Department of Computer Science Engineering

2017 Regulation

EC8681 Microprocessor and Microcontroller Laboratory Manual

(Anna University Regulation 2017)

Name	:
Register Number	:
Lab Name/Code	:
Semester/Year	:

St, Anne's College of Engineering and Technology

Affiliated to Anna University and Approved by AICTE, New Delhi

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

BONAFIDE CERTIFICATE

This is to Certified that bonafide record of work done by Mr./Ms./Mrs. ________________________ in the **EC8681 Microprocessor and Microcontroller Laboratory** for the course of Electronics and Communication Engineering during VIth Semester of academic year 2019:20.

STAFF IN-CHARGE **HOD**

Register No.: _______________

This record is submitted for VIth semester Electronics and Communication Engineering practical examination of Anna University, Chennai held on__________.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

INDEX

S.No	Date	Name of the Experiment	Page No.	Marks	Initial of Faculty
DESIGN AND TEST EXPERIMENTS					

Average:

Date :

Ex.No.: 1

ADDITION AND SUBTRACTION OF TWO 16 BIT NUMBERS USING 8086

AIM:

To add and subtract two 16-Bit numbers stored at consecutive memory locations.

APPARATUS REQUIRED:

8086 kit

ALGORITHM: (16 bit addition)

1. Start the program.
2. Load the first data in AX register.
3. Load the second data in BX register.
4. Clear the CL registers for carry.
5. Add the two data and get the sum in AX REGISTER.
6. Store the sum in memory location.
7. Check for carry. If carry flag is set then go to next step, otherwise go to step8
8. Increment the carry in memory.
9. Store the carry in memory.
10. Stop the program.

PROGRAM

Address	Label	Opcode	Mnemonics	Operand	Comments
1000		8B 06 00 20	MOV	AX,[2000]	Load the first data
1004		8B 1E 02 20	MOV	BX,[2002]	Load the second data
1008		C6 C1 00	MOV	CL,00	Clear the CL register for carry
100B		01 D8	ADD	AX,BX	Add two number sum ill be AX
100D		73 02	JNC	# (1011)	Check the status of carry flag
100F		FE C1	INC	CL	If carry flag is set, increment CL
1011	#	89 06 04 20	MOV	[2004],AX	Store the sum result
1015		88 0E 06 20	MOV	[2006],CL	Store the carry
1019		F4	HLT	-	Stop the program

OBSERVATION:

	Input		Output	
	Address	**Data**	**Address**	**Data**
ADDITION	2000		2004	
	2001		2005	
	2002		2006	
	2003			

OBSERVATION:

	Input		Output	
	Address	**Data**	**Address**	**Data**
SUBTRACTION	2000		2004	
	2001		2005	
	2002		2006	
	2003			

ALGORITHM: (16 bit Subtraction)

1. Start the program.
2. Set SI register as pointer for data.
3. Get the minuend AX register.
4. Get the subtrahend in BX register.
5. Clear CL register to account for sign.
6. Subtract the content of BX from AX, the difference will be in AX.]
7. Check for carry, if carry flag is set then go to next step, otherwise go to step 9.
8. Increment CL register by 1.
9. Take 2's complement of the difference in AX register.
10. Store the magnitude of the difference in memory.
11. Store the sign bit in memory.
12. Stop the program.

SUBTRACTION

Address	Label	Opcode	Mnemonics	Operand	Comments
1000		C7 C6 00 20	MOV	SI,2000	Initialize the SI value
1004		8B 04	MOV	AX,[SI]	First input value move to AX
1006		8B 5C 02	MOV	BX,[SI+2]	Second value move to bx
1009		29 D8	SUB	AX,BX	Sub AX and BX
100B		73 08	JNC	# (1015)	Check the condition
100D		FE C1	INC	CL	Increment CL value
100F		F7 D0	NOT	AX	Ones complements
1011		81 C0 01 00	ADD	AX,0001	Twos complements
1015	#	89 44 04	MOV	[SI+4],AX	Store the result
1018		88 4C 06	MOV	[SI+6],CL	Store the carry value
101B		F4	HLT	-	stop

RESULT :

Date :

Ex.No.:

MULTIPLICATION AND DIVISION OF TWO 16 BIT NUMBERS USING 8086

AIM:

To write an assembly language program for the multiplication and division of two 16 bit numbers using 8086 microprocessor kit

APPARATUS REQUIRED:

8086 kit

ALGORITHM: (16 BIT MULTIPLICATION)

1. Load the address of data in SI register.
2. Get the first data in AX register.
3. Get the second data in BX register.
4. Multiply the content of AX & BX.
5. The product will be in AX & DX
6. Save the product (AX & BX) in memory.
7. Stop the program.

PROGRAM:

MULTIPLICATION

Address	Opcode	Mnemonics	Operand	Comments
1000	C7 C6 00 11	MOV	SI,1100	Set SI as a pointer for data
1004	8B 04	MOV	AX,[SI]	Get the first data in AX register
1006	8B 5C 02	MOV	BX,[SI+2]	Get the second data in BX
1009	F7 E3	MUL	BX	Multiply AX and BX
100B	89 44 04	MOV	[SI+4],AX	The product will be in AX and DX register
100E	89 54 06	MOV	[SI+6],DX	Save the lower, upper 16 bits of the product in memory
1011	F4	HLT	-	Stop

OBSERVATION:

	Input		Output	
	Address	**Data**	**Address**	**Data**
MULTIPLICATION	1100		1104	
	1101		1105	
	1102		1106	
	1103		1107	

OBSERVATION:

	Input		Output	
	Address	**Data**	**Address**	**Data**
DIVISION	1100		1104	
	1101		1105	
	1102		1106	
	1103		1107	

ALGORITHM: (16 BIT DIVISION)

1. Load the address of data in SI register.
2. Get the dividend in AX register.
3. Get the divisor in BX register.
4. Divide the two numbers.
5. Store the result in memory.
6. Stop the program

PROGRAM

DIVISION

Address	Opcode	Mnemonics	Operand	Comments
1000	8B 06 00 11	MOV	AX,[1100]	Get the first data in AX.
1004	8B 1E 02 11	MOV	BX,[1102]	Get the second data in BX
1008	F7 F3	DIV	BX	Divide AX by BX.
100A	89 06 04 11	MOV	[1104],AX	The Quotient stored in 3000.
100E	89 16 06 11	MOV	[1106],DX	The Remainder stored in 3002
1012	F4	HLT	-	Stop

RESULT:

Date :

Ex.No.:

LOGICAL OPERATION USING 8086

AIM:

To write an assembly language program for one's complement and AND operation using 8086 microprocessor kit.

APPARATUS REQUIRED:

8086 kit

ALGORITHM:

1. Start the program.
2. Move the data to accumulator.
3. Give the instruction for ONES complement and AND operation.
4. Store the result in respective address.
5. Stop the program.

PROGRAM

ONES COMPLEMENT

Address	Label	Opcode	Mnemonics	Operand	Comments
1000		C7,C0,34,12	MOV	AX,1234	Move 1234 to accumulator
1004		F7,D0	NOT	AX	Ones complement of AX
1006		89,06,00,14	MOV	[1400],AX	Move AX to address of 1400
100A		F4	HLT	-	Stop

OBSERVATION:

ONES COMPLEMENT

INPUT	**OUTPUT**	
DATA	**ADDRESS**	**DATA**
	1400	
	1401	

AND OPERATION

INPUT		**OUTPUT**	
ADDRESS	**DATA**	**ADDRESS**	**DATA**
1200		1400	
1201		1401	

AND OPERATION

Address	Label	Opcode	Mnemonics	Operand	Comments
1000		8B,06,00,12	MOV	AX,[1200]	Move the content of 1200 to accumulator
1004		81,E0,0F,0F	AND	AX,0F0F	And 0F0F with AX
1008		89,06,00,14	MOV	[1400],AX	Move AX to address of 1400
100C		F4	HLT	-	Stop

RESULT:

Date :

Ex.No.: 2

Block Transfer without Overlap

AIM :

To write an Assembly Language Program (ALP) for moving a data block without overlap using 8086.

APPARATUS REQUIRED:

8086 kit

PROGRAM:

Address	Label	Opcode	Mnemonics	Operand	Comments
1000		C7,C6,00,20	MOV	SI,2000	Load Source Address in SI
1004		C7, C7,00,21	MOV	DI,2100	Load Destination address in DI
1008		C7,C1,00,FF	MOV	CX,00FF	Load number of bytes transferred to CX reg.
100C		FC	CLD	-	Clear Direction Flag
100D	L1	A4	MOVSB	-	Move a block of string byte from the source to the destination
100E		E2,FD	LOOP	L1 (100D)	Facilitate auto incrementing of the index register
1010		F4	HLT		Stop

OBSERVATION

Memory Address	Data
Input [2000] to [20FE]	
Output [2100] to [21FE]	

RESULT:

Date :

Ex.No.: 4

STRING OPERATIONS

AIM:

To perform string manipulation operations using 8086 string primitive.

APPARATUS REQUIRED:

8086 kit

THEORY:

The 8086 instruction set includes called the string primitives. Each string primitive instruction performs a sequence of operations normally handled by an instruction loop. The string primitive instruction performs an operation specified by the primitive, then increments or decrements the pointer registers involved in the operation. On each iteration the affected pointer registers can be either incremented or decremented by 1 or 2.

