Assignment NO. 2.

D D M M Y Y Y Y

Q1) Explain different ways of specifying target address in control transfer instructions with.

→

The target address in a control transfer instruction can be specified in one of the following ways.

[a] Immidate constant:

In this method, target address is specified along with instruction as an immidiate data.

Eg.

(a) jmp 0x100

Target address = A + 0x100H

where A = Address of next instruction

(b) jmp * 0x100

Target address = 0x100H

[b] Register Addressing:

In this method, target address is specified using a register.

Eg:

jmp *%eax

jmp %ebx

[c] ~~Memory~~ Addressing Mode:-

In this method target instruction is read from memory.

Eg:

jmp (%eax)

Here address of target location instruction is stored in memory location whose address is in ~~regular~~ register eax.

Q2. Write short note on function call and return.

→ Functions are subprograms to which the control of execution is passed temporarily. After executes of the called function is completed the control is transferred back to the calling program.

The most important reason is to reuse the code. Example: If in a program several strings are to be displayed on the screen of different time, it is better to write function to displayed, this function can be called.

Writing function in a program to make it readable and understable.

Example:

```
CALL FUNCTION
0x100C  PUSH Y.EAX
0x1100  FUNCTION:
```

During execution of CALL instruction, address of next instruction is saved on stack. Stack pointer is decremented by 4. the register EIP changed to 0x00001100 there by control transfer to function FUNCTION. IA 32 Archecteure provide an instruction called RET to return control to the caller after execution of the called function is completed.

The RET instruction pops a 32 bit offset from top of the stack into EIP register thereby causing a control transfer.

Q3. Write 80386 assembly language program for function call and return.

→

```
.section .data
output
        .output :
                .asciz "This is section %d \n"

.section .text
.globl _start
_start:  pushl  $1
         pushl  $output
         call   printf
         addl   $8, %esp
         call   function
         pushl  $3
         pushl  $output
         call   printf
         addl   $8, %esp

movl  $1, %eax
movl  $0, %ebx
int   $0x80

function:  pushl  %ebp
           movl   %esp, %ebp
           pushl  $2
           pushl  $output
           call   printf
           addl   $8, %esp
           movl   %ebp, %esp
           popl   %ebp
           ret
```

Q4. Write 8086 assembly language program to find Largest number from an array of 32 bit number

→

```
.section .data
value:
    .int 100, 56, 130, 65, 12

.section .text
.globl _start
_start: nop

    movl value, %eax
    movl $1, %edi
back: movl value(, %edi, 4), %ebx
    cmpl %eax, %ebx
    cmovbe %ebx, %eax
    inc %edi
    cmpl $5, %edi
    jne back

    movl $1, %eax
    movl $0, %ebx
    int $0x80.
```

Q5. The values of registers are given to be all 0 prior to the execution of the following sequence of instructions.

```
pushl $0x487
pushl $0x53
pushl $0xffeeccdd
inc %esp
pop %ax
```

inc %esp
pop %cx.
find value of ax, cx and esp after execution of all instruction

→ Assuming, esp (stack pointer intially 100)

dd	92
cc	93
ee	94
ff	95
33	96
00	97
87	98
04	99
xx	100

Before execution starts:
esp = 100 (containing Garbage value)
ax = 0
cx = 0

(i) pushw $0x4487
pushes 16 bit data on the stack and stack pointer is decremented by 2.
∴ esp = 98

(ii) pushw $0x33
pushes 16 bit data on the stack and stack pointer is decremented by 2
∴ esp = 96

(iii) pushl $0xfreeccdd
pushes 32 bit data on the stack and stack pointer is decremented by 4
∴ esp = 92

(iv) inc %esp
Increments stack pointer by 1
∴ esp = 93

(v) pop %.ax

pops out 16 bit data from top of stack and stores to ax register and esp incremented by 2

∴ value at ax = eecc and esp = 95

(vi) inc %.esp → ∴ esp = 9c

(vii) pop %.cx → ∴ cx = 0033 and esp = 98

After Execution:

ax = 0xeecc , cx = 0x33 , esp = 98

Q7. Explain following instruction with instruction example

(a) cmovbe

cmovbe src, dest

If (dest <= src) move the content of source to destination

source can be register or memory but destination is always register. This instruction is used immediately after compare instruction

Eg: cmovebe %.eax, %.ebx.

(b) xchg:

xchg r_1, r_2

It exchanges the contents of r_1 and r_2.

r_1 and r_2 can be registers or memory locations.

Eg: (i) xchg %.eax, %.ebx (Register to Register transfer)

(ii) xchg %.eax, (%.esi)

or (Register ↔ memory transfer)

xchg (%.esi), %.eax

(c) bswap

bswap %eax

It converts big endian type data to little endian type because Intel doesn't support big endian.

Eg: value = 0x11223344.

Before bswap:

44	33	22	11
3	2	1	0

After bswap:

11	22	33	44
3	2	1	0

(d) cmpl:

cmpl src dest.

If dest - src > 0, carry flag is reset else carry flag is set

i.e. if dest - src > 0 ⟶ cy = 0

dest - src < 0 ⟶ cy = 1

Register value are not changed after execution, only flags are updated.

Eg: cmpl %eax, %ebx

(e) ~~ebtw~~:

This instruction converts byte (8bit) to word (16 bit). sign bit (7th bit) of the source is extended.

Eg: Before execution, value = 1111 0000

After execution, value = 1111 1111 1111 0000

(f) pushl: pushl src

The source operand of push instruction can be an immediate data, registers or memory variable.

The source contents are located on the stack & stack pointer (esp) is decremented.

(g) popl : popl dest.

The operand can be only registers or memory variable..

The value at the top of the stack is written to destination and stack pointer is incremented.

Eg: popl %eax, popl %ax.

Q8. Write assembly language program to demonstrate function parameter passing by address and reference.

→

```
.section .data
num1:
    .int 0xff
num2:
    .int 0xee

.section .text
.globl _start
_start: nop.
        pushl $num1
        pushl $num2
        call exchange
        addl $8, %.esp
        movl $1, %.eax
        movl $0, %.ebx
        int $0x80.

exchange: pushl %.ebp
          movl %.esp, %.ebp
          movl 8(%.ebp), %.eax
          movl 12(%.ebp), %.ebx.
```

```
xchg  (%.eax), %.ecx
xchg  (%.ebx), %.ecx
xchg  (%.eax), %.ecx
movl  %.ebp, %.esp
popl  %.ebp
ret
```