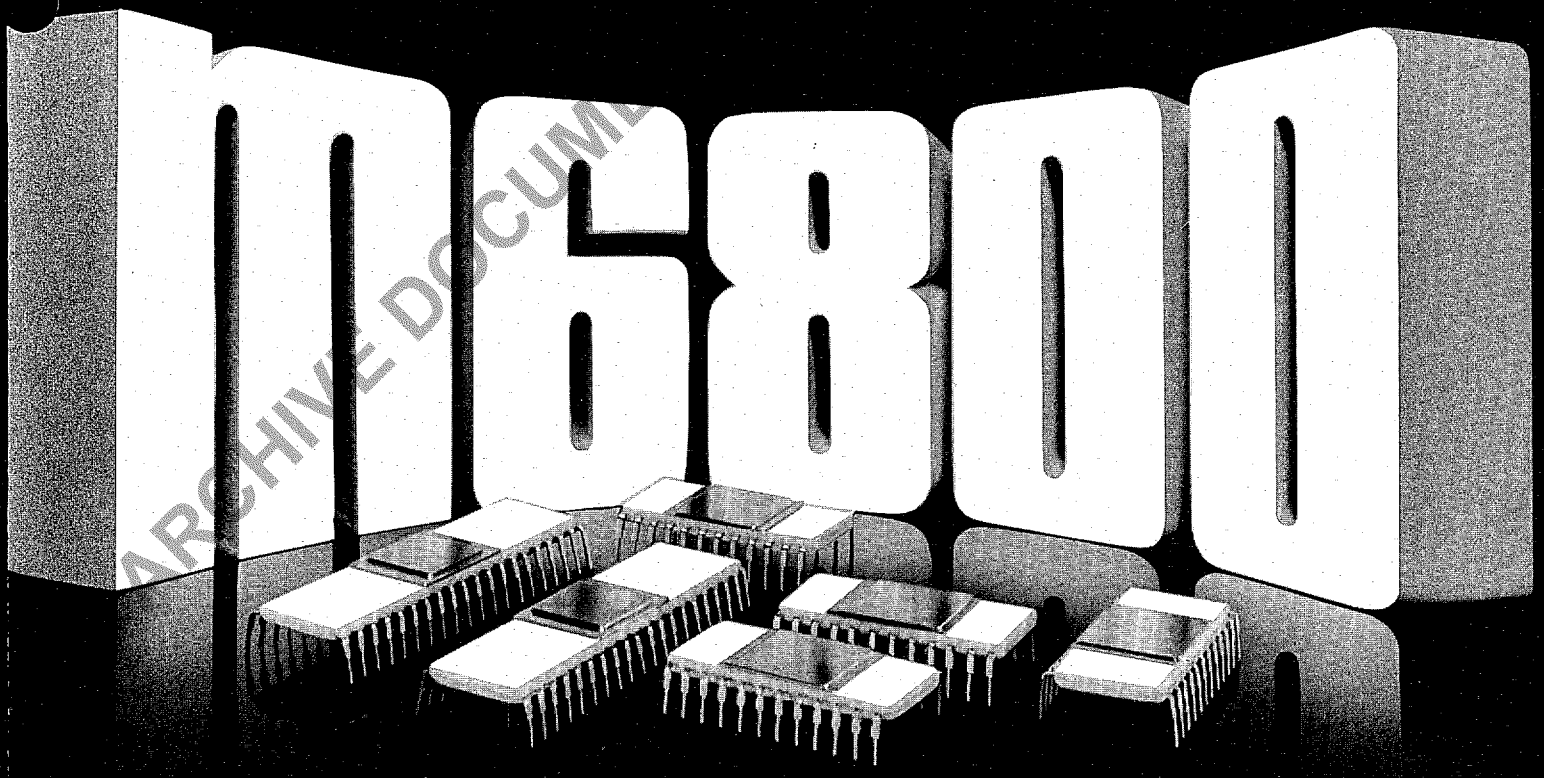
MOTOROLA

M68PRM(D)

M6800
PROGRAMMING REFERENCE MANUAL



M68PRM(D)
Nov., 1976

M6800 PROGRAMMING REFERENCE MANUAL

First Edition

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

CHAPTER 1
INTRODUCTION

1.0 INTRODUCTION

Motorola Microsystem's software and development tools for the M6800 have been designed to simplify the implementation of systems using the M6800 Microcomputer Family. The M6800 Programming Reference Manual is the basic software reference document to be used as a supplement to reference manuals for specific software products. It includes descriptions of:

- M6800 Program-visible Registers
- Interrupts and Stack Operations
- M6800 Addressing Modes
- M6800 Instruction Set
- Commands for
  - — EXbug
  - — MIKbug
  - — MINIbug II
  - — MINIbug III

The manual also includes descriptions of basic M6800 Microcomputer Family components:

- — MPU
- — ROM
- — RAM
- — PIA
- — ACIA

Available User's Guides and software reference manuals include:

- M6800 EXORciser User's Guide
- EDOS II Operator's Manual
- Co-Resident Assembler Reference Manual
- Co-Resident Editor Reference Manual
- Linking Loader Reference Manual
- Macro Assembler Reference Manual
- Resident FORTRAN Reference Manual
- M68SAM Cross Assembler Reference Manual
- M68EML Simulator Reference Manual
- M68MPL Compiler Reference Manual

CHAPTER 2
HARDWARE DESCRIPTION

2.0 INTRODUCTION

The MC6800 MPU is the nucleus of a series of fully bus-compatible, silicon gate NMOS building blocks which are interconnected into the desired microcomputer system configuration.

Development tools are also available which emulate system function and performance so that hardware may be evaluated and system software and firmware generated and debugged. The most powerful is the M6800 EXORciser. By configuring its modules in his system's likeness, the user possesses a surrogate prototype with which to edit, assemble and modify his programs in real time on the actual hardware.

2.1 THE BASIC MICROCOMPUTER COMPONENTS

A minimum system can be assembled with four LSI (Large Scale Integration) bus oriented parts:

MPU — Microprocessor
RAM — 128 X 8 Random Access Memory
ROM — 1024 X 8 Read Only Memory
PIA — Peripheral Interface Module

2.1.1 A Minimum System

These parts can be interconnected without interface parts making a minimum functional system (Figure 2-1). Such a system can easily be adapted for a number of small scale applications by simply changing the application program content of the ROM.

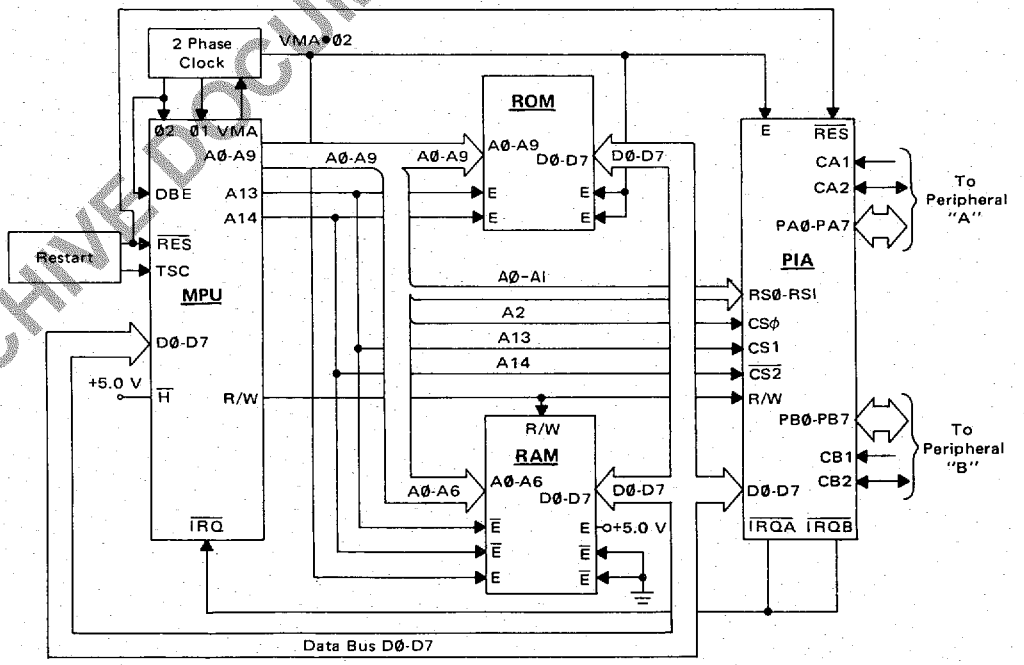


FIGURE 2-1. Minimum System Configuration

The minimum system may be expanded without the addition of TTL integrated circuits providing the load does not exceed the capacity of the MPU. The MPU has capacity to drive a load of one standard TTL input and 130 picofarads at one megahertz.

2.1.1.1 *MPU – Microprocessing Unit*

Some of the more important features of the MC6800 Microprocessing Unit that contribute to the "ease of use" in a system are:

- Eight-bit parallel processing
- Bi-directional data bus
- Sixteen-bit address bus — 65k bytes of addressing
- 72 instructions — variable length
- Seven addressing modes — Direct, Relative, Immediate, Indexed, Extended, Implied, and Accumulator
- Interrupt vectoring
- Two Accumulators
- Index Register
- Program Counter
- Stack Pointer and variable length stack
- Condition Code Register (6 codes)
- Separate Non-Maskable Interrupt
- Direct Memory Access (DMA) and multiple processor capability
- Clock operating rates up to 1 MHz
- Simple interface without TTL
- Halt/ Go and single instruction execution capability
- 40 pin package

A programming model of the microprocessing unit is represented in Figure 2-2. This comprises all of the registers in the MPU which are controlled explicitly by programs. The inputs and outputs of the MPU form five functional groups (see Figure 2-3):

1. 8-bit data bus
2. 16-bit address bus
3. Controls
4. Clock input signals
5. Power supply and ground

This manual, being concerned with the programming of the MPU, primarily considers the information being transferred on the data bus and the address bus connected to the MPU.

In this manual, the control signals are considered only as they affect execution of a program. The control signals are as follows:

- Reset
- Halt
- Non-Maskable Interrupt
- Interrupt Request
- Read/ Write
- Data Bus Enable
- Valid Memory Address
- Three-state Control
- Bus Available

Reset, Halt, Non-Maskable Interrupt, and Interrupt Request are considered in Chapter 3. Read/ Write, Data Bus Enable, and Valid Memory Address are considered only as they relate to the

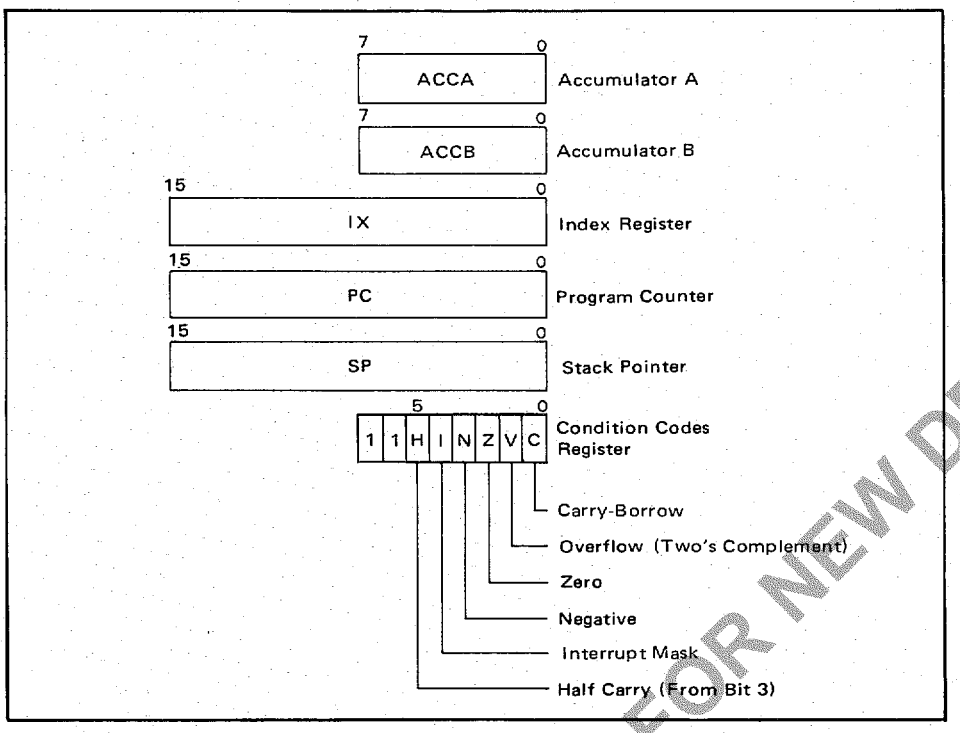


FIGURE 2-2. Programming Model of the M6800 Microprocessor

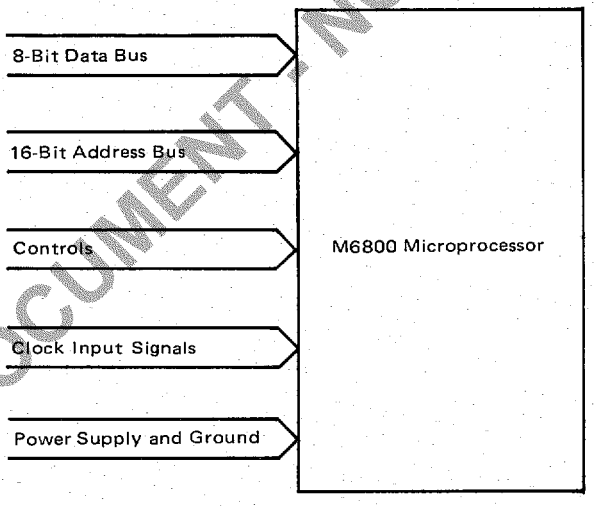


FIGURE 2-3. M6800 Microprocessor — Inputs and Outputs

addressing of the memory and interface units. Three-State Control, Bus Available, and the Halt control lines function in Direct Memory Access (DMA) and multiple processor operations.

The execution time of any instruction in a program is directly proportional to the clock period and is, therefore, expressed as a number of clock cycles. Apart from the times of execution of instructions, the clock inputs are not otherwise considered in this manual.

The information contained in this programming manual applies to programs written for execution by the microprocessing unit, and is not restricted to any particular set of parts with which the MPU may be interconnected in any system. Motorola offers a family of parts that are compatible with the MPU and which may be easily assembled into a computing system giving the user the most cost effective solution to his problem.

2.1.1.2 *ROM – 1024 X 8-Bit Read Only Memory*

The MCM6830 is a mask-programmable byte-organized memory designed for use in bus-organized systems. It is fabricated with N-channel silicon-gate technology. The ROM operates from a single power supply, is compatible with TTL and DTL, and needs no clocks or refreshing because of static operation.

The memory is compatible with the M6800 Microcomputer Family, providing read only storage in byte increments. Memory expansion is provided through multiple chip select inputs. The active level of the chip select inputs and the memory content are defined by the customer. Some of the important features of the ROM are:

- Organized as 1024 bytes (1 byte = 8 bits)
- Static operation
- Three-state data output
- Four chip select inputs (mask option)
- Single 5 volt power supply
- TTL/ DTL compatible
- Maximum access time = 575 ns

2.1.1.3 *RAM – 128 X 8-Bit Static Random Access Memory*

The MCM6810 is a byte-organized memory designed for use in bus-organized systems. It is fabricated with N-channel silicon-gate technology. The RAM operates from a single power supply, has compatibility with TTL and DTL, and needs no clocks or refreshing because of static operation.

The memory is compatible with the M6800 Microcomputer Family, providing random storage in byte increments. Memory expansion is provided through multiple chip select inputs. Some of the important features of the RAM are:

- Organized as 128 bytes (1 byte = 8 bits)
- Static operation
- Bi-directional, three-state, data input/ output
- Six chip select inputs (four active low, two active high)
- Single 5-volt power supply
- TTL/ DTL compatible
- Maximum access time = 1.0 us for MCM6810L
  575 ns for MCM6810L-1

2.1.1.4 *PIA – Peripheral Interface Adapter*

The MC6820 Peripheral Interface Adapter provides an effective means of interfacing peripheral equipment to the MC6800 Microprocessing Unit (MPU). This device is capable of interfacing the MPU to peripherals through two, 8-bit, bi-directional, peripheral data buses and four control lines. No external logic is required for interfacing to most peripheral devices.

The functional configuration of the PIA is programmed by the MPU during system initialization. Each of the peripheral data lines can be programmed to act as an input or output, and each of the four control/ interrupt lines may be programmed for one of several control modes, as shown in Figure 2-4. This allows a high degree of flexibility in the over-all operation of the interface. Some of the important features of the PIA are:

- An 8-bit bi-directional data bus for communication with the MPU
- Two bi-directional 8-bit buses for interface to peripherals
- Two programmable control registers

Determine Active CA1 (CB1) Transition for Setting Interrupt Flag IRQA(B)1 – (bit b7)

b1 = 0 : IRQA(B)1 set by high-to-low transition on CA1 (CB1).

b1 = 1 : IRQA(B)1 set by low-to-high transition on CA1 (CB1).

CA1 (CB1) Interrupt Request Enable/Disable

b0 = 0 : Disables IRQA(B) MPU Interrupt by CA1 (CB1) active transition.[1]

b0 = 1 : Enable IRQA(B) MPU Interrupt by CA1 (CB1) active transition.

1. IRQA(B) will occur on next (MPU generated) positive transition of b0 if CA1 (CB1) active transition occurred while interrupt was disabled.

IRQA(B) 1 Interrupt Flag (bit b7)

Goes high on active transition of CA1 (CB1); Automatically cleared by MPU Read of Output Register A(B). May also be cleared by hardware Reset.

b7	b6	b5	b4	b3	b2	b1	bϕ
IRQA(B)1 Flag	IRQA(B)2 Flag	CA2(CB2) Control			DDR Access	CA1(CB1) Control	

IRQA(B)2 Interrupt Flag (bit b6)

CA2 (CB2) Established as Input (b5 = 0): Goes high on active transition of CA2 (CB2); Automatically cleared by MPU Read of Output Register A(B). May also be cleared by hardware Reset.

CA2 (CB2) Established as Output (b5 = 1): IRQA(B)2 = 0, not affected by CA2 (CB2) transitions.

Determines Whether Data Direction Register Or Output Register is Addressed

b2 = 0 : Data Direction Register selected.

b2 = 1 : Output Register selected.

CA2 (CB2) Established as Output by b5 = 1

b5 b4 b3 (Note that operation of CA2 and CB2 output functions are not identical)

1 0

→ CA2

b3 = 0 : Read Strobe With CA1 Restore

CA2 goes low on first high-to-low E transition following an MPU Read of Output Register A; returned high by next active CA1 transition.

b3 = 1 : Read Strobe with E Restore

CA2 goes low on first high-to-low E transition following an MPU Read of Output Register A; returned high by next high-to-low E transition.

→ CB2

b3 = 0 : Write Strobe With CB1 Restore

CB2 goes on low on first low-to high E transition following an MPU Write into Output Register B; returned high by the next active CB1 transition.

b3 = 1 : Write Strobe With E Restore

CB2 goes low on first low-to-high E transition following an MPU Write into Output Register B; returned high by the next low-to-high E transition.

b5 b4 b3

1 1

→ Set/Reset CA2 (CB2)

CA2 (CB2) goes low as MPU writes b3 = 0 into Control Register.

CA2 (CB2) goes high as MPU writes b3 = 1 into Control Register.