Pointer registers will be incremented if the value of the Direction Flag in the Flags Register is 0; affected pointer will be decremented if the value of the Direction Flag is 1. The affected pointer registers will be incremented or decremented by 1 if the low-order bit of the string primitive operation code is 0. If the low-order bit of the string primitive operation code is 1,the affected pointer registers will be incremented or decremented by 2.

There are five primitives;

MOV -	Move 8 or 16 bit data in memory
LODS -	Load 8 or 16 bits of data from memory into AL or AX register
STOS -	Store the AL (8-bit operation) or AX (16-bit operation) register into memory
SCAS -	Compare the AL (8-bit operation) or AX (16-bit operation) register with memory
CMPS -	Compare two strings of memory locations.

Use of index registers and string primitives along with direction flag status enables efficient array and string manipulation as shall be evident from the following examples.

Since the 8086 includes the string primitives which require initialization of the index register the SI and DI registers are initialized to start of the source and start of the destination array respectively. The direction flag is cleared to facilitate auto-incrementing of the index registers. The CX register is used to perform the operation repeatedly. The string primitive is used in MOVS. In the case of MOVE operation, the status of the direction flag is however immaterial.

PROGRAM1:

Input [2000] to [20FE]	
Output [2100] to [21FE]	

PROGRAM 2:

Input [AL]	
Output [2000]	

EXAMPLE-1:

The data for the source array has to be initially entered. Hence let us fill the source locations starting from 2000 using the FILL command in the kit. Fill locations from 2000 to 20FF with data XX.

PROGRAM-1 :

Address	Label	Opcode	Mnemonic	Operand	Comments
1000		C7,C6,00,20	MOV	SI,2000	Load Source Address in SI
1004		C7, C7,00,21	MOV	DI,2100	Load Destination address in DI
1008		C7,C1,00,FF	MOV	CX,00FF	Load number of bytes transferred to CX reg.
100C		FC	CLD	-	Clear Direction Flag
100D	L1	A4	MOVSB	-	Move a block of string byte from the source to the destination
100E		E2,FD	LOOP	L1 (100D)	Facilitate auto incrementing of the index register
1010		F4	HLT	-	Stop

EXAMPLE-2:

This program uses the string primitive STOS. The function of this is that it will store the byte in AL or the word in AX depending upon the operand size from the location pointed to by the destination index DI. So if we want to fill a block with a particular data then we should set destination index to the beginning of the block and then use the STOSW instruction or the STOSB instruction and use CXto get the required length. S_ARRAY is the location 2000 in this program.

PROGRAM -2:

Address	Label	Opcode	Mnemonic	Operand	Comments
1000		C7,C1,00,FF	MOV	CX,00FF	00FF move to CX
1004		C7,C7,00,20	MOV	DI,2000	Get 2000 in DI
1008		C7,C0,	MOV	AL, Data	Move To AL
100C		FC	CLD	-	Clear
100D	L	FE	STOSB	-	Store byte value
100E		E2,FD	LOOP	L (100D)	Continue
1010		F4	HLT	-	Stop

PROGRAM -3:

INPUT :

Input [AX]	

OUTPUT :

Address	**Data**
2000	
2001	
2002	
2003	
.	.
20FE	

PROGRAM 3 :

Address	Label	Opcode	Mnemonic	Operand	Comments
1000		C7,C1,00,FF	MOV	CX,00FF	00FF move to CX
1004		C7,C7,00,20	MOV	DI,2000	Get address in DI
1008		C7,C0,	MOV	AX, Data	Get the data
100C		FC	CLD	-	Clear Direction Flag
100D	L	FD	STOSW	-	Store the word
100E		E2,FD	LOOP	L (100D)	Continue till string is stored
1010		F4	HLT	-	Stop

PROCEDURE:

1. Enter the program from location 1000
2. Fill FF locations as stated above with particular data.
3. Execute the program.
4. Check if the contents are duplicated to another 255 locations using the compare command in the kit

RESULT:

Date :
Ex.No. : 4

SORTING AN ARRAY USING 8086

AIM:

To write an assembly language program to sort an array of data in ascending and descending order using 8086 microprocessor kit.

APPARATUS REQUIRED:

8086 kit

ALGORITHM:

1. Set SI register as pointer for array.
2. Set CL register as count for N-1 repetitions Initialize array pointer.
3. Set CH as count for N – 1 comparison.
4. Increment the array in AL register.
5. Get the element of array in AL register.
6. Increment the array in pointer.
7. Compare the next element of array in AL
8. Check carry flag, if carry is set then go to step – 12, otherwise go to next step
9. Exchange the content of memory pointed by SI and the content of previous memory location.[for this exchange AL and memory pointed by SI and then exchange AL and memory pointed by SI – 1.

PROGRAM

Address	Label	Opcode	Mnemonics	Operand	Comments
1000		C7 C6 00 11	MOV	SI,1100	Set SI register as pointer for array
1004		8A 0C	MOV	CL,[SI]	Set CL as count for N-1repletion's
1006		FE C9	DEC	CL	Decrement CL
1008	@	C7 C6 00 11	MOV	SI,1100	Initialize pointer
100C		8A 2C	MOV	CH,[SI]	Set CH as count for N-1 comparisons
100E		FE CD	DEC	CH	Decrement CH

OBSERVATION:

ASCENDING ORDER :

Input		**Output**	
Address	**Data**	**Address**	**Data**
1100	(Number of elements)		
1101		1101	
1102		1102	
1103		1103	
1104		1104	
1105		1105	

1010		46	INC	SI	Increment SI
1011	%	8A 0A	MOV	AL,[SI]	Get an element of array in AL register
1013		46	INC	SI	Increment SI
1014		3A 0A	CMP	AL,[SI]	Compare with next element of array in memory
1016		72 00	JC	# (1018)	If AL is less than memory, then go to #
1008	#	FE CD	DEC	CH	Decrement count for comparisons
101A		75 F5	JNZ	% (1011)	Repeat comparison until CH count is zero
101C		FE C9	DEC	CL	Decrement the count for repeat ions
101E		75 E8	JNZ	@ (1008)	Repeat N-1 comparisons until CL count is zero
1020		F4	HLT	-	Stop

OBSERVATION:

DESCENDING ORDER :

Input		**Output**	
Address	**Data**	**Address**	**Data**
1100	(No. of elements)		
1101		1101	
1102		1102	
1103		1103	
1104		1104	
1105		1105	

PROGRAM

Address	Label	Opcode	Mnemonics	Operand	Comments
1000		C7 C6 00 11	MOV	SI,1100	Set SI register as pointer for array
1004		8A 0C	MOV	CL,[SI]	Set CL as count for N-1 repetitions
1006		FE C9	DEC	CL	Decrement CL
1008	@	C7 C6 00 11	MOV	SI,1100	Initialize pointer
100C		8A 2C	MOV	CH,[SI]	Set CH as count for nN-1 comparisons
100E		FE CD	DEC	CH	Decrement CH
1010		46	INC	SI	Increment SI
1011	%	8A 04	MOV	AL,[SI]	Get an element of array in AL register
1013		46	INC	SI	Increment SI
1014		3A 04	CMP	AL,[SI]	Compare with next element of array in memory
1016		73 05	JNC	# (101D)	If AL is not less than memory, then go to #
1018		86 04	XCHG	AL,[SI]	If AL is less than memory then exchange the content of memory pointed by SI
101A		86 44 FF	XCHG	AL,[SI-1]	If AL is less than memory then exchange the content of memory pointed by previous memory location
101D	#	FE CD	DEC CH	CH	Decrement count for comparisons
101F		75 0F	JNZ	% (1011)	Repeat comparison until CH count is zero
1021		FE C9	DEC	CL	Decrement the count for repetitions
1023		75 E3	JNZ	@ (1008)	Repeat N-1 comparisons until CL count is zero
1025		F4	HLT	-	Stop

RESULT :

Date :

Ex. No.:

LARGEST & SMALLEST NUMBER IN AN ARRAY USING 8086

AIM:

To find the largest and smallest number in a given array using 8086 microprocessor.

