Saveetha Institute of Medical And Technical Sciences
(Declared as Deemed to be University under Section 3 of UGC Act 1956)

ECA10

MICROPROCESSORS AND MICROCONTROLLERS

MASTER RECORD
WITH
TEST CASES

INDEX

1) Arithmetic Operations in 8085 Microprocessors

a. 8-BIT ADDITION WITH CARRY USING DIRECT ADDRESSING

AIM:

To write an assembly language program to add two numbers of 8-bit data stored in memory locations 4200H and 4201H and store the result in 4202H and 4203H with carry using direct addressing.

APPARATUS REQUIRED:

1. 8085 microprocessor kit 1
2. Power card 1
3. Keyboard 1

ALGORITHM:

1. Load the first data from memory to the accumulator and move it to B register.
2. Load the second data from memory to the accumulator.
3. Add the content of B – register to accumulator
4. If Carry flag = 0 then jump to step 6
5. Increment C register to count the carry
6. Store the sum in memory.
7. Move the carry to accumulator and store in memory.
8. Stop.

PROGRAM TO ADD TWO 8-BIT DATA

Memory address	Label	Instruction	Opcode	Comments
4100		LDA 4200H		Get 1st data in A and save in B.
4103		MOV B, A		
4104		LDA 4201H		Get 2nd data in A-register
4107		ADD B		Get the sum in A register
4108		JNC SKIP		If CY=0 Then skip next step
410B		INR C		Increment C register to count the carry
410C	SKIP	STA 4202H		Store the sum in memory
410F		MOV A,C		Move the carry to accumulator and store in memory
4110		STA 4203H		
4113		HLT		Stop the Execution

Input		Output	
Address	**Data**	**Address**	**Data**
4200	**CF**	**4202**	**6C (Sum)**
4201	**9D**	**4203**	**01 (Carry)**

RESULT:

Thus, an assembly language program for addition of two 8-bit numbers with carry was written, executed and Verified the Result successfully using 8085 kit.

b. 8-BIT ADDITION USING INDIRECT ADDRESSING

AIM:

To write an assembly language program to add two numbers of 8-bit data stored in memory locations 4200H and 4201H and store the result in 4202H and 4203H using indirect addressing modes

APPARATUS REQUIRED:

1. 8085 microprocessor kit 1
2. Power card 1
3. Keyboard 1

ALGORITHM:

1. Load the first data from memory to the accumulator.
2. Add the content of Memory to accumulator
3. If Carry flag = 0 then jump to step 5
4. Increment C register to count the carry
5. Store the sum in memory.
6. Store the carry in memory from C-register.
7. Stop.

PROGRAM TO ADD TWO 8-BIT DATA

Memory address	Label	Instruction	Opcode	Comments
4100		LXI H,4200H		Load the HL with data address
4103		MVI C,00		Clear C-Reg to count carry
4105		MOV A, M		Get 1st data in A
4106		INX H		
4107		ADD M		Add reg-A with Memory and Get the sum in A register
4108		JNC SKIP		If CY=0, Skip next step
410B		INR C		Increment C-Reg
410C	SKIP	INX H		
410D		MOV M, A		Store the sum in memory
410E		INX H		
410F		MOV M, C		Store the Carry in memory
4110		HLT		Stop the Execution

SAMPLE DATA:

Input		Output	
Address	**Data**	**Address**	**Data**
4200	**2F**	**4202**	**18 (Sum)**
4201	**E9**	**4203**	**01 (Carry)**

RESULT:

Thus, an assembly language program for addition of two 8-bit numbers with carry was written, executed and Verified the Result successfully using 8085 kit.

c. 8-BIT SUBTRACTION WITHOUT BORROW

AIM:

To write an assembly language program to subtract two numbers of 8-bit data stored in memory 4200H and 4201H. Store the magnitude of the result in 4202H using 8085 Microprocessor.

APPARATUS REQUIRED:

1. 8085 microprocessor kit 1
2. Power card 1
3. Keyboard 1

ALGORITHM:

1. Load the subtrahend (the data to be subtracted) from memory to accumulator and move if to B-register.
2. Load the minuend from memory to accumulator.
3. Subtract the content of B-register (subtrahend) from the content of accumulator (minuend).
4. Store the difference in memory.
5. Stop.

PROGRAM TO SUBTRACT TWO 8-BIT DATA

Memory address	Label	Instruction	Opcode	Comments
4100		LDA 4201H		; Get the subtrahend in B register.
4103		MOV B,A		
4104		LDA 4200H		;Get the minuend in A register
4107		SUB B		; Get the difference in A register.
4108		STA 4202H		Store the result in memory
410B		HLT		Stop the Execution

Sample data

Address	Input Data	Address	Output Data
4200	**D5**	**4202**	**8B (Difference)**
4201	**4A**		

RESULT:

Thus, an assembly language program for subtraction of two 8-bit numbers without borrow was written, executed and Verified the Result successfully using 8085 kit.

d. 8-BIT SUBTRACTION WITH BORROW USING DIRECT ADDRESSING

AIM:
To write an assembly language program to subtract tow numbers of 8-bit data stored in memory locations 4200H and 4201H and store the result in 4202H and 4203H with borrow using direct addressing.

APPARATUS REQUIRED:
1. 8085 microprocessor kit 1
2. Power card 1
3. Keyboard 1

ALGORITHM:
1. Load the second data from memory to the accumulator and move it to the B register.
2. Load the first data from memory to the accumulator.
3. Subtract the content of B – register from accumulator
4. If Carry flag = 0 then jump to step 5 & 6
5. Increment C register to count the borrow
6. Take two's complement of the difference
7. Store the Difference in memory.
8. Move the borrow to accumulator and store in memory.
9. Stop.

PROGRAM TO SUBTRACT TWO 8-BIT DATA

Memory address	Label	Instruction	Opcode	Comments
4100		LDA 4201H		Get 2nd data in A and save in B.
4103		MOV B, A		
4104		LDA 4200H		Get 1st data in A-register
4107		SUB B		Subtract B-Reg from A register
4108		JNC SKIP		If CY=0 Then skip next two steps
410B		INR C		Increment C register to count the carry
410C		CMA		Take two's complement of difference
410D		INR A		
410E	SKIP	STA 4202H		Store the Difference in memory
4111		MOV A,C		Move the Borrow to accumulator and store in memory
4112		STA 4203H		
4115		HLT		Stop the Execution

SAMPLE DATA:

Input		Output	
Address	**Data**	**Address**	**Data**
4200	**CF**	**4202**	**0E (Sum)**
4201	**DD**	**4203**	**01 (Borrow)**

RESULT:

Thus, an assembly language program for subtraction of two 8-bit numbers with borrow was written, executed and Verified the Result successfully using 8085 kit.

e. 8-BIT MULTIPLICATION OPERATIONS USING 8085 MICROPROCESSOR

AIM:

To write an assembly language program to multiply two numbers of 8-bit data stored in memory 4200H and 4201H and store the product in 4202H and 4203H.

APPARATUS REQUIRED:

1.	8085 microprocessor kit	1
2.	Power card	1
3.	Keyboard	1

ALGORITHM:

- Load the first data in ACC and move to E.
- Load the second data ACC and move to B (count)
- Clear HL pair (Initial sum)
- Clear D for overflow (carry)
- Add the content of DE to HL
- Decrement the count.
- Check whether the count has reached zero.
- Check the zero flag. If ZF = 0, repeat addition or If ZF = 1, go to next step
- Store the content of HL in memory. (Least significant 16 bits of the product)
- Stop.

PROGRAM TO MULTIPLY TWO NUMBERS OF 8-BIT DATA

Memory address	Label	Instruction		Opcode	Comments
4100		LDA	4200H		;Get 1 st data in A
4103		MOV	E, A		;Save 1st data in E
4104		LDA	4201H		;Get 2nd data in A
4107		MOV	B, A		;save 2nd data in B
4108		LXI	H,0000H		;Clear HL pair(initial sum=0)
410B		MVI	D,00H		;Clear E for accounting overflow.
410D	NEXT:	DAD	D		;Add the content of DE to sum(HL)
410E		DCR	B		Decrement data 2 for every addition
410F		JNZ	NEXT		;Repeat Addition until count is zero.
4112		SHLD	4202H		;Store the product in memory
4115		HLT			Stop the Execution

Sample data

Address	Input Data	Address	Output Data
4200	6D (Data-1)	4202	26 (Lower byte of product)
4201	FE (Data-2)	4203	6C (Higher byte of product)

RESULT:

Thus, an assembly language program to multiply two numbers of 8-bit data was written, executed and Verified the Result successfully using 8085 kit.

f. 8-BIT DIVISION OPERATIONS USING 8085 MICROPROCESSOR

AIM:

To write an ALP to perform division of two 8 bit numbers Stored in memory location 4200H, 4201 H and Store the remainder in 4202H and the quotient in 4203H.

APPARATUS REQUIRED:

1. 8085 microprocessor kit 1
2. Power card 1
3. Keyboard 1

ALGORITHM:

- Load the divisor in accumulator and move if to B-register
- Load the dividend in the accumulator.
- Clear C-register to account for quotient
- Check whether divisor is less than dividend
- If divisor is less than dividend, go to step 8, otherwise go to next step
- Subtract the content of B-register (quotient)
- Increment the content of C-register (quotient)
- Go to step 4
- Store the content of the accumulator (remainder) in memory.
- Move the content of C-register (quotient) to accumulator and store in memory
- Stop.