CA2 (CB2) Established as Input by b5 = 0

b5 b4 b3

0

→ CA2 (CB2) Interrupt Request Enable/Disable

b3 = 0 : Disables IRQA(B) MPU Interrupt by CA2 (CB2) active transition.[1]

b3 = 1 : Enables IRQA(B) MPU Interrupt by CA2 (CB2) active transition.

1. IRQA(B) will occur on next (MPU generated) positive transition of b3 if CA2 (CB2) active transition occurred while interrupt was disabled.

→ Determines Active CA2 (CB2) Transition for Setting Interrupt Flag IRQA(B)2 – (bit b6)

b4 = 0 : IRQA(B)2 set by high-to-low transition on CA2 (CB2).

b4 = 1 : IRQA(B)2 set by low-to-high transition on CA2 (CB2).

FIGURE 2-4. PIA Control Register Format

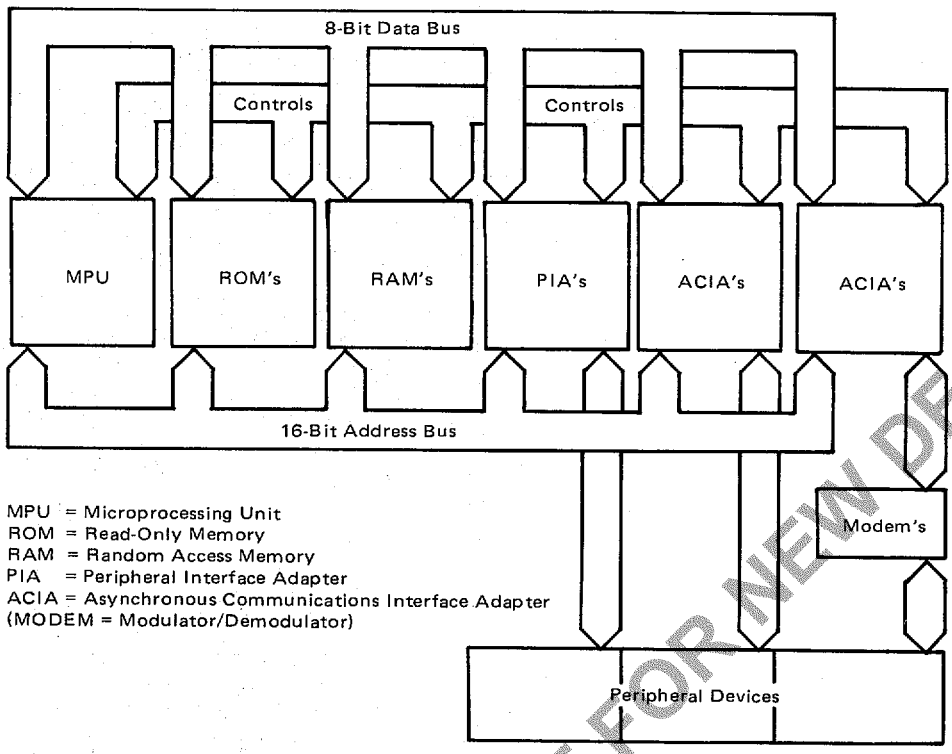


FIGURE 2-5. Expanded M6800 Microcomputer

- Two programmable data direction registers
- Four individually controlled interrupt input lines; Two usable as peripheral control outputs
- Handshake control logic for input and output peripheral operation
- High-impedance three-state and direct transistor drive peripheral lines
- Program controlled interrupt and interrupt disable capability
- CMOS compatible peripheral lines

2.1.2 Expanding the Basic System

The minimum system can be expanded with other family parts to meet the needs of more complex systems (see Figure 2-5). Motorola also is continually developing new parts to add to the family.

2.1.2.1 *ACIA – Asynchronous Communications Interface Adapter*

The MC6850 Asynchronous Communications Interface Adapter provides the data formatting and control to interface serial asynchronous data communications information to bus organized systems such as the MC6800 Microprocessing Unit.

The bus interface of the MC6850 includes select, enable, read/ write, interrupt and bus interface logic to allow data transfer over an 8-bit bi-directional data bus. The parallel data of the bus system is serially transmitted and received by the asynchronous data interface, with proper formatting and error checking. The functional configuration of the ACIA is programmed via the data bus during system initialization. A programmable control register provides variable word lengths, clock division ratios, transmit control, receive control, and interrupt control (see Figures 2-6 and 2-7). For peripheral or modem operation, three control lines are provided. These lines allow the ACIA to interface directly with the MC6860L 0-600 bps Digital Modem. Some of the features of the ACIA are:

- Eight and nine-bit transmission

- Optional even and odd parity
- Parity, overrun and framing error checking
- Programmable control register
- Optional ÷ 1, ÷ 16, and ÷ 64 clock modes
- Up to 500 Kbps transmission
- False start bit detection
- Peripheral/ modem control functions
- Double buffered
- One or two stop bit operation

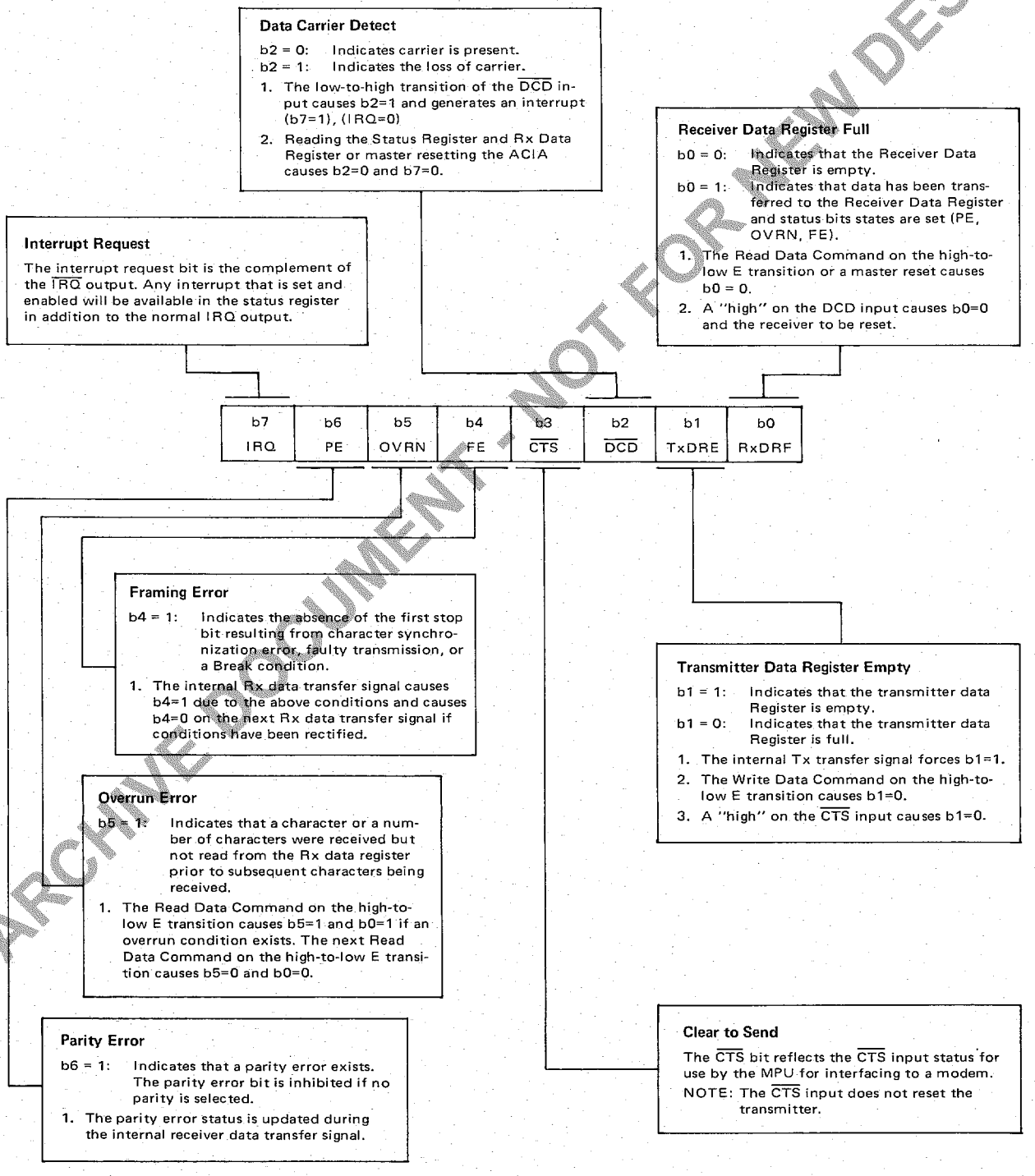


FIGURE 2-6. ACIA Status Register Format

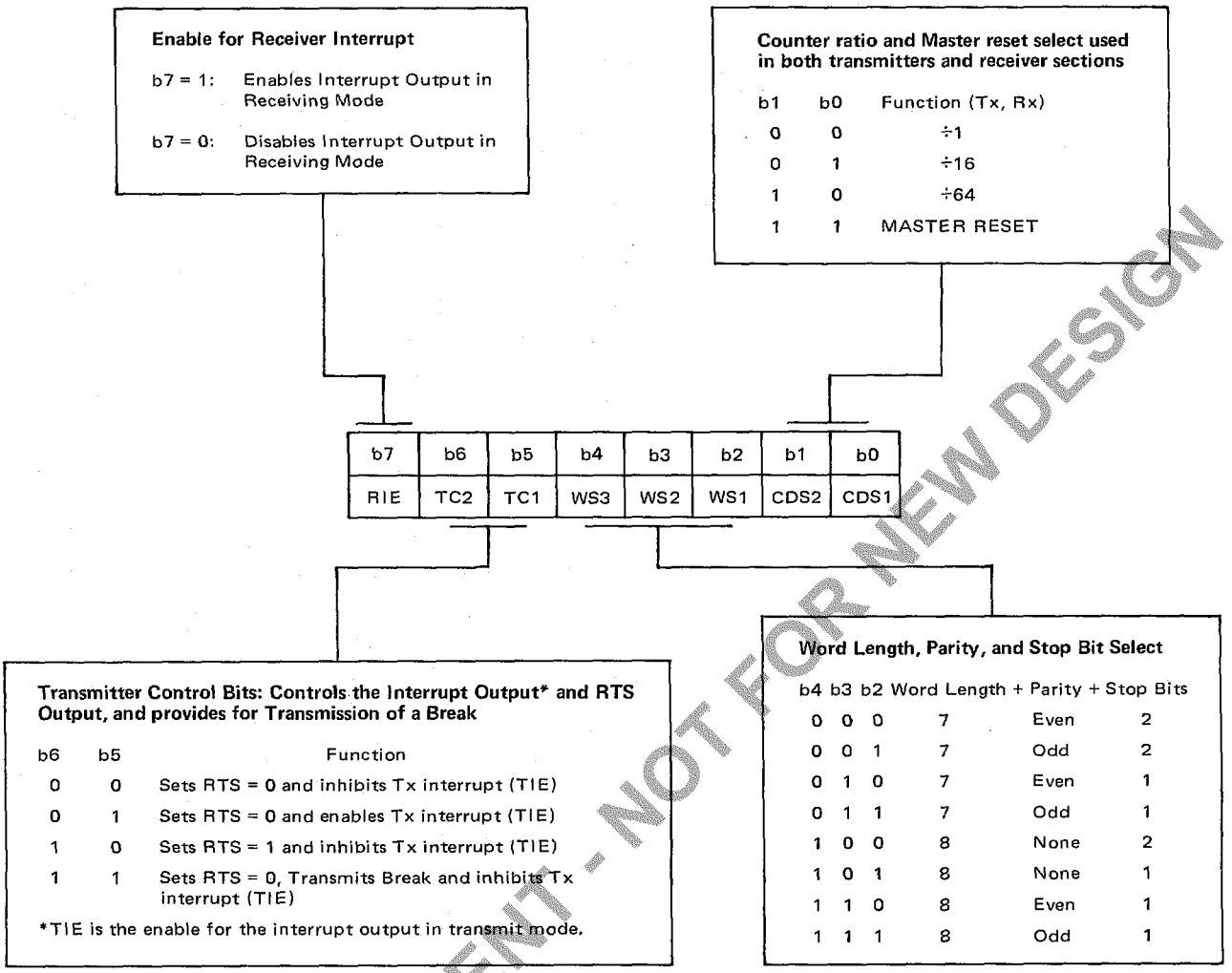


FIGURE 2-7. ACIA Control Register Format

CHAPTER 3
PROGRAMMING THE M6800 MICROPROCESSOR

3.0 MACHINE CODE

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 in Chapter 4).

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 Figure 3-1. 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. This is explained along with a description of the different addressing modes in Chapter 4 of this manual.

3.1 STACK AND STACK POINTER

The stack consists of any number of locations in RAM memory. The stack provides for temporary storage and retrieval of successive bytes of information, which may include any of the following items:

- current status of the MPU
- return address
- data

The stack can be used for the following purposes:

- interrupt control
- subroutine linkage
- temporary storage of data (under control of the program)
- reentrant code

The microprocessing unit includes a 16-bit stack pointer. This contains an address which enables the MPU to find the current location of the stack.

When a byte of information is stored in the stack, it is stored at the address which is contained in the stack pointer. The stack pointer is decremented (by one) immediately following the storage in the stack of each byte of information. Conversely, the stack pointer is incremented (by one) immediately before retrieving each byte of information from the stack, and the byte is then obtained from the address contained in the stack pointer. The programmer must ensure that the stack pointer is initialized to the required address before the first execution of an instruction which manipulates the stack.

Normally, the stack will consist of a single block of successive memory locations. However, some instructions in the source language change the address contained in the stack pointer without storing or retrieving information into or from the stack. The use of these instructions may result in the stack being other than one continuous sequence of memory locations. In such a case, it may alternatively be considered that there exist two or more stacks, each of which consists of a block of successive locations in the memory.

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	LDA	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	CLV		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		REL	CD	*		
0E	CLI		4E	*			8E	LDS		IMM	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		DIR	DC	*		
1D	*		5D	TST	B		9D	*			DD	*		
1E	*		5E	*			9E	LDS		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	LDX		IND
2F	BLE	REL	6F	CLR		IND	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	DES		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	ADC	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	IMM	= Immediate
B	= Accumulator B	DIR	= Direct
REL	= Relative		
IND	= Indexed		

2. Unassigned code indicated by "*".

TABLE 3-1. Hexadecimal Values of Machine Codes

3.2 SAVING MPU STATUS

The status of the microprocessing unit is saved in the stack during the following operations:

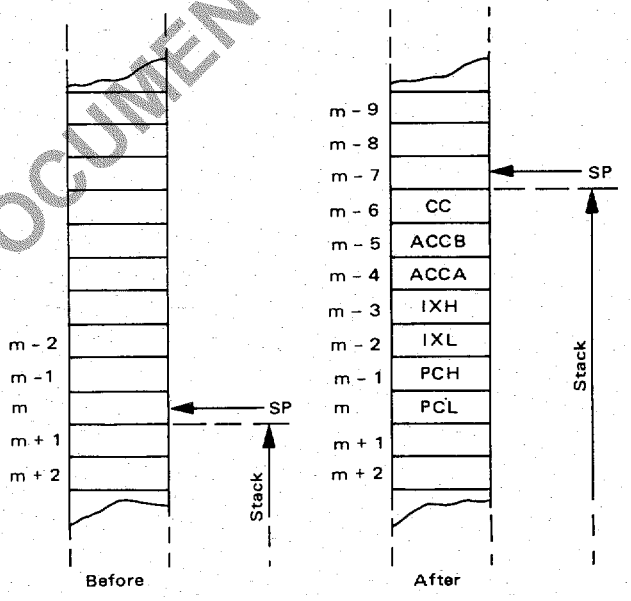
- in response to an external condition indicated by a negative edge on the Non-Maskable Interrupt control input signal to the MPU.
- during execution of the machine code corresponding to either of the source language instructions SWI (Software Interrupt) or WAI (Wait for Interrupt).
- during servicing of an interrupt from a peripheral device, in response to a negative edge on the Interrupt Request control input signal to the MPU and provided the interrupt mask bit (I) is clear.

The status is stored in the stack in accordance with the scheme shown in Figure 3-2. Before storing the status, the stack pointer contains the address of a memory location represented in Figure 3-2 by "m." The stack, if any, extends from location "m + 1" to higher locations. The status is stored in seven bytes of memory, beginning with the byte at location "m," and ending with the byte at location "m – 6." The stack pointer is decremented after each byte of information is entered into the stack.

The information which is saved in the stack consists of the numerical content of all of the registers of the programming model, shown in Figure 2-2, except the stack pointer.

The value stored for the program counter (PCH and PCL) is in accordance with the following rules:

1. In response to a Non-Maskable Interrupt or to an interrupt from a peripheral device, the value saved for the program counter is the address of that instruction which would next be executed, if the interrupt had not occurred.
2. During execution of a SWI or WAI instruction, the value saved for the program counter is the address of that SWI or WAI instruction, plus one.



SP = Stack Pointer
CC = Condition Codes (Also called the Processor Status Byte)
ACCB = Accumulator B
ACCA = Accumulator A
IXH = Index Register, Higher Order 8 Bits
IXL = Index Register, Lower Order 8 Bits
PCH = Program Counter, Higher Order 8 Bits
PCL = Program Counter, Lower Order 8 Bits

FIGURE 3-2. Saving the Status of the Microprocessor in the Stack

The values stored for the other registers (CC, ACCB, ACCA, IXH and IXL) are in accordance with the following rules:

1. In response to a Non-Maskable Interrupt, or an interrupt from a peripheral device, the values saved are those which resulted from the last instruction executed before the interrupt was serviced.
2. During execution of a SWI or WAI instruction, the values saved are those which resulted from the last instruction executed before the SWI or WAI instruction.
3. The condition codes H, I, N, Z, V, and C, in bit positions 5 thru 0 of the processor condition code register, are stored respectively in bit positions 5 thru 0 of the applicable memory location in the stack. Bit positions 7 and 6 of that memory location are set (go to the 1 state).

3.3 INTERRUPT POINTERS

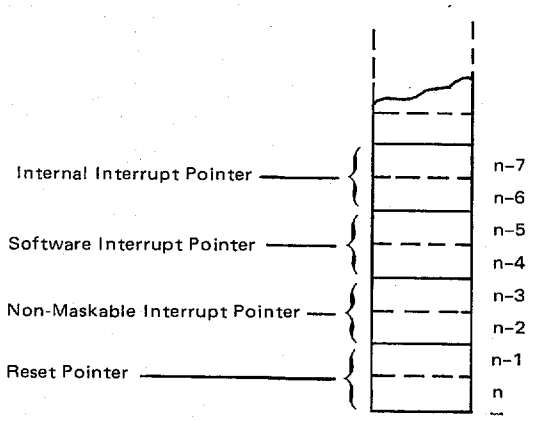
A block of memory is reserved for pointers, which provide for read-only storage of the addresses of programs which are to be executed in the event of a reset (or power on), a low state of the Non-Maskable Interrupt control input, a software interrupt, or a response to an interrupt signal from a peripheral device. The respective pointers each occupy two bytes of memory and are disposed at locations from "n – 7" to "n," as indicated in Figure 3-3.

The location indicated in Figure 3-3 by "n" is that location which is addressed when all the lines of the address bus are in the high ("1") state. In most systems, the location "n" will be the highest address in the memory. However, the correspondence of "n" to a particular numerical value depends on the hardwired interconnections of the parts of the programmable system to the address bus.

