ASSIGNMENT #01

ASSEMBLY LANGUAGE

I,D # 13131

BS(COMPUTER SCIENCE):

Q1) Solve each of the following.

1/a: $64_{10} = (?)_2$

Ans) Sol:-

2	64
2	32 - 0
2	16 - 0
2	8 - 0
2	4 - 0
2	2 - 0
	1 - 0

$64_{10} = (1000000)_2$

1/b: $(01111111)_2 = (?)_{10}$

Sol:-

$= (0\times2^7)+(1\times2^6)+(1\times2^5)+(1\times2^4)+(1\times2^3)+(1\times2^2)+(1\times2^1)+(1\times2^0)$

$= 0 + 64 + 32 + 16 + 8 + 4 + 2 + 1$

$= 127.$

$(01111111)_2 = (127)_{10}$

1/c: $4D7F_{16} = (?)_{10}$

Sol:-

$= (4\times16^3)+(13\times16^2)+(7\times16^1)+(15\times16^0)$

$= (4\times4096)+(13\times256+(7\times16)+(15\times1)$

$= 16384 + 3328 + 112 + 15$

$= 19,839$

1/d: $128_{10} = (?)_{16}$

Sol:-

16	128
	8 - 0

$128_{10} = (80)_{16}$

1/e: $3AGF_{16} = (?)_2$

Sol:-

3 A G F

0011 1010 0110 1111

$(0011101001101111)_2$

1/f: $(1100001111100101)_2 = (?)_{16}$

Sol:-

1100 0011 1110 0101

C 3 E 5

$(C3E5)_{16}$

1/g): $(11111111)_2 = \pm (?)_{10}$

Sol:-

MSB is 1 Shows the number is negative

$= -(1 \times 2^7) + (1 \times 2^6) + (1 \times 2^5) + (1 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)$

$= -128 + (64 + 32 + 16 + 8 + 4 + 2 + 1)$

$= -128 + 127$

$= -1$

1/h: $-16_{10} = (?)_2$

Sol:-

2	16
2	8 - 0
2	4 - 0
2	2 - 0
	1 - 0

$(0001\,0000)_2$

Taking 2's complement

0001 0000

1110 1111

+ 1

$(1111\,0000)_2$ Ans.

1/i: $01111111_2 - 00000111_2$

Sol:-

01111111

− 00000111

01111000

Taking 2's complement

10000111

+ 1

$(1000\,1000)_2$ Ans.

1/j: $6D_{16} - 3F_{16}$

Sol:-

6 D − 3 F

0110 1101 0011 1111

6D = 0110 1101

3F = 0011 1111

Taking first number of second number.

0111111

1000000

Adding both numbers

1101101

1000000

Carry (1) 01011101

0101101

+ 1

$0010\,1110_2$ $(2E)_{16}$ Ans.

E

(Q2) write short note on each of the following.

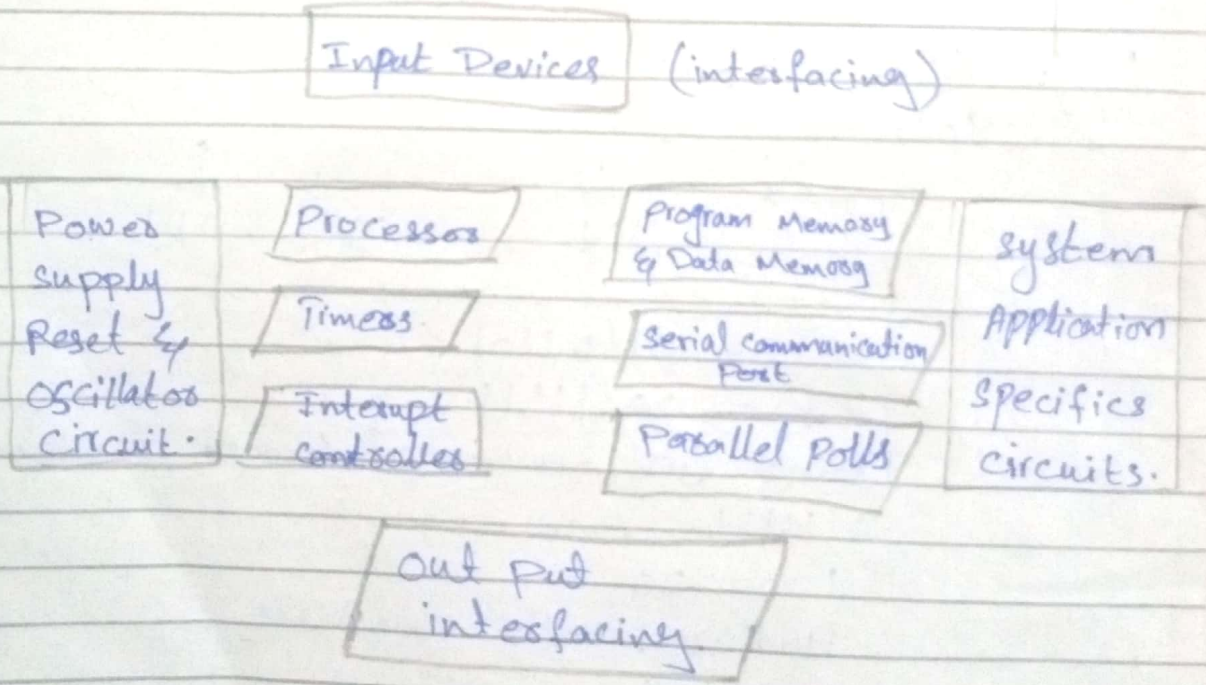
2/a: Embedded Systems.

