Assignment No - 3

19UCS122

D D M M Y Y Y Y

Arithmetic, BCD, Logical, String bitwise operations

Q1. Write short note on:

(a) Arithmetic Instructions:

→ (1) ADD: This instruction adds the value stored in the source operand into destination operand

Ex: add %.eax, %.ebx

(2) ADC: This instruction adds the value stored in source operand or value of into destination operand

Ex: adc %.eax, %.ebx

(3) SUB: This instruction subtract the value stored in the source operand into destination operand

Ex: SUB %.eax, %.ebx

(4) SBB: This instruction subtract the value stored in source operand and the value of CF into destination operand.

Ex: Sbb %.eax, %.ebx

(5) inc %.eax: Used to increment %.eax by 1

(6) dec %.eax: Used to decrement %.eax by 1

(7) neg %.eax: NEG - This instruction negate the operand

(8) cmp: This instruction is similar to substraction except the destination is not modified. This instruction compare source and destination operand & accordingly set the Flags.

19UCS122

Ex: cmp src, dest

(9) MUL = It performs unsigned multiplication
Ex: mull %ebx

(10) IMUL = It performs signed multiplication
Ex: imull %ebx

(11) DIV = It performs unsigned division
Ex: div %ecx

(12) IDIV = It performs signed division
Ex: idiv %ecx

Q1.

(b) Shift and rotate instructions.

→ 1) The shift instructions are primarily for multiplication or division by a number (2^n)

2) There are two kinds of shift instructions.
(a) for signed numbers (b) for unsigned numbers.

3) Shift operation for signed numbers are also known as arithmetic shifts.

4) Shift operations for unsigned numbers are known as logical shifts

5) Arithmetic shifts : SAR counts, dest
SAL/SHL Count, dest

6) Logical shifts : SHR Count, dest.

(c) String instruction

→ (1) MOVSB / MOVSW / MOVSL : instruction copies a byte/word/long word from a source string to destination string.

(2) After each copy, the addressed stored in ESI and EDI registers are incremented or decremented depend upon the value of direction flag and size of each element

(3) REP prefix : It is used to repeat a string instruction a number of items controlled by counter value ECX.

(4) CMPSB / CMPSW / CMPSL / :- compare an element of source string with an element of destination string.

Load: Memory to register
store: opposite

D D M M Y Y Y Y

19UCS122

(5) LODSB | LODSW | LODSL :- Provides address of string in registers esi. It loads string from memory to register al/ax/eax.

(6) STOSB | STOSW | STOSL :- Provides address of string in register edi. It store string element from register al/ax/eax to memory location whose offset is stored in edi register.

(D) Bit testing and modification

→ (1) BSF : This instruction is used to return inter of LSB that is set to 1 in source operand.

Ex: BSF src, dest.

(2) BSR : This instruction is used to return index of MSB that is set to 1 in source operand.

Ex: BSR src, dest.

(3) Bt bitoffset, dest : It copies the specified bit from the destination operand into carry flag.

bts :- bit is set to 1

btr :- bit is set to 0

Q2. Explain the following BCD instruction with an instruction example.

(a) AAA :

→ This instruction is executed after an ADD instruction that adds two ascii coded operands to give a byte of result to AL. It converts the resulting contents of AL to unpacked decimal digits.

```
Ex: movb $0x4, %al
    movb $0x4, %bl
    addb %bl, %al        ; al = 0x8
    aaa                  ; al = 0108
```

(2) AAS:

This instruction is executed after execution converts the product available in al into unpacked BCD format

```
Ex: movb $0x4, %al
    movb $0x4, %bl
    mul %bl, %al         ; al = 0x24
    aam
```

D D M M Y Y Y Y

19/05/22

(b) AAS:

→ This instruction is executed after a SUB instruction that substracts two ascii coded operand to give a byte of result in AL

```
Ex: movw   $.0x103, %.ax
    movb   $0x9, %.bl
    subb   %.bl, %.al
    aas
```

(c) AAM:

→ This instruction after execution converts the product available in al into unpacked BCD format

```
Ex: movb   $0x4, %.al
    movb   $0x4, %.bl
    mul    %.bl, %.al          ; al = 0x24
    aam                        ; al = 0306
```

(D) AAD:

→ This instruction converts two unpacked BCD digits and AL to the equivalent binary number in al. This adjustment must be made before division

```
Ex: movb   $4, %.bl
    movw   $0x0105, %.ax
    aad                        ; ax = 0xf
    div %.bl                   ; ah = 0x03 (Remainder)
                               al = 0x03 (Gradient)
```

(E) DAA:

→ This instruction is used to convert the result of the addition of two packed bcd numbers to a valid bcd number

D D M M Y Y Y Y

19UCS122

Ex: AL = 53, CL = 29

```
add b %.cl, %.al    ; al = 0x7c
daa                 ; al = 0x82
```

Q3. Write an 80386 assembly language program to perform following operations.