APPARATUS REQUIRED:

8086 kit

PROGRAM

LARGEST NUMBER IN A DATA ARRAY:

Address	**Label**	**Opcode**	**Mnemonics**	**Operand**	**Comments**
1000		C7 C60030	MOV	SI,2000	Get the address in SI
1004		8B 0C	MOV	CX,[SI]	Content of 2000 mov to CX
1006		C7 C0 00 00	MOV	AX, 0000	Clear AX
100A	BACK	46	INC	SI	Increment the address
100B		46	INC	SI	Increment the address
100C		3B 04	CMP	AX,[SI]	Compare contents
100E		73 02	JAE	GO	Check above or below
1010		8B 04	MOV	AX,[SI]	Content of 2000 mov to AX
1012	GO	E2 F6	DEC	CX	Jump
1013		75 F5	JNZ	BACK	Jump if CX is not zero
1015		89 06 51 30	MOV	[2051],AX	Store the result in 2051
1019		F4	HLT	-	Stop

OBSERVATION

Address	Input Data
2000 (No. of 16 bit numbers in the array)	
2001(No. of 16 bit numbers in the array)	
2002	
2003	
2004	
2005	
2006	
2007	
2008	
2009	
200A	
200B	
200C	
200D	

OUTPUT FOR LARGEST NUMBER

Address	Data
2051	
2052	

PROGRAM

SMALLEST NUMBER IN A DATA ARRAY:

Address	Label	Opcode	Mnemonics	Operand	Comments
1000		C7 C6 00 30	MOV	SI,2000	Get the address in SI
1004		8B 0C	MOV	CX,[SI]	Content of 2000 move to CX
1006		C7 C0 99 99	MOV	AX, 9999	Clear AX
100A	BACK	46	INC	SI	Increment the address
100B		46	INC	SI	Increment the address
100C		3B 04	CMP	AX,[SI]	Compare contents
100E		72 02	JB	GO	Check above or below
1010		8B 04	MOV	AX,[SI]	Content of [SI] moved to AX
1012	GO	49	DEC	CX	Decrement Count
1013		75 F5	JNZ	BACK	Jump if CX is not zero
1015		89 06 51 30	MOV	[2051],AX	Store the result in 2051
1019		F4	HLT	-	Stop

OBSERVATION

Address	Input Data
2000 (No. of 16 bit numbers in the array)	
2001(No. of 16 bit numbers in the array)	
2002	
2003	
2004	
2005	
2006	
2007	
2008	
2009	
200A	
200B	
200C	
200D	

OUTPUT FOR SMALLEST NUMBER

Address	Data
2051	
2052	

RESULT :

Date :

Ex.No.:

ARITHMETIC AND LOGICAL OPERATIONS USING MASM SOFTWARE

AIM:

To write ALP for Arithmetic and logic operations using MASAM.

SOFTWARE REQUIRED:

Pc with windows (95/98/XP/NT/2000)
MASM Software.

PROCEDURE :

1. Go to command prompt and type ‘edit’
2. In the edit window type the program.
3. Save the program as ‘add.asm’
4. Exit from edit window and in the command prompt following operations are performed:

```
D:/8086>masm add.asm (press enter)
D:/8086> link add.obj (press enter)
D:/8086> debug add.exe        (press enter)
- e 2000 01 02 08 05  (press enter)
- g = 1000       (press enter)
```

Program terminated correctly

PROGRAM:

16-BIT ADDITION

```
Code segment
Assume CS : code, DS: code
Org 1000h
Mov si, 2000h
Mov cl,00h
Mov ax,[si]
Mov bx,[si+2]
```

```
        Add ax,bx
```

Addition:

<u>Input:</u>

```
-e 2000
```

<u>Output:</u>

```
-e 2004
```

```
        Jnc L1
         Inc cl
L1:     Mov [si+4], ax
        Mov [si+6], cl
        Mov ah,4ch
        Int 21h
        Code ends
        End
```

16-BIT SUBTRACTION

```
        Code segment
        Assume CS : code, DS: code
        Org 1000h
        Mov si, 2000h
        Mov cl,00h
        Mov ax, [si]
        Mov bx,[si+2]
        Sub ax,bx
        Jnc L1
        Inc cl
        Not ax
        Add ax,0001h
L1:      Mov [si+4], ax
        Mov [si+6], cl
        Mov ah,4ch
        Int 21h
        Code ends
        End
```

Subtraction:

Input:

-e 2000

Output:

-e 2004

16-BIT MULTIPLICATION

```
Code segment
Assume CS : code, DS: code
Org 1000h
Mov si, 2000h
Mov ax,[si]
Mov bx,[si+2]
Mul bx
Mov [si+4], ax
Mov [si+6], dx
Mov  ah,4ch
Int 21h
Code ends
End
```

16-BIT DIVISION

```
Code segment
Assume CS : code, DS: code
Org 1000h
Mov si, 2000h
Mov ax,[si]
Mov bx,[si+2]
Div bx
Mov [si+4], ax
Mov [si+6], dx
Mov  ah,4ch
Int 21h
Code ends
```

End

Multiplication:

Input :

-e 2000

Output:

-e 2004

Division:

Input :

-e 2000

Output:

-e 2004

-e 2006

Logical OR:

Output:

-e 2000

Logical AND:

Output:

-e 2000

LOGICAL AND:

```
Code segment
Assume CS : code, DS: code
Org 1000h
Mov si, 2000h
Mov al, 07h
Mov bl, 02h
And al, bl
Mov [si],al
Int 21h
Code ends
End
```

LOGICAL OR:

```
Code segment
Assume CS : code, DS: code
Org 1000h
Mov si, 2000h
Mov al, 07h
Mov bl, 02h
Or al, bl
Mov [si],al
Int 21h
Code ends
End
```

LOGICAL XOR:

```
Code segment
Assume CS : code, DS: code
Org 1000h
Mov si, 2000h
Mov al, 07h
Mov bl, 02h
Xor al, bl
Mov [si],al
Int 21h
Code ends
End
```

Logical XOR:

Output:

-e 2000

Logical NOT:

Output:

-e 2000

NOT OPERATION:

```
Code segment
Assume CS : code, DS: code
Org 1000h
Mov si, 2000h
Mov Al, 07h
Not  Al
Mov [si],Al
Int 21h
Code ends
End
```

RESULT:

Date :

Ex.No.

BIOS / DOS CALL - STRING MANIPULATION

AIM:

To perform string manipulation and BIOS / DOS call using MASM software.

APPARATUS REQUIRED:

PC loaded with MASM software.

PROCEDURE:

1. Go to command prompt and type ‘edit’
2. In the edit window type the program.
3. Save the program as ‘str.asm’
4. Exit from edit window and in the command prompt following operations are performed:

```
D:/8086> masm str.asm (press enter 3 times)
D:/8086> link str.obj (press enter 3 times)
D:/8086> debug str.exe          (press enter)
- f 2000 20ff  45 (any data)    (press enter)
- g = 1000       (press enter)
```

Program terminated correctly

PROGRAM :

```
Code segment
        Assume CS : code, DS: code
        Org 1000h
        Mov si, 2000h
        Mov cx, 00ffh
        Mov di, 3000h
Move: movsb
        Loop Move
        Mov ah, 4ch
        Int 21h
        Code ends
        End
```

<u>Output:</u>

```
-e 3000 to 30FF
```

RESULT :

Date :

Ex.No.:

BIOS / DOS CALL – SEARCH AN ELEMENT

AIM :

To search an element in an array and BIOS / DOS call using MASM software.

APPARATUS REQUIRED:

PC loaded with MASM software.

PROCEDURE :

1. Go to command prompt and type 'edit'
2. In the edit window type the program.
3. Save the program as 'search.asm'
4. Exit from edit window and in the command prompt following operations are performed:

```
D:/8086>masm search.asm (press enter)
D:/8086> link search.obj       (press enter)
D:/8086> debug search.exe      (press enter)
- e 2000 01 02 08 05 08        (press enter)
- g = 1000      (press enter)
```

Program terminated correctly

PROGRAM:

```
Code segment
Assume CS:code, DS: code
Org 1000h
Mov di, 2100h
Mov bx, 2000h
Mov dh, 05h (No. of elements in the array)
Mov ch, 08h (Data which needs to be searched)
Mov cl,00h
```

Output:

```
-e 2100
```

```
L2:     mov ax,[bx]
        Cmp al, ch
        Jnz L1
        Inc cl
L1:     inc bx
        Dec dh
        Jnz L2
        Mov [di], cl
        Mov ah,4ch
        Int 21h
        Code ends
        End
```

RESULT :

Date :
Ex.No :7

TRAFFIC LIGHT CONTROLLER USING 8086

AIM:

The objective of this experiment is to simulate a traffic lights system.

APPARATUS REQUIRED:

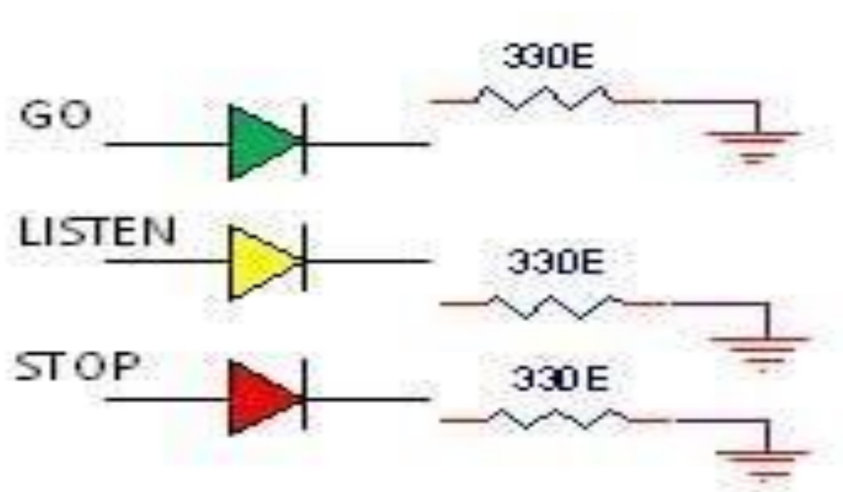
8086 kit and Interfacing card

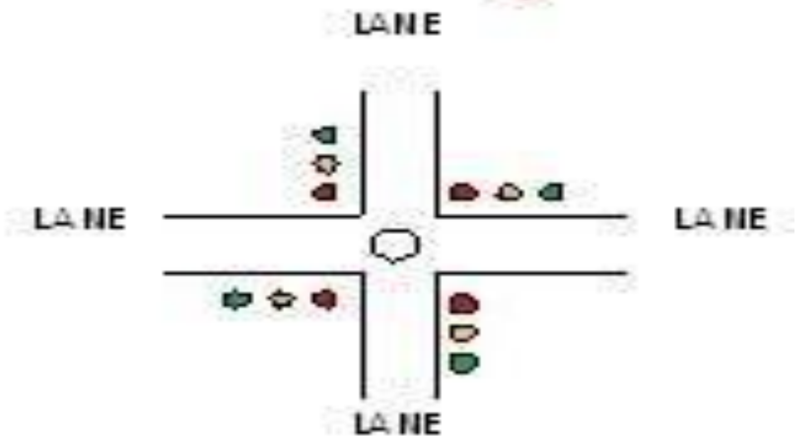
THEORY:

Traffic light control using microcontroller 8051 can be done easily with parallel ports. The port pins can be connected to each light, LED, or group of LEDs through a proper driver circuit. The data in parallel ports can be changed using the program, for turning on and off the lights. The port pins of Port A and Port B are used. The least significant three bits of Port A are used for the west direction. The Port A pins 3,4,5 are used for are used for the north direction. Similarly, the least significant bits of Port B are used for the lights in east direction and pins 3,4,5 are used for the south direction.

PROGRAM

Address	Label	Opcode	Mnemonics
1000	START:	C7 C6 00 12	MOV SI, 1200H
1004		C7 C1 04 00	MOV CX, 0004H
1008		8A 0A	MOV AL,[SI]
100A		C7 C2 83 00	MOV DX, 0083H [Control Word]
100E		EE	OUT DX, AL
100F		46	INC SI
1010	NEXT:	8A 04	MOV AL,[SI]
1012		C7 C2 80 00	MOV DX, 0080H [PA]
1016		EE	OUT DX,AL
1017		46	INC SI
1018		8A 04	MOV AL,[SI]
101A		C7 C2 81 00	MOV DX, 0081H [PB]





Make high to - LED On

Make low to – LED Off

101E		EE	OUT DX,AL
101F		E8 DE 00	CALL DELAY 1
1022		46	INC SI
1023		8A 04	MOV AL,[SI]
1025		C7 C2 80 00	MOV DX,0080H
1029		EE	OUT DX,AL
102A		46	INC SI
102B		8A 04	MOV AL,[SI]
102D		C7 C2 81 00	MOV DX,0081H
1031		EE	OUT DX,AL
1032		E8 1B 01	CALL DELAY 2
1035		46	INC SI
1036		49	DEC CX
1037		75 D7	JNZ NEXT
1039		E9 C4 FF	JMP START
1100	DELAY 1:	C7 C2 14 00	MOV DX,0014H
1104	%	C7 C3 FF FF	MOV BX,FFFF
1108	#	90	NOP
1109		90	NOP
111A		90	NOP
111B		90	NOP
110C		4B	DEC BX
110D		75 F9	JNZ #
110F		4A	DEC DX
1110		75 F2	JNZ %
1112		C3	RET
1150	DELAY 2:	C7 C2 05 00	MOV DX,0005H
1154	@	C7 C3 FF FF	MOV BX,FFFF
1158	$	90	NOP

Data for port pins for traffic light control

Sequence	**SG PB.5**	**SY PB.4**	**SR PB.3**	**EG PB.2**	**EY PB.1**	**ER PB.0**	**NG PA.5**	**NY PA.4**	**NR PA.3**	**WG PA.2**	**WY PA.1**	**WR PA.0**	**PA Data**	**PB Data**
Sequence 1	0	0	1	0	0	1	0	0	1	1	0	0	0CH	09H
	0	0	1	0	0	1	0	1	0	1	0	0	14H	09H
Sequence 2	0	0	1	0	0	1	1	0	0	0	0	1	21H	09H
	0	0	1	0	1	0	1	0	0	0	0	1	21H	0AH
Sequence 3	0	0	1	1	0	0	0	0	1	0	0	1	09H	0CH
	0	1	0	1	0	0	0	0	1	0	0	1	09H	14H
Sequence 4	1	0	0	0	0	0	0	1	1	0	0	1	09H	21H
	1	0	0	0	0	1	0	0	1	0	1	0	0AH	21H

Software: Control Word: For initialization of 8255.

Ports/control Register	**Address lines** A7 A6 A5 A4 A3 A2 A1 A0	**Address**
Port A	**1 0 0 0 0 0 0 0**	**80H**
Port B	**1 0 0 0 0 0 0 1**	**81H**
Port C	**1 0 0 0 0 0 1 0**	**82H**
Control Register	**1 0 0 0 0 0 1 1**	**83H**

BSR/IO	**MODE A**	**P_A**	**PC_H**	**MODE B**	**P_B**	**PC_L**
1	0 0	0	X	0	0	X

Control Word

1159		90	NOP
115A		90	NOP
115B		90	NOP
115C		75 FA	JNZ $
115E		4A	DEC DX
115F		75 F3	JNZ @
1161		C3	RET

LOOK UP TABLE

Address	**Opcode**
1200	80H
1201	0CH,09H,14H,09H (WEST WAY)
1205	21H,09H,21H,0AH (NORTH WAY)
1209	09H,0CH,09H,14H (EAST WAY)
120D	09H,21H,0AH,21H (SOUTH WAY)

RESULT:

Date :
Ex.No. : 8

STEPPER MOTOR INTERFACING WITH 8086

AIM:

To interface a stepper motor with 8086 microprocessor, operate it in clockwise and anticlockwise direction and control its speed of operation.

THEORY:

A motor in which the rotor is able to assume only discrete stationary angular position is a stepper motor. The rotary motion occurs in a step-wise manner from one equilibrium position to the next. Stepper Motors are used very wisely in position control systems like printers, disk drives, process control machine tools, etc.

The basic two-phase stepper motor consists of two pairs of stator poles. Each of the four poles has its own winding. The excitation of any one winding generates a North Pole. A South Pole gets induced at the diametrically opposite side. The rotor magnetic system has two end faces. It is a permanent magnet with one face as South Pole and the other as North Pole.

The Stepper Motor windings A1, A2, B1, B2 are cyclically excited with a DC current to run the motor in clockwise direction. By reversing the phase sequence as A1, B2, A2, B1, anticlockwise stepping can be obtained.

2-PHASE SWITCHING SCHEME:

In this scheme, any two adjacent stator windings are energized. The switching scheme is shown in the table. This scheme produces more torque.

ADDRESS DECODING LOGIC:

The 74138 chip is used for generating the address decoding logic to generate the device select pulses; CS1 & CS2 for selecting the IC 74175.The 74175 latches the data bus to the stepper motor driving circuitry.

Switching scheme of stepper motor :

Anticlockwise						**Clockwise**					
STEP	**A1**	**A2**	**B1**	**B2**	**DATA**	**STEP**	**A1**	**A2**	**B1**	**B2**	**DATA**
1	1	0	0	1	09h	1	1	0	1	0	0Ah
2	0	1	0	1	05h	2	0	1	1	0	06h
3	0	1	1	0	06h	3	0	1	0	1	05h
4	1	0	1	0	0Ah	4	1	0	0	1	09h

Stepper Motor requires logic signals of relatively high power. Therefore, the interface circuitry that generates the driving pulses uses silicon Darlington pair transistors. The inputs for the interface circuit are TTL pulses generated under software control using the Microcontroller Kit. The TTL level of pulse sequence from the data bus is translated to high voltage output pulses using a buffer 7407 with open collector.

PROCEDURE:

Enter the above program starting from location 1018.and execute the same. The stepper motor rotates. Varying the count at R4 and R5 can vary the speed. Entering the data in the look-up TABLE in the reverse order can vary direction of rotation.

PROGRAM:

Address	Label	Mnemonics	Operand	Comments
1000	START	MOV	DI,1018	Load the start address of switching scheme data TABLE into DI register
1004		MOV	CL,04	Load the count in CL
1007	#	MOV	AL,[DI]	Load the number in TABLE into AL
1009		OUT	C0,AL	Send the value in A to stepper Motor port address
100B		MOV	DX,1010	Delay loop to cause a specific amount of time delay before next data item is sent to the Motor
100F	@	DEC	DX	Decrement the Dx value for delay
1010		JNZ	@ (100F)	Go to 100F till DX=0

Stepper motor Stepping Sequence Look up table :

Address	Data (Clockwise Rotation)	Data (Anti- Clockwise Rotation)
1018	09	0A
1019	05	06
101A	06	05
101B	0A	09

1012		INC	DI	Increment the DI value to point out to the next value
1013		LOOP	# (1007)	Repeat the process
1015		JMP	1000	Jump to starting address and continue the processs

RESULT:

Date :

Ex.No :

INTERFACING 8279 WITH 8086

AIM:

To display Rolling message in display using 8279 programmable keyboard/display controller by interfacing it with 8086.