PROGRAM TO DIVIDE TWO NUMBERS OF 8-BIT DATA

Memory address	Label	Instruction	Comments
4100		LDA 4201H	
4103		MOV B,A	;Get the divisor in B register
4104		LDA 4200H	;Get the dividend in A register
4107		MVI C,00H	;Clear C register for quotient
4109	AGAIN:	CMP B	
410A		JC STORE	;If divisor is less than dividend go to store
410D		SUB B	;Subtract divisor from dividend. Increment
410E		INR C	;quotient by one for each subtraction.
410F		JMP AGAIN	
4112	STORE:	STA 4203H	;Store the remainder in memory
4115		MOV A,C	
4116		STA 4202H	;Stare the quotient in memory
4119		HLT	Stop the Execution

Sample data

Address	Input Data	Address	Output Data
4200	9F (Dividend)	4202	0F (Quotient)
4201	0A (Divisor)	4203	09 (Remainder)

RESULT:

Thus, an assembly language program to Divide two numbers of 8-bit data was written, executed and Verified the Result successfully using 8085 kit.

Test Cases: Arithmetic Operations of 8085 Microprocessor

Test Case Question 1: These test cases are for the arithmetic operations such as addition and subtraction. Each test case includes the initial values of registers and memory locations, the operation to be performed, and the expected results.

4. Initial Register Values:

Accumulator (A) = 35H
B Register (B) = 25H
Operation: What will be the value in the accumulator (A) and the carry flag (CY) after executing the instruction ADD B?

Output:

- The value in the accumulator (A) after the operation will be **5AH**.
- The carry flag (CY) will be **0**.

5. Initial Register Values:

Accumulator (A) = 6EH
B Register (B) = 3CH
Operation: What will be the value in the accumulator (A) and the carry flag (CY) after executing the instruction SUB B?

Output:

- The value in the accumulator (A) after the operation will be **32H**.
- The carry flag (CY) will be **0** (no borrow).

Test Case Question 2: These test cases are for the arithmetic operations such as multiplication, and division. Each test case includes the initial values of registers and memory locations, the operation to be performed, and the expected results.

3) Initial Register Values:
Accumulator (A) = 14H
B Register (B) = 0AH
Operation: What will be the values in the accumulator (A) and the B Register (B) after executing the instruction MUL B?

Output:

1. The value in the accumulator (A) after the operation will be **C8H**.
2. The value in the B register (B) will remain **0AH**, as typically multiplication affects only the accumulator.

4) Initial Register Values:
Accumulator (A) = 63H (Dividend)
B Register (B) = 09H (Divisor)
Operation: What will be the values in the accumulator (A) and the B Register (B) after executing the instruction DIV B?

Output:

4. The value in the accumulator (A) after the operation will be **0BH** (quotient).
5. The value in the B register (B) will be **00H** (remainder).

2a. MASKING AND SETTING OF LOWER NIBBLES ON GIVEN DATA

AIM:

To write and execute an assembly language program for performing Masking, Setting, One's and Two's Complement of given data of 8-bit numbers using 8085 Microprocessor.

APPARATUS REQUIRED:

8085 microprocessor kit 1
Power card 1
Keyboard 1

MASKING OF BITS ALGORITHM:

1. Load the Data in A-register.
2. Logically AND the content of A with 0FH.
3. Store the result in memory location.
4. Stop the program

PROGRAM:

i) By using 8086 kit:

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
4100		LDA 4200		Load A-register with Data
4103		ANI, 0FH		AND the content of A with 0FH
4105		STA 4201		Store the Result
4108		HLT		Stop the program

OUTPUT:

INPUT		OUTPUT	
Address	**Data**	**Address**	**Data**
4200H	**4A**	**4201H**	**0A**

2b) SETTING OF BITS ALGORITHM:

1. Load the Data in A-register.
2. Logically ORI the content of A with 0FH..
3. Store the result in memory location.
4. Stop the program

PROGRAM:

ii) By using 8086 kit:

ADDRESS	OPCODE	LABEL	MNEMONICS	COMMENTS
4100			LDA 4200	Load A-register with Data
4103			ORI, 0FH	OR the content of A with 0FH
4105			STA 4201	Store the Result
4108			HLT	Stop the program

OUTPUT:

INPUT		OUTPUT	
Address	**Data**	**Address**	**Data**
4200H	**C5**	**4201H**	**CF**

RESULT:

Thus, an assembly language program for performing logical Masking and Setting of bits were executed using 8085 kit.

2c.ONE’S AND TWO’S COMPLEMENT

AIM:

To write and execute an assembly language program for performing One’s and Two’s Complement of given 8-bit numbers using 8085 Microprocessor.

APPARATUS:

8085 microprocessor kit 1
Power card 1
Keyboard 1

MASKING OF BITS

ALGORITHM:

Load the Data in A-register.
Logically NOT the content of A.
Store the One’s complement in memory location.
Increment the content of A.
Store the Two’s complement in memory location.
Stop the program

PROGRAM:

iii) By using 8086 kit:

ADDRESS	OPCODE	LABEL	MNEMONICS	COMMENTS
4100			LDA 4200	Load AL-register with 1st Data
4103			CMA	NOT the content of AX
4104			STA 4201	Store the One’s complement in memory location.
4107			INR A	Increment the content of AX.
4108			STA 4202	Store the Two’s complement in memory location
410B			HLT	Stop the program

OUTPUT:

INPUT		OUTPUT	
Address	**Data**	**Address**	**Data**
4200H	**AB**	**4201H**	**54**
		4202H	**55**

RESULT:

Thus, an assembly language program for performing One’s and Two’s Complement of bits were executed using 8085 kit.

Test Cases: Logical operation using 8085 Microprocessor

Test case question 1: Load the accumulator with a random value like 0xB5 and apply ANI 0F.What is the accumulator value when a random bit pattern like 0xB5 is masked with ANI 0F?

10110101
AND 00001111

00000101

OUTPUT:

After executing **ANI 0x0F**, the accumulator will contain **0x05**.

Test case question 2: Load the accumulator with 0x1F and apply the ANI instruction with the immediate data 0x0F.Does the ANI instruction correctly preserve the lower nibble when applied to 0x1F with ANI 0F?

00011111
AND 00001111

00001111

OUTPUT:

After executing ANI 0x0F, the accumulator will contain 0x0F. The lower nibble is correctly preserved.

Test case question 3: Load the accumulator with 0x00 and apply the ORI instruction with the immediate data 0x0F.What is the result in the accumulator when ORI 0F is executed on 0x00?

00000000
OR 00001111

00001111

OUTPUT: After executing ORI 0x0F, the accumulator will contain 0x0F.

Test case question 4: Load the accumulator with 0xFF and perform the one's complement.What happens to 0xFF after performing the one's complement?

~11111111

00000000

OUTPUT: After performing the one's complement on 0xFF, the accumulator will contain 0x00.

Test case question 5: Load the accumulator with 0x80 (representing -128 in signed 8-bit) and perform the one's complement.How does the one's complement work when the accumulator contains a signed negative number like 0x80?

~10000000

01111111

OUTPUT:

After performing the one's complement on **0x80**, the accumulator will contain **0x7F** (which is **127** in signed 8-bit).

5) ADDITION OF 16 BIT NUMBERS WITH CARRY

AIM:
To write and execute an assembly language program to add two 16-bit unsigned numbers with carry in 8086 kit.

APPARATUS:

1. 8086 microprocessor kit 1
2. Power card 1
3. Keyboard 1

ALGORITHM:
1. Load the First Data in AX-register.
2. Load the First Data in BX-register.
2. Add the two data and get the sum in AX-register.
3. If C=0 then skip next step.
4. Increment CX Reg for carry
5. Store the sum in memory locations.
6. Store the Carry in memory location.
7. Stop the program.

PROGRAMM

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENT
1100		**MOV CX,0000H**		Initialize counter CX
1104		**MOV AX,[1200]**		Get the first data in AX register.
1108		**MOV BX,[1202]**		Get the second data in BX register.
110C		**ADD AX,BX**		Add the contents of both the register AX & BX
110E		**JNC L1**		Check for carry
1110		**INC CX**		If carry exists, increment the CX
1111	**LI**	**MOV [1206],CX**		Store the carry
1113		**MOV [1204],AX**		Store the sum
1117		**HLT**		Stop the program

OUTPUT FOR ADDITION:

INPUT		OUTPUT	
Address	**Data**	**Address**	**Data**
1200	**ABCD**	**1204**	**9ADF**
1202	**EF12**	**1206**	**0001**

RESULT

Thus, an assembly language program for addition with carry of given 16-bit numbers was written, executed and Verified the Result successfully using 8086 kit

.

4.SUBTRACTION OF 16 BIT NUMBERS WITH BORROW

AIM

To write and execute an assembly language program to subtract two 16-bit unsigned numbers with borrow in 8086 kit.

APPARATUS:

1. 8086 microprocessor kit 1
2. Power card 1
3. Keyboard 1

ALGORITHM:

1. Load the second data from memory to the accumulator and move it to B register.
2. Load the first data from memory to the accumulator.
3. Subtract the content of B – register from accumulator
4. If Carry flag = 0 then jump to step 5 & 6
5. Increment C register to count the borrow
6. Take two's complement of the difference
7. Store the Difference in memory.
8. Move the borrow to accumulator and store in memory.
9. Stop.