(a) 64 bit addition

→
```
.section .data
val 1:
    .quad 0x1122334455667788
val 2:
    .quad 0x1111222233334444

.section .text
.globl _start
_start: nop
```

```
movl  value1 , %.eax
movl  value1+4, %.ebx
movl  value2 ,  %.ecx
movl  value2+4, %.edx
addl  %.ecx , %.eax
adcl  %.ebx , %.edx
```

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

output

$eax : 0x8899bbcc

$edx : 0x22335566

(b) 32 bit unsigned multiplication

→

```
.section .data
value1:
    .int 0xf
value2:
    .int 0x2

.section .text
.globl _start
_start : nop

    movl value1 , %.eax
    movl value2 , %.ebx
    mul %.ebx

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

output:
  $eax = 0x1e
```

19UCS122

(c) 64/32 unsigned division

→
```
.section .data
value 1:
        .quad 0xf
value 2:
        .int 0x2

.section .text
.globl _start
_start: nop

    movl value 1 , %eax
    movl value 1+4 , %edx
    movl value 2 , %ecx

    div %ecx

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

output: %eax : 0x7 (quotient)
%edx : 0x1 (Remainder)

Q4. Describe Attribute of string in detail.

→ Attributes of string :-

(1) Address: Address of string provides address in memory of first / Last element of string

(2) Size of each element: string instruction also supports size of an element to be 8/16/32 bit
eg: b, w, l.

19UCS122

(3) Number of element: Register ECX is used in a string instruction to store the count of number of elements

(4) Direction for address adjustment:
In IA32 architecture, DF is used to represent the direction for address adjustment.
~~IEDF~~ =0 If DF=0, address is incremented.
If DF=1, address is decremented.

Q5. Write an 80386 assembly language program to count the number of one's in a given 32 bit number stored in memory location using bitwise instruction

→
```
.section .data
value1:
    .int 0xf

.section .text
.globl _start
_start: nop
    movl value1, %ebx
    movl $12, %ecx
    xor %eax, %eax
back: bsf %ebx, %ecx
    jz end
    btr %ecx, %ebx
    inc %eax
    jmp back
end:
    movl $1, %eax
    movl $0, %ebx
    int $0x80
```

output:
$eax = 0x4

19UCS122

Q6. Write an 80386 assembly language program to count the largest sequence of consecutive one's in a given 32 bit number stored in memory location

→
```
.section .data
value1:
    .int 0x7e0f3

.section .text
.globl _start
_start: nop
    movl value1, %ebx
    jz end
    xor %edx, %edx
    movl %ebx, %ecx
back: addl %ebx, %ebx
    and %ecx, %ebx
    jz next
    inc %edx
    jmp back
    next: addl $1, %edx
end:
    movl $1, %eax
    movl $0, %ebx
    int $0x80
```

Q7. Write an 80386 assembly language program to move a string from source to destination memory location.

→
```
.section .data
value1:
    .ascii "This is a test string \n"
```

```
.section .bss
    .lcomm output ,23

.section .text
.globl _start
_start: nop
    movl $value1, %esi
    movl $output, %edi
    movl $23 , %eax
    cld
    rep movsb
movl $1 , %eax
movl $0 , %ebx
int $0x80

output: "this is a test string \n"
```

Q8. Write an 8086 assembly language program to find the length of the given strings

→

```
.section .data
str:
    .asciz "DKTE"

.section .text
.globl _start
_start: nop
    movl $str , %esi
    xor %edx , %edx.
    cld
back: movb (%esi), %al
    cmpb $0x0 , %al
    je end
    inc %edx
    inc %esi
    jmp back

end:
    movl $1 , %eax
    movl $0 , %ebx
    int $0x80
```