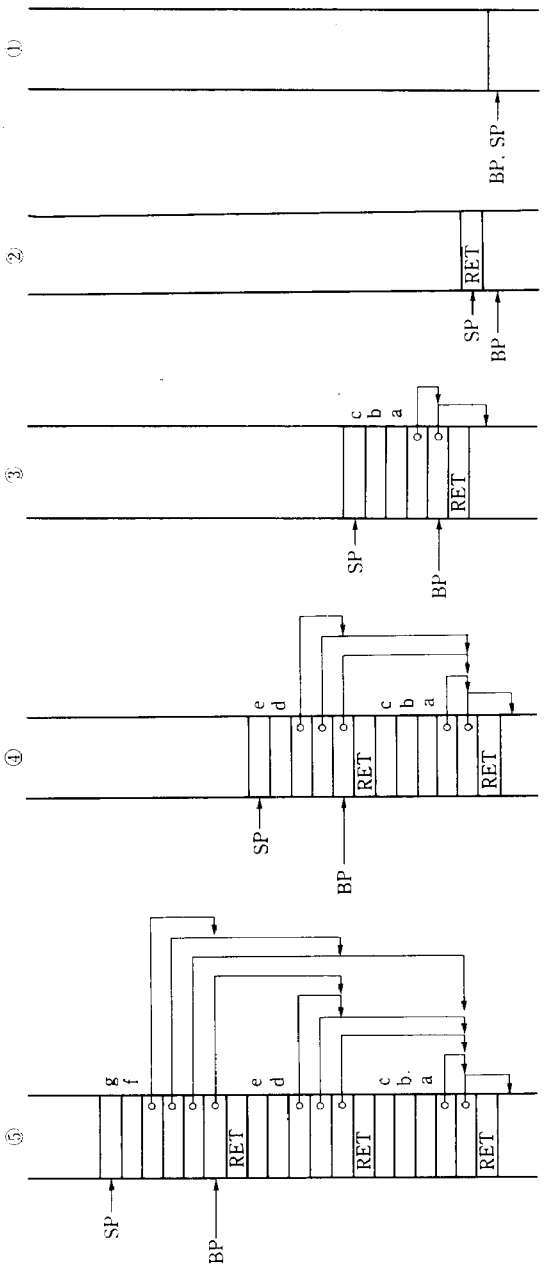
Accordingly, the global variables of a, b and c are referenced from the procedure Q, and a, b, c, d and e from the procedure R.

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

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+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 refer-

enced 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+offset 1))+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

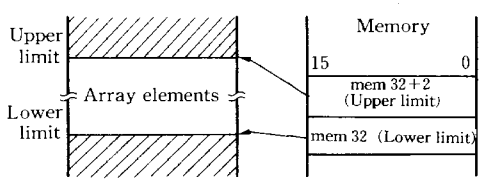
Check Array Boundary Instruction

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



Mode Operating Instructions

The LH70108 has two operating modes (figure 7). One is the native mode, and 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.

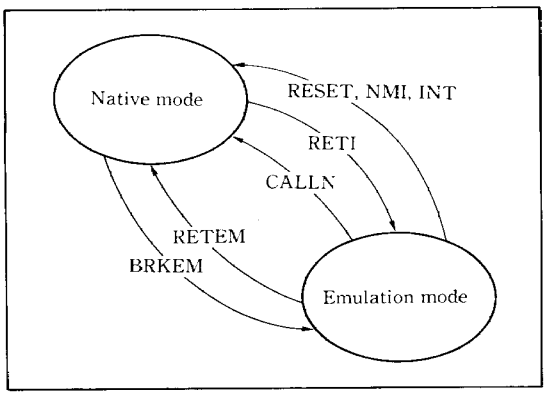


Fig. 7 V20 Modes

(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 LH70108.

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

	8080A	LH70108
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

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_0, and DS_1) 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_0 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 a 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 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 initerrupt 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 LH70108 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 Emmulation 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.

