MICROCONTROLLERS AND APPLICATIONS

BY

Dr. **P. VENKATESWARAN** M.E. Tel. E. (J.U.), Ph. D. (JU)
MISTE MIETE F.I.E.(I) SMIEEE (USA) [redacted]
[redacted]
[redacted] OF [redacted]
[redacted] - [redacted] 30. [redacted] Dt.

ACKNOWLEDGEMENTS

[redacted] DEPT. OF E.T.E
JADAVPUR UNIVERSITY CALCUTTA - 700 032.

Mr. VIVEK G. KHEKALE
CENTRE FOR ELECTRONICS DESIGN TECHNOLOGY
CEDT - AURANGABAD MAHARASTRA

TOPICS

1. Introduction

"The microprocessors made an impact on industries of vast diversification in nature such as machine tools [1], bio-medical instrumentation and chemical processes [1], etc., thus proved its utility as a general purpose system design (GPSD) tool [2], with *suitable interfacing devices*. The necessity to evolve a higher performance architecture, to be used as a specific purpose system design (SPSD) tool [2] for *dedicated* applications leads to the development of *microcontrollers*.

The microprocessor architecture has three basic segments : arithmatic / logic unit (ALU), register unit and control unit [1]. Other hardware requirements such as, for clock generator, RAM, ROM / EPROM, parallel and serial I / O ports are to be supported in the form of external chips and that is the difference between microprocessors and microcontrollers [2].

Microcontrollers, also called as Single-chip microcomputers are designed on a single-chip such that besides microprocessing unit, it includes RAM, ROM / EPROM, I / O ports, timer / counter and other circuitry required for microcomputer working [3]. This is shown in Fig. 1.

Microcontrollers are used primarily to perform dedicated functions [1]. They are also used as independent controllers in machines or as slave processors in the distributed processing environment [1].

