# 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
9. 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 qoutient |
|  | 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 MOV  INX | H,4200 B,M  H | Set pointer for array Load the Count |
| MOV | A,M | Set 1st 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 MOV  INX | H,4200 B,M  H | Set pointer for array Load the Count |
| MOV | A,M | Set 1st 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:

|  |  |  |
| --- | --- | --- |
| REPEAT: | LXI MOV DCR  MOV | H,4200 C,M  C 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:

|  |  |  |
| --- | --- | --- |
| REPEAT: | LXI MOV DCR  MOV | H,4200 C,M  C 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 RLC RLC RLC RLC  CALL | F0  SUB1 | Mask Lower Nibble  Get ASCII code for lower nibble |
|  | STA HLT | 4202 |  |
| SUB1: | CPI | 0A |  |
|  | JC | SKIP |  |
|  | ADI | 07 |  |
| SKIP: | ADI  RET | 30 |  |

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.

AIM:

## ASCII TO HEX CONVERSION

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 ant 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 |

1= ENABLE 0 = DISABLE

PARITY ENABLE

0 =ODD 1 = EVEN

EVEN PARITY GEN/CHECK

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

1. 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 |

1=Enable 0 = Disable

TRANSMIT ENABLE

HIGH will force DTR

Output to Zero

DATA TERMINAL READY

1=Enable 0 = Disable

RECEIVE ENABLE

1 = Forces TXD LOW

0 = Normal Operation

SEND BREAK CHARACTER

1=Reset Error Flags

PE,OE,FE

ERROR RESET

HIGH will force RTS

Output to Zero

REQUEST TO SEND

HIGH Returns 8251 to

Mode Instruction Format

INTERNAL RESET

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

ENTER HUNT 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 |
| OBSERVATION: | HLT |  |

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:

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

PROGRAM:

1. 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  RET | L1 |

1. Saw tooth Wave Generation

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

1. Triangular Wave Generation

|  |  |  |
| --- | --- | --- |
| START:  L1: | MVI  MOV | L,00  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 (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  RET | LOOP2 |

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:

|  |  |  |
| --- | --- | --- |
| START: | ORG  MOV | 4100  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:

1. Square Wave Generation
2. Move the port address of DAC to DPTR
3. Load the initial value (00) to Accumulator and move it to DAC
4. Call the delay program
5. Load the final value(FF) to accumulator and move it to DAC
6. Call the delay program.
7. Repeat Steps 2 to 5
8. Saw tooth Wave Generation
9. Move the port address of DAC to DPTR
10. Load the initial value (00) to Accumulator
11. Move the accumulator content to DAC
12. Increment the accumulator content by 1.
13. Repeat Steps 3 and 4.
14. Triangular Wave Generation
15. Move the port address of DAC to DPTR
16. Load the initial value (00) to Accumulator
17. Move the accumulator content to DAC
18. Increment the accumulator content by 1.
19. If accumulator content is zero proceed to next step. Else go to step 3.
20. Load value (FF) to Accumulator
21. Move the accumulator content to DAC
22. Decrement the accumulator content by 1.
23. If accumulator content is zero go to step2. Else go to step 7.

PROGRAM:

1. 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  RET SJMP | R1,LOOP  START |

1. Saw tooth Wave Generation

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

1. 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 KeilµVision2 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

1. Read the key from user
2. Compare with the initialized password
3. 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