PROGRAMM

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENT
1100		MOV CX,0000H		Initialize counter CX
1104		MOV AX,[1300]		Get the first data in AX register
1108		MOV BX,[1302]		Get the second data in BX register.
110C		SUB AX,BX		Subtract the contents of both the register AX & BX
110E		JNC SKIP		Check the Borrow.
1110		INC CX		If carry exists, increment the CX
1111		NEG AX		Take two's complement of the difference
1113	SKIP	MOV [1306],CX		Store the Borrow.
1117		MOV [1304],AX		Store the difference.
111A		HLT		Stop the program

OUTPUT FOR SUBTRACTION:

INPUT		OUTPUT	
Address	**Data**	**Address**	**Data**
1200	**ABCD**	**1204**	**4345**
1202	**EF12**	**1206**	**0001**

RESULT

Thus, an assembly language program for subtraction with borrow of given 16-bit numbers was written, executed and Verified the Result successfully using 8086 kit.

5. MULTIPLICATION OF 16 BIT NUMBERS

AIM

To write and execute an assembly language program to Multiply two 16-bit unsigned numbers in 8086 kit.

APPARATUS:

8086 microprocessor kit 1
Power card 1
Keyboard 1

ALGORITHM:

1. Load the multiplier from memory to accumulator.
2. Load the Multiplicand from memory to BX Reg .
3. Multiply AX with BX.
4. Store the Lower word in memory from AX.
5. Store the Higher word in memory from DX.
6. Stop.

PROGRAMM

ADDRESS	LABEL	MNEMONIC	OPCODE	COMMENTS
1100		MOV AX, [1200]		Load AX-register with 1st data
1104		MOV BX,[1202]		Load BX-register with 2nd data
1105		MUL BX		Multiply the contents of AX with BX-register
1106		MOV [1204],AX		Store the Lower word
1109		MOV [1206],DX		Store the Higher word
110D		HLT		Stop the program

OUTPUT:

INPUT		OUTPUT	
Address	Data	Address	Data
1200	**ABCD**	**1204**	**776A**
1202	**EF12**	**1206**	**A070**

RESULT

Thus, an assembly language program for multiplication of given 16-bit numbers was written, executed and Verified the Result successfully using 8086 kit.

6. DIVISION OF 16 BIT NUMBERS

AIM

To write and execute an assembly language program to Divide two 16-bit unsigned numbers in an 8086 kit.

APPARATUS:

8086 microprocessor kit 1
Power card 1
Keyboard 1

ALGORITHM:

1. Load the Divisor from memory to accumulator.
2. Load the Divisor from memory to BX Reg .
3. Divide DXAX by BX.
4. Store the Quotient in memory from AX.
5. Store the Reminder in memory from DX.
6. Stop.

PROGRAMM

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
1100		MOV DX,0000		Initialize DX-register with Lsb of Dividend
1104		MOV AX, [1200]		Load AX-register with Dividend
1108		MOV BX, [1202]		Load BX-register with Divisor
1109		DIV CX		Divide AX by BX-register
110A		MOV [1204], AX		Store the Quotient
110D		MOV [1206], DX		Store the Remainder
1112		HLT		Stop the program

OUTPUT:

INPUT		OUTPUT	
Address	Data	Address	Data
1200	**EF12**	**1204**	**0001**
1202	**ABCD**	**1206**	**4345**

RESULT

Thus, an assembly language program for Division of given 16-bit numbers was written, executed and Verified the Result successfully using 8086 kit.

Test Cases: Arithmetic Operations of 8086 Microprocessor

Test Case Question 1: These test cases are for the arithmetic operations such as addition and subtraction. Each test case includes the initial values of registers and memory locations, the operation to be performed, and the expected results.

(i) Initial Register Values:
AX Register = 1234H
BX Register = 5678H
Operation: What will be the value in the AX and the carry flag (CY) after executing the instruction ADD BX?

(ii) Initial Register Values:
AX Register = 6A19H
BX Register = 5C15H
Operation: What will be the value in the AX and the carry flag (CY) after executing the instruction SUB BX?

Test Case Question 2: These test cases are for the arithmetic operations such as multiplication, and division. Each test case includes the initial values of registers and memory locations, the operation to be performed, and the expected results.

(i) Initial Register Values:
AL Register = CDH
BL Register = A2H
Operation: What will be the values in the accumulator (A) and the B Register (B) after executing the instruction MUL B?

(ii) Initial Register Values:
AX = 2C5B (Dividend)
BL = 56H (Divisor)
Operation: What will be the values in the AX and the BL Register after executing the instruction DIV B?

EXPERIMENT NO 7: **Double Precision (32-bit) Addition Using 8086 Microprocessor**

Aim:

To add two 32-bit numbers stored as double words in memory and store the 32-bit result in memory.

Algorithm:

3. Load the lower word of the first number into AX.
4. Add the lower word of the second number to AX.
5. Store the result (lower word) in memory.
6. Load the higher word of the first number into AX.
7. Add the higher word of the second number to AX, using the ADC instruction.
8. Store the result (higher word) in memory.
9. Transfer the flag register contents into AH using LAHF to check carry.
10. Store AH in memory to record the carry flag status.
11. Halt the program.

Program:

```
MOV AX, [1100]      ; Load low word of first number
ADD AX, [1104]      ; Add low word of second number
MOV [1200], AX      ; Store low word result

MOV AX, [1102]      ; Load high word of first number
ADC AX, [1106]      ; Add high word + carry
MOV [1202], AX      ; Store high word result

LAHF                ; Load flag register into AH
MOV [1204], AH      ; Store flag status (to check final carry)
HLT                 ; Stop execution
```

Sample Data:

Address	Data (Input/Output)	Description
1100	5678h	Lower word of 1st number
1102	1234h	Higher word of 1st number
1104	5678h	Lower word of 2nd number
1106	1234h	Higher word of 2nd number
1200	**ACF0h**	Result (Low word)
1202	**2468h**	Result (High word)
1204	**00h**	Flag (Carry not set)

Output = 2468ACF0h

Result:

The program successfully performs 32-bit addition using the `ADD` and `ADC` instructions.
For the given inputs, the result stored in memory is:

8a. Logical Operation – Masking of bits

AIM:
To write and execute an assembly language program for performing Masking, Setting, One's and Two's Complement of given 16-bit numbers using 8086 Microprocessor.

APPARATUS:

8086 microprocessor kit 1
Power card 1
Keyboard 1

MASKING OF BITS

ALGORITHM:
Load the Data in AX-register.
Logically AND the content of AX with 0F0FH.
Store the result in memory location.
Stop the program

PROGRAM:

By using 8086 kit:

ADDRESS	OPCODE	LABEL	MNEMONICS	COMMENTS
1100			MOV AX,[1200]	Load AL-register with 1st Data
1104			AND AX, 0F0FH	AND the content of AX with 0F0FH
1108			MOV [1202],AX	Store the Result
110C			HLT	Stop the program

OUTPUT:

INPUT		OUTPUT	
Address	**Data**	**Address**	**Data**
1200H	**ABCD**	**1202H**	**0B0D**

8b. Logical Operation -Setting of Bits

ALGORITHM:

1. Load the Data in AX-register.
2. Logically OR the content of AX with 0F0FH.
3. Store the result in memory location.
4. Stop the program

PROGRAM:

iv) By using 8086 kit:

ADDRESS	OPCODE	LABEL	MNEMONICS	COMMENTS
1100			MOV AX,[1200]	Load AL-register with 1st Data
1104			OR AX, 0F0FH	AND the content of AX with 0F0FH
1108			MOV [1202],AX	Store the Result
110C	F4		HLT	Stop the program

OUTPUT:

INPUT		OUTPUT	
Address	**Data**	**Address**	**Data**
1200H	**ABCD**	**1202H**	**FBFD**

RESULT:

Thus, an assembly language program for performing logical Masking and Setting of bits were executed using 8086 kit.

9. Logical Operation - One's and Two's Complement

AIM:

To write and execute an assembly language program for performing One's and Two's Complement of given 16-bit numbers using 8086 Microprocessor.

APPARATUS:

1. 8086 microprocessor kit 1
2. Power card 1
3. Keyboard 1

MASKING OF BITS

ALGORITHM:

1. Load the Data in AX-register.
2. Logically NOT the content of AX.
3. Store the One's complement in memory location.
4. Increment the content of AX.
5. Store the Two's complement in memory location.
6. Stop the program

PROGRAM:

v) By using 8086 kit:

ADDRESS	OPCODE	LABEL	MNEMONICS	COMMENTS
1100			MOV AX,[1200]	Load AL-register with 1st Data
1104			NOT AX	NOT the content of AX
1108			MOV [1202],AX	Store the One's complement in memory location.
110C			INC AX	Increment the content of AX.
110D			MOV [1204],AX	Store the Two's complement in memory location
1110	F4		HLT	Stop the program

OUTPUT:

INPUT		OUTPUT	
Address	**Data**	**Address**	**Data**
1200H	**12AB**	**1202H**	**ED54**
		1204H	**ED55**

RESULT:

Thus, an assembly language program for performing One's and Two's Complement of bits were executed using 8086 kit.

