

8-Bit Microprocessing Unit (MPU)

The MC6800 is a monolithic 8-bit microprocessor forming the central control function for Motorola's M6800 Family. Compatible with TTL, the MC6800, as with all M6800 system parts, requires only one +5.0-volt power supply and no external TTL devices for bus interface.

The MC6800 is capable of addressing 64K bytes of memory with its 16-bit address lines. The 8-bit data bus is bidirectional as well as three-state, making direct memory addressing and multiprocessing applications realizable.

- 8-Bit Parallel Processing
- Bidirectional Data Bus
- 16-Bit Address Bus — 64K Bytes of Addressing
- 72 Instructions — Variable Length
- Seven Addressing Modes — Direct, Relative, Immediate, Indexed, Extended, Implied, and Accumulator
- Variable Length Stack
- Vectored Restart
- Maskable Interrupt Vector
- Separate Nonmaskable Interrupt — Internal Registers Saved in Stack
- Six Internal Registers — Two Accumulators, Index Register, Program Counter, Stack Pointer and Condition Code Register
- Direct Memory Addressing (DMA) and Multiple Processor Capability
- Simplified Clocking Characteristics
- Clock Rates as High as 2.0 MHz
- Simple Bus Interface without TTL
- Halt and Single Instruction Execution Capability

This document contains information on a new product. Specifications and information herein are subject to change without notice.

MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage	V _{CC}	-0.3 to +7.0	V
Input Voltage	V _{in}	-0.3 to +7.0	V
Operating Temperature Range MC6800, MC68A00, MC68B00, MC6800C, MC68A00C	T _A	T _L to T _H -0 to 70 -40 to +85	°C
Storage Temperature Range	T _{stg}	-55 to +150	°C

THERMAL RESISTANCE

Rating	Symbol	Value	Unit
Plastic Package	θ _{JA}	100	°C/W
Cerdip Package		60	

POWER CONSIDERATIONS

The average chip-junction temperature, T_J, in °C can be obtained from:

$$T_J = T_A + (P_D \cdot \theta_{JA}) \quad (1)$$

where:

T_A = Ambient Temperature, °C
 θ_{JA} = Package Thermal Resistance,
 Junction-to-Ambient, °C/W

P_D = P_{INT} + P_{PORT}

P_{INT} = I_{CC} × V_{CC}, Watts — Chip Internal Power

P_{PORT} = Port Power Dissipation, Watts — User Determined

For most applications P_{PORT} < P_{INT} and can be neglected. P_{PORT} may become significant if the device is configured to drive Darlington bases or sink LED loads.

An approximate relationship between P_D and T_J (if P_{PORT} is neglected) is:

$$P_D = K \div (T_J + 273^\circ\text{C}) \quad (2)$$

Solving equations (1) and (2) for K gives:

$$K = P_D \cdot (T_A + 273^\circ\text{C}) + \theta_{JA} \cdot P_D^2 \quad (3)$$

where K is a constant pertaining to the particular part. K can be determined from equation (3) by measuring P_D (at equilibrium) for a known T_A. Using this value of K, the values of P_D and T_J can be obtained by solving equations (1) and (2) iteratively for any value of T_A.

DC ELECTRICAL CHARACTERISTICS (V_{CC} = 5.0 Vdc, ±5%, V_{SS} = 0, T_A = T_L to T_H unless otherwise noted)

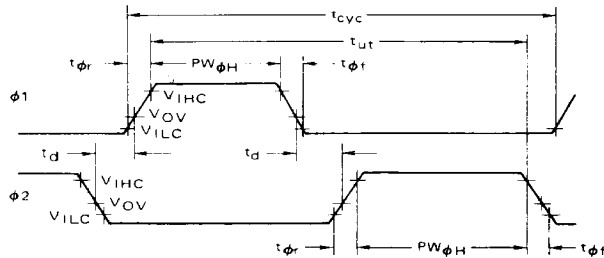
Characteristic	Symbol	Min	Typ	Max	Unit	
Input High Voltage	Logic	V _{IH}	V _{SS} + 2.0	—	V _{CC}	
	φ1, φ2	V _{IHC}	V _{CC} - 0.6	—	V _{CC} + 0.3	
Input Low Voltage	Logic	V _{IL}	V _{SS} - 0.3	—	V _{SS} + 0.8	
	φ1, φ2	V _{ILC}	V _{SS} - 0.3	—	V _{SS} + 0.4	
Input Leakage Current (V _{in} = 0 to 5.25 V, V _{CC} = Max) (V _{in} = 0 to 5.25 V, V _{CC} = 0 V to 5.25 V)	Logic	I _{in}	—	1.0	2.5	
	φ1, φ2		—	—	100	
Hi-Z Input Leakage Current (V _{in} = 0.4 to 2.4 V, V _{CC} = Max)	D0-D7	I _{Iz}	—	2.0	10	
	A0-A15, R/W		—	—	100	
Output High Voltage (I _{Load} = -205 μA, V _{CC} = Min) (I _{Load} = -145 μA, V _{CC} = Min) (I _{Load} = -100 μA, V _{CC} = Min)	D0-D7	V _{OH}	V _{SS} + 2.4	—	—	
	A0-A15, R/W, VMA		V _{SS} + 2.4	—	—	
	BA		V _{SS} + 2.4	—	—	
Output Low Voltage (I _{Load} = 1.6 mA, V _{CC} = Min)		V _{OL}	—	—	V _{SS} + 0.4	
Internal Power Dissipation (Measured at T _A = T _L)		P _{INT}	—	0.5	1.0	
Capacitance (V _{in} = 0, T _A = 25°C, f = 1.0 MHz)	φ1	C _{in}	—	25	35	
	φ2		—	45	70	
	D0-D7		—	10	12.5	
	Logic Inputs	—	—	6.5	10	
	A0-A15, R/W, VMA	C _{out}	—	—	12	pF

MC6800

CLOCK TIMING (V_{CC}=5.0 V, ±5%, V_{SS}=0, T_A=T_L to T_H unless otherwise noted)

Characteristic	Symbol	Min	Typ	Max	Unit	
Frequency of Operation	MC6800 MC68A00 MC68B00	f	0.1 0.1 0.1	— 1.5 2.0	MHz	
Cycle Time (Figure 1)	MC6800 MC68A00 MC68B00	t _{cyc}	1.000 0.666 0.500	— 10 10	μs	
Clock Pulse Width (Measured at V _{CC} -0.6 V)	φ1, φ2 - MC6800 φ1, φ2 - MC68A00 φ1, φ2 - MC68B00	PW _{φH}	400 230 180	— 9500 9500	ns	
Total φ1 and φ2 Up Time	MC6800 MC68A00 MC68B00	t _{ut}	900 600 440	— — —	ns	
Rise and Fall Time (Measured between V _{SS} +0.4 and V _{CC} -0.6)		t _r , t _f	—	—	100	ns
Delay Time or Clock Separation (Figure 1) (Measured at V _{OY} =V _{SS} +0.6 V@t _r =t _f ≤100 ns) (Measured at V _{OY} =V _{SS} +1.0 V@t _r =t _f ≤35 ns)		t _d	0 0	— —	9100 9100	ns

FIGURE 1 — CLOCK TIMING WAVEFORM



NOTES:

1. Voltage levels shown are V_L ≤ 0.4, V_H ≥ 2.4 V, unless otherwise specified.
2. Measurement points shown are 0.8 V and 2.0 V, unless otherwise noted.

READ/WRITE TIMING (Reference Figures 2 through 6, 8, 9, 11, 12 and 13)

Characteristic	Symbol	MC6800			MC68A00			MC68B00			Unit
		Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
Address Delay C = 90 pF C = 30 pF	t _{AD}	—	—	270 250	—	—	180 165	—	—	150 135	ns
Peripheral Read Access Time t _{acc} = t _{ut} - (t _{AD} + t _{DSR})	t _{acc}	530	—	—	360	—	—	250	—	—	ns
Data Setup Time (Read)	t _{DSR}	100	—	—	60	—	—	40	—	—	ns
Input Data Hold Time	t _H	10	—	—	10	—	—	10	—	—	ns
Output Data Hold Time	t _H	10	25	—	10	25	—	10	25	—	ns
Address Hold Time (Address, R/W, VMA)	t _{AH}	30	50	—	30	50	—	30	50	—	ns
Enable High Time for DBE Input	t _{EH}	450	—	—	280	—	—	220	—	—	ns
Data Delay Time (Write)	t _{DDW}	—	—	225	—	—	200	—	—	160	ns
Processor Controls											
Processor Control Setup Time	t _{PCS}	200	—	—	140	—	—	110	—	—	ns
Processor Control Rise and Fall Time	t _{PCr} , t _{PCf}	—	—	100	—	—	100	—	—	100	
Bus Available Delay	t _{BA}	—	—	250	—	—	165	—	—	135	
Hi-Z Enable	t _{TSE}	0	—	40	0	—	40	0	—	40	
Hi-Z Delay	t _{TSD}	—	—	270	—	—	270	—	—	220	
Data Bus Enable Down Time During φ1 Up Time	t _{DBE}	150	—	—	120	—	—	75	—	—	
Data Bus Enable Rise and Fall Times	t _{DBEr} , t _{DBEf}	—	—	25	—	—	25	—	—	25	



FIGURE 2 — READ DATA FROM MEMORY OR PERIPHERALS

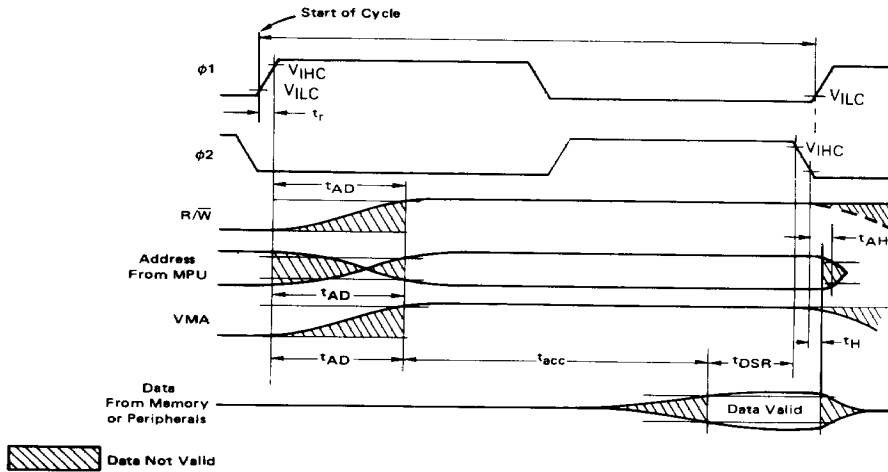
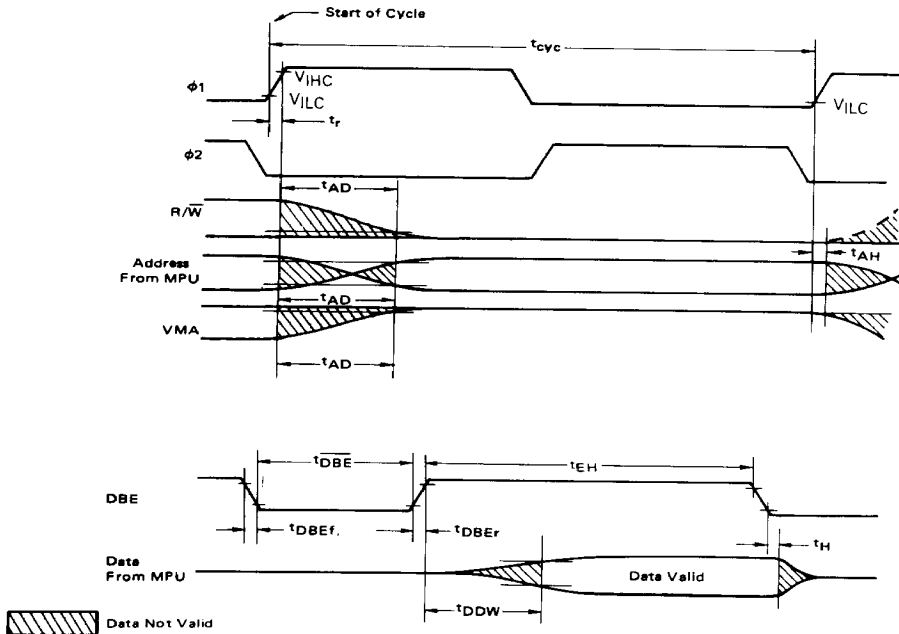


FIGURE 3 — WRITE IN MEMORY OR PERIPHERALS



NOTES:

1. Voltage levels shown are $V_L \leq 0.4$, $V_H \geq 2.4$ V, unless otherwise specified.
2. Measurement points shown are 0.8 V and 2.0 V, unless otherwise noted.

3

FIGURE 4 — TYPICAL DATA BUS OUTPUT DELAY versus CAPACITIVE LOADING (t_{DDW})

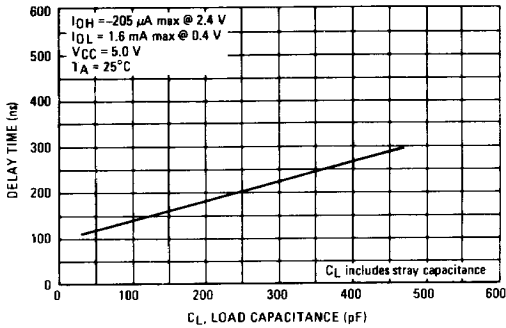


FIGURE 5 — TYPICAL READ/WRITE, VMA, AND ADDRESS OUTPUT DELAY versus CAPACITIVE LOADING (t_{AD})

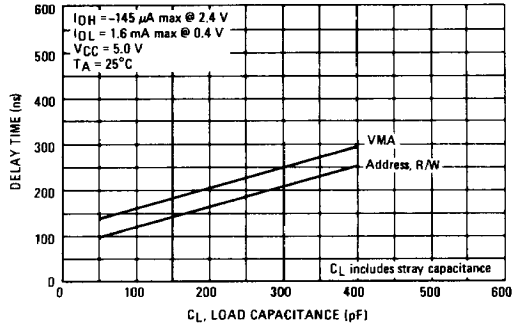
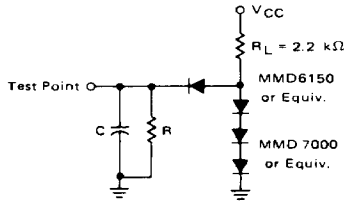


FIGURE 6 — BUS TIMING TEST LOADS



- C = 130 pF for D0-D7, E
- = 90 pF for A0-A15, R/W, and VMA (Except t_{AD2})
- = 30 pF for A0-A15, R/W, and VMA (t_{AD2} only)
- = 30 pF for BA
- R = 11.7 kΩ for D0-D7
- = 16.5 kΩ for A0-A15, R/W, and VMA
- = 24 kΩ for BA

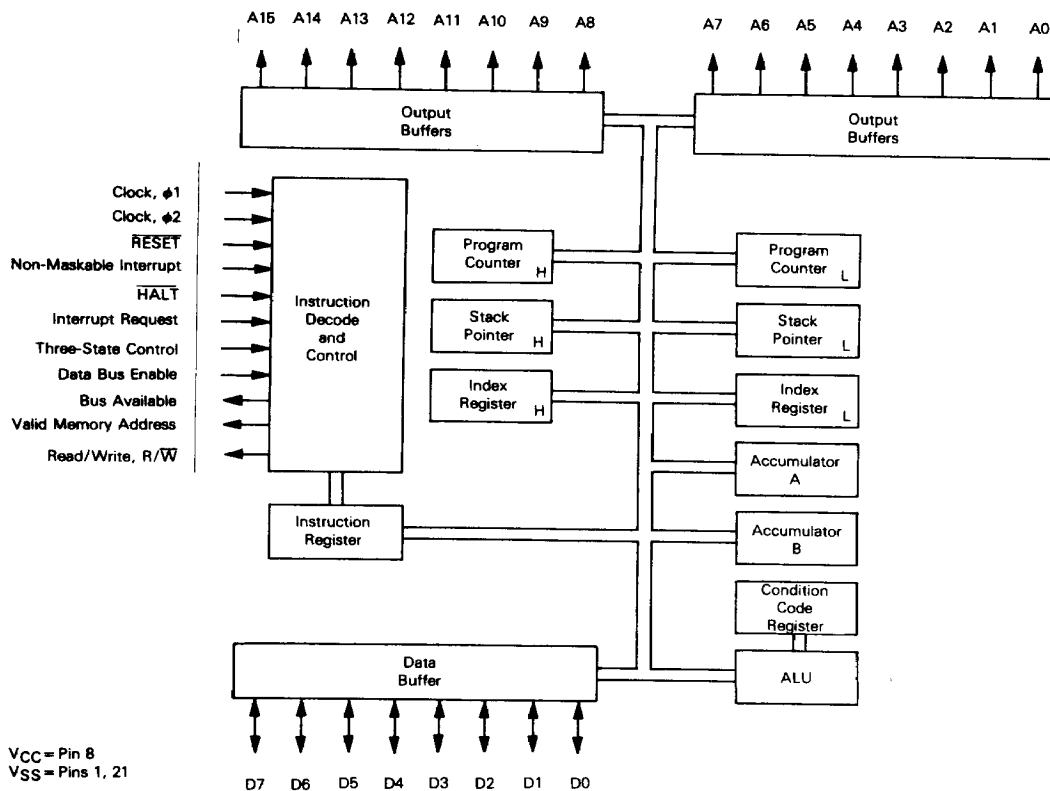
TEST CONDITIONS

The dynamic test load for the Data Bus is 130 pF and one standard TTL load as shown. The Address, R/W, and VMA outputs are tested under two conditions to allow optimum operation in both buffered and unbuffered systems. The resistor (R) is chosen to insure specified load currents during V_{OH} measurement.

Notice that the Data Bus lines, the Address lines, the Interrupt Request line, and the DBE line are all specified and tested to guarantee 0.4 V of dynamic noise immunity at both "1" and "0" logic levels.

MC6800

FIGURE 7 — EXPANDED BLOCK DIAGRAM



3

IMAGE UNAVAILABLE

■ 9004697 0743074 6T3 ■

3

FIGURE 8 — RESET TIMING

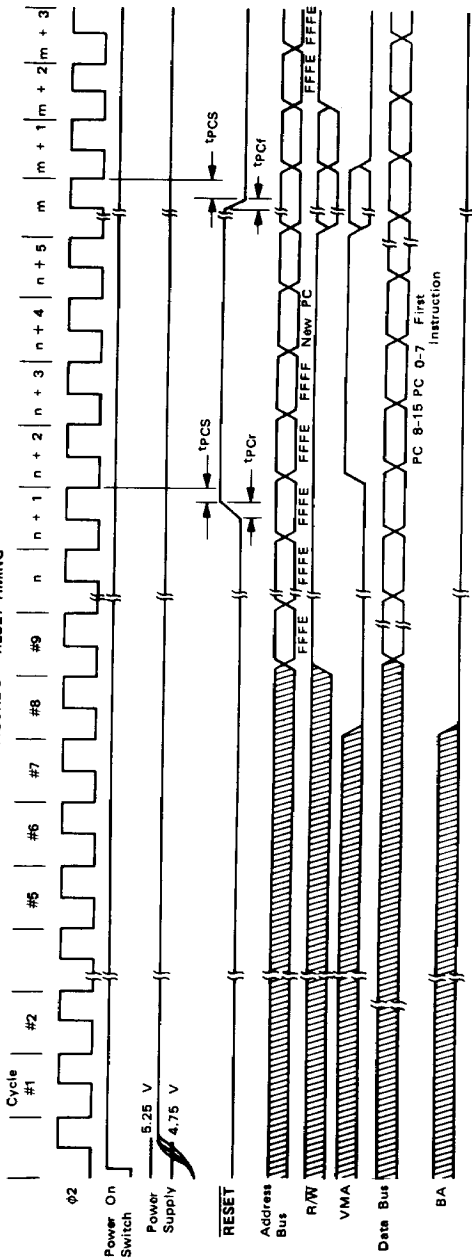
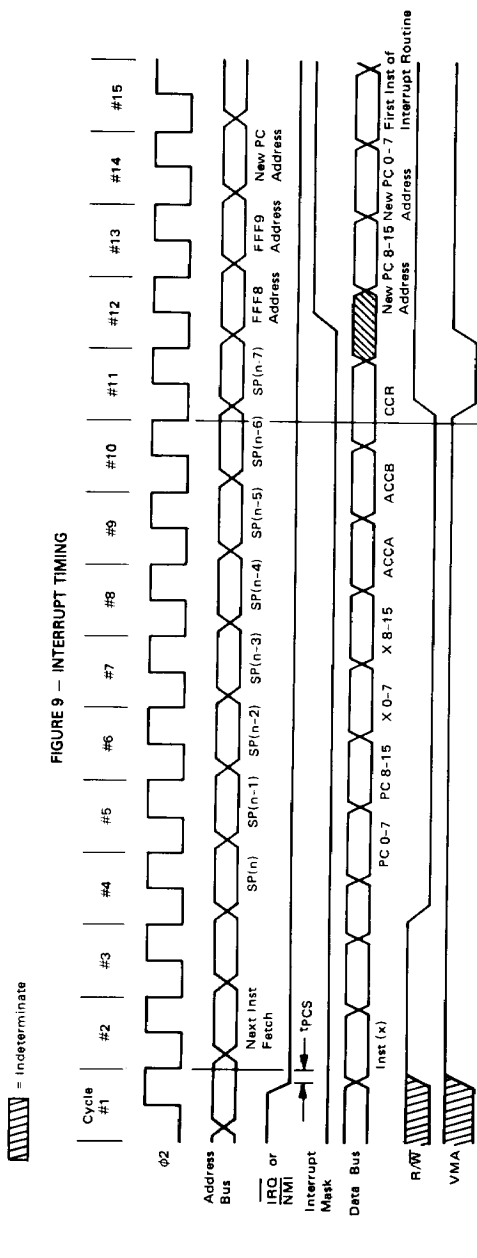


FIGURE 9 — INTERRUPT TIMING



The $\overline{\text{HALT}}$ line must be in the high state for interrupts to be serviced. Interrupts will be latched internally while $\overline{\text{HALT}}$ is low.

The $\overline{\text{IRQ}}$ has a high-impedance pullup device internal to the chip; however, a 3 k Ω external resistor to V_{CC} should be used for wire-OR and optimum control of interrupts.

Non-Maskable Interrupt (NMI) and Wait for Interrupt (WAI) — The MC6800 is capable of handling two types of interrupts: maskable ($\overline{\text{IRQ}}$) as described earlier, and non-maskable (NMI) which is an edge sensitive input. $\overline{\text{IRQ}}$ is maskable by the interrupt mask in the condition code register while 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 Figure 9 which details the MPU response to an interrupt while the MPU is executing the control program. The interrupt shown could be either $\overline{\text{IRQ}}$ or NMI and can be asynchronous with respect to $\phi 2$. The interrupt is shown going low at time t_{PCS} in cycle #1 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 NMI interrupt and from FFF8, FFF9 for an $\overline{\text{IRQ}}$ interrupt. Upon completion of the interrupt service routine, the execution of RTI will pull the PC, IX, ACCX, and CCR off the stack; the Interrupt Mask bit is restored to its condition prior to Interrupts (see Figure 10).

Figure 11 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 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.

A 3-10 k Ω external resistor to V_{CC} should be used for wire-OR and optimum control of interrupts.

MEMORY MAP FOR INTERRUPT VECTORS

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

Refer to Figure 10 for program flow for Interrupts.

Three-State Control (TSC) — When the level sensitive Three-State Control (TSC) line is a logic "1", the Address Bus and the R/ \overline{W} line are placed in a high-impedance state. VMA and BA are forced low when TSC="1" 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 $\phi 1$ (or $\phi 2$) 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 $\phi 1$ clock can be stopped for a maximum

time $PW_{\phi 1H}$ without destroying data within the MPU. TSC then can be used in a short Direct Memory Access (DMA) application.

Figure 12 shows the effect of TSC on the MPU. TSC must have its transitions at t_{TSE} (three-state enable) while holding $\phi 1$ high and $\phi 2$ low as shown. 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 $\phi 2$ is being held low since $DBE = \phi 2$. 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 90 pF may be directly driven by this active high signal.

$\overline{\text{HALT}}$ — When this level sensitive input is in the low state, all activity in the machine will be halted. This input is level sensitive.

The $\overline{\text{HALT}}$ line provides an input to the MPU to allow control of program execution by an outside source. If $\overline{\text{HALT}}$ 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 NMI or $\overline{\text{IRQ}}$ 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 RESET 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 RESET is low. As soon as the RESET line goes high, the MPU will go to locations FFFE and FFFF for the address of the reset routine.

Figure 13 shows the timing relationships involved when halting the MPU. The instruction illustrated is a one byte, 2 cycle instruction such as CLRA. When $\overline{\text{HALT}}$ goes low, the MPU will halt after completing execution of the current instruction. The transition of $\overline{\text{HALT}}$ must occur t_{PCS} before the trailing edge of $\phi 1$ of the last cycle of an instruction (point A of Figure 13). $\overline{\text{HALT}}$ must not go low any time later than the minimum t_{PCS} specified.

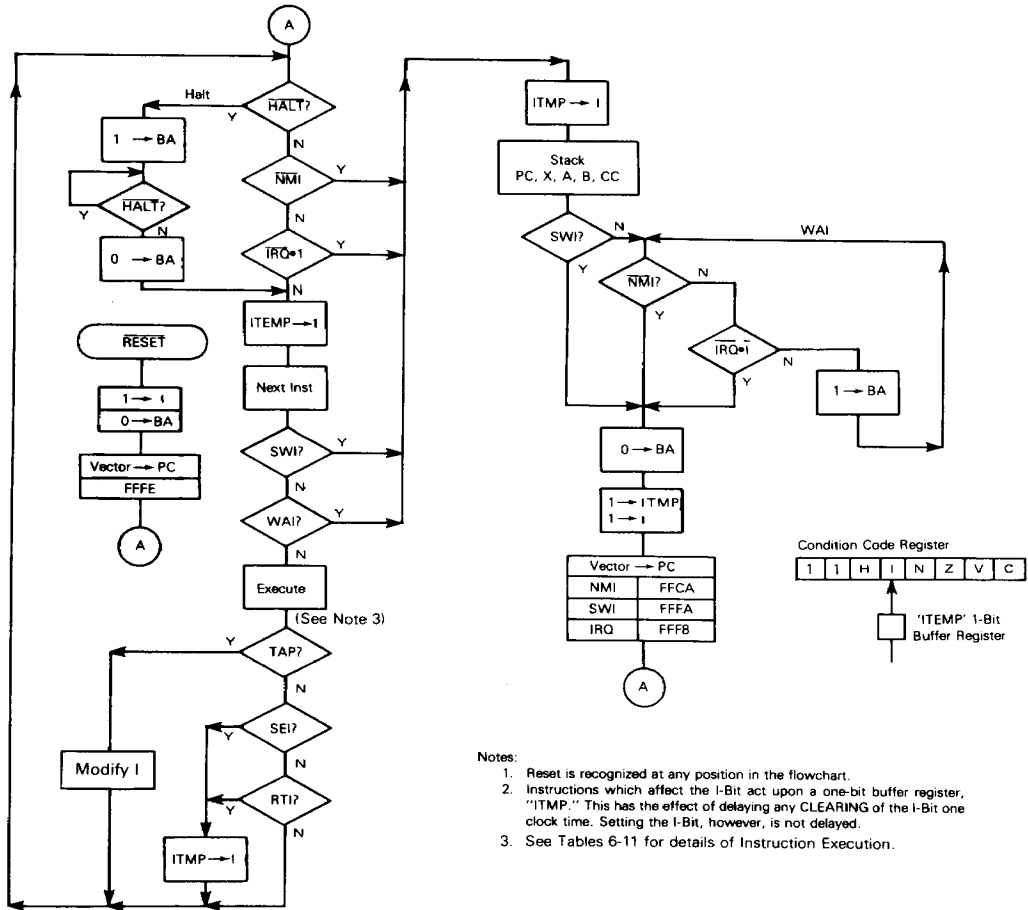
The fetch of the OP code by the MPU is the first cycle of the instruction. If $\overline{\text{HALT}}$ had not been low at Point A but went low during $\phi 2$ of that cycle, the MPU would have halted after completion of the following instruction. BA will go high by time t_{BA} (bus available delay time) 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.

3

To debug programs it is advantageous to step through programs instruction by instruction. To do this, HALT must be brought high for one MPU cycle and then returned low as shown at point B of Figure 13. Again, the transitions of HALT must occur tPCS before the trailing edge of $\phi 1$. BA will go low at tBA after the leading edge of the next $\phi 1$, indicating that the Address Bus, Data Bus, VMA and R/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 tBA 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.

FIGURE 10 — MPU FLOWCHART



Notes:

1. Reset is recognized at any position in the flowchart.
2. Instructions which affect the I-Bit act upon a one-bit buffer register, "ITEMP." This has the effect of delaying any CLEARING of the I-Bit one clock time. Setting the I-Bit, however, is not delayed.
3. See Tables 6-11 for details of Instruction Execution.

FIGURE 11 — WAIT INSTRUCTION TIMING

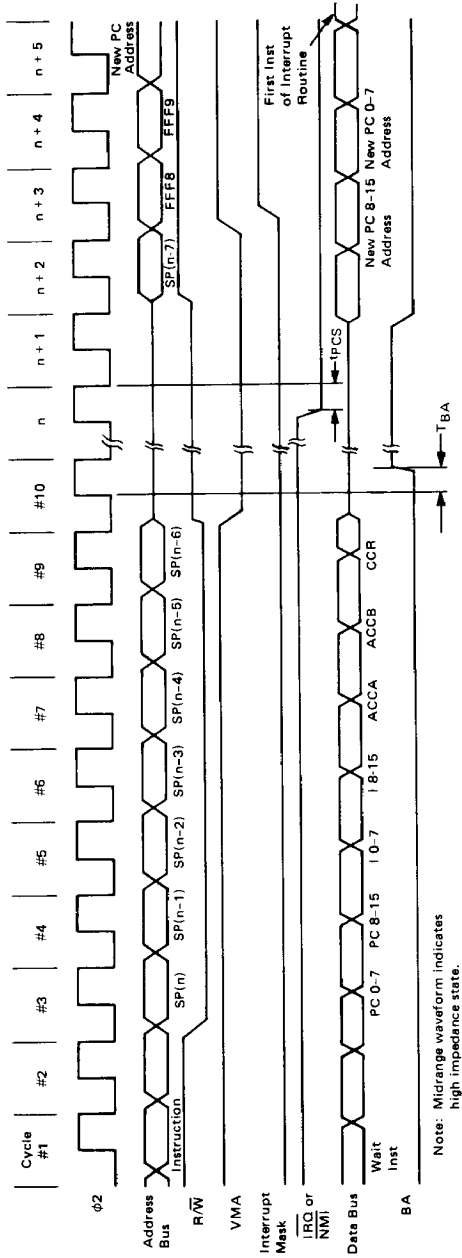


FIGURE 12 — THREE-STATE CONTROL TIMING

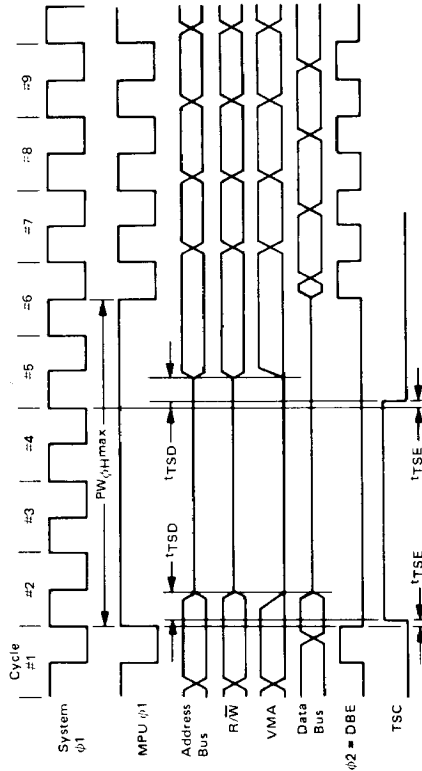
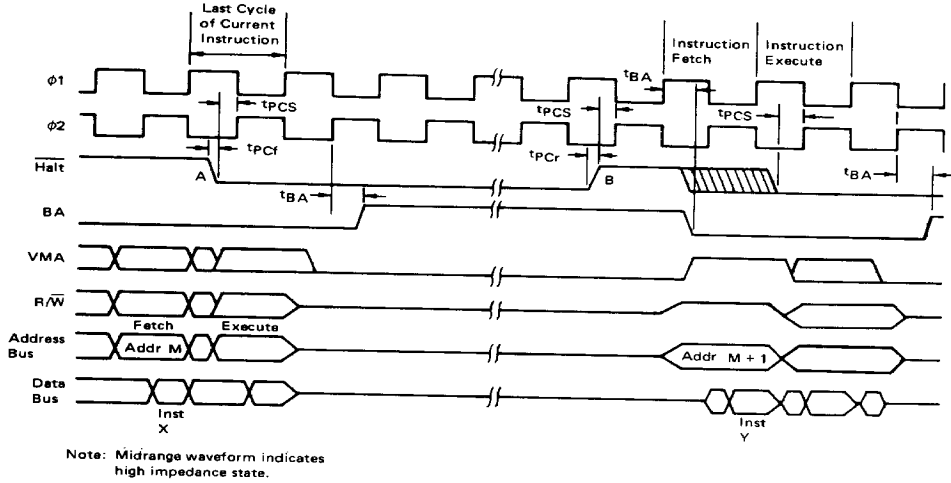


FIGURE 13 — $\overline{\text{HALT}}$ AND SINGLE INSTRUCTION EXECUTION FOR SYSTEM DEBUG



3

MPU REGISTERS

The MPU has three 16-bit registers and three 8-bit registers available for use by the programmer (Figure 14).

Program Counter — The program counter is a two byte (16 bits) register that points to the current program address.

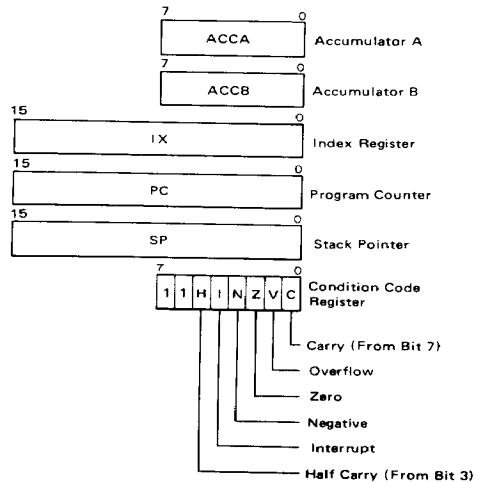
Stack Pointer — The stack pointer is a two byte register that contains the address of the next available location in an external push-down/pop-up stack. This stack is normally a random access Read/Write memory that may have any location (address) that is convenient. In those applications that require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

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

Condition Code Register — 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 ones.

FIGURE 14 — PROGRAMMING MODEL OF THE MICROPROCESSING UNIT



MPU INSTRUCTION SET

The MC6800 instructions are described in detail in the M6800 Programming Manual. This Section will provide a brief introduction and discuss their use in developing MC6800 control programs. The MC6800 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 1. 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 the second and third bytes contain(s) an operand, an address, or information from which an address is obtained during execution.

Microprocessor instructions are often divided 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 MC6800 performs the same operation on both its internal accumulators and the external memory locations. In addition, the MC6800 interface adapters (PIA and ACIA) 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 MC6800's instruction set: (1) Accumulator and memory operations; (2) Program control operations; (3) Condition Code Register operations.

TABLE 1 — HEXADECEMAL VALUES OF MACHINE CODES

00	.	40	NEG	A	80	SUB	A	IMM	C0	SUB	B	IMM	
01	NOP	41	.	.	81	CMP	A	IMM	C1	CMP	B	IMM	
02	.	42	.	.	82	SBC	A	IMM	C2	SBC	B	IMM	
03	.	43	COM	A	83	.	.	.	C3	.	.	.	
04	.	44	LSR	A	84	AND	A	IMM	C4	AND	B	IMM	
05	.	45	.	.	85	BIT	A	IMM	C5	BIT	B	IMM	
06	TAP	46	ROR	A	86	LOA	A	IMM	C6	LDA	B	IMM	
07	TPA	47	ASR	A	87	.	.	.	C7	.	.	.	
08	INX	48	ASL	A	88	EOR	A	IMM	C8	EOR	B	IMM	
09	DEX	49	ROL	A	89	ADC	A	IMM	C9	ADC	B	IMM	
0A	CLY	4A	DEC	A	8A	ORA	A	IMM	CA	ORA	B	IMM	
0B	SEV	4B	.	.	8B	ADD	A	IMM	CB	ADD	B	IMM	
0C	CLC	4C	INC	A	8C	CPX	A	IMM	CC	.	.	.	
0D	SEC	4D	TST	A	8D	BSR	.	.	CD	.	.	.	
0E	CLI	4E	.	.	8E	LDS	.	.	CE	LDX	.	IMM	
0F	SEI	4F	CLR	A	8F	.	.	.	CF	.	.	.	
10	SBA	50	NEG	B	90	SUB	A	DIR	D0	SUB	B	DIR	
11	CBA	51	.	.	91	CMP	A	DIR	D1	CMP	B	DIR	
12	.	52	.	.	92	SBC	A	DIR	D2	SBC	B	DIR	
13	.	53	COM	B	93	.	.	.	D3	.	.	.	
14	.	54	LSR	B	94	AND	A	DIR	D4	AND	B	DIR	
15	.	55	.	.	95	BIT	A	DIR	D5	BIT	B	DIR	
16	TAB	56	ROR	B	96	LDA	A	DIR	D6	LDA	B	DIR	
17	TBA	57	ASR	B	97	STA	A	DIR	D7	STA	B	DIR	
18	.	58	ASL	B	98	EOR	A	DIR	D8	EOR	B	DIR	
19	DAA	59	ROL	B	99	ADC	A	DIR	D9	ADC	B	DIR	
1A	.	5A	DEC	B	9A	ORA	A	DIR	DA	ORA	B	DIR	
1B	ABA	5B	.	.	9B	ADD	A	DIR	DB	ADD	B	DIR	
1C	.	5C	INC	B	9C	CPX	.	.	DC	.	.	.	
1D	.	5D	TST	B	9D	.	.	.	DD	.	.	.	
1E	.	5E	.	.	9E	LOS	.	DIR	DE	LDX	.	DIR	
1F	.	5F	CLR	B	9F	STS	.	DIR	DF	STX	.	DIR	
20	BRA	REL	60	NEG	IND	A0	SUB	A	IND	E0	SUB	B	IND
21	.	.	61	.	.	A1	CMP	A	IND	E1	CMP	B	IND
22	BHI	REL	62	.	.	A2	SBC	A	IND	E2	SBC	B	IND
23	BLS	REL	63	COM	IND	A3	.	.	E3	.	.	.	
24	BCC	REL	64	LSR	IND	A4	AND	A	IND	E4	AND	B	IND
25	BCS	REL	65	.	.	A5	BIT	A	IND	E5	BIT	B	IND
26	BNE	REL	66	ROR	IND	A6	LDA	A	IND	E6	LDA	B	IND
27	BEQ	REL	67	ASR	IND	A7	STA	A	IND	E7	STA	B	IND
28	BVC	REL	68	ASL	IND	A8	EOR	A	IND	E8	EOR	B	IND
29	BVS	REL	69	ROL	IND	A9	ADC	A	IND	E9	ADC	B	IND
2A	BPL	REL	6A	DEC	IND	AA	ORA	A	IND	EA	ORA	B	IND
2B	BMI	REL	6B	.	.	AB	ADD	A	IND	EB	ADD	B	IND
2C	BGE	REL	6C	INC	IND	AC	CPX	.	IND	EC	.	.	.
2D	BLT	REL	6D	TST	IND	AD	JSR	.	IND	ED	.	.	.
2E	BGT	REL	6E	JMP	IND	AE	LDS	.	IND	EE	.	.	.
2F	BLE	REL	6F	CLR	EXT	AF	STS	.	IND	EF	STX	.	IND
30	TSX	.	70	NEG	EXT	B0	SUB	A	EXT	F0	SUB	B	EXT
31	INS	.	71	.	.	B1	CMP	A	EXT	F1	CMP	B	EXT
32	PUL	A	72	.	.	B2	SBC	A	EXT	F2	SBC	B	EXT
33	PUL	B	73	COM	EXT	B3	.	.	F3	.	.	.	
34	OES	.	74	LSR	EXT	B4	AND	A	EXT	F4	AND	B	EXT
35	TXS	.	75	.	.	B5	BIT	A	EXT	F5	BIT	B	EXT
36	PSH	A	76	ROR	EXT	B6	LDA	A	EXT	F6	LDA	B	EXT
37	PSH	B	77	ASR	EXT	B7	STA	A	EXT	F7	STA	B	EXT
38	.	.	78	ASL	EXT	B8	EOR	A	EXT	F8	EOR	B	EXT
39	RTS	.	79	ROL	EXT	B9	ADC	A	EXT	F9	ADC	B	EXT
3A	.	.	7A	DEC	EXT	BA	ORA	A	EXT	FA	ORA	B	EXT
3B	RTI	.	7B	.	.	BB	ADD	A	EXT	FB	ADD	B	EXT
3C	.	.	7C	INC	EXT	BC	CPX	.	EXT	FC	.	.	.
3D	.	.	7D	TST	EXT	BD	JSR	.	EXT	FD	.	.	.
3E	WAI	.	7E	JMP	EXT	BE	LDS	.	EXT	FE	LDX	.	EXT
3F	SWI	.	7F	CLR	EXT	BF	STS	.	EXT	FF	STX	.	EXT

Notes: 1. Addressing Modes:

A = Accumulator A
 B = Accumulator B
 REL = Relative
 IND = Indexed
 IMM = Immediate
 DIR = Direct

2. Unassigned code indicated by " * * * ".

3

PROGRAM CONTROL OPERATIONS

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

Index Register/Stack Pointer Operations

The instructions for direct operation on the MPU's Index Register and Stack Pointer are summarized in Table 3. 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 M6800 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 MC6800 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 Figures 15 and 16. 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 that 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 saved on the stack as shown in Figures 18 through 20. The stack is decremented after each byte of the return address is pushed onto the stack. For both of 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 Figure 21.

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 (IRQ) and non-maskable (NMI) hardware interrupts all cause the MPU's internal registers (except for the Stack Pointer itself) to be stacked as shown in Figure 23. MPU status is restored by the Return from Interrupt, RTI, as shown in Figure 22.

Jump and Branch Operation

The Jump and Branch instructions are summarized in Table 4. These instructions are used to control the transfer or 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.

TABLE 3 — INDEX REGISTER AND STACK POINTER INSTRUCTIONS

POINTER OPERATIONS	MNEMONIC	IMMED		DIRECT		INDEX		EXTND		IMPLIED		BOOLEAN/ARITHMETIC OPERATION	COND. CODE REG.							
		OP	~	≠	OP	~	≠	OP	~	≠	OP		~	≠	H	I	N	Z	V	C
Compare Index Reg	CPX	8C	3	3	9C	4	2	AC	6	2	8C	5	3	$X_H - M, X_L \rightarrow (M + 1)$	•	•	①	1	②	•
Decrement Index Reg	DEX													$X - 1 \rightarrow X$	•	•	•	•	•	•
Decrement Stack Ptr	DES													$SP - 1 \rightarrow SP$	•	•	•	•	•	•
Increment Index Reg	INX													$X + 1 \rightarrow X$	•	•	•	•	•	•
Increment Stack Ptr	INS													$SP + 1 \rightarrow SP$	•	•	•	•	•	•
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 Ptr	LDS	BE	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 Ptr	STS				9F	5	2	AF	7	2	BF	6	3	$SP_H \rightarrow M, SP_L \rightarrow (M + 1)$	•	•	③	•	R	•
Index Reg → Stack Ptr	TXS													$X - 1 \rightarrow SP$	•	•	•	•	•	•
Stack Ptr → Index Reg	TSX													$SP + 1 \rightarrow X$	•	•	•	•	•	•

- 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 16 — STACK OPERATION, PUSH INSTRUCTION

