Thakre Sir

T.Y

Electronics

Unit - IV :-

8051 - Programming :

Assembly Language Programming :-

Assembly language is referred as a low-level language because it deals directly with the internal structure of the CPU. To program in Assembly language, the programmer must know all the registers of the CPU, the size of each register, size of internal memory, instruction set, addressing modes, method of flowcharting, etc. An assembly language program is a series of statements, or lines consisting of Assembly language instructions like ADD, MOV or statements called directives. The instructions tells the CPU what to do while directives (also called pseudo-instructions) gives directions to the assembler. ORG and END are directives to the assembler. ORG tells the assembler to place the opcode at memory location 0 while End directive indicates to the assembler the end of the source code. ORG is used at the start of the program and END is used at the end of the program.

Assembly language programs must be translated into machine code by a program called an assembler. A program that consists of 0's and 1's is called machine language which is then processed by CPU. An assembler is used to translate an Assembly language program into machine code. It is also called object code or opcode for operation code. The high-level languages like BASIC, Pascal, C, C++, Java etc are translated into machine code by a program called a compiler.

An assembly language instruction consists of four fields—

label:	mnemonic	operands	comment
	ADD	A, B	Add contents of Reg. B to Reg. A

The label field allows the program to refer to a line of code by name. It has limited no. of characters.

The assembly language mnemonic (instruction) and operand(s) field together perform the real work of the program and completes the task for which the program was written.

A sample of Assembly language program is shown belo

Label	Mnemonic	operand	Comment
	ORG	0H	;Start (origin) at location 0.
	MOV	R5, #25H	; load 25H into R5.
	MOV	R7, #34H	; load 34H into R7.
	MOV	A, #0	; load 0 into A.
	ADD	A, R5	; add contents of R5 to A.
			; now A = A + R5
	ADD	A, R7	; add contents of R7 to A.
			; now A = A + R7
	ADD	A, #12H	; add to A value 12H
			; now A = A + 12H
HERE:	SJMP HERE		; Stay in this loop
	END		; end of asm source file.

The directives do not generate any machine code (opcode) and are used only by the assembler. But, the instructions are translated to machine code (opcode) for the CPU to execute. The comment field begins with ";" semicolon and assembler ignores comments. A label referring to an instruction must be followed by a ":" colon symbol. as show above HERE:.

④

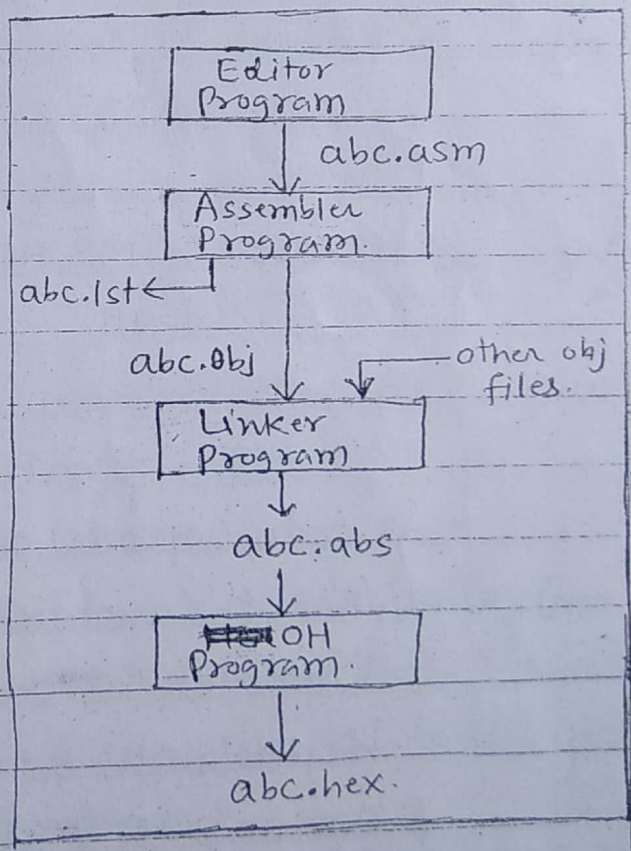
In programming techniques the five different points are considered.

Step 1 - Define the problem to be solved.
Step 2 - Solution Plan. (Algorithm)
Step 3 - Flowchart
Step 4 - Write a program.
Step 5 - Check the result.