Test Cases: Logical Operations of 8086 Microprocessor

Test case question 1: Load AX with 0x3A5B and apply AND with 0x0FFF.What happens when the upper nibble is masked out using AND 0x0FFF on 0x3A5B?
Test case question 2: Load AX with 3F0Fh and apply AND with 0008h.What is the result after masking a random value like 0x78AB with AND 0xF0F0?
Test case question 3: Load AX with a value like 0x1234, perform a series of arithmetic operations that affect flags, then apply AND to isolate the carry or zero flag.How does the AND operation interact with flag settings after arithmetic operations?
Test case question 4:Load a register (e.g., AX) with 0x1234 and apply the OR operation with 0x000F.What is the result in AX after setting the lower nibble using OR 0x000F on 0x1234?
Test case question 5: Load AX with 3F0Fh and apply XOR with 0008h.How does XORing with 0008h modify the value 3F0Fh?

```
1.AND AX, 0008H
2.AND AX, BX
3.AND AX, [5000H]
4.AND [5000H], DX
```

If the content of AX is 3F0FH, the first example instruction will carry out the operation as given below. The result 3F9FH will be stored in the AX register.

0 0 1 1	1 1 1 1	0 0 0 0	1 1 1 1	= 3F0F H [AX]
↓ ↓ ↓ ↓	↓ ↓ ↓ ↓	↓ ↓ ↓ ↓	↓ ↓ ↓ ↓	AND
0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 0	= 0008 H
0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 0	= 0008 H [AX]

The result 0008H will be in AX.

```
1.XOR   AX, 0098H
2.XOR   AX, BX
3.XOR   AX, [5000H]
```

If the content of AX is 3F0FH, then the first example instruction will be executed as explained. The result 3F97H will be stored in AX.

AX = 3F0FH =	0 0 1 1	1 1 1 1	0 0 0 0	1 1 1 1
XOR	↓ ↓ ↓ ↓	↓ ↓ ↓ ↓	↓ ↓ ↓ ↓	↓ ↓ ↓ ↓
0098H =	0 0 0 0	0 0 0 0	1 0 0 1	1 0 0 0
AX = Result =	0 0 1 1	1 1 1 1	1 0 0 1	0 1 1 1
= 3F97H				

10) String Operation using 8086 Microprocessors: Move a block of data from source to destination

AIM:

To write and execute an assembly language program for transferring data from one block to another block without overlapping using 8086 kit and MASM.

APPARATUS:

1. 8086 microprocessor kit 1
2. Power card 1
3. Keyboard 1

ALGORITHM:

1. Initialize counter.
2. Initialize source block pointer.
3. Initialize destination block pointer.
4. Get the byte from the source block.
5. Store the byte in the destination block.
6. Increment source, destination pointers and decrement counter.
7. Repeat steps 4, 5 and 6 until the counter equal to zero.
8. Stop.

PROGRAM:

i) By using 8086 kit:

ADDRESS	OPCODE	LABEL	MNEMONICS	COMMENTS
1100	C7 C6 0012		MOV SI, 1150H	Initialize the source address.
1104	C7 C7 0013		MOV DI,1250H	Initialize the destination address.
1108	C7 C1 0600		MOV CX,0006 H	Initialize count value to the count register.
110C	FC	REPEAT:	CLD	Clear the direction flag.
110D	A4		MOVSB	Move the string byte.
110E	E2,F3		LOOP REPEAT	Unconditional loop to address specified by the label REPEAT.
1111	F4		HLT	Stop the program

OUTPUT:

INPUT		OUTPUT	
Address	**Data**	**Address**	**Data**
1150.	52.	1250.	52.
1151.	53.	1251.	53.
1152.	54.	1252.	54.
1153.	55.	1253.	55.
1154.	56.	1254.	56.

RESULT:

Thus, an assembly language program for transferring data from one block to another block without overlapping was executed using 8086 kit.

Test Cases : String Operations of 8086 Microprocessor

Test Case Question 1:

Test Case Scenario:

Moving a block of data from a source memory location to a destination memory location using the 8086 microprocessors. The source and destination addresses are provided. You need to ensure that the data is correctly moved from the source to the destination, and the original data in the source location is preserved.

Test Case Steps:

Load the source memory address (e.g., DS:SI) with a known source address.

Load the destination memory address (e.g., ES:DI) with a known destination address.

Load the count of bytes to move (e.g., CX) with the number of bytes in the block.

Execute a string move instruction (e.g., MOVSB) to move a single byte from the source to the destination.

Repeat the move instruction for the desired number of bytes (specified by CX) until the entire block of data is moved.

Output:

Set the OFFSET S_ARRAY to 200C and OFFSET D_ARRAY to 300C.

200C: 20h

300C :20h

11. SUM OF N NUMBERS IN A WORD ARRAY

AIM:

To write and execute an assembly language program for adding N Numbers in a word array using 8086 kit.

APPARATUS:

1. 8086 microprocessor kit 1
2. Power card 1
3. Keyboard 1

ALGORITHM:

1. Initialize counter.
2. Initialize source block pointer.
3. Initialize destination block pointer.
4. Get the byte from the source block.
5. Store the byte in the destination block.
6. Increment source, destination pointers and decrement counter.
7. Repeat steps 4, 5 and 6 until the counter is equal to zero.
8. Stop.

PROGRAM:

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
1100		MOV DX,00H		Move 00H to DX register
1102		MOV SI,1250H		Move Source Index to 1250H
1105		MOV CX,03H		Move 03H to CX register
1107		MOV AX, [SI]		Move Source Index value to Ax register
110A	A1:	INC SI		Increment The Source Index by one
110B		INC SI		Increment The Source Index by one
110C		ADD AX, [SI]		Add Data in SI with Data in AX register
110D		JNC NEXT		Jump No carry to Label NEXT
110E		INC DX		Increment DX register
110F	NEXT:	LOOP A1		
1110		INC SI		Increment SI
1111		INC SI		Increment SI
1112		MOV [1300H], AX		Move AX register data to 1300H
1116		MOV [1302H], DX		Move DX register data to 1302
1119		HLT		

OUTPUT:

INPUT		OUTPUT	
Address	**Data**	**Address**	**Data**
1250	**ABCD**	**1300**	**0B64**
1252	**EF98**	**1302**	**0003**
1254	**DCBA**		
1256	**9345**		

RESULT:

Thus, an assembly language program for transferring data from one block to another block without overlapping was executed using 8086 kit.

Test Cases: Array Operations of 8086 Microprocessor

Test Case 1: Basic Functionality

- **Input:**
  - Array: [1000h, 2000h, 3000h]
  - N = 3 (length of the array)
- **Expected Output:**
  - Sum = 6000h (stored in DX)

Test Case 2: Mixed Values

- **Input:**
  - Array: [FFFFh, 0001h, 0001h]
  - N = 3
- **Expected Output:**
  - Sum = 0001h (since FFFFh + 1h = 0000h, and adding 1h gives 0001h)

summing an array of N numbers in a word array using the 8086 microprocessor, you can assess various scenarios for understanding assembly language concepts.

12.FACTORIAL OF NUMBER USING 8086 MICROPROCESSOR

AIM:

To write and execute an assembly language program to find the factorial of a number using 8086.

APPARATUS:

- 8086 Microprocessor kit 1
- Power card 1
- Keyboard 1

ALGORITHM:

1. Input the Number whose factorial is to be find and Store that Number in CX Register (Condition for LOOP Instruction)
2. Insert 0001 in AX(Condition for MUL Instruction) and 0000 in DX
3. Multiply CX with AX until CX become Zero(0) using LOOP Instruction
4. Copy the content of AX to memory location 0600
5. Copy the content of DX to memory location 0601
6. Stop Execution

PROGRAM:

ADDRESS	OPCODE	LABEL	PROGRAM	COMMENTS
4100			MOV CX, [4500]	CX <- [4500]
4104			MOV AX, 0001	AX <- 0001
4107			MOV DX, 0000	DX <- 0000
410A		LOOP	MUL CX	DX:AX <- AX * CX
410C			LOOP 410A	Go To [040A] till CX->00
4110			MOV [4600], AX	[4600]<-AX
4114			MOV [4601], DX	[4601]<-DX
4118			HLT	Stop the program

OUTPUT:

INPUT		OUTPUT	
Register	**Data**	**Address**	**Data**
4500	**04**	**4600**	**18**
		4601	**00**
4500	**06**	**4600**	**D0**
		4601	**02**

RESULT:

Thus, an assembly language program for finding the factorial of numbers using 8086 were performed and its outputs were verified.

13. SQUARE OF NUMBER USING 8086 MICROPROCESSOR

AIM:
To write and execute an assembly language program to find the square of a number using 8086.

APPARATUS:
- 8086 Microprocessor kit 1
- Power card 1
- Keyboard 1

ALGORITHM:
- Store 500 to SI and Load data from offset 500 to register CL and set register CH to 00 (for count).
- Increase the value of SI by 1.
- Load first number(value) from next offset (i.e 501) to register AL.
- Multiply the value in register AL by itself.
- Store the result (value of register AL) to memory offset SI.
- Increase the value of SI by 1.
- Loop above 2 till register CX gets 0.

