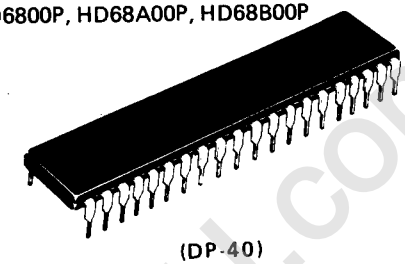
HD6800, HD68A00, HD68B00—
MPU (Micro Processing Unit)

The HD6800 is a monolithic 8-bit microprocessor forming the central control function for Hitachi's HMCS6800 family. Compatible with TTL, the HD6800 as with all HMCS6800 system parts, requires only one 5V power supply, and no external TTL devices for bus interface. The HD68A00 and HD68B00 are high speed versions.

The HD6800 is capable of addressing 65k bytes of memory with its 16-bit address lines. The 8-bit data bus is bi-directional as well as 3-state, making direct memory addressing and multiprocessing applications realizable.

HD6800P, HD68A00P, HD68B00P

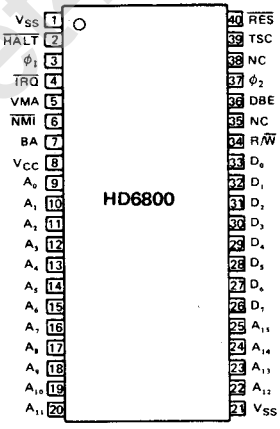


(DP-40)

■ FEATURES

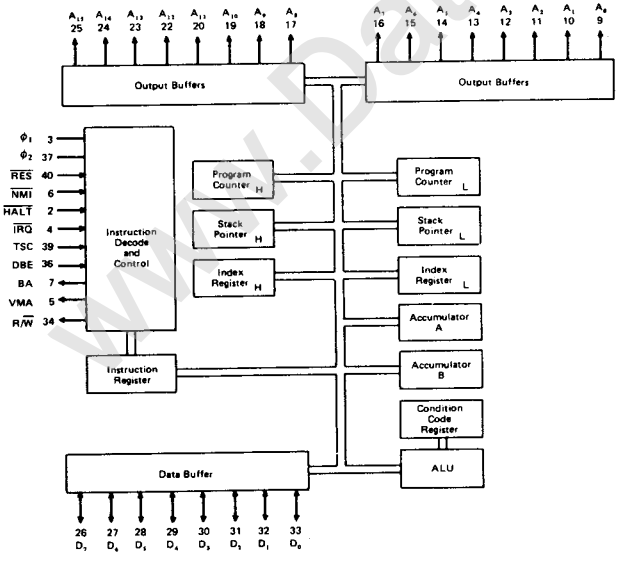
- Versatile 72 Instruction – Variable Length (1~3 Byte)
- Seven Addressing Modes – Direct, Relative, Immediate, Indexed, Extended, Implied and Accumulator
- Variable Length Stack
- Vectored Restart
- Maskable Interrupt
- Separate Non-Maskable Interrupt – Internal Registers Saved in Stack
- Six Internal Registers – Two Accumulators, Index Register, Program Counter, Stack Pointer and Condition Code Register
- Direct Memory Accessing (DMA) and Multiple Processor Capability
- Clock Rates as High as 2.0 MHz (HD6800 ··· 1 MHz, HD68A00 ··· 1.5 MHz, HD68B00 ··· 2.0 MHz)
- Halt and Single Instruction Execution Capability
- Compatible with MC6800, MC68A00 and MC68B00

■ PIN ARRANGEMENT



(Top View)

■ BLOCK DIAGRAM



www.DataSheet4U.com

■ ABSOLUTE MAXIMUM RATINGS

Item	Symbol	Value	Unit
Supply Voltage	V_{CC}*	−0.3 ~ +7.0	V
Input Voltage	V_{in}*	−0.3 ~ +7.0	V
Operating Temperature	T_{opr}	− 20 ~ + 75	°C
Storage Temperature	T_{stg}	− 55 ~ +150	°C

* With respect to V_{SS} (SYSTEM GND)
(NOTE) Permanent LSI damage may occur if maximum rating are exceeded. Normal operation should be under recommended operating conditions. If these conditions are exceeded, it could affect reliability of LSI.

■ RECOMMENDED OPERATING CONDITION

Item	Symbol	min	typ	max	Unit
Supply Voltage	V_{CC}*	4.75	5.0	5.25	V
Input Voltage	V_{IL}*	−0.3	–	0.8	V
	V_{IH}*	2.0	–	V_{CC}	V
Operating Temperature	T_{opr}	−20	25	75	°C

* With respect to V_{SS} (SYSTEM GND)

■ ELECTRICAL CHARACTERISTICS

● DC CHARACTERISTICS (V_{CC} = 5V ± 5%, V_{SS} = 0V, Ta = −20~+75°C, unless otherwise noted.)

Item		Symbol	Test Condition	min	typ*	max	Unit
Input "High" Voltage	Logic**	V_{IH}		2.0	–	V_{CC}	V
Input "Low" Voltage	Logic**	V_{IL}		− 0.3	—	0.8	V
Clock Input "High" Voltage	ϕ_1, ϕ_2	V_{IHC}		V_{CC} − 0.6	–	V_{CC} + 0.3	V
Clock Input "Low" Voltage	ϕ_1, ϕ_2	V_{ILC}		−0.3	–	0.4	V
Output "High" Voltage	D_0~D_7	V_{OH}	I_{OH} = −205μA	2.4	–	–	V
	A_0~A_{15}, R/$\overline{W}$ VMA		I_{OH} = −145μA	2.4	–	–	V
	BA		I_{OH} = −100μA	2.4	–	–	V
Output "Low" Voltage		V_{OL}	I_{OL} = 1.6mA	–	–	0.4	V
Input Leakage Current	Logic***	I_{in}	V_{in} = 0~5.25V, All other pins are connected to GND	-2.5	–	2.5	μA
	ϕ_1, ϕ_2			-100	–	100	μA
Three-State (Off-state) Input Current	D_0~D_7	I_{TSI}	V_{in} = 0.4 ~ 2.4V	-10	–	10	μA
	A_0~A_{15}, R/$\overline{W}$			-100	–	100	μA
Power Dissipation		P_D		–	0.5	1.0	W
Input Capacitance	Logic***	C_{in}	V_{in} = 0V, Ta = 25°C, f = 1 MHz	–	6.5	10	pF
	D_0~D_7			–	10	12.5	pF
	ϕ_1			–	25	35	pF
	ϕ_2			–	45	70	pF
Output Capacitance	A_0~A_{15}, R/$\overline{W}$ VMA, BA	C_{out}	V_{in} = 0V, Ta = 25°C, f = 1 MHz	–	–	12	pF

* Ta = 25°C, V_{CC} = 5V
** All inputs except ϕ_1 and ϕ_2
*** All inputs except ϕ_1, ϕ_2 and D_0~D_7

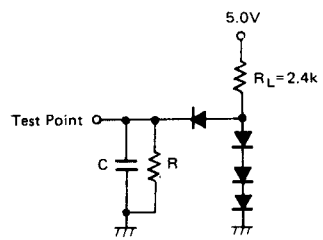
• AC CHARACTERISTICS (V_{CC} = 5V ± 5%, V_{SS} = 0V, Ta = -20~+75°C, unless otherwise noted.)

1. TIMING CHARACTERISTICS OF CLOCK PULSE ϕ_1 and ϕ_2

Item		Symbol	Test Condition	HD6800			HD68A00			HD68B00			Unit
				min	typ	max	min	typ	max	min	typ	max	
Frequency of Operation		f		0.1	–	1.0	0.1	–	1.5	0.1	–	2.0	MHz
Cycle Time		t_{cyc}	Fig. 10	1.000	–	10	0.666	–	10	0.500	–	10	μs
Clock Pulse Width	ϕ_1, ϕ_2	PW_{CH1}, PW_{CH2}	Fig. 10	400	–	4,500	230	–	4,500	180	–	4,500	ns
Rise and Fall Times	ϕ_1, ϕ_2	t_r, t_f	Fig. 10	–	–	100	–	–	100	–	–	100	ns
Delay Time (Clock Internal)		t_d	Fig. 10	0	–	4,500	0	–	4,500	0	–	4,500	ns
Clock "High" Level Time		t_{UT}	Fig. 10	900	–	–	600	–	–	440	–	–	ns

2. READ/WRITE CHARACTERISTICS

Item		Symbol	Test Condition	HD6800			HD68A00			HD68B00			Unit
				min	typ	max	min	typ	max	min	typ	max	
Address Delay Time	C=90pF	t_{AD1}	Fig. 11, Fig. 12	–	–	270	–	–	180	–	–	150	ns
	C=30pF	t_{AD2}	Fig. 11, Fig. 12	–	–	250	–	–	165	–	–	135	ns
Data Setup Time (Read)		t_{DSR}	Fig. 11	100	–	–	60	–	–	40	–	–	ns
Peripheral Read Access Time $t_{acc} = t_{UT} - (t_{AD} + t_{DSR})$		t_{acc}	Fig. 11	530	–	–	360	–	–	250	–	–	ns
Input Data Hold Time		t_H	Fig. 11	10	–	–	10	–	–	10	–	–	ns
Output Data Hold Time		t_H	Fig. 12	20	–	–	20	–	–	20	–	–	ns
Address Hold Time (Address, R/$\overline{W}$, VMA)		t_{AH}	Fig. 11, Fig. 12	10	–	–	10	–	–	10	–	–	ns
Enable "High" Time for DBE Input		t_{EH}	Fig. 12	450	–	–	280	–	–	220	–	–	ns
Data Delay Time (Write)		t_{DDW}	Fig. 12	–	–	225	–	–	200	–	–	160	ns
Data Bus Enable Down Time (During ϕ_1 Up Time)		$t_{\overline{DBE}}$	Fig. 12	150	–	–	120	–	–	75	–	–	ns
Data Bus Enable Delay Time		t_{DBED}	Fig. 12	300	–	–	250	–	–	180	–	–	ns
Data Bus Enable Rise and Fall Times		t_{DBEr} t_{DBEf}	Fig. 12	–	–	25	–	–	25	–	–	25	ns
Processor Control Setup Time		t_{PCS}		200	–	–	140	–	–	110	–	–	ns
Processor Control Rise and Fall Times		t_{PCr} t_{PCf}		–	–	100	–	–	100	–	–	100	ns
Bus Available Delay Time (BA)		t_{BA}		–	–	250	–	–	165	–	–	135	ns
Three-State Delay Time		t_{TSD}		–	–	270	–	–	270	–	–	220	ns



C = 130pF for D_0~D_7
= 90pF for A_0~A_{15}, R/$\overline{W}$, and VMA
= 30pF for BA
R = 11kΩ for D_0~D_7
= 16kΩ for A_0~A_{15}, R/$\overline{W}$ and VMA
= 24kΩ for BA
C includes Stray Capacitance.
All diodes are 1S2074 Ⓗ or equivalent

Figure 1 Bus Timing Test Load

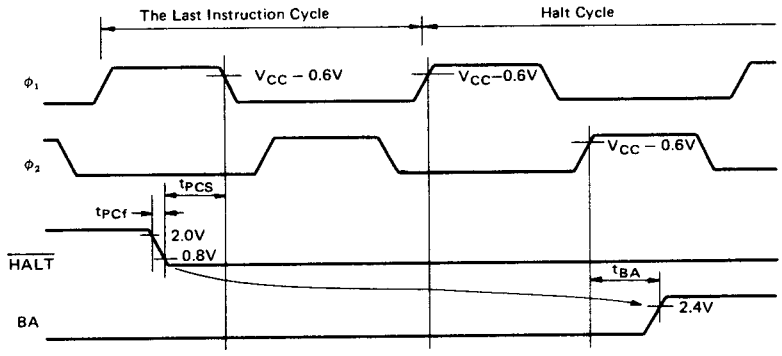


Figure 2 Timing of $\overline{\text{HALT}}$ and BA

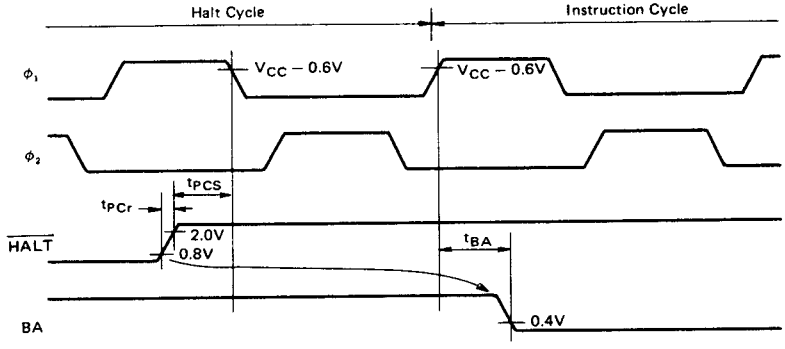


Figure 3 Timing of $\overline{\text{HALT}}$ and BA

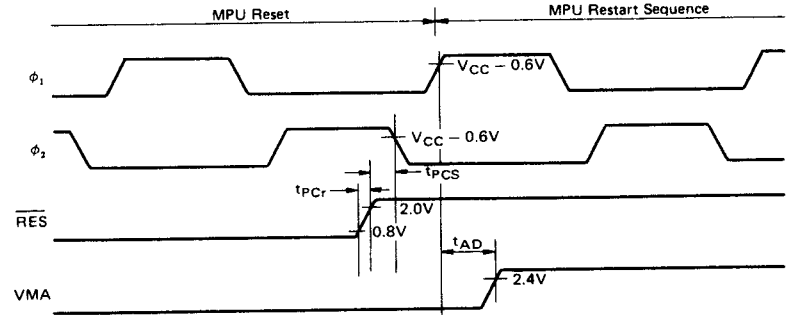


Figure 4 $\overline{\text{RES}}$ and MPU Restart Sequence

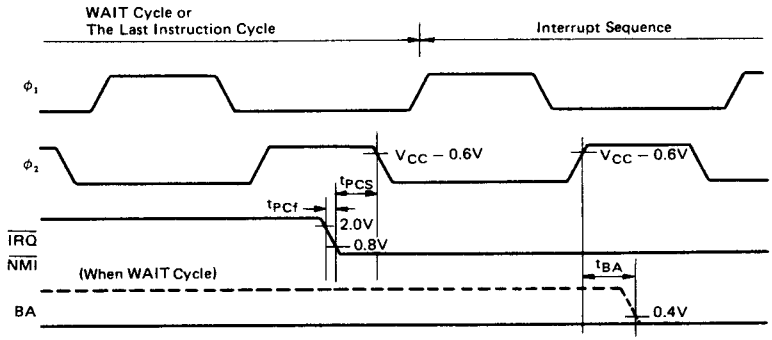


Figure 5 $\overline{IRQ}$ and $\overline{NMI}$ Interrupt Timing

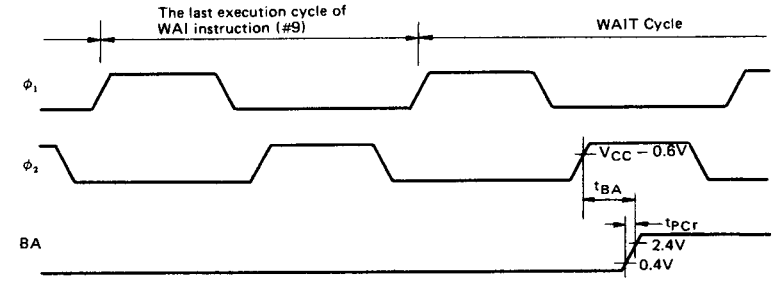


Figure 6 WAI Instruction and BA Timing

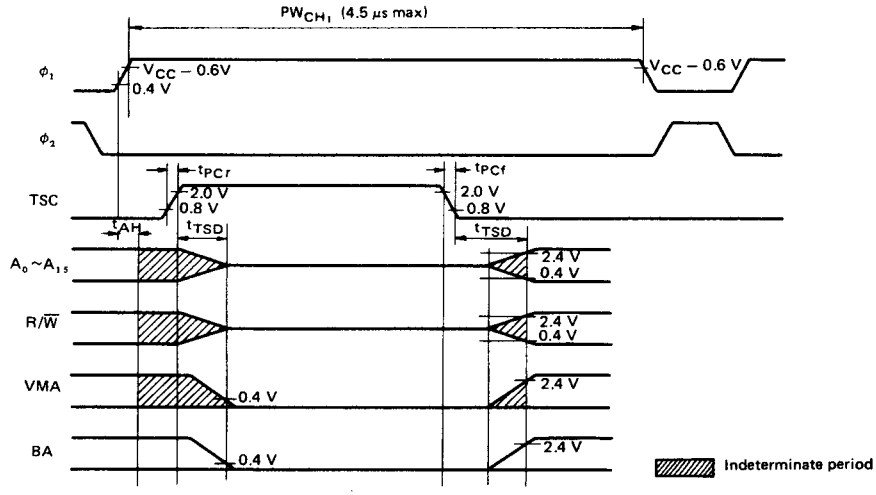


Figure 7 TSC Input and MPU Output

■ MPU REGISTERS

The MPU provides several registers in Fig. 8, which is available for use by the programmer.