3.3.1 Reset (or Power On)

The Reset control input to the MPU is used to start the execution of the program, either for initial start-up or from a power down condition following a power failure. When a positive edge is detected on this input line, the program counter is loaded with the address stored in the restart pointer at locations "n – 1" and "n" of memory (see Figure 3-3). The MPU then proceeds with execution of a Restart Program, which begins with the instruction addressed by the program counter. The restart and the continued execution, however, depends on the Go/ Halt control input being in the "Go" condition.

When the Go/ Halt control input is in the high state, the machine will fetch the instruction addressed by the program counter and start execution. When this line changes to a low, execution will



n = Memory Location Addressed When All Lines of The Address Bus are in the High (1) State.

FIGURE 3-3. Reset and Interrupt Pointers

stop. The stop may become effective at the completion of execution of the current instruction. Alternatively, one more instruction may be executed before the stop becomes effective, due to the look-ahead capability described in Section 3.3.8. Execution of the program will not be resumed until the "Go" condition is restored.

TheGo/ Halt input must remain in the "Go" condition for the interrupt sequences to be completed. Otherwise, the machine will stop execution at the end of an instruction. The following sections of this manual, which describe the interrupt operations, assume that the "Go" state is maintained.

3.3.2 NMI — Non-Maskable Interrupt

The sequence of operations, which occurs following a non-maskable interrupt, is initiated by a negative edge on the Non-Maskable Interrupt control input to the MPU. Execution of the current instruction is completed. The response of the MPU to the Non-Maskable Interrupt signal may begin on the completion of execution of the current instruction. Alternatively, one or more instructions in the program may first be executed, due to the look-ahead capability of the MPU described in Section 3.3.8.

The status of the MPU is then saved in the stack, as described in Section 3.2 and the program counter is loaded with the address stored in the Non-Maskable Interrupt pointer at locations "n – 3" and "n – 2" of memory (see Figure 3-3). The MPU then starts execution of the Non-Maskable Interrupt Program, which begins with the instruction which is now addressed by the program counter..

3.3.3 SWI — Software Interrupt

During execution of the SWI instruction, the status of the MPU is saved in the stack, as described in Section 3.2. The value saved for the program counter is the address of the SWI instruction, plus one.

After the status has been saved, the interrupt mask bit "I" is set (I = 1). The MPU will not respond to an interrupt request from a peripheral device while the interrupt mask bit is set.

The program counter is then loaded with the address stored in the software interrupt pointer at locations "n – 5" and "n – 4" of memory (see Figure 3-3). The MPU then proceeds with execution of a Software Interrupt Program, which begins with the instruction whose address is now in the program counter.

The MPU will remain insensitive to an interrupt request from any peripheral device (signalled by a "low" state of the Interrupt Request control input signal to the MPU) until the interrupt mask bit has been reset by execution of the programmed instructions.

3.3.4 IRQ — Interrupt Request

A request for an interrupt by a peripheral device is signalled by a low state of the Interrupt Request control input to the MPU (IRQ).

The MPU will not respond to an Interrupt Request while the interrupt mask bit is set (I = 1). Normal execution of the program continues until the interrupt mask bit is reset (I = 0)enabling the MPU to respond to the Interrupt Request.

Execution of the current instruction will always be completed before the MPU responds to an Interrupt Request. The response of the MPU to the Interrupt Request may begin on the completion of the current instruction. Alternatively, one more instruction in the program may first be executed, due

to the look-ahead capability of the MPU described in Section 3.3.8. The Response of the MPU to the Interrupt Request then proceeds as follows:

1. Saving the Status
   Provided the last instruction executed was not a WAI instruction, the status of the MPU is saved in the stack, as described in Section 3.2. The value saved for the program counter is the address of the instruction which would be the next to be executed if the interrupt had not occurred. If the last instruction executed was a WAI instruction, the address of the next instruction is not saved since PC and MPU status were already saved by the WAI instruction in preparation for an interrupt.
2. Interrupt Mask
   The interrupt mask bit is then set (I = 1). This prevents the MPU from responding to further interrupt requests until the interrupt mask bit has been cleared by execution of programmed instructions.
3. Internal Interrupt Pointer and Program
   The program counter is loaded with the address stored in the internal interrupt pointer at locations "n – 7" and "n – 6" of memory (see Figure 3-3). The MPU then proceeds with execution of an internal interrupt program, which begins with the instruction currently being addressed by the program counter. The internal interrupt pointer is selected by logic which is internal to the MPU. At the point when execution of the internal interrupt program begins, no distinction will have been made regarding the source of the interrupt request. In a system in which there is more than one possible source of interrupt request, the internal interrupt program must include a routine for identifying the origin of the request. In a system composed of the Motorola Microcomputer Kit, this routine would consist of a programmed interrogation of the addressable registers of the PIAs and ACIAs, in order to identify the peripheral device which has requested the interrupt.

3.3.5 WAI — Wait Instruction

During execution of the WAI instruction, the status of the MPU is saved in the stack, as described in Section 3.2. The value saved for the program counter is the address of the WAI instruction, plus one.

Execution of the WAI instruction does not change the interrupt mask bit.

If the interrupt mask bit is set (I = 1), the MPU cannot respond to an interrupt request from any peripheral device. Execution stops after MPU status is saved and can be resumed only via a Non-Maskable Interrupt or a reset interrupt.

If the interrupt mask bit is in the reset state (I = 0), the MPU will service any interrupt request which may be present. If the Interrupt Request input is in the high state, execution will be suspended, and the MPU will wait for an Interrupt Request to be signalled. If an Interrupt Request is signalled by the Interrupt Request input changing to the low state, the interrupt will be serviced as previously described: the interrupt mask bit will be set, the program counter will be loaded with the address stored in the internal interrupt pointer, and execution of the internal interrupt program will begin.

3.3.6 Manipulation of the Interrupt Mask Bit

The interrupt mask bit is affected by execution of the source language instructions SWI and RTI, and by the servicing of an interrupt request from a peripheral device, as has been previously

described. The interrupt mask may also be manipulated by the use of any of the following instructions:

- CLI — clear interrupt mask bit
- SEI — set interrupt mask bit
- TAP — transfer accumulator A to processor condition codes register

The state of the interrupt mask bit can also be affected as a result of the following instruction:

- TPA — transfer the processor condition codes register to accumulator A.

During execution of the TPA instruction, the condition codes H, I, N, Z, V, and C, in bit positions 5 thru 0 of the processor condition codes register are stored respectively in bit positions 5 thru 0 of accumulator A. Bit positions 7 and 6 of accumulator A are set (i.e. go to the 1 state). After execution of the TAP instruction, the state of each of the condition codes (H, I, N, Z, V, C) will be whatever is retrieved from the respective bit positions (5 thru 0) of accumulator A.

3.3.7 Special Programming Requirements

A comprehensive program should make provision for the following special requirements:

(a) Pointers:
The program should place the addresses of the reset and interrupt routine in the respective pointers (see Figure 3-3) at the high-address end of memory. The addresses would usually be placed in the pointers by use of the FDB assembler directive in the source program.

(b) Reset and Interrupt Sequences:
The sequences of instructions to be addressed by the Reset pointer, the Non-Maskable Interrupt pointer, the Software Interrupt pointer, and the Internal Interrupt pointer, should be provided in the program.

(c) Input and Output:
The program would normally include provisions for inputs and outputs relating to peripheral devices. In a programmable system composed of the parts of the Motorola Microcomputer Family, the input and output routines would involve reading and writing coded data from and into the addressable registers of the PIAs and ACIAs. The input and output routines would normally be reached via conditional branch instructions in the Internal Interrupt Program.

3.3.8 Look-Ahead Feature

The MPU responds, at the completion of the instruction being executed, to any of the following signals:

- Halt
- Non-Maskable Interrupt
- Interrupt Request (when the interrupt mask is in the reset state).

However, if the interrupt occurs during the last cycle of an instruction, the look-ahead to the next instruction feature will mask the interrupt until the completion of the next instruction.

3.3.9 RTI — Return from Interrupt

The source language instruction RTI assembles into one byte of the machine code. Execution of this instruction consists of the restoration of the MPU to a state pulled from the stack.

The information which is obtained from the stack provides for the numerical content of the registers of the programming model shown in Figure 2-2. The operation is the reverse of that represented in Figure 3-2. Seven bytes of information are pulled from the stack and stored in

respective registers of the MPU. The address stored in the stack pointer is incremented before each byte of information is pulled from the stack.

After execution of the RTI instruction, the state of each of the condition codes (H, I, N, Z, V, and C) will be whatever is retrieved from the respective bit positions (5 thru 0) of the applicable memory location in the stack. In particular, it should be noted that the interrupt mask bit (I-bit) may be either set or reset by execution of the RTI instruction.

3.4 SUBROUTINE LINKAGE

The stack provides an orderly method of calling a subroutine and returning from the subroutine. Use of a stack allows subroutine calls when in a subroutine (subroutine nesting).

3.4.1 Call Subroutine (BSR or JSR)

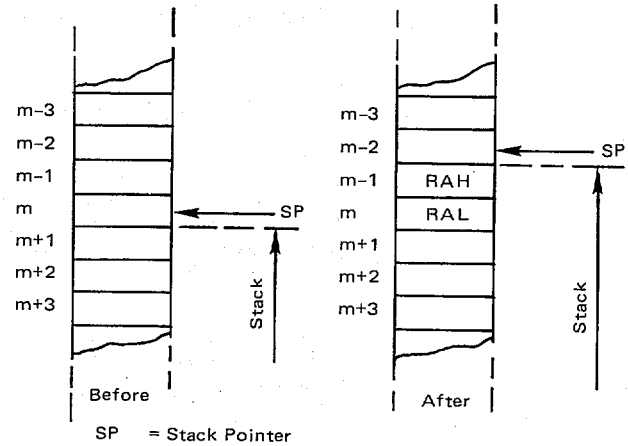
A return address is saved in the stack during execution of the machine code corresponding to either of the source language instructions BSR (branch to subroutine) or JSR (jump to subroutine).

The return address is stored in the stack in accordance with the scheme shown in Figure 3-4. Before storing the return address, the stack pointer contains the address of a memory location represented in Figure 3-4 by "m." The stack, if any, extends from memory location "m + 1" to higher locations. The return address is stored in two bytes of memory, at locations "m − 1" and "m." The stack pointer is decremented after each byte of the return address is pushed into the stack.

For either of the instructions (BSR or JSR), the return address saved in the stack is that of the next byte of memory following the bytes of code which correspond to the BSR or JSR instruction. Thus, for the BSR instruction, the return address is equal to the address of the BSR instruction, plus two. For the JSR instruction, the return address is equal to the address of the JSR instruction, plus three or plus two; according to whether the instruction is used with the extended or the indexed mode of addressing.

3.4.2 RTS — Return From Subroutine

During execution of the RTS instruction, the return address is obtained from the stack and loaded into the program counter. The address stored in the stack pointer is incremented before each byte of the return address is pulled from the stack. This operation is the reverse of that represented in Figure 3-4.



SP = Stack Pointer
RAH = Return Address, Higher Order 8-Bits
RAL = Return Address, Lower Order 8-Bits

FIGURE 3-4. Saving a Return Address in the Stack

3.5 DATA STORAGE IN THE STACK

The source language instruction PSH is used for storing a single byte of data in the stack. This instruction addresses either register A or register B. The contents of the specified register is stored in the stack, in accordance with the scheme represented in Figure 3-5. The address contained in the stack pointer is decremented.

Conversely, the source language instruction PUL retrieves data from the stack. This instruction addresses either register A or register B. The address contained in the stack pointer is incremented. A single byte of data is then obtained from the stack and loaded into the specified register. The operation is the reverse of that represented in Figure 3-5.

3.6 REENTRANT CODE

Reentrant code is an attribute of a program that allows the program to be interrupted during execution, entered by another user, and subsequently, reentered at the point of interruption by the first user, thus producing the desired results for all users: a program with an intermediate state of execution that is totally restorable when it is reentered after an interruption.

The instruction TSX allows data on the stack to be manipulated by the indexed mode of addressing.

3.7 MANIPULATION OF THE STACK POINTER

The address saved in the stack pointer is affected by execution of the source language instructions (SWI, WAI, RTI, BSR, JSR, RTS, PSH, and PUL) and also by the servicing of a Non-Maskable Interrupt or an Interrupt Request from a peripheral device, as previously described. In these operations, the stack pointer is coordinated with the storing and retrieval of information in the stack.

The address in the stack pointer may also be manipulated without storing or retrieving information in the stack. This is carried out by the following source language instructions:

- DES — decrement stack pointer
- INS — increment stack pointer
- LDS — load the stack pointer
- TXS — transfer index register to stack pointer

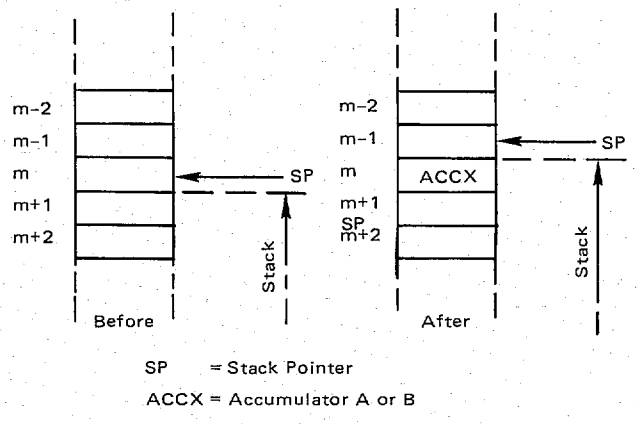


FIGURE 3-5. Data Storage in the Stack

The use of any of these four instructions can result in the stack being other than a block of successive locations in memory.

The content of the stack pointer is also involved in execution of the following instructions:

- STS — store the stack pointer
- TSX — transfer stack pointer to index register

The instruction TSX loads the index register with a value equal to the contents of the stack pointer, plus 1. The instruction TXS loads the stack pointer with a value equal to the contents of the index register, minus 1. This is in accordance with the operation of the stack pointer during execution of the instructions SWI, WAI, BSR, JSR, or PSH, or during servicing of an interrupt from a peripheral device; in which case the stack pointer is set to one less than the address of the last byte stored in the stack.

CHAPTER 4
M6800 MICROPROCESSOR ADDRESSING MODES

4.0 ADDRESSING MODES

The assembler scans the operator and operand to determine the proper addressing mode. The addressing modes are:

- Dual Addressing
- Accumulator Addressing
- Inherent Addressing
- Immediate Addressing
- Relative Addressing
- Indexed Addressing
- Direct and Extended Addressing

4.1 DUAL ADDRESSING

Eleven of the executable instructions require addressing of two operands in the operand field. These instructions are indicated in Figure 4-1 by the column headed Dual Operand. For all of these operators the first operand must be either accumulator A or accumulator B. This is specified respectively by A or B as the first character in the operand field, the second character in the operand field being a SPACE.

For dual addressing the specification of the first operand (either A or B) is separated from that of the second operand by one or more SPACE characters.

The second operand is specified in the operand field in accordance with the rules for immediate, direct, extended, or indexed addressing (as subsequently defined); depending on which modes of addressing are valid for the individual operators. (For nmemonic operators which employ dual addressing, it is permissible to omit the SPACE between the operator and the first operand field — LDAA LABEL).

4.2 ACCUMULATOR ADDRESSING (SINGLE OPERAND)

Thirteen of the operators address a single operand from the operand field and, thus, can address either accumulator A or accumulator B in the microprocessing unit. These operators are indicated by the column headed ACCX in Figure 4-1. This mode of addressing is specified by writing an operand field consisting only of the single character A or B (corresponding to accumulator A or accumulator B). For this type of addressing, it is then permissible to omit the SPACE between the operator and the operand field.

For this type of addressing, the assembly of a source instruction results in one byte of instruction in the machine language. For operators PUL and PSH, the accumulator mode is the only valid mode of addressing. The remaining eleven operators capable of this mode of addressing can alternatively be used with extended or indexed addressing.

4.3 INHERENT ADDRESSING

In many cases, the mnemonic operator itself specifies one or more registers which contain operands or in which results are saved. For example, the operator ABA requires two operands which

are located in accumulator A and accumulator B of the microprocessor. The operator also determines that the result of execution will be saved in accumulator A.

For some executable instructions, all of the information which may be required for the addressing is contained in the mnemonic operator, and no operand field is used in the source statement. There are 25 such instructions. These are indicated by the column headed Inherent in Figure 4-1.

Assembly of this type of source instruction results in only one byte of machine language code. Some other operators which contain addressing information inherently in the mnemonic code also require further addressing or operand information which is then placed in an operand field. Examples are the operators CPX, LDS, LDX, STS and STX.

4.4 IMMEDIATE ADDRESSING

The operators with which the immediate mode of addressing is permissible are indicated by the column headed Immediate in Figure 4-1. This mode of addressing is selected by beginning the specification of the corresponding operand (in the operand field of a source statement) with the pound character "#".

	(Dual Operand)	ACCX	Immediate	Direct	Extended	Indexed	Inherent	Relative
ABA		•	•	•	•	•	2	•
ADC	x	•	2	3	4	5	•	•
ADD	x	•	2	3	4	5	•	•
AND	x	•	2	3	4	5	•	•
ASL		2	•	•	6	7	•	•
ASR		2	•	•	6	7	•	•
BCC		•	•	•	•	•	•	4
BCS		•	•	•	•	•	•	4
BEA		•	•	•	•	•	•	4
BGE		•	•	•	•	•	•	4
BGT		•	•	•	•	•	•	4
BHI		•	•	•	•	•	•	4
BIT	x	•	2	3	4	5	•	•
BLE		•	•	•	•	•	•	4
BLS		•	•	•	•	•	•	4
BLT		•	•	•	•	•	•	4
BMI		•	•	•	•	•	•	4
BNE		•	•	•	•	•	•	4
BPL		•	•	•	•	•	•	4
BRA		•	•	•	•	•	•	4
BSR		•	•	•	•	•	•	8
BVC		•	•	•	•	•	•	4
BVS		•	•	•	•	•	•	4
CBA		•	•	•	•	•	2	•
CLC		•	•	•	•	•	2	•
CLI		•	•	•	•	•	2	•
CLR		2	•	•	6	7	•	•
CLV		•	•	•	•	•	2	•
CMP	x	•	2	3	4	5	•	•
COM		2	•	•	6	7	•	•
CPX		•	3	4	5	6	•	•
DAA		•	•	•	•	•	2	•
DEC		2	•	•	6	7	•	•
DES		•	•	•	•	•	4	•
DEX		•	•	•	•	•	4	•
EOR	x	•	2	3	4	5	•	•

	(Dual Operand)	ACCX	Immediate	Direct	Extended	Indexed	Inherent
INC		2	•	•	6	7	•
INS		•	•	•	•	•	4
INX		•	•	•	•	•	4
JMP		•	•	•	3	4	•
JSR		•	•	•	9	8	•
LDA	x	•	2	3	4	5	•
LDS		•	3	4	5	6	•
LDX		•	3	4	5	6	•
LSR		2	•	•	6	7	•
NEG		2	•	•	6	7	•
NOP		•	•	•	•	•	2
ORA	x	•	2	3	4	5	•
PSH		4	•	•	•	•	•
PUL		4	•	•	•	•	•
ROL		2	•	•	6	7	•
ROR		2	•	•	6	7	•
RTI		•	•	•	•	•	10
RTS		•	•	•	•	•	5
SBA		•	•	•	•	•	2
SBC	x	•	2	3	4	5	•
SEC		•	•	•	•	•	2
SEI		•	•	•	•	•	2
SEV		•	•	•	•	•	2
STA	x	•	•	4	5	6	•
STS		•	•	5	6	7	•
STX		•	•	5	6	7	•
SUB	x	•	2	3	4	5	•
SWI		•	•	•	•	•	12
TAB		•	•	•	•	•	2
TAP		•	•	•	•	•	2
TBA		•	•	•	•	•	2
TPA		•	•	•	•	•	2
TST		2	•	•	6	7	•
TSX		•	•	•	•	•	4
TXS		•	•	•	•	•	4
WAI		•	•	•	•	•	9

NOTE: Interrupt time is 12 cycles from the end of the instruction being executed, except following a WAI instruction. Then it is 4 cycles.

FIGURE 4-1. Instruction Addressing Modes and Execution Times (Times in Machine Cycles)

With the immediate mode of addressing, the operand field of the source statement either contains the actual value of the operand, or it includes a symbol or an expression which has an algebraic value equal to the value of the operand. The operand may be specified in accordance with any of the following formats:

Number
Symbol
Expression
'C

In the first three of these alternate forms, the assembler will find or compute a numerical value of the operand. For any executive instruction in the immediate mode of addressing (except CPX, LDS, or LDX), the numeric values must be an integer from 0 to 255 (decimal). For the operators CPX, LDS, or LDX, any value from 0 to 65535 (decimal) is valid.

In the last of the alternate forms (#'C), the apostrophe instructs the assembler to translate the next character into the corresponding 7-bit ASCII code. The ASCII code so obtained is then the value of the operand. The single character "C" can be any character of the ASCII character set with a hexadecimal value from 20 (SP) thru 5F (_____).

For the immediate mode of addressing, the assembler inserts the actual value of the operand into the machine code. Except for the three operators (CPX, LDS, and LDX), an instruction in the immediate mode is assembled into two bytes of machine code and the value of the operand is entered in the second byte. When it is a number, the operand is entered in the memory in unsigned 8-bit binary code. When it is an ASCII character, the corresponding 7-bit ASCII code applies, using bits 0-6 with bit 7 set to zero.

For the three operators (CPX, LDS, or LDX) used in the immediate mode, the source statement is assembled into three bytes of machine code. The numerical operand (which can have any value from 0 thru FFFF) will be entered in the second and third bytes. The second byte will contain the most significant part of the operand and the third byte will contain the least significant part of the operand. Both parts are entered into the respective bytes of the memory in unsigned 8-bit binary code.

The operators (CPX, LDS, or LDX) in the immediate mode are not normally used with an operand in the format #'C. However, in such a case, the assembler would place the ASCII coded character "C" in the third byte of the machine code corresponding to the source instruction.

When the immediate mode of addressing is used, the numerical address is in effect that of the second byte of machine code which results from assembly of the source instruction. Data flow for the immediate addressing mode is shown in Figure 4-2.

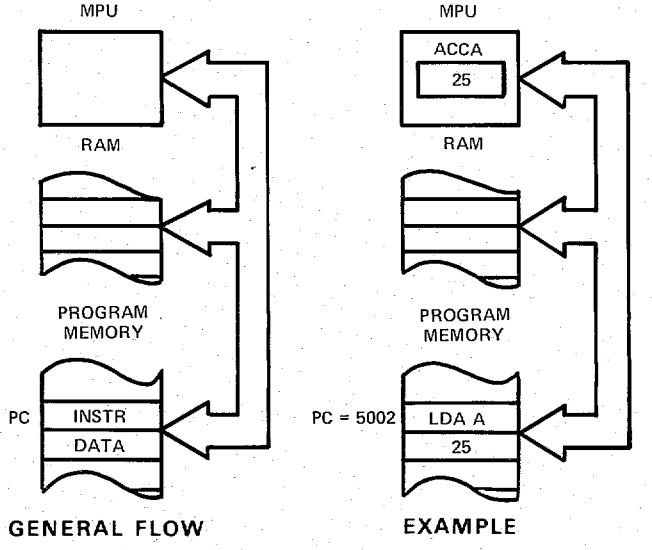


FIGURE 4-2. Immediate Addressing Mode Data Flow

4.5 RELATIVE ADDRESSING

For the relative addressing mode to be valid, there is a rule which limits the distance in the machine language program from the branch instruction to the destination of the branch. The rule which applies to the relative addressing mode is that the address of the destination of the branch must be within the range specified by:

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

where

PC = address of the first byte of the branch instruction.

D = address of the destination of the branch instruction.

When it is desired to transfer control beyond the range of the branch instructions, this can be done by using JMP (unconditional jump) or JSR (jump to subroutine). These instructions do not use the relative mode of addressing.

The assembler translates a branch instruction into two bytes of the machine code. The second byte contains a relative address. This is stored as a number in 8-bit, two's complement, binary form, with a decimal value in the range of −128 to +127. These numbers correspond to the limits of the range of a branch instruction, as described above.

The relationship between the relative address and the absolute address of the destination of a branch instruction is expressed by:

$$D = (PC + 2) + R$$

where:

PC = address of the first byte of the branch instruction

D = address of the destination of the branch instruction

R = the 8-bit, two's complement, binary number, stored in the second byte of the branch instruction.

The relative addressing mode is available only to the conditional branch instructions, the unconditional branch instruction (BRA), and the branch to subroutine (BSR). None of these source instructions can use any other of the several modes of addressing. The three-character mnemonic instruction, therefore, is sufficient to determine when the relative mode of addressing will be used for the assembler. An example of the data flow for the relative addressing mode is shown in Figure 4-3.

4.6 INDEXED ADDRESSING

The Indexed column of Figure 4-1 indicates the instructions for which indexed addressing is valid.

With this mode of addressing, the numerical address is variable; depending on the contents of the index register. The current address is obtained whenever it is required during the execution of a program, rather than being predetermined by the assembler as it is for the other addressing modes. The operand field of the source statement contains a numerical value which, when added to the contents of the index register during execution of the program, will provide the numerical address. Alternatively, the operand field may contain a symbol or an expression which the assembler is able to replace by the value which is to be added to the contents of the index register. An example of the indexed addressing mode is shown in Figure 4-4.

In indexed addressing, the data for obtaining the numerical address may be written in any of the formats:

X
,X
Number,X
Symbol,X
Expression,X

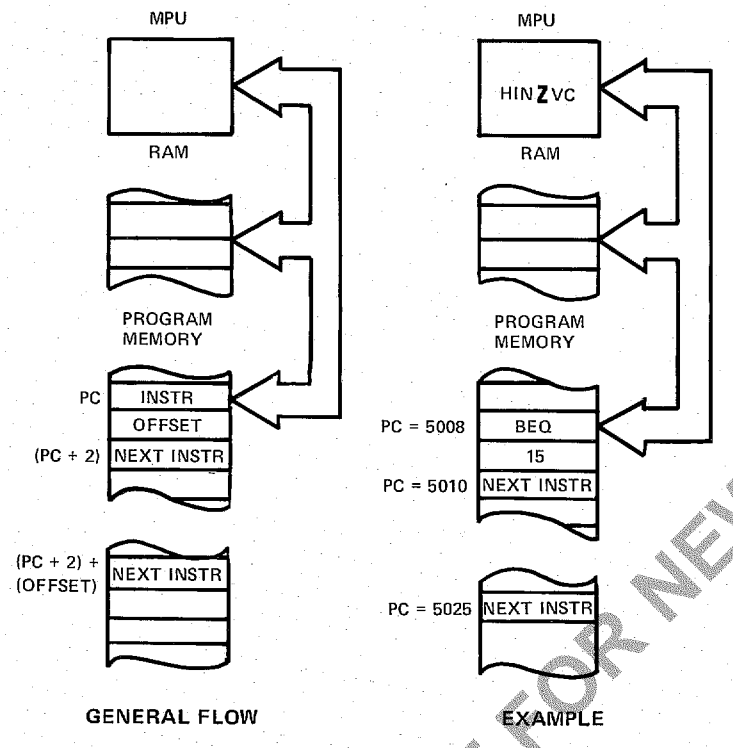


FIGURE 4-3. Relative Addressing Mode Data Flow

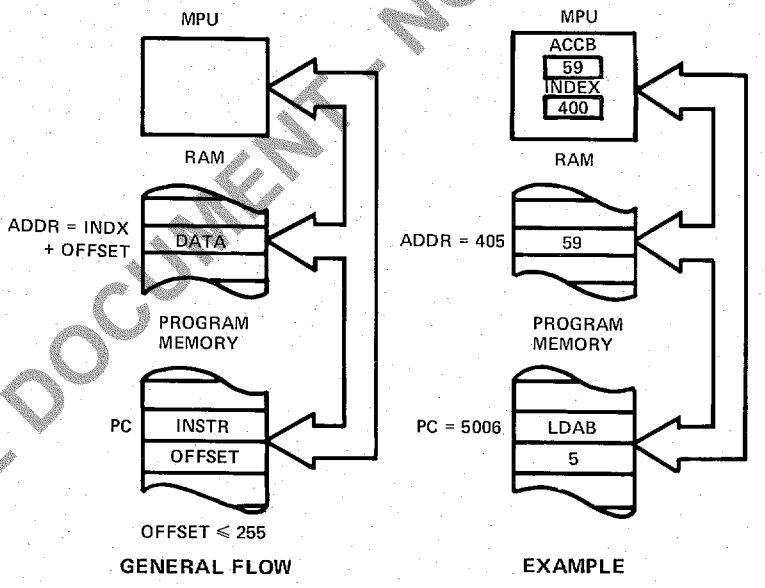


FIGURE 4-4. Indexed Addressing Mode

The single character X informs the assembler that the indexed mode is to be used (the character X being reserved to denote the index register).

The format X, when used alone, instructs the assembler that the address of the operand is identical with the contents of the index register. This format has the same effect on the assembly as if 0,X had been written.

If a symbol or an expression is used rather than a number, the assembler will find or compute a numerical value of that symbol or expression. The source program must then include other statements which define a numerical value for the symbol or which enable the assembler to compute a numerical value for the symbol or expression. Only values from zero to FF (hexadecimal) are valid.

This value is added to the contents of the index register during execution to obtain the numerical address as shown in the following formula:

D = numerical value + X

where

X = contents of index register

D = numerical address

For indexed addressing, the source instruction is translated into two bytes of the machine code. The second byte contains the number, in unsigned 8-bit binary form, which is added to the contents of the index register during execution of the instruction. The number thus obtained is the numerical address (in accordance with the previous formula).

4.7 DIRECT AND EXTENDED ADDRESSING

In direct addressing, the source instruction is translated into two bytes of machine code. The second byte will contain the address in unsigned 8-bit binary form.

In extended addressing, the source instruction is translated into three bytes of machine language. The second of these bytes will contain the highest 8 bits of the address. The third byte will contain the lowest 8 bits of the address. The contents of the second and third bytes will both be coded in unsigned 8-bit binary form.

For both direct and extended addressing, the address, which is placed by the assembler into the second or third bytes of the machine code, is the absolute numerical address.

As may be seen in Figure 4-1, there are several instructions for which the extended mode of addressing is valid and not the direct mode. For these instructions, when using any of the number, symbol, or expression formats, the assembler will select the extended mode of addressing, regardless of the value of the numerical address. The source statement will be translated into three bytes of the machine code.

For those instructions which may use the direct mode of addressing as well as the extended mode, the assembler selects the mode according to the following rule: The assembler will select direct addressing if the numerical address is in the range from 0 to 255 (decimal) and will select extended addressing if the numerical address exceeds 255 (decimal). Examples of the direct and extended addressing modes are shown in Figures 4-5 and 4-6.

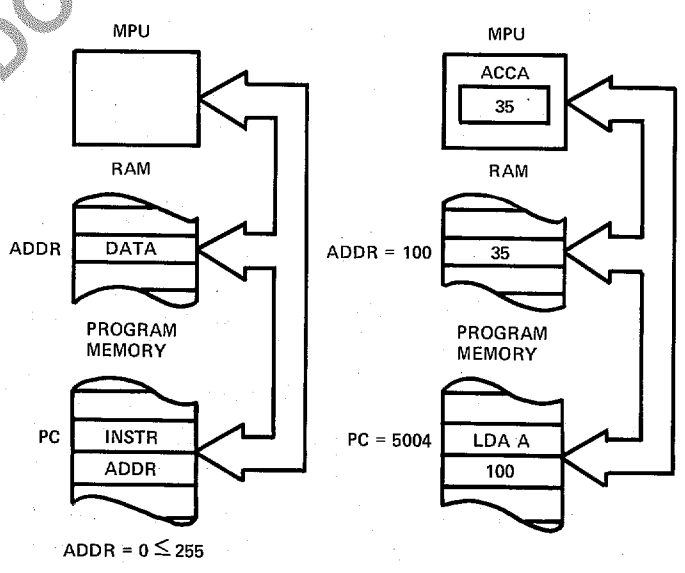


FIGURE 4-5. Direct Addressing Mode Data Flow

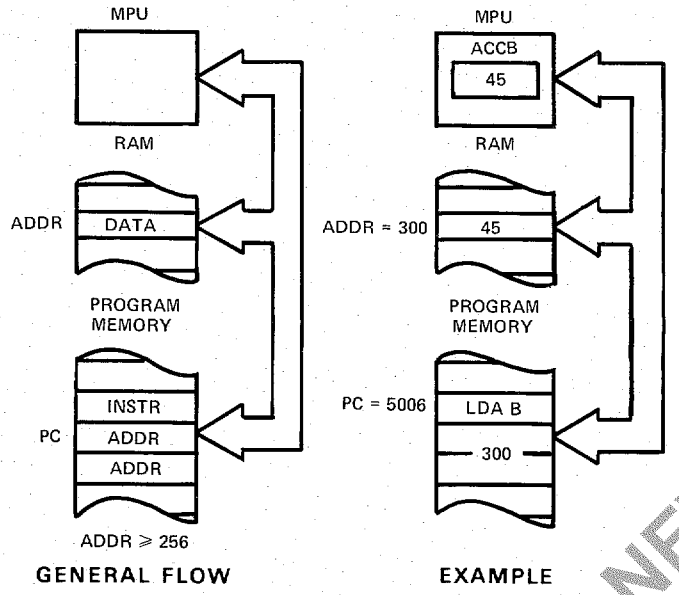


FIGURE 4-6. Extended Addressing Mode Data Flow

APPENDIX A
Definition of the Executable Instructions

A.1 Nomenclature

The following nomenclature is used in the subsequent definitions.

(a) *Operators*

( )	=	contents of
←	=	is transferred to
↑	=	"is pulled from stack"
↓	=	"is pushed into stack"
·	=	Boolean AND
⊙	=	Boolean (Inclusive) OR
⊕	=	Exclusive OR
≈	=	Boolean NOT

(b) *Registers in the MPU*

ACCA	=	Accumulator A
ACCB	=	Accumulator B
ACCX	=	Accumulator ACCA or ACCB
CC	=	Condition codes register
IX	=	Index register, 16 bits
IXH	=	Index register, higher order 8 bits
IXL	=	Index register, lower order 8 bits
PC	=	Program counter, 16 bits
PCH	=	Program counter, higher order 8 bits
PCL	=	Program counter, lower order 8 bits
SP	=	Stack pointer
SPH	=	Stack pointer high
SPL	=	Stack pointer low

(c) *Memory and Addressing*

M	=	A memory location (one byte)
M +1	=	The byte of memory at 0001 plus the address of the memory location indicated by "M."
Rel	=	Relative address (i.e. the two's complement number stored in the second byte of machine code corresponding to a branch instruction).

(d) *Bits 0 thru 5 of the Condition Codes Register*

C	=	Carry — borrow	bit — 0
V	=	Two's complement overflow indicator	bit — 1
Z	=	Zero indicator	bit — 2
N	=	Negative indicator	bit — 3
I	=	Interrupt mask	bit — 4
H	=	Half carry	bit — 5

(e) *Status of Individual Bits BEFORE Execution of an Instruction*

An	=	Bit n of ACCA (n=7,6,5,...,0)
Bn	=	Bit n of ACCB (n=7,6,5,...,0)
IXHn	=	Bit n of IXH (n=7,6,5,...,0)

IXLn = Bit n of IXL (n=7,6,5,...,0)
Mn = Bit n of M (n=7,6,5,...,0)
SPHn = Bit n of SPH (n=7,6,5,...,0)
SPLn = Bit n of SPL (n=7,6,5,...,0)
Xn = Bit n of ACCX (n=7,6,5,...,0)

(f) *Status of Individual Bits of the RESULT of Execution of an Instruction*

(i) For 8-bit Results

Rn = Bit n of the result (n =7,6,5,...,0)

This applies to instructions which provide a result contained in a single byte of memory or in an 8-bit register.

(ii) For 16-bit Results

RHn = Bit n of the more significant byte of the result (n =7,6,5,...,0)

RLn = Bit n of the less significant byte of the result (n =7,6,5,...,0)

This applies to instructions which provide a result contained in two consecutive bytes of memory or in a 16-bit register.

A.2 Executable Instructions (definition of)

Detailed definitions of the 72 executable instructions of the source language are provided on the following pages.

APPENDIX A
Definition of the Executable Instructions

A.1 Nomenclature

The following nomenclature is used in the subsequent definitions.

(a) *Operators*

( )	=	contents of
←	=	is transferred to
↑	=	"is pulled from stack"
↓	=	"is pushed into stack"
·	=	Boolean AND
⊙	=	Boolean (Inclusive) OR
⊕	=	Exclusive OR
≈	=	Boolean NOT

(b) *Registers in the MPU*

ACCA	=	Accumulator A
ACCB	=	Accumulator B
ACCX	=	Accumulator ACCA or ACCB
CC	=	Condition codes register
IX	=	Index register, 16 bits
IXH	=	Index register, higher order 8 bits
IXL	=	Index register, lower order 8 bits
PC	=	Program counter, 16 bits
PCH	=	Program counter, higher order 8 bits
PCL	=	Program counter, lower order 8 bits
SP	=	Stack pointer
SPH	=	Stack pointer high
SPL	=	Stack pointer low

(c) *Memory and Addressing*

M	=	A memory location (one byte)
M +1	=	The byte of memory at 0001 plus the address of the memory location indicated by "M."
Rel	=	Relative address (i.e. the two's complement number stored in the second byte of machine code corresponding to a branch instruction).

(d) *Bits 0 thru 5 of the Condition Codes Register*

C	=	Carry — borrow	bit — 0
V	=	Two's complement overflow indicator	bit — 1
Z	=	Zero indicator	bit — 2
N	=	Negative indicator	bit — 3
I	=	Interrupt mask	bit — 4
H	=	Half carry	bit — 5

(e) *Status of Individual Bits BEFORE Execution of an Instruction*

An	=	Bit n of ACCA (n=7,6,5,...,0)
Bn	=	Bit n of ACCB (n=7,6,5,...,0)
IXHn	=	Bit n of IXH (n=7,6,5,...,0)

IXLn = Bit n of IXL (n=7,6,5,...,0)
Mn = Bit n of M (n=7,6,5,...,0)
SPHn = Bit n of SPH (n=7,6,5,...,0)
SPLn = Bit n of SPL (n=7,6,5,...,0)
Xn = Bit n of ACCX (n=7,6,5,...,0)

(f) *Status of Individual Bits of the RESULT of Execution of an Instruction*

(i) For 8-bit Results

Rn = Bit n of the result (n =7,6,5,...,0)

This applies to instructions which provide a result contained in a single byte of memory or in an 8-bit register.

(ii) For 16-bit Results

RHn = Bit n of the more significant byte of the result (n =7,6,5,...,0)

RLn = Bit n of the less significant byte of the result (n =7,6,5,...,0)

This applies to instructions which provide a result contained in two consecutive bytes of memory or in a 16-bit register.

A.2 Executable Instructions (definition of)

Detailed definitions of the 72 executable instructions of the source language are provided on the following pages.

Add Accumulator B to Accumulator A **ABA**

Operation: ACCA ← (ACCA) + (ACCB)

Description: Adds the contents of ACCB to the contents of ACCA and places the result in ACCA.

Condition Codes:
- H: Set if there was a carry from bit 3; cleared otherwise.
- I: Not affected.
- N: Set if most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Set if there was two's complement overflow as a result of the operation; cleared otherwise.
- C: Set if there was a carry from the most significant bit of the result; cleared otherwise.

Boolean Formulae for Condition Codes:

$$H = A_3 \cdot B_3 + B_3 \cdot \overline{R}_3 + \overline{R}_3 \cdot A_3$$
$$N = R_7$$
$$Z = \overline{R}_7 \cdot \overline{R}_6 \cdot \overline{R}_5 \cdot \overline{R}_4 \cdot \overline{R}_3 \cdot \overline{R}_2 \cdot \overline{R}_1 \cdot \overline{R}_0$$
$$V = A_7 \cdot B_7 \cdot \overline{R}_7 + \overline{A}_7 \cdot \overline{B}_7 \cdot R_7$$
$$C = A_7 \cdot B_7 + B_7 \cdot \overline{R}_7 + \overline{R}_7 \cdot A_7$$

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
Inherent	2	1	1B	033	027

ADC

Add with Carry

Operation: ACCX ← (ACCX) + (M) + (C)

Description: Adds the contents of the C bit to the sum of the contents of ACCX and M, and places the result in ACCX.

Condition Codes:
- H: Set if there was a carry from bit 3; cleared otherwise.
- I: Not affected.
- N: Set if most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Set if there was two's complement overflow as a result of the operation; cleared otherwise.
- C: Set if there was a carry from the most significant bit of the result; cleared otherwise.

Boolean Formulae for Condition Codes:

$$H = X_3 \cdot M_3 + M_3 \cdot \overline{R}_3 + \overline{R}_3 \cdot X_3$$
$$N = R_7$$
$$Z = \overline{R}_7 \cdot \overline{R}_6 \cdot \overline{R}_5 \cdot \overline{R}_4 \cdot \overline{R}_3 \cdot \overline{R}_2 \cdot \overline{R}_1 \cdot \overline{R}_0$$
$$V = X_7 \cdot M_7 \cdot \overline{R}_7 + \overline{X}_7 \cdot \overline{M}_7 \cdot R_7$$
$$C = X_7 \cdot M_7 + M_7 \cdot \overline{R}_7 + \overline{R}_7 \cdot X_7$$

Addressing Formats:

See Table A-1

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

(DUAL OPERAND)

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
A IMM	2	2	89	211	137
A DIR	3	2	99	231	153
A EXT	4	3	B9	271	185
A IND	5	2	A9	251	169
B IMM	2	2	C9	311	201
B DIR	3	2	D9	331	217
B EXT	4	3	F9	371	249
B IND	5	2	E9	351	233

Add Without Carry **ADD**

Operation: ACCX ← (ACCX) + (M)

Description: Adds the contents of ACCX and the contents of M and places the result in ACCX.

Condition Codes:
- H: Set if there was a carry from bit 3; cleared otherwise.
- I: Not affected.
- N: Set if most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Set if there was two's complement overflow as a result of the operation; cleared otherwise.
- C: Set if there was a carry from the most significant bit of the result; cleared otherwise.

Boolean Formulae for Condition Codes:

$$H = X_3 \cdot M_3 + M_3 \cdot \overline{R}_3 + \overline{R}_3 \cdot X_3$$
$$N = R_7$$
$$Z = \overline{R}_7 \cdot \overline{R}_6 \cdot \overline{R}_5 \cdot \overline{R}_4 \cdot \overline{R}_3 \cdot \overline{R}_2 \cdot \overline{R}_1 \cdot \overline{R}_0$$
$$V = X_7 \cdot M_7 \cdot \overline{R}_7 + \overline{X}_7 \cdot \overline{M}_7 \cdot R_7$$
$$C = X_7 \cdot M_7 + M_7 \cdot \overline{R}_7 + \overline{R}_7 \cdot X_7$$

Addressing Formats:

See Table A-1

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

(DUAL OPERAND)

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
A IMM	2	2	8B	213	139
A DIR	3	2	9B	233	155
A EXT	4	3	BB	273	187
A IND	5	2	AB	253	171
B IMM	2	2	CB	313	203
B DIR	3	2	DB	333	219
B EXT	4	3	FB	373	251
B IND	5	2	EB	353	235

AND

Logical AND

Operation: ACCX ← (ACCX) · (M)

Description: Performs logical "AND" between the contents of ACCX and the contents of M and places the result in ACCX. (Each bit of ACCX after the operation will be the logical "AND" of the corresponding bits of M and of ACCX before the operation.)

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Cleared.
- C: Not affected.

Boolean Formulae for Condition Codes:

$N = R_7$

$Z = \overline{R}_7 \cdot \overline{R}_6 \cdot \overline{R}_5 \cdot \overline{R}_4 \cdot \overline{R}_3 \cdot \overline{R}_2 \cdot \overline{R}_1 \cdot \overline{R}_0$

$V = 0$

Addressing Formats:

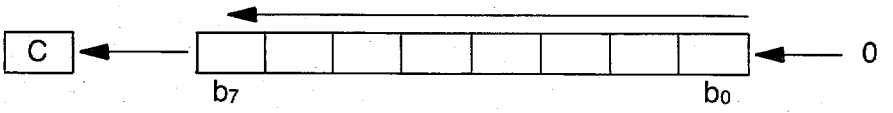
See Table A-1

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
A IMM	2	2	84	204	132
A DIR	3	2	94	224	148
A EXT	4	3	B4	264	180
A IND	5	2	A4	244	164
B IMM	2	2	C4	304	196
B DIR	3	2	D4	324	212
B EXT	4	3	F4	364	244
B IND	5	2	E4	344	228

Arithmetic Shift Left **ASL**

Operation:



Description: Shifts all bits of the ACCX or M one place to the left. Bit 0 is loaded with a zero. The C bit is loaded from the most significant bit of ACCX or M.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Set if, after the completion of the shift operation, EITHER (N is set and C is cleared) OR (N is cleared and C is set); cleared otherwise.
- C: Set if, before the operation, the most significant bit of the ACCX or M was set; cleared otherwise.

Boolean Formulae for Condition Codes:

$N = R_7$

$Z = \overline{R}_7 \cdot \overline{R}_6 \cdot \overline{R}_5 \cdot \overline{R}_4 \cdot \overline{R}_3 \cdot \overline{R}_2 \cdot \overline{R}_1 \cdot \overline{R}_0$

$V = N \oplus C = [N \cdot \overline{C}] \odot [\overline{N} \cdot C]$

(the foregoing formula assumes values of N and C after the shift operation)

$C = M_7$

Addressing Formats

See Table A-3

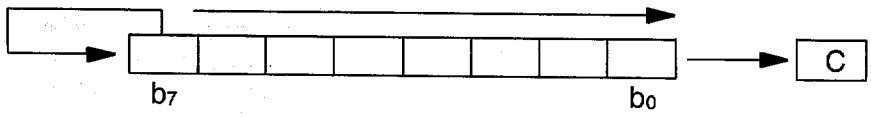
Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
A	2	1	48	110	072
B	2	1	58	130	088
EXT	6	3	78	170	120
IND	7	2	68	150	104

ASR

Arithmetic Shift Right

Operation:



Description: Shifts all bits of ACCX or M one place to the right. Bit 7 is held constant. Bit 0 is loaded into the C bit.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Set if, after the completion of the shift operation, EITHER (N is set and C is cleared) OR (N is cleared and C is set); cleared otherwise.
- C: Set if, before the operation, the least significant bit of the ACCX or M was set; cleared otherwise.

Boolean Formulae for Condition Codes:

$N = R_7$

$Z = \overline{R}_7 \cdot \overline{R}_6 \cdot \overline{R}_5 \cdot \overline{R}_4 \cdot \overline{R}_3 \cdot \overline{R}_2 \cdot \overline{R}_1 \cdot \overline{R}_0$

$V = N \oplus C = [N \cdot \overline{C}] \odot [\overline{N} \cdot C]$

(the foregoing formula assumes values of N and C after the shift operation)

$C = M_0$

Addressing Formats:

See Table A-3

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
A	2	1	47	107	071
B	2	1	57	127	087
EXT	6	3	77	167	119
IND	7	2	67	147	103

Branch if Carry Clear **BCC**

Operation: PC ← (PC) + 0002 + Rel if (C)=0

Description: Tests the state of the C bit and causes a branch if C is clear.

See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Addressing Formats:

See Table A-8.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
REL	4	2	24	044	036

BCS

Branch if Carry Set

Operation: PC ← (PC) + 0002 + Rel if (C)=1

Description: Tests the state of the C bit and causes a branch if C is set.
See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Addressing Formats:

See Table A-8.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
REL	4	2	25	045	037

Branch if Equal

BEQ

Operation: PC ← (PC) + 0002 + Rel if (Z)=1

Description: Tests the state of the Z bit and causes a branch if the Z bit is set.

See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Addressing Formats:

See Table A-8.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
REL	4	2	27	047	039

BGE

Branch if Greater than or Equal to Zero

Operation: $PC \leftarrow (PC) + 0002 + Rel$ if $(N) \oplus (V) = 0$

i.e. if $(ACCX) \geq (M)$

(Two's complement numbers)

Description: Causes a branch if (N is set and V is set) OR (N is clear and V is clear).

If the BGE instruction is executed immediately after execution of any of the instructions CBA, CMP, SBA, or SUB, the branch will occur if and only if the two's complement number represented by the minuend (i.e. ACCX) was greater than or equal to the two's complement number represented by the subtrahend (i.e. M).

See BRA instruction for details of the branch.

Condition Codes: Not affected.

Addressing Formats:

See Table A-8.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
REL	4	2	2C	054	044

Branch if Greater than Zero **BGT**

Operation: $PC \leftarrow (PC) + 0002 + Rel$ if $(Z) \odot [(N) \oplus (V)] = 0$

i.e. if (ACCX) > (M)
(two's complement numbers)

Description: Causes a branch if [Z is clear] AND [(N is set and V is set) OR (N is clear and V is clear)].

If the BGT instruction is executed immediately after execution of any of the instructions CBA, CMP, SBA, or SUB, the branch will occur if and only if the two's complement number represented by the minuend (i.e. ACCX) was greater than the two's complement number represented by the subtrahend (i.e. M).

See BRA instruction for details of the branch.

Condition Codes: Not affected.

Addressing Formats:

See Table A-8.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
REL	4	2	2E	056	046

BHI

Branch if Higher

Operation: PC ← (PC) + 0002 + Rel if (C) · (Z)=0

i.e. if (ACCX) > (M)
(unsigned binary numbers)

Description: Causes a branch if (C is clear) AND (Z is clear).

If the BHI instruction is executed immediately after execution of any of the instructions CBA, CMP, SBA, or SUB, the branch will occur if and only if the unsigned binary number represented by the minuend (i.e. ACCX) was greater than the unsigned binary number represented by the subtrahend (i.e. M).

See BRA instruction for details of the execution of the branch.

Condition Codes: Not affected.

Addressing Formats:

See Table A-8.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
REL	4	2	22	042	034

Bit Test **BIT**

Operation: (ACCX) · (M)

Description: Performs the logical "AND" comparison of the contents of ACCX and the contents of M and modifies condition codes accordingly. Neither the contents of ACCX or M operands are affected. (Each bit of the result of the "AND" would be the logical "AND" of the corresponding bits of M and ACCX.)

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result of the "AND" would be set; cleared otherwise.
- Z: Set if all bits of the result of the "AND" would be cleared; cleared otherwise.
- V: Cleared.
- C: Not affected.

Boolean Formulae for Condition Codes:

$N = R_7$

$Z = \overline{R}_7 \cdot \overline{R}_6 \cdot \overline{R}_5 \cdot \overline{R}_4 \cdot \overline{R}_3 \cdot \overline{R}_2 \cdot \overline{R}_1 \cdot \overline{R}_0$

$V = 0$

Addressing Formats:

See Table A-1.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
A IMM	2	2	85	205	133
A DIR	3	2	95	225	149
A EXT	4	3	B5	265	181
A IND	5	2	A5	245	165
B IMM	2	2	C5	305	197
B DIR	3	2	D5	325	213
B EXT	4	3	F5	365	245
B IND	5	2	E5	345	229

BLE

Branch if Less than or Equal to Zero

Operation: $PC \leftarrow (PC) + 0002 + Rel$ if $(Z) \odot [(N) \oplus (V)] = 1$

i.e. if (ACCX) ≤ (M)
(two's complement numbers)

Description: Causes a branch if [Z is set] OR [(N is set and V is clear) OR (N is clear and V is set)].

If the BLE instruction is executed immediately after execution of any of the instructions CBA, CMP, SBA, or SUB, the branch will occur if and only if the two's complement number represented by the minuend (i.e. ACCX) was less then or equal to the two's complement number represented by the subtrahend (i.e. M).

See BRA instruction for details of the branch.

Condition Codes: Not affected.

Addressing Formats:

See Table A-8.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
REL	4	2	2F	057	047

Branch if Lower or Same **BLS**

Operation: PC ← (PC) + 0002 + Rel if (C)⊙(Z) = 1
i.e. if (ACCX) ≤ (M)
(unsigned binary numbers)

Description: Causes a branch if (C is set) OR (Z is set).

If the BLS instruction is executed immediately after execution of any of the instructions CBA, CMP, SBA, or SUB, the branch will occur if and only if the unsigned binary number represented by the minuend (i.e. ACCX) was less than or equal to the unsigned binary number represented by the subtrahend (i.e. M).

See BRA instruction for details of the execution of the branch.

Condition Codes: Not affected.

Addressing Formats:

See Table A-8.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
REL	4	2	23	043	035

BLT

Branch if Less than Zero

Operation: $PC \leftarrow (PC) + 0002 + Rel$ if $(N) \oplus (V) = 1$

i.e. if (ACCX) < (M)
(two's complement numbers)

Description: Causes a branch if (N is set and V is clear) OR (N is clear and V is set).

If the BLT instruction is executed immediately after execution of any of the instructions CBA, CMP, SBA, or SUB, the branch will occur if and only if the two's complement number represented by the minuend (i.e. ACCX) was less than the two's complement number represented by the subtrahend (i.e. M).

See BRA instruction for details of the branch.

Condition Codes: Not affected.

Addressing Formats:

See Table A-8.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
REL	4	2	2D	055	045

Branch if Minus **BMI**

Operation: PC← (PC) + 0002 + Rel if (N) =1

Description: Tests the state of the N bit and causes a branch if N is set.
See BRA instruction for details of the execution of the branch.

Condition Codes: Not affected.

Addressing Formats:

See Table A-8.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
REL	4	2	2B	053	043

BNE

Branch if Not Equal

Operation: PC ← (PC) + 0002 + Rel if (Z) = 0

Description: Tests the state of the Z bit and causes a branch if the Z bit is clear.
See BRA instruction for details of the execution of the branch.

Condition Codes: Not affected.

Addressing Formats:

See Table A-8.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
REL	4	2	26	046	038

Branch if Plus **BPL**

Operation: PC ← (PC) + 0002 + Rel if (N) =0

Description: Tests the state of the N bit and causes a branch if N is clear.
See BRA instruction for details of the execution of the branch.

Condition Codes: Not affected.

Addressing Formats:

See Table A-8.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
REL	4	2	2A	052	042

BRA

Branch Always

Operation: PC ← (PC) + 0002 + Rel

Description: Unconditional branch to the address given by the foregoing formula, in which R is the relative address stored as a two's complement number in the second byte of machine code corresponding to the branch instruction.

Note: The source program specifies the destination of any branch instruction by its absolute address, either as a numerical value or as a symbol or expression which can be numerically evaluated by the assembler. The assembler obtains the relative address R from the absolute address and the current value of the program counter PC.

Condition Codes: Not affected.

Addressing Formats:

See Table A-8.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
REL	4	2	20	040	032

Branch to Subroutine **BSR**

Operation: PC ← (PC) + 0002
↓ (PCL)
SP ← (SP) − 0001
↓ (PCH)
SP ← (SP) − 0001
PC ← (PC) + Rel

Description: The program counter is incremented by 2. The less significant byte of the contents of the program counter is pushed into the stack. The stack pointer is then decremented (by 1). The more significant byte of the contents of the program counter is then pushed into the stack. The stack pointer is again decremented (by 1). A branch then occurs to the location specified by the program.

See BRA instruction for details of the execution of the branch.

Condition Codes: Not affected.

Addressing Formats:

See Table A-8.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
REL	8	2	8D	215	141

BRANCH TO SUBROUTINE EXAMPLE

			Memory Location	Machine Code (Hex)	Label	Assembler Language Operator	Operand
A.	*Before*						
	PC	←	$1000	8D		BSR	CHARLI
			$1001	50			
	SP	←	$EFFF				
B.	*After*						
	PC	←	$1052	**	CHARLI	***	*****
	SP	←	$EFFD				
			$EFFE	10			
			$EFFF	02			

BVC

Branch if Overflow Clear

Operation: PC ← (PC) + 0002 + Rel if (V) = 0

Description: Tests the state of the V bit and causes a branch if the V bit is clear. See BRA instruction for details of the execution of the branch.

Condition Codes: Not affected.

Addressing Formats:

See Table A-8.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
REL	4	2	28	050	040

Branch if Overflow Set **BVS**

Operation: PC ← (PC) + 0002 + Rel if (V) =1

Description: Tests the state of the V bit and causes a branch if the V bit is set.

See BRA instruction for details of the execution of the branch.

Condition Codes: Not affected.

Addressing Formats:

See Table A-8.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
REL	4	2	29	051	041

CBA

Compare Accumulators

Operation: (ACCA) − (ACCB)

Description: Compares the contents of ACCA and the contents of ACCB and sets the condition codes, which may be used for arithmetic and logical conditional branches. Both operands are unaffected.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result of the subtraction would be set; cleared otherwise.
- Z: Set if all bits of the result of the subtraction would be cleared; cleared otherwise.
- V: Set if the subtraction would cause two's complement overflow; cleared otherwise.
- C: Set if the subtraction would require a borrow into the most significant bit of the result; clear otherwise.

Boolean Formulae for Condition Codes:

$$N = R_7$$
$$Z = \overline{R}_7 \cdot \overline{R}_6 \cdot \overline{R}_5 \cdot \overline{R}_4 \cdot \overline{R}_3 \cdot \overline{R}_2 \cdot \overline{R}_1 \cdot \overline{R}_0$$
$$V = A_7 \cdot \overline{B}_7 \cdot \overline{R}_7 + \overline{A}_7 \cdot B_7 \cdot R_7$$
$$C = \overline{A}_7 \cdot B_7 + B_7 \cdot R_7 + R_7 \cdot \overline{A}_7$$

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
INHERENT	2	1	11	021	017

Clear Carry **CLC**

Operation: C bit ← 0

Description: Clears the carry bit in the processor condition codes register.

Condition Codes:
H: Not affected.
I: Not affected.
N: Not affected.
Z: Not affected.
V: Not affected.
C: Cleared

Boolean Formulae for Condition Codes:
C = 0

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
INHERENT	2	1	0C	014	012

CLI

Clear Interrupt Mask

Operation: I bit ← 0

Description: Clears the interrupt mask bit in the processor condition codes register. This enables the microprocessor to service an interrupt from a peripheral device if signalled by a high state of the "Interrupt Request" control input.

Condition Codes:
H: Not affected.
I: Cleared.
N: Not affected.
Z: Not affected.
V: Not affected.
C: Not affected.

Boolean Formulae for Condition Codes:
I = 0

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
INHERENT	2	1	0E	016	014

Clear **CLR**

Operation: ACCX ← 00
or: M ← 00

Description: The contents of ACCX or M are replaced with zeros.

Condition Codes:
H: Not affected.
I: Not affected.
N: Cleared
Z: Set
V: Cleared
C: Cleared

Boolean Formulae for Condition Codes:

N = 0
Z = 1
V = 0
C = 0

Addressing Formats:

See Table A-3.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
A	2	1	4F	117	079
B	2	1	5F	137	095
EXT	6	3	7F	177	127
IND	7	2	6F	157	111

CLV

Clear Two's Complement Overflow Bit

Operation: V bit ← 0

Description: Clears the two's complement overflow bit in the processor condition codes register.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Not affected.
- Z: Not affected.
- V: Cleared.
- C: Not affected.

Boolean Formulae for Condition Codes:

V = 0

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
INHERENT	2	1	0A	012	010

Compare **CMP**

Operation: (ACCX) − (M)

Description: Compares the contents of ACCX and the contents of M and determines the condition codes, which may be used subsequently for controlling conditional branching. Both operands are unaffected.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result of the subtraction would be set; cleared otherwise.
- Z: Set if all bits of the result of the subtraction would be cleared; cleared otherwise.
- V: Set if the subtraction would cause two's complement overflow; cleared otherwise.
- C: Carry is set if the absolute value of the contents of memory is larger than the absolute value of the accumulator; reset otherwise.

Boolean Formulae for Condition Codes:

$$N = R_7$$
$$Z = \overline{R}_7 \cdot \overline{R}_6 \cdot \overline{R}_5 \cdot \overline{R}_4 \cdot \overline{R}_3 \cdot \overline{R}_2 \cdot \overline{R}_1 \cdot \overline{R}_0$$
$$V = X_7 \cdot \overline{M}_7 \cdot \overline{R}_7 + \overline{X}_7 \cdot M_7 \cdot R_7$$
$$C = \overline{X}_7 \cdot M_7 + M_7 \cdot R_7 + R_7 \cdot \overline{X}_7$$

Addressing Formats:

See Table A-1.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

(DUAL OPERAND)

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
A IMM	2	2	81	201	129
A DIR	3	2	91	221	145
A EXT	4	3	B1	261	177
A IND	5	2	A1	241	161
B IMM	2	2	C1	301	193
B DIR	3	2	D1	321	209
B EXT	4	3	F1	361	241
B IND	5	2	E1	341	225

COM

Complement

Operation: $ACCX \leftarrow \approx (ACCX) = FF - (ACCX)$
or: $M \leftarrow \approx (M) = FF - (M)$

Description: Replaces the contents of ACCX or M with its one's complement. (Each bit of the contents of ACCX or M is replaced with the complement of that bit.)

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Cleared.
- C: Set.

Boolean Formulae for Condition Codes:

$N = R_7$
$Z = \overline{R}_7 \cdot \overline{R}_6 \cdot \overline{R}_5 \cdot \overline{R}_4 \cdot \overline{R}_3 \cdot \overline{R}_2 \cdot \overline{R}_1 \cdot \overline{R}_0$
$V = 0$
$C = 1$

Addressing Formats:

See Table A-3.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
A	2	1	43	103	067
B	2	1	53	123	083
EXT	6	3	73	163	115
IND	7	2	63	143	099

Compare Index Register

CPX

Operation: (IXL) − (M+1)
(IXH) − (M)

Description: The more significant byte of the contents of the index register is compared with the contents of the byte of memory at the address specified by the program. The less significant byte of the contents of the index register is compared with the contents of the next byte of memory, at one plus the address specified by the program. The Z bit is set or reset according to the results of these comparisons, and may be used subsequently for conditional branching.

The N and V bits, though determined by this operation, are not intended for conditional branching.

The C bit is not affected by this operation.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result of the subtraction from the more significant byte of the index register would be set; cleared otherwise.
- Z: Set if all bits of the results of both subtractions would be cleared; cleared otherwise.
- V: Set if the subtraction from the more significant byte of the index register would cause two's complement overflow; cleared otherwise.
- C: Not affected.

Boolean Formulae for Condition Codes:

$$N = RH_7$$

$$Z = (\overline{RH_7} \cdot \overline{RH_6} \cdot \overline{RH_5} \cdot \overline{RH_4} \cdot \overline{RH_3} \cdot \overline{RH_2} \cdot \overline{RH_1} \cdot \overline{RH_0}) \cdot (\overline{RL_7} \cdot \overline{RL_6} \cdot \overline{RL_5} \cdot \overline{RL_4} \cdot \overline{RL_3} \cdot \overline{RL_2} \cdot \overline{RL_1} \cdot \overline{RL_0})$$

$$V = IXH_7 \cdot \overline{M_7} \cdot \overline{RH_7} + \overline{IXH_7} \cdot M_7 \cdot RH_7$$

Addressing Formats:

See Table A-5.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
IMM	3	3	8C	214	140
DIR	4	2	9C	234	156
EXT	5	3	BC	274	188
IND	6	2	AC	254	172

DAA

Decimal Adjust ACCA

Operation: Adds hexadecimal numbers 00, 06, 60, or 66 to ACCA, and may also set the carry bit, as indicated in the following table:

State of C-bit before DAA (Col. 1)	Upper Half-byte (bits 4-7) (Col. 2)	Initial Half-carry H-bit (Col.3)	Lower to ACCA (bits 0-3) (Col. 4)	Number Added after by DAA (Col. 5)	State of C-bit DAA (Col. 6)
0	0-9	0	0-9	00	0
0	0-8	0	A-F	06	0
0	0-9	1	0-3	06	0
0	A-F	0	0-9	60	1
0	9-F	0	A-F	66	1
0	A-F	1	0-3	66	1
1	0-2	0	0-9	60	1
1	0-2	0	A-F	66	1
1	0-3	1	0-3	66	1

Note: Columns (1) through (4) of the above table represent all possible cases which can result from any of the operations ABA, ADD, or ADC, with initial carry either set or clear, applied to two binary-coded-decimal operands. The table shows hexadecimal values.

Description: If the contents of ACCA and the state of the carry-borrow bit C and the half-carry bit H are all the result of applying any of the operations ABA, ADD, or ADC to binary-coded-decimal operands, with or without an initial carry, the DAA operation will function as follows.

Subject to the above condition, the DAA operation will adjust the contents of ACCA and the C bit to represent the correct binary-coded-decimal sum and the correct state of the carry.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Not defined.
- C: Set or reset according to the same rule as if the DAA and an immediately preceding ABA, ADD, or ADC were replaced by a hypothetical binary-coded-decimal addition.

Boolean Formulae for Condition Codes:

$N = R_7$

$Z = \overline{R}_7 \cdot \overline{R}_6 \cdot \overline{R}_5 \cdot \overline{R}_4 \cdot \overline{R}_3 \cdot \overline{R}_2 \cdot \overline{R}_1 \cdot \overline{R}_0$

C = See table above.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
INHERENT	2	1	19	031	025

DEC

Decrement

Operation: ACCX ← (ACCX) − 01
or: M ← (M) − 01

Description: Subtract one from the contents of ACCX or M.

The N, Z, and V condition codes are set or reset according to the results of this operation.

The C bit is not affected by the operation.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Set if there was two's complement overflow as a result of the operation; cleared otherwise. Two's complement overflow occurs if and only if (ACCX) or (M) was 80 before the operation.
- C: Not affected.

Boolean Formulae for Condition Codes:

$$N = R_7$$
$$Z = \overline{R}_7 \cdot \overline{R}_6 \cdot \overline{R}_5 \cdot \overline{R}_5 \cdot \overline{R}_4 \cdot \overline{R}_3 \cdot \overline{R}_2 \cdot \overline{R}_1 \cdot \overline{R}_0$$
$$V = X_7 \cdot \overline{X}_6 \cdot \overline{X}_5 \cdot \overline{X}_4 \cdot \overline{X}_3 \cdot \overline{X}_2 \cdot \overline{X}_0 = \overline{R}_7 \cdot R_6 \cdot R_5 \cdot R_4 \cdot R_3 \cdot R_2 \cdot R_1 \cdot R_0$$

Addressing Formats:

See Table A-3.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
A	2	1	4A	112	074
B	2	1	5A	132	090
EXT	6	3	7A	172	122
IND	7	2	6A	152	106

Decrement Stack Pointer **DES**

Operation: SP ← (SP) − 0001

Description: Subtract one from the stack pointer.

Condition Codes: Not affected.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
INHERENT	4	1	34	064	052

DEX

Decrement Index Register

Operation: $IX \leftarrow (IX) - 0001$

Description: Subtract one from the index register.

Only the Z bit is set or reset according to the result of this operation.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Not affected.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Not affected.
- C: Not affected.

Boolean Formulae for Condition Codes:

$$Z = (\overline{RH_7} \cdot \overline{RH_6} \cdot \overline{RH_5} \cdot \overline{RH_4} \cdot \overline{RH_3} \cdot \overline{RH_2} \cdot \overline{RH_1} \cdot \overline{RH_0}) \cdot (\overline{RL_7} \cdot \overline{RL_6} \cdot \overline{RL_5} \cdot \overline{RL_4} \cdot \overline{RL_3} \cdot \overline{RL_2} \cdot \overline{RL_1} \cdot \overline{RL_0})$$

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
INHERENT	4	1	09	011	009

Exclusive OR **EOR**

Operation: ACCX ← (ACCX) ⊕ (M)

Description: Perform logical "EXCLUSIVE OR" between the contents of ACCX and the contents of M, and place the result in ACCX. (Each bit of ACCX after the operation will be the logical "EXCLUSIVE OR" of the corresponding bit of M and ACCX before the operation.)

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Cleared
- C: Not affected.

Boolean Formulae for Condition Codes:

$N = R_7$

$Z = \overline{R}_7 \cdot \overline{R}_6 \cdot \overline{R}_5 \cdot \overline{R}_4 \cdot \overline{R}_3 \cdot \overline{R}_2 \cdot \overline{R}_1 \cdot \overline{R}_0$

$V = 0$

Addressing Formats:

See Table A-1.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
A IMM	2	2	88	210	136
A DIR	3	2	98	230	152
A EXT	4	3	B8	270	184
A IND	5	2	A8	250	168
B IMM	2	2	C8	310	200
B DIR	3	2	D8	330	216
B EXT	4	3	F8	370	248
B IND	5	2	E8	350	232

INC

Increment

Operation: ACCX ← (ACCX) + 01
or: M ← (M) + 01

Description: Add one to the contents of ACCX or M.

The N, Z, and V condition codes are set or reset according to the results of this operation.

The C bit is not affected by the operation.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Set if there was two's complement overflow as a result of the operation; cleared otherwise. Two's complement overflow will occur if and only if (ACCX) or (M) was 7F before the operation.
- C: Not affected.

Boolean Formulae for Condition Codes:

$N = R_7$

$Z = \overline{R}_7 \cdot \overline{R}_6 \cdot \overline{R}_5 \cdot \overline{R}_4 \cdot \overline{R}_3 \cdot \overline{R}_2 \cdot \overline{R}_1 \cdot \overline{R}_0$

$V = \overline{X}_7 \cdot X_6 \cdot X_5 \cdot X_4 \cdot X_3 \cdot X_2 \cdot X_1 \cdot X_0$

$C = \overline{R}_7 \cdot \overline{R}_6 \cdot \overline{R}_5 \cdot \overline{R}_4 \cdot \overline{R}_3 \cdot \overline{R}_2 \cdot \overline{R}_1 \cdot \overline{R}_0$

Addressing Formats:

See Table A-3.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
A	2	1	4C	114	076
B	2	1	5C	134	092
EXT	6	3	7C	174	124
IND	7	2	6C	154	108

Increment Stack Pointer

Operation: SP ← (SP) + 0001

Description: Add one to the stack pointer.

Condition Codes: Not affected.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
INHERENT	4	1	31	061	049

INX

Increment Index Register

Operation: $IX \leftarrow (IX) + 0001$

Description: Add one to the index register.

Only the Z bit is set or reset according to the result of this operation.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Not affected.
- Z: Set if all 16 bits of the result are cleared; cleared otherwise.
- V: Not affected.
- C: Not affected.

Boolean Formulae for Condition Codes:

$$Z = (\overline{RH_7} \cdot \overline{RH_6} \cdot \overline{RH_5} \cdot \overline{RH_4} \cdot \overline{RH_3} \cdot \overline{RH_2} \cdot \overline{RH_1} \cdot \overline{RH_0}) \cdot (\overline{RL_7} \cdot \overline{RL_6} \cdot \overline{RL_5} \cdot \overline{RL_4} \cdot \overline{RL_3} \cdot \overline{RL_2} \cdot \overline{RL_1} \cdot \overline{RL_0})$$

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
INHERENT	4	1	08	010	008

Jump **JMP**

Operation: PC ← numerical address

Description: A jump occurs to the instruction stored at the numerical address. The numerical address is obtained according to the rules for EXTended or INDexed addressing.

Condition Codes: Not affected.

Addressing Formats:

See Table A-7.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal).

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
EXT	3	3	7E	176	126
IND	4	2	6E	156	110

JSR

Jump to Subroutine

Operation:

Either: PC ← (PC) + 0003 (for EXTended addressing)
or: PC ← (PC) + 0002 (for INDexed addressing)
Then: ↓ (PCL)
SP ← (SP) − 0001
↓ (PCH)
SP ← (SP) − 0001
PC ← numerical address

Description: The program counter is incremented by 3 or by 2, depending on the addressing mode, and is then pushed onto the stack, eight bits at a time. The stack pointer points to the next empty location in the stack. A jump occurs to the instruction stored at the numerical address. The numerical address is obtained according to the rules for EXTended or INDexed addressing.

Condition Codes: Not affected.

Addressing Formats:

See Table A-7.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
EXT	9	3	BD	275	189
IND	8	2	AD	255	173

JUMP TO SUBROUTINE EXAMPLE (extended mode)

			Memory Location	Machine Code (Hex)	Assembler Language Label	Operator	Operand
A.	*Before:*						
	PC	→	$0FFF	BD		JSR	CHARLI
			$1000	20			
			$1001	77			
	SP	←	$EFFF				
B.	*After:*						
	PC	→	$2077	**	CHARLI	***	*****
	SP	→	$EFFD				
			$EFFE	10			
			$EFFF	02			

Load Accumulator **LDA**

Operation: ACCX ← (M)

Description: Loads the contents of memory into the accumulator. The condition codes are set according to the data.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Cleared.
- C: Not affected.

Boolean Formulae for Condition Codes:

$N = R_7$

$Z = \overline{R}_7 \cdot \overline{R}_6 \cdot \overline{R}_5 \cdot \overline{R}_4 \cdot \overline{R}_3 \cdot \overline{R}_2 \cdot \overline{R}_1 \cdot \overline{R}_0$

$V = 0$

Addressing Formats:

See Table A-1.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

(DUAL OPERAND)

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
A IMM	2	2	86	206	134
A DIR	3	2	96	226	150
A EXT	4	3	B6	266	182
A IND	5	2	A6	246	166
B IMM	2	2	C6	306	198
B DIR	3	2	D6	326	214
B EXT	4	3	F6	366	246
B IND	5	2	E6	346	230

LDS

Load Stack Pointer

Operation: SPH ← (M)
SPL ← (M+1)

Description: Loads the more significant byte of the stack pointer from the byte of memory at the address specified by the program, and loads the less significant byte of the stack pointer from the next byte of memory, at one plus the address specified by the program.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the stack pointer is set by the operation; cleared otherwise.
- Z: Set if all bits of the stack pointer are cleared by the operation; cleared otherwise.
- V: Cleared.
- C: Not affected.

Boolean Formulae for Condition Codes:

$N = RH_7$

$Z = (\overline{RH_7} \cdot \overline{RH_6} \cdot \overline{RH_5} \cdot \overline{RH_4} \cdot \overline{RH_3} \cdot \overline{RH_2} \cdot \overline{RH_1} \cdot \overline{RH_0}) \cdot (\overline{RL_7} \cdot \overline{RL_6} \cdot \overline{RL_5} \cdot \overline{RL_4} \cdot \overline{RL_3} \cdot \overline{RL_2} \cdot \overline{RL_1} \cdot \overline{RL_0})$

$V = 0$

Addressing Formats:

See Table A-5.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
IMM	3	3	8E	216	142
DIR	4	2	9E	236	158
EXT	5	3	BE	276	190
IND	6	2	AE	256	174

Load Index Register **LDX**

Operation: IXH ← (M)
IXL ← (M+1)

Description: Loads the more significant byte of the index register from the byte of memory at the address specified by the program, and loads the less significant byte of the index register from the next byte of memory, at one plus the address specified by the program.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the index register is set by the operation; cleared otherwise.
- Z: Set if all bits of the index register are cleared by the operation; cleared otherwise.
- V: Cleared.
- C: Not affected.

Boolean Formulae for Condition Codes:

$$N = RH_7$$

$$Z = (\overline{RH_7} \cdot \overline{RH_6} \cdot \overline{RH_5} \cdot \overline{RH_4} \cdot \overline{RH_3} \cdot \overline{RH_2} \cdot \overline{RH_1} \cdot \overline{RH_0}) \cdot (\overline{RL_7} \cdot \overline{RL_6} \cdot \overline{RL_5} \cdot \overline{RL_4} \cdot \overline{RL_3} \cdot \overline{RL_2} \cdot \overline{RL_1} \cdot \overline{RL_0})$$

$$V = 0$$

Addressing Formats:

See Table A-5.

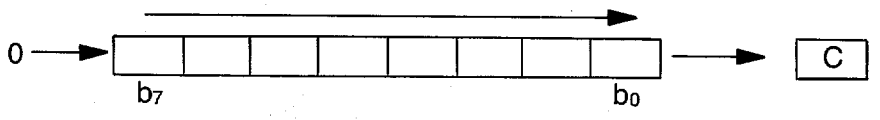
Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
IMM	3	3	CE	316	206
DIR	4	2	DE	336	222
EXT	5	3	FE	376	254
IND	6	2	EE	356	238

LSR

Logical Shift Right

Operation:



Description: Shifts all bits of ACCX or M one place to the right. Bit 7 is loaded with a zero. The C bit is loaded from the least significant bit of ACCX or M.

Condition Codes:

H: Not affected.
I: Not affected.
N: Cleared.
Z: Set if all bits of the result are cleared; cleared otherwise.
V: Set if, after the completion of the shift operation, EITHER (N is set and C is cleared) OR (N is cleared and C is set); cleared otherwise.
C: Set if, before the operation, the least significant bit of the ACCX or M was set; cleared otherwise.

Boolean Formulae for Condition Codes:

$N = 0$

$Z = \overline{R_7} \cdot \overline{R_6} \cdot \overline{R_5} \cdot \overline{R_4} \cdot \overline{R_3} \cdot \overline{R_2} \cdot \overline{R_1} \cdot \overline{R_0}$

$V = N \oplus C = [N \cdot \overline{C}] \odot [\overline{N} \cdot C]$

(the foregoing formula assumes values of N and C after the shift operation).

$C = M_0$

Addressing Formats:

See Table A-3.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
A	2	1	44	104	068
B	2	1	54	124	084
EXT	6	3	74	164	116
IND	7	2	64	144	100

Negate **NEG**

Operation: ACCX ← − (ACCX) = 00 − (ACCX)
or: M ← − (M) = 00 − (M)

Description: Replaces the contents of ACCX or M with its two's complement. Note that 80 is left unchanged.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Set if there would be two's complement overflow as a result of the implied subtraction from zero; this will occur if and only if the contents of ACCX or M is 80.
- C: Set if there would be a borrow in the implied subtraction from zero; the C bit will be set in all cases except when the contents of ACCX or M is 00.

Boolean Formulae for Condition Codes:

$$N = R_7$$
$$Z = \overline{R}_7 \cdot \overline{R}_6 \cdot \overline{R}_5 \cdot \overline{R}_4 \cdot \overline{R}_3 \cdot \overline{R}_2 \cdot \overline{R}_1 \cdot \overline{R}_0$$
$$V = R_7 \cdot \overline{R}_6 \cdot \overline{R}_5 \cdot \overline{R}_4 \cdot \overline{R}_3 \cdot \overline{R}_2 \cdot \overline{R}_1 \cdot \overline{R}_0$$
$$C = R_7 + R_6 + R_5 + R_4 + R_3 + R_2 + R_1 + R_0$$

Addressing Formats:

See Table A-3.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
A	2	1	40	100	064
B	2	1	50	120	080
EXT	6	3	70	160	112
IND	7	2	60	140	096

NOP

No Operation

Description: This is a single-word instruction which causes only the program counter to be incremented. No other registers are affected.

Condition Codes: Not affected.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
INHERENT	2	1	01	001	001

Inclusive OR **ORA**

Operation: ACCX ← (ACCX)⊙(M)

Description: Perform logical "OR" between the contents of ACCX and the contents of M and places the result in ACCX. (Each bit of ACCX after the operation will be the logical "OR" of the corresponding bits of M and of ACCX before the operation).

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Cleared.
- C: Not affected.

Boolean Formulae for Condition Codes:

$N = R_7$

$Z = \overline{R}_7 \cdot \overline{R}_6 \cdot \overline{R}_5 \cdot \overline{R}_4 \cdot \overline{R}_3 \cdot \overline{R}_2 \cdot \overline{R}_1 \cdot \overline{R}_0$

$V = 0$

Addressing Formats:

See Table A-1.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

(DUAL OPERAND)

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
A IMM	2	2	8A	212	138
A DIR	3	2	9A	232	154
A EXT	4	3	BA	272	186
A IND	5	2	AA	252	170
B IMM	2	2	CA	312	202
B DIR	3	2	DA	332	218
B EXT	4	3	FA	372	250
B IND	5	2	EA	352	234

PSH

Push Data Onto Stack

Operation: ↓ (ACCX)
SP ← (SP) − 0001

Description: The contents of ACCX is stored in the stack at the address contained in the stack pointer. The stack pointer is then decremented.

Condition Codes: Not affected.

Addressing Formats:

See Table A-4.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
A	4	1	36	066	054
B	4	1	37	067	055

Pull Data from Stack **PUL**

Operation: SP ← (SP) + 0001
↑ ACCX

Description: The stack pointer is incremented. The ACCX is then loaded from the stack, from the address which is contained in the stack pointer.

Condition Codes: Not affected.

Addressing Formats:

See Table A-4.

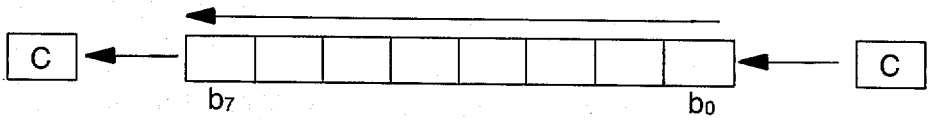
Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
A	4	1	32	062	050
B	4	1	33	063	051

ROL

Rotate Left

Operation:



Description: Shifts all bits of ACCX or M one place to the left. Bit 0 is loaded from the C bit. The C bit is loaded from the most significant bit of ACCX or M.

Condition Codes:

H: Not affected.
I: Not affected.
N: Set if most significant bit of the result is set; cleared otherwise.
Z: Set if all bits of the result are cleared; cleared otherwise.
V: Set if, after the completion of the operation, EITHER (N is set and C is cleared) OR (N is cleared and C is set); cleared otherwise.
C: Set if, before the operation, the most significant bit of the ACCX or M was set; cleared otherwise.

Boolean Formulae for Condition Codes:

$N = R_7$

$Z = \overline{R}_7 \cdot \overline{R}_6 \cdot \overline{R}_5 \cdot \overline{R}_4 \cdot \overline{R}_3 \cdot \overline{R}_2 \cdot \overline{R}_1 \cdot \overline{R}_0$

$V = N \oplus C = [N \cdot \overline{C}] \odot [\overline{N} \cdot C]$

(the foregoing formula assumes values of N and C after the rotation)

$C = M_7$

Addressing Formats:

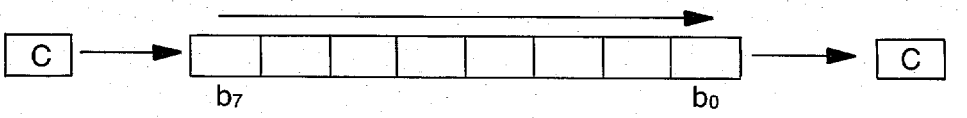
See Table A-3

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
A	2	1	49	111	073
B	2	1	59	131	089
EXT	6	3	79	171	121
IND	7	2	69	151	105

Rotate Right **ROR**

Operation:



Description: Shifts all bits of ACCX or M one place to the right. Bit 7 is loaded from the C bit. The C bit is loaded from the least significant bit of ACCX or M.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Set if, after the completion of the operation, EITHER (N is set and C is cleared) OR (N is cleared and C is set); cleared otherwise.
- C: Set if, before the operation, the least significant bit of the ACCX or M was set; cleared otherwise.

Boolean Formulae for Condition Codes:

$N = R_7$

$Z = \overline{R}_7 \cdot \overline{R}_6 \cdot \overline{R}_5 \cdot \overline{R}_4 \cdot \overline{R}_3 \cdot \overline{R}_2 \cdot \overline{R}_1 \cdot \overline{R}_0$

$V = N \oplus C = [N \cdot \overline{C}] \odot [\overline{N} \cdot C]$

(the foregoing formula assumes values of N and C after the rotation)

$C = M_0$

Addressing Formats:

See Table A-3

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
A	2	1	46	106	070
B	2	1	56	126	086
EXT	6	3	76	166	118
IND	7	2	66	146	102

RTI

Return from Interrupt

Operation:
SP ← (SP) + 0001 , ↑CC
SP ← (SP) + 0001 , ↑ACCB
SP ← (SP) + 0001 , ↑ACCA
SP ← (SP) + 0001 , ↑IXH
SP ← (SP) + 0001 , ↑IXL
SP ← (SP) + 0001 , ↑PCH
SP ← (SP) + 0001 , ↑PCL

Description: The condition codes, accumulators B and A, the index register, and the program counter, will be restored to a state pulled from the stack. Note that the interrupt mask bit will be reset if and only if the corresponding bit stored in the stack is zero.

Condition Codes: Restored to the states pulled from the stack.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
INHERENT	10	1	3B	073	059

Return from Interrupt

Example

		Memory Location	Machine Code (Hex)	Label	Assembler Language Operator	Operand
A.	*Before*					
	PC →	$D066	3B		RTI	
	SP →	$EFF8				
		$EFF9	11HINZVC	(binary)		
		$EFFA	12			
		$EFFB	34			
		$EFFC	56			
		$EFFD	78			
		$EFFE	55			
		$EFFF	67			
B.	*After*					
	PC →	$5567	**		***	*****
		$EFF8				
		$EFF9	11HINZVC	(binary)		
		$EFFA	12			
		$EFFB	34			
		$EFFC	56			
		$EFFD	78			
		$EFFE	55			
	SP →	$EFFF	67			

CC = HINZVC (binary)
ACCB = 12 (Hex) IXH = 56 (Hex)
ACCA = 34 (Hex) IXL = 78 (Hex)

Return from Subroutine

RTS

Operation: SP ← (SP) + 0001
↑ PCH
SP ← (SP) + 0001
↑ PCL

Description: The stack pointer is incremented (by 1). The contents of the byte of memory, at the address now contained in the stack pointer, are loaded into the 8 bits of highest significance in the program counter. The stack pointer is again incremented (by 1). The contents of the byte of memory, at the address now contained in the stack pointer, are loaded into the 8 bits of lowest significiance in the program counter.

Condition Codes: Not affected.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
INHERENT	5	1	39	071	057

Return from Subroutine

EXAMPLE

		Memory Location	Machine Code (Hex)	Assembler Language Label	Operator	Operand
A.	*Before*					
	PC	$30A2	39		RTS	
	SP	$EFFD				
		$EFFE	10			
		$EFFF	02			
B.	*After*					
	PC	$1002	**		***	*****
		$EFFD				
		$EFFE	10			
	SP	$EFFF	02			

SBA

Subtract Accumulators

Operation: ACCA ← (ACCA) − (ACCB)

Description: Subtracts the contents of ACCB from the contents of ACCA and places the result in ACCA. The contents of ACCB are not affected.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Set if there was two's complement overflow as a result of the operation.
- C: Carry is set if the absolute value of accumulator B plus previous carry is larger than the absolute value of accumulator A; reset otherwise.

Boolean Formulae for Condition Codes:

$$N = R_7$$
$$Z = \overline{R}_7 \cdot \overline{R}_6 \cdot \overline{R}_5 \cdot \overline{R}_4 \cdot \overline{R}_3 \cdot \overline{R}_2 \cdot \overline{R}_1 \cdot \overline{R}_0$$
$$V = A_7 \cdot \overline{B}_7 \cdot \overline{R}_7 + \overline{A}_7 \cdot B_7 \cdot R_7$$
$$C = \overline{A}_7 \cdot B_7 + B_7 \cdot R_7 + R_7 \cdot \overline{A}_7$$

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
INHERENT	2	1	10	020	016

Subtract with Carry

SBC

Operation: ACCX ← (ACCX) − (M) − (C)

Description: Subtracts the contents of M and C from the contents of ACCX and places the result in ACCX.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Set if there was two's complement overflow as a result of the operation; cleared otherwise.
- C: Carry is set if the absolute value of the contents of memory plus previous carry is larger than the absolute value of the accumulator; reset otherwise.

Boolean Formulae for Condition Codes:

$$N = R_7$$
$$Z = \overline{R_7} \cdot \overline{R_6} \cdot \overline{R_5} \cdot \overline{R_4} \cdot \overline{R_3} \cdot \overline{R_2} \cdot \overline{R_1} \cdot \overline{R_0}$$
$$V = X_7 \cdot \overline{M_7} \cdot \overline{R_7} + \overline{X_7} \cdot M_7 \cdot R_7$$
$$C = \overline{X_7} \cdot M_7 + M_7 \cdot R_7 + R_7 \cdot \overline{X_7}$$

Addressing Formats:

See Table A-1.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

(DUAL OPERAND)

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
A IMM	2	2	82	202	130
A DIR	3	2	92	222	146
A EXT	4	3	B2	262	178
A IND	5	2	A2	242	162
B IMM	2	2	C2	302	194
B DIR	3	2	D2	322	210
B EXT	4	3	F2	362	242
B IND	5	2	E2	342	226

SEC

Set Carry

Operation: C bit ← 1

Description: Sets the carry bit in the processor condition codes register.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Not affected.
- Z: Not affected.
- V: Not affected.
- C: Set.

Boolean Formulae for Condition Codes:

C = 1

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
INHERENT	2	1	0D	015	013

Set Interrupt Mask **SEI**

Operation: I bit ← 1

Description: Sets the interrupt mask bit in the processor condition codes register. The microprocessor is inhibited from servicing an interrupt from a peripheral device, and will continue with execution of the instructions of the program, until the interrupt mask bit has been cleared.

Condition Codes:
H: Not affected.
I: Set.
N: Not affected.
Z: Not affected.
V: Not affected.
C: Not affected.

Boolean Formulae for Condition Codes:
I = 1

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
INHERENT	2	1	0F	017	015

SEV

Set Two's Complement Overflow Bit

Operation: V bit ← 1

Description: Sets the two's complement overflow bit in the processor condition codes register.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Not affected.
- Z: Not affected.
- V: Set.
- C: Not affected.

Boolean Formulae for Condition Codes:

V = 1

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
INHERENT	2	1	0B	013	011

Store Accumulator **STA**

Operation: M ← (ACCX)

Description: Stores the contents of ACCX in memory. The contents of ACCX remains unchanged.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the contents of ACCX is set; cleared otherwise.
- Z: Set if all bits of the contents of ACCX are cleared; cleared otherwise.
- V: Cleared.
- C: Not affected.

Boolean Formulae for Condition Codes:

$N = X_7$

$Z = \overline{X}_7 \cdot \overline{X}_6 \cdot \overline{X}_5 \cdot \overline{X}_4 \cdot \overline{X}_3 \cdot \overline{X}_2 \cdot \overline{X}_1 \cdot \overline{X}_0$

$V = 0$

Addressing Formats:

See Table A-2.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
A DIR	4	2	97	227	151
A EXT	5	3	B7	267	183
A IND	6	2	A7	247	167
B DIR	4	2	D7	327	215
B EXT	5	3	F7	367	247
B IND	6	2	E7	347	231

STS

Store Stack Pointer

Operation: $M \leftarrow (SPH)$
$M + 1 \leftarrow (SPL)$

Description: Stores the more significant byte of the stack pointer in memory at the address specified by the program, and stores the less significant byte of the stack pointer at the next location in memory, at one plus the address specified by the program.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the stack pointer is set; cleared otherwise.
- Z: Set if all bits of the stack pointer are cleared; cleared otherwise.
- V: Cleared.
- C: Not affected.

Boolean Formulae for Condition Codes:

$$N = SPH_7$$
$$Z = (\overline{SPH_7} \cdot \overline{SPH_6} \cdot \overline{SPH_5} \cdot \overline{SPH_4} \cdot \overline{SPH_3} \cdot \overline{SPH_2} \cdot \overline{SPH_1} \cdot \overline{SPH_0}) \cdot (\overline{SPL_7} \cdot \overline{SPL_6} \cdot \overline{SPL_5} \cdot \overline{SPL_4} \cdot \overline{SPL_3} \cdot \overline{SPL_2} \cdot \overline{SPL_1} \cdot \overline{SPL_0})$$
$$V = 0$$

Addressing Formats:

See Table A-6.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
DIR	5	2	9F	237	159
EXT	6	3	BF	277	191
IND	7	2	AF	257	175

Store Index Register

STX

Operation: $M \leftarrow (IXH)$
$M + 1 \leftarrow (IXL)$

Description: Stores the more significant byte of the index register in memory at the address specified by the program, and stores the less significant byte of the index register at the next location in memory, at one plus the address specified by the program.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if the most significant bite of the index register is set; cleared otherwise.
- Z: Set if all bits of the index register are cleared; cleared otherwise.
- V: Cleared.
- C: Not affected.

Boolean Formulae for Condition Codes:

$$N = IXH_7$$
$$Z = (\overline{IXH_7} \cdot \overline{IXH_6} \cdot \overline{IXH_5} \cdot \overline{IXH_4} \cdot \overline{IXH_3} \cdot \overline{IXH_2} \cdot \overline{IXH_1} \cdot \overline{IXH_0}) \cdot (\overline{IXL_7} \cdot \overline{IXL_6} \cdot \overline{IXL_5} \cdot \overline{IXL_4} \cdot \overline{IXL_3} \cdot \overline{IXL_2} \cdot \overline{IXL_1} \cdot \overline{IXL_0})$$
$$V = 0$$

Addressing Formats:

See Table A-6.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
DIR	5	2	DF	337	223
EXT	6	3	FF	377	255
IND	7	2	EF	357	239

SUB

Subtract

Operation: ACCX ← (ACCX) − (M)

Description: Subtracts the contents of M from the contents of ACCX and places the result in ACCX.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- V: Set if there was two's complement overflow as a result of the operation; cleared otherwise.
- C: Set if the absolute value of the contents of memory are larger than the absolute value of the accumulator; reset otherwise.

Boolean Formulae for Condition Codes:

$N = R_7$

$Z = \overline{R}_7 \cdot \overline{R}_6 \cdot \overline{R}_5 \cdot \overline{R}_4 \cdot \overline{R}_3 \cdot \overline{R}_2 \cdot \overline{R}_1 \cdot \overline{R}_0$

$V = X_7 \cdot \overline{M}_7 \cdot \overline{R}_7 \cdot \overline{X}_7 \cdot M_7 \cdot R_7$

$C = \overline{X}_7 \cdot M_7 + M_7 \cdot R_7 + R_7 \cdot \overline{X}_7$

Addressing Formats:

See Table A-1.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

(DUAL OPERAND)

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
A IMM	2	2	80	200	128
A DIR	3	2	90	220	144
A EXT	4	3	B0	260	176
A IND	5	2	A0	240	160
B IMM	2	2	C0	300	192
B DIR	3	2	D0	320	208
B EXT	4	3	F0	360	240
B IND	5	2	E0	340	224

Software Interrupt **SWI**

Operation:

PC ← (PC) + 0001
↓ (PCL) , SP ← (SP)-0001
↓ (PCH) , SP ← (SP)-0001
↓ (IXL) , SP ← (SP)-0001
↓ (IXH) , SP ← (SP)-0001
↓ (ACCA) , SP ← (SP)-0001
↓ (ACCB) , SP ← (SP)-0001
↓ (CC) , SP ← (SP)-0001
I ← 1
PCH ← (n-0005)
PCL ← (n-0004)

Description: The program counter is incremented (by 1). The program counter, index register, and accumulator A and B, are pushed into the stack. The condition codes register is then pushed into the stack, with condition codes H, I, N, Z, V, C going respectively into bit positions 5 thru 0, and the top two bits (in bit positions 7 and 6) are set (to the 1 state). The stack pointer is decremented (by 1) after each byte of data is stored in the stack.

The interrupt mask bit is then set. The program counter is then loaded with the address stored in the software interrupt pointer at memory locations (n-5) and (n-4), where n is the address corresponding to a high state on all lines of the address bus.

Condition Codes:
H: Not affected.
I: Set.
N: Not affected.
Z: Not affected.
V: Not affected.
C: Not affected.

Boolean Formula for Condition Codes:
I = 1

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
INHERENT	12	1	3F	077	063

Software Interrupt

EXAMPLE

A. *Before:*

CC = HINZVC (binary)
ACCB = 12 (Hex) IXH = 56 (Hex)
ACCA = 34 (Hex) IXL = 78 (Hex)

		Memory Location	Machine Code (Hex)	Assembler Language Label	Operator	Operand
PC	→	$5566	3F		SWI	
SP	→	$EFFF				
		$FFFA	D0			
		$FFFB	55			

B. *After:*

PC	→	$D055				
SP	→	$EFF8				
		$EFF9	11HINZVC	(binary)		
		$EFFA	12			
		$EFFB	34			
		$EFFC	56			
		$EFFD	78			
		$EFFE	55			
		$EFFF	67			

Note: This example assumes that FFFF is the memory location addressed when all lines of the address bus go to the high state.

Transfer from Accumulator A to Accumulator B

TAB

Operation: ACCB ← (ACCA)

Description: Moves the contents of ACCA to ACCB. The former contents of ACCB are lost. The contents of ACCA are not affected.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the contents of the accumulator is set; cleared otherwise.
- Z: Set if all bits of the contents of the accumulator are cleared; cleared otherwise.
- V: Cleared.
- C: Not affected.

Boolean Formulae for Condition Codes:

$N = R_7$

$Z = \overline{R}_7 \cdot \overline{R}_6 \cdot \overline{R}_5 \cdot \overline{R}_4 \cdot \overline{R}_3 \cdot \overline{R}_2 \cdot \overline{R}_1 \cdot \overline{R}_0$

$V = 0$

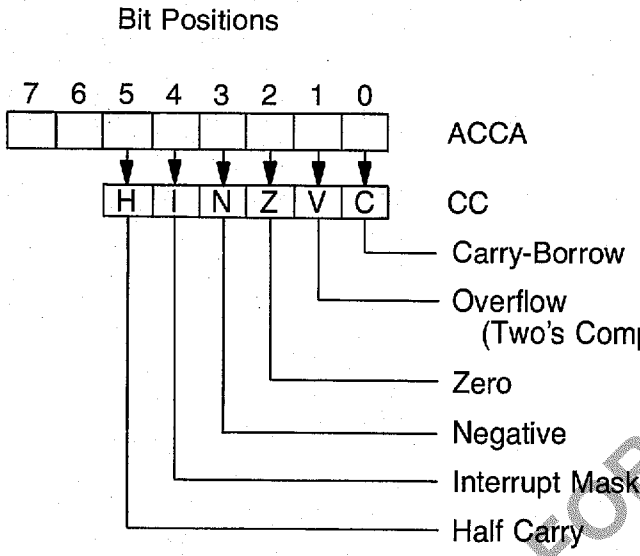
Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
INHERENT	2	1	16	026	022

TAP

Transfer from Accumulator A to Processor Condition Codes Register

Operation: CC ← (ACCA)



Description: Transfers the contents of bit positions 0 thru 5 of accumulator A to the corresponding bit positions of the processor condition codes register. The contents of accumulator A remain unchanged.

Condition Codes: Set or reset according to the contents of the respective bits 0 thru 5 of accumulator A.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
INHERENT	2	1	06	006	006

Transfer from Accumulator B to Accumulator A

TBA

Operation: ACCA ← (ACCB)

Description: Moves the contents of ACCB to ACCA. The former contents of ACCA are lost. The contents of ACCB are not affected.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if the most significant accumulator bit is set; cleared otherwise.
- Z: Set if all accumulator bits are cleared; cleared otherwise.
- V: Cleared.
- C: Not affected.

Boolean Formulae for Condition Codes:

$N = R_7$

$Z = \overline{R}_7 \cdot \overline{R}_6 \cdot \overline{R}_5 \cdot \overline{R}_4 \cdot \overline{R}_3 \cdot \overline{R}_2 \cdot \overline{R}_1 \cdot \overline{R}_0$

$V = 0$

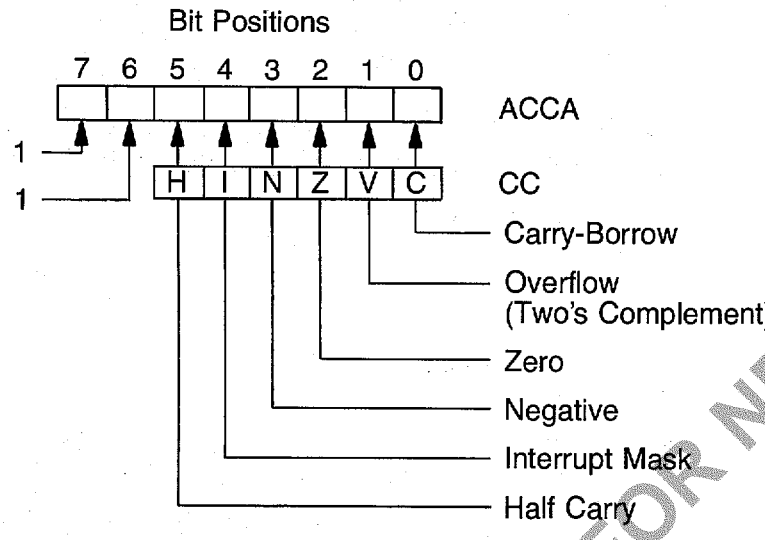
Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
INHERENT	2	1	17	027	023

TPA

Transfer from Processor Condition Codes Register to Accumulator A

Operation: ACCA ← (CC)



Description: Transfers the contents of the processor condition codes register to corresponding bit positions 0 thru 5 of accumulator A. Bit positions 6 and 7 of accumulator A are set (i.e. go to the "1" state). The processor condition codes register remains unchanged.

Condition Codes: Not affected.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
INHERENT	2	1	07	007	007

Test **TST**

Operation: (ACCX) − 00
(M) − 00

Description: Set condition codes N and Z according to the contents of ACCX or M.

Condition Codes:
- H: Not affected.
- I: Not affected.
- N: Set if most significant bit of the contents of ACCX or M is set; cleared otherwise.
- Z: Set if all bits of the contents of ACCX or M are cleared; cleared otherwise.
- V: Cleared.
- C: Cleared.

Boolean Formulae for Condition Codes:

$N = M_7$
$Z = \overline{M}_7 \cdot \overline{M}_6 \cdot \overline{M}_5 \cdot \overline{M}_4 \cdot \overline{M}_3 \cdot \overline{M}_2 \cdot \overline{M}_1 \cdot \overline{M}_0$
$V = 0$
$C = 0$

Addressing Formats:

See Table A-3.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
A	2	1	4D	115	077
B	2	1	5D	135	093
EXT	6	3	7D	175	125
IND	7	2	6D	155	109

TSX

Transfer from Stack Pointer to Index Register

Operation: IX ← (SP) + 0001

Description: Loads the index register with one plus the contents of the stack pointer. The contents of the stack pointer remain unchanged.

Condition Codes: Not affected.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
INHERENT	4	1	30	060	048

Transfer From Index Register to Stack Pointer **TXS**

Operation: SP ← (IX) − 0001

Description: Loads the stack pointer with the contents of the index register, minus one. The contents of the index register remain unchanged.

Condition Codes: Not affected.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
INHERENT	4	1	35	.065	053

WAI

Wait for Interrupt

Operation:

PC ← (PC) + 0001
↓ (PCL) , SP ← (SP)-0001
↓ (PCH) , SP ← (SP)-0001
↓ (IXL) , SP ← (SP)-0001
↓ (IXH) , SP ← (SP)-0001
↓ (ACCA) , SP ← (SP)-0001
↓ (ACCB) , SP ← (SP)-0001
↓ (CC) , SP ← (SP)-0001

Condition Codes: Not affected.

Description: The program counter is incremented (by 1). The program counter, index register, and accumulators A and B, are pushed into the stack. The condition codes register is then pushed into the stack, with condition codes H, I, N, Z, V, C going respectively into bit positions 5 thru 0, and the top two bits (in bit positions 7 and 6) are set (to the 1 state). The stack pointer is decremented (by 1) after each byte of data is stored in the stack.

Execution of the program is then suspended until an interrupt from a peripheral device is signalled, by the interrupt request control input going to a low state.

When an interrupt is signalled on the interrupt request line, and provided the I bit is clear, execution proceeds as follows. The interrupt mask bit is set. The program counter is then loaded with the address stored in the internal interrupt pointer at memory locations (n-7) and (n-6), where n is the address corresponding to a high state on all lines of the address bus.

Condition Codes:

H: Not affected.
I: Not affected until an interrupt request signal is detected on the interrupt request control line. When the interrupt request is received the I bit is set and further execution takes place, provided the I bit was initially clear.
N: Not affected.
Z: Not affected.
V: Not affected.
C: Not affected.

Addressing Modes, Execution Time, and Machine Code (hexadecimal/ octal/ decimal):

Addressing Modes	Execution Time (No. of cycles)	Number of bytes of machine code	Coding of First (or only) byte of machine code		
			HEX.	OCT.	DEC.
INHERENT	9	1	3E	076	062

Addressing Mode of Second Operand	First Operand	
	Accumulator A	Accumulator B
IMMediate	CCC A #number CCC A #symbol CCC A #expression CCC A #'C	CCC B #number CCC B #symbol CCC B #expression CCC B #'C
DIRect or EXTended	CCC A number CCC A symbol CCC A expression	CCC B number CCC B symbol CCC B expression
INDexed	CCC A X CCC Z ,X CCC A number,X CCC A symbol,X CCC A expression,X	CCC B X CCC B ,X CCC B number,X CCC B symbol,X CCC B expression,X

Notes: 1. CCC = mnemonic operator of source instruction.
2. "symbol" may be the special symbol "*".
3. "expression" may contain the special symbol "*".
4. space may be omitted before A or B.

Applicable to the following source instructions:

ADC ADD AND BIT CMP
EOR LDA ORA SBC SUB

*Special symbol indicating program-counter.

TABLE A-1. Addressing Formats (1)

Addressing Mode of Second Operand	First Operand	
	Accumulator A	Accumulator B
DIRect or EXTended	STA A number STA A symbol STA A expression	STA B number STA B symbol STA B expression
INDexed	STA A X STA A ,X STA A number,X STA A symbol,X STA A expression,X	STA B X STA B ,X STA B number,X STA B symbol,X STA B expression,X

Notes: 1. "symbol" may be the special symbol "*".
2. "expression" may contain the special symbol "*".
3. Space may be omitted before A or B.

Applicable to the source instruction:

STA

*Special symbol indicating program-counter.

TABLE A-2. Addressing Formats (2)

Operand or Addressing Mode	Formats
Accumulator A	CCC A
Accumulator B	CCC B
EXTended	CCC number CCC symbol CCC expression
INDexed	CCC X CCC ,X CCC number,X CCC symbol,X CCC expression,X

Notes: 1. CCC = mnemonic operator of source instruction.
2. "symbol" may be the special symbol "*".
3. "expression" may contain the special symbol "*".
4. Space may be omitted before A or B.

Applicable to the following source instructions:

ASL ASR CLR COM DEC INC
LSR NEG ROL ROR TST

*Special symbol indicating program-counter.

TABLE A-3. Addressing Formats (3)

Operand	Formats
Accumulator A	CCC A
Accumulator B	CCC B

Notes: 1. CCC = mnemonic operator of source instruction.
2. Space may be omitted before A or B.

Applicable to the following source instructions:

PSH PUL

TABLE A-4. Addressing Formats (4)

Addressing Mode	Formats
IMMediate	CCC #number CCC #symbol CCC #expression CCC #'C
DIRect or EXTended	CCC number CCC symbol CCC expression
INDexed	CCC X CCC ,X CCC number,X CCC symbol,X CCC expression,X

Notes: 1. CCC = mnemonic operator of source instruction.
2. "symbol" may be the special symbol "*".
3. "expression" may contain the special symbol "*".

Applicable to the following source instructions:

CPX LDS LDX

*Special symbol indicating program-counter.

TABLE A-5. Addressing Formats (5)

Addressing Mode	Formats
DIRect or EXTended	CCC number CCC symbol CCC expression
INDexed	CCC X CCC ,X CCC number,X CCC symbol,X CCC expression,X

Notes: 1. CCC = mnemonic operator of source instruction.
2. "symbol" may be the special symbol "*".
3. "expression" may contain the special symbol "*".

Applicable to the following source instructions:

STS STX

*Special symbol indicating program-counter.

TABLE A-6. Addressing Formats (6)

Addressing Mode	Formats
EXTended	CCC number CCC symbol CCC expression
INDexed	CCC X CCC ,X CCC number,X CCC symbol,X CCC expression,X

Notes: 1. CCC = mnemonic operator of source instruction.
2. "symbol" may be the special symbol "*".
3. "expression" may contain the special symbol "*".

Applicable to the following source instructions:

JMP JSR

*Special symbol indicating program-counter.

TABLE A-7. Addressing Formats (7)

Addressing Mode	Formats
RELative	CCC number CCC symbol CCC expression

Notes: 1. CCC = mnemonic operator of source instruction.
2. "symbol" may be the special symbol "*".
3. "expression" may contain the special symbol "*".

Applicable to the following source instructions:

BCC BCS BEQ BGE BGT BHI BLE BLS
BLT BMI BNE BPL BRA BSR BVC BVS

*Special symbol indicating program-counter.

TABLE A-8. Addressing Formats (8)

APPENDIX B
EXbug COMMANDS

EXbug COMMANDS	DESCRIPTION
LOAD	Initiates memory loader function.
VERF	Compares contents of memory with tape data. Where unequal, prints location in hexadecimal.
PNCH	Instructs EXORciser to punch an absolute formatted binary object tape.
PRNT	Causes terminal to print the contents of memory in hexadecimal followed by the literal ASCII characters.
SRCH	Searches tape for header record. Stops reader at first record encountered and prints that record.
S10, S30, & S120	Inserts nulls for proper printing during terminal operation at 110, 300, and 1200 Baud: θ, or θ, or 3 nulls, respectively, inserted after standard ASCII characters; θ, or 4, or 23, respectively, inserted following carriage return character.
S240 (EXbug 1.2 only)	Inserts nulls for proper printing during 2400 Baud terminal operation. Seven nulls inserted after standard ASCII characters; 47 nulls inserted following carriage return character.
TERM (EXbug 1.2 only)	Prints the number of nulls currently being inserted after standard ASCII and carriage return characters. Permits either or both to be changed.
MAID	
n/	Print the contents of memory location n and enable the EXORciser to change the contents of this memory location.
n;0	Calculate the address offset (for relative addressing mode instructions.
(LF)	Print the contents of the next sequential memory location and enable the EXORciser to change the contents of this memory location (LF — Line Feed character).
↑	Print the contents of the previous sequential memory location and enable the EXORciser to change the contents of this memory location (↑ — up arrow character, or SHIFT key, or N character).

EXbug COMMANDS	DESCRIPTION
(CR)	Return the displayed contents to memory and accept next command (CR — Carriage Return character).
n;V	Enter a breakpoint at memory location n.
$V	Display the memory location of each breakpoint.
n;P	Continue executing from the selected breakpoint until this breakpoint is encountered n times.
;U	Remove all the breakpoints.
n;U	Remove the breakpoint at memory location n.
n;W	Search for the n bit pattern.
$M	Display the search mask.
;G	Execute the user's program starting at the auto restart memory location.
n;G	Execute user's program starting at memory location n.
$R	Display/change the user's program registers.
;P	Continue executing from the current program counter setting.
;N	Trace one instruction.
N	Trace one instruction.
n;N	Trace n instructions.
$T	Set the trace mode.
;T	Reset the trace mode.
$S	Display and set the stop-on-address compare — Scope trigger pulse.
;S	Reset the stop-on-address compare — Scope trigger pulse.
#n=	Convert the decimal number n to its hexadecimal equivalent.
#$n=	Convert the hexadecimal number n to its decimal equivalent.
#@n=	Convert the octal number n to its hexadecimal equivalent.

APPENDIX C
MIKbug COMMANDS

OPERATION	DESCRIPTION
L	Load tape.
M	Memory change.
R	Display registers CC, B, A, X, P, S.
P	Print/ punch tape.
G	Go to location.

APPENDIX D
MINIbug II COMMANDS

COMMAND	DESCRIPTION
L	Load tape.
M	Memory Change.
P	Print/ punch dump.
R	Display registers CC, B, A, X, P, S.
S	Set terminal speed: 1 — set speed for 10 cps 3 — set speed for 30 cps
G	Go to location nnnn.
W	Memory test.
Y	Punch binary tape.
Z	Load binary tape.
See Evaluation Module II User's Guide for detailed description.	

APPENDIX E
MINIbug III COMMANDS

COMMAND	DESCRIPTION
L	Load.
M	Memory change.
P	Print/ punch dump.
R	Display registers CC, B, A, X, P, S.
S	Set terminal speed: 1 — set speed for 10 cps 3 — set speed for 30 cps
B	Print out all breakpoints.
C	Continue execution from current location.
N	Next instruction.
T	Trace "N" instructions.
G	Go to location "N".
D	Delete all breakpoints.
U	Reset breakpoint with address "N".
V	Set a breakpoint with address "N".

APPENDIX F
ASCII CODE CONVERSION TABLE

BITS 4 thru 6 → / BITS 0 thru 3 ↓	—	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P		p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	”	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	’	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	/	l	/
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	≈
F	SI	US	/	?	O	—	o	DEL

APPENDIX G
HEXADECIMAL AND DECIMAL CONVERSION

From hex: locate each hex digit in its corresponding column position and note the decimal equivalents. Add these to obtain the decimal value.

From decimal: (1) locate the largest decimal value in the table that will fit into the decimal number to be converted, and (2) note its hex equivalent and hex column position. (3) Find the decimal remainder. Repeat the process on this and subsequent remainders.

HEXADECIMAL COLUMNS											
6		5		4		3		2		1	
HEX = DEC		HEX = DEC		HEX = DEC		HEX = DEC		HEX = DEC		HEX = DEC	
0	0	0	0	0	0	0	0	0	0	0	0
1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	14,680,064	E	917,504	E	54,344	E	3,584	E	224	E	14
F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15
0 1 2 3		4 5 6 7		0 1 2 3		4 5 6 7		0 1 2 3		4 5 6 7	
BYTE				BYTE				BYTE			

POWERS OF 2

2^n	n
256	8
512	9
1 024	10
2 048	11
4 096	12
8 192	13
16 384	14
32 768	15
65 536	16
131 072	17
262 144	18
524 288	19
1 048 576	20
2 097 152	21
4 194 304	22
8 388 608	23
16 777 216	24

$2^0 = 16^0$
$2^4 = 16^1$
$2^8 = 16^2$
$2^{12} = 16^3$
$2^{16} = 16^4$
$2^{20} = 16^5$
$2^{24} = 16^6$
$2^{28} = 16^7$
$2^{32} = 16^8$
$2^{36} = 16^9$
$2^{40} = 16^{10}$
$2^{44} = 16^{11}$
$2^{48} = 16^{12}$
$2^{52} = 16^{13}$
$2^{56} = 16^{14}$
$2^{60} = 16^{15}$

POWERS OF 16

16^n	n
1	0
16	1
256	2
4 096	3
65 536	4
1 048 576	5
16 777 216	6
268 435 456	7
4 294 967 296	8
68 719 476 736	9
1 099 511 627 776	10
17 592 186 044 416	11
281 474 976 710 656	12
4 503 599 627 370 496	13
72 057 594 037 927 936	14
1 152 921 504 606 846 976	15

MOTOROLA Semiconductor Products Inc.

P.O. BOX 20912 • PHOENIX, ARIZONA 85036 • A SUBSIDIARY OF MOTOROLA INC.

A10803-18 PRINTED IN USA 1/92 IMPERIAL LITHO 83311 2,500 MCU YGACAA