PROGRAM:

ADDRESS	OPCODE	LABEL	PROGRAM	COMMENTS
4100			MOV SI, 4500	set the value of SI to 4500
4103			MOV CL, [SI]	load data from offset SI to register CL
4105			MOV CH, 00	set value of register CH to 00
4107			INC SI	increase value of SI by 1
4108		LOOP	MOV AL, [SI]	load value from offset SI to register AL
410A			MUL AL	multiply value of register AL by AL.
410C			MOV [SI], AL	store value of register AL at offset SI.
410E			INC SI	increase the value of SI by 1.
410F			LOOP 4108	jump to address 4108 if CX not 0 and CX=CX-1.
4111			HLT	Stop the program

OUTPUT:

INPUT		OUTPUT	
Register	**Data**	**Address**	**Data**
4500	04		
4501	03	4601	09
4502	01	4600	01
4503	02	4601	04
4504	05	4602	25

RESULT:

Thus, an assembly language program for finding the factorial of numbers using 8086 were performed and its outputs were verified.

14.ADDITION OPERATION USING 8051 MICROCONTROLLERS

AIM:

To write and execute an assembly language program to Add two 8-bit numbers using 8051.

APPARATUS:

- 8051 microcontroller kit1
- Power card 1
- Keyboard 1

ALGORITHM:

- Load the First Data in A-register.
- Load the Second Data in B-register.
- Add the two data with carry.
- Store the sum in memory location.
- Stop the program.

PROGRAM:

ADDITION

ADDRESS	OPCODE	LABEL	PROGRAM	COMMENTS
4100	74,05		MOV A,#data	Load data 1 in the accumulator.
4102	75,F0,05		MOV B,#data	Load data 2 in B-register
4105	35,F0		ADDC A,B	Add the contents of the accumulator and B-reg with carry.
4107	90,11,00		MOV DPTR,#4500_H	Initialize DPTR with address 4500_H
410A	F0		MOVX @ DPTR,A	Store the Sum in 4500_H
410B	80, FE	STOP:	SJMP STOP	Stop the program

OUTPUT:

INPUT		OUTPUT	
Register	**Data**	**Address**	**Data**
4101	02H	**4500**	05H
4104	03H		

RESULT:

Thus, an assembly language program for addition of given two 8-bit number was written, executed and Verified the Result successfully using 8051 kit

15.SUBTRACTION OPERATION USING 8051 MICROCONTROLLER

AIM:

To write and execute an assembly language program to subtract two 8-bit numbers using 8051.

APPARATUS:

- 8051 microcontroller kit1
- Power card 1
- Keyboard 1

SUBTRACTION

ALGORITHM:

- Load the First Data in A-register.
- Load the Second Data in B-register.
- Subtract the two data with borrow.
- Store the sum in memory location.
- Stop the program.

PROGRAM:

ADDRESS	OPCODE	LABEL	PROGRAM	COMMENTS
4100	74,05		MOV A,#data	Load data 1 in the accumulator.
4102	75,F0,04		MOV B,#data	Load data 2 in B-register
4105	95,F0		SUBB A,B	Subtract the contents of B-reg from accumulator with borrow.
4107	90 11 00		MOV DPTR,#4500$_H$	Initialize DPTR with address 4500$_H$
410A	F0		MOVX @ DPTR,A	Store the difference in 4500$_H$
410B	80, FE	STOP:	SJMP STOP	Stop the program

OUTPUT:

INPUT		OUTPUT	
Register	**Data**	**Address**	**Data**
4101	0Ah	**4500**	08h
4104	02h		

RESULT:

Thus, an assembly language program for subtraction of given two 8-bit number was written, executed and Verified the Result successfully using 8051 kit

16 .MULTIPLICATION OPERATION USING 8051 MICROCONTROLLER

AIM:

To write and execute an assembly language program to multiply two 8-bit numbers using 8051.

APPARATUS:

1. 8051 microcontroller kit 1
2. Power card 1
3. Keyboard 1

MULTIPLICATION

ALGORITHM:

1. Get the multiplier in the accumulator.
2. Get the multiplicand in the B register.
3. Multiply A with B.
4. Store the product in memory locations.
5. Stop the program.

PROGRAM:

ADDRESS	OPCODE	LABEL	PROGRAM	COMMENTS
4100	74,05		MOV A,#data	Load data 1 in the accumulator.
4102	75,F0,05		MOV B,#data	Load data 2 in B-register
4105	A4		MUL AB	A*B, Higher byte of result in B and lower byte of result in A.
4106	90,11,00		MOV DPTR,#4500_H	Initialize DPTR with address 1100_H
4109	F0		MOVX @ DPTR,A	Store the LSB in 4500_H
410A	A3		INC DPTR	Increment Data pointer
410B	E5,F0		MOV A,B	Copy the content of B-reg to A-register.
410D	F0		MOVX @ DPTR,A	Store the MSB in 4501_H
410E	80, FE	STOP:	SJMP STOP	Stop the program

OUTPUT:

INPUT		OUTPUT	
REGISTER	**DATA**	**ADDRESS**	**DATA**
4101	05h	**4500**	0Ah
4104	02h	**4501**	

RESULT:

Thus, an assembly language program for multiplication of given two 8-bit number was written, executed and Verified the Result successfully using 8051 kit

17. DIVISION OPERATION USING 8051 MICROCONTROLLER

AIM:

To write and execute an assembly language program to divide two 8-bit numbers using 8051.

APPARATUS:

1. 8051 microcontroller kit 1
2. Power card 1
3. Keyboard 1

DIVISION

ALGORITHM:

1. Get the Dividend in the accumulator.
2. Get the Divisor in the B register.
3. Divide A by B.
4. Store the Quotient and Remainder in memory.
5. Stop the program.

PROGRAM:

ADDRESS	OPCODE	LABEL	PROGRAM	COMMENTS
4100	74,data1		MOV A,#CF	Load data 1 in the accumulator.
4102	75,data2		MOV B,#21	Load data 2 in B-register
4104	84		DIV AB	Divide. Remainder in A and quotient in B
4105	90,11,00		MOV DPTR,#4500_H	Initialize DPTR with address 1100_H
4108	F0		MOVX @ DPTR,A	Store the quotient in 4500_H
4109	A3		INC DPTR	Increment Data pointer
410A	E5,F0		MOV A,B	Copy the content of B-reg to A-register.
410C	F0		MOVX @ DPTR,A	Store the Remainder in 4501_H
410D	80, FE	STOP:	SJMP STOP	Stop the program

OUTPUT:

INPUT		OUTPUT	
REGISTER	**DATA**	**ADDRESS**	**DATA**
4101	CF	**4500**	**6(quotient)**
4104	21	**4501**	**9(remainder)**

RESULT:

Thus, an assembly language program for Division of given two 8-bit number was written, executed and Verified the Result successfully using 8051 kit

Test Cases: Arithmetic Operations of 8051 Microcontroller

[a] 16-bit Addition

Test Case 1: Basic Addition

ii) **Input:**
1. Number 1: 1234h
2. Number 2: 5678h

iii) **Expected Output:**
1. Result: 68ACh (No carry)

Test Case 2: Addition with Carry

(iii) **Input:**
a. Number 1: FFFFh
b. Number 2: 0001h

(iv) **Expected Output:**
a. Result: 0000h, Carry = 1

Test Case 3: Adding Zero

g. **Input:**
a. Number 1: 4321h
b. Number 2: 0000h

h. **Expected Output:**
a. Result: 4321h, Carry = 0

[b] 16-bit Subtraction

Test Case 1: Basic Subtraction

57. **Input:**
a. Number 1: 5678h
b. Number 2: 1234h

58. **Expected Output:**
a. Result: 4444h, Borrow = 0

Test Case 2: Subtraction with Borrow

1. **Input:**
    1. Number 1: 1234h
    2. Number 2: 5678h
2. **Expected Output:**
    1. Result: BBBC, Borrow = 1

Test Case 3: Subtraction of Zero

- **Input:**
    - Number 1: 1234h

- Number 2: 0000h
- **Expected Output:**
  - Result: 1234h, Borrow = 0

[c] 16-bit Multiplication

Test Case 1: Basic Multiplication

1. **Input:**
   - Number 1: 1234h
   - Number 2: 0002h
2. **Expected Output:**
   - Result: 02468h (16-bit low in ACC, high byte in B)

Test Case 2: Multiplication Leading to Overflow

- **Input:**
  - Number 1: FFFFh
  - Number 2: 0002h
- **Expected Output:**
  - Result: 1FFFEh (requires handling overflow in high byte)

[d] 16-bit Division

Test Case 1: Basic Division

8. **Input:**
   a. Dividend: 1234h
   b. Divisor: 0002h
9. **Expected Output:**
   a. Quotient: 091Ah
   b. Remainder: 0000h

Test Case 2: Division by Larger Number

- **Input:**
  - Dividend: 1234h
  - Divisor: 5678h
- **Expected Output:**
  - Quotient: 0000h
  - Remainder: 1234h

18. LOGICAL OPERATIONS USING 8051

AIM:
To write and execute an assembly language program for Setting and Masking of a given 8-bit number using 8051.

APPARATUS REQUIRED:

1. 8051 microcontroller kit 1
2. Power card 1
3. Keyboard 1

SETTING OF BITS

ALGORITHM:
1. Load the Data in A-register.
2. Load 0F to set the lower nibble in B-register.
3. Perform OR operation with B-register.
4. Store the Result in memory location.
5. Stop the program.