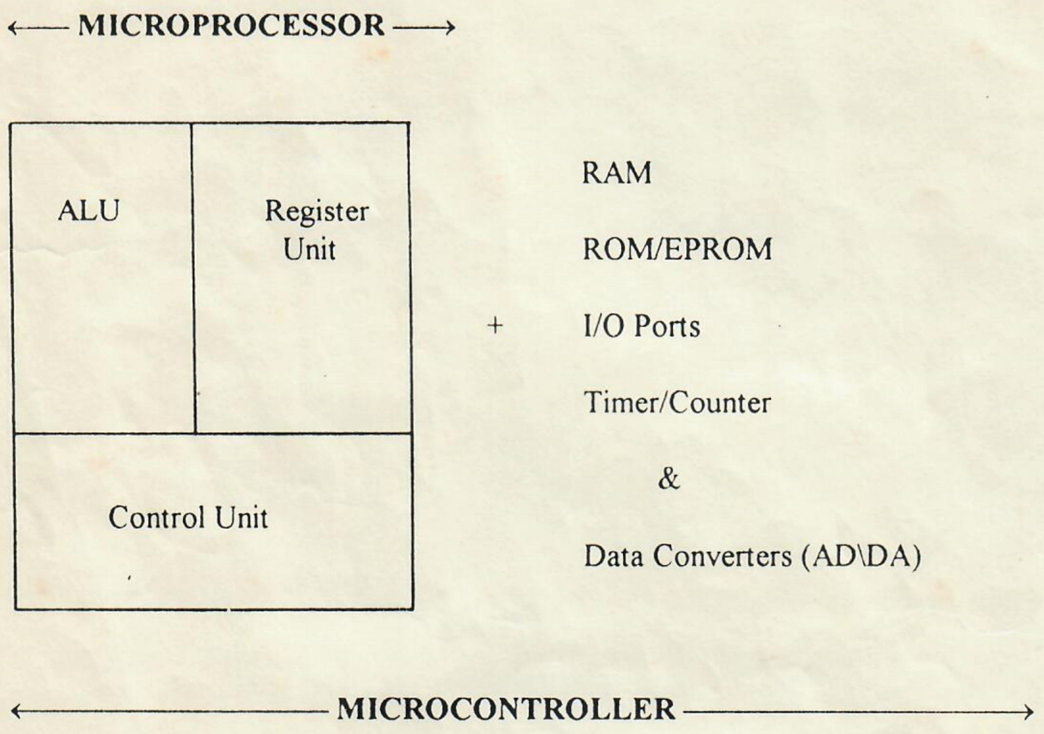

Fig. 1 : Block diagram of Microcontroller

LDR and Stepper Motor

Sensor and *actuator* are the two major cmponents of a positioning system. The requirements of the control task determines the nature of these two components and subsequently the cost, also. Light dependant resistor (LDR), which exhibits light

II. EVOLUTION OF MICROCONTROLLERS

In 1971, Intel corporation developed the first 4 bit microprocesor architecture. Soon, Motorola, Zilog and other manufactures brought out their respective microprocessors to the market. The 8085, an 8-bit micrprocesor from Intel, made a big impact in the market.

One design trend is to go for 16 and higher bit microprocessors to improve the data handling capabilities and speed of execution. 8086 and 8088 are the 16-bit micro processors from Intel Corporation. The 8088 is a proceessor with 8-bit data bus and 16-bit capabilities. Another design trend is to go for higher performance architecture for dedicated applications, meant to be used as specific purpose system design tool.

For example microcomputer design based on microprocessor involves the interfacing of several hardwares such as memory, parallel I/O ports, serial I/O ports, timer, etc. Thus any microprocessor based application design necessitates the handling of multiple chips, resulting in increased PCB size and lesser reliability in the functioning of the circuit. This concept was totally changed with the introduction of single-chip microcomputer/microcontroller by Intel Corporation in 1976. Because of recent advances in semiconductor design and fabrication technology, it is possible to integrate the entire system on a single chip.

For example, Intel's MCS-48 microcontroller family has RAM, ROM/EPROM, Parallel I/O ports, timer/counter besides an 8-bit CPU. Thus with microcontrollers, the designer can make the system design more compact at less cost and yet powerful. In 1980, Intel came up with MCS-51 series of microcontrollers featuring serial interface facility, also. This second generation microcontrollers are more popular for sophisticated real-time instrumentation and industrial control applications. This was followed by MCS-96 family of microcontrollers, fastest among Intel's series, incorporating A/D converter also, with a 16 bit CPU.

Intel products are upward compatible with successive architectures. Peripherals and interfacing chips are easily available in the market for Intel products. Moreover, Intel provides well documentation support and hence Intel products are more popular.

III. MCS - 51 Family and its features :

The 8051 is the core of MCS-51 family and its **features** are (14.1):
8-bit CPU meant for control applications
Incorporated with Boolean (single-bit) processor
128 bytes Scratch-pad memory (on –chip data RAM)
4K bytes on-chip program memory (PM)
64K expandable program and data memory (PM and DM) space
32 bidierctional, individually accessible I/O lines, and can be accessed as a group of 8 lines.
Two 16-bit Timer/Counter
Serial interface facility
6/5 vecter interrupts with two levels of priority
On-chip clock oscillator

IV. Pin description :

Fig 2 shows the MCS-51 Pin diagram. Various signals available at the 40 pins can be classified under the following categories:

1. Port 0
2. Port 1
3. Port 2
4. Port 3
5. Bus-control signals
6. On-chip oscillator signals
7. Power supply signals

4 copy

4. MCS-51 Addressing modes and Instruction Set

4.1 Introduction

All MCS-51 family members execute the *same* instruction set, which is designed to suit *8-bit control applications* [14.a]. The instruction set is supported by 111 instructions (64 single-cycle) [14.c]. The boolean processing (*single-bit*) facilities, thus treating one-bit variables as a separate data entity, is typical of microcontroller instruction set. This allows direct bit manipulation in control and switching (logic) applications [14.a]. MCS-51 instruction set provides various addressing modes, of which *bit addressing* is unique to microcontrollers. This chapter begins with a discussion on 'Addressing modes' followed by 'Instruction Set'. Few sample programs are also included at the end of this chapter.