APPARATUS REQUIRED:

S.No	Apparatus	Quantity
1	Microprocessor kit-8086	1
2	Keyboard display Interface-8279	1
3	Connecting cable	-

DESCRIPTION:

The INTEL 8279 is specially developed for interfacing keyboard and display devices to 8085/8086/8088 microprocessor based system.

The important features of 8279 are,

- Simultaneous keyboard and display operations.
- Scanned keyboard mode.
- Scanned sensor mode.
- 8-character keyboard FIFO.
- 16-character display.
- Right or left entry 1 6-byte display RAM.
- Programmable scan timing.

DISPLAY SECTION:

- The display section has eight output lines divided into two groups A0-A3 and B0-B3.
- The output lines can be used either as a single group of eight lines or as two groups of our lines, in conjunction with the scan lines for a multiplexed display.
- The output lines are connected to the anodes through driver transistor in case of common cathode 7-segment LEDs.
- The cathodes are connected to scan lines through driver transistors.

Display mode setup command: [10]

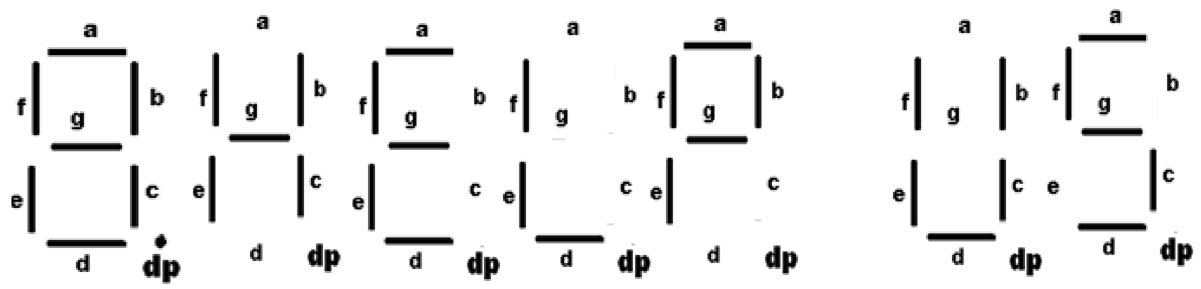
0	0	0	D	D	K	K	K
0	0	0	1	0	0	0	0

Clear Display Command: [CC]

1	1	0	CD2	CD1	CD0	CF	CA
1	1	0	0	1	1	0	0

Display RAM Command: [90]

1	0	0	AI	A	A	A	A
1	0	0	1	0	0	0	0



Seven segment code for the message :

Character	d	c	b	a	Dp	g	f	e	Seven segment code
H	1	0	0	1	1	0	0	0	98
E	0	1	1	0	1	0	0	0	68
L	0	1	1	1	1	1	0	0	7C
P	1	1	0	0	1	0	0	0	C8
U	0	0	0	1	1	1	0	0	1C
S	0	0	1	0	1	0	0	1	29

- The display can be blanked by BD (low) line.
- The display section consists of 16 x 8 display RAM. The CPU can read from or write into any location of the display RAM.

In common anode type seven segment display, 0 is used for a segment to glow and 1 for a segment to remain in off condition.

ALGORITHM :

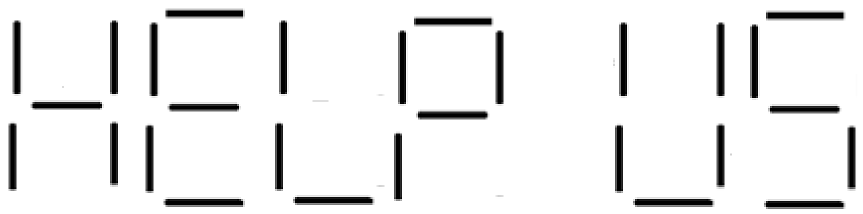
1. Store the look up table which contains the common anode seven segment code for the message ‘HELP US’ from memory location 1200.
2. Load the number of characters to be displayed in CX reg.
3. Move display mode set up command to acc. And then load it in command reg.
4. Move clear command to acc. and then load it in command reg.
5. Move display RAM command to acc. And then load it in command reg.
6. Then move common anode seven segment codes one by one for the character to be displayed to accumulator from memory and then load it in data reg.
7. Call delay subroutine between each code.
8. Repeat step 2 to 7 to get continuous display of message ‘HELP US’.

PROGRAM -1 :

To Display ‘A’ in the first digit:

Address	Label	Opcode	Mnemonics
1000	START	C6C000	MOV AL,00
1003		E6C2	OUT C2,AL
1005		C6C0CC	MOV AX,00CC
1008		E6C2	OUT C2,AL
100A		C6C090	MOV AL,90
100D		E6C0	OUT C2,AL
100F		C6C088	MOV AL,88(Data)
1012		E6C0	OUT C0,AL
1014		C6C0FF	MOV AX,00FF
1017		C7C10500	MOV CX,0005
101B	NEXT	E6C0	OUT C0,AL
101D		E2FC	LOOP NEXT
101F		F4	HLT

OUTPUT:



PROGRAM – 2 :

ROLLING DISPLAY:

Address	Label	Opcode	Mnemonics
1000	START	C7 C6 00 12	MOV SI,1200
1004		C7 C1 0F 00	MOV CX,000F
1008		C6 C0 10	MOV AL,10
100B		E6 C2	OUT C2,AL
100D		C6 C0 CC	MOV AL,CC
1010		E6 C2	OUT C2,AL
1012		C6 C0 90	MOV AL,90
1015		E6 C2	OUT C2,AL
1017	NEXT	8A 04	MOV AL,[SI]
1019		E6 C0	OUT C0,AL
101B		E8 E2 04	CALL DELAY
101E		46	INC SI
101F		E2 F6	LOOP NEXT
1021		E9 DC FF	JMP START
1500	DELAY	C7 C2 FF A0	MOV DX,0A0FF
1504	LOOP1	4A	DEC DX
1505		75 FD	JNZ LOOP1
1507		C3	RET

LOOK UP TABLE:

1200	FF	FF	FF	FF
1204	FF	FF	FF	FF
1208	98	68	7C	C8
120C	FF	1C	29	FF

RESULT:

Mode Instruction Word Format: (4Eh)

S2	S1	EP	PEN	L2	L1	B2	B1

=1Even =1 Parity Char. Length Baud rate factor
S2 S1 Stop bits parity enable 0 0 5 bits 0 0 Sync. Mode
0 0 Invalid =0 Odd 0 1 6 bits 0 1 1xAsync.
0 1 1 parity 1 0 7 bits 1 0 16xAsync.
1 0 1½ 1 1 8 bits 1 1 64xAsync.
1 1 2

E	1	0	0	1	1	1	0

Command Instruction Word Format: (37h)

EH	IR	RTS	ER	SBRK	RxE	DTR	TxEN
0	0	1	1	0	1	1	1

EH: Enter Hunt mode (No effect in Async.)
IR: Internal Reset
RTS : Request To Send
ER : Error Reset
SBRK: Send Break Character
RxE : Receive Enable
DTR : Data Terminal Ready
TxEN : Transmit Enable

8254 Timer's registers address

Register	Address
Control Register	CE
Data Register	C8

USART 8251's registers address

Register	Address
Control Register	C2
Data Register	C0

8254 Timer's registers address **USART 8251's registers address**

Register	Address	Register	Address
Control Register	CE	Control Register	C2
Data Register	C8	Data Register	C0

Date :
Ex.No.:

INTERFACING 8251 WITH 8086

AIM:

To initialize the USART (Universal Synchronous Asynchronous Receiver Transmitter) 8251 and check the serial data transmission and reception of character with 8086.

APPARATUS REQUIRED:

S.No	Apparatus	Quantity
1	Microprocessor kit 8086	1
2	Interface kit 8251	1
3	Connecting cable	-

DESCRIPTION:

USART-INTEL 8251A:

1. The 8251A is a programmable serial communication interface chip designed for synchronous and asynchronous serial data communication.
2. It supports the serial transmission of data.
3. It is packed in a 28 pin DIP.

ALGORITHM :

1. Clock for serial transmission and reception is generated using Intel’s Programmable Interval Timer 8254, which is made to function in mode 3 (Square wave generator). For that first the control word for 8254 (36H) is loaded in its control register. Then count value (0AH) is loaded in counter 0’s count register.
2. Mode instruction word for 8251 (4EH) is loaded in its control register.
3. Then command instruction word (37H) is loaded in control register.
4. Data to be transmitted is loaded in 8251’s data register.
5. In program 2 get the data in data register to check whether the data is received at the receiver side properly.
6. Store the received data in accumulator in memory Location 1250.