PROGRAM:

ADDRESS	OPCODE	LABEL	PROGRAM	COMMENTS
4100	74,05		MOV A,#C3	Load data 1 in the accumulator.
4102	75,F0,05		MOV B,#0F	Load data 2 in B-register
4105	35,F0		ORL A,B	OR the contents of accumulator and B-reg.
4107	90,11,00		MOV DPTR,#4500_H	Initialize DPTR with address 4500_H
410A	F0		MOVX @ DPTR,A	Store the Result in 4500_H
410B	80, FE	STOP:	SJMP STOP	Stop the program

OUTPUT:

INPUT		OUTPUT	
Register	**Data**	**Address**	**Data**
4101	**C3**	**4500**	**CF**

MASKING OF BITS

ALGORITHM:
1. Load the Data in A-register.
2. Load 0F to mask the higher nibble in B-register.
3. Perform AND operation with B-register.
4. Store the Result in memory location.
5. Stop the program.

PROGRAM:

ADDRESS	OPCODE	LABEL	PROGRAM	COMMENTS
4100	74,05		MOV A,#4D	Load data 1 in the accumulator.
4102	75,F0,05		MOV B,#0F	Load data 2 in B-register
4105	35,F0		ANL A,B	AND the contents of accumulator and B-reg.
4107	90,11,00		MOV DPTR,#4500_H	Initialize DPTR with address 4500_H
410A	F0		MOVX @ DPTR,A	Store the Result in 4500_H
410B	80, FE	STOP:	SJMP STOP	Stop the program

OUTPUT:

INPUT		OUTPUT	
Register	**Data**	**Address**	**Data**
4101	**4D**	**4500**	**0D**

RESULT:

Thus, an assembly language program for Setting and Masking of 8-bit numbers using 8051 were performed and its outputs were verified.

19) Traffic Light Controller using 8051 Microcontroller (AT89C51)

AIM:

To write an assembly language program using Keil µVision to simulate the working of a block traffic light control system using AT89C51 microcontroller, controlling RED, YELLOW, and GREEN lights in a timed sequence.

APPARATUS REQUIRED:

S.No	Component	Specification
1	Microcontroller	AT89C51 / 8051
2	Software	Keil µVision & Proteus
3	Output Interface	LEDs (Red, Yellow, Green)
4	Clock Frequency	11.0592 MHz

THEORY:

Traffic light control is a time-based sequence operation where each signal (Red, Yellow, Green) is turned ON for a specific interval. Using the microcontroller 8051, the ports are configured as output to drive LEDs.

- **Port Configuration:**
  - P1.0 → RED Light
  - P1.1 → YELLOW Light
  - P1.2 → GREEN Light

ALGORITHM:

1. Start the program.
2. Initialize Port 1 as output.
3. Turn **RED ON** for 5 seconds, YELLOW and GREEN OFF.
4. Turn **YELLOW ON** for 2 seconds, RED and GREEN OFF.
5. Turn **GREEN ON** for 5 seconds, RED and YELLOW OFF.
6. Repeat the sequence continuously.
7. End program (in loop).

PROCEDURE / STEPS IN KEIL:

3. Open **Keil µVision**.
4. Create a **new project** and select **AT89C51** microcontroller.
5. Add a new assembly file (.asm) and write the program.
6. Set **Port1 pins** to output.
7. Implement delay using timer or software loop.
8. Build and debug the program.
9. Open **Proteus**, place **AT89C51 and LEDs**, connect LEDs to P1.0, P1.1, P1.2.

10. Load the **hex file** into Proteus and run the simulation.
11. Observe the traffic light sequence.

ASSEMBLY LANGUAGE PROGRAM (Keil – AT89C51)

```
ORG 0000H       ; Start of program

START: MOV P1, #00H  ; Clear all lights (initialize Port1)

RED:   MOV P1, #01H  ; RED ON (P1.0)
       ACALL DELAY5S

YELLOW:MOV P1, #02H  ; YELLOW ON (P1.1)
       ACALL DELAY2S

GREEN: MOV P1, #04H  ; GREEN ON (P1.2)
       ACALL DELAY5S
       SJMP START    ; Repeat cycle

;--------------------------------
; Delay of approx 5 seconds
;--------------------------------
DELAY5S:
       MOV R2, #5    ; 5 loops of 1 second
DLY5:  ACALL DELAY1S
       DJNZ R2, DLY5
       RET

;--------------------------------
; Delay of approx 2 seconds
;--------------------------------
DELAY2S:
       MOV R2, #2
DLY2:  ACALL DELAY1S
       DJNZ R2, DLY2
       RET

;--------------------------------
; Delay of approx 1 second
;--------------------------------
DELAY1S:
       MOV R3, #255
D1:    MOV R4, #255
D2:    DJNZ R4, D2
       DJNZ R3, D1
       RET

END
```

INPUT / OUTPUT TABLE

Time (Seconds)	P1.0 (RED)	P1.1 (YELLOW)	P1.2 (GREEN)	LED Status
0 – 5	1	0	0	RED ON
5 – 7	0	1	0	YELLOW ON
7 – 12	0	0	1	GREEN ON
Repeat	Cycle repeats continuously			

RESULT:

The program successfully simulates a traffic light control system using the AT89C51 microcontroller in Keil. LEDs connected to Port1 blink in the correct sequence (RED → YELLOW → GREEN) with specified timing delays.

20) Multi-byte Addition Using 8051 Microcontroller (AT89C51) in Assembly Language (Keil Software)

AIM:

To write and execute an Assembly Language Program (ALP) using Keil software for adding two multi-byte numbers stored in memory and store the result including carry using the AT89C51 microcontroller.

APPARATUS REQUIRED:

S.No	Component	Description
1	Microcontroller	AT89C51 / 8051
2	Software	Keil µVision, Proteus
3	Memory	Internal RAM
4	Clock	11.0592 MHz Crystal

THEORY:

- A microcontroller can only add one byte at a time.
- For **multi-byte addition**, the addition starts from the **least significant byte (LSB)** and proceeds to the **most significant byte (MSB)**, propagating any carry between bytes.

ALGORITHM:

1. Start the program.
2. Initialize the data segment and pointer register.
3. Load the total number of bytes to be added.
4. Clear the carry flag.
5. Add the bytes one by one from memory.
6. Store result and propagate carry to the next byte.
7. If carry remains after final addition, store it separately.
8. End program.

PROCEDURE (Keil & Proteus Simulation Steps):

1. Open **Keil µVision** → Create a New Project → Select **AT89C51**.
2. Add a new assembly file → Save as .asm.
3. Write and assemble the code.
4. Build the project to generate .hex file.
5. Open **Proteus** → Place AT89C51 and RAM blocks (or memory).
6. Connect output LEDs to monitor the result.
7. Load the .hex file into the microcontroller in Proteus.

8. Run simulation and observe the output memory or LEDs.

ASSEMBLY LANGUAGE PROGRAM (Keil - AT89C51)

Assumption:

- First number stored from address 30H (LSB) to 33H (MSB)
- Second number stored from address 40H (LSB) to 43H (MSB)
- Result stored starting from 50H (LSB) to 54H (MSB including carry)

```
ORG 0000H

MOV R0, #30H      ; Pointer for first number
MOV R1, #40H      ; Pointer for second number
MOV R2, #50H      ; Pointer for result location
MOV R3, #04H      ; Number of bytes (4 bytes)

CLR C             ; Clear carry flag

BACK: MOV A, @R0   ; Load byte from first number
     ADDC A, @R1  ; Add byte with carry from second number
     MOV @R2, A   ; Store result byte

     INC R0       ; Increment pointers
     INC R1
     INC R2
     DJNZ R3, BACK ; Loop until all bytes are added

; Store final carry if any
MOV A, #00H
ADDC A, #00H      ; Add carry to zero
MOV @R2, A        ; Store final carry in next location

SJMP $

END
```

INPUT (Memory Representation)

Address	**30H**	**31H**	**32H**	**33H**
Data 1	56H	A2H	11H	20H
Address	**40H**	**41H**	**42H**	**43H**
Data 2	50H	01H	22H	10H

OUTPUT (After Execution)

Address	50H	51H	52H	53H	54H
Result	A6H	A3H	33H	30H	00H

(Last byte 54H stores carry. If any carry is generated, it will be stored here.)

RESULT:

The Assembly Language Program for multi-byte addition was successfully written, compiled in Keil, and simulated using Proteus. The program correctly added two 4-byte numbers with carry propagation and stored the result in consecutive memory locations.

21) Block Move from one memory location to another using Keil software for the 8051 microcontroller (AT89C51)

AIM:
To write and execute an Assembly Language Program using Keil software for the 8051 microcontroller (AT89C51) to move a block of data from one memory location to another.

APPARATUS REQUIRED:

S.No	Apparatus / Software	Quantity
1	AT89C51 Microcontroller / Keil µVision	1
2	USB Programmer / Proteus Simulator	1
3	PC with Keil µVision Software Installed	1

ALGORITHM:

1. Start the program.
2. Initialize the **Data Pointer (DPTR)** to point to the **source address** of the data block.
3. Load the **R0 register** with the **destination address** where data is to be moved.
4. Load the **counter register (R2)** with the **number of bytes** to be moved.
5. Move one byte from the source to the accumulator (A).
6. Transfer the accumulator content to the destination location.
7. Increment both source and destination pointers.
8. Decrement the counter.
9. Repeat the process until the counter becomes zero.
10. Stop the program.