Ans) The electronic system which integrates the hardware circuitry with software programming techniques for providing project solutions is called as embedded system.

Purpose:-

By using this embedded system technology. The complexity of the circuits can reduced to a greater extent which fusthes reduces the cost and size.

Design:-



Application:-

Embedded systems find numerous. application in various field such as digital, electronics, Telecommunication

computing, Network, Smart cards, satellite system, military defense system equipment, research equipment Ep so on.

2/b: Device Drivers :-

Device are programs that translate general operating system commands into specific refrences to hardware details.

Printer manufactuss, for example create different MS-Window, device drivers for each modul they sell often these devices drivers contain significant amount of assembly languages code.

2/c: Virtual machine Concept :-

An effective way to explain how a computer hardware and software are related is called the virtual machine concept.

Programming Language Analogy :-

=) Each computer has a native machine language. (Language L0) that runs directly on its hardware.

=) A more humen friendly language is usually constructed above machine Language called Language L1. Program written in L1 can run two different ways.

* Interpretation :-

L0 program interprets

Ep executes L1 instruction one by one.

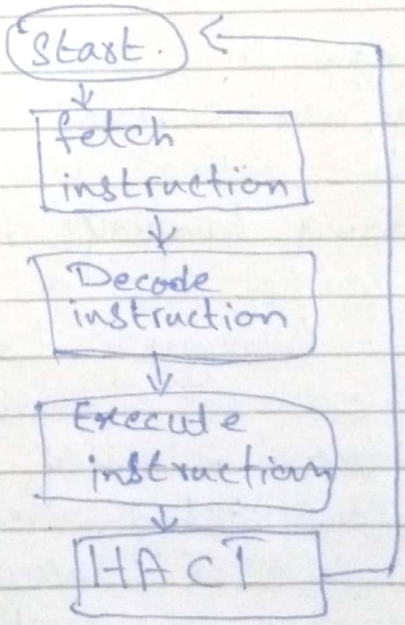
* Translation:-
L1 program is completely translated into an Lo program which runs on the computer Hardware.

2/d: Instruction execution cycle:-

=> The time period during which one instruction is fetched from Ep excute when computer given an instruction in machine language.

=> Each instruction is further divided into sequence of phases.

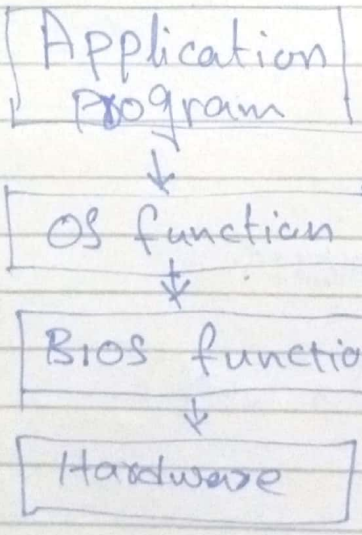
=> After excution of program Counter is incoemented in point the next instruction.



