UNIT - III
ASSEMBLER AND ADDRESSING MODES

ASSEMBLING AND RUNNING AN 8051 PROGRAM

1) First we use an editor to type a program,many excellent editors or word processorsare available that can be used to createand/or edit the program

- Notice that the editor must be able toproducean ASCII file
- For many assemblers, the file names followthe usual DOS conventions, but thesourcefile has the extension “asm“ or “src”,depending on which assembly you are using

2) The “asm” source file containing the program codecreated in step 1 is fed to an 8051assembler

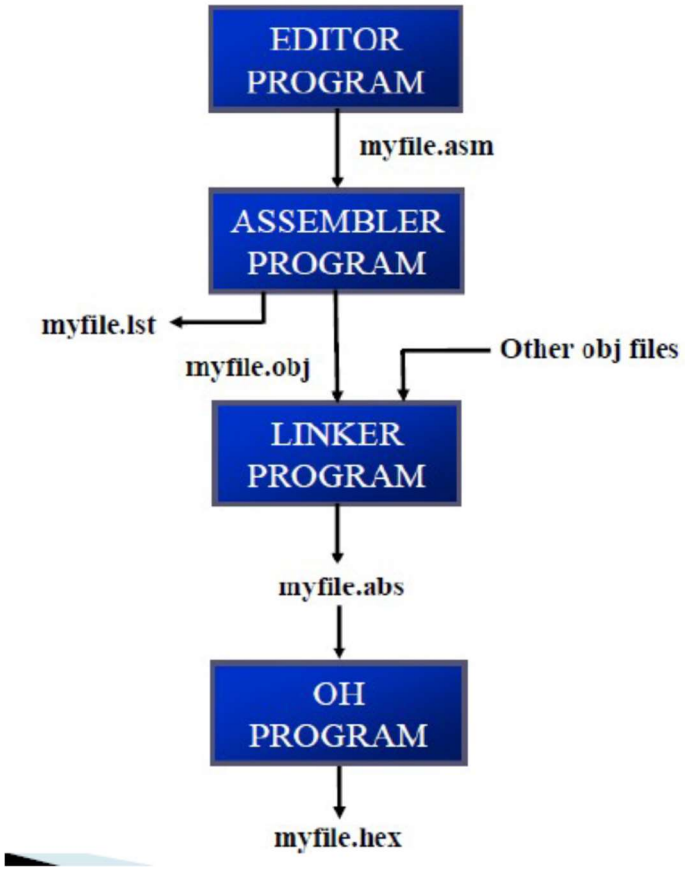
- The assembler converts the instructions into machinecode
- The assembler will produce an object file and a list file
- The extension for the object file is “obj” while the extension for the list file is “lst”

3) Assembler require a third step called linking

- The linker program takes one or more object code filesand produce an absolute object filewith the extension“abs”
- This abs file is used by 8051 trainers that have amonitor program

4) Next the “abs” file is fed into a programcalled “OH” (object to hex converter) whichcreates a file with extension “hex” that isready to burn into ROM

- This program comes with all 8051assemblers
- Recent Windows-based assemblers combinestep 2 through 4 into one step



STRUCTURE OF ASSEMBLY LANGUAGE PROGRAM

An instruction may be represented on a line of maximum 128 characters, the general form being:

[<label>:] [<opcode>[<operatives>] [;<comments>]]

Where:

<label> is a name, maximum 31 characters (letters, numbers or special characters _? @,), the first character being a letter or one of the special characters. Each label has a value attached and also a relative address in the segment where it belongs to.

<opcode> the mnemonic of the instruction.

<operatives> the operative (or operatives) associated with the instruction concordant to the syntax required for the instruction. It may be a constant, a symbol or expressions containing these.

<comments> a certain text forego of the character ";". comment is optional. It is for readability

ASSEMBLER DIRECTIVES

- It is a pseudo instruction.
- It is not an executable.
- It gives directions to Assembler

Some examples of Assembler Directives

ORG, END, EQU, DB, DW, DATA

ORG:

It gives direction to Assembler that the program should be started at the Address following ORG.

Ex. ORG 8000h

END:

It gives direction to Assembler that the program ends at that point

Ex. END

EQU:

It is used to give direction to Assembler to assign some value to some variable.