PROCEDURE / STEPS:

1. Open **Keil µVision** software.
2. Create a **new project** and select the **AT89C51** device.
3. Write the assembly language code in the **editor window**.
4. Save the program with the extension **.asm**.
5. Assemble the program and correct any syntax errors.
6. Simulate the program using the **Proteus simulator** (optional).
7. Observe the data transfer from source to destination memory locations.
8. Verify the result.

PROGRAM: (Block Move in Assembly Language)

```
ORG 0000H

MOV DPTR, #3000H     ; Source address of block
MOV R0, #4000H       ; Destination address
MOV R2, #05H         ; Number of bytes to move
```

```
BACK: MOVX A, @DPTR   ; Get data from source
INC DPTR              ; Increment source address
MOV @R0, A            ; Move data to destination
INC R0                ; Increment destination address
DJNZ R2, BACK         ; Repeat until count = 0

HERE: SJMP HERE       ; Stop program (infinite loop)

END
```

INPUT:

Memory Address	Data (Hex)	Description
3000H	12H	Source Data 1
3001H	34H	Source Data 2
3002H	56H	Source Data 3
3003H	78H	Source Data 4
3004H	9AH	Source Data 5

EXPECTED OUTPUT:

Destination Address	Data (Hex)	Description
4000H	12H	Moved Data 1
4001H	34H	Moved Data 2
4002H	56H	Moved Data 3
4003H	78H	Moved Data 4
4004H	9AH	Moved Data 5

RESULT:

The assembly language program for **block move from one memory address to another** using **Keil Software for AT89C51** was successfully written, executed, and verified. The data bytes from **3000H–3004H** were correctly moved to **4000H–4004H**.

22) Interface Stepper Motor with 8-bit Microprocessor to Run in Clockwise and Anticlockwise direction

AIM:

To write and execute an assembly language Program to run a stepper motor at different speed, and to control its
direction using 8085 Microprocessor

APPARATUS:

1. 8085 microprocessor kit 1
2. Stepper Motor 1
3. Stepper Motor Interface board 1
4. Power card 1
5. Keyboard 1

PROGRAM:

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
4100	START	LXI H, 4200		Initialize HL with 4200H
4103		MVI C, 04		Copy the value 04 to C- register
4105	NEXT	MOV A, M		Copy the content M to A-register
4106		OUT C0		The content of A is moved to Out port
4108		LXI D, 1010		Copy the data 1010 to DE-reg Pair
410B	loop	DCX D		Decrement DE-register
410C		MOV A,E		
410D		ORA D		Check DE = 0000
410E		JNZ loop		Jump on no zero to loop
4111		INX H		Increment HL -register Pair
4112		DCR C		Decrement the count
4113		JNZ NEXT		Jump to NEXT if Z flag is zero
4115		JMP START		Jump to label START
4118		HLT		Stop the program.
4200	TABLE	09 05 06 0A	01100110	Lookup table for **clockwise direction**
4200	TABLE	0A 06 05 09		Lookup table for **Counter clockwise direction**

OUTPUT

Switching sequence	Clockwise rotation				Anticlockwise rotation			
	PA_3	PA_2	PA_1	PA_0	PA_3	PA_2	PA_1	PA_0
Sequence-1	1	1	0	0	0	0	1	1
Sequence-2	0	1	1	0	0	1	1	0
Sequence-3	0	0	1	1	1	1	0	0
Sequence-4	1	0	0	1	1	0	0	1

RESULT:

Thus, an assembly language Program to run the stepper motor in both forward and reverse direction with delay was executed and its output was verified.

TEST CASE :Interface Stepper Motor Controller with 8085 Microprocessor

Test Case Question:
i) Write an 8085 assembly program that makes the motor rotate clockwise for a set period, then stops it.

Input: Port 1: 01H (clockwise rotation).

Expected Output:

The motor should rotate clockwise for a specific duration (e.g., 5 seconds).

After the set duration, the motor should stop, and Port 2 should output 00H.

ii) Write an 8085 assembly program that handles rapid changes in direction inputs (e.g., alternating between 01H and 02H) without causing mechanical issues to the motor.

Input: Port 1: Alternating between 01H and 02H at quick intervals.

Expected Output:

The motor should handle the rapid changes in direction smoothly without damage or failure.

Port 2 should output the corresponding bit patterns (0AH and 05H) appropriately.

23.KEYBOARD AND DISPLAY INTERFACING

AIM:
To write and execute an assembly language Program to display a character "7" and the rolling message "HELP US" in the display.

APPARATUS:

1. 8086 microprocessor kit 1
2. 8279 Interface board 1
6. Power card 1
7. Keyboard 1
8.

ROLLING MESSAGE "HELP US"

ALGORITHM:

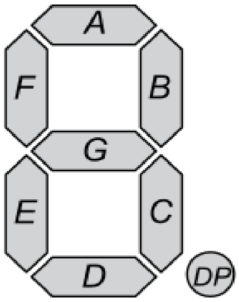
- Display of rolling message "HELP US "
- Initialize the counter
- Set 8279 for 8-digit character display, right entry
- Set 8279 for clearing the display
- Write the command to display
- Load the character into accumulator and display it
- Introduce the delay
- Repeat from step 1.

PROGRAM:

ADDRESS	LABEL	PROGRAM	OPCODE	COMMENTS
1100	START	MOV SI,1200H		Initialize array
1104		MOV CX,000FH		Initialize array size
1108		MOV AL,10		Store the control word for display mode
110B		OUT C2,AL		Send through output port
110D		MOV AL,0CC		Store the control word to clear display
1110		OUT C2,AL		Send through output port
1112		MOV AL,90		Store the control word to write display
1115		OUT C2,AL		Send through output port
1117	NEXT	MOV AL,[SI]		Get the first data
1119		OUT C0,AL		Send through output port
111B	DELAY	MOV DX,0FFFFH		Store 16bit count value
111F	LOOP1	DEC DX		Decrement count value
1120		JNZ LOOP1		Loop until count values becomes zero
1122		INC SI		Go & get next data
1123		LOOP NEXT		Loop until all the data has been taken
1125		JMP START		Go to starting location
1127		HLT		

LOOK-UP TABLE:

1200	98	68	7C	C8
1204	FF	1C	29	FF



OUTPUT:

ON – 0   OFF - 1

MEMORY LOCATION	**Message**	**7-SEGMENT LED FORMAT**								**HEX CODE**
		D	**C**	**B**	**A**	**DP**	**G**	**F**	**E**	
1200H	**H**	1	0	0	1	1	0	0	0	**98**
1201H	**E**	0	1	1	0	1	0	0	0	**68**
1202H	**L**	0	1	1	1	1	1	0	0	**7C**
1203H	**P**	1	1	0	0	1	0	0	0	**C8**
1204H		1	1	1	1	1	1	1	1	**FF**
1205H	**U**	0	0	0	0	1	1	0	0	**1C**
1206H	**S**	0	0	1	0	1	0	0	1	**29**
1207H		1	1	1	1	1	1	1	1	**FF**

24) Interface 8279 with 8086 for: Rolling display using Keyboard Display Controller

ALGORITHM:

- Set 8279 for 8-digit character display, right entry
- Set 8279 for clearing the display
- Write the command to display
- Load the character into accumulator and display it
- Repeat from step 1.

PROGRAM:

ADDRESS	LABEL	PROGRAM	OPCODE	COMMENTS
1100		MOV AL,00	C6 C0 00	Store the control word for display mode
1103		OUT C2,AL	E6 C2	Send through output port
1105		MOV AL,0CC	C6 C0 CC	Store the control word to clear display
1108		OUT C2,AL	E6 C2	Send through output port
110A		MOV AL,90	C6 C0 90	Store the control word to write display
110D		OUT C2,AL	E6 C2	Send through output port
110F		MOV AL,8F	C6 C0 8F	Get the first data
1112		OUT C0,AL	E6 C0	Send through output port
1114		HLT	F4	Stop the program

INPUT		OUTPUT	
Address	**Data**	**Address**	**Data**
1111	**98**	**Display**	**H**

RESULT:

Thus, the rolling message “HELP US” and the character “H” are displayed using an 8279-interface kit with 8086 Microprocessor.

Test Cases: 8279 Keyboard Display Controller

Test Case Question 1:

Test case to demonstrate displaying a character "A" on a common cathode 7-segment display using the 8279 and an 8086 microprocessor:

Test Case: Display Character "A" on a 7-Segment Display

Test Case Scenario:

· Initialize the 8279 Keyboard Display Controller for 7-segment display mode.

· Send the ASCII code for the character 'A' to the 8279.

· Observe the character 'A' displayed on the 7-segment display.

Test Case Steps:

· Initialize the 8279 controller:

· Write the Control Word 1 (CW1) to enable the display.

· Write the Control Word 2 (CW2) to set the display mode for a 7-segment display (Set 0 or Set 1).

· Set other relevant configuration settings as needed.

· Send the ASCII code for the character 'A' (which is 0x41 in hexadecimal) to the 8279 data buffer.

· Wait for a short delay to allow the 8279 to process the data.