2/e: Motherboard Chipset :-

Living on the motherboard, PC's Chipset controls the communications between the CPU, RAM, Storage & other peripherals. The chipset determines how many high speed components or USB devices your motherboard can support PC chipset are designed by Intel & AMP but are found on motherboards from a variety of third Party vendors such as ISI, Asus & ASRock.

2/f: Access levels for input-output operations :-



level 3:-

call library functions to Perform genuine text I/O and file based I/O

Level 2 :-

call the operating system

function. If the OS uses GUI it was function as display graphics in a devices independant way.

Level 1 :-
call BIOS function to control colors, graphics, sound, keyboard input & low level disk I/O

Level 0 :-
send & receive data from H/w ports.

2/g: Basic parts of an assembly language instruction:-

1) Lebel (optional)
2) Mnemonin (lequired)
3) Operand (depends on the instructio)
4) Commend (optional)

- Data Lebel
- Code label

- Constiant expressin
- constant
- Memory (data lebel)
- registar.

Q3) Differentiate between each of the folowing.

a: Assembly language and high-level language.

Assembly language	high-level language.
⇒ Program written for one processor will not run on another type of Processor	⇒ Programs run independently of processor or type
⇒ Performance & accuracy of Assembly language are both high level	⇒ It not as accurate as assembly language.
⇒ Directly read pointers at a physical address	⇒ Not possible to read pointer at a Physical address.
⇒ Assembler is used	⇒ Doesn't need to know about details
⇒ Should know about registers	hardware like register.

3/b: Protected mode and real address mode.

Real Mode	Protected Mode.
⇒ only 1 MByte of memory can be addressed from hexadecimal 0000 to ffff.	⇒ It assigns each process of 4 GByte of memory
⇒ The Processor can only run one programs a time.	⇒ Processor can run muliple programs at the same time.
⇒ Application programs are permitted to access any memory location including address that are directly link to system H/W e.g MS-Dos.	⇒ Each program can be assigned its own reserved memory area and Program are prevented accedintly accessing each other's code & data e.g. MS window, Linux.

3/c: Assembler and linker.

Assembler	Linker
=> Translates assembly language program into machine code. The output of assembler is called an object.	=> computer program that links & merges various object files together in order to make executable file.

3/d: Instruction and directive.

Instruction	Directive.
=> Its a task to be carried out by the processor at run time Instruction are assembled into machine code & eventually linked into the final executable.	=> Its run instruction to the assembler telling it how to treat the data its asked to assembler for instance => It might specify the location in memory where you want the Program to be coded once its been assembl

3/e: Code label and data label.

Code label.	Data label.
=> These are just the names which are used by other instruction to jump from particular position to that position where data label is located. It must end with (:)	=> These are just like variable is any other language. e.g count DWORD 100 Data labels are declared in data segment of the code.

code labels are declared in coding segment.
e.g. Target : mov ax, bx
jmp @ target.

3/f: Line comment and block comment.

Line comment	Block comment.
=> Single line comments beginning with a semicolon character (;). All the character following semicolon on the same line ignored by assembler.	=> Beginning with comment Directive and user-specified symbol. All the subsequent lines of text are ignored by assembler until the same user specified symbol appears again. comments!

3/g: Equal-sign directive and EQU directive.

Equal-sign Directive	EQU Directive.
o Syntax => name = expression => expression is 32-bit integer (expression/ constant) => may be redifined => name is called symbolic constant. o good Programming style is use symbols COUNT = 500 : : mov al, Cont.	=> Define a symbol either an integer or text expression => Cant redefined => Syntax name EQU expression name EQU Symbol name EQU <text> eg: metric EQU 10*10 DI EQU <3+4h>

Q4 Give answer to each of the following.

a. Explain the concept of probability as it applies to programming languages.

Concept of Probability:

A language whose source Programe can be compiled and seen on a wide variety of Computer system is Said to be Portable.

(b) Why would a high-level language not be an ideal tool for writing a program that directly accesses a Particular brand of printer?

A high level language may not be Provided for direct hardware access. Even if does awkward Coding techniques must often be used, resulting in possible maintain problem.

(c) Why was Unicode invented?

Uniwell is universal computing standard to represent Text to most writting system. It was incented to store most of the words character. It is Stated during 1987.

Q4 (d) If W = 11101100, X = 00010011, and Y = 00111100, then find $Z = W \vee X \wedge \neg Y$

W = 11101100 , X = 00010011
Y = 00111100
$Z = W \vee X \wedge \neg Y$
Y = 00111100
$\neg Y$ = 11000011
$X \wedge \neg Y$ = 00010011
AND 11000011
00000011
$W \vee (X \wedge \neg Y)$
11101100
OR 00000011
11101111 Ans.

4/(e) Create a truth table to show all Possible inputs and outputs for the Boolean function described by $\neg(A \vee B)$

A	B	A∨B	¬(A∨B)
F	F	F	T
F	T	T	F
T	F	T	F
T	T	T	F

4/(f) Why does memory access take more machine cycle than registers access?

Conventional memory is outside the CPU and responds more slowly to access the request registers are hard wired inside the CPU.

4/(g) Discuss the basic Programe execution registeres used in x86 32 Bit processors.

XMM Registers

The x86 architeture also contain eight 128-Bit register called Streaming SIMD extension to the instruction.

MMX Registers:

MMX technology improves the Performace of intel Processor. when implemmenting advanced multimedia and communication application. The 64bit MMX register suppose special instruction called SIMD (single instruction multiple data)

Segment Registers:

In real address mode 16 bit Segment register indicate base addresses of Preassigned memory areas named segment.

Basic Programming Execution Registers:

Registers are high speed storage location inside the CPU designed to be accessed at much higher speed Than Conventional memory when a processing loop is optimized for speed.

(Q 5) Discuss the following MASM directive in detail.

INCLUDE :-

Insert source code from the source file given filename into the current source file during assembly.

Syntax.

INCLUDE file name.

386 :-

Enable assembly of non privileged instructions for the 80386 processor disable assembly of instructions introduced with later processors.

MODEL :-

Intialize the program memory model. The memory model can by TINY, SMALL, COMPACT, MEDIUM, CARGE, HUGE, or FIAT. The langtype can be, C, BASIC, FORTRAN.

STACK :-

When used with, Model defines a stack segment. The optional size specifies the number of bytes for the stack.

PROTO :-

The Proto directives by using the invoke directive.

Syntax:

Label PROTO [distance] [Language type] [[parameter]: Tag]].

DATA :-

When used with model, starts a near data segment: for intialized data.

CODE:-
When used with model indicates the start of a code segment called.

PROC:-
Set of code, that can be browsed to and returned from in such a way.

ENBP:-
Mark the end of the prodrse name previously begin with PROC.

END:-
Mark the end of Model and optionally sets the program entry point to address.

Q6)a: write a program that calculates the following expressions: A=(A+B)-(C+D)

```
.386
.model flat, stdcall
.stack 4096
 ExitProcess PROTO
 dw ExitCode: DWORD
.code
 Main PROC
  mov eax, 3h
  mov ebx, 8h
  mov ecx, 1h
  move dx, 8h
  add eax, ebx
  add ecx, edx
  sub eax, ecx
 Invoke, ExitProcess, 6
```

main ENDP:
END main.

Q6/b: Show the order of individual bytes in memory for the following doubleword variable using little endian order:

dval DWORD 12345678h.

0000	78
0001	56
0002	34
0003	12

6/c: write a statement that causes the assembler to calculate the number of in the following string. and assign the value to a symbolic constant named stringsize: string1 byte "Assembly language is easy", 0

```
.data
string1 byte "Assembly language is easy", 0
stringsize byte ?
.code
moveax, SIZE of string1
move stringsize, eax.
```

6/d: write a program that performs arithmetic operations on different register operands and stores the result in memory. Give stepwise explanation of each statement.

Let the Arithmetic operations is

$$x = -m + (n-o)$$

```
.data
X DWORD ? ; uninitialized variable.
```

```
m DWORD 10  ; initialized variable m
n DWORD 25  ; initialized variable n
O DWORD 17  ; initialized variable O
.code
main PROC
mov eax, m  ; moves value of m into eax
neg eax     ; EAX = -12
mov ebx, n  ; move value n in ebx
mov ebx, O  ; Ebx = 13
add eax, ebx ; -12 + 13
mov x, eax  ; (1)
call DumpRegs
exit
main ENDP
END main
```