Each register is described below.

- **Program Counter (PC)**

The program counter is a two byte (16-bit) register that points to the current program address.

- **Stack Pointer (SP)**

The stack pointer is a two byte register that contains the address of the next available location in an external push-down/pop-up stack. This stack is normally a random access Read/Write memory that may have any location (address) that is convenient. In those applications that require storage of information in the stack when power is lost, the stack must be non-volatile.

- **Index Register (IX)**

The index register is a two byte register that is used to store data or a sixteen bit memory address for the Indexed mode of memory addressing.

- **Accumulators (ACCA, ACCB)**

The MPU contains two 8-bit accumulators that are used to hold operands and results from an arithmetic logic unit (ALU).

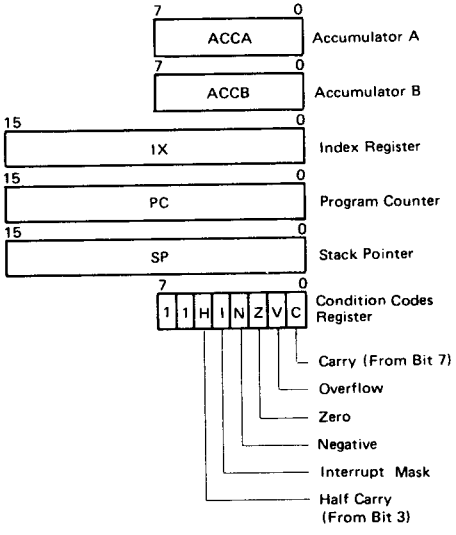


Figure 8 Programming Model of the Microprocessing Unit

- **Condition Code Register (CCR)**

The condition code register indicates the results of an Arithmetic Logic Unit operation: Negative (N), Zero (Z), Overflow (V), Carry from bit 7 (C), and half carry from bit 3(H). These bits of the Condition Code Register are used as testable conditions for the conditional branch instructions. Bit 4 is the interrupt mask bit (I). The unused bits of the Condition Code Register (b6 and b7) are "1". The detail block diagram of the microprossing unit is shown in Fig. 9.

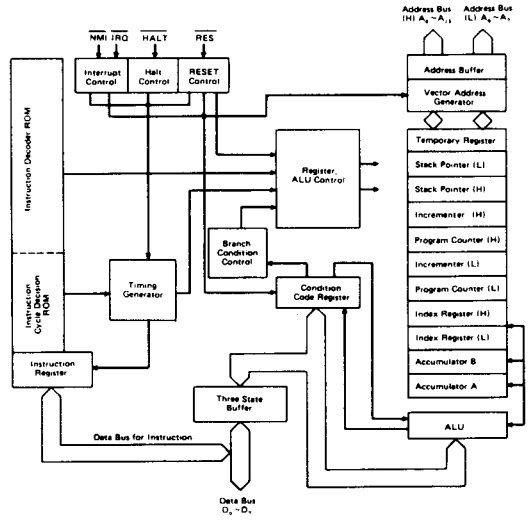


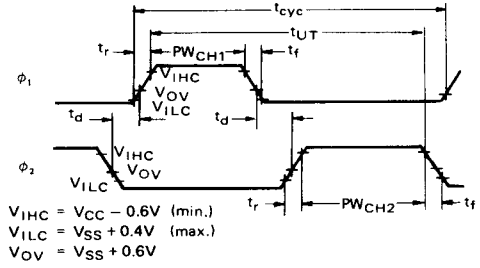
Figure 9 Internal Block Diagram of MPU

■ MPU SIGNAL DESCRIPTION

Proper operations of the MPU requires that certain control and timing signals (Fig. 9) be provided to accomplish specific functions. The functions of pins are explained in this section.

- **Clock (ϕ_2, ϕ_2)**

Two pins are used to provide the clock signals. A two-phase non-overlapping clock is provided as shown in Fig. 10.



$V_{IHC} = V_{CC} - 0.6V$ (min.)
$V_{ILC} = V_{SS} + 0.4V$ (max.)
$V_{OV} = V_{SS} + 0.6V$

Figure 10 Clock Timing Waveform

- **Address Bus ($A_0 \sim A_{15}$)**

Sixteen pins are used for the address bus. The outputs are three-state bus drivers capable of driving one standard TTL load and 90pF. When the output is turned off, it is essentially an open circuit. This permits the MPU to be used in DMA applications. Putting TSC in its high state forces the Address bus to go into the three-state mode.

- **Data Bus ($D_0 \sim D_7$)**

Eight pins are used for the data bus. It is bidirectional, transferring data to and from the memory and peripheral devices. It also has three-state output buffers capable of driving one standard TTL load and 130pF. Data Bus is placed in the three-state mode when DBE is "Low."

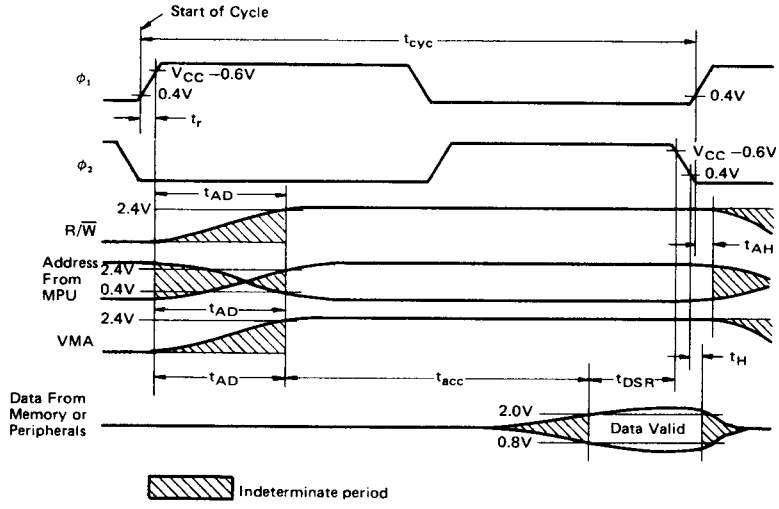


Figure 11 Read from Memory or Peripherals

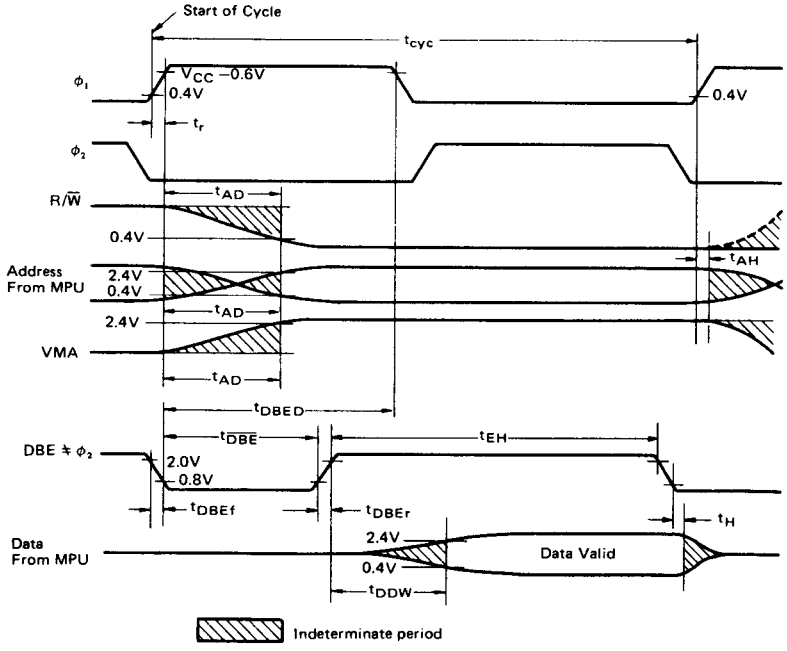


Figure 12 Write to Memory or Peripherals

- **Data Bus Enable (DBE)**

This input is the three-state control signal for the MPU data bus and will enable the bus drivers when in the "High" state; will make the bus driver off when in the "Low" state. This input is TTL compatible; however in normal operation, it would be driven by ϕ_2 clock. During an MPU read cycle, the data bus drivers will be disabled internally. When it is desired that another device control the data bus such as in Direct Memory Access (DMA) applications, DBE should be held "Low."

If additional data setup or hold time is required on an MPU write, the DBE down time can be decreased as shown in Fig. 13 (DBE $\downarrow \phi_2$). The minimum down time for DBE is $t_{\overline{DBE}}$ as shown and must occur within ϕ_1 up time. As for the characteristical values in Fig. 12, refer to the table of electrical characteristics.

- **Bus Available (BA)**

The BA signal will normally be in the "Low" state. When activated, it will go to the "High" state indicating that the microprocessor has stopped and that the address bus is available. This will occur if the $\overline{HALT}$ line is in the "Low" state or the processor is in the WAIT state as a result of the execution of a WAIT instruction. At such time, all three-state output drivers will go to their off state and other outputs to their normally inactive level. The processor is removed from the WAIT state by the occurrence of a maskable (mask bit I = 0) or nonmaskable interrupt. This output is capable of driving one standard TTL load and 30pF. If TSC is in the "High" state, Bus Available will be "Low".

- **Read/Write (R/$\overline{W}$)**

This TTL compatible output signals the peripherals and memory devices whether the MPU is in a Read ("High") or Write ("Low") state. The normal standby state of this signal is Read ("High"). Three-State Control going "High" will turn R/$\overline{W}$ to the off (high impedance) state. Also, when the processor is halted, it will be in the off state. This output is capable of driving one standard TTL load and 90pF.

- **Reset ($\overline{RES}$)**

The $\overline{RES}$ input is used to reset and start the MPU from a power down condition resulting from a power failure or initial start-up of the processor. This input can also be used to re-initialize the machine at any time after start-up.

If a "High" level is detected in this input, this will signal the MPU to begin the reset sequence. During the reset sequence, the contents of the last two locations (FFFE, FFFF) in memory will be loaded into the Program Counter to point to the beginning of the reset routine. During the reset routine, the interrupt mask bit is set and must be cleared under program control before the MPU can be interrupted by $\overline{IRQ}$. While $\overline{RES}$ is "Low" (assuming a minimum of 8 clock cycles have occured) the MPU output signals will be in the following states; VMA = "Low", BA = "Low", Data Bus = high impedance, R/$\overline{W}$ = "High" (read state), and the Address Bus will contain the reset address FFFE. Fig. 13 illustrates a power up sequence using the Reset control line. After the power supply reaches 4.75V, a minimum of eight clock cycles are required for the processor to stabilize in preparation for restarting. During these eight cycles, VMA will be in an indeterminate state so any devices that are enabled by VMA which could accept a false write during this time (such as a battery-backed RAM) must be disabled until VMA is forced "Low" after eight cycles. $\overline{RES}$ can go "High" asynchronously with the system clock any time after the eighth cycle.

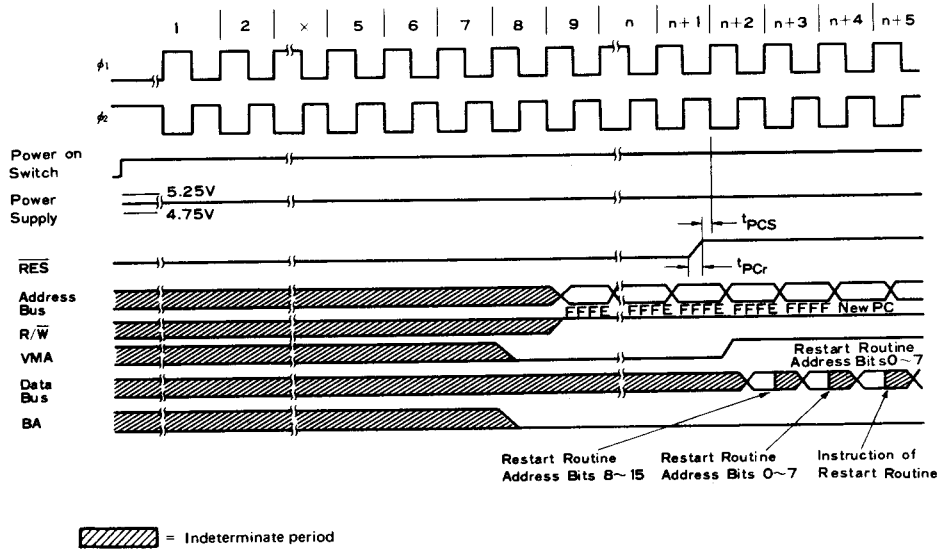


Figure 13 $\overline{RES}$ Timing

The Reset control line may also be used to reinitialize the MPU system at any time during its operation. This is accomplished by pulsing $\overline{RES}$ "Low" for the duration of a minimum of three complete ϕ_2 cycles. The $\overline{RES}$ pulse can be completely asynchronous with the MPU system clock and will be recognized during ϕ_2 if setup time t_{PCS} is met.

- **Interrupt Request ($\overline{IRQ}$)**

This level sensitive input requests that an interrupt sequence be generated within the machine. The processor will wait until it completes the current instruction that is being executed before it recognizes the request. If the interrupt mask bit in the Condition Code Register is not set, the machine will begin an interrupt sequence. The Program Counter, Index Register, Accumulators, and Condition Code Register are stored away on the stack.

Next the MPU will respond to the interrupt request by setting the interrupt mask bit "1" so that no further interrupts may occur. At the end of the cycle, a 16-bit address will be loaded that points to a vectoring address which is located in memory locations FFF8 and FFF9. An address loaded at these locations causes the MPU to branch to an interrupt routine in memory. Interrupt timing is shown in Fig. 14.

The $\overline{HALT}$ line must be in the "High" state for interrupts to be serviced. Interrupts will be latched internally while $\overline{HALT}$ is "Low". The $\overline{IRQ}$ has a high impedance pullup device internal to the chip; however a 3kΩ external resistor to V_{CC} should be used for wire-OR and optimum control of interrupts.

- **Non-Maskable Interrupt ($\overline{NMI}$) and Wait for Interrupt (WAI)**

The MPU is capable of handling two types of interrupts: maskable ($\overline{IRQ}$) as described earlier, and non-maskable ($\overline{NMI}$). $\overline{IRQ}$ is maskable by the interrupt mask in the Condition Code Register while $\overline{NMI}$ is not maskable. The handling of these interrupts by the MPU is the same except that each has its own vector address. The behavior of the MPU when interrupted is shown in Fig. 14 which details the MPU response to an interrupt while the MPU is executing the control program. The interrupt shown could be either $\overline{IRQ}$ or $\overline{NMI}$ and can be asynchronous with respect to ϕ_2. The interrupt is shown going "Low" at time t_{PCS} in cycle #0 which precedes the first cycle of an instruction (OP code fetch). This instruction is not executed but instead the Program Counter (PC), Index Register (IX), Accumulators (ACCX), and the Condition Code Register (CCR) are pushed onto the stack.