· Verify that the 7-segment display connected to the 8279 is showing the character 'A' in a 7-segment format.

INPUT		OUTPUT	
Address	**Data**	**Address**	**Data**
1111	**88**	**Display**	**A**

25 .INTERFACE SWITCHES WITH 8086 THROUGH 8255

AIM:

To write and execute an assembly language Program to Interface 8 switches with 8086 Microprocessor through 8255 PPI.

APPARATUS:

1. 8086 microprocessor kit 1
2. 8255 Interface board 1
3. Power card 1
4. Keyboard 1

ALGORITHM:

1. Configure the 8255 port A as input port with the control reg value as "90H"
2. Read the port A switch status through C0.
3. Store the output in 1250.
4. Stop

PROGRAM:

ADDRESS	LABEL	PROGRAM	OPCODE	COMMENTS
1100		MOV AL,90		Load the AL with control word
1103		OUT C2,AL		Send the control word to control reg of 8255
1105		IN AL,C0		Read port A
1108		MOV [1250],AL		Store the result on memory
1114		HLT	F4	Stop the program

INPUT	OUTPUT	
VARY THE SWITCH POSITIONS ON OFF ON ON OFF ON OFF ON	Address	Data
	1250	B5

RESULT

Thus, an assembly language program for Interfacing of switches with 8086 through 8255 PPI was written, executed and Verified the Result successfully.

Test Cases: 8255 Programmable Input Output Port

Test Case 1: Verify Data Transfer to Port A

Test Case Objective:

To verify that data is correctly transferred to Port A of the 8255 PPI interface.

Test Scenario:

Write a specific data value to Port A and check if it is correctly reflected.

Test Steps:

3. Initialize the 8255 PPI in Mode 0.
4. Set the control word to configure Port A as an output port.
5. Write a data value (e.g., 0xAA) to Port A.
6. Read back the value from Port A.

Input:

- Control Word: 0x80 (Configures Port A as output, Mode 0)
- Data Value: 0xAA

Expected Output:

2. Port A should reflect the written data value 0xAA.

Register Address:

4. Control Word Register: 0x03F3

Port A Address: 0x03F0

Test Case 2: Verify Data Reception from Port B

Test Case Objective:

To verify that data is correctly received from Port B of the 8255 PPI interface.

Test Scenario:

Read data from Port B and check if it matches the expected input value.

Test Steps:

- Initialize the 8255 PPI in Mode 0.
- Set the control word to configure Port B as an input port.
- Simulate or input a data value (e.g., 0x55) on Port B.
- Read the value from Port B.

Input:

3. Control Word: 0x90 (Configures Port B as input, Mode 0)
4. Expected Data Value: 0x55

Expected Output:

9. The data read from Port B should be 0x55.

Register Address:

- Control Word Register: 0x03F3

Port B Address: 0x03F1

26 : Interfacing EEPROM (24C02) with Microcontroller using I²C

Aim
To simulate and verify I²C communication between a microcontroller and an external EEPROM (24C02) using Proteus and to write and read data from the EEPROM.

Apparatus Required

- PC with **Proteus** and **Keil µVision / Arduino IDE**
- **8051 Microcontroller / Arduino UNO**
- **24C02 EEPROM (I²C device)**
- Virtual **Serial Monitor / LCD**
- Connecting wires and pull-up resistors (4.7 kΩ)

Algorithm

1. **Start condition** – Send START signal on I²C bus.
2. **Send Device Address** (1010 + A2 A1 A0 + R/W bit).
3. **Send Memory Location Address** where data is to be written.
4. **Send Data Byte** to be stored.
5. **Send STOP condition.**
6. **Wait for some time** (internally EEPROM writes data).
7. **Send START condition** again for reading.
8. **Send Device Address with Read bit = 1.**
9. **Read the Data Byte** from EEPROM.
10. **Display the read data** on LCD or Serial Terminal.

Program (8051 Assembly in Keil or Arduino Code)

Option A – 8051 (C Language in Keil)

```
#include <reg51.h>
#define SDA P2_0
#define SCL P2_1

void delay() {
  unsigned int i;
  for(i=0; i<200; i++);
}

void I2C_Start() {
  SDA = 1; SCL = 1; delay();
  SDA = 0; delay();
  SCL = 0;
}

void I2C_Stop() {
  SDA = 0; SCL = 1; delay();
  SDA = 1;
```

```
}

void I2C_Write(unsigned char dat) {
  unsigned char i;
  for(i=0;i<8;i++) {
    SDA = (dat & 0x80)>>7;
    SCL = 1; delay();
    SCL = 0;
    dat <<= 1;
  }
}

unsigned char I2C_Read() {
  unsigned char i, dat=0;
  SDA = 1;
  for(i=0;i<8;i++) {
    SCL = 1; delay();
    dat = (dat<<1) | SDA;
    SCL = 0; delay();
  }
  return dat;
}

void main() {
  unsigned char x;
  I2C_Start();
  I2C_Write(0xA0);   // Write address
  I2C_Write(0x00);   // Memory address
  I2C_Write('A');    // Data to write
  I2C_Stop();

  delay(); delay();  // Internal write delay

  I2C_Start();
  I2C_Write(0xA0);   // Device address
  I2C_Write(0x00);   // Memory address
  I2C_Start();
  I2C_Write(0xA1);   // Read mode
  x = I2C_Read();
  I2C_Stop();

  P1 = x;            // Display read data on Port 1 LEDs
  while(1);
}
```

Option B – Arduino (C++)

```
#include <Wire.h>

void setup() {
  Wire.begin();
  Serial.begin(9600);

  // Write 'A' (0x41) to memory address 0x00
  Wire.beginTransmission(0x50);
  Wire.write(0x00);
  Wire.write('A');
  Wire.endTransmission();
```

```
  delay(10);

  // Read back
  Wire.beginTransmission(0x50);
  Wire.write(0x00);
  Wire.endTransmission();

  Wire.requestFrom(0x50, 1);
  if (Wire.available()) {
    char data = Wire.read();
    Serial.print("Read from EEPROM: ");
    Serial.println(data);
  }
}

void loop() { }
```

Steps to Simulate in Proteus

1. Open **Proteus** and place the following components:
   - 8051 or Arduino UNO
   - **24C02 EEPROM**
   - **Virtual Terminal / LCD**
2. Connect SDA → P2.0 and SCL → P2.1 (or A4/A5 for Arduino).
3. Add **4.7kΩ pull-up resistors** to SDA and SCL lines.
4. Load the compiled **HEX file** into the microcontroller.
5. Run the simulation.
6. Observe the output on **Port LEDs / Virtual Terminal**.

Output

- The microcontroller writes ‘A’ to EEPROM address 0x00 and reads it back.
- On the **Virtual Terminal** or Port LEDs, the data **‘A’** (ASCII 65 or binary 01000001) is displayed.

Result

The simulation of I²C communication between microcontroller and 24C02 EEPROM was successfully verified using Proteus. Data was correctly written and read back from the EEPROM.

27: Interfacing Real-Time Clock (DS1307) with Microcontroller using I²C

Aim

To simulate and verify I²C communication between a microcontroller and an RTC (DS1307) in Proteus to display the current time on an LCD.

Apparatus Required

- PC with Proteus, Keil µVision / Arduino IDE
- 8051 / Arduino UNO
- RTC DS1307
- 16x2 LCD Display
- Pull-up Resistors (4.7 kΩ)

Algorithm

1. Initialize LCD.
2. Initialize I²C bus.
3. Set initial time data to DS1307 (hours, minutes, seconds).
4. Continuously read time data (hours, minutes, seconds) from DS1307.
5. Convert BCD data to ASCII.
6. Display time on the LCD.

Program (Arduino)

```
#include <Wire.h>
#include "RTClib.h"
#include <LiquidCrystal.h>

RTC_DS1307 rtc;
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);

void setup() {
  Wire.begin();
  lcd.begin(16, 2);
  rtc.begin();

  // Set time (only once, comment after setting)
  // rtc.adjust(DateTime(2025, 10, 16, 10, 45, 0)); // YYYY, MM, DD, HH,
MM, SS
}

void loop() {
  DateTime now = rtc.now();
  lcd.setCursor(0, 0);
  lcd.print("Time: ");
  if (now.hour() < 10) lcd.print("0");
  lcd.print(now.hour());
  lcd.print(":");
  if (now.minute() < 10) lcd.print("0");
  lcd.print(now.minute());
  lcd.print(":");
```

```
  if (now.second() < 10) lcd.print("0");
  lcd.print(now.second());
  delay(1000);
}
```

Steps to Simulate in Proteus

3. Open **Proteus** and place:
    - **Arduino UNO**, **DS1307**, **16x2 LCD**, **Pull-up resistors (4.7kΩ)**
4. Connect SDA → A4, SCL → A5.
5. Connect LCD pins RS→7, EN→6, D4→5, D5→4, D6→3, D7→2.
6. Power DS1307 (VCC, GND) and connect a **32.768 kHz crystal** with pins 1 and 2 of DS1307.
7. Load Arduino HEX file.
8. Run simulation.

Output

- The LCD displays the current time, e.g.
- `Time: 10:45:08`
- The seconds increment every second.

Result

The simulation of I²C communication between microcontroller and RTC (DS1307) was successfully verified in Proteus. The current time was displayed and updated on the LCD screen.