Running an 8051 program:-

The steps to create or run an assemby language program is shown in figure.

1. First we use an editor program to type the assembly language program. An Editor MSDOS EDIT program or Notepad in windows are used for this purpose. The Editor must be able to produce ASCII file. The source file has the extension ".asm" or ".src" depending on the assembler which is used. The "asm" file is also called the source file.



2. The "asm" source file containing the program code created in step1 is fed to an 8051 assembler. The assembler converts the instructions into machine code. The assembler will produce an object file and a list file. The extension for the object file is "obj" while for the list file is "lst". The assembler converts the "asm" file in to machine language and provides "obj" extension i.e. object file. It also creates a list file with extension "lst", This file lists all the opcodes and addresses, as well as errors that the assembler detected. The programmer uses the list file to find syntax errors. When all the errors are removed in the "lst" file, the "obj" file is ready to be input to the linker program.

3. The assembler requires a third step called linking. The Link program takes one or more object files and produces an absolute object file with the extension "abs". This "abs" file is used by 8051 trainers that have a monitor program.

4. Next, the "abs" file is fed into a program called "OH" (object to Hex converter) which creates a file with extension "hex" that is ready to burn into ROM.

Recent windows-based assemblers combine all steps in to one compiler like, uVision, uVision2, uVision3, uVision4 etc. versions from keil with advanced facilities to observe the results on screen of computer monitor.

Examples –

1. Arithmatic Assembly language Programs :-

Program 1 – Addition of two 8-bit numbers.

① Assuming two numbers are available in A and B registers, write a program in assembly language of 8051 to add two 8 bit numbers.

② Solution Plan :-

ⓐ Consider that a byte of data is present in the A-register and second byte of data is present in the B register.

ⓑ We have to add the byte in B from the byte in A. Using ADD instruction add the contents of two registers.

ⓒ Result will be stored in the A-register.