OUTPUT:

Transmitted Data	**Received Data**	
	Address	
	Data	

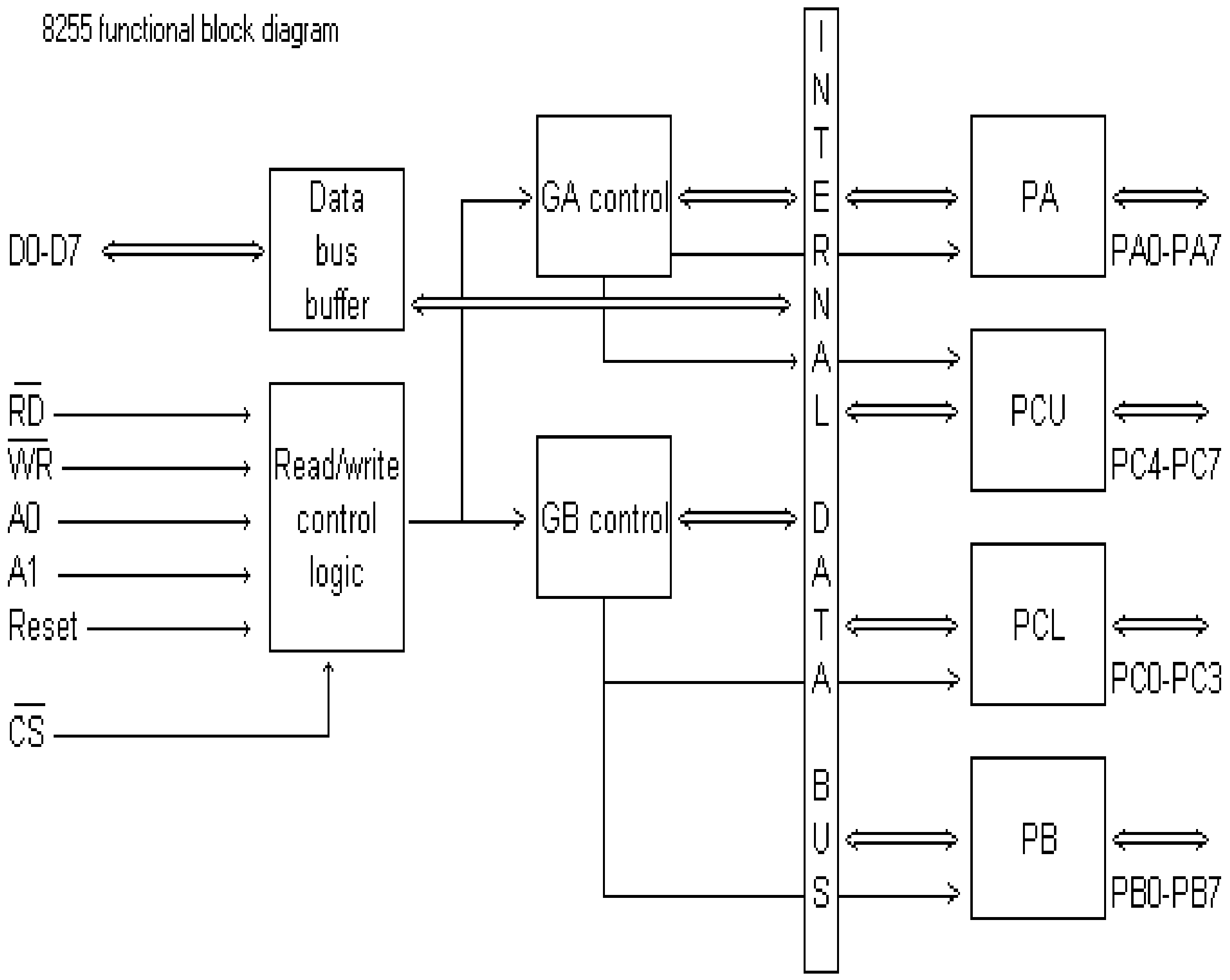
PROGRAM (TX):

Address	Opcode	Mnemonic	Operand
1000	B0 36	MOV	AL, 36
1002	E6 CE	OUT	CE,AL
1004	B0 10	MOV	AL,OA
1006	E6 C8	OUT	C8,AL
1008	B0 00	MOV	AL,00
100A	E6 C8	OUT	C8,AL
100C	B0 4E	MOV	AL,4E
100E	E6 C2	OUT	C2,AL
1010	B0 37	MOV	AL,37
1012	E6 C2	OUT	C2,AL
1014	C6 C0 (DATA)	MOV	AL, DATA
1016	E6 C0	OUT	C0,AL
1018	F4	HLT	-

PROGRAM (RX):

Address	Opcode	Mnemonic	Operand
1200	E4 C0	IN	AL,C0
1202	BB 50 12	MOV	BX,1250
1205	88 07	MOV	[BX],AL
1207	F4	HLT	-

RESULT:


8255 functional block diagram
D0-D7
Data bus buffer
RD
WR
A0
A1
Reset
CS
Read/write control logic
GA control
GB control
INTERNAL DATA BUS
PA
PA0-PA7
PCU
PC4-PC7
PCL
PC0-PC3
PB
PB0-PB7

Date :

Ex.No. :

INTERFACING 8255 WITH 8086

AIM:

To interface programmable peripheral interface 8255 with 8085 and study its characteristics in mode 0.

APPARATUS REQUIRED:

8086 µp kit, 8255Interface board and Interface card.

THEORY:

The 8255 Programmable Peripheral Interface (PPI) is a very popular and versatile input / output chip that are easily configured to function in several different configurations. This chip allows to do both digital input and output (DIO) with PC. The functional configuration of the 8255A is programmed by the systems software so that normally no external logic is necessary to interface peripheral devices or structures. There are 3-stable bi-directional 8-bit buffer is used to interface the 8255A to the systems data bus. Data is transmitted or received by the buffer upon execution of input or output instructions by the CPU. Read/Write and Control Logic blockmanages all the Internal and External transfers of both Data and Control or Status words.

I/O MODES:

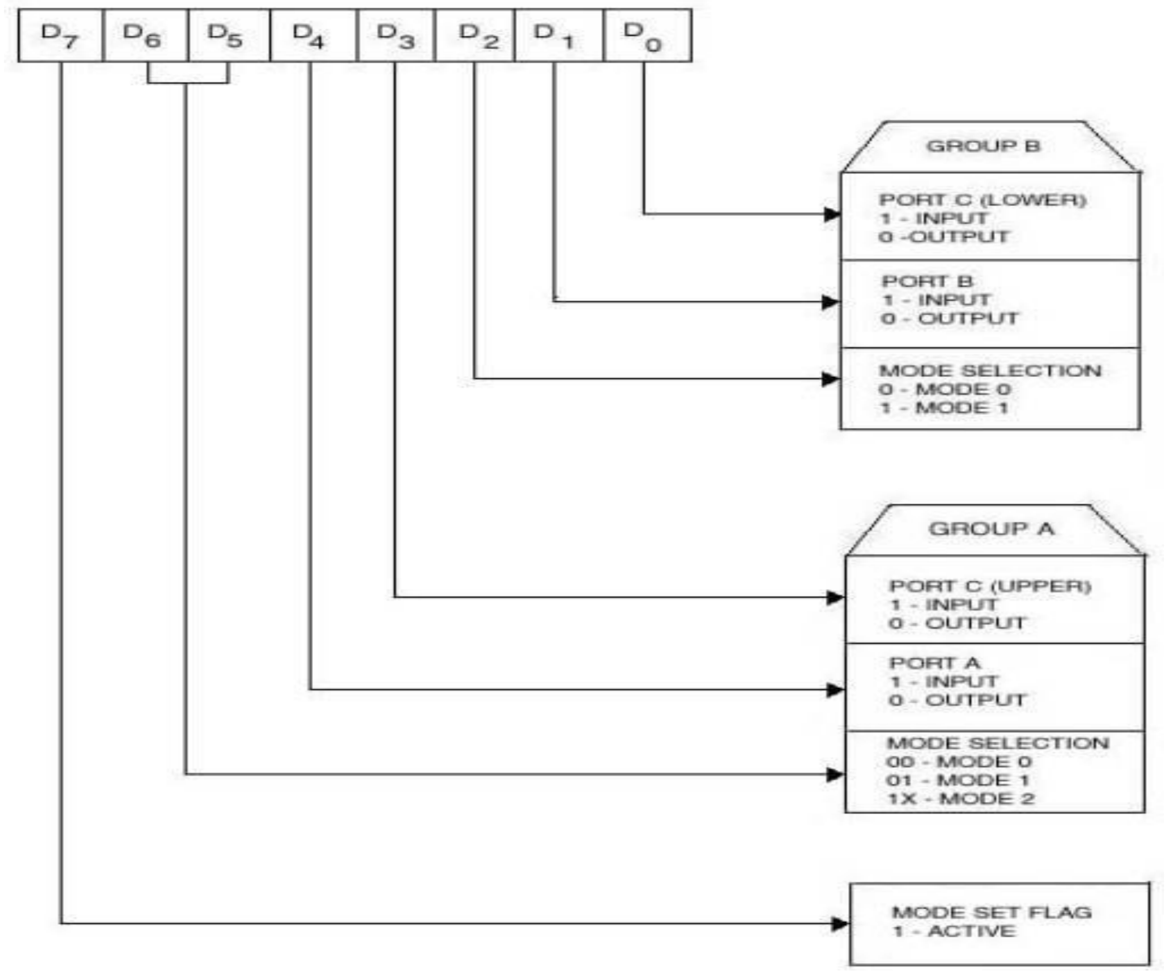
MODE 0 – SIMPLE I/O MODE:

This mode provides simple I/O operations for each of the three ports and is suitable for synchronous data transfer. In this mode all the ports can be configured either as input or output port.