4.2 Addressing modes

Different addressing modes of MCS-51 instruction set, are as follows [3] :

1. Direct addressing

The 8-bit address of the operand is given in the instruction.

(e.g.) MOV A, direct	(The content of memory location 'direct', is moved to accumulator)

Only internal (on-chip) data RAM and special function registers(SFR) can be directly addressed.

2. Register Direct addressing

(e.g.) ADD A, Rn	(The content of register 'Rn' is added with the content of accumulator and the result is stored in the accumulator)

The register Rn (corresponds to R0-R7) is pointed in the current register bank. Here, one of the operand is taken from the registers, R0 through R7.

678162

3. Register Indirect addressing

This mode points towards internal data RAM, addressed indirectly through Ri (corresponds to R0 or R1).

(e.g.) MOV A, @Ri	(The content of internal data RAM pointed at by R0 or R1 is transferred to accumulator)

4. Immediàte addressing

The 8- or 16-bit operand is the part of the instruction itself.

(e.g.) ORL A,# data	(The 8-bit data given in the instruction is logically 'OR'ed with the content of accumulator and the result is stored in the accumulator).

5. Absolute Near addressing

This mode is used by 'ACALL' and 'AJMP' instructions. The 11-bit absolute address of the 16-bit call/jump location is provided in the given 2-byte instruction. By *concatenating* the *binary format* of the content of upper 5-bits of program counter (PC), upper 3-bits and lower 8-bits of the given ACALL/AJMP instruction, the 16-bit call/jump location is obtained. (Details of this are given under 'Unconditional Jump' of program branching instructions, in Section 4.3).

6. Relative addressing

An 8-bit *signed offset* is provided in the instruction to specify the jump location. This mode is used by short jump-SJMP and conditional jump instructions. The range of the jump is -128 to +127 memory bytes relative to the *first* instruction following the SJMP/conditional jump instructions.

7. Bit addressing

This addressing mode is used to access any *'bit addresable locations'* in the internal data RAM or SFR.

(e.g.) SETB 20H (Set the bit with address 20H in the internal data RAM)

8. Indexed addressing

This addressing mode, which can access only *program memory* is meant for reading *lookup tables*. Either data pointer (DPTR) or program counter (PC) is used as a 16-bit *base* register. The accumulator must be provided with the *table entry-number*. By adding the contents of accumulator and base pointer, the *address* of table entry-number is obtained [14.a]. (The two instructions used for reading lookup tables is given in Table 4.3.C. of Section 4.3). Another use ofthis addressing mode is in computing the destination address as the sum of *base* register and accumulator contents, called 'Case Jump'. (Details of this are given under 'Unconditional Jump' of Program branching instructions, in Section 4.3).

4.3 Instruction Set

The various instructions in the MCS-51 instruction set can be classified under the following five categories :

1. Arithmatic instructions
2. Logical instructions
3. Data transfer instructions
4. Boolean instructions
5. Program branching instructions.

The menu and the description of instructions under each category is given below.

1. ARITHMATIC INSTRUCTIONS

The arithmatic instructions are listed in Table 4.1 [14.a]. The different addressing modes available to access the operand '<byte>' in each instruction, is also shown in the table. For ADD A, <byte> instruction, the operand <byte> may reside in the internal data RAM. Then, it can be accessed by *direct* addressing, as

```
ADD   A,30H
```

The operand may reside in any one of the registers R0-R7 in the selected register bank. By *register direct* addressing, the operand can then be obtained, as given below :

```
ADD   A,R5
```

The <byte> operand, which is available in the internal data RAM can also be obtained through *register indirect* addressing, as given below :

```
ADD   A,@R1
```

Table 4.1 : **MCS-51 Arithmatic instructions** [14.a]

Mnemonic		Operation		Addressing modes		
			Dir.	R.D.	R.I.	Imm.
ADD	A,<byte>	A = A + <byte>	✓	✓	✓	✓
ADDC	A,<byte>	A = A + <byte> + C	✓	✓	✓	✓
SUBB	A,,byte>	A = A - <byte> - C	✓	✓	✓	✓
INC	A	A = A + 1		accumulator only		
INC	<byte>	<byte> = <byte> + 1	✓	✓	✓	X
INC	DPTR	DPTR = DPTR + 1		data pointer only		
DEC	A	A = A - 1		accumulator only		
DEC	<byte>	<byte> = <byte> - 1	✓	✓	✓	X
MUL	AB	A x B : (B) = high byte		A(acc) and B only		
		(A) = low-byte				
DIV	AB	A/B : (A) = quot., (B) = rema.		A(acc) and B only		
DA	A	Decimal Adjust		accumulator only		

In this case, the register R1 must hold the address of the internal data RAM, where the operand <byte> is available. Or the operand <byte> may be the *data* itself, accessed by *immediate* addressing, as

ADD A, #OFH

Increment or decrement operations can be done on any byte in the internal data RAM *directly*, without the need to go through the accumulator.

Data pointer (DPTR) holds the 16-bit address of the external memory and access to external memory is through DPTR *only*. Increment operation can be done on DPTR also.

MUL AB instruction multiplies the 8-bit contents of accumulator and B register and leaves the high-byte of the result in B register, low-byte in accumulator.

DIV AB instruction divides the 8-bit contents of accumulator by the 8-bit contents of B register and leaves the quotient and remainder in accumulator and B register, respectively.

During BCD arithmatic, the addition instructions (ADD & ADDC) must be followed by decimal adjust (DA) operation, thus ensuring that the result is also in BCD. DA A does not convert a binary number to BCD.

2. LOGICAL INSTRUCTIONS

Table 4.2 [14.a] shows the MCS-51 logical instructions. Any boolean operations, such as AND, OR, EX-OR, NOT on bytes are performed on a bit-by-bit basis. For example, ORing the data 73H (0111 0011B) with the accumulator content 27H (0010 0111B) will leave the result 77H (0111 0111B) in the accumulator.

For ANL A,<byte> instruction, the <byte> operand can be accessed by *any* one of the addressing modes, as given below :

ANL	A,30H	(Dir - direct addressing)
ANL	A,R5	(R.D. - register direct)
ANL	A,@R1	(R.I. - register indirect)
ANL	A,#OFH	(Imm. - immediate addressing)

Boolean operation can be performed with immediate constant using direct addressing, without the need to use accumulator. Otherwise, previous accumulator contents are to be stacked, for any such operation, thus saving the time.

(e.g.) ANL 30H,#2CH	(The content of internal data RAM with address 30H is logically ANDed with the data 2CH and the result is with 30H).

Accumulator contents can be shifted to left/right by 1 bit, also through carry by *rotate* instructions.

Table 4.2 : **MCS-51 Logical instructions** [14.a]

Mnemonic		Operation	Addressing modes Dir.	R.D.	R.I.	Imm.
ANL	A,<byte>	A = A AND <byte>	✓	✓	✓	✓
ANL	<byte>,A	<byte> = <byte> AND A	✓	X	X	X
ANL	<byte>,#data	<byte> = <byte> AND data	✓	X	X	X
ORL	A,<byte>	A = A OR <byte>	✓	✓	✓	✓
ORL	<byte>,A	<byte> = <byte> OR A	✓	X	X	X
ORL	<byte>,#data	<byte> = <byte> OR data	✓	X	X	X
XRL	A,<byte>	A = A XOR <byte>	✓	✓	✓	✓
XRL	<byte>, A	<byte> = <byte> XOR A	✓	X	X	X
XRL	<byte>,#data	<byte> = <byte> XOR data	✓	X	X	X
CRL	A	A = 00H		accumulator only		
CPL	A	$A = \overline{A}$		accumulator only		
RL	A	Rotate acc left by 1 bit		accumulator only		
RLC	A	Rotate acc left through carry		accumulator only		
RR	A	Rotate acc right by 1 bit		accumulator only		
RRC	A	Rotate acc right through carry		accumulator only		
SWAP	A	Swap nibbles in $A(A_{3-0} \leftrightarrow A_{7-4})$		accumulator only		

The high and low nibbles of accumulator gets *interchanged* after the execution of SWAP A instruction. This instruction can be conveniently used during BCD manipulation.

3. DATA TRANSFER INSTRUCTIONS

Table 4.3.A [14.a] shows the instructions available for moving data around within internal RAM, and also the addressing modes for each instructions.

Data can be transferred between any two internal RAM or SFR locations by using MOV <dest>, <Src> instruction, *without* the need to go through the accumulator. (In 8052, the upper 128 bytes of data RAM can be accessed only by *indirect* addressing, and SFR space only by *direct* addressing).

The MOV DPTR, #data 16, instruction initializes the data pointer with 16-bit address for external (program/data) memory access.

For all MCS-51 devices, the stack is with on-chip RAM and growing upwardly. During PUSH execution, the stack pointer (SP) is first incremented and then the byte is copied into the stack. During POP execution, first the stack pointer (SP) is decremented and the byte is copied from the stack location into destination address. To identify the byte being saved and retrieved, PUSH and POP use only direct addressing. But the stack itself is identified using SP by *indirect* addressing. It implies that the stack can go into the upper 128 bytes, when implemented as in 8052, but *not* into the SFR space. For the devices 8031 and 8051, which do not implement upper 128 bytes, if the SP points to the upper 128 byte, then the bytes that are PUSHed will be *lost* and the POPed (retrieved) bytes will be *indeterminate*.

Table 4.3.A : **MCS-51 Data transfer instructions that access internal data RAM** [14.a]

Mnemonic		Operation	Addressing modes			
			Dir.	R.D.	R.I.	Imm.
MOV	A,<Src>	A = <Src>	✓	✓	✓	✓
MOV	<dest>,A	<dest> = A	✓	✓	✓	X
MOV	<dest>,<Src>	<dest> = <Src>	✓	✓	✓	✓
MOV	DPTR,#data 16	DPTR = 16-bit immediate constant	X	X	X	✓
PUSH	<Src>	INC SP, MOV "@SP", <Src>	✓	X	X	X
POP	<dest>	DEC SP, MOV <dest>, @SP"	✓	X	X	X
XCH	A,<byte>	ACC and <byte> exchange	✓	✓	✓	X
XCHD	A, @Ri	ACC and @Ri exchange low nibbles	X	X	✓	X

The accumulator contents and addressed byte got their data *exchanged*, after the execution of XCH A,<byte> instruction. Only the *lower nibbles* can be exchanged with XCHD A,@Ri instruction.

External RAM/ROM

Table 4.3.B [14.a] shows the data transfer instructions that access external (program/data) memory space. The external, one-byte address can be accessed indirectly through Ri (R0 or R1), while the two-byte is through data pointer (@DPTR). In case of two-byte external addressing, the entire Port 0 and 2 are to be used for the memory access. For any external access, either the source or the destination of the data is *accumulator only*.

Lookup Tables

Table 4.3.C. shows the two instructions available for reading lookup tables in progam memory, using *indexed* addressing. As the access is with program memory, the lookup tables can be read but cannot be updated. The mnemonic MOVC stands for 'Move Constant'. (Lookup table read operation is given in Section 4.2 under 'Indexed addressing').

Table 4.3.B : **MCS-51 Data transfer instructions that access external RAM/ROM** [14.a]

Address width	Mnemonic		Operation	Addressing modes			
				Dir.	R.D.	R.I.	Imm.
8-bits	MOVX	A,@Ri	Read external RAM/ROM @Ri	X	X	With RO/RI only	X
8-bits	MOVX	@Ri,A	Write external RAM @Ri	X	X		X
16-bits	MOVX	A,@DPTR	Read external RAM/ROM @DPTR	X	X	with DPTR only	X
16-bits	MOVX	@DPTR,A	Write external RAM @DPTR	X	X		X

Table 4.3.C : **MCS-51 Data transfer instruction :**
Lookup table read instructions [14.a]

MOVC	A,@A+DPTR	Read program memory at (A + DPTR)
MOVC	A,@A+PC	Read program memory at (A + PC)

4. BOOLEAN INSTRUCTIONS

All MCS-51 devices incorporates *boolean* (single-bit) *processor*. A complete and compact boolean instruction set provides move, clear, set, complement along with conditional jump instructions. Bit accessing is unique to microcontroller architecture. There are 128 bit addressable locations in the internal data RAM in the address range 20H to 7FH. There are 88 (128 in 8052/8032) more bit addressable locations in the SFR area between the addresses 80H-FFH. All the bit access is through *direct* addressing.

Table 4.4 [14-a] shows the Bolean instruction set of MCS - 51.

All the bits including the carry bit in the program status word (PSW) is bit addressable, and CY acts as the Boolean Accumulator. The given bit addressable location,also the complement of it, can be logically ANDed with the carry bit. Similarly OR logic operation also can be performed. As Ex-OR operation can be implemented by *software*, separate EX-OR menu is not included.

Move opeation can be performed between the carry bit and any other bit addressable locations.

Carry bit can be cleared, set as well as complemented. Complement operation can be performed on any other bit addressable locations also.

Table 4.4 : **MCS-51 Boolean Instructions** [14-a]

Mnemonic	Operation
ANL C,bit	C = C. AND. bit
ANL C, /bit	C = C. AND $\overline{\text{bit}}$
ORL C, bit	C = C OR bit
ORL C, /bit	C = C. OR $\overline{\text{bit}}$
MOV C, bit	C = bit
MOV bit, C	bit = C
CLR C	C = 0
SETB C	C = 1
CPL C	C = $\overline{\text{C}}$
CPL bit	bit = $\overline{\text{bit}}$
JC, rel	Jump if C = 1
JNC, rel	Jump if C = 0
JB bit, rel	Jump if bit = 1
JNB bit, rel	Jump if bit = 0
JBC bit, rel	Jump if bit = 1; CLR bit

There are jump instruction, which execute a jump, if carry/bit addressable location is set (JC/JB). Jump can also be implemented when carry/bit addressable location is reset (JNC/JNB). 'JBC' jumps to the address if bit is set and before making a jump, it clears the bit.

5. PROGRAM BRANCHING INSTRUCTIONS

Table 4.5 A [14-a] shows the program branching instruction, that executes a jump *unconditionally*.

Table 4.5 A : **Program branching : Unconditional Jump instructions** [14-a]

Mnemonic	Operation
SJMP rel	Short Jump (relative address)
LJMP addr 16	Long Jump
AJMP addr 11	Absolute Jump
JMP @ A + DPTR	Jump to (A+DPTR)

Unconditional Jump

For short jump instruction - SJMP, the jump distance is limited to a range of - 128 to +127 memory bytes *relative* to the instruction following it. SJMP instruction is 2 bytes wide, consisting of opcode and *relative offset byte*. The relative offest byte, given in signed magnitude (two's complement) form represents the jump distance, which is added to the PC in two's complement arithmatic in order to calculate the destination jump address, during the execution of SJMP. (For such instructions, when *assembler* is available, it is suffice to specify the destination address as a lable or as 8-bit/16-bit constants. The assembler will take care of inserting the destination address in the correct format for the given instruction)

The long jump - LJMP instruction is 3 bytes wide, consisting of opcode and 16-bit destination address as operand, which can be anywhere in the 64k program memory area.

The absolute jump - AJMP instruction incorporates lower 11-bits of the 16-bit destination address. The AJMP instruction is 2 bytes wide, consisting of opcode, which itself contains 3 of the 11-address bits, followed by another byte which is the lower 8-bits of the destination address, as mentioned below :

Let the 2-bytes long AJMP, add11 be written as four nibbles

AJMP, add11 == YY XX ....(4.1)

Writing the four nibbles, bit-wise,

$Y_7\,Y_6\,Y_5\,Y_4 \quad Y_3\,Y_2\,Y_1\,Y_0 \quad X_7X_6X_5X_4 \quad X_3\,X_2X_1X_0$ .... (4.2)

Then, the lower 11-address bits of the 16-bit jump location is obtained by concatenating the upper 3-bits and lower 8-bits of equ (4.2), as given below.

$$\begin{array}{ccccccccccc} \mathbf{A_{10}} & \mathbf{A_9} & \mathbf{A_8} & \mathbf{A_7} & \mathbf{A_6} & \mathbf{A_5} & \mathbf{A_4} & \mathbf{A_3} & \mathbf{A_2} & \mathbf{A_1} & \mathbf{A_0} \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ Y_7 & Y_6 & Y_5 & X_7 & X_6 & X_5 & X_4 & X_3 & X_2 & X_1 & X_0 \end{array} \qquad \text{........ (4.3)}$$

When the AJMP, add11 instruction is executed, these 11-bits are simply substituted for low 11-bits in the *PC*. The high 5-bits of the PC remains the same, as given below

$$\begin{array}{cccc@{\quad}cccc@{\quad}cccc@{\quad}cccc}
\mathbf{A_{15}} & \mathbf{A_{14}} & \mathbf{A_{13}} & \mathbf{A_{12}} & \mathbf{A_{11}} & \mathbf{A_{10}} & \mathbf{A_{9}} & \mathbf{A_{8}} & \mathbf{A_{7}} & \mathbf{A_{6}} & \mathbf{A_{5}} & \mathbf{A_{4}} & \mathbf{A_{3}} & \mathbf{A_{2}} & \mathbf{A_{1}} & \mathbf{A_{0}} \\
\downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
PC_7 & PC_6 & PC_5 & PC_4 & PC_3 & Y_7 & Y_6 & Y_5 & X_7 & X_6 & X_5 & X_4 & X_3 & X_2 & X_1 & X_0
\end{array} \quad \ldots\ldots (4.4)$$

By this way, the complete 16-bit addres of the jump location is obtained [3]. As the high 5-bits are always the same, the destination jump location must be within the same 2K block of the instruction following the AJMP [14.a].

As the three bits (Y_7 Y_6 Y_5), from the opcode, can have $2^3 = 8$ different combinations from (0 0 0) to (1 1 1), the MCS - 51 instruction set is supplied with eight AJMP instructions with opcodes O1H, 2IH, 4IH, 6IH, 8IH, AIH, CIH, EIH [2]. Thus, depends on the *page* of the jump location required, an AJMP instruction with appropriate opcode can be chosen.

With JMP @ A+DPTR, the destination address is computed as the sum of accumulator and DPTR contents, an example of *case jump*.

Unconditional Call

Table 4.5 B shows the MCS-51 unconditional call and return instruction.

Table 4.5 B : **Program branching : Unconditional Call and Return Instructions** [14.a]

Mnemonic	Operation
LCALL addr 16	Long Subroutine Call
ACALL addr 11	Absolute Subroutine Call
RET	Return from Subroutine
RETI	Return from Interrupt

The long call-LCALL instruction is 3-bytes long with 16-bit address as the operand and the subroutine can be anywhere in the 64k program memory area. The absolute call-ACALL instruction is 2-bytes long, which contains 11-bits of the 16-bit call location and the subroutine must be in the same 2K block of the instruction, following the ACALL. As in the case of AJMP instructions, depends an the *page* of the call location, appropriate ACALL instruction can be chosen among the eight ACALL instructions with the opcodes 11H, 3IH, 5IH, 7IH, 9IH, B1H , DIH and FIH.

To return the execution to the main program after a call, subroutines must end with RET instruction. RETI is used during the return from the interrupt service routine.

Conditional Jump

Table 4.5.C [14.a] shows the MCS-51 Conditional Jump instructions. In all the cases, the destination address is limited to the jump distance of -128 to +127 memory bytes

Table 4.5.C : **Program Branching : Conditional Jump** [14.a]

Mnemonic	Operation	Addressing modes Dir.	R.D.	R.I.	Imm.
JZ rel	Jump if A = O		Accumulator only		
JNZ rel	Jump if A ≠ O		Accumulator only		
DJNZ <byte>, rel	Decrement and jump, if not Zero	✓	✓	X	X
CJNE A,<byte> rel	Jump if A ≠ <byte>	✓	X	X	✓
CJNE <byte>, # data, rel	Jump if <byte> ≠ # data	✓	X	X	✓

NOP	No Operation

relative to the first byte, following the conditional Jump instructions, which is represented as a *relative offset byte.*

As there is no 0 bit in the program status word (PSW), the JZ and JNZ instructions check the accumulator data for that conditions.

The Decrement and Jump, if not Zero - DJNZ instruction is meant for loop control, and the compare and jump, if not Zero - CJNZ instruction also can be used for it.

4.3 Sample Programs

1(a) Write a program to add the content of any two registers in the internal data RAM amd store the result at 30H, in the internal data RAM (b) change the register bank and repeat the same task [3].

Program :

```
1(a)    MOV  RO, # 05H
        MOV  R1, # 01H
        MOV  A, RO
        ADD  A, R1
        MOV  30H, A
LPO :   SJMP LPO          (Halt)
```

Note : By default, register bank O is always selected.

```
1(b)   SETB  D3
       SETB  D4              (Register Bank 3 is selected)
       MOV   RO # 05H
       MOV   R1 # O1H
       MOV   A, RO
       ADD   A, R1
       MOV   30H, A
LPI :  SJMP  LP1
```

2. Write a program to subtract the contents of internal data RAM location 40H from 41H using register *indirect addressing* [3].

Program :

Let the content of 40H and 41H be 01H and 05H, respectively.

i.e (40H) = 01H & (41H) = 05H.

```
       CLR   C
       MOV   R1, # 40H
       MOV   R0, # 41H
       MOV   A, @ R0   → R1
       SUBB  A, @ R1   → RO
       MOV   42H, A
LP2 :  SJMP  LP2
```

The result of the subtraction is stroed in 42H, in the internal data memory.

Note : SUBB implies subtract with borrow. When borrow is not required during the subtraction process, *clear* the carry flag, beforehand.

3. Write a program to add the datas 10H and 20H, using *immediate* addressing [3]

Program :

```
        10H + 20H

        MOV A, # 10H

        ADD A, # 20H

LP3 :   SJMP  LP3
```

4. Write a program to (a) multiply any two 8-bit datas (b) divide any two 8-bit data [3].

Program :

```
4(a)   O5H x 02H

       MOV  A, # 05H

       MOV  addrB, # 02H

       MUL  AB

LP41 : SJMP  LP41


4(b)   04H / 02H
       MOV  A, # 04H
       MOV  addrB, # 02H
       DIV  AB
LP42 : SJMP  LP42
```