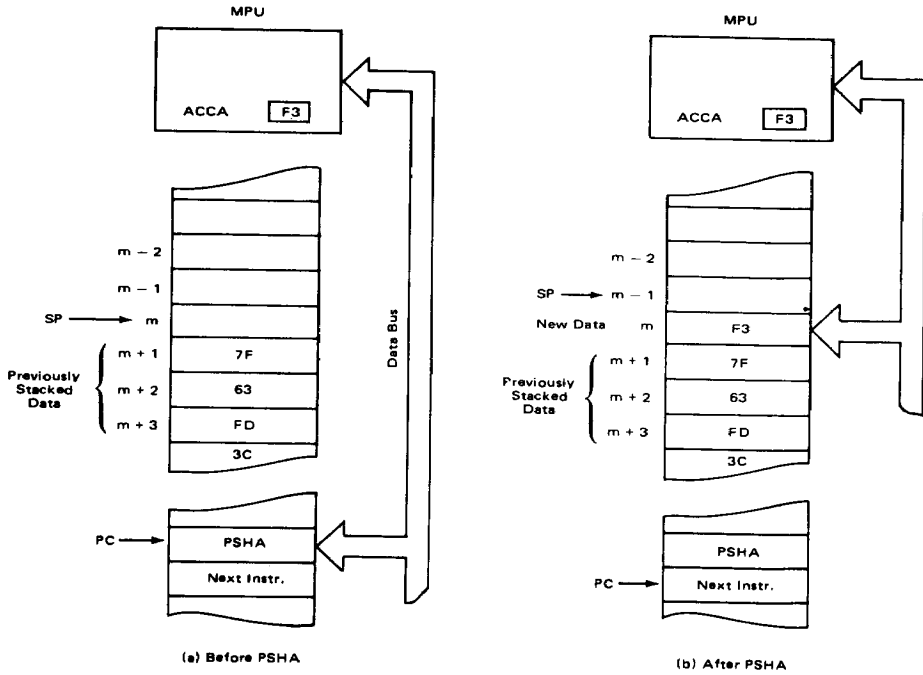


FIGURE 16 — STACK OPERATION, PULL INSTRUCTION

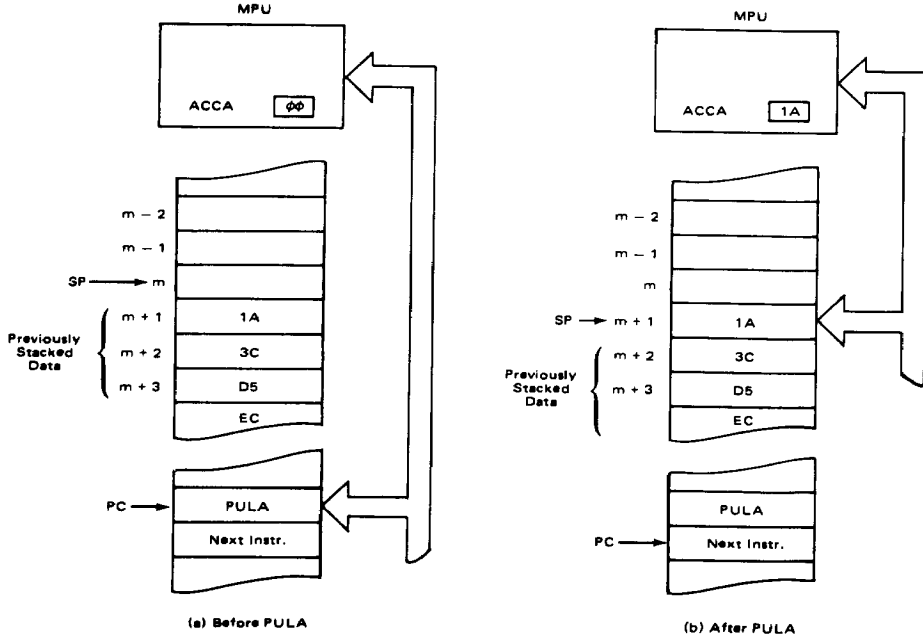


FIGURE 18 — PROGRAM FLOW FOR BSR

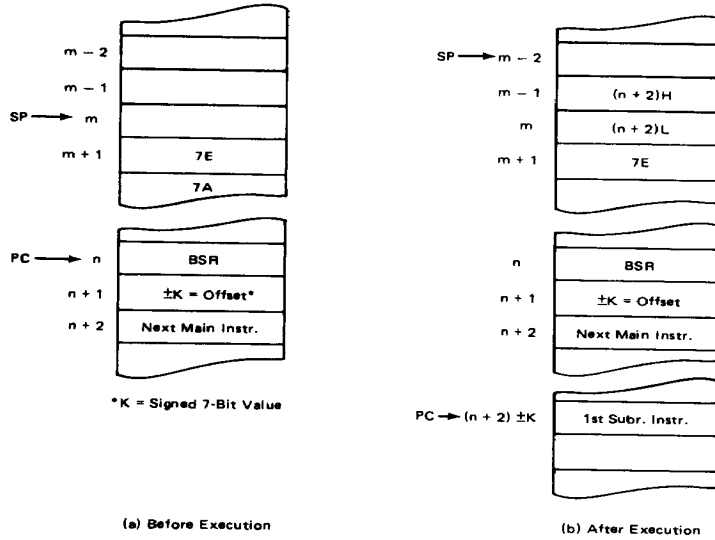


FIGURE 19 — PROGRAM FLOW FOR JSR (EXTENDED)

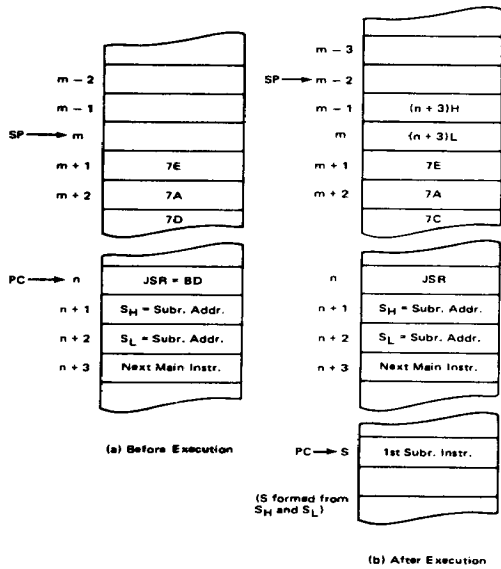


FIGURE 20 — PROGRAM FLOW FOR JSR (INDEXED)

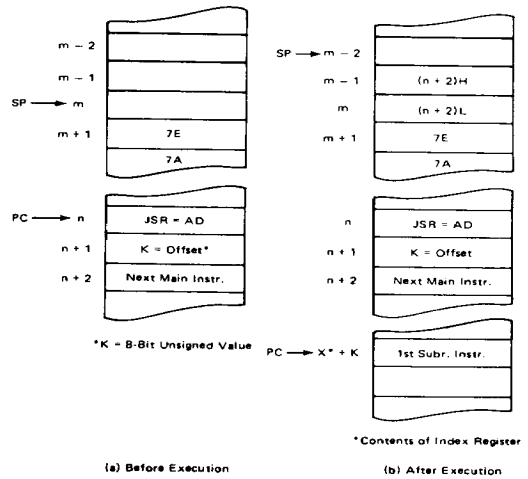


FIGURE 21 — PROGRAM FLOW FOR RTS

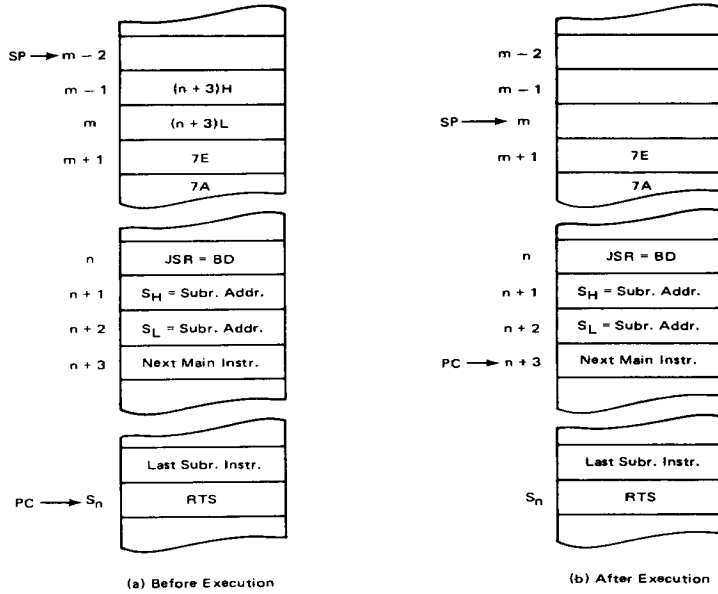
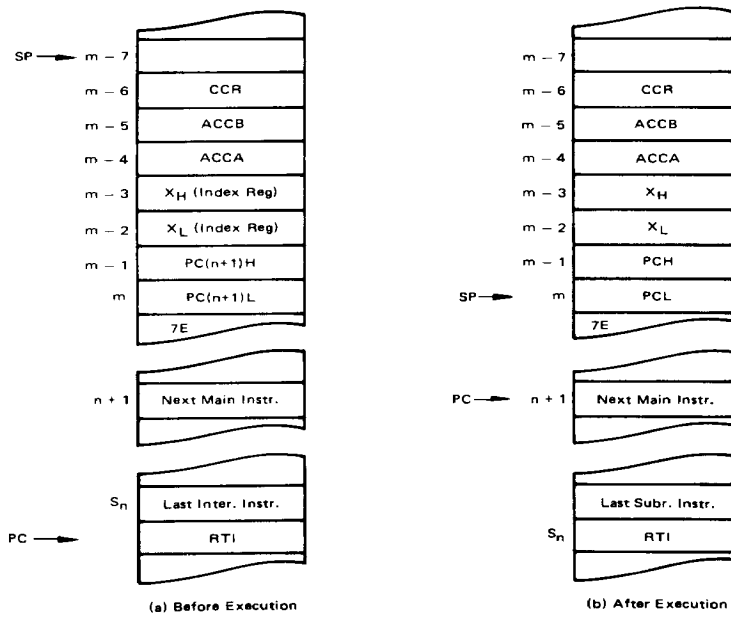
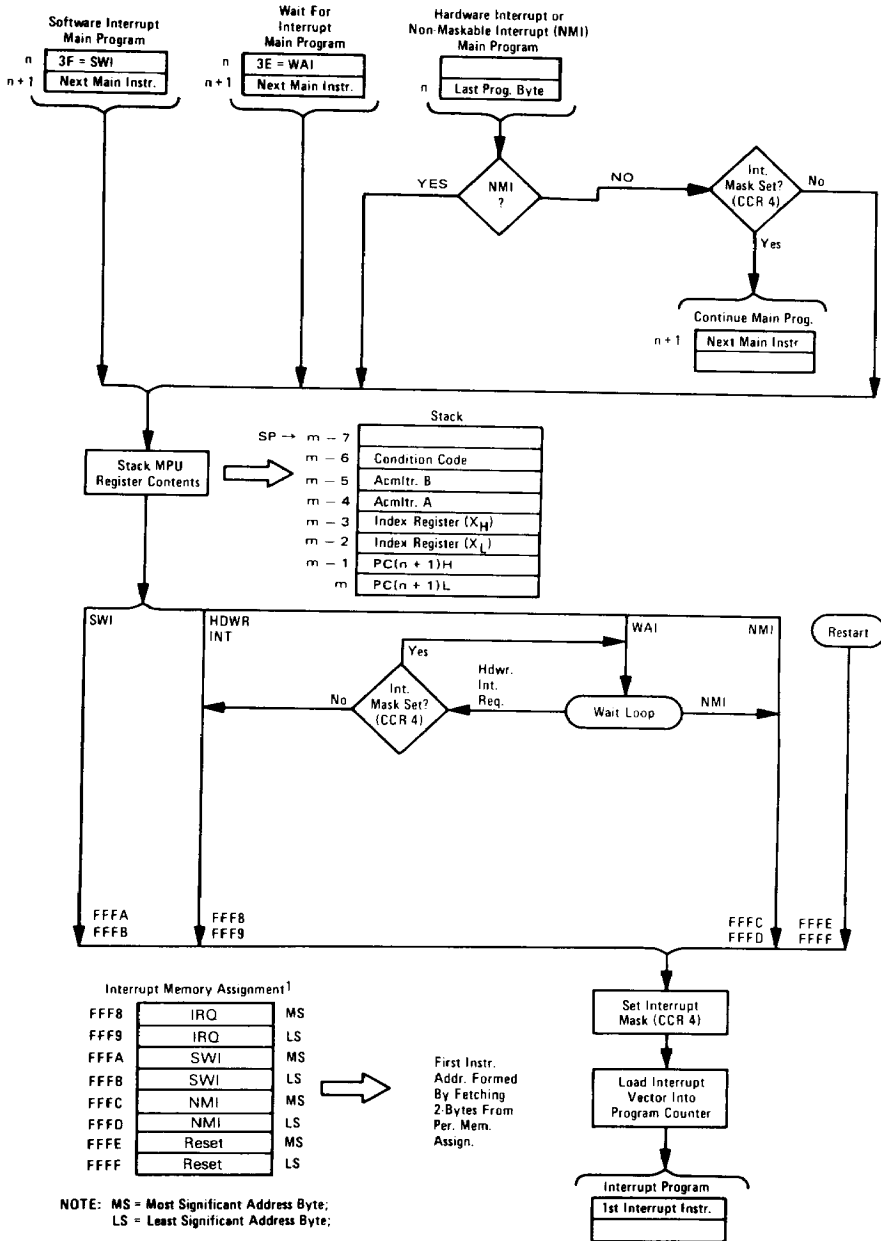


FIGURE 22 — PROGRAM FLOW FOR RTI



3

FIGURE 23 — PROGRAM FLOW FOR INTERRUPTS



3

FIGURE 24 — CONDITIONAL BRANCH INSTRUCTIONS

BMI :	N = 1 ;	BEQ :	Z = 1 ;
BPL :	N = ϕ ;	BNE :	Z = ϕ ;
BVC :	V = ϕ ;	BCC :	C = ϕ ;
BVS :	V = 1 ;	BCS :	C = 1 ;
BHI :	C + Z = ϕ ;	BLT :	N \oplus V = 1 ;
BLS :	C + Z = 1 ;	BGE :	N \oplus V = ϕ ;
BLE :	Z + (N \oplus V) = 1 ;		
BGT :	Z + (N \oplus V) = ϕ ;		

The conditional branch instructions, Figure 24, 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 zero. 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) to 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 \oplus V = 1 and N \oplus 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 zero.

The last pair, Branch On Less Than Or Equal Zero (BLE) and Branch On Greater Than Zero (BGT) test the status bits for Z \oplus (N + V) = 1 and Z \oplus (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 zero. Conversely, BGT is similar to BGE except that no branch will occur following a zero 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 Figure 25.

The instructions shown in Table 5 are available to the user for direct manipulation of the CCR.

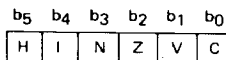
A CLI-WAI instruction sequence operated properly, with early MC6800 processors, only if the preceding instruction was odd (Least Significant Bit = 1). Similarly it was advisable

to precede any SEI instruction with an odd opcode — such as NOP. These precautions are not necessary for MC6800 processors indicating manufacture in November 1977 or later.

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

3

FIGURE 25 — CONDITION CODE REGISTER BIT DEFINITION



- 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 or software interrupt or SEI instruction; cleared by CLI instruction. (Normally not used in arithmetic operations.) Restored to a zero as a result of an RTI instruction if I_m stored on the stacked is low.
- 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.

TABLE 5 — CONDITION CODE REGISTER INSTRUCTIONS

OPERATIONS	MNEMONIC	IMPLIED		BOOLEAN OPERATION	COND. CODE REG.					
		DP	~ =		5	4	3	2	1	0
					H	I	N	Z	V	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
Accmtr A → CCR	TAP	06	2 1	A - CCR	<div style="border: 1px solid black; padding: 2px; display: inline-block;"> 1 </div>					
CCR → Accmtr A	TPA	07	2 1	CCR → A						

R = Reset
 S = Set
 ● = Not affected

① (ALL) Set according to the contents of Accumulator A.

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 M6800 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.

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 inherent 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 (hexadecimal 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 assemble 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 Figure 26.

Inherent (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.

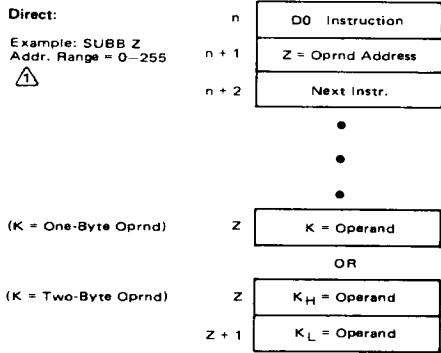
The addition instruction, ADD, provides an example of dual addressing in the operand field:

```

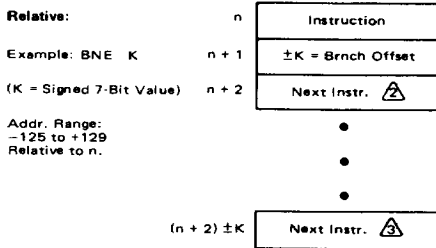
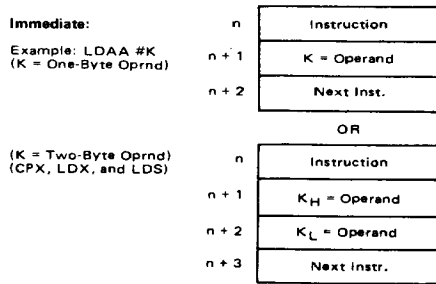
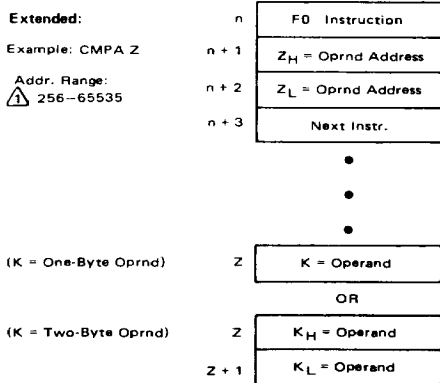
Operator Operand      Comment
  ADDA  MEM12  ADD CONTENTS OF MEM12 TO ACCA
or
  ADDB  MEM12  ADD CONTENTS OF MEM12 TO ACCB
    
```

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:

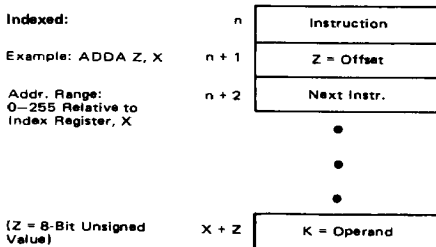
FIGURE 26 — ADDRESSING MODE SUMMARY



⚠ If Z ≤ 255, Assembler Select Direct Mode
 ⚠ If Z > 255, Extended Mode is selected



⚠ If Brnch Tst False, ⚡ If Brnch Tst True.



	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 27 and 28. 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 inherent mode is shown in Table 6.

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 Figure 29.

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 to 255. Since Compare Index Register (CPX), Load Index Register (LDX), and Load Stack Pointer (LDS), require 16-bit values, the immediate mode for these three instructions require two-byte operands. In the Immediate addressing

mode, the "address" of the operand is effectively the memory location immediately following the instruction itself. Table 7 shows the cycle-by-cycle operation for the immediate addressing mode.

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 through 255; a two byte operand is generated for Extended addressing, enabling the MPU to reach the remaining memory locations, 256 through 65535. An example of Direct addressing and its effect on program flow is illustrated in Figure 30.

The MPU, after encountering the opcode for the instruction 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. Table 8 shows the cycle-by-cycle operation for the direct mode of addressing.

Extended addressing, Figure 31, 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 any place 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 9 for Extended Addressing.

FIGURE 27 — INHERENT ADDRESSING

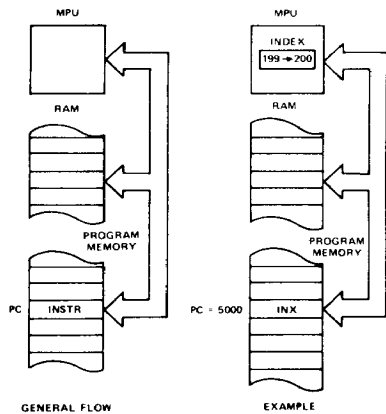
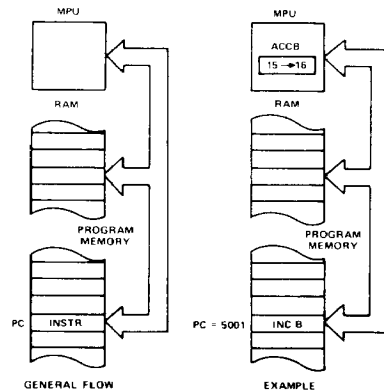


FIGURE 28 — ACCUMULATOR ADDRESSING



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 Figure 32). 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 results 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 bytes 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:

$$(PC + 2) - 127 \leq D \leq (PC + 2) + 127$$

or

$$PC - 125 \leq D \leq PC + 129$$

that is, the destination of the branch instruction must be within -125 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 Figure 32, 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 Figure 32). 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 10 for relative addressing.

Indexed Addressing Mode — With Indexed addressing, the numerical address is variable and depends 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



TABLE 6 — INHERENT MODE CYCLE-BY-CYCLE OPERATION

Address Mode and Instructions	Cycles	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 2	1 1	Op Code Address Op Code Address + 1	1 1	Op Code Op Code of Next Instruction
DES DEX INS INX	4	1 2 3 4	1 1 0 0	Op Code Address Op Code Address + 1 Previous Register Contents New Register Contents	1 1 1 1	Op Code Op Code of Next Instruction Irrelevant Data (Note 1) Irrelevant Data (Note 1)
PSH	4	1 2 3 4	1 1 1 0	Op Code Address Op Code Address + 1 Stack Pointer Stack Pointer - 1	1 1 0 1	Op Code Op Code of Next Instruction Accumulator Data Accumulator Data
PUL	4	1 2 3 4	1 1 0 1	Op Code Address Op Code Address + 1 Stack Pointer Stack Pointer + 1	1 1 1 1	Op Code Op Code of Next Instruction Irrelevant Data (Note 1) Operand Data from Stack
TSX	4	1 2 3 4	1 1 0 0	Op Code Address Op Code Address + 1 Stack Pointer New Index Register	1 1 1 1	Op Code Op Code of Next Instruction Irrelevant Data (Note 1) Irrelevant Data (Note 1)
TXS	4	1 2 3 4	1 1 0 0	Op Code Address Op Code Address + 1 Index Register New Stack Pointer	1 1 1 1	Op Code Op Code of Next Instruction Irrelevant Data Irrelevant Data
RTS	5	1 2 3 4 5	1 1 0 1 1	Op Code Address Op Code Address + 1 Stack Pointer Stack Pointer + 1 Stack Pointer + 2	1 1 1 1 1	Op Code Irrelevant Data (Note 2) Irrelevant Data (Note 1) Address of Next Instruction (High Order Byte) Address of Next Instruction (Low Order Byte)

TABLE 6 — INHERENT MODE CYCLE-BY-CYCLE OPERATION (CONTINUED)

Address Mode and Instructions	Cycles	Cycle #	VMA Line	Address Bus	R/W Line	Data Bus
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/W, and Data Bus are all in the high impedance state.

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, DEC, 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 Figure 33.

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 11 shows the cycle-by-cycle operation for the Indexed Mode of Addressing.

FIGURE 29 — IMMEDIATE ADDRESSING MODE

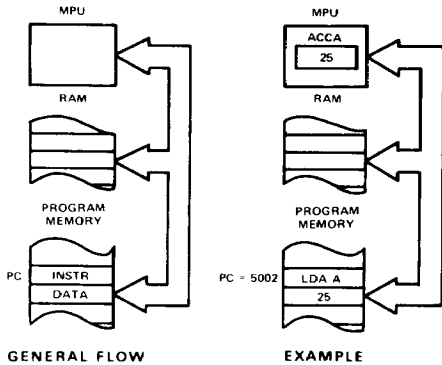


FIGURE 30 — DIRECT ADDRESSING MODE

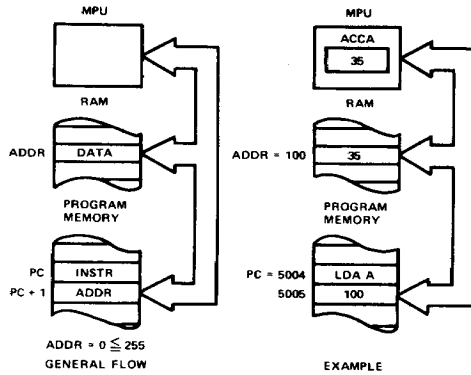


TABLE 7 — IMMEDIATE MODE CYCLE-BY-CYCLE OPERATION

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

TABLE 8 — DIRECT MODE CYCLE-BY-CYCLE OPERATION

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

3

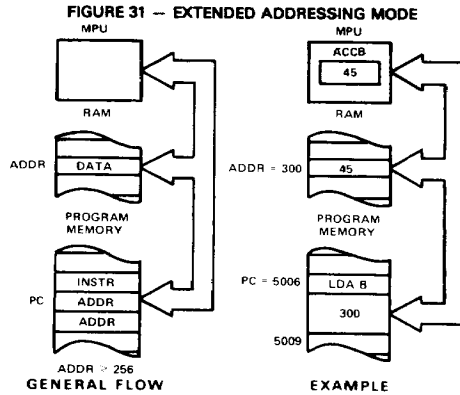


TABLE 9 -- EXTENDED MODE CYCLE-BY-CYCLE

Address Mode and Instructions	Cycles	Cycle #	VMA Line	Address Bus	R/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 32 — RELATIVE ADDRESSING MODE

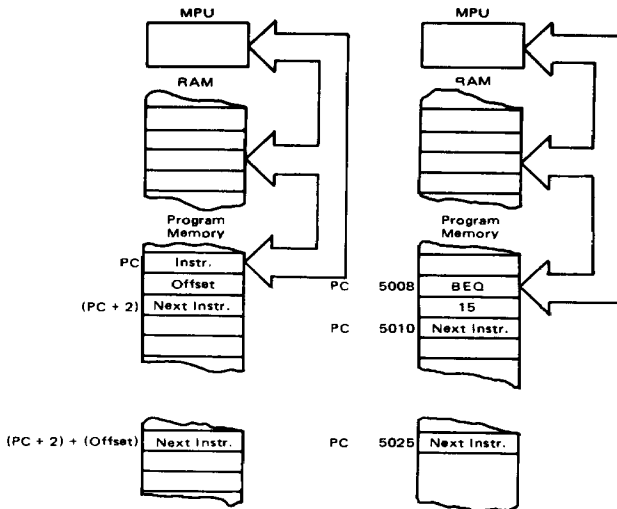


FIGURE 33 — INDEXED ADDRESSING MODE

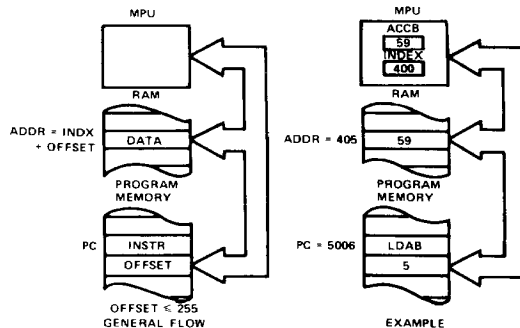


TABLE 10 — RELATIVE MODE CYCLE-BY-CYCLE OPERATION

Address Mode and Instructions	Cycles	Cycle #	VMA Line	Address Bus	R/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.

TABLE 11 — INDEXED MODE CYCLE-BY-CYCLE

Address Mode and Instructions	Cycles	Cycle #	VMA Line	Address Bus	R/W Line	Data Bus
INDEXED						
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 ORA 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 Order 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.

ORDERING INFORMATION

Package Type	Frequency (MHz)	Temperature	Order Number
Cerdip S Suffix	1.0	0°C to 70°C	MC6800S
	1.0	-40°C to 85°C	MC6800CS
	1.5	0°C to 70°C	MC68A00S
	1.5	-40°C to 85°C	MC68A00CS
	2.0	0°C to 70°C	MC68B00S
Plastic P Suffix	1.0	0°C to 70°C	MC6800P
	1.0	-40°C to 85°C	MC6800CP
	1.5	0°C to 70°C	MC68A00P
	1.5	-40°C to 85°C	MC68A00CP
	2.0	0°C to 70°C	MC68B00P

PIN ASSIGNMENT