Let us initialize port A as input port and port B as output port

I/O MODES:

Control Word:



PROGRAM :

To initialize port A as an input port in Mode -0 and to input the data set by the SPDT switches through port A, output the same to LEDs connected to Port B and store the data at RAM location 1100.

Address	Opcodes	Mnemonics	Operand	Comments
1000	B0 90	MOV	AL, 90	Initialize port A as Input port in mode 0.
1002	E6 C6	OUT	C6,AL	Send Mode Control word
1004	E4 C0	IN	AL,C0	Read port A
1006	E6 C2	OUT	C2,AL	Send output to port B
1008	BE 00 11	MOV	SI, 1100	Initialize SI register for output address
100B	88 04	MOV	[SI],AL	Store output at 1100
100D	F4	HLT	-	Stop the program

OPERATION OF 8255 IN MODE -1 :

In this mode, the ports are divided into two groups, A and B, each of which consists of an 8 bit data port and 4 bit control lines, which are used for strobed I/O data transfer. The ports can be configured either as input or as output.

EXAMPLE FOR PORT – B AS OUTPUT PORT IN MODE - 1 :

With this configuration, the port C lines PC1 and PC2 acts as OBF and ACK signals respectively. The given program initialize port B as output port. The control word for this is 84. Then it writes data 45 into port B. This write operation generates WR signal to 8255. Hence at the trailing edge of WR, the OBF signal goes low, which can be seen by the corresponding LED going off. Now, if INT 0 switch is pressed, a ACK goes low and corresponding LED will glow.

OBSERVATION

MODE -0

INPUT – PORT – A SPDT switch position:

OUTPUT :

PORT B – LED Condition:

Memory Address: 1100

Data:

MODE -1

OUTPUT:

PORT B – LED Condition:

MODE - 2

INPUT – PORT – A SPDT switch position:

OUTPUT:

PORT B – LED Condition:

PROGRAM :

Address	Opcode	Mnemonics	Operand	Comments
1000	B0 90	MOV	AL, 84	Initialize port A as Input port in mode 0.
1002	E6 C6	OUT	C6,AL	Send Mode Control word
1004	B0 45	MOV	AL, 45	Load data in AL
1006	E6 C2	OUT	C2,AL	Send output to port B
100D	F4	HLT	-	Stop the program

OPERATION OF 8255 IN MODE -2 :

Mode 2 provides the facility of data transfer with an I/O in both directions using strobed I/O mode. This mode is available only for Group A, that is, port A, A part from the 8-bit bidirectional port, there are 5 control lines (PC3 – PC7), which are used for handshaking and interrupt request. Data can be latched in both directions.

EXAMPLE FOR PORT – A AS INPUT PORT IN MODE - 2 :

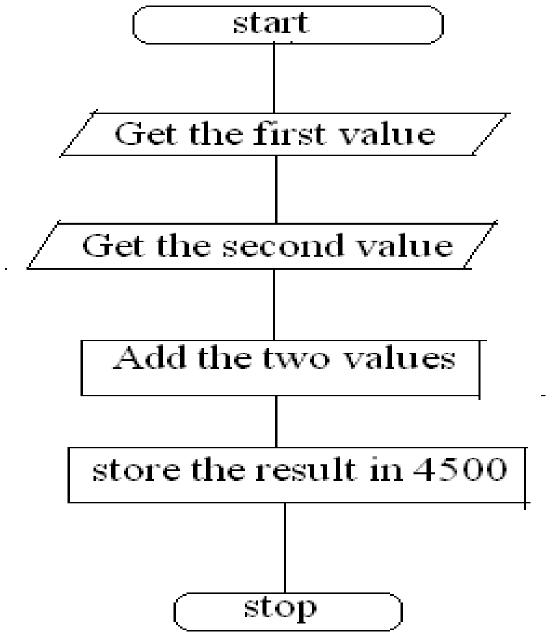
In mode -2, port A acts as a bidirectional I/O. In this mode, lines PC4 and PC5 of port C as STB and IBF signals respectively. Configuring port A as input port and pins PC6 and PC7 acts as ACK and OBF respectively configuring port A as output port.

PROGRAM :

Address	Opcodes	Mnemonics	Operand	Comments
1000	B0 90	MOV	AL, C0	Initialize port A as Input port in mode 0.
1002	E6 C6	OUT	C6,AL	Send Mode Control word
1004	F4	HLT	-	Stop the program

RESULT:

FLOW CHART: ADDTION



OUTPUT:

	Address	Data
Input	-	(Data 1) (Data 2)
Output	4150	(sum)

Date :
Ex.No. :

ARITHEMETIC OPERATIONS USING 8051

AIM:

To write an assembly language program to perform
1. Addition
2. Subtraction
3. Multiplication
4. Division

Using 8051 microcontroller.

APPARATUS REQUIRED:

Microcontroller kit-8051

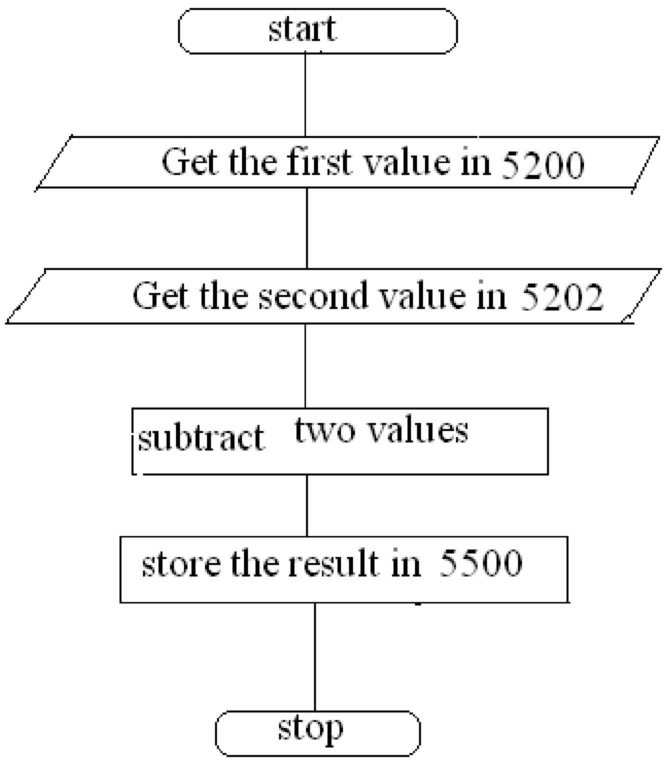
ALGORITHM: ADDITION

1. Start the program.
2. Get the 1st and 2nd value.
3. Add the 2 values.
4. Store the result in a memory location.
5. Stop the program.

PROGRAM:

Address	Label	Opcode	Mnemonics	Operand	Comments
4100		E4	CLR	A	Clear A register
4101		04	MOV	A,#data1	Get immediate data Of register A
4103		03	ADD	A,#data2	Add with data 1
4105		90,41,50	MOV	DPTR,#4150	Initialize data pointer
4108		F0	MOVX	@DPTR,A	Store the results in DPTR
4109	Here	80,FE	SJMP	Here (4109)	Loop is terminated

FLOW CHART: SUBTRACTION



OUTPUT

	Address	**Data**
Input	-	(data 1) (data 2)
Outpu T	5500H	(difference)

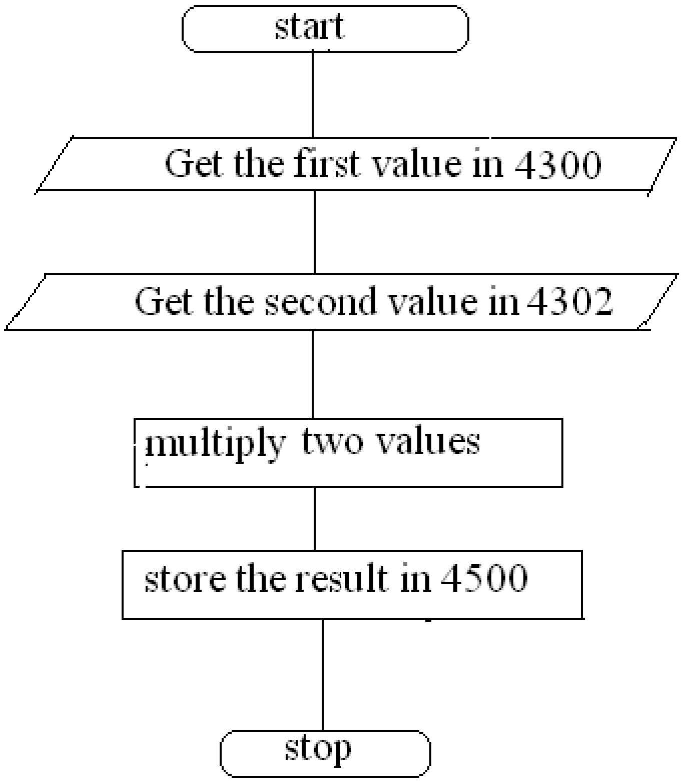
ALGORITHM: SUBTRACTION

1. Start the program
2. Get the 1st and second value.
3. Subtract the two values.
4. Store the results in the memory location.
5. Stop the program