The Interrupt Mask bit is set to prevent further interrupts. The address of the interrupt service routine is then fetched from FFFC, FFFD for an $\overline{NMI}$ interrupt and from FFF8, FFF9 for an $\overline{IRQ}$ interrupt. Upon completion of the interrupt service routine, the execution of RTI will pull the PC, IX, ACCX, and CCR off of the stack; the Interrupt Mask bit is restored to its condition prior to interrupts. Fig. 15 is a similar interrupt sequence, except in this case, a WAIT instruction has been executed in preparation for the interrupt. This technique speeds up the MPU's response to the interrupt because the stacking of

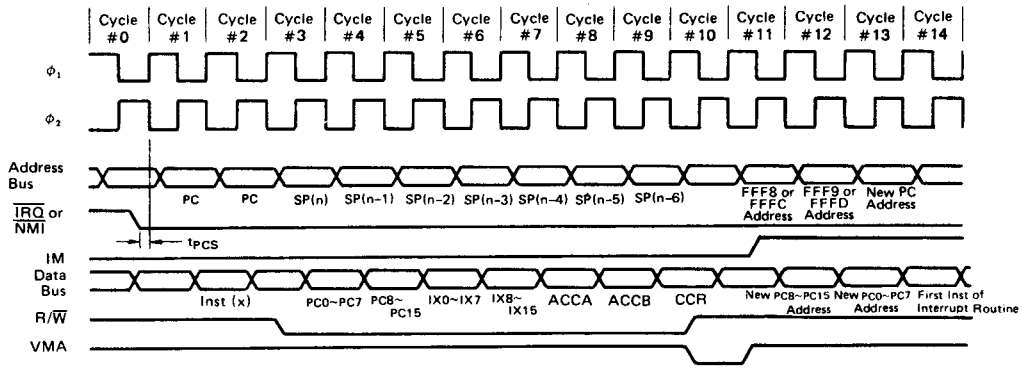
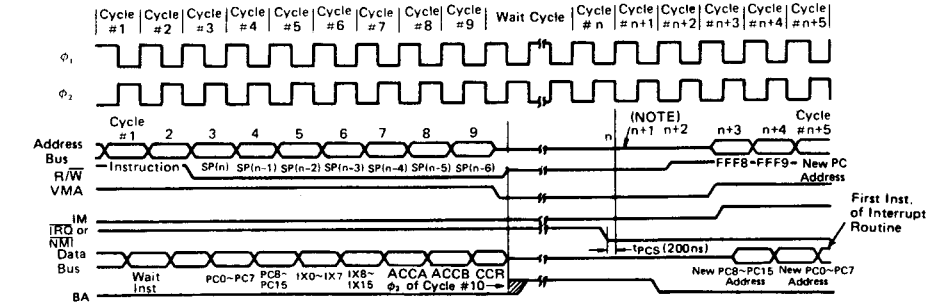


Figure 14 Interrupt Timing



(NOTE) Midrange waveform indicates high impedance state.

Figure 15 WAI Instruction Timing

the PC, IX, ACCX, and the CCR is already done.

While the MPU is waiting for the interrupt, Bus Available will go "High" indicating the following states of the control lines: VMA is "Low", and the Address Bus, R/$\overline{W}$ and Data Bus are all in the high impedance state. After the interrupt occurs, it is serviced as previously described.

Table 1 Memory Map for Interrupt Vectors

Vector		Description
MS	LS	
FFFE	FFFF	Restart
FFFC	FFFD	Non-maskable Interrupt
FFFA	FFFB	Software Interrupt
FFF8	FFF9	Interrupt Request

Refer to Figure 18 for program flow for Interrupts.

- **Three State Control (TSC)**

When the Three State Control (TSC) line is "High" level, the Address Bus and the R/$\overline{W}$ line are placed in a high impedance State. VMA and BA are forced "Low" when TSC = "High" to prevent false reads or writes on any device enabled by VMA. It is necessary to delay program execution while TSC is held "High". This is done by insuring that no transitions of ϕ_1 (or ϕ_2) occur during this period. (Logic levels of the clocks are irrelevant so long as they do not change.)

Since the MPU is a dynamic device, the ϕ_1 clock can be stopped for a maximum time PW_{CH1} without destroying data within the MPU. TSC then can be used in a short Direct Memory Access (DMA) application.

Fig. 16 shows the effect of TSC on the MPU. The Address Bus and R/$\overline{W}$ line will reach the high impedance state at t_{TSD} (three-state delay), with VMA being forced "Low". In this example, the Data Bus is also in the high impedance state while ϕ_2 is being held "Low" since DBE=ϕ_2. At this point in time, a DMA transfer could occur on cycles #3 and #4. When TSC is returned "Low," the MPU address and R/$\overline{W}$ lines return to the bus. Because it is too late in cycle #5 to access memory, this cycle is dead and used for synchronization. Program execution resumes in cycle #6.

- **Valid Memory Address (VMA)**

This output indicates to peripheral devices that there is a valid address on the address bus. In normal operation, this signal should be utilized for enabling peripheral interfaces such as the PIA and ACIA. This signal is not three-state. One standard TTL load and 90pF may be directly driven by this active "High" signal.

- **Halt ($\overline{HALT}$)**

When this input is in the "Low" state, all activity in the machine will be halted. This input is level sensitive.

The $\overline{HALT}$ line provides an input to the MPU to allow control or program execution by an outside source. If $\overline{HALT}$ is "High", the MPU will execute the instructions; if it is "Low", the MPU will go to a halted or idle mode. A response signal, Bus Available (BA) provides an indication of the current MPU status. When BA is "Low", the MPU is in the process of executing the control program; if BA is "High", the MPU has halted and all internal activity has stopped.

When BA is "High", the Address Bus, Data Bus, and R/$\overline{W}$ line will be in a high impedance state, effectively removing the MPU from the system bus. VMA is forced "Low" so that the floating system bus will not activate any device on the bus that is enabled by VMA.

While the MPU is halted, all program activity is stopped, and if either an $\overline{NMI}$ or $\overline{IRQ}$ interrupt occurs, it will be latched into the MPU and acted on as soon as the MPU is taken out of the halted mode. If a $\overline{RES}$ command occurs while the MPU is halted, the following states occur: VMA = "Low", BA = "Low", Data Bus = high impedance, R/$\overline{W}$ = "High" (read state), and the Address Bus will contain address FFFE as long as $\overline{RES}$ is "Low". As soon as the $\overline{RES}$ line goes "High", the MPU will go to locations FFFE and FFFF for the address of the reset routine.

Fig. 18 shows the timing relationships involved when halting the MPU. The instruction illustrated is a one byte, 2 cycle instruction such as CLRA. When $\overline{HALT}$ goes "Low", the MPU will halt after completing execution of the current instruction. The transition of $\overline{HALT}$ must occur t_{PCS} before the trailing edge of ϕ_1 of the last cycle of an instruction (point A of Fig. 18). $\overline{HALT}$ must not go "Low" any time later than the minimum t_{PCS} specified.

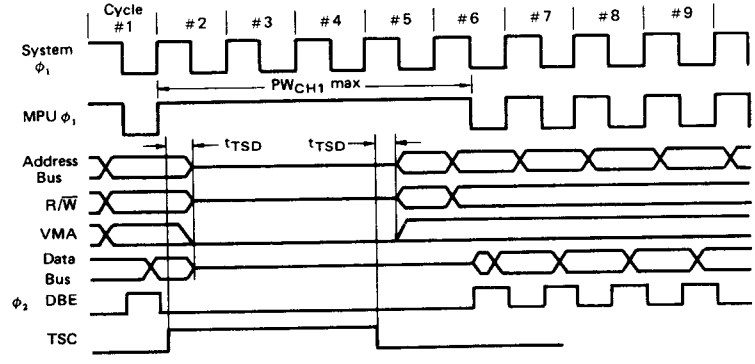


Figure 16 TSC Control Timing

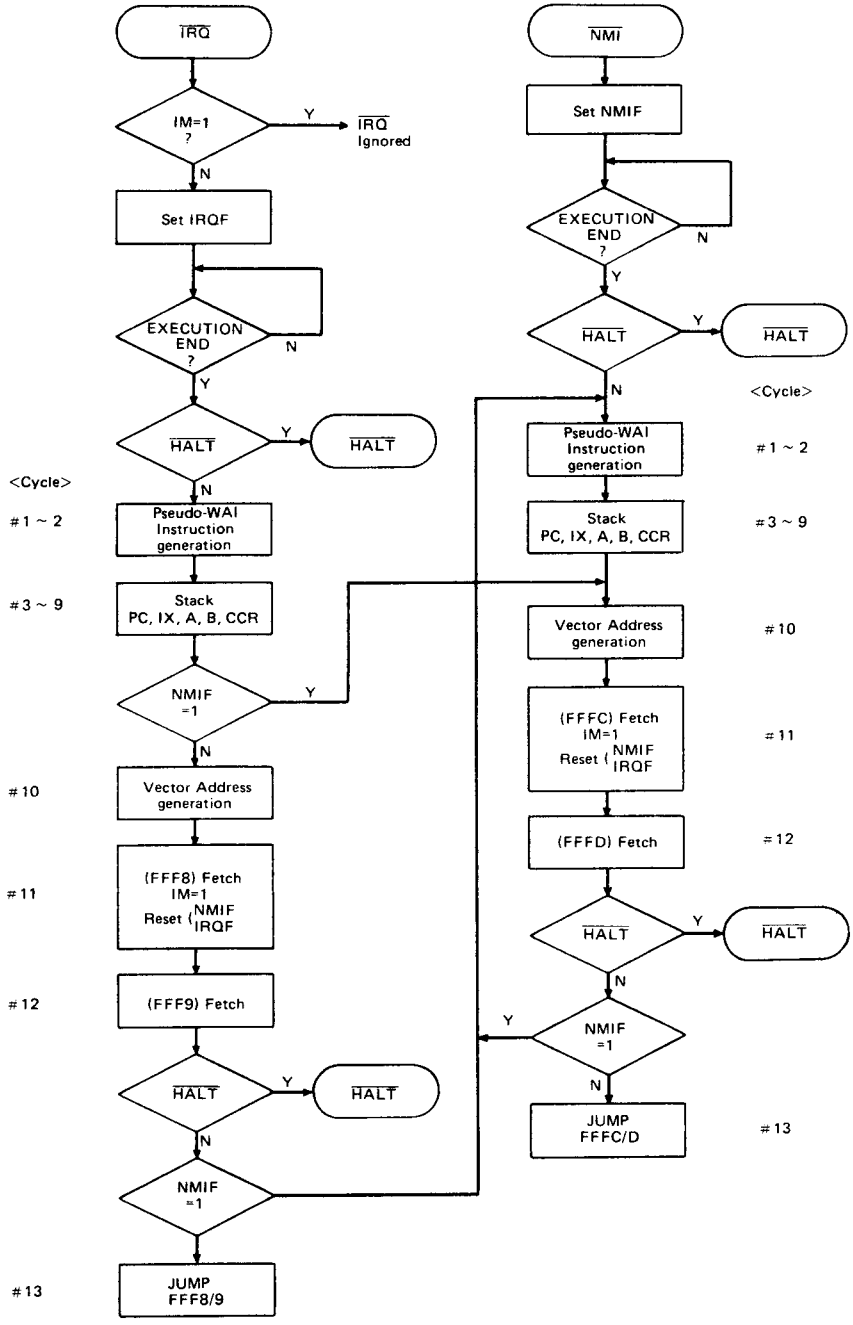
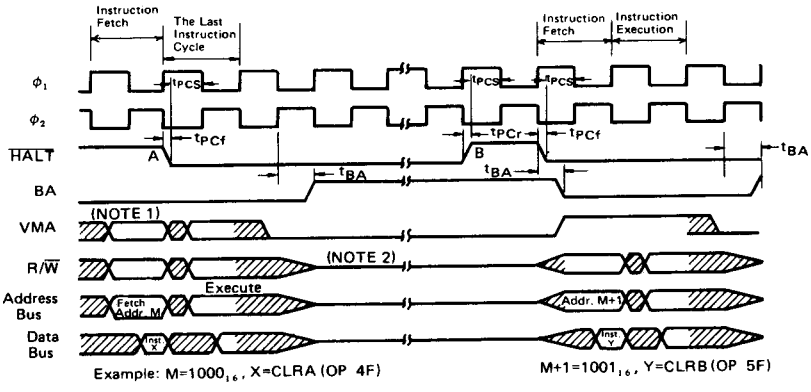


Figure 17 MPU Interrupt Flow Chart



(NOTE) 1. Oblique lines indicate indeterminate range of data.
2. Midrange waveform indicates high impedance, state.

Figure 18 $\overline{\text{HALT}}$ and Single Instruction Execution for System Dubug

Table 2 Operation States of MPU and Signal Outputs (Except the Execution of Instruction)

Signals	Halt state	Reset state	Halt and Reset state	WAI state	TSC state
BA	"H"	"L"	"L"	"H"	"L"
VMA	"L"	"L"	"L"	"L"	"L"
R/$\overline{W}$	"T"	"H"	"H"	"T"	"T"
$A_0 \sim A_{15}$	"T"	$(FFFE)_{16}$	$(FFFE)_{16}$	"T"	"T"
$D_0 \sim D_7$	"T"	"T"	"T"	"T"	–

"T" indicates high impedance state.

The fetch of the OP code by the MPU is the first cycle of the instruction. If $\overline{\text{HALT}}$ had not been "Low" at Point A but went "Low" during ϕ_2 of the cycle, the MPU would have halted after completion of the following instruciton. BA will go "High" by time t_{BA} (bus available delay time) after the last instruction cycle. At this point in time, VMA is "Low" and R/$\overline{W}$, Address Bus, and the Data Bus are in the high impedance state.

To debug programs it is advantageous to step through programs instruction by instruction. To do this, $\overline{\text{HALT}}$ must be brought "High" for one MPU cycle and then returned "Low" as shown at point B of Fig. 18. Again, the transitions of $\overline{\text{HALT}}$ must occur t_{PCS} before the trailing edge of ϕ_1. BA will go "Low" at t_{BA} after the leading edge of the next ϕ_1, indicating that the Address Bus, Data Bus, VMA and R/$\overline{W}$ lines are back on the bus. A single byte, 2 cycle instruction such as LSR is used for this example also. During the first cycle, the instruction Y is fetched from address M+1. BA returns "High" at t_{BA} on the last cycle of the instruction indicating the MPU is off the bus, if instruction Y had been three cycles, the width of the BA "Low" time would have been increased by one cycle.

Table 2 shows the relation between the state of MPU and signal outputs.

■ MPU INSTRUCTION SET

This Section will provide a brief introduction and discuss their use in developing HD6800 MPU control programs. The HD6800 MPU has a set of 72 different executable source instructions. Included are binary and decimal arithmetic, logical, shift, rotate, load, store, conditional or unconditional branch, interrupt and stack manipulation instructions.

Each of the 72 executable instructions of the source language assembles into 1 to 3 bytes of machine code. The number of bytes depends on the particular instruction and on the addressing mode. (The addressing modes which are available for use with the various executive instructions are discussed later.)

The coding of the first (or only) byte corresponding to an executable instruction is sufficient to identify the instruction and the addressing mode. The hexadecimal equivalents of the binary codes, which result from the translation of the 72 instructions in all valid modes of addressing, are shown in Table 3. There are 197 valid machine codes, 59 of the 256 possible codes being unassigned.

When an instruction translates into two or three bytes of code, the second byte, or second and third bytes contain(s) an operand, an address, or information from which an address is obtained during execution.

Microprocessor instructions are often devided into three general classifications; (1) memory reference, so called because they operate on specific memory locations; (2) operating instructions that function without needing a memory reference; (3) I/O instructions for transferring data between the microprocessor and peripheral devices.

In many instances, the HD6800 MPU performs the same operation on both its internal accumulators and the external memory locations. In addition, the HD6800 MPU allow the MPU to treat peripheral devices exactly like other memory locations, hence, no I/O instructions as such are required. Because of these features, other classifications are more suitable for introducing the HD6800's instruction set: (1) Accumulator and memory operations; (2) Program control operations; (3) Condition Code Register operations.

For Accumulator and Memory Operations, refer to Table 4.