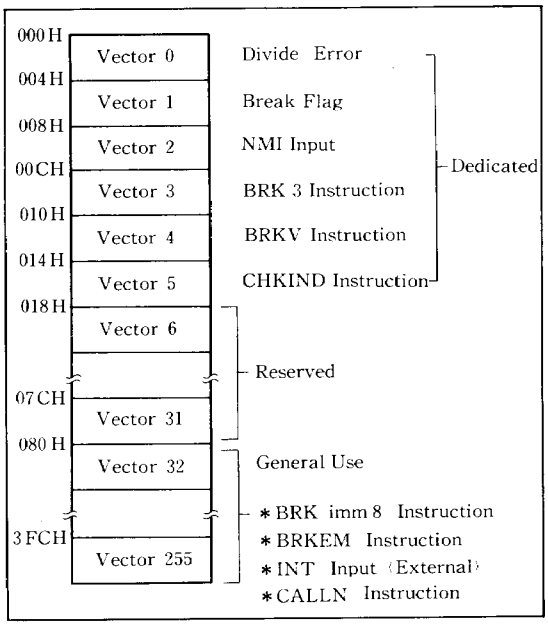


Fig. 8 Interrupt Vector Table

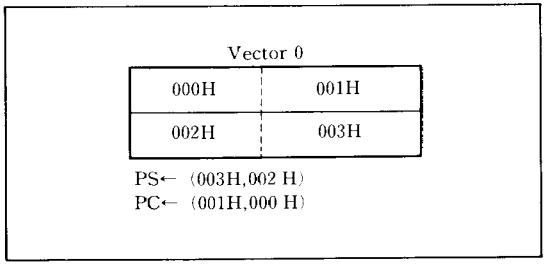


Fig. 9 Interrupt Vector 0

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.

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

■ Standby Function

The LH70108 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.

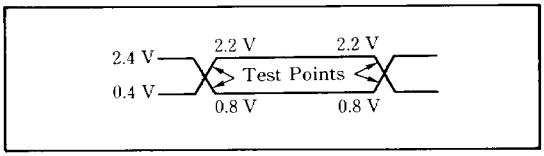
During standby mode, all control outputs are disabled and the address/data bus will be at either high or low levels.

■ I/O Address Reservation

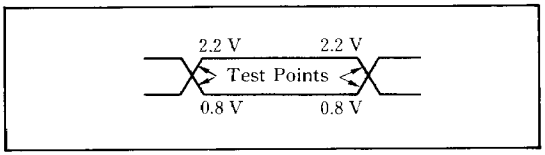
Reserve upper 256 bytes of I/O address (FF00H-FFFH) in case it may be used in future.

■ Timing Diagram

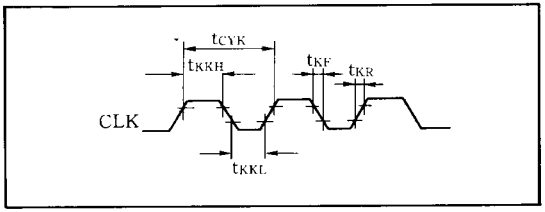
(1) AC test input waveform (Except CLK)



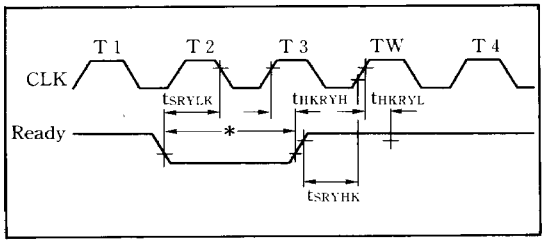
(2) AC output test points



(3) Clock timing

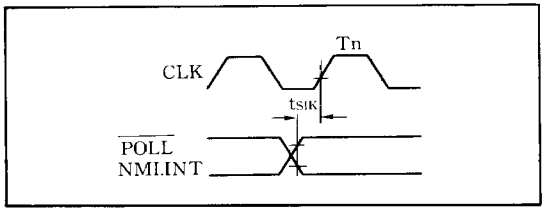


(4) Wait (Ready) timing

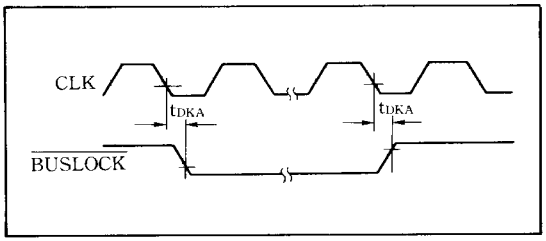


*Read signal must be held at LOW or HIGH during this period.