Ex. EQU PI 3.14

DB: Define Byte

It directs the Assembler that the number following this DB is byte

Ex. DB 39h

DB 00110101b

DATA:

It gives direction to Assembler that the numbers following DATA are the data

Ex. DATA 32, 43,65,23,01

DIFFERENT ADDRESSING MODES

The method of specifying the data in instruction is called Addressing

Types of Addressing Modes

1. Register Addressing
2. Indirect Addressing
3. Direct Addressing
4. Immediate Addressing
5. Index Addressing

Register Addressing

In this Addressing mode, the registers (R0…R7, A, B, DPTR, CARRY) are used as operands. R0…R7 can be selected in any one of four modes. The modes can be selected in PSW register.

Ex.
MOV A, R1
In this mode, the content of R1 is moved to A
ADD A, R3
In this mode, the content of R3 is added with the content of A
ANL A, R1
In this mode, the content of R1 is AND immediate with the content of A

Indirect Addressing

In this Addressing mode, the operand's address is specified in register R0,R1or DPTR To access the address register, the symbol '@' is preceded with above register.
Both internal and external RAM can be accessed in this mode. 8 bit address is specified in R0 an R1 registers. 16 bit address is specified in DPTR
Ex.
MOV A,@R1
In this mode, the content of address specified in R1 is moved to A
ADD A,@R0
In this mode, the content of address specified in R0 is added with the content of A
MOVX A,@DPTR
In this mode, the content of external memory address specified in DPTR is moved to A.
'X' in the MOVX represent External memory.

Direct Addressing

In this mode of addressing, 8 bit address of operand is specified in instruction. Address of internal RAM is specified.
Ex.
MOV A,34h
In this mode, the content of address 34h is moved to A
ADD A,23h
In this mode, the content of address 23h is added with the content of A
ANL A,45h
In this mode, the content of address 45h is AND immediate with the content of A

Immediate Addressing

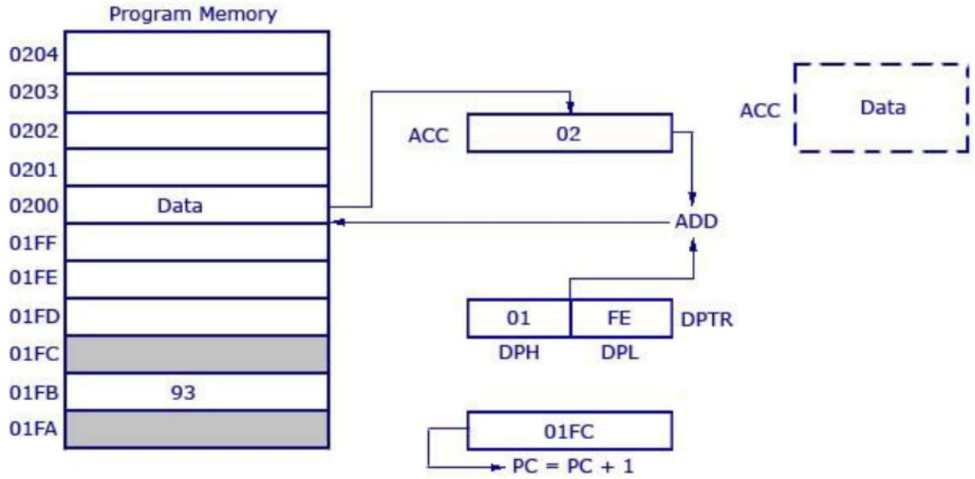
In this addressing mode, the operand is data. The actual data is specified in the instruction itself. The symbol '#' is proceeded with data. The data is accessed immediately.
Ex.
MOV A, #67h
In this mode, the hexadecimal data 67 is moved to A
ADD A, #89h
In this mode, the hexadecimal data 89 is added with the content of A
ANL A, #91
In this mode the hexadecimal data 91 is AND immediate with the content of A

Index Addressing

This addressing mode is used access lookup table in program memory.
Ex.
MOVC A, @A+DPTR

Indexed Addressing Mode

Instruction	Opcode	Bytes	Cycles
MOVC A,@A +DPTR	93H	1	2



PROGRAMS

MULTIBYTE

Multi byte numbers like 8254h and 65f3h may be added.