For example –

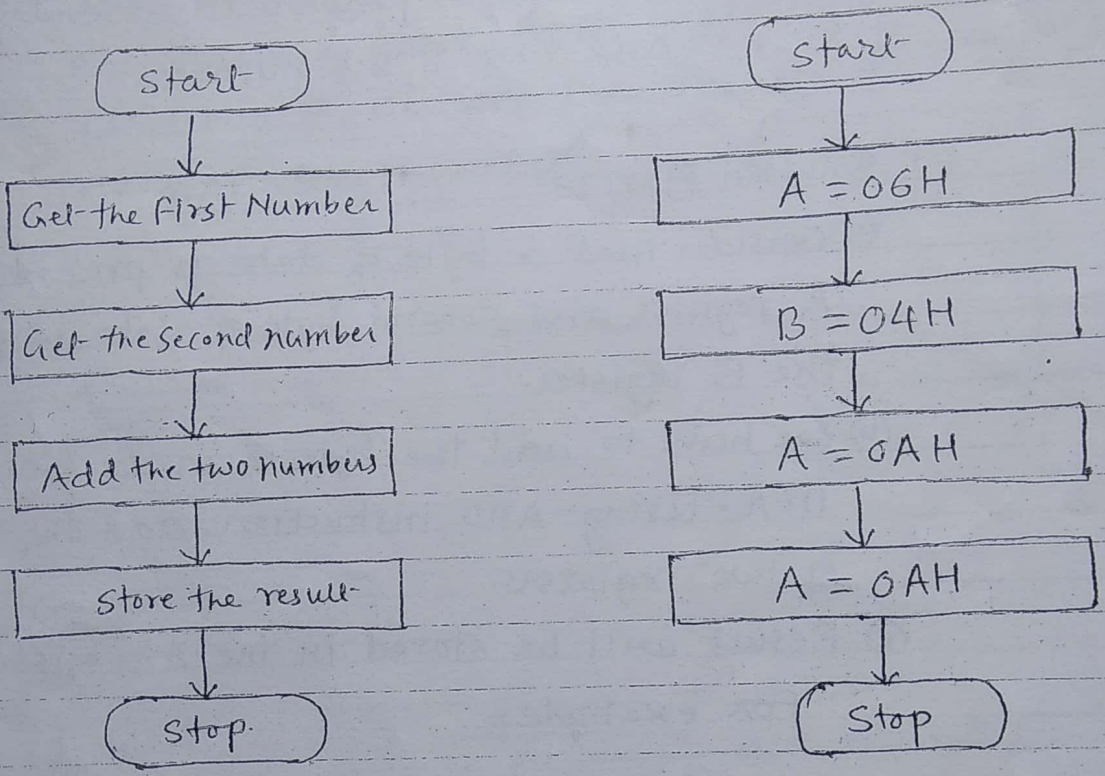
A = 06 H
B = 04 H

06 H	$(06)_{10}$
+ 04 H	+ $(04)_{10}$
0A H	$(10)_{10}$

(8) Algorithm :-

Step 1 - Get the first number in A-register.
Step 2 - Get the second number in B-register
Step 3 - Add the two numbers.
Step 4 - Stop.

Flowchart :-



Program -

Instruction	Comment	operation.
MOV A, #06H	; Load the first no. in A	A = 06H
MOV B, #04H	; load the second no in B	B = 04H
ADD A, B	; Compute addition. Result is stored in A.	A = 0AH
END	; End program.	END

② write program to add two 16-bit numbers, the numbers are FC45H and 02ECH

When adding two 16 bit data operands, we need to take care of the propogation of a carry from the lower byte to the higher byte. The instruction ADDC (add with carry) is used on such occassion.

Program -

```
CLR  C          ; make CY=0
MOV  A, #45H    ; Load the low byte into A
ADD  A, #0ECH   ; Add the low byte, now A=31, CY=1
MOV  R0, A      ; save the low byte of sum in R0
MOV  A, #02H    ; load the high byte in to A
ADDC A, #0FCH   ; Add the high bytes with carry
                ; 02+FCH+1 = FFH
MOV  R1, A      ; Save the high byte of result in R1
```

Finally we get the result as R0 = 31H & R1 = FFH
The answer is 31FFH.

(10)

③ Subtraction of two 8-bit numbers:-

There are two different instructions for subtraction SUB and SUBB (subtract with borrow) In 8051 we have only SUBB. There are two cases for the SUBB instruction -

(1) with CY=0 and

(2) with CY=1

(1) SUBB (subtract with borrow) when CY=0.

The 8051 uses adder circuitry to perform the subtraction command. The steps involved in subtraction is as follows -

① Take the 2's complement of the subtrahend (source operand)

② Add it to the minuend.

③ Invert the carry.

Program -

```
CLR  C          ; make CY=0
MOV  A, #3FH    ; Load 3FH into A (A=3FH)
MOV  R3, #23H   ; Load 23H into R3 (R3=23H)
SUBB A, R3      ; Subtract R3 from A and place
                  result in A.
```

A = 3F 0011 1111 0011 1111 → minuend

R3 = 23 0010 0011 + 1101 1101 → subtrahend (2's Complement i.e. 1's comp + 1)

1C 0001 1100 1 0001 1100

1 C 0 0001 1100 - carry is inverted i.e. CF=0

(2) SUBB (subtract with borrow) when CY=1

This instruction is used for multibyte numbers and will take care of the borrow of the lower operand. If CY=1 prior to executing the SUBB instruction, it also subtracts 1 from the result. If the CY=0 after the execution of SUBB, the result is positive; if CY=1, the result is negative and the destination has the 2's complement of the result. Normally, the result is left in 2's complement, but the CPL (complement) and INC (increment) Instruction can be used to Change it. The CPL instruction performs the 1's complement of the operand, then the operand is incremented (INC) to get the 2's complement.

Program:-

```
        CLR C           ; make CY=0
        MOV A, #4CH     ; load A with value 4CH (A=4CH)
        SUBB A, #6EH    ; subtract 6EH from A
        JNC NEXT        ; If CY=0 then jump to NEXT target
        CPL A           ; If CY=1 then take 1's complement
        INC A           ; and increment to get 2's complement
NEXT:   MOV R1, A       ; Save A in R1.
        END             ; End program.
```

(2) 8051-subtraction

4C	0100 1100	0100 1100	
-6E	-0110 1110	+1001 0010	2's complement
-22	0010 0010	0 1101 1110	invert the carry
		1 1101 1110	since CY=1; the result is negative

Program Statement – Subtraction of two 8-bit numbers.

Write a program to subtract two 8 bit numbers assuming two numbers are available in A and B register. A = 0AH and B = 04H.

A = 0AH	0AH	$(10)_{10}$
B = 04H	-04H	$(4)_{10}$
	06H	$(6)_{10}$

Algorithm – ① Get the first no. in A register
② Get the second no. in B register
③ subtract the two numbers.
④ Stop.

Instruction	Comment	Operation.
MOV A, #0AH	; load the first no. in reg. A.	A = 0AH
MOV B, #04H	; load the second no. in reg B.	B = 04H
SUBB A,B	; compute subtraction; store the result in A.	A = 06H
END	; End Program.	END.

Program Statement :- Subtraction of two 16-bit numbers.

Write a program to subtract two 16-bit numbers. Let the two numbers be 7856H and 1234H, store the result of ~~fraction~~ subtraction in the DPTR register.

Algorithm -

1. Start
2. Get the first 16 bit number in DPTR
3. Get the LSB of the second no. in reg. A.
4. Get the MSB of the second no. in reg. B.
5. Subtract the LSB of two numbers.
6. Store the result of LSB subtraction in register DPL.
7. Load the MSB of the first no. in accumulator
8. Subtract the MSB of two numbers.
9. Store the result of MSB subtraction in reg. DPH.
10. Stop.

Program :-

Instruction	comment	operation.
MOV DPTR, #1234H;	load the first no. in DPTR	DPTR=1234H
MOV A, #56H	; load the LSB of second no. in reg. A.	A=56H
MOV B, #78H	; load the MSB of second no. in reg. B	B=78H
CLR C	; clean the carry	
SUBB A, DPL	; subtract the two LSB's	A=56-34=1
MOV DPL, A	; Store the result of subtraction in DPL.	DPL=12H
MOV A, B	; load the MSB of second no. in reg. A.	A=78H
SUBB A, DPH	; subtract the two MSB's	A=78-12=
MOV DPH, A	; store the result of subtraction in DPH	DPH=66H
END	; END Program.	

(14) Program statement - Multiplication of two 8-bit numbers

Multiply two 8 bit numbers stored in external memory locations 3000H and 3001H. Store the result in memory locations 3020H and 3021H.

Let 3000H = 09H and 3001H = 02H

∴ A = 09H

B = 02H

$$\begin{array}{r} 09H \\ \times\ 02H \\ \hline 0012H \end{array}$$

then 3020H = 12H

and 3021H = 00H

Algorithm:-

① Get the first number

② Get the second number

③ Multiply the two numbers.

④ Store the result and stop.

Instruction	Comment	Operation
MOV DPTR, #3000H	; Initialize DPTR as memory pointer	DPTR = 3000H
MOVX A, @DPTR	; Get the first no. in reg A.	A = 09H
MOV B, A	; B = First number.	B = 09H
INC DPTR	; Increment memory pointer to point the second number.	DPTR = 3001H
MOVX A, @DPTR	; Get the second number in reg A.	A = 02H
MUL A, B	; Compute multiplication A = LSB digit B = MSB digit after multiplication.	A = 12H; B = 00H
MOV DPTR, #3020H	; Initialize Memory pointer for storing result	DPTR = 3020H
MOVX @DPTR, A	; Store the LSB digit obtained at location 3020H	3020H = 12H (LSB)
INC DPTR	; Increment memory pointer	DPTR = 3021H
MOV A, B.	; Store the MSB digit obtained in reg A.	A = 00H
MOVX @DPTR, A	; Store the MSB digit obtained at location 3021H	3021H = 00H (MSB)
END	; END program	

Program statement - Division of two 8-bit numbers.

- Divide two 8-bit numbers stored in memory locations 3000H & 3001H store the result in memory locations 3020H and 3021H. The quotient is to be stored at memory location 3020H and remainder at memory location 3021H.

Let 3000H = 0FH and 3001H = 08H Thus, A = 0FH and B = 08H

```
08 ) 000F ( 01 - Quotient
     - 08
     ----
      07 - remainder
```

∴ 3020H = 01H
& 3021H = 07H

Algorithm -

1. Get the first number
2. Get the second number
3. Divide the two numbers
4. Store the result and stop.

Program Instruction	comment	operation
MOV DPTR, #3000H	; Initialize DPTR as memory pointer	DPTR = 3000H
MOVX A, @DPTR	; Get the first no. in reg. A	A = 0FH
MOV R0, A	; R0 = First number	R0 = 0FH
INC DPTR	; Increment memory pointer to point the second number.	DPTR = 3001H
MOVX A, DPTR	; Get the second number in reg. A	A = 08H
MOV B, A	; load the divisor in register B.	B = 08H
MOV A, R0	; load the dividend in register A	A = 0FH
DIV A, B.	; Compute division A = Quotient and B = remainder.	A = 01H, B = 07
MOV DPTR, #3020H	; Initialize DPTR as memory pointer for storing the result.	DPTR = 3020H
MOVX @DPTR, A	; Store the quotient obtained at location 3020H	3020H = 01H
INC DPTR	; Increment memory pointer.	DPTR = 3021H
MOV A, B.	; Store the remainder obtained in register A.	A = 07H
MOVX @DPTR, A	; store the remainder obtained at Location 3021H	3021H = 07H
	; stop	
END		

(16) Program statement - Program to unpack the Packed BCD num

A two digit BCD number is stored in the register A. Wri a program in ALP of 8051 to unpack this BCD number store the MSB digit in register R1 and LSB digit in register R0 of the register bank 3.

Explaination -

A digit BCD number is available in register A. we have to unpack this BCD number i.e we have to seperate the BCD digits.
example - of the number = 92H then in unpack form the two digits will be 02H and 09H i.e we have to mask the lower nibble, first and rotate fourtimes to the right to get the MSB digit or we the SWAP instruction.
Then to get the LSB digit mask the upper nibble. Masking lower nibble means ANDing the number with 0F0 to get MSB.

Algorithm -

① Load number in to register A.
② Mask the lower nibble
③ Rotate 4 times left to make MSB digit = LSB.
④ Store the digit in register R1.
⑤ Load number in register A.
⑥ Mask upper nibble.
⑦ Store the digit in register R0.
⑧ Stop.

Program.

Instruction	Comment	operation
MOV A, #92H	; load the no. in accumulator	A=92H
MOV B, A	; store the no. in reg. B.	B=92H
ANL A, #0F0H	; Mask lower nibble	A=90H
SWAP A	; Make the MSB digit ↔ LSB digit	A=09H
MOV R1, A	; store the result in register R1 of register bank 3.	R1=09H(MSB
MOV A, B	; load the number back in accumulator	A=92H
ANL A, #0FH	; Mask upper nibble	A=02H
MOV R0, A	; Store the result in register R0 of register bank 3.	R0=02H(LSB)
END	; End program.	

Program statement - Program to find sum of series.
To add block of data assuming the sum to be 8-bit
Write a program to add ten bytes in internal RAM.
Assume that the starting location of the block is 40H.
Assume sum to be 8 bit. store the result in register
R0 of bank 1.

Explaination - Consider that a block of 10 bytes is present
at source location i.e. 40H. We have to add these 10 bytes
we will initialize this as count in the R0 register.
The register R1 will act as pointer to point the block.

(18)

Using Add instruction add the contents, byte by byte of the block. Increment R1 to point to next element. Decrement the counter and continue till all the contents are added. Result is stored in A. This result is stored in register R0 of bank 1.

For example -

Block data : 01 02 03 04 05 06 07 08 09 0A

Result = 01+02+03+04+05+06+07+08+09 = 37H

Algorithm :-

① Initialise R1 as pointer with source address.
② Initialize R0 register with count.
③ Add data, byte by byte.
④ Increment pointer.
⑤ Decrement pointer.
⑥ check for count, if not zero go to step ③ else go to step ⑦.
⑦ store the result of addition.
⑧ Stop.

Program -

Label	Instruction	Comment	operation
	CLR PSW.3 CLR PSW.4 }	; select register bank	register bank 0 is selected.
	MOV R0, #0AH	; initialize reg. R0 as counter	R0 = 0AH
	MOV R1, #40H	; initialize reg. R1 as memory pointer.	R1 = 40H
	MOV A, #00H	; Initialize result = 0.	A = 00 H
L1:	ADD A, @R1	; compute addition	A = A + @R1
	INC R1	; Increment R1 to point next memory location.	R1 = R1 + 1
	DJNZ R0, L1	; Check if count = 0	IS R0 = 0 if not continue looping L1
	MOV R0, A	; Store the result in reg R0 of bank 1.	R0 = 37H
	END	; END Program	

Result = 37H

step (9)

Program:- Program to find Average of block of N-bytes.

- write an ALP to find the average of block of N-bytes.

Explaination:-

Consider that a block of data N-bytes is present at the source location. Let the number of bytes N=10. We have to initialize this count in the R0-register. Our task is to find out the average of these 10 bytes i.e. to find out average of 10 numbers.

$$\text{Average} = \frac{\text{Sum of N bytes.}}{N}$$

Assume that the source address is in the R1-register. Initially, we will add the N-bytes, byte by byte using ADD instruction. Once the addition is compute, result of addition will be stored in register A. Now, we will divide this sum by number of bytes N to find the average. The result of division i.e. quotient is present in the A register and remainder in the B-register.

for example.

Block data = 01 02 03 04 05 06 07 08 09 0A.

$\therefore$ Sum = 01+02+03+04+05+06+07+08+09+0AH

= 37H.

$$\text{average} = \frac{37H}{0AH} = \frac{(55)_{10}}{(10)_{10}}$$

= 5H.

Algorithm :-

1. Initialize R1 with source address
2. Initialize R0 register with the count
3. Add the N bytes, byte by byte
4. Increment pointer R1
5. Decrement count
6. check for count in R0, if not zero go to step (2)
7. Initialize register B with count of numbers.
8. Find average = $\frac{\text{Sum}}{\text{count}}$
9. Stop.

Program :-

Instruction	Comment	Operation
MOV R1, #20H	; initialize reg. R1 as with source address	R1 = 20H
MOV R0, #0AH	; initalize reg. R0 with count of numbers.	R0 = 0AH
MOV B, #00H	; initialize result in reg. B.	B = 00H
L1: MOV A, @R1	; Load the number in accumulator	A = @R1
ADD A, B	; Add the data byte by byte	A = A + B
MOV B, A	; Store the result of addition in reg. B.	B = A
INC R1	; Increment source memory pointer to next location.	R1 = R1 + 1
DJNZ R0, L1	; Decrement counter and continue till count = 0	If R0 ≠ 0 then R0 = R0 - 1 and continue executing L1
MOV A, B	; store the result of addition in reg A:	A = B
MOV B, #0AH	; Initialize register B with count of numbers	B = 0AH
DIV AB	; average = sum/count. A = quotient and B = remainder	A = 05H B = 05H
END.	; End Program.	End.

(22) Program statement:- 1's complement of 8-bit number

Write a program to find the 1's complement of the number. Assume that the number 20H is stored in register R3 of the register bank 0.

Explaination:- One's complement of a number means to invert each bit of that number. We will first load the number whose 1's complement is to be found in the accumulator. Using the CPL instruction we will complement the number in accumulator. The result will be stored in register B or desired location.

Algorithm

① Select register bank 0.
② Get the no. in accumulator
③ complement the number
④ Store the result.

Program:-

Instruction	Comment	Operation
CLR PSW.3 CLR PSW.4	; select register Bank 0.	Register bank 0 selected
MOV A, R3	; Load the no. in accumulator	A = 20H [0010 0000]
CPL A	; Compute 1's complement	A = DFH [1101 1111]
MOV B, A	; Store the result	A = DFH B = DFH
END	; End program.	

Program Statement:- 2's complement of a number.

Write a program to find the 2's complement of the number. Assume that the number is stored in register R3 of the register bank 0.

Explaination:-

Two's complement of number means add 1 to the 1's complement of that number. One's complement of number means to invert each bit of that number. We will first load the number whose 1's complement is to be found in the accumulator. Using the CPL instruction we will complement the accumulator. This is 1's complement of the number. Now add 1 to this number so that 2's complement is computed. Store the result in register R3 of register bank 0.

Algorithm:-

① Select register bank 0.
② Get the number in accumulator
③ Complement the number.
④ Add 1 to compute 2's complement.
⑤ Store the result.

Program:

Instruction	Comment	operation.
CLR PSW.3 CLR PSW.4	; select register bank 0	Register bank 0 selected
MOV A, R3	; load the no. in Accumulator	A = 20H 0010 0000
CPL A	; Compute 1's complement	A = DFH 1101 1111
ADD A, #01H	; compute 2's complement	A = E0H
MOV A, R3	; Store the result	R3 = E0H
END	; End Program.	

(24) Program statement - Program for Binary-Gray conversion

Write a program in the ALP to convert a given binary number into its Gray code equivalent. Store the Gray equivalent in register A.

Explaination -

For Binary-Gray conversion -

① Record the MSB as it is

② Add this bit to the next position recording the sum and neglecting carry if any.

③ Record successive sums untill completed.

e.g. to Convert 0AH i.e. 1010 binary to gray.

Given binary number → 1 0 1 0

Gray code → 1 1 1 1 LSB = (0F)

For programming it, i.e. Binary gray conversion first we will add the number with itself. Then We will X-OR this added number with the original number. Then we will shift this X-ORed number by 1-bit position to the right. This gives the equivalent gray code. Display this gray code

For e.g. the number is 0A

```
  0AH      X-ORed added number  0001 0100 (14H)
+ 0AH      with original number 0000 1010 (0AH)
-----                           ---------
  14H                          0001 1110
                               ---------
shift by 1 bit to right →      0000 1111 (0FH)
```

Algorithm:-

1. Get the number whose gray code equivalent is to be found.
2. Add number with itself
3. XOR this added result with the original number.
4. Shift the XORed number by 1 bit position to the right to get the gray equivalent.
5. Store the result.
6. Stop.

Program:-

Instruction	comment	operation
MOV A, #0AH	; Load number in register A	A = 0AH
MOV B, A	; Get the no. in register B also	B = 0AH
ADD A, A	; Add contents of A with itself	A = 14H PSW = 40H
XRL A, B	; XOR the added contents with no. itself	A = 1EH PSW = 40H
RR A	; Roll by 1 bit to right to get gray equivalent.	A = 0FH
END	; END Program.	

(26)

Program Statement:- Largest of N numbers.
Program to find the maximum/largest no. in the array.

Given - an array of N-numbers.

Write a program in ALP of 8051 to find the maximum number amongst the N numbers. Assume the length of array is stored in register R0 of the register bank 0. Assume that the array begins from memory location 3000H. Store the maximum or the largest number in the register R1 of register bank 0.

Explaination:-

We have an array of 10 numbers. So we initialize the counter with 10. Also we initialize a pointer to point these numbers. Compare first number with initial maximum number i.e. zero. If number > maximum number, save number otherwise increment pointer to compare next number. Decrement counter and compare till all the numbers are compared. Store the maximum number in register R1.

Algorithm:-

① Initialize memory pointer.
② Load data pointed by memory pointer i.e. OP1 and store it in register B.
③ Decrement counter.
④ Check if count = 0 If yes go to step 10.

⑤ Increment memory pointer to next memory location.
⑥ Load data pointed by pointer i.e. OP2.
⑦ Compare the two numbers i.e. OP1 and OP2 If OP1 = OP2 go to step ③
⑧ If OP2 < OP1 go to step ③.
⑨ If OP2 > OP1 go to step ②
⑩ Store the result in register R1.
⑪ Stop.

Program:

Instruction	comment	operation
MOV DPTR, #3000H	; Load DPTR with the address of data	DPTR = 3000H
MOVX A, @DPTR	; Load accumulator with data	A = @DPTR
L1: MOV B, A	; Store the number in register B.	B = A
L3: DJNZ R0, L2	; Decrement counter	If R0 ≠ 0 then R0 = R0 - 1 and go to L2.
SJMP STP	; Stop if counter = 0	Stop
L2: INC DPTR	; Increment data pointer to point next memory location.	DPTR = DPTR + 1
MOVX A, @DPTR	; Load accumulator with data	A = @DPTR
CJNE A, B, NEQ	; Find the maximum number	B = 64H
SJMP L3	; If A = B, continue	
NEQ: JC L3	; If A < B, continue	
SJMP L1	; If A > B, Exchange the contents.	
STP: MOV R1, B.	; Store the largest no. in register R1	R1 = 64H
End	; End the program.	

Program Statement - Program to find the least/smallest no. in the array.

Given - an array of N numbers.

Write a program in ALP of 8051 to find the minimum number amongst the N-numbers. Assuming the length of array is stored in register R0 of the register bank 0. Assume that the array begins from memory location 3000H. Store the smallest or the least number in the register R1 of register bank 0.