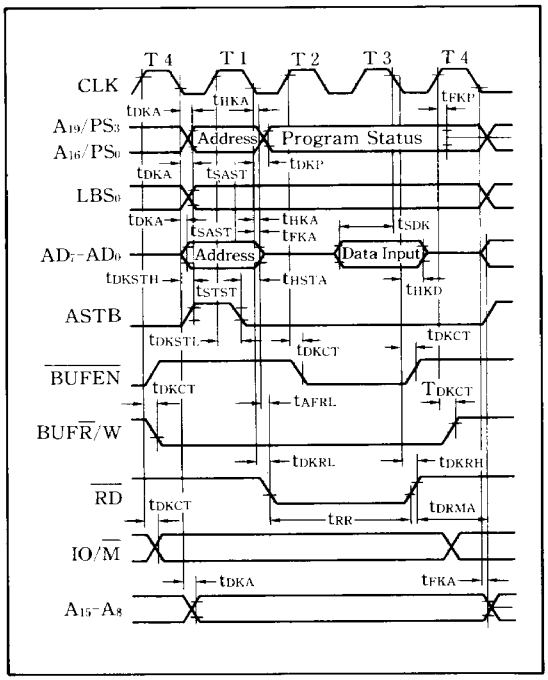
(5) $\overline{\text{POLL}}$, NMI, INT input timing



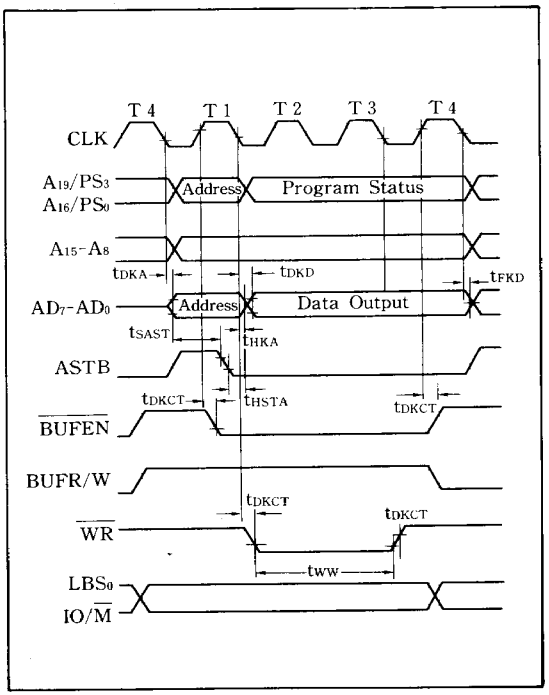
(6) $\overline{\text{BUSLOCK}}$ output timing



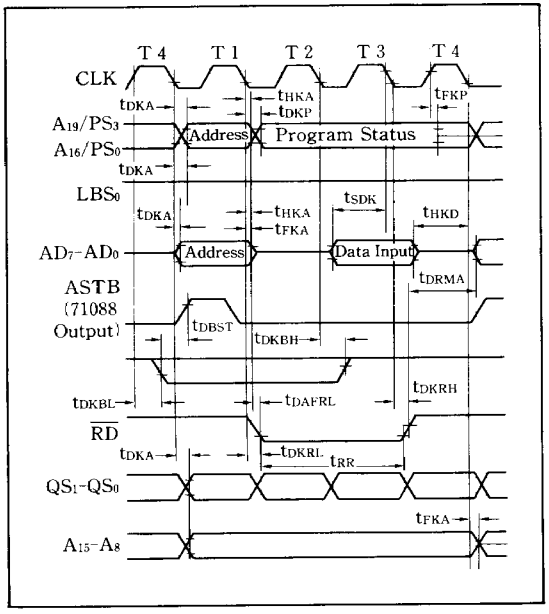
(7) Read timing (small scale)