PROGRAM:

Address	Label	Opcode	Mnemonics	Operand	Comments
5200		E4	CLR	A	Clear A register
5201		74 06	MOV	A,#data1	Get minuend value in A register
5203		94 05	SUBB	A,#data2	Subtract data 2 from minuend
5205		90 55 00	MOV	DPTR,#5500	Initialize data pointer
5208		F0	MOVX	@DPTR,A	Store different pointer in data
5209	Here	80,FE	SJMP	Here (5209)	Loop is terminated

FLOW CHART: MULTIPLICATION



OUTPUT:

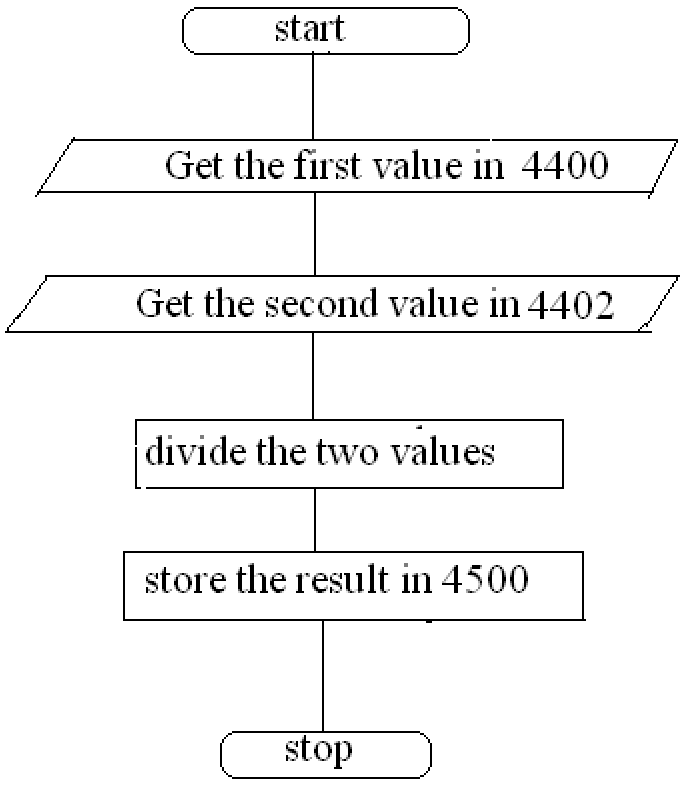
	Address	Data
Input	-	(data1) (data2)
Output	4500H 4501H	(Lower Byte) (Higher Byte)

ALGORITHM: MULTIPLICATION

1. Start the program
2. Get the first and second value
3. Divide the two values
4. Store the result in data pointer
5. Stop the program

Address	Label	Opcode	Mnemonics	Operand	Comments
4300		74 02	MOV	A,#data1	Immediate data1 is moved to A
4302		75 F0 03	MOV	B,#data2	Data2 is moved to B
4305		A4	MUL	AB	Multiply A and B
4306		90 45 00	MOV	DPTR,#4500	Move content from 4500 to DPTR
4309		F0	MOVX	@DPTR,A	Move accumulator content to DPTR
430A		A3	INC	DPTR	Increment data pointer value
430B		E5 F0	MOV	A,B	Move contents of B to A
430D		F0	MOVX	@DPTR,A	Move A to DPTR
430E	Here	80 F5	SJMP	Here (430E)	Loop is terminated

FLOW CHART: DIVISION



OUTPUT:

	Address	Data
Input	-	(data 1) (data 2)
Output	4500H 4501H	(Quotient) (Remainder)

ALGORITHM: DIVISION

1. Start the program
2. Get the first and second value
3. Divide the two values
4. Store the result in data pointer
5. Stop the program

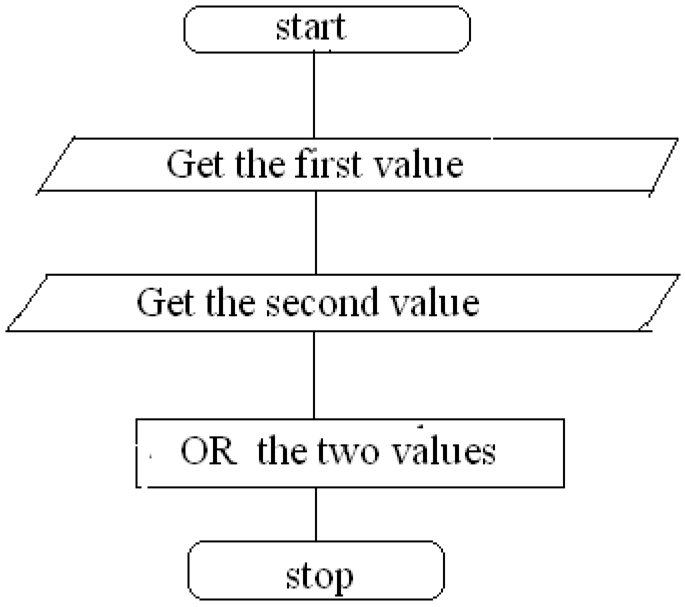
PROGRAM:

Address	Label	Opcode	Mnemonics	Operand	Comments
4400		74 04	MOV	A,#data1	Data1 is moved to A
4402		75 F0 02	MOV	B,#data2	Data2 is moved to B
4405		84	DIV	AB	Divide A and B
4406		90 45 00	MOV	DPTR,#4500	Move content from 4500 to DPTR
4409		F0	MOVX	@DPTR,A	Move A to DPTR
440A		A3	INC	DPTR	Increment DPTR value
440B		E5 F0	MOV	A,B	Move contents of B to A
440D		F0	MOVX	@DPTR,A	Store the result in DPTR
440E	Here	80 FE	SJMP	Here (440E)	Loop is terminated

Particulars	Max. marks	Marks awarded
Algorithm	20	
Program Coding	40	
Execution	20	
Output	10	
Viva Voce	10	

RESULT:

FLOW CHART: LOGICAL OR



OUTPUT:

	Address	Data
Input	-	(data) (data)
Output	4150H	(logical OR)

Date :
Ex.No. :

LOGICAL AND & OR OPERATIONS USING 8051

AIM:

To write an assembly language program for logical operations using 8051 microprocessor.

APPARATUS REQUIRED:

Microcontroller kit-8051

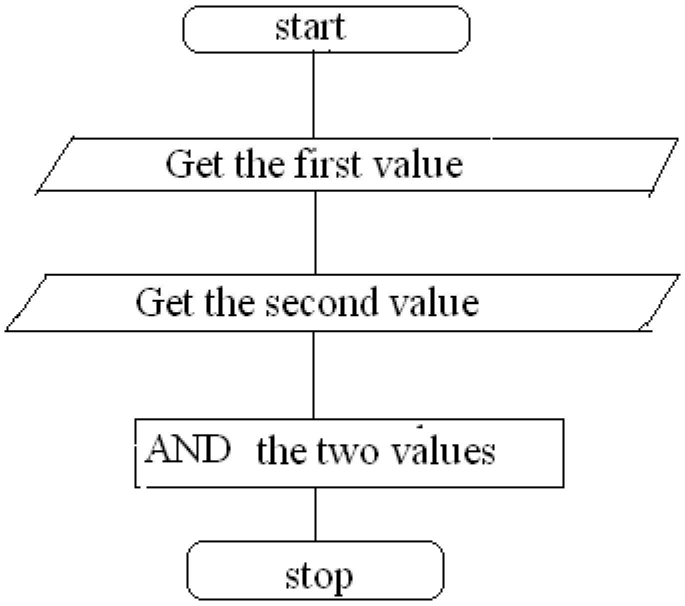
ALGORITHM:

1. Start the program
2. Get the 1st data.
3. Perform OR of 1st and the 2 nd data.
4. Store the results in the memory location.
5. Stop the program

PROGRAM:
LOGICAL OR OPERATION

Address	Label	Opcode	Mnemonics	Operand	Comments
5000		E4	CLR	A	Clear A register
5001		74	MOV	A, #data1	Move data1 to A
5003		54	ORL	A,#data2	Perform OR with data2
5005		90	MOV	DPTR, #4150	Initialize DPTR
5008		F0	MOVX	@DPTR,A	Store result in DPTR
5009	Here	80	SJMP	Here(5009)	Loop is terminated

FLOW CHART: LOGICAL AND OPERATION



OUTPUT:

	Address	Data
Input	-	Data 1: Data 2:
Output	4150H	

PROGRAM:
LOGICAL AND OPERATION

Address	Label	Opcode	Mnemonics	Operand	Comments
4100		74	MOV	A, #data1	Move data1 to A register
4102		74	MOV	A, #data2	Move data2 to B register
4105		72 AB	ANL	A,B	ANL A with B
4107		F0	MOV	DPTR,#4150	Move the address 4150 to DPTR
410A		F0	MOVX	@DPTR,A	Store result in DPTR address
410B	Here	80 FE	SJMP	Here (410B)	Stop the program

RESULT: