Mahatma Gandhi Mission

Jawaharlal Nehru Engineering College

Aurangabad, Maharashtra

Affiliated to Dr. B. A. Technological University, Lonere

NAAC 'A' Grade, ISO 9001:2015, 14001:2015 Certified, AICTE Approved.

Second Year B. Tech

Department of Information Technology

Lab Book

BTITC 401 : MICROPROCESSOR & MICROCONTROLLERS

Name: ____

Class:_________ **Roll No:** ________ **Year:** ___________

Exam No.:___________________________

Mahatma Gandhi Mission

Jawaharlal Nehru Engineering College

Aurangabad, Maharashtra

Affiliated to Dr. B. A. Technological University, Lonere

NAAC 'A' Grade, ISO 9001:2015, 14001:2015 Certified, AICTE Approved.

Second Year B. Tech

Department of Information Technology

Lab Book

BTITC 401 : MICROPROCESSOR & MICROCONTROLLERS

Prepared by
Prof. Vijaya Ahire
Assistant Professor
Lab Incharge

Reviewed by
Dr. S. C. Tamane
Associate Professor
Head of Department

Approved by
Dr. H. H. Shinde
Principal

Vision of Information Technology Department:

To develop expertise of budding technocrats by imparting technical knowledge and human value based education.

Mission of Information Technology Department:

A. Equipping the students with technical skills, soft skills and professional attitude.
B. Providing the state of art facilities to the students to excel as competent professionals, entrepreneurs and researchers.

Programme Educational Objectives:

PEO1. The graduates will utilize their **expertise** in IT industry and solve industry technological problems.
PEO2. Graduates should excel in **engineering positions** in industry and other organizations that emphasize design & implementation of IT applications.
PEO3. Graduates will be **innovators & professionals** in technology development, deployment & system implementation.
PEO4. Graduates will be pioneers in engineering, engineering management, research and **higher education**.
PEO5. Graduates will be good citizens & cultured human being with full appreciation of importance of IT **professional ethical & social** responsibilities.

Program specific outcomes

- PSO1. An ability to design, develop and implement computer programs in the areas related to Algorithms, Multimedia, Website Design, System Software, DBMS and Networking.
- PSO2. Develop software systems that would perform tasks related to Research, Education and Training and/or E governance.
- PSO3. Design, develop, test and maintain application software that would perform tasks related to information management and mobiles by utilizing new technologies to an individual or organizations.

Lab outcomes: After the completion of this course students will be able to,

LO1 Write simple assembly language programs by learning instruction set of MP & MC

LO2 Develop assembly language program for arithmetic operations,

LO3 Understand BIOS and DOS interrupts

LO4 To design I/O circuits and Memory Interfacing circuits.

Mandatory instructions for students

1. Students should report to the concerned labs as per the given timetable.
2. Students should make an entry in the log book whenever they enter the labs during practical or for their own personal work.
3. When the experiment is completed, students should shut down the computers and make the counter entry in the logbook.
4. Any damage to the lab computers will be viewed seriously.

Students should not leave the lab without concerned faculty's permission

List of experiments and progressive Assessment for TERM WORK

Name of student: ______________________ Roll No: _____________Batch: __

No	Title of Practical	Date of performance	Date of evaluation	Marks/ Grade	Page No	Sign of teacher	Remark
1	Study of block diagram, pin diagram & instruction set of 8086 microprocessor						
2	Study of Emulator EMU8086 and MASM						
3	Arithmetic operations using 8086 instructions. i. 8bit & 16bit addition. ii. Read two numbers from user & perform addition.						
4	Arithmetic operations using 8086 instructions i. 8 bit & 16 bit Subtraction. ii. 8 bit & 16 bit Multiplication. iii. Division						
5	ALP to find smallest/largest number from an array of N numbers						
6	ALP for String Operations. i. Accept string from user ii. Find length of a given string						
7	ALP for String Operations. i. Compare two Strings ii. String Reverse						
8	Assembly Language Programs using Macros and procedures						
9	To study KEIL and perform arithmetic operations for microcontroller 8051 using KEIL						
10	Keyboard Interfacing						

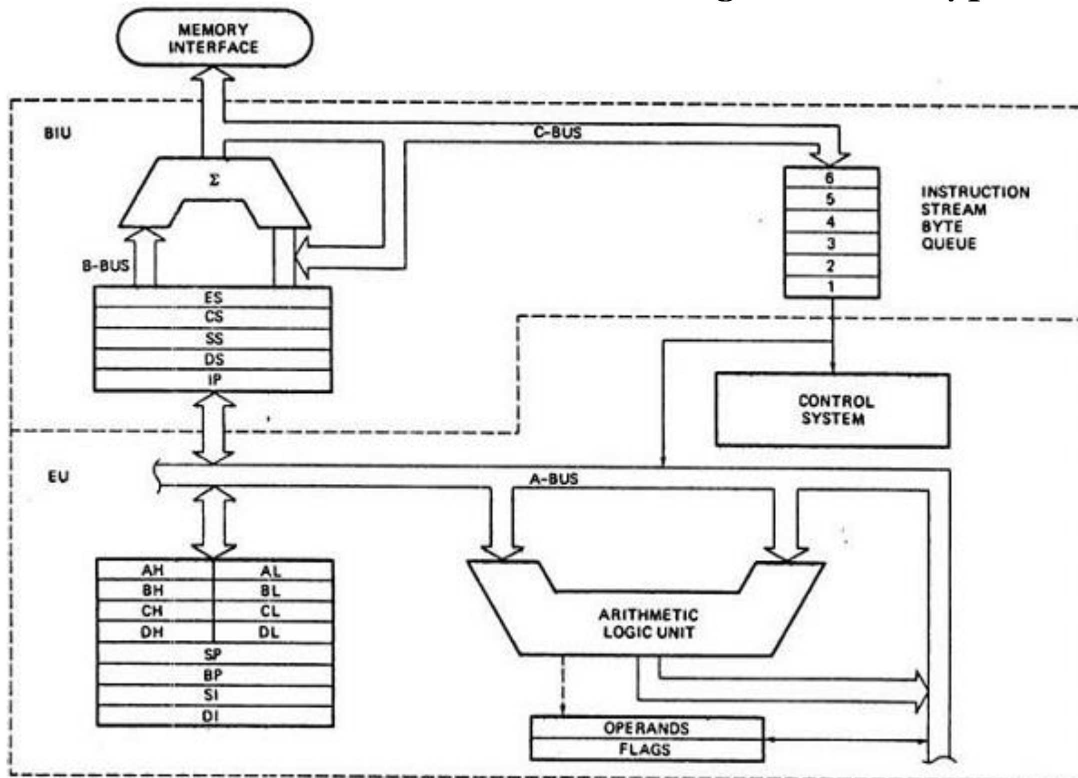
Submission Details: Date:

Name & Sign of Faculty Sign of HOD

Experiment :-01

Title:- Study of block diagram, pin diagram, instruction set of 8086 microprocessor

1.a Block diagram of 8086 μp



8086 ARCHITECTURE

Student Exercise

i. A microprocessor is a _______ chip integrating all the functions of a CPU of a computer.
   A. multiple B. single C. double D. triple

ii. Microprocessor is a/an _______ circuit that functions as the CPU of the compute
   A. electronic B. mechanic C. integrating D. processing

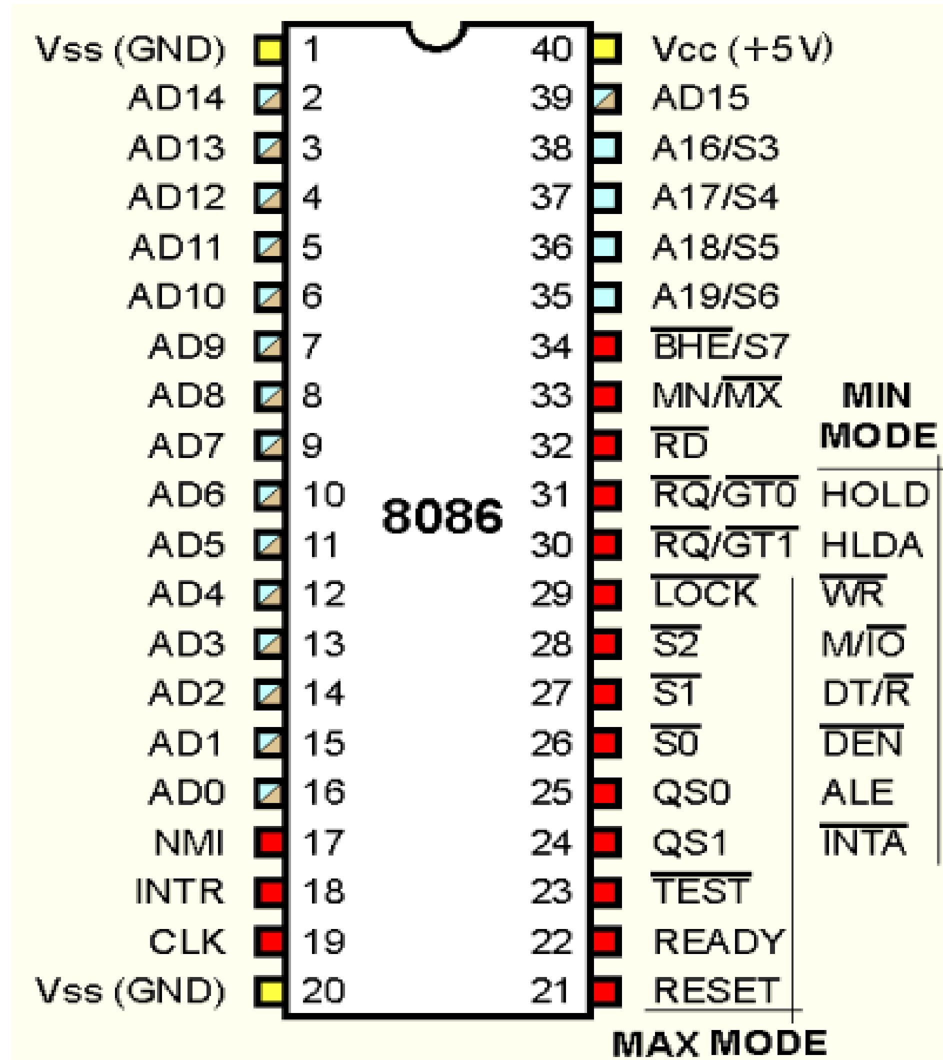
iii. The intel 8086 microprocessor is a _______ processor
   A. 8 bit B. 16 bit C. 32 bit D. 4 bit

iv. The microprocessor can read/write 16 bit data from or to ________
   A. memory B. I /O device C. processor D. register

v. In 8086 microprocessor , the address bus is ________ bit wide

A. 12 bit B. 10 bit C. 16 bit D. 20 bit

1.b <u>Pin diagram of 8086 µp</u>



Write functions of pins of 8086

1. AD0-AD15

____

____

____

____

2. A16/S3, A17/S4, A18/S5, A19/S6

3. S0, S1, S2

4 DEN:

5 ALE:

6. HLDA:

7. M/IO:

8. DT/R

9. BHE (Active Low)/S7 (Output)

10. MN/ MX

11. RD (Read) (Active Low)

12 TEST

13 READY

14 RESET (Input)

15 CLK

16 INTR Interrupt Request

17 $M/\overline{IO}$

18 $\overline{INTA}$

19 $\overline{QS_0}$, $\overline{QS_1}$

20 $\overline{RQ}/\overline{GT_0}$, $\overline{RQ}/\overline{GT_1}$

21 LOCK

1.c 8086 Instruction Set

Write one sentence explanation of following 8086 instructions :

Data Transfer Instructions

MOV : ______________________________

IN, OUT : ______________________________

LEA : ______________________________

LDS, LES : ______________________________

PUSH, POP : ______________________________

XCHG : ______________________________

XLAT : ______________________________

Logical Instructions

NOT : ______________________________

AND : ______________________________

OR : ______________________________

XOR : ______________________________

TEST : ______________________________

Shift and Rotate Instructions

SHL, SHR : ______________________________

SAL, SAR : ______________________________

ROL, ROR : ____

RCL, RCR : ____

Arithmetic Instructions

ADD, SUB :____

ADC, SBB : ____

INC, DEC : ____

NEG : ____

CMP : ____

MUL, DIV : ____

IMUL, IDIV : ____

CBW, CWD : ____

AAA, AAS, AAM, AAD: ____

____

DAA, DAS : ____

Program control Transfer Instructions

JMP : ____

JA (JNBE) : ____

JAE (JNB) : ____

JB (JNAE) : ____

JBE (JNA) : ____

JE (JZ) : ____

JG (JNLE) : ____

JGE (JNL) : ____

JL (JNGE) : ____

JLE (JNG) : ____

JC, JNC : ____

JO, JNO : ____

JS, JNS : ____

JNP (JPO) : ____

JP (JPE) : ____

LOOP : ____

LOOPE (LOOPZ) : ____

LOOPNE (LOOPNZ) : ____

JCXZ : ____

Subroutine and Interrupt Instructions

CALL, RET : ____

INT, INTO : ____

IRET : ____

String Instructions

MOVS : ____

MOVSB, MOVSW : ____

CMPS : ____

SCAS : ____

LODS, STOS : ____

REP : ____

REPE, REPZ : ____

REPNE, REPNZ : ____

Processor Control Instructions

STC, CLC, CMC : ____

STD, CLD : ____

STI, CLI : ____

LAHF, SAHF : ____

PUSHF, POPF : ____

ESC : ____

LOCK : ____

NOP : ____

WAIT : ____

HLT : ____

Student Exercise

Fill in the blanks

1. The BIU prefetches the instruction from memory and store them in _________
2. The 1 MB byte of memory can be divided into _______ segment
3. The DS is called as ________
4. The CS register stores instruction _______________ in code segment
5. The IP is _________ bits in length
6. The push source copies a word from source to _______
7. INC destination increments the content of destination by ________
8. IMUL source is a signed ___________
9. The JS is called as _______

10. The __________ instruction translates a byte from one code to another code
11. The 8086 fetches instruction one after another from ___________ of memory
12. _________ is used to write into memory
13. If MN/MX is low the 8086 operates in ___________ mode
14. The Microprocessor places ___________ address on the address bus
15. 8086 and 8088 contains ________ transistors
16. ALE stands for ____________
17. DEN is ________________?
18. The First Microprocessor was___________.
19. Status register is also called as ____________.
20. A 20-bit address bus can locate _________locations .
21. A 20-bit address bus allows access to a memory of _______________capacity
22. ____________ microprocessor accepts the program written for 8086 without any changes?
23. ________________ group of instructions do not affect the flags?
24. The result of MOV AL, 65 is to store _____________________

Conclusion

____

____

Experiment:-02

Title:- Study of Emulator for 8086-EMU8086 and MASM.

Theory:

- MASM (Microsoft Macro Assembler) is a very efficient assembly language programming tool for windows and MS-DOS. It is not a emulator but an actual programming tool helps in programming with processor.
- Emulator EMU8086 is an emulator for 8086 providing an easy and user friendly environment for assembly language programming for 8086.

ASSEMBLY LANGUAGE PROGRAM DEVELOPMENT TOOLS:

1. **EDITOR:**

   An editor is a program, which allows you to create a file containing the assembly language statements for your program.

2. **ASSEMBLER:**

   An assembler program is used to translate the assembly language mnemonic instructions to the corresponding binary codes. The second file generated by assembler is called the assembler List file.

3. **LINKER:**

   A Linker is a program used to join several object files in to one large object file. The linkers produce link files with the .EXE extension.

ASSEMBLER DIRECTIVES:

An assembler is a program used to convert an assembly language program into the equivalent machine code modules. The assembler decides the address of each label and substitutes the values for each of the constants and variables.

It then forms the machine code for mnemonics and data in assembly language program. Assembler directives help the assembler to correctly understand assembly language programs to prepare the codes.

Commonly used assembler directives are DB, DD, DW, DUP, ASSUME, BYTE, SEGMENT, MACRO, PROC, OFFSET, NEAR, FAR, EQU, STRUC, PTR, END, ENDM, ENDP etc.

Explain briefly following Assembler Directives

DB :- ____

BYTE PTR :- ____

SEGMENT :- ____

DUP (Duplicate) :- ____

____

ASSUME : - ____

____

EQU : - ____

ORG : - ____

PROC and ENDP : - ____

____

____

____

____

Execution of assembly language programming in masm software:

Assembly language programming has 4 steps.

1. Entering Program
2. Compile Program
3. Linking a Program
4. Debugging a Program

PROCEDURE:

1. **Entering Program:-**

Start Menu

Run

Cmd C:\cd MASM

C:\ MASM> edit filename.asm

C:\MASM\filename.asm This is editor Enter program here

After entering program save & exit (ALT F & Press S or ALT F &Press X)

C:\MASM>

2. Compile the Program:-

C:\MASM> MASM filename.asm

Microsoft @macro assembler version 5.10 Copy rights reserved© Microsoft Corp 1981 All rights reserved Object filename [OBJ]; List filename [NUL, LIST]; Cross Reference [NUL, CRF];

Press enter the screen shows c>

3. **Linking a Program:-**

C:> link filename.obj

Microsoft @ overlay linker version 3.64 Copy rights reserved© Microsoft corp. 1983-88. All rights reserved Object module [.OBJ]; Run file [.EXE]; List [NUL MAP]; Libraries [LIB]; Press enter till screen chows c>

4. **Debug a Program**:-

C:\> debug filename.exe - (Screen shows only dash) - t 't' for trace the program execution by single stepping starting from the address SEG.OFFSET. 'q' for Quit from Debug & return to DOS.

Program coding Format:

	CODE SEGMENT	start of code segment
START:	ASSUME CS:CODE,DS:DATA,ES:EXTRA,SS:STACK	assign names to logical segments
	MOV DX,DATA MOV DS,DX	Initialize segment registers
	…… …… …..	Program instructions
	MOV AH, 4CH	Terminate program with return code
	INT 21H	
	CODE ENDS	End of code segment
	DATA SEGMENT	start of code segment
	NUM1 DB 10H NUM2 DB 20H	Assign values to different data types in data segment
	DATA ENDS	End of data segment
	EXTRA SEGMENT	
	……………. ………..	Assign values to different data types in extra segment
	EXTRA ENDS	End of extra segment
	STACK SEGMENT	
	……….. ……….	Assign values to different data types in stacksegment
	STACK ENDS	End of stack segment
	END	End of program

Student Exercise

Write comments about the instructions and Check the values of different variables stored in data segment.

Label	Program Code	Comments
	Data segment	
	Num1 db 40h	
	Num2 db 60h	
	Num3 dw 1000h	
	Num4 dw 20A0h	
	Data ends	
	Code segment	
START:	ASSUME CS:code,DS:data,ES:extra,SS:stack	
	MOV AX, DATA	
	MOV DS, AX	
	MOV AL, NUM1	
	MOV BL, NUM2	
	MOV CX,NUM3	
	MOV DX, NUM4	
	MOV AH,4CH	
	INT 21H	
	CODE ENDS	
	END START	

WHAT ARE CONTENTS OF REGISTERS	
AL	
BX	
CX	
DX	

Conclusion:

____

____

Experiment :-03

Title:- Arithmetic Operations In 8086 (Addition)

Arithmetic Addition Instructions

ADD – ADD Destination, Source **ADC –** ADC Destination, Source

- These instructions add a number from some *source* to a number in some *destination* and put the result in the specified destination.
- The ADC also adds the status of the carry flag to the result. The source may be an immediate number, a register, or a memory location. The destination may be a register or a memory location.
- The source and the destination in an instruction cannot both be memory locations. The source and the destination must be of the same type (bytes or words).
- If you want to add a byte to a word, you must copy the byte to a word location and fill the upper byte of the word with 0's before adding.
- Flags affected: AF, CF, OF, SF, ZF.

☐ ADD AL, 74H ; Add immediate number 74H to content of AL. Result in AL

☐ ADC CL, BL ;

Add content of BL plus carry status to content of CL(CL = CL+BL+Carry Flag)

☐ ADD DX, BX ; Add content of BX to content of DX

☐ ADD DX, [SI] ;Add word from memory at offset [SI] in DS to content of DX

PROGRAM -1 (Using Registers and Immediate Data (Result is with carry))

MOV AL, 0F0H ; *Load the value to 0F0H*
MOV BL, 10H ;*Load the value to 10H*
ADD AL,BL ;

Write contents of Registers

Before execution AL :-_______________ BL:-____________________

After Execution AL :-_______________ BL:-____________________

PROGRAM -2 (Using Registers and Immediate Data (Result is without carry))

MOV AL, 01H ; *Load the value to 01H*
MOV BL, 02H ;*Load the value to 02H*
ADD AL,BL ;

Write contents of Registers

Before execution AL :-______________ BL:-__________________

After Execution AL :-______________ BL:-__________________

PROGRAM -3 (Using Registers and Immediate Data)

MOV AL, 01H ; *Load the value to 01H*
ADD AL,02H ;

Write contents of Registers

Before execution **After Execution**

AL :-______________ **AL :-______________**

PROGRAM -4 (Using Registers and Memory)

MOV AL, 01H ; *Load the value to 01H*
ADD AL,02H ;
MOV [1234H], AL ;

Physical Address=DS * 10 + 1234H .

If DS=0700H then PA = _________H

Write OUTPUT

____

____

PROGRAM -4 (To demonstrate the ADC Instruction)

```
MOV AL, 0F0H ;  Load the value to 0F0H
MOV BL, 10H  ;Load the value to 10H
ADD AL,BL  ;
ADC AL,20H ;
```

Write OUTPUT

____

____

WAP in Assembly language to read two 8 bit numbers from user and perform addition

____

____

____

____

____

____

____

____

____

____

____

____

____

____

____

WAP in Assembly language Addition of two BCD numbers

____

____

____

Conclusion:

____

____

Experiment :-04

Title:- Arithmetic Operations In 8086

i. 8 bit & 16 bit Subtraction.
ii. 8 bit & 16 bit Multiplication.
iii. Division

SUB – SUB Destination, Source   SBB – SBB Destination, Source

- These instructions subtract the number in some *source* from the number in some *destination* and put the result in the destination.
- The SBB instruction also subtracts the content of carry flag from the destination.
- The source may be an immediate number, a register or memory location. The destination can also be a register or a memory location. However, the source and the destination cannot both be memory location.
- The source and the destination must both be of the same type (bytes or words). If you want to subtract a byte from a word, you must first move the byte to a word location such as a 16-bit register and fill the upper byte of the word with 0's. Flags affected: AF, CF, OF, PF, SF, ZF.

8. Bit subtraction (Using registers)

```
MOV AL,09H      ;Load immediate data 09h to register AL
MOV BL,06H      ; Load immediate data to register BL
SUB AL, BL      ;AL=AL-BL
```

Write OutPut

____

____

(Using register and immediate data)

MOV AL, 09H ; *Load immediate data 09h to register AL*

SUB AL, 02H ; AL=AL-2

Write OutPut

____

____

(Using register and memory write explaination for each instruction)

MOV AL,07H ____

MOV BL,04H ____

SUB AL,BL ____

MOV[1234H], AL; ; ____

Write OutPut

____

____

____

16 bit Program:

(Using Registers)

MOV AX, 1F00H ; ____

MOV BX, 1234H ; ____

SUB AX, BX ;*AX=AX-BX*

Write OutPut

____

____

(Using register and immediate data)

MOV AX,1F00H ;MOVE 1FOOH 16 BIT DATA TO REGISTER AX

SUB AX,1111H ; ;AX=AX-1111H

Write OutPut

____

____

(Using registers and memory)

MOV AX,1F00H;

MOV BX,1111H;

SUB AX,BX;

MOV[1234H],AX;

Write OutPut

____

WAP in Assembly language to read two 8 bit input numbers from user and perform subtraction

____

____

____

8 bit multiplication

```
MOV AX,04H        ;MOVE 04H TO REGISTER AX
MOV BX,05H        ;MOVE 05H TO REGISTER BX
MUL BX            ;AX=AX*BX
```

Write OutPut

16 bit multiplication

```
MOV AX, 0111H     ;MOVE 0111H TO REGISTER AX
MOV BX, 1212H     ;MOVE 1212H TO REGISTER BX
MUL BX            ;AX=AX*BX
```

Write OutPut

8 bit division

```
MOV AX,20H
MOV BX,10H
DIV BX
```

Write OutPut

16 bit division

```
MOV DX,0102H
MOV AX,1000H
MOV BX,1010H
DIV BX
```

Write OutPut

____

Conclusion

____

Experiment :-05

Title: ALP to find smallest/largest number from an array of N numbers

Theory :

Data definition directives to the assembler are used for allocating storage for variables. The variable could also be initialized with some specific value. The initialized value could be specified in hexadecimal, decimal or binary form.

For example, we can define a word variable 'months' in either of the following way –

```
MONTHS      DW    12

MONTHS      DW    0CH

MONTHS      DW    0110B
```

1. Declare an array

The data definition directives can a be used for defining a one-dimensional array. Let us define a one-dimensional array of numbers.

```
NUMBERS     DW 34, 45, 56, 67, 75, 89
```

The above definition declares an array of six words each initialized with the numbers 34, 45, 56, 67, 75, 89. This allocates 2x6 = 12 bytes of consecutive memory space. The symbolic address of the first number will be NUMBERS and that of the second number will be NUMBERS + 2 and so on.

```
INVENTORY   DW  0, 0 , 0 , 0 , 0 , 0 , 0 , 0
```

The TIMES directive can also be used for multiple initializations to the same value. Using TIMES, the INVENTORY array can be defined as:

```
INVENTORY TIMES 8 DW 0
```

To access an array in assembly language, we use a *pointer*. A pointer is simply a register or variable that contains a memory address.

The value in the pointer is computed as shown in the previous sections by adding the base address of the array and the offset of the desired element.

Part of the computation can be done using offset addressing mode, but note that the offset in offset addressing mode is in bytes, and does not account for the size of an element. For example, if working with words in MAL, we must manually multiply the offset by 4.

To declare an array, specify the name of your array, the dimension of your array, the size of every element and the special system words DUP.
Size of element can be DB(for byte) or DW (for word that means 2 bytes).
Example

1 A DB 1, 2, 3, 4, 5, 6, 7, 8, 10

2 B DB DIM DUP(?)

A is an array with inital values. B is an array of dimension DIM (a constant) without initial values.

2. Access to array elements
This part is easy similar to other language, you need the right index and then you access your element.
That's means if i want to access the first element of the array A, I just write: A[0].

ALP to find smallest/largest number from an array of N numbers
____

____

____

____

Conclusion

Experiment :-06

Title:- ALP for String Operations.

A string is a collection of objects stored in contiguous memory locations. Strings are usually arrays of bytes, words, or (on 80386 and later processors) double words.

The 80x86 microprocessor family supports several instructions specifically designed to cope with strings. The 8088, 8086, 80186, and 80286 can process two types of strings: byte strings and word strings

How the String Instructions Operate

The string instructions operate on blocks (contiguous linear arrays) of memory. For example, the **movs** instruction moves a sequence of bytes from one memory location to another. The **cmps** instruction compares two blocks of memory. The **scas** instruction scans a block of memory for a particular value.

These string instructions often require three operands, a destination block address, a source block address, and (optionally) an element count. For example, when using the **movs** instruction to copy a string, you need a source address, a destination address, and a count (the number of string elements to move).

Unlike other instructions which operate on memory, the string instructions are single-byte instructions which don't have any explicit operands. The operands for the string instructions include the

- SI (source index) register,
- DI (destination index) register,
- CX (count) register,
- AX register, and
- Direction flag in the FLAGS register.

For example, one variant of the movs (move string) instruction copies a string from the source address specified by **DS:SI** to the destination address specified by ES:DI, of length cx. Likewise, the cmps instruction compares the string pointed at by ds:si, of length cx, to the string pointed at by **ES:DI**. Not all instructions have source and destination operands (only movs and cmps support them). For example, the **SCAS** instruction (scan a string) compares the value in the accumulator to values in memory.

Despite their differences, the 80x86's string instructions all have one thing in common – using them requires that you deal with two segments, the data segment and the extra segment. We have already used variable length strings in our previous examples. The variable length strings can have as many characters as required. Generally, we specify the length of the string by either of the two ways –

- Explicitly storing string length
- Using a sentinel character

We can store the string length explicitly by using the $ location counter symbol that represents the current value of the location counter. In the following example –

```
msg  db  'Hello, world!',0xa ;our dear string

len  equ  $ - msg          ;length of our dear string
```

$ points to the byte after the last character of the string variable *msg*. Therefore, ***$-msg*** gives the length of the string. We can also write

```
msg db 'Hello, world!',0xa ;our dear string

len equ 13              ;length of our dear string
```

Alternatively, you can store strings with a trailing sentinel character to delimit a string instead of storing the string length explicitly. The sentinel character should be a special character that does not appear within a string.

For example –

```
message DB 'I am loving it!', 0
```

String Instructions

Each string instruction may require a source operand, a destination operand or both. For 32-bit segments, string instructions use ESI and EDI registers to point to the source and destination operands, respectively.

For 16-bit segments, however, the SI and the DI registers are used to point to the source and destination, respectively.

There are five basic instructions for processing strings. They are –

- **MOVS** – This instruction moves 1 Byte, Word or Doubleword of data from memory location to another.
- **LODS** – This instruction loads from memory. If the operand is of one byte, it is loaded into the AL register, if the operand is one word, it is loaded into the AX register and a doubleword is loaded into the EAX register.
- **STOS** – This instruction stores data from register (AL, AX, or EAX) to memory.
- **CMPS** – This instruction compares two data items in memory. Data could be of a byte size, word or doubleword.
- **SCAS** – This instruction compares the contents of a register (AL, AX or EAX) with the contents of an item in memory.

Each of the above instruction has a byte, word, and doubleword version, and string instructions can be repeated by using a repetition prefix.

These instructions use the ES:DI and DS:SI pair of registers, where DI and SI registers contain valid offset addresses that refers to bytes stored in memory. SI is normally associated with DS (data segment) and DI is always associated with ES (extra segment).

The DS:SI (or ESI) and ES:DI (or EDI) registers point to the source and destination operands, respectively. The source operand is assumed to be at DS:SI (or ESI) and the destination operand at ES:DI (or EDI) in memory.

For 16-bit addresses, the SI and DI registers are used, and for 32-bit addresses, the ESI and EDI registers are used.

The following table provides various versions of string instructions and the assumed space of the operands.

Basic Instruction	Operands at	Byte Operation	Word Operation	Double word Operation
MOVS	ES:DI, DS:SI	MOVSB	MOVSW	MOVSD
LODS	AX, DS:SI	LODSB	LODSW	LODSD

STOS	ES:DI, AX	STOSB	STOSW	STOSD
CMPS	DS:SI, ES: DI	CMPSB	CMPSW	CMPSD
SCAS	ES:DI, AX	SCASB	SCASW	SCASD

Repetition Prefixes

The REP prefix, when set before a string instruction, for example - REP MOVSB, causes repetition of the instruction based on a counter placed at the CX register. REP executes the instruction, decreases CX by 1, and checks whether CX is zero. It repeats the instruction processing until CX is zero.

The Direction Flag (DF) determines the direction of the operation.

- Use CLD (Clear Direction Flag, DF = 0) to make the operation left to right.
- Use STD (Set Direction Flag, DF = 1) to make the operation right to left.

The REP prefix also has the following variations:

- REP: It is the unconditional repeat. It repeats the operation until CX is zero.
- REPE or REPZ: It is conditional repeat. It repeats the operation while the zero flag indicates equal/zero. It stops when the ZF indicates not equal/zero or when CX is zero.
- REPNE or REPNZ: It is also conditional repeat. It repeats the operation while the zero flag indicates not equal/zero. It stops when the ZF indicates equal/zero or when CX is decremented to zero.

I. **WAP in Assembly language to accept string from user**

____

____

____

II. **WAP in Assembly language to find length of a given string**

III. WAP in Assembly language To move a sting or a block of data from one segment to the other segment

___

___

Experiment :-07

Title:- ALP for String Operations

i. Compare two Strings
ii. String Reverse

i. **WAP in Assembly language to Compare two Strings**

____

____

____

____

____

____

____

____

____

____

____

____

____

____

____

____

____

ii. WAP in Assembly language to String Reverse

____

____

____

____

____

____

____

Conclusion

____

____

Experiment :-08

Assembly Language Programs using Macros and procedures

Procedures or subroutines are very important in assembly language, as the assembly language programs tend to be large in size. Procedures are identified by a name. Following this name, the body of the procedure is described which performs a well-defined job. End of the procedure is indicated by a return statement.

Syntax

Following is the syntax to define a procedure –

```
proc_name:
  procedure body
  ...
  ret
```

The procedure is called from another function by using the CALL instruction. The CALL instruction should have the name of the called procedure as an argument as shown below –

```
CALL proc_name
```

The called procedure returns the control to the calling procedure by using the RET instruction.

Writing a macro is another way of ensuring modular programming in assembly language.

- A macro is a sequence of instructions, assigned by a name and could be used anywhere in the program.
- In NASM, macros are defined with **%macro** and **%endmacro**directives.

- The macro begins with the %macro directive and ends with the %endmacro directive.

The Syntax for macro definition –

```
%macro macro_name  number_of_params
<macro body>
%endmacro
```

Where, *number_of_params* specifies the number parameters, *macro_name*specifies the name of the macro.

The macro is invoked by using the macro name along with the necessary parameters. When you need to use some sequence of instructions many times in a program, you can put those instructions in a macro and use it instead of writing the instructions all the time.

For example, a very common need for programs is to write a string of characters in the screen. For displaying a string of characters, you need the following sequence of instructions –

```
mov     edx,len     ;message length
mov     ecx,msg     ;message to write
mov     ebx,1       ;file descriptor (stdout)
mov     eax,4       ;system call number (sys_write)
int     0x80        ;call kernel
```

In the above example of displaying a character string, the registers EAX, EBX, ECX and EDX have been used by the INT 80H function call. So, each time you need to display on screen, you need to save these registers on the stack, invoke INT 80H and then restore the original value of the registers from the stack. So, it could be useful to write two macros for saving and restoring data.

We have observed that, some instructions like IMUL, IDIV, INT, etc., need some of the information to be stored in some particular registers and even return values in some specific register(s). If the program was already using those registers for keeping important data, then the existing data from these registers should be saved in the stack and restored after the instruction is executed.

Example

```
DATA SEGMENT
        A DB 01H
        B DB 02H
DATA ENDS
STACK SEGMENT
                DW 20 DUP(0)
                TOP DB ?
 STACK ENDS

EXTRA SEGMENT
ADDITION PROC FAR
     ASSUME DS:DATA
      MOV DX,DATA
      MOV DS,DX
      MOV AL,A
      MOV BL,B
      ADD AL,BL
      ADD AL,30H
      MOV DL,AL
      MOV AH,02H
      INT 21H
      RET
      ADDITION ENDP
EXTRA ENDS
  CODE SEGMENT
  ASSUME DS:DATA,CS:CODE,SS:STACK,ES:EXTRA
  START:
      MOV AX,DATA
      MOV DS,AX

  MOV BX,STACK
```

```
MOV SS,BX
MOV AX,EXTRA
MOV ES,AX
MOV SP,OFFSET TOP
CALL ADDITION
MOV AH,4CH
INT 21H
CODE ENDS
END START
```

Write OUTPUT

____

____

____

____

____

WAP in Assembly language using procedure to

i. **convert Hex number to Decimal**
ii. **convert Hex number to Decimal**

____

____

____

____

____

____

____

____

Conclusion

____

____

Experiment :-09

Title - To study KEIL and perform arithmetic operations for microcontroller 8051 using KEIL

Theory-

Assembler-

- An assembler allows to you to writer program using MCU instructions. It is used where almost speed, small code size and exact hardware control is essential.
- The KEIL assembler translate symbolic assembler language mnemonics into executable machine code while supporting source level , symbolic debugger powerful capabilities like macro processing.
- The assembler translates assembly source files into reloadable object modules and optionally creates listing files with symbol tables and cross reference details, complete line number, symbol type information is written to generate object files.
- This information allows exact display of program variables in your debugger. Line numbers are used for source level debugging at the other third-party debugging tools.
- KEIL assembler supports several different types of macro processors depending on architecture. Standard macro processor is easier macro processor to use. It allows you to define and use macros in your assembly programs using syntax that is compatible with that used in many other assemblers.
- Macro processing language at the MPL is a string replacement facility that is compatible with in ASM-51. Micro processor MPL has several predefined macro processor. Macros save development and maintenance since common sequence need to be developed once.

Simulation-

- Simulator mode configures the μvision debugger as software only product that accurate simulates target systems including systems including instruction and on chip peripherals. This allows application code testing before hardware is available and gives you several benefits for rapid, reliable embedded software development. Simulation allows software testing on your desktop with no hardware environment.

- Early software debugging on functional bases improves overall software reliability. Simulation allows break points that are impossible with hardware debuggers. Simulation offers optimal inputs signals hardware debugger add extra noise. Single stepping through single processing algorithm is possible. External signal are stepped when CPU halts. Failure entries that would destroy real hardware peripherals are easily chore.

KEIL Compiler-

- The μvision IDE is the easiest way for the most developers to create embedded SIM programs. To launch μvision click on your icon desktop at the selected KEIL μvision3 for start menu.
- The μvision screen provides a menu bar that contains means of commands various tool bars that contains buttons for common commands and window that displays project details, source files, dialogue box, and other information of course, multiple windows can be open simultaneously.

Simulator and programmer for 8051-

A simulator is software to mimic a microcontroller operation with a Personal Computer. This helps in running the assembly language program off-line and debug for errors. This is also a powerful learning tool before actually working with a Microcontroller A programmer is hardware used to transfer the machine code to the internal program memory of a microcontroller.

Working with KEIL Compiler μVision3-

KEIL Compiler μVision3 is a simulator/assembler for 8051 microcontroller to write and edit the code in assembly language, compile it and also to run the code. Output of the assembly language program can be verified using simulator.

Steps to use KEIL Compiler μVision3-

1. After installing the software, open8051 IDE
2. To create new project, go to: Project New Project, Save project (save project without any extension) select Microcontroller.
3. To write the assembly code in the editor , go to File New

4. After writing the assembly code in the editor, save the file with .asm extension.
5. Then go to work space and right click on source group and then add saved file.
6. Then write click on the added file and select built option to check the errors.
7. Check for the errors in the output window View Output
8. Once the error free code was made, debug the code. Debug Start debug session
9. Debug options are
   a. Step into – Each time only one instruction will be executed (single step mode).
   b. Run– To run the whole code at once.
10. Additional things:
   a. To view RAM, program memory, SFRs, and External memory, serial window, logic analyzer,
   performance analyzer window use the option VIEW.
   b. To set break points in the code (where debugging stops at that point) Debug Insert Break Point.
11. To stop the simulation : Debug stop session

After checking the code in the simulator, the code (file with .HEX extension in Intel HEX format) is loaded into Atmel 89C51 microcontroller using Universal Programmer.

To perform the arithmetic operation (HEX and BCD add and subtract) using 8051 microcontroller

1) 8 BIT ADDITION

```
MOV A, #05
MOV B,#06
ADD A,B
RET
```

INPUT		OUTPUT	
REGISTER	**DATA**	**REGISTER**	**DATA**
A	05	A	______

B	06		
A	12	A	______
B	05		

8 BIT SUBSTRACTION

```
MOV A, #04
MOV B,#02
SUBB A,B
RET
```

INPUT		OUTPUT	
REGISTER	**DATA**	**REGISTER**	**DATA**
A	04	A	02
B	02		

Students exercise

1. Write a program to add 8 bit (BCD) number

____

____

____

____

____

____

____

____

2. Write a program to substract 8 bit (BCD) number

3.Write a program for multiplication of two 8 bit (BCD) number

4. Write a program for division of two 8 bit (BCD) number

____

Conclusion:

____

____

____

____

Experiment :-10

Keyboard Interfacing