

8085 MICROPROCESSOR PROGRAMS

ADDITION OF TWO 8 BIT NUMBERS

AIM:

To perform addition of two 8 bit numbers using 8085.

ALGORITHM:

- 1) Start the program by loading the first data into Accumulator.
- 2) Move the data to a register (B register).
- 3) Get the second data and load into Accumulator.
- 4) Add the two register contents.
- 5) Check for carry.
- 6) Store the value of sum and carry in memory location.
- 7) Terminate the program.

PROGRAM:

| | | | |
|-------|-----|-------|--|
| | MVI | C, 00 | Initialize C register to 00 |
| | LDA | 4150 | Load the value to Accumulator. |
| | MOV | B, A | Move the content of Accumulator to B register. |
| | LDA | 4151 | Load the value to Accumulator. |
| | ADD | B | Add the value of register B to A |
| | JNC | LOOP | Jump on no carry. |
| | INR | C | Increment value of register C |
| LOOP: | STA | 4152 | Store the value of Accumulator (SUM). |
| | MOV | A, C | Move content of register C to Acc. |
| | STA | 4153 | Store the value of Accumulator (CARRY) |
| | HLT | | Halt the program. |

OBSERVATION:

| | |
|---------|------------------------|
| Input: | 80 (4150) 80 (4251) |
| Output: | 00 (4152) 01 (4153) |

RESULT:

Thus the program to add two 8-bit numbers was executed.

SUBTRACTION OF TWO 8 BIT NUMBERS

AIM:

To perform the subtraction of two 8 bit numbers using 8085.

ALGORITHM:

1. Start the program by loading the first data into Accumulator.
2. Move the data to a register (B register).
3. Get the second data and load into Accumulator.
4. Subtract the two register contents.
5. Check for carry.
6. If carry is present take 2's complement of Accumulator.
7. Store the value of borrow in memory location.
8. Store the difference value (present in Accumulator) to a memory location and terminate the program.

PROGRAM:

| | | |
|-------|----------|--|
| MVI | C, 00 | Initialize C to 00 |
| LDA | 4150 | Load the value to Acc. |
| MOV | B, A | Move the content of Acc to B register. |
| LDA | 4151 | Load the value to Acc. |
| SUB | B | |
| JNC | LOOP | Jump on no carry. |
| CMA | | Complement Accumulator contents. |
| INR | A | Increment value in Accumulator. |
| INR | C | Increment value in register C |
| LOOP: | STA 4152 | Store the value of A-reg to memory address. |
| | MOV A, C | Move contents of register C to Accumulator. |
| | STA 4153 | Store the value of Accumulator memory address. |
| | HLT | Terminate the program. |

OBSERVATION:

Input: 06 (4150)
02 (4251)
Output: 04 (4152)
01 (4153)

RESULT:

Thus the program to subtract two 8-bit numbers was executed.

MULTIPLICATION OF TWO 8 BIT NUMBERS

AIM:

To perform the multiplication of two 8 bit numbers using 8085.

ALGORITHM:

- 1) Start the program by loading HL register pair with address of memory location.
- 2) Move the data to a register (B register).
- 3) Get the second data and load into Accumulator.
- 4) Add the two register contents.
- 5) Check for carry.
- 6) Increment the value of carry.
- 7) Check whether repeated addition is over and store the value of product and carry in memory location.
- 8) Terminate the program.

PROGRAM:

| | | | |
|-------|-----|---------|---------------------------------------|
| | MVI | D, 00 | Initialize register D to 00 |
| | MVI | A, 00 | Initialize Accumulator content to 00 |
| | LXI | H, 4150 | |
| | MOV | B, M | Get the first number in B - reg |
| | INX | H | |
| | MOV | C, M | Get the second number in C- reg. |
| LOOP: | ADD | B | Add content of A - reg to register B. |
| | JNC | NEXT | Jump on no carry to NEXT. |
| | INR | D | Increment content of register D |
| NEXT: | DCR | C | Decrement content of register C. |
| | JNZ | LOOP | Jump on no zero to address |
| | STA | 4152 | Store the result in Memory |
| | MOV | A, D | |
| | STA | 4153 | Store the MSB of result in Memory |
| | HLT | | Terminate the program. |

OBSERVATION:

Input: FF (4150)
FF (4151)
Output: 01 (4152)
FE (4153)

RESULT:

Thus the program to multiply two 8-bit numbers was executed.

DIVISION OF TWO 8 BIT NUMBERS

AIM:

To perform the division of two 8 bit numbers using 8085.

ALGORITHM:

- 1) Start the program by loading HL register pair with address of memory location.
- 2) Move the data to a register(B register).
- 3) Get the second data and load into Accumulator.
- 4) Compare the two numbers to check for carry.
- 5) Subtract the two numbers.
- 6) Increment the value of carry .
- 7) Check whether repeated subtraction is over and store the value of product and carry in memory location.
- 8) Terminate the program.

PROGRAM:

| | | | |
|-------|-----|---------|----------------------------------|
| | LXI | H, 4150 | |
| | MOV | B, M | Get the dividend in B – reg. |
| | MVI | C, 00 | Clear C – reg for quotient |
| | INX | H | |
| | MOV | A, M | Get the divisor in A – reg. |
| NEXT: | CMP | B | Compare A - reg with register B. |
| | JC | LOOP | Jump on carry to LOOP |
| | SUB | B | Subtract A – reg from B- reg. |
| | INR | C | Increment content of register C. |
| | JMP | NEXT | Jump to NEXT |
| LOOP: | STA | 4152 | Store the remainder in Memory |
| | MOV | A, C | |
| | STA | 4153 | Store the quotient in memory |
| | HLT | | Terminate the program. |

OBSERVATION:

Input: FF (4150)
FF (4251)

Output: 01 (4152) ---- Remainder
FE (4153) ---- Quotient

RESULT:

Thus the program to divide two 8-bit numbers was executed.

LARGEST NUMBER IN AN ARRAY OF DATA

AIM:

To find the largest number in an array of data using 8085 instruction set.

ALGORITHM:

- 1) Load the address of the first element of the array in HL pair
- 2) Move the count to B – reg.
- 3) Increment the pointer
- 4) Get the first data in A – reg.
- 5) Decrement the count.
- 6) Increment the pointer
- 7) Compare the content of memory addressed by HL pair with that of A - reg.
- 8) If Carry = 0, go to step 10 or if Carry = 1 go to step 9
- 9) Move the content of memory addressed by HL to A – reg.
- 10) Decrement the count
- 11) Check for Zero of the count. If ZF = 0, go to step 6, or if ZF = 1 go to next step.
- 12) Store the largest data in memory.
- 13) Terminate the program.

PROGRAM:

| | | | |
|--------|-----|--------|---|
| | LXI | H,4200 | Set pointer for array |
| | MOV | B,M | Load the Count |
| | INX | H | |
| | MOV | A,M | Set 1 st element as largest data |
| | DCR | B | Decrement the count |
| LOOP: | INX | H | |
| | CMP | M | If A- reg > M go to AHEAD |
| | JNC | AHEAD | |
| | MOV | A,M | Set the new value as largest |
| AHEAD: | DCR | B | |
| | JNZ | LOOP | Repeat comparisons till count = 0 |
| | STA | 4300 | Store the largest value at 4300 |
| | HLT | | |

OBSERVATION:

Input: 05 (4200) ----- Array Size
 0A (4201)
 F1 (4202)
 1F (4203)
 26 (4204)
 FE (4205)

Output: FE (4300)

RESULT:

Thus the program to find the largest number in an array of data was executed

SMALLEST NUMBER IN AN ARRAY OF DATA

AIM:

To find the smallest number in an array of data using 8085 instruction set.

ALGORITHM:

- 1) Load the address of the first element of the array in HL pair
- 2) Move the count to B – reg.
- 3) Increment the pointer
- 4) Get the first data in A – reg.
- 5) Decrement the count.
- 6) Increment the pointer
- 7) Compare the content of memory addressed by HL pair with that of A - reg.
- 8) If carry = 1, go to step 10 or if Carry = 0 go to step 9
- 9) Move the content of memory addressed by HL to A – reg.
- 10) Decrement the count
- 11) Check for Zero of the count. If ZF = 0, go to step 6, or if ZF = 1 go to next step.
- 12) Store the smallest data in memory.
- 13) Terminate the program.

PROGRAM:

| | | | |
|--------|-----|--------|---|
| | LXI | H,4200 | Set pointer for array |
| | MOV | B,M | Load the Count |
| | INX | H | |
| | MOV | A,M | Set 1 st element as largest data |
| | DCR | B | Decrement the count |
| LOOP: | INX | H | |
| | CMP | M | If A- reg < M go to AHEAD |
| | JC | AHEAD | |
| | MOV | A,M | Set the new value as smallest |
| AHEAD: | DCR | B | |
| | JNZ | LOOP | Repeat comparisons till count = 0 |
| | STA | 4300 | Store the largest value at 4300 |
| | HLT | | |

OBSERVATION:

Input: 05 (4200) ----- Array Size
 0A (4201)
 F1 (4202)
 1F (4203)
 26 (4204)
 FE (4205)

Output: 0A (4300)

RESULT:

Thus the program to find the smallest number in an array of data was executed

ARRANGE AN ARRAY OF DATA IN ASCENDING ORDER

AIM:

To write a program to arrange an array of data in ascending order

ALGORITHM:

1. Initialize HL pair as memory pointer
2. Get the count at 4200 into C – register
3. Copy it in D – register (for bubble sort (N-1) times required)
4. Get the first value in A – register
5. Compare it with the value at next location.
6. If they are out of order, exchange the contents of A –register and Memory
7. Decrement D –register content by 1
8. Repeat steps 5 and 7 till the value in D- register become zero
9. Decrement C –register content by 1
10. Repeat steps 3 to 9 till the value in C – register becomes zero

PROGRAM:

| | | |
|---------|-----|--------|
| | LXI | H,4200 |
| | MOV | C,M |
| | DCR | C |
| REPEAT: | MOV | D,C |
| | LXI | H,4201 |
| LOOP: | MOV | A,M |
| | INX | H |
| | CMP | M |
| | JC | SKIP |
| | MOV | B,M |
| | MOV | M,A |
| | DCX | H |
| | MOV | M,B |
| | INX | H |
| SKIP: | DCR | D |
| | JNZ | LOOP |
| | DCR | C |
| | JNZ | REPEAT |
| | HLT | |

OBSERVATION:

Input: 4200 05 (Array Size)
 4201 05
 4202 04
 4203 03
 4204 02
 4205 01

Output: 4200 05(Array Size)
 4201 01
 4202 02
 4203 03
 4204 04
 4205 05

RESULT:

Thus the given array of data was arranged in ascending order.

ARRANGE AN ARRAY OF DATA IN DESCENDING ORDER

AIM:

To write a program to arrange an array of data in descending order

ALGORITHM:

1. Initialize HL pair as memory pointer
2. Get the count at 4200 into C – register
3. Copy it in D – register (for bubble sort (N-1) times required)
4. Get the first value in A – register
5. Compare it with the value at next location.
6. If they are out of order, exchange the contents of A –register and Memory
7. Decrement D –register content by 1
8. Repeat steps 5 and 7 till the value in D- register become zero
9. Decrement C –register content by 1
10. Repeat steps 3 to 9 till the value in C – register becomes zero

PROGRAM:

| | | |
|---------|-----|--------|
| | LXI | H,4200 |
| | MOV | C,M |
| | DCR | C |
| REPEAT: | MOV | D,C |
| | LXI | H,4201 |
| LOOP: | MOV | A,M |
| | INX | H |
| | CMP | M |
| | JNC | SKIP |
| | MOV | B,M |
| | MOV | M,A |
| | DCX | H |
| | MOV | M,B |
| | INX | H |
| SKIP: | DCR | D |
| | JNZ | LOOP |
| | DCR | C |
| | JNZ | REPEAT |
| | HLT | |

OBSERVATION:

Input: 4200 05 (Array Size)
 4201 01
 4202 02
 4203 03
 4204 04
 4205 05

Output: 4200 05(Array Size)
 4201 05
 4202 04
 4203 03
 4204 02
 4205 01

RESULT:

Thus the given array of data was arranged in descending order.

BCD TO HEX CONVERSION

AIM:

To convert two BCD numbers in memory to the equivalent HEX number using 8085 instruction set

ALGORITHM:

- 1) Initialize memory pointer to 4150 H
- 2) Get the Most Significant Digit (MSD)
- 3) Multiply the MSD by ten using repeated addition
- 4) Add the Least Significant Digit (LSD) to the result obtained in previous step
- 5) Store the HEX data in Memory

PROGRAM:

| | | |
|-----|--------|---------------------------|
| LXI | H,4150 | |
| MOV | A,M | Initialize memory pointer |
| ADD | A | MSD X 2 |
| MOV | B,A | Store MSD X 2 |
| ADD | A | MSD X 4 |
| ADD | A | MSD X 8 |
| ADD | B | MSD X 10 |
| INX | H | Point to LSD |
| ADD | M | Add to form HEX |
| INX | H | |
| MOV | M,A | Store the result |
| HLT | | |

OBSERVATION:

Input: 4150 : 02 (MSD)
 4151 : 09 (LSD)

Output: 4152 : 1D H

RESULT:

Thus the program to convert BCD data to HEX data was executed.

HEX TO BCD CONVERSION

AIM:

To convert given Hexa decimal number into its equivalent BCD number using 8085 instruction set

ALGORITHM:

- 1) Initialize memory pointer to 4150 H
- 2) Get the Hexa decimal number in C - register
- 3) Perform repeated addition for C number of times
- 4) Adjust for BCD in each step
- 5) Store the BCD data in Memory

PROGRAM:

| | | | |
|--------|-----|--------|--|
| | LXI | H,4150 | Initialize memory pointer |
| | MVI | D,00 | Clear D- reg for Most significant Byte |
| | XRA | A | Clear Accumulator |
| | MOV | C,M | Get HEX data |
| LOOP2: | ADI | 01 | Count the number one by one |
| | DAA | | Adjust for BCD count |
| | JNC | LOOP1 | |
| | INR | D | |
| LOOP1: | DCR | C | |
| | JNZ | LOOP2 | |
| | STA | 4151 | Store the Least Significant Byte |
| | MOV | A,D | |
| | STA | 4152 | Store the Most Significant Byte |
| | HLT | | |

OBSERVATION:

Input: 4150 : FF

Output: 4151 : 55 (LSB)
4152 : 02 (MSB)

RESULT:

Thus the program to convert HEX data to BCD data was executed.

HEX TO ASCII CONVERSION

AIM:

To convert given Hexa decimal number into its equivalent ASCII number using 8085 instruction set.

ALGORITHM:

1. Load the given data in A- register and move to B – register
2. Mask the upper nibble of the Hexa decimal number in A – register
3. Call subroutine to get ASCII of lower nibble
4. Store it in memory
5. Move B –register to A – register and mask the lower nibble
6. Rotate the upper nibble to lower nibble position
7. Call subroutine to get ASCII of upper nibble
8. Store it in memory
9. Terminate the program.

PROGRAM:

| | | |
|-------|------|---------------------------------|
| LDA | 4200 | Get Hexa Data |
| MOV | B,A | |
| ANI | 0F | Mask Upper Nibble |
| CALL | SUB1 | Get ASCII code for upper nibble |
| STA | 4201 | |
| MOV | A,B | |
| ANI | F0 | Mask Lower Nibble |
| RLC | | |
| CALL | SUB1 | Get ASCII code for lower nibble |
| STA | 4202 | |
| HLT | | |
| | | |
| SUB1: | CPI | 0A |
| | JC | SKIP |
| | ADI | 07 |
| SKIP: | ADI | 30 |
| | RET | |

OBSERVATION:

Input: 4200 E4(Hexa data)

Output: 4201 34(ASCII Code for 4)
 4202 45(ASCII Code for E)

RESULT:

Thus the given Hexa decimal number was converted into its equivalent ASCII Code.

ASCII TO HEX CONVERSION

AIM:

To convert given ASCII Character into its equivalent Hexa Decimal number using 8085 instruction set.

ALGORITHM:

1. Load the given data in A- register
2. Subtract 30 H from A – register
3. Compare the content of A – register with 0A H
4. If A < 0A H, jump to step6. Else proceed to next step.
5. Subtract 07 H from A – register
6. Store the result
7. Terminate the program

PROGRAM:

| | |
|-------|----------|
| LDA | 4500 |
| SUI | 30 |
| CPI | 0A |
| JC | SKIP |
| SUI | 07 |
| SKIP: | STA 4501 |
| | HLT |

OBSERVATION:

Input: 4500 31

Output: 4501 0B

RESULT:

Thus the given ASCII character was converted into its equivalent Hexa Value.

SQUARE OF A NUMBER USING LOOK UP TABLE

AIM:

To find the square of the number from 0 to 9 using a Table of Square.

ALGORITHM:

1. Initialize HL pair to point Look up table
2. Get the data .
3. Check whether the given input is less than 9.
4. If yes go to next step else halt the program
5. Add the desired address with the accumulator content
6. Store the result

PROGRAM:

| | | |
|--------|----------|----------------------------------|
| LXI | H,4125 | Initialsie Look up table address |
| LDA | 4150 | Get the data |
| CPI | 0A | Check input > 9 |
| JC | AFTER | if yes error |
| MVI | A,FF | Error Indication |
| STA | 4151 | |
| HLT | | |
| AFTER: | MOV C,A | Add the desired Address |
| | MVI B,00 | |
| | DAD B | |
| | MOV A,M | |
| | STA 4151 | Store the result |
| | HLT | Terminate the program |

LOOKUP TABLE:

| | |
|------|----|
| 4125 | 01 |
| 4126 | 04 |
| 4127 | 09 |
| 4128 | 16 |
| 4129 | 25 |
| 4130 | 36 |
| 4131 | 49 |
| 4132 | 64 |
| 4133 | 81 |

OBSERVATION:

Input: 4150: 05

Output: 4151 25 (Square)

Input: 4150: 11

Output: 4151: FF (Error Indication)

RESULT:

Thus the program to find the square of the number from 0 to 9 using a Look up table was executed.

INTERFACING WITH 8085

INTERFACING 8251 (USART) WITH 8085 PROCESSOR

AIM:

To write a program to initiate 8251 and to check the transmission and reception of character

THEORY:

The 8251 is used as a peripheral device for serial communication and is programmed by the CPU to operate using virtually any serial data transmission technique. The USART accepts data characters from the CPU in parallel format and then converts them into a continuous serial data stream for transmission. Simultaneously, it can receive serial data streams and convert them into parallel data characters for the CPU. The CPU can read the status of USART at any time. These include data transmission errors and control signals.

Prior to starting data transmission or reception, the 8251 must be loaded with a set of control words generated by the CPU. These control signals define the complete functional definition of the 8251 and must immediately follow a RESET operation. Control words should be written into the control register of 8251. These control words are split into two formats:

1. MODE INSTRUCTION WORD
2. COMMAND INSTRUCTION WORD

1. MODE INSTRUCTION WORD

This format defines the Baud rate, Character length, Parity and Stop bits required to work with asynchronous data communication. By selecting the appropriate baud factor sync mode, the 8251 can be operated in Synchronous mode.

Initializing 8251 using the mode instruction to the following conditions

- 8 Bit data
- No Parity
- Baud rate Factor (16X)
- 1 Stop Bit

gives a mode command word of 01001110 = 4E (HEX)

MODE INSTRUCTION - SYNCHRONOUS MODE

| | | | | | | | |
|----|----|----|-----|----|----|----|----|
| S2 | S1 | EP | PEN | L2 | L1 | B2 | B1 |
|----|----|----|-----|----|----|----|----|

| BAUD RATE FACTOR | | | | |
|------------------|------|---|-------|-------|
| 0 | 1 | 0 | 1 | |
| 0 | | 0 | 1 | 1 |
| SYNC MODE | (1X) | | (16X) | (64X) |

| CHARACTR LENGTH | | | | |
|-----------------|--------|--------|--------|--|
| 0 | 1 | 0 | 1 | |
| 0 | 0 | 1 | 1 | |
| 5 BITS | 6 BITS | 7 BITS | 8 BITS | |

| PARITY ENABLE | | | | |
|---------------|-------------|--|--|--|
| 1= ENABLE | 0 = DISABLE | | | |

| EVEN PARITY GEN/CHECK | | | | |
|-----------------------|----------|--|--|--|
| 0 = ODD | 1 = EVEN | | | |

| NUMBER OF STOP BITS | | | | |
|---------------------|-------|---------|-------|--|
| 0 | 1 | 0 | 1 | |
| 0 | 0 | 1 | 1 | |
| INVALID | 1 BIT | 1.5 BIT | 2 BIT | |

MODE INSTRUCTION - ASYNCHRONOUS MODE

| | | | | | | | |
|----|----|----|-----|----|----|----|----|
| S2 | S1 | EP | PEN | L2 | L1 | B2 | B1 |
|----|----|----|-----|----|----|----|----|

| CHARACTER LENGTH | | | |
|------------------|--------|--------|--------|
| 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 5 BITS | 6 BITS | 7 BITS | 8 BITS |

| PARITY ENABLE | |
|---------------|-------------|
| 1= ENABLE | 0 = DISABLE |

| EVEN PARITY GEN/CHECK | |
|-----------------------|----------|
| 0 = ODD | 1 = EVEN |

| EXTERNAL SYNC DETECTS | |
|--------------------------|--|
| 1 = SYSDET IS AN INPUT | |
| 0 = SYSDET IS AN IOUTPUT | |

| SINGLE CHARACTER SYNC | |
|---------------------------|--|
| 1 = SINGLE SYNC CHARACTER | |
| 0 = DOUBLE SYNC CHARACTER | |

2. COMMAND INSTRUCTION WORD

This format defines a status word that is used to control the actual operation of 8251. All control words written into 8251 after the mode instruction will load the command instruction.

The command instructions can be written into 8251 at any time in the data block during the operation of the 8251. To return to the mode instruction format, the master reset bit in the command instruction word can be set to initiate an internal reset operation which automatically places the 8251 back into the mode instruction format. Command instructions must follow the mode instructions or sync characters.

Thus the control word 37 (HEX) enables the transmit enable and receive enable bits, forces DTR output to zero, resets the error flags, and forces RTS output to zero.

| | | | | | | | |
|----|----|-----|----|------|-----|-----|------|
| EH | IR | RTS | ER | SBRK | RXE | DTR | TXEN |
|----|----|-----|----|------|-----|-----|------|

| |
|-----------------|
| TRANSMIT ENABLE |
|-----------------|

1=Enable 0 = Disable

| |
|---------------------|
| DATA TERMINAL READY |
|---------------------|

HIGH will force DTR
Output to Zero

| |
|----------------|
| RECEIVE ENABLE |
|----------------|

1=Enable 0 = Disable

| |
|----------------------|
| SEND BREAK CHARACTER |
|----------------------|

1 = Forces TXD LOW
0 = Normal Operation

| |
|-------------|
| ERROR RESET |
|-------------|

1=Reset Error Flags
PE,OE,FE

| |
|-----------------|
| REQUEST TO SEND |
|-----------------|

HIGH will force RTS
Output to Zero

| |
|----------------|
| INTERNAL RESET |
|----------------|

HIGH Returns 8251 to
Mode Instruction Format

| |
|-----------------|
| ENTER HUNT MODE |
|-----------------|

1= Enable a Search for
Sync Characters(Has
No Effect in Async mode)

COMMAND INSTRUCTION FORMAT

ALGORITHM:

1. Initialise timer (8253) IC
2. Move the mode command word (4E H) to A -reg
3. Output it to port address C2
4. Move the command instruction word (37 H) to A -reg
5. Output it to port address C2
6. Move the the data to be transferred to A -reg
7. Output it to port address C0
8. Reset the system
9. Get the data through input port address C0
10. Store the value in memory
11. Reset the system

PROGRAM:

| | |
|-----|--------|
| MVI | A,36H |
| OUT | CEH |
| MVI | A,0AH |
| OUT | C8H |
| MVI | A,00 |
| OUT | C8H |
| LXI | H,4200 |
| MVI | A,4E |
| OUT | C2 |
| MVI | A,37 |
| OUT | C2 |
| MVI | A,41 |
| OUT | C0 |
| RST | 1 |
| ORG | 4200 |
| IN | C0 |
| STA | 4500 |
| RST | 1 |

OBSERVATION:

Output: 4500 41

RESULT:

Thus the 8251 was initiated and the transmission and reception of character was done successfully.

INTERFACING ADC WITH 8085 PROCESSOR

AIM:

To write a program to initiate ADC and to store the digital data in memory

PROGRAM:

| | | |
|-------|-----|------|
| | MVI | A,10 |
| | OUT | C8 |
| | MVI | A,18 |
| | OUT | C8 |
| | MVI | A,10 |
| | OUT | D0 |
| | XRA | A |
| | XRA | A |
| | XRA | A |
| | MVI | A,00 |
| | OUT | D0 |
| LOOP: | IN | D8 |
| | ANI | 01 |
| | CPI | 01 |
| | JNZ | LOOP |
| | IN | C0 |
| | STA | 4150 |
| | HLT | |

OBSERVATION:

Compare the data displayed at the LEDs with that stored at location 4150

RESULT:

Thus the ADC was initiated and the digital data was stored at desired location

INTERFACING DAC WITH 8085

AIM:

To interface DAC with 8085 to demonstrate the generation of square, saw tooth and triangular wave.

APPARATUS REQUIRED:

- 8085 Trainer Kit
- DAC Interface Board

THEORY:

DAC 0800 is an 8 – bit DAC and the output voltage variation is between – 5V and + 5V. The output voltage varies in steps of $10/256 = 0.04$ (appx.). The digital data input and the corresponding output voltages are presented in the Table1.

| Input Data in HEX | Output Voltage |
|-------------------|----------------|
| 00 | - 5.00 |
| 01 | - 4.96 |
| 02 | - 4.92 |
| ... | ... |
| 7F | 0.00 |
| ... | ... |
| FD | 4.92 |
| FE | 4.96 |
| FF | 5.00 |

Referring to Table1, with 00 H as input to DAC, the analog output is – 5V. Similarly, with FF H as input, the output is +5V. Outputting digital data 00 and FF at regular intervals, to DAC, results in different wave forms namely square, triangular, etc,. The port address of DAC is 08 H.

ALGORITHM:

(a) Square Wave Generation

1. Load the initial value (00) to Accumulator and move it to DAC
2. Call the delay program
3. Load the final value(FF) to accumulator and move it to DAC
4. Call the delay program.
5. Repeat Steps 2 to 5

(b) Saw tooth Wave Generation

1. Load the initial value (00) to Accumulator
2. Move the accumulator content to DAC
3. Increment the accumulator content by 1.
4. Repeat Steps 3 and 4.

(c) Triangular Wave Generation

2. Load the initial value (00) to Accumulator
3. Move the accumulator content to DAC
4. Increment the accumulator content by 1.
5. If accumulator content is zero proceed to next step. Else go to step 3.
6. Load value (FF) to Accumulator
7. Move the accumulator content to DAC
8. Decrement the accumulator content by 1.
9. If accumulator content is zero go to step2. Else go to step 7.

PROGRAM:

(a) Square Wave Generation

| | | |
|--------|------|---------------------|
| START: | MVI | A,00 |
| | OUT | Port address of DAC |
| | CALL | DELAY |
| | MVI | A,FF |
| | OUT | Port address of DAC |
| | CALL | DELAY |
| | JMP | START |
| DELAY: | MVI | B,05 |
| L1: | MVI | C,FF |
| L2: | DCR | C |
| | JNZ | L2 |
| | DCR | B |
| | JNZ | L1 |
| | RET | |

(b) Saw tooth Wave Generation

| | | |
|--------|-----|---------------------|
| START: | MVI | A,00 |
| L1: | OUT | Port address of DAC |
| | INR | A |
| | JNZ | L1 |
| | JMP | START |

(c) Triangular Wave Generation

| | | |
|--------|-----|---------------------|
| START: | MVI | L,00 |
| L1: | MOV | A,L |
| | OUT | Port address of DAC |
| | INR | L |
| | JNZ | L1 |
| | MVI | L,FF |
| L2: | MOV | A,L |
| | OUT | Port address of DAC |
| | DCR | L |
| | JNZ | L2 |
| | JMP | START |

RESULT:

Thus the square, triangular and saw tooth wave form were generated by interfacing DAC with 8085 trainer kit.

INTERFACING 8253 (TIMER IC) WITH 8085 PROCESSOR

AIM:

To interface 8253 Programmable Interval Timer to 8085 and verify the operation of 8253 in six different modes.

APPARATUS REQUIRED:

- 1) 8085 Microprocessor toolkit.
- 2) 8253 Interface board.
- 3) VXT parallel bus.
- 4) Regulated D.C power supply.
- 5) CRO.

MODE 0-Interrupt On Terminal Count:-

The output will be initially low after mode set operation. After loading the counter, the output will remain low while counting and on terminal count, the output will become high until reloaded again.

Let us see the channel in mode0. Connect the CLK 0 to the debounce circuit and execute the following program.

PROGRAM:

```
MVI A, 30H ;Channel 0 in mode 0.  
OUT CEH  
MVI A, 05H ;LSB of count.  
OUT C8H  
MVI A, 00H ;MSB of count.  
OUT C8H  
HLT
```

It is observed in CRO that the output of channel 0 is initially low. After giving 'x' clock pulses, we may notice that the output goes high.

MODE 1-Programmable One Shot:-

After loading the count, the output will remain low following the rising edge of the gate input. The output will go high on the terminal count. It is retriggerable; hence the output will remain low for the full count after any rising edge of the gate input.

The following program initializes channel 0 of 8253 in Mode 1 and also initializes triggering of gate. OUT 0 goes low as clock pulses and after triggering It goes back to high level after five clock pulses. Execute the program and give clock pulses through the debounce logic and verify using CRO.

PROGRAM:

```
MVI A, 32H ;Channel 0 in mode 1.  
OUT CEH ;  
MVI A, 05H ;LSB of count.  
OUT C8H  
MVI A, 00H ;MSB of count.  
OUT C8H  
OUT DOH ;Trigger Gate 0.  
HLT
```

MODE 2-Rate Generator:

It is a simple divide by N counter. The output will be low for one period of the input clock. The period from one output pulse to next equals the number of input count in the count register. If the count register is reloaded between output pulses, the present period will not be affected, but the subsequent period will reflect a new value.

MODE 3-Square Generator:

It is similar to mode 2 except that the output will remain high until one half of the count and goes low for the other half provided the count is an even number. If the count is odd the output will be high for $(\text{count} + 1)/2$ counts. This mode is used for generating baud rate of 8251.

PROGRAM:

```
MVI A, 36H ;Channel 0 in mode 3.  
OUT CEH ;  
MVI A, 0AH ;LSB of count.  
OUT C8H  
MVI A, 00H ;MSB of count.  
OUT C8H  
HLT
```

We utilize mode 3 to generate a square wave of frequency 150 kHz at Channel 0. Set the jumper so that the clock of 8253 is given a square wave of Frequency 1.5 MHz. This program divides the program clock by 10 and thus the Output at channel 0 is 150 KHz.

MODE 4-Software Triggered Strobe:

The output is high after the mode is set and also during counting. On Terminal count, the output will go low for one clock period and becomes high again. This mode can be used for interrupt generation.

MODE 5-Hardware Triggered Strobe:

Counter starts counting after rising edge of trigger input and the output goes low for one clock period. When the terminal count is reached, the counter is retrigerrable. On terminal count, the output will go low for one clock period and becomes high again. This mode can be used for interrupt generation.

RESULT:

Thus the 8253 PIT was interfaced to 8085 and the operations for mode 0, Mode 1 and mode 3 was verified.

INTERFACING 8279 KEYBOARD/DISPLAY CONTROLLER WITH 8085 MICROPROCESSOR

AIM:

To interface 8279 Programmable Keyboard Display Controller to 8085 Microprocessor.

APPARATUS REQUIRED:

- 1) 8085 Microprocessor toolkit.
- 2) 8279 Interface board.
- 3) VXT parallel bus.
- 4) Regulated D.C power supply.

PROGRAM:

| | | |
|--------|------|----------------------------|
| START: | LXI | H,4130H |
| | MVI | D,0FH ;Initialize counter. |
| | MVI | A,10H |
| | OUT | C2H ;Set Mode and Display. |
| | MVI | A,CCH;Clear display. |
| | OUT | C2H |
| | MVI | A,90H ;Write Display |
| | OUT | C2H |
| LOOP: | MOV | A,M |
| | OUT | C0H |
| | CALL | DELAY |
| | INX | H |
| | DCR | D |
| | JNZ | LOOP |
| | JMP | START |
| DELAY: | MVI | B, A0H |
| LOOP2: | MVI | C, FFH |
| LOOP1: | DCR | C |
| | JNZ | LOOP1 |
| | DCR | B |
| | JNZ | LOOP2 |
| | RET | |

Pointer equal to 4130 .FF repeated eight times.

| | |
|------|------|
| 4130 | - FF |
| 4131 | -FF |
| 4132 | -FF |
| 4133 | -FF |
| 4134 | -FF |
| 4135 | -FF |
| 4136 | -FF |
| 4137 | -FF |
| 4138 | -98 |
| 4139 | -68 |
| 413A | -7C |
| 413B | -C8 |
| 413C | -1C |
| 413D | -29 |
| 413E | -FF |
| 413F | -FF |

RESULT:

Thus 8279 controller was interfaced with 8085 and program for rolling display was executed successfully.

8051 MICROCONTROLLER PROGRAMS

ADDITION OF TWO 8 – BIT NUMBERS

AIM:

To perform addition of two 8 – bit numbers using 8051 instruction set.

ALGORITHM:

1. Clear C – register for Carry
2. Get the data immediately .
3. Add the two data
4. Store the result in memory pointed by DPTR

PROGRAM:

| | | |
|-------|------|------------|
| | ORG | 4100 |
| | CLR | C |
| | MOV | A,#data1 |
| | ADD | A,#data2 |
| | MOV | DPTR,#4500 |
| | MOVX | @DPTR,A |
| HERE: | SJMP | HERE |

OBSERVATION:

Input: 66
 23

Output: 89 (4500)

RESULT:

Thus the program to perform addition of two 8 – bit numbers using 8051 instruction set was executed.

SUBTRACTION OF TWO 8 – BIT NUMBERS

AIM:

To perform Subtraction of two 8 – bit numbers using 8051 instruction set.

ALGORITHM:

1. Clear C – register for Carry
2. Get the data immediately .
3. Subtract the two data
4. Store the result in memory pointed by DPTR

PROGRAM:

| | |
|-------|------------|
| ORG | 4100 |
| CLR | C |
| MOV | A,#data1 |
| SUBB | A,#data2 |
| MOV | DPTR,#4500 |
| MOVX | @DPTR,A |
| HERE: | SJMP HERE |

OBSERVATION:

Input: 66
 23

Output: 43 (4500)

RESULT:

Thus the program to perform subtraction of two 8 – bit numbers using 8051 instruction set was executed.

MULTIPLICATION OF TWO 8 – BIT NUMBERS

AIM:

To perform multiplication of two 8 – bit numbers using 8051 instruction set.

ALGORITHM:

1. Get the data in A – reg.
2. Get the value to be multiplied in B – reg.
3. Multiply the two data
4. The higher order of the result is in B – reg.
5. The lower order of the result is in A – reg.
6. Store the results.

PROGRAM:

| | |
|-------|------------|
| ORG | 4100 |
| CLR | C |
| MOV | A,#data1 |
| MOV | B,#data2 |
| MUL | AB |
| MOV | DPTR,#4500 |
| MOVX | @DPTR,A |
| INC | DPTR |
| MOV | A,B |
| MOVX | @DPTR,A |
| HERE: | SJMP HERE |

OBSERVATION:

Input: 80
 80

Output: 00 (4500)
 19 (4501)

RESULT:

Thus the program to perform multiplication of two 8 – bit numbers using 8051 instruction set was executed.

DIVISION OF TWO 8 – BIT NUMBERS

AIM:

To perform division of two 8 – bit numbers using 8051 instruction set.

ALGORITHM:

1. Get the data in A – reg.
2. Get the value to be divided in B – reg.
3. Divide the two data
4. The quotient is in A – reg.
5. The remainder is in B – reg.
6. Store the results.

PROGRAM:

| | |
|-------|------------|
| ORG | 4100 |
| CLR | C |
| MOV | A,#data1 |
| MOV | B,#data2 |
| DIV | AB |
| MOV | DPTR,#4500 |
| MOVX | @DPTR,A |
| INC | DPTR |
| MOV | A,B |
| MOVX | @DPTR,A |
| HERE: | SJMP HERE |

OBSERVATION:

Input: 05
 03

Output: 01 (4500)
 02 (4501)

RESULT:

Thus the program to perform multiplication of two 8 – bit numbers using 8051 instruction set was executed.

RAM ADDRESSING

AIM:

To exhibit the RAM direct addressing and bit addressing schemes of 8051 microcontroller.

ALGORITHM:

1. For Bit addressing, Select Bank 1 of RAM by setting 3rd bit of PSW
2. Using Register 0 of Bank 1 and accumulator perform addition
3. For direct addressing provide the address directly (30 in this case)
4. Use the address and Accumulator to perform addition
5. Verify the results

PROGRAM:

Bit Addressing:

| | |
|-------|------------|
| SETB | PSW.3 |
| MOV | R0,#data1 |
| MOV | A,#data2 |
| ADD | A,R0 |
| MOV | DPTR,#4500 |
| MOVX | @DPTR,A |
| HERE: | SJMP HERE |

Direct Addressing:

| | |
|-------|------------|
| MOV | 30,#data1 |
| MOV | A,#data2 |
| ADD | A,30 |
| MOV | DPTR,#4500 |
| MOVX | @DPTR,A |
| HERE: | SJMP HERE |

OBSERVATION:

Bit addressing:

Input: 54
 25

Output: 79 (4500)

Direct addressing:

Input: 54
 25

Output: 79 (4500)

RESULT:

Thus the program to exhibit the different RAM addressing schemes of 8051 was executed.

INTERFACING STEPPER MOTOR WITH 8051

AIM:

To interface stepper motor with 8051 parallel port and to vary speed of motor, direction of motor.

APPARATUS REQUIRED:

- 8051 Trainer Kit
- Stepper Motor Interface Board

THEORY:

A motor in which the rotor is able to assume only discrete stationary angular position is a stepper motor. The rotor motion occurs in a stepwise manner from one equilibrium position to next.

The motor under our consideration uses 2 – phase scheme of operation. In this scheme, any two adjacent stator windings are energized. The switching condition for the above said scheme is shown in Table.

| Clockwise | | | | Anti - Clockwise | | | |
|-----------|----|----|----|------------------|----|----|----|
| A1 | B1 | A2 | B2 | A1 | B1 | A2 | B2 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |

In order to vary the speed of the motor, the values stored in the registers R1, R2, R3 can be changed appropriately.

ALGORITHM:

1. Store the look up table address in DPTR
2. Move the count value (04) to one of the register (R0)
3. Load the control word for motor rotation in accumulator
4. Push the address in DPTR into stack
5. Load FFC0 in to DPTR.
6. Call the delay program
7. Send the control word for motor rotation to the external device.
8. Pop up the values in stack and increment it.
9. Decrement the count in R0. If zero go to next step else proceed to step 3.
10. Perform steps 1 to 9 repeatedly.

PROGRAM:

| | | |
|--------|------|-------------|
| | ORG | 4100 |
| START: | MOV | DPTR,#4500H |
| | MOV | R0,#04 |
| AGAIN: | MOVX | A,@DPTR |
| | PUSH | DPH |
| | PUSH | PDL |
| | MOV | DPTR,#FFC0H |
| | MOV | R2, 04H |
| | MOV | R1,#FFH |
| DLY1: | MOV | R3, #FFH |
| DLY: | DJNZ | R3,DLY |
| | DJNZ | R1,DLY1 |
| | DJNZ | R2,DLY1 |
| | MOVX | @DPTR,A |
| | POP | DPL |
| | POP | DPH |
| | INC | DPTR |
| | DJNZ | R0, AGAIN |
| | SJMP | START |

DATA:

4500: 09, 05, 06, 0A

RESULT:

Thus the speed and direction of motor were controlled using 8051 trainer kit.

INTERFACING DAC WITH 8051

AIM:

To interface DAC with 8051 parallel port to demonstrate the generation of square, saw tooth and triangular wave.

APPARATUS REQUIRED:

- 8051 Trainer Kit
- DAC Interface Board

THEORY:

DAC 0800 is an 8 – bit DAC and the output voltage variation is between – 5V and + 5V. The output voltage varies in steps of $10/256 = 0.04$ (appx.). The digital data input and the corresponding output voltages are presented in the Table below

| Input Data in HEX | Output Voltage |
|-------------------|----------------|
| 00 | - 5.00 |
| 01 | - 4.96 |
| 02 | - 4.92 |
| ... | ... |
| 7F | 0.00 |
| ... | ... |
| FD | 4.92 |
| FE | 4.96 |
| FF | 5.00 |

Referring to Table1, with 00 H as input to DAC, the analog output is – 5V. Similarly, with FF H as input, the output is +5V. Outputting digital data 00 and FF at regular intervals, to DAC, results in different wave forms namely square, triangular, etc.,

ALGORITHM:

(a) Square Wave Generation

1. Move the port address of DAC to DPTR
2. Load the initial value (00) to Accumulator and move it to DAC
3. Call the delay program
4. Load the final value(FF) to accumulator and move it to DAC
5. Call the delay program.
6. Repeat Steps 2 to 5

(b) Saw tooth Wave Generation

1. Move the port address of DAC to DPTR
2. Load the initial value (00) to Accumulator
3. Move the accumulator content to DAC
4. Increment the accumulator content by 1.
5. Repeat Steps 3 and 4.

(c) Triangular Wave Generation

1. Move the port address of DAC to DPTR
2. Load the initial value (00) to Accumulator
3. Move the accumulator content to DAC
4. Increment the accumulator content by 1.
5. If accumulator content is zero proceed to next step. Else go to step 3.
6. Load value (FF) to Accumulator
7. Move the accumulator content to DAC
8. Decrement the accumulator content by 1.
9. If accumulator content is zero go to step2. Else go to step 7.

PROGRAM:**(a) Square Wave Generation**

| | | |
|--------|-------|--------------------------|
| | ORG | 4100 |
| | MOV | DPTR,PORT ADDRESS OF DAC |
| START: | MOV | A,#00 |
| | MOVX | @DPTR,A |
| | LCALL | DELAY |
| | MOV | A,#FF |
| | MOVX | @DPTR,A |
| | LCALL | DELAY |
| | LJUMP | START |
| DELAY: | MOV | R1,#05 |
| LOOP: | MOV | R2,#FF |
| HERE: | DJNZ | R2,HERE |
| | DJNZ | R1,LOOP |
| | RET | |
| | SJMP | START |

(b) Saw tooth Wave Generation

| | | |
|-------|------|--------------------------|
| | ORG | 4100 |
| | MOV | DPTR,PORT ADDRESS OF DAC |
| | MOV | A,#00 |
| LOOP: | MOVX | @DPTR,A |
| | INC | A |
| | SJMP | LOOP |

(c) Triangular Wave Generation

| | | |
|--------|------|--------------------------|
| | ORG | 4100 |
| | MOV | DPTR,PORT ADDRESS OF DAC |
| START: | MOV | A,#00 |
| LOOP1: | MOVX | @DPTR,A |
| | INC | A |
| | JNZ | LOOP1 |
| | MOV | A,#FF |
| LOOP2: | MOVX | @DPTR,A |
| | DEC | A |
| | JNZ | LOOP2 |
| | LJMP | START |

RESULT:

Thus the square, triangular and saw tooth wave form were generated by interfacing DAC with 8051 trainer kit.

PROGRAMMING 8051 USING KEIL SOFTWARE

AIM:

To perform arithmetic operations in 8051 using keil software.

PROCEDURE:

1. Click Keil\u201cVision2 icon in the desktop
2. From Project Menu open New project
3. Select the target device as ATMEL 89C51
4. From File Menu open New File
5. Type the program in Text Editor
6. Save the file with extension “.asm”
7. In project window click the tree showing TARGET
8. A source group will open.
9. Right Click the Source group and click “Add files to Source group”
10. A new window will open. Select our file with extension “.asm”
11. Click Add.
12. Go to project window and right click Source group again
13. Click Build Target (F7).
14. Errors if any will be displayed.
15. From Debug menu, select START/STOP Debug option.
16. In project window the status of all the registers will be displayed.
17. Click Go from Debug Menu.
18. The results stored in registers will be displayed in Project window.
19. Stop the Debug process before closing the application.

PROGRAM:

| | |
|-----|--------|
| ORG | 4100 |
| CLR | C |
| MOV | A,#05H |
| MOV | B,#02H |
| DIV | AB |

OBSERVATION:

A: 02
B: 01
SP:07

Note that Stack pointer is initiated to 07H

RESULT:

Thus the arithmetic operation for 8051 was done using Keil Software.

SYSTEM DESIGN USING MICROCONTROLLER

AIM:

To Design a microcontroller based system for simple applications like security systems combination lock etc.

PROCEDURE:

1. Read number of bytes in the password
2. Initialize the password
3. Initialize the Keyboard Display IC (8279) to get key and Display
4. Blank the display
5. Read the key from user
6. Compare with the initialized password
7. If it is not equal, Display 'E' to indicate Error.
8. Repeat the steps 6 and 7 to read next key
9. If entered password equal to initialized password, Display 'O' to indicate open.

PROGRAM:

| | |
|-----|------------|
| MOV | 51H,# |
| MOV | 52H,# |
| MOV | 53H,# |
| MOV | 54H,# |
| MOV | R1,#51 |
| MOV | R0,#50 |
| MOV | R3,#04 |
| MOV | R2,#08 |
| MOV | DPTR,#FFC2 |
| MOV | A,#00 |

| | | |
|--------|------|------------|
| | MOVX | @DPTR,A |
| | MOV | A,#CC |
| | MOVX | @DPTR,A |
| | MOV | A,#90 |
| | MOVX | @DPTR,A |
| | MOV | A,#FF |
| | MOV | DPTR,#FFCO |
| LOOP: | MOVX | @DPTR,A |
| | DJNZ | R2,LOOP |
| AGAIN: | MOV | DPTR,#FFC2 |
| WAIT: | MOVX | A,@DPTR |
| | ANL | A,#07 |
| | JZ | WAIT |
| | MOV | A,#40 |
| | MOVX | @DPTR,A |
| | MOV | DPTR,#FFCO |
| | MOVX | A,@DPTR |
| | MOV | @R0,A |
| | MOV | A,@R1 |
| | CJNE | A,50H,NEQ |
| | INC | R1 |
| | DJNZ | R3, AGAIN |
| | MOV | DPTR,#FFCO |
| | MOV | A,#OC |
| | MOVX | @DPTR,A |
| XX: | SJMP | XX |

| | | |
|------|------|------------|
| NEQ: | MOV | DPTR,#FFCO |
| | MOV | A,#68 |
| | MOVX | @DPTR,A |
| YY: | SJMP | YY |

RESULT:

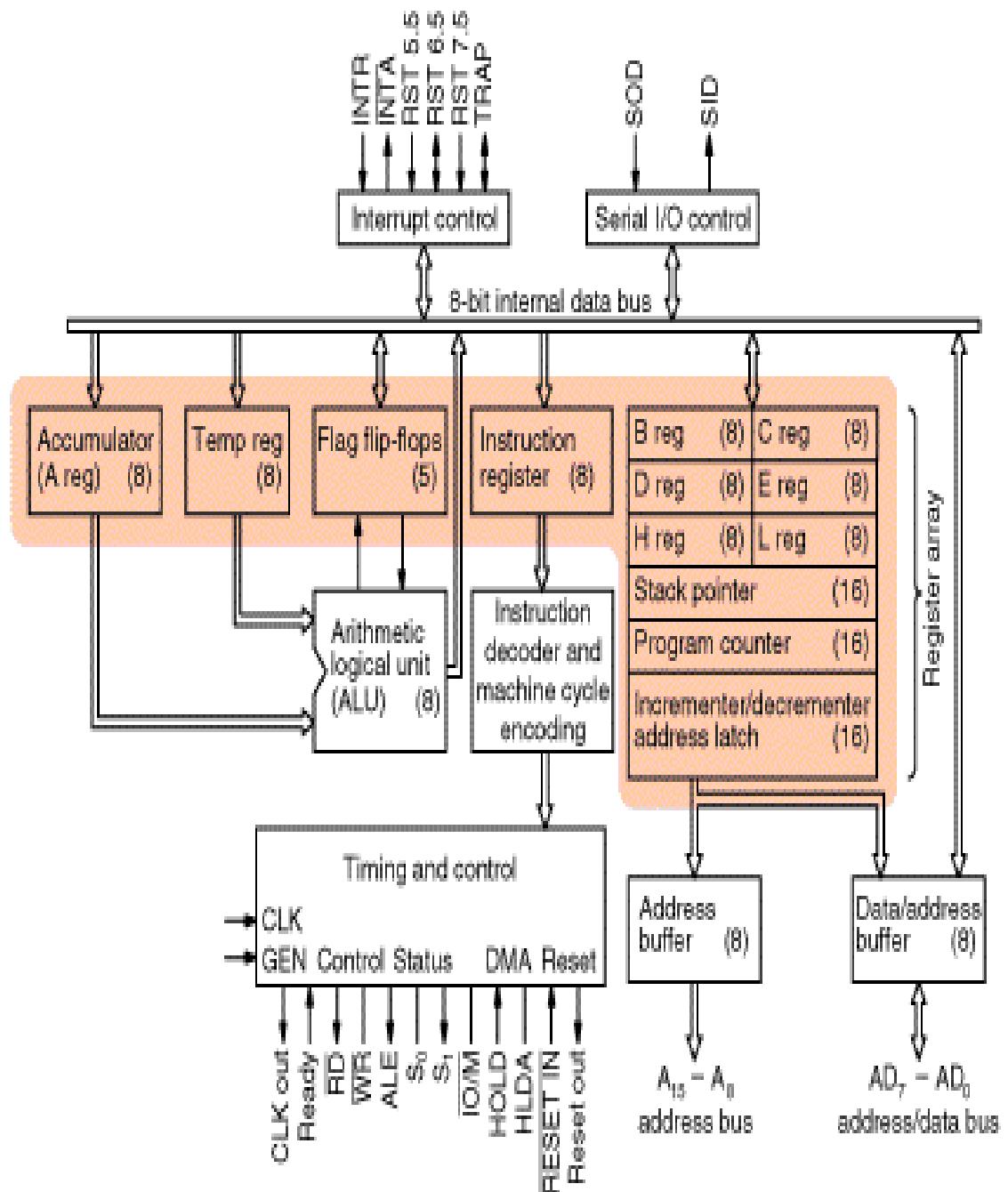
Thus the program for security lock system was executed

Tutorial
On
Introduction to 8085 Architecture and Programming

Contents

1. Internal architecture of 8085 microprocessor
2. 8085 system bus
3. 8085 pin description.
4. 8085 functional description.
5. Programming model of 8085 microprocessor
6. Addressing modes.
7. Instruction set classification.
8. Instruction format.
9. Sample programs.

1. Internal Architecture of 8085 Microprocessor



Control Unit

Generates signals within uP to carry out the instruction, which has been decoded. In reality causes certain connections between blocks of the uP to be opened or closed, so that data goes where it is required, and so that ALU operations occur.

Arithmetic Logic Unit

The ALU performs the actual numerical and logic operation such as ‘add’, ‘subtract’, ‘AND’, ‘OR’, etc. Uses data from memory and from Accumulator to perform arithmetic. Always stores result of operation in Accumulator.

Registers

The 8085/8080A-programming model includes six registers, one accumulator, and one flag register, as shown in Figure. In addition, it has two 16-bit registers: the stack pointer and the program counter. They are described briefly as follows.

The 8085/8080A has six general-purpose registers to store 8-bit data; these are identified as B,C,D,E,H, and L as shown in the figure. They can be combined as register pairs - BC, DE, and HL - to perform some 16-bit operations. The programmer can use these registers to store or copy data into the registers by using data copy instructions.

Accumulator

The accumulator is an 8-bit register that is a part of arithmetic/logic unit (ALU). This register is used to store 8-bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator. The accumulator is also identified as register A.

Flags

The ALU includes five flip-flops, which are set or reset after an operation according to data conditions of the result in the accumulator and other registers. They are called Zero(Z), Carry (CY), Sign (S), Parity (P), and Auxiliary Carry (AC) flags; they are listed in the Table and their bit positions in the flag register are shown in the Figure below. The most commonly used flags are Zero, Carry, and Sign. The microprocessor uses these flags to test data conditions.

For example, after an addition of two numbers, if the sum in the accumulator is larger than eight bits, the flip-flop used to indicate a carry -- called the Carry flag (CY) -- is set to one. When an arithmetic operation results in zero, the flip-flop called the Zero(Z) flag is set to one. The first Figure shows an 8-bit register, called the flag register, adjacent to the accumulator. However, it is not used as a register; five bit positions out of eight are used to store the outputs of the five flip-flops. The flags are stored in the 8-bit register so that the programmer can examine these flags (data conditions) by accessing the register through an instruction.

These flags have critical importance in the decision-making process of the microprocessor. The conditions (set or reset) of the flags are tested through the software instructions. For example, the instruction JC (Jump on Carry) is implemented to change the sequence of a program when CY flag is set. The thorough understanding of flag is essential in writing assembly language programs.

Program Counter (PC)

This 16-bit register deals with sequencing the execution of instructions. This register is a memory pointer. Memory locations have 16-bit addresses, and that is why this is a 16-bit register.

The microprocessor uses this register to sequence the execution of the instructions. The function of the program counter is to point to the memory address from which the next byte is to be fetched. When a byte (machine code) is being fetched, the program counter is incremented by one to point to the next memory location

Stack Pointer (SP)

The stack pointer is also a 16-bit register used as a memory pointer. It points to a memory location in R/W memory, called the stack. The beginning of the stack is defined by loading 16-bit address in the stack pointer. The stack concept is explained in the chapter "Stack and Subroutines."

Instruction Register/Decoder

Temporary store for the current instruction of a program. Latest instruction sent here from memory prior to execution. Decoder then takes instruction and 'decodes' or interprets the instruction. Decoded instruction then passed to next stage.

Memory Address Register

Holds address, received from PC, of next program instruction. Feeds the address bus with addresses of location of the program under execution.

Control Generator

Generates signals within uP to carry out the instruction which has been decoded. In reality causes certain connections between blocks of the uP to be opened or closed, so that data goes where it is required, and so that ALU operations occur.

Register Selector

This block controls the use of the register stack in the example. Just a logic circuit which switches between different registers in the set will receive instructions from Control Unit.

General Purpose Registers

uP requires extra registers for versatility. Can be used to store additional data during a program. More complex processors may have a variety of differently named registers.

Microprogramming

How does the μ P know what an instruction means, especially when it is only a binary number? The microprogram in a uP/uC is written by the chip designer and tells the uP/uC the meaning of each instruction. uP/uC can then carry out operation.

2. 8085 System Bus

Typical system uses a number of busses, collection of wires, which transmit binary numbers, one bit per wire. A typical microprocessor communicates with memory and other devices (input and output) using three busses: Address Bus, Data Bus and Control Bus.

Address Bus

One wire for each bit, therefore 16 bits = 16 wires. Binary number carried alerts memory to 'open' the designated box. Data (binary) can then be put in or taken out. The Address Bus consists of 16 wires, therefore 16 bits. Its "width" is 16 bits. A 16 bit binary number allows 2^{16} different numbers, or 32000 different numbers, ie 0000000000000000 up to 1111111111111111. Because memory consists of boxes, each with a unique address, the size of the address bus determines the size of memory, which can be used. To communicate with memory the microprocessor sends an address on the address bus, eg 0000000000000011 (3 in decimal), to the memory. The memory then selects box number 3 for reading or writing data. Address bus is unidirectional, ie numbers only sent from microprocessor to memory, not other way.

Question?: If you have a memory chip of size 256 kilobytes (256 x 1024 x 8 bits), how many wires does the address bus need, in order to be able to specify an address in this memory? **Note:** the memory is organized in groups of 8 bits per location, therefore, how many locations must you be able to specify?

Data Bus

Data Bus: carries 'data', in binary form, between μ P and other external units, such as memory. Typical size is 8 or 16 bits. Size determined by size of boxes in memory and μ P size helps determine performance of μ P. The Data Bus typically consists of 8 wires. Therefore, 2^8 combinations of binary digits. Data bus used to transmit "data", ie information, results of arithmetic, etc, between memory and the microprocessor. Bus is bi-directional. Size of the data bus determines what arithmetic can be done. If only 8 bits wide then largest number is 11111111 (255 in decimal). Therefore, larger numbers have to be broken down into chunks of 255. This slows microprocessor. Data Bus also carries instructions from memory to the microprocessor. Size of the bus therefore limits the number of possible instructions to 256, each specified by a separate number.

Control Bus

Control Bus are various lines which have specific functions for coordinating and controlling uP operations. Eg: Read/NotWrite line, single binary digit. Control whether memory is being ‘written to’ (data stored in mem) or ‘read from’ (data taken out of mem) 1 = Read, 0 = Write. May also include clock line(s) for timing/synchronising, ‘interrupts’, ‘reset’ etc. Typically μ P has 10 control lines. Cannot function correctly without these vital control signals.

The Control Bus carries control signals partly unidirectional, partly bi-directional. Control signals are things like "read or write". This tells memory that we are either **reading from** a location, specified on the address bus, or **writing to** a location specified. Various other signals to control and coordinate the operation of the system. Modern day microprocessors, like 80386, 80486 have much larger busses. Typically 16 or 32 bit busses, which allow larger number of instructions, more memory location, and faster arithmetic. Microcontrollers organized along same lines, except: because microcontrollers have memory etc inside the chip, the busses may all be internal. In the microprocessor the three busses are external to the chip (except for the internal data bus). In case of external busses, the chip connects to the busses via buffers, which are simply an electronic connection between external bus and the internal data bus.

3. 8085 Pin description.

Properties

- Single + 5V Supply
- 4 Vectored Interrupts (One is Non Maskable)
- Serial In/Serial Out Port
- Decimal, Binary, and Double Precision Arithmetic
- Direct Addressing Capability to 64K bytes of memory

The Intel 8085A is a new generation, complete 8 bit parallel central processing unit (CPU). The 8085A uses a multiplexed data bus. The address is split between the 8bit address bus and the 8bit data bus. Figures are at the end of the document.

Pin Description

The following describes the function of each pin:

A6 - A1s (Output 3 State)

Address Bus; The most significant 8 bits of the memory address or the 8 bits of the I/O address,3 stated during Hold and Halt modes.

AD0 - 7 (Input/Output 3state)

Multiplexed Address/Data Bus; Lower 8 bits of the memory address (or I/O address) appear on the bus during the first clock cycle of a machine state. It then becomes the data bus during the second and third clock cycles. 3 stated during Hold and Halt modes.

ALE (Output)

Address Latch Enable: It occurs during the first clock cycle of a machine state and enables the address to get latched into the on chip latch of peripherals. The falling edge of ALE is set to guarantee setup and hold times for the address information. ALE can also be used to strobe the status information. ALE is never 3stated.

SO, S1 (Output)

Data Bus Status. Encoded status of the bus cycle:

| S1 | S0 | |
|----|----|-------|
| 0 | 0 | HALT |
| 0 | 1 | WRITE |
| 1 | 0 | READ |
| 1 | 1 | FETCH |

S1 can be used as an advanced R/W status.

RD (Output 3state)

READ; indicates the selected memory or 1/0 device is to be read and that the Data Bus is available for the data transfer.

WR (Output 3state)

WRITE; indicates the data on the Data Bus is to be written into the selected memory or 1/0 location. Data is set up at the trailing edge of WR. 3stated during Hold and Halt modes.

READY (Input)

If Ready is high during a read or write cycle, it indicates that the memory or peripheral is ready to send or receive data. If Ready is low, the CPU will wait for Ready to go high before completing the read or write cycle.

HOLD (Input)

HOLD; indicates that another Master is requesting the use of the Address and Data Buses. The CPU, upon receiving the Hold request, will relinquish the use of buses as soon as the completion of the current machine cycle. Internal processing can continue.

The processor can regain the buses only after the Hold is removed. When the Hold is acknowledged, the Address, Data, RD, WR, and IO/M lines are 3stated.

HLDA (Output)

HOLD ACKNOWLEDGE; indicates that the CPU has received the Hold request and that it will relinquish the buses in the next clock cycle. HLDA goes low after the Hold request is removed. The CPU takes the buses one half clock cycle after HLDA goes low.

INTR (Input)

INTERRUPT REQUEST; is used as a general purpose interrupt. It is sampled only during the next to the last clock cycle of the instruction. If it is active, the Program Counter (PC) will be inhibited from incrementing and an INTA will be issued. During this cycle a RESTART or CALL instruction can be inserted to jump to the interrupt service routine. The INTR is enabled and disabled by software. It is disabled by Reset and immediately after an interrupt is accepted.

INTA (Output)

INTERRUPT ACKNOWLEDGE; is used instead of (and has the same timing as) RD during the Instruction cycle after an INTR is accepted. It can be used to activate the 8259 Interrupt chip or some other interrupt port.

RST 5.5

RST 6.5 - (Inputs)

RST 7.5

RESTART INTERRUPTS; These three inputs have the same timing as INTR except they cause an internal RESTART to be automatically inserted.

RST 7.5 ~ Highest Priority

RST 6.5

RST 5.5 o Lowest Priority

The priority of these interrupts is ordered as shown above. These interrupts have a higher priority than the INTR.

TRAP (Input)

Trap interrupt is a nonmaskable restart interrupt. It is recognized at the same time as INTR. It is unaffected by any mask or Interrupt Enable. It has the highest priority of any interrupt.

RESET IN (Input)

Reset sets the Program Counter to zero and resets the Interrupt Enable and HLDA flipflops. None of the other flags or registers (except the instruction register) are affected. The CPU is held in the reset condition as long as Reset is applied.

RESET OUT (Output)

Indicates CPIJ is being reset. Can be used as a system RESET. The signal is synchronized to the processor clock.

X1, X2 (Input)

Crystal or R/C network connections to set the internal clock generator X1 can also be an external clock input instead of a crystal. The input frequency is divided by 2 to give the internal operating frequency.

CLK (Output)

Clock Output for use as a system clock when a crystal or R/ C network is used as an input to the CPU. The period of CLK is twice the X1, X2 input period.

IO/M (Output)

IO/M indicates whether the Read/Write is to memory or I/O Tristated during Hold and Halt modes.

SID (Input)

Serial input data line The data on this line is loaded into accumulator bit 7 whenever a RIM instruction is executed.

SOD (output)

Serial output data line. The output SOD is set or reset as specified by the SIM instruction.

Vcc

+5 volt supply.

Vss

Ground Reference.

| | | | |
|------------------|----|-------|-----------|
| X_1 | 1 | 40 | V_{CC} |
| X_2 | 2 | 39 | HOLD |
| RESET OUT | 3 | 38 | HLDA |
| SOD | 4 | 37 | CLK (OUT) |
| SID | 5 | 36 | RESET IN |
| TRAP | 6 | 35 | READY |
| RST 7.5 | 7 | 34 | IO/M |
| RST 6.5 | 8 | 33 | S_1 |
| RST 5.5 | 9 | 32 | RD |
| INTR | 10 | 8085A | 31 |
| INTA | 11 | 30 | ALE |
| AD_0 | 12 | 29 | S_0 |
| AD_1 | 13 | 28 | A_{15} |
| AD_2 | 14 | 27 | A_{14} |
| AD_3 | 15 | 26 | A_{13} |
| AD_4 | 16 | 25 | A_{12} |
| AD_5 | 17 | 24 | A_{11} |
| AD_6 | 18 | 23 | A_{10} |
| AD_7 | 19 | 22 | A_9 |
| V_{SS} | 20 | 21 | A_8 |

8085 Pinout

4. 8085 Functional Description

The 8085A is a complete 8 bit parallel central processor. It requires a single +5 volt supply. Its basic clock speed is 3 MHz thus improving on the present 8080's performance with higher system speed. Also it is designed to fit into a minimum system of three IC's: The CPU, a RAM/ IO, and a ROM or PROM/IO chip.

The 8085A uses a multiplexed Data Bus. The address is split between the higher 8bit Address Bus and the lower 8bit Address/Data Bus. During the first cycle the address is sent out. The lower 8bits are latched into the peripherals by the Address Latch Enable (ALE). During the rest of the machine cycle the Data Bus is used for memory or I/O data.

The 8085A provides RD, WR, and IO/Memory signals for bus control. An Interrupt Acknowledge signal (INTA) is also provided. Hold, Ready, and all Interrupts are synchronized. The 8085A also provides serial input data (SID) and serial output data (SOD) lines for simple serial interface.

In addition to these features, the 8085A has three maskable, restart interrupts and one non-maskable trap interrupt. The 8085A provides RD, WR and IO/M signals for Bus control.

Status Information

Status information is directly available from the 8085A. ALE serves as a status strobe. The status is partially encoded, and provides the user with advanced timing of the type of bus transfer being done. IO/M cycle status signal is provided directly also. Decoded S0, S1 Carries the following status information:

HALT, WRITE, READ, FETCH

S1 can be interpreted as R/W in all bus transfers. In the 8085A the 8 LSB of address are multiplexed with the data instead of status. The ALE line is used as a strobe to enter the lower half of the address into the memory or peripheral address latch. This also frees extra pins for expanded interrupt capability.

Interrupt and Serial I/O

The 8085A has 5 interrupt inputs: INTR, RST5.5, RST6.5, RST 7.5, and TRAP. INTR is identical in function to the 8080 INT. Each of the three RESTART inputs, 5.5, 6.5, 7.5, has a programmable mask. TRAP is also a RESTART interrupt except it is non-maskable.

The three RESTART interrupts cause the internal execution of RST (saving the program counter in the stack and branching to the RESTART address) if the interrupts

are enabled and if the interrupt mask is not set. The non-maskable TRAP causes the internal execution of a RST independent of the state of the interrupt enable or masks.

The interrupts are arranged in a fixed priority that determines which interrupt is to be recognized if more than one is pending as follows: TRAP highest priority, RST 7.5, RST 6.5, RST 5.5, INTR lowest priority. This priority scheme does not take into account the priority of a routine that was started by a higher priority interrupt. RST 5.5 can interrupt a RST 7.5 routine if the interrupts were re-enabled before the end of the RST 7.5 routine. The TRAP interrupt is useful for catastrophic errors such as power failure or bus error. The TRAP input is recognized just as any other interrupt but has the highest priority. It is not affected by any flag or mask. The TRAP input is both edge and level sensitive.

Basic System Timing

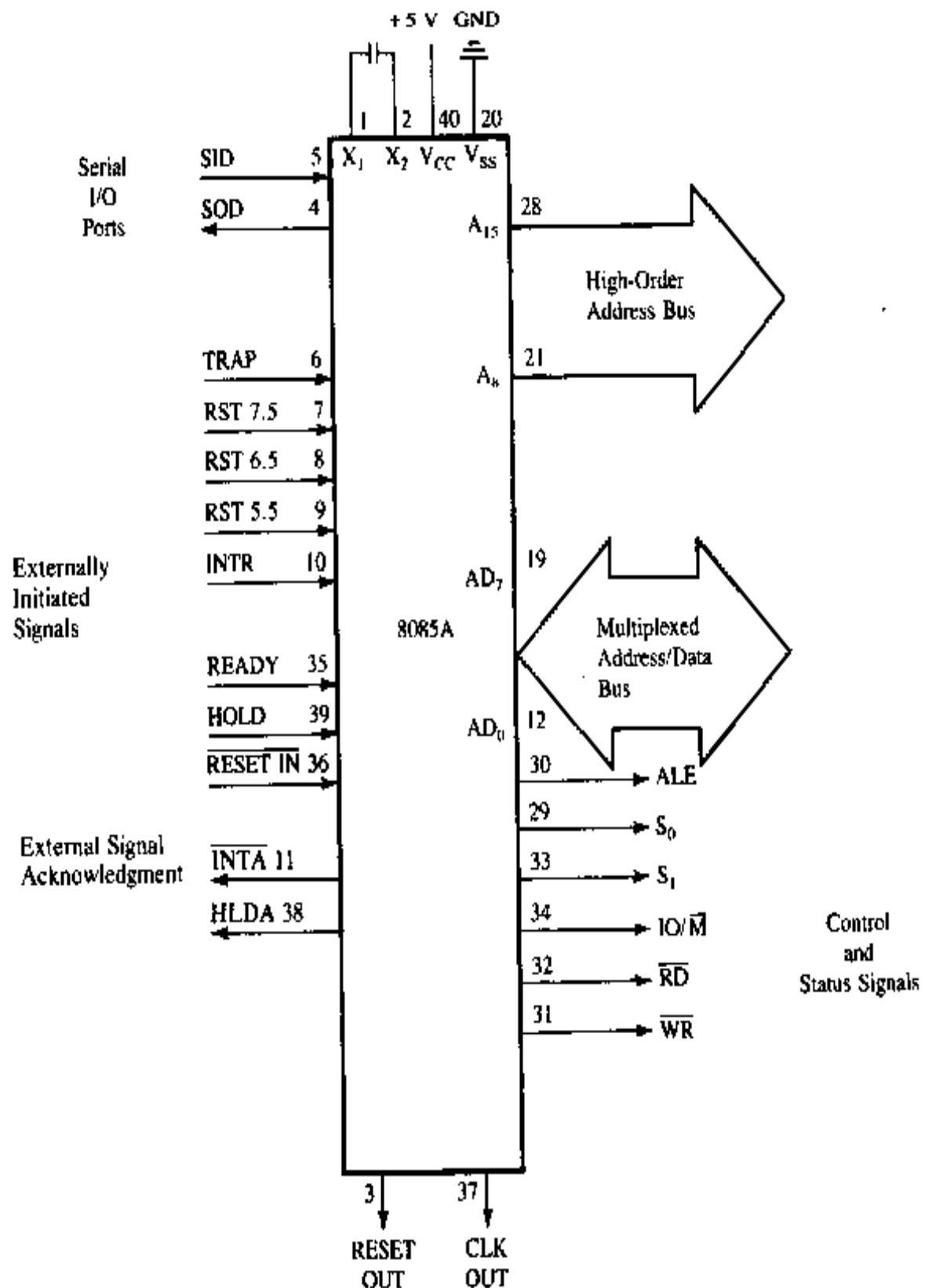
The 8085A has a multiplexed Data Bus. ALE is used as a strobe to sample the lower 8 bits of address on the Data Bus. Figure 2 shows an instruction fetch, memory read and I/O write cycle (OUT). Note that during the I/O write and read cycle that the I/O port address is copied on both the upper and lower half of the address. As in the 8080, the READY line is used to extend the read and write pulse lengths so that the 8085A can be used with slow memory. Hold causes the CPU to relinquish the bus when it is through with it by floating the Address and Data Buses.

System Interface

8085A family includes memory components, which are directly compatible to the 8085A CPU. For example, a system consisting of the three chips, 8085A, 8156, and 8355 will have the following features:

- 2K Bytes ROM
- 256 Bytes RAM
- 1 Timer/Counter
- 4 8bit I/O Ports
- 1 6bit I/O Port
- 4 Interrupt Levels
- Serial In/Serial Out Ports

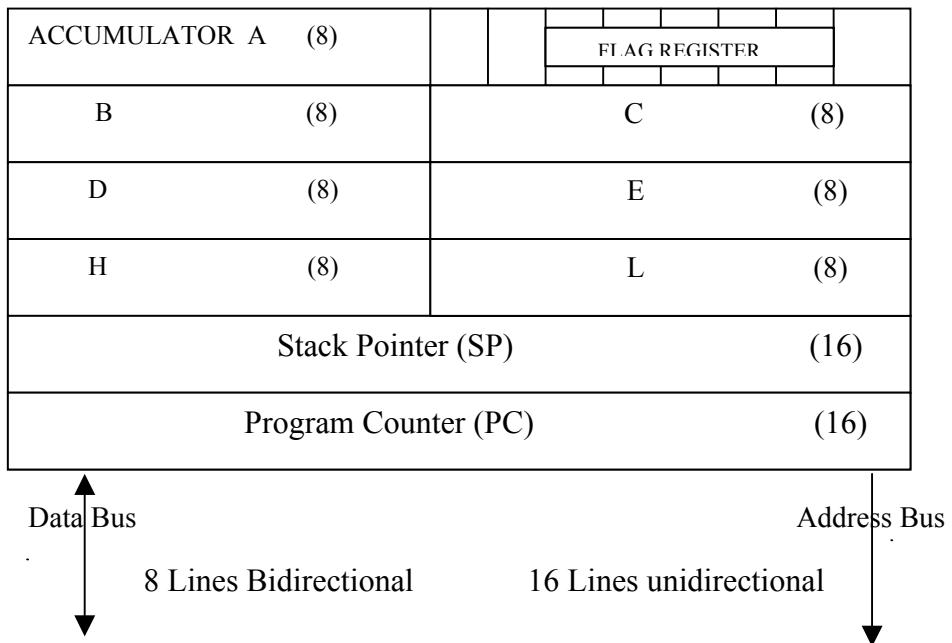
In addition to standard I/O, the memory mapped I/O offers an efficient I/O addressing technique. With this technique, an area of memory address space is assigned for I/O address, thereby, using the memory address for I/O manipulation. The 8085A CPU can also interface with the standard memory that does not have the multiplexed address/data bus.



5. The 8085 Programming Model

In the previous tutorial we described the 8085 microprocessor registers in reference to the internal data operations. The same information is repeated here briefly to provide the continuity and the context to the instruction set and to enable the readers who prefer to focus initially on the programming aspect of the microprocessor.

The 8085 programming model includes six registers, one accumulator, and one flag register, as shown in Figure. In addition, it has two 16-bit registers: the stack pointer and the program counter. They are described briefly as follows.



Registers

The 8085 has six general-purpose registers to store 8-bit data; these are identified as B,C,D,E,H, and L as shown in the figure. They can be combined as register pairs - BC, DE, and HL - to perform some 16-bit operations. The programmer can use these registers to store or copy data into the registers by using data copy instructions.

Accumulator

The accumulator is an 8-bit register that is a part of arithmetic/logic unit (ALU). This register is used to store 8-bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator. The accumulator is also identified as register A.

Flags

The ALU includes five flip-flops, which are set or reset after an operation according to data conditions of the result in the accumulator and other registers. They are called Zero(Z), Carry (CY), Sign (S), Parity (P), and Auxiliary Carry (AC) flags; their bit positions in the flag register are shown in the Figure below. The most commonly used flags are Zero, Carry, and Sign. The microprocessor uses these flags to test data conditions.

| | | | | | | | |
|----|----|----|----|----|----|----|----|
| | | | | | | | |
| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| S | Z | | AC | | P | | CY |

For example, after an addition of two numbers, if the sum in the accumulator is larger than eight bits, the flip-flop used to indicate a carry -- called the Carry flag (CY) -- is set to one. When an arithmetic operation results in zero, the flip-flop called the Zero(Z) flag is set to one. The first Figure shows an 8-bit register, called the flag register, adjacent to the accumulator. However, it is not used as a register; five bit positions out of eight are used to store the outputs of the five flip-flops. The flags are stored in the 8-bit register so that the programmer can examine these flags (data conditions) by accessing the register through an instruction.

These flags have critical importance in the decision-making process of the microprocessor. The conditions (set or reset) of the flags are tested through the software instructions. For example, the instruction JC (Jump on Carry) is implemented to change the sequence of a program when CY flag is set. The thorough understanding of flag is essential in writing assembly language programs.

Program Counter (PC)

This 16-bit register deals with sequencing the execution of instructions. This register is a memory pointer. Memory locations have 16-bit addresses, and that is why this is a 16-bit register.

The microprocessor uses this register to sequence the execution of the instructions. The function of the program counter is to point to the memory address from which the next byte is to be fetched. When a byte (machine code) is being fetched, the program counter is incremented by one to point to the next memory location.

Stack Pointer (SP)

The stack pointer is also a 16-bit register used as a memory pointer. It points to a memory location in R/W memory, called the stack. The beginning of the stack is defined by loading 16-bit address in the stack pointer.

This programming model will be used in subsequent tutorials to examine how these registers are affected after the execution of an instruction.

6. The 8085 Addressing Modes

The instructions MOV B, A or MVI A, 82H are to copy data from a source into a destination. In these instructions the source can be a register, an input port, or an 8-bit number (00H to FFH). Similarly, a destination can be a register or an output port. The sources and destination are operands. The various formats for specifying operands are called the ADDRESSING MODES. For 8085, they are:

1. Immediate addressing.
2. Register addressing.
3. Direct addressing.
4. Indirect addressing.

Immediate addressing

Data is present in the instruction. Load the immediate data to the destination provided.
Example: MVI R,data

Register addressing

Data is provided through the registers.
Example: MOV Rd, Rs

Direct addressing

Used to accept data from outside devices to store in the accumulator or send the data stored in the accumulator to the outside device. Accept the data from the port 00H and store them into the accumulator or Send the data from the accumulator to the port 01H.

Example: IN 00H or OUT 01H

Indirect Addressing

This means that the Effective Address is calculated by the processor. And the contents of the address (and the one following) is used to form a second address. The second address is where the data is stored. Note that this requires several memory accesses; two accesses to retrieve the 16-bit address and a further access (or accesses) to retrieve the data which is to be loaded into the register.

7. Instruction Set Classification

An **instruction** is a binary pattern designed inside a microprocessor to perform a specific function. The entire group of instructions, called the **instruction set**, determines what functions the microprocessor can perform. These instructions can be classified into the following five functional categories: data transfer (copy) operations, arithmetic operations, logical operations, branching operations, and machine-control operations.

Data Transfer (Copy) Operations

This group of instructions copy data from a location called a source to another location called a destination, without modifying the contents of the source. In technical manuals, the term *data transfer* is used for this copying function. However, the term *transfer* is misleading; it creates the impression that the contents of the source are destroyed when, in fact, the contents are retained without any modification. The various types of data transfer (copy) are listed below together with examples of each type:

| Types | Examples |
|---|---|
| 1. Between Registers. | 1. Copy the contents of the register B into register D. |
| 2. Specific data byte to a register or a memory location. | 2. Load register B with the data byte 32H. |
| 3. Between a memory location and a register. | 3. From a memory location 2000H to register B. |
| 4. Between an I/O device and the accumulator. | 4. From an input keyboard to the accumulator. |

Arithmetic Operations

These instructions perform arithmetic operations such as addition, subtraction, increment, and decrement.

Addition - Any 8-bit number, or the contents of a register or the contents of a memory location can be added to the contents of the accumulator and the sum is stored in the accumulator. No two other 8-bit registers can be added directly (e.g., the contents of register B cannot be added directly to the contents of the register C). The instruction DAD is an exception; it adds 16-bit data directly in register pairs.

Subtraction - Any 8-bit number, or the contents of a register, or the contents of a memory location can be subtracted from the contents of the accumulator and the results stored in the accumulator. The subtraction is performed in 2's compliment, and the results if negative, are expressed in 2's complement. No two other registers can be subtracted directly.

Increment/Decrement - The 8-bit contents of a register or a memory location can be incremented or decremented by 1. Similarly, the 16-bit contents of a register pair (such as BC) can be incremented or decremented by 1. These increment and decrement operations differ from addition and subtraction in an important way; i.e., they can be performed in any one of the registers or in a memory location.

Logical Operations

These instructions perform various logical operations with the contents of the accumulator.

AND, OR Exclusive-OR - Any 8-bit number, or the contents of a register, or of a memory location can be logically ANDed, Ored, or Exclusive-OREd with the contents of the accumulator. The results are stored in the accumulator.

Rotate- Each bit in the accumulator can be shifted either left or right to the next position.

Compare- Any 8-bit number, or the contents of a register, or a memory location can be compared for equality, greater than, or less than, with the contents of the accumulator.

Complement - The contents of the accumulator can be complemented. All 0s are replaced by 1s and all 1s are replaced by 0s.

Branching Operations

This group of instructions alters the sequence of program execution either conditionally or unconditionally.

Jump - Conditional jumps are an important aspect of the decision-making process in the programming. These instructions test for a certain conditions (e.g., Zero or Carry flag) and alter the program sequence when the condition is met. In addition, the instruction set includes an instruction called *unconditional jump*.

Call, Return, and Restart - These instructions change the sequence of a program either by calling a subroutine or returning from a subroutine. The conditional Call and Return instructions also can test condition flags.

Machine Control Operations

These instructions control machine functions such as Halt, Interrupt, or do nothing.

The microprocessor operations related to data manipulation can be summarized in four functions:

1. copying data
2. performing arithmetic operations
3. performing logical operations
4. testing for a given condition and alerting the program sequence

Some important aspects of the instruction set are noted below:

1. In data transfer, the contents of the source are not destroyed; only the contents of the destination are changed. The data copy instructions do not affect the flags.
2. Arithmetic and Logical operations are performed with the contents of the accumulator, and the results are stored in the accumulator (with some expectations). The flags are affected according to the results.
3. Any register including the memory can be used for increment and decrement.
4. A program sequence can be changed either conditionally or by testing for a given data condition.

8. Instruction Format

An **instruction** is a command to the microprocessor to perform a given task on a specified data. Each instruction has two parts: one is task to be performed, called the **operation code** (opcode), and the second is the data to be operated on, called the **operand**. The operand (or data) can be specified in various ways. It may include 8-bit (or 16-bit) data, an internal register, a memory location, or 8-bit (or 16-bit) address. In some instructions, the operand is implicit.

Instruction word size

The 8085 instruction set is classified into the following three groups according to word size:

1. One-word or 1-byte instructions
2. Two-word or 2-byte instructions
3. Three-word or 3-byte instructions

In the 8085, "byte" and "word" are synonymous because it is an 8-bit microprocessor. However, instructions are commonly referred to in terms of bytes rather than words.

One-Byte Instructions

A 1-byte instruction includes the opcode and operand in the same byte. Operand(s) are internal register and are coded into the instruction.

For example:

| Task | Op code | Operand | Binary Code | Hex Code |
|--|---------|---------|-------------|----------|
| Copy the contents of the accumulator in the register C. | MOV | C,A | 0100 1111 | 4FH |
| Add the contents of register B to the contents of the accumulator. | ADD | B | 1000 0000 | 80H |
| Invert (compliment) each bit in the accumulator. | CMA | | 0010 1111 | 2FH |

These instructions are 1-byte instructions performing three different tasks. In the first instruction, both operand registers are specified. In the second instruction, the operand

B is specified and the accumulator is assumed. Similarly, in the third instruction, the accumulator is assumed to be the implicit operand. These instructions are stored in 8-bit binary format in memory; each requires one memory location.

MOV rd, rs

rd <-> rs copies contents of rs into rd.

Coded as 01 ddd sss where ddd is a code for one of the 7 general registers which is the destination of the data, sss is the code of the source register.

Example: MOV A,B

Coded as 01111000 = 78H = 170 octal (octal was used extensively in instruction design of such processors).

ADD r

A <-> A + r

Two-Byte Instructions

In a two-byte instruction, the first byte specifies the operation code and the second byte specifies the operand. Source operand is a data byte immediately following the opcode. For example:

| Task | Opcode | Operand | Binary Code | Hex Code | |
|---|--------|---------|-------------------|------------|---------------------------|
| Load an 8-bit data byte in the accumulator. | MVI | A, Data | 0011 1110 DATA | 3E Data | First Byte Second Byte |

Assume that the data byte is 32H. The assembly language instruction is written as

| Mnemonics | Hex code |
|------------|----------|
| MVI A, 32H | 3E 32H |

The instruction would require two memory locations to store in memory.

MVI r,data

r <-> data

Example: MVI A,30H coded as 3EH 30H as two contiguous bytes. This is an example of immediate addressing.

ADI data

A <-> A + data

OUT port

where port is an 8-bit device address. (Port) <-- A. Since the byte is not the data but points directly to where it is located this is called direct addressing.

Three-Byte Instructions

In a three-byte instruction, the first byte specifies the opcode, and the following two bytes specify the 16-bit address. Note that the second byte is the low-order address and the third byte is the high-order address.

opcode + data byte + data byte

For example:

| Task | Opcode | Operand | Binary code | Hex Code | | | | |
|---|--------|---------|---|-----------|-----------|-----------|----------------|---|
| Transfer the program sequence to the memory location 2085H. | JMP | 2085H | <table border="1"><tr><td>1100 0011</td></tr><tr><td>1000 0101</td></tr><tr><td>0010 0000</td></tr></table> | 1100 0011 | 1000 0101 | 0010 0000 | C3 85 20 | First byte Second Byte Third Byte |
| 1100 0011 | | | | | | | | |
| 1000 0101 | | | | | | | | |
| 0010 0000 | | | | | | | | |

This instruction would require three memory locations to store in memory.

Three byte instructions - opcode + data byte + data byte

LXI rp, data16

rp is one of the pairs of registers BC, DE, HL used as 16-bit registers. The two data bytes are 16-bit data in L H order of significance.

rp <-- data16

Example:

LXI H,0520H coded as 21H 20H 50H in three bytes. This is also immediate addressing.

LDA addr

A <-- (addr) Addr is a 16-bit address in L H order. Example: LDA 2134H coded as 3AH 34H 21H. This is also an example of direct addressing.

9. Sample Programs

Write an assembly program to add two numbers

Program

```
MVI D, 8BH  
MVI C, 6FH  
MOV A, C
```

```
ADD D  
OUT PORT1  
HLT
```

Write an assembly program to multiply a number by 8

Program

```
MVI A, 30H  
RRC  
RRC  
RRC  
OUT PORT1  
HLT
```

Write an assembly program to find greatest between two numbers

Program

```
MVI B, 30H  
MVI C, 40H  
MOV A, B  
CMP C  
JZ EQU  
JC GRT  
OUT PORT1  
HLT
```

EQU: MVI A, 01H
OUT PORT1
HLT

GRT: MOV A, C
OUT PORT1
HLT

Instruction Set Design

- One goal of instruction set design is to minimize instruction length
- Many instructions were designed with compilers in mind.
- Determining how operands are addressed is a key component of instruction set design

Instruction Format

- Defines the layout of bits in an instruction
- Includes opcode and includes implicit or explicit operand(s)
- Usually there are several instruction formats in an instruction set
- Huge variety of instruction formats have been designed; they vary widely from processor to processor

Instruction Length

- The most basic issue
- Affected by and affects:
 - Memory size
 - Memory organization
 - Bus structure
 - CPU complexity
 - CPU speed
- Trade off between a powerful instruction repertoire and saving space with shorter instructions

Instruction format trade-offs

- Large instruction set => small programs
- Small instruction set => large programs
- Large memory => longer instructions
- Fixed length instructions same size or multiple of bus width => fast fetch
- Variable length instructions may need extra bus cycles
- Processor may execute faster than fetch
 - Use cache memory or use shorter instructions
- Note complex relationship between word size, character size, instruction size and bus transfer width
 - In almost all modern computers these are all multiples of 8 and related to each other by powers of 2

Allocation of bits

- Determines several important factors
- Number of addressing modes
 - Implicit operands don't need bits
 - X86 uses 2-bit mode field to specify interpretation of 3-bit operand fields
- Number of operands
 - 3 operand formats are rare
 - For two operand instructions we can use one or two operand mode indicators
 - X86 uses only one 2-bit indicator
- Register versus memory
 - Tradeoff between # of registers and program size
 - Studies suggest optimal number between 8 and 32
 - Most newer architectures have 32 or more
 - X86 architecture allows some computation in memory

Allocation of bits

- Number of register sets
 - RISC architectures tend to have larger sets of uniform registers
 - Small register sets require fewer opcode bits
 - Specialized register sets can reduce opcode bits further by implicit reference (address vs. data registers)
- Address range
 - Large address space requires large instructions for direct addressing
 - Many architectures have some restricted or short forms of displacement addressing
 - Ex: x86 short jumps and loops, PowerPC 16-bit displacement addressing
- Address granularity
 - Size of object addressed.
 - Typically 8, 16, 32 and 64 instruction variants

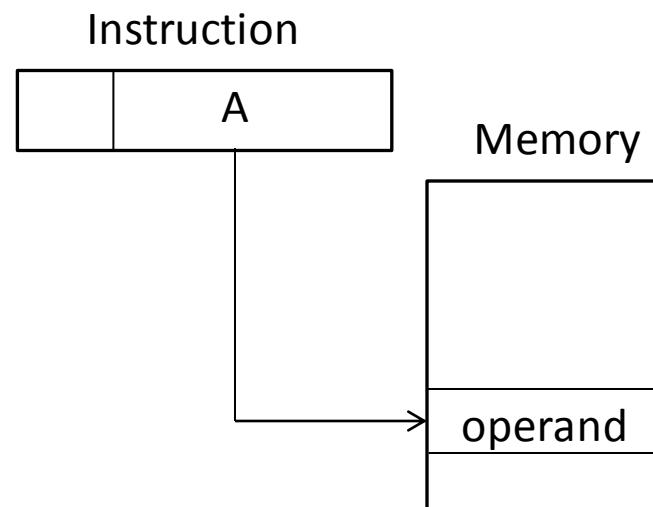
Addressing Modes

Addressing Modes

- For a given instruction set architecture, addressing modes define how machine language instructions identify the operand (or operands) of each instruction.
- An addressing mode specifies how to calculate the effective memory address of an operand by using information held in registers and/or constants contained within a machine instruction or elsewhere.
- Different types of addresses involve tradeoffs between instruction length, addressing flexibility, and complexity of address calculation
- Common addressing modes
 - Direct
 - Immediate
 - Indirect
 - Register
 - Register indirect
 - Displacement
 - Implied (stack)

Direct Addressing

- The instruction tells where the value can be found, but the value itself is out in memory.
- The address field contains the address of the operand
- Effective address (EA) = address field (A)
- In a high level language, direct addressing is frequently used for things like global variables.
- Advantage
 - Single memory reference to access data
 - More flexible than immediate.



Direct Addressing

for the following examples, assume an accumulator machine structure and that an add instruction is stored in memory, beginning at location 12

memory

| assembly lang. | addr | contents | hardware actions |
|----------------|------|----------|-------------------------|
| ... | | ... | |
| add(one) | 12 | 40 | acc <- acc + memory[24] |
| | 13 | 24 | = acc + 1 |
| ... | | ... | |
| word(one,1) | 24 | 1 | effective address = 24 |
| ... | | ... | |

so, when the PC points to 12:

40 (that is, the contents of location 12) is interpreted as an opcode

24 (that is, the contents of location 13) is interpreted as an address

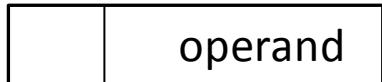
1 (that is, the contents of location 24) is interpreted as data

note that there are no tags or other indicators that the number 40 in location 12 has to be an opcode; it could just as well be used as an address or as data

Immediate Addressing

- the instruction itself contains the value to be used; located in the addresss field of the instruction
- the value is stored in memory immediately after the instruction opcode in memory
- Similar to using a constant in a high level language
- Advantage
 - fast since the value is included in the instruction; no memory reference to fetch data
- Disadvantage
 - not flexible, since the value is fixed at translation-time
 - can have limited range in machines with fixed length instructions

Instruction



Immediate Addressing

for the following example, assume an accumulator machine structure and that an add instruction is stored in memory, beginning at location 12

memory

| assembly lang. | addr | contents | hardware actions |
|------------------|------|----------|---|
| ... | | ... | |
| add_immediate(1) | 12 | 41 | acc <- acc + 1 |
| | 13 | 1 | |
| ... | | ... | no additional memory fetch for data beyond the instruction fetch (since the instruction contains the data being used) |

since an add must have different hardware actions than an add_immediate,
add_immediate has to be a different opcode (or there has to be an extra type-of-addressing-mode code in the instruction format to go along with the opcode)

Example of direct and immediate addressing

Suppose we have a statement in C like

$b = a + 10;$

a and b are variables, so they are out in memory.

To execute this statement, we will need to fetch a from memory, and write our result to b .

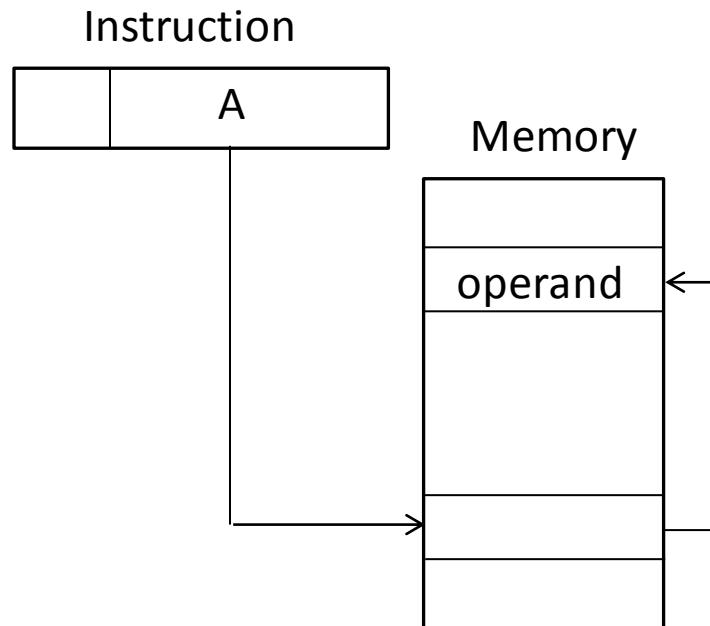
That means the instructions we generate need to have the addresses of a and b , and need to read and write those addresses as appropriate.

The number 10 is an actual value appearing in the statement. So, our code needs to include 10 itself.

Memory-Indirect Addressing

The memory cell pointed to by the address field contains the address of (pointer to) the operand

- $EA = (A)$



Indirect Addressing

for the following examples, assume an accumulator machine structure and that an add instruction is stored in memory, beginning at location 12

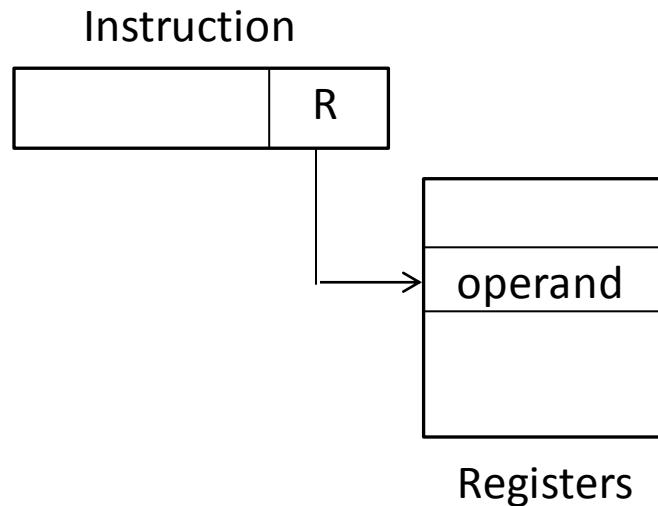
memory

| assembly lang. | addr | contents | hardware actions |
|-------------------|------|----------|---------------------------------|
| ... | | ... | |
| add_indirect(ptr) | 12 | 42 | acc <- acc + memory[memory[36]] |
| | 13 | 36 | = acc + memory[24] |
| ... | | ... | = acc + 1 |
| word(one,1) | 24 | 1 | |
| ... | | ... | effective address = 24 |
| word(ptr,one) | 36 | 24 | |
| ... | | ... | |

the address included in the instruction is that of a pointer, that is, a word that holds another address

Register Addressing

- Operand(s) is (are) registers
- $EA = R$
 - Register R is EA (not contents of R)



Register Addressing

- There is a limited number of registers
 - A very small address field is needed
 - Shorter instructions
 - Faster instruction fetch
 - X86: 3 bits used to specify one of 8 registers
- No memory access needed to fetch EA
- Very fast execution
- Very limited address space
- Multiple registers can help performance
- Requires good assembly programming or compiler writing

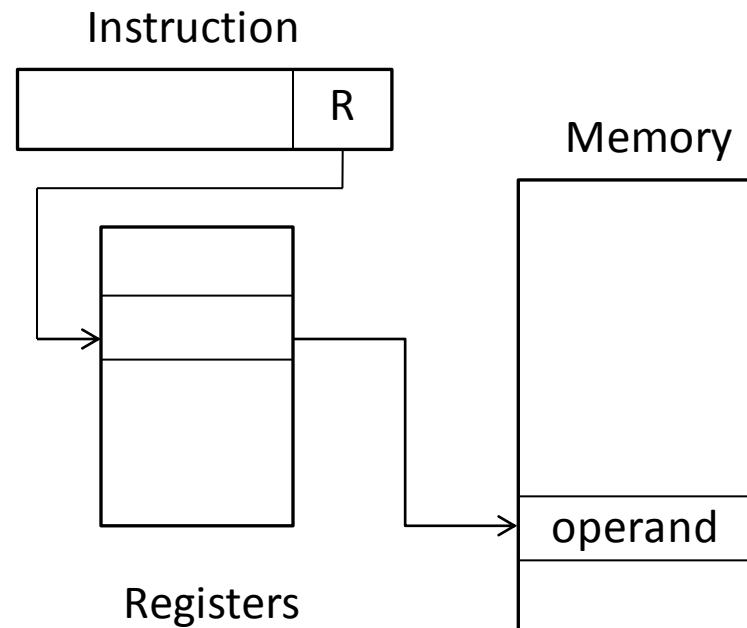
Note: in C you can specify register variables

```
register int a;
```

- This is only advisory to the compiler; no guarantees

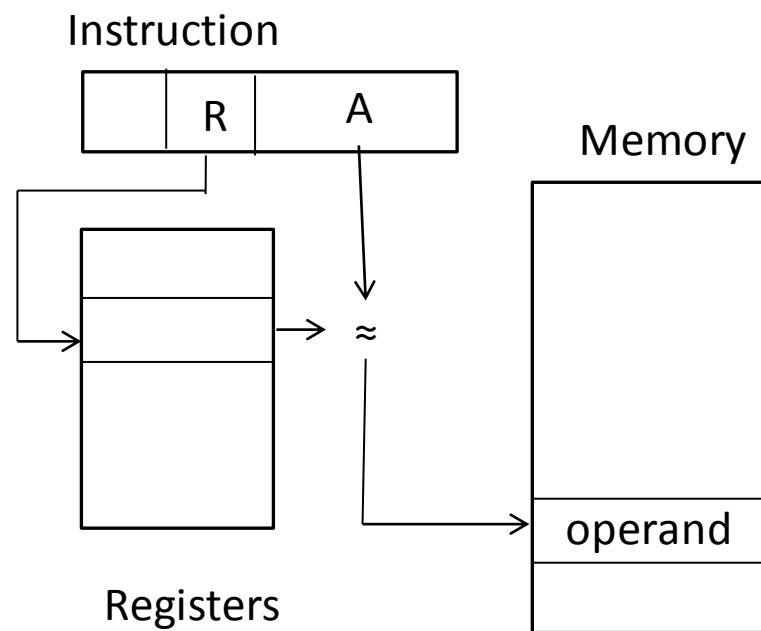
Register-Indirect Addressing

- Similar to memory-indirect addressing
- $EA = (R)$
- Operand is in memory cell pointed to by contents of register R
- Large address space (2^n)
- One fewer memory address than memory-indirect



Displacement Addressing

- Combines register-indirect addressing and direct addressing
- $EA = A + (R)$
- Address field holds two values
 - A = base value
 - R = register that holds displacement
 - Or visa versa



Types of Displacement Addressing

- Relative Addressing
- Base-register addressing
- Indexing

Relative Addressing

- $EA = A + (PC)$
- Address field A is treated as 2's complement integer to allow backward references
- Fetch operand from $PC+A$
- Can be very efficient because of locality of reference & cache usage
 - But in large programs code and data may be widely separated in memory

Base-Register Addressing

- A holds displacement
- R holds pointer to base address
- R may be explicit or implicit
 - E.g. segment registers in 80x86 are base registers and are involved in all EA computations
 - X86 processors have a wide variety of base addressing

Indexed Addressing

- $A = \text{Base}$
- $R = \text{displacement}$
- $EA = A + R$
- Good for accessing arrays
 - $EA = A + R$
 - $R++$
- Iterative access to sequential memory locations is very common
- Some architectures provide auto-increment or auto-decrement
- Preindex $EA = A + (R++)$
- Postindex $EA = A + (++R)$

Indexed Addressing

for the following examples, assume an accumulator machine structure and that an add instruction is stored in memory, beginning at location 12

memory

| assembly lang. | addr | contents | hardware actions |
|-------------------|------|----------|------------------------------------|
| ... | ... | ... | |
| add_indexed(b0,x) | 12 | 43 | acc <- acc + memory[20+memory[36]] |
| | 13 | 20 | = acc + memory[20+4] |
| | 14 | 36 | = acc + memory[24] |
| ... | | ... | = acc + 1 |
| word(b0,5) | 20 | 5 | |
| word(b1,-2) | 21 | -2 | effective address = 24 |
| word(b2,3) | 22 | 3 | |
| word(b3,9) | 23 | 9 | |
| word(b4,1) | 24 | 1 | |
| ... | | ... | |
| word(x,4) | 36 | 4 | |
| ... | | ... | |

Addressing modes using registers

on machines with multiple registers, addresses and index values can be held in registers, for example:

| | | |
|---|-------------------|--|
| Direct | load(x,r1) | // r1 <- memory[x] |
| immediate | load_imm(3,r2) | // r2 <- 3 |
| indexed for array access (fixed array base address and index in a register) | load_ind(a,r3,r4) | // r4 <- memory[a + r3] |
| register indirect as part of indexed (i.e., a pointer is in a register) | load_ind(0,r5,r6) | // r6 <- memory[0 + r5] |
| base plus displacement as part of indexed (i.e., structure access w/ ptr. in reg. and constant offset) | load_ind(2,r7,r8) | // r8 <- memory[2 + r7] // accesses 3rd word of // a structure |

Branch addressing modes

direct addressing, such as the accumulator machine

| Assembly lang | addr | contents | hardware actions |
|---------------|------|----------|-------------------------------|
| ... | | ... | |
| ba(target) | 20 | 70 | pc <- 30 |
| | 21 | 30 | |
| ... | | ... | |
| label(target) | 30 | | next instruction after branch |
| ... | | ... | |

Branch addressing modes

pc-relative addressing, such as the JVM Memory

| Assembly lang | addr | contents | hardware actions |
|---------------|------|----------|-----------------------------|
| ... | | ... | |
| goto(target) | 20 | 167 | $pc \leftarrow 21 + 9$ |
| | 21 | 9 | $= 30$ |
| ... | | ... | |
| label(target) | 30 | | next instruction after goto |
| ... | | ... | |

note that other machines may make the offset relative to the address of the branch (e.g., 20 above) or the fully-updated pc (e.g., 22 above)

Stack Addressing

- Operand is implicitly on top of stack
 - PUSH
 - POP

Alternate Addressing

- Offset addressing – offset is added or subtracted from value in base register
- Preindex addressing
 - Memory address is formed the same way as offset addressing, but the memory address is written back to the base register after adding or subtracting the displacement
 - The writeback occurs before the store to memory
- Postindex addressing
 - Similar to preindex addressing, but the writeback of the effective address occurs after the store to memory

DTEL

(Department for Technology Enhanced Learning)
The Centre for Technology enabled Teaching & Learning



Teaching Innovation - Entrepreneurial - Global



NAGAR YUWAK SHIKSHAN SANSTHA'S SHRI DATTA MEGHE POLYTECHNIC

DEPARTMENT OF COMPUTER TECHNOLOGY

Microprocessor and Programming

AUTHORS

MANOJ JETHWA

CONTENT: MICROPROCESSOR AND PROGRAMMING



CHAPTER 1: Basics of Microprocessor



CHAPTER 2: 16 Bit Microprocessor: 8086



CHAPTER 3: Instruction Set of 8086 Microprocessor



CHAPTER 4: The Art of Assembly Language Programming



CHAPTER 5: 8086 Assembly Language Programming.



CHAPTER 6: Procedure and Macro in Assembly Language Program

SYLLABUS GENERAL OBJECTIVE

The student will be able to:



Understand What is microprocessor Architecture.



Understand the execution of instructions in pipelining and address generation.



Apply instructions in Assembly Language Program for different problem statements.



Use the procedures and macros in assembly language programming.

CHAPTER-1 Basics of Microprocessor



Topic 1: Evolution of Microprocessor and types



Topic 2: 8085 Microprocessor,



Topic 3: Salient features of 8085



Topic 4: Architecture of 8085 - Functional Block diagram,



Topic 5: Pin description,

CHAPTER-1 SPECIFIC OBJECTIVE / COURSE OUTCOME

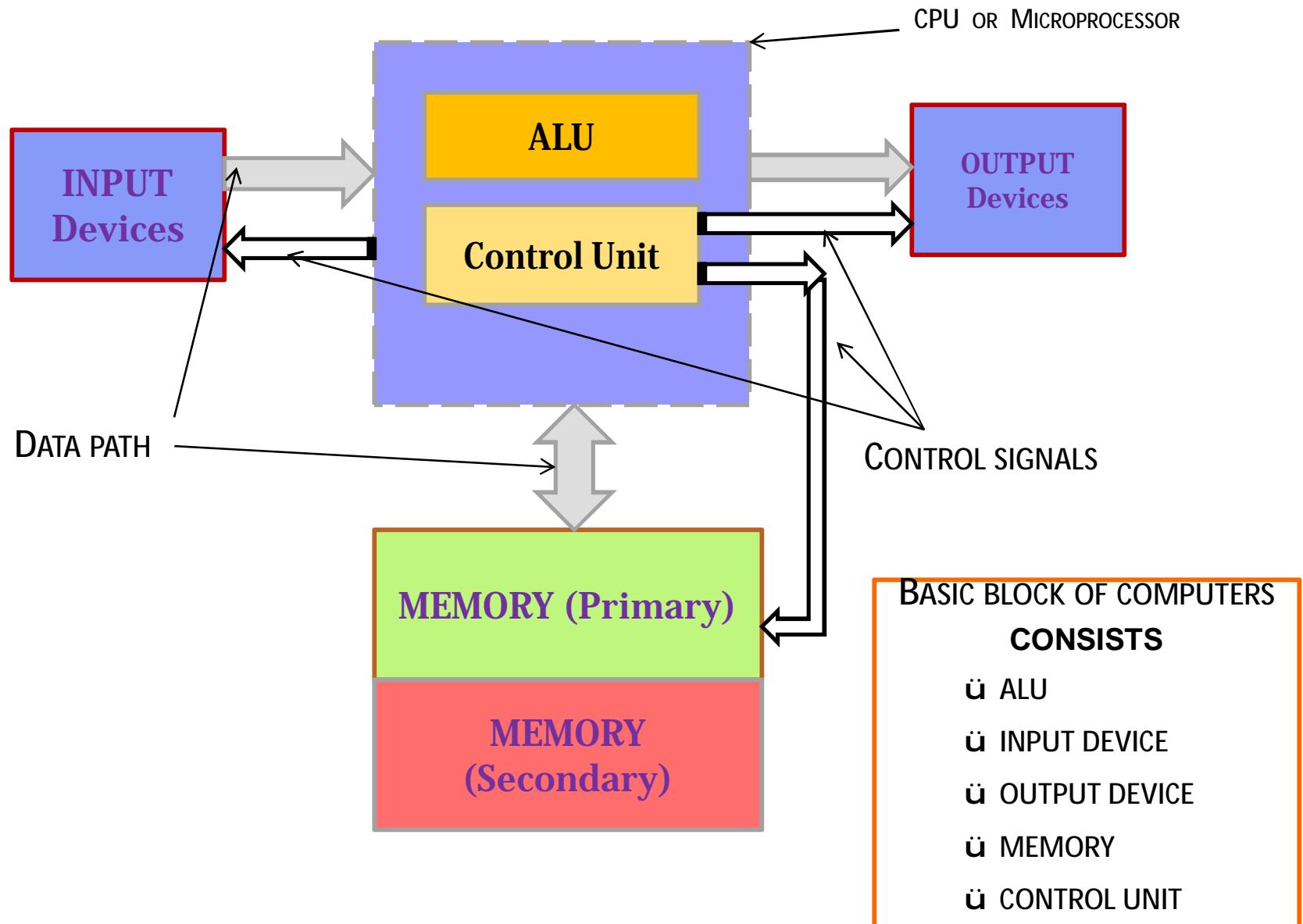
The student will be able to:



Draw the architecture of 8085 and understand the functions of different pins of 8085



Identify status of different flags and understand register organisation of 8085



The typical Computer system consists of:

- § CPU (central processing unit)
 - ü ALU (arithmetic-logic unit)
 - ü Control Logic
 - ü Registers, etc...
- § Memory
- § Input / Output interfaces

Interconnections between these units are through 3 basic buses:

- § Address Bus
- § Data Bus
- § Control Bus

- ü The main function of ALU is to perform arithmetic and logical operations on binary numbers.
- ü The Input Device is used to feed data and command for the CPU.
- ü The output device is used for display of result /data /program etc.
- ü The memory is used for storing information.
- ü The control unit Synchronizes operation of ALU with IO and Memory.

The interconnections (known as Interfacing) between the 5 units of computer system is carried by 3 basic buses i) Address Bus ii) Data Bus iii) Control Bus. A bus(from the Latin *omnibus*, meaning "for all") is essentially a set of wires which is used in computer system to carry information of the same logical functionality. The function of the 3 buses is

- ü The address bus selects memory location or an I/O device for the CPU.
- ü The data bus transfers information between the microprocessor and its memory or I/O device. Data transfer can vary in size, from 8-bits wide to 64 bits wide in various members of microprocessors.
- ü The Control bus generates command signals to synchronise the CPU operation with IO and Memory devices.

| Processor | Date of Launch | Clock speed | Data Bus Width | Address Bus | Addressable Memory Size |
|-------------------------|----------------|-------------|----------------|-------------|-------------------------|
| 4004 | 1971 | 740 khz | 4 bit | 12 | 4 KB |
| 8-BIT PROCESSOR | | | | | |
| 8008 | 1972 | 800 KHz | 8 bit | 14 | 16 Kb |
| 8080 | 1974 | 2 Mhz | 8 bit | 16 | 64 kb |
| 8085 | 1976 | 3 Mhz | 8 bit | 16 | 64 kb |
| 16-BIT PROCESSOR | | | | | |
| 8086 | 1978 | 5 Mhz | 16 | 20 | 1M |
| 80286 | 1982 | 16 Mhz | 16 | 24 | 16 M |

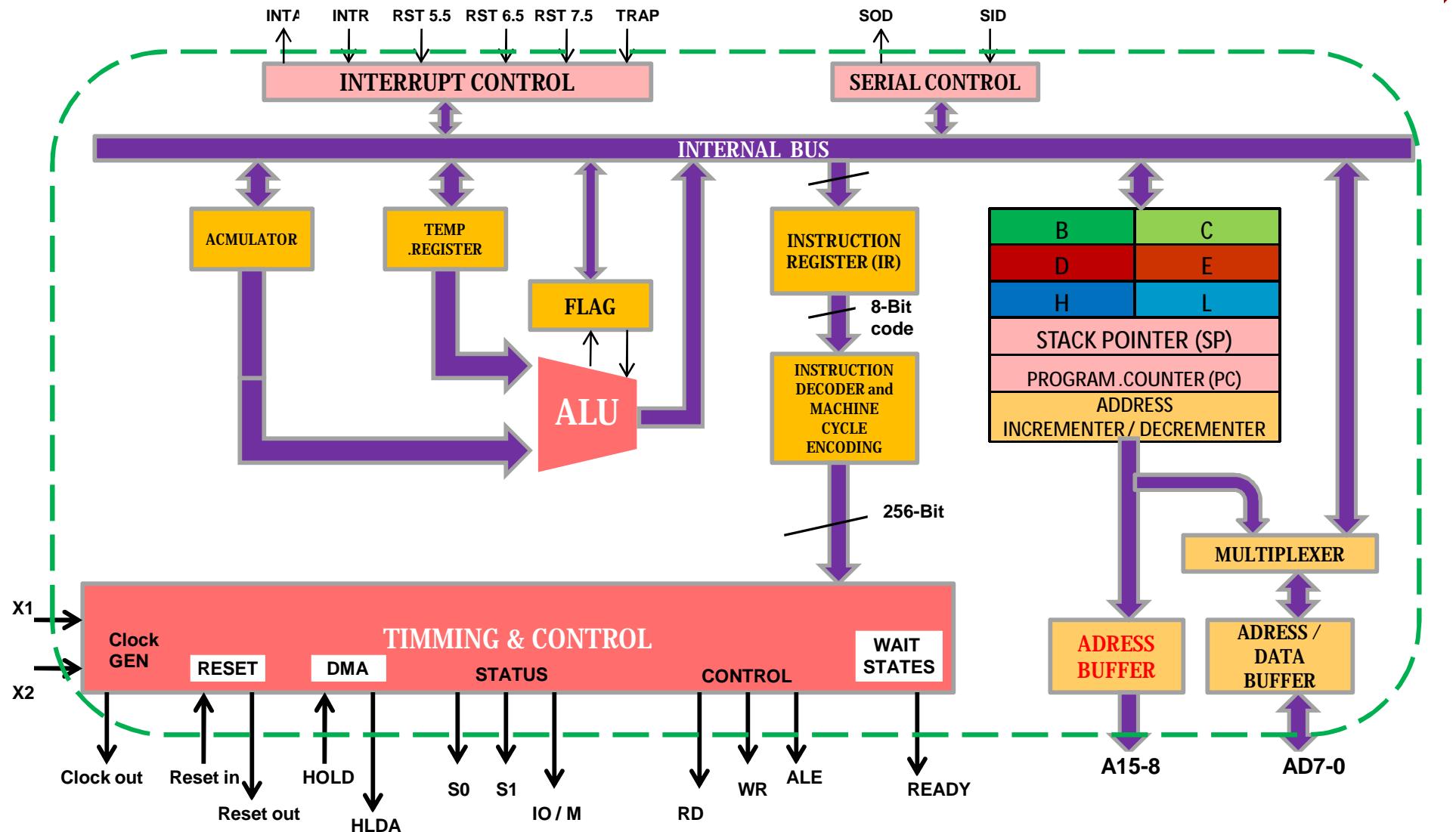
LECTURE 1:-

Evolution of Microprocessor

| Processor | Date of Launch | Clock speed | Data Bus Width | Address Bus | Addressable Memory Size |
|-------------------------|----------------|-------------|----------------------------|-------------|--|
| 32-BIT PROCESSOR | | | | | |
| 80386 | 1985 | 33 Mhz | 32 | 32 | 4 G |
| 80486 | 1989 | 40 Mhz | 32 | 32 | 4G+ 8k cache |
| Petium I | 1993 | 100 Mhz | 32 | 32 | 4G+16k cache |
| Petium II | 1997 | 233 Mhz | 32 | 32 | 4G+16k cache + L2 256 Cache |
| Petium III | 1999 | 1.4 Ghz | 32 | 32 | 4G+32k cache + L2 256 Cache |
| Petium IV | 2000 | 2.66 Ghz | 32 Internal 64 External | 32 | 4G+32k cache + L2 256 Cache |
| 64-BIT PROCESSOR | | | | | |
| Dual Core | 2006 | 2.66 Ghz | 64 | 36 | 64G+Independent L1 64 Kb+ Common L2 256 kb Cache |
| Core 2 Duo | 2006 | 3 Ghz | 64 | 36 | 64G+Independent L1 128 Kb+ Common L2 4 Mb Cache |
| I7 | 2008 | 3.33 Ghz | 64 | 36 | 64G+Independent L1 64 Kb+ Common L2 256 kb Cache + 8 Mb L3 Cache |

LECTURE 2:-

8085 ARCHITECTURE



ü Arithmetic and Logic Unit

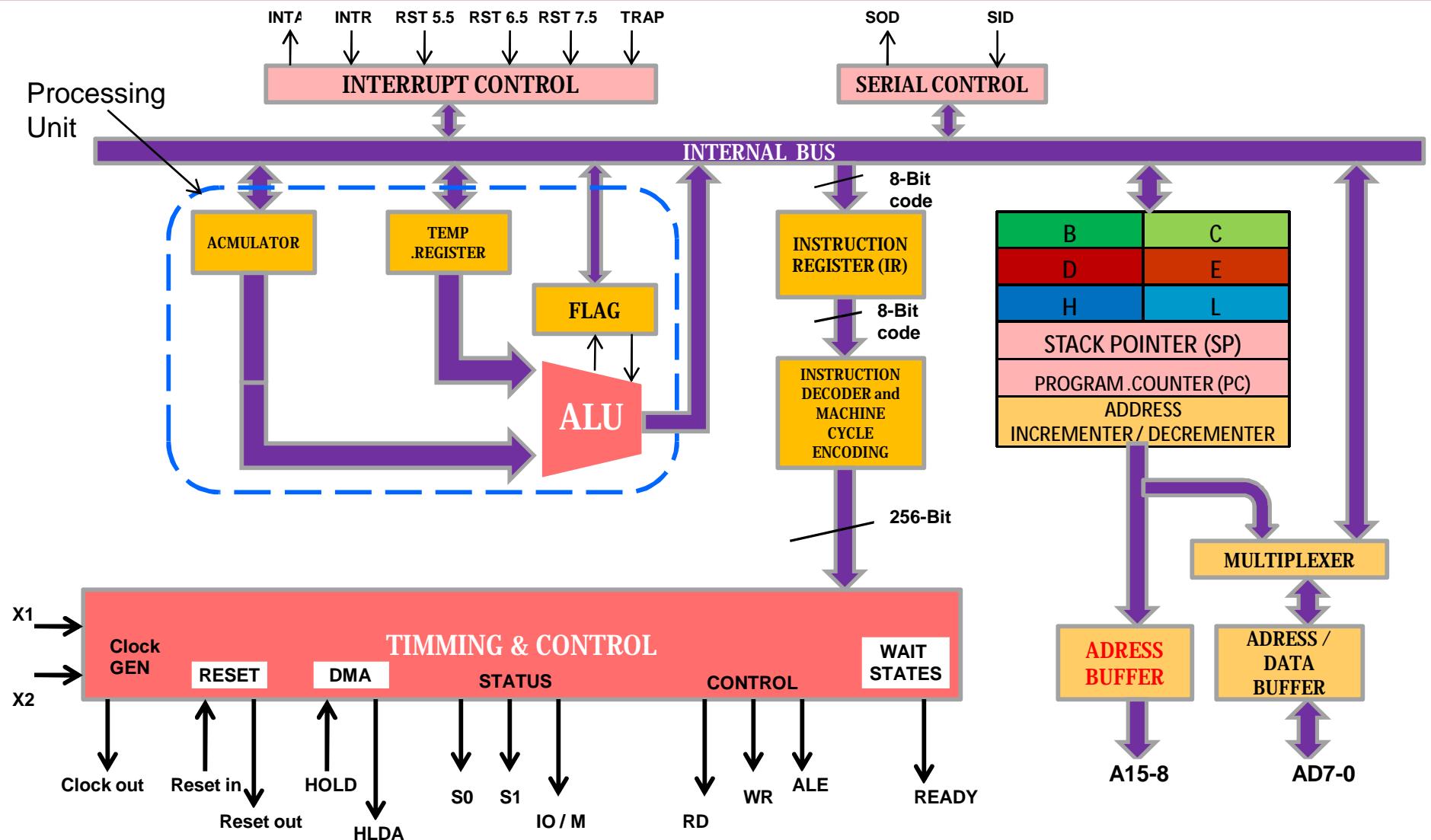
ü Accumulator

ü Status Flags

ü Temporary Register

LECTURE 2:-

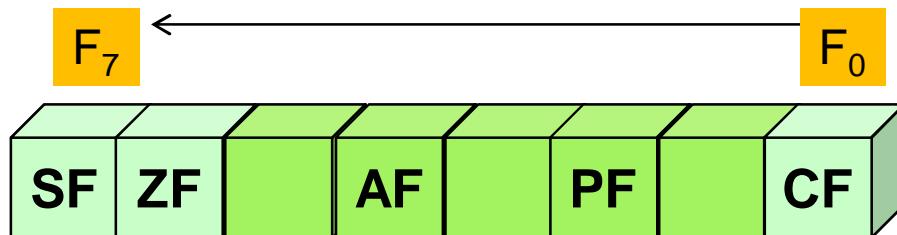
Processing Unit



- ü It performs Arithmetic and logic operations on binary nos.
- ü The result is stored in accumulator in most cases, hence A is known as accumulator.
- ü Arithmetic Operations:
 - ü Addition, Subtraction, Increment, Decrement .
- ü Logic Operations:
 - ü AND, OR, X-OR, Complement .

- ü It is the main register of microprocessor directly connected with the ALU.
- ü It is also called register 'A'.
- ü It is an 8-bit register.
- ü It is used in the arithmetic and logic operations.
- ü It always contains one of the operands on which arithmetic/logic has to be performed.
- ü After the arithmetic/logic operation, the contents of accumulator are replaced by the result.

Status Flag



The 5 Status Flags are affected immediately after an arithmetic or logical operation performed by the ALU. The SET or RESET condition of each flag is used to indicate the status of the result generated by the ALU.

Ø Status

| | |
|-----|----------------------|
| CF: | Carry flag |
| PF: | Parity flag |
| AF: | Auxiliary carry flag |
| ZF: | Zero flag |
| SF: | Sign flag |
| OF: | Overflow flag |

Status Flag

- ü Sign Flag: It is used to indicate whether the result is positive or negative. It will set ($SF=1$) if the result is -ve and if the result +ve then $SF=0$.
- ü Zero Flag: It is used to indicate whether the result is a Zero or non-zero. It will set ($ZF=1$) if the result is zero else $ZF=0$.
- ü Auxiliary carry Flag: It is used to indicate whether or not the ALU has generated a carry/Borrow from D3 bit position to D4 bit. It will set if there was a carry out from bit 3 to bit 4 of the result else $AF=0$. The auxiliary carry flag is used for binary coded decimal (BCD) operations.
- ü Parity Flag: It is used to indicate parity (Even or Odd) of the result. It will set if the parity is even else $PF =0$.
- ü Carry Flag: It is used to indicate whether a carry/Borrow has been generated /occurred during addition/subtraction It will set if there was a carry is generated from the MS-bit during addition, or borrow during subtraction/comparison else $CF=0$.

Program Status Word (PSW)

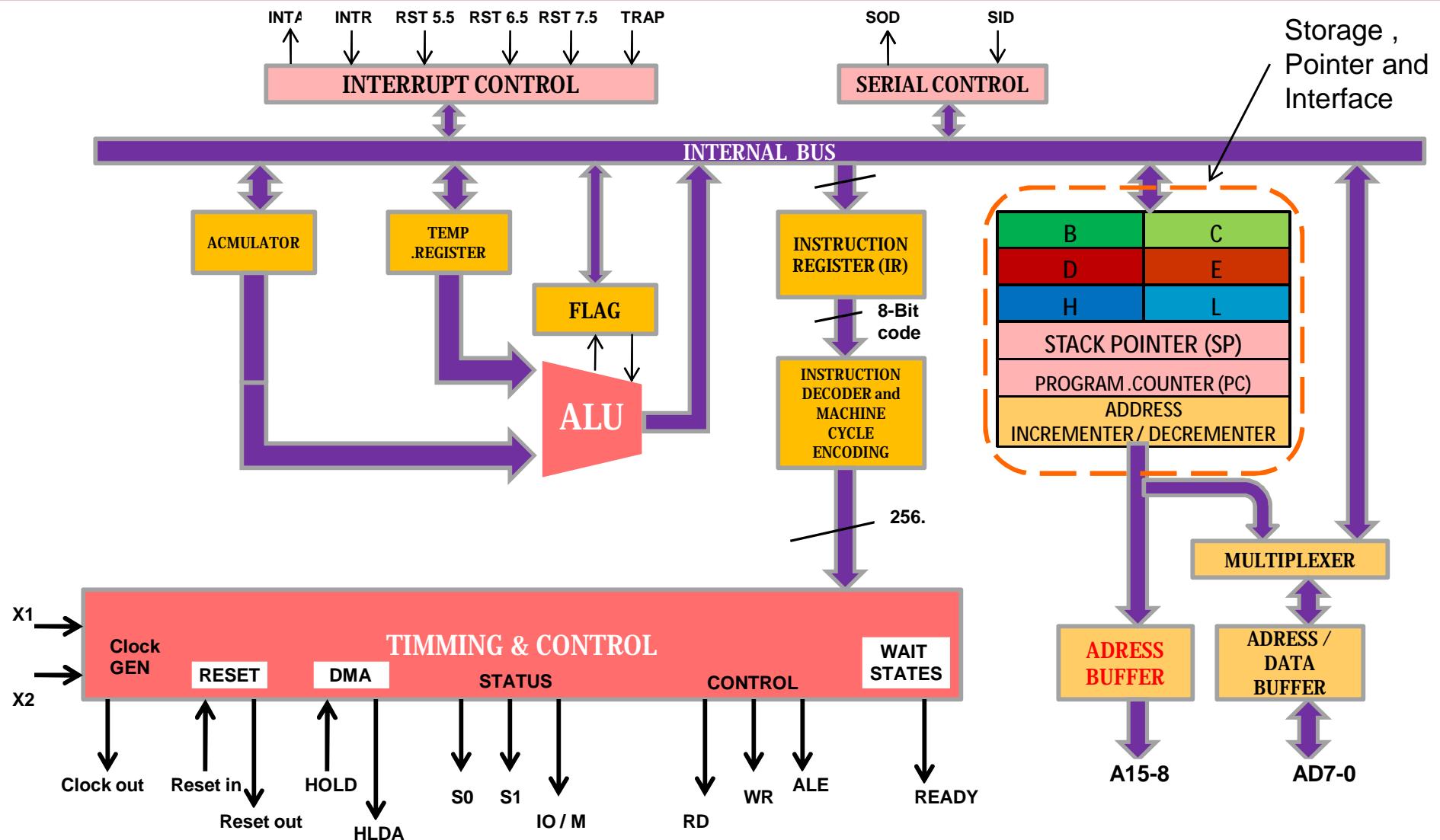
- The Accumulator and Status Flags clubbed together is known as Program Status Word (PSW).
- It is a 16-bit word.

Accumulator (8)

FLAGS (8)

LECTURE 3:-

Register sets & pointer



ü The 8085 has set of 8 register (of 8-bit) and 2 memory pointers (of 16-bit) . The register A and flag are directly connected with ALU, While B,C,D,E,H,& L are indirectly connected through internal bus. The register A is used to store data as well as result of an operation performed by the ALU. The Flag is used to store status of result. The register B,C,D,E,H,L are used to store 8-bit data. It can also be paired to store 16-bit data. The pairing combination can be,

B-C D-E H-L

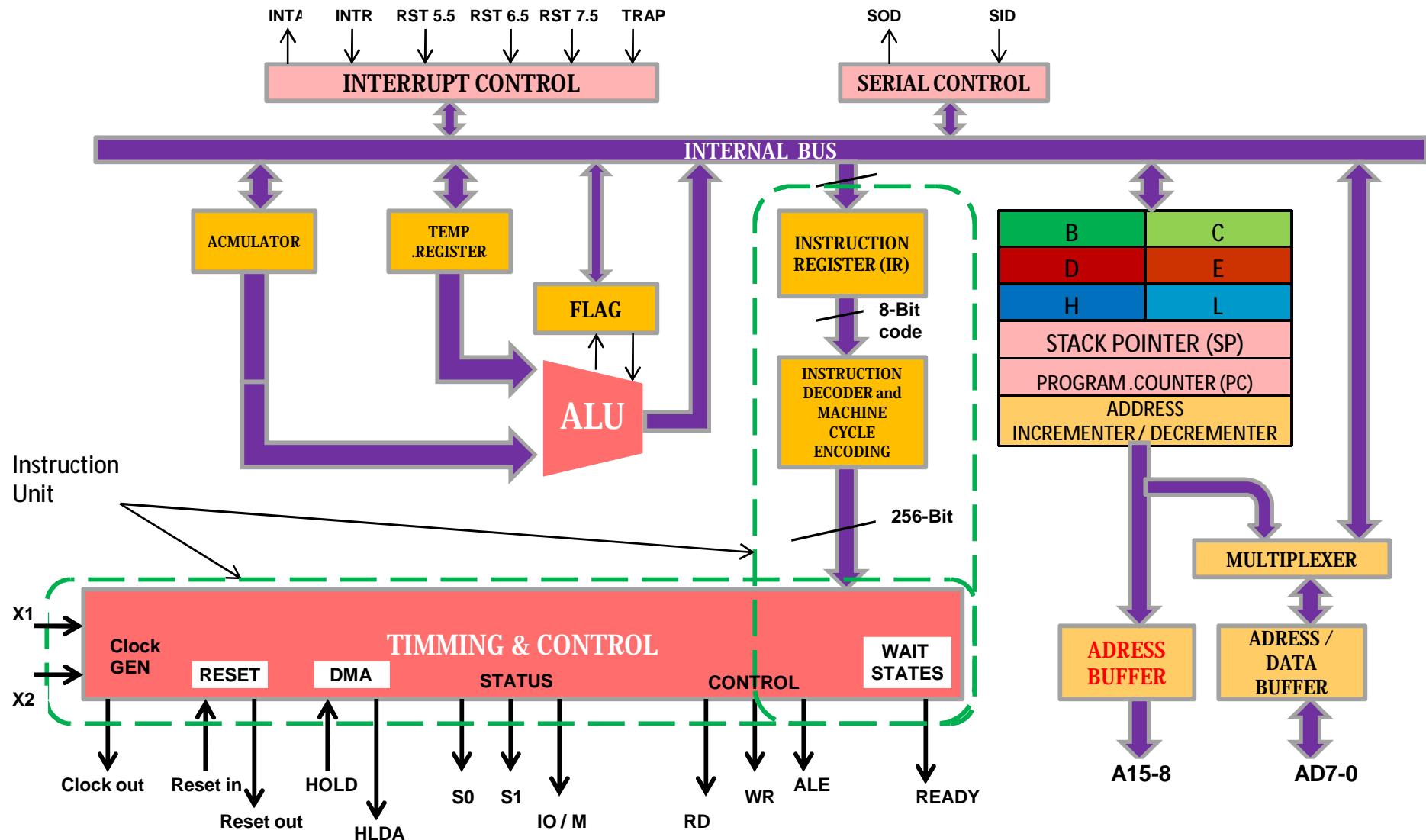
The register pairs can also be used to generate 16-bit address.

ü The pointers are used to generate 16-bit address for selection of memory location. The PC generates address during execution of a program. It contains the memory address (16 bits) of the instruction that will be executed in the next step.

While SP generates address during stack operation.

LECTURE 3:-

Timing-Control and IR -Decoder

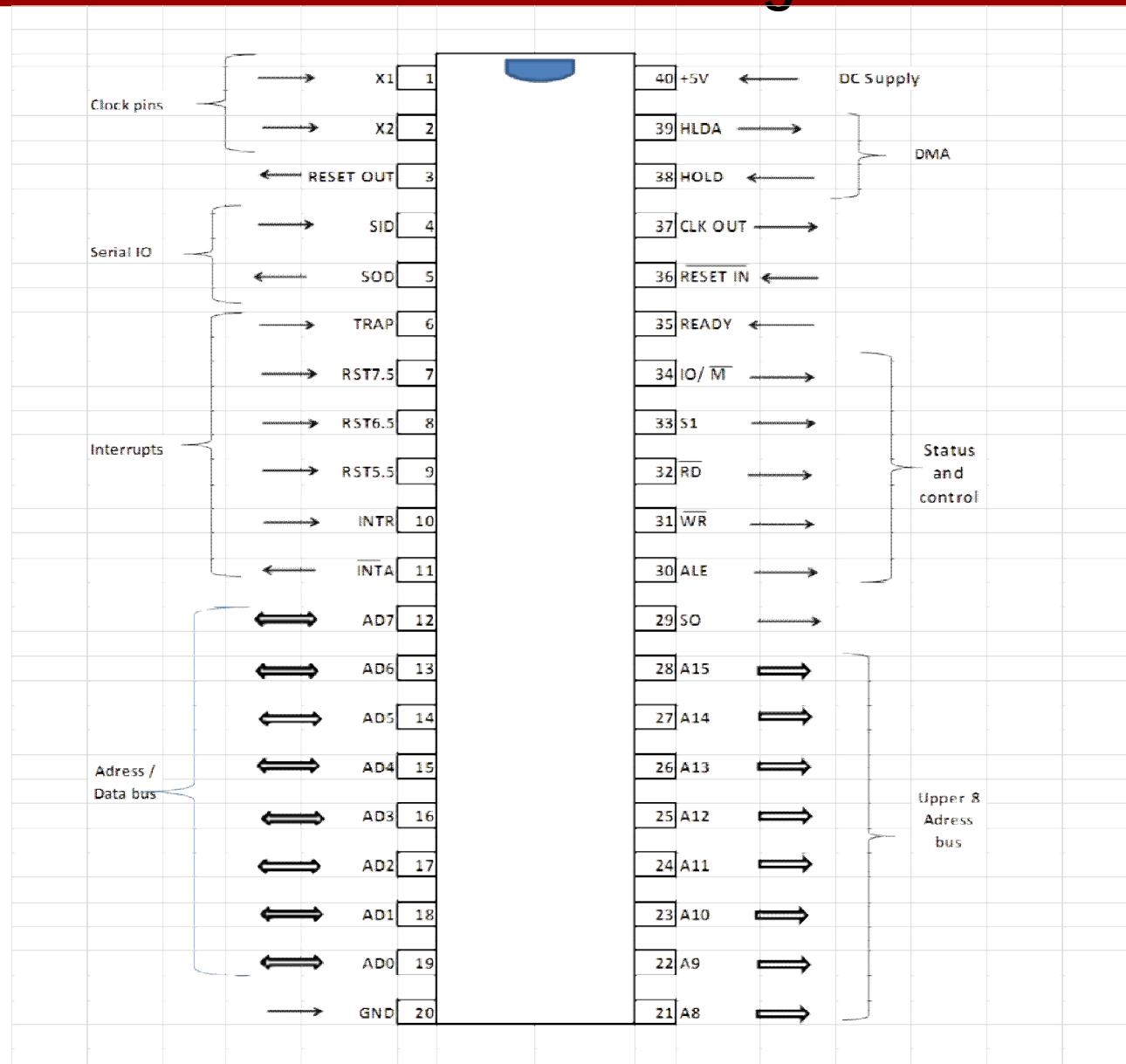


- Instruction Register : It is used to store the current instruction code which is fetched from the memory It is an 8-bit register.
- Instruction Decoder : Instruction is decoded and the meaning in the form of signal is given to TC.
- Timing and Control Unit : It generates internal and external signals to execute the code

PIN DIAGRAM OF 8085

LECTURE 4:-

Pin diagram



- ü It was introduced in 1977 by Intel.
- ü It is 8-bit microprocessor.
- ü It is NMOS device consisting of 6200 transistors.
- ü Its data bus is 8-bit and address bus is 16-bit.
- ü Its clock speed was 3 MHz. Could execute 7,69,230 instructions per second.
- ü Its data bus is 8-bit and address bus is 16-bit.
- ü It had 6,500 transistors.
- ü It is 40 pins Dual-Inline-Package (DIP).



- It is also called Oscillator Pins to which crystal of max 6.14 Mhz should be connected so that 8085 can generate clock signals internally.
- The internal Clock pulses will be $\frac{1}{2}$ times crystal value i.e, 3.07 Mhz.

- RESET IN is used to reset the microprocessor by making the pin Low. When the signal on this pin is low for at least 3 clock cycles, it forces the microprocessor to reset . Thereby ,
 1. Clear the PC and IR.
 2. Disable all the interrupts (except TRAP) and the SOD pin.
 3. HIGH output pulse to RESET OUT pin.
- Reset OUT is used to reset the external peripheral devices and ICs on the circuit. It is active high signal. The output on this pin goes high whenever RESET IN pin is made low .

- SID (Serial Input Data): It receives 1-bit from external device and Stores the bit at the MSB of the Accumulator. RIM (Read Interrupt Mask) instruction is used to transfer the bit from SID MS Bit of Acc.
- SOD (Serial Output Data): It transmits MSB of Accumulator through this pin. SIM (Set Interrupt Mask) instruction is used to transfer the MS bit of Acc through SOD.

– Interrupt: (INTR,RST5.5,RST6.6,RST7.5 and TRAP pins)

- It allows external devices to *interrupt* the normal program execution of the microprocessor.
- When microprocessor receives interrupt signal, it temporarily stops current program and starts executing new program indicated by the interrupt signal.
- Interrupt signals are generated by external peripheral devices like keyboard , sensors, printers etc.
- After execution of the new program, microprocessor returns back to the previous program.

- The 5 Hardware Interrupt Pins are TRAP , RST 7.5 , RST 6.5 , RST 5.5 and INTR. Interrupts can be classified as,
 1. Maskable and Non-Maskable
 2. Vectored and Non-Vectored
 3. Edge Triggered and Level Triggered
 4. Priority Based Interrupts

Maskable and Non-Maskable

Maskable interrupts are those interrupts which can be *enabled* or *disabled*. Enabling and Disabling can be done by software instructions like EI , DI and SIM. The interrupt pins RST7.5, RST6.5 ,RST5.5 and INTR are Maskable.

The interrupts which cannot be *disabled* are called non-maskable interrupts. These interrupts can never be disabled by any software instruction. TRAP is a non-maskable interrupt.Such pins are normally used for emergency cases like fire alarming , fire extinguisher system ,intruder detector etc.

Vectored and Non-Vectored

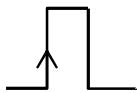
- Vectored interrupts which have particular memory location where program control is transferred when interrupt occurs. Each vectored interrupt points to the particular location in memory. RST 7.5 , RST 6.5 , RST 5.5, TRAP are vectored Interrupts.
 - The addresses to which program control is transferred are :

| Name | Vectored Address |
|---------|-----------------------|
| RST 7.5 | 003C H (7.5 x 0008 H) |
| RST 6.5 | 0034 H (6.5 x 0008 H) |
| RST 5.5 | 002C H (5.5 x 0008 H) |
| TRAP | 0024 H (4.5 x 0008 H) |

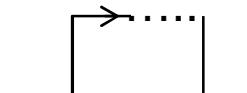
- Absolute address is calculated by multiplying the RST no with 0008 H.
- Non-Vectored interrupts don't have fixed memory location for transfer of program control. The address of the memory location is given by interrupting device to the processor along with the interrupt. INTR is a non-vectored interrupt.

Edge Triggered and Level Triggered

The interrupts that are triggered at leading or trailing edge are called edge triggered interrupts. RST 7.5 is an edge triggered interrupt. It is triggered during the leading (positive) edge.



The interrupts which are triggered at high or low level are called level triggered interrupts. RST 6.5,RST 5.5, INTR, are level triggered interrupt.



The TRAP is edge and level triggered interrupt.

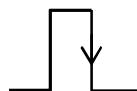
Priority Based Interrupts

When there is a simultaneous interrupt request at two or more interrupt pins then the microprocessor will execute program of that pin that has higher priority. To avoid confusion in such cases all microprocessor assigns priority level to each interrupt pins. Priority is considered by microprocessor only when there are simultaneous requests. The priority of 8085 pins are:

| Interrupt | Priority |
|-----------|----------|
| TRAP | 1 |
| RST 7.5 | 2 |
| RST 6.5 | 3 |
| RST 5.5 | 4 |
| INTR | 5 |

- **Address Bus pins $A_{15}-A_8$ and AD_7-AD_0** : The address bus are used to send address for memory or IO device. It selects one of the many locations in memory or a particular IO port. 8085 has 16-bit bus.
- **Data Bus pins AD_7-AD_0** : It is used to transfer data between microprocessor and memory /IO Device. 8085 Data bus is of 8-bit.

- On this pin 8085 generates a pulse in the T1 clock of each Machine cycle to latch lower byte address from $AD_7 - AD_0$ pins. From mid of T2 to T3 the bus can transfer data.



- It indicates whether the bus has address or data . Since the bus $AD_7 - AD_0$ is used for Address as well as data therefore it is known as Multiplexed bus. Due to multiplexing 8085 has less no of pins.

- S_0 and S_1 are called Status Pins. They indicate the status of current operation which is in progress by 8085. The 4 status indicated by 8085 are

| S_0 | S_1 | Operation |
|-------|-------|--------------|
| 0 | 0 | Halt |
| 0 | 1 | Write |
| 1 | 0 | Read |
| 1 | 1 | Opcode Fetch |

- This pin indicates whether I/O or memory operation is being performed by microprocessor.
- If $\text{IO}/\overline{\text{M}} = 1$ then
 - I/O operation is being performed.
- If $\text{IO}/\overline{\text{M}} = 0$ then
 - Memory operation is being performed.

- It is a control signal used to perform Read operation from memory or from Input device. It is active low signal. A low signal indicates that data on the data bus must be placed by the selected memory location or input device.

- It is a control signal used to perform Write operation into memory location or to output device. It is also active low signal. A low signal indicates that data on the data bus must be written into selected memory location or to output device.

- This pin is used to synchronize slower peripheral devices with high speed of microprocessor.
- A low pulse in T2 causes the microprocessor to enter into *wait state*.
- The microprocessor remains in wait state until the input at this pin goes high.

- HOLD pin is used to request the microprocessor for DMA transfer.
- A high signal on this pin is a request to microprocessor, by external device, to relinquish the hold on buses.
- The request is sent by external device through DMA controller.

- The HLDA signal is send to DMA Controller as acknowledgement to DMA controller to indicate that microprocessor has relinquished the system bus.
- After data transfer When HOLD is made low by DMA controller, HLDA is also made low by 8085 so that the microprocessor takes control of the buses again.

- +5V DC power supply is connected to V_{CC} to bias internal circuit.
- Ground signal is connected to V_{SS} .

- What are the technical features of 8085?
- Explain the function of ALU section of 8085.
- Describe the function of the following blocks of 8085
- ALU ii) Timming & control iii) Instruction Decoder
- Explain the function of various registers of 8085.
- Draw the Block (Architecture) of 8085 and explain IR, stack pointer and programme counter.
- What are the various Flag of 8085?
- What are the pointers of 8085.Explain the function of Pointers of 8085?
- Explain the function of Interrupt section of 8085.
- List Maskable and non-maskable Interrupts of 8085.
- Explain the function of SID & SOD of 8085.
- Describe microprocessor evolution with suitable example?
- Differentiate, any six ,between 8085 & 8086.

1. The typical Computer system consists of:

- ü ALU (arithmetic-logic unit)
- ü Control Logic
- ü Memory
- ü Input devices
- ü Output devices

2. A bus (from the Latin *omnibus*, meaning "for all") is essentially a set of wires which is used in computer system to carry information of the same logical functionality. The interconnections (known as Interfacing) between the 5 units of computer system is carried by 3 basic buses

- i) Address Bus ii) Data Bus iii) Control Bus.

3. 8085 can be divided into sections like i) Processing unit ii) Register & pointers iii) instruction register-decoder-timing & control.

4. The 8085 has 8-bit flag but 5 are affected by Arithmetic / logical operation.

CHAPTER-2 16 Bit Microprocessor: 8086



Topic 1: Salient features of 8086



Topic 2: Architecture of 8086 - Functional Block diagram,



Topic 3: Register organization,



Topic 4: Concepts of pipelining, Memory segmentation



Topic 5: Operating Modes of 8086

CHAPTER-2 SPECIFIC OBJECTIVE / COURSE OUTCOME

The student will be able to:



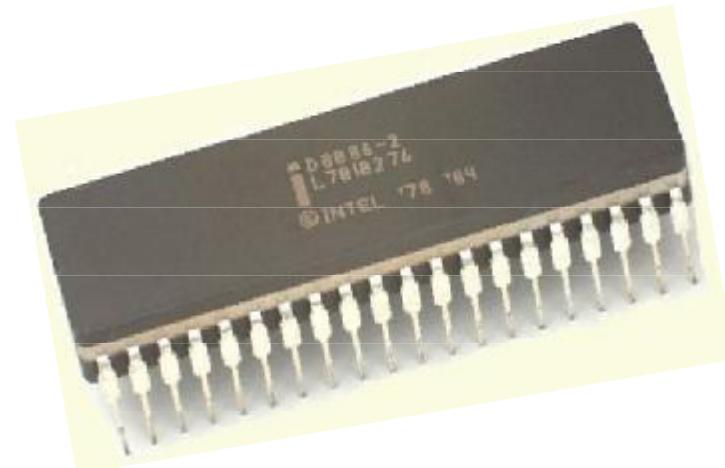
Draw the architecture of 8086 and understand the functions of different pins of 8086

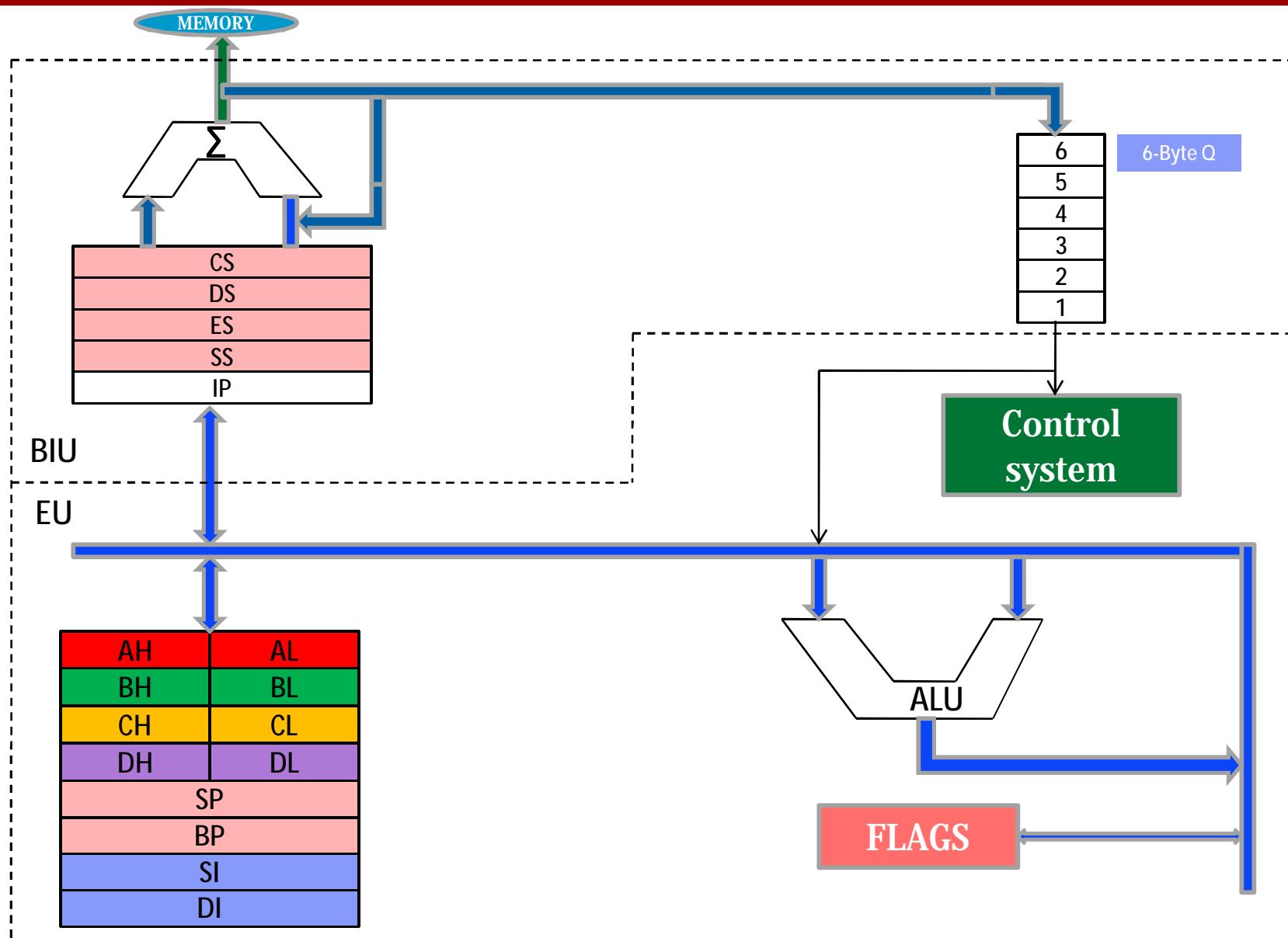


Understand the operating modes of 8086

Key Features:

- ü Introduction date: March 1978
- ü It is 16-bit HMOS microprocessor implemented with 29,000 transistors
- ü It can be operated with clock Frequency of 5MHz
- ü Technology: HMOS
- ü Number of Pins : 40
- ü It has 20-bit Address lines and hence it can address $2^{20} = 1$ Mbytes memory location.
- ü It can generate 16-bit address for IO devices and can address $2^{16} = 64K$ IO ports.
- ü It can be operated in two Modes : Maximum and Minimum
- ü It has two stage pipeline architecture.
- ü Number of instructions: 135 instructions with eight 8-bit registers and eight 16-bit registers
- ü DC Power Supply +5v





- The architecture of 8086 provides a number of improvements over 8085 architecture. It supports a 16-bit ALU, a set of 16-bit registers and provides segmented memory addressing capability, a rich instruction set, powerful interrupt structure, fetched instruction queue for overlapped fetching and execution.
- The complete architecture of 8086 can be logically divided into two units a) Bus Interface Unit (BIU) and (b) Execution Unit (EU). Both units operate asynchronously to provide the 8086 to overlap instruction fetch and execution operation , which is called as parallel processing. This results in efficient use of the system bus and enhance system performance.

An instruction pipeline is a technique used in the design of microprocessors to increase the number of instructions that can be executed in a unit of time. Pipeline technique is used in advanced microprocessors where the microprocessor begins operation on next instruction before it has completed operation on the previous. That is, several instructions are simultaneously in the *pipeline* at a different stage of processing. The pipeline is divided into different Stages and each Stage can perform its particular operation simultaneously with the other stages. When a stage completes an operation, it passes the result to the next stage in the pipeline and fetches the next operation from the preceding stage. The final results of each instruction emerge at the end of the pipeline in rapid succession. Since all units perform operation concurrently on different instructions , it is known as parallel processor.

Pipelining of 8086

| INSTRUCTION NO. | EXECUTION PHASES | | | | | | | |
|-----------------|------------------|----------|-----------|-----------|-----------|-----------|-----------|-----------|
| 1 | Fetch-1 | Decode-1 | Execute-1 | | | | | |
| 2 | | Fetch-2 | Decode-2 | Execute-2 | | | | |
| 3 | | | Fetch-3 | Decode-3 | Execute-3 | | | |
| 4 | | | | Fetch-4 | Decode-4 | Execute-4 | | |
| 5 | | | | | Fetch-5 | Decode-5 | Execute-5 | |
| 6 | | | | | | Fetch-6 | Decode-6 | Execute-6 |
| Machine cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

Non-Pipelining Process of 8085

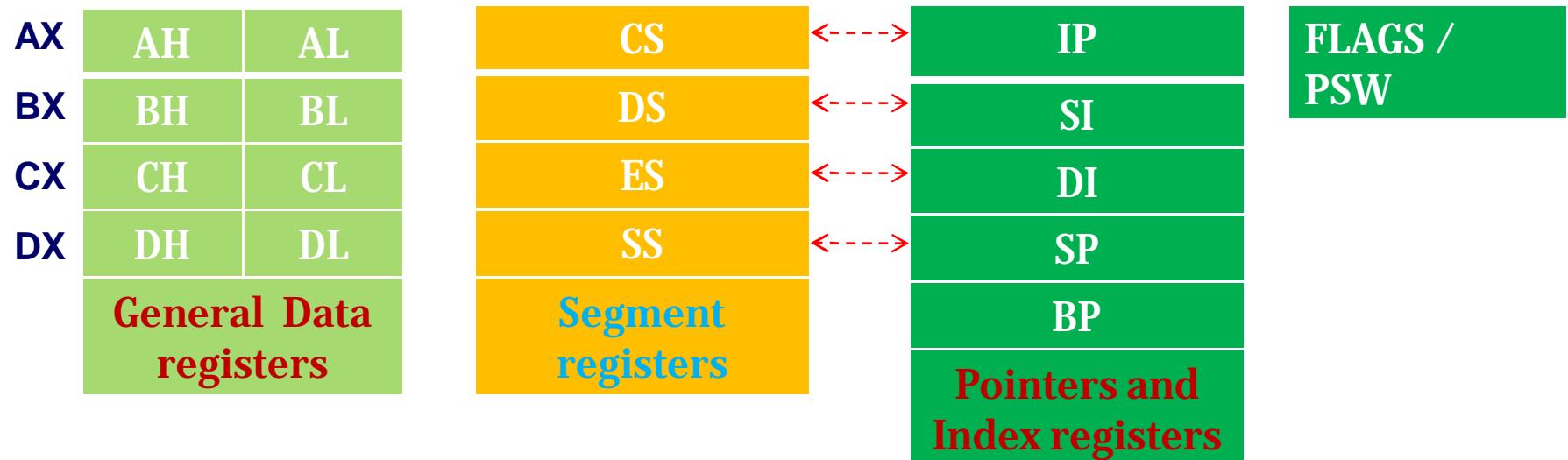
| | Instruction-1 | | | Instruction-2 | | | Instruction-3 | | |
|----------|---------------|----------|-----------|---------------|----------|-----------|---------------|----------|-----------|
| | Fetch-1 | Decode-1 | Execute-1 | Fetch-2 | Decode-2 | Execute-2 | Fetch-3 | Decode-3 | Execute-3 |
| M. cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

8086 has a powerful set of registers that can be grouped as

- Ø General Data register
- Ø Segment registers
- Ø Pointers & Index registers

Ø FLAG

- Ø Only GPRs can be accessed as
8/16-bit while others as 16-bit only

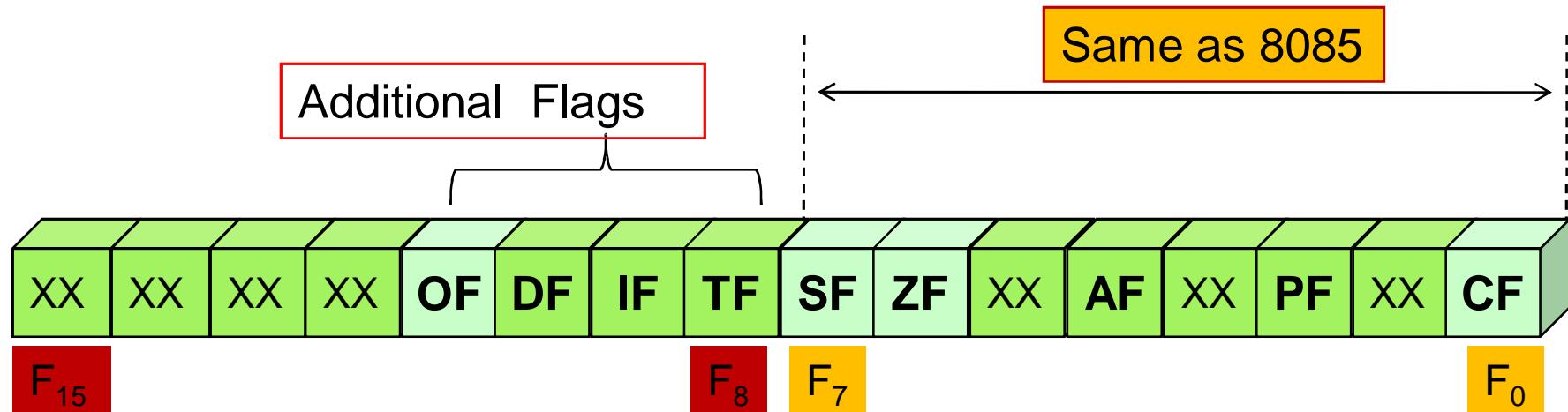


- Special Purpose Registers: The special purpose registers are
 - Ø Segment registers
 - Ø Pointers and index registers
- Segment Registers : Unlike 8085, the 8086 addresses a segmented memory of 1MB, which the 8086 is able to address. The 1 MB is divided into 16 logical segments ($16 \times 64 \text{ KB} = 1024 \text{ KB} = 1 \text{ MB}$). Each segment thus contains 64 Kbytes of memory. There are four segment registers, viz. Code Segment Register (CS), Data Segment Register (DS), Extra Segment Register (ES) and Stack Segment Register (SS).

Pointers and Index Registers

The pointers contain offset within the particular segments. The pointers IP, BP and SP usually contain offsets within the code, data and stack segments respectively. The index registers are used as general purpose registers as well as for offset storage in case of indexed, based indexed and relative based indexed addressing modes. The register SI is generally used to store the offset of source data in DMS while the register DI is used to store the offset of destination in DMS or EMS. The index registers are particularly useful for string manipulations.

- The FLAG is nothing but group of flip-flops which are affected (SET or RESET) immediately after an arithmetic or logical operation performed by the ALU.
- The flags of 8086 can be divided into two types: Conditional Flags and Control Flags
- Conditional Flags are affected immediately after an arithmetic or logical operation performed by the ALU. The SET or RESET condition of each flag is used to indicate the status of the result generated by the ALU. The 8086 has 6 conditional flags, out of which 5 are similar to the 8085 while Overflow flag is the additional flag.
- Control Flag are not affected by Arithmetic or logical operation performed by the ALU but programmer can SET or RESET these Flags to Control certain operation/Instructions.

**∅ Control Flags**

- IF: Interrupt enable flag
- DF: Direction flag
- TF: Trap flag

XX: Don't Care

∅ Status Flags

- CF: Carry flag
- PF: Parity flag
- AF: Auxiliary carry flag
- ZF: Zero flag
- SF: Sign flag
- OF: Overflow flag

The 6 Status or Conditional Flags are affected immediately after an arithmetic or logical operation performed by the ALU. The SET or RESET condition of each flag is used to indicate the status of the result generated by the ALU.

- **Sign Flag:** It is used to indicate whether the result is positive or negative. It will set ($SF=1$) if the result is –ve and if the result +ve then $SF=0$.
- **Zero Flag:** It is used to indicate whether the result is a Zero or non-zero. It will set ($ZF=1$) if the result is zero else $ZF=0$.
- **Auxiliary carry Flag:** It is used to indicate whether or not the ALU has generated a carry/Borrow from D3 bit position to D4 bit. It will set if there was a carry out from bit 3 to bit 4 of the result else $AF=0$. The auxiliary carry flag is used for binary coded decimal (BCD) operations.

- Parity Flag: It is used to indicate parity (Even or Odd) of the result. It will set if the parity is even else PF =0.
- Carry Flag: It is used to indicate whether a carry/Borrow has been generated /occurred during addition/subtraction It will set if there was a carry is generated from the MS-bit during addition, or borrow during subtraction/comparison else CF=0.
- Overflow Flag: The OF indicates a signed arithmetic result overflow. If result of an operation is too large a positive number or too small a negative number to fit in the destination then OF will SET, else it will RESET.

TF (Trap Flag) : It is used for Single step operation .If TF=1 then 8086 executes single instruction at a time and stop momentarily. If TF=0 then 8086 executes the given programme in natural sequence.

IF (Interrupt-enable flag) : When IF=1 then maskable Interrupt INTR will cause the CPU to transfer control to an interrupt vector location.

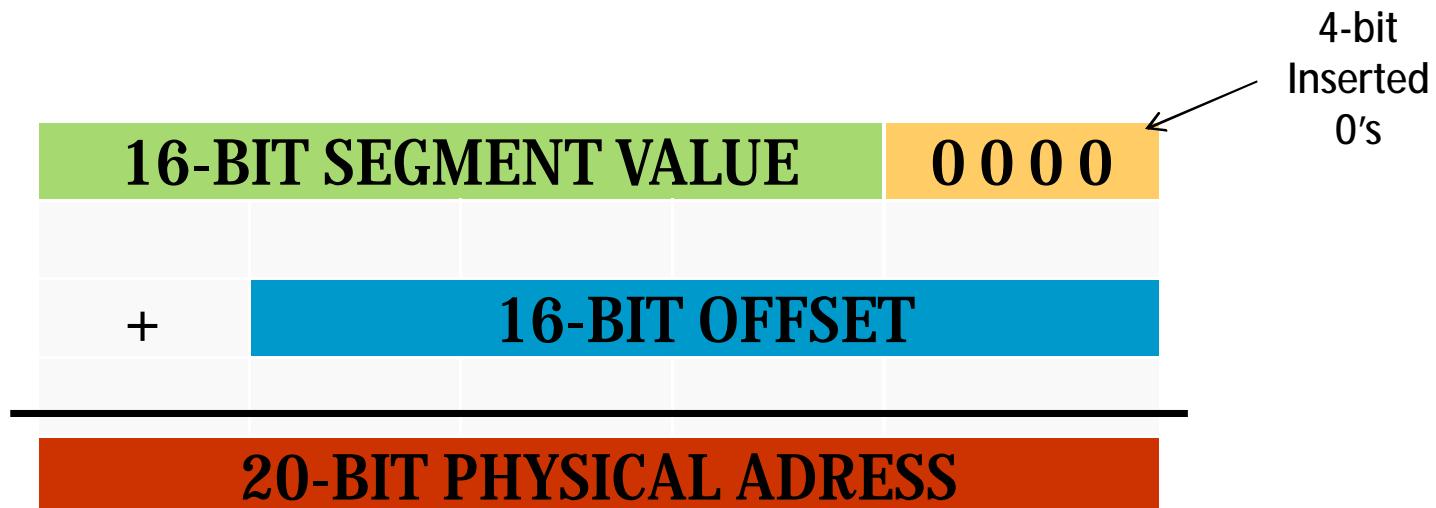
DF (Direction flag) : Causes string instructions to auto decrement/increment the index registers (SI/DI) by 1 (for byte operation) or 2 by word operation). If DF=1 will decrement and DF=0 will increment index registers.

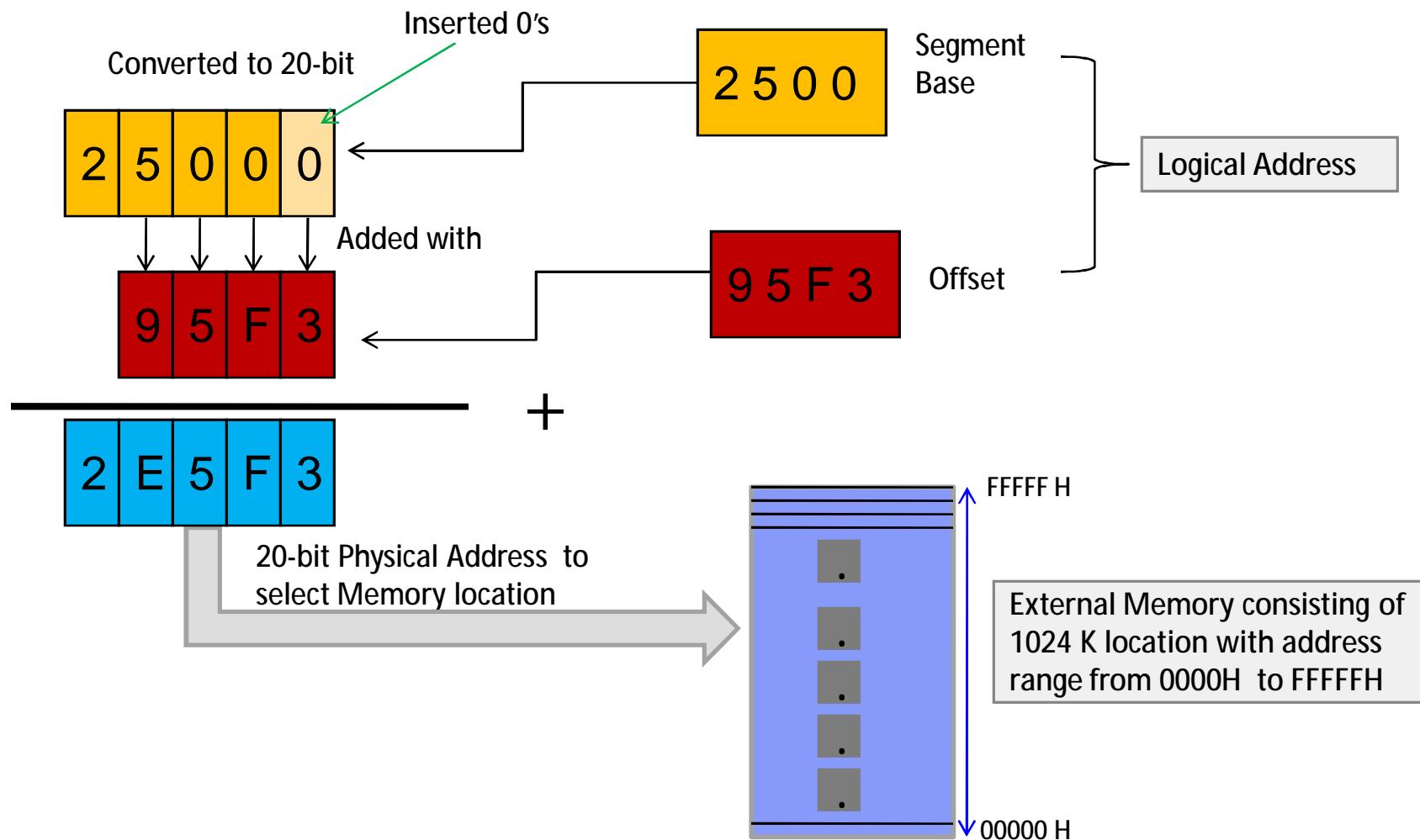
- Ü Since the 8086 can generate 20-bit physical address therefore it can access $2^{20} = 1048576$ locations or 1024 Kbytes location or 1 Mbytes locations addressed from 00000h TO FFFFFh .
- Ü For programme flexibility the 1Mbytes location is logically segmented (divided or organized) into
 - Ø Code Memory Segment (CMS),
 - Ø Data Memory Segment (DMS),
 - Ø Extra Memory Segment (EMS) and
 - Ø Stack Memory Segment (SMS).

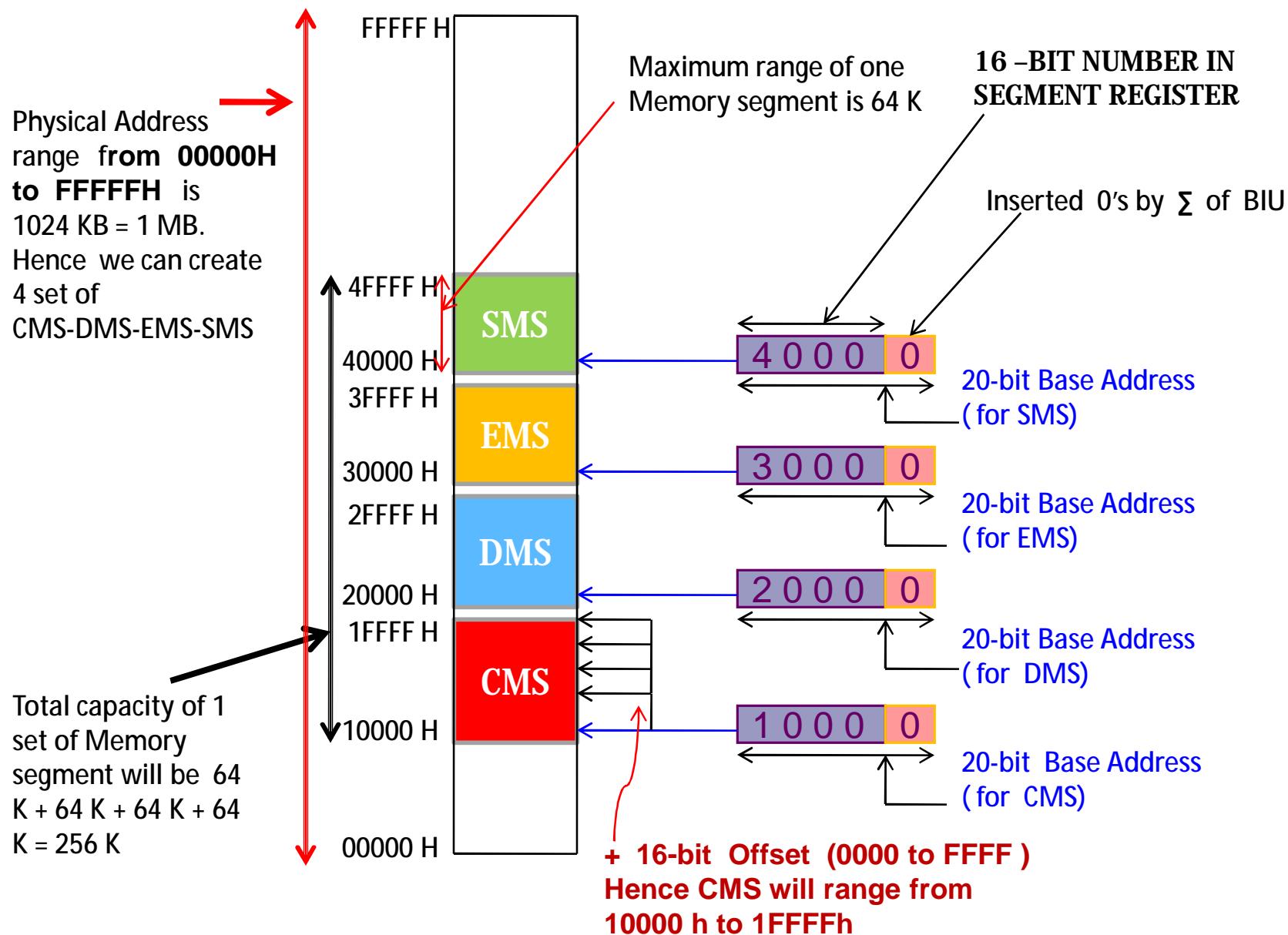
- ü Each memory segment can be maximum of 64 Kbytes.
- ü To access a particular location of memory segment the 20-bit physical address is generated by the addition of Base Address (BA) provided by the segment register and 16/8 bit offset address/displacements (OA) is provided by Pointers/index registers.

The selection of segment registers and pointers/index registers is according to the rule given in the table.

| Name of Memory segment | Segment register used for base value | Default Pointers/ Index register used for offset address | Memory segment used for | Segment selection rule |
|------------------------------|--------------------------------------|--|-------------------------|--|
| CMS (Code memory Segment) | CS | IP | Instructions | Automatic during execution of a programme to prefetch code. |
| DMS (Data memory Segment) | DS | BX/SI/16/8bit displacement | Local data | During execution of a string instruction or data transfer. |
| (Data memory Segment) | ES | DI/16/8bit displacement | External data | During execution of a string instruction or data transfer from IO. |
| SMS (Data memory Segment) | SS | SP/BP | Stack | During execution of a stack instruction. |







- ü Allows the memory capacity to be 1Mb even though the addresses associated with the individual register / instructions are only 16 bits wide.
- ü Facilitate the use of separate memory areas for the program, its data and the stack and allows a program and/or its data to be put into different areas of memory each time the program is executed. **Due to which relocability of information becomes efficient.**

ü The greatest advantage of segmented memory is that programs that reference logical addresses only can be loaded and run anywhere in memory. This is because the logical addresses always range from 00000h to 0FFFFh, independent of the code segment base. Such programs are said to be relocatable, meaning that they can be executed at any location in memory. The requirements for writing relocatable programs are

1. No reference should be made to physical addresses, and
2. No changes to the segment registers be allowed once initialised.

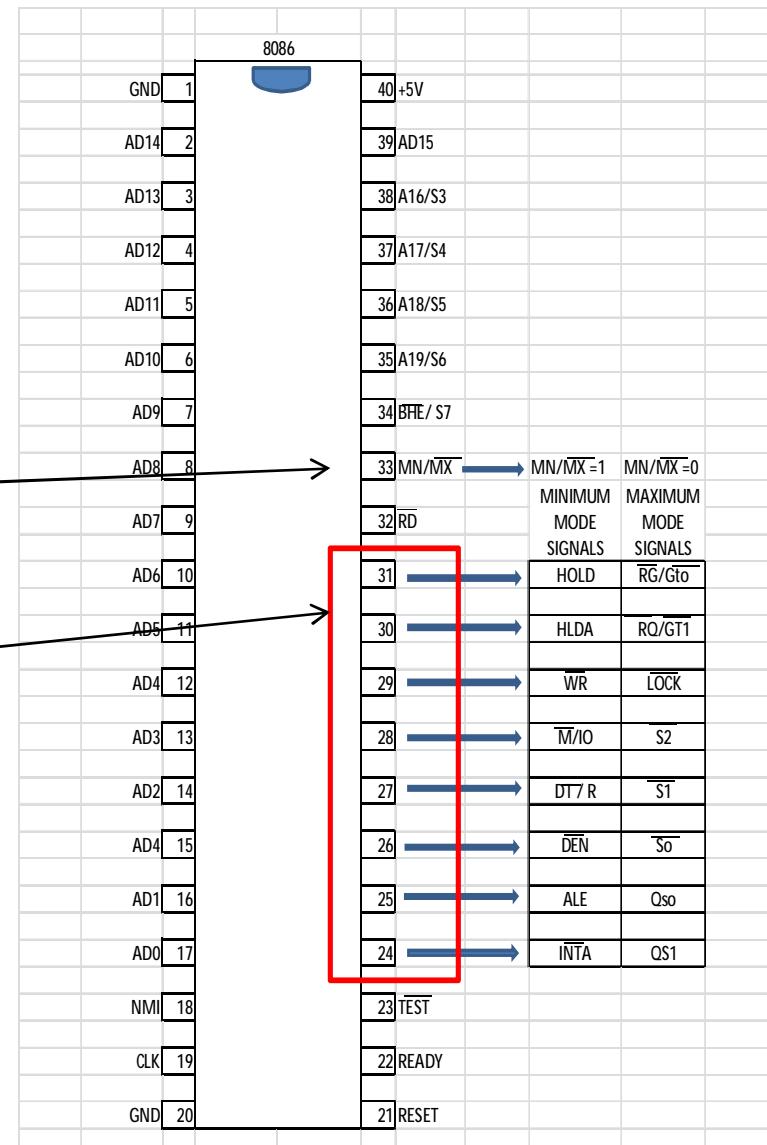
ü Since more than 1 set of CMS-DMS-EMS-SMS can be created therefore multiprogramming can be implemented easily. Also sharing of segments by different process is also possible.

8086 works in two modes:

- 1) Minimum Mode and 2) Maximum Mode

If pin 33 MN/MX is high, it works in minimum mode and If pin 33 MN/MX is low, it works in maximum mode.

Pins 24 to 31 generates two different sets of signals. One set of signals is generated in minimum mode. Other set of signals is generated in maximum mode.



- Pin 24 is an interrupt acknowledge. When microprocessor receives INTR signal, it uses this pin to send acknowledgment by generating 3 active low signal.
- Pin 25 is an Address Latch Enable signal. It indicates that valid address is generated on bus $AD_{15} - AD_0$.It generates a pulse during T_1 state.It is connected to enable external latch .
- Pin 26 is a Data Enable signal. This signal is used to enable the external transceiver like 8286. Transceiver is used to separate the data from the address/data bus $AD_{15} - AD_0$.It is an active low signal.
- Pin 27 is a Data Transmit/Receive signal. It controls the direction of data flow through the transceiver. When it is high, data is transmitted out.When it is low, data is received in.

- Pin 28 is issued by the microprocessor to distinguish whether memory or /O access. When it is high, memory can be accessed. When it is low, I/O devices can be accessed.
- Pin 29 is a Write signal. It is used to write data in memory or output device depending on the status of M/IO signal. It is an active low signal.
- Pin 30 is a Hold Acknowledgement signal. It is issued after receiving the HOLD signal. It is an active high signal.
- Pin 31 During DMA operation microprocessor receives HOLD signal from DMA controller.

QS_1 and QS_0
Pin 24 and 25

- These pins provide the status of internal instruction queue.

| QS_1 | QS_0 | Status |
|--------|--------|---|
| 0 | 0 | No operation |
| 0 | 1 | 1 st byte of opcode from queue |
| 1 | 0 | Empty queue |
| 1 | 1 | Subsequent byte from queue |

$\overline{S_0}, \overline{S_1}, \overline{S_2}$
Pin 26, 27, 28

- These status signals indicate the operation being to be performed by the microprocessor.
- These information decoded by the Bus Controller 8288 which generates all memory and I/O control signals.

| S_2 | S_1 | S_0 | Status |
|-------|-------|-------|-----------------------|
| 0 | 0 | 0 | Interrupt Acknowledge |
| 0 | 0 | 1 | I/O Read |
| 0 | 1 | 0 | I/O Write |
| 0 | 1 | 1 | Halt |
| 1 | 0 | 0 | Opcode Fetch |
| 1 | 0 | 1 | Memory Read |
| 1 | 1 | 0 | Memory Write |
| 1 | 1 | 1 | Passive |

LOCK

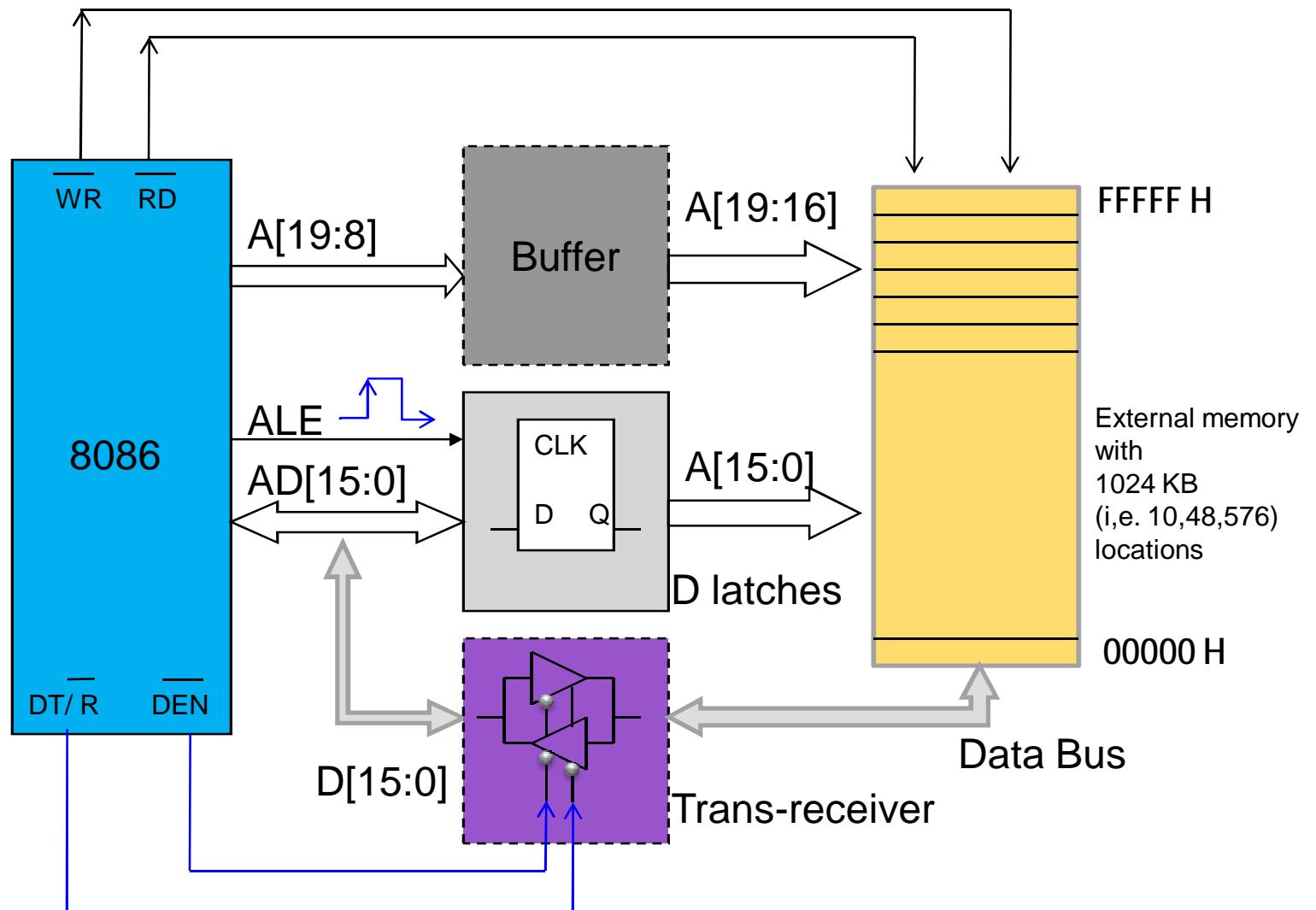
Pin 29

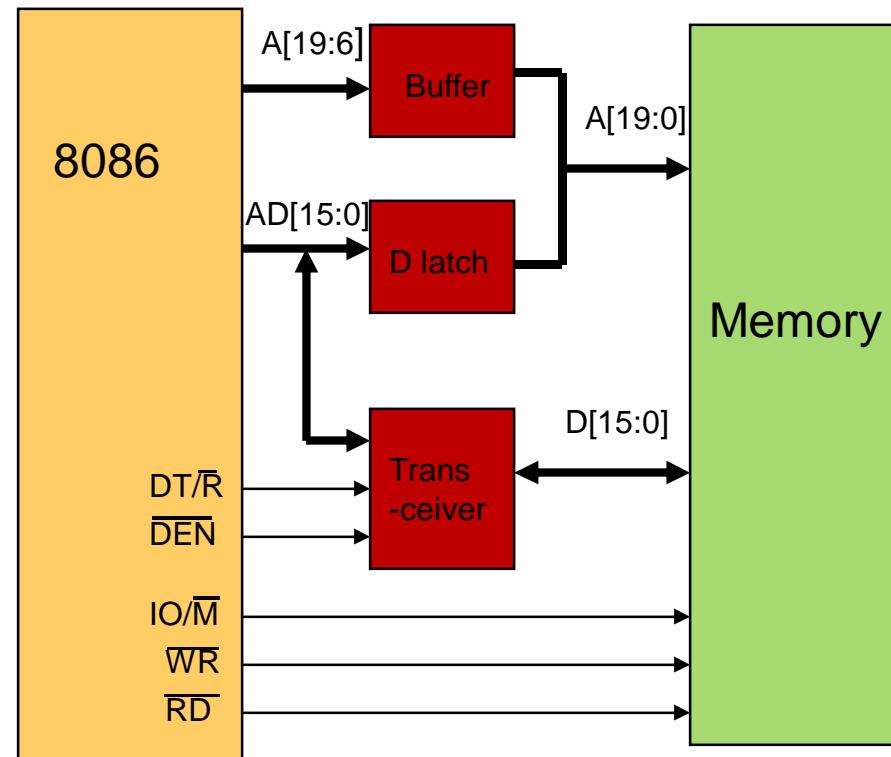
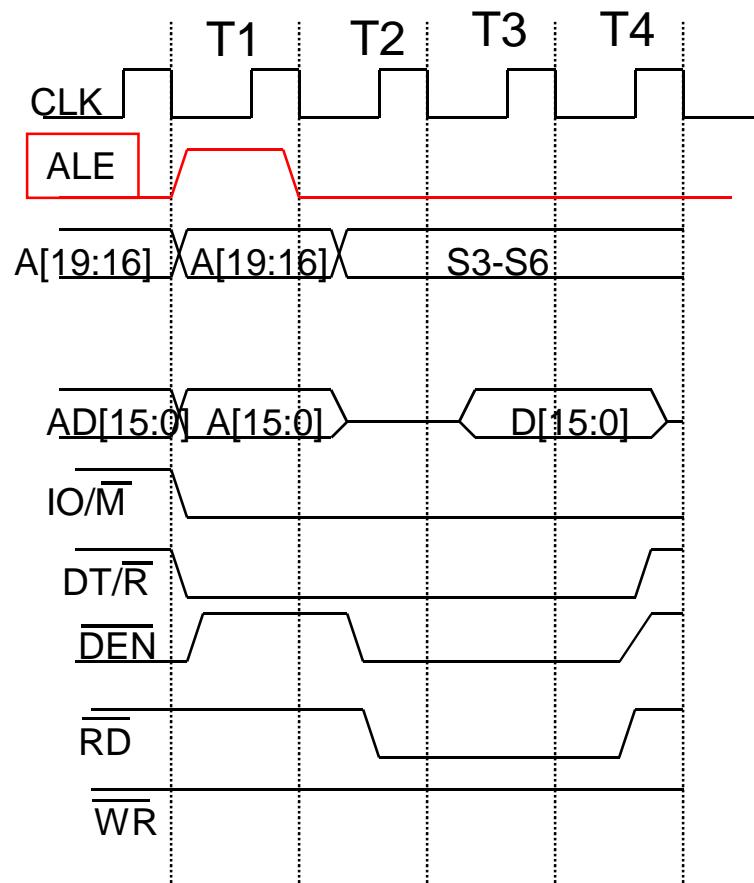
- This signal indicates that external processors like 8087 should not request CPU to relinquish the system bus as it is locked with important operation. This pin is activated by using LOCK prefix before any instruction.
- When it goes low, all interrupts are masked and HOLD request is not granted.

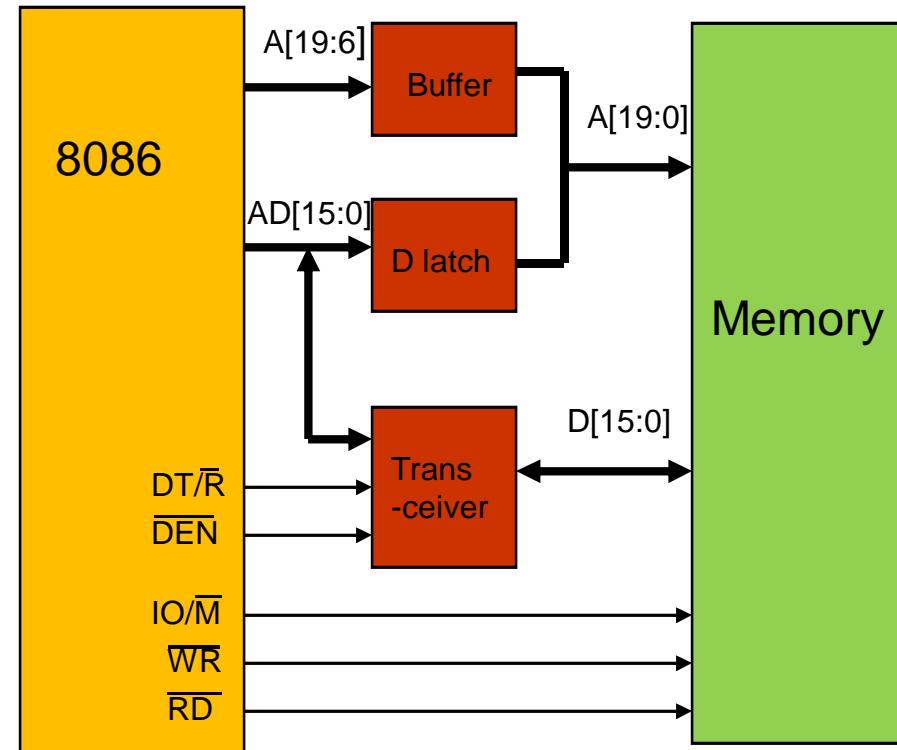
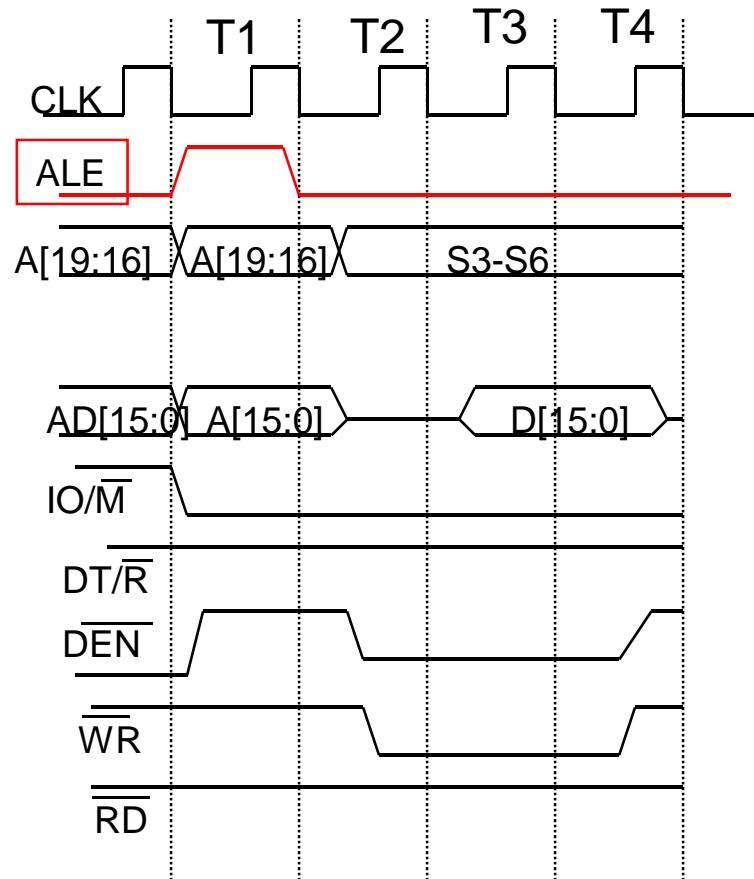
$\overline{RQ}/\overline{GT}_1$ and $\overline{RQ}/\overline{GT}_0$

Pin 30 and 31 (Bi-directional)

- These are Request/Grant pins.
- External processors like 8087 can request the CPU through these lines to release the system bus.
- After receiving the request, CPU sends acknowledge signal on the same lines.
- RQ/GT_0 has higher priority than RQ/GT_1 .







- What is pipeline?
- Explain the function of Q 8086.
- Describe the function EU & BIU.
- Explain the function of various registers of 8086.
- Explain function of segment register & pointer .
- What are the various Flag of 8086?
- How 20-bit address is generated in 8086?
- Explain the Minimum and Maximum mode of 8086.
- Explain the timing diagram of memory read 8086.
- Explain the timing diagram of memory write 8086.

1. The 8086 logically divided into:

- ü BIU &
- ü EU

Both units operate asynchronously to give the 8086 an overlapping instruction fetch and execution mechanism which is called as Pipelining.

2. For programme flexibility the 1Mbytes location is logically segmented (divided or organized) into Code Memory Segment (CMS), Data Memory Segment (DMS), Extra Memory Segment (EMS) and Stack Memory Segment (SMS).

3. To access a particular location of memory segment the 20-bit physical address is generated by the addition of Base Address (BA) provided by the segment register and 16/8 bit offset address/displacements (OA) is provided by Pointers/index registers.

4. The flags of 8086 can be divided into two types: Conditional Flags and Control Flags

5. **8086 works in two modes:** 1) Minimum Mode 2) Maximum Mode.

CHAPTER-3 Instruction Set of 8086 Microprocessor



Topic 1: Machine Language Instruction format,
addressing modes



Topic 2: Instruction set,



Topic 3: Groups of Instructions,

CHAPTER-3 SPECIFIC OBJECTIVE / COURSE OUTCOME

The student will be able to:



Understand the different types of instructions



Identify the addressing modes of instruction and the operation of an instructions

A programme is nothing but set of Instructions written sequentially one below the other and stored in computers memory for execution by microprocessor.

Instruction consists of a mnemonic and one or two operands (data).

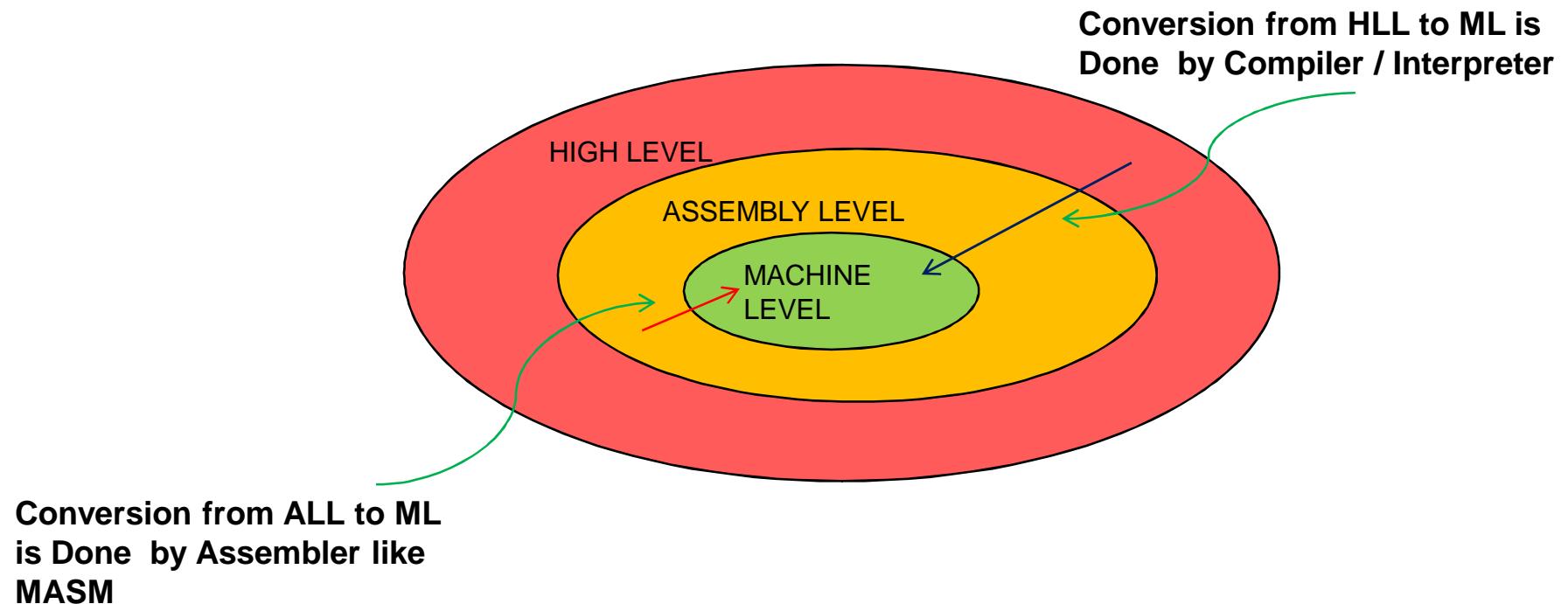
Ø Machine Language: In this Programs consist of 0s and 1s.

Ø Assembly Languages : It uses short form notations , called , **mnemonics** , to write a programme .The Mnemonics are like MOV , ADD , SUB, etc.

Ø High level languages: It uses English like sentences with proper syntax to write a programme.

Ø Assembler translates Assembly language program into machine code.

Ø Compilers like Pascal, Basic, C etc **translate the HLL program into machine code.** The programmer does not have to be concerned with internal details of the CPU.



q General Format of Instructions is

Label: **Opcode** **Operands** ; **Comment**

Ø Label: It provides a symbolic address that can be used in branch instructions

Ø Opcode: It specifies the type of instructions

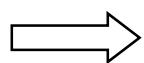
Ø Operands: Instructions of 8086 family can have one, two, or zero operand

Ø Comments: programmers can use for effective reference

q Machine Code Format



MOV AL, BL



1000100011000011

binary code of MOV

- Ø Addressing modes is define as the way in which data is addressed in the operand part of the instruction. It indicates the CPU finds from where to get data and where to store results
- Ø When a CPU executes an instruction, it needs to know where to get data and where to store results. Such information is specified in the operand fields of the instruction.

1 0 0 0 1 0 0 0 1 1 0 0 0 0 1 1 → **MOV AL, BL**



- Ø An operand can be:
 - A datum
 - A register
 - A memory location

| <i>Addressing Modes</i> | <i>Examples</i> |
|--|-------------------|
| 1] Immediate addressing | MOV AL, 1FH |
| 2] Register addressing | MOV AL, BH |
| 3] Direct addressing | MOV [0200H], AH |
| 4] Register Indirect addressing | MOV DX, [SI] |
| 5] Relative Based addressing | MOV BX, [SI+4] |
| 6] Relative Indexed addressing | MOV [DI+8], BL |
| 7] Based indexed addressing | MOV [BP+SI], AL |
| 8] Relative Based indexed with displacement addressing | MOV AL, [BX+SI+2] |

In this AM 8/16-bit Data is specified in the operand part of instruction immediately

- MOV AL,1Fh
- MOV AX,0FC8h
- MOV AH,4Eh
- MOV DX,1F00h

- Ø In this data is specified through register in operand part of instruction
- Ø Operands are the names of internal register. The processor gets data from the register specified by instruction .

For Example: *move the value of register BL to register AL*



- q If AX = 1F00H and BX=8086H, after the execution of MOV AL, BL what are the new values of AX and BX?

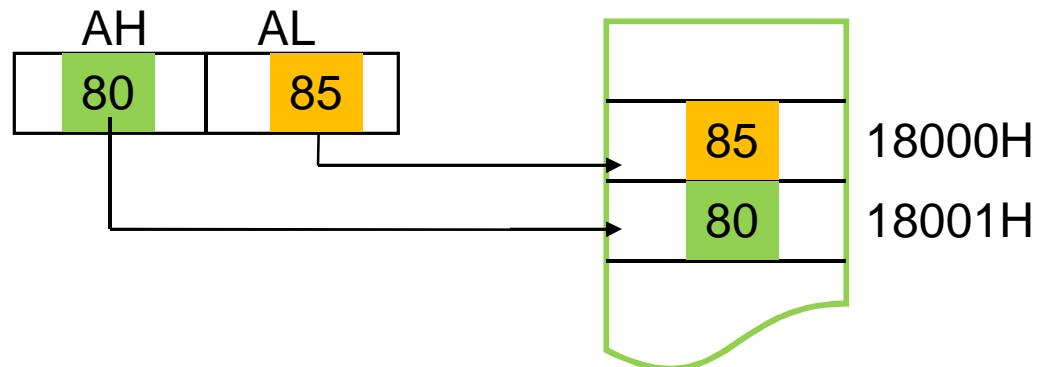
- Ø In this AM 16-bit OFFSET address is specified , with symbol [] , in the operand part of the instruction.
- Ø The processor will access memory location by adding this OFFSET with Base address given by DS.

$$DS \times 10H + \text{OFFSET} = \text{Memory location}$$

- Example: If $DS = 1000H$, then explain the operation

MOV AX, 8085 H
MOV [8000H], AX

$$\begin{array}{r} DS: 10000 \\ + \text{Disp: } 8000 \\ \hline 18000 \end{array}$$



- Example: If $DS = 1000H$, then explain the operation

1 MOV AX, F2A5 H
 MOV [8000H], AH

2 MOV AX, F2A5 H
 MOV [8000H], AL

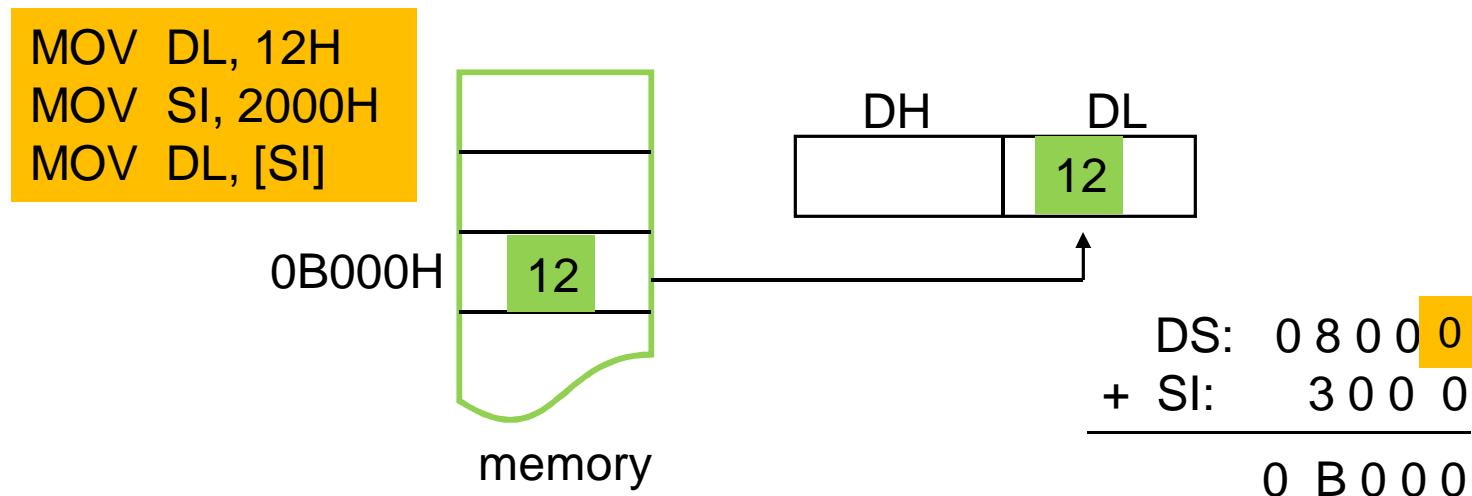
- Ø In this AM OFFSET address is specified indirectly through one of the registers BX, SI, DI in the instruction operand.
- Ø The index register is specified using symbol [].
- Ø This value is added with DS to generate 20-bit Physical address

For Example: **MOV DL, [SI]**

- Ø Memory address is calculated as following:

$$\left[\begin{array}{l} \text{DS} \end{array} \right] \times 10H + \left[\begin{array}{l} \text{BX} \\ \text{SI} \\ \text{DI} \end{array} \right] = \text{20-bit Memory address}$$

Ø Example 1: assume $DS = 0800H$ then explain the operation



Ø Example 2: assume $DS = 0900H$, $BX=3000H$

```
MOV DL,E7H  
MOV [BX], DL
```



- Ø In this AM OFFSET address is specified indirectly by adding an 8-bit (or 16-bit) constant (displacement) with one of the registers BX, BP in the instruction operand.
- Ø If BX appears in the instruction operand field, segment register DS is used in address calculation and If BP appears in the instruction operand field, segment register SS is used in address calculation
- Ø Calculation of memory address

$$\begin{bmatrix} DS \\ SS \end{bmatrix} \times 10H + \begin{bmatrix} BX \\ BP \end{bmatrix} + \begin{bmatrix} 8/16\text{-bit Displacement} \end{bmatrix} = \text{20-bit Memory address}$$

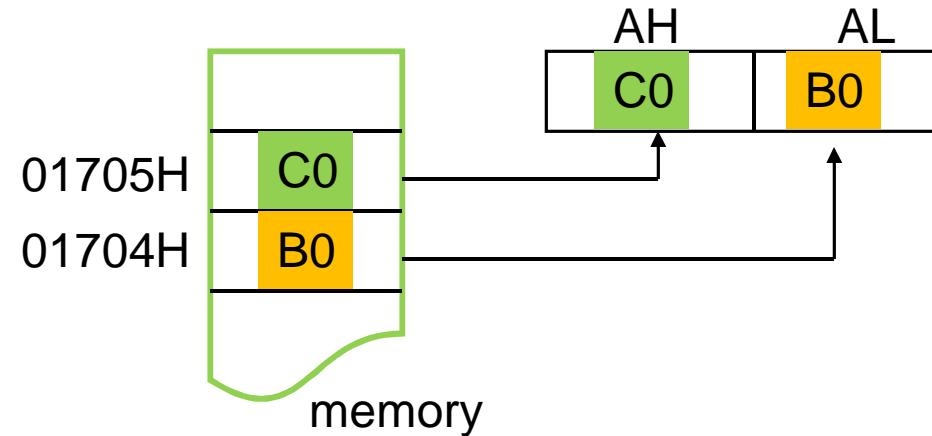
For Example: If DS = 2000H then explain

MOV AX, [BX+4]

Ø Example 1: assume $DS = 0100H$, $BX=0700H$

```
MOV AX, F4E0H  
MOV AX, [ BX+4 ]
```

$$\begin{array}{r} \text{DS: } 0\ 1\ 0\ 0\ 0 \\ + \text{ BX: } 0\ 7\ 0\ 0 \\ + \text{ Disp.: } 0\ 0\ 0\ 4 \\ \hline 0\ 1\ 7\ 0\ 4 \end{array}$$



Ø Example 2: assume $SS = 0A00H$, $BP=0012H$, $CH = ABH$

```
MOV [BP +7], CH
```



Ø In this AM OFFSET address is specified indirectly by adding an 8-bit (or 16-bit) constant (displacement) with one of the Index registers SI, DI in the instruction operand. This value is then added with DS to generate 20-bit Physical address

Ø Calculation for memory address

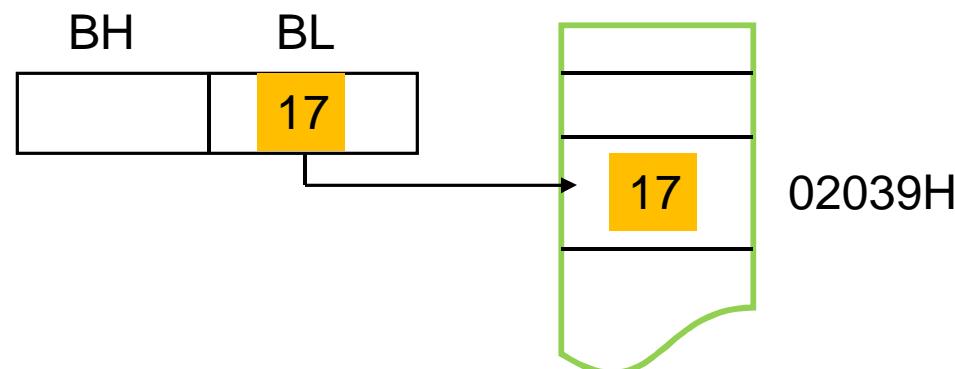
$$DS \times 10H + \begin{bmatrix} SI \\ DI \end{bmatrix} + [8 / 16 \text{ bit Displacement}] = \text{Memory address}$$

For Example:

```
MOV BL,17H
MOV DI,0030H
MOV [DI+9],BL
```

Ø Example: assume DS = 0200H

$$\begin{array}{r} \text{MOV [DI+9], BL} \\ \text{DS: } 0\ 2\ 0\ 0\ 0 \\ + \text{ DI: } 0\ 0\ 3\ 0 \\ - \text{ Disp.: } 0\ 0\ 0\ 9 \\ \hline 0\ 2\ 0\ 3\ 9 \end{array}$$



Ø In this AM OFFSET address is specified indirectly by adding one of the Index registers SI /DI with based register BX / BP in the instruction operand. This value is added with DS to generate 20-bit Physical address.

Ø Calculation for memory address

$$\begin{bmatrix} \text{DS} \\ \text{SS} \end{bmatrix} \times 10H + \begin{bmatrix} \text{BX} \\ \text{BP} \end{bmatrix} + \begin{bmatrix} \text{SI} \\ \text{DI} \end{bmatrix} = \text{20-bit Memory address}$$

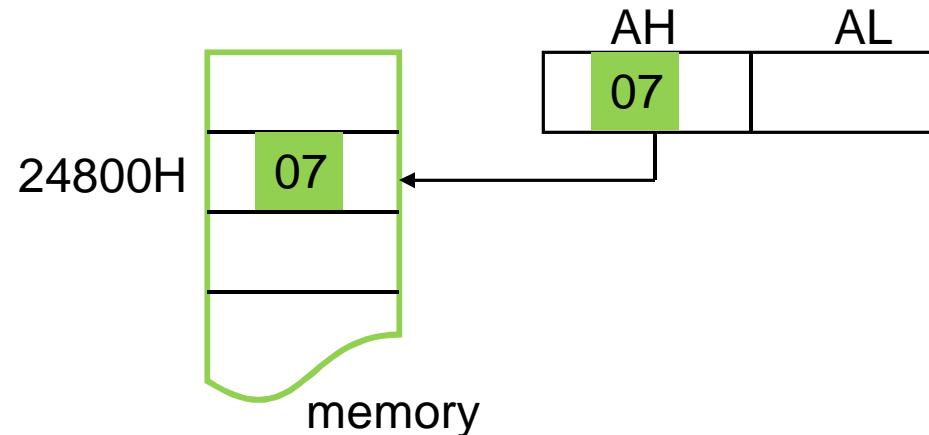
For Example: MOV [BX][SI], AH
 or
 MOV [BX+SI], AH

- q If BX appears in the instruction operand field, then segment register DS is used in address calculation
- q If BP appears in the instruction operand field, segment register SS is used in address calculation

Ø Example 1: assume SS = 2000H explain the operation

```
MOV AH,07H  
MOV SI, 0800H  
MOV BP,4000H  
MOV [BP][SI], AH
```

$$\begin{array}{r} \text{SS: } 2\ 0\ 0\ 0\ 0 \\ + \text{BP: } 4\ 0\ 0\ 0 \\ + \text{SI.: } 0\ 8\ 0\ 0 \\ \hline 2\ 4\ 8\ 0\ 0 \end{array}$$



Ø Example 2: assume DS = 0B00H, BX=0112H, DI = 0003H, CH=ABH

MOV [BX+DI], CH



- Ø In this AM OFFSET address is specified indirectly by adding 8/16-bit displacement with one of the Index registers SI /DI with based register BX / BP in the instruction operand. This value is added with DS to generate 20-bit Physical address.
- Ø Calculate memory address

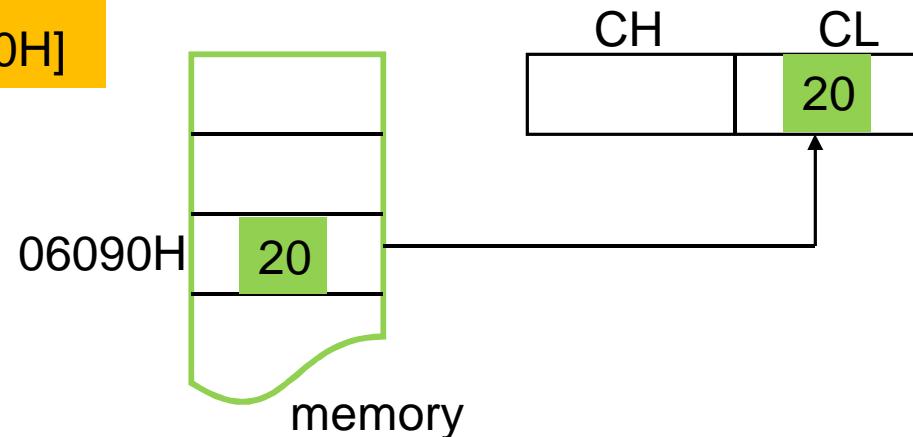
$$\begin{bmatrix} \text{DS} \\ \text{SS} \end{bmatrix} \times 10\text{H} + \begin{bmatrix} \text{BX} \\ \text{BP} \end{bmatrix} + \begin{bmatrix} \text{SI} \\ \text{DI} \end{bmatrix} + \begin{bmatrix} \text{8/16-bit} \\ \text{displacement} \end{bmatrix}$$

For Example: MOV CL, [BX+DI+2080H]

Ø Example 1: assume $DS = 0300H$, $BX=1000H$, $DI=0010H$

```
MOV BX,1000H  
MOV DI, 0010H  
MOV CL, [BX+DI+2080H]
```

$$\begin{array}{r} \text{DS: } 0\ 3\ 0\ 0\ 0 \\ + \text{ BX: } 1\ 0\ 0\ 0 \\ + \text{ DI.: } 0\ 0\ 1\ 0 \\ + \text{ Disp. } 2\ 0\ 8\ 0 \\ \hline 0\ 6\ 0\ 9\ 0 \end{array}$$



Ø Example 2: assume $SS = 1100H$, $BP=0110H$, $SI = 000AH$, $CH=ABH$

```
MOV [BP+SI+0010H], CH
```



- 1] Data transfer instructions
- 2] Arithmetic instructions
- 3] String instructions
- 4] Bit manipulation instructions
- 5] Loop and jump instructions
- 6] Subroutine and interrupt instructions
- 7] Processor control instructions

Various Data transfer Instructions are**a) Memory/Register Transfers**

| | |
|------|---|
| MOV | Move byte or word to register or memory |
| XCHG | Exchange byte or word |
| XLAT | Translate byte using look-up table |

b) Stack Transfers

| | |
|-------|-----------------------|
| PUSH | Push data onto stack |
| PUSHF | Push flags onto stack |
| POP | Pop data from stack |
| POPF | Pop flags off stack |

c) AH/Flags Transfers

| | |
|------|---------------------|
| LAHF | Load AH from flags |
| SAHF | Store AH into flags |

d) Address Translation

LEA

Load effective address

LDS

Load pointer using data segment

LES

Load pointer using extra segment

e) Port I/O Instructions

IN

Input byte or word from port

OUT

Output word to port

| Addition | |
|----------------|-----------------------------------|
| ADD | Add byte or word |
| ADC | Add byte or word with carry |
| INC | Increment byte or word by 1 |
| AAA | ASCII adjust for addition |
| DAA | Decimal adjust for addition |
| Subtraction | |
| SUB | Subtract byte or word |
| SBB | Subtract byte or word with borrow |
| DEC | Decrement byte or word by 1 |
| NEG | Negate byte or word |
| AAS | ASCII adjust for subtraction |
| DAS | Decimal adjust for subtraction |
| Multiplication | |
| MUL | Multiply byte or word unsigned |
| IMUL | Integer multiply byte or word |
| AAM | ASCII adjust for multiplication |
| Division | |
| DIV | Divide byte or word unsigned |
| IDIV | Integer divide byte or word |
| AAD | ASCII adjust for division |
| CBW | Convert byte to word |
| CWD | Convert word to double word |

a) Logical Instructions

q NOT *Destination*

- § Inverts each bit of the destination
- § Destination can be a register or a memory location

q AND *Destination, Source*

- § Performs logic AND operation for each bit of the destination with corresponding source bit and stores result into destination
- § Source can be immediate no while destination can be register or memory
- § Destination and source can not be both memory locations at the same time

q OR *Destination, Source*

- § Performs logic OR operation for each bit of the destination with source; stores result into destination
- § Source can be immediate no while destination can be register or memory
- § Destination and source can not be both memory locations at the same time

a) Logical Instructions

q XOR *Destination, Source*

- § Performs logic XOR operation for each bit of the destination with source; stores result into destination
- § Source can be immediate no while destination can be register or memory
- § Destination and source can not be both memory locations at the same time

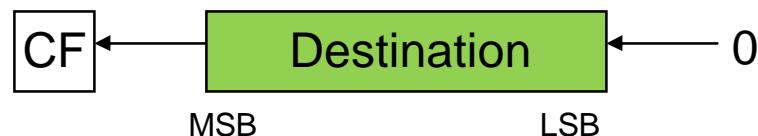
q TEST *Destination, Source*

- § Performs logic AND operation for each bit of the destination with source
- § Updates Flags depending on the result of AND operation
- § Do not store the result of AND operation anywhere

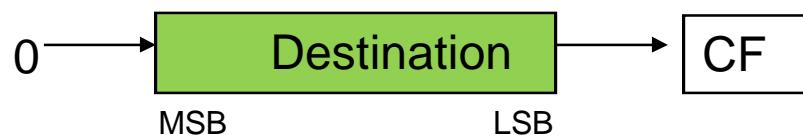
b) Shift Instruction

q SHL *Destination, Count*

- § SHift LEFT destination bits; the number of times bits shifted is given by CL
- § During the shift operation, the MSB of the destination is shifted into CF and zero is shifted into the LSB of the destination
- § Destination can be a register or a memory location

q SHR *Destination, Count*

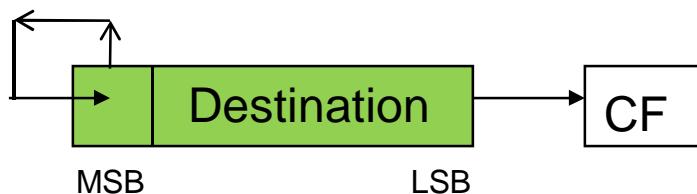
- § SHift RIGHT destination bits; the number of times bits shifted is given by CL
- § During the shift operation, the LSB of the destination is shifted into CF and zero is shifted into the MSB of the destination
- § Destination can be a register or a memory location



b) Shift Instructions

q SAR *Destination, Count*

- § Shift RIGHT destination bits; the number of times bits shifted is given by CL
- § The LSB of the destination is shifted into CF and the MSB of the destination is copied in the MSB itself i.e, it remains the same
- § Destination can be a register or a memory location



c) Rotate Instructions

q ROL *Destination, Count*

- § Left shift destination bits; the number of times bits shifted is given by CL
- § The MSB of the destination is shifted into CF, it is also rotated into the LSB .
- § Destination can be a register or a memory location

q ROR *Destination, Count*

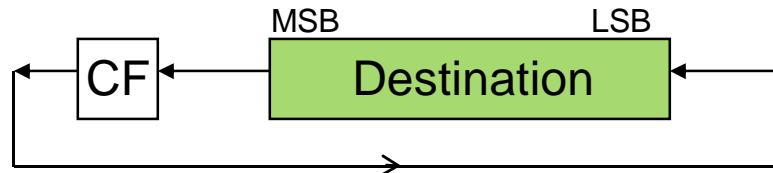
- § Right shift destination bits; the number of times bits shifted is given by CL
- § The LSB of the destination is shifted into CF, it is also rotated into the MSB .
- § Destination can be a register or a memory location



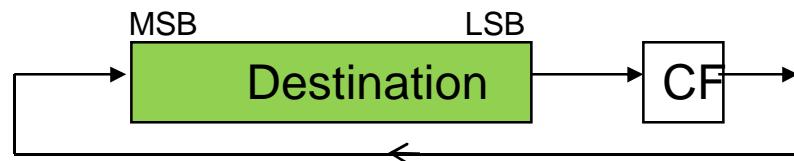
c) Rotate Instructions

q RCL *Destination, Count*

- § Left shift destination bits; the number of times bits shifted is given by CL
- § The MSB of the destination is shifted into CF; the old CF value is rotated into the LSB.
- § Destination can be a register or a memory location

q RCR *Destination, Count*

- § Right shift destination bits; the number of times bits shifted is given by CL
- § The LSB of the destination is shifted into CF, the old CF value is rotated into the MSB.
- § Destination can be a register or a memory location



- q String is a collection of bytes or words stored in successive memory locations of DMS or EMS that can be up to 64KB in length .
- q String instructions can have two operands. One is source string and the second is destination string .
 - § Source string is located in Data Segment and SI register points to the current element of the source string
 - § Destination string is located in Extra Segment and DI register points to the current element of the destination string

| DS : SI | |
|-----------|----|
| 0510:0000 | 5F |
| 0510:0001 | 4E |
| 0510:0002 | 4A |
| 0510:0003 | 5B |
| 0510:0004 | D0 |
| 0510:0005 | CA |
| 0510:0006 | 55 |

Source String

| ES : DI | |
|-----------|----|
| 02A8:2000 | 5F |
| 02A8:2001 | 4E |
| 02A8:2002 | 4A |
| 02A8:2003 | 5B |
| 02A8:2004 | D0 |
| 02A8:2005 | CA |
| 02A8:2006 | 55 |

Destination String

Repeat Prefix Instructions

q REP *String Instruction*

— The prefix instruction repeatedly execute the instruction until CX AUTO-decrements to 0 (During the execution, CX is decremented by one after execution of the string instruction).

— For Example: **MOV CX, 09**
REP MOVSB

By the above two instructions, the microprocessor will execute MOVSB 9 times.

— Execution flow of REP MOVSB is as below:

While (CX!=0)
{
 CX = CX - 1;
 MOVSB;
}
OR

Check_CX: If CX!=0 Then
 CX = CX - 1;
 MOVSB;
 goto Check_CX;
next instruction

Repeat Prefix Instructions

q REPZ *String Instruction*

§ Repeatedly execute the string instruction until CX=0 OR zero flag is clear

q REPNZ *String Instruction*

§ Repeatedly execute the string instruction until CX=0 OR zero flag is set

q REPE *String Instruction*

§ Repeatedly execute the string instruction until CX=0 OR zero flag is clear

q REPNE *String Instruction*

§ Repeatedly execute the string instruction until CX=0 OR zero flag is set

q MOVSB (MOVSW)

- § Move a byte (word) at source memory location of DMS (DS:SI) to destination memory location (ES:DI) and update SI and DI according to status of DF.
- § After transfer Increment SI/DI by 1 (or 2) if DF=0 and Decrement SI/DI if DF=1.

Example:

| | DS : SI | ES : DI |
|---------------|-----------------|-----------------|
| MOV AX, 0510H | 0510:0000 5E | 0300:0100 5E |
| MOV DS, AX | 0510:0001 48 | 0300:0101 ? |
| MOV SI, 0 | 0510:0002 4F | 0300:0102 ? |
| MOV AX, 0300H | 0510:0003 50 | 0300:0103 ? |
| MOV ES, AX | 0510:0004 50 | 0300:0104 ? |
| MOV DI, 100H | 0510:0005 45 | |
| CLD | 0510:0006 52 | |
| MOV CX, 5 | | |
| REP MOVSB | | |
| INT 21 | | |

Source String Destination String

q MOVSB (MOVSW)

- § Move a byte (word) at source memory location of DMS (DS:SI) to destination memory location (ES:DI) and update SI and DI according to status of DF.
- § After transfer Increment SI/DI by 1 (or 2) if DF=0 and Decrement SI/DI if DF=1.
- § Example:

| | DS : SI | ES : DI |
|---------------|--------------|--------------|
| MOV AX, 0510H | 0510:0000 5E | 0300:0100 5E |
| MOV DS, AX | 0510:0001 48 | 0300:0101 48 |
| MOV SI, 0 | 0510:0002 4F | 0300:0102 4F |
| MOV AX, 0300H | 0510:0003 50 | 0300:0103 50 |
| MOV ES, AX | 0510:0004 50 | 0300:0104 50 |
| MOV DI, 100H | 0510:0005 45 | |
| CLD | 0510:0006 52 | |
| MOV CX, 5 | | |
| REP MOVSB | | |
| INT 21 | | |

Source String Destination String

q CMPSB (CMPSW)

- § Compare bytes (words) at memory locations DS:SI and ES:DI;
update SI and DI according to DF and the width of the data being compared

§ Example:

Assume: ES = 02A8H

DI = 2000H

DS = 0510H

SI = 0000H

CLD

MOV CX, 7

REPZ CMPSB

INT 21

What will be the values of CX after

The execution?

| DS : SI | | ES : DI | |
|-----------|----|-----------|----|
| 0510:0000 | 4D | 02A8:2000 | 4D |
| 0510:0001 | 4A | 02A8:2001 | 4A |
| 0510:0002 | 45 | 02A8:2002 | 45 |
| 0510:0003 | 54 | 02A8:2003 | 54 |
| 0510:0004 | 48 | 02A8:2004 | 48 |
| 0510:0005 | 57 | 02A8:2005 | 57 |
| 0510:0006 | 41 | 02A8:2006 | 4E |

q SCASB (SCASW)

§ Compare byte in AL (or word in AX) with data at memory location ES:DI;
It updates DI depending status of DF and the length of the data being compare

q LODSB (LODSW)

§ Load byte (word) at memory location DS:SI to AL (AX);
It updates SI depending status of DF and the length of the data being transferred

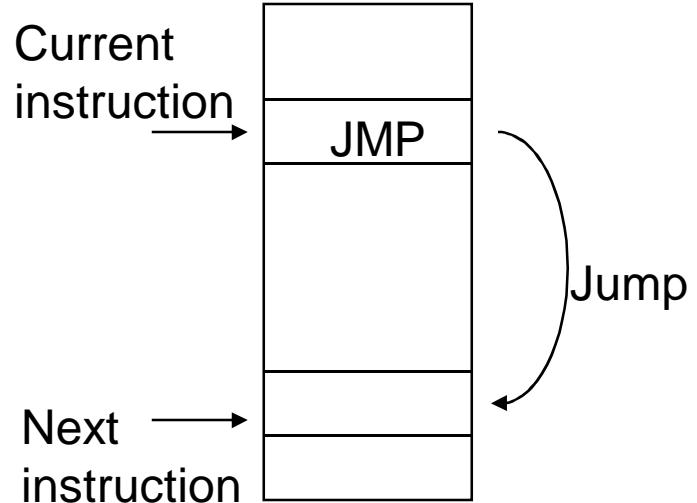
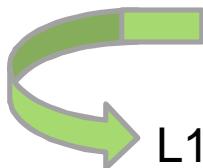
q STOSB (STOSW)

§ Store byte (word) at in AL (AX) to memory location ES:DI;
It updates DI depending status of DF and the length of the data being transferred

q JMP Label

- § Unconditionally Jump to specified Label or address location.
- § Label can be represented by a word or Alphabet with no.

```
MOV CX, 0007h  
MOV AX,F2FEh  
ADD AH,CL  
JMP L1  
SUB AH,CL  
L1: MOV [0200],AH  
INT21
```



Ø Conditional Jumps

q JZ: *Label_1*

§ If ZF =1, jump to the target address labeled by *Label_1*; else do not jump

q JNZ: *Label_1*

§ If ZF =0, jump to the target address labeled by *Label_1*; else do not jump

Ø Other Conditional Jumps

| | | | | | | |
|-----|-----|-----|------|------|------|-----|
| JNC | JAE | JNB | JC | JB | JNAE | JNG |
| JNE | JE | JNS | JS | JNO | JO | JNP |
| JPO | JP | JPE | JA | JBNE | JBE | JNA |
| JGE | JNL | JL | JNGE | JG | JNLE | JLE |

| Mnemonic | Meaning | Jump Condition |
|----------|------------------------|-------------------|
| JA | Jump if Above | CF = 0 and ZF = 0 |
| JAE | Jump if Above or Equal | CF = 0 |
| JB | Jump if Below | CF = 1 |
| JBE | Jump if Below or Equal | CF = 1 or ZF = 1 |
| JC | Jump if Carry | CF = 1 |
| JE | Jump if Equal | ZF = 1 |
| JNC | Jump if Not Carry | CF = 0 |
| JNE | Jump if Not Equal | ZF = 0 |
| JNZ | Jump if Not Zero | ZF = 0 |
| JPE | Jump if Parity Even | PF = 1 |
| JPO | Jump if Parity Odd | PF = 0 |
| JZ | Jump if Zero | ZF = 1 |

q `LOOP Label`

$CX = CX - 1$
If $CX \neq 0$ Then
JMP Label else
Next Instruction

q `LOOPE/LOOPZ Label`

$CX = CX - 1$
If $CX \neq 0 \& ZF=1$ Then
JMP Label else
Next Instruction

q `LOOPNE/LOOPNZ Label`

$CX = CX - 1$
If $CX \neq 0 \& ZF=0$ Then
JMP Label else
Next Instruction

- q CLC *Clear carry flag*
- q STC *Set carry flag*
- q CMC *Complement carry flag*
- q CLD *Clear direction flag*
- q STD *Set direction flag*
- q CLI *Clear interrupt-enable flag*
- q STI *Set interrupt-enable flag*
- q HLT *Halt microprocessor operation*
- q NOP *No operation*
- q LOCK *Lock Bus During Next Instruction*

- Ø A subroutine or procedure is a collection of instructions, written separately from main program, and can be called from a program.
- Ø Instruction used is *CALL Procedure-Name*
- Ø RET instruction lets the microprocessor to return from a subroutine to the called program.

Example

```
...  
MOV AL, 1  
CALL M1  
MOV BL, 3
```

...

```
M1 PROC  
    MOV CL, 2  
    RET  
M1 ENDP
```

The order of execution will be :

```
MOV AL, 1  
MOV CL, 2  
MOV BL, 3
```

- What Opcode & Operand?
- List various Instructions of 8086.
- Describe the Data transfer Instruction.
- Describe the Arithmetic Instruction.
- Describe the Data Bit manipulation Instruction.
- Explain Various Program control Instruction.
- Describe the Processor Control Instruction.
- What Adressing mode ?
- Explain various Adressing modes of 8086.

1. Addressing modes is define as the way in which data is addressed in the operand part of the instruction. It indicates the CPU finds from where to get data and where to store results.
2. 8086 has 8 Adressing modes a] Immediate addressing b] Register addressing c] Direct addressing d] Register Indirect addressing e] Relative Based f] Relative Indexed addressing g] Based indexed addressing h] Relative Based indexed with displacement addressing
3. 8086 Instructions cab be grouped as ,
 - 1] Data transfer instructions
 - 2] Arithmetic instructions
 - 3] String instructions
 - 4] Bit manipulation instructions
 - 5] Loop and jump instructions
 - 6] Subroutine and interrupt instructions
 - 7] Processor control instructions

CHAPTER-4 The Art of Assembly Language Programming



Topic 1: Program development steps



Topic 2: Assembly Language Programming Tools



Topic 3: Assembler directives and Operators

CHAPTER-4 SPECIFIC OBJECTIVE / COURSE OUTCOME

The student will be able to:



Know the program development steps



Use the different program development tools

A programme is nothing but set of Instructions written sequentially one below the other and stored in computers memory for execution by microprocessor.

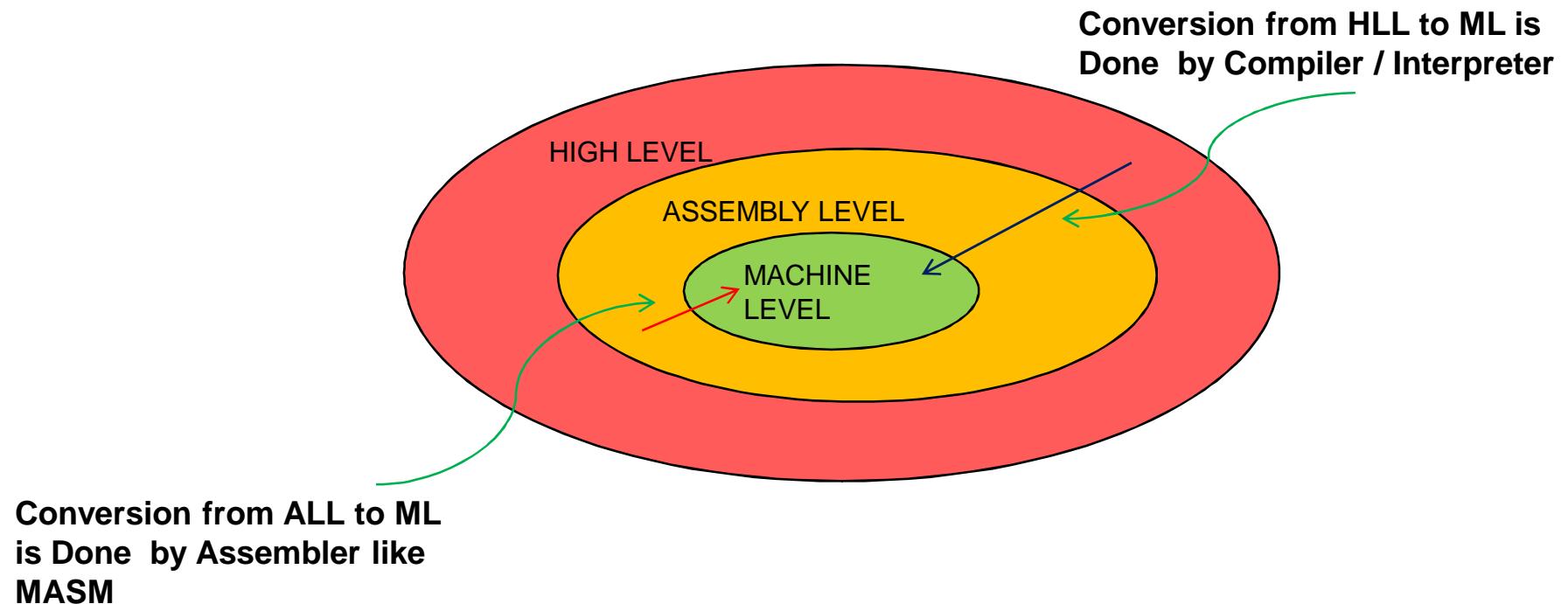
Instruction consists of a mnemonic and one or two operands (data).

Ø Machine Language: In this Programs is written in 0s and 1s.

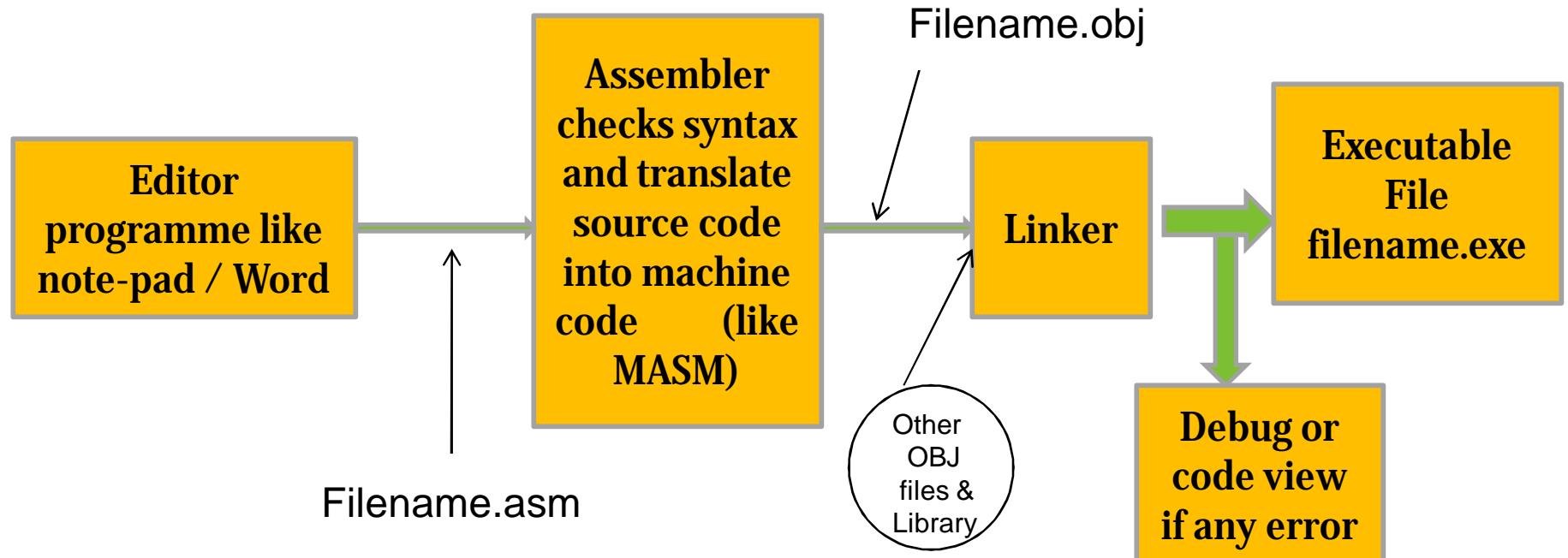
Ø Assembly Languages : It uses short form notations , called , ***mnemonics*** , to write a programme .The Mnemonics are like MOV , ADD , SUB, etc.

Ø High level languages: It uses English like sentences with proper syntax to write a programme.

Ø Assembler translates Assembly language program into machine code.



| Step & operation | Input | Software | Output |
|-------------------------------|------------------|--------------------|--------------|
| 1.Editing / writing programme | Note pad or Word | Note pad / MS-Word | Filename.asm |
| 2.Asemble | Filename.asm | MASM | Filename.obj |
| 3.Link | Filename.obj | LINK | Filename.exe |



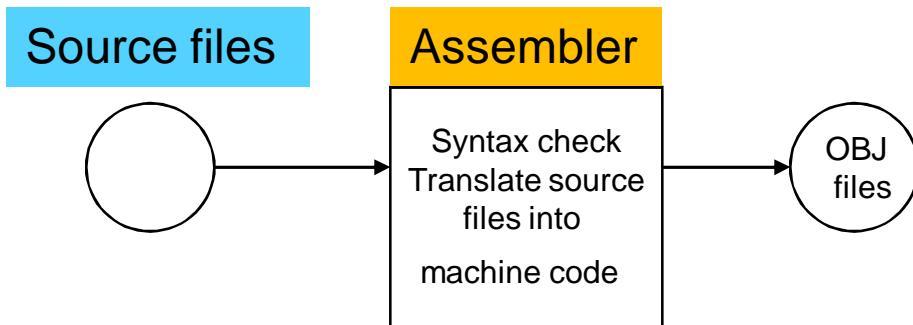
Editor

A text editor is required in order to create assembly language source files, where you'll be writing your code. You can use Notepad, DOS editor, or any other editor of your choice that produces plain ASCII text files.

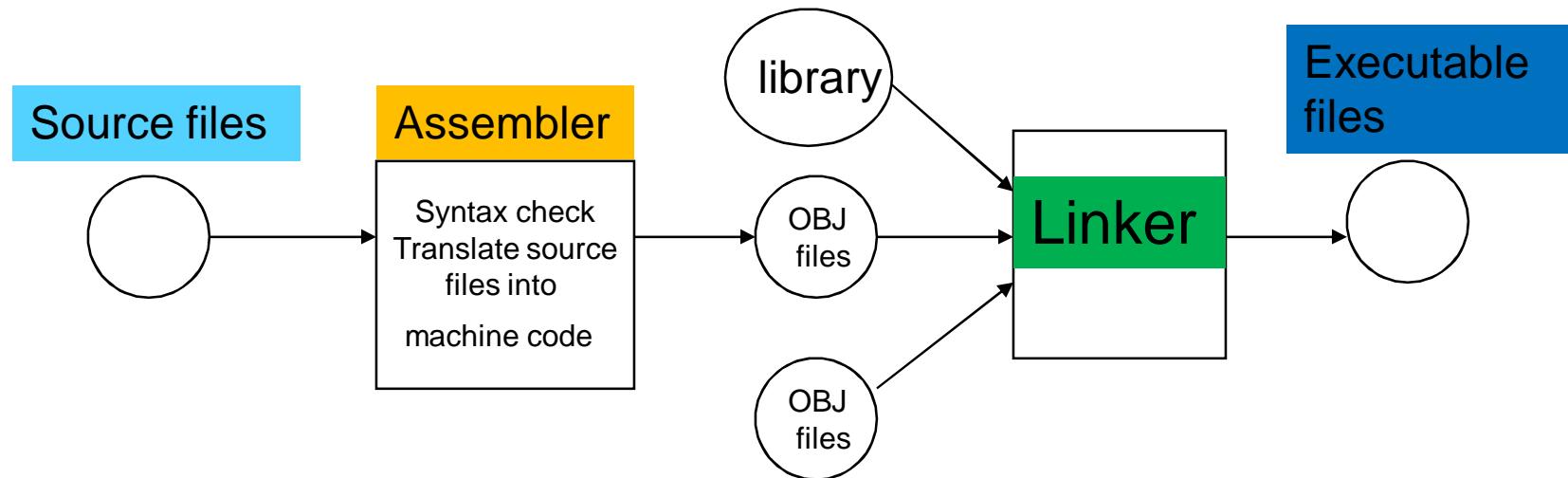
Debugger

A debugger program allows tracing of program execution and examination of registers and memory content.

For *16-bit programs*, MASM's debugger named *CodeView* can be used to debug .



An assembler is a program that converts source-code programs written in assembly language into object files in machine language. Popular assemblers include MASM (Macro Assembler from Microsoft), TASM (Turbo Assembler from Borland), NASM (Netwide Assembler for both Windows and Linux), and GNU assembler distributed by the free software foundation.



A linker program combines your program's object file created by the assembler with other object files and link libraries, producing a single executable program. You need a linker utility to produce executable files.

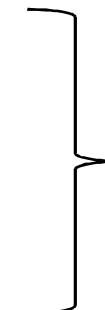
Two linkers: LINK.EXE and LINK32.EXE are provided with the MASM 6.15 distribution to link 16-bit real-address mode and 32-bit protected-address mode programs respectively.

It provides information to assist the assembler in producing executable code. It creates storage for a variable and initialize it.

Assembler directives (pseudo-instructions) give directions to the assembler about how it should translate the Assembly language instructions into machine code, Where as other instructions discussed in the above section give command to the 8086 microprocessor. Assembler directives are specific for a particular assembler. However all the popular assemblers like the Intel 8086 macro assembler, the turbo assembler and the IBM macro assembler use common assembler directives. The basic structure of a program in ASM will look like this

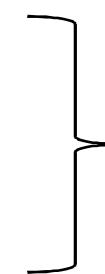
; Program for addition of two 8-bit nos. Comments / Remark

```
ASSUME CS: Code DS: Data  
Data SEGMENT  
    N1 DB 2FH  
    N2 DB 0EH  
    SUM DB 1 DUP(?)  
Data ENDS
```



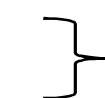
Assembler directives

```
Code SEGMENT  
    MOV AL,N1  
    MOV BL,N2  
    ADD AL,BL  
    MOV SUM,AL
```



Program

```
Code ENDS  
END
```



Assembler directives

1) The ASSUME directive tells the assembler the name of the logical segment it should use for a specified segment. It associates segment names with segment registers. For example

ASSUME CS: Code tells assembler that the instructions for programme are in the logical segment named Code.

Similarly for

ASSUME DS: Data tells assembler that for any program if instruction refers to the data segment then it should use the logical segment named Data. For example in the instruction MOV AX, [BX] the memory segment referred to by [BX] is in logical segment Data.

The 8086 contains a segment register (DS) that is dedicated to a data memory segment. This register is the default segment register used for all memory references used for data. The user is responsible loading the DS register with the appropriate value and telling the assembler where the DS register points so that it can calculate the offsets correctly. The standard is to define a segment to be a data segment. This is a convenient way of keeping data and code separate. The most common way of doing this is:

Data SEGMENT

...

Data ENDS ;indicates the end of the data segment

Hence the Assume directive is required to inform assembler of location of DS pointer by : ASSUME DS: Data;

2) Data storage directive

Each variable has a data type and is assigned a memory address by the program. Data directives are used to reserve and provide name for memory location in data segments. The symbols used for data types are:

| Data type | Symbol |
|-------------|--------|
| Byte | B |
| Word | W |
| Double word | D |
| Quad Word | Q |
| Ten Bytes | T |

For byte variable we should use
ü DB for declaration

N1 DB 4 ; initialise variable N1 with value 4 in decimal
N2 DB AFH ; initialise variable N2 with value AF in Hex
Name DB "JETHWA" ; allocates 6 byte with variable Name

| | |
|---|----|
| J | 4A |
| E | 45 |
| T | 54 |
| H | 48 |
| W | 57 |
| A | 41 |

DUP Operator is used to create arrays of elements whose initialize value is same.

The basic syntax is count DUP (initial value)

Example :

N1 DB 100 DUP(0) ; create 100 bytes arrays with value 0 in each

N2 DW 5 DUP (?) ; CREATE 5 arrays of uninitialized words

L1 DB 5, 4, 3 DUP(2,3 DUP(0),1) is same as

L1 DB 5,4,2,0,0,0,1,2,0,0,0,1,2,0,0,0,1

- What is Machine language?
- What is Assembly language?
- What is High level language?
- Describe the function of Linker.
- Describe the function of Assembler & debugger.
- What is Assemble directives?
- Explain various Assemble directives.

1. A programme is nothing but set of Instructions written sequentially one below the other and stored in computers memory for execution by microprocessor. Program can be written in 3 levels a) Machine Language b) Assembly Languages c) High level languages
2. **Assembler** translates Assembly language program into machine code.
3. **Compilers** like Pascal, Basic, C etc ***translate the HLL program into machine code.*** The programmer does not have to be concerned with internal details of the CPU.
4. A text editor is required in order to create assembly language source files, where you'll be writing your code. You can use Notepad, DOS editor, or any other editor of your choice that produces plain ASCII text files.
5. A debugger program allows tracing of program execution and examination of registers and memory content.
6. An assembler is a program that converts source-code programs written in assembly language into object files in machine language.
7. A linker program combines your program's **object file created by the assembler with other object files and link libraries, producing a single executable program. You need a linker utility to produce executable files.**
8. **Assembler directives** provides information to assist the assembler in producing executable code. It creates storage for a variable and initialize it. **Assembler directives** (pseudo-instructions) give directions to the assembler about how it should translate the Assembly language instructions into machine code

CHAPTER-5 8086 Assembly Language Programming.



Topic 1: Model of 8086 assembly language programs



Topic 2: Programming using assembler -

CHAPTER-5 SPECIFIC OBJECTIVE / COURSE OUTCOME

The student will be able to:



Write appropriate programs using editor



Run program using assembler & linker and Debug program using debugger

; Program for addition of two 8-bit nos. Comments / Remark

```
ASSUME CS: Code DS: Data
Data SEGMENT
    N1 DB 2FH
    N2 DB 0EH
    SUM DB 1 DUP(?)
Data ENDS
```

```
Code SEGMENT
    MOV AX, Data
    MOV DS,AX
    MOV AL,N1
    MOV BL,N2
    ADD AL,BL
    MOV SUM,AL
Code ENDS
END
```

; Program for addition of two 16-bit nos. Comments / Remark

```
ASSUME CS: Code DS: Data
Data SEGMENT
    N1 DW 002FH
    N2 DW 000EH
    SUM DW 1 DUP (?)
Data ENDS
```

```
Code SEGMENT
    MOV AX, Data
    MOV DS, AX
    MOV AX, N1
    MOV BX, N2
    ADD AX, BX
    MOV SUM, AX
Code ENDS
END
```

; Program for MULTIPLICATION of two 16-bit nos. Comments / Remark

```
ASSUME CS: Code DS: Data
Data SEGMENT
    N1 DW 002FH
    N2 DW 000EH
    RES DW 2 DUP (?)
Data ENDS

Code SEGMENT
    MOV AX, Data
    MOV DS, AX
    MOV AX, N1
    MOV BX, N2
    MUL BX
    MOV RES, AX
    MOV RES+2, BX
Code ENDS
END
```

; Program for DIVISION of 16-bit no. Comments / Remark

```
ASSUME CS: Code DS: Data
Data SEGMENT
    N1 DW 0F2FH
    N2 DB 0EH
    RESQ DB 2 DUP (?)
Data ENDS

Code SEGMENT
    MOV AX, Data
    MOV DS, AX
    MOV AX, N1
    MOV BL, N2
    DIV BL
    MOV RESQ, AL
    MOV RESQ+1, AH
Code ENDS
END
```

; Program for DIVISION of 32-bit no. Comments / Remark

ASSUME CS: Code DS: Data

Data SEGMENT

N1 DD FE000F2FH

N2 DW E40EH

RESQ DW 2 DUP (?)

Data ENDS

Code SEGMENT

MOV AX, Data

MOV DS,AX

MOV AX,N1

MOV DX,N1+2

MOV BX,N2

DIV BX

MOV RESQ,AX

MOV RESQ+2,DX

Code ENDS

END

- Write program to transfer a block of 50 bytes B1 to another block B2. The block B1 begins with offset address 1000h and block B2 from 2000h?
- Write program to exchange data of block of 10 bytes B1 to with another block B2. The block B1 begins with offset address 0200h and block B2 from 0300h?
- Write program to arrange a block of 50 bytes in ascending order. The block begins with offset address 1000h?
- Write program to arrange a block of 50 bytes in descending order. The block begins with offset address 1000h?

CHAPTER-6 Procedure and Macro in Assembly Language Program



Topic 1: Procedure



Topic 2: Defining Macros.

CHAPTER-6 SPECIFIC OBJECTIVE / COURSE OUTCOME

The student will be able to:



Understand the purpose of procedure and macros



Use procedure and macros

Procedure is a part of code that can be called from your program in order to make some specific task. Procedures make program more structural and easier to understand. Generally procedure returns to the same point from where it was called.

The syntax for procedure declaration:

name PROC

; here goes the code
; of the procedure ...

RET

name ENDP

name - is the procedure name, the same name should be in the top and the bottom, this is used to check correct closing of procedures.

RET instruction is used from procedure

PROC and ENDP are compiler directives, so they are not assembled into any real machine code. Compiler just remembers the address of procedure.

CALL instruction is used to call a procedure.

```
ORG 100h
MOV AL,2FH
MOV BL,F2H
CALL m1
MOV [SI] , AX
RET      ; return to operating system.
```

Procedure for multiplication

```
m1 PROC
MUL BL
RET      ; return to caller.
m1 ENDP
END
```

There are several ways to pass parameters to procedure, the easiest way to pass parameters is by using registers, here is another example of a procedure that receives two parameters in AL and BL registers, multiplies these parameters and returns the result in AX register:

```
ORG 100h
MOV AL, 1
MOV BL, 2
CALL m2
CALL m2
CALL m2
CALL m2
RET      ; return to operating
system.
```

In this example value of **AL** register is update every time the procedure is called, **BL** register stays unchanged, so this algorithm calculates 2 in power of 4,

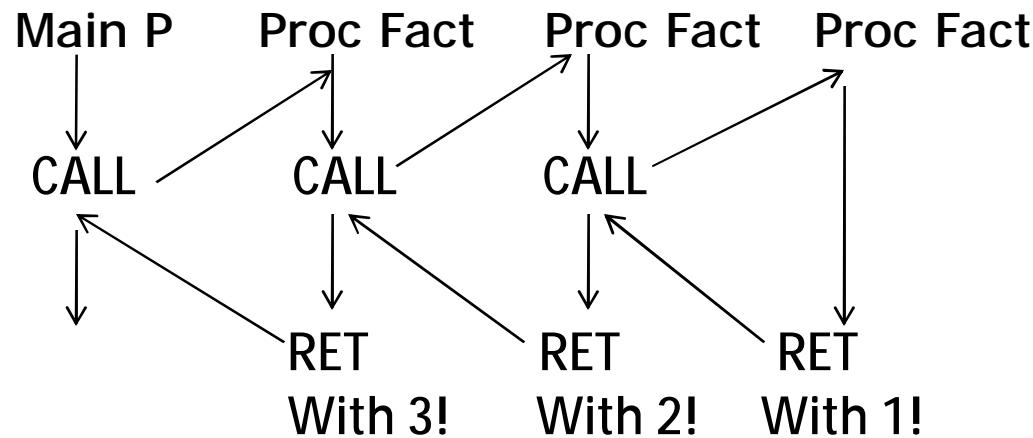
so final result in **AX** register is **16** (or **10h**).

```
m2 PROC
MUL BL      ; AX = AL * BL.
RET      ; return to caller.
m2 ENDP
END
```

A near CALL is a call to a procedure which is in the same code memory segment as that of CALL instruction in this the 8086 decrements Stack pointer by 2 and copies the IP on the STACK.

A far CALL is a call to a procedure which is in different code segment as that of CALL instruction. . In this the 8086 decrements Stack pointer by 2 and copies the CS first on the STACK and then again decrement SP by 2 to copy IP on the STACK.

Recursive Procedure: A recursive procedure is procedure which calls itself. It is used to work with complex data structures called trees. If the procedure is called with N (known as recursion depth) =3 then the n is decremented by 1 after each procedure CALL and the procedure is called until n=0 as shown in the diagram below:



LECTURE 2

To find out factorial of number

;Program to find out factorial of number Using Recursion

Data SEGMENT

 NUMBER DB 03H

 FACTORIAL DW 1DUP(?)

ENDS

Stack SEGMENT

 DW 128 DUP(0)

ENDS

Code SEGMENT

ASSUME CS:Code, DS:Data, SS:Stack

; INITIALISE SEGMENT REGISTERS:

 MOV AX, Data

 MOV DS, AX

 MOV AX, Stack

 MOV SS, AX

 MOV CX, NUMBER

 CALL FACT

 RET

```
;PROCEDURE FOR FACTORIAL PROGRAM
;CX CONTAINS INPUT NUMBER
;DX CONTAINS RESULT
FACT PROC NEAR
    CMP CX, 01H
    JNE CONT
    MOV DX,01H
    RET
CONT: PUSH CX      ; FOR BACKUP
    DEC CX
    CALL FACT
    POP AX      ; BACKUP OF CX IE N
    MUL DX      ; N*(N-1)!
    MOV DX, AX ; RESULT INTO DX
    RET
FACT ENDP
ENDS
```

Reentrant Procedure: A program or subroutine is called reentrant if it can be interrupted in the mid (i.e. the control flow is transferred outside of the subroutine, either due to an internal action such as a jump or call, or by an external action such as a hardware interrupt or signal), and then can then safely be called again before its previous invocation has been completed, and once the reentered invocation completes, the previous invocations should be able to resume execution correctly.

If procedure1 is called from main program and procedure2 is called from procedure1 and procedure1 again from procedure2 then such is called as reentrant procedure as shown below:

Macros

Macros are just like procedures, but not really. Macros look like procedures, but they exist only until your code is compiled, after compilation all macros are replaced with real instructions. For Macro assembler generates the code in program each time where the macro is “called”. If you declared a macro and never used it in your code, compiler will simply ignore it.

Macro definition:

```
name MACRO  
[parameters,...]  
<instructions>  
ENDM
```

```
MyMacro MACRO p1, p2, p3  
MOV AX, p1  
MOV BX, p2  
MOV CX, p3  
ENDM
```

```
ORG 100h  
MyMacro 1, 2, 3  
MyMacro 4, 5, DX  
RET
```

The code is expanded into:

```
MOV AX, 00001h  
MOV BX, 00002h  
MOV CX, 00003h  
MOV AX, 00004h  
MOV BX, 00005h  
MOV CX, DX
```

Procedure

Accessed by CALL & RET instruction

Machine code for instruction is put only once in the memory

With procedures less memory is required

Parameters can be passed in registers, memory locations or stack

Macro

Accessed during assembly with name given during program execution to macro when defined

Machine code is generated for instruction each time when macro is called.

With macro more memory is required

Parameters passed as part of statement which calls macro

Advantages of MACRO

- ü Program written with MACRO is more readable
- ü MACRO can be called by just writing its name along with its parameters;
hence no extra code is required like CALL & RET.
- ü Execution time is less as compared to Procedure
- ü Finding errors is easy

Disadvantages of MACRO

- ü Object code is generated every time Macro is called, hence object file becomes lengthy
- ü For large group of instruction macro is not preferred

- What is Procedures?
- What are the instructions to implement Procedures?
- What is Re-entrant Procedures?
- Describe the function MACROS.
- What are the differences between Procedures & MACROS.
- List various Advantages and disadvantages of MACROS.

1. Procedure is a part of code that can be called from your program in order to make some specific task. Procedures make program more structural and easier to understand. Generally procedure returns to the same point from where it was called. CALL instruction is used to call a procedure.
2. A near CALL is a call to a procedure which is in the same code memory segment as that of CALL instruction in this the 8086 decrements Stack pointer by 2 and copies the IP on the STACK.
3. A far CALL is a call to a procedure which is in different code segment as that of CALL instruction. . In this the 8086 decrements Stack pointer by 2 and copies the CS first on the STACK and then again decrement SP by 2 to copy IP on the STACK.
4. Reentrant Procedure: A program or subroutine is called reentrant if it can be interrupted in the middle (i.e. the control flow is transferred outside of the subroutine, either due to an internal action such as a jump or call, or by an external action such as a hardware interrupt or signal), can then safely be called again before its previous invocations have been completed, and once the reentered invocation completes, the previous invocations should be able to resume execution correctly.
5. Macros are just like procedures, but not really. Macros look like procedures, but they exist only until your code is compiled, after compilation all macros are replaced with real instructions. For Macro assembler generates the code in program each time where the macro is “called”. If you declared a macro and never used it in your code, compiler will simply ignore it.

Recommended Books:

1. Advanced Microprocessor and Peripherals (Architecture, Programming & Interfacing) by A.K. Roy & K.M. Bhurchandi, Tata Mcgraw Hill
2. Fundamentals of Microprocessors by B RAM, Dhanpat Rai Publications
3. Microprocessors by A.P.Godse, Technical Publications
4. 8085 Microprocessor: Programming And Interfacing 1st Edition
Author: N.K.Srinath, PHI Learning Private Limited
5. Microprocessor 8085 And Its Interfacing, Author A.P.Mathur , PHI Learning Pvt. Ltd.
6. The 8088 and 8086 Microprocessors: , Author: Walter A. Triebel, Avtar Singh,
7. Microprocessor 8085, 8086 , by Abhishek Yadav
8. Microprocessors: Theory and Applications : Intel and Motorola
by Mohamed Rafiquzzaman
9. "Microcomputer Systems: The 8086/88 Family", Liu, Gibson, 2nd Edition, PHI Learning Private Limited

References Books:

1. Microprocessor Architecture, Programming and Applications with the 8085 by Ramesh S. Gaonkar , Penram International Publishing (India)
2. Microprocessor & interfacing (programming & hardware) Revised Second Edition by Douglas V. Hall , Tata McGraw Hill
3. The Intel Microprocessors: 8086/8088, 80186/80188, 80286, 80386 Barry Bray , Pearson Education; Eighth edition (2011)
4. The 8086/8088 Family: Design, Programming And Interfacing 1st Edition, John Uffenbeck, Prentice-Hall
5. Microcomputer Systems the 8086/8088 Family : Architecture, Programming and Design 2nd Edition Author: Gleen A.Gibson , Prentice-Hall
6. 8085 Microprocessor: Programming And Interfacing 1st Edition Author: N.K.Srinath, PHI Learning Private Limited
7. The 8086 Microprocessor :Programming & Interfacing the PC with CD Kenneth Ayala, Publisher: Cengage Learning
8. Assembly programming and the 8086 microprocessor, Douglas Samuel Jones ,Oxford University Press

References Web:

1. www.intel.com
2. www.pcguide.com/ref/CPU
3. www.CPU-World.com /Arch /
4. [www.techsource .com](http://www.techsource.com) / Engineering parts/ microprocessor.html
5. www.slideshare.net
6. www.powershow.com
7. www.authorstream.com
8. www.youtube.com
9. www.scribd.com
10. www.eazynotes.com
11. www.electronicstutorialsblog.com
12. ece.uprm.edu

8085 MICROPROCESSOR PROGRAMS

ADDITION OF TWO 8 BIT NUMBERS

AIM:

To perform addition of two 8 bit numbers using 8085.

ALGORITHM:

- 1) Start the program by loading the first data into Accumulator.
- 2) Move the data to a register (B register).
- 3) Get the second data and load into Accumulator.
- 4) Add the two register contents.
- 5) Check for carry.
- 6) Store the value of sum and carry in memory location.
- 7) Terminate the program.

PROGRAM:

| | | | |
|-------|-----|-------|--|
| | MVI | C, 00 | Initialize C register to 00 |
| | LDA | 4150 | Load the value to Accumulator. |
| | MOV | B, A | Move the content of Accumulator to B register. |
| | LDA | 4151 | Load the value to Accumulator. |
| | ADD | B | Add the value of register B to A |
| | JNC | LOOP | Jump on no carry. |
| | INR | C | Increment value of register C |
| LOOP: | STA | 4152 | Store the value of Accumulator (SUM). |
| | MOV | A, C | Move content of register C to Acc. |
| | STA | 4153 | Store the value of Accumulator (CARRY) |
| | HLT | | Halt the program. |

OBSERVATION:

| | |
|---------|-----------|
| Input: | 80 (4150) |
| | 80 (4251) |
| Output: | 00 (4152) |
| | 01 (4153) |

RESULT:

Thus the program to add two 8-bit numbers was executed.

SUBTRACTION OF TWO 8 BIT NUMBERS

AIM:

To perform the subtraction of two 8 bit numbers using 8085.

ALGORITHM:

1. Start the program by loading the first data into Accumulator.
2. Move the data to a register (B register).
3. Get the second data and load into Accumulator.
4. Subtract the two register contents.
5. Check for carry.
6. If carry is present take 2's complement of Accumulator.
7. Store the value of borrow in memory location.
8. Store the difference value (present in Accumulator) to a memory location and terminate the program.

PROGRAM:

| | | |
|-------|----------|--|
| MVI | C, 00 | Initialize C to 00 |
| LDA | 4150 | Load the value to Acc. |
| MOV | B, A | Move the content of Acc to B register. |
| LDA | 4151 | Load the value to Acc. |
| SUB | B | |
| JNC | LOOP | Jump on no carry. |
| CMA | | Complement Accumulator contents. |
| INR | A | Increment value in Accumulator. |
| INR | C | Increment value in register C |
| LOOP: | STA 4152 | Store the value of A-reg to memory address. |
| | MOV A, C | Move contents of register C to Accumulator. |
| | STA 4153 | Store the value of Accumulator memory address. |
| | HLT | Terminate the program. |

OBSERVATION:

Input: 06 (4150)
02 (4251)
Output: 04 (4152)
01 (4153)

RESULT:

Thus the program to subtract two 8-bit numbers was executed.

MULTIPLICATION OF TWO 8 BIT NUMBERS

AIM:

To perform the multiplication of two 8 bit numbers using 8085.

ALGORITHM:

- 1) Start the program by loading HL register pair with address of memory location.
- 2) Move the data to a register (B register).
- 3) Get the second data and load into Accumulator.
- 4) Add the two register contents.
- 5) Check for carry.
- 6) Increment the value of carry.
- 7) Check whether repeated addition is over and store the value of product and carry in memory location.
- 8) Terminate the program.

PROGRAM:

| | | | |
|-------|-----|---------|---------------------------------------|
| | MVI | D, 00 | Initialize register D to 00 |
| | MVI | A, 00 | Initialize Accumulator content to 00 |
| | LXI | H, 4150 | |
| | MOV | B, M | Get the first number in B - reg |
| | INX | H | |
| | MOV | C, M | Get the second number in C- reg. |
| LOOP: | ADD | B | Add content of A - reg to register B. |
| | JNC | NEXT | Jump on no carry to NEXT. |
| | INR | D | Increment content of register D |
| NEXT: | DCR | C | Decrement content of register C. |
| | JNZ | LOOP | Jump on no zero to address |
| | STA | 4152 | Store the result in Memory |
| | MOV | A, D | |
| | STA | 4153 | Store the MSB of result in Memory |
| | HLT | | Terminate the program. |

OBSERVATION:

Input: FF (4150)
FF (4151)
Output: 01 (4152)
FE (4153)

RESULT:

Thus the program to multiply two 8-bit numbers was executed.

DIVISION OF TWO 8 BIT NUMBERS

AIM:

To perform the division of two 8 bit numbers using 8085.

ALGORITHM:

- 1) Start the program by loading HL register pair with address of memory location.
- 2) Move the data to a register(B register).
- 3) Get the second data and load into Accumulator.
- 4) Compare the two numbers to check for carry.
- 5) Subtract the two numbers.
- 6) Increment the value of carry .
- 7) Check whether repeated subtraction is over and store the value of product and carry in memory location.
- 8) Terminate the program.

PROGRAM:

| | | | |
|-------|-----|---------|----------------------------------|
| | LXI | H, 4150 | |
| | MOV | B, M | Get the dividend in B – reg. |
| | MVI | C, 00 | Clear C – reg for quotient |
| | INX | H | |
| | MOV | A, M | Get the divisor in A – reg. |
| NEXT: | CMP | B | Compare A - reg with register B. |
| | JC | LOOP | Jump on carry to LOOP |
| | SUB | B | Subtract A – reg from B- reg. |
| | INR | C | Increment content of register C. |
| | JMP | NEXT | Jump to NEXT |
| LOOP: | STA | 4152 | Store the remainder in Memory |
| | MOV | A, C | |
| | STA | 4153 | Store the quotient in memory |
| | HLT | | Terminate the program. |

OBSERVATION:

Input: FF (4150)
FF (4251)

Output: 01 (4152) ---- Remainder
FE (4153) ---- Quotient

RESULT:

Thus the program to divide two 8-bit numbers was executed.

LARGEST NUMBER IN AN ARRAY OF DATA

AIM:

To find the largest number in an array of data using 8085 instruction set.

ALGORITHM:

- 1) Load the address of the first element of the array in HL pair
- 2) Move the count to B – reg.
- 3) Increment the pointer
- 4) Get the first data in A – reg.
- 5) Decrement the count.
- 6) Increment the pointer
- 7) Compare the content of memory addressed by HL pair with that of A - reg.
- 8) If Carry = 0, go to step 10 or if Carry = 1 go to step 9
- 9) Move the content of memory addressed by HL to A – reg.
- 10) Decrement the count
- 11) Check for Zero of the count. If ZF = 0, go to step 6, or if ZF = 1 go to next step.
- 12) Store the largest data in memory.
- 13) Terminate the program.

PROGRAM:

| | | | |
|--------|-----|--------|---|
| | LXI | H,4200 | Set pointer for array |
| | MOV | B,M | Load the Count |
| | INX | H | |
| | MOV | A,M | Set 1 st element as largest data |
| | DCR | B | Decrement the count |
| LOOP: | INX | H | |
| | CMP | M | If A- reg > M go to AHEAD |
| | JNC | AHEAD | |
| | MOV | A,M | Set the new value as largest |
| AHEAD: | DCR | B | |
| | JNZ | LOOP | Repeat comparisons till count = 0 |
| | STA | 4300 | Store the largest value at 4300 |
| | HLT | | |

OBSERVATION:

Input: 05 (4200) ----- Array Size
 0A (4201)
 F1 (4202)
 1F (4203)
 26 (4204)
 FE (4205)

Output: FE (4300)

RESULT:

Thus the program to find the largest number in an array of data was executed

SMALLEST NUMBER IN AN ARRAY OF DATA

AIM:

To find the smallest number in an array of data using 8085 instruction set.

ALGORITHM:

- 1) Load the address of the first element of the array in HL pair
- 2) Move the count to B – reg.
- 3) Increment the pointer
- 4) Get the first data in A – reg.
- 5) Decrement the count.
- 6) Increment the pointer
- 7) Compare the content of memory addressed by HL pair with that of A - reg.
- 8) If carry = 1, go to step 10 or if Carry = 0 go to step 9
- 9) Move the content of memory addressed by HL to A – reg.
- 10) Decrement the count
- 11) Check for Zero of the count. If ZF = 0, go to step 6, or if ZF = 1 go to next step.
- 12) Store the smallest data in memory.
- 13) Terminate the program.

PROGRAM:

| | | | |
|--------|-----|--------|---|
| | LXI | H,4200 | Set pointer for array |
| | MOV | B,M | Load the Count |
| | INX | H | |
| | MOV | A,M | Set 1 st element as largest data |
| | DCR | B | Decrement the count |
| LOOP: | INX | H | |
| | CMP | M | If A- reg < M go to AHEAD |
| | JC | AHEAD | |
| | MOV | A,M | Set the new value as smallest |
| AHEAD: | DCR | B | |
| | JNZ | LOOP | Repeat comparisons till count = 0 |
| | STA | 4300 | Store the largest value at 4300 |
| | HLT | | |

OBSERVATION:

Input: 05 (4200) ----- Array Size
 0A (4201)
 F1 (4202)
 1F (4203)
 26 (4204)
 FE (4205)

Output: 0A (4300)

RESULT:

Thus the program to find the smallest number in an array of data was executed

ARRANGE AN ARRAY OF DATA IN ASCENDING ORDER

AIM:

To write a program to arrange an array of data in ascending order

ALGORITHM:

1. Initialize HL pair as memory pointer
2. Get the count at 4200 into C – register
3. Copy it in D – register (for bubble sort (N-1) times required)
4. Get the first value in A – register
5. Compare it with the value at next location.
6. If they are out of order, exchange the contents of A –register and Memory
7. Decrement D –register content by 1
8. Repeat steps 5 and 7 till the value in D- register become zero
9. Decrement C –register content by 1
10. Repeat steps 3 to 9 till the value in C – register becomes zero

PROGRAM:

| | | |
|---------|-----|--------|
| | LXI | H,4200 |
| | MOV | C,M |
| | DCR | C |
| REPEAT: | MOV | D,C |
| | LXI | H,4201 |
| LOOP: | MOV | A,M |
| | INX | H |
| | CMP | M |
| | JC | SKIP |
| | MOV | B,M |
| | MOV | M,A |
| | DCX | H |
| | MOV | M,B |
| | INX | H |
| SKIP: | DCR | D |
| | JNZ | LOOP |
| | DCR | C |
| | JNZ | REPEAT |
| | HLT | |

OBSERVATION:

Input: 4200 05 (Array Size)
 4201 05
 4202 04
 4203 03
 4204 02
 4205 01

Output: 4200 05(Array Size)
 4201 01
 4202 02
 4203 03
 4204 04
 4205 05

RESULT:

Thus the given array of data was arranged in ascending order.

ARRANGE AN ARRAY OF DATA IN DESCENDING ORDER

AIM:

To write a program to arrange an array of data in descending order

ALGORITHM:

1. Initialize HL pair as memory pointer
2. Get the count at 4200 into C – register
3. Copy it in D – register (for bubble sort (N-1) times required)
4. Get the first value in A – register
5. Compare it with the value at next location.
6. If they are out of order, exchange the contents of A –register and Memory
7. Decrement D –register content by 1
8. Repeat steps 5 and 7 till the value in D- register become zero
9. Decrement C –register content by 1
10. Repeat steps 3 to 9 till the value in C – register becomes zero

PROGRAM:

| | | |
|---------|-----|--------|
| | LXI | H,4200 |
| | MOV | C,M |
| | DCR | C |
| REPEAT: | MOV | D,C |
| | LXI | H,4201 |
| LOOP: | MOV | A,M |
| | INX | H |
| | CMP | M |
| | JNC | SKIP |
| | MOV | B,M |
| | MOV | M,A |
| | DCX | H |
| | MOV | M,B |
| | INX | H |
| SKIP: | DCR | D |
| | JNZ | LOOP |
| | DCR | C |
| | JNZ | REPEAT |
| | HLT | |

OBSERVATION:

Input: 4200 05 (Array Size)
 4201 01
 4202 02
 4203 03
 4204 04
 4205 05

Output: 4200 05(Array Size)
 4201 05
 4202 04
 4203 03
 4204 02
 4205 01

RESULT:

Thus the given array of data was arranged in descending order.

BCD TO HEX CONVERSION

AIM:

To convert two BCD numbers in memory to the equivalent HEX number using 8085 instruction set

ALGORITHM:

- 1) Initialize memory pointer to 4150 H
- 2) Get the Most Significant Digit (MSD)
- 3) Multiply the MSD by ten using repeated addition
- 4) Add the Least Significant Digit (LSD) to the result obtained in previous step
- 5) Store the HEX data in Memory

PROGRAM:

| | | |
|-----|--------|---------------------------|
| LXI | H,4150 | |
| MOV | A,M | Initialize memory pointer |
| ADD | A | MSD X 2 |
| MOV | B,A | Store MSD X 2 |
| ADD | A | MSD X 4 |
| ADD | A | MSD X 8 |
| ADD | B | MSD X 10 |
| INX | H | Point to LSD |
| ADD | M | Add to form HEX |
| INX | H | |
| MOV | M,A | Store the result |
| HLT | | |

OBSERVATION:

Input: 4150 : 02 (MSD)
 4151 : 09 (LSD)

Output: 4152 : 1D H

RESULT:

Thus the program to convert BCD data to HEX data was executed.

HEX TO BCD CONVERSION

AIM:

To convert given Hexa decimal number into its equivalent BCD number using 8085 instruction set

ALGORITHM:

- 1) Initialize memory pointer to 4150 H
- 2) Get the Hexa decimal number in C - register
- 3) Perform repeated addition for C number of times
- 4) Adjust for BCD in each step
- 5) Store the BCD data in Memory

PROGRAM:

| | | | |
|--------|-----|--------|--|
| | LXI | H,4150 | Initialize memory pointer |
| | MVI | D,00 | Clear D- reg for Most significant Byte |
| | XRA | A | Clear Accumulator |
| | MOV | C,M | Get HEX data |
| LOOP2: | ADI | 01 | Count the number one by one |
| | DAA | | Adjust for BCD count |
| | JNC | LOOP1 | |
| | INR | D | |
| LOOP1: | DCR | C | |
| | JNZ | LOOP2 | |
| | STA | 4151 | Store the Least Significant Byte |
| | MOV | A,D | |
| | STA | 4152 | Store the Most Significant Byte |
| | HLT | | |

OBSERVATION:

Input: 4150 : FF

Output: 4151 : 55 (LSB)
4152 : 02 (MSB)

RESULT:

Thus the program to convert HEX data to BCD data was executed.

HEX TO ASCII CONVERSION

AIM:

To convert given Hexa decimal number into its equivalent ASCII number using 8085 instruction set.

ALGORITHM:

1. Load the given data in A- register and move to B – register
2. Mask the upper nibble of the Hexa decimal number in A – register
3. Call subroutine to get ASCII of lower nibble
4. Store it in memory
5. Move B –register to A – register and mask the lower nibble
6. Rotate the upper nibble to lower nibble position
7. Call subroutine to get ASCII of upper nibble
8. Store it in memory
9. Terminate the program.

PROGRAM:

| | | |
|-------|------|---------------------------------|
| LDA | 4200 | Get Hexa Data |
| MOV | B,A | |
| ANI | 0F | Mask Upper Nibble |
| CALL | SUB1 | Get ASCII code for upper nibble |
| STA | 4201 | |
| MOV | A,B | |
| ANI | F0 | Mask Lower Nibble |
| RLC | | |
| CALL | SUB1 | Get ASCII code for lower nibble |
| STA | 4202 | |
| HLT | | |
| | | |
| SUB1: | CPI | 0A |
| | JC | SKIP |
| | ADI | 07 |
| SKIP: | ADI | 30 |
| | RET | |

OBSERVATION:

Input: 4200 E4(Hexa data)

Output: 4201 34(ASCII Code for 4)
 4202 45(ASCII Code for E)

RESULT:

Thus the given Hexa decimal number was converted into its equivalent ASCII Code.

ASCII TO HEX CONVERSION

AIM:

To convert given ASCII Character into its equivalent Hexa Decimal number using 8085 instruction set.

ALGORITHM:

1. Load the given data in A- register
2. Subtract 30 H from A – register
3. Compare the content of A – register with 0A H
4. If A < 0A H, jump to step6. Else proceed to next step.
5. Subtract 07 H from A – register
6. Store the result
7. Terminate the program

PROGRAM:

| | |
|-------|----------|
| LDA | 4500 |
| SUI | 30 |
| CPI | 0A |
| JC | SKIP |
| SUI | 07 |
| SKIP: | STA 4501 |
| | HLT |

OBSERVATION:

Input: 4500 31

Output: 4501 0B

RESULT:

Thus the given ASCII character was converted into its equivalent Hexa Value.

SQUARE OF A NUMBER USING LOOK UP TABLE

AIM:

To find the square of the number from 0 to 9 using a Table of Square.

ALGORITHM:

1. Initialize HL pair to point Look up table
2. Get the data .
3. Check whether the given input is less than 9.
4. If yes go to next step else halt the program
5. Add the desired address with the accumulator content
6. Store the result

PROGRAM:

| | | |
|--------|----------|--------------------------------------|
| LXI | H,4125 | Initialsie Look up table address |
| LDA | 4150 | Get the data |
| CPI | 0A | Check input > 9 |
| JC | AFTER | if yes error |
| MVI | A,FF | Error Indication |
| STA | 4151 | |
| | HLT | |
| AFTER: | | MOV C,A Add the desired Address |
| | MVI B,00 | |
| | DAD B | |
| | MOV A,M | |
| | STA 4151 | Store the result |
| | HLT | Terminate the program |

LOOKUP TABLE:

| | |
|------|----|
| 4125 | 01 |
| 4126 | 04 |
| 4127 | 09 |
| 4128 | 16 |
| 4129 | 25 |
| 4130 | 36 |
| 4131 | 49 |
| 4132 | 64 |
| 4133 | 81 |

OBSERVATION:

Input: 4150: 05

Output: 4151 25 (Square)

Input: 4150: 11

Output: 4151: FF (Error Indication)

RESULT:

Thus the program to find the square of the number from 0 to 9 using a Look up table was executed.

INTERFACING WITH 8085

INTERFACING 8251 (USART) WITH 8085 PROCESSOR

AIM:

To write a program to initiate 8251 and to check the transmission and reception of character

THEORY:

The 8251 is used as a peripheral device for serial communication and is programmed by the CPU to operate using virtually any serial data transmission technique. The USART accepts data characters from the CPU in parallel format and then converts them into a continuous serial data stream for transmission. Simultaneously, it can receive serial data streams and convert them into parallel data characters for the CPU. The CPU can read the status of USART at any time. These include data transmission errors and control signals.

Prior to starting data transmission or reception, the 8251 must be loaded with a set of control words generated by the CPU. These control signals define the complete functional definition of the 8251 and must immediately follow a RESET operation. Control words should be written into the control register of 8251. These control words are split into two formats:

1. MODE INSTRUCTION WORD
2. COMMAND INSTRUCTION WORD

1. MODE INSTRUCTION WORD

This format defines the Baud rate, Character length, Parity and Stop bits required to work with asynchronous data communication. By selecting the appropriate baud factor sync mode, the 8251 can be operated in Synchronous mode.

Initializing 8251 using the mode instruction to the following conditions

- 8 Bit data
- No Parity
- Baud rate Factor (16X)
- 1 Stop Bit

gives a mode command word of 01001110 = 4E (HEX)

MODE INSTRUCTION - SYNCHRONOUS MODE

| | | | | | | | |
|----|----|----|-----|----|----|----|----|
| S2 | S1 | EP | PEN | L2 | L1 | B2 | B1 |
|----|----|----|-----|----|----|----|----|

| BAUD RATE FACTOR | | | | |
|------------------|------|---|-------|-------|
| 0 | 1 | 0 | 1 | |
| 0 | | 0 | 1 | 1 |
| SYNC MODE | (1X) | | (16X) | (64X) |

| CHARACTR LENGTH | | | | |
|-----------------|--------|--------|--------|--|
| 0 | 1 | 0 | 1 | |
| 0 | 0 | 1 | 1 | |
| 5 BITS | 6 BITS | 7 BITS | 8 BITS | |

| PARITY ENABLE | | | | |
|---------------|-------------|--|--|--|
| 1= ENABLE | 0 = DISABLE | | | |

| EVEN PARITY GEN/CHECK | | | | |
|-----------------------|----------|--|--|--|
| 0 = ODD | 1 = EVEN | | | |

| NUMBER OF STOP BITS | | | | |
|---------------------|-------|---------|-------|--|
| 0 | 1 | 0 | 1 | |
| 0 | 0 | 1 | 1 | |
| INVALID | 1 BIT | 1.5 BIT | 2 BIT | |

MODE INSTRUCTION - ASYNCHRONOUS MODE

| | | | | | | | |
|----|----|----|-----|----|----|----|----|
| S2 | S1 | EP | PEN | L2 | L1 | B2 | B1 |
|----|----|----|-----|----|----|----|----|

| CHARACTER LENGTH | | | |
|------------------|--------|--------|--------|
| 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 5 BITS | 6 BITS | 7 BITS | 8 BITS |

| PARITY ENABLE | |
|---------------|-------------|
| 1= ENABLE | 0 = DISABLE |

| EVEN PARITY GEN/CHECK | |
|-----------------------|----------|
| 0 = ODD | 1 = EVEN |

| EXTERNAL SYNC DETECTS | |
|--------------------------|--|
| 1 = SYSDET IS AN INPUT | |
| 0 = SYSDET IS AN IOUTPUT | |

| SINGLE CHARACTER SYNC | |
|---------------------------|--|
| 1 = SINGLE SYNC CHARACTER | |
| 0 = DOUBLE SYNC CHARACTER | |

2. COMMAND INSTRUCTION WORD

This format defines a status word that is used to control the actual operation of 8251. All control words written into 8251 after the mode instruction will load the command instruction.

The command instructions can be written into 8251 at any time in the data block during the operation of the 8251. To return to the mode instruction format, the master reset bit in the command instruction word can be set to initiate an internal reset operation which automatically places the 8251 back into the mode instruction format. Command instructions must follow the mode instructions or sync characters.

Thus the control word 37 (HEX) enables the transmit enable and receive enable bits, forces DTR output to zero, resets the error flags, and forces RTS output to zero.

| | | | | | | | |
|----|----|-----|----|------|-----|-----|------|
| EH | IR | RTS | ER | SBRK | RXE | DTR | TXEN |
|----|----|-----|----|------|-----|-----|------|

| |
|-----------------|
| TRANSMIT ENABLE |
|-----------------|

1=Enable 0 = Disable

| |
|---------------------|
| DATA TERMINAL READY |
|---------------------|

HIGH will force DTR
Output to Zero

| |
|----------------|
| RECEIVE ENABLE |
|----------------|

1=Enable 0 = Disable

| |
|----------------------|
| SEND BREAK CHARACTER |
|----------------------|

1 = Forces TXD LOW
0 = Normal Operation

| |
|-------------|
| ERROR RESET |
|-------------|

1=Reset Error Flags
PE,OE,FE

| |
|-----------------|
| REQUEST TO SEND |
|-----------------|

HIGH will force RTS
Output to Zero

| |
|----------------|
| INTERNAL RESET |
|----------------|

HIGH Returns 8251 to
Mode Instruction Format

| |
|-----------------|
| ENTER HUNT MODE |
|-----------------|

1= Enable a Search for
Sync Characters(Has
No Effect in Async mode)

COMMAND INSTRUCTION FORMAT

ALGORITHM:

1. Initialise timer (8253) IC
2. Move the mode command word (4E H) to A -reg
3. Output it to port address C2
4. Move the command instruction word (37 H) to A -reg
5. Output it to port address C2
6. Move the the data to be transferred to A -reg
7. Output it to port address C0
8. Reset the system
9. Get the data through input port address C0
10. Store the value in memory
11. Reset the system

PROGRAM:

| | |
|-----|--------|
| MVI | A,36H |
| OUT | CEH |
| MVI | A,0AH |
| OUT | C8H |
| MVI | A,00 |
| OUT | C8H |
| LXI | H,4200 |
| MVI | A,4E |
| OUT | C2 |
| MVI | A,37 |
| OUT | C2 |
| MVI | A,41 |
| OUT | C0 |
| RST | 1 |
| ORG | 4200 |
| IN | C0 |
| STA | 4500 |
| RST | 1 |

OBSERVATION:

Output: 4500 41

RESULT:

Thus the 8251 was initiated and the transmission and reception of character was done successfully.

INTERFACING ADC WITH 8085 PROCESSOR

AIM:

To write a program to initiate ADC and to store the digital data in memory

PROGRAM:

| | | |
|-------|-----|------|
| | MVI | A,10 |
| | OUT | C8 |
| | MVI | A,18 |
| | OUT | C8 |
| | MVI | A,10 |
| | OUT | D0 |
| | XRA | A |
| | XRA | A |
| | XRA | A |
| | MVI | A,00 |
| | OUT | D0 |
| LOOP: | IN | D8 |
| | ANI | 01 |
| | CPI | 01 |
| | JNZ | LOOP |
| | IN | C0 |
| | STA | 4150 |
| | HLT | |

OBSERVATION:

Compare the data displayed at the LEDs with that stored at location 4150

RESULT:

Thus the ADC was initiated and the digital data was stored at desired location

INTERFACING DAC WITH 8085

AIM:

To interface DAC with 8085 to demonstrate the generation of square, saw tooth and triangular wave.

APPARATUS REQUIRED:

- 8085 Trainer Kit
- DAC Interface Board

THEORY:

DAC 0800 is an 8 – bit DAC and the output voltage variation is between – 5V and + 5V. The output voltage varies in steps of $10/256 = 0.04$ (appx.). The digital data input and the corresponding output voltages are presented in the Table1.

| Input Data in HEX | Output Voltage |
|-------------------|----------------|
| 00 | - 5.00 |
| 01 | - 4.96 |
| 02 | - 4.92 |
| ... | ... |
| 7F | 0.00 |
| ... | ... |
| FD | 4.92 |
| FE | 4.96 |
| FF | 5.00 |

Referring to Table1, with 00 H as input to DAC, the analog output is – 5V. Similarly, with FF H as input, the output is +5V. Outputting digital data 00 and FF at regular intervals, to DAC, results in different wave forms namely square, triangular, etc,. The port address of DAC is 08 H.

ALGORITHM:

(a) Square Wave Generation

1. Load the initial value (00) to Accumulator and move it to DAC
2. Call the delay program
3. Load the final value(FF) to accumulator and move it to DAC
4. Call the delay program.
5. Repeat Steps 2 to 5

(b) Saw tooth Wave Generation

1. Load the initial value (00) to Accumulator
2. Move the accumulator content to DAC
3. Increment the accumulator content by 1.
4. Repeat Steps 3 and 4.

(c) Triangular Wave Generation

2. Load the initial value (00) to Accumulator
3. Move the accumulator content to DAC
4. Increment the accumulator content by 1.
5. If accumulator content is zero proceed to next step. Else go to step 3.
6. Load value (FF) to Accumulator
7. Move the accumulator content to DAC
8. Decrement the accumulator content by 1.
9. If accumulator content is zero go to step2. Else go to step 7.

PROGRAM:

(a) Square Wave Generation

| | | |
|--------|------|---------------------|
| START: | MVI | A,00 |
| | OUT | Port address of DAC |
| | CALL | DELAY |
| | MVI | A,FF |
| | OUT | Port address of DAC |
| | CALL | DELAY |
| | JMP | START |
| DELAY: | MVI | B,05 |
| L1: | MVI | C,FF |
| L2: | DCR | C |
| | JNZ | L2 |
| | DCR | B |
| | JNZ | L1 |
| | RET | |

(b) Saw tooth Wave Generation

| | | |
|--------|-----|---------------------|
| START: | MVI | A,00 |
| L1: | OUT | Port address of DAC |
| | INR | A |
| | JNZ | L1 |
| | JMP | START |

(c) Triangular Wave Generation

| | | |
|--------|-----|---------------------|
| START: | MVI | L,00 |
| L1: | MOV | A,L |
| | OUT | Port address of DAC |
| | INR | L |
| | JNZ | L1 |
| | MVI | L,FF |
| L2: | MOV | A,L |
| | OUT | Port address of DAC |
| | DCR | L |
| | JNZ | L2 |
| | JMP | START |

RESULT:

Thus the square, triangular and saw tooth wave form were generated by interfacing DAC with 8085 trainer kit.

INTERFACING 8253 (TIMER IC) WITH 8085 PROCESSOR

AIM:

To interface 8253 Programmable Interval Timer to 8085 and verify the operation of 8253 in six different modes.

APPARATUS REQUIRED:

- 1) 8085 Microprocessor toolkit.
- 2) 8253 Interface board.
- 3) VXT parallel bus.
- 4) Regulated D.C power supply.
- 5) CRO.

MODE 0-Interrupt On Terminal Count:-

The output will be initially low after mode set operation. After loading the counter, the output will remain low while counting and on terminal count, the output will become high until reloaded again.

Let us see the channel in mode0. Connect the CLK 0 to the debounce circuit and execute the following program.

PROGRAM:

```
MVI A, 30H ;Channel 0 in mode 0.  
OUT CEH  
MVI A, 05H ;LSB of count.  
OUT C8H  
MVI A, 00H ;MSB of count.  
OUT C8H  
HLT
```

It is observed in CRO that the output of channel 0 is initially low. After giving 'x' clock pulses, we may notice that the output goes high.

MODE 1-Programmable One Shot:-

After loading the count, the output will remain low following the rising edge of the gate input. The output will go high on the terminal count. It is retriggerable; hence the output will remain low for the full count after any rising edge of the gate input.

The following program initializes channel 0 of 8253 in Mode 1 and also initializes triggering of gate. OUT 0 goes low as clock pulses and after triggering It goes back to high level after five clock pulses. Execute the program and give clock pulses through the debounce logic and verify using CRO.

PROGRAM:

```
MVI A, 32H ;Channel 0 in mode 1.  
OUT CEH ;  
MVI A, 05H ;LSB of count.  
OUT C8H  
MVI A, 00H ;MSB of count.  
OUT C8H  
OUT DOH ;Trigger Gate 0.  
HLT
```

MODE 2-Rate Generator:

It is a simple divide by N counter. The output will be low for one period of the input clock. The period from one output pulse to next equals the number of input count in the count register. If the count register is reloaded between output pulses, the present period will not be affected, but the subsequent period will reflect a new value.

MODE 3-Square Generator:

It is similar to mode 2 except that the output will remain high until one half of the count and goes low for the other half provided the count is an even number. If the count is odd the output will be high for $(\text{count} + 1)/2$ counts. This mode is used for generating baud rate of 8251.

PROGRAM:

```
MVI A, 36H ;Channel 0 in mode 3.  
OUT CEH ;  
MVI A, 0AH ;LSB of count.  
OUT C8H  
MVI A, 00H ;MSB of count.  
OUT C8H  
HLT
```

We utilize mode 3 to generate a square wave of frequency 150 kHz at Channel 0. Set the jumper so that the clock of 8253 is given a square wave of Frequency 1.5 MHz. This program divides the program clock by 10 and thus the Output at channel 0 is 150 KHz.

MODE 4-Software Triggered Strobe:

The output is high after the mode is set and also during counting. On Terminal count, the output will go low for one clock period and becomes high again. This mode can be used for interrupt generation.

MODE 5-Hardware Triggered Strobe:

Counter starts counting after rising edge of trigger input and the output goes low for one clock period. When the terminal count is reached, the counter is retrigerrable. On terminal count, the output will go low for one clock period and becomes high again. This mode can be used for interrupt generation.

RESULT:

Thus the 8253 PIT was interfaced to 8085 and the operations for mode 0, Mode 1 and mode 3 was verified.

INTERFACING 8279 KEYBOARD/DISPLAY CONTROLLER WITH 8085 MICROPROCESSOR

AIM:

To interface 8279 Programmable Keyboard Display Controller to 8085 Microprocessor.

APPARATUS REQUIRED:

- 1) 8085 Microprocessor toolkit.
- 2) 8279 Interface board.
- 3) VXT parallel bus.
- 4) Regulated D.C power supply.

PROGRAM:

| | | |
|--------|------|----------------------------|
| START: | LXI | H,4130H |
| | MVI | D,0FH ;Initialize counter. |
| | MVI | A,10H |
| | OUT | C2H ;Set Mode and Display. |
| | MVI | A,CCH;Clear display. |
| | OUT | C2H |
| | MVI | A,90H ;Write Display |
| | OUT | C2H |
| LOOP: | MOV | A,M |
| | OUT | C0H |
| | CALL | DELAY |
| | INX | H |
| | DCR | D |
| | JNZ | LOOP |
| | JMP | START |
| DELAY: | MVI | B, A0H |
| LOOP2: | MVI | C, FFH |
| LOOP1: | DCR | C |
| | JNZ | LOOP1 |
| | DCR | B |
| | JNZ | LOOP2 |
| | RET | |

Pointer equal to 4130 .FF repeated eight times.

| | |
|------|------|
| 4130 | - FF |
| 4131 | -FF |
| 4132 | -FF |
| 4133 | -FF |
| 4134 | -FF |
| 4135 | -FF |
| 4136 | -FF |
| 4137 | -FF |
| 4138 | -98 |
| 4139 | -68 |
| 413A | -7C |
| 413B | -C8 |
| 413C | -1C |
| 413D | -29 |
| 413E | -FF |
| 413F | -FF |

RESULT:

Thus 8279 controller was interfaced with 8085 and program for rolling display was executed successfully.

8051 MICROCONTROLLER PROGRAMS

ADDITION OF TWO 8 – BIT NUMBERS

AIM:

To perform addition of two 8 – bit numbers using 8051 instruction set.

ALGORITHM:

1. Clear C – register for Carry
2. Get the data immediately .
3. Add the two data
4. Store the result in memory pointed by DPTR

PROGRAM:

| | |
|-------|------------|
| ORG | 4100 |
| CLR | C |
| MOV | A,#data1 |
| ADD | A,#data2 |
| MOV | DPTR,#4500 |
| MOVX | @DPTR,A |
| HERE: | SJMP HERE |

OBSERVATION:

Input: 66
 23

Output: 89 (4500)

RESULT:

Thus the program to perform addition of two 8 – bit numbers using 8051 instruction set was executed.

SUBTRACTION OF TWO 8 – BIT NUMBERS

AIM:

To perform Subtraction of two 8 – bit numbers using 8051 instruction set.

ALGORITHM:

1. Clear C – register for Carry
2. Get the data immediately .
3. Subtract the two data
4. Store the result in memory pointed by DPTR

PROGRAM:

| | |
|-------|------------|
| ORG | 4100 |
| CLR | C |
| MOV | A,#data1 |
| SUBB | A,#data2 |
| MOV | DPTR,#4500 |
| MOVX | @DPTR,A |
| HERE: | SJMP HERE |

OBSERVATION:

Input: 66
 23

Output: 43 (4500)

RESULT:

Thus the program to perform subtraction of two 8 – bit numbers using 8051 instruction set was executed.

MULTIPLICATION OF TWO 8 – BIT NUMBERS

AIM:

To perform multiplication of two 8 – bit numbers using 8051 instruction set.

ALGORITHM:

1. Get the data in A – reg.
2. Get the value to be multiplied in B – reg.
3. Multiply the two data
4. The higher order of the result is in B – reg.
5. The lower order of the result is in A – reg.
6. Store the results.

PROGRAM:

| | |
|-------|------------|
| ORG | 4100 |
| CLR | C |
| MOV | A,#data1 |
| MOV | B,#data2 |
| MUL | AB |
| MOV | DPTR,#4500 |
| MOVX | @DPTR,A |
| INC | DPTR |
| MOV | A,B |
| MOVX | @DPTR,A |
| HERE: | SJMP HERE |

OBSERVATION:

Input: 80
 80

Output: 00 (4500)
 19 (4501)

RESULT:

Thus the program to perform multiplication of two 8 – bit numbers using 8051 instruction set was executed.

DIVISION OF TWO 8 – BIT NUMBERS

AIM:

To perform division of two 8 – bit numbers using 8051 instruction set.

ALGORITHM:

1. Get the data in A – reg.
2. Get the value to be divided in B – reg.
3. Divide the two data
4. The quotient is in A – reg.
5. The remainder is in B – reg.
6. Store the results.

PROGRAM:

| | |
|-------|------------|
| ORG | 4100 |
| CLR | C |
| MOV | A,#data1 |
| MOV | B,#data2 |
| DIV | AB |
| MOV | DPTR,#4500 |
| MOVX | @DPTR,A |
| INC | DPTR |
| MOV | A,B |
| MOVX | @DPTR,A |
| HERE: | SJMP HERE |

OBSERVATION:

Input: 05
 03

Output: 01 (4500)
 02 (4501)

RESULT:

Thus the program to perform multiplication of two 8 – bit numbers using 8051 instruction set was executed.

RAM ADDRESSING

AIM:

To exhibit the RAM direct addressing and bit addressing schemes of 8051 microcontroller.

ALGORITHM:

1. For Bit addressing, Select Bank 1 of RAM by setting 3rd bit of PSW
2. Using Register 0 of Bank 1 and accumulator perform addition
3. For direct addressing provide the address directly (30 in this case)
4. Use the address and Accumulator to perform addition
5. Verify the results

PROGRAM:

Bit Addressing:

| | |
|-------|------------|
| SETB | PSW.3 |
| MOV | R0,#data1 |
| MOV | A,#data2 |
| ADD | A,R0 |
| MOV | DPTR,#4500 |
| MOVX | @DPTR,A |
| HERE: | SJMP HERE |

Direct Addressing:

| | |
|-------|------------|
| MOV | 30,#data1 |
| MOV | A,#data2 |
| ADD | A,30 |
| MOV | DPTR,#4500 |
| MOVX | @DPTR,A |
| HERE: | SJMP HERE |

OBSERVATION:

Bit addressing:

Input: 54
 25

Output: 79 (4500)

Direct addressing:

Input: 54
 25

Output: 79 (4500)

RESULT:

Thus the program to exhibit the different RAM addressing schemes of 8051 was executed.

INTERFACING STEPPER MOTOR WITH 8051

AIM:

To interface stepper motor with 8051 parallel port and to vary speed of motor, direction of motor.

APPARATUS REQUIRED:

- 8051 Trainer Kit
- Stepper Motor Interface Board

THEORY:

A motor in which the rotor is able to assume only discrete stationary angular position is a stepper motor. The rotor motion occurs in a stepwise manner from one equilibrium position to next.

The motor under our consideration uses 2 – phase scheme of operation. In this scheme, any two adjacent stator windings are energized. The switching condition for the above said scheme is shown in Table.

| Clockwise | | | | Anti - Clockwise | | | |
|-----------|----|----|----|------------------|----|----|----|
| A1 | B1 | A2 | B2 | A1 | B1 | A2 | B2 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |

In order to vary the speed of the motor, the values stored in the registers R1, R2, R3 can be changed appropriately.

ALGORITHM:

1. Store the look up table address in DPTR
2. Move the count value (04) to one of the register (R0)
3. Load the control word for motor rotation in accumulator
4. Push the address in DPTR into stack
5. Load FFC0 in to DPTR.
6. Call the delay program
7. Send the control word for motor rotation to the external device.
8. Pop up the values in stack and increment it.
9. Decrement the count in R0. If zero go to next step else proceed to step 3.
10. Perform steps 1 to 9 repeatedly.

PROGRAM:

| | | |
|--------|------|-------------|
| | ORG | 4100 |
| START: | MOV | DPTR,#4500H |
| | MOV | R0,#04 |
| AGAIN: | MOVX | A,@DPTR |
| | PUSH | DPH |
| | PUSH | PDL |
| | MOV | DPTR,#FFC0H |
| | MOV | R2, 04H |
| | MOV | R1,#FFH |
| DLY1: | MOV | R3, #FFH |
| DLY: | DJNZ | R3,DLY |
| | DJNZ | R1,DLY1 |
| | DJNZ | R2,DLY1 |
| | MOVX | @DPTR,A |
| | POP | DPL |
| | POP | DPH |
| | INC | DPTR |
| | DJNZ | R0, AGAIN |
| | SJMP | START |

DATA:

4500: 09, 05, 06, 0A

RESULT:

Thus the speed and direction of motor were controlled using 8051 trainer kit.

INTERFACING DAC WITH 8051

AIM:

To interface DAC with 8051 parallel port to demonstrate the generation of square, saw tooth and triangular wave.

APPARATUS REQUIRED:

- 8051 Trainer Kit
- DAC Interface Board

THEORY:

DAC 0800 is an 8 – bit DAC and the output voltage variation is between – 5V and + 5V. The output voltage varies in steps of $10/256 = 0.04$ (appx.). The digital data input and the corresponding output voltages are presented in the Table below

| Input Data in HEX | Output Voltage |
|-------------------|----------------|
| 00 | - 5.00 |
| 01 | - 4.96 |
| 02 | - 4.92 |
| ... | ... |
| 7F | 0.00 |
| ... | ... |
| FD | 4.92 |
| FE | 4.96 |
| FF | 5.00 |

Referring to Table1, with 00 H as input to DAC, the analog output is – 5V. Similarly, with FF H as input, the output is +5V. Outputting digital data 00 and FF at regular intervals, to DAC, results in different wave forms namely square, triangular, etc.,

ALGORITHM:

(a) Square Wave Generation

1. Move the port address of DAC to DPTR
2. Load the initial value (00) to Accumulator and move it to DAC
3. Call the delay program
4. Load the final value(FF) to accumulator and move it to DAC
5. Call the delay program.
6. Repeat Steps 2 to 5

(b) Saw tooth Wave Generation

1. Move the port address of DAC to DPTR
2. Load the initial value (00) to Accumulator
3. Move the accumulator content to DAC
4. Increment the accumulator content by 1.
5. Repeat Steps 3 and 4.

(c) Triangular Wave Generation

1. Move the port address of DAC to DPTR
2. Load the initial value (00) to Accumulator
3. Move the accumulator content to DAC
4. Increment the accumulator content by 1.
5. If accumulator content is zero proceed to next step. Else go to step 3.
6. Load value (FF) to Accumulator
7. Move the accumulator content to DAC
8. Decrement the accumulator content by 1.
9. If accumulator content is zero go to step2. Else go to step 7.

PROGRAM:**(a) Square Wave Generation**

| | | |
|--------|-------|--------------------------|
| | ORG | 4100 |
| | MOV | DPTR,PORT ADDRESS OF DAC |
| START: | MOV | A,#00 |
| | MOVX | @DPTR,A |
| | LCALL | DELAY |
| | MOV | A,#FF |
| | MOVX | @DPTR,A |
| | LCALL | DELAY |
| | LJUMP | START |
| DELAY: | MOV | R1,#05 |
| LOOP: | MOV | R2,#FF |
| HERE: | DJNZ | R2,HERE |
| | DJNZ | R1,LOOP |
| | RET | |
| | SJMP | START |

(b) Saw tooth Wave Generation

| | | |
|-------|------|--------------------------|
| | ORG | 4100 |
| | MOV | DPTR,PORT ADDRESS OF DAC |
| | MOV | A,#00 |
| LOOP: | MOVX | @DPTR,A |
| | INC | A |
| | SJMP | LOOP |

(c) Triangular Wave Generation

| | | |
|--------|------|--------------------------|
| | ORG | 4100 |
| | MOV | DPTR,PORT ADDRESS OF DAC |
| START: | MOV | A,#00 |
| LOOP1: | MOVX | @DPTR,A |
| | INC | A |
| | JNZ | LOOP1 |
| | MOV | A,#FF |
| LOOP2: | MOVX | @DPTR,A |
| | DEC | A |
| | JNZ | LOOP2 |
| | LJMP | START |

RESULT:

Thus the square, triangular and saw tooth wave form were generated by interfacing DAC with 8051 trainer kit.

PROGRAMMING 8051 USING KEIL SOFTWARE

AIM:

To perform arithmetic operations in 8051 using keil software.

PROCEDURE:

1. Click Keil\u201cVision2 icon in the desktop
2. From Project Menu open New project
3. Select the target device as ATMEL 89C51
4. From File Menu open New File
5. Type the program in Text Editor
6. Save the file with extension “.asm”
7. In project window click the tree showing TARGET
8. A source group will open.
9. Right Click the Source group and click “Add files to Source group”
10. A new window will open. Select our file with extension “.asm”
11. Click Add.
12. Go to project window and right click Source group again
13. Click Build Target (F7).
14. Errors if any will be displayed.
15. From Debug menu, select START/STOP Debug option.
16. In project window the status of all the registers will be displayed.
17. Click Go from Debug Menu.
18. The results stored in registers will be displayed in Project window.
19. Stop the Debug process before closing the application.

PROGRAM:

| | |
|-----|--------|
| ORG | 4100 |
| CLR | C |
| MOV | A,#05H |
| MOV | B,#02H |
| DIV | AB |

OBSERVATION:

A: 02
B: 01
SP:07

Note that Stack pointer is initiated to 07H

RESULT:

Thus the arithmetic operation for 8051 was done using Keil Software.

SYSTEM DESIGN USING MICROCONTROLLER

AIM:

To Design a microcontroller based system for simple applications like security systems combination lock etc.

PROCEDURE:

1. Read number of bytes in the password
2. Initialize the password
3. Initialize the Keyboard Display IC (8279) to get key and Display
4. Blank the display
5. Read the key from user
6. Compare with the initialized password
7. If it is not equal, Display 'E' to indicate Error.
8. Repeat the steps 6 and 7 to read next key
9. If entered password equal to initialized password, Display 'O' to indicate open.

PROGRAM:

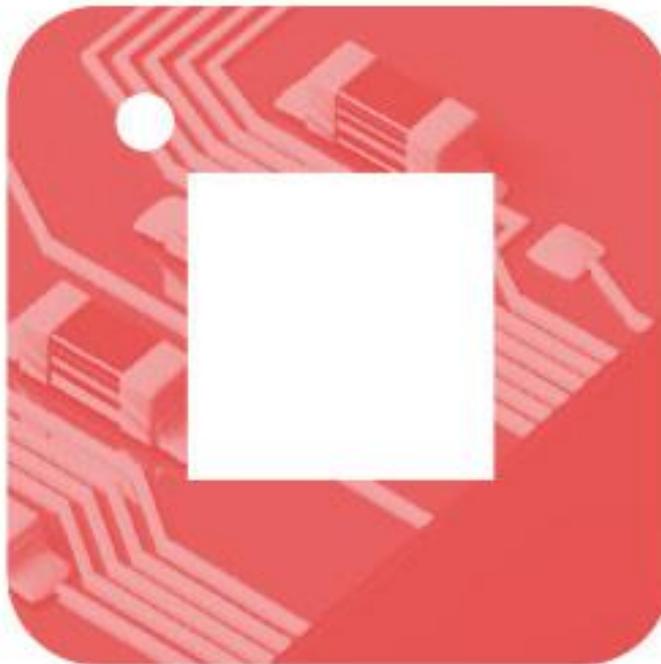
| | |
|-----|------------|
| MOV | 51H,# |
| MOV | 52H,# |
| MOV | 53H,# |
| MOV | 54H,# |
| MOV | R1,#51 |
| MOV | R0,#50 |
| MOV | R3,#04 |
| MOV | R2,#08 |
| MOV | DPTR,#FFC2 |
| MOV | A,#00 |

| | | |
|--------|------|------------|
| | MOVX | @DPTR,A |
| | MOV | A,#CC |
| | MOVX | @DPTR,A |
| | MOV | A,#90 |
| | MOVX | @DPTR,A |
| | MOV | A,#FF |
| | MOV | DPTR,#FFCO |
| LOOP: | MOVX | @DPTR,A |
| | DJNZ | R2,LOOP |
| AGAIN: | MOV | DPTR,#FFC2 |
| WAIT: | MOVX | A,@DPTR |
| | ANL | A,#07 |
| | JZ | WAIT |
| | MOV | A,#40 |
| | MOVX | @DPTR,A |
| | MOV | DPTR,#FFCO |
| | MOVX | A,@DPTR |
| | MOV | @R0,A |
| | MOV | A,@R1 |
| | CJNE | A,50H,NEQ |
| | INC | R1 |
| | DJNZ | R3, AGAIN |
| | MOV | DPTR,#FFCO |
| | MOV | A,#OC |
| | MOVX | @DPTR,A |
| XX: | SJMP | XX |

| | | |
|------|------|------------|
| NEQ: | MOV | DPTR,#FFCO |
| | MOV | A,#68 |
| | MOVX | @DPTR,A |
| YY: | SJMP | YY |

RESULT:

Thus the program for security lock system was executed



MICROPROCESSORS

tutorialspoint
SIMPLY EASY LEARNING

www.tutorialspoint.com



<https://www.facebook.com/tutorialspointindia>



<https://twitter.com/tutorialspoint>

About the Tutorial

A microprocessor is a controlling unit of a micro-computer, fabricated on a small chip capable of performing Arithmetic Logical Unit (ALU) operations and communicating with the other devices connected to it.

In this tutorial, we will discuss the architecture, pin diagram and other key concepts of microprocessors.

Audience

This tutorial is designed for all those readers pursing either Bachelor's or Master's degree in Computer Science. It will help them understand the basic concepts related to Microprocessors.

Prerequisites

In this tutorial, all the topics have been explained from elementary level. Therefore, a beginner can understand this tutorial very easily. However if you have a prior knowledge of computer architecture in general, then it will be quite easy to grasp the concepts explained here.

Disclaimer & Copyright

© Copyright 2016 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com.

Table of Contents

| | |
|---|--------|
| About the Tutorial | i |
| Audience..... | i |
| Prerequisites..... | i |
| Disclaimer & Copyright..... | i |
| Table of Contents | ii |
| MICROPROCESSOR..... | 1 |
| 1. Microprocessor – Overview | 2 |
| How does a Microprocessor Work? | 2 |
| Features of a Microprocessor..... | 3 |
| 2. Microprocessor – Classification..... | 4 |
| RISC Processor | 4 |
| CISC Processor | 6 |
| Special Processors | 7 |
| 8085 MICROPROCESSOR..... | 9 |
| 3. 8085 – Architecture..... | 10 |
| 8085 Microprocessor – Functional Units..... | 10 |
| 8085 Architecture..... | 12 |
| 4. 8085 – Pin Configuration..... | 13 |
| 5. 8085 – Addressing Modes & Interrupts..... | 16 |
| Addressing Modes in 8085 | 16 |
| Interrupts in 8085..... | 16 |
| 6. 8085 – Instruction Sets..... | 19 |
| Control Instructions..... | 19 |
| Branching Instructions..... | 21 |
| Arithmetic Instructions..... | 24 |
| Data Transfer Instructions | 26 |
| 8085 – Demo Programs | 29 |
| 8086 MICROPROCESSOR..... | 32 |
| 7. 8086 – Overview | 33 |
| Features of 8086..... | 33 |
| Comparison between 8085 &8086 Microprocessor..... | 33 |
| Architecture of 8086..... | 34 |
| 8. 8086 – Functional Units..... | 35 |
| EU (Execution Unit) | 35 |
| BIU (Bus Interface Unit)..... | 36 |

| | |
|--|-----------|
| 9. 8086 – Pin Configuration | 38 |
| 10. 8086 – Instruction Sets..... | 43 |
| Data Transfer Instructions | 43 |
| Arithmetic Instructions..... | 44 |
| Bit Manipulation Instructions..... | 45 |
| String Instructions | 46 |
| Program Execution Transfer Instructions (Branch & Loop Instructions) | 46 |
| Processor Control Instructions | 47 |
| Iteration Control Instructions..... | 47 |
| Interrupt Instructions | 48 |
| 11. 8086 – Interrupts | 49 |
| Hardware Interrupts..... | 49 |
| Software Interrupts | 50 |
| 12. 8086 – Addressing Modes | 53 |
| MULTIPROCESSOR CONFIGURATION | 55 |
| 13. Multiprocessor Configuration – Overview..... | 56 |
| Coprocessor Configuration | 56 |
| Closely Coupled Configuration | 57 |
| Loosely Coupled Configuration | 58 |
| 14. 8087 Numeric Data Processor | 60 |
| 8087 Architecture..... | 60 |
| 8087 Pin Description | 61 |
| I/O INTERFACING | 63 |
| 15. I/O Interfacing – Overview..... | 64 |
| 16. 8279 – Programmable Keyboard | 66 |
| Operational Modes of 8279 | 70 |
| 17. 8257 – DMA Controller | 71 |
| How DMA Operations are Performed? | 71 |
| Features of 8257..... | 71 |
| 8257 Architecture..... | 72 |
| MICROCONTROLLERS..... | 76 |
| 18. Microcontrollers – Overview..... | 77 |
| Difference between Microprocessor and Microcontroller..... | 77 |
| Types of Microcontrollers..... | 77 |
| Applications of Microcontrollers | 78 |

| | |
|--|---------------|
| 19. 8051 – Architecture..... | 79 |
| 20. 8051 – Pin Description | 80 |
| 21. 8051 – Input Output Ports..... | 82 |
| Pins Current Limitations | 83 |
| 22. 8051 – Interrupts | 84 |
| PERIPHERAL DEVICES | 86 |
| 23. Intel 8255A – Programmable Peripheral Interface | 87 |
| Ports of 8255A..... | 87 |
| Operating Modes..... | 87 |
| Features of 8255A..... | 88 |
| 8255..... | 88 |
| Architecture..... | 88 |
| 24. Intel 8255A – Pin Description..... | 89 |
| 25. Intel 8253 – Programmable Interval Timer..... | 91 |
| Difference between 8253 and 8254 | 91 |
| Features of 8253 / 54 | 91 |
| 8254 Architecture..... | 92 |
| 8254 Pin Description | 92 |
| 26. Intel 8253/54 – Operational Modes | 95 |

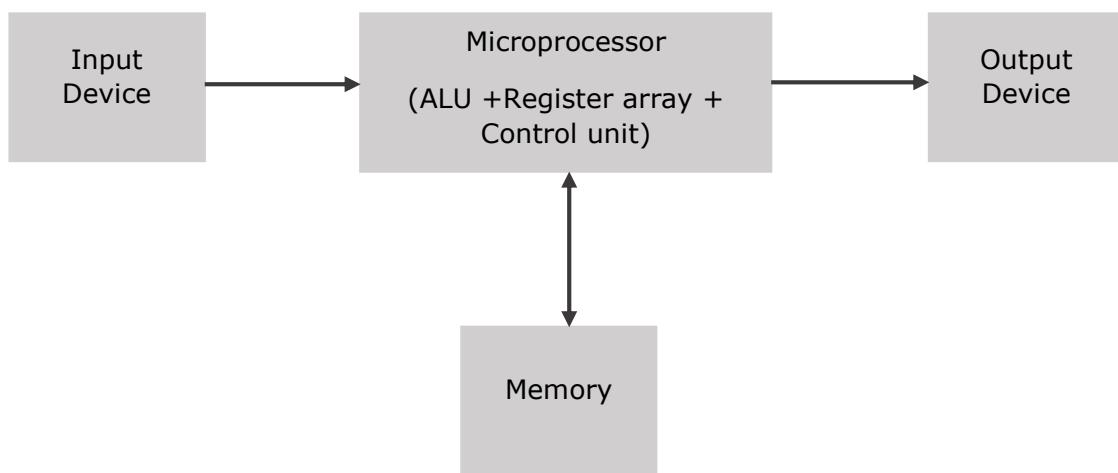
Microprocessor

1. Microprocessor – Overview

Microprocessor is a controlling unit of a micro-computer, fabricated on a small chip capable of performing ALU (Arithmetic Logical Unit) operations and communicating with the other devices connected to it.

Microprocessor consists of an ALU, register array, and a control unit. ALU performs arithmetical and logical operations on the data received from the memory or an input device. Register array consists of registers identified by letters like B, C, D, E, H, L and accumulator. The control unit controls the flow of data and instructions within the computer.

Block Diagram of a Basic Microcomputer



How does a Microprocessor Work?

The microprocessor follows a sequence: Fetch, Decode, and then Execute.

Initially, the instructions are stored in the memory in a sequential order. The microprocessor fetches those instructions from the memory, then decodes it and executes those instructions till STOP instruction is reached. Later, it sends the result in binary to the output port. Between these processes, the register stores the temporarily data and ALU performs the computing functions.

List of Terms Used in a Microprocessor

Here is a list of some of the frequently used terms in a microprocessor:

- **Instruction Set:** It is the set of instructions that the microprocessor can understand.
- **Bandwidth:** It is the number of bits processed in a single instruction.
- **Clock Speed:** It determines the number of operations per second the processor can perform. It is expressed in megahertz (MHz) or gigahertz (GHz). It is also known as Clock Rate.
- **Word Length:** It depends upon the width of internal data bus, registers, ALU, etc. An 8-bit microprocessor can process 8-bit data at a time. The word length ranges from 4 bits to 64 bits depending upon the type of the microcomputer.
- **Data Types:** The microprocessor has multiple data type formats like binary, BCD, ASCII, signed and unsigned numbers.

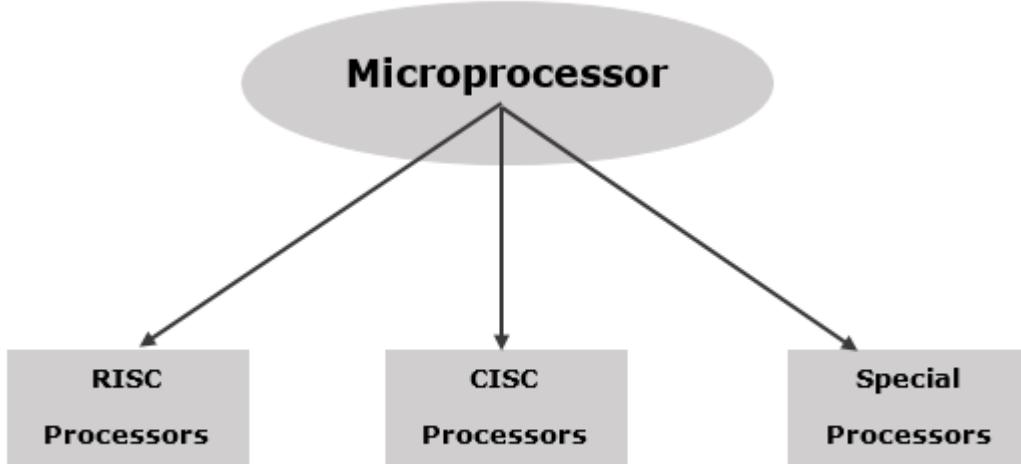
Features of a Microprocessor

Here is a list of some of the most prominent features of any microprocessor:

- **Cost-effective:** The microprocessor chips are available at low prices and results its low cost.
- **Size:** The microprocessor is of small size chip, hence is portable.
- **Low Power Consumption:** Microprocessors are manufactured by using metal-oxide semiconductor technology, which has low power consumption.
- **Versatility:** The microprocessors are versatile as we can use the same chip in a number of applications by configuring the software program.
- **Reliability:** The failure rate of an IC in microprocessors is very low, hence it is reliable.

2. Microprocessor – Classification

A microprocessor can be classified into three categories:



RISC Processor

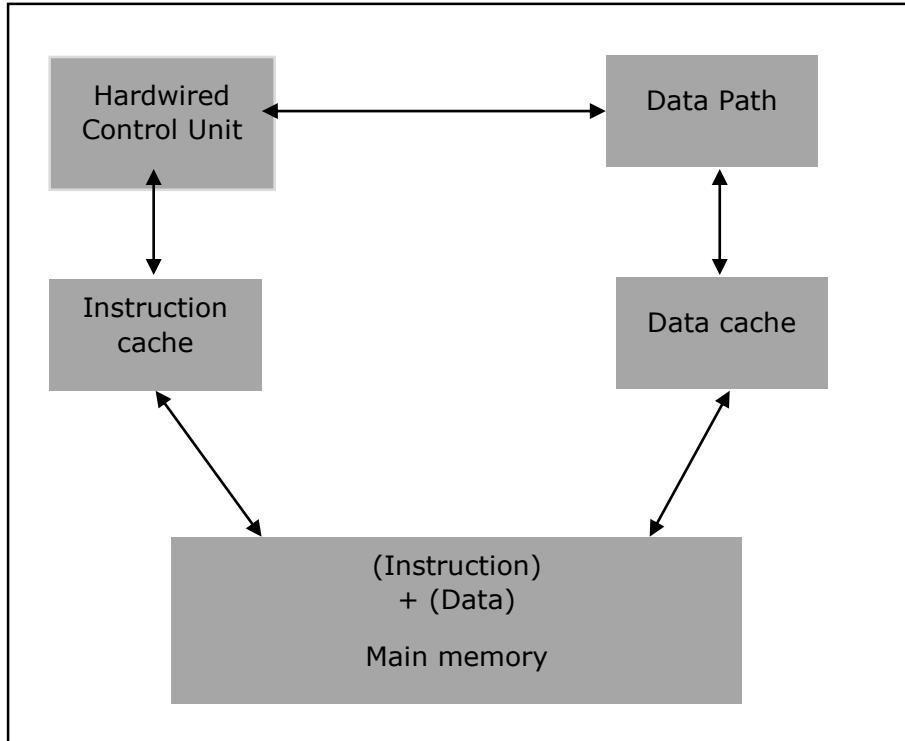
RISC stands for **Reduced Instruction Set Computer**. It is designed to reduce the execution time by simplifying the instruction set of the computer. Using RISC processors, each instruction requires only one clock cycle to execute resulting in uniform execution time. This reduces the efficiency as there are more lines of code, hence more RAM is needed to store the instructions. The compiler also has to work more to convert high-level language instructions into machine code.

Some of the RISC processors are:

- Power PC: 601, 604, 615, 620
- DEC Alpha: 210642, 211066, 21068, 21164
- MIPS: TS (R10000) RISC Processor
- PA-RISC: HP 7100LC

Architecture of RISC

RISC microprocessor architecture uses highly-optimized set of instructions. It is used in portable devices like Apple iPod due to its power efficiency.



Characteristics of RISC

The major characteristics of a RISC processor are as follows:

- It consists of simple instructions.
- It supports various data-type formats.
- It utilizes simple addressing modes and fixed length instructions for pipelining.
- It supports register to use in any context.
- One cycle execution time.
- "LOAD" and "STORE" instructions are used to access the memory location.
- It consists of larger number of registers.
- It consists of less number of transistors.

CISC Processor

CISC stands for **Complex Instruction Set Computer**. It is designed to minimize the number of instructions per program, ignoring the number of cycles per instruction. The emphasis is on building complex instructions directly into the hardware.

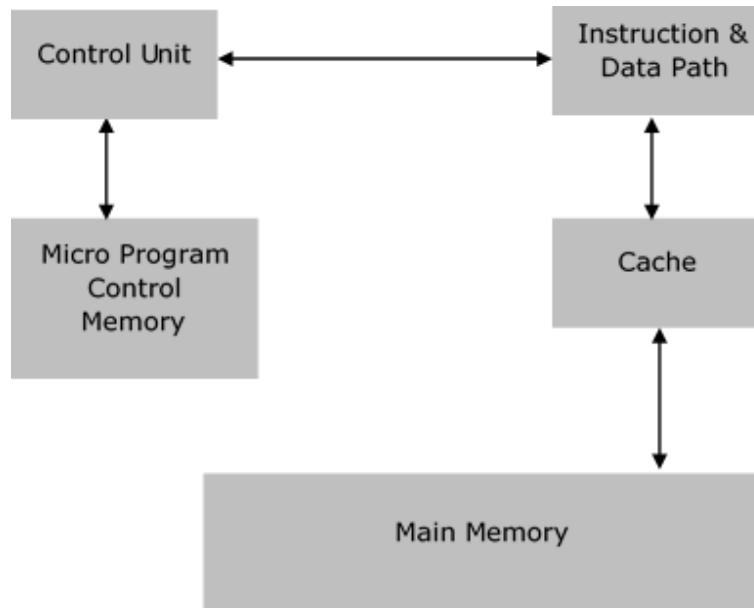
The compiler has to do very little work to translate a high-level language into assembly level language/machine code because the length of the code is relatively short, so very little RAM is required to store the instructions.

Some of the CISC Processors are:

- IBM 370/168
- VAX 11/780
- Intel 80486

Architecture of CISC

Its architecture is designed to decrease the memory cost because more storage is needed in larger programs resulting in higher memory cost. To resolve this, the number of instructions per program can be reduced by embedding the number of operations in a single instruction.



Characteristics of CISC

- Variety of addressing modes.
- Larger number of instructions.

- Variable length of instruction formats.
- Several cycles may be required to execute one instruction.
- Instruction-decoding logic is complex.
- One instruction is required to support multiple addressing modes.

Special Processors

These are the processors which are designed for some special purposes. Few of the special processors are briefly discussed:

Coprocessor

A coprocessor is a specially designed microprocessor, which can handle its particular function many times faster than the ordinary microprocessor.

For example: Math Coprocessor.

Some Intel math-coprocessors are:

- 8087-used with 8086
- 80287-used with 80286
- 80387-used with 80386

Input/Output Processor

It is a specially designed microprocessor having a local memory of its own, which is used to control I/O devices with minimum CPU involvement.

For example:

- DMA (direct Memory Access) controller
- Keyboard/mouse controller
- Graphic display controller
- SCSI port controller

Transputer (Transistor Computer)

A transputer is a specially designed microprocessor with its own local memory and having links to connect one transputer to another transputer for inter-processor communications. It was first designed in 1980 by Inmos and is targeted to the utilization of VLSI technology.

A transputer can be used as a single processor system or can be connected to external links, which reduces the construction cost and increases the performance.

For example: 16-bit T212, 32-bit T425, the floating point (T800, T805 & T9000) processors.

DSP (Digital Signal Processor)

This processor is specially designed to process the analog signals into a digital form. This is done by sampling the voltage level at regular time intervals and converting the voltage at that instant into a digital form. This process is performed by a circuit called an analogue to digital converter, A to D converter or ADC.

A DSP contains the following components:

- **Program Memory:** It stores the programs that DSP will use to process data.
- **Data Memory:** It stores the information to be processed.
- **Compute Engine:** It performs the mathematical processing, accessing the program from the program memory and the data from the data memory.
- **Input/Output:** It connects to the outside world.

Its applications are:

- Sound and music synthesis
- Audio and video compression
- Video signal processing
- 2D and 3d graphics acceleration.

For example: Texas Instrument's TMS 320 series, e.g., TMS 320C40, TMS320C50.

8085 Microprocessor

3. 8085 – Architecture

8085 is pronounced as "eighty-eighty-five" microprocessor. It is an 8-bit microprocessor designed by Intel in 1977 using NMOS technology.

It has the following configuration:

- 8-bit data bus
- 16-bit address bus, which can address upto 64KB
- A 16-bit program counter
- A 16-bit stack pointer
- Six 8-bit registers arranged in pairs: BC, DE, HL
- Requires +5V supply to operate at 3.2 MHZ single phase clock

It is used in washing machines, microwave ovens, mobile phones, etc.

8085 Microprocessor – Functional Units

8085 consists of the following functional units:

Accumulator

It is an 8-bit register used to perform arithmetic, logical, I/O & LOAD/STORE operations. It is connected to internal data bus & ALU.

Arithmetic and logic unit

As the name suggests, it performs arithmetic and logical operations like Addition, Subtraction, AND, OR, etc. on 8-bit data.

General purpose register

There are 6 general purpose registers in 8085 processor, i.e. B, C, D, E, H & L. Each register can hold 8-bit data.

These registers can work in pair to hold 16-bit data and their pairing combination is like B-C, D-E & H-L.

Program counter

It is a 16-bit register used to store the memory address location of the next instruction to be executed. Microprocessor increments the program whenever an instruction is being executed, so that the program counter points to the memory address of the next instruction that is going to be executed.

Stack pointer

It is also a 16-bit register works like stack, which is always incremented/decremented by 2 during push & pop operations.

Temporary register

It is an 8-bit register, which holds the temporary data of arithmetic and logical operations.

Flag register

It is an 8-bit register having five 1-bit flip-flops, which holds either 0 or 1 depending upon the result stored in the accumulator.

These are the set of 5 flip-flops:

- Sign (S)
- Zero (Z)
- Auxiliary Carry (AC)
- Parity (P)
- Carry (C)

Its bit position is shown in the following diagram:

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|----|----|----|----|----|----|----|----|
| S | Z | | AC | | P | | CY |

Instruction register and decoder

It is an 8-bit register. When an instruction is fetched from memory then it is stored in the Instruction register. Instruction decoder decodes the information present in the Instruction register.

Timing and control unit

It provides timing and control signal to the microprocessor to perform operations. Following are the timing and control signals, which control external and internal circuits:

- Control Signals: READY, RD', WR', ALE
- Status Signals: S0, S1, IO/M'
- DMA Signals: HOLD, HLDA
- RESET Signals: RESET IN, RESET OUT

Interrupt control

As the name suggests it controls the interrupts during a process. When a microprocessor is executing a main program and whenever an interrupt occurs, the microprocessor shifts the control from the main program to process the incoming request. After the request is completed, the control goes back to the main program.

There are 5 interrupt signals in 8085 microprocessor: INTR, RST 7.5, RST 6.5, RST 5.5, TRAP.

Serial Input/output control

It controls the serial data communication by using these two instructions: SID (Serial input data) and SOD (Serial output data).

Address buffer and address-data buffer

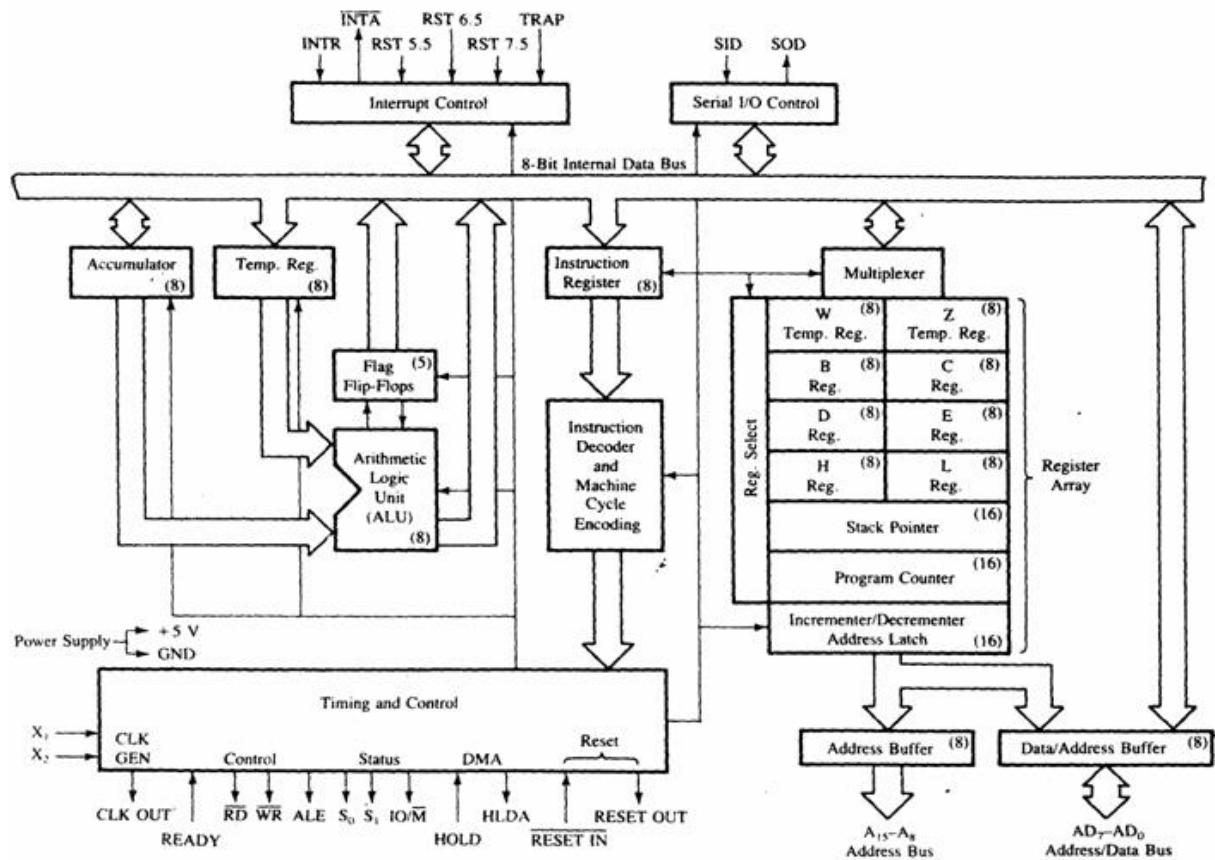
The content stored in the stack pointer and program counter is loaded into the address buffer and address-data buffer to communicate with the CPU. The memory and I/O chips are connected to these buses; the CPU can exchange the desired data with the memory and I/O chips.

Address bus and data bus

Data bus carries the data to be stored. It is bidirectional, whereas address bus carries the location to where it should be stored and it is unidirectional. It is used to transfer the data & Address I/O devices.

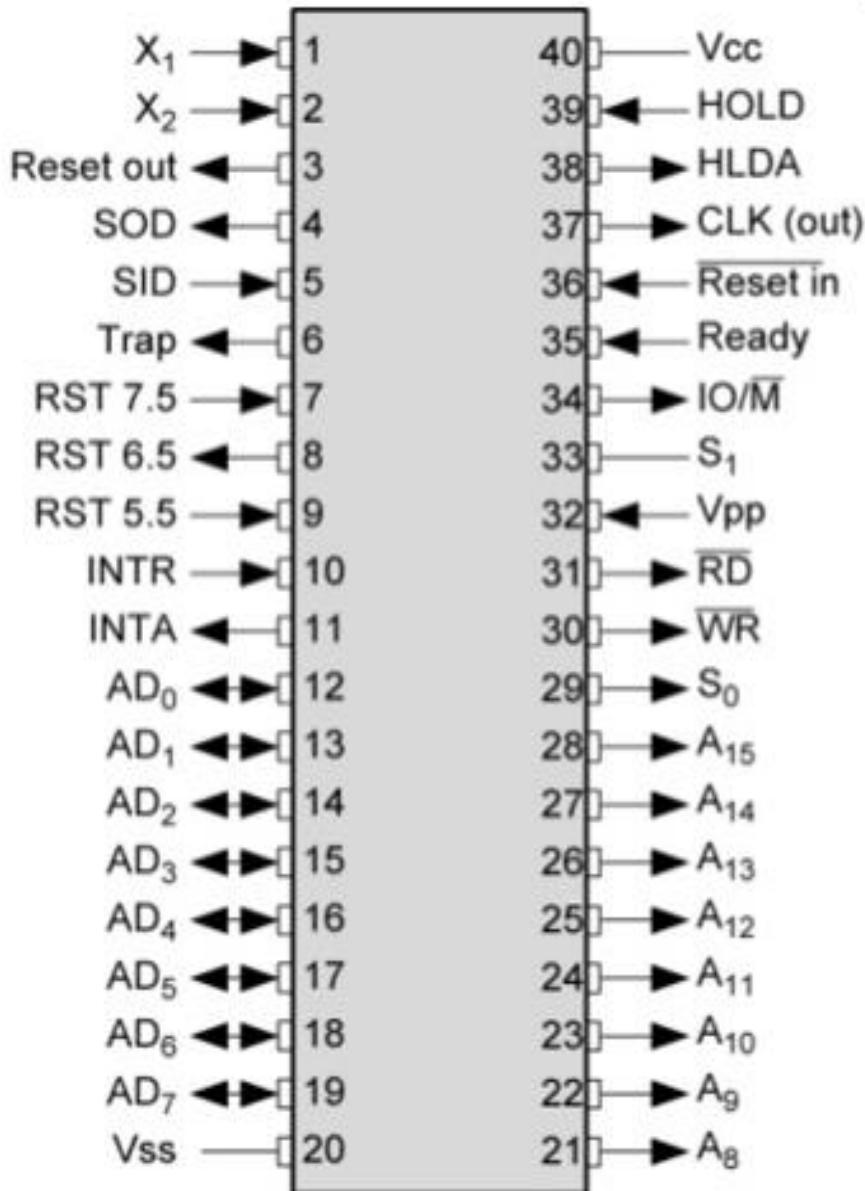
8085 Architecture

We have tried to depict the architecture of 8085 with this following image:



4. 8085 – Pin Configuration

The following image depicts the pin diagram of 8085 Microprocessor:



The pins of a 8085 microprocessor can be classified into seven groups:

Address bus

A15-A8, it carries the most significant 8-bits of memory/IO address.

Data bus

AD7-AD0, it carries the least significant 8-bit address and data bus.

Control and status signals

These signals are used to identify the nature of operation. There are 3 control signal and 3 status signals.

Three control signals are RD, WR & ALE.

- **RD:** This signal indicates that the selected IO or memory device is to be read and is ready for accepting data available on the data bus.
- **WR:** This signal indicates that the data on the data bus is to be written into a selected memory or IO location.
- **ALE:** It is a positive going pulse generated when a new operation is started by the microprocessor. When the pulse goes high, it indicates address. When the pulse goes down it indicates data.

Three status signals are IO/M, S0 & S1.

IO/M

This signal is used to differentiate between IO and Memory operations, i.e. when it is high indicates IO operation and when it is low then it indicates memory operation.

S1 & S0

These signals are used to identify the type of current operation.

Power supply

There are 2 power supply signals: VCC & VSS. VCC indicates +5v power supply and VSS indicates ground signal.

Clock signals

There are 3 clock signals, i.e. X1, X2, CLK OUT.

- **X1, X2:** A crystal (RC, LC N/W) is connected at these two pins and is used to set frequency of the internal clock generator. This frequency is internally divided by 2.

- **CLK OUT:** This signal is used as the system clock for devices connected with the microprocessor.

Interrupts & externally initiated signals

Interrupts are the signals generated by external devices to request the microprocessor to perform a task. There are 5 interrupt signals, i.e. TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR. We will discuss interrupts in detail in interrupts section.

- **INTA:** It is an interrupt acknowledgment signal.
- **RESET IN:** This signal is used to reset the microprocessor by setting the program counter to zero.
- **RESET OUT:** This signal is used to reset all the connected devices when the microprocessor is reset.
- **READY:** This signal indicates that the device is ready to send or receive data. If READY is low, then the CPU has to wait for READY to go high.
- **HOLD:** This signal indicates that another master is requesting the use of the address and data buses.
- **HLDA (HOLD Acknowledge):** It indicates that the CPU has received the HOLD request and it will relinquish the bus in the next clock cycle. HLDA is set to low after the HOLD signal is removed.

Serial I/O signals

There are 2 serial signals, i.e. SOD and SID and these signals are used for serial communication.

- **SOD** (Serial output data line): The output SOD is set/reset as specified by the SIM instruction.
- **SID** (Serial input data line): The data on this line is loaded into accumulator whenever a RIM instruction is executed.

End of ebook preview
If you liked what you saw...
Buy it from our store @ <https://store.tutorialspoint.com>

LABORATORY MANUAL

EE0310 – MICROPROCESSOR & MICROCONTROLLER LAB



DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING
FACULTY OF ENGINEERING & TECHNOLOGY
SRM UNIVERSITY, Kattankulathur – 603 203

LIST OF EXPERIMENTS

| Sl.No. | Name of the Experiments | Page No. |
|---------------|--|-----------------|
| 1 | Induction to 8085 Microprocessor | |
| 2 | a) Addition of 2 - 8 bit numbers b) Subtraction of 2 - 8 bit numbers | |
| 3 | a) Addition of 2 - 16 bit numbers b) Subtraction of 2 – 16 bit numbers | |
| 4 | a) Multiplication of 2 - 8 numbers b) Division of 2 - 8 bit numbers | |
| 5 | a) Ascending order b) Descending order | |
| 6 | a) Fibonaci Series b) Sum of finite series | |
| 7 | Factorial of Given Numbers | |
| 8 | a) Multiplication of 2 - 16 bit numbers b) Division of 2 - 16 bit numbers | |
| 9 | a) Binary to BCD code conversions b) BCD to Binary code conversions | |
| 10 | a) Rolling Display b) Flashing Display | |
| 11 | Stepper motor rotate forward and reverse direction | |
| 12 | Digital analog conversion | |
| 13 | Analog digital conversion | |
| 14 | Microcontroller a) Addition b) Subtraction c) Multiplication d) Division | |

INTRODUCTION TO MICROPROCESSOR 8085

Aim

To study the microprocessor 8085

Architecture of 8085 Microprocessor

a) General purpose register

It is an 8 bit register i.e. B,C,D,E,H,L. The combination of 8 bit register is known as register pair, which can hold 16 bit data. The HL pair is used to act as memory pointer is accessible to program.

b) Accumulator

It is an 8 bit register which hold one of the data to be processed by ALU and stored the result of the operation.

c) Program counter (PC)

It is a 16 bit pointer which maintain the address of a byte entered to line stack.

d) Stack pointer (Sp)

It is a 16 bit special purpose register which is used to hold line memory address for line next instruction to be executed.

e) Arithmetic and logical unit

It carries out arithmetic and logical operation by 8 bit address it uses the accumulator content as input the ALU result is stored back into accumulator.

f) Temporary register

It is an 8 bit register associated with ALU hold data, entering an operation, used by the microprocessor and not accessible to programs.

g) Flags

Flag register is a group of five, individual flip flops line content of line flag register will change after execution of arithmetic and logic operation. The line states flags are

- i) Carry flag (C)
- ii) Parity flag (P)
- iii) Zero flag (Z)
- iv) Auxiliary carry flag (AC)
- v) Sign flag (S)

h) Timing and control unit

Synchronous all microprocessor, operation with the clock and generator and control signal from it necessary to communicate between controller and peripherals.

i) Instruction register and decoder

Instruction is fetched from line memory and stored in line instruction register decoder the stored information.

j) Register Array

These are used to store 8 bit data during execution of some instruction.

PIN Description

Address Bus

1. The pins $A_0 - A_{15}$ denote the address bus.
2. They are used for most significant bit

Address / Data Bus

1. $AD_0 - AD_7$ constitutes the address / Data bus
2. These pins are used for least significant bit

ALE : (Address Latch Enable)

1. The signal goes high during the first clock cycle and enables the lower order address bits.

IO / M

1. This distinguishes whether the address is for memory or input.
2. When this pins go high, the address is for an I/O device.

$S_0 - S_1$

S_0 and S_1 are status signal which provides different status and functions.

RD

1. This is an active low signal
2. This signal is used to control READ operation of the microprocessor.

WR

1. WR is also an active low signal
2. Controls the write operation of the microprocessor.

HOLD

1. This indicates if any other device is requesting the use of address and data bus.

HLDA

1. HLDA is the acknowledgement signal for HOLD
2. It indicates whether the hold signal is received or not.

INTR

1. INTE is an interrupt request signal
2. IT can be enabled or disabled by using software

INTA

1. Whenever the microprocessor receives interrupt signal
2. It has to be acknowledged.

RST 5.5, 6.5, 7.5

1. These are nothing but the restart interrupts
2. They insert an internal restart junction automatically.

TRAP

1. Trap is the only non-maskable interrupt
2. It cannot be enabled (or) disabled using program.

RESET IN

1. This pin resets the program counter to 0 to 1 and results interrupt enable and HLDA flip flops.

X₁, X₂

These are the terminals which are connected to external oscillator to produce the necessary and suitable clock operation.

SID

This pin provides serial input data

SOD

This pin provides serial output data

V_{CC} and V_{SS}

1. V_{CC} is +5V supply pin
2. V_{SS} is ground pin

Specifications**1. Processors**

Intel 8085 at E144 MHz clock

2. Memory

| | |
|-------------------|---------------|
| Monitor RAM: | 0000 – IFFF |
| EPROM Expansion: | 2000 – 3FFF's |
| | 0000 – FFF |
| System RAM: | 4000 – 5FFF |
| Monitor data area | 4100 – 5FFF |
| RAM Expansion | 6000 – BFFF |

3. Input / Output

Parallel: A8 TTL input timer with 2 number of 32-55 only input timer available in μ-85 EBI.

Serial: Only one number RS 232-C, Compatible, crucial interface using 8281A

Timer: 3 channel -16 bit programmable units, using 8253 channel '0' used for no band late. Clock generator. Channel '1' is used for single stopping used program.

Display: 6 digit – 7 segment LED display with filter 4 digit for adder display and 2 digit for data display.

Key board: 21 keys, soft keyboard including common keys and hexa decimal keys.

RES: Reset keys allow to terminate any present activity and retain to μ - 85 its on initialize state.

INT: Maskable interrupt connect to CPU's RST 7.5 interrupt

DEC: Decrement the adder by 1

EXEC: Execute line particular value after selecting address through go command.

NEXT: Increment the address by 1 and then display its content.

Key Functions:

| | |
|---|-----|
| 0 | E |
| | SUB |

- i. Hex entry key '0'
- ii. Substituting memory content where "next" key is pressed immediately after 1, take used to set cutting address.
- iii. Register key 'E'

| | |
|---|-----|
| 1 | RD |
| | REG |

- i) Hex code entry (1)
- ii) Register key 'D'

| | |
|---|----|
| 2 | C |
| | TN |

- i) Hex code entry '2'
- ii) Retrieve data from data 'memory' to data top
- iii) Register key 'C'

| | |
|---|----|
| 3 | B |
| | TR |

- i) Hex code entry '3'
- ii) Retrieve data from memory to top
- iii) Register key 'B'

| | |
|---|------|
| 4 | F |
| | BLOC |

- i) Hex key entry 'C'
- ii) Block search from byte
- iii) Register key 'F'

| | |
|---|------|
| 5 | A |
| | FILL |

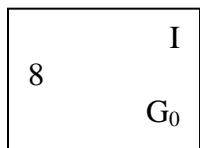
- i) Hex key entry '5'
- ii) Fill block of RAM memory with desired data
- iii) Register key 'A'

| | |
|---|-----|
| 6 | L |
| | SER |

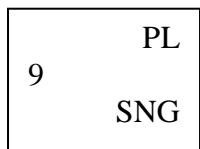
- i) Hex key entry '6'
- ii) TN/TI used for sending (or) receiving
- iii) Register key 'H'

| | |
|---|----------------|
| 7 | H |
| | F ₂ |

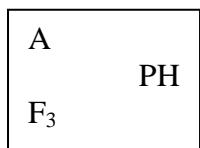
- i) Hex key entry '7'
- ii) Register key 'H'



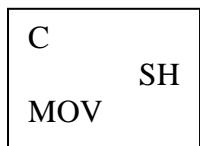
- i) Register key ‘S’
- ii) Register key ‘I’



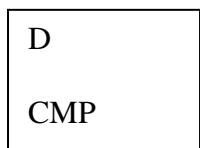
- i) Hex key entry ‘A’
- ii) Function key F₃
- iii) Register key “ph”



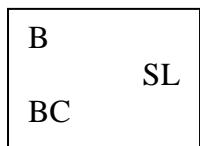
- i) Hex key entry “y”
- ii) Signal step program (instruction by instruction)



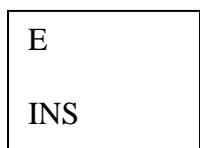
- i) Hex key entry “c”
- ii) Much a block of memory from a linear block
- iii) Register key “S_H”



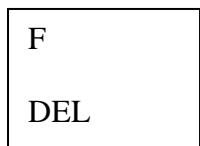
- i) Hex key D
- ii) Compare 2 memory block



- i) Hex key entry ‘B’
- ii) Check a block from flame
- iii) Register key “SPL”



- i) Hex key ‘E’
- ii) Insert by test into memory (RAM)



- i) Hex key ‘F’
- ii) Delete byte from memory RAM

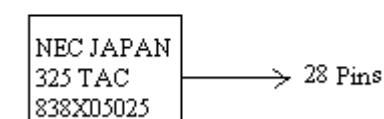
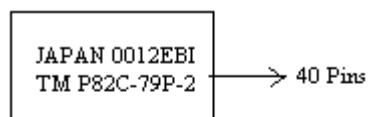
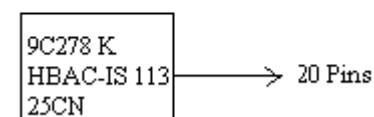
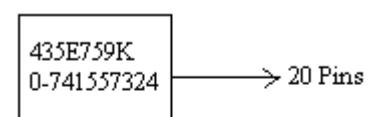
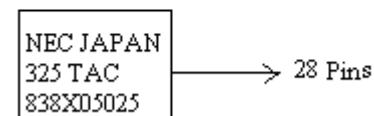
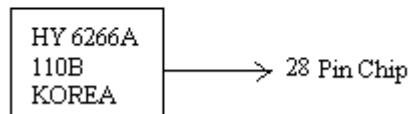
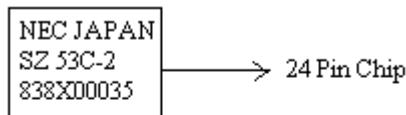
System Power Consumption

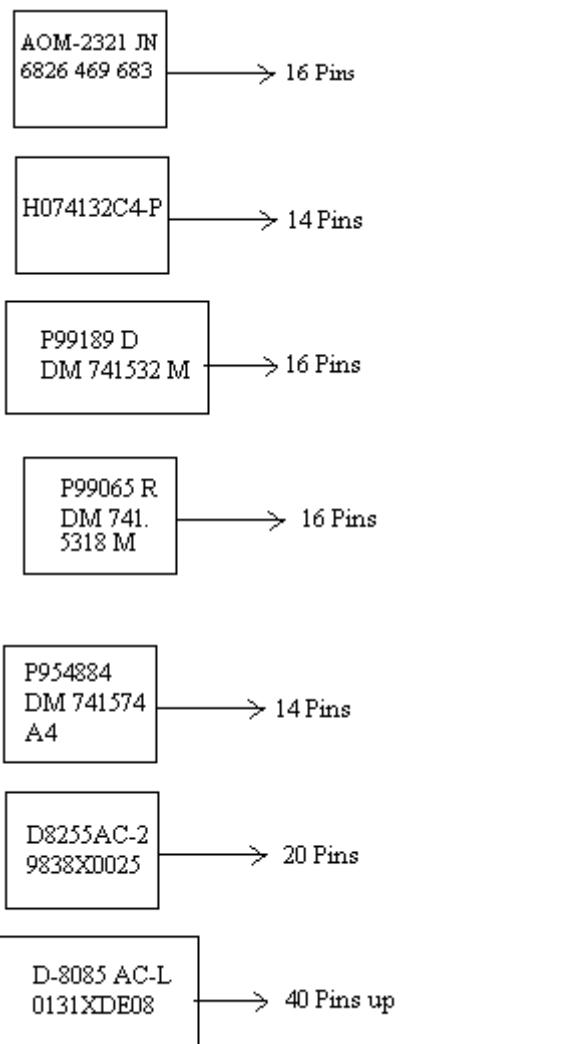
Micro BSEB2
+5V @ 1Amp
+12V @ 200 mA
- 12V @ 100 mA

MICRO SSEB
+5V@ 800 mA

Power Supply Specification
MICRO SSEM
230V, AC @ 80 Hz
+5V @ 600 mA

Key Function





IC's Used

| | | |
|-------|---|---|
| 8085 | - | 8 bit μp |
| 8253 | - | programmable internal timer |
| 8255 | - | programmable peripheral interface |
| 8279 | - | programmable key boards / display interface |
| 8251 | - | programmable communication interface |
| 2764 | - | 8 KV VV EPROM |
| 6264 | - | 8K STATIC PROM |
| 7414 | - | Hex inverter |
| 7432 | - | Quad 21/p OR GATE |
| 7409 | - | Quad 21/p AND GATE |
| 7400 | - | NAND Gate |
| 7404 | - | Dual D-FF |
| 74373 | - | Octal 'D' Latch |
| 74139 | - | Dual 2 to 4 line decoder |
| 74138 | - | 3 to 8 line decoder |

In Enter Program into Trainer Kit

1. Press ‘RESET’ key
2. Sub (key processor represent address field)
3. Enter the address (16 bit) and digit in hex
4. Press ‘NEXT’ key
5. Enter the data
6. Again press “NEXT”
7. Again after taking the program, are use HLT instruction its Hex code
8. Press “NEXT”

How to execute program

1. Press “RESET”
2. Press “GO”
3. Enter the address location in which line program was executed
4. Press “Execute” key

Result:

Thus 8085 microprocessor was studied successfully.

ADDITION OF TWO 8-BIT NUMBERS

Aim:

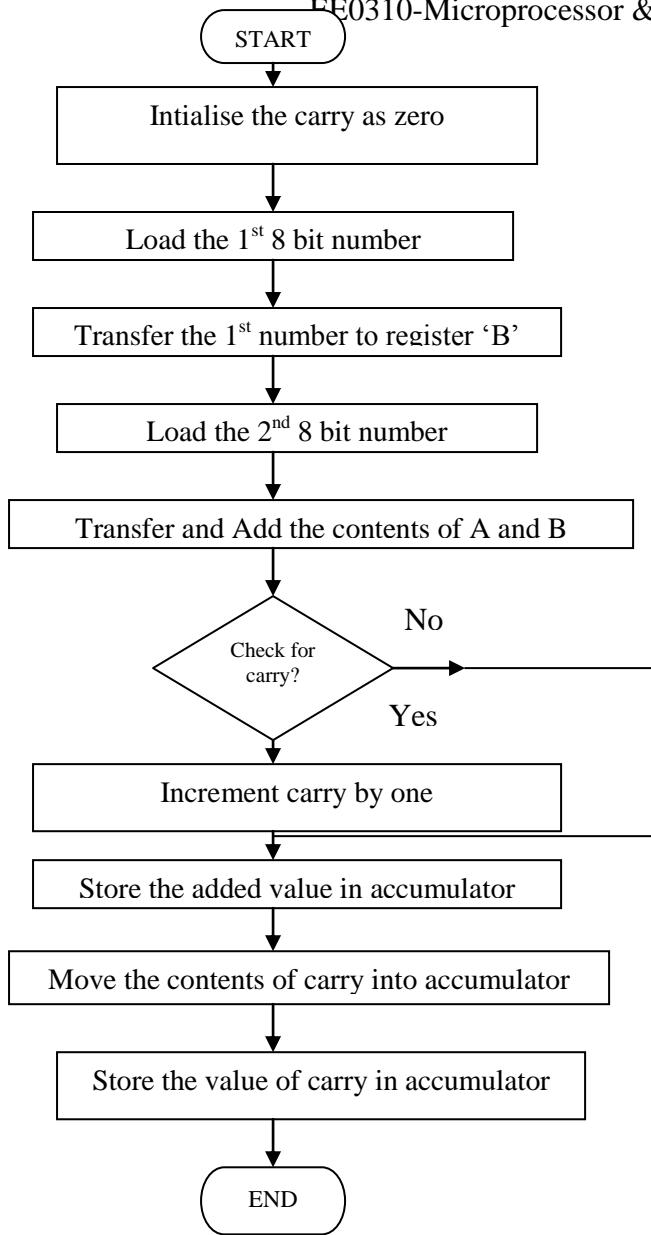
To write an assembly language for adding two 8 bit numbers by using micro processor kit.

Apparatus required:

8085 micro processor kit
(0-5V) DC battery

Algorithm:

- | | | |
|---------|---|--|
| Step 1 | : | Start the microprocessor |
| Step 2 | : | Intialize the carry as 'Zero' |
| Step 3 | : | Load the first 8 bit data into the accumulator |
| Step 4 | : | Copy the contents of accumulator into the register 'B' |
| Step 5 | : | Load the second 8 bit data into the accumulator. |
| Step 6 | : | Add the 2 - 8 bit datas and check for carry. |
| Step 7 | : | Jump on if no carry |
| Step 8 | : | Increment carry if there is |
| Step 9 | : | Store the added request in accumulator |
| Step 10 | : | More the carry value to accumulator |
| Step 11 | : | Store the carry value in accumulator |
| Step 12 | : | Stop the program execution. |



| Address | Label | Mnemonics | Hex Code | Comments |
|---------|-------|-----------|--------------|--|
| 4100 | | MVI C,00 | OE, 00 | Initialize the carry as zero |
| 4102 | | LDA 4300 | 3A, (00, 43) | Load the first 8 bit data |
| 4105 | | MOV, B,A | 47 | Copy the value of 8 bit data into register B |
| 4106 | | LDA 4301 | 3A, (01, 43) | Load the second 8 bit data into the accumulator |
| 4109 | | ADD B | 80 | Add the two values |
| 410A | | JNC | D2, 0E, 41 | Jump on if no carry |
| 410D | | INR C | OC | If carry is there increment it by one |
| 410E | Loop | STA 4302 | 32 (02, 43) | Store the added value in the accumulator |
| 4111 | | MOV A,C | 79 | Move the value of carry to the accumulator from register C |
| 4112 | | STA 4303 | 32 (03, 43) | Store the value of carry in the accumulator |
| 4115 | | HLT | 76 | Stop the program execution |

Input

Without carry

| Input Address | Value |
|---------------|-------|
| 4300 | 04 |
| 4301 | 02 |

Output

| Output Address | Value |
|----------------|------------|
| 4302 | 06 |
| 4303 | 00 (carry) |

With carry

| Input Address | Value |
|---------------|-------|
| 4300 | FF |
| 4301 | FF |

| Output Address | Value |
|----------------|------------|
| 4302 | FE |
| 4303 | 01 (carry) |

Calculation

| | |
|----------|------|
| 1111 | 1111 |
| 1111 | 1111 |
| ----- | |
| (1) 1111 | 1110 |
| ===== | |
| F | E |

Result:

The assembly language program for 8 bit addition of two numbers was executed successfully by using 8085 micro processing kit.

SUBTRACTION OF TWO 8 BIT NUMBERS

Aim:

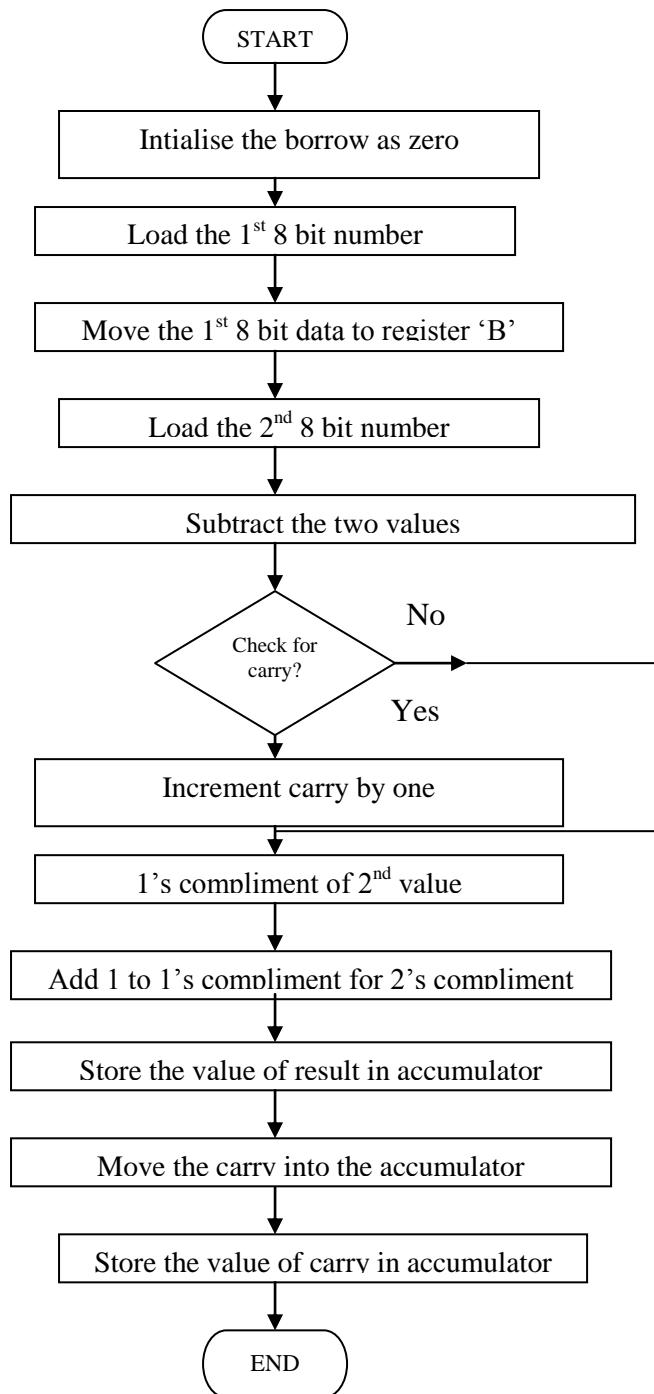
To write a assembly language program for subtracting 2 bit (8) numbers by using- 8085 micro processor kit.

Apparatus required:

8085 micro processor kit
(0-5V) DC battery

Algorithm:

- | | | |
|---------|---|---|
| Step 1 | : | Start the microprocessor |
| Step 2 | : | Intialize the carry as ‘Zero’ |
| Step 3 | : | Load the first 8 bit data into the accumulator |
| Step 4 | : | Copy the contents of contents into the register ‘B’ |
| Step 5 | : | Load the second 8 bit data into the accumulator. |
| Step 6 | : | Subtract the 2 8 bit datas and check for borrow. |
| Step 7 | : | Jump on if no borrow |
| Step 8 | : | Increment borrow if there is |
| Step 9 | : | 2’s compliment of accumulator is found out |
| Step 10 | : | Store the result in the accumulator |
| Step 11 | : | Move the borrow value from ‘c’ to accumulator |
| Step 12 | : | Store the borrow value in the accumulator |
| Step 13 | : | Stop program execution |



| Address | Label | Mnemonics | Hex Code | Comments |
|----------------|--------------|------------------|-----------------|--|
| 4100 | | MVI C,00 | OE, 00 | Initialize the carry as zero |
| 4102 | | LDA 4300 | 3A, (00, 43) | Load the first 8 bit data into the accumulator |
| 4105 | | MOV, B,A | 47 | Copy the value into register 'B' |
| 4106 | | LDA 4301 | 3A, (01, 43) | Load the 2 nd 8 bit data into the accumulator |
| 4109 | | SUB B | 90 | Subtract both the values |
| 410A | Loop | INC | D2, 0E, 41 | Jump on if no borrow |
| 410D | | INR C | OC | If borrow is there, increment it by one |
| 410E | Loop | CMA | 2F | Compliment of 2 nd data |
| 410F | | ADI, 01 | 6, 01 | Add one to 1's compliment of 2 nd data |
| 4111 | | STA 4302 | 32,02,43 | Store the result in accumulator |
| 4114 | | MOV A,C | 79 | Move the value of borrow into the accumulator |
| 4115 | | STA 4303 | 32,03,43 | Store the result in accumulator |
| 4118 | | HLT | 76 | Stop Program execution |

Input

Without borrow

| Input Address | Value |
|---------------|-------|
| 4300 | 05 |
| 4301 | 07 |

Output

| Output Address | Value |
|----------------|-------------|
| 4302 | 02 |
| 4303 | 00 (borrow) |

With carry borrow

| Input Address | Value |
|---------------|-------|
| 4300 | 07 |
| 4301 | 05 |

| Output Address | Value |
|----------------|-------------|
| 4302 | 02 |
| 4303 | 01 (borrow) |

Calculation $05 - 07$
 $07 - 0111$

CMA 1000

ADJ 0.1 0001

1001

05 - 0101

1110 (-2)

Result:

The assembly language program subtraction of two 8 bit numbers was executed successfully by using 8085 micro processing kit.

ADDITION OF TWO 16 – BIT NUMBERS

Aim:

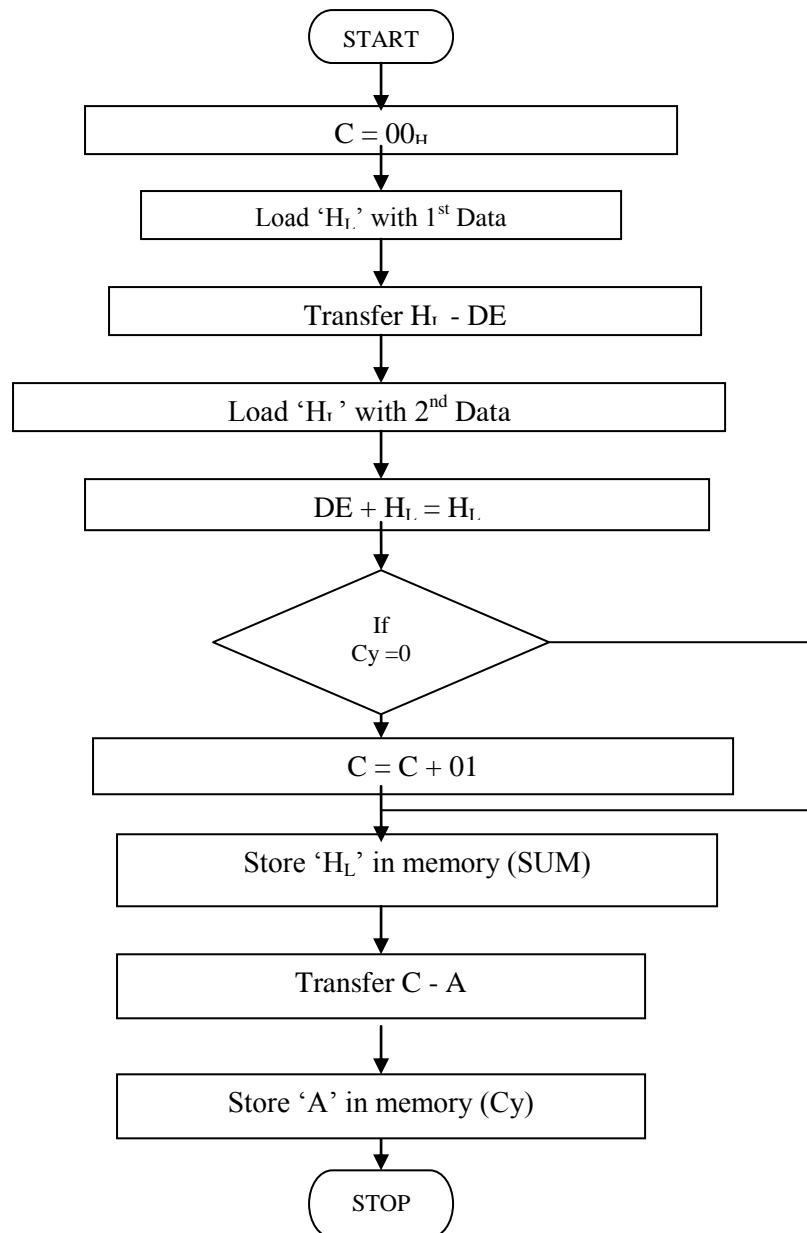
To write an assembly language program for adding two 16 bit numbers using 8085 micro processor kit.

Apparatus required:

8085 micro processor kit
(0-5V) DC battery

Algorithm:

- Step 1 : Start the microprocessor
- Step 2 : Get the 1st 8 bit in ‘C’ register (LSB) and 2nd 8 bit in ‘H’ register (MSB) of 16 bit number.
- Step 3 : Save the 1st 16 bit in ‘DE’ register pair
- Step 4 : Similarly get the 2nd 16 bit number and store it in ‘HL’ register pair.
- Step 5 : Get the lower byte of 1st number into ‘L’ register
- Step 6 : Add it with lower byte of 2nd number
- Step 7 : Store the result in ‘L’ register
- Step 8 : Get the higher byte of 1st number into accumulator
- Step 9 : Add it with higher byte of 2nd number and carry of the lower bit addition.
- Step 10 : Store the result in ‘H’ register
- Step 11 : Store 16 bit addition value in ‘HL’ register pair
- Step 12 : Stop program execution



| Address | Label | Mnemonics | Hex Code | Comments |
|---------|-------|-----------|---------------|---------------|
| 4500 | | MVI | C,00 | 0E |
| 4501 | | | 00 | |
| 4502 | | LHLD | 4800 | 2A |
| 4503 | | | | 00 |
| 4504 | | | | 48 |
| 4505 | | XCHG | | EB |
| 4506 | | LHLD | 4802 | 2A |
| 4507 | | | | 02 |
| 4508 | | | | 48 |
| 4509 | | DAD | D | 19 |
| 450A | | JNC | Ahead 450E | D2 |
| 450B | | | | 0E |
| 450C | | | | 45 |
| 450D | | INR | C | 0C |
| 450E | AHEAD | SHLD | 4804 | 22 |
| 450F | | | | 04 |
| 4510 | | | | 48 |
| 4511 | | MOV | C,A | 79 |
| 4512 | | STA | 4806 | 32 |
| 4513 | | | | 06 |
| 4514 | | | | 48 |
| 4515 | | HLT | | 76 |
| | | | | Stop excution |

Input
Without

| Input Address | Value |
|---------------|-------------|
| 4800 | 01 (addend) |
| 4801 | 04 |
| 4802 | 02 (augend) |
| 4803 | 03 (augend) |

Output

| Output Address | Value |
|----------------|------------|
| 4804 | 03 (sum) |
| 4805 | 07 (sum) |
| 4806 | 00 (carry) |

Calculation

| | | | |
|-------|------|------|------|
| 0000 | 0100 | 0000 | 0001 |
| 0000 | 0011 | 0000 | 0010 |
| ----- | | | |
| 0000 | 0111 | 0000 | 0011 |
| 0 | 7 | 0 | 3 |

With carry

| Input Address | Value |
|---------------|-------------|
| 4800 | FF (addend) |
| 4801 | DE (addend) |
| 4802 | 96 (augend) |
| 4803 | DF (augend) |

| Output Address | Value |
|----------------|------------|
| 4804 | 95 (sum) |
| 4805 | BE (sum) |
| 4806 | 01 (carry) |

Calculation

| | | | |
|-------|------|------|------|
| 1101 | 1110 | 1111 | 1111 |
| 1101 | 1111 | 1001 | 0101 |
| ----- | | | |
| 1011 | 1110 | 1001 | 0101 |
| B | E | 9 | 5 |

Result:

The assembly language program for addition of two 16 bit numbers was executed using 8085 micro processing kit.

SUBTRACTION OF TWO 16 – BIT NUMBERS

Aim:

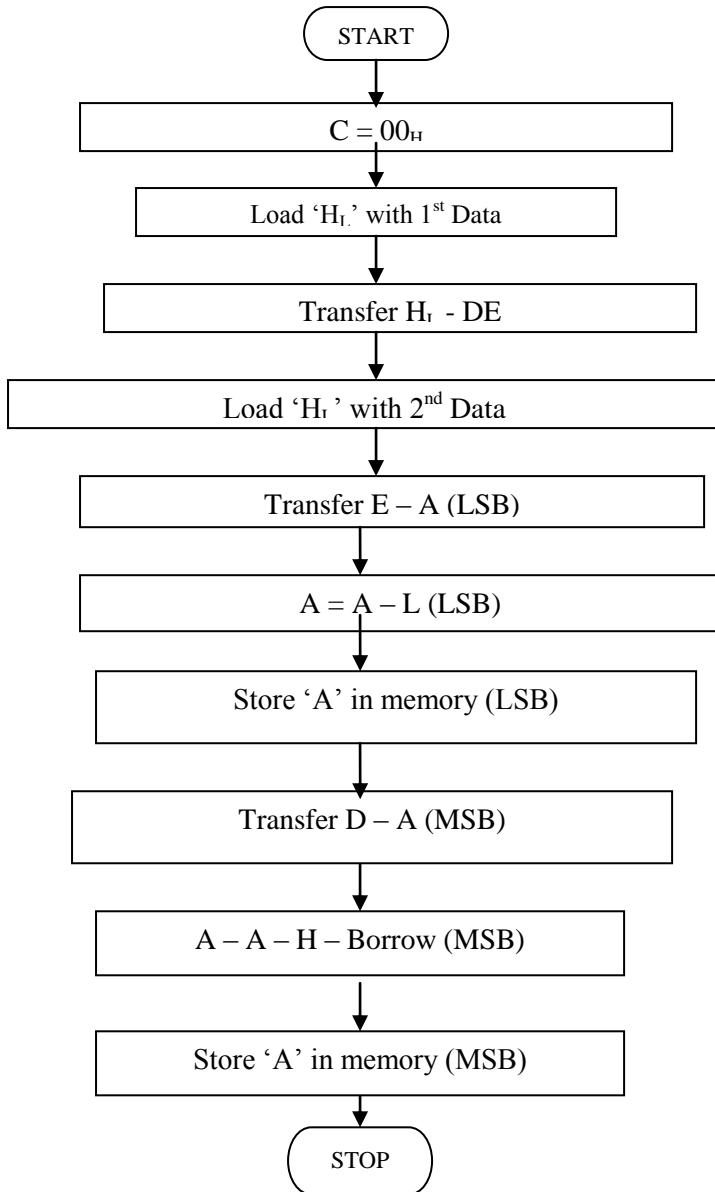
To write an assembly language program for subtracting two 16 bit numbers using 8085 microprocessor kit.

Apparatus required:

8085 microprocessor kit
(0-5V) DC battery

Algorithm:

- | | | |
|---------|---|--|
| Step 1 | : | Start the microprocessor |
| Step 2 | : | Get the 1 st 16 bit in ‘HL’ register pair |
| Step 3 | : | Save the 1 st 16 bit in ‘DE’ register pair |
| Step 4 | : | Get the 2 nd 16 bit number in ‘HL’ register pair |
| Step 5 | : | Get the lower byte of 1 st number |
| Step 6 | : | Get the subtracted value of 2 nd number of lower byte by subtracting it with lower byte of 1 st number |
| Step 7 | : | Store the result in ‘L’ register |
| Step 8 | : | Get the higher byte of 2 nd number |
| Step 9 | : | Subtract the higher byte of 1 st number from 2 nd number with borrow |
| Step 10 | : | Store the result in ‘HL’ register |
| Step 11 | : | Stop the program execution |



| Address | Label | Mnemonics | | Hex Code | Comments |
|---------|-------|-----------|------|----------|--------------------------|
| 4500 | | MVI | C,00 | 0E | C = 00 _H |
| 4501 | | | | 00 | |
| 4502 | | LHLD | 4800 | 2A | L – 1 st No. |
| 4503 | | | | 00 | |
| 4504 | | | | 48 | |
| 4505 | | XLHG | | EB | HL – DE |
| 4506 | | LHLD | 4802 | 2A | HL – 2 nd No. |
| 4507 | | | | 02 | |
| 4508 | | | | 48 | |
| 4509 | | MOV | A,E | 7B | LSB of '1' to 'A' |
| 450A | | SUB | L | 95 | A – A – L |
| 450B | | STA | 4804 | 32 | A – memory |
| 450C | | | | 04 | |
| 450D | | | | 48 | |
| 450E | | MOV | A,D | 7A | MSB of 1 to A |
| 450F | | SBB | H | 9C | A- A – H |
| 4510 | | STA | 4805 | 32 | A – memory |
| 4511 | | | | 05 | |
| 4512 | | | | 48 | |
| 4513 | | HLT | | 76 | Stop execution |

Input

Without borrow

| Input Address | Value |
|---------------|-------|
| 4800 | 07 |
| 4801 | 08 |
| 4802 | 05 |
| 4803 | 06 |

Output

| Output Address | Value |
|----------------|-------|
| 4804 | 02 |
| 4805 | 02 |
| 4807 | 00 |

With borrow

| Input Address | Value |
|---------------|-------|
| 4800 | 05 |
| 4801 | 06 |
| 4802 | 07 |
| 4803 | 08 |

| Output Address | Value |
|----------------|-------|
| 4804 | 02 |
| 4805 | 02 |
| 4806 | 01 |

Calculation

$$\begin{array}{r}
 \begin{array}{rr} 05 & 06 \end{array} \\
 - \quad \quad \quad \begin{array}{rr} 07 & 08 \end{array} \\
 \begin{array}{rr} 05 & 06 \\ \text{CMA} & \\ \text{ADI} & \end{array} \quad \begin{array}{rr} 0101 & 0110 \\ 1010 & 1001 \\ 0000 & 0001 \end{array} \quad \begin{array}{rr} 07 & 08 \\ \text{CMA} & \\ \text{ACI} & \end{array} \quad \begin{array}{rr} 0111 & 1000 \\ 1000 & 0111 \\ 0000 & 0001 \end{array} \\
 \hline
 \begin{array}{rr} 1010 & 1010 \end{array} \quad \quad \quad \begin{array}{rr} 1000 & 1000 \end{array}
 \end{array}$$

$$\begin{array}{r}
 \begin{array}{rr} 05 & 06 \end{array} \\
 + \quad \quad \quad \begin{array}{rr} 07 & 08 \end{array} \\
 \begin{array}{rr} 1010 & 1010 \\ 1000 & 1000 \end{array} \\
 \hline
 \begin{array}{rr} (1) & 0010 \\ 02 & 02 \end{array}
 \end{array}$$

Result:

The assembly language program for subtraction of two 16 bit numbers was executed by using 8085 micro processing kit.

MULTIPLICATION OF TWO 8 – BIT NUMBERS**Aim:**

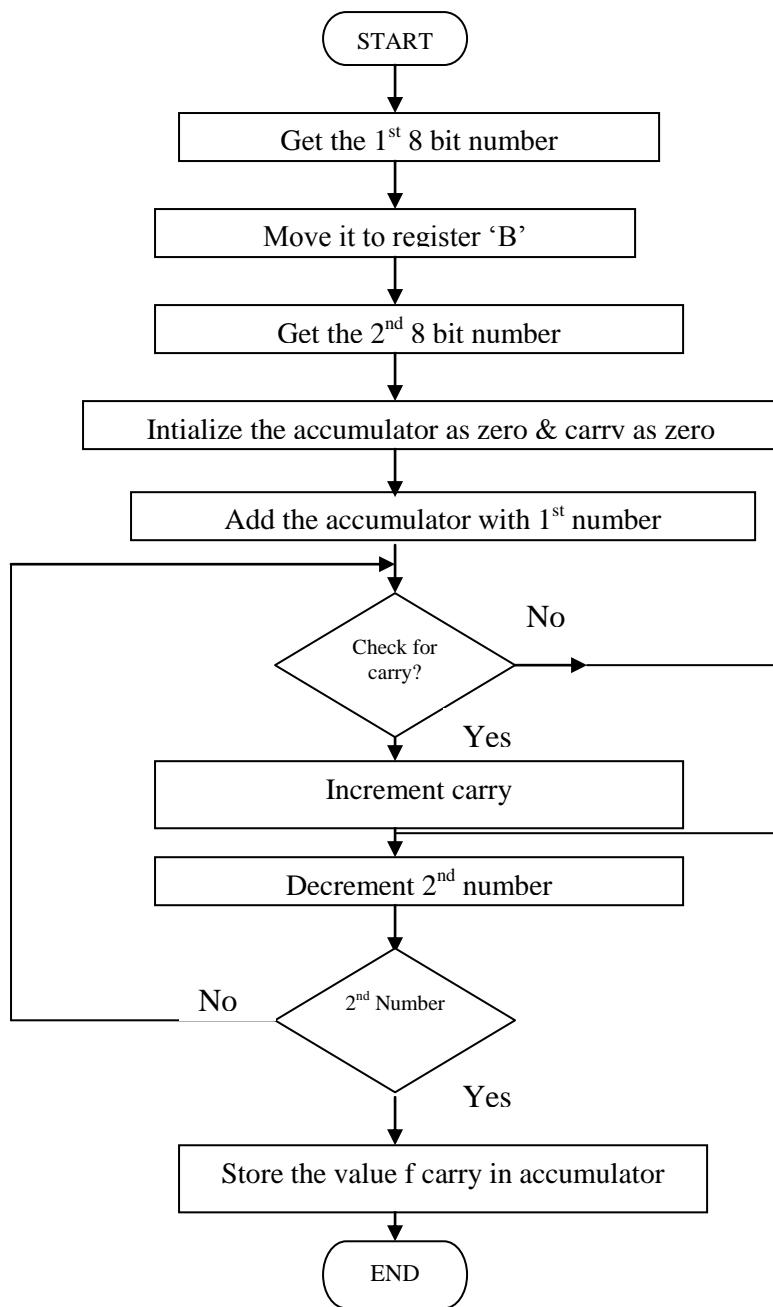
To write an assembly language for multiplying two 8 bit numbers by using 8085 micro processor kit.

Apparatus required:

8085 microprocessor kit
(0-5V) DC battery

Algorithm:

- | | | |
|---------|---|--|
| Step 1 | : | Start the microprocessor |
| Step 2 | : | Get the 1 st 8 bit numbers |
| Step 3 | : | Move the 1 st 8bit number to register ‘B’ |
| Step 4 | : | Get the 2 nd 8 bit number |
| Step 5 | : | Move the 2 nd 8 bit number to register ‘C’ |
| Step 6 | : | Intialise the accumulator as zero |
| Step 7 | : | Intialise the carry as zero |
| Step 8 | : | Add both register ‘B’ value as accumulator |
| Step 9 | : | Jump on if no carry |
| Step 10 | : | Increment carry by 1 if there is |
| Step 11 | : | Decrement the 2 nd value and repeat from step 8, till the 2 nd value becomes zero. |
| Step 12 | : | Store the multiplied value in accumulator |
| Step 13 | : | Move the carry value to accumulator |
| Step 14 | : | Store the carry value in accumulator |



| Address | Label | Mnemonics | Hex Code | Comments |
|---------|-------|-----------|------------|---|
| 4100 | | LDA 4500 | 3A, 00, 45 | Load the first 8 bit number |
| 4103 | | MOV B,A | 47 | Move the 1 st 8 bit data to register 'B' |
| 4104 | | LDA 4501 | 3A, 01, 45 | Load the 2 nd 16 it number |
| 4107 | | MOV C,A | 4F | Move the 2 nd 8 bit data to register 'C' |
| 4108 | | MVI A, 00 | 3E, 00 | Intialise the accumulator as zero |
| 410A | | MVI D, 00 | 16, 00 | Intialise the carry as zero |
| 410C | | ADD B | 80 | Add the contents of 'B' and accumulator |
| 410D | | INC | D2 11, 41 | Jump if no carry |
| 4110 | | INR D | 14 | Increment carry if there is |
| 4111 | | DCR C | OD | Decrement the value 'C' |
| 4112 | | JNZ | C2 0C, 41 | Jump if number zero |
| 4115 | | STA 4502 | 32 02, 45 | Store the result in accumulator |
| 4118 | | MOV A,D | 7A | Move the carry into accumulator |
| 4119 | | STA 4503 | 32,03,45 | Store the result in accumulator |
| 411C | | HLT | 76 | Stop the program execution |

Input

| Input Address | Value |
|---------------|-------|
| 4500 | 04 |
| 4501 | 02 |

Output

| Output Address | Value |
|----------------|-------|
| 4502 | 08 |
| 4503 | 00 |

Result:

The assembly language program for multiplication of two 8 bit numbers was executed using 8085 micro processing kit.

DIVISION OF TWO 8 – BIT NUMBERS

Aim:

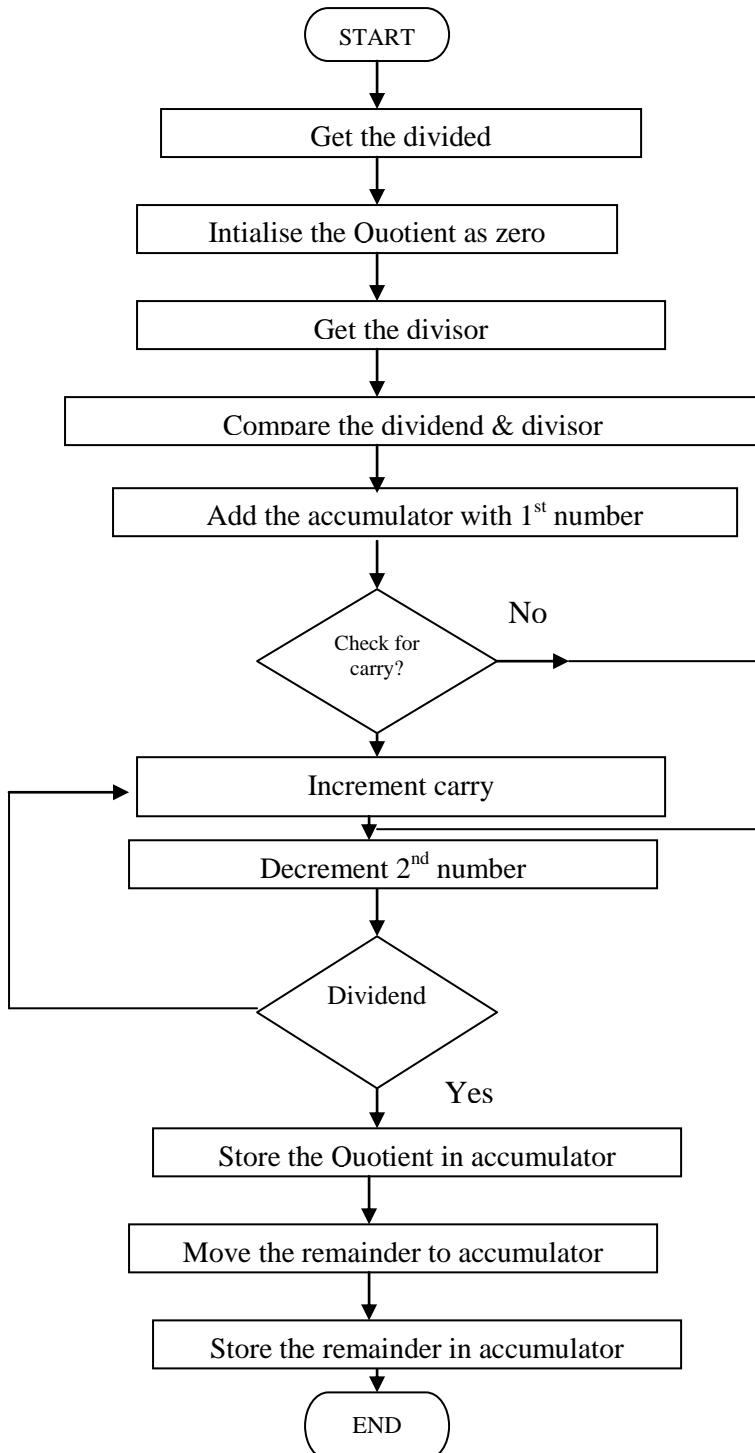
To write an assembly language program for dividing two 8 bit numbers using microprocessor kit.

Apparatus required:

8085 microprocessor kit
(0-5V) DC battery

Algorithm:

- | | | |
|---------|---|--|
| Step 1 | : | Start the microprocessor |
| Step 2 | : | Intialise the Quotient as zero |
| Step 3 | : | Load the 1 st 8 bit data |
| Step 4 | : | Copy the contents of accumulator into register ‘B’ |
| Step 5 | : | Load the 2 nd 8 bit data |
| Step 6 | : | Compare both the values |
| Step 7 | : | Jump if divisor is greater than dividend |
| Step 8 | : | Subtract the dividend value by divisor value |
| Step 9 | : | Increment Quotient |
| Step 10 | : | Jump to step 7, till the dividend becomes zero |
| Step 11 | : | Store the result (Quotient) value in accumulator |
| Step 12 | : | Move the remainder value to accumulator |
| Step 13 | : | Store the result in accumulator |
| Step 14 | : | Stop the program execution |



| Address | Label | Mnemonics | Hex Code | Comments |
|---------|--------|---------------|------------|---|
| 4100 | | MVI C, 00 | 0E, 00 | Initialise Quotient as zero |
| 4102 | | LDA, 4500 | 3A 00, 45 | Get the 1 st data |
| 4105 | | MOV B,A | 47 | Copy the 1 st data into register 'B' |
| 4106 | | LDA, 4501 | 3A 01, 45 | Get the 2 nd data |
| 4109 | | CMP B | B8 | Compare the 2 values |
| 410A | | JC (LDP) | DA 12,41 | Jump if dividend lesser than divisor |
| 410D | Loop 2 | SUB B | 90 | Subtract the 1 st value by 2 nd value |
| 410E | | INR C | 0C | Increment Quotient (410D) |
| 410F | | JMP (LDP, 41) | C3, 0D, 41 | Jump to Loop 1 till the value of dividend becomes zero |
| 4112 | Loop 1 | STA 4502 | 32 02,45 | Store the value in accumulator |
| 4115 | | MOV A,C | 79 | Move the value of remainder to accumulator |
| 4116 | | STA 4503 | 32 03,45 | Store the remainder value in accumulator |
| 4119 | | HLT | 76 | Stop the program execution |

Input

| Input Address | Value |
|---------------|-------|
| 4500 | 09 |
| 4501 | 02 |

Output

| Output Address | Value |
|----------------|----------------|
| 4502 | 04 (quotient) |
| 4503 | 01 (remainder) |

Quotient
Carry

1001
0010 – I

0111
0010 – II

0101
0010 – III

0011
0010 – IV

0001 – carry

- 04
- 01

Result:

The assembly language program for division of two 8 bit numbers was executed using 8085 micro processing kit.

ASCENDING ORDER

Aim:

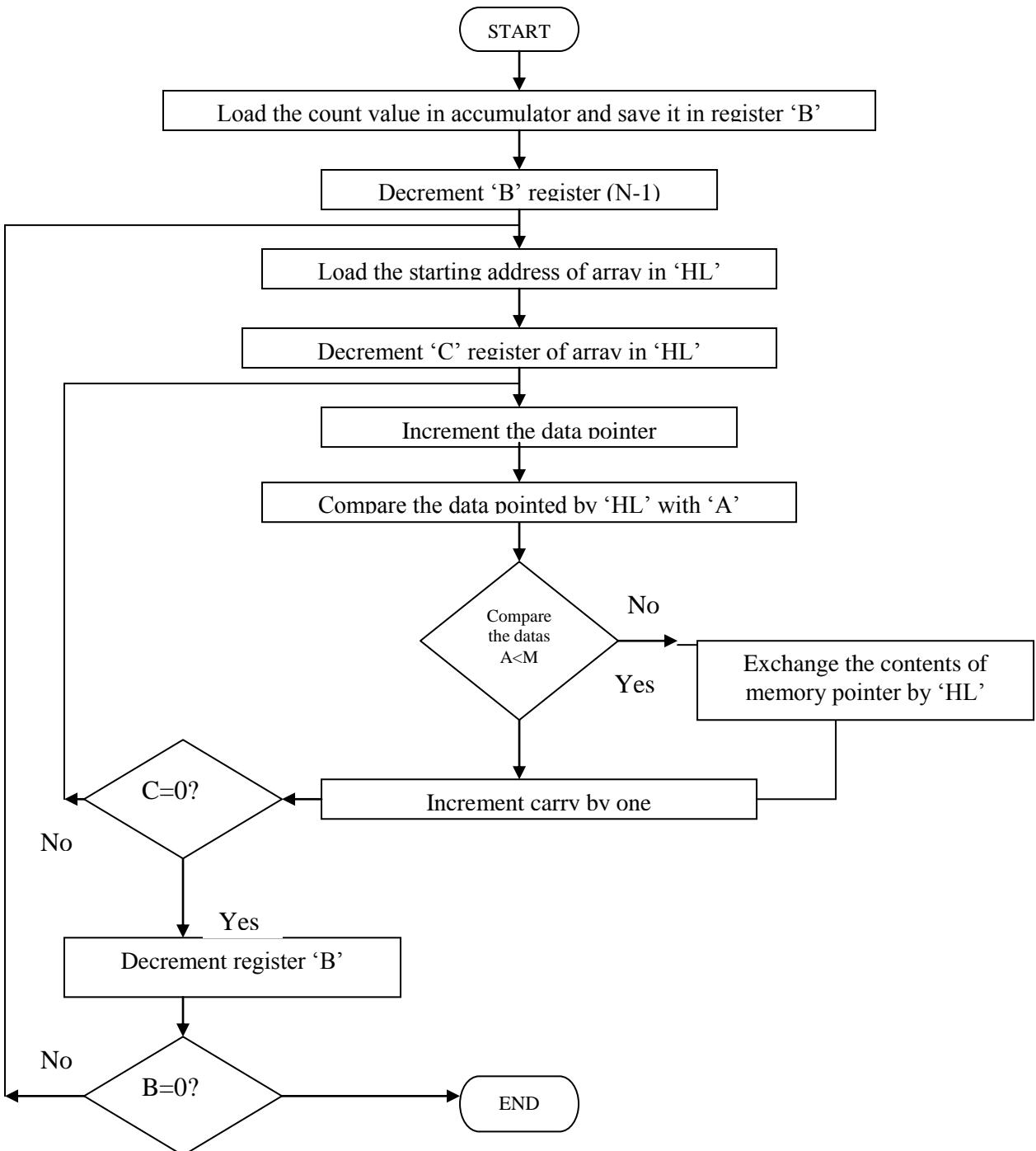
To write a program to sort given ‘n’ numbers in ascending order

Apparatus required:

8085 microprocessor kit
(0-5V) DC battery

Algorithm:

- | | | |
|---------|---|---|
| Step 1 | : | Start the microprocessor |
| Step 2 | : | Accumulator is loaded with number of values to sorted and it is saved |
| Step 3 | : | Decrement 8 register (N-1) Repetitions |
| Step 4 | : | Set ‘HL’ register pair as data array |
| Step 5 | : | Set ‘C’ register as counter for (N-1) repetitions |
| Step 6 | : | Load a data of the array in accumulator |
| Step 7 | : | Compare the data pointed in ‘HL’ pair |
| Step 8 | : | If the value of accumulator is smaller than memory, then jump to step 10. |
| Step 9 | : | Otherwise exchange the contents of ‘HL’ pair and accumulator |
| Step 10 | : | Decrement ‘C’ register, if the value of ‘C’ is not zero go to step 6 |
| Step 11 | : | Decrement ‘B’ register, if value of ‘B’ is not zero, go step 3 |
| Step 12 | : | Stop the program execution |



| Address | Label | Mnemonics | Hex Code | Comments |
|---------|--------|-------------|------------|---|
| 4100 | | LDA 4500 | 3A, 00,45 | Load the number of values |
| 4103 | | MOV B,A | 47 | Move it 'B' register |
| 4104 | | DCR B | 05 | For (N-1) comparisons |
| 4105 | Loop 3 | LXI H, 4500 | 21, 00,45 | Set the pointer for array |
| 4108 | | MOV C,M | 4E | Count for (N-1) comparisons |
| 4109 | | DCR C | 0D | For (N-1) comparisons |
| 410A | | INX H | 23 | Increment pointer |
| 410B | Loop 2 | MOV A,M | 7E | Get one data in array 'A' |
| 410C | | INX H | 23 | Increment pointer |
| 410D | | CMP M | BE | Compare next with accumulator |
| 410E | | JC | DA, 16, 41 | If content less memory go ahead |
| 4111 | | MOV D,M | 56 | If it is greater than interchange it |
| 4112 | | MOV M,A | 77 | Memory content |
| 4113 | | DCX H | 2B | Exchange the content of memory pointed by 'HL' by previous location |
| 4114 | | MOV M,D | 72 | One in by 'HL' and previous location |
| 4115 | | INX H | 23 | Increment pointer |
| 4116 | Loop 1 | DCR C | 0D | Decrement 'C' register |
| 4117 | | JNZ Loop 1 | C2, 0B, 41 | Repeat until 'C' is zero |
| 411A | | DCR B | 05 | Decrement in 'B' values |
| 411B | | JNZ Loop 2 | C2, 05, 41 | Repeat till 'B' is zero |
| 411E | | HLT | 76 | Stop the program execution |

Input

| Input Address | Value |
|---------------|-------|
| 4500 | 04 |
| 4501 | AB |
| 4502 | BC |
| 4503 | 01 |
| 4504 | 0A |

Output Address & Value

| Output Address | Value |
|----------------|-------|
| 4500 | 04 |
| 4501 | 01 |
| 4502 | 0A |
| 4503 | AB |
| 4504 | BC |

Result:

The assembly language program for sorting numbers in ascending order was executed by microprocessor kit.

DESCENDING ORDER**Aim:**

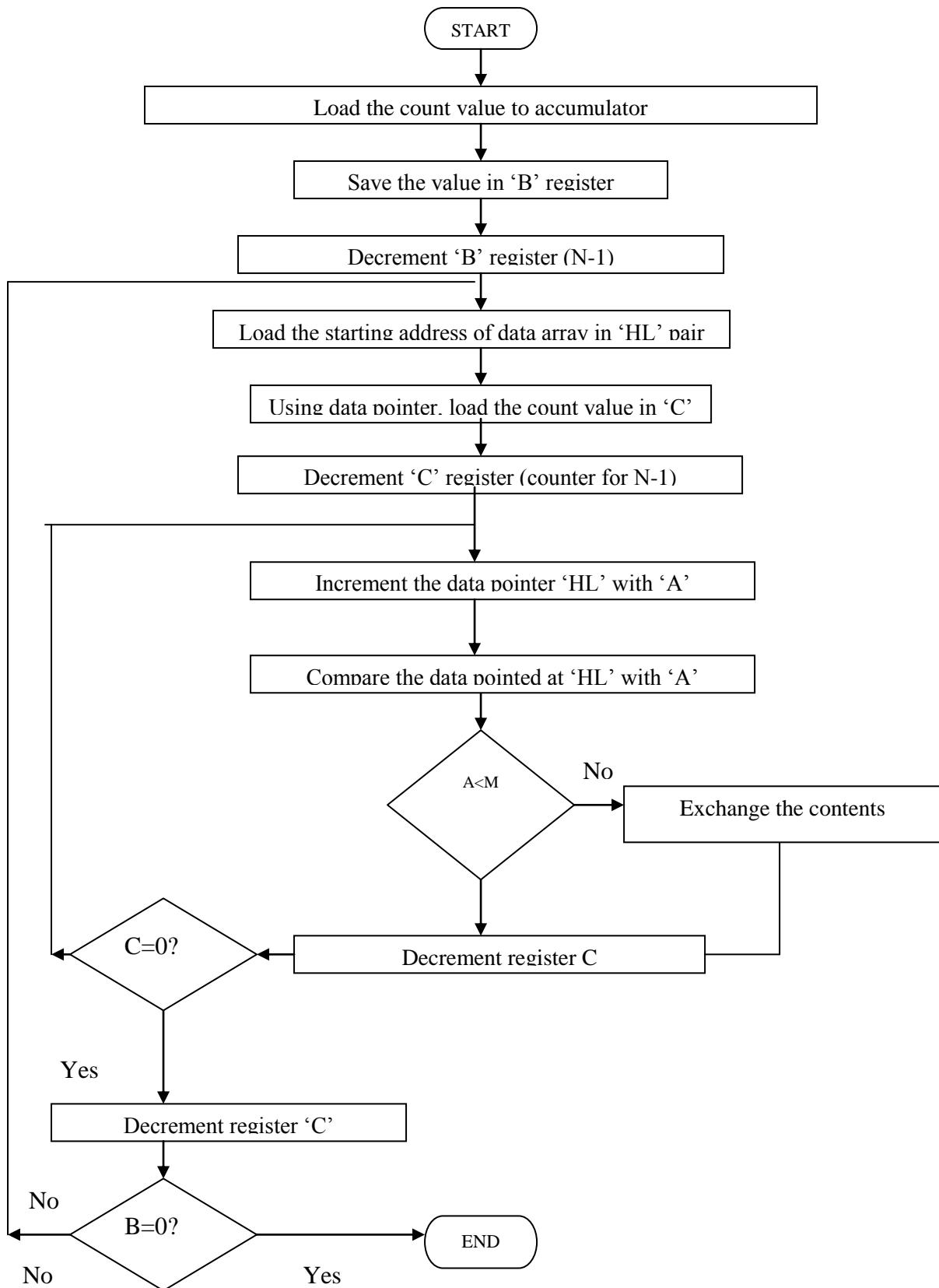
To write a program to sort given ‘n’ numbers in descending order

Apparatus required:

8085 microprocessor kit
(0-5V) DC battery

Algorithm:

- Step 1 : Start the microprocessor
- Step 2 : Load the number of values into accumulator and save the number of values in register ‘B’
- Step 3 : Decrement register ‘B’ for (N-1) Repetitions
- Step 4 : Set ‘HL’ register pair as data array address pointer and load the data of array in accumulator
- Step 5 : Set ‘C’ register as counter for (N-1) repetitions
- Step 6 : Increment ‘HL’ pair (data address pointer)
- Step 7 : Compare the data pointed by ‘HL’ with accumulator
- Step 8 : If the value of accumulator is larger than memory, then jump to step 10, otherwise next step.
- Step 9 : Exchange the contents of memory pointed by ‘HL’ and accumulator
- Step 10 : Decrement ‘C’ register, if the of ‘C’ is not zero go to step 6, otherwise next step.
- Step 11 : Decrement ‘B’ register, if ‘B’ is not zero, go step 3, otherwise next step.
- Step 12 : Stop the program execution



| Address | Label | Mnemonics | Hex Code | Comments |
|---------|--------|-------------|------------|---|
| 4100 | | LDA 4500 | 3A, 00,45 | Load the number of values in accumulator |
| 4103 | | MOV B,A | 47 | Move it to 'B' register |
| 4104 | | DCR B | 05 | For (N-1) comparisons |
| 4105 | Loop 3 | LXI H, 4500 | 21, 00,45 | Set the pointer for array |
| 4108 | | MOV C,M | 4E | Count for (N-1) comparisons |
| 4109 | | DCR C | 0D | For (N-1) comparisons |
| 410A | | INX H | 23 | Increment pointer |
| 410B | Loop 2 | MOV A,M | 7E | Get one data from array |
| 410C | | INX H | 23 | Increment pointer |
| 410D | | CMP M | BE | Compare next with number |
| 410E | | ICE, Loop 1 | D2, 16,41 | If content 'A' is greater than content of 'HL' pair |
| 4111 | | MOV D,M | 56 | If it is greater than interchange the datas |
| 4112 | | MOV M,A | 77 | Accumulator to memory value |
| 4113 | | DCX H | 2B | Decrement memory pointer |
| 4114 | | MOV M,D | 72 | Move the old to 'HL' and previous location |
| 4115 | | INX H | 23 | Increment pointer |
| 4116 | Loop 1 | DCR C | 0D | Decrement 'C' register |
| 4117 | | JNZ Loop 2 | C2, 0B, 41 | Repeat till 'C' is zero |
| 411A | | DCR B | 05 | Decrement in 'B' values |
| 411B | | JNZ Loop 3 | C2, 05, 41 | Jump to loop till the value of 'B' be |
| 411E | | HLT | 76 | Stop the program execution |

Input

| Input Address | Value |
|---------------|-------|
| 4500 | 04 |
| 4501 | AB |
| 4502 | BC |
| 4503 | 01 |
| 4504 | 0A |

Output Address & Value

| Output Address | Value |
|----------------|-------|
| 4500 | 04 |
| 4501 | BC |
| 4502 | AB |
| 4503 | 0A |
| 4504 | 01 |

Result:

The assembly language program for sorting '4' numbers in descending order was executed successfully using microprocessor kit.

SUM OF DATAS

Aim:

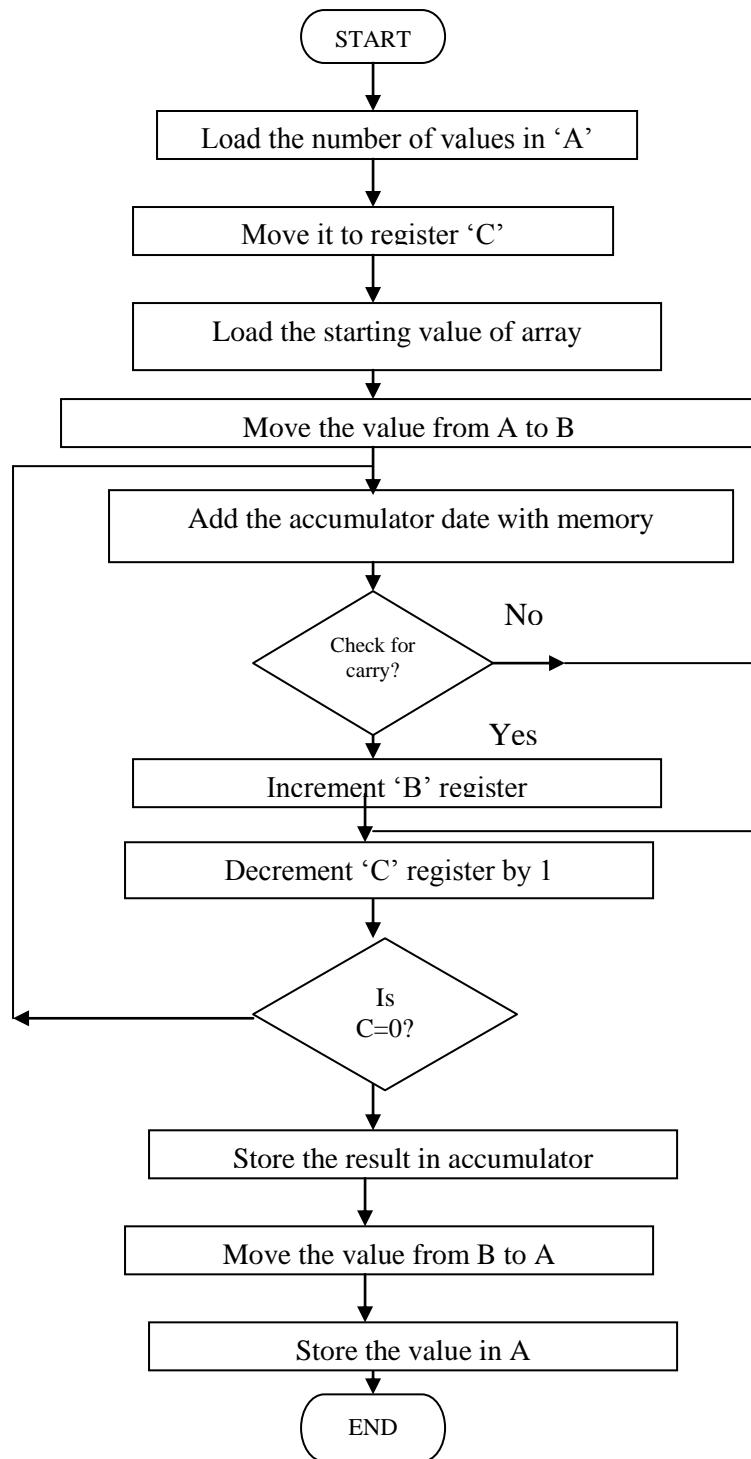
To write an assembly language program to calculate the sum of datas using 8085 microprocessor kit.

Apparatus required:

8085 microprocessor kit
(0-5V) DC battery

Algorithm:

- Step 1 : Start the microprocessor
- Step 2 : Load the number of values in series in accumulator and move it to register C and load the starting address of array
- Step 3 : Initialize the value of A as '00'
- Step 4 : Move the value of 'A' to 'B' register
- Step 5 : Add the content of accumulator with the data pointed by 'HL' pair
- Step 6 : If there exists a carry, increment 'B' by 1, if not continue
- Step 7 : Increment the pointer to next data
- Step 8 : Decrement the value of 'C' by 1, which is used as counter
- Step 9 : If 'C' is equal to zero, go to step 10 if not go to step 5.
- Step 10 : Store the value of 'A' to memory, it shows the result
- Step 11 : Move the content of B to A
- Step 12 : Store the value of A to memory
- Step 13 : Stop the program



| Address | Label | Mnemonics | Hex Code | Comments |
|---------|-------|-------------|------------|--|
| 4100 | | LDA 4200 | 3A 00,42 | Load the accumulator with number of values |
| 4103 | | MOV B,A | 4F | Move it from A to C |
| 4104 | | LXI H, 4201 | 21,01,42 | Load the starting address of data array |
| 4107 | | SUB A | 97 | Initialise 'A' as 00 |
| 4108 | | MOV B,A | 47 | Initialise 'B' as 00 |
| 4109 | Loop | ADD M | 86 | Add the previous sum with next data |
| 410A | | JNC Skip | D2, 0E, 41 | Jump on if no carry |
| 410D | | INR B | 04 | Increment carry by one |
| 410E | Skip | INX H | 23 | Increment pointer for next data |
| 410F | | DCR C | 0D | Decrement 'C' by one |
| 4110 | | JNZ Loop | C2, 09, 41 | Jump if not zero |
| 4113 | | STA 4400 | 32,00,44 | Store the sum in accumulator |
| 4116 | | MOV A,B | 78 | Move the value of carry to A from B |
| 4117 | | STA 4401 | 32,01,44 | Store the carry in memory |
| 411A | | HLT | 76 | End of program |

Input

| Input Address | Value |
|---------------|-------|
| 4200 | 04 |
| 4201 | 07 |
| 4202 | 09 |
| 4203 | 03 |
| 4204 | 04 |

Output

| Output Address | Value |
|----------------|-------|
| 4400 | 17 |
| 4401 | 00 |

$$07 + 09 + 03 + 04 = 23$$

$$= 17 \text{ (in Hexa decimal)} \\ (0F + 8 = 233)$$

$$\begin{array}{rcl}
 0F & = & 0000 \quad 1111 \\
 08 & = & 0000 \quad 1000 \\
 \hline
 & & 0001 \quad 0111 \\
 & & 1 \quad \quad 7
 \end{array}$$

Result:

The assembly language program for sum of datas was executed successfully using 8085 microprocessor kit.

FACTORIAL OF 8 BIT NUMBER

Aim:

To write a program to calculate the factorial of a number (between 0 to 8)

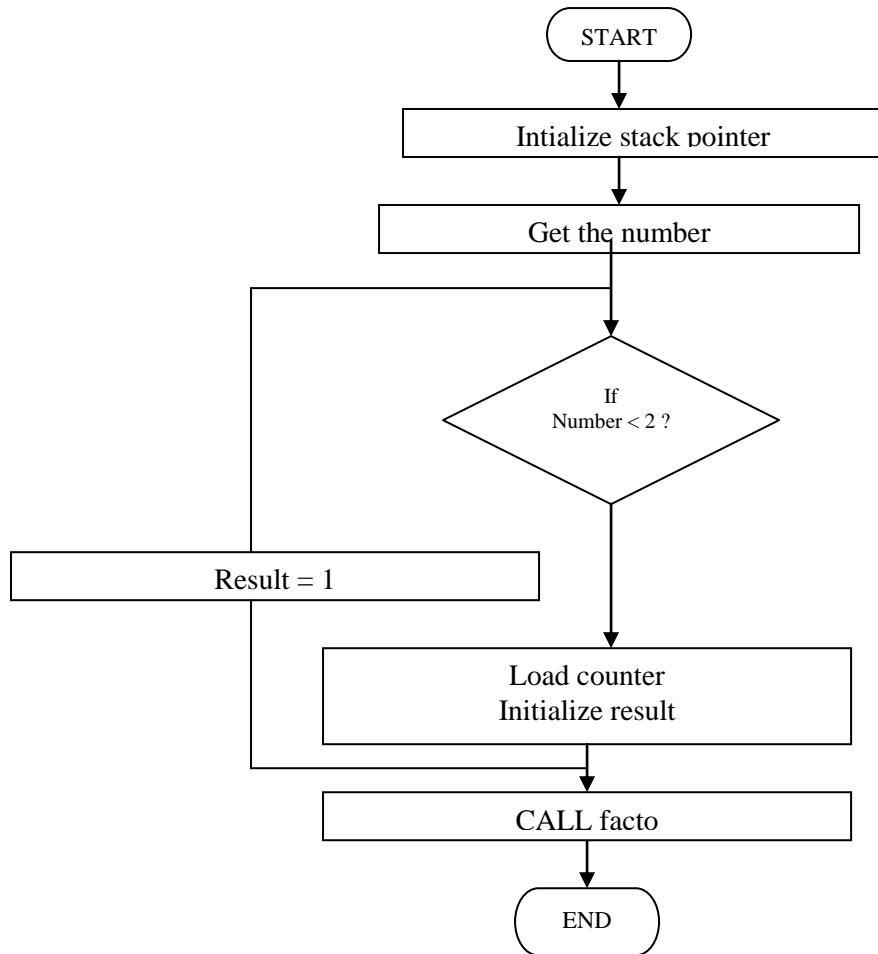
Apparatus required:

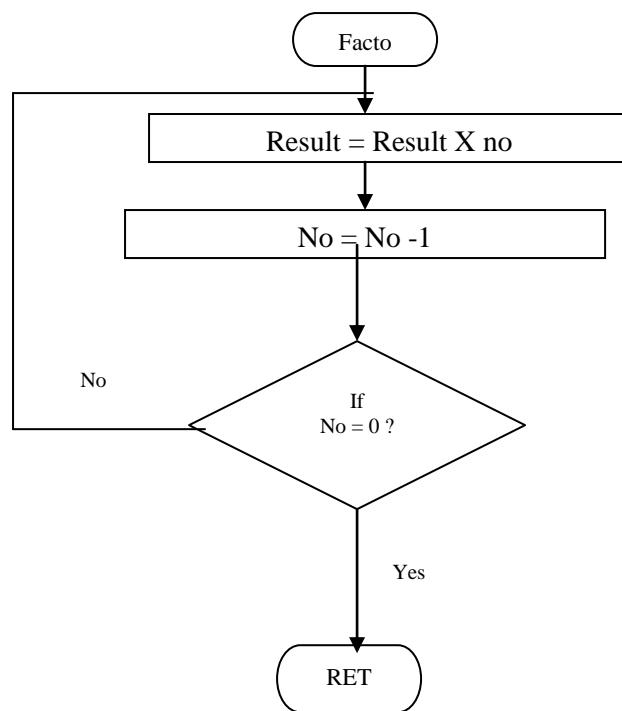
8085 microprocessor kit
(0-5V) power supply

Algorithm:

- Step 1 : Initialize the stack pointer
- Step 2 : Get the number in accumulator
- Step 3 : Check for if the number is greater than 1. If no store the result otherwise go to next step.
- Step 4 : Load the counter and initialize result
- Step 5 : Now factorial program in sub-routine is called.
- Step 6 : In factorial,
initialize H_L RP with 0.
Move the count value to B
Add H_L content with R_p.
Decrement count (for multiplication)
- Step 7 : Exchange content of Rp (DE) with HL.
- Step 8 : Decrement counter (for factorial) till zero flag is set.
- Step 9 : Store the result
- Step 10 : Hault

| Memory address | Content |
|----------------|----------------------|
| 4250 | 05 |
| 4251 | (120 ₁₀) |





| Memory Location | Hex Code | Label | Mnemonics | | Comments |
|--------------------|----------|--------|-----------|---------------------|---|
| | | | Op code | Operand | |
| 4200 | 3A | | LDA | 4250 | Get the number in accumulator |
| 4201 | 50 | | | | |
| 4202 | 42 | | | | |
| 4203 | FE | | CPI | 02H | Compare data with 2 and check it is greater than 1 |
| 4204 | 02 | | | | |
| 4205 | DA | | JC | Loop 1 | If cy =1 jump to loop 1 If cy = 0 proceed |
| 4206 | 17 | | | | |
| 4207 | 42 | | | | |
| 4208 | 5F | | MOV | E,A | Move content of A to E |
| 4209 | 16 | | MVI | D,00 | Load this term as a result |
| 420A | 00 | | | | |
| 420B | 3D | | DCR | A | Decrement accumulator by 1 |
| 420C | 4F | | MOV | C,A | Move 'A' content to 'C' (counter 1 less than A) |
| 420D | CD | | CALL | Facto | Call sub routine programme Facto |
| 420E | 00 | | | | |
| 420F | 46 | | | | |
| 4210 | EB | | XCHG | | Exchange (DE) – (HL) |
| 4211 | 22 | | SHLD | 4251 | Store content of HL in specified memory location |
| 4212 | 51 | | | | |
| 4213 | 42 | | | | |
| 4214 | C3 | | JMP | Loop 3 | Jump to Loop 3 |
| 4215 | 1D | | | | |
| 4216 | 42 | | | | |
| 4217 | 21 | Loop 1 | LXI | H,0001 _H | HL is loaded with data 01 |
| 4218 | 00 | | | | |
| 4219 | 01 | | | | |
| 421A | 22 | | SHLD | 4251 | Store the result in memory |
| 421B | 51 | | | | |
| 421C | 42 | | | | |
| 421D | 76 | Loop 3 | HLT | | Terminate the program |
| Sub Routine | | | | | |
| 4600 | 21 | Facto | LXI | H,0000 | Initialize HL pair |
| 4601 | 00 | | | | |
| 4602 | 00 | | | | |
| 4603 | 41 | | MOV | B,C | Content of 'C' is moved to B |
| 4604 | 19 | Loop 2 | DAD | D | Content of DE is added with HL |
| 4605 | 05 | | | | |
| 4606 | C2 | | DCR | B | 'B' is decremented |
| 4607 | 04 | | JNZ | Loop 2 | Multiply by successive addition till zero flag is set |
| 4608 | 46 | | | | |

| | | | | | |
|------|----|--|------|-------|--|
| 4609 | EB | | XCHG | | [DE] – [HL] |
| 460A | 0D | | DCR | C | Decrement counter value |
| 460B | C4 | | CNZ | Facto | Call on no zero to facto (i.e repeat process till zero flag for c = 1) |
| 460C | 00 | | | | |
| 460D | 46 | | | | |
| 460E | C9 | | RET | | Return to main program |

| Memory address | Content |
|----------------|---------|
| 4250 | 04 |
| 4251 | 18 |

$$1 \times 2 \times 3 \times 4 = 24$$

Hexadecimal

$$16 \overline{)24} \\ 1-8$$

Result:

Thus, factorial program was done successfully

FIBANOCCI SERIES

Aim:

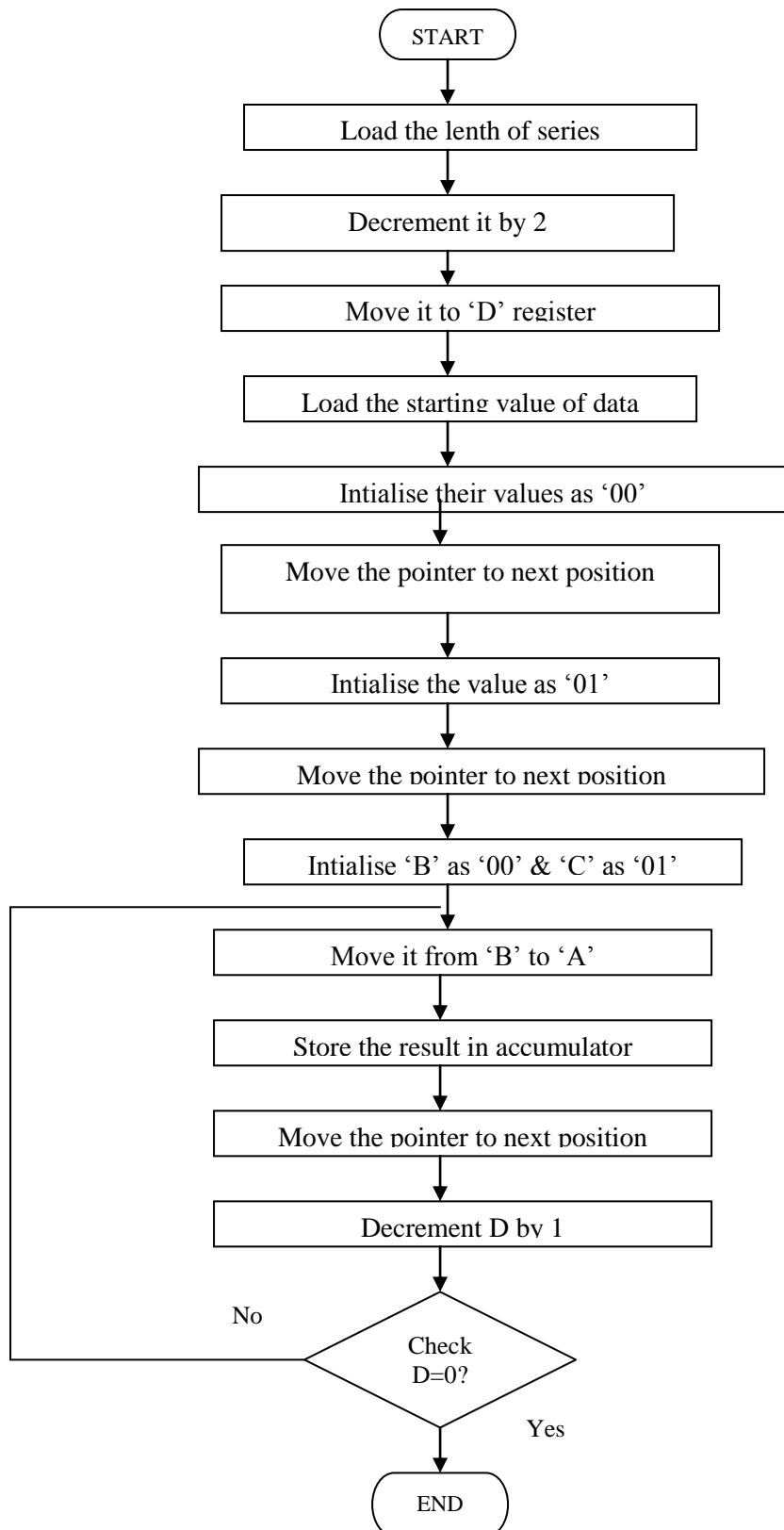
To write an assembly language program to dispel Fibanocci Series.

Apparatus required:

8085 microprocessor kit
(0-5V) DC battery

Algorithm:

- | | | |
|---------|---|--|
| Step 1 | : | Start the microprocessor |
| Step 2 | : | Load the length of series in the accumulator and decrement it by 2 |
| Step 3 | : | Move the value to register 'D' |
| Step 4 | : | Load the starting value of data value address |
| Step 5 | : | Initialise the 1 st number as 00 |
| Step 6 | : | Move the pointer to 2 nd data and initialise them as '01' |
| Step 7 | : | Move the pointer to next position for next data |
| Step 8 | : | Initialise B as '00' and C as '01' for calculations |
| Step 9 | : | Copy the contents of 'B' to accumulator |
| Step 10 | : | Add the content of 'C' register to accumulator |
| Step 11 | : | Move the content 'C' to 'B' and 'A' to C |
| Step 12 | : | Now store the result to memory pointed by 'HL' pair |
| Step 13 | : | Move the pointer to next pointer |
| Step 14 | : | Decrement 0 by 1 for counter |
| Step 15 | : | If 'D' is not zero, go to step 9 |
| Step 16 | : | if 'D' is zero, end the program |



| Address | Label | Mnemonics | Hex Code | Comments |
|---------|-------|-------------|------------|------------------------------------|
| 4200 | | LDA 4300 | 3A, 00, 43 | Store the length of series in 'A' |
| 4203 | | SUI 02 | D6, 02 | Decrement 'A' by 02 |
| 4205 | | MOV D,A | 57 | Move 'A' to 'D' (counter) |
| 4206 | | LXI H, 4301 | 21,01,43 | Load the starting address of array |
| 4209 | | MVI M,00 | 36,00 | Intialise 4301 as '00' |
| 420B | | INX H | 23 | Increment pointer |
| 420C | | MVI M, 01 | 36,01 | Initialize 2 nd as '01' |
| 420E | | INX H | 23 | Increment pointer |
| 420F | | MVI B,00 | 06,00 | Intialise 'B' as '00' |
| 4211 | | MVI, C, 01 | 0E, 01 | Intialise 'C' as '01' |
| 4213 | Loop | MOV A,B | 78 | Move B to A |
| 4214 | | ADD C | 81 | Add 'A' and 'C' |
| 4215 | | MOV B,C | 41 | Move C to B |
| 4216 | | MOV C,A | 4F | Move A to C |
| 4217 | | MOV M,A | 77 | Move the result to memory |
| 4218 | | INX H | 23 | Increment pointer |
| 4219 | | DCR D | 15 | Decrement counter |
| 421A | | JNZ loop | C2, 13,42 | If D = 0, jump to loop |
| 421D | | HLT | 76 | Stop the program |

Input

| Input Address | Value |
|---------------|-------|
| 4300 | 05 |

Output

| Output Address | Value |
|----------------|-------|
| 4301 | 00 |
| 4302 | 01 |
| 4303 | 01 |
| 4304 | 02 |
| 4305 | 03 |

$$00 + 01 = 01$$

$$01 + 01 = 02$$

$$02 + 01 = 03$$

Result:

The assembly language for Fibonacci series was executed successfully using 8085 microprocessor kit.

16 – BIT MULTIPLICATION

Aim:

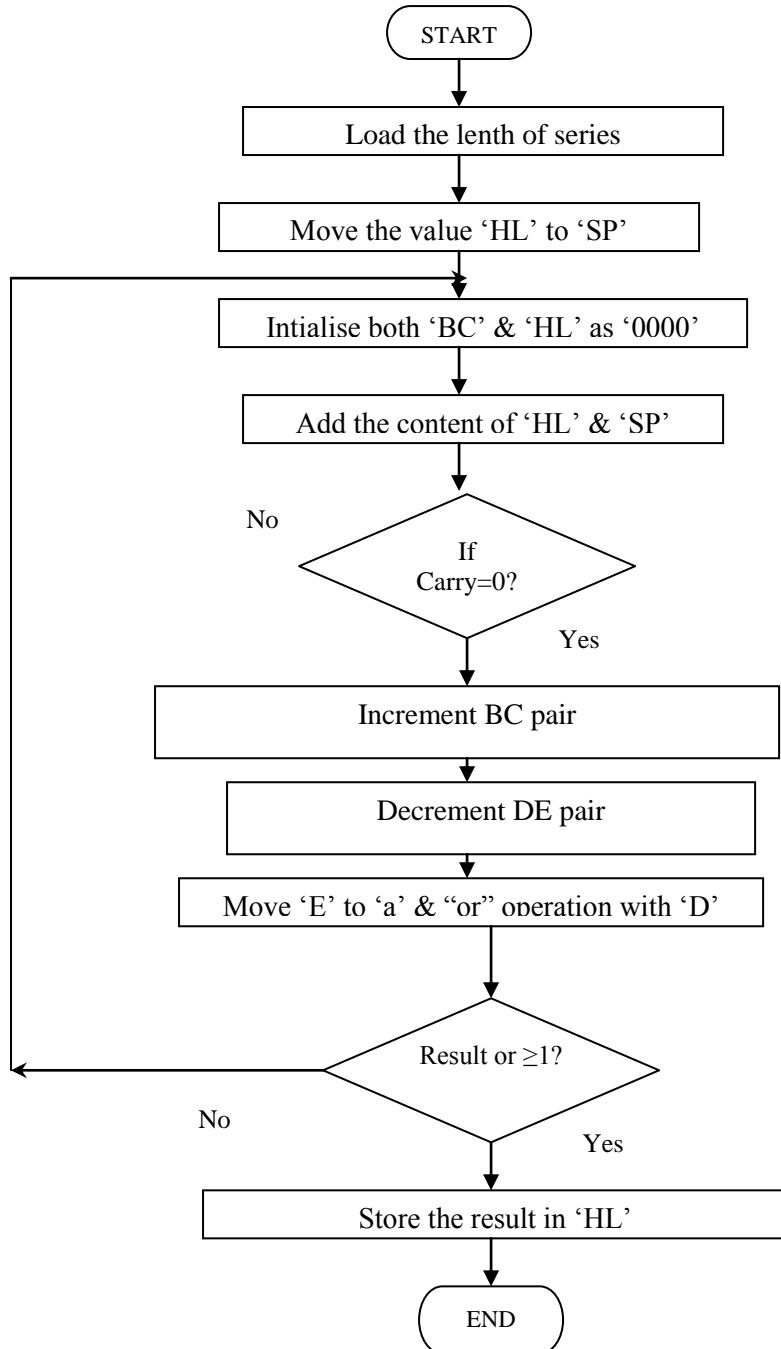
To write an assembly language program for 16 bit multiplication by using 8085 microprocessor kit.

Apparatus required:

8085 microprocessor kit
(0-5V) DC battery

Algorithm:

- Step 1 : Start the microprocessor
- Step 2 : Load the 1st data in ‘HL’ register pair
- Step 3 : Move content of ‘HL’ pair to stack pointer
- Step 4 : Load the 2nd data in ‘HL’ and move it to ‘DE’
- Step 5 : Make ‘HL’ pair as ‘00’ and ‘00’
- Step 6 : Add ‘HL’ pair and ‘SP’
- Step 7 : Check for carry condition, if carry is present increment it by one else move to next step.
- Step 8 : Decrement DE register
- Step 9 : Then move E to ‘A’ and perform ‘OR’ operation with ‘a’ and ‘D’
- Step 10 : The value of operation is zero, then store the value else go to step 3
- Step 11 : Stop the program



| Memory Location | Hex Code | Label | Mnemonics | | Comments |
|-----------------|------------|-------|-----------|---------|------------------------------------|
| | | | Op code | Operand | |
| 4100 | 2A,00,42 | | LHLD | 4200 | Get the 1 st data in HL |
| 4103 | F9 | | SP HL | | Save it in stack pointer |
| 4106 | 2A,02,42 | | LHLD | 4202 | Get the 2 nd data in HL |
| 4107 | EB | | XCHG | | Exchange 'HL' and 'DC' |
| 4108 | 21,00,00 | | LXI H | 0000 | Make HL – 0000 |
| 410B | 01,00,00 | | LXI B | 0000 | Make BC – 0000 |
| 410E | 39 | Next | DAD | SP | Add 'SP' and 'HL' |
| 410F | D2, 13, 41 | | JNC | Loop | Jump to loop if no carry |
| 4112 | 03 | | INX | B | Increment 'BC' by one |
| 4113 | 1B | Loop | DCX | D | Decrement 'DE' by one |
| 4114 | 7B | | MOV | A,E | Make E – A |
| 4115 | B2 | | ORA | D | 'OR' gate between A & D |
| 4116 | C2,0E,41 | | JNZ | Next | Jump on if number zero |
| 4119 | 22,04,42 | | SHLD | 4204 | Store the LSB in memory |
| 411C | 69 | | MOV | L,C | Make C to L |
| 411D | 60 | | MOV | H,B | Make B to H |
| 411E | 22,06,42 | | SHLD | 4206 | Store the MSB in memory |
| 4121 | 76 | | HLT | | Stop the program |

Input

| Input Address | Value |
|---------------|-------|
| 4200 | 04 |
| 4201 | 07 |
| 4202 | 02 |
| 4203 | 01 |

Output

| Output Address | Value |
|----------------|-------|
| 4204 | 08 |
| 4205 | 12 |
| 4206 | 01 |
| 4207 | 00 |

Result:

Thus the assembly language program for 16 bit multiplication was executed successfully.

16 – BIT DIVISION**Aim:**

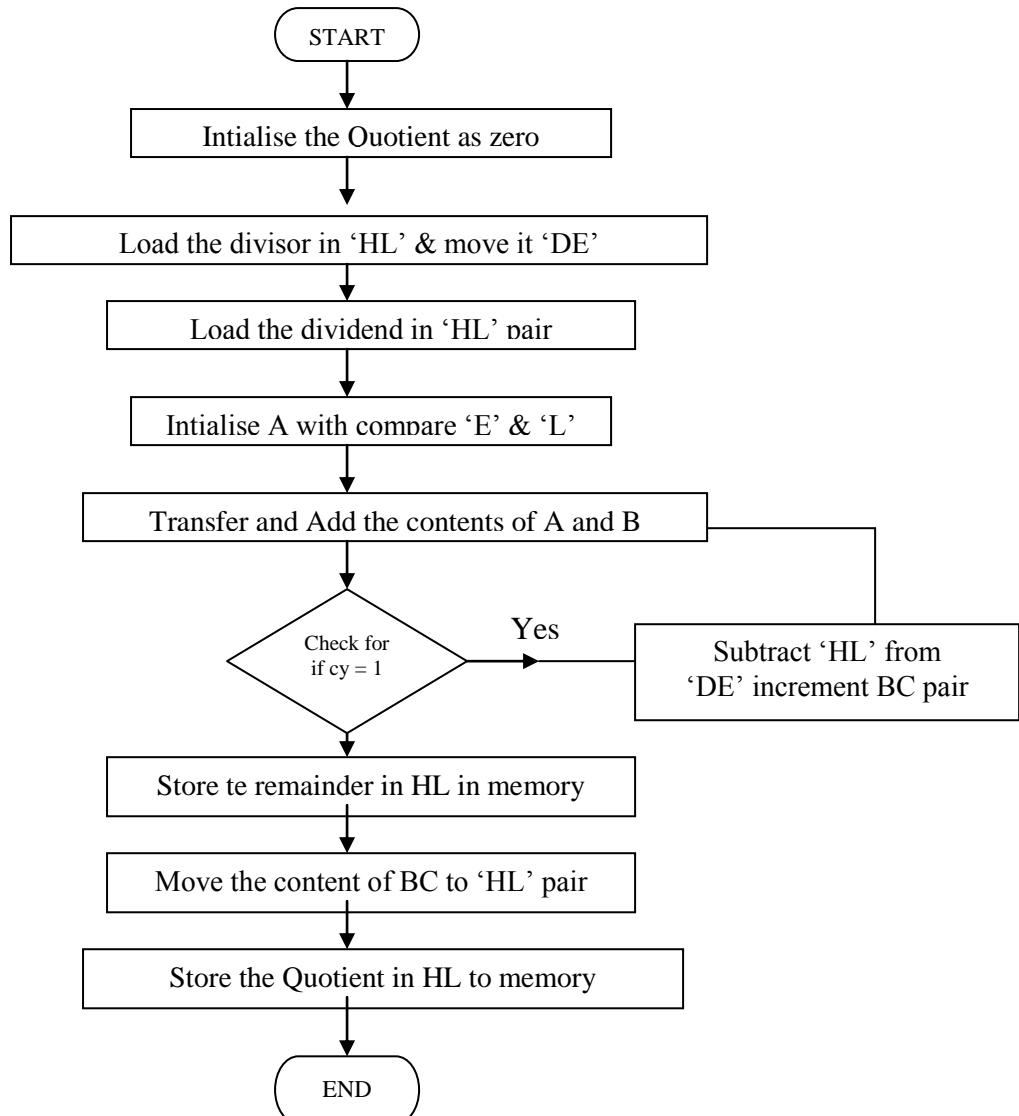
To write an assembly language program for 16 bit division in 8085 microprocessor.

Apparatus required:

8085 microprocessor kit
(0-5V) DC battery

Algorithm:

- | | | |
|---------|---|---|
| Step 1 | : | Start the microprocessor |
| Step 2 | : | Initialise ‘BC’ as ‘0000’ for Quotient |
| Step 3 | : | Load the divisor in ‘HL’ pair and save it in ‘DE’ register pair |
| Step 4 | : | Load the dividend in ‘HL’ pair |
| Step 5 | : | Move the value of ‘a’ to register ‘E’ |
| Step 6 | : | Subtract the content of accumulator with ‘E’ register |
| Step 7 | : | Move the content ‘A’ to ‘C’ & ‘H’ to ‘A’ |
| Step 8 | : | Subtract with borrow, the content of ‘A’ with ‘D’ |
| Step 9 | : | Move the value of ‘a’ to ‘H’ |
| Step 10 | : | If cy = 1, go to step 12, otherwise next step |
| Step 11 | : | Increment ‘B’ register & jump to step ‘4’ |
| Step 12 | : | Add both contents of ‘DC’ and ‘HL’ |
| Step 13 | : | Store the remainder in memory |
| Step 14 | : | Move the content of ‘C’ to ‘L’ & ‘B’ to ‘H’ |
| Step 15 | : | Store the Quotient in memory |
| Step 16 | : | Stop the program |



| Address | Label | Mnemonics | Hex Code | Comments |
|---------|--------|------------|------------|-------------------------------|
| 4500 | | LXI B,0000 | 0,00,00 | Initialise Quotient as '0000' |
| 4503 | | LHLD 4802 | 2A,02,48 | Load the divisor in 'HL' |
| 4506 | | XCHG | EB | Exchange 'HL' and 'DE' |
| 4507 | | LHLD 4800 | 2A,00,48 | Load the dividend |
| 450A | Loop 2 | MOV A,L | 7D | Move the 'L' value to 'A' |
| 450B | | SUB E | 93 | (A-E) - A |
| 450C | | MOV L,A | 6F | A - L (A value is moved to L) |
| 450D | | MOV A,H | 7C | H - A (A is stored with H) |
| 450E | | SBB D | 9A | Subtract 'D' from 'A' |
| 450F | | MOV H,A | 67 | Then A is moved to 'H' |
| 4510 | | JC loop 1 | DA,17,45 | If cy is present go to loop 1 |
| 4513 | | INX B | 03 | Increment BC pair by 1 |
| 4514 | | JMP loop 2 | C3, 0A, 45 | Jump to loop 2 |
| 4517 | Loop 1 | DAD 'D' | 19 | 'DE' and 'HL' pair all added |
| 4518 | | SHLD 4806 | 22,06,48 | HL is stored in memory |
| 451B | | MOV L,C | 69 | Move 'C' register data to 'L' |
| 451C | | MOV H,B | 60 | Move 'B' register data to 'H' |
| 451D | | SHLD 4804 | 22,04,48 | Store the result in 'HL' pair |
| 4520 | | HLT | 76 | Stop the program |

Input

| Input Address | Value |
|---------------|-------|
| 4800 | 04 |
| 4801 | 00 |
| 4802 | 02 |
| 4803 | 00 |

Output

| Output Address | Value |
|----------------|-------|
| 4804 | 02 |
| 4805 | 00 |
| 4806 | FE |
| 4807 | FF |

Result:

Thus the assembly language program for 16 bit division was executed successfully.

BINARY TO BCD CONVERSION

Aim:

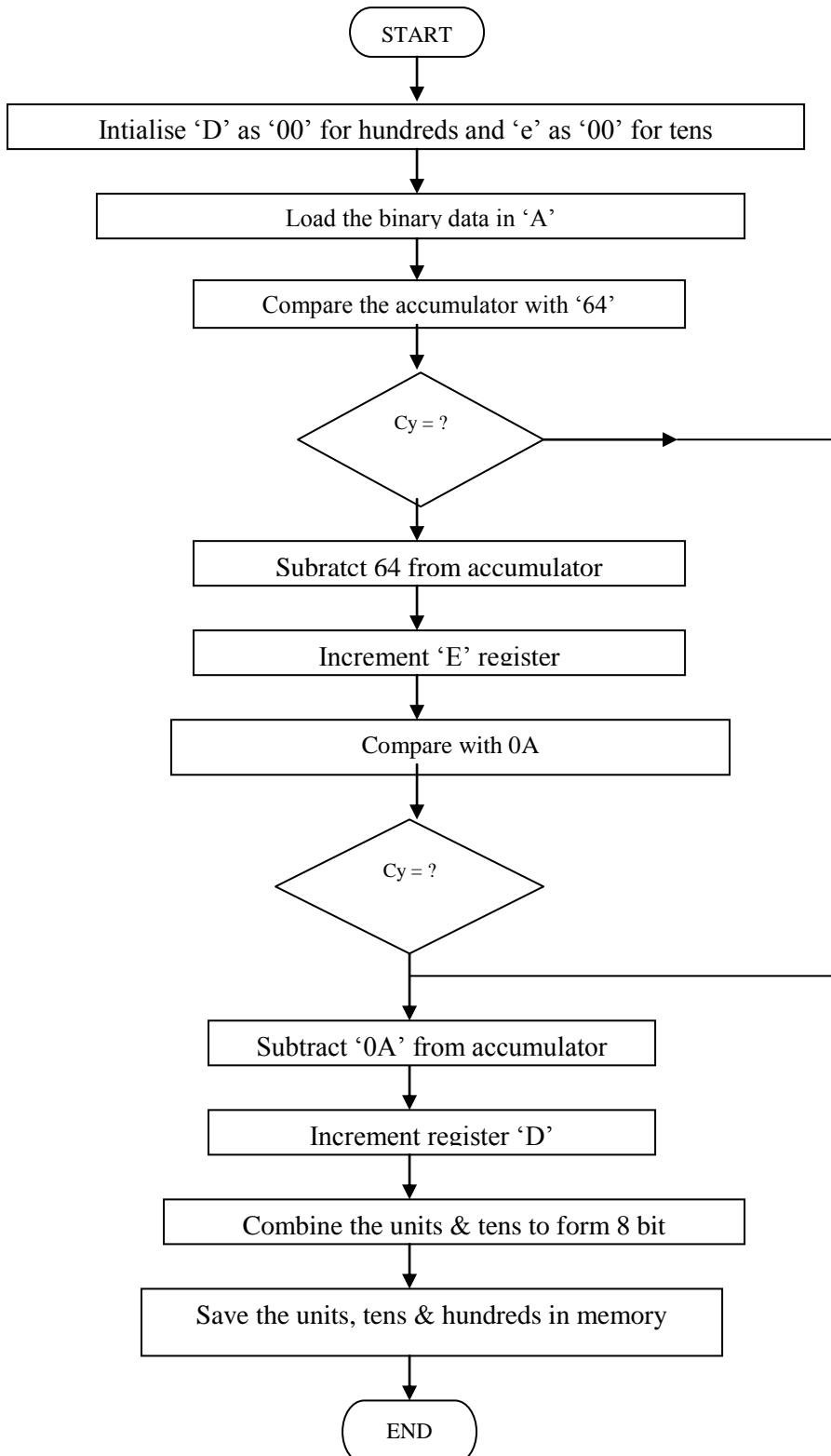
To write an assembly language program to convert an 8 bit binary data to BCD using 8085 microprocessor kit.

Apparatus required:

8085 microprocessor kit
(0-5V) power supply

Algorithm:

- | | | |
|---------|---|--|
| Step 1 | : | Start the microprocessor |
| Step 2 | : | Clear 'D' and 'E' register to account for hundred's and ten's load the binary data in accumulator |
| Step 3 | : | Compare 'A' with 64 if cy = 01, go step C otherwise next step |
| Step 4 | : | Subtract 64 from (64+1) 'A' register |
| Step 5 | : | Increment 'E' register |
| Step 6 | : | Compare the register 'A' with '0A', if cy=1, go to step 11, otherwise next step |
| Step 7 | : | Subtract ($0A_H$) from 'A' register |
| Step 8 | : | Increment D register |
| Step 9 | : | Go to step 7 |
| Step 10 | : | Combine the units and tens to form 8 bit result |
| Step 11 | : | Save the units, tens and hundred's in memory |
| Step 12 | : | Stop the program execution |



| Memory Location | Hex Code | Label | Mnemonics | | Comments |
|-----------------|----------|-------|-----------|---------|--------------------------------|
| | | | Op code | Operand | |
| 4100 | 0E,00 | | MVI | E,00 | Clear ‘E’ register (Hund) |
| 4102 | 53 | | MOV | D,E | Clear ‘D’ register (tens) |
| 4103 | 3A,00,42 | | LDA | 4200 | Get the data in ‘A’ |
| 4106 | C3,06,41 | HUND | CPI | 64 | Compare the data with 64 |
| 4108 | DA,11,41 | | JC | TEN | If content is less jump to ten |
| 410B | D6, 64 | | SUI | 64 | Subtract data by 64 |
| 410D | IC | | INR | E | Increment carry each time |
| 410E | C3,06,41 | | JMP | HUND | Jump to hundred & repeat |
| 4111 | C3, 0A | TEN | CPI | 0A | Compare the data with 0A |
| 4113 | DA,1C,41 | | JC | UNIT | If data is less jump to unit |
| 4116 | D6, 0A | | SUI | 0A | Subtract the data by 0A |
| 4118 | 14 | | INR | D | Increment ‘D’ each time |
| 4119 | C3,11,41 | | JMP | TEN | Jump to ten & repeat |
| 411C | 4F | UNIT | MOV | 4A | Move the value ‘A’ to ‘C’ |
| 411D | 7A | | MOV | A,D | Move the value ‘D’ to ‘A’ |
| 411E | 07 | | RLC | | Rotate the value of ‘A’ |
| 411F | 07 | | RLC | | Of ‘A’ so that |
| 4120 | 07 | | RLC | | Lower and upper niddle |
| 4121 | 07 | | RLC | | Gets exchanged |
| 4122 | 81 | | ADD | C | Add ‘A’ and ‘C’ |
| 4123 | 32,50,42 | | STA | 42,50 | Save ten’ & units in ‘M’ |
| 4126 | 7B | | MOV | A,E | Move to E to A |
| 4127 | 32,51,42 | | STA | 4251 | Save hundreds unit in ‘A’ |
| 412A | 76 | | HLT | | Stop the program execution |

Input

| Input Address | Value |
|---------------|-------|
| 4200 | 54 |

Output

| Output Address | Value |
|----------------|-------|
| 4250 | 84 |
| 4251 | 00 |

Result:

Thus the binary to BCD conversion was executed successfully

BCD TO BINARY**Aim:**

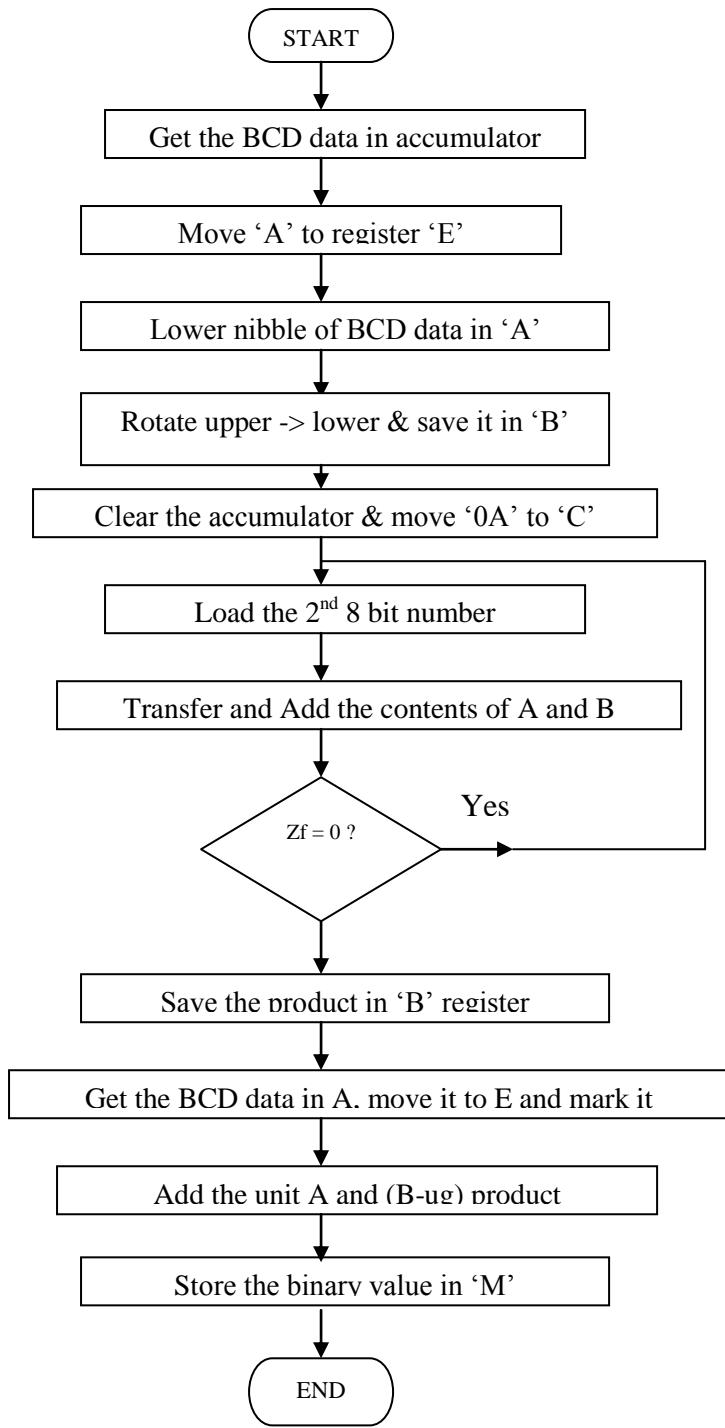
To write an assembly language program to convert BCD data to Binary data using 8085 microprocessor kit.

Apparatus required:

8085 microprocessor kit
(0-5V) power supply

Algorithm:

- | | | |
|---------|---|---|
| Step 1 | : | Start the microprocessor |
| Step 2 | : | Get the BCD data in accumulator and save it in register 'E' |
| Step 3 | : | Mark the lower nibble of BCD data in accumulator |
| Step 4 | : | Rotate upper nibble to lower nibble and save it in register 'B' |
| Step 5 | : | Clear the accumulator |
| Step 6 | : | Move $0A_H$ to 'C' register |
| Step 7 | : | Add 'A' and 'B' register |
| Step 8 | : | Decrement 'C' register. If $zf = 0$, go to step 7 |
| Step 9 | : | Save the product in 'B' |
| Step 10 | : | Get the BCD data in accumulator from 'E' register and mark the upper nibble |
| Step 11 | : | Add the units (A_{ug}) to product (B_{ug}) |
| Step 12 | : | Store the binary value in memory |
| Step 13 | : | End the program |



| Memory Location | Hex Code | Label | Mnemonics | | Comments |
|-----------------|----------|-------|-----------|---------|----------------------------|
| | | | Op code | Operand | |
| 4100 | 3A,00,42 | | LDA | 4200 | Get the data in 'A' |
| 4103 | 5E | | MOV | E,A | Save in 'E' register |
| 4104 | E6, F0 | | ANI | F0 | Mark the lower nibble |
| 4106 | 07 | | RLC | | Rotate the upper |
| 4107 | 07 | | RLC | | To lower nibble |
| 4108 | 07 | | RLC | | And save in |
| 4109 | 07 | | RLC | | Register B |
| 410A | 47 | | MOV | B,A | Move it from 'A' to 'B' |
| 410B | AF | | XRA | A | Clear the accumulator |
| 410C | 0E,0A | | MVI | C,0A | Initialise 'C' as '0A' |
| 410E | 08 | | REP | | |
| 410F | 0D | | DCR | C | Decrement 'C' register |
| 4110 | C2,0E,41 | | JNZ | | Jump till value 'C' is 0 |
| 4113 | 47 | | MOV | B,A | Move the value A to B |
| 4114 | 7B | | MOV | A,E | Get the BCD in 'A' |
| 4115 | E6, 0F | | ANI | 0F | Mark the upper nibble |
| 4117 | 80 | | ADD | B | Add 'A' and 'B' |
| 4118 | 32,01,42 | | STA | 4201 | Save the binary data |
| 411B | 76 | | HLT | | Stop the program execution |

Input

| Input Address | Value |
|---------------|-------|
| 4200 | 68 |

Output

| Output Address | Value |
|----------------|-------|
| 4201 | 44 |

$$\begin{array}{c|c} 16 & 68 \\ \hline & 4-4 \end{array}$$

Result:

Thus the BCD to binary conversion was executed successfully

SPEED CONTROL OF STEPPER MOTOR

Aim:

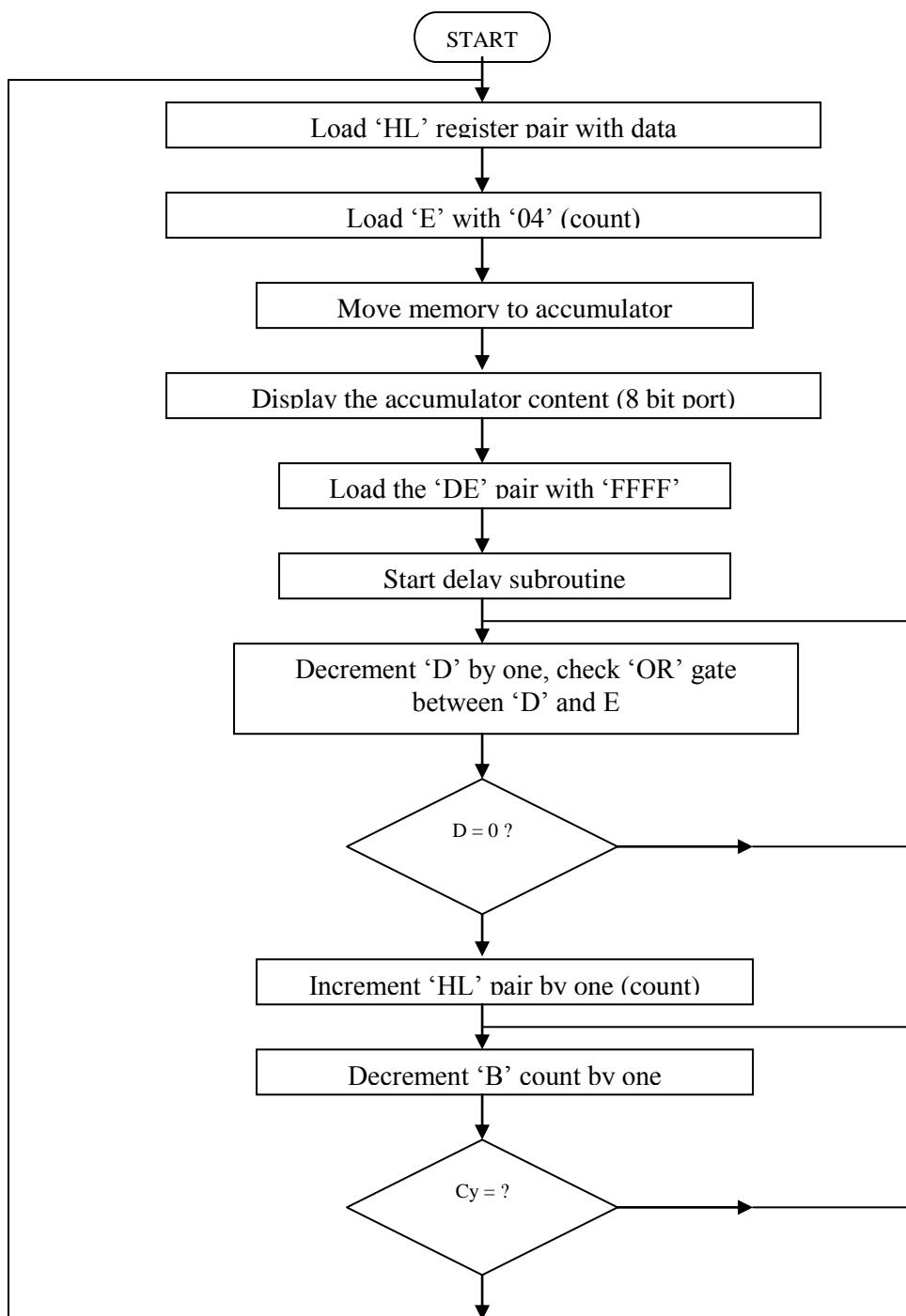
To write an assembly program to make the stepper motor run in forward and reverse direction.

Apparatus required:

Stepper motor
8085 microprocessor kit
(0-5V) power supply

Algorithm:

- Step 1 : Load the 'HL' pair with value from table
- Step 2 : Move it to 'B' register for setting the counter
- Step 3 : Move the memory value to accumulator and display it by control word
- Step 4 : Load 'DE' register pair with FFFF for starting delay subroutine
- Step 5 : Run the delay loop control D-register becomes zero.
- Step 6 : Increment 'H' address for next value from table
- Step 7 : Jump on no zero
- Step 8 : When B = 0, go to start and restart the program



| Memory Location | Hex Code | Label | Mnemonics | | Comments |
|-----------------|----------|-------|-----------|-----------|-------------------------|
| | | | Op code | Operand | |
| 4100 | Start | LXI | H,Look up | 21,1A,41 | Load the ‘HL’ with data |
| 4103 | | MVI | B,04 | 06,04 | B = 04 |
| 4105 | Repeat | MOV | A,M | 7E | Memory value to ‘A’ |
| 4106 | | OUT | C0 | D3, C0 | Display it |
| 4108 | | LXI | D,03,03 | 11 | Load ‘DE’ with FFFF |
| 410B | Delay | NOP | | 00 | Start delay loop |
| 410C | | DCX | D | 1B | Decrement DE by 1 |
| 410D | | MOV | A,E | 7B | Move ‘E’ to ‘A’ |
| 410E | | ORA | D | B2 | Check De = 0 or not |
| 410F | | JNZ | DELAY | C2, 0B,41 | Jump on zero |
| 4112 | | INX | H | 23 | Increment HL by 1 |
| 4113 | | DCR | B | 05 | Decrement B by 1 |
| 4114 | | JNZ | Repeat | C2,05,41 | Jump on no zero |
| 4117 | | JMP | START | C3,00,41 | Jump to start |

Input

| Input Address | Value |
|---------------|-------|
| 411A | 0A |
| 411B | 06 |
| 411C | 05 |
| 411D | 09 |

Reverse Direction

| Output Address | Value |
|----------------|-------|
| 411A | 09 |
| 411B | 05 |
| 411C | 06 |
| 411D | 0A |

Result:

Thus, an assembly language program to control of stepper motor was written using 8085 microprocessor kit.

FLASHING DISPLAY

Aim:

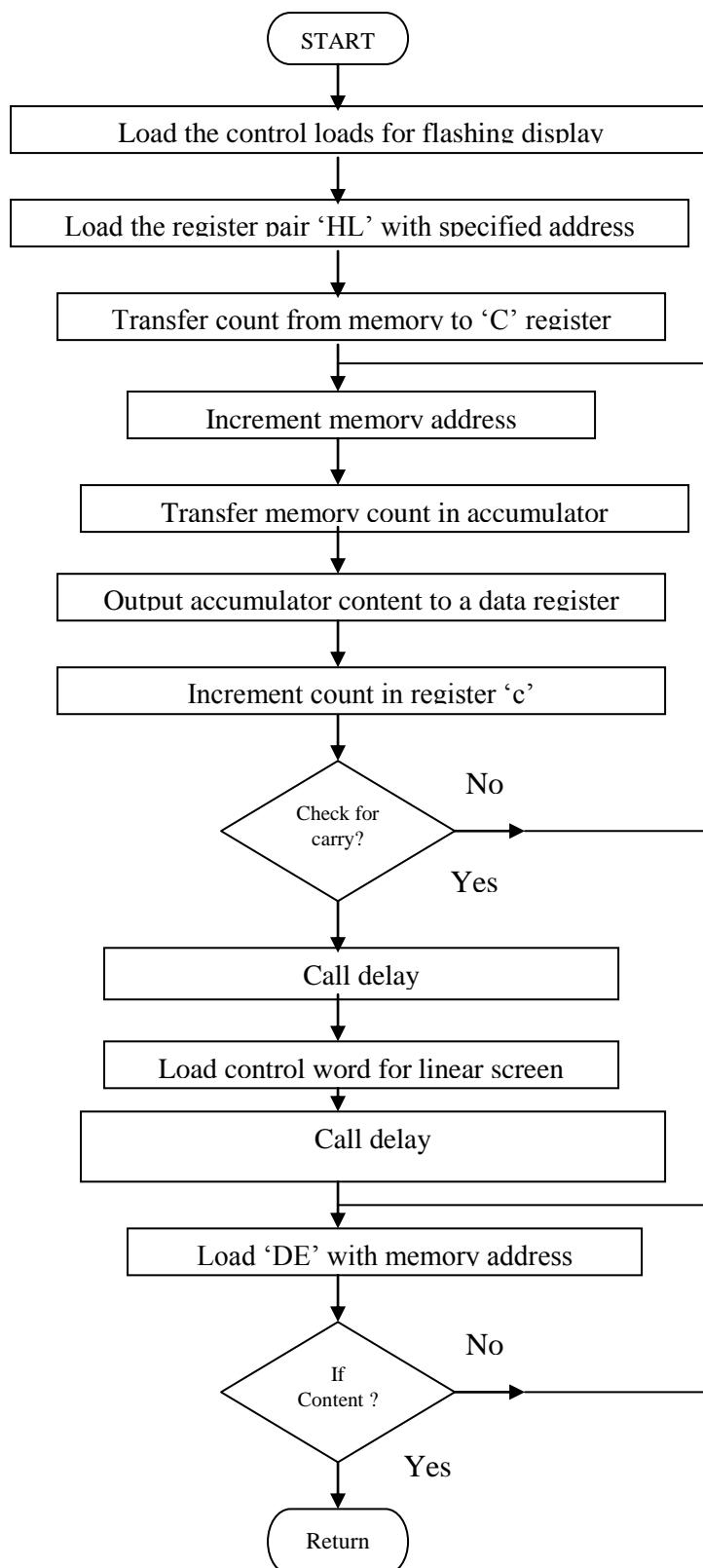
To write an assembly language program to obtain the following flashing display of a particular data.

Apparatus required:

8085 micro processing kit
(0-5V) power supply

Algorithm:

- | | | |
|---------|---|--|
| Step 1 | : | Get the control words in accumulator and output words through 8 bit address |
| Step 2 | : | Load ‘HL’ register pair with memory address |
| Step 3 | : | Get the count value in ‘C’ register |
| Step 4 | : | Increment the register pair by one and display the character and call for delay. |
| Step 5 | : | Clear the display and call delay routine to step 7 |
| Step 6 | : | Go to step 7 |
| Step 7 | : | Load ‘DE’ register pair with memory address |
| Step 8 | : | Decrement ‘DE’ pair with memory address |
| Step 9 | : | If the content is not equal to zero, go to step 8 |
| Step 10 | : | Return to main program |



| Memory Location | Hex Code | Label | Mnemonics | | Comments |
|-----------------|----------|-------|-----------|----------|------------------------------------|
| | | | Op code | Operand | |
| 4300 | | MVI | A,00 | 3E,00 | Initialise 'A' as '00' |
| 4302 | | OUT | 01 | DE,01 | Out the control word through 8 bit |
| 4304 | | MVI | A,90 | 3E,90 | Initialise 'a' with cw for RAM |
| 4306 | | OUT | 01 | D3,01 | Out the cw |
| 4308 | | MVI | A,CC | 3E,CC | A = CC |
| 430A | | OUT | 01 | 0D,01 | Out the cw |
| 430C | Loop 2 | LXI | H,5000 | 21,00,50 | Load 'HL' with |
| 430F | | MOV | C,M | 4E | M to C |
| 4310 | Loop 1 | INX | H | 23 | Increment 'H' by |
| 4311 | | MOV | A,M | 7E | Move M to A |
| 4312 | | OUT | 00 | D3, 00 | Out the character |
| 4314 | | DCR | C | 0D | Decrement 'C' by 1 |
| 4315 | | JNZ | Loop 1 | C2,10,43 | Check for zero |
| 4318 | | CALL | Delay | C0,00,46 | Call subroutine |
| 431B | | MVI | A,DC | 3E,DC | A <- 0C |
| 431D | | OUT | 01 | D3, 01 | A<-01 |
| 431F | | CALL | Delay | CD,00,46 | Call subroutine |
| 4322 | | JMP | Loop 2 | C3 0C,43 | Check for zf |
| 4600 | Delay | LXI | D,FFFF | 11,FF,FF | Initialise DE=FFFF |
| 4603 | Loop 3 | DCX | D | 1B | Decrement DE by 1 |
| 4604 | | MOV | A,E | 7B | Move 'E' to 'A' |
| 4605 | | ORA | D | B2 | Check 'De' = '00' |
| 4606 | | JNZ | Loop 3 | C2,03,46 | Jump on no zero |
| 4609 | | RET | C9 | C9 | Return to main program |

Input

| Input Address | Value |
|---------------|-------|
| 5000 | 05 |
| 5001 | 68 |
| 5002 | 68 |
| 5003 | 68 |
| 5004 | FD |
| 5005 | 88 |

Output

EEE – A

Result:

Thus, an assembly language program to obtain flashing display of a particular data was written using 8085 microprocessor kit.

ROLLING DISPLAY

Aim:

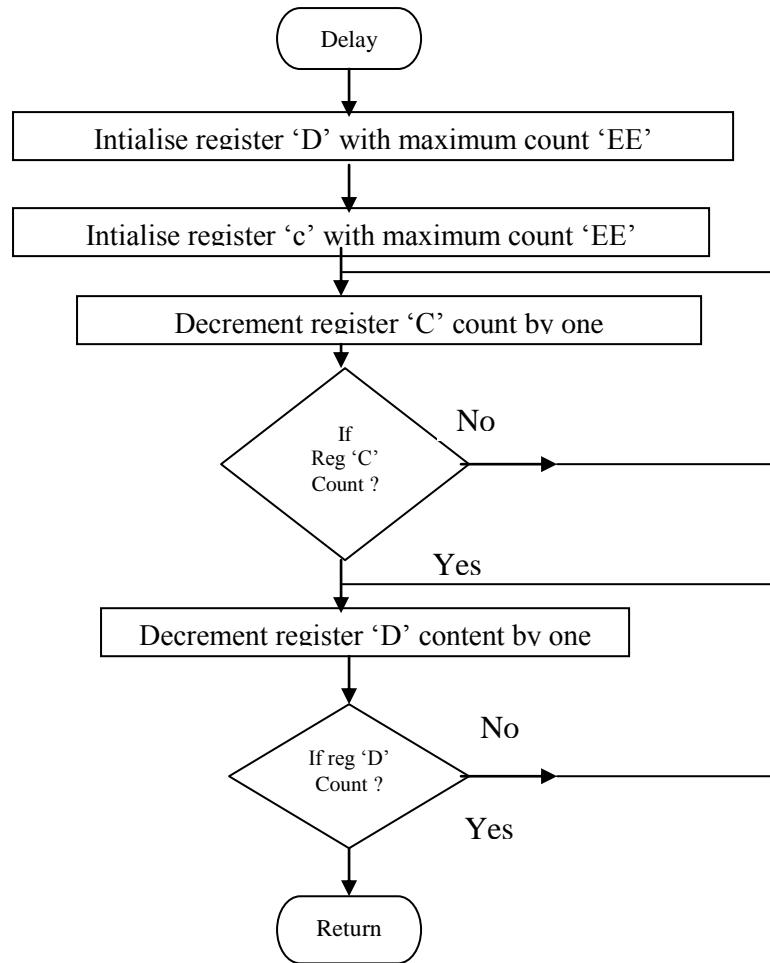
To write an assembly language program to obtain a rolling display of a particular data by using 8085 microprocessor

Apparatus required:

8085 micro processing kit
(0-5V) power supply

Algorithm:

- Step 1 : Get the control words in accumulator and output the control words through 8 bit port address
- Step 2 : Load ‘HL’ register pair with memory address and transfer memory content to ‘C’ register
- Step 3 : Increment ‘HL’ pair with one and transfer the particular bit pattern through 8 bit port address
- Step 4 : Call subroutine delay at step 6
- Step 5 : If the count value in ‘C’ is not equal to zero then go to step 3 else go to step 2
- Step 6 : Load ‘DE’ register pair by memory address
- Step 7 : Decrement ‘DE’ register pair by one
- Step 8 : If DE is not equal to zero, go to step 7 else main program



| Memory Location | Hex Code | Label | Mnemonics | | Comments |
|-----------------|----------|-------|-----------|----------|------------------------------|
| | | | Op code | Operand | |
| 4500 | | MVI | A,00 | 3E,00 | Initialise A 00 |
| 4502 | | OUT | 01 | DE, 01 | Control word through 8 bit |
| 4504 | | MVI | A,90 | 3E, 90 | A = RAM cw |
| 4506 | | OUT | 01 | DE,01 | Output cw through 8 bit port |
| 4508 | | MVI | A,CC | 3E,CC | A = CC |
| 450A | | OUT | 01 | DE,01 | Output cw through 8 bit port |
| 450C | Loop 2 | LXI | H,5000 | 21,00,50 | Memory -> HL location |
| 450F | | MOV | C,M | 4E | M -> C |
| 4510 | Loop 1 | INX | H | 23 | Increment 'HL' |
| 4511 | | MOV | A,M | 7E | Move 'H' to 'A' |
| 4512 | | OUT | 00 | DE, 00 | Output the character |
| 4514 | | CALL | Loop | CD,00,46 | Call the subroutine |
| 4517 | | DCR | C | 0D | Decrement 'C' by one |
| 4518 | | JNZ | Loop 1 | C2,10,45 | Jump on no zero |
| 451B | | JMP | Loop 2 | C3,0C,45 | Jump to L2 |
| 4600 | Loop | LXI | D,FFFF | 11,FFFF | Load DE-FFFF |
| 4603 | Loop 3 | DCX | D | 1B | Decrement 'DE' by 1 |
| 4604 | | MOV | A,D | 7A | Move 'D' to 'A' |
| 4605 | | ORA | E | B3 | (A) = (A) check |
| 4606 | | JNZ | Loop 3 | C2,03,46 | Jump on no zero |
| 4609 | | RET | | C9 | Return to main program |

Input

| Input Address | Value |
|---------------|-------|
| 5000 | 06 |
| 5001 | 98 |
| 5002 | 68 |
| 5003 | 7A |
| 5004 | C8 |
| 5005 | 1A |
| 5006 | 2C |

Output

HELPUS

Result:

Thus, an assembly language program to obtain rolling display of a particular value written using 8085 microprocessor kit.

SQUARE WAVE GENERATOR

Aim:

To write a program and to generate square generator using DAC.

Apparatus required:

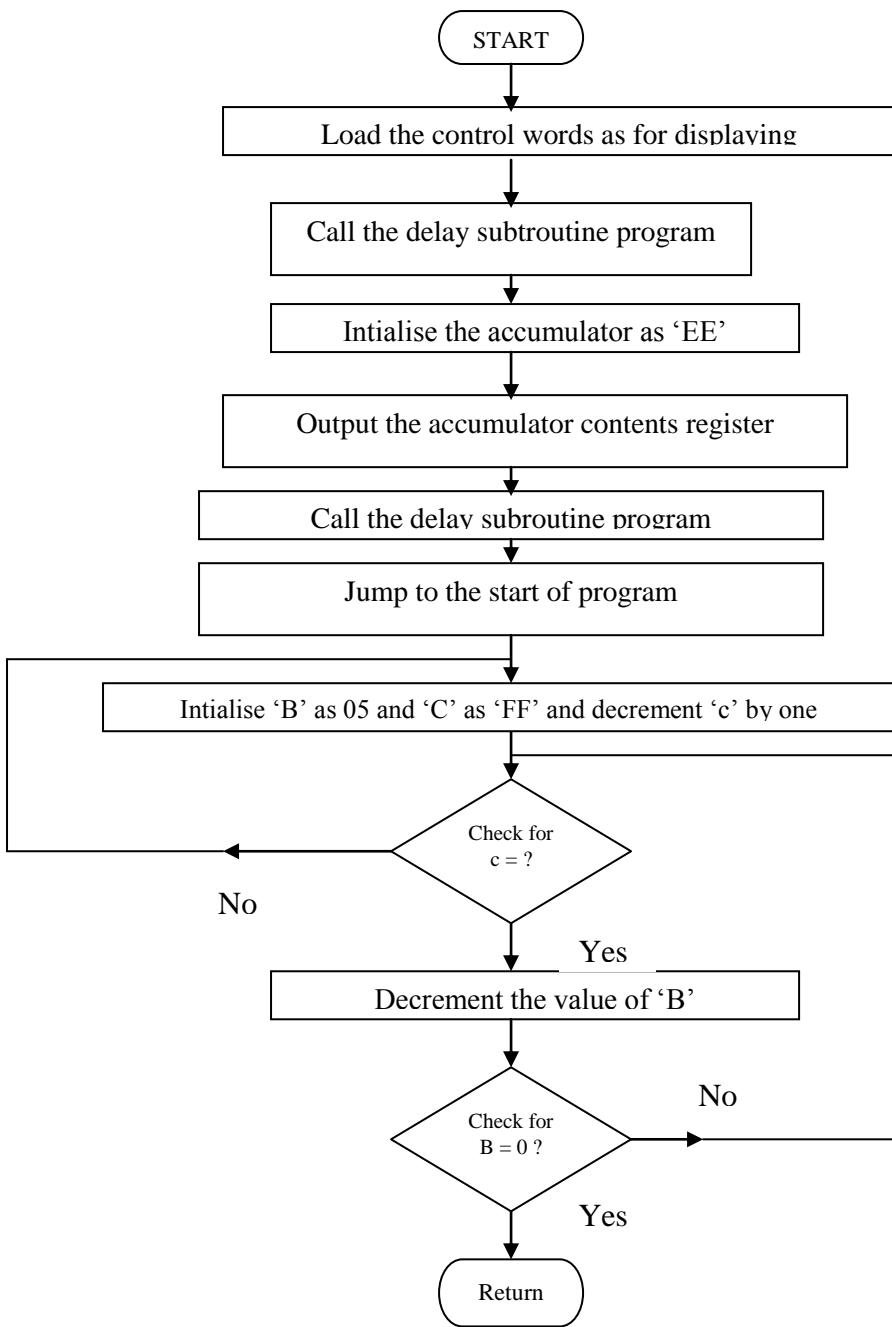
8085 microprocessor kit
(0-5V) power supply

Algorithm:

| | | |
|--------|---|---|
| Step 1 | : | Initialise 'A' as '00' and take data pointer to port C8 |
| Step 2 | : | Call delay |
| Step 3 | : | Move FF to A and take port 'C8' |
| Step 4 | : | Call delay |
| Step 5 | : | Go to step 1 |

Delay Subroutine

| | | |
|--------|---|--------------------------------------|
| Step 1 | : | Counter 1 = 05 |
| Step 2 | : | Counter 2 = FF |
| Step 3 | : | Decrement counter 2 |
| Step 4 | : | Check if c= 0, if no jump to step 3 |
| Step 5 | : | Decrement counter 1 |
| Step 6 | : | Check if B = 0, if no jump to step 2 |
| Step 7 | : | Return to main program |



| Memory Location | Hex Code | Label | Mnemonics | | Comments |
|------------------------|-----------------|--------------|------------------|----------------|-------------------------|
| | | | Op code | Operand | |
| 4100 | 3E,00 | Start | MVI | A,00 | Intialise 'A' with '00' |
| 4102 | D3,C8 | | OUT | C8 | Load the control words |
| 4104 | CD,11,41 | | CALL | Delay | Call delay subroutine |
| 4107 | 3E,FF | | MVI | A,FF | Intialise 'A' with 'FF' |
| 4109 | D3,C8 | | OUT | C8 | A -> C8 |
| 410B | CD,11,41 | | CALL | Delay | Call delay subroutine |
| 410E | C3,00,41 | | JMP | Start | Jump to start |
| 4111 | 06,05 | Delay | MVI | B,05 | B -> 05 |
| 4113 | 0E | Loop 1 | MVI | C,FF | [C] => FF |
| 4115 | OD | Loop 2 | DCR | C | Decrement 'C' register |
| 4116 | C2,15,41 | | JNZ | Loop 2 | Jump on no zero |
| 4119 | 05 | | DCR | B | Decrement 'B' register |
| 411A | C2,13,41 | | JNZ | Loop 1 | Jump on n zero |
| 411D | C9 | | RET | | Return to main program |

Result:

Thus square wave was generated using 8085 microprocessor kit.

TRIANGULAR WAVE GENERATOR

Aim:

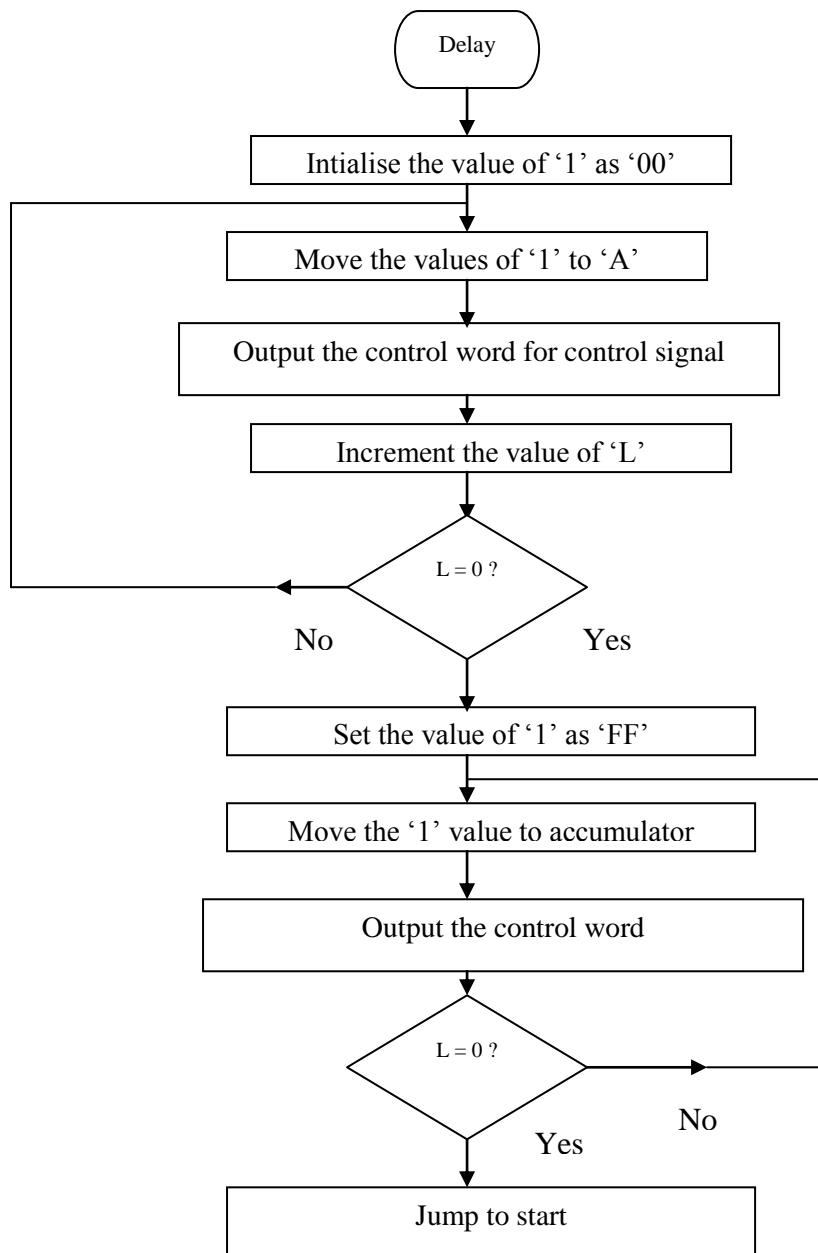
To write an assembly language program for generating triangular wave using DAC.

Apparatus required:

8085 micro processor kit
(0-5V) DC battery

Algorithm:

- | | | |
|--------|---|--|
| Step 1 | : | Move content of 'C' to 'A' where 'L' is initialised to '00' |
| Step 2 | : | Output content of C8 |
| Step 3 | : | Increment L till zf = 0 |
| Step 4 | : | Initialise 'L' register with FF |
| Step 5 | : | Move content of 'L' to accumulator and output to port |
| Step 6 | : | Decrement 'L' if not equal to zero jump else go to next step |
| Step 7 | : | Jump on next step |



| Memory Location | Hex Code | Label | Mnemonics | | Comments |
|-----------------|----------|--------|-----------|---------|---------------------------|
| | | | Op code | Operand | |
| 4300 | 2E,00 | Start | MVI | L,00 | Initialise 'L' as '00' |
| 4302 | 7D | Loop 1 | MOV | A,L | [L] -> [A] |
| 4303 | D3,C8 | | OUT | C8 | Load the control words |
| 4305 | 2C | | INR | L | Increment register 'L' |
| 4306 | C2,02,43 | | JNZ | Loop 1 | Jump on no zero to loop 1 |
| 4309 | 2E, FF | | MVI | L,FF | L = FF |
| 430B | 70 | Loop 2 | MOV | A,L | L -> A |
| 430C | D3,C8 | | OUT | C8 | [C8] -> [A] |
| 430E | 2D | | DCR | L | Decrement L by one |
| 430F | C2,0B,43 | | JNZ | Loop 2 | Jump on no zero to 430B |
| 4312 | C3,00.43 | | JMP | Start | Repeat process |

Result:

Thus the triangular wave was generated using 8085 microprocessor kit.

SAWTOOTH WAVE GENERATOR

Aim:

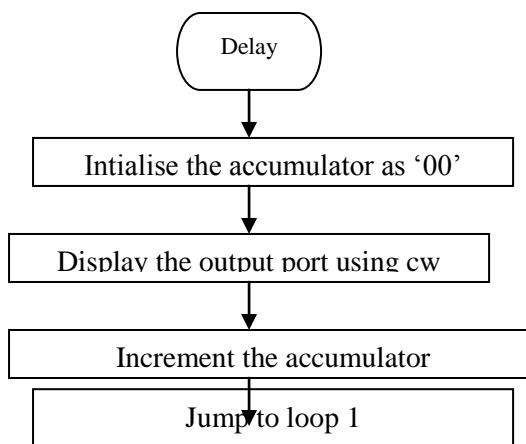
To write an assembly language program for generating Sawtooth waveform by using microprocessor 8085.

Apparatus required:

8085 microprocessor kit
(0-5V) power supply

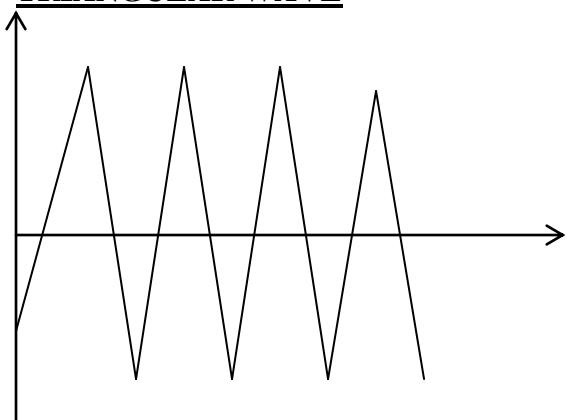
Algorithm:

- Step 1 : Intialise accumulator with '00'
- Step 2 : Output current address specified
- Step 3 : Increment accumulator by one
- Step 4 : Jump to step one

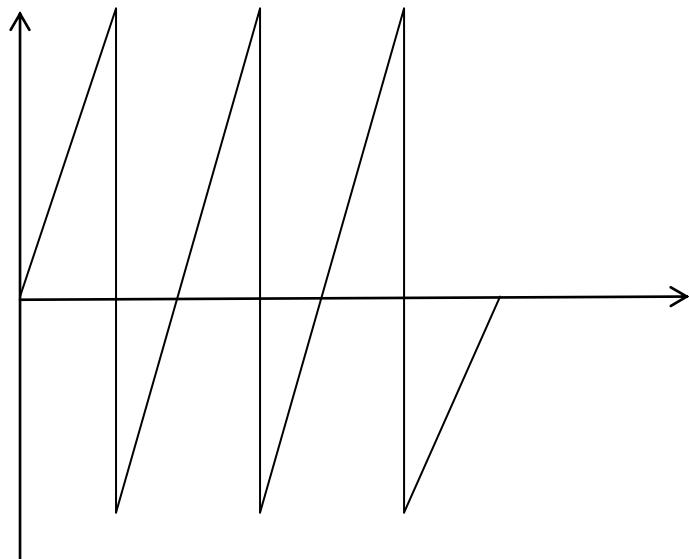


| Memory Location | Hex Code | Label | Mnemonics | | Comments |
|-----------------|----------|--------|-----------|---------|-----------------------|
| | | | Op code | Operand | |
| 4500 | 3E,00 | Start | MVI | A,00 | Intialise 'A' as '00' |
| 4502 | D3, C8 | Loop 1 | OUT | C8 | A = [C8] |
| 4504 | 3C | | INR | A | Increment 'A' by one |
| 4505 | C3,02,45 | | JMP | Loop 1 | Jump to loop one |

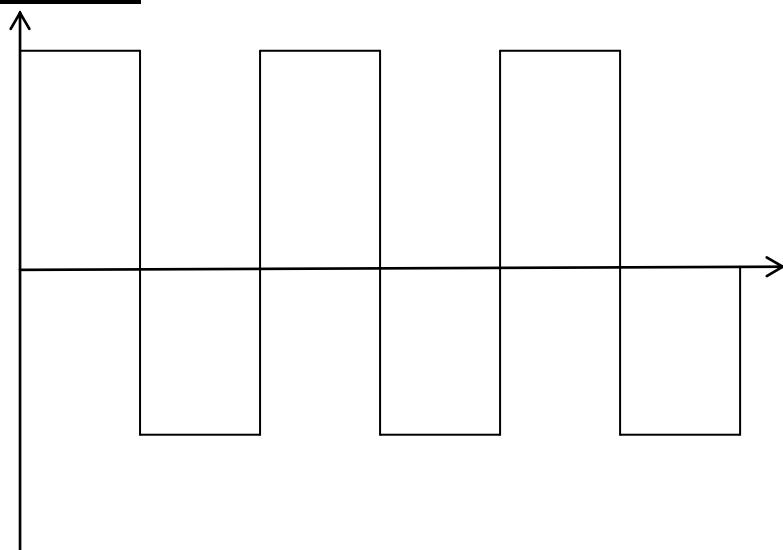
TRIANGULAR WAVE



SAW TOOTH WAVE



SQUARE WAVE



Result:

Thus the Sawtooth wave was generated using 8085 microprocessor kit.

ANALOG TO DIGITAL CONVERTER

Aim:

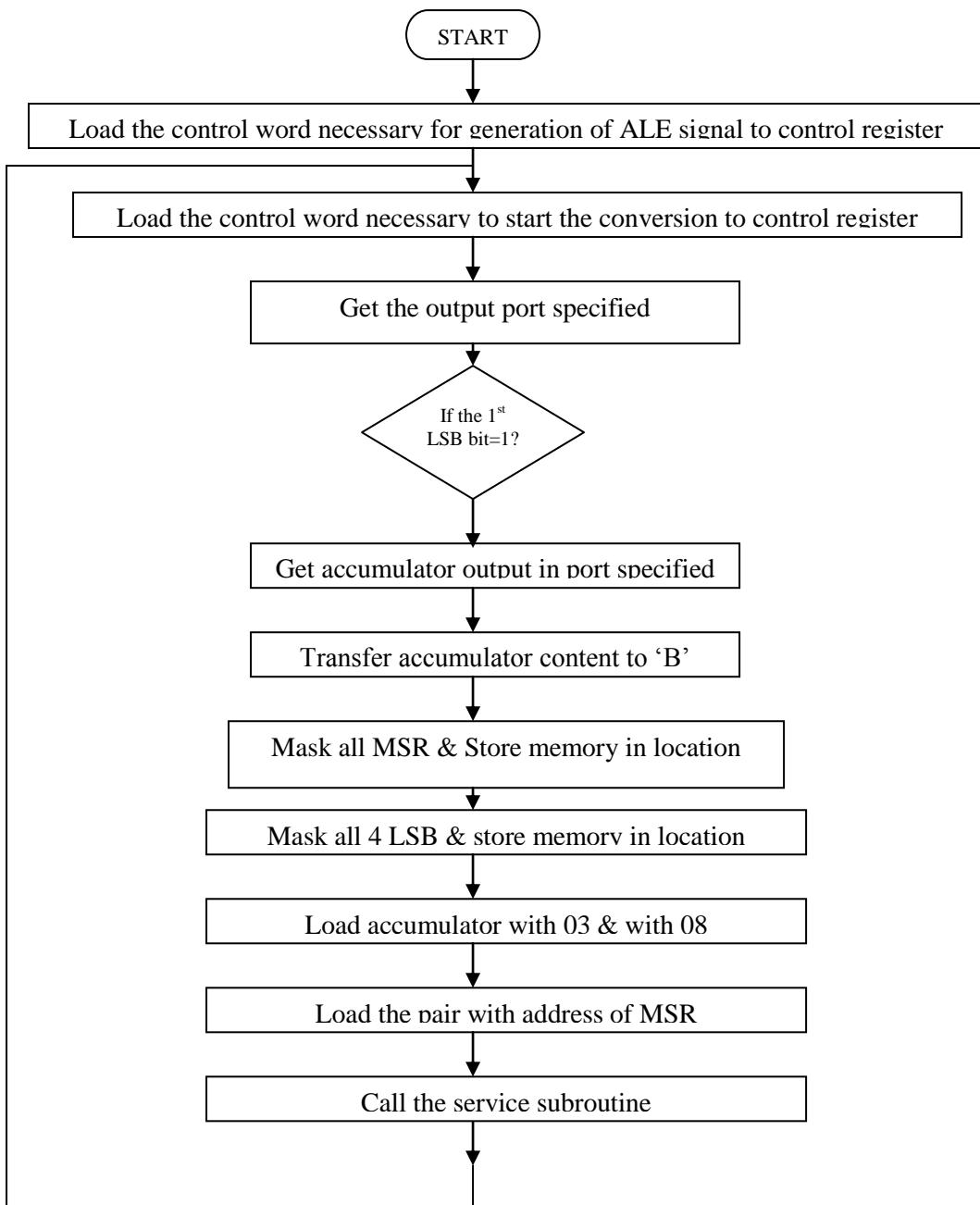
To write an assembly language program to convert analog to digital signal and to display it in 7 segment LED display

Apparatus required:

8085 microprocessor kit
(0-5V) power supply

Algorithm:

- Step 1 : Access the channel of ADC
- Step 2 : Initialise the accumulator with start of conversion signal & output it to the ADC
- Step 3 : Send '0' signal for ending the conversion for ADC
- Step 4 : Get the analog value converted to display from ADC
- Step 5 : The digital signal is separated into two nibbles and displayed in hexadecimal from by calling service subroutine.
- Step 6 : Go to step 1



| Memory Location | Hex Code | Label | Mnemonics | | Comments |
|-----------------|----------|-------|-----------|---------|--|
| | | | Op code | Operand | |
| 5000 | 3E,10 | | MVI | A,10 | Initialise 'a' with 10 |
| 5002 | D3,C | | OUT | C8 | Output channel through |
| 5004 | 3E,18 | | MVI | A,18 | Initialise 'A' with 18 |
| 5006 | D3,C8 | | OUT | C8 | Output channel through 8 bit port |
| 5008 | 00 | | NOP | | No operation |
| 5009 | 00 | | NOP | | No operation |
| 500A | 3E,10 | | MVI | A,10 | Initialise 'A' with 2 nd signal |
| 500C | D3,C8 | | OUT | C8 | Output channel through 8 bit port |
| 500E | 3E,01 | L2 | MVI | A,01 | Initialise 'A' with 2 nd |
| 5010 | D3,D0 | | OUT | D0 | Output through 8 bit |
| 5012 | 00 | | NOP | | |
| 5013 | 00 | | NOP | | |
| 5014 | 00 | | NOP | | |
| 5015 | 3E,00 | | MVI | A,00 | |
| 5017 | D3,D0 | | OUT | D0 | |
| 5019 | DB,D8 | L1 | IN | D8 | |
| 501B | E6,01 | | ANI | 01 | |
| 501D | CA,19,50 | | JZ | L1 | |
| 5020 | DB,C0 | | IN | C0 | Get input from |
| 5022 | 47 | | MOV | B,A | B -> A |
| 5023 | E6,0F | | ANI | 0F | And of with 'A' |
| 5025 | 32,51,51 | | STA | 5151 | Store in 5151 |
| 5028 | 78 | | MOV | A,B | B -> A |
| 5029 | E6,F0 | | ANI | F0 | And F0 with A |
| 502B | 0F | | RRC | | Rotate content 'A' |
| 502C | 0F | | RRC | | |
| 502E | 0F | | RRC | | |
| 502F | 32,50,51 | | STA | 550 | Store MSB in 5150 |
| 5032 | 3E,03 | | MVI | A,03 | 03 -> A |
| 5034 | 0E,08 | | MVI | C,08 | 08 -> C |
| 5036 | 21,50,51 | | LXI H | 5150 | Load 'HL' pair with 5150 |
| 5039 | CD,05,00 | | CALL | 0005 | Call device subroutine |
| 503C | C3,0E,50 | | JMP | 500E | Jump to 500E |

Result:

Thus the analog to digital conversion was done microprocessor.

ARITHMETIC OPERATIONS USING 8051

Aim:

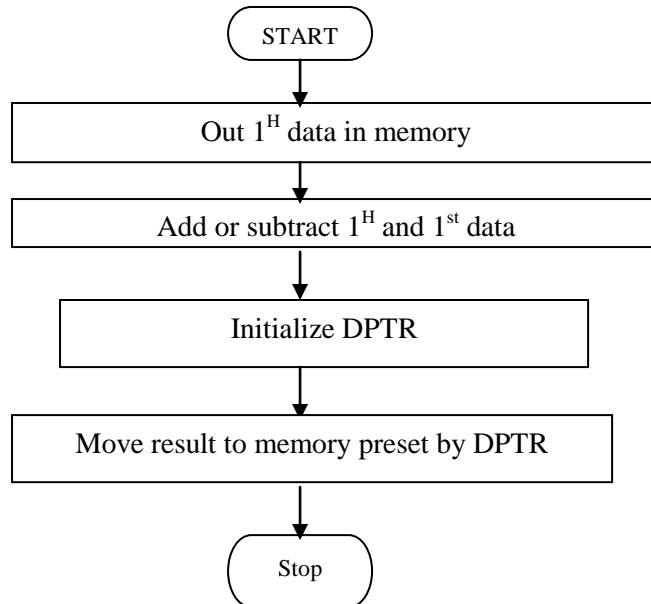
To do the arithmetic operations using 8051 microprocessor

Apparatus required:

8085 microprocessor kit
DAC interface kit
Keyboard

Algorithm:**Addition / Subtraction**

- Step 1 : Move 1^H data to memory
- Step 2 : Add or subtract 1^H data with 2nd data
- Step 3 : Initialize data pointer.
- Step 4 : Move result to memory pointed by DPTR.



Program: 8-bit Addition:

| Memory Location | Label | Opcode | Mnemonics | Comments |
|------------------------|--------------|---------------|------------------|--|
| 4100 | Start | C3 | CLR C | Clear the carry flag |
| 4101 | | 74DA | MOV A, # data 1 | Moves data 1 to register A |
| 4103 | | 24DA | ADD A, # data 2 | Add content of A and data 2 and store in A |
| 4105 | | 464500 | MOV DPTR, # 4500 | Moves data 4500 to DPTR |
| 4108 | | F0 | MOV A @ DPTR, A | Moves control of A to location pointed by DPTR |
| 4109 | | 80 FE | SJMP 4109 | Short jump to 4109 |

Execution:**Addition:**

| ML | Input |
|------|-------|
| 4103 | 0L |
| 4109 | 03 |

| ML | Output |
|------|--------|
| 4500 | 05 |

Program: 8-bit Subtraction:

| Memory Location | Label | Opcode | Mnemonics | Comments |
|------------------------|--------------|---------------|------------------|---|
| 4100 | Start | C3 | CLR C | Clear the carry flag |
| 4101 | | 74DA | MOV A, # data 1 | Moves data 1 to register A |
| 4103 | | 24DA | SUB B, # data 2 | Subtract data 2 from content of A and store result in A |
| 4105 | | 464500 | MOV DPTR, # 4500 | Moves 4500 to DPTR |
| 4108 | | F0 | MOV X @ DPTR, A | Moves result by location by DPTR |
| 4109 | | 80 FE | SJMP 4109 | Short jump to 4109 |

Execution:**Subtraction:**

| ML | Input |
|------|-------|
| 4101 | 05 |
| 4103 | 02 |

| ML | Output |
|------|--------|
| 4500 | 03 |

Result:

Thus 8-bit addition, subtraction is performed using 8051.

ARITHMETIC OPERATIONS USING 8051

Aim:

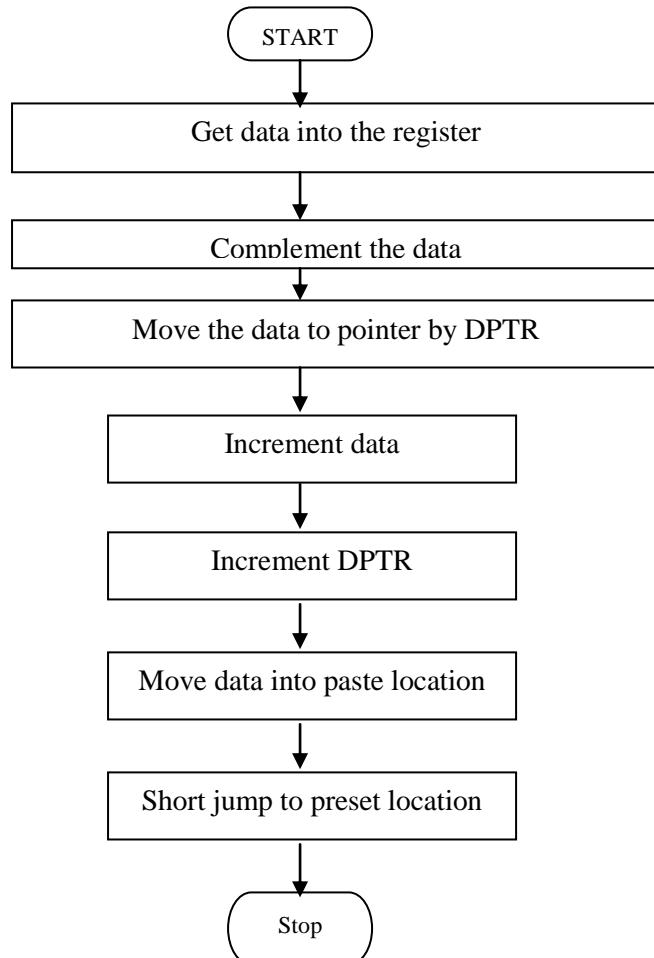
To do the arithmetic operations using 8051 microprocessor

Apparatus required:

8085 microprocessor kit
DAC interface kit
Keyboard

Algorithm:**Multiplication / Division**

- Step 1 : Get 1^H data and 2nd data to memory
- Step 2 : Multiply or divide 1^H data with 2nd data
- Step 3 : Initialize data pointer.
- Step 4 : Move result to memory pointed by DPTR (first port)
- Step 5 : Increment DPTR
- Step 6 : Move 2nd part of result to register A
- Step 7 : Move result to 2nd memory location pointer by DPTR



Program: 8-bit Multiplication:

| Memory Location | Label | Opcode | Mnemonics | Comments |
|------------------------|--------------|---------------|------------------|---|
| 4100 | Start | 7403 | MOV A, # data 1 | Move immediate data to accumulator |
| 4101 | | 75F003 | MOV B, # data 2 | Move 2 nd data to B register |
| 4105 | | A4 | MUL A B | Get the product in A & B |
| 4106 | | 904500 | MOV DPTR, # 4500 | Load data in 4500 location |
| 4109 | | F0 | MOV X @DPTR, A | Move A to ext RAM |
| 410B | | E5F0 | MOV A,B | Move 2 nd data in A |
| 410D | | F0 | MOV A @ DPTR | Same the ext RAM |
| 410E | | 80FE | SJMP 410E | Remain idle in infinite loop |

Execution:**Multiplication:**

| ML | Input | Output Address | Value |
|------|-------|----------------|-------|
| 4101 | 0L | 4500 | 08 |
| 4103 | 04 | | |

Program: 8-bit Division:

| Memory Location | Label | Opcode | Mnemonics | Comments |
|------------------------|--------------|---------------|------------------------|--------------------------------------|
| 4100 | Start | 7408 | MOV A, # data 1 | Move immediate data to accumulator |
| 4102 | | 75F002 | MOV B, @ data 2 DIV AB | Move immediate to B reg. |
| 4105 | | 84 | DIV AB | Divide content of A & B |
| 4106 | | 904500 | MOV DPTR, # 4500 | Load data pointer with 4500 location |
| 4109 | | F0 | MOV X @ DPTR, A | Move A to ext RAM |
| 410A | | A3 | INC DPTR | Increment data pointer |
| 410B | | ESF0 | MOV A,B | Move remainder to A |
| 410D | | F0 | MOV @ DPTR, A | Move A to ext RAM |
| 410E | | 80FE | SJMP 410E | Remain idle in infinite loop |

Execution:**Division:**

| ML | Input | Output Address | Value |
|------|-------|----------------|-------|
| 4101 | 08 | 4500 | 02 |
| 4103 | 04 | | |

Result:

Thus 8-bit multiplication & division is performed using 8051.

The 8085 Microprocessor

1. Draw the pin configuration and functional pin diagram of μ P 8085.

Ans. The pin configuration and functional pin diagram of μ P 8085 are shown below:

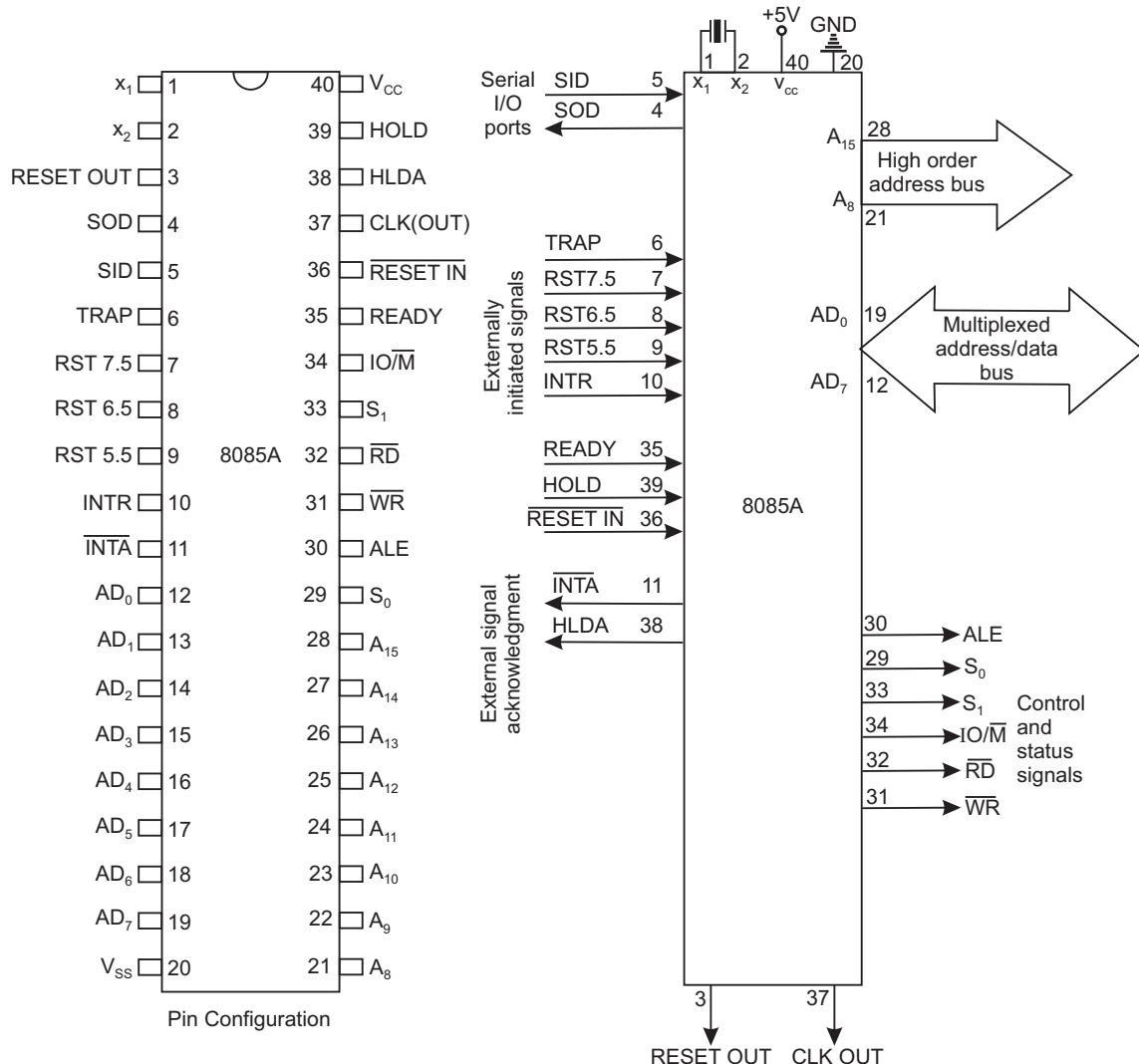


Fig. 2.1: Functional pin diagram

2. In how many groups can the signals of 8085 be classified?

Ans. The signals of 8085 can be classified into seven groups according to their functions. These are:

- (1) Power supply and frequency signals
- (2) Data and Address buses
- (3) Control bus
- (4) Interrupt signals
- (5) Serial I/O signals
- (6) DMA signals
- (7) Reset signals.

3. Draw the architecture of 8085 and mention its various functional blocks.

Ans. The architecture of 8085 is shown below:

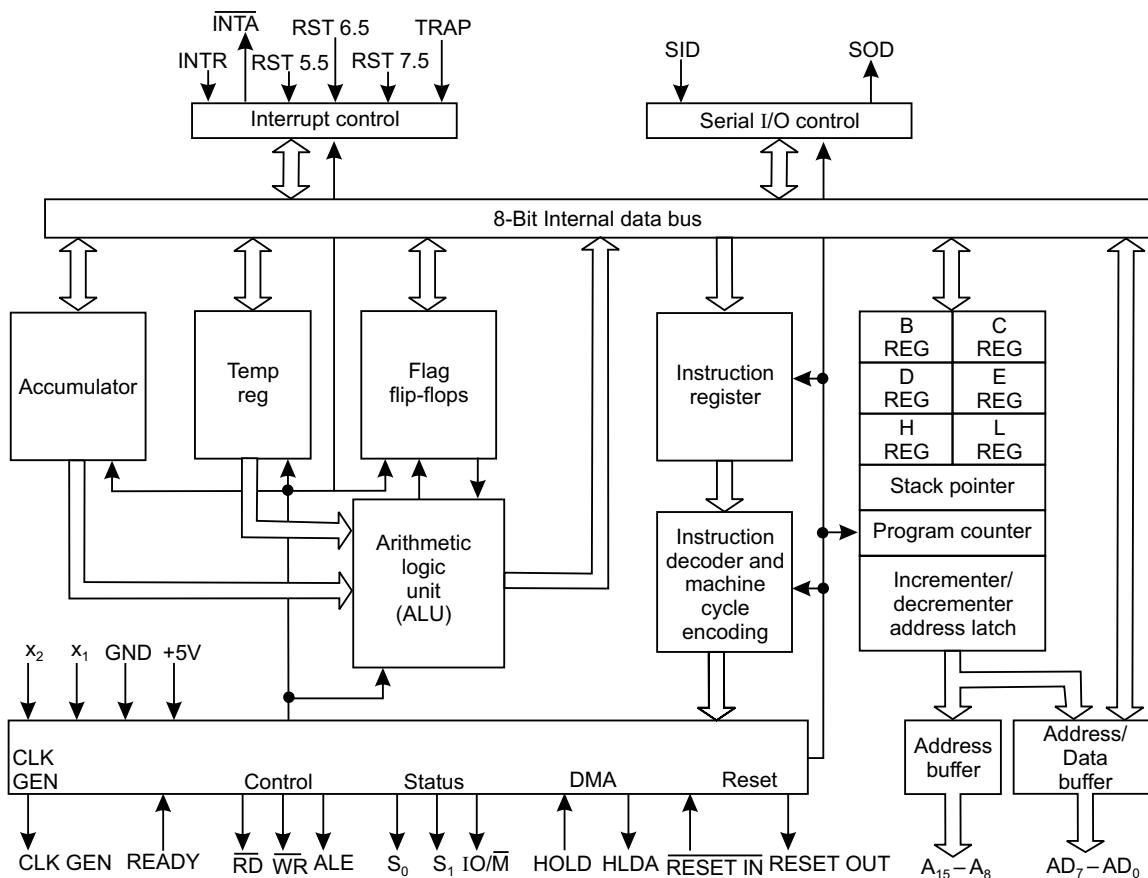


Fig. 2.2: Architecture of 8085

The various functional blocks of 8085 are as follows:

- Registers
- Arithmetic logic unit
- Address buffer
- Incrementer/decrementer address latch
- Interrupt control
- Serial I/O control
- Timing and control circuitry
- Instructions decoder and machine cycle encoder.

4. What is the technology used in the manufacture of 8085?

Ans. It is an NMOS device having around 6200 transistors contained in a 40 pin DIP package.

5. What is meant by the statement that 8085 is a 8-bit microprocessor?

Ans. A microprocessor which has n data lines is called an n-bit microprocessor i.e., the width of the data bus determines the size of the microprocessor. Hence, an 8-bit microprocessor like 8085 can handle 8-bits of data at a time.

6. What is the operating frequency of 8085?

Ans. 8085 operates at a frequency of 3 MHz, and the minimum frequency of operation is 500 kHz.

The version 8085 A-2 operates at a maximum frequency of 5 MHz.

7. Draw the block diagram of the built-in clock generator of 8085.

Ans. The built-in clock generator of 8085, in block schematic, is shown below:

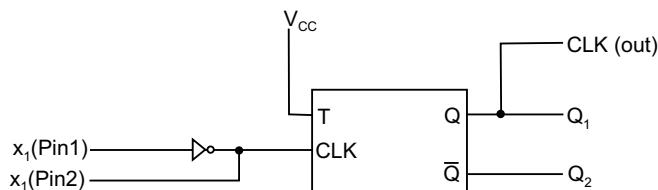


Fig. 2.3: Block diagram of built-in clock generator

The internal built-in clock generator, LC or RC tuned circuits, piezo-electric crystal or external clock source acts as an input to generate the clock. The T F/F, shown in Fig. 2.3 divides the input frequency by 2. Thus the output frequency of 8085 (obtained from pin 37) is half the input frequency.

8. What is the purpose of CLK signal of 8085?

Ans. The CLK (out) signal obtained from pin 37 of 8085 is used for synchronizing external devices.

9. Draw the different clock circuits which can be connected to pins 1 and 2 of 8085.

Ans. The different external clock circuits which can be connected to pins 1 and 2 of 8085 are shown below in Fig. 2.4:

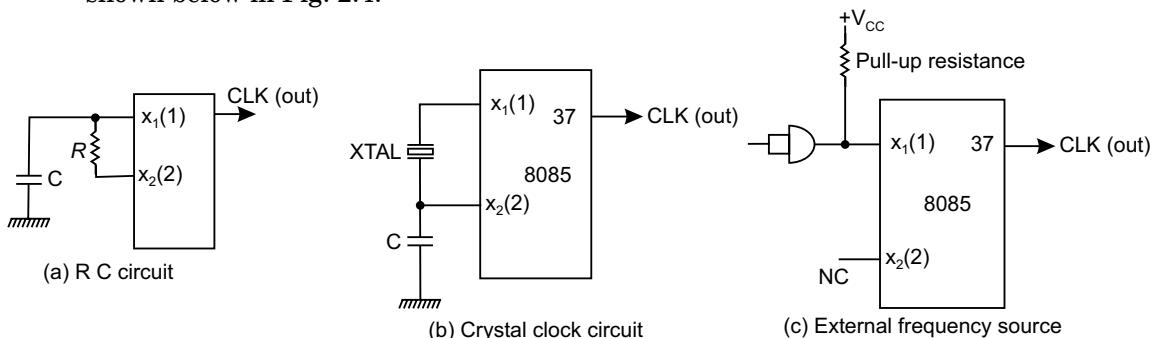


Fig. 2.4: The different external clock circuits

The output frequency obtained from pin 37 of Fig. 2.4(b) is more stable than the RC circuit of Fig. 2.4(a).

10. What are the widths of data bus (DB) and address bus (AB) of 8085?

Ans. The width of DB and AB of 8085 are 8-bits (1 byte) and 16-bits (2 bytes) respectively.

11. What is the distinguishing feature of DB and AB?

Ans. While the data bus is bidirectional in nature, the address bus is unidirectional.

Since the μ P can input or output data from within it, hence DB is bidirectional. Again the microprocessor addresses/communicates with peripheral ICs through the address bus, hence it is unidirectional, the address comes out via the AB of μ P.

12. The address capability of 8085 is 64 KB. Explain.

Ans. Microprocessor 8085 communicates via its address bus of 2-bytes width – the lower byte $AD_0 - AD_7$ (pins 12-19) and upper byte $D_8 - D_{15}$ (pins 21–28). Thus it can address a maximum of 2^{16} different address locations. Again each address (memory location) can hold 1 byte of data/instruction. Hence the maximum address capability of 8085 is

$$\begin{aligned} &= 2^{16} \times 1 \text{ Byte} \\ &= 65,536 \times 1 \text{ Byte} \\ &= 64 \text{ KB } (\text{where } 1 \text{ K} = 1024 \text{ bytes}) \end{aligned}$$

13. Does 8085 have serial I/O control?

Ans. 8085 has serial I/O control via its SOD and SID pins (pins 4 and 5) which allows it to communicate serially with external devices.

14. How many instructions 8085 can support?

Ans. 8085 supports 74 different instructions.

15. Mention the addressing modes of 8085.

Ans. 8085 has the following addressing modes: Immediate, Register, Direct, Indirect and Implied.

16. What jobs ALU of 8085 can perform?

Ans. The Arithmetic Logic Unit (ALU) of 8085 can perform the following jobs:

- 8-bit binary addition with or without carry.
- 16-bit binary addition.
- 2-digit BCD addition.
- 8-bit binary subtraction with or without borrow.
- 8-bit logical OR, AND, EXOR, complement (NOT function).
- bit shift operation.

17. How many hardware interrupts 8085 supports?

Ans. It supports five (5) hardware interrupts—TRAP, RST 7.5, RST 6.5, RST 5.5 and INTR.

18. How many I/O ports can 8085 access?

Ans. It provides 8-bit I/O addresses. Thus it can access $2^8 = 256$ I/O ports.

19. Why the lower byte address bus ($A_0 - A_7$) and data bus ($D_0 - D_7$) are multiplexed?

Ans. This is done to reduce the number of pins of 8085, which otherwise would have been a 48 pin chip. But because of multiplexing, external hardware is required to demultiplex the lower byte address cum data bus.

20. List the various registers of 8085.

Ans. The various registers of 8085, their respective quantities and capacities are tabulated below:

Table 2.1: List of Various Registers in 8085

| S. No. | Name of the Register | Quantity | Capacity |
|--------|---|----------|------------|
| 1. | Accumulator (or) Register A | 1 | 8-bit |
| 2. | Temporary register | 1 | 8-bit |
| 3. | General purpose registers (B, C, D, E, H and L) | 6 | 8-bit each |
| 4. | Stack pointer (SP) | 1 | 16-bit |
| 5. | Program counter (PC) | 1 | 16-bit |
| 6. | Instruction register | 1 | 8-bit |
| 7. | Incrementer/Decrementer address latch | 1 | 16-bit |
| 8. | Status flags register | 1 | 8-bit |

21. Describe the accumulator register of 8085.

Ans. This 8-bit register is the most important one amongst all the registers of 8085. Any data input/output to/from the microprocessor takes place via the accumulator (register). It is generally used for temporary storage of data and for the placement of final result of arithmetic/logical operations.

Accumulator (ACC or A) register is extensively used for arithmetic, logical, store and rotate operations.

22. What are the temporary registers of 8085?

Ans. The temporary registers of 8085 are temporary data register and W and Z registers. These registers are not available to the programmer, but 8085 uses them internally to hold temporary data during execution of some instructions.

23. Describe W and Z registers of 8085.

Ans. W and Z are two 8-bit temporary registers, used to hold 8-bit data/address during execution of some instructions.

CALL-RET instructions are used in subroutine operations. On getting a CALL in the main program, the current program counter content is pushed into the stack and loads the PC with the first memory location of the subroutine. The address of the first memory location of the subroutine is temporarily stored in W and Z registers.

Again, XCHG instruction exchanges the contents H and L with D and E respectively. W and Z registers are used for temporary storage of such data.

24. Describe the temporary data register of 8085.

Ans. The temporary data register of 8085 is an 8-bit register, which is not available to the programmer, but is used internally for execution of most of the arithmetic and logical operations.

ADD D instruction adds the contents of accumulator with the content of D. The content of D is temporarily brought into the temporary data register. Thus the two inputs to the ALU are—one from the accumulator and the other from the temporary data register. The result is stored in the accumulator.

25. Describe the general purpose registers of 8085?

Ans. The general purpose registers of 8085 are: B, C, D, E, H and L. They are all 8-bit registers but can also be used as 16-bit register pairs—BC, DE and HL. These registers are also known as scratch pad registers.

26. In what other way HL pair can be used?

Ans. HL register pair can be used as a data pointer or memory pointer.

27. Mention the utility of the general purpose registers.

Ans. General purpose registers store temporary data during program execution, which can also be stored in different accessible memory locations. But storing temporary data in memory requires bus access—hence more time is needed to store. Thus it is always advisable to store data in general purpose registers.

The more the number of general purpose registers, the more is flexibility in programming—so a microprocessor having more such registers is always advantageous.

28. Which are the sixteen bit registers of 8085?

Ans. 8085 has three (3) sixteen bit registers—Program Counter (PC), Stack Pointer (SP) and Incrementer/Decrementer address latch register.

29. Discuss the two registers program counter and stack pointer.

Ans. Program counter (PC) is a sixteen bit register which contains the address of the instruction to be executed just next. PC acts as a address pointer (also known as memory pointer) to the next instruction. As the processor executes instructions one after another, the PC is incremented—the number by which the PC increments depends on the nature of the instruction. For example, for a 1-byte instruction, PC is incremented by one, while for a 3-byte instruction, the processor increments PC by three address locations.

Stack pointer (SP) is a sixteen bit register which points to the ‘stack’. The stack is an area in the R/W memory where temporary data or return addresses (in cases of subroutine CALL) are stored. Stack is a auto-decrement facility provided in the system. The stack top is initialised by the SP by using the instruction LXI SP, memory address.

In the memory map, the program should be written at one end and stack should be initialised at the other end of the map—this is done to avoid crashing of program. If sufficient

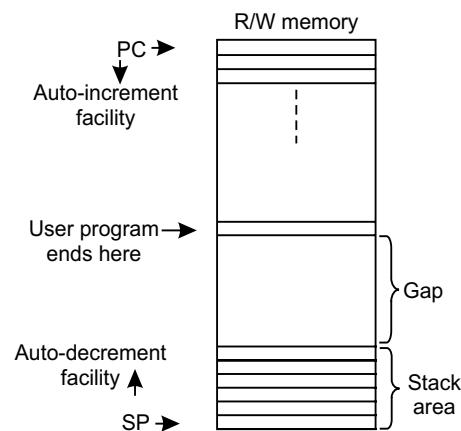


Fig. 2.5: Auto-increment and auto-decrement facility for PC and SP respectively

gap is not maintained between program memory location and stack, then when the stack gets filled up by PUSH or subroutine calls, the stack top may run into the memory area where program has been written. This is shown in Fig. 2.5.

30. Describe the instruction register of 8085.

Ans. Program written by the programmer resides in the R/W memory. When an instruction is being executed by the system, the opcode of the instruction is fetched from the memory and stored in the instruction register. The opcode is loaded into the instruction register during opcode fetch cycle. It is then sent to the instruction decoder.

31. Describe the (status) flag register of 8085.

Ans. It is an 8-bit register in which five bit positions contain the status of five condition flags which are Zero (Z), Sign (S), Carry (CY), Parity (P) and Auxiliary carry (AC). Each of these five flags is a 1 bit F/F. The flag register format is shown in Fig. 2.6:

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|----|----|----|----|----|----|----|----|
| S | Z | X | AC | X | P | X | CY |

Fig. 2.6: The flag register format

- *Sign (S) flag*: – If the MSB of the result of an operation is 1, this flag is set, otherwise it is reset.
- *Zero (Z) flag*:– If the result of an instruction is zero, this flag is set, otherwise reset.
- *Auxiliary Carry (AC) flag*:– If there is a carry out of bit 3 and into bit 4 resulting from the execution of an arithmetic operation, it is set otherwise reset.

This flag is used for BCD operation and is not available to the programmer to change the sequence of an instruction.

- *Carry (CY) flag*:– If an instruction results in a carry (for addition operation) or borrow (for subtraction or comparison) out of bit D₇, then this flag is set, otherwise reset.
- *Parity (P) flag*:– This flag is set when the result of an operation contains an even number of 1's and is reset otherwise.

32. State the characteristics of the flag register.

Ans. The following are the characteristics of flag register:

- It is an 8-bit register.
- It contains five flags—each of one bit.
- The flag register can't be written into.

33. What is the purpose of incrementer/decrementer address latch register?

Ans. This 16-bit register increments/decrements the contents of PC or SP when instructions related to them are executed.

34. Mention the blocks on which ALU operates?

Ans. The ALU functions as a part which includes arithmetic logic group of circuits. This includes accumulator, flags F/Fs and temporary register blocks.

35. What is the function of the internal data bus?

Ans. The width of the internal data bus is 8-bit and carries instructions/data between the CPU registers. This is totally separate from the external data bus which is connected to memory chips, I/O, etc.

The internal and external data bus are connected together by a logic called a bidirectional bus (transceiver).

36. Describe in brief the timing and control circuitry of 8085.

Ans. The T&C section is a part of CPU and generates timing and control signals for execution of instructions. This section includes Clock signals, Control signals, Status signals, DMA signals as also the Reset section. This section controls fetching and decoding operations. It also generates appropriate control signals for instruction execution as also the signals required to interface external devices.

37. Mention the following:

- (a) **Control and Status signals**
- (b) **Interrupt signals**
- (c) **Serial I/O signals**
- (d) **DMA signals**
- (e) **Reset signals.**

Ans. The control and status signals are ALE, \overline{RD} , \overline{WR} , IO/M, S₀, S₁ and READY.

The interrupt signals are TRAP, RST 7.5, RST 6.5, RST 5.5, INTR. \overline{INTA} is an interrupt acknowledgement signal indicating that the processor has acknowledged an INTR interrupt.

Serial I/O signals are SID and SOD

DMA signals are HOLD and HLDA

Reset signals are RESET IN and RESET OUT.

38. What is the function of ALE and how does it function?

Ans. Pin 30 of 8085 is the ALE pin which stands for ‘Address Latch Enable’. ALE signal is used to demultiplex the lower order address bus (AD₀ – AD₇).

Pins 12 to 19 of 8085 are AD₀ – AD₇ which is the multiplexed address-data bus. Multiplexing is done to reduce the number of pins of 8085.

Lower byte of address (A₀ – A₇) are available from AD₀ – AD₇ (pins 12 to 19) during T₁ of machine cycle. But the lower byte of address (A₀ – A₇), along with the upper byte A₈ – A₁₅ (pins 21 to 28) must be available during T₂ and rest of the machine cycle to access memory location or I/O ports.

Now ALE signal goes high at the beginning of T₁ of each machine cycle and goes low at the end of T₁ and remains low during the rest of the machine cycle. This high to low transition of ALE signal at the end of T₁ is used to latch the lower order address byte (A₀ – A₇) by the latch IC 74LS373, so that the lower byte A₀ – A₇ is continued to be available till the end of the machine cycle. The situation is explained in the following figure:

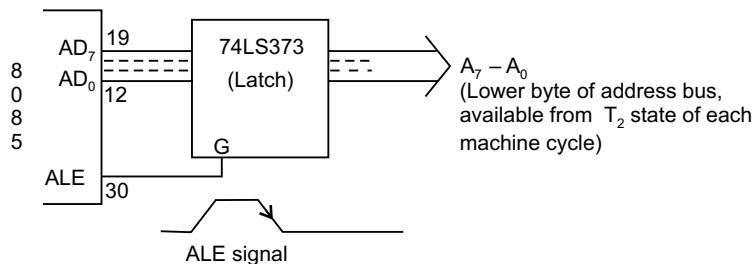


Fig. 2.7: Lower byte of address latching achieved by the H to L transition of ALE signal, which occurs at the end of T₁ of each machine cycle

39. Explain the function of the two DMA signals HOLD and HLDA.

Ans. DMA mode of data transfer is fastest and pins 39 and 38 (HOLD and HLDA) become active only in this mode.

When DMA is required, the DMA controller IC (8257) sends a 1 to pin 39 of 8085. At the end of the current instruction cycle of the microprocessor it issues a 1 to pin 38 of the controller. After this the bus control is totally taken over by the controller.

When 8085 is active and 8257 is idle, then the former is MASTER and the latter is SLAVE, while the roles of 8085 and 8257 are reversed when 8085 is idle and 8257 becomes active.

40. Discuss the three signals IO/M̄, S₀ and S₁.

Ans. IO/M̄ signal indicates whether I/O or memory operation is being carried out. A high on this signal indicates I/O operation while a low indicates memory operation. S₀ and S₁ indicate the type of machine cycle in progress.

41. What happens when RESET IN signal goes low?

Ans. RESET IN is an input signal which is active when its status is low. When this pin is low, the following occurs:

- The program counter is set to zero (0000_H).
- Interrupt enable and HLDA F/Fs are resetted.
- All the buses are tri-stated.
- Internal registers of 8085 are affected in a random manner.

42. Is there any minimum time required for the effective RESET IN signal?

Ans. For proper resetting to take place, the reset signal RESET IN must be held low for at least 3 clock cycles.

43. Indicate the function of RESET OUT signal.

Ans. When this signal is high, the processor is being reset. This signal is synchronised to the processor clock and is used to reset other devices which need resetting.

44. Write the advantages/disadvantages of having more number of general purpose registers in a microprocessor.

Ans. Writing of a program becomes more convenient and flexible by having more number of general purpose registers.

But there are certain disadvantages of having more GPRs. These are as follows:

The more the number of GPRs in a microprocessor, more number of bits would be required to identify individual registers. This would reduce the number of operations that can be provided by the microprocessor.

In programs involving subroutine CALL, if more GPRs are involved, then their status are to be saved in stack and on return from the subroutine, they are to be restored from the stack. This will thus put considerable overhead on the microprocessor.

If more number of GPRs are used in a microprocessor, considerable area of the chip is used up in accommodating the GPRs. Thus there may be some problem in implementing other functions on the chip.

45. Draw the lower and higher order address bus during the machine cycles.

Ans. The lower byte of address ($AD_0 - AD_7$) is available on the multiplexed address/data bus during T_1 state of each machine cycle, except during the bus idle machine cycle, shown in Fig. 2.8.

The higher byte of address ($A_8 - A_{15}$) is available during T_1 to T_3 states of each machine cycle, except during the bus idle machine cycle, shown in Fig. 2.9.

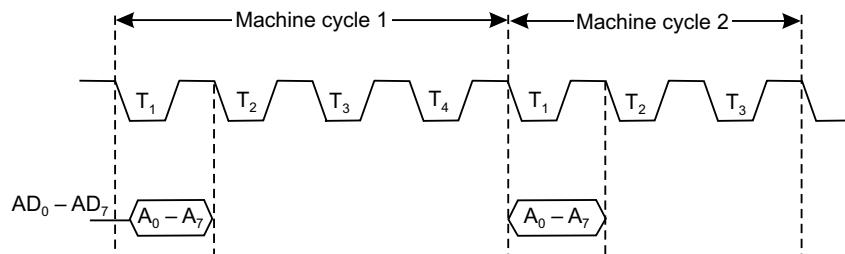


Fig. 2.8: Lower byte address on the multiplexed bus

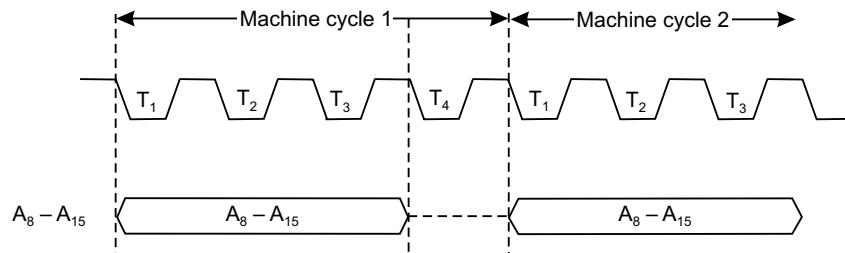


Fig. 2.9: Higher byte address on $A_8 - A_{15}$

46. Draw the appearance of data in the read and write machine cycles.

Ans. Data transfer from memory or I/O device to microprocessor or the reverse takes place during T_2 and T_3 states of the machine cycles.

In the read machine cycle, data appears at the beginning of T_3 state, whereas in the write machine cycle, it appears at the beginning of T_2 , shown in Fig. 2.10.

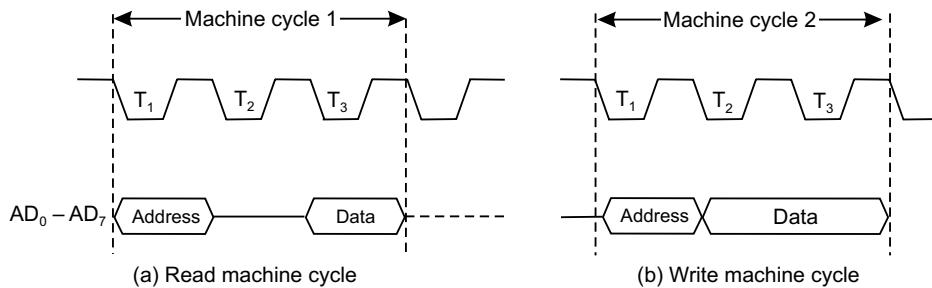


Fig. 2.10: Data bus

47. Draw the status signals during opcode fetch and memory read machine cycles.

Ans. The status signals are $\overline{IO/M}$, S_0 and S_1 . Their conditions indicate the type of machine cycle that the system is currently passing through. These three status signals remain active right from the beginning till the end of each machine cycle, shown in Fig. 2.11.

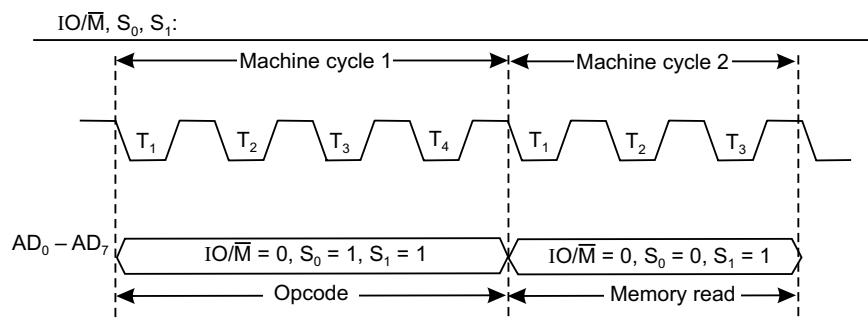


Fig. 2.11: Status signals

48. Show the \overline{RD} and \overline{WR} signals during the Read cycle and Write cycle.

Ans. When \overline{RD} is active, microprocessor reads data from either memory or I/O device while when \overline{WR} is active, it writes data into either memory or I/O device.

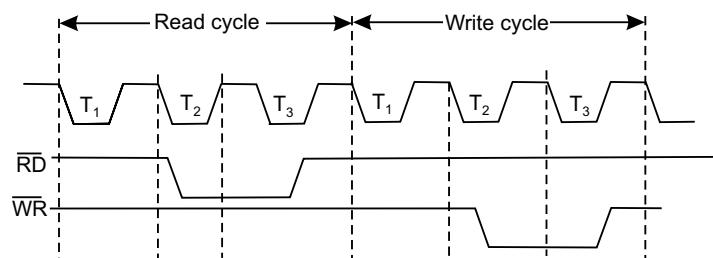


Fig. 2.12: \overline{RD} and \overline{WR} signals

Data transfer (reading/writing) takes place during T_2 and T_3 states of read cycle or write cycle and is shown in Fig. 2.12.

49. Indicate the different machine cycles of 8085.

Ans. 8085 has seven different machine cycles. These are:

- (1) Opcode Fetch
- (2) Memory Read
- (3) Memory Write
- (4) I/O Read
- (5) I/O Write
- (6) Interrupt Acknowledge
- (7) Bus Idle.

50. Draw the Opcode Fetch machine cycle of 8085 and discuss.

Ans. The first machine cycle of every instruction is the Opcode Fetch. This indicates the kind of instruction to be executed by the system. The length of this machine cycle varies between 4T to 6T states—it depends on the type of instruction. In this, the processor places the contents of the PC on the address lines, identifies the nature of machine cycle (by $\overline{IO/M}$, S_0 , S_1) and activates the ALE signal. All these occur in T_1 state.

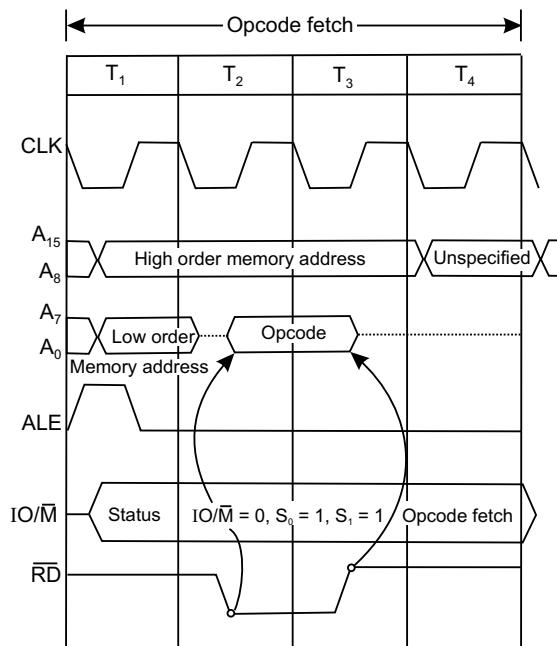


Fig. 2.13: Opcode fetch machine cycle

In T_2 state, \overline{RD} signal is activated so that the identified memory location is read from and places the content on the data bus ($D_0 - D_7$).

In T_3 , data on the data bus is put into the instruction register (IR) and also raises the \overline{RD} signal thereby disabling the memory.

In T_4 , the processor takes the decision, on the basis of decoding the IR, whether to enter into T_5 and T_6 or to enter T_1 of the next machine cycle.

One byte instructions that operate on eight bit data are executed in T_4 . Examples are ADD B, MOV C, B, RRC, DCR C, etc.

51. Briefly describe Memory Read and Write machine cycles and show the waveforms.

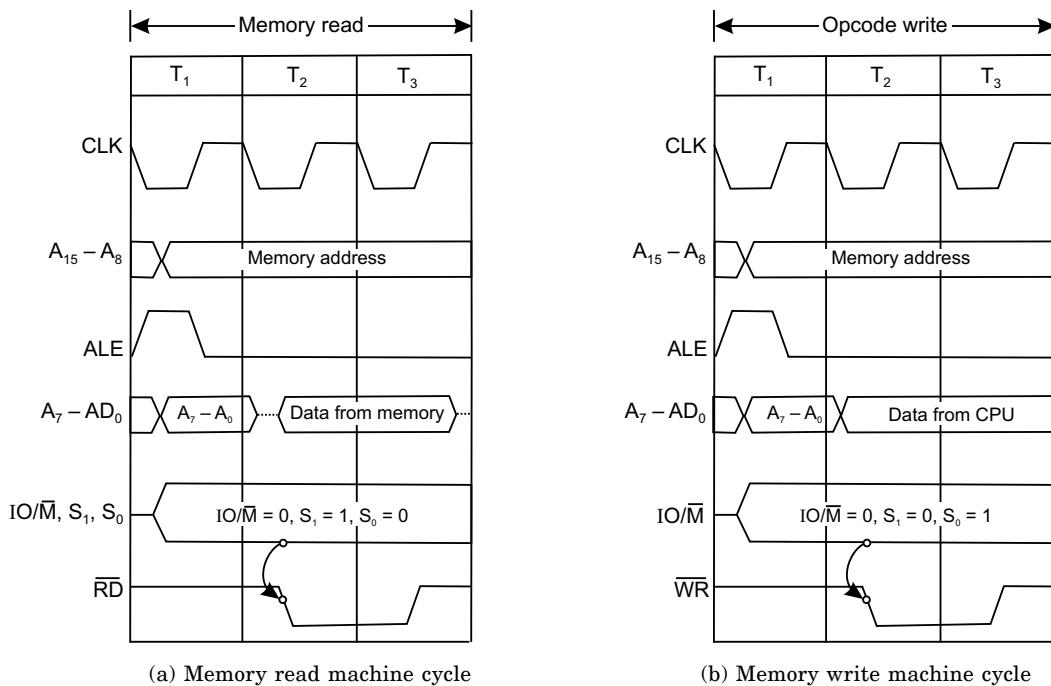


Fig. 2.14: Memory read and write machine cycle

Ans. Both the Memory Read and Memory Write machine cycles are 3T states in length. In Memory Read the contents of R/W memory (including stack also) or ROM are read while in Memory Write, it stores data into data memory (including stack memory).

As is evident from Fig. 2.14 during T₂ and T₃ states data from either memory or CPU are made available in Memory Read or Memory Write machine cycles respectively. The status signal (IO/M, S₀, S₁) states are complementary in nature in Memory Read and Memory Write cycles. Reading or writing operations are performed in T₂.

In T₃ of Memory Read, data from data bus are placed into the specified register (A, B, C, etc.) and raises RD so that memory is disabled while in T₃ of Memory Write WR signal is raised which disables the memory.

52. Draw the I/O Read and I/O Write machine cycles and discuss.

Ans. I/O Read and Write machine cycles are almost similar to Memory Read and Write machine cycles respectively. The difference here is in the IO/M signal status which remains 1 indicating that these machine cycles are related to I/O operations. These machine cycles take 3T states.

In I/O read, data are available in T₂ and T₃ states, while during the same time (T₂ and T₃) data from CPU are made available in I/O write.

The I/O read and write machine cycles are shown in Fig. 2.15.

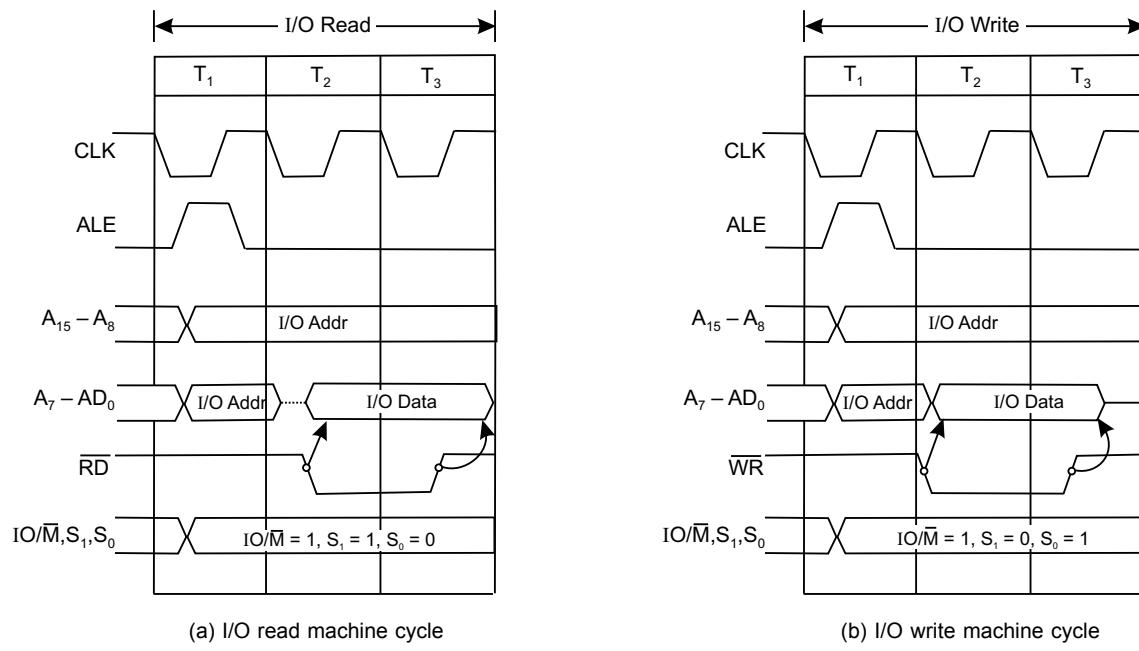


Fig. 2.15: I/O read and write machine cycles

53. Draw the Interrupt Acknowledge cycles for (a) RST instruction (b) CALL instruction.

Ans. The following figure shows the Interrupt Acknowledge cycle for RST instruction.

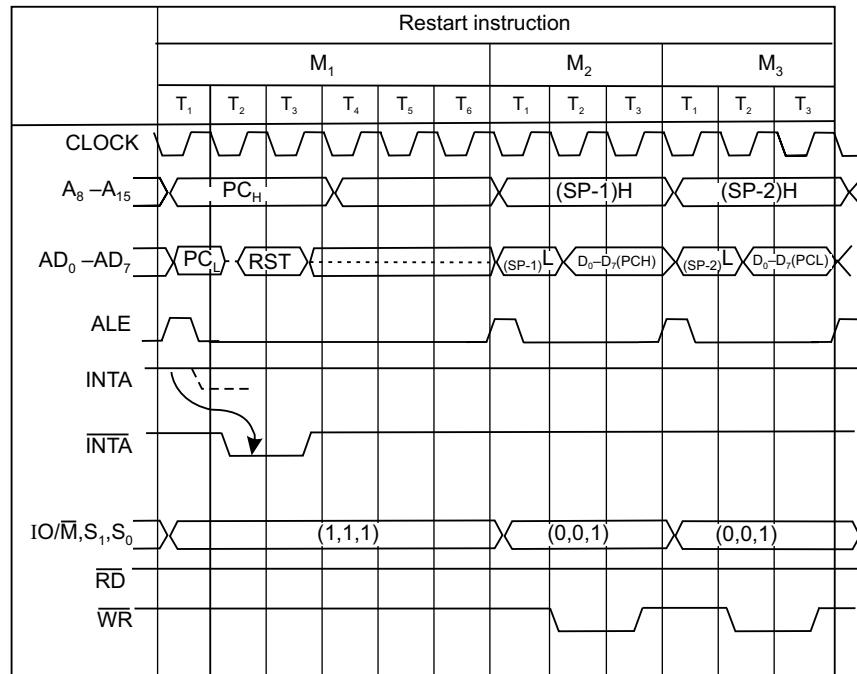


Fig. 2.16: Restart instruction

In M₁, RST is decoded. This initiates a CALL to the specific vector location. Contents of the PC are stored in stack in machine cycles M₂ and M₃.

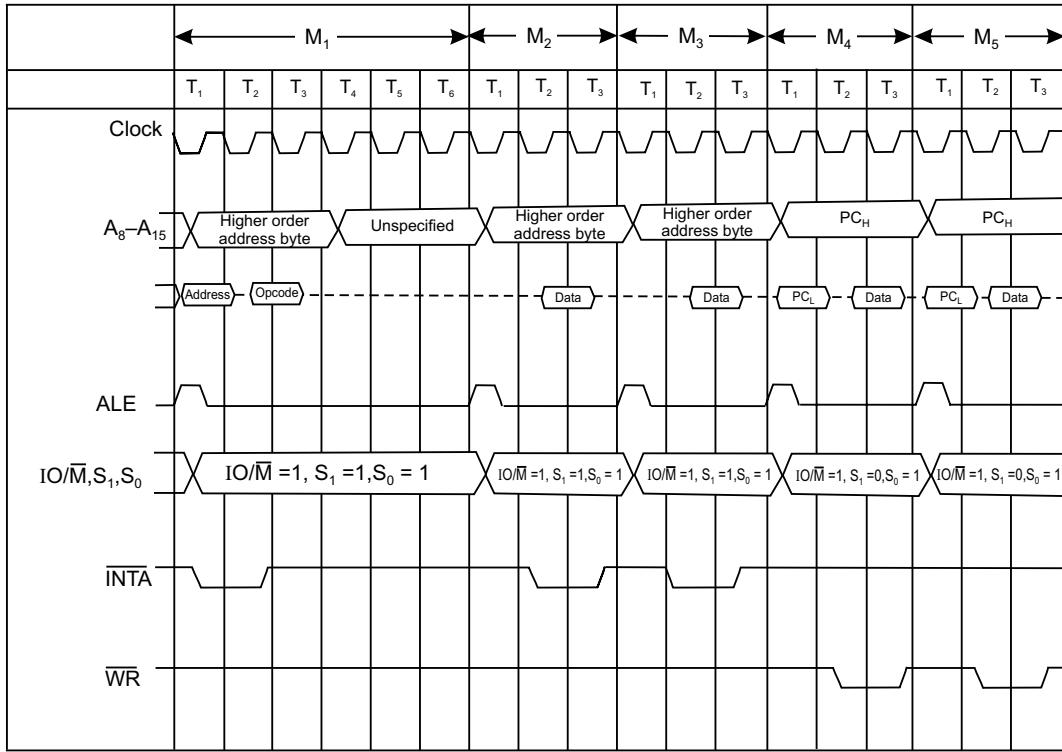


Fig. 2.17: Timing diagram of INTA machine cycle and execution of call instruction

The above figure shows an Interrupt Acknowledge cycle for CALL instruction. M₂ and M₃ machine cycles are required to call the 2 bytes of the address following the CALL. Memory write are done in machine cycles M₄ and M₅ in which contents of PC are stored in stack and then a new instruction cycle begins.

54. What is meant by Bus Idle Machine cycle?

Ans. There are a few situations in which machine cycles are neither Read or Written into. These are called Bus Idle Machine cycle.

Such situations arise when the system executes a DAD or during the internal opcode generation for the RST or TRAP interrupts.

The ALE signal changes state during T₁ of each machine cycle, but in Bus Idle Machine cycles, ALE does not change state.

55. Explain the DAD instruction and draw its timing diagram.

Ans. DAD instruction adds the contents of a specified register pair to the contents of H and L.

For execution of DAD, 10 T-states are needed. Instead of having a single machine cycle having 10 T-states, it consists of the Opcode Fetch machine cycle (4T states) and 6 extra T-states divided into two machine cycles. These two extra machine cycles are Bus Idle Machine cycles which do not involve either memory or I/O.

The timing diagram for DAD instruction is shown below:

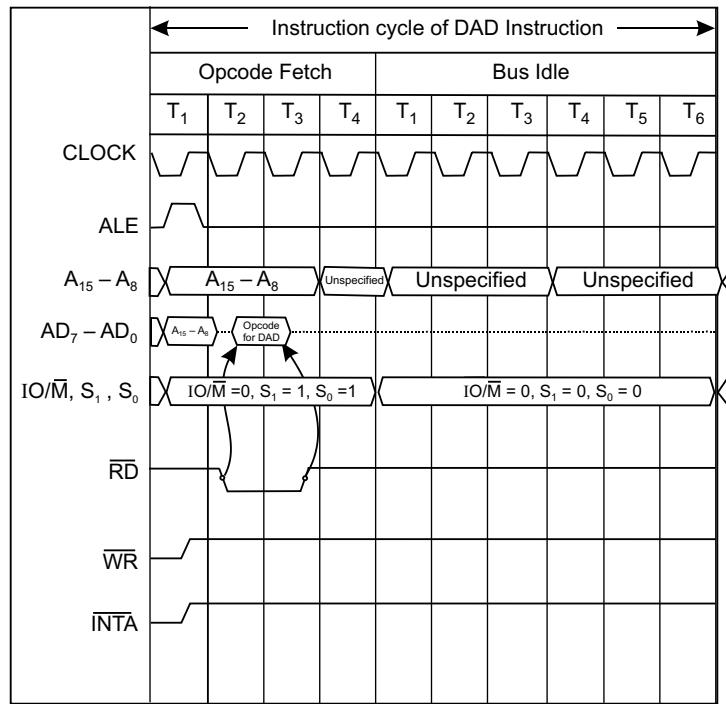


Fig. 2.18: Timing diagram for DAD instruction

56. Discuss the concept of WAIT states in microprocessors.

Ans. So many times it may happen that there is speed incompatibility between microprocessor and its memory and I/O systems. Mostly the microprocessor is having higher speed.

So in a given situation, if the microprocessor is ready to accept data from a peripheral device while there is no valid data in the device (e.g. an ADC), then the system enters into WAIT states and the READY pin (an input pin to the microprocessor, pin no. 35 for 8085) is put to a low state by the device.

Once the device becomes ready with some valid data, it withdraws the low state on the READY pin of 8085. Then 8085 accepts the data from the peripheral by software instructions.

57. Does 8085 have multiplication and division instructions?

Ans. No, 8085 does not have the above two instructions. It can neither multiply nor divide two 8-bit numbers. The same are executed by the processor following the process of repetitive addition or subtraction respectively.

58. Indicate the bus drive capability of 8085.

Ans. 8085 buses can source up to 400 μ A and sink 2 mA of current. Hence 8085 buses can drive a maximum of one TTL load.

Thus the buses need bus drivers/buffers to enhance the driving capability of the buses to ensure that the voltage levels are maintained at appropriate levels and malfunctioning is avoided.

59. What are the buffers needed with the buses of 8085?

Ans. An 8-bit unidirectional buffer 74LS244 is used to buffer the higher order address bus ($A_8 - A_{15}$). It consists of eight non-inverting buffers with tri-state outputs. Each pin can sink 24 mA and source 15 mA of current.

A bidirectional buffer 74LS245 (also called octal bus transreceivers) can be used to drive the bidirectional data bus ($D_0 - D_7$) after its demultiplexing. The DIR pin of the IC controls the direction of flow of data through it.

60. Explain the instruction cycle of a microprocessor.

Ans. When a processor executes a program, the instructions (1 or 2 or 3 bytes in length) are executed sequentially by the system. The time taken by the processor to complete one instruction is called the Instruction Cycle (IC).

An IC consists of Fetch Cycle (FC) and an Execute Cycle (EC). Thus $IC = FC + EC$. It is shown in Fig. 2.19. Depending on the type of instruction, IC time varies.

61. Explain a typical fetch cycle (FC).

Ans. The time required to fetch an opcode from a memory location is called Fetch Cycle.

A typical FC may consist of 3T states. In the first T-state, the memory address, residing in the PC, is sent to the memory. The content of the addressed memory (i.e., the opcode residing in that memory location) is read in the second T-state, while in the third T-state this opcode is sent via the data bus to the instruction register (IR). For slow memories, it may take more time in which case the processor goes into ‘wait cycles’. Most microprocessors have provision of wait cycles to cope with slow memories.

A typical FC may look like the following:

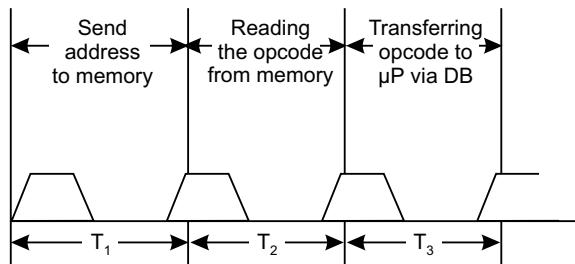


Fig. 2.20: A typical FC

62. Draw the block schematic of a typical Instruction Word flow diagram and explain the same.

Ans. There are two kinds of words—instruction word and data word. The 2 byte content of the PC is transferred to a special register—called memory address register (MAR) or simply address register (AR) at the beginning of the fetch cycle. Since the content of MAR is an address, it is thus sent to memory via the address bus. The content of the addressed memory is then read under ‘Read control’ generated by T&C section of the

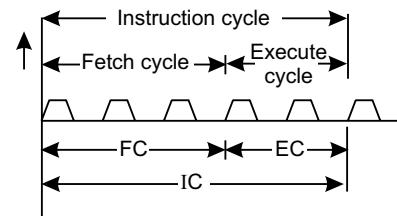


Fig. 2.19: Instruction cycle showing FC, EC and IC

microprocessor. This is then sent via the data bus to the memory data register (MDR) or simply data register (DR) existing in CPU. This is placed in the instruction register (IR) and is decoded by the instruction decoder and subsequently executed. The PC is then incremented if the subsequent memory location is to be accessed.

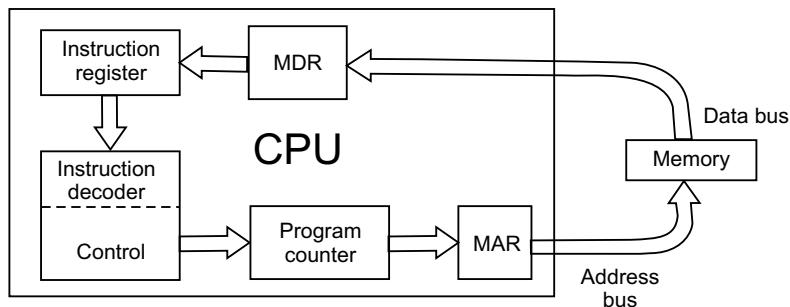


Fig. 2.21: Flow of instruction word

63. Draw the block schematic of a typical data word flow diagram and explain the same.

Ans. The data word flows via the data bus into the accumulator. The data source can be a memory device or an input device. Data from the accumulator is then manipulated in ALU under control of T & C unit. The manipulated data is then put back in the accumulator and can be sent to memory or output devices.

The block schematic of data flow is shown below.

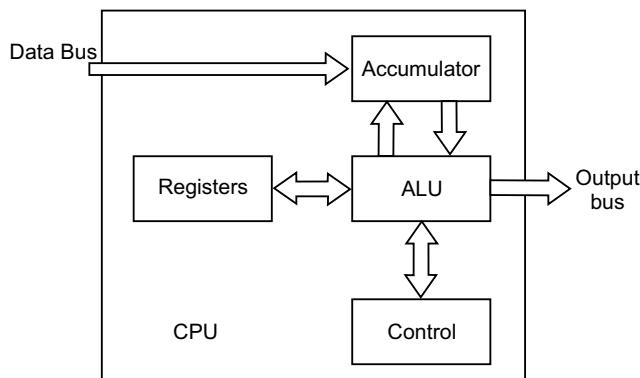


Fig. 2.22: Flow of data word

64. Why a microprocessor based system is called a sequential machine?

Ans. It can perform the jobs in a sequential manner, one after the other. That is why it is called a sequential machine.

65. Why a microprocessor based system is called a synchronous one?

Ans. All activities pertaining to the pP takes place in synchronism with the clock. Hence it is called a synchronous device.

66. Draw the diagram which will show the three buses separately, with the help of peripheral ICs.

Ans. The scheme of connections is shown below. The octal bus driver 74LS244 drives the higher order address bus $A_{15} - A_8$ while 74LS373 Latch drives the lower order address bus $A_7 - A_0$ (with the help of ALE signal). The bidirectional bus driver 74LS245 drives the data bus $D_7 - D_0$ while the 74LS138 (a 3 to 8 decoder chip) outputs at its output pins \overline{IOW} , \overline{IOR} , \overline{MEMW} , \overline{MEMR} signals from the $\overline{IO/M}$, \overline{RD} and \overline{WR} signals of 8085.

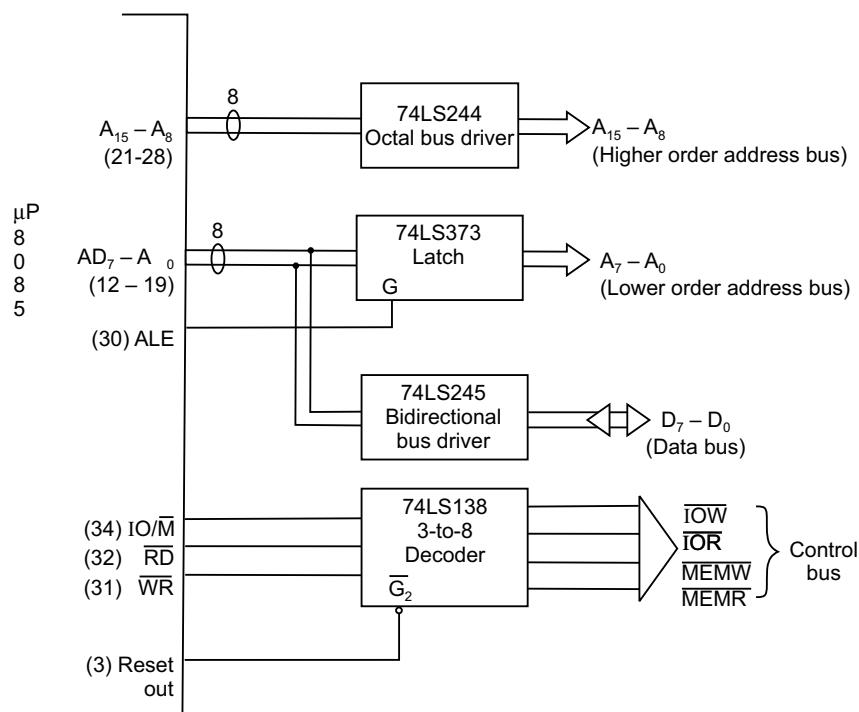


Fig. 2.23: Demultiplexing of address/data bus and separation of control signals

67. Discuss the status of \overline{WR} and \overline{RD} signals of 8085 at any given instant of time.

Ans. At any given instant, the status of the two signals \overline{WR} and \overline{RD} will be complementary to each other.

It is known that microprocessor based system is a sequential device, so that at any given point of time, it does the job of reading or writing—and definitely not the two jobs at the same time. Hence if \overline{WR} signal is low, then \overline{RD} signal must be high or vice versa.

68. Which registers of 8085 are programmable?

Ans. The registers B, C, D, E, H and L are programmable registers in that their contents can be changed by programming.

69. Suggest the type of operations possible on data from a look of the architecture of 8085.

Ans. The architecture of 8085 suggests that the following operations are possible.

- Can store 8-bits of data.
- Uses the temporary registers during arithmetic operations.
- Checks for the condition of resultant data (like carry, etc.) from the flag register.

70. What are the externally initiated operations supported by 8085?

Ans. 8085 supports several externally initiated operations at some of its pins. The operations possible are:

- Resetting (via Reset pin)
- Interruptions (via Trap, RST 7.5, RST 6.5, RST 5.5 and Interrupt pin)
- Ready (via Ready pin)
- Hold (via Hold pin)

71. Name the registers not accessible to the programmer.

Ans. The following registers cannot be accessed by the programmer:

- Instruction register (IR)
- Memory address register (MAR) or supply address register (AR)
- Temporary registers.

72. Name the special purpose registers of 8085.

Ans. The special purpose registers used in 8085 microprocessor are:

- Accumulator register (A)
- Program counter register (PC)
- Status (or Flag) register
- Stack pointer register (SP)

73. What should be the size of the Instruction Register if an arbitrary microprocessor has only 25 instructions?

Ans. The length of the Instruction Register would be 5-bits, since $2^5 = 32$, since a 5-bit Instruction Register can decode a maximum of 32 instructions.

74. Explain the difference between HLT and HOLD states.

Ans. HLT is a software instruction. Its execution stops the processor which enters into a HALT state and the buses of the processor are driven into tri-state.

HOLD is a hardware input to the processor. When HOLD input = 1, the processor goes into the HOLD state, but the buses don't go into tri-state. The processor gives out a high HLDA (hold acknowledge) signal which can be utilised by an external device (like a DMA controller) to take control of the processor buses. HOLD is acknowledged by the processor at the end of the current instruction execution.

75. Indicate the length of the Program Counter (PC) to access 1 KB and 1 MB memory.

Ans. 1 KB = 1024 bytes

and 1 MB = 1024 K bytes

Thus the required number of bits in PC to access 1 KB are 10 ($2^{10} = 1024$) and 20 ($2^{20} = 1024$ K) respectively.

76. What determines the number of bytes to be fetched from memory to execute an instruction?

Ans. An instruction normally consists of two fields. These are:

| | |
|--------|---------|
| Opcode | Operand |
|--------|---------|

Thus, while the system starts executing an instruction, it first decodes the opcode which then decides how many more bytes are to be brought from the memory—its minimum value is zero (like RAR) while the maximum value is two (like STA 4059 H).

77. A Microprocessor's control logic is 'micropogrammed'. Explain.

Ans. It implies that the architecture of the control logic is much alike the architecture of a very special purpose microprocessor.

78. Does the ALU have any storage facility?

Ans. No, it does not have any storage facility. For this reason, the need for temporary data registers arise in ALU—it has two inputs: one provided by the accumulator and the other from the temporary data register. The result of summation is stored in the accumulator.

3

Instruction Types and Timing Diagrams

1. What is an instruction?

Ans. An instruction is a command given to the microcomputer to perform a specific task or function on a given data.

2. What is meant by instruction set?

Ans. An instruction set is a collection of instructions that the microprocessor is designed to perform.

3. In how many categories the instructions of 8085 be classified?

Ans. Functionally, the instructions can be classified into five groups:

- data transfer (copy) group
- arithmetic group
- logical group
- branch group
- stack, I/O and machine control group.

4. What are the different types of data transfer operations possible?

Ans. The different types of data transfer operations possible are cited below:

- Between two registers.
- Between a register and a memory location.
- A data byte can be transferred between a register and a memory location.
- Between an I/O device and the accumulator.
- Between a register pair and the stack.

The term 'data transfer' is a misnomer—actually data is not transferred, but copied from source to destination.

5. Mention the different types of operations possible with arithmetic, logical, branch and machine control operations.

Ans. The arithmetic operations possible are addition, subtraction, increment and decrement.

The logical operations include AND, OR, EXOR, compare, complement, while branch operations are Jump, Call, Return and Restart instructions.

The machine control operations are Halt, Interrupt and NOP (no operation).

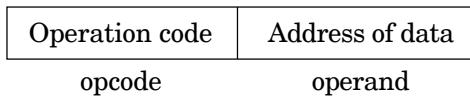
6. What are the different instruction word sizes in 8085?

Ans. The instruction word sizes are of the following types:

- 1-byte instruction
- 2-byte instruction
- 3-byte instruction.

7. What an instruction essentially consists of?

Ans. An instruction comprises of an operation code (called ‘opcode’) and the address of the data (called ‘operand’), on which the opcode operates. This is the structure on which an instruction is based. The opcode specifies the nature of the task to be performed by an instruction. Symbolically, an instruction looks like

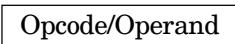


8. Give one example each of 1-byte, 2-byte and 3-byte instructions.

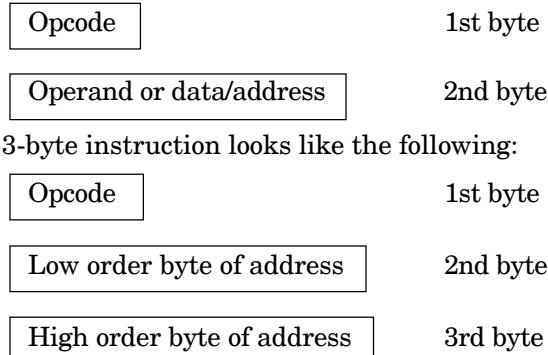
Ans. The examples are given below:

- 1-byte instruction : ADD B
- 2-byte instruction : MVIC, 07
- 3-byte instruction : LDA 4400

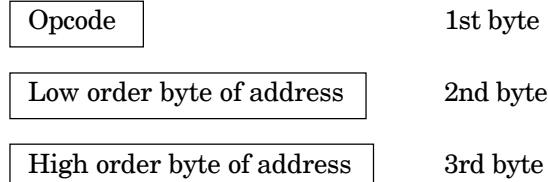
In 1-byte instruction, the opcode and the operand are in the same byte i.e.,



A 2-byte instruction looks like this:



While a 3-byte instruction looks like the following:



9. What is meant by ‘addressing mode’? Mention the different addressing modes.

Ans. Each instruction indicates an operation to be performed on certain data. There are various methods to specify the data for the instructions, known as ‘addressing modes’.

For 8085 microprocessor, there are five addressing modes. These are:

- Direct addressing
- Register addressing
- Register indirect addressing
- Immediate addressing
- Implicit addressing.

10. Give one example each of the five types of addressing modes.

Ans. The examples for each type of addressing mode are given below:

- (a) *Direct Addressing*: In this mode, the operand is specified within the instruction itself.

Examples of this type are:

LDA 4000_H, STA 5513_H, etc.

IN/OUT instructions (like IN PORT C, OUT PORT B, etc.) also falls under this category.

- (b) *Register Addressing*: In this mode of addressing, the operand are in the general purpose registers.

Examples are: MOV A, B ; ADD D, etc.

- (c) *Register Indirect Addressing*: MOV A, M; ADD M are examples of this mode of addressing. These instructions utilise 1-byte. In this mode, instead of specifying a register, a register pair is specified to accommodate the 16-bit address of the operand.

- (d) *Immediate Addressing*: MVI A, 07; ADI 0F are examples of Immediate Addressing mode.

The operand is specified in the instruction in this mode. Here, the operand address is not specified.

- (e) *Implicit Addressing*: In this mode of addressing, the operand is fully absent. Examples are RAR, RAL, CMA, etc.

- 11. Let at the program memory location 4080, the instruction MOV B, A (opcode 47_H) is stored while the accumulator content is FF_H. Illustrate the execution of this instruction by timing diagram.**

Ans. Since the program counter sequences the execution of instructions, it is assumed that the PC holds the address 4080_H. While the system executes the instruction, the following takes place one after another.

1. The CPU places the address 4080_H (residing in PC) on the address bus—40_H on the high order bus A₁₅ – A₈ and 80_H on the low order bus AD₇ – AD₀.

2. The CPU raises the ALE signal to go high—the H to L transition of ALE at the end of the first T state demultiplexes the low order bus.

3. The CPU identifies the nature of the machine cycle by means of the three status signals IO/M, S₀ and S₁.

$$\text{IO/} = 0, \text{S}_1 = 1, \text{S}_0 = 1$$

4. In T₂, memory is enabled by the signal.

The content of PC i.e., 47_H is placed on the data bus. PC is incremented to 4081_H.

5. In T₃, CPU reads 47_H and places it in the instruction register.

6. In T₄, CPU decodes the instruction, places FF_H (accumulator content) in the temporary register and then transfers it to register B. Figure 3.1 shows the execution of the above. It consists of 4T states.

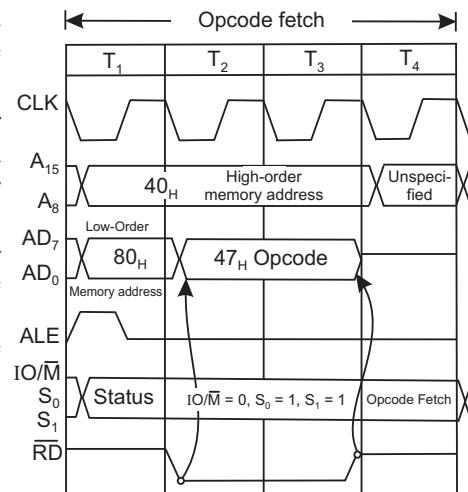


Fig. 3.1: 8085 timing for execution of the instruction (MOV B, A)

4

8085 Interrupts

1. Mention the interrupt pins of 8085.

Ans. There are five (5) interrupt pins of 8085—from pin 6 to pin 10. They represent TRAP, RST 7.5, RST 6.5, RST 5.5 and INTR interrupts respectively. These five interrupts are ‘hardware’ interrupts.

2. Explain maskable and non-maskable interrupts.

Ans. An interrupt which can be disabled by software means, is called a maskable interrupt. Thus an interrupt which cannot be masked is an unmaskable interrupt.

3. Which is the non-maskable interrupt for 8085?

Ans. TRAP interrupt is the non-maskable interrupt for 8085. It means that if an interrupt comes via TRAP, 8085 will have to recognise the interrupt.

4. Do the interrupts of 8085 have priority?

Ans. Yes, the interrupts of 8085 have their priorities fixed—TRAP interrupt has the highest priority, followed by RST 7.5, RST 6.5, RST 5.5 and lastly INTR.

5. What is meant by priority of interrupts?

Ans. It means that if 8085 is interrupted by more than one interrupt at the same time, the one which is having highest priority will be serviced first, followed by the one(s) which is (are) having just next priority and so on.

For example, if 8085 is interrupted by RST 7.5, INTR and RST 5.5 at the same time, then the sequence in which the interrupts are going to be serviced are as follows: RST 7.5, RST 5.5 and INTR respectively.

6. Mention the types of interrupts that 8085 supports.

Ans. 8085 supports two types of interrupts—hardware and software interrupts.

7. What are the software interrupts of 8085? Mention the instructions, their hex codes and the corresponding vector addresses.

Ans. 8085 has eight (8) software interrupts from RST 0 to RST 7. The instructions, hex codes and the vector locations are tabulated in Table 4.1:

Table 4.1: Vector addresses for software interrupts

| Instruction | Corresponding HEX code | Vector addresses |
|-------------|------------------------|------------------|
| RST 0 | C7 | 0000H |
| RST 1 | CF | 0008H |
| RST 2 | D7 | 0010H |
| RST 3 | DF | 0018H |
| RST 4 | E7 | 0020H |
| RST 5 | EF | 0028H |
| RST 6 | F7 | 0030H |
| RST 7 | FF | 0038H |

8. How the vector address for a software interrupt is determined?

Ans. The vector address for a software interrupt is calculated as follows:

$$\text{Vector address} = \text{interrupt number} \times 8$$

For example, the vector address for RST 5 is calculated as

$$5 \times 8 = 40_{10} = 28_H$$

∴ Vector address for RST 5 is 0028_H .

9. In what way INTR is different from the other four hardware interrupts?

Ans. There are two differences, which are discussed below:

1. While INTR is not a vectored interrupt, the other four, viz., TRAP, RST 7.5, RST 6.5 and RST 5.5 are all vectored interrupts. Thus whenever an interrupt comes via any one of these four interrupts, the internal control circuit of 8085 produces a CALL to a predetermined vector location. At these vector locations the subroutines are written.
On the other hand, INTR receives the address of the subroutine from the external device itself.
2. Whenever an interrupt occurs via TRAP, RST 7.5, RST 6.5 or RST 5.5, the corresponding returns address (existing in program counter) is auto-saved in STACK, but this is not so in case of INTR interrupt.

10. Indicate the nature of signals that will trigger TRAP, RST 7.5, RST 6.5, RST 5.5 and INTR.

Ans. TRAP interrupt is both positive edge and level triggered, RST 7.5 is positive edge triggered while RST 6.5, RST 5.5 and INTR are all level triggered.

11. Why the TRAP input is edge and level sensitive?

Ans. TRAP input is edge and level sensitive to avoid false triggering caused by noise and transients.

12. Draw the TRAP interrupt circuit diagram and explain the same.

Ans.

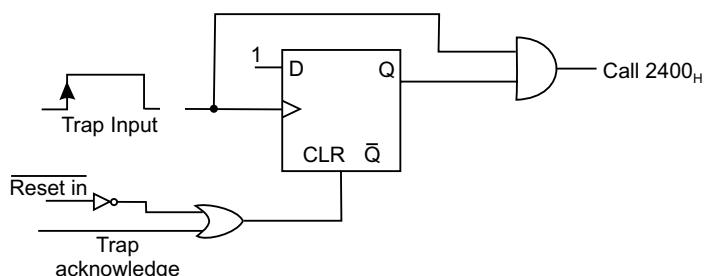


Fig. 4.1: The TRAP interrupt circuit

The positive edge of the TRAP signal sets the D F/F, so that Q becomes 1. It thus enables the AND gate, but the AND gate will output a 1 for a sustained high level at the TRAP input. In that case the subroutine written at vector memory location 2400_H corresponding to TRAP interrupt starts executing. 2400_H is the starting address of an interrupt service routine for TRAP.

There are two ways to clear the TRAP interrupt:

1. When the microprocessor is reset, then via the inverter, a high comes out of the OR gate, thereby clearing the D F/F, making Q = 0.
2. When a TRAP is acknowledged by the system, an internal TRAP ACKNOWLEDGE is generated thereby clearing the D F/F.

13. Discuss the INTR interrupt of 8085.

Ans. The following are the characteristics of INTR interrupt of 8085:

- It is a maskable interrupt
- It has lowest priority
- It is a non-vectorized interrupt.

Sequentially, the following occurs when INTR signal goes high:

1. 8085 checks the status of INTR signal during execution of each instruction.
2. If INTR signal remains high till the completion of an instruction, then 8085 sends out an active low interrupt acknowledge (INTA) signal.
3. When INTA signal goes low, external logic places an instruction OPCODE on the data bus.
4. On receiving the instruction, 8085 saves the address of next instruction (which would have otherwise been executed) in the STACK and starts executing the ISS (interrupt service subroutine).

14. Draw the diagram that outputs RST 3 instruction opcode on acknowledging the interrupt.

Ans. The diagram is shown below:

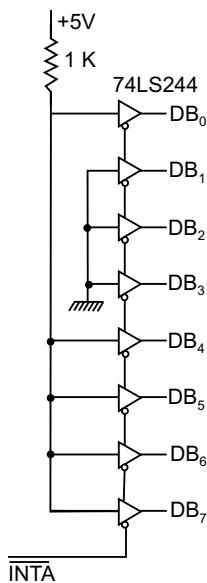


Fig. 4.2: Hardware circuit diagram to implement RST 3 (Opcode DF H)

When an INTR is acknowledged by the microprocessor, it outputs a low INTA. Thus an RST 3 is gated onto the system bus. The processor then first saves the PC in the STACK and branches to the location 0018_H . At this address, the ISS begins and ends with a RET instruction. On encountering RET instruction in the last line of the ISS, the return address saved in the stack is restored in the PC so that normal processing in the main program (at the address which was left off when the program branched to ISS) begins.

15. What is to be done if a particular part of a program is not to be interrupted by RST 7.5, RST 6.5, RST 5.5 and INTR?

Ans. Two software instructions—EI and DI are used at the beginning and end of the particular portion of the program respectively. The scheme is shown schematically as follows:

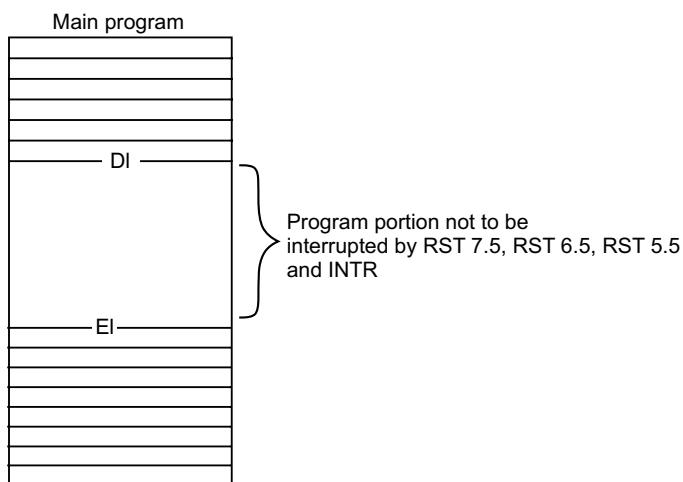


Fig. 4.3: Employing EI and DI in a program

16. Explain the software instructions EI and DI.

Ans. The EI instruction sets the interrupt enable flip-flop, thereby enabling RST 7.5, RST 6.5, RST 5.5 and INTR interrupts.

The DI instruction resets the interrupt enable flip-flop, thereby disabling RST 7.5, RST 6.5, RST 5.5 and INTR interrupts.

17. When returning back to the main program from Interrupt Service Subroutine (ISS), the software instruction EI is inserted at the end of the ISS. Why?

Ans. When an interrupt (either via RST 7.5, RST 6.5, RST 5.5, INTR) is acknowledged by the microprocessor, ‘any interrupt acknowledge’ signal resets the interrupt enable F/F. It thus disables RST 7.5, RST 6.5, RST 5.5 and INTR interrupts. Thus any future interrupt coming via RST 7.5, RST 6.5, RST 5.5 or INTR will not be acknowledged unless the software instruction EI is inserted which thereby sets the interrupt enable F/F.

18. Mention the ways in which the three interrupts RST 7.5, RST 6.5 and RST 5.5 are disabled?

Ans. The three interrupts can be disabled in the following manner:

- Software instruction DI
- RESET IN signal
- Any interrupt acknowledge signal.

19. Draw the SIM instruction format and discuss.

Ans. The SIM instruction format is shown below:

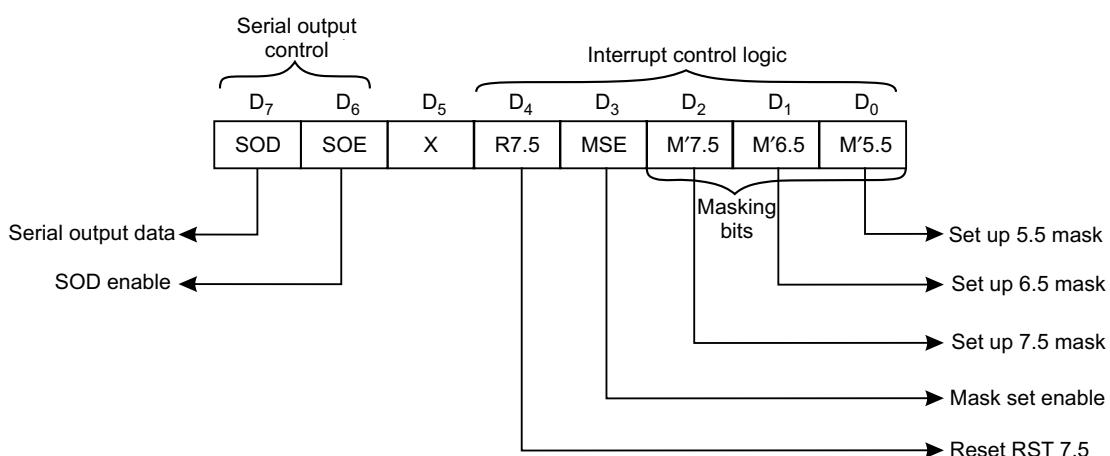


Fig. 4.4: The SIM instruction format

D₇ and D₆ bits are utilised for serial outputting of data from accumulator. D₅ bit is a don't care bit, while bits D₄–D₀ are used for interrupt control.

D₄ bit can clear the D F/F associated with RST 7.5.

D₃ bit is mask set enable (MSE) bit, while bits D₂–D₀ are the masking bits corresponding to RST 7.5, RST 6.5 and RST 5.5 respectively.

None of the flags are affected by SIM instruction.

By employing SIM instruction, the three interrupts RST 7.5, RST 6.5 and RST 5.5 can be masked or unmasked. For masking any one of these three interrupts, MSE (i.e., bit D₃) bit must be 1.

For example let RST 7.5 is to be masked (disabled), while RST 6.5 and RST 5.5 are to be unmasked (enabled), then the content of the bits of the SIM instruction will be like

0000 1100 = 0C_H

For this to be effective the following two instructions are written,

MVI A, 0C_H

SIM

Execution of SIM instruction allows copying of the contents of the accumulator into the interrupt masks.

20. Show the RIM instruction format and discuss the same.

Ans. RIM stands for ‘Read interrupt mask’ and its format is as follows:

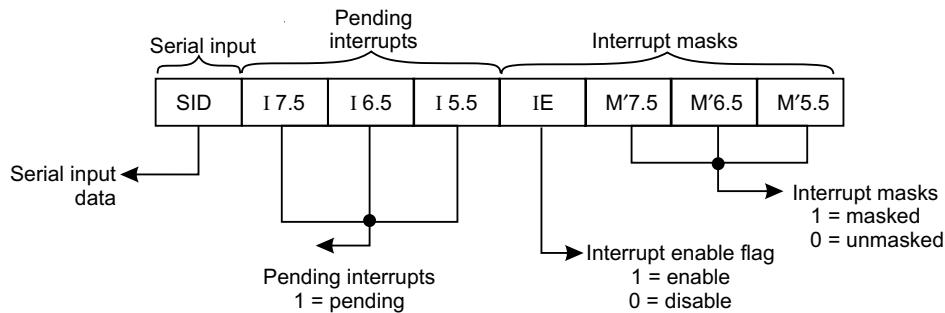


Fig. 4.5: The RIM instruction format

When RIM instruction is executed in software, the status of SID, pending interrupts and interrupt masks are loaded into the accumulator. Thus their status can be monitored. It may so happen that when one interrupt is being serviced, other interrupt(s) may occur. The status of these pending interrupts can be monitored by the RIM instruction. None of the flags are affected by RIM instruction.

21. Write a program which will call the interrupt service subroutine (at 3C00H) corresponding to RST 7.5 if it is pending. Let the ACC content is 20 H on executing the RIM instruction.

Ans. The program for the above will be as hereunder:

| | | SID I7.5 I6.5 I5.5 IE M'7.5 M'6.5 M'5.5 |
|-----------|--|--|
| RIM | \Rightarrow ACC | 0 0 1 0 0 0 0 0 |
| ANI 40 H | \Rightarrow AND immediate with 40 H | 0 1 0 0 0 0 0 0 |
| CNZ 3C00H | \Rightarrow Call interrupt service subroutine corresponding to RST 7.5, if RST 7.5 is pending. | Isolate 17.5 bit |

22. Write a program which will call the subroutine (say named ‘SR’) if RST 6.5 is masked. Let content of ACC is 20 H on executing the RIM instruction.

Ans. RST 6.5 is masked if bit M'6.5 (D₁ bit of RIM) is a 1 and also D₃ bit (i.e., IE) is 1

The program for the above will be as hereunder:

| | | SID I7.5 I6.5 I5.5 IE M'7.5 M'6.5 M'5.5 |
|----------|---|--|
| RIM | \Rightarrow ACC | 0 0 1 0 0 0 0 0 |
| ANI 0A H | \Rightarrow AND immediate with 0A H | 0 0 0 0 1 0 1 0 |
| JNZ SR | \Rightarrow Jump to subroutine “SR” if RST 6.5 is masked. | Isolate M'6.5 bit |

23. For what purpose TRAP interrupt is normally used?

Ans. TRAP interrupt is a non-maskable one i.e., if an interrupt comes via the TRAP input, the system will have to acknowledge that. That is why it is used for vital purposes which require immediate attention like power failure.

If the microprocessor based system loses power, the filter capacitors hold the supply voltage for several mili seconds.

During this time, data in the RAM can be written in a disk or E²PROM for future usage.

24. Draw the interrupt circuit diagram for 8085 and explain.

Ans. Figure 4.6 is the interrupt circuit diagram of 8085. It shows the five hardware interrupts TRAP, RST 7.5, RST 6.5, RST 5.5 and INTR along with the software interrupts RST n: n = 0 to 7.

Trap is both edge and level sensitive interrupt. A short pulse should be applied at the trap input, but the pulse width must be greater than a normal noise pulse width and also long enough for the μP to complete its current instruction. The trap input must come down to low level for it to be recognised for the second time by the system. It is having highest priority.

Next highest priority interrupt is RST 7.5 which responds to the positive edge (low to high transition) of a pulse. Like trap, it also has a D F/F whose output becomes 1 on accepting the RST 7.5 input, but final call to vector location 3C00H is reached only if RST 7.5 remains unmasked and the program has an EI instruction inserted already. These are evident from the circuit. If R 7.5 (bit D₄ of SIM instruction) is 1, then RST 7.5 instruction will be overlooked i.e., it can override any RST 7.5 interrupt.

Like RST 7.5, final call locations 3400_H and 2C00_H corresponding to interrupts at RST 6.5 and RST 5.5 are reached only if the two interrupts remain unmasked and the software instruction EI is inserted.

The three interrupts RST 7.5, RST 6.5 and RST 5.5 are disabled once the system accepts an interrupt input via any one of these pins—this is because of the generation of ‘any interrupt acknowledge’ signal which disables them.

Any of the software RST instructions (RST n : n = 0 to 7) can be utilised by using INTR instruction and hardware logic. RST instructions are utilised in breakpoint service routine to check register(s) or memory contents at any point in the program.

25. The process of interrupt is asynchronous in nature. Why?

Ans. Interrupts may come and be acknowledged (provided masking of any interrupt is not done) by the microprocessor without any reference to the system clock. That is why interrupts are asynchronous in nature.

26. In how many categories can ‘interrupt requests’ be classified?

Ans. The ‘interrupt requests’ can be classified into two categories—maskable interrupt and non-maskable interrupt.

A maskable interrupt can either be ignored or delayed as per the needs of the system while a non-maskable interrupt has to be acknowledged.

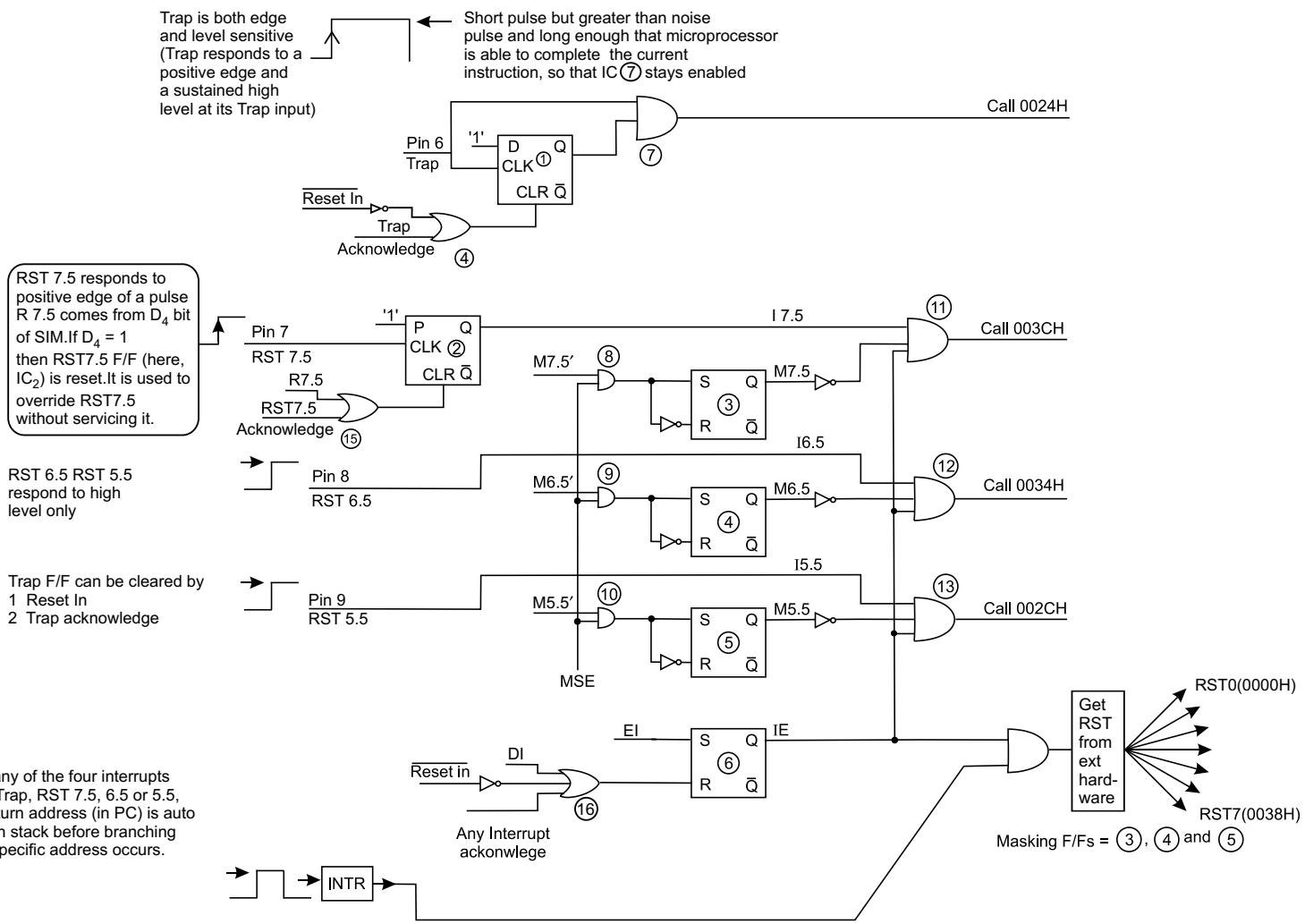


Fig. 4.6: Interrupt circuit description

27. When the interrupt pins of 8085 are checked by the system?

Ans. Microprocessor checks (samples) interrupt pins one cycle before the completion of an instruction cycle, on the falling edge of the clock pulse. An interrupt occurring at least 160 ns (150 ns for 8085A-2, since it is faster in operation) before the sampling time is treated as a valid interrupt.

28. Is there a minimum pulse width required for the INTR signal?

Ans. Microprocessor issues a low $\overline{\text{INTA}}$ signal as an acknowledgement on receiving an INTR interrupt input signal. A CALL instruction is then issued so that the program branches to Interrupt Service Subroutine (ISS). Now the CALL requires 18 T-states to complete. Hence the INTR pulse must remain high for at least 17.5 T-states. If 8085 is operated at 3 MHz clock frequency, then the INTR pulse must remain high for at least 5.8 μs .

29. Can the microprocessor be interrupted before completion of existing Interrupt Service Subroutine (ISS)?

Ans. Yes, the microprocessor can be interrupted before the completion of the existing ISS. Let after acknowledging the INTR, the microprocessor is in the ISS executing instructions one by one. Now, for a given situation, if the interrupt system is enabled (by inserting an EI instruction) just after entering the ISS, the system can be interrupted again while it is in the first ISS.

If an interrupt service subroutine be interrupted again then this is called ‘nested interrupt’.

30. Bring out one basic difference between SIM and DI instructions.

Ans. While by using SIM instruction any combinations or all of RST 7.5, RST 6.5 and RST 5.5 can be disabled, on the other hand DI disables RST 7.5, RST 6.5, RST 5.5 and in addition INTR interrupt also.

31. What is RIM instruction and what does it do?

Ans. The instruction RIM stands for Read Interrupt Mask. By executing this instruction in software, it is possible to know the status of interrupt mask, pending interrupt(s), and serial input.

32. In an interrupt driven system, EI instruction should be incorporated at the beginning of the program. Why?

Ans. A program, written by a programmer in the RAM location, is started first by system reset and loading the PC with the starting address of the program.

Now, with a system reset, all maskable interrupts are disabled. Hence, an EI instruction must be put in at the beginning of the program so that the maskable interrupts, which should remain unmasked in a program, remain so.

33. How the system can handle multiple interrupts?

Ans. Multiple interrupts can be handled if a separate interrupt is allocated to each peripheral. The programmable interrupt controller IC 8259 can also be used to handle multiple interrupts when they are interfaced through INTR.

34. When an interrupt is acknowledged, maskable interrupts are automatically disabled. Why?

Ans. This is done so that the interrupt service subroutine (ISS) to which the program has entered on receiving the interrupt, has a chance to complete its own task.

35. What is meant by ‘nested interrupts’? What care must be taken while handling nested interrupts?

Ans. Interrupts occurring within interrupts are called ‘nested interrupts’.

While handling nested interrupts, care must be taken to see that the stack does not grow to such an extent as to foul the main program—in that case the system program fails.

36. ‘A RIM instruction should be performed immediately after TRAP occurs’—Why?

Ans. This is so as to enable the pre-TRAP interrupt status to be restored with the implementation of a SIM instruction.

37. What does the D₄ bit of SIM do?

Ans. Bit D₄ of SIM is R 7.5 which is connected to RST 7.5 F/F via a OR gate. If D₄ of SIM is made a 1, then it resets RST 7.5 F/F. This thus can be used to override RST 7.5 without servicing it.

38. Comment on the TRAP input of 8085.

Ans. Trap input is both edge and level sensitive. It is a narrow pulse, but the pulse width should be more than normal noise pulse width. This is done so that noise cannot affect the TRAP input with a false triggering. Again the pulse width should be such that the TRAP input which is directly connected to the gate stays high till the completion of current instruction by the μP. In that case, only the program gets diverted to vector call location 2400 H.

TRAP cannot respond for a second time until the first TRAP goes through a high to low transition.

TRAP interrupt, once acknowledged, goes to 2400 H vector location without any external hardware or EI instruction, as is the case for other interrupt signals to be acknowledged.

39. Discuss about the triggering levels of RST 7.5, RST 6.5 and RST 5.5.

Ans. RST 7.5 is positive edge sensitive and responds to a short trigger pulse. The interrupt that comes via RST 7.5 is stored in a D F/F, internal to μP. The final vector call location 3C00 H is invoked only if RST 7.5 remains unmasked via SIM and software instruction EI is inserted in the program.

RST 6.5 and 5.5 respond to high level (i.e., level sensitive) at their input pins—thus these two pins must remain high until microprocessor completes the execution of the current instruction.

40. Discuss the utility of RST software instruction.

Ans. For an RST instruction (RST n, n : 0 to 7) to become effective, external hardware is necessary, along with INTR interrupt instruction.

When debugging is required in a program to know the register(s) or memory contents, breakpoints are inserted via RST instruction.

A breakpoint is an RST instruction inserted in a program. When an RST instruction is recognised, the program control is transferred to the corresponding RST vector location. From this vector location, it is again transferred to the breakpoint service routine so that programmer can check the contents of any register or memory content on pressing specified key(s). After testing, the routine returns to the breakpoint in the main program.

Thus RST instructions can be inserted within a program to examine the register/memory content as per the requirement.

41. Under what condition, an RST instruction is going to be recognised?

Ans. Any RST instruction is recognised only if the EI instruction is incorporated via software.

42. Can the 'TRAP' interrupt be disabled by a combination of hardware and software?

Ans. Yes, it can be disabled by SIM instruction and hardware, as shown in Fig. 4.7.

The following two instructions are executed.

MVIA, 40 H

SIM

It ensures that a '0' logic comes out via SOD pin (pin 4) of 8085. This is then ANDed with TRAP input.

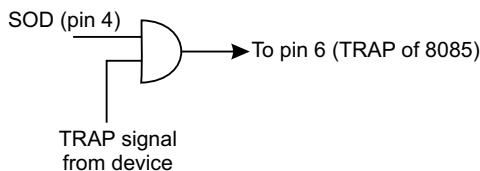


Fig. 4.7

Thus pin 6 (TRAP) always remains at '0' logic and hence TRAP input is disabled or 'MASKED'.

43. Level wise, how the interrupts can be classified? Distinguish them.

Ans. Level wise, interrupts can be classified as

- single level interrupts
- multi level interrupts

Their distinguishing features are shown below:

| Single level interrupt | Multi level interrupt |
|--|--|
| <ol style="list-style-type: none"> 1. Interrupts are fed via a single pin of microprocessor (like INTR of 8085) 2. CPU polls the I/O devices to identify the interrupting device. 3. Slower because of sl. no. 2. | <ol style="list-style-type: none"> 1. Interrupts are fed via different pins of microprocessor (like RST 7.5, RST 6.5 etc), each interrupt requiring a separate pin of microprocessor. 2. Since each interrupt pin corresponds to a single I/O device, polling is not necessary. 3. Faster because of sl. no. 2. |