i) Add f3 and 54 using ADD and store the result in memory location

ii) Add 82 and 65 using ADDC and store the result in another location

Label	Mnemonics	Comments
	MOV DPTR ,#8200H	memory address is loaded to store result
	MOV R1,#00H	R1 is initialized .It is used to store carry
	MOV A,#F3H	LSB of second data is moved to A
	ADD A,#54H	LSB of first data is added with A
	MOVX @DPTR,A	added value in A is moved to memory
	INC DPTR	DPTR is incremented
	MOV A,#65H	MSB of second data is moved to A
	ADDC A,#82H	MSB of first data is added with carry in A
	MOVX @DPTR,A	added value is moved to memory
	INC DPTR	DPTR is incremented to store 1 if carry
	JNC NEXT	if carry, jump to labeled NEXT
	INC R1	R1 is incremented if carry available
NEXT	MOV A,R1	R1 value is moved to A
	MOVX @DPTR,A	value in A is moved to memory
HLT	SJMP HLT	

8 BIT MULTIPLICATION

Assume that 8 bit data are available in memory address 8400h and 8401h and the result is to be stored in 8402h, 8403h

Label	Mnemonics	Comments
	MOV DPTR,#8400H	DPTR is initialized
	MOVX A,@DPTR	data in memory address 8400 is moved to A
	MOV B,A	value in A is moved to B
	INC DPTR	DPTR is incremented
	MOVX A,@DPTR	next data in address 8401 is moved to A
	MUL AB	both data are multiplied
	INC DPTR	DPTR is incremented
	MOVX @DPTR,A	low byte answer is moved to memory
	MOV A,B	high byte answer in B is moved to A
	INC DPTR	DPTR incremented
	MOVX @DPTR,A	high answer in A is moved to memory
HLT	SJMP HLT	

8 bit DIVISION

Assume that 8 bit data (denominator) is available in memory address 8400h and data (Numerator) in 8401h and the quotient is to be stored in 8402h and remainder is to be stored in 8403h

Label	Mnemonics	Comments
	MOV DPTR,#8400H	DPTR is initialized
	MOVX A,@DPTR	data(divisor) in memory address 8400 is moved toA
	MOV B,A	value in A is moved to B
	INC DPTR	DPTR is incremented
	MOVX A,@DPTR	next data(dividend) in address 8401 is moved to A
	DIV AB	data in A is divided by data in B
	INC DPTR	DPTR is incremented
	MOVX @DPTR,A	answer quotient is moved to memory
	MOV A,B	answer remainder in B is moved to A
	INC DPTR	DPTR incremented
	MOVX @DPTR,A	answer remainder in A is moved to memory
HLT	SJMP HLT	

BIGGEST NUMBER

Assume that the data to be arranged are available in array which starts from 8401h and the array length is available in 8400h.

The result is to be stored in 8500h

Label	Mnemonics	Comments
	MOV B, #00H	to hold the biggest number, B is initialised
	MOV DPTR ,#8400H	DPTR initialized with8400
	MOVX A, @DPTR	array length in 8400 in moved to A
	MOV R0,A	value (array length) in A is copied into R0
AGAIN	INC DPTR	DPTR incremented
	MOVX A, @DPTR	data in memory is moved to A
	CJNE A,B , NEXT	A and B compared. Carry will be generated if B is bigger
NEXT	JC L1	if carry ,jump to label L1
	MOV B,A	if no carry, move the value in A to B

L1	DJNZ R0, AGAIN	array length is decremented and jump to label AGAIN till array length is 0
	MOV DPTR,#8500H	DPTR is initialized with 8500 to store bigger value in B
	MOV A,B	value in B (bigger value) is moved to A
	MOVX @DPTR,A	this value (bigger value) is moved to memory addressed by DPTR(8500)
HLT	SJMP HLT	

SMALLEST NUMBER

Assume that the data are available in array which starts from 8401h and the array length is available in 8400h.The result is to be stored in 8500h