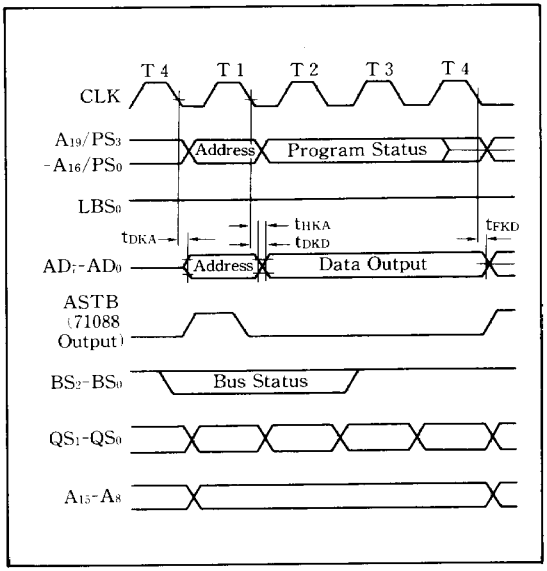
(8) Write timing (small scale)



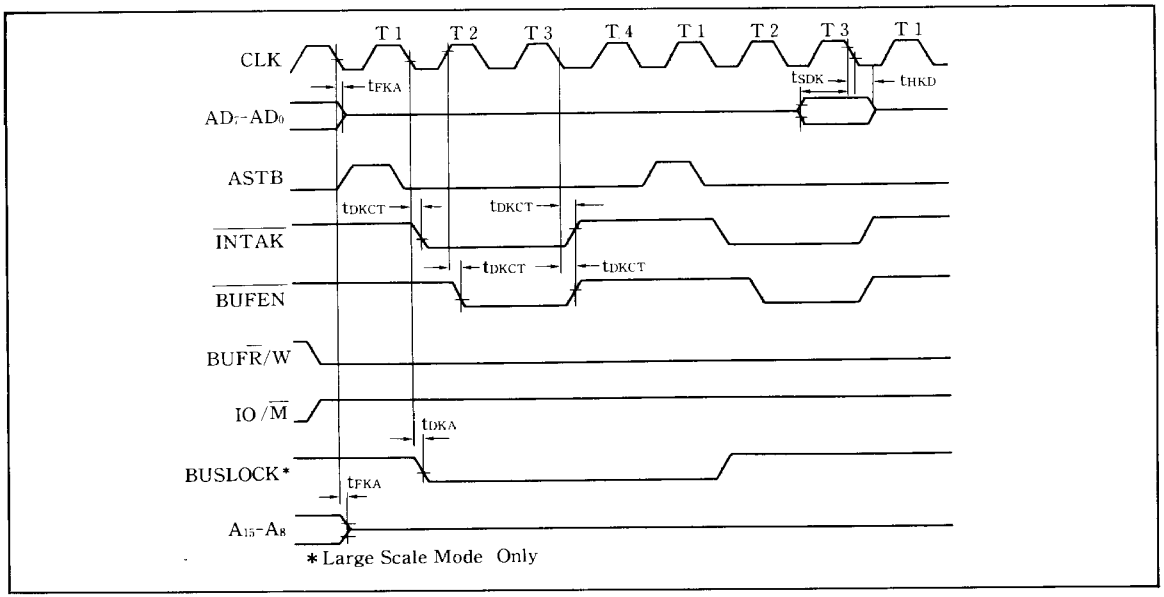
(9) Read timing (large scale)