Table 3 Hexadecimal Values of Machine Codes

MSB \ LSB	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	•	NOP (IMP)	•	•	•	•	TAP (IMP)	TPA (IMP)	INX (IMP)	DEX (IMP)	CLV (IMP)	SEV (IMP)	CLC (IMP)	SEC (IMP)	CLI (IMP)	SEI (IMP)
1	SBA (A, B)	CBA (A, B)	•	•	•	•	TAB (IMP)	TBA (IMP)	•	DAA (IMP)	•	ABA (IMP)	•	•	•	•
2	BRA (REL)	•	BHI (REL)	BLS (REL)	BCC (REL)	BCS (REL)	BNE (REL)	BEQ (REL)	BVC (REL)	BVS (REL)	BPL (REL)	BMI (REL)	BGE (REL)	BLT (REL)	BGT (REL)	BLE (REL)
3	TSX (IMP)	INS (IMP)	PUL (A)	PUL (B)	DES (IMP)	TXS (IMP)	PSH (A)	PSH (B)	•	RTS (IMP)	•	RTI (IMP)	•	•	WAI (IMP)	SWI (IMP)
4	NEG (A)	•	•	COM (A)	LSR (A)	•	ROR (A)	ASR (A)	ASL (A)	ROL (A)	DEC (A)	•	INC (A)	TST (A)	•	CLR (A)
5	NEG (B)	•	•	COM (B)	LSR (B)	•	ROR (B)	ASR (B)	ASL (B)	ROL (B)	DEC (B)	•	INC (B)	TST (B)	•	CLR (B)
6	NEG (IND)	•	•	COM (IND)	LSR (IND)	•	ROR (IND)	ASR (IND)	ASL (IND)	ROL (IND)	DEC (IND)	•	INC (IND)	TST (IND)	JMP (IND)	CLR (IND)
7	NEG (EXT)	•	•	COM (EXT)	LSR (EXT)	•	ROR (EXT)	ASR (EXT)	ASL (EXT)	ROL (EXT)	DEC (EXT)	•	INC (EXT)	TST (EXT)	JMP (EXT)	CLR (EXT)
8	SUB (IMM)(A)	CMP (IMM)(A)	SBC (IMM)(A)	•	AND (IMM)(A)	BIT (IMM)(A)	LDA (IMM)(A)	•	EOR (IMM)(A)	ADC (IMM)(A)	ORA (IMM)(A)	ADD (IMM)(A)	CPX (IMM)(A)	BSR (REL)	LDS (IMM)	•
9	SUB (DIR)(A)	CMP (DIR)(A)	SBC (DIR)(A)	•	AND (DIR)(A)	BIT (DIR)(A)	LDA (DIR)(A)	STA (DIR)(A)	EOR (DIR)(A)	ADC (DIR)(A)	ORA (DIR)(A)	ADD (DIR)(A)	CPX (DIR)(A)	•	LDS (DIR)	STS (DIR)
A	SUB (IND)(A)	CMP (IND)(A)	SBC (IND)(A)	•	AND (IND)(A)	BIT (IND)(A)	LDA (IND)(A)	STA (IND)(A)	EOR (IND)(A)	ADC (IND)(A)	ORA (IND)(A)	ADD (IND)(A)	CPX (IND)(A)	JSR (IND)	LDS (IND)	STS (IND)
B	SUB (EXT)(A)	CMP (EXT)(A)	SBC (EXT)(A)	•	AND (EXT)(A)	BIT (EXT)(A)	LDA (EXT)(A)	STA (EXT)(A)	EOR (EXT)(A)	ADC (EXT)(A)	ORA (EXT)(A)	ADD (EXT)(A)	CPX (EXT)(A)	JSR (EXT)	LDS (EXT)	STS (EXT)
C	SUB (IMM)(B)	CMP (IMM)(B)	SBC (IMM)(B)	•	AND (IMM)(B)	BIT (IMM)(B)	LDA (IMM)(B)	•	EOR (IMM)(B)	ADC (IMM)(B)	ORA (IMM)(B)	ADD (IMM)(B)	•	•	LDX (IMM)	•
D	SUB (DIR)(B)	CMP (DIR)(B)	SBC (DIR)(B)	•	AND (DIR)(B)	BIT (DIR)(B)	LDA (DIR)(B)	STA (DIR)(B)	EOR (DIR)(B)	ADC (DIR)(B)	ORA (DIR)(B)	ADD (DIR)(B)	•	•	LDX (DIR)(B)	STX (DIR)(B)
E	SUB (IND)(B)	CMP (IND)(B)	SBC (IND)(B)	•	AND (IND)(B)	BIT (IND)(B)	LDA (IND)(B)	STA (IND)(B)	EOR (IND)(B)	ADC (IND)(B)	ORA (IND)(B)	ADD (IND)(B)	•	•	LDX (IND)	STX (IND)
F	SUB (EXT)(B)	CMP (EXT)(B)	SBC (EXT)(B)	•	AND (EXT)(B)	BIT (EXT)(B)	LDA (EXT)(B)	STA (EXT)(B)	EOR (EXT)(B)	ADC (EXT)(B)	ORA (EXT)(B)	ADD (EXT)(B)	•	•	LDX (EXT)	STX (EXT)

DIR = Direct Addressing Mode
EXT = Extended Addressing Mode
IMM = Immediate Addressing Mode
IND = Index Addressing Mode
IMP = Implied Addressing Mode
REL = Relative Addressing Mode
A = Accumulator A
B = Accumulator B

Table 4 Accumulator and Memory Operations

Operation	Mnemonic	IMMED OP	IMMED ~	IMMED #	DIRECT OP	DIRECT ~	DIRECT #	INDEX OP	INDEX ~	INDEX #	EXTND OP	EXTND ~	EXTND #	IMPLIED OP	IMPLIED ~	IMPLIED #	Boolean/ Arithmetic Operation	5 H	4 I	3 N	2 Z	1 V	0 C
Add	ADDA	8B	2	2	9B	3	2	AB	5	2	BB	4	3				A + M → A	↕	•	↕	↕	↕	↕
	ADDB	CB	2	2	DB	3	2	EB	5	2	FB	4	3				B + M → B	↕	•	↕	↕	↕	↕
Add Acmltrs	ABA													1B	2	1	A + B → A	↕	•	↕	↕	↕	↕
Add with Carry	ADCA	89	2	2	99	3	2	A9	5	2	B9	4	3				A + M + C → A	↕	•	↕	↕	↕	↕
	ADCB	C9	2	2	D9	3	2	E9	5	2	F9	4	3				B + M + C → B	↕	•	↕	↕	↕	↕
And	ANDA	84	2	2	94	3	2	A4	5	2	B4	4	3				A • M → A	•	•	↕	↕	R	•
	ANDB	C4	2	2	D4	3	2	E4	5	2	F4	4	3				B • M → B	•	•	↕	↕	R	•
Bit Test	BITA	85	2	2	95	3	2	A5	5	2	B5	4	3				A • M	•	•	↕	↕	R	•
	BITB	C5	2	2	D5	3	2	E5	5	2	F5	4	3				B • M	•	•	↕	↕	R	•
Clear	CLR							6F	7	2	7F	6	3				00 → M	•	•	R	S	R	R
	CLRA													4F	2	1	00 → A	•	•	R	S	R	R
	CLRB													5F	2	1	00 → B	•	•	R	S	R	R
Compare	CMPA	81	2	2	91	3	2	A1	5	2	B1	4	3				A − M	•	•	↕	↕	↕	↕
	CMPB	C1	2	2	D1	3	2	E1	5	2	F1	4	3				B − M	•	•	↕	↕	↕	↕
Compare Acmltrs	CBA													11	2	1	A − B	•	•	↕	↕	↕	↕
Complement, 1's	COM							63	7	2	73	6	3				M̄ → M	•	•	↕	↕	R	S
	COMA													43	2	1	Ā → A	•	•	↕	↕	R	S
	COMB													53	2	1	B̄ → B	•	•	↕	↕	R	S
Complement, 2's (Negate)	NEG							60	7	2	70	6	3				00 − M → M	•	•	↕	↕	①	②
	NEGA													40	2	1	00 − A → A	•	•	↕	↕	①	②
	NEGB													50	2	1	00 − B → B	•	•	↕	↕	①	②
Decimal Adjust, A	DAA													19	2	1	Converts Binary Add of BCD Characters into BCD Format	•	•	↕	↕	↕	③
Decrement	DEC							6A	7	2	7A	6	3				M − 1 → M	•	•	↕	↕	④	•
	DECA													4A	2	1	A − 1 → A	•	•	↕	↕	④	•
	DECB													5A	2	1	B − 1 → B	•	•	↕	↕	④	•
Exclusive OR	EORA	88	2	2	98	3	2	A8	5	2	B8	4	3				A ⊕ M → A	•	•	↕	↕	R	•
	EORB	C8	2	2	D8	3	2	E8	5	2	F8	4	3				B ⊕ M → B	•	•	↕	↕	R	•
Increment	INC							6C	7	2	7C	6	3				M + 1 → M	•	•	↕	↕	⑤	•
	INCA													4C	2	1	A + 1 → A	•	•	↕	↕	⑤	•
	INCB													5C	2	1	B + 1 → B	•	•	↕	↕	⑤	•
Load Acmltr	LDAA	86	2	2	96	3	2	A6	5	2	B6	4	3				M → A	•	•	↕	↕	R	•
	LDAB	C6	2	2	D6	3	2	E6	5	2	F6	4	3				M → B	•	•	↕	↕	R	•
Or, Inclusive	ORAA	8A	2	2	9A	3	2	AA	5	2	BA	4	3				A + M → A	•	•	↕	↕	R	•
	ORAB	CA	2	2	DA	3	2	EA	5	2	FA	4	3				B + M → B	•	•	↕	↕	R	•
Push Data	PSHA													36	4	1	A → Msp, SP − 1 → SP	•	•	•	•	•	•
	PSHB													37	4	1	B → Msp, SP − 1 → SP	•	•	•	•	•	•
Pull Data	PULA													32	4	1	SP + 1 → SP, Msp → A	•	•	•	•	•	•
	PULB													33	4	1	SP + 1 → SP, Msp → B	•	•	•	•	•	•
Rotate Left	ROL							69	7	2	79	6	3				M	•	•	↕	↕	⑥	↕
	ROLA													49	2	1	A	•	•	↕	↕	⑥	↕
	ROLB													59	2	1	B — C b7 ← b0	•	•	↕	↕	⑥	↕
Rotate Right	ROR							66	7	2	76	6	3				M	•	•	↕	↕	⑥	↕
	RORA													46	2	1	A	•	•	↕	↕	⑥	↕
	RORB													56	2	1	B — C b7 → b0	•	•	↕	↕	⑥	↕
Shift Left, Arithmetic	ASL							68	7	2	78	6	3				M	•	•	↕	↕	⑥	↕
	ASLA													48	2	1	A	•	•	↕	↕	⑥	↕
	ASLB													58	2	1	B — C b7 b0 ← 0	•	•	↕	↕	⑥	↕
Shift Right, Arithmetic	ASR							67	7	2	77	6	3				M	•	•	↕	↕	⑥	↕
	ASRA													47	2	1	A	•	•	↕	↕	⑥	↕
	ASRB													57	2	1	B — b7 b0 C →	•	•	↕	↕	⑥	↕
Shift Right, Logic	LSR							64	7	2	74	6	3				M	•	•	R	↕	⑥	↕
	LSRA													44	2	1	A	•	•	R	↕	⑥	↕
	LSRB													54	2	1	B — b7 b0 C → 0	•	•	R	↕	⑥	↕
Store Acmltr	STAA				97	4	2	A7	6	2	B7	5	3				A → M	•	•	↕	↕	R	•
	STAB				D7	4	2	E7	6	2	F7	5	3				B → M	•	•	↕	↕	R	•
Subtract	SUBA	80	2	2	90	3	2	A0	5	2	B0	4	3				A − M → A	•	•	↕	↕	↕	↕
	SUBB	C0	2	2	D0	3	2	E0	5	2	F0	4	3				B − M → B	•	•	↕	↕	↕	↕
Subtract Acmltrs	SBA													10	2	1	A − B → A	•	•	↕	↕	↕	↕
Subtr with Carry	SBCA	82	2	2	92	3	2	A2	5	2	B2	4	3				A − M − C → A	•	•	↕	↕	↕	↕
	SBCB	C2	2	2	D2	3	2	E2	5	2	F2	4	3				B − M − C → B	•	•	↕	↕	↕	↕
Transfer Acmltrs	TAB													16	2	1	A → B	•	•	↕	↕	R	•
	TBA													17	2	1	B → A	•	•	↕	↕	R	•
Test Zero or Minus	TST							6D	7	2	7D	6	3				M − 00	•	•	↕	↕	R	R
	TSTA													4D	2	1	A − 00	•	•	↕	↕	R	R
	TSTB													5D	2	1	B − 00	•	•	↕	↕	R	R

LEGEND:

OP Operation Code (Hexadecimal)
~ Number of MPU Cycles
# Number of Program Bytes
+ Arithmetic Plus
− Arithmetic Minus
• Boolean AND
Msp Contents of memory location pointed to be Stack Pointer
+ Boolean Inclusive OR
⊕ Boolean Exclusive OR
M̄ Complement of M
→ Transfer Into
0 Bit = Zero
00 Byte = Zero

CONDITION CODE SYMBOLS:

H Half-carry from bit 3
I Interrupt mask
N Negative (sign bit)
Z Zero (byte)
V Overflow, 2's complement
C Carry from bit 7
R Reset Always
S Set Always
↕ Test and set if true, cleared otherwise
• Not Affected

(Note) Accumulator addressing mode instructions are included in the column for IMPLIED addressing.

CONDITION CODE REGISTER NOTES:
(Bit set if test is true and cleared otherwise)

① (Bit V) Test: Result = 10000000?
② (Bit C) Test: Result ≠ 00000000?
③ (Bit C) Test: Decimal value of most significant BCD Character greater than nine? (Not cleared if previously set.)
④ (Bit V) Test: Operand = 10000000 prior to execution?
⑤ (Bit V) Test: Operand = 01111111 prior to execution?
⑥ (Bit V) Test: Set equal to result of N⊕C after shift has occurred.

■ PROGRAM CONTROL OPERATIONS

Program Control operation can be subdivided into two categories: (1) Index Register/Stack Pointer instructions: (2) Jump and Branch of operations.

● Index Register/Stack Pointer Operations

The instructions for direct operation on the MPU's Index Register and Stack Pointer are summarized in Table 5. Decrement (DEX, DES), increment (INX, INS), load (LDX, LDS), and store (STX, STS) instructions are provided for both. The Compare instruction, CPX, can be used to compare the Index Register to a 16-bit value and update the Condition Code Register accordingly.

The TSX instruction causes the Index Register to be loaded with the address of the last data byte put onto the "stack". The TXS instruction loads the Stack Pointer with a value equal to one less than the current contents of the Index Register. This causes the next byte to be pulled from the "stack" to come from the location indicated by the Index Register. The utility of these two instructions can be clarified by describing the "stack" concept relative to the HMCS 6800 system.

The "stack" can be thought of as a sequential list of data stored in the MPU's read/write memory. The Stack Pointer contains a 16-bit memory address that is used to access the list from one end on a last-in-first-out (LIFO) basis in contrast to the random access mode used by the MPU's other addressing modes.

The HD6800 MPU instruction set and interrupt structure allow extensive use of the stack concept for efficient handling of data movement, subroutines and interrupts. The instructions can be used to establish one or more "stacks" anywhere in read/write memory. Stack length is limited only by the amount of memory that is made available.

Operation of the Stack Pointer with the Push and Pull instructions is illustrated in Figs. 19 and 20. The Push instruction (PSHA) causes the contents of the indicated accumulator (A in this example) to be stored in memory at the location indicated by the Stack Pointer. The Stack Pointer is automatically decremented by one following the storage operation and is "pointing" to the next empty stack location.

The Pull instruction (PULA or PULB) causes the last byte stacked to be loaded into the appropriate accumulator. The Stack Pointer is automatically incremented by one just prior to the data transfer so that it will point to the last byte stacked rather than the next empty location. Note that the PULL instruction does not "remove" the data from memory; in the example, 1A is still in location (m+1) following execution of PULA. A subsequent PUSH instruction would overwrite than location with the new "pushed" data.

Execution of the Branch to Subroutine (BSR) and Jump to Subroutine (JSR) instructions cause a return address to be save on the stack as shown in Figs. 21 through 23. The stack is decremented after each byte of the return address is pushed onto the stack. For both of the these instructions, the return address is the memory location following the bytes of code that correspond to the BSR and JSR instruction. The code required for BSR or JSR may be either two or three bytes, depending on whether the JSR is in the indexed (two bytes) or the extended (three bytes) addressing mode. Before it is stacked, the Program Counter is automatically incremented the correct number of times to be pointing at the location of the next instruction. The Return from Subroutine instruction, RTS, causes the return address to be retrieved and loaded into the Program Counter as shown in Fig. 24.

There are several operations that cause the status of the MPU to be saved on the stack. The Software Interrupt (SWI) and Wait for Interrupt (WAI) instructions as well as the maskable ($\overline{IRQ}$) and non-maskable ($\overline{NMI}$) hardware interrupts all cause the MPU's internal registers (except for the Stack Pointer itself) to be stacked as shown in Fig. 25. MPU status is restored by the Return from interrupt, RTI, as shown in Fig. 26.

Table 5 Index Register and Stack Pointer Instructions

Operation	Mnemonic	Addressing Modes														Boolean/ Arithmetic Operation	Cond. Code Reg.					
		IMMED			DIRECT			INDEX			EXTND			IMPLIED				5	4	3	2	1	0
		OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#		H	I	N	Z	V	C
Compare Index Reg	CPX	8C	3	3	9C	4	2	AC	6	2	BC	5	3				$(X_H) - (M), (X_L) - (M+1)$	•	•	①	↕	②	•
Decrement Index Reg	DEX													09	4	1	$X - 1 \rightarrow X$	•	•	•	↕	•	•
Decrement Stack Pntr	DES													34	4	1	$SP - 1 \rightarrow SP$	•	•	•	•	•	•
Increment Index Reg	INX													08	4	1	$X + 1 \rightarrow X$	•	•	•	↕	•	•
Increment Stack Pntr	INS													31	4	1	$SP + 1 \rightarrow SP$	•	•	•	•	•	•
Load Index Reg	LDX	CE	3	3	DE	4	2	EE	6	2	FE	5	3				$M \rightarrow X_H, (M+1) \rightarrow X_L$	•	•	③	↕	R	•
Load Stack Pntr	LDS	8E	3	3	9E	4	2	AE	6	2	BE	5	3				$M \rightarrow SP_H, (M+1) \rightarrow SP_L$	•	•	③	↕	R	•
Store Index Reg	STX				DF	5	2	EF	7	2	FF	6	3				$X_H \rightarrow M, X_L \rightarrow (M+1)$	•	•	③	↕	R	•
Store Stack Pntr	STS				9F	5	2	AF	7	2	BF	6	3				$SP_H \rightarrow M, SP_L \rightarrow (M+1)$	•	•	③	↕	R	•
Index Reg → Stack Pntr	TXS													35	4	1	$X - 1 \rightarrow SP$	•	•	•	•	•	•
Stack Pntr → Index Reg	TSX													30	4	1	$SP + 1 \rightarrow X$	•	•	•	•	•	•

① (Bit N) Test: Sign bit of most significant (MS) byte of result = 1?
② (Bit V) Test: 2's complement overflow from subtraction of ms bytes?
③ (Bit N) Test: Result less than zero? (Bit 15 = 1)

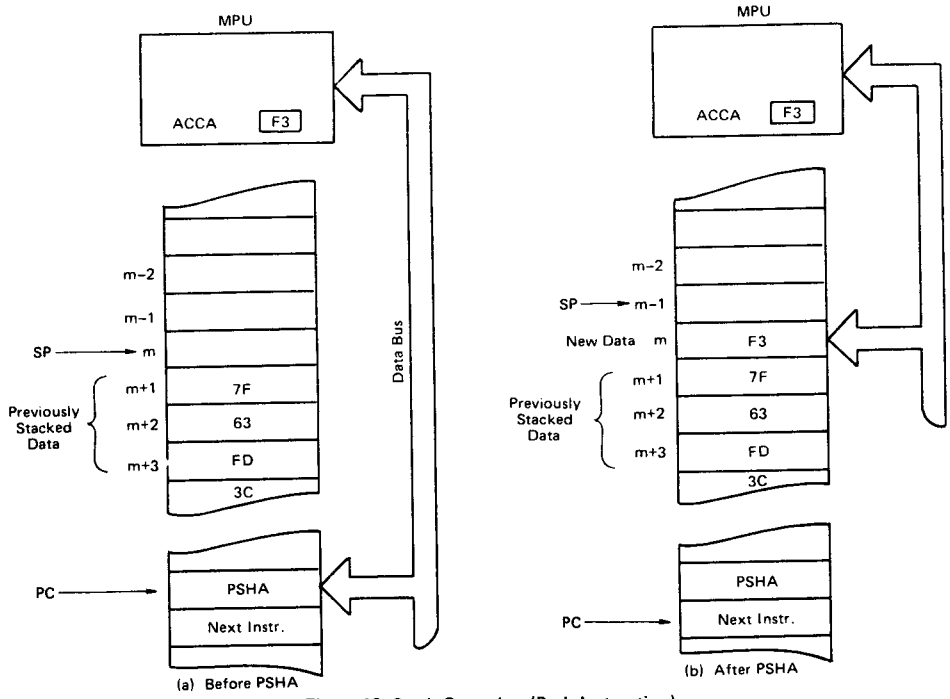


Figure 19 Stack Operation (Push Instruction)

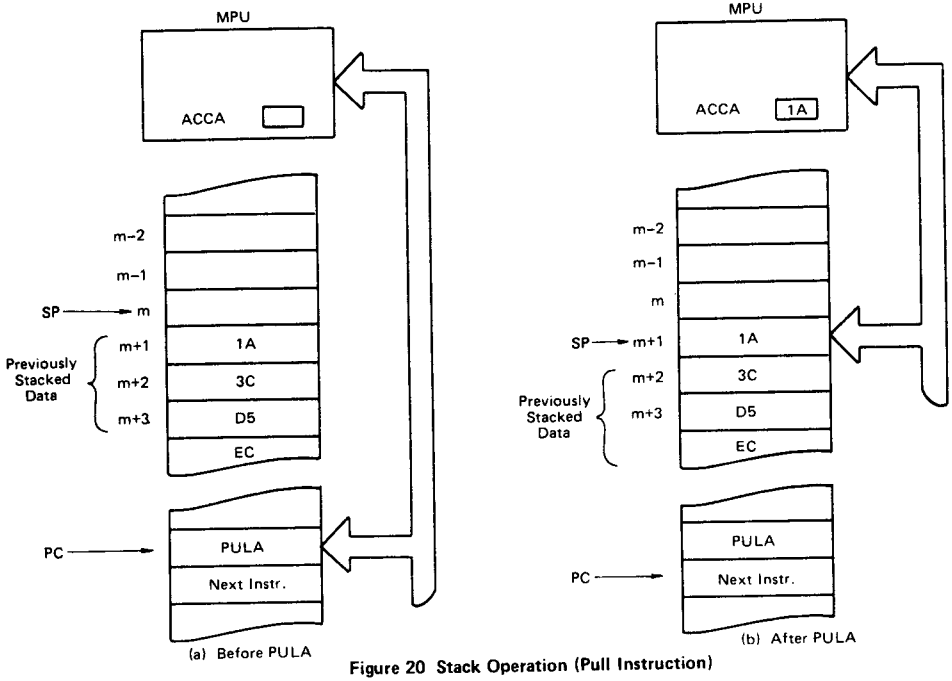


Figure 20 Stack Operation (Pull Instruction)

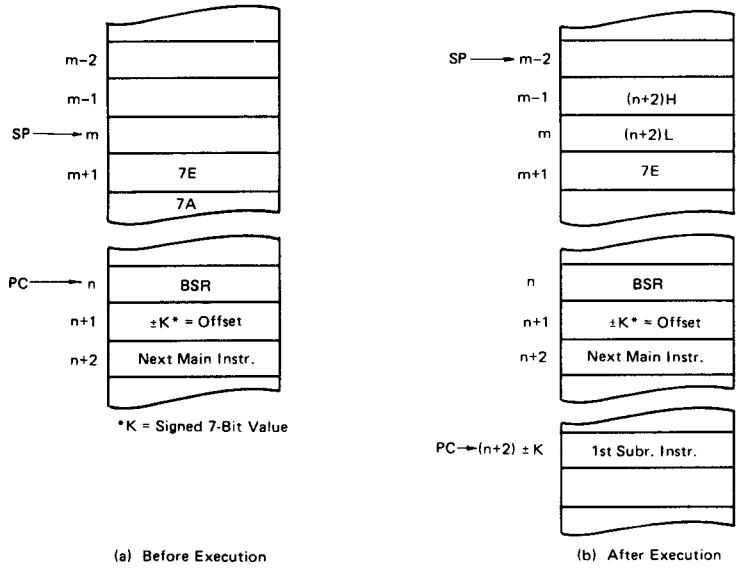


Figure 21 Program Flow for BSR

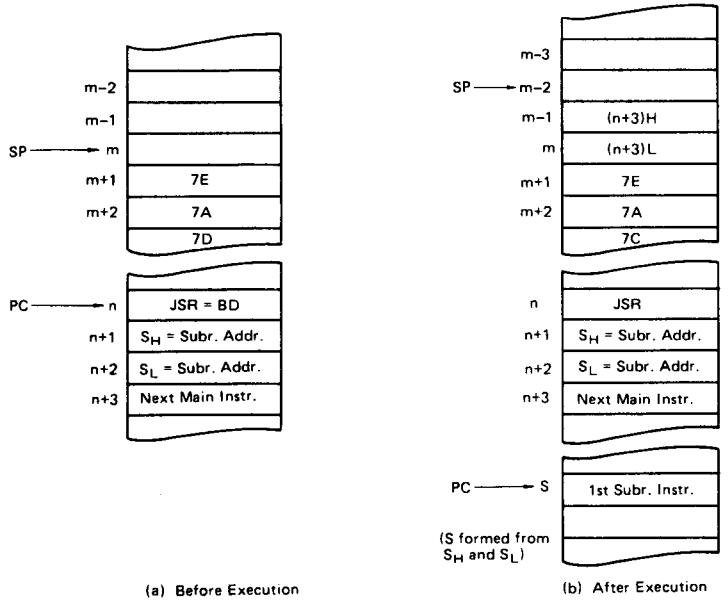


Figure 22 Program Flow for JSR (Extended)

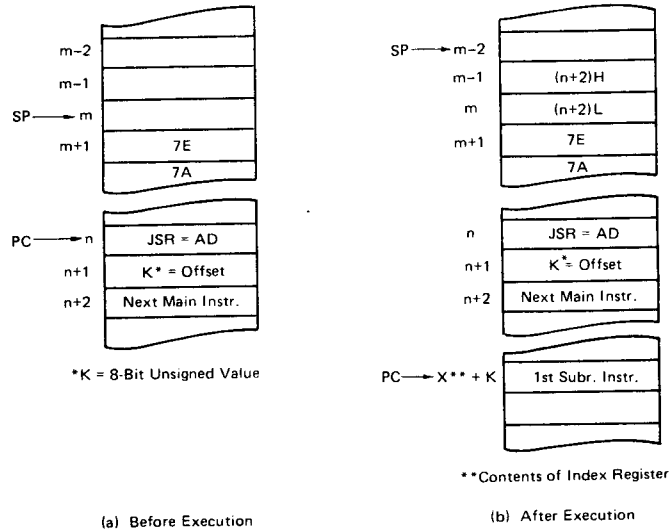


Figure 23 Program Flow for JSR (Indexed)

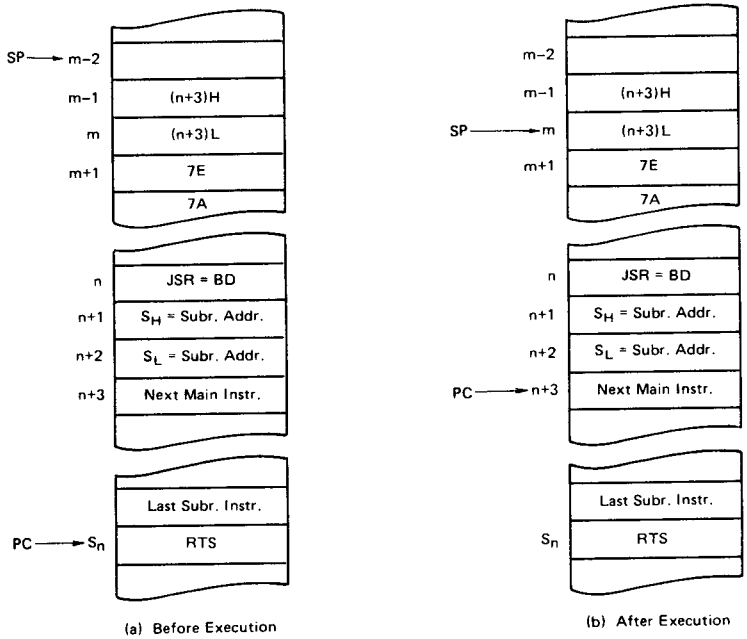


Figure 24 Program Flow for RTS

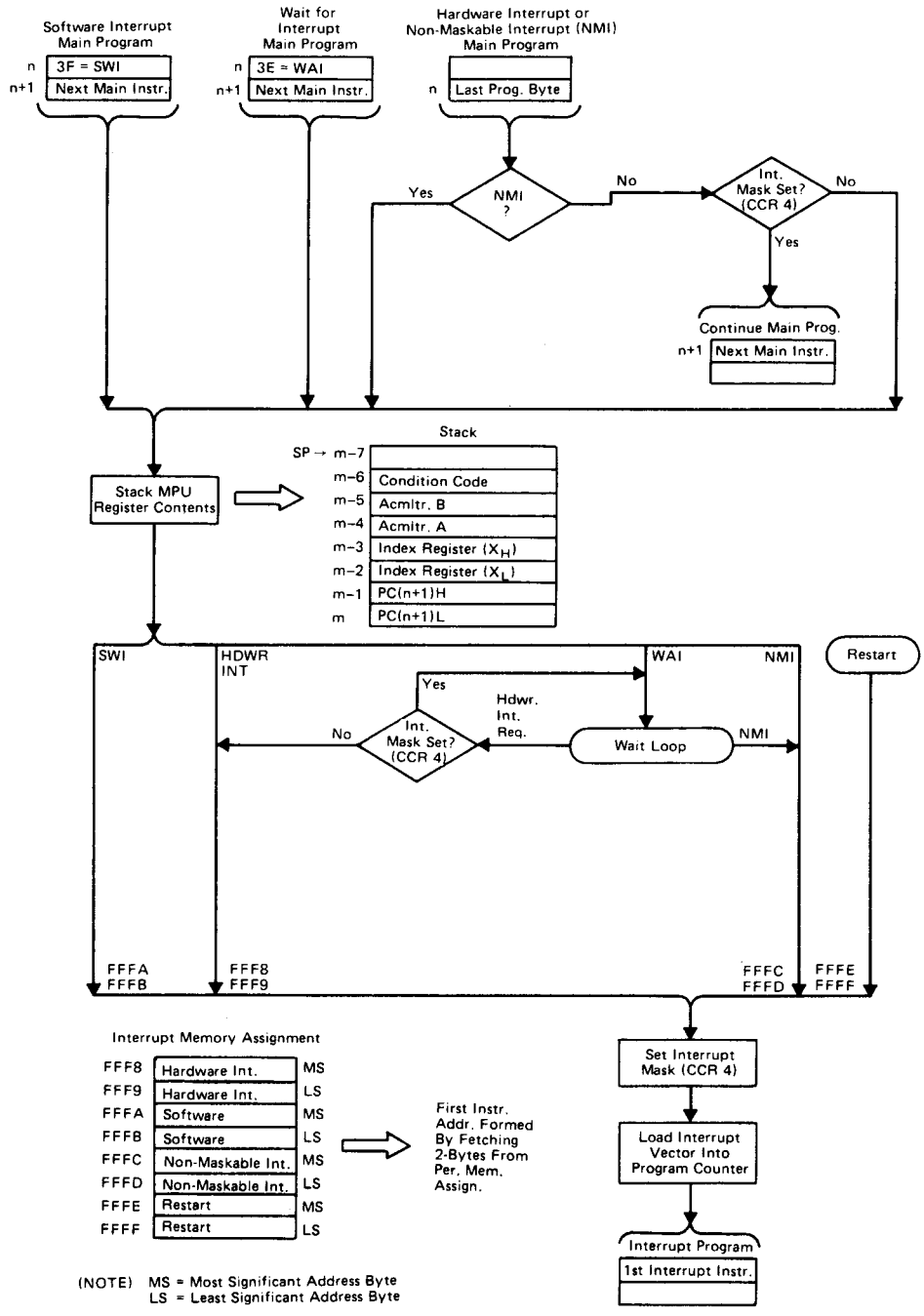


Figure 25 Program Flow for Interrupts

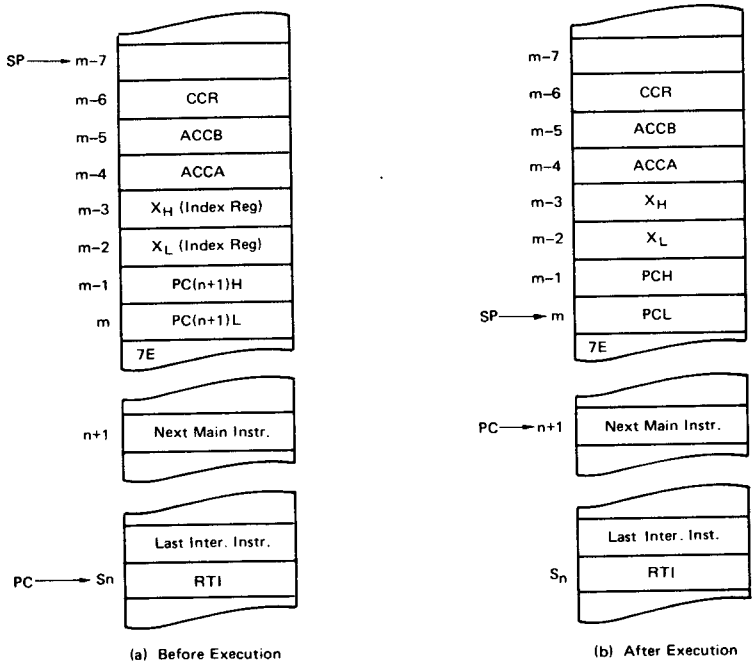


Figure 26 Program Flow for RTI

- **Jump and Branch Operation**

The Jump and Branch instructions are summarized in Table 6. These instructions are used to control the transfer of operation from one point to another in the control program.

The No Operation instruction, NOP, while included here, is a jump operation in a very limited sense. Its only effect is to increment the Program Counter by one. It is useful during program development as a "stand-in" for some other instruction that is to be determined during debug. It is also used for equalizing the execution time through alternate paths in a control program.

Execution of the Jump Instruction, JMP, and Branch Always, BRA, affects program flow as shown in Fig. 27. When the MPU encounters the Jump (Index) instruction, it adds the offset to the value in the Index Register and uses the result as the address of the next instruction to be executed. In the extended addressing mode, the address of the next instruction to be executed is fetched from the two locations immediately following the JMP instruction. The Branch Always (BRA) instruction is similar to the JMP (extended) instruction except that the relative addressing mode applies and the branch is limited to the range within −125 or +127 bytes of the branch instruction itself. The opcode for the BRA instruction requires one less byte than JMP (extended) but takes one more cycle to execute.

The effect on program flow for the Jump to Subroutine (JSR) and Branch to Subroutine (BSR) is shown in Figs. 21 through 23. Note that the Program Counter is properly incremented to be pointing at the correct return address before it is stacked. Operation of the Branch to Subroutine and Jump to Subroutine (extended) instruction is similar except for the range. The BSR instruction requires less opcode than JSR (2 bytes versus 3 bytes) and also executes one cycle faster than JSR. The Return from Subroutine, RTS, is used at the end of a subroutine to return to the main program as indicated in Fig. 24.

The effect of executing the Software Interrupt, SWI, and the Wait for Interrupt, WAI, and their relationship to the hardware interrupts is shown in Fig. 25. SWI causes the MPU contents to be stacked and then fetches the starting address of the interrupt routine from the memory locations that respond to the addresses FFFA and FFFB. Note that as in the case of the subroutine instructions, the Program Counter is incremented to point at the correct return address before being stacked. The Return from Interrupt instruction, RTI, (Fig. 26) is used at the end of an interrupt routine to restore control to the main program. The SWI instruction is useful for inserting break points in the control program, that is, it can be used to stop operation and put the MPU registers in memory where they can be examined. The WAI instruction is used to decrease the time required to service a hardware interrupt; it stacks the MPU contents and then waits for the interrupt to occur, effectively removing the stacking time from a hardware interrupt sequence.

Table 6 JUMP/BRANCH Instruction

Operation	Mnemonic	Addressing Modes												Branch Test	Cond. Code Reg.					
		RELATIVE			INDEX			EXTND			IMPLIED				5	4	3	2	1	0
		OP	~	#	OP	~	#	OP	~	#	OP	~	#		H	I	N	Z	V	C
Branch Always	BRA	20	4	2										None	•	•	•	•	•	•
Branch If Carry Clear	BCC	24	4	2										C = 0	•	•	•	•	•	•
Branch If Carry Set	BCS	25	4	2										C = 1	•	•	•	•	•	•
Branch If = Zero	BEQ	27	4	2										Z = 1	•	•	•	•	•	•
Branch If ≧ Zero	BGE	2C	4	2										N ⊕ V = 0	•	•	•	•	•	•
Branch If > Zero	BGT	2E	4	2										Z + (N ⊕ V) = 0	•	•	•	•	•	•
Branch If Higher	BHI	22	4	2										C + Z = 0	•	•	•	•	•	•
Branch If ≦ Zero	BLE	2F	4	2										Z + (N ⊕ V) = 1	•	•	•	•	•	•
Branch If Lower Or Same	BLS	23	4	2										C + Z = 1	•	•	•	•	•	•
Branch If < Zero	BLT	2D	4	2										N ⊕ V = 1	•	•	•	•	•	•
Branch If Minus	BMI	2B	4	2										N = 1	•	•	•	•	•	•
Branch If Not Equal Zero	BNE	26	4	2										Z = 0	•	•	•	•	•	•
Branch If Overflow Clear	BVC	28	4	2										V = 0	•	•	•	•	•	•
Branch If Overflow Set	BVS	29	4	2										V = 1	•	•	•	•	•	•
Branch If Plus	BPL	2A	4	2										N = 0	•	•	•	•	•	•
Branch To Subroutine	BSR	8D	8	2											•	•	•	•	•	•
Jump	JMP				6E	4	2	7E	3	3					•	•	•	•	•	•
Jump To Subroutine	JSR				AD	8	2	BD	9	3					•	•	•	•	•	•
No Operation	NOP										01	2	1	Advances Prog Cntr Only	•	•	•	•	•	•
Return From Interrupt	RTI										3B	10	1		———			①	———	
Return From Subroutine	RTS										39	5	1		•	•	•	•	•	•
Software Interrupt	SWI										3F	12	1		•	S	•	•	•	•
Wait for Interrupt	WAI										3E	9	1		•	②	•	•	•	•

① (All) Load Condition Code Register from Stack. (See Special Operations)
② (Bit I) Set when interrupt occurs. If previously set, a Non-Maskable interrupt is required to exit the wait state.

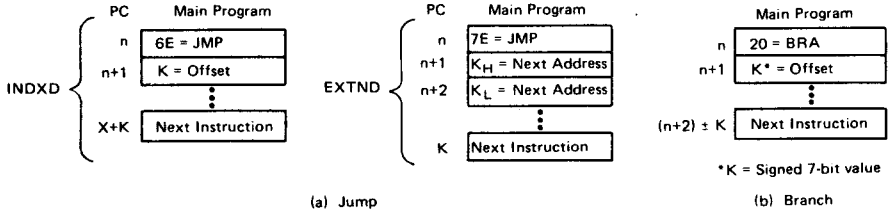


Figure 27 Program Flow for JUMP/BRANCH Instructions

BMI : N = 1 ; BEQ : Z = 1 ;
BPL : N = 0 ; BNE : Z = 0 ;

BVC : V = 0 ; BCC : C = 0 ;
BVS : V = 1 ; BCS : C = 1 ;

BHI : C + Z = 0 ; BLT : N ⊕ V = 1 ;
BLS : C + Z = 1 ; BGE : N ⊕ V = 0 ;

BLE : Z + (N ⊕ V) = 1 ;
BGT : Z + (N ⊕ V) = 0 ;

Figure 28 Conditional Branch Instructions

The conditional branch instructions, Fig. 28, consists of seven pairs of complementary instructions. They are used to test the results of the preceding operation and either continue with the next instruction in sequence (test fails) or cause a branch to another point in the program (test succeeds).

Four of the pairs are used for simple tests of status bits N, Z, V, and C:

1. Branch on Minus (BMI) and Branch On Plus (BPL) tests the sign bit, N, to determine if the previous result was negative or positive, respectively.
2. Branch On Equal (BEQ) and Branch On Not Equal (BNE) are used to test the zero status bit, Z, to determine whether or not the result of the previous operation was equal to "0". These two instructions are useful following a Compare (CMP) instruction to test for equality between an accumulator and the operand. They are also used following the Bit Test (BIT) to determine whether or not the same bit positions are set in an accumulator and the operand.

3. Branch On Overflow Clear (BVC) and Branch On Overflow Set (BVS) tests the state of the V bit to determine if the previous operation caused an arithmetic overflow.
4. Branch On Carry Clear (BCC) and Branch On Carry Set (BCS) tests the state of the C bit to determine if the previous operation caused a carry to occur. BCC and BCS are useful for testing relative magnitude when the values being tested are regarded as unsigned binary numbers, that is, the values are in the range "00" (lowest) of "FF" (highest). BCC following a comparison (CMP) will cause a branch if the (unsigned) value in the accumulator is higher than or the same as the value of the operand. Conversely, BCS will cause a branch if the accumulator value is lower than the operand.

The Fifth complementary pair, Branch On Higher (BHI) and Branch On Lower or Same (BLS) are in a sense complements to BCC and BCS. BHI tests for both C and Z = "0", if used following a CMP, it will cause a branch if the value in the accumulator is higher than the operand. Conversely, BLS will cause a branch if the unsigned binary value in the accumulator is lower than or the same as the operand.

The remaining two pairs are useful in testing results of operations in which the values are regarded as signed two's complement numbers. This differs from the unsigned binary case in the following sense: In unsigned, the orientation is higher or lower; in signed two's complement, the comparison is between larger or smaller where the range of values is between -128 and +127.

Branch On Less Than Zero (BLT) and Branch On Greater Than Or Equal Zero (BGE) test the status bits for N⊕V = "1" and N ⊕ V = "0", respectively. BLT will always cause a branch following an operation in which two negative numbers were added. In addition, it will cause a branch following a CMP in which the value in the accumulator was negative and the operand was positive. BLT will never cause a branch following a CMP in which the accumulator value was positive and the operand negative. BGE, the complement to BLT, will cause a branch following operations in which two positive values were added or in which the result was "0".

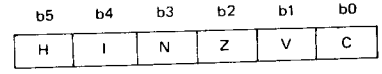
The last pair, Branch On Less Than Or Equal Zero (BLE) and Branch On Greater Than Zero (BGT) test the status bits for Z ⊕ (N + V) = "1" and Z ⊕(N + V) = "0", respectively, The action of BLE is identical to that for BLT except that a branch will also occur if the result of the previous result was "0". Conversely, BGT is similar to BGE except that no branch will occur following a "0" result.

■ CONDITION CODE REGISTER OPERATIONS

The Condition Code Register (CCR) is a 6-bit register within the MPU that is useful in controlling program flow during system operation. The bits are defined in Fig. 29.

The instructions shown in Table 7 are available to the user for direct manipulation of the CCR. In addition, the MPU automatically sets or clears the appropriate status bits as many of the other instructions on the condition code register was indicated as they were introduced.

Systems which require an interrupt window to be opened under program control should use a CLI-NOP-SEI sequence rather than CLI-SEI.



H = Half-carry; set whenever a carry from b3 to b4 of the result is generated by ADD, ABA, ADC; cleared if no b3 to b4 carry; not affected by other instructions.

I = Interrupt Mask; set by hardware of software interrupt or SEI instruction; cleared by CLI instruction. (Normally not used in arithmetic operations.) Restored to a "0" as a result of an RTI instruction if IM stored on the stacked is "0"

N = Negative; set if high order bit (b7) of result is set; cleared otherwise.

Z = Zero; set if result = "0"; cleared otherwise.

V = Overflow; set if there was arithmetic overflow as a result of the operation; cleared otherwise.

C = Carry; set if there was a carry from the most significant bit (b7) of the result; cleared otherwise.

Figure 29 Condition Code Register Bit Definition

■ ADDRESSING MODES

The MPU operates on 8-bit binary numbers presented to it via the Data Bus. A given number (byte) may represent either data or an instruction to be executed, depending on where it is encountered in the control program. The HD6800 MPU has 72 unique instructions, however, it recognizes and takes action on 197 of the 256 possibilities that can occur using an 8-bit word length. This larger number of instructions results from the fact that many of the executive instructions have more than one addressing mode.

Table 7 Condition Code Register Instructions

Operations	Mnemonic	Addressing Mode IMPLIED OP	~	#	Boolean Operation	Cond. Code Reg. 5 H	4 I	3 N	2 Z	1 V	0 C
Clear Carry	CLC	0C	2	1	0 → C	•	•	•	•	•	R
Clear Interrupt Mask	CLI	0E	2	1	0 → I	•	R	•	•	•	•
Clear Overflow	CLV	0A	2	1	0 → V	•	•	•	•	R	•
Set Carry	SEC	0D	2	1	1 → C	•	•	•	•	•	S
Set Interrupt Mask	SEI	0F	2	1	1 → I	•	S	•	•	•	•
Set Overflow	SEV	0B	2	1	1 → V	•	•	•	•	S	•
Acmltr A → CCR	TAP	06	2	1	A → CCR	①					
CCR → Acmltr A	TPA	07	2	1	CCR → A	•	•	•	•	•	•

R = Reset
S = Set
• = Not affected
① (ALL) Set according to the contents of Accumulator A.

These addressing modes refer to the manner in which the program causes the MPU to obtain its instructions and data. The programmer must have a method for addressing the MPU's internal registers and all of the external memory locations.

Selection of the desired addressing mode is made by the user as the source statements are written. Translation into appropriate opcode then depends on the method used. If manual translation is used, the addressing mode is implied in the opcode. For example, the Immediate, Direct, Indexed, and Extended modes may all be used with the ADD instruction. The proper mode is determined by selecting (hexidecimal notation) 8B, 9B, AB, or BB, respectively.

The source statement format includes adequate information for the selection if an assembler program is used to generate the opcode. For instance, the Immediate mode is selected by the Assembler whenever it encounters the "#" symbol in the operand field. Similarly, an "X" in the operand field causes the Indexed mode to be selected. Only the Relative mode applies to the branch instructions, therefore, the mnemonic instruction itself is enough for the Assembler to determine addressing mode.

For the instructions that use both Direct and Extended modes, the Assembler selects the Direct mode if the operand value is in the range 0~255 and Extended otherwise. There are a number of instructions for which the Extended mode is valid but the Direct is not. For these instructions, the Assembler automatically selects the Extended mode even if the operand is in the 0~255 range. The addressing modes are summarized in Fig. 30.

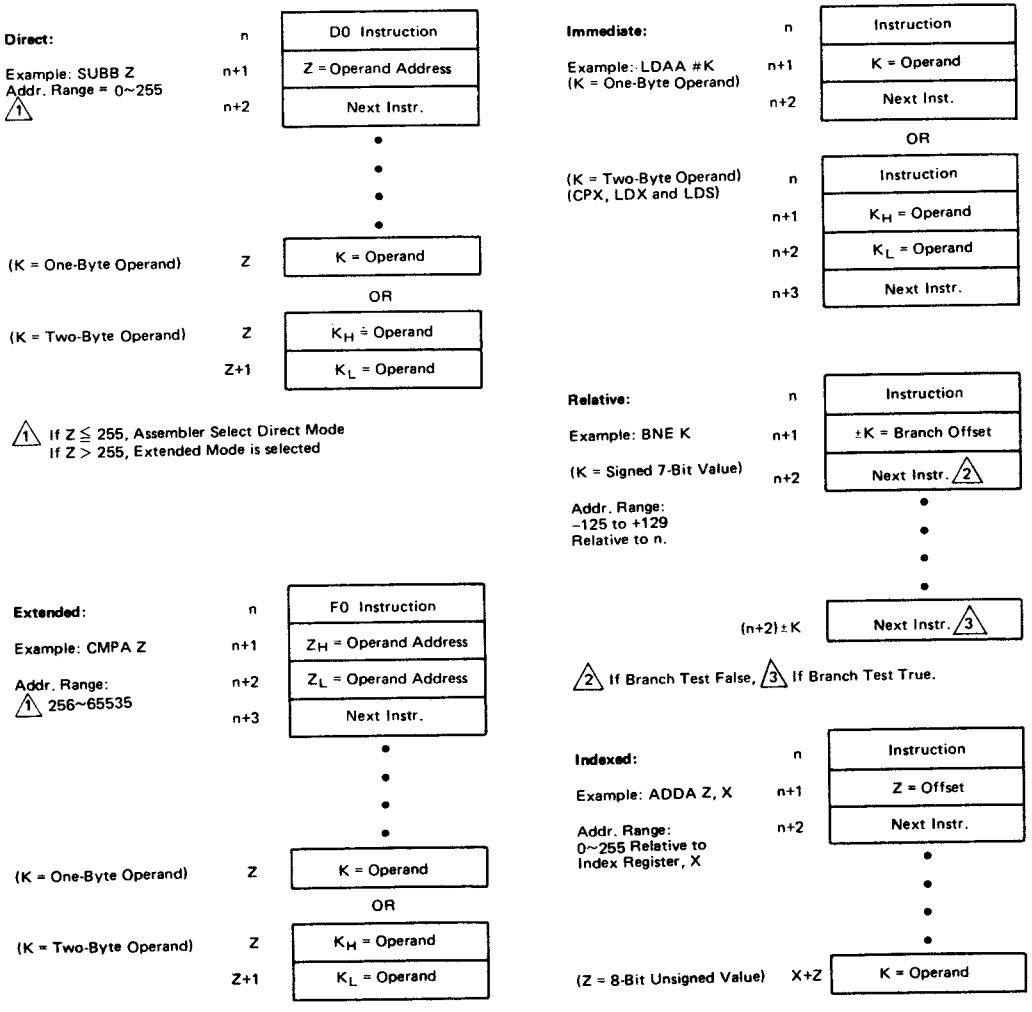


Figure 30 Addressing Mode Summary

- **Implied (Includes "Accumulator Addressing" Mode)**

The successive fields in a statement are normally separated by one or more spaces. An exception to this rule occurs for instructions that use dual addressing in the operand field and for instructions that must distinguish between the two accumulators. In these cases, A and B are "operands" but the space between them and the operator may be omitted. This is commonly done, resulting in apparent four character mnemonics for those instructions.

The addition instruction, ADD, provides an example of dual addressing in the operand fields;

Operator	Operand	Comment
ADDA	MEM12	ADD CONTENTS OF MEM12 TO ACCA
or ADDB	MEM12	ADD CONTENTS OF MEM12 TO ACCB

The example used earlier for the test instruction, TST, also applies to the accumulators and uses the "accumulator addressing mode" to designate which of the two accumulators is being tested:

Operator	Comment
TSTB	TEST CONTENTS OF ACCB
or TSTA	TEST CONTENTS OF ACCA

A number of the instructions either alone or together with an accumulator operand contain all of the address information that is required, that is, "inherent" in the instruction, itself. For instance, the instruction ABA causes the MPU to add the contents of accumulators A and B together and place the result in accumulator A. The instruction INCB, another example of "accumulator addressing", causes the contents of accumulator B to be increased by one. Similarly, INX, increment the Index Register, causes the contents of the Index Register to be increased by one.

Program flow for instructions of this type is illustrated in Figures 31 and 32. In these figures, the general case is shown on the left and a specific example is shown on the right. Numerical examples are in decimal notation. Instructions of this type require only one byte of opcode. Cycle-by-cycle operation of the implied mode is shown in Table 8.

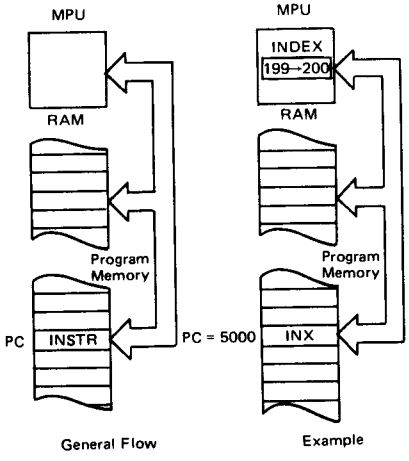


Figure 31 Implied Addressing

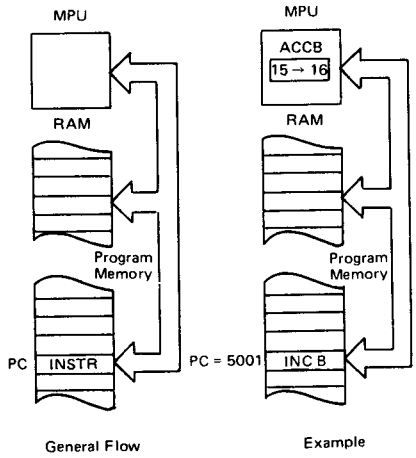


Figure 32 Accumulator Addressing

- **Immediate Addressing Mode**

In the Immediate addressing mode, the operand is the value that is to be operated on. For instance, the instruction

Operator	Operand	Comment
LDAA	#25	LOAD 25 INTO ACCA

causes the MPU to "immediately load accumulator A with the value 25"; no further address reference is required. The Immediate mode is selected by preceding the operand value with the "#" symbol. Program flow for this addressing mode is illustrated in Fig. 33.

The operand format allows either properly defined symbols or numerical values. Except for the instructions CPX, LDX, and LDS, the operand may be any value in the range 0 ~ 255. Since Compare Index Register (CPX), Load Index Register (LDX), Load Stack Pointer (LDS), require 16-bit values, the immediate mode for these three instructions requie two-byte operands.

Table 9 shows the cycle-by-cycle operation for the immediate addressing mode.

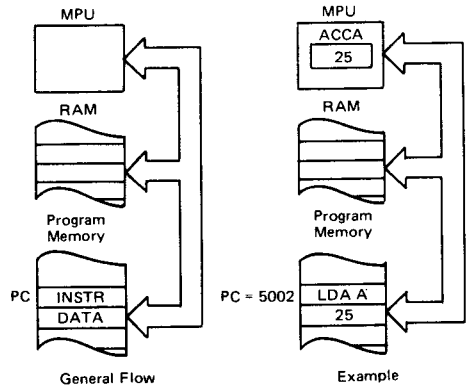


Figure 33 Immediate Addressing Mode

Table 8 Implied Mode Cycle by Cycle Operation

Address Mode and Instructions	Cycle	Cycle #	VMA Line	Address Bus	R/W Line	Data Bus
ABA DAA SEC ASL DEC SEI ASR INC SEV CBA LSR TAB CLC NEG TAP CLI NOP TBA CLR ROL TPA CLV ROR TST COM SBA	2	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Op Code of Next Instruction
DES DEX INS INX	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Op Code of Next Instruction
		3	0	Previous Register Contents	1	Irrelevant Data (NOTE 1)
		4	0	New Register Contents	1	Irrelevant Data (NOTE 1)
PSH	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Op Code of Next Instruction
		3	1	Stack Pointer	0	Accumulator Data
		4	0	Stack Pointer − 1	1	Accumulator Data
PUL	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Op Code of Next Instruction
		3	0	Stack Pointer	1	Irrelevant Data (NOTE 1)
		4	1	Stack Pointer + 1	1	Operand Data from Stack
TSX	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Op Code of Next Instruction
		3	0	Stack Pointer	1	Irrelevant Data (NOTE 1)
		4	0	New Index Register	1	Irrelevant Data (NOTE 1)
TXS	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Op Code of Next Instruction
		3	0	Index Register	1	Irrelevant Data
		4	0	New Stack Pointer	1	Irrelevant Data
RTS	5	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Irrelevant Data (NOTE 2)
		3	0	Stack Pointer	1	Irrelevant Data (NOTE 1)
		4	1	Stack Pointer + 1	1	Address of Next Instruction (High Order Byte)
		5	1	Stack Pointer + 2	1	Address of Next Instruction (Low Order Byte)
WAI	9	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Op Code of Next Instruction
		3	1	Stack Pointer	0	Return Address (Low Order Byte)
		4	1	Stack Pointer − 1	0	Return Address (High Order Byte)
		5	1	Stack Pointer − 2	0	Index Register (Low Order Byte)
		6	1	Stack Pointer − 3	0	Index Register (High Order Byte)
		7	1	Stack Pointer − 4	0	Contents of Accumulator A
		8	1	Stack Pointer − 5	0	Contents of Accumulator B
		9	1	Stack Pointer − 6 (NOTE 3)	1	Contents of Cond. Code Register
RTI	10	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Irrelevant Data (NOTE 2)
		3	0	Stack Pointer	1	Irrelevant Data (NOTE 1)
		4	1	Stack Pointer + 1	1	Contents of Cond. Code Register from Stack
		5	1	Stack Pointer + 2	1	Contents of Accumulator B from Stack
		6	1	Stack Pointer + 3	1	Contents of Accumulator A from Stack
		7	1	Stack Pointer + 4	1	Index Register from Stack (High Order Byte)
		8	1	Stack Pointer + 5	1	Index Register from Stack (Low Order Byte)
		9	1	Stack Pointer + 6	1	Next Instruction Address from Stack (High Order Byte)
		10	1	Stack Pointer + 7	1	Next Instruction Address from Stack (Low Order Byte)
SWI	12	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Irrelevant Data (NOTE 1)
		3	1	Stack Pointer	0	Return Address (Low Order Byte)
		4	1	Stack Pointer − 1	0	Return Address (High Order Byte)
		5	1	Stack Pointer − 2	0	Index Register (Low Order Byte)
		6	1	Stack Pointer − 3	0	Index Register (High Order Byte)
		7	1	Stack Pointer − 4	0	Contents of Accumulator A
		8	1	Stack Pointer − 5	0	Contents of Accumulator B
		9	1	Stack Pointer − 6	0	Contents of Cond. Code Register
		10	0	Stack Pointer − 7	1	Irrelevant Data (NOTE 1)
		11	1	Vector Address FFFA (Hex)	1	Address of Subroutine (High Order Byte)
		12	1	Vector Address FFFB (Hex)	1	Address of Subroutine (Low Order Byte)

NOTE 1. If device which is addressed during this cycle uses VMA, then the Data Bus will go to the high impedance three-state condition. Depending on bus capacitance, data from the previous cycle may be retained on the Data Bus.
NOTE 2. Data is ignored by the MPU.
NOTE 3. While the MPU is waiting for the interrupt, Bus Available will go "High" indicating the following states of the control lines: VMA is "Low"; Address Bus, R/$\overline{W}$, and Data Bus are all in the high impedance state.

Table 9 Immediate Mode Cycle by Cycle Operation

Address Mode and Instructions	Cycle	Cycle #	VMA Line	Address Bus	R/$\overline{W}$ Line	Data Bus
ADC EOR ADD LDA AND ORA BIT SBC CMP SUB	2	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Operand Data
CPX LDS LDX	3	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Operand Data (High Order Byte)
		3	1	Op Code Address + 2	1	Operand Data (Low Order Byte)

- **Direct and Extended Addressing Modes**

In the Direct and Extended modes of addressing, the operand field of the source statement is the address of the value that is to be operated on. The Direct and Extended modes differ only in the range of memory locations to which they can direct the MPU. Direct addressing generates a single 8-bit operand and, hence, can address only memory locations 0 ~ 255; a two byte operand is generated for Extended addressing, enabling the MPU to reach the remaining memory locations, 256 ~ 65535. An example of Direct addressing and its effect on program flow is illustrated in Fig. 34.

Table 10 shows the cycle-by-cycle operations of this mode.

The MPU, after encountering the opcode for the instrution LDAA (Direct) at memory location 5004 (Program Counter = 5004), looks in the next location, 5005, for the address of the operand. It then sets the program counter equal to the value found there (100 in the example) and fetches the operand, in this case a value to be loaded into accumulator A, from that location. For instructions requiring a two-byte operand such as LDX (Load the Index Register), the operand bytes would be retrieved from locations 100 and 101.

Extended addressing, Fig. 35, is similar except that a two-byte address is obtained from locations 5007 and 5008 after the LDAB (Extended) opcode shows up in location 5006. Extended addressing can be thought of as the "standard" addressing mode, that is, it is a method of reaching anyplace in memory. Direct addressing, since only one address byte is required, provides a faster method of processing data and generates fewer bytes of control code. In most applications, the direct addressing range, memory locations 0 ~ 255, are reserved for RAM. They are used for data buffering and temporary storage of system variables, the area in which faster addressing is of most value, Cycle-by-cycle operation is shown in Table 11 for Extended Addressing.

Table 10 Direct Mode Cycle by Cycle Operation

Address Mode and Instructions	Cycle	Cycle #	VMA Line	Address Bus	R/$\overline{W}$ Line	Data Bus
ADC EOR ADD LDA AND ORA BIT SBC CMP SUB	3	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand
		3	1	Address of Operand	1	Operand Data
CPX LDS LDX	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand
		3	1	Address of Operand	1	Operand Data (High Order Byte)
		4	1	Operand Address + 1	1	Operand Data (Low Order Byte)
STA	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Destination Address
		3	0	Destination Address	1	Irrelevant Data (NOTE 1)
		4	1	Destination Address	0	Data from Accumulator
STS STX	5	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand
		3	0	Address of Operand	1	Irrelevant Data (NOTE 1)
		4	1	Address of Operand	0	Register Data (High Order Byte)
		5	1	Address of Operand + 1	0	Register Data (Low Order Byte)

NOTE 1. If device which is address during this cycle uses VMA, then the Data Bus will go to the high impedance three-state condition. Depending on bus capacitance, data from the previous cycle may be retained on the Data Bus.

Table 11 Extended Mode Cycle by Cycle

Address Mode and Instructions	Cycle	Cycle #	VMA Line	Address Bus	R/$\overline{W}$ Line	Data Bus
STS STX	6	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand (High Order Byte)
		3	1	Op Code Address + 2	1	Address of Operand (Low Order Byte)
		4	0	Address of Operand	1	Irrelevant Data (NOTE 1)
		5	1	Address of Operand	0	Operand Data (High Order Byte)
		6	1	Address of Operand + 1	0	Operand Data (Low Order Byte)
JSR	9	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Subroutine (High Order Byte)
		3	1	Op Code Address + 2	1	Address of Subroutine (Low Order Byte)
		4	1	Subroutine Starting Address	1	Op Code of Next Instruction
		5	1	Stack Pointer	0	Return Address (Low Order Byte)
		6	1	Stack Pointer − 1	0	Return Address (High Order Byte)
		7	0	Stack Pointer − 2	1	Irrelevant Data (NOTE 1)
		8	0	Op Code Address + 2	1	Irrelevant Data (NOTE 1)
		9	1	Op Code Address + 2	1	Address of Subroutine (Low Order Byte)
JMP	3	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Jump Address (High Order Byte)
		3	1	Op Code Address + 2	1	Jump Address (Low Order Byte)
ADC EOR ADD LDA AND ORA BIT SBC CMP SUB	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand (High Order Byte)
		3	1	Op Code Address + 2	1	Address of Operand (Low Order Byte)
		4	1	Address of Operand	1	Operand Data
CPX LDS LDX	5	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand (High Order Byte)
		3	1	Op Code Address + 2	1	Address of Operand (Low Order Byte)
		4	1	Address of Operand	1	Operand Data (High Order Byte)
		5	1	Address of Operand + 1	1	Operand Data (Low Order Byte)
STA A STA B	5	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Destination Address (High Order Byte)
		3	1	Op Code Address + 2	1	Destination Address (Low Order Byte)
		4	0	Operand Destination Address	1	Irrelevant Data (NOTE 1)
		5	1	Operand Destination Address	0	Data from Accumulator
ASL LSR ASR NEG CLR ROL COM ROR DEC TST INC	6	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Address of Operand (High Order Byte)
		3	1	Op Code Address + 2	1	Address of Operand (Low Order Byte)
		4	1	Address of Operand	1	Current Operand Data
		5	0	Address of Operand	1	Irrelevant Data (NOTE 1)
		6	1/0 (NOTE 2)	Address of Operand	0	New Operand Data (NOTE 2)

NOTE 1. If device which is addressed during this cycle uses VMA, then the Data Bus will go to the high impedance three-state condition. Depending on bus capacitance, data from the previous cycle may be retained on the Data Bus.
NOTE 2. For TST, VMA = 0 and Operand data does not change.

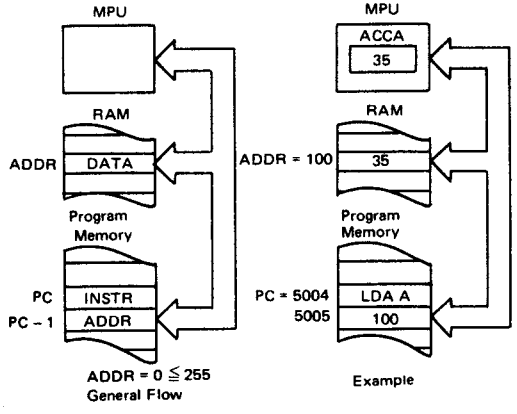


Figure 34 Direct Addressing Mode

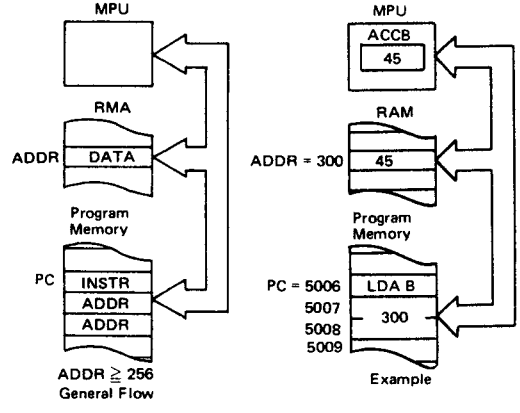


Figure 35 Extended Addressing Mode

- **Relative Address Mode**

In both the Direct and Extended modes, the address obtained by the MPU is an absolute numerical address. The Relative addressing mode, implemented for the MPU's branch instructions, specifies a memory location relative to the Program Counter's current location. Branch instructions generate two bytes of machine code, one for the instruction opcode and one for the "relative" address (see Fig. 36). Since it is desirable to be able to branch in either direction, the 8-bit address byte is interpreted as a signed 7-bit value; the 8th bit of the operand is treated as a sign bit, "0" = plus and "1" = minus. The remaining seven bits represent the numerical value. This result in a relative addressing range of ±127 with respect to the location of the branch instruction itself. However, the branch range is computed with respect to the next instruction that would be executed if the branch conditions are not satisfied. Since two byte are generated, the next instruction is located at PC+2. If, D is defined as the address of the branch destination, the range is then;

$$(PC+2) - 128 \leqq D \leqq (PC+2) + 127$$

or

$$PC - 126 \leqq D \leqq PC + 129$$

that is, the destination of the branch instruction must be within -126 to +129 memory locations of the branch instruction itself. For transferring control beyond this range, the unconditional jump (JMP), jump to subroutine (JSR), and return from subroutine (RTS) are used.

In Fig. 36, when the MPU encounters the opcode for BEQ (Branch if result of last instruction was zero), it tests the Zero bit in the Condition Code Register. If that bit is "0", indicating a non-zero result, the MPU continues execution with the next instruction (in location 5010 in Fig. 36). If the previous result was zero, the branch condition is satisfied and the MPU adds the offset, 15 in this case, to PC+2 and branches to location 5025 for the next instruction.

The branch instructions allow the programmer to efficiently direct the MPU to one point or another in the control program depending on the outcome of test results. Since the control program is normally in read-only memory and cannot be changed, the relative address used in execution of branch instructions is a constant numerical value. Cycle-by-cycle operation is shown in Table 12 for relative addressing.

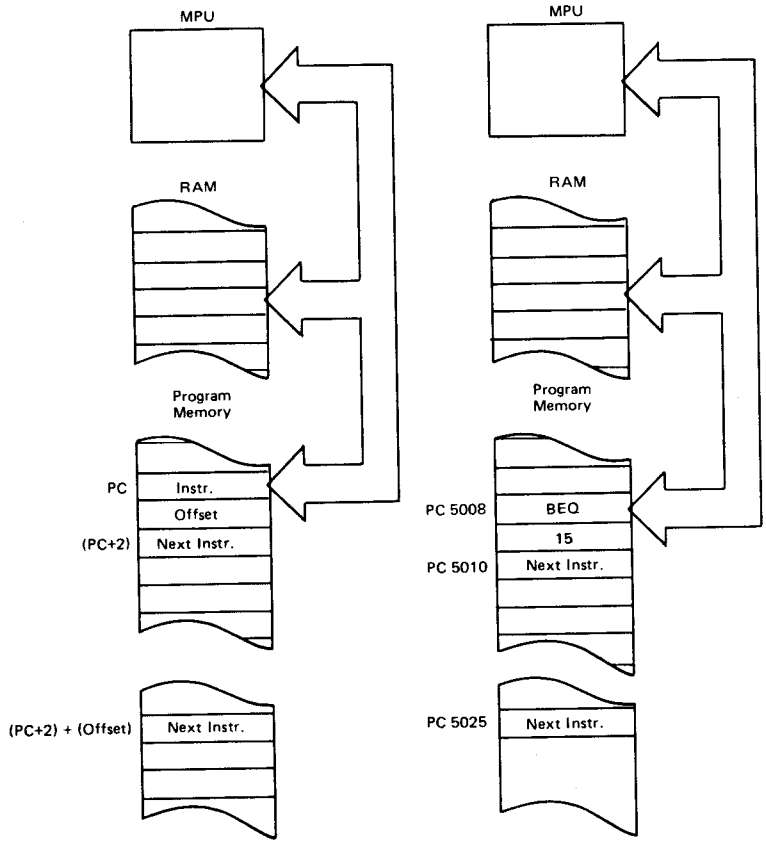


Figure 36 Relative Addressing Mode

Table 12 Relative Mode Cycle-by-Cycle Operation

Address Mode and Instructions	Cycle	Cycle #	VMA Line	Address Bus	R/$\overline{W}$ Line	Data Bus
BCC BHI BNE BCS BLE BPL BEQ BLS BRA BGE BLT BVC BGT BMI BVS	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Branch Offset
		3	0	Op Code Address + 2	1	Irrelevant Data (NOTE 1)
		4	0	Branch Address	1	Irrelevant Data (NOTE 1)
BSR	8	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Branch Offset
		3	0	Return Address of Main Program	1	Irrelevant Data (NOTE 1)
		4	1	Stack Pointer	0	Return Address (Low Order Byte)
		5	1	Stack Pointer − 1	0	Return Address (High Order Byte)
		6	0	Stack Pointer − 2	1	Irrelevant Data (NOTE 1)
		7	0	Return Address of Main Program	1	Irrelevant Data (NOTE 1)
		8	0	Subroutine Address	1	Irrelevant Data (NOTE 1)

NOTE 1. If device which is addressed during this cycle uses VMA, then the Data Bus will go to the high impedance three-state condition. Depending on bus capacitance, data from the previous cycle may be retained on the Data Bus.

- **Indexed Addressing Mode**

With Indexed addressing the numerical address is variable and depend on the current contents of the Index Register. A source statement such as

```
Operator   Operand        Comment
STAA       X          PUT A IN INDEXED LOCATION
```

causes the MPU to store the contents of accumulator A in the memory location specified by the contents of the Index Register (recall that the label X is reserved to designate the Index Register). Since there are instructions for manipulating X during program execution (LDX, INX, DEX, etc.), the Indexed addressing mode provides a dynamic "on the fly" way to modify program activity.

The operand field can also contain a numerical value that will be automatically added to X during execution. This format is illustrated in Fig. 37.

When the MPU encounters the LDAB (Indexed) opcode in location 5006, it looks in the next memory location for the value to be added to X (5 in the example) and calculates the required address by adding 5 to the present Index Register value of 400. In the operand format, the offset may be represented by a label or a numerical value in the range 0 ~ 255 as in the example. In the earlier example, STAA X, the operand is equivalent to 0, X , that is, the "0" may be omitted when the desired address is equal to X. Table 13 shows the cycle-by-cycle operation for the Indexed Mode of Addressing.

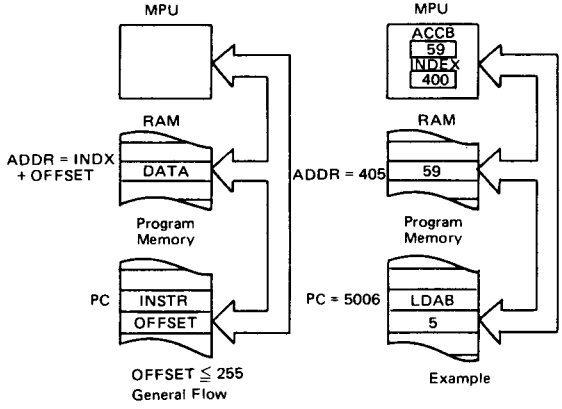


Figure 37 Indexed Addressing Mode

Table 13 Indexed Mode Cycle by Cycle

Address Mode and Instructions	Cycle	Cycle #	VMA Line	Address Bus	R/$\overline{W}$ Line	Data Bus
JMP	4	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (NOTE 1)
		4	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (NOTE 1)
ADC EOR ADD LDA AND QRA BIT SBC CMP SUB	5	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (NOTE 1)
		4	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (NOTE 1)
		5	1	Index Register Plus Offset	1	Operand Data
CPX LDS LDX	6	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (NOTE 1)
		4	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (NOTE 1)
		5	1	Index Register Plus Offset	1	Operand Data (High Order Byte)
		6	1	Index Register Plus Offset + 1	1	Operand Data (Low Order Byte)
STA	6	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (NOTE 1)
		4	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (NOTE 1)
		5	0	Index Register Plus Offset	1	Irrelevant Data (NOTE 1)
		6	1	Index Register Plus Offset	0	Operand Data
ASL LSR ASR NEG CLR ROL COM ROR DEC TST INC	7	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (NOTE 1)
		4	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (NOTE 1)
		5	1	Index Register Plus Offset	1	Current Operand Data
		6	0	Index Register Plus Offset	1	Irrelevant Data (NOTE 1)
		7	1/0 (NOTE 2)	Index Register Plus Offset	0	New Operand Data (NOTE 2)
STS STX	7	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (NOTE 1)
		4	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (NOTE 1)
		5	0	Index Register Plus Offset	1	Irrelevant Data (NOTE 1)
		6	1	Index Register Plus Offset	0	Operand Data (High Order Byte)
		7	1	Index Register Plus Offset + 1	0	Operand Data (Low Oder Byte)
JSR	8	1	1	Op Code Address	1	Op Code
		2	1	Op Code Address + 1	1	Offset
		3	0	Index Register	1	Irrelevant Data (NOTE 1)
		4	1	Stack Pointer	0	Return Address (Low Order Byte)
		5	1	Stack Pointer – 1	0	Return Address (High Order Byte)
		6	0	Stack Pointer – 2	1	Irrelevant Data (NOTE 1)
		7	0	Index Register	1	Irrelevant Data (NOTE 1)
		8	0	Index Register Plus Offset (w/o Carry)	1	Irrelevant Data (NOTE 1)

NOTE 1. If Device which is addressed during this cycle uses VMA, then the Data Bus will go to the high impedance three-state condition. Depending on bus capacitance, data from the previous cycle may be retained on the Data Bus.

NOTE 2. For TST, VMA = 0 and Operand data does not change.

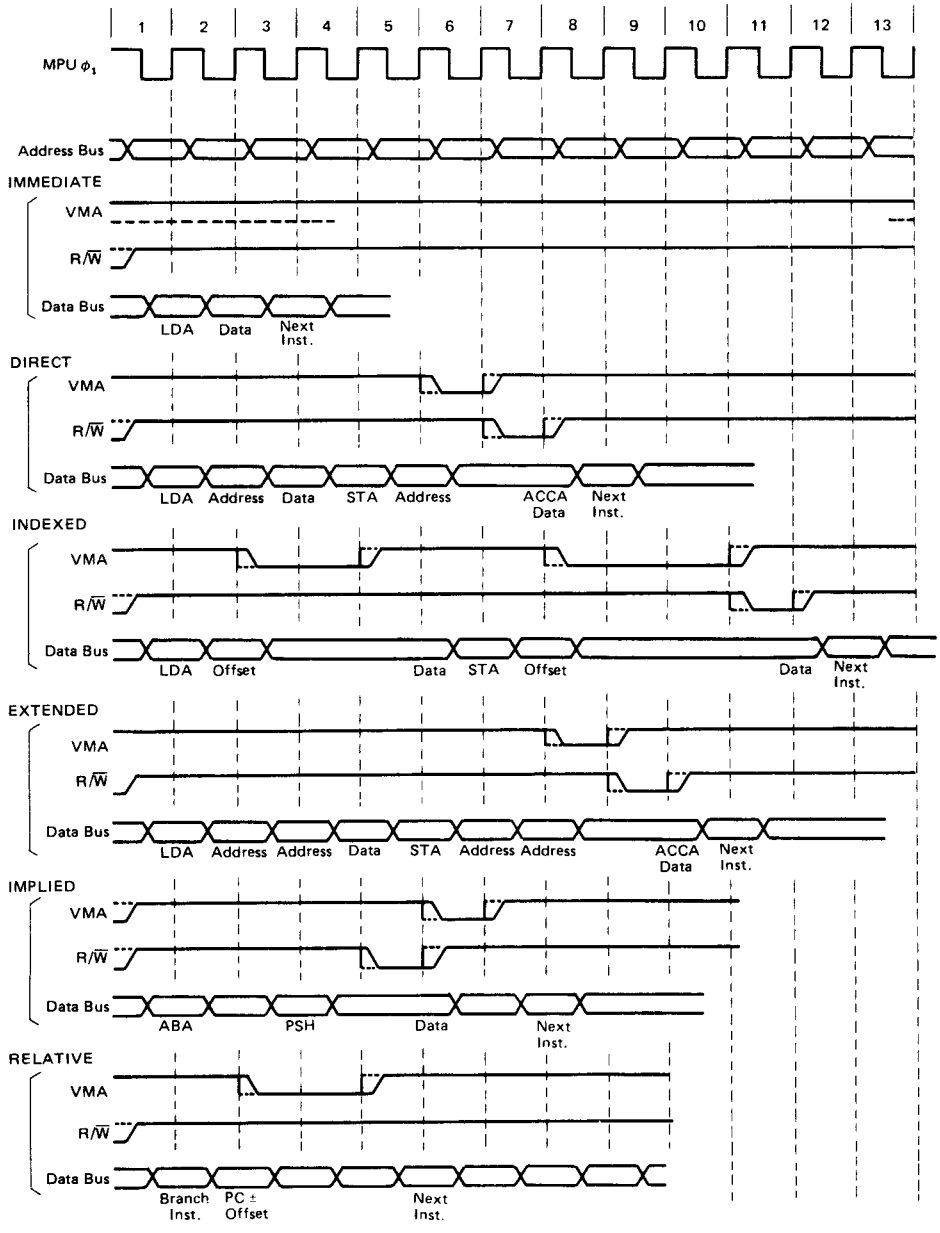


Figure 38 Example of Excution Timing in Each Addressing Mode

■ NOTE FOR THE RELATION BETWEEN WAI INSTRUCTION AND HALT OPERATION OF HD6800

When $\overline{HALT}$ input signal is asserted to "Low" level, the MPU will be halted after the execution of the current instruction except WAI instruction.

The "Halt" signal is not accepted after the fetch cycle of the WAI instruction (See ① in Fig. 39). In the case of the "WAI" instruction, the MPU enters the "WAIT" cycle after stacking the internal registers and outputs the "High" level on the BA line.

When an interrupt request signal is input to the MPU, the MPU accepts the interrupt regardless the "Halt" signal and releases the "WAIT" state and outputs the interrupt's vector address. If the "Halt" signal is "Low" level, the MPU halts after the fetch of new PC contents. The sequense is shown below.

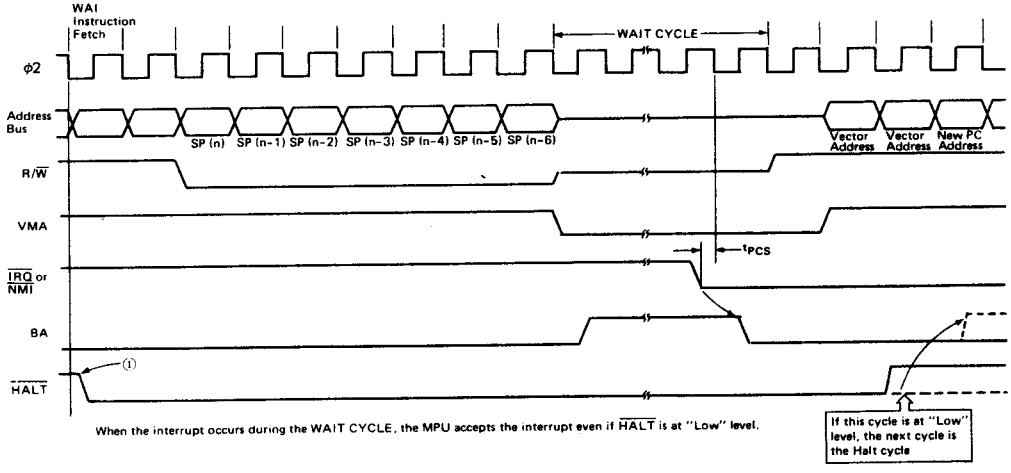


Figure 39 HD6800 WAIT CYCLE & $\overline{HALT}$ Request