

# Assignment No - 3

19UCJ122

D	D	M	M	Y	Y	Y
---	---	---	---	---	---	---

## Arithmetic, BCD, Logical, String, bitwise operations

Q1. Write short note on:

(a) Arithmetic Instructions:

→ (i) ADD : This instruction adds the value stored in the source operand into destination operand  
Ex: add %eax, %ebx

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

Ex: adc %eax, %ebx

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

Ex: sub %eax, %ebx

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

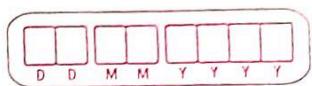
Ex: sbb %eax, %ebx

(v) inc %eax : Used to increment %eax by 1

(vi) dec %eax : Used to decrement %eax by 1

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

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



### 19UGS12.2

Ex: cmp src, dest

(9) MUL = It performs unsigned multiplication

Ex: mull 1.ebx out broadword full

(10) IMUL = It performs signed multiplication

Ex: imull 1.ebx 3,807 add word

(11) DIV = It performs unsigned division

Ex: div 1.ecx

(12) IDIV = It performs signed division

Ex: idiv 1.ecx 17 (applied to b)

Q1.

(b) Shift and rotate instructions.

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

ii) There are two kinds of shift instructions.

(a) for signed numbers (b) for unsigned numbers.

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

iv) Shift operations for unsigned numbers are known as logical shifts.

v) Arithmetic shifts : SAR, count, dest

SAL/SHL count, dest

vi) Logical shifts : SHR, count, dest

(c) String instruction

→ i) MRSB | MRSW | MRSL : instruction copies a byte | word | long word from source string to destination string.

ii) After each copy, the addresses stored in EDI and EDI registers are incremented or decremented depend upon the value of direction flag and size of each element.

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

iv) CMPSB | CMPSH | CMPSL : compare an element of source string with an element of destination string.

Load: Memory to register  
Store: opposite



### 19UCS12.2

- (a) `lodsb | lodsh | lodsl` :- provides address of string in registers esi. It loads string from memory to register al/ax/eax.  
(c) `stosb | stos h | stosl` :- provides address of string in register edi. It stores string element from register al/ax/cx to memory location whose offset is stored in edi register.

### (d) Bit testing and modification

→ (1) `BSF` : This instruction is used to return index 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  
      add b %bl, %al  
      aaa  
      ; al = 0103

This instruction is executed after execution converter the product available in AL to into unpacked BCD format.

Ex: movb \$0x4, %al  
      movb \$0x4, %bl  
      mulb %bl, %al  
      aam  
      ; al = 0x24, bl = 0x00



19UCS122

(b) AAS:

- This instruction is executed after a SUB instruction that subtracts two ASCII coded operand to give a byte of result in AL.

Ex:   
 mow \$0x103, i. ax  
 mow b \$0x9, i. bl  
 subb i. bl, i. al

(c) AAM:

- This instruction after execution converts the product available in AL into unpacked BCD format.

Ex:   
 mowb \$0x9, i. al  
 mowb \$0x404, i. bl ; al = 0x100  
 mul i. bl, i. al ; al = 0x24  
 aam ; al = 0306

(d) AAD:

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

Ex:   
 mowb \$4, i. bl  
 mow \$0x0105, i. ax ; ax = 0xf  
 aad ; ax = 0xf  
 div i. bl ; ah = 0x03 (Remainder)  
 ; al = 0x03 (Quotient)

(e) DAA:

- This instruction is used to convert the result of the addition of two packed BCD numbers to a valid BCD number.

• In 8088, if a result has a  
negative sign bit



Ex:  $AL = 53, CL = 29$

19UCS122

add b [1.cl], [1.al] ;  $al = 0x7C$

dqa [1.cl], [1.al] ;  $al = 0xB2$

Q3. Write an 8086 assembly language program to perform following operations.

(a) 64-bit addition

→ .section .data

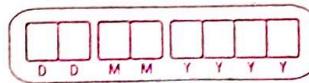
val1: .quad 0x1122334455667788

val2: .quad 0x1111222233334444

.section .text

.globl \_start

\_start: nop



19UCS122

```

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

movl \$1, %eax	output = 1
movl \$0, %ebx	\$eax: 0x8899bbcc
int \$0x80	\$edx: 0x22335566

(b) 32 bit unsigned multiplication

```

→ .section .data
value1: .int 0xf
value2: .int 0x2
.value1, %eax
.value2, %ebx
mul %ebx, %eax
movl $1, %eax
movl $0, %ebx
int $0x80
output: $eax = 0x1e
  
```

D	D	M	M	Y	Y	Y	Y
---	---	---	---	---	---	---	---

19UCS122

(c) 64/32 Unsigned division

→ .section .data

value 1: .quad 0x1000000000000000

.quad 0x1000000000000000

value 2: .int 0x2

.int 0x2

.section .text

.globl \_start

\_start: .nop

move value1, %eax

move value1, %edx

move value2, %ecx

div %ecx

move \$L, %eax

move \$0, %edx

int \$0x80

Output: %eax : 0x7 (Quotient)

%edx : 0x1 (Remainder)

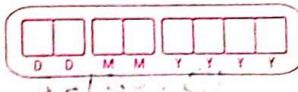
Q1. Describe Attribute of string in detail.

→ 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.

~~IE~~DF = 0 If DF = 0, address is incremented.

If DF = 1, address is decremented.

Q5. Write an 8086 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 0x1f

.section .text

.globl \_start

\_start: nop

movl value1, %ebx

movl \$12, %ecx

xorl %eax, %eax

back: bsf %ebx, %ecx

jz end

bt %ebx,%ecx, %ebx, %eax

inc %eax

jmp back

end:

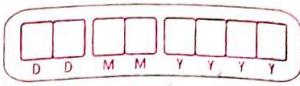
movl \$1, %eax

movl \$0, %ebx

int \$0x80

Output:

\$eax = 0x9



19UCS122

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

```

→ .section .data
value1: dd 4142434445464748494A4B4C4D4E4F4G4H4I4L4J4K4L4M4N4O4P4Q4R4S4T4U4V4W4X4Y4Z4
        ; initialization of 32 bits of value with ones

→ .section .text
        ; code section
        .globl _start
_start:    nop
        movl $value1, %eax
        addl $1, %eax
        jz end
        xorl %eax, %eax
        movl %eax, %ebx
back:     addl %ebx, %ebx
        andl %ebx, %ebx
        jz next
        inc %ebx
        jmp back
next:    addl $1, %ebx
end:      movl $1, %eax
        movl %eax, %ebx
        int $0x80
    
```

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

```

→ .section .data
value1: db 'This is a test string\r\n'
        ; source
        .section .text
        movl value1, %eax
        movl $value1, %ebx
        ; destination
        addl $1, %eax
        addl $1, %ebx
        ; loop
        jne end
        ; end
        addl $1, %ebx
    
```

IGUCS122

D	D	M	M	Y	Y	Y	Y
---	---	---	---	---	---	---	---

.section .bss  
.lcomm output, 23

.section .text  
.globl \_start  
\_start:  
nop  
movl \$value, %esi  
movl \$output, %edi  
movl \$23, %eax

output: "This is a test string\n" 0

Q8. Write an 80386 assembly language program to find the length of the given strings.

→ .section .data  
str:  
.asciz "DKE"

.section .text  
.globl \_start  
\_start:  
nop  
movl \$str, %esi  
xor %edx, %edx  
clt

inc %edx  
inc %esi  
jmp back  
movl \$1, %eax  
movl \$0, %ebx  
int \$0x80