Explaination:-

We have an array of 10 numbers. So, we initialize the counter with 10. Also, we initialize a pointer to point these numbers. Compare first number with initial minimum number i.e zero. If number < minimum number, save number otherwise increment pointer to compare next number. Decrement counter, compare till all the numbers are compared. Store the smallest number in register R1.

Algorithm:-

① Initialize memory pointer.
② Load data pointed by memory pointer. i.e. OP1 and store it in register B.
③ Decrement counter.
④ Check if count = 0, if yes go to step ⑩

(5) Increment memory pointer to next memory location.
(6) Load data pointed by pointer i.e. OP2.
(7) Compare the two number ; i.e. OP1 and OP2.
If OP1 = OP2 go to step (9).
(8) If OP2 > OP1 ; go to step (5)
(9) If OP2 < OP1 ; go to step (2)
(10) Store the result in register R1.
(11) Stop.

Program :-

Instruction	Command	operation
	MOV DPTR, #3000H ; Load DPTR with the address of data	DPTR=3000H
	MOVX A, @DPTR ; Load accumulator with data	A=@DPTR
	MOV B, A ; Store the no. in register B.	B=A
L1:	MOV B, A	
L3:	DJNZ R0, L2 ; Decrement counter	If R0 ≠ 0 then R0=R0-1 and go to L2
	SJMP STP ; Stop if counter = 0	→Stop
L2:	INC DPTR ; Increment data pointer to point next memory location.	DPTR=DPTR+1
	MOVX A, @DPTR ; Load accumulator with data.	A=@DPTR
	CJNE A, B, NEQ ; Find the minimum number.	B=[AH]
	SJMP L3 ; If A=B, continue	
NEQ:	JNC L3 ; If A>B, continue	
	SJMP L1 ; If A<B; Exchange the contents	
STP:	MOV R1, B. ; Store the smallest number in register R1.	R1=[AH]
	END ; End the program.	

(24)

Program Statement:- Program to move the block of Memory (source) to other area (destination)

Write an ALP to move a block of N-bytes of data from source to destination. Assume that the length of block is stored in register R2 of register bank 0. The source block starts from memory location 30H and the destination block begins from memory location 25H.

Explaination:-

Consider that a block of data of N bytes is present at source location. Now this blocks of N bytes is to be moved from source location to a destination location. The number of bytes, N (Let N = 10) (i.e. 0AH) is stored in register R2 of bank 2. We will have to initialize this as count. For pointing the source address we will use the R0 register and for the destination address we will use the R1 register. Transfer the data byte by byte from source to destination block. Initialise R2 register with the count.

Algorithm:-

(1) Initialize R2 register with the count.
(2) Initialize R0 and R1 with the source and destination address of the last element in the block.
(3) Transfer the data block byte by byte to destination.

④ Decrement count

⑤ Decrement source and destination memory pointer.

⑥ Check for count in R2, if not zero go to step ③ else go to step ⑦.

⑦ stop.

Program :-

Instruction	Comment	operation
CLR PSW.3 CLR PSW.4	; Select Register bank 0	Register Bank 0 selected.
MOV R0, #20H	; Initialise register R0 as with source address	R0 = 20H
MOV R1, #25H	; Initialize reg. R1 as with destination address	R1 = 25H
MOV A, R2	; Load accumulator with the size of block	A = 0AH
ADD A, R0	;	A = 2AH
MOV R0, A	; R0 contains address of the number after the last element in the source block.	R0 = 2AH
MOV A, R2	; load accumulator with the size of block.	A = 0AH
ADD A, R1	;	A = 2FH
MOV R1, A	; R1 contains address of the number after the last element in the destination block.	R1 = 2FH
DEC R0	; R0 contains address of the last element of the source.	R0 = 29H
DEC R1	; R1 contains address of the last element of the destination.	R1 = 2EH
L1: MOV A, @R0	; Load accumulator with numbe from source block.	[loop for transfer byte by byte from source to destinati e.g. data at meme location 20H i.e. 01H is transfere to memory locat 25H.
MOV @R1, A	; Store the data in desired memory location.	
DEC R0	; Decrement source memory pointer.	
DEC R1	; Decrement destination memory pointer	
DJNZ R2, L1	; check if count = 0 ?	
END	; End program.	END.