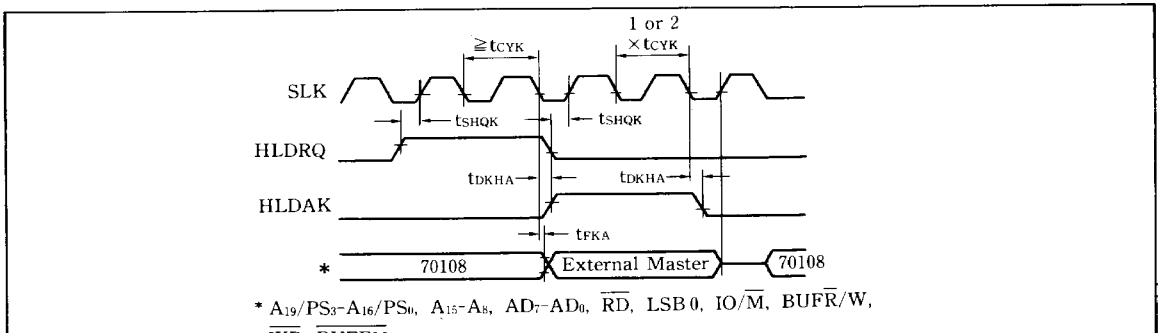
(10) Write timing (large scale)



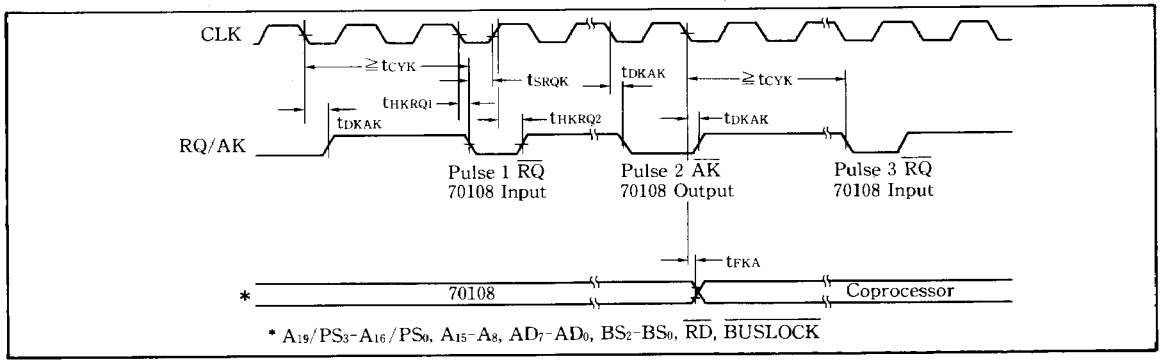
(11) Interrupt acknowledge timing



(12) Hold request/acknowledge timing (small scale)



(13) Bus request/acknowledge timing (large scale)



■ Instruction Set

The following tables briefly describe the LH70108's instruction set.

- ☐ Operation and Operand Type-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 sffect, their operation codes the number of bytes in the instruction, the number of clocks required for execution, and the effect on the LH70108 flags.

Table 1 Operand Types

Identifier	Description
reg	8- or 16-bit general-purpose register
reg8	8-bit general-pourpose 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 whin 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

Table 2 Operation Code Types

Identifier	Description
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

Table 3 Operational Description

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)
BP	Base pointer (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 bit)
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 bit) TA: Temporary resister A (16 bits) TB: Temporary resister B (16 bits) TC: Temporary resister C (16 bits)

Identifier	Description
tmpcy	Temporary carry flag (1 bit)
seg	Immediate segment data (16 bits)
offset	Immediate offset data (16 bit)
←	Transfer direction
+	Addition
−	Subtraction
×	Multiplication
÷	Division
%	Modulo
AND A	Logical product
OR V	Logical sum
XOR ∀	Exclusive logical sum
XXH	Two-digit hexadecimal value
XXXXH	Four-digit hexadecimal value

Table 4 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

Table 5 Memory Addressing

	mod		
men	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_1
01	PS
10	SS
11	DS_0

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 opereands 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.

"No. of Clocks"includes these times:

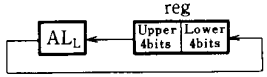
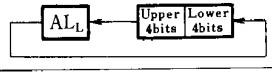
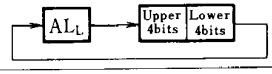
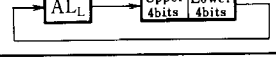
- Decoding
- Effective address generation
- Operand fetch
- Execution

It assumes that the instruction bytes have been prefetched.

Mnemonic	Operand	Operation	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
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	
	dmen, 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	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	14	2-4	
	DS0, reg16, mem32	reg16←(mem32) DS0←(mem32+2)	1 1 0 0 0 1 0 1	mod reg mem	26	2-4	
	DS1, reg16, mem32	reg16←(mem32) DS1←(mem32+2)	1 1 0 0 0 1 0 0	mod reg mem	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 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		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 0		2	1	

Mnemonic	Operand	Operation	Operation Code	No. of Clocks	No. of Bytes	Flags
Repeat Prefixes (cont)						
REP REPE REPZ		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, If 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 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 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−2	1 0 1 0 0 1 0 W	11+8n 11+16n	1	
CMPBK	src-block, dst-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 7+22n	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 7+14n	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 7+13n	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 7+8n	1	

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			AC	CY	V	P	S	Z
Bit Field Instructions																										
INS	reg8, reg8	16-Bit field←AW	0	0	0	0	1	1	1	1	0	0	1	1	0	0	0	1	35-133	3						
			1	1		reg			reg																	
	reg8, imm4	16-Bit field←AW	0	0	0	0	1	1	1	1	0	0	1	1	1	0	0	1	35-133	4						
			1	1	0	0	0		reg																	
Bit Field Transfer Instructions (cont)																										
EXT	reg8, reg8	AW←16-Bit field	0	0	0	0	1	1	1	1	0	0	1	1	0	0	1	1	34-59	3						
			1	1		reg			reg																	
	reg8, imm4	AW←16-Bit filed	0	0	0	0	1	1	1	1	0	0	1	1	1	0	1	1	34-59	4						
			1	1	0	0	0		reg																	
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 Instrctions																										
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 9+16n	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 9+16n	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
	regm 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		reg			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	No. of Clocks	No. of Bytes	Flags
			7 6 5 4 3 2 1 0 / 7 6 5 4 3 2 1 0			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
	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−scrBCD string	0 0 0 0 1 1 1 1 / 0 0 1 0 0 1 1 0 n: number of BCD digits divided by 2	7+19n	2	u x u u u x
ROL4	reg8		0 0 0 0 1 1 1 1 / 1 1 0 0 0 reg	13	3	
	mem8		0 0 0 0 1 1 1 1 / mod 0 0 mem	28	3-5	
ROR4	reg8		0 0 0 0 1 1 1 1 / 1 1 0 0 0 reg	17	3	
	mem8		0 0 0 0 1 1 1 1 / mod 0 0 0 mem	32	3-5	

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			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			16/24	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			16/24	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			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×(mem)8 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×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	1	0	1	reg			41-47	2	u	x	x	u	u	u
	mem16	DW, AW←AW×(mem)8 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			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			38-44	3-5	u	x	x	u	u	u
	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	reg			reg			36-42	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	0	1	mod		reg			mem			46-52	4-6	u	x	x	u	u	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			AC	CY	V	P	S	Z
Unsigned Division Instructions																										
DIVU	reg8	temp←AW When temp÷reg8≦FFH AH←temp%reg8, AL←temp÷reg8 When temp÷reg8>FFH 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	0	1	1	1	1	0	reg			19	2	u	u	u	u	u	u
	mem8	temp←AW When temp÷(mem8)≦FFH AH←temp%(mem8), AL←temp÷(mem8) When temp÷(mem8)>FFH TA←(001H, 000H), TC←(003H, 002H) SP←SP−2,(SP+1, SP)←PSW, 1E←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	0	mod		1	1	0	mem			24	2-4	u	u	u	u	u	u
	reg16	temp←DW, AW When temp÷reg16≦FFFFH DW←temp%reg16, AW←temp÷reg16 When temp÷reg16>FFFFH 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	0	reg			25	2	u	u	u	u	u	u
	mem16	temp←DW, AW When temp÷(mem16)≦FFFFH DW←temp%(mem16),AW←temp÷(mem16) When temp÷(mem16)>FFFFH 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	mod		1	1	0	mem			34	2-4	u	u	u	u	u	u
Signed Division Instructions																										
DIV	reg8	temp←AW When temp÷reg8>0, temp÷reg8≦7FH or temp÷reg8<0, temp÷reg8>0−7FH−1, AH←temp%reg8, AL←temp÷reg8 When temp÷reg8>0, temp÷reg8>7FH or temp÷reg8<0, temp÷temp8≦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	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	Operation Code 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, 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 0	mod 1 1 1 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 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 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	mod 1 1 1 mem	47-52	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		7	1	x x u u u u
ADJ4A		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		3	1	x x u x x x
ADJSB		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, AC←1 When AL>9FH or CY=1 AL←AL−60H, CY←1	0 0 1 0 1 1 1 1		3	1	x x u 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			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 imm8	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			16/24	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
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	1	W									4	2-3	u	0	0	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			AC CY V P S Z
Logical Operation Instructions (cont)						
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						
TEST1	reg8, CL	reg8 bit no. CL=0: Z←1 reg8 bit no. CL=1: Z←0	2nd byte★: 0 0 0 1 0 0 0 0 | 3rd byte★: 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 1 | mod 0 0 0 mem	8	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 0 | mod 0 0 0 mem	12	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	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 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	2nd byte★: 0 0 0 1 1 0 0 1 | 3rd byte★: mod 0 0 0 mem ★Note: First byte=0FH	13	4-6	u 0 0 u u x
NOT1	reg8, CL	reg8 bit no. CL←$\overline{\text{reg8 bit no. CL}}$	2nd byte★: 0 0 0 1 0 1 1 0 | 3rd byte★: 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	13	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. LC←$\overline{\text{(mem16) bit no. CL}}$	0 0 0 1 0 1 1 1 | mod 0 0 0 mem	21	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	14	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←$\overline{\text{(mem16) bit no. imm4}}$	2nd byte★: 0 0 0 1 1 1 1 1 | 3rd byte★: mod 0 0 0 mem ★Note: First byte=0FH	22	4-6	
	CY	CY←$\overline{\text{CY}}$	1 1 1 1 0 1 0 1	2	1	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			ACY V P S Z
Bit Operation Instructions (cont)						
			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	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 2nd byte★ / 3rd byte★ ★Note: First byte=0FH	23	4-6	
	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	
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	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 ★Note: First byte=0FH	0 0 0 1 1 1 0 1 / mod 0 0 0 mem 2nd byte★ / 3rd byte★	22	4-6	
	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 or 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 or (mem), (mem)←(mem)×2 When MSB or (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 or 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 or 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 or (mem), (mem)←(mem)×2, temp←temp−1 n: number of shifts	1 1 0 0 0 0 0 W / mod 1 0 0 mem	19/27+n	3-5	u x u 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			AC	CY	V	P	S	Z
Shift Instructions (cont)																										
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
	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	u	x	x	x	x	x
	reg, CL	temp←CL, while time≠0, repeat this operation, CY←LSB or reg, reg←reg÷2, temp←temp−1	1	1	0	1	0	0	0	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 or (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 or 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		19/27+n	3-5	u	x	u	x	x	x
			n: number of shifts																							
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		19/27+n	3-5	u	x	u	x	x	x
			n: number of shifts																							

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	Flags AC CY V P S Z
Rotation 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 reg	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	mod 0 0 0 mem	19/27+n	3-5	x u
			n: number of shifts				
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 leg←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	mod 0 0 1 mem	19/27+n	2-4	x u
			n: number of shifts				

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			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																							
Rotation Instructions																										
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														No. of Clocks	No. of Bytes	Flags					
			7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0			AC	CY	V	P	S	Z
Rotation 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←1	1	1	0	1	0	0	0	W	1	1	1	0	1	reg			2	2		x			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	x
	reg, CL	temp←CL, while temp≠0, repeat this operatios, 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	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+n	2-4		x			u	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	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			19/27+n	3-5		x			u	u
											n: number of shifts															
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									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			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			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									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			47	2-4						

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			AC	CY	V	P	S	Z
Subroutine Control Instructions (cont)																										
RET		PC←(SP+1, SP), SP←SP+2	1	1	0	0	0	0	1	1									19	1						
	pop-value	PC←(SP+1, SP), SP←SP+2, SP←SP+pop-value	1	1	0	0	0	0	1	0									24	3						
		PC←(SP+1, SP), PS←(SP+3, SP+2) SP←SP+4	1	1	0	0	1	0	1	1									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									32	3						
Stack Manipulation Instructions																										
PUSH	mem16	SP←SP−2 (SP+1, SP)←(mem16)	1	1	1	1	1	1	1	1	mod		1	1	0		mem		26	2-4						
	reg16	SP←SP−2 (SP+1, SP)←reg16	0	1	0	1	0		reg										12	1						
	sreg	SP←SP−2 (SP+1, SP)←sreg	0	0	0	sreg		1	1	0									12	1						
	PSW	SP←SP−2 (SP+1, SP)←PSW	1	0	0	1	1	1	0	0									12	1						
	R	Push registers on the stack	0	1	1	0	0	0	0	0									67	1						
	imm	(SP−1, SP−2)←imm SP←SP−2, when S=1, sign extension	0	1	1	0	1	0	S	0									11/12	2-3						
POP	mem16	(mem16)←(SP+1, SP), SP←SP+2	1	0	0	0	1	1	1	1	mod		0	0	0		mem		25	2-4						
	reg16	reg 16←(SP+1, SP), SP←SP+2	0	1	0	1	1		reg										12	1						
	sreg	sreg←(SP+1, SP) sreg: SS, DS0 DS1 SP←SP+2	0	0	0	sreg		1	1	1									12	1						
	PSW	PSW←(SP+1, SP), SP←SP+2	1	0	0	1	1	1	0	1									12	1	R	R	R	R	R	R
	R	Pop registers from the stack	0	1	1	0	0	0	0	1									75	1						
PREPARE	imm16,imm8	Prepare new stack frame *: imm8<0: 16 imm8<1: 23+16 (imm8−1)	1	1	0	0	1	0	0	0									*	4						
DISPOSE		Dispose of stack frame	1	1	0	0	1	0	0	1									10	1						
Branch Instructions																										
BR	near-label	PC←PC+disp	1	1	1	0	1	0	0	1									13	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	1	0	0	reg		11	2						
	memptr16	PC←(memptr16)	1	1	1	1	1	1	1	1	mod		1	0	0		mem		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		35	2-4						
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+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						

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	Flags AC CY V P S Z
Conditional Branch Instructions (cont)							
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	TA←(00DH, 00CH), TA←(00FH, 00EH) 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 0 0 1 1 0 0		50	1	
	imm8 (≠3)	TA←(4n+1, 4n), TC←(4n+3, 4n+2) n=Imm8 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 0 0 1 1 0 1		50	2	
BRKV		When V=1 TA←(011H, 010H), TC←(013H, 012H) 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 0 0 1 1 1 0		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		39	1	R R R R R R
CHKIND	reg16, mem32	When (mem32)>reg 16 or (mem32+2)<reg16 TA←(4n+1, 4n), TC←(4n+3, 4n+2) n=imm8 SP←SP−2, (SP+1, SP)←PSW, MD←0 MD Bit Write Enable SP←SP−2, (SP+1, SP)←PS, PS←TC SP←SP−2, (SP+1, SP)←PC, PC←TA	0 1 1 0 0 0 1 0	mod reg mem	73-76/ 26	2-4	
BRKEM	imm8	TA←(015H, 014H), TC←(017H, 016H) 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	0 0 0 0 1 1 1 1	1 1 1 1 1 1 1 1	50	3	

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	Flags ACCY V P S Z
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	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	15	2-4	
POLL		Poll and wait n: number of times POLL pin is sampled	1 0 0 1 1 0 1 1		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, MD Bit Write Disable	1 1 1 0 1 1 0 1	1 1 1 1 1 1 0 1	39	2	R R R R R R
CALLN	imm8	TA←(4n+1, 4n), TC←(4n+3, 4n+2) n=imm8 SP←SP−2, (SP+1, SP)←PSW, MD←1 SP←SP−2, (SP+1, SP)←PS, PS←TC SP←SP−2, (SP+1, SP)←PC, PC←TA	1 1 1 0 1 1 0 1	1 1 1 0 1 1 0 1	58	3	