Label	Mnemonics	Comments
	MOV B, #00H	to hold the biggest number, B is initialised
	MOV DPTR ,#8400H	DPTR initialized with8400
	MOVX A, @DPTR	array length in 8400 in moved to A
	MOV R0,A	value(array length) in A is copied into R0
AGAIN	INC DPTR	DPTR incremented
	MOVX A, @DPTR	data in memory is moved to A
	CJNE A,B , NEXT	A and B compared. Carry will be generated if B is bigger
NEXT	JC L1	if carry ,jump to label L1
	MOV B,A	if no carry, move the value in A to B
L1	DJNZ R0, AGAIN	array length is decremented and jump to label AGAIN till array length is 0
	MOV DPTR,#8500H	DPTR is initialized with 8500 to store bigger value in B
	MOV A,B	value in B (bigger value) is moved to A
	MOVX @DPTR,A	this value (bigger value) is moved to memory addressed by DPTR(8500)
HLT	SJMP HLT	

ASCENDING ORDER

Assume that the data to be arranged are available in array which starts from 8401h and the array length is assumed as 09 .

Label	Mnemonics	Comments
	MOV R0, #08H	array length 08h(09-01) is stored
AGAIN	MOV A, R0	08h is moved to R1
	MOV R1, A	Data moved to R1
	MOV DPTR, #8401H	DPTR is initialized with 8401h
BACK	PUSH DPH	84 is saved in stack
	PUSH DPL	01 is saved in stack
	MOVX A, @DPTR	first data is moved to A
	MOV B, A	this data is copied into B
	INC DPTR	DPTR incremented
	MOVX A, @DPTR	second data is moved to A
	CJNE A,B, LOOP	first data in B and next data in A are compared. Carry will generate if B value is bigger

LOOP	JNC NEXT	if no carry(second data is bigger)instruction labeled NEXT will be executed
	POP DPL	01 from stack is moved to DPL
	POP DPH	84 from stack is moved to DPH
	MOVX @DPTR, A	second data is moved is moved to first location
	INC DPTR	DPTR incremented
	MOV A ,B	first data in B is moved to A
	MOVX @DPTR,A	this first data is moved to second location
NEXT	DJNZ R1, BACK	jump for next two data comparison
	DJNZ R0, AGAIN	jump for next scan
HLT	SJMP HLT	stay at here

DESCENDING ORDER

Assume that the data to be arranged are available in array which starts from 8401h and the array length is assumed as 09 .

Label	Mnemonics	Comments
	MOV R0, #08H	array length 08h(09-01) is stored
AGAIN	MOV A, R0	08h is moved to R1
	MOV R1, A	Data moved to R1
	MOV DPTR, #8401H	DPTR is initialized with 8401h
BACK	PUSH DPH	84 is saved in stack
	PUSH DPL	01 is saved in stack
	MOVX A, @DPTR	first data is moved to A
	MOV B, A	this data is copied into B
	INC DPTR	DPTR incremented
	MOVX A, @DPTR	second data is moved to A
	CJNE A,B, LOOP	first data in B and next data in A are compared. Carry will generate if B value is bigger
LOOP	JC NEXT	if carry(second data is smaller)instruction labeled NEXT will be executed
	POP DPL	01 from stack is moved to DPL
	POP DPH	84 from stack is moved to DPH
	MOVX @DPTR, A	second data is moved is moved to first location
	INC DPTR	DPTR incremented
	MOV A ,B	first data in B is moved to A
	MOVX @DPTR,A	this first data is moved to second location
NEXT	DJNZ R1, BACK	jump for next two data comparison
	DJNZ R0, AGAIN	jump for next scan
HLT	SJMP HLT	stay at here

BCD TO ASCII CONVERSION

To covert BCD number, each digit is separately considered and equivalent ASCII value is generated.

Ex. To convert 65, 5is converted into ASCII value 35 and 6 is converted into ASCII 36.

Assume that the BCD value 65 is stored in memory location 8400h and ASCII values are stored in 8401, 8402.

