FAIRCHILD
A Schlumberger Company

F6800/F68A00/F68B00

8-Bit Microprocessing Unit

Microprocessor Product

Description

The F6800 is a monolithic 8-bit microprocessing unit (MPU) forming the central control function for the Fairchild F6800 family. Compatible with TTL, the F6800, as with all F6800 system parts, requires only one +5.0 V power supply and no external TTL devices for bus interface.

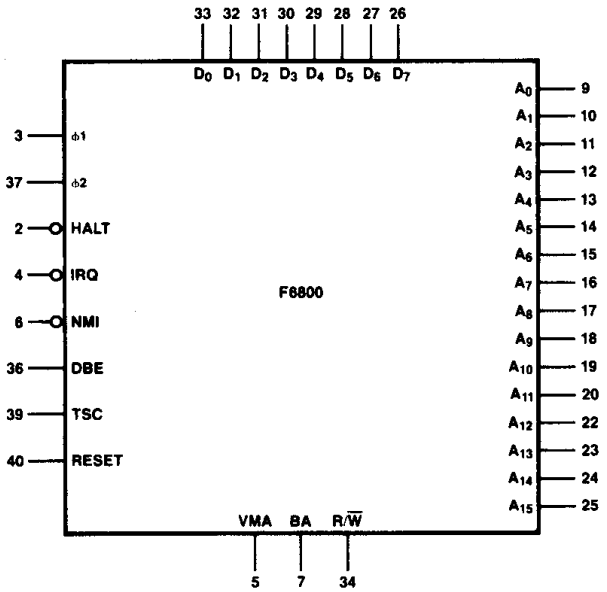
The F6800 is capable of addressing 65K bytes of memory with its 16-bit address lines. The 8-bit data bus is bidirectional as well as 3-state, making direct memory addressing and multiprocessing applications realizable.

- **8-Bit Parallel Processing**
- **Bidirectional Data Bus**
- **16-Bit Address Bus — 65K Bytes of Addressing**
- **72 Instructions — Variable Length**
- **7 Addressing Modes — Direct, Relative, Immediate, Indexed, Extended, Implied and Accumulator**
- **Variable Length Stack**
- **Vectored Restart**
- **Maskable Interrupt Vector**
- **Separate Non-Maskable Interrupt — Internal Registers Saved in Stack**
- **6 Internal Registers — 2 Accumulators, Index Register, Program Counter, Stack Pointer, and Condition Code Register**
- **Direct Memory Addressing (DMA) and Multiple Processor Capability**
- **Simplified Clocking Characteristics**
- **Clock Rates 1 MHz (F6800), 1.5 MHz (F68A00), and 2 MHz (F68B00)**
- **Simple Bus Interface Without TTL**
- **Halt and Single Instruction Execution Capability**

Pin Names

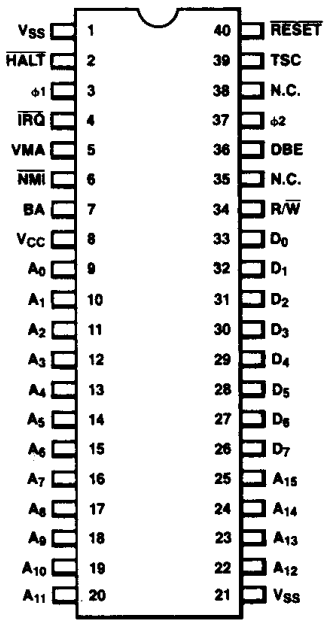
D_0-D_7	Bidirectional Data Bus
$\overline{HALT}$	Halt Input
$\phi 1$, $\phi 2$	Clock Inputs
$\overline{IRQ}$	Interrupt Request Input
$\overline{NMI}$	Non-Maskable Interrupt Input
DBE	Data Bus Enable Input
TSC	3-State Control Input
$\overline{RESET}$	Reset Input
VMA	Valid Memory Address Output
BA	Bus Available Output
A_0-A_{15}	Address Bus Outputs
R/$\overline{W}$	Read/Write Output
V_{CC}	+5 V Power Supply Input
V_{SS}	Ground

Logic Symbol



V_{CC} = Pin 8
V_{SS} = Pins 1, 21

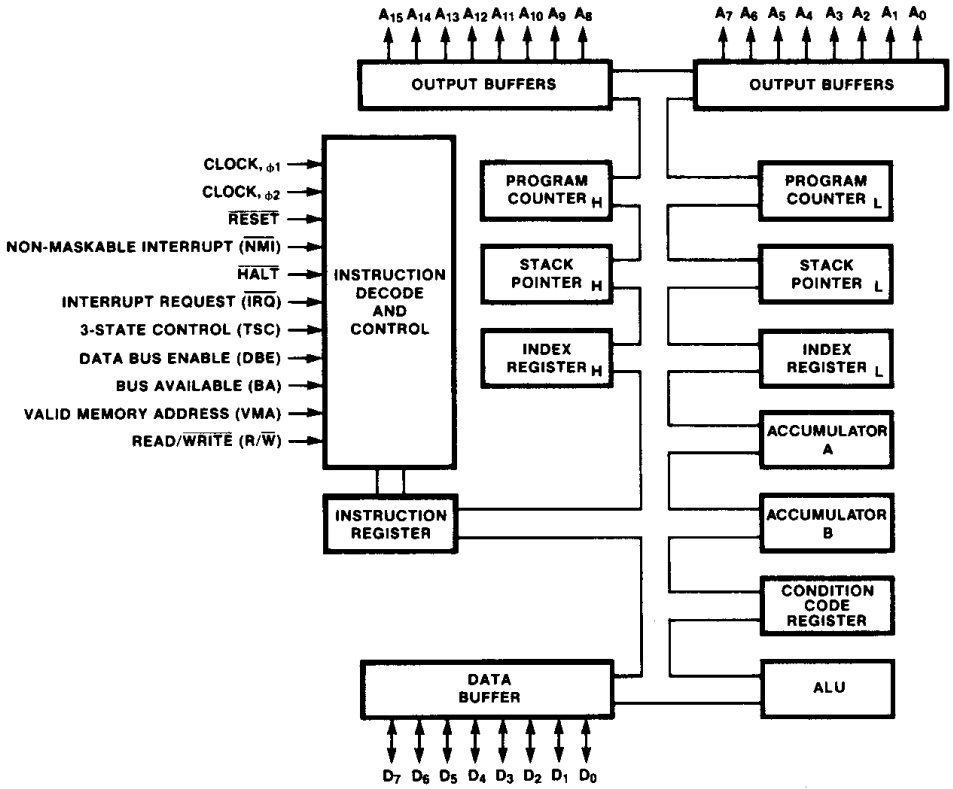
Connection Diagram
40-Pin DIP



(Top View)

www.DataSheet4U.com

Block Diagram



MPU Signal Description

Proper operation of the MPU requires that certain control and timing signals be provided to accomplish specific functions and that other signal lines be monitored to determine the state of the processor.

Clocks Phase One and Phase Two (ϕ1,ϕ2)
Two pins are used for a 2-phase non-overlapping clock that runs at the V_{CC} voltage level.

Figure 27 shows the microprocessor clocks, and the *Clock Timing* table shows the static and dynamic clock specifications. The HIGH level is specified at V_{IHC} and the LOW level is specified at V_{ILC}. The allowable clock frequency is specified by f (frequency). The minimum ϕ1 and ϕ2 HIGH level pulse widths are specified by $PW_{\phi H}$ (pulse width HIGH time). To guarantee the required access time for the peripherals, the clock up time, t_{ut}, is specified. Clock separation, t_d, is measured at a maximum voltage of V_{OV} (overlap voltage). This allows for a multitude of clock variations at the system frequency rate.

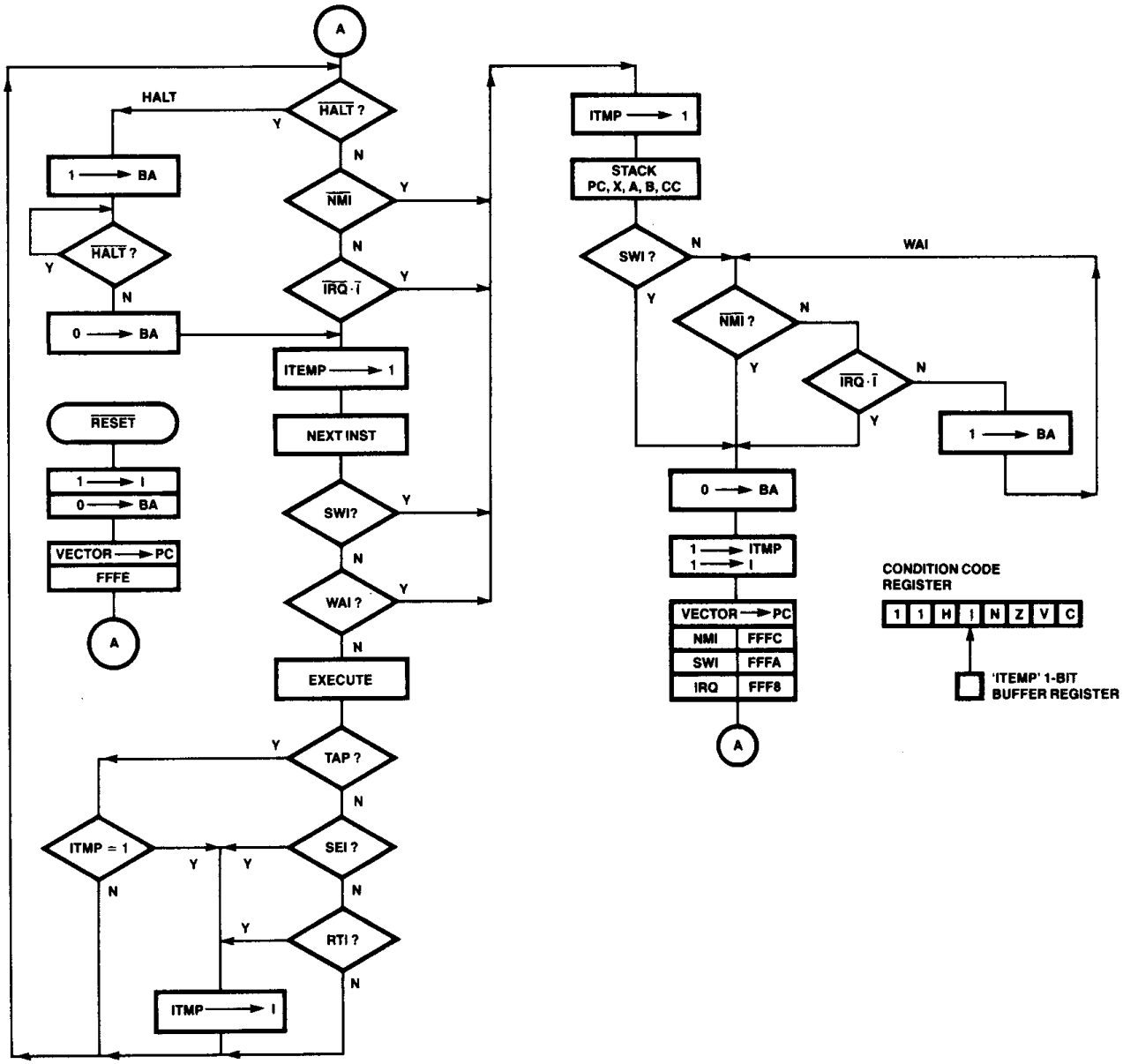
Address Bus (A_0–A_{15})
Sixteen pins are used for the address bus. The outputs are 3-state bus drivers capable of driving one standard TTL load and 90 pF. When the output is turned off, it is essentially an open circuit. This permits the MPU to be used in DMA applications. Putting TSC in its HIGH state forces the address bus to go into the 3-state mode.

Data Bus (D_0–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 3-state output buffers capable of driving one standard TTL load and 130 pF. The data bus is placed in the 3-state mode when DBE is LOW.

Data Bus Enable (DBE)
This input is the 3-state control signal for the MPU data bus and will enable the bus drivers when in the HIGH state. This input is TTL-compatible; however, in normal operation it would be driven by the phase two 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

www.DataSheet4U.com

Fig. 1 MPU Flow Chart



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, "ITMP". This has the effect of delaying any clearing of the I-Bit one clock time. Setting the I-Bit, however, is not delayed.
3. Refer to tables 8 through 13 for details of instruction execution.

www.DataSheet4U.com

in Direct Memory Access (DMA) applications, DBE should be held LOW.

If additional data set-up or hold time is required on an MPU write, the DBE down time can be decreased as shown in *Figure 29* (DBE ≠ ϕ2). The minimum down time for DBE is $t_{\overline{DBE}}$ as shown and must occur within ϕ1 up time. The minimum delay from the trailing edge of DBE to the trailing edge of ϕ1 is $t_{\overline{DBED}}$. By skewing DBE with respect to E in this manner, data set-up or hold time can be increased.

Bus Available (BA)
The Bus Available 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 3-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 30 pF. 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). 3-State Control (TSC) going HIGH will turn Read/$\overline{Write}$ 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 90 pF.

Reset ($\overline{RESET}$)
The $\overline{RESET}$ 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 reinitialize 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 $\overline{RESET}$ is LOW (assuming a minimum of eight clock cycles have occurred) 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. *Figure 2* illustrates a power-up sequence using the $\overline{RESET}$ control line. After the power supply reaches 4.75 V 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. $\overline{RESET}$ can go HIGH asynchronously with the system clock any time after the eighth cycle.

Reset timing is shown in *Figure 2* and the *Read/Write Timing* table. The maximum rise and fall transition times are specified by t_{PCr} and t_{PCf}. If $\overline{RESET}$ is HIGH at t_{PCS} (processor control set-up time) as shown in *Figure 2* in any given cycle, then the restart sequence will begin on the next cycle as shown. The $\overline{RESET}$ control line may also be used to reinitialize the MPU system at any time during its operation. This is accomplished by pulsing $\overline{RESET}$ LOW for the duration of a minimum of three complete ϕ2 cycles. The Reset pulse can be completely asynchronous with the MPU system clock and will be recognized during ϕ2 if set-up 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. At that time, if the interrupt mask bit in the condition code register is not set, the machine will begin an interrupt sequence. The index register, program counter, 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 HIGH so that 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 *Figure 3*.

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 3 kΩ 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 F6800 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 *Figure 3* which details the MPU response to an interrupt while the MPU is executing the

www.DataSheet4U.com

Fig. 2 Reset Timing

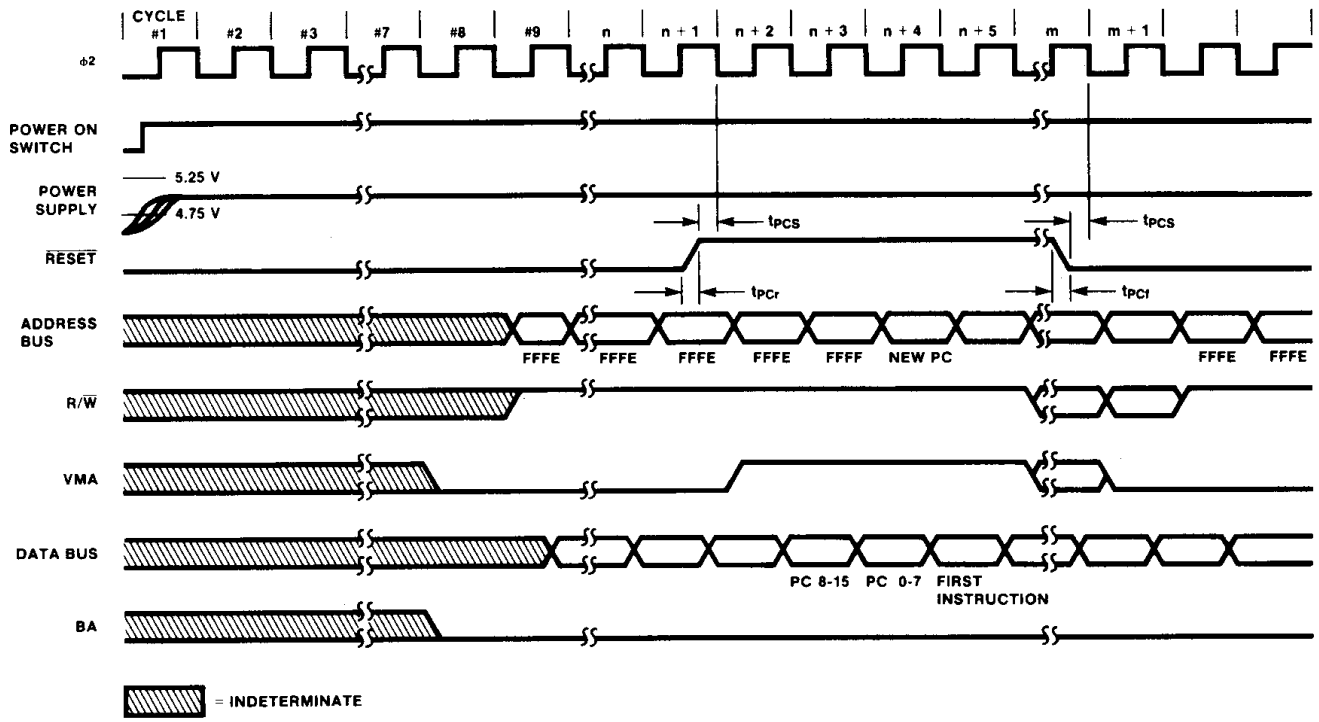
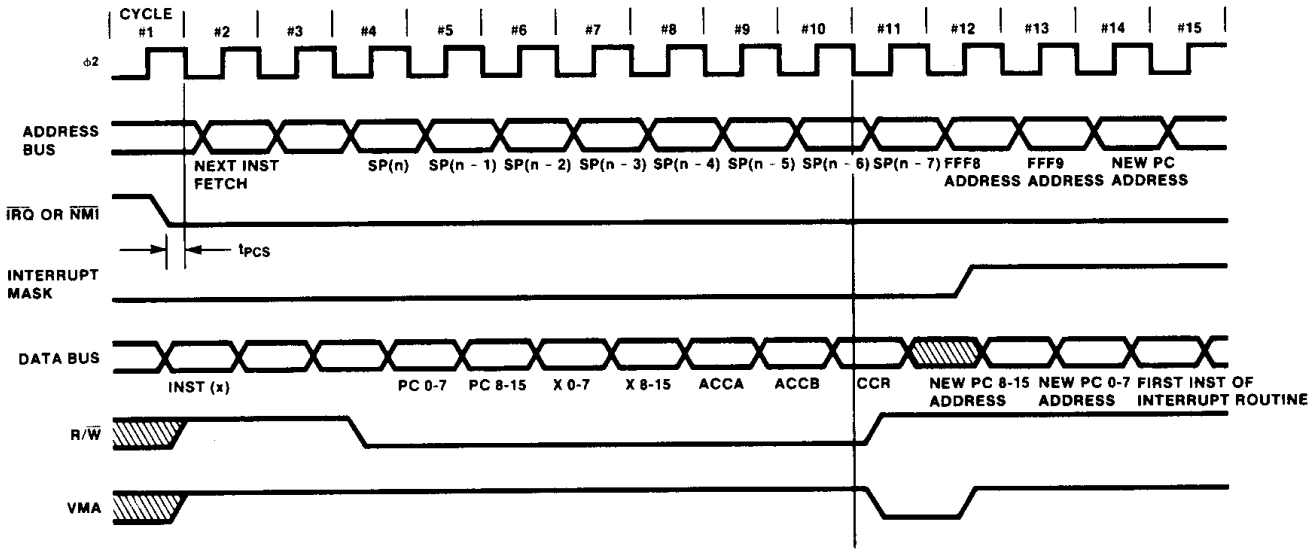


Fig. 3 Interrupt Timing



www.DataSheet4U.com

control program. The interrupt shown could be either $\overline{IRQ}$ or $\overline{NMI}$ 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 $\overline{NMI}$ 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.

Figure 4 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.

Table 1 Memory Map for Interrupt Vectors

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

Refer to *Figure 4* for program flow for Interrupts.

3-State Control (TSC)
When the 3-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 $PW_{\phi H}$ without destroying data within the MPU. TSC then can be used in a short Direct Memory Access (DMA) application.

Figure 5 shows the effect of TSC on the MPU. TSC must have its transitions at t_{TSE} (3-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} (3-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 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 3-state. One standard TTL load and 90 pF may be directly driven by this active HIGH signal.

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

The $\overline{HALT}$ line provides an input to the MPU to allow control of 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.

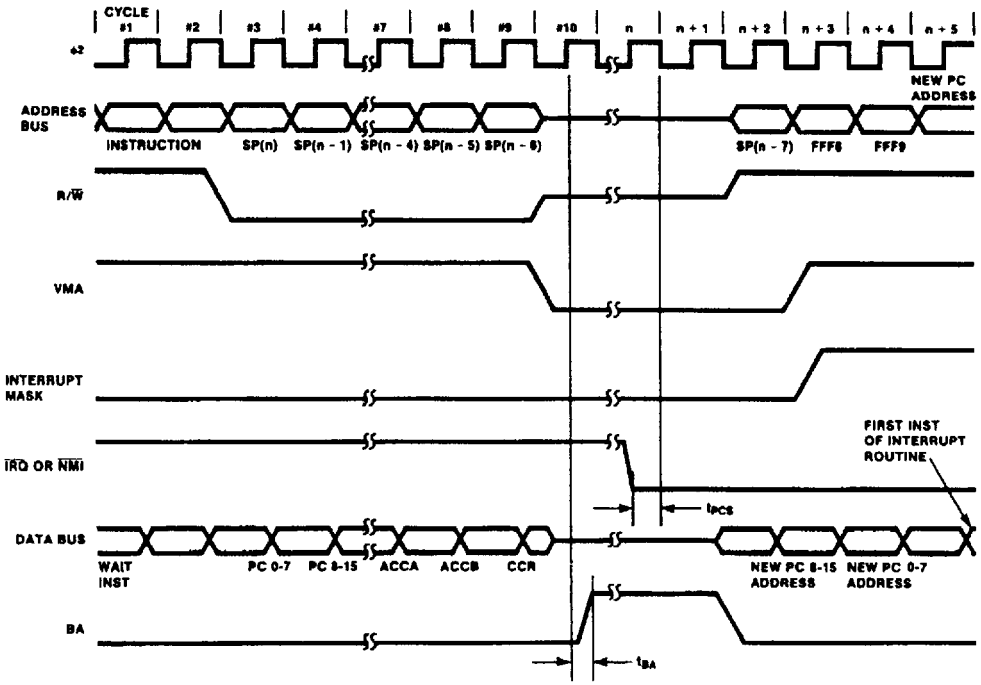
When BA is HIGH, the address bus, data bus, and R/$\overline{W}$ line will be in a high impedance state, effectively removing the MPU from the system bus. VMA is forced LOW so that the floating system bus will not activate any device on the bus that is enabled by VMA.

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 $\overline{RESET}$ 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 $\overline{RESET}$ is LOW. As soon as the $\overline{HALT}$ line goes HIGH, the MPU will go to locations FFFE and FFFF for the address of the reset routine.

Figure 6 shows the timing relationships involved when halting the MPU. The instruction illustrated is a 1-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 $\phi 1$ of the last cycle of an instruction (Point A of

www.DataSheet4U.com

Fig. 4 Wait Instruction Timing



5

Note
Midrange waveform indicates high-impedance state.

www.DataSheet4U.com

C 9004697 0293033 006

Fig. 5 3-State Control Timing

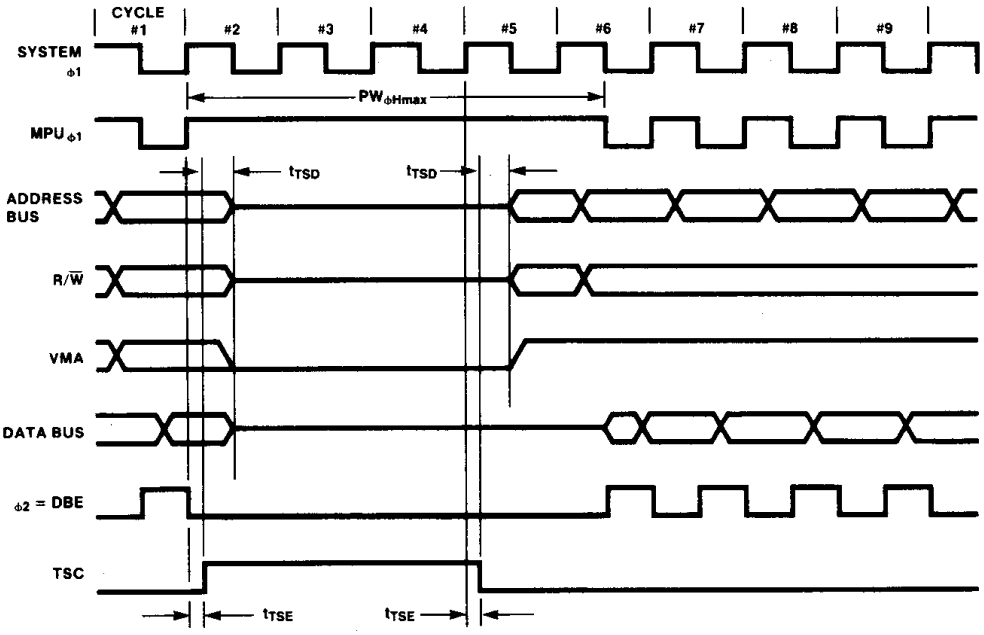
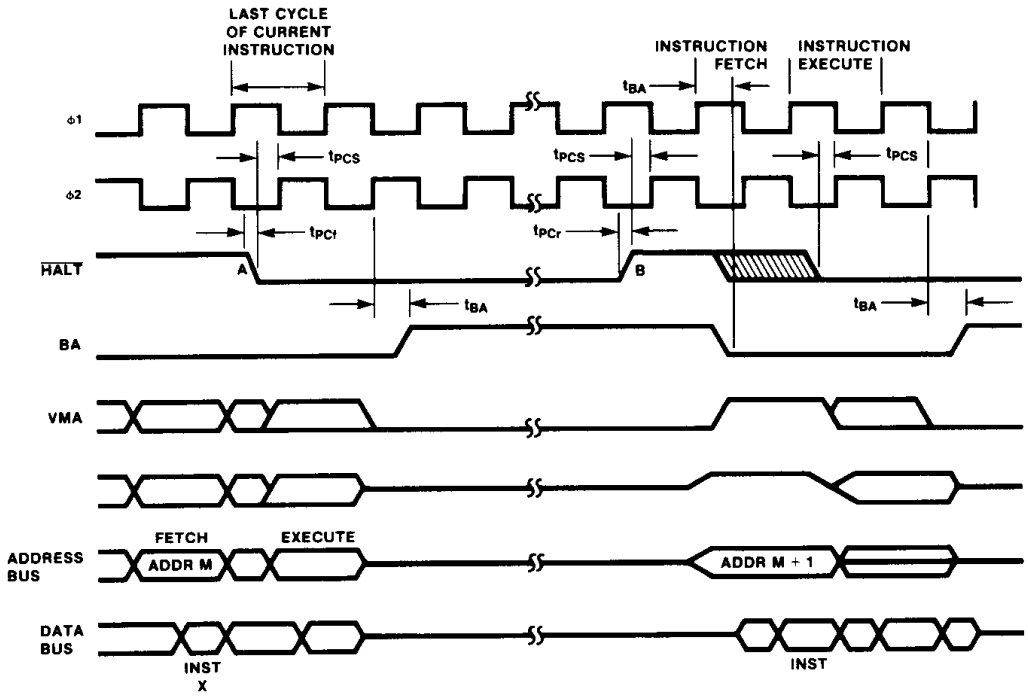


Fig. 6 Halt and Single Instruction Execution for System Debug



Note
Midrange waveform indicates high impedance state.

www.DataSheet4U.com

Figure 6). $\overline{HALT}$ 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 $\overline{HALT}$ 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 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, $\overline{HALT}$ must be brought HIGH for one MPU cycle and then returned LOW as shown at Point B of *Figure 6*. Again, the transitions of $\overline{HALT}$ must occur t_{PCS} before the trailing edge of the next $\phi 1$, 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.

MPU Registers

The MPU has three 16-bit registers and three 8-bit registers available for use by the programmer (*Figure 7*).

Program Counter
The program counter is a 2-byte (16 bits) register that points to the current program address.

Stack Pointer
The stack pointer is a 2-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 2-byte register that is used to store data or a 16-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 (bit 6 and bit 7) are ones.

MPU Instruction Set

The F6800 instructions are described in detail in the F6800 Programming Manual. This section will provide a brief introduction and discuss their use in developing F6800 control programs. The F6800 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.

5

Each of the 72 executable instructions of the source language assembles into one to three 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 2*. 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 F6800 performs the same operation on both its internal accumulators and the external memory locations. In addition, the F6800 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 F6800's instruction set: (1) accumulator and memory operations; (2) program control operations; (3) condition code register operations.

www.DataSheet4U.com

Fig. 7 Programming Model of The Microprocessing Unit

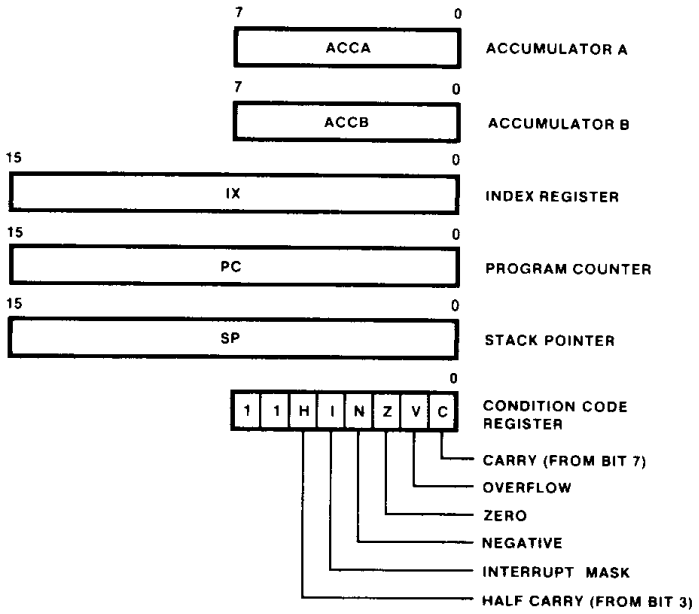


Table 2 Microprocessor Instruction Set—Alphabetic Sequence

ABA	Add Accumulators
ADC	Add with Carry
ADD	Add
AND	Logical And
ASL	Arithmetic Shift Left
ASR	Arithmetic Shift Right
BCC	Branch if Carry Clear
BCS	Branch if Carry Set
BEQ	Branch if Equal to Zero
BGE	Branch if Greater or Equal Zero
BGT	Branch if Greater than Zero
BHI	Branch if Higher
BIT	Bit Test
BLE	Branch if Less or Equal
BLS	Branch if Lower or Same
BLT	Branch if Less than Zero
BMI	Branch if Minus
BNE	Branch if Not Equal to Zero
BPL	Branch if Plus
BRA	Branch Always
BSR	Branch to Subroutine
BVC	Branch if Overflow Clear
BVS	Branch if Overflow Set
CBA	Compare Accumulators
CLC	Clear Carry
CLI	Clear Interrupt Mask
CLR	Clear
CLV	Clear Overflow
CMP	Compare
COM	Complement
CPX	Compare Index Register
DAA	Decimal Adjust
DEC	Decrement
DES	Decrement Stack Pointer
DEX	Decrement Index Register
EOR	Exclusive OR
INC	Increment
INS	Increment Stack Pointer
INX	Increment Index Register
JMP	Jump
JSR	Jump to Subroutine
LDA	Load Accumulator
LDS	Load Stack Pointer
LDX	Load Index Register
LSR	Logical Shift Right
NEG	Negate
NOP	No Operation
ORA	Inclusive OR Accumulator
PSH	Push Data
PUL	Pull Data
ROL	Rotate Left
ROR	Rotate Right
RTI	Return from Interrupt
RTS	Return from Subroutine
SBA	Subtract Accumulators
SBC	Subtract with Carry
SEC	Set Carry
SEI	Set Interrupt Mask
SEV	Set Overflow
STA	Store Accumulator
STS	Store Stack Register
STX	Store Index Register
SUB	Subtract
SWI	Software Interrupt
TAB	Transfer Accumulators
TAP	Transfer Accumulators to Condition Code Reg.
TBA	Transfer Accumulators
TPA	Transfer Condition Code Reg. to Accumulator
TST	Test
TSX	Transfer Stack Pointer to Index Register
TXS	Transfer Index Register to Stack Pointer
WAI	Wait for Interrupt

www.DataSheet4U.com

Table 3 Hexadecimal Values of Machine Codes

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

Notes

1. Addressing Modes: A = Accumulator A, B = Accumulator B, IMM = Immediate, DIR = Direct, REL = Relative, IND = Indexed
2. Unassigned code indicated by an asterisk (*)

www.DataSheet4U.com

Table 4 Accumulator and Memory Operations

The accumulator and memory operations and their effect on the CCR are shown in *Table 4*. Included are Arithmetic Logic, Data Test and Data Handling instructions.

Operations	Mnemonic	Addressing Modes												Boolean/Arithmetic Operation (All register labels refer to contents	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
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, 1s	COM							63	7	2	73	6	3				M → M	•	•	↕	↕	R	S
	COMA													43	2	1	A → A	•	•	↕	↕	R	S
	COMB													53	2	1	B → B	•	•	↕	↕	R	S
Complement, 2s	NEG							60	7	2	70	6	3				00 - M → M	•	•	↕	↕	1	2
(Negate)	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 → M_{SP}, SP - 1 → SP	•	•	•	•	•	•
	PSHB													37	4	1	B → M_{SP}, SP - 1 → SP	•	•	•	•	•	•
Pull Data	PULA													32	4	1	SP + 1 → SP, M_{SP} → A	•	•	•	•	•	•
	PULB													33	4	1	SP + 1 → SP, M_{SP} → B	•	•	•	•	•	•

www.DataSheet4U.com

Table 4 Accumulator and Memory Operations (Cont.)

Operations	Mnemonic	Addressing Modes: Immed OP	Immed ~	Immed #	Direct OP	Direct ~	Direct #	Index OP	Index ~	Index #	Extnd OP	Extnd ~	Extnd #	Implied OP	Implied ~	Implied #	Boolean/Arithmetic Operation (All register labels refer to contents)	Cond. Code Reg. * 5 H	4 I	3 N	2 Z	1 V	0 C
Rotate Left	ROL							69	7	2	79	6	3				M	•	•	↕	↕	6	↕
	ROLA													49	2	1	A	•	•	↕	↕	6	↕
	ROLB													59	2	1	B	•	•	↕	↕	6	↕
Rotate Right	ROR							66	7	2	76	6	3				M	•	•	↕	↕	6	↕
	RORA													46	2	1	A	•	•	↕	↕	6	↕
	RORB													56	2	1	B	•	•	↕	↕	6	↕
Shift Left,	ASL							68	7	2	78	6	3				M	•	•	↕	↕	6	↕
Arithmetic	ASLA													48	2	1	A	•	•	↕	↕	6	↕
	ASLB													58	2	1	B	•	•	↕	↕	6	↕
Shift Right,	ASR							67	7	2	77	6	3				M	•	•	↕	↕	6	↕
Arithmetic	ASRA													47	2	1	A	•	•	↕	↕	6	↕
	ASRB													57	2	1	B	•	•	↕	↕	6	↕
Shift Right,	LSR							64	7	2	74	6	3				M	•	•	R	↕	6	↕
Logic	LSRA													44	2	1	A	•	•	R	↕	6	↕
	LSRB													54	2	1	B	•	•	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	60	2	2	90	3	2	A0	5	2	80	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	TST							6D	7	2	7D	6	3				M - 00	•	•	↕	↕	R	R
or Minus	TSTA													4D	2	1	A - 00	•	•	↕	↕	R	R
	TSTB													5D	2	1	B - 00	•	•	↕	↕	R	R
																		H	I	N	Z	V	C

Note
Accumulator addressing mode instructions are included in the column for IMPLIED addressing
*See condition code register notes page 26

Legend:
OP Operation Code (Hexadecimal);
~ Number of MPU Cycles;
# Number of Program Bytes;
+ Arithmetic Plus;
− Arithmetic Minus;
• Boolean AND;
M_{SP} Contents of memory location pointed to be Stack Pointer;
+ Boolean Inclusive OR;
⊕ Boolean Exclusive OR;
M 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

www.DataSheet4U.com

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 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 F6800 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 F6800 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.

Table 5 Index Register and Stack Pointer Instructions

Pointer Operations	Mnemonic	Immed			Direct			Index			Extend			Implied			Boolean/Arithmetic Operation	Cond. Code Reg.*					
		OP	~	#	OP	~	#	OP	~	#	OP	~	#	OP	~	#		5	4	3	2	1	0
																		H	I	N	Z	V	C
Compare Index Reg	CPX	8C	3	3	9C	4	2	AC	6	2	BC	5	3				$X_H - M, X_L - (M + 1)$	•	•	⑦	↕	⑧	•
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				$M \rightarrow X_H, (M + 1) \rightarrow X_L$	•	•	⑨	↕	R	•
Load Stack Pntr	LDS	8E	3	3	9E	4	2	AE	6	2	BE	5	3				$M \rightarrow SP_H, (M + 1) \rightarrow SP_L$	•	•	⑨	↕	R	•
Store Index Reg	STX				DF	5	2	EF	7	2	FF	6	3				$X_H \rightarrow M, X_L \rightarrow (M + 1)$	•	•	⑨	↕	R	•
Store Stack Pntr	STS				9F	5	2	AF	7	2	BF	6	3				$SP_H \rightarrow M, SP_L \rightarrow (M + 1)$	•	•	⑨	↕	R	•
Indx Reg → Stack Pntr	TXS													35	4	1	$X - 1 \rightarrow SP$	•	•	•	•	•	•
Stack Pntr → Indx Reg	TSX													30	4	1	$SP + 1 \rightarrow X$	•	•	•	•	•	•

*See condition code register notes page 26

www.DataSheet4U.com

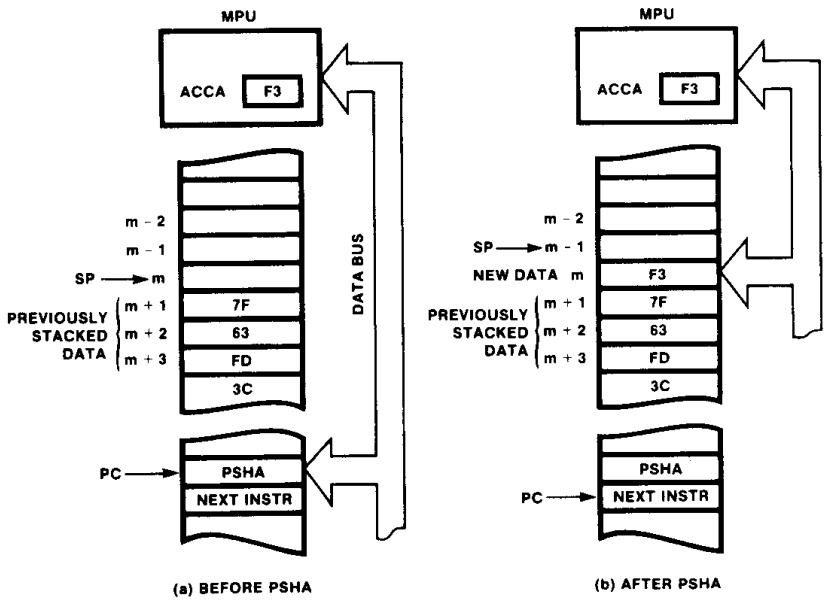
Operation of the stack pointer with the push and pull instructions is illustrated in *Figures 8* and *9*. The push instruction (PSHA) causes the contents of the indicated accumulator (A in this example) to be stored in memory at the location indicated by the stack pointer. The stack pointer is automatically 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 11* through *13*. 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 14*.

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 *Figure 16*. MPU status is restored by the return from interrupt, RTI, as shown in *Figure 15*.

5

Fig. 8 Stack Operation, Push Instruction



www.DataSheet4U.com

Fig. 9 Stack Operation, Pull Instruction

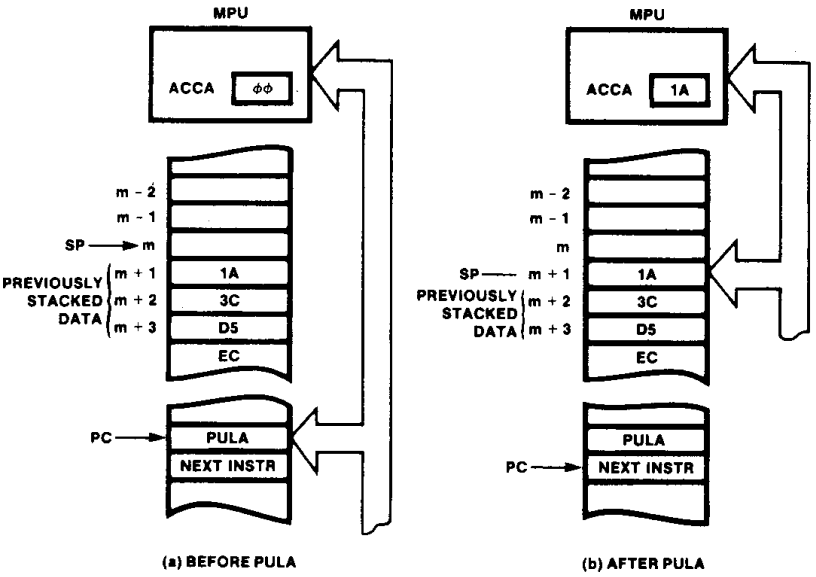


Fig. 10 Program Flow for Jump and Branch Instructions

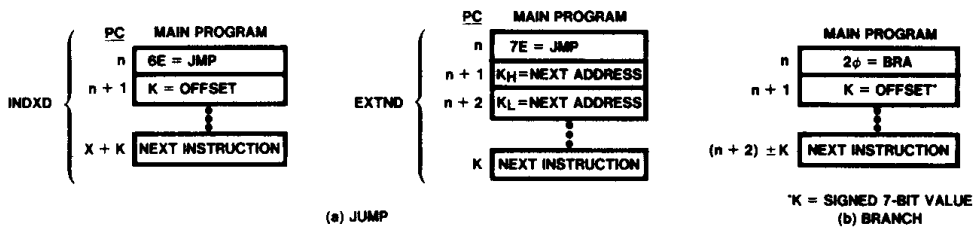
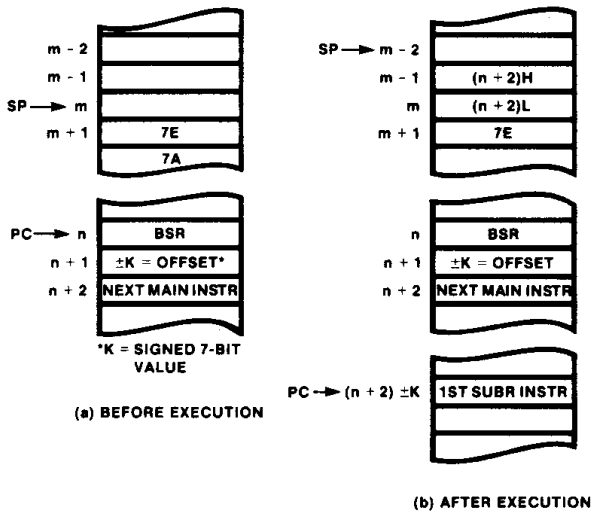


Fig. 11 Program Flow for BSR



www.DataSheet4U.com

Fig. 12 Program Flow for JSR (Extended)

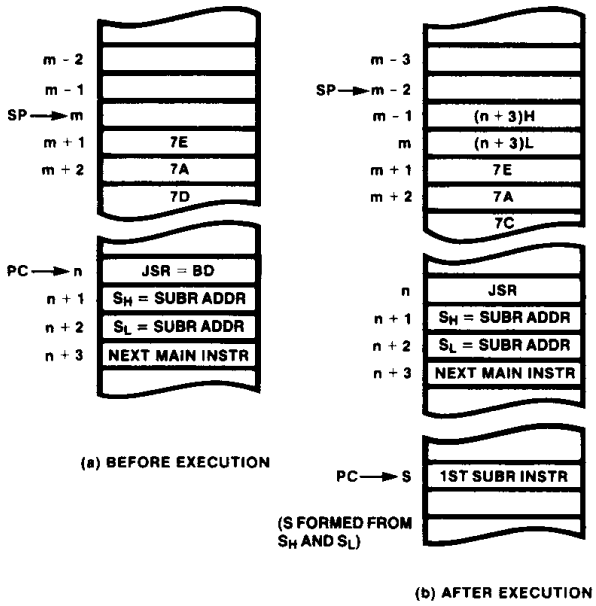
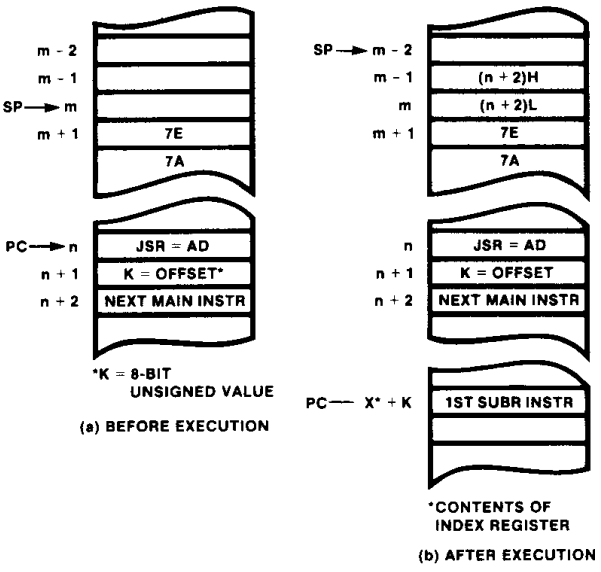


Fig. 13 Program Flow for JSR (Indexed)



www.DataSheet4U.com

Fig. 14 Program Flow for RTS

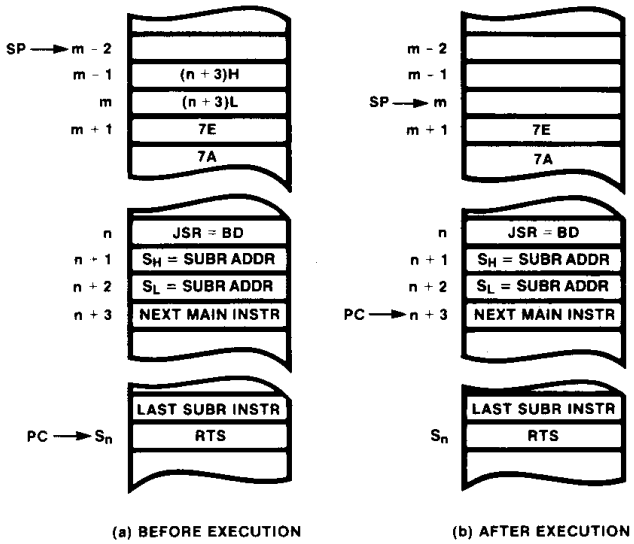
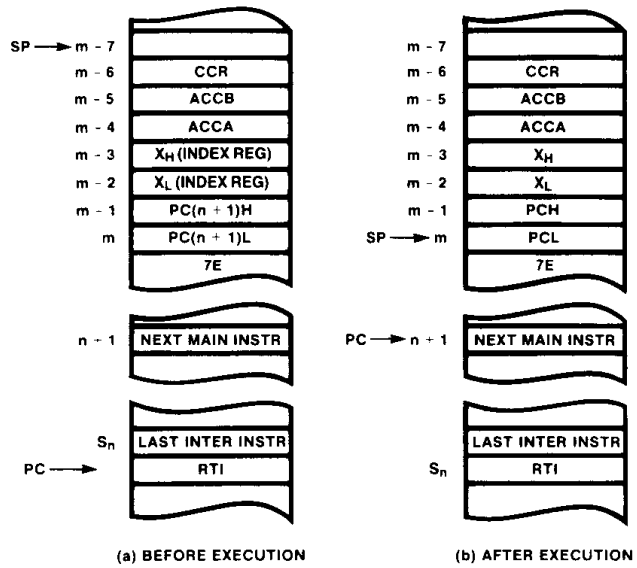
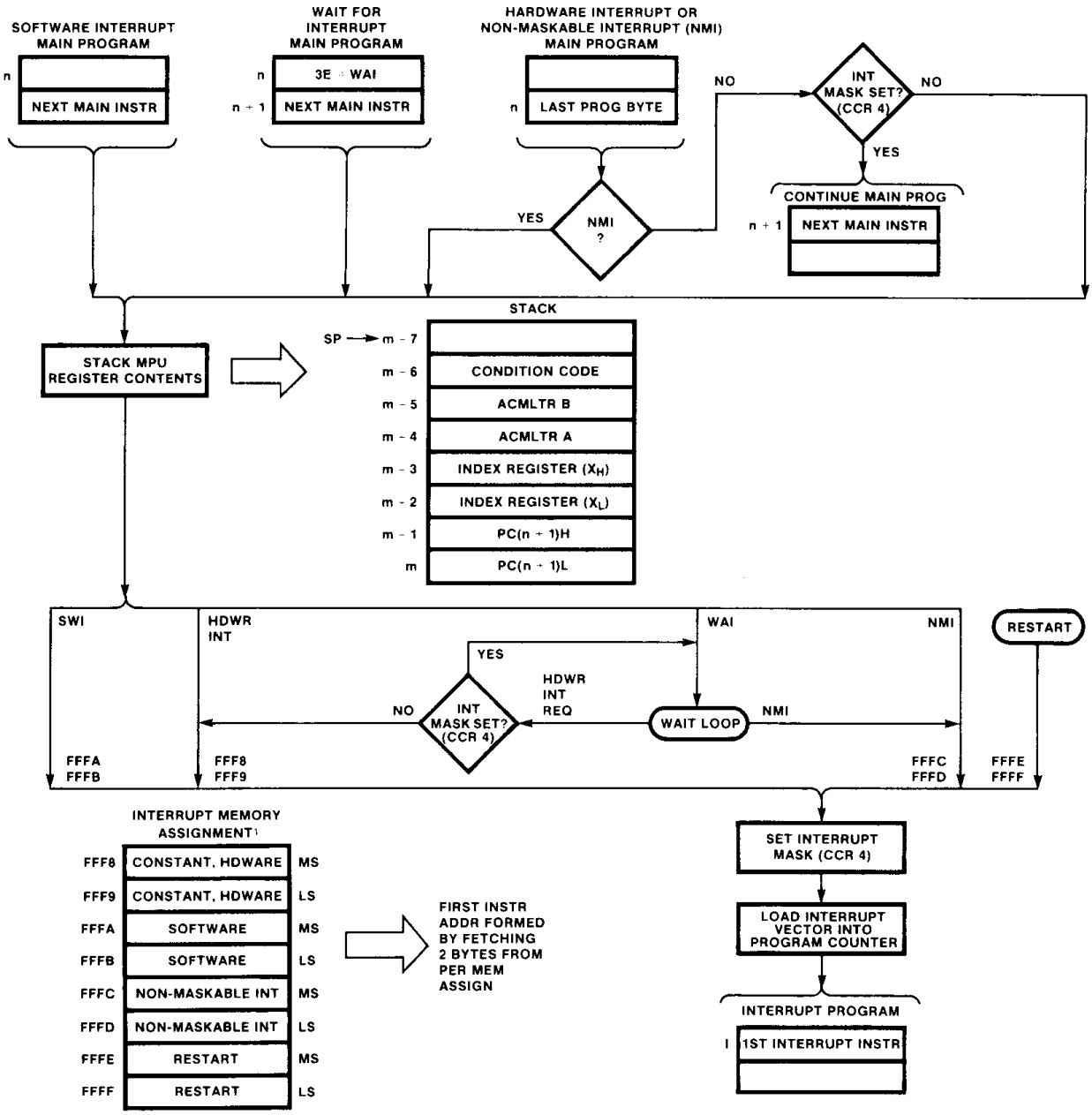


Fig. 15 Program Flow for RTI



www.DataSheet4U.com

Fig. 16 Program Flow for Interrupts



Note
MS = Most Significant Address Byte
LS = Least Significant Address Byte

www.DataSheet4U.com

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 *Figure 10*. When the MPU encounters the jump (indexed) 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.

Table 6 Jump and Branch Instructions

Operations	Mnemonic	Relative			Index			Extend			Implied			Branch Test	Cond. Code Reg. †					
															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				See Special Operations	•	•	•	•	•	•
Jump to Subroutine	JSR				AD	8	2	BD	9	3				See Special Operations	•	•	•	•	•	•
No Operation	NOP										01	2	1	Advances Prog. Cntr. Only	•	•	•	•	•	•
Return from Interrupt	RTI										3B	10	1		———		⑩	———		
Return from Subroutine	RTS										39	5	1	See Special Operations	•	•	•	•	•	•
Software Interrupt	SWI										3F	12	1	See Special Operations	•	•	•	•	•	•
Wait for Interrupt*	WAI										3E	9	1	See Special Operations	•	⑪	•	•	•	•

*WAI puts address bus, R/$\overline{W}$, and data bus in the 3-state mode while VMA is held LOW.
†See condition code register notes page 26.

www.DataSheet4U.com

The effect on program flow for the jump to subroutine (JSR) and branch to subroutine (BSR) is shown in *Figures 11* through *13*. 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 *Figure 14*.

The effect of executing the software interrupt, SWI, and the wait for interrupt, WAI, and their relationship to the hardware interrupts is shown in *Figure 15*. 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, (*Figure 15*) 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.

Fig. 17 Conditional Branch Instructions

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

The conditional branch instructions, *Figure 17*, consist 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

www.DataSheet4U.com

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 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 18*.

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 were indicated as they were introduced.

Table 7 Condition Code Register Instructions

Operations	Mnemonic	Implied			Boolean Operation	Cond. Code Reg.*					
						5	4	3	2	1	0
		OP	~	#		H	I	N	Z	V	C
Clear Carry	CLC	OC	2	1	0 → C	●	●	●	●	●	R
Clear Interrupt Mask	CLI	OE	2	1	0 → I	●	R	●	●	●	●
Clear Overflow	CLV	OA	2	1	0 → V	●	●	●	●	R	●
Set Carry	SEC	OD	2	1	1 → C	●	●	●	●	●	S
Set Interrupt Mask	SEI	OF	2	1	1 → I	●	S	●	●	●	●
Set Overflow	SEV	OB	2	1	1 → V	●	●	●	●	S	●
Acmltr A → CCR	TAP	06	2	1	A → CCR	(12)					
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.
*See Condition Code Register notes below

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.

7 (Bit N) Test: Sign bit of most significant (MS) byte = 1?

8 (Bit V) Test: 2s complement overflow from subtraction of MS bytes?

9 (Bit N) Test: Result less than "0"? (Bit 15 = 1)

10 (All) Load condition code register from stack. (See Special Operations)

11 (Bit I) Set when interrupt occurs. If previously set, a non-maskable interrupt is required to exit the wait state.

12 (All) Set according to the contents of accumulator A.

www.DataSheet4U.com

Fig. 18 Condition Code Register Bit Definition

b_5	b_4	b_3	b_2	b_1	b_0
H	I	N	Z	V	C

H = Half-carry; set whenever a carry from b_3 to b_4 of the result is generated by ADD, ABA, ADC; cleared if no b_3 to b_4 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 stack is LOW.

N = Negative; set if high order bit (b_7) of result is set; cleared otherwise

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

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

C = Carry; set if there were a carry from the most significant bit (b_7) of the result; cleared otherwise.

A CLI-WAI instruction sequence operated properly with early F6800 processors only if the preceding instruction were 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 F6800 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.

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 F6800 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 assembler to determine addressing mode.

5

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 19*.

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:

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

www.DataSheet4U.com

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, incrementing the index register, causes the contents of the index register to be increased by one.

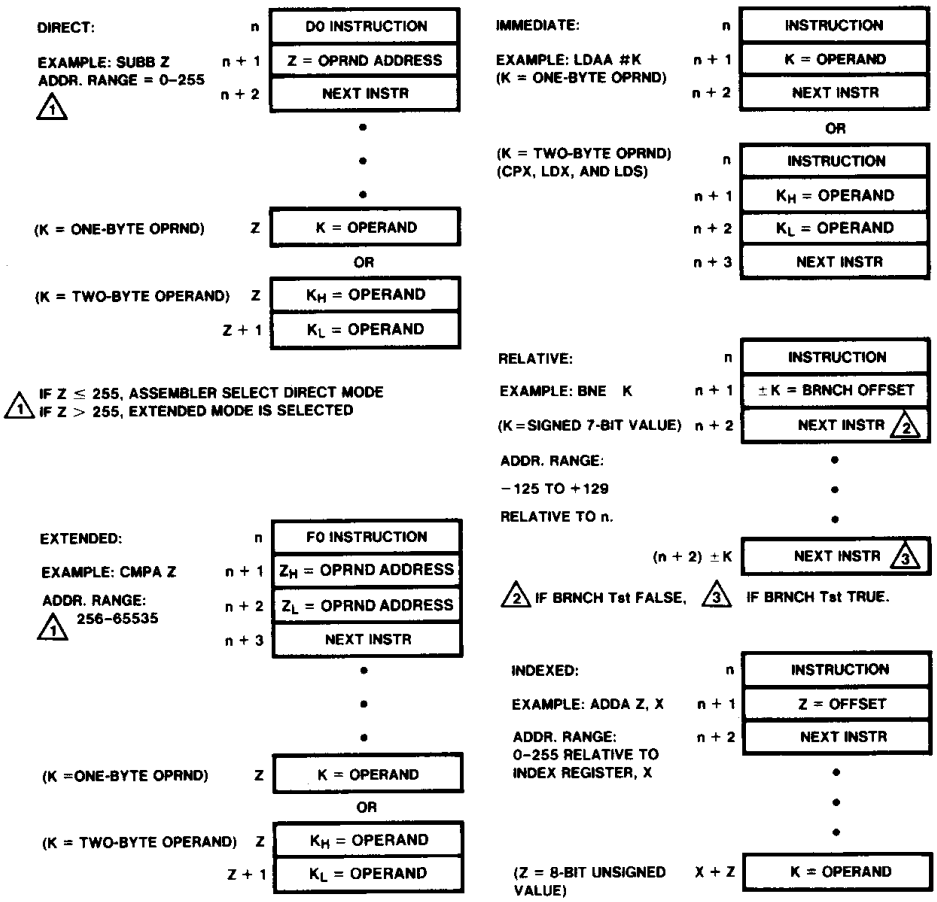
Program flow for instructions of this type is illustrated in *Figures 20* and *21*. In these figures, the general case is shown on the left and a specific example is shown on the right. Numerical examples are in decimal notation. Instructions of this type require only one byte of opcode. Cycle-by-cycle operation of the inherent mode is shown in *Table 8*.

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 23*.

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

Fig. 19 Addressing Mode Summary



www.DataSheet4U.com

Table 8 Inherent Mode Cycle-by-Cycle Operation

Address Mode and Instructions	Cycles	Cycle #	VMA Line	Address Bus	R/$\overline{W}$ Line	Data Bus
Inherent						
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)

www.DataSheet4U.com

Table 8 Inherent Mode Cycle-by-Cycle Operation (Cont.)

Address Mode and Instructions	Cycles	Cycle #	VMA Line	Address Bus	R/$\overline{W}$ Line	Data Bus
Inherent (Cont'd)						
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)

Notes

1. If device which is addressed during this cycle uses VMA, then the data bus will go to the high impedance 3-state condition. Depending on bus capacitance, data from the previous cycle may be retained on the data bus.
2. Data is ignored by the MPU.
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.

www.DataSheet4U.com

Fig. 20 Inherent Addressing

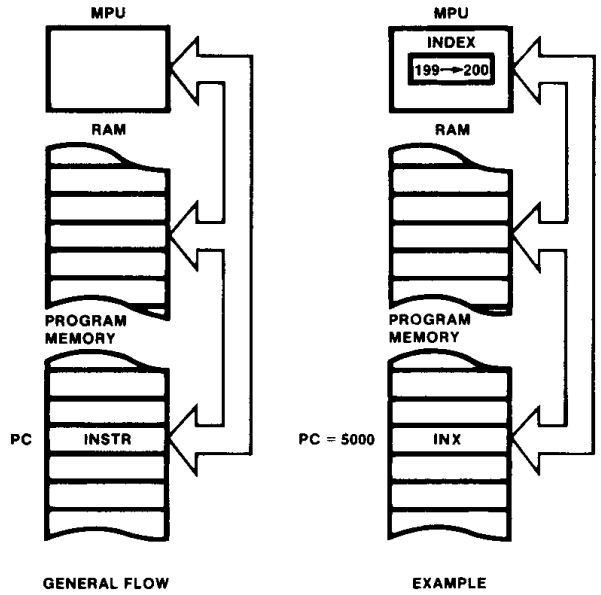


Fig. 21 Accumulator Addressing

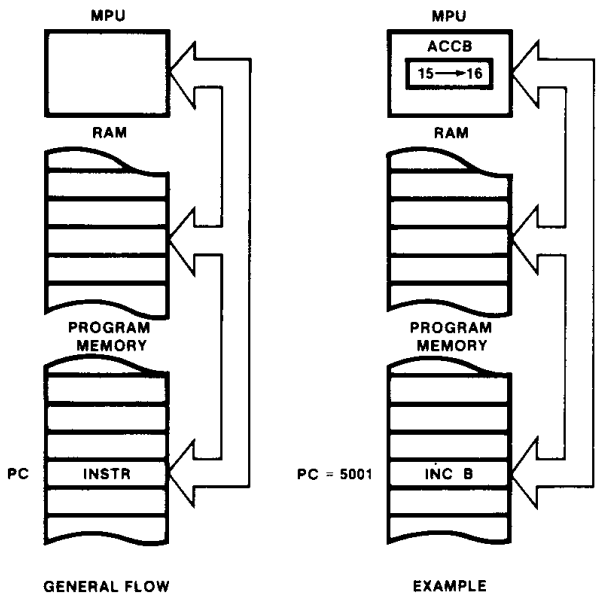


Fig. 22 Immediate Addressing Mode

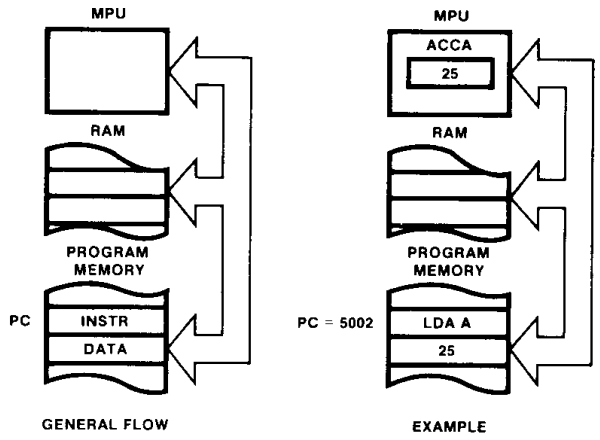
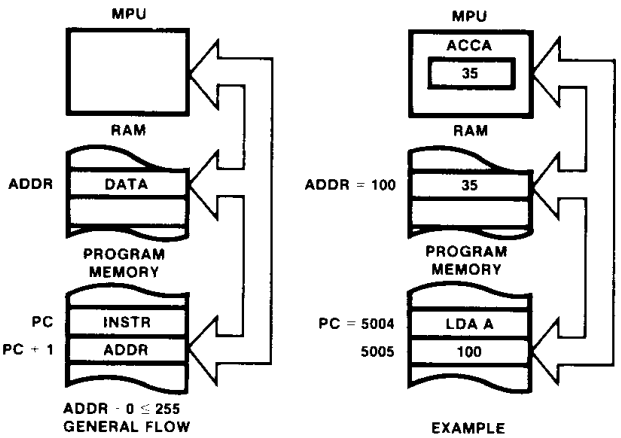


Fig. 23 Direct Addressing Mode



5

www.DataSheet4U.com

Table 9 Immediate Mode Cycle-by-Cycle Operation

Address Mode and Instructions	Cycles	Cycle #	VMA Line	Address Bus	R/$\overline{W}$ Line	Data Bus
Immediate						
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 10 Direct Mode Cycle-by-Cycle Operation

Address Mode and Instructions	Cycles	Cycle #	VMA Line	Address Bus	R/$\overline{W}$ Line	Data Bus
Direct						
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) 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) Register Data (High Order Byte) Register Data (Low Order Byte)

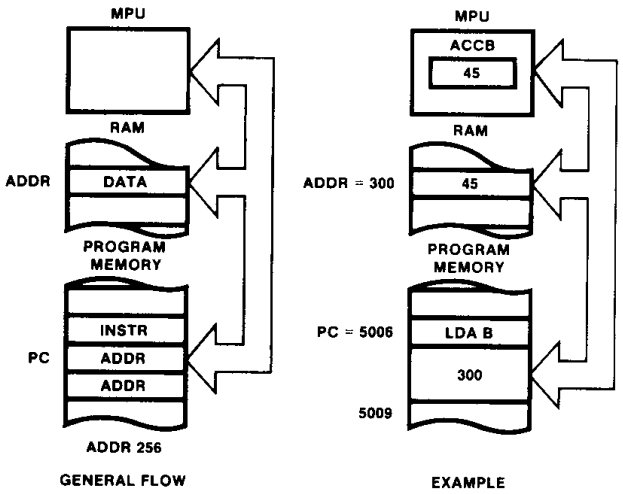
Note
If device which is addressed during this cycle uses VMA, then the data bus will go to the high impedance 3-state condition. Depending on bus capacitance, data from the previous cycle may be retained on the data bus.

www.DataSheet4U.com

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 2-byte operand such as LDX (load the index register), the operand bytes would be retrieved from locations 100 and 101. *Table 10* shows the cycle-by-cycle operations for the direct mode of addressing.

Extended addressing, *Figure 24*, 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 11* for extended addressing.

Fig. 24 Extended Addressing Mode



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 22*.

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 requires two-byte operands. In the immediate addressing mode, the "address" of the operand is effectively the memory location immediately following the instruction itself. *Table 9* shows the cycle-by-cycle operation for the immediate addressing mode.

Relative Addressing 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 25*.) 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 designation, 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 instructions itself. For transferrring control beyond this range, the unconditional jump (JMP), jump to subroutine (JSR), and return from subroutine (RTS) are used.

In *Figure 25*, 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 25*). If the previous result were 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.

www.DataSheet4U.com

Table 11 Extended Mode Cycle-by-Cycle Operation

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

Notes

1. If device which is addressed during this cycle uses VMA, then the data bus will go to the high impedance 3-state condition. Depending on bus capacitance, data from the previous cycle may be retained on the data bus.
2. For TST, VMA = "0" and operand data does not change.

www.DataSheet4U.com

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.

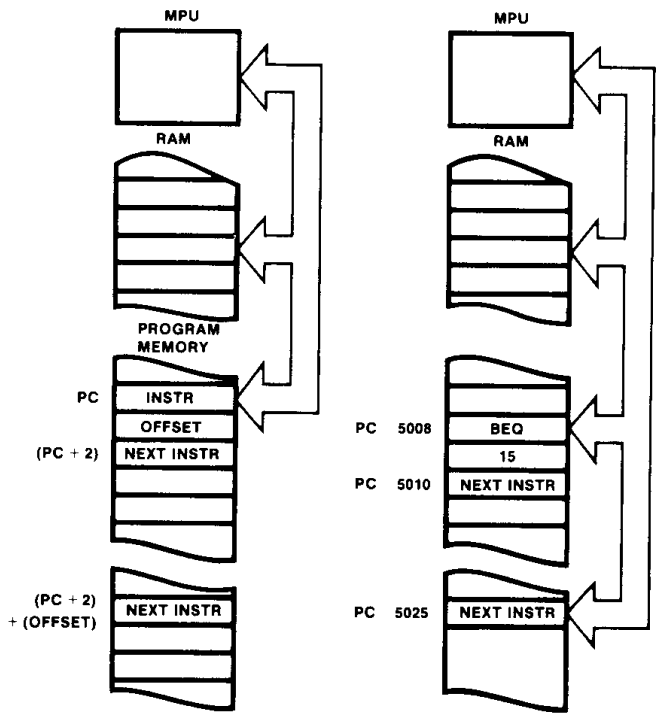
Indexed Addressing Mode

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 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.

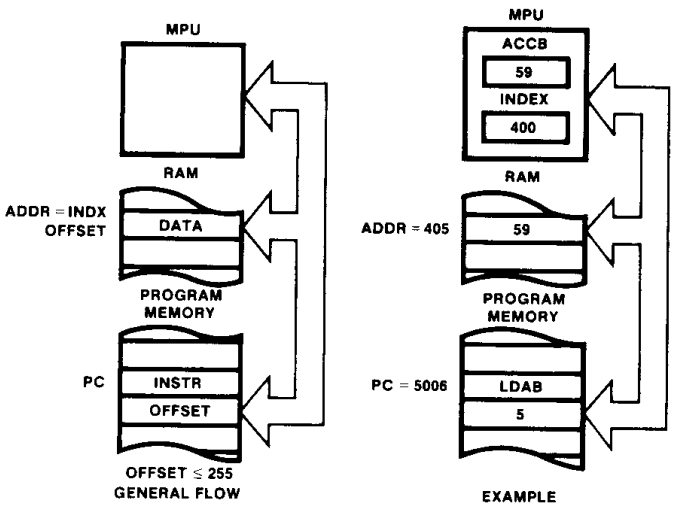
Fig. 25 Relative Addressing Mode



The operand field can also contain a numerical value that will be automatically added to X during execution. This format is illustrated in *Figure 26*.

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.

Fig. 26 Indexed Addressing Mode



www.DataSheet4U.com

Table 12 Relative Mode Cycle-by-Cycle Operation

Address Mode and Instructions	Cycles	Cycle #	VMA Line	Address Bus	R/$\overline{W}$ Line	Data Bus
Relative						
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)
		4	0	Branch Address	1	Irrelevant Data (Note)
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)
		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)
		7	0	Return Address of Main Program	1	Irrelevant Data (Note)
		8	0	Subroutine Address	1	Irrelevant Data (Note)

Note
If device which is addressed during this cycle uses VMA, then the data bus will go to the high impedance 3-state condition. Depending on bus capacitance, data from the previous cycle may be retained on the data bus.

Table 13 Indexed Mode Cycle-by-Cycle Operation

Address Mode and Instructions	Cycles	Cycle #	VMA Line	Address Bus	R/$\overline{W}$ 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

www.DataSheet4U.com

Table 13 Indexed Mode Cycle-by-Cycle Operation (Cont.)

Address Mode and Instructions	Cycles	Cycle #	VMA Line	Address Bus	R/$\overline{W}$ Line	Data Bus
Indexed (Cont.)						
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 3-state condition. Depending on bus capacitance, data from the previous cycle may be retained on the data bus.

2. For TST, VMA = "0" and operand data does not change.

www.DataSheet4U.com

Absolute Maximum Ratings

Supply Voltage	−0.3 V, +7.0 V
Input Voltage	−0.3 V, +7.0 V
Operating Temperature Range—T_L to T_H	
F6800, F68A00, F68B00	0°C, +70°C
F6800C, F68A00C, F68B00C	−40°C, +85°C
F6800DM, F68A00DM, F68B00DM	−55°C, +125°C
Storage Temperature Range	−55°C, +150°C
Thermal Resistance	
Plastic Package	70°C/W
Ceramic Package	50°C/W

This device contains circuitry to protect the inputs against damage due to high static voltages or electric fields; however, it is advised that normal precautions be taken to avoid application of any voltage higher than maximum rated voltages to this high-impedance circuit.

DC Characteristics V_{CC} = 5.0 V ± 5%, V_{SS} = 0, $T_A = T_L$ to T_H, unless otherwise noted

Symbol	Characteristic	Min	Typ	Max	Unit	Conditions
V_{IH} V_{IHC}	Input HIGH Voltage — Logic $\phi1, \phi2$	V_{SS} + 2.0 V_{CC} − 0.6		V_{CC} V_{CC} + 0.3	V	
V_{IL} V_{ILC}	Input LOW Voltage — Logic $\phi1, \phi2$	V_{SS} − 0.3 V_{SS} − 0.3		V_{SS} + 0.8 V_{SS} + 0.4	V	
I_{IN}	Input Leakage Current — Logic $\phi1, \phi2$		1.0	2.5 100	μA	V_{IN} = 0 to 5.25 V, V_{CC} = Max V_{IN} = 0 to 5.25 V, V_{CC} = 0.0 V
I_{TSI}	3-State (OFF State) Input Current — D_0-D_7 A_0-A_{15}, R/$\overline{W}$		2.0	10 100	μA	V_{IN} = 0.4 to 2.4 V, V_{CC} = Max
V_{OH}	Output HIGH Voltage — D_0-D_7 A_0-A_{15}, R/$\overline{W}$, VMA BA	V_{SS} + 2.4 V_{SS} + 2.4 V_{SS} + 2.4			V	I_{Load} = −205 μA, V_{CC} = Min I_{Load} = −145 μA, V_{CC} = Min I_{Load} = −100 μA, V_{CC} = Min
V_{OL}	Output LOW Voltage			V_{SS} + 0.4	V	I_{Load} = 1.6 mA, V_{CC} = Min
P_D	Power Dissipation		0.5	1.0	W	
C_{IN}	Input Capacitance — $\phi1$ $\phi2$ D_0-D_7 Logic Inputs		25 45 10 6.5	35 70 12.5 10	pF	V_{IN} = 0, T_A = 25°C, f = 1.0 MHz
C_{OUT}	Output Capacitance A_0-A_{15}, R/$\overline{W}$, VMA			12	pF	

www.DataSheet4U.com

Clock Timing V_{CC} = 5.0 V ± 5%, V_{SS} = 0, $T_A = T_L$ to T_H, unless otherwise noted

Symbol	Characteristic		Min	Typ	Max	Unit	Conditions
f	Frequency of Operation						
		F6800	0.1		1.0	MHz	
		F68A00	0.1		1.5		
		F68B00	0.1		2.0		
t_{cyc}	Cycle Time *(Figure 27)*	F6800	1.000		10	μs	
		F68A00	0.666		10		
		F68B00	0.500		10		
$PW_{\phi H}$	Clock Pulse Width						
	$\phi1, \phi2$ -	F6800	400		9500	ns	V_{CC} − 0.6 V
	$\phi1, \phi2$ -	F68A00	230		9500		
	$\phi1, \phi2$ -	F68B00	180		9500		
t_{ut}	Total $\phi1$ and $\phi2$ Up Time	F6800	900			ns	
		F68A00	600				
		F68B00	440				
$t_{\phi r}, t_{\phi f}$	Rise and Fall Times				100	ns	Measured Between V_{SS} + 0.4 V and V_{CC} − 0.6 V
t_d	Delay Time or Clock Separation *(Figure 27)*		0		9100	ns	$V_{OV} = V_{SS}$ + 0.6 V @ $t_r = t_f \leq$ 100 ns
			0		9100		$V_{OV} = V_{SS}$ + 1.0 V @ $t_r = t_f \leq$ 35 ns

Read/Write Timing (Reference *Figures 28* through *32*)

Symbol	Characteristic	F6800			F68A00			F68B00			Unit
		Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
t_{AD}	Address Delay										ns
	C = 90 pF			270			180			150	
	C = 30 pF			250			165			135	
t_{acc}	Peripheral Read Access Time $t_{acc} = t_{ut} - (t_{AD} + t_{DSR})$			530			360			250	ns
t_{DSR}	Data Set-up Time (Read)	100			60			40			ns
t_H	Input Data Hold Time	10			10			10			ns
t_H	Output Data Hold Time	10	25		10	25		10	25		ns
t_{AH}	Address Hold Time (Address, R/$\overline{W}$, VMA)	30	50		30	50		30	50		ns
t_{EH}	Enable HIGH Time for DBE Input	450			280			220			ns
t_{DDW}	Data Delay Time (Write)			225			200			160	ns
	Processor Controls										
t_{PCS}	Processor Control Set-up Time	200			140			110			ns
t_{PCr}, t_{PCf}	Processor Control Rise and Fall Time			100			100			100	ns
t_{BA}	Bus Available Delay			250			165			135	ns
t_{TSE}	3-State Enable			40			40			40	ns
t_{TSD}	3-State Delay			270			270			220	ns
$t_{\overline{DBE}}$	Data Bus Enable Down Time During $\phi1$ Up Time	150			120			75			ns
t_{DBEr}, t_{DBEf}	Data Bus Enable Rise and Fall Times			25			25			25	ns

www.DataSheet4U.com

Fig. 27 Clock Timing Waveform

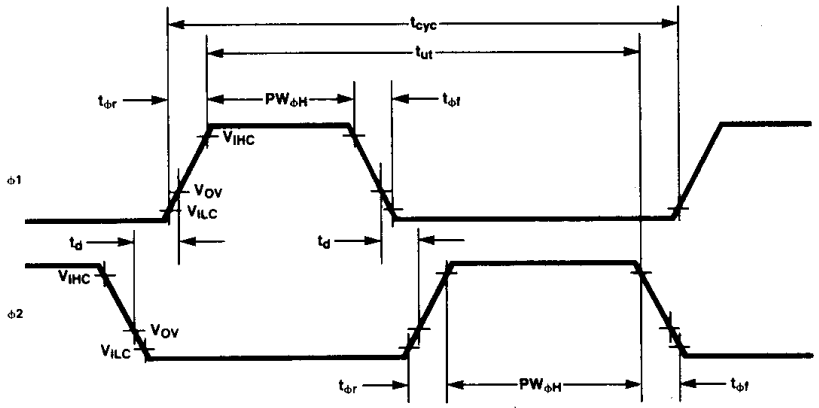
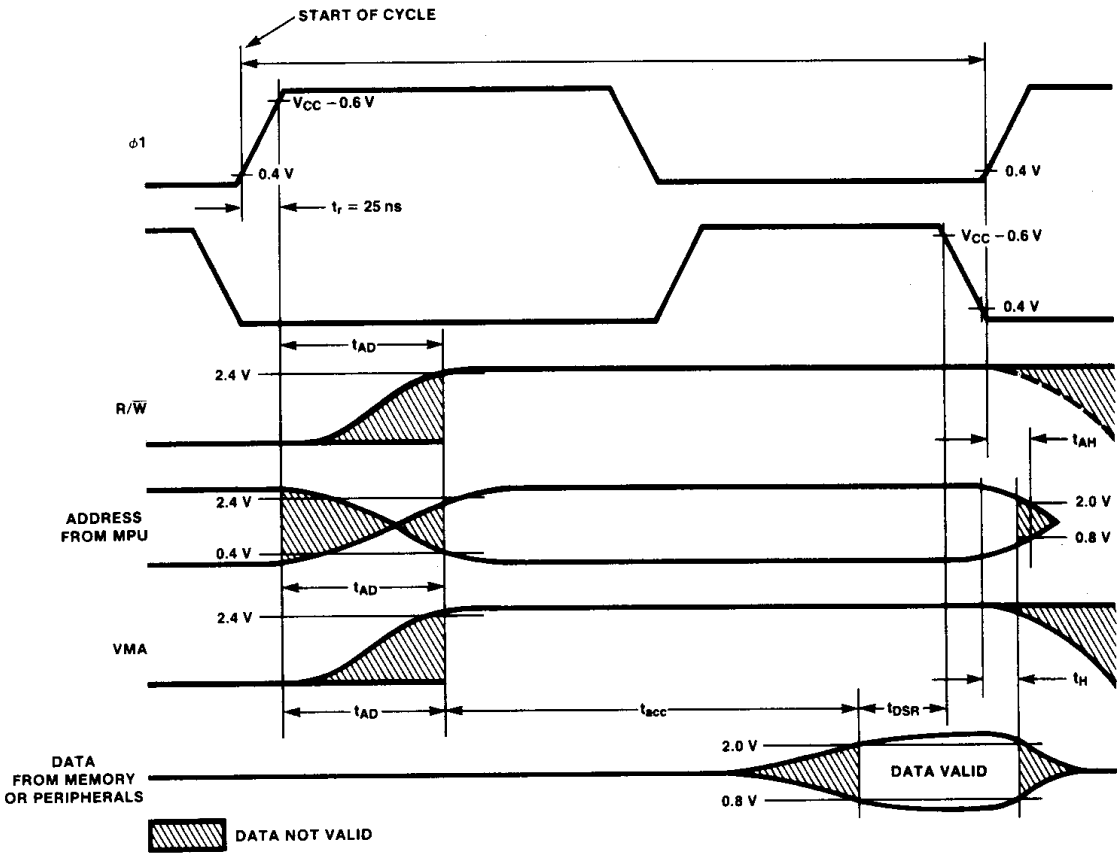
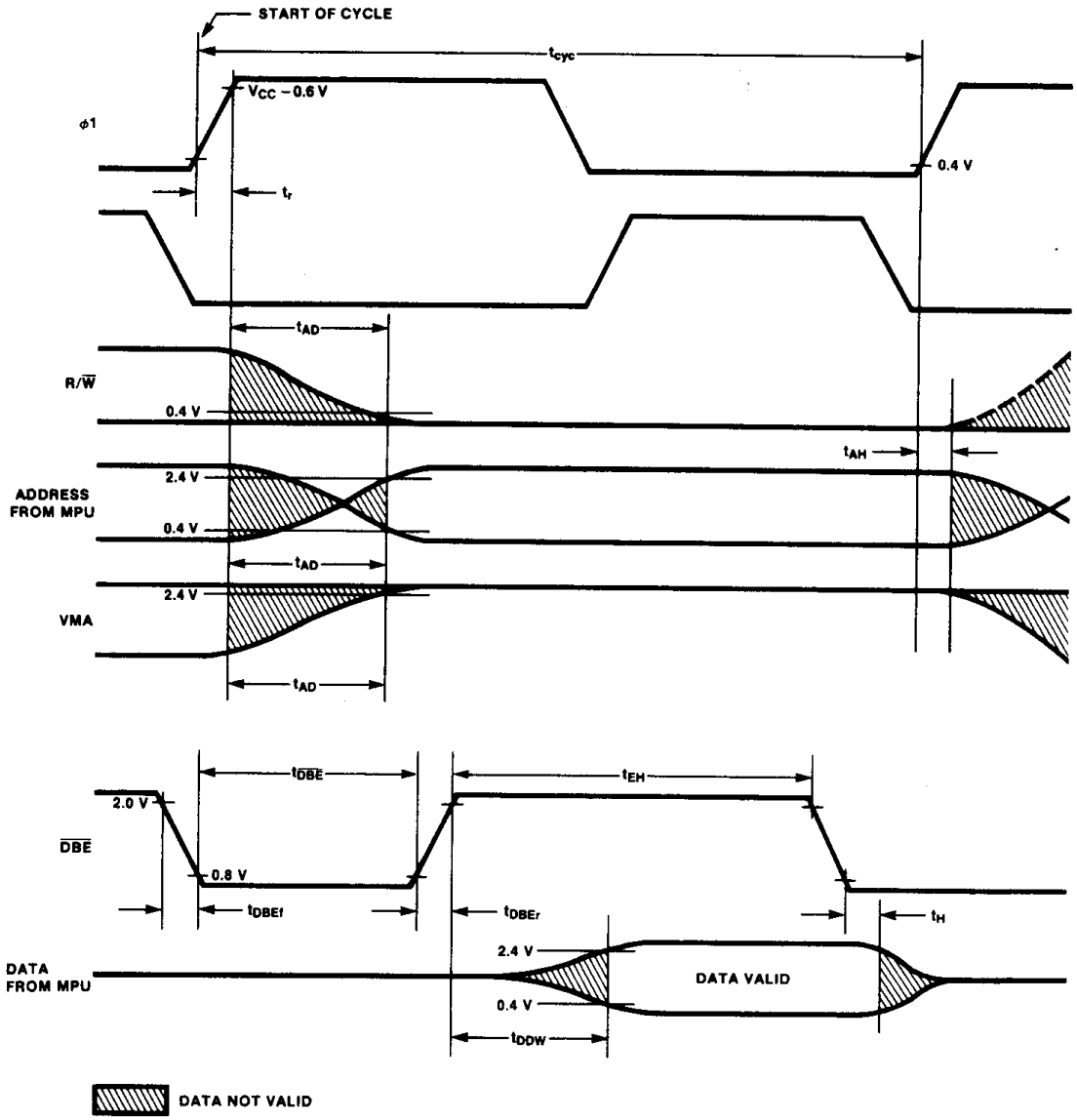


Fig. 28 Read Data From Memory or Peripherals



www.DataSheet4U.com

Fig. 29 Write In Memory or Peripherals



www.DataSheet4U.com

Fig. 30 Typical Data Bus Output Delay vs Capacitive Loading (t_{DDW})

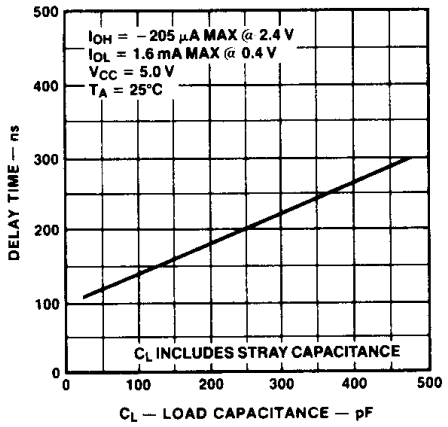


Fig. 31 Typical READ/WRITE, VMA, and Address Output Delay vs Capacitive Loading (t_{AD})

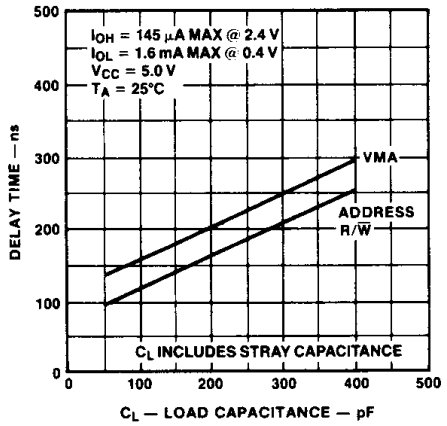
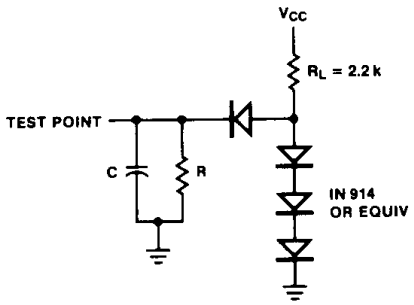


Fig. 32 Bus Timing Test Loads



Test Conditions

The dynamic test load for the data bus is 130 pF and one standard TTL load, as shown. The Address, R/$\overline{W}$, 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.

C = 130 pF for D_0-D_7, E
= 90 pF for A_0-A_{15}, R/$\overline{W}$, and VMA (except t_{AD2})
= 30 pF for A_0-A_{15}, R/$\overline{W}$, and VMA (t_{AD2} only)
= 30 pF for BA.

R = 11.7 kΩ for D_0-D_7
= 16.5 kΩ for A_0-A_{15}, R/$\overline{W}$, and VMA
= 24 kΩ for BA

www.DataSheet4U.com

Ordering Information

Speed	Order Code	Temperature Range
1.0 MHz	F6800P,S	0 to +70°C
	F6800CP,CS	−40 to +85°C
	F6800DM	−55 to +125°C
1.5 MHz	F68A00P,S	0 to +70°C
	F68A00C,CS	−40 to +85°C
	F68A00DM	−55 to +125°C
2.0 MHz	F68B00P,S	0 to +70°C
	F68B00C,CS	−40 to +85°C
	F68B00DM	−55 to +125°C

*P = plastic package, S = CER-DIP package.

www.DataSheet4U.com