Label	Mnemonics	Comments
	MOV DPTR,#8400H	DPTR is initialized with 8400h
	MOV XA, @DPTR	First Data Is Moved To A
	MOV R1,A	Data moved to R1
	ANL A,#0FH	Get the Lower data
	ORL A,#30H	OR logic for 30h
	INC DPTR	DPTR incremented
	MOVX @DPTR,A	Second Data Is Moved To A
	INC DPTR	DPTR incremented
	MOV A,R1	Data moved to R1
	ANL A,#F0H	Get the Upper data
	SWAP A	Move the lower value
	ORL A,#30H	OR logic for 30h
	MOVX @DPTR,A	Store the Result
HLT	SJMP HLT	stay at here

ASCII TO BINARY CONVERSION

Assume that ASCII value is stored in 8400h and answer to be stored in 8401h

Label	Mnemonics	Comments
	MOV DPTR, #8400H	DPTR is initialized with 8400h
	MOV A, @DPTR	First Data Is Moved To A
	MOV R1, A	Data moved to R1
	CJNE A,#40H, NEXT	first data in 40h and next data in A are compared. Carry will generate if B value is bigger
NEXT:	JC LOOP	if carry(second data is smaller)instruction labeled NEXT will be executed
	CLR C	Clear carry flag
	SUBB A, #07H	Subtract the value Data from 07 h
LOOP:	CLR C	Clear carry flag
	SUBB A, #30H	Subtract the value Data from 30 h
	INC DPTR	DPTR incremented
	MOVX @DPTR ,A	Store the Result
HLT	SJMP HLT	stay at here

ODD PARITY GENERATOR

Odd parity means that the total number of 1's in data as well as in parity bit is ODD. Assume that the data 42H for which the parity is to be generated and the result is to be Stored in 8400h.

The data 42h has 2 nos. of '1's. So one bit is generated to make ODD parity.

Label	Mnemonics	Comments
	MOV DPTR, #8400H	memory address is loaded in DPTR
	MOV R1,#08	to rotate the data 8times,R1 is initialised
	MOV R2,#00	counter initialized to count no.of '1's
	MOV A,#42h	the data 42h is stored in A
BACK:	RRC A	the data is rotated
	JNC XXX	if not carry, jump to label XXX
	INC R2	if carry, R2 is incremented
XXX	DJNZ R1, BACK	8 rotation is checked
	MOV A,R2	no. of 1 s in R2 is moved to A

	MOV B,#02	to test the even ,that should be
	DIV AB	divide by 02
	MOV A,B	the remainder(it is 0 for even no of 1's in data) is Move to A
	CPL A	A value is complemented
	MOVX @DPTR,A,	Complemented value is stored in memory
HLT	SJMP HLT	

EVEN PARITY GENERATOR

Even parity means that the total number of 1's in data as well as in parity bit is even. Assume that the data 42H for which the parity is to be generated and the result is to be Stored in 8400h.The data 42h has 2 nos. of '1's. So NO need of one bit is generated to make even parity.

Label	Mnemonics	Comments
	MOV DPTR, #8400H	memory address is loaded in DPTR
	MOV R1,#08	to rotate the data 8times,R1 is initialised
	MOV R2,#00	counter initialized to count no.of '1's
	MOV A,#42h	the data 42h is stored in A
BACK:	RRC A	the data is rotated
	JNC XXX	if not carry, jump to label XXX
	INC R2	if carry, R2 is incremented
XXX	DJNZ R1, BACK	8 rotation is checked
	MOV A,R2	no. of 1 s in R2 is moved to A
	MOV B,#02	to test the even ,that should be
	DIV AB	divide by 02
	MOV A,B	the remainder(it is 0 for even no of 1's in data) is Move to A
	MOVX @DPTR,A,	Complemented value is stored in memory
HLT	SJMP HLT	stay at here

TIME DELAY ROUTINE

To hold the output or operation of controller, the controller must be forced into delay Routine. This delay may be increased by increasing number of loops.

Delay Routine using single loop

Label	Mnemonics	Comments
	MOV R1,#FFH	value FF is loaded in to R1
	DJNZ R1, WAIT	R1 is decremented till reach to zero
HLT	SJMP HLT	stay at here

Delay Routine using double loop

Label	Mnemonics	Comments
	MOV R1,#FFH	R1 is initialized with FF
LOOP	MOV R2, #FEH	R2 is initialized with FE
WAIT	DJNZ R2, WAIT	R2 is decremented till reach to zero
	DJNZ R1, LOOP	once R2 reach zero, R1 is decremented till to zero and again R2 initialised with FE
HLT	SJMP HLT	