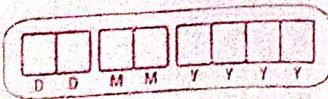


Assignment No - 3



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 or 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.

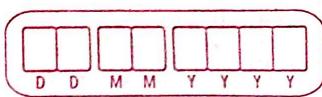
Ex: sbb %eax, %ebx

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

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

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

(8) CMP: This instruction is similar to subtraction except the destination is not modified. This instruction compare source and destination operand & accordingly set the flags.



19UGUJ22

Ex: cmp src, dest

(9) MUL = It performs unsigned multiplication

Ex: mull bl, ebx

(10) IMUL = It performs signed multiplication

Ex: imull bl, ebx

(11) DIV = It performs unsigned division

Ex: div bl, cx

(12) IDIV = It performs signed division

Ex: idiv bl, cx

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, optimally.

ii) After each copy, the addresses stored in ESI 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 values. E.g.,

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



19UCS122

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

(6) STOSB | STOSH | STOSL :- Provides address of string in register edi. It stores 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 index of MSB 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) BTF 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



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: moww \$0x103, 'i. ax
moxb \$0x9, 'i. bl
subb 'i. bl, 'i. al
aas

(C) AAM:

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

Ex: morb \$0x4, 'i. al
morb \$0x4, 'i. bl
mul 'i. bl, 'i. al ; AL = 0 x 24
aam ; AL = 0306

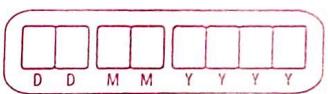
(D) AAD:

- 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: morb d4, 'i. bl
moww \$0x0105, 'i. ax
aad ; AX = AXF
dir 'i. bl ; AH = AX03 (Remainder)
AL = AX03 (Quotient)

(E) DAA:

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



Ex: AL = 53, CL = 29 ; addl \$1, %al ; al = 0x7C

19UC5122

addl \$1, %cl, %al ; al = 0x7C

addl \$1, %al ; al = 0x82

So numbers written in hex are loaded.

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

(a) 64 bit addition

→ .section .data

val1:

.quad 0x1122334455667788

val2:

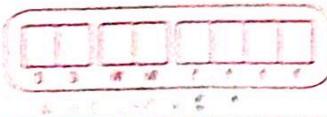
.quad 0x1111222233334444

.section .text

.global _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
movl \$0, %ebx
int \$0x80

output	0x00000000
\$eax:	0x55555555
\$edx:	0x22335566

(b) 32 bit unsigned multiplication

→ .section .data

value1:

.int 0xf

value2:

.int 0xe

.section .text

.global _start

_start: nop

movl value1, %eax

movl value2, %ebx

mult %ebx

movl \$1, %eax

movl \$0, %ebx

int \$0x80

output:

\$eax = 0x1e

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

19UCS122

(c) 64/32 unsigned division

→ .section .data

value 1:

.quad 0xf

value 2:

.int 0x2

.section .text

.globl _start

_start: .nop

movl value1, %eax

movl value1+4, %edx

movl value2, %ecx

divl %ecx

movl \$1, %eax

movl \$0, %ebx

int \$0x80

Output: %eax : 0x7 (Quotient)

%edx : 0x1 (Remainder)

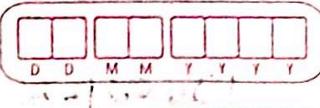
Q4. Describe Attribute of string in detail.

→ Attributes of string :-

(i) Address: Address of string provides address in memory of first | Last element of string

(ii) 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 8086 assembly language program to count the number of ones in a given 32 bit number stored in memory location using bitwise instruction

→ .section .data

value1:

.int 0x1

.section .text

.globl _start

_start: nop

more 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 = 0x9

D	D	M	M	Y	Y	Y

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:

.int 0x7e0f3

.section .text

.globl _start

_start: nop

more value1, _i.ebx

j2 end

xor _i.edx, _i.edx

more _i.ebx, _i.ecx

back: addl _i.ebx, _i.ebx

and _i.ecx, _i.ebx

j2 next

inc _i.edx

jmp back

next: addl \$1, _i.edx

end :

morel \$1, _i.eax

more \$0, _i.ebx

int 3

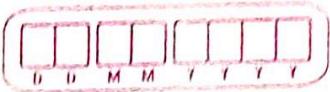
Q7. Write an 80386 assembly language program to move a string from source to destination memory location.

→ .section .data

value1:

.asci "this is a test string \n"

19UCS122



.section .bss
.lcomm output , 23

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

output: "This is a test string \n"

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

→ .section .data

str:

.asciz "DETE"

.section .text
.globl _start
_start: nop
 movl \$str, %esi
 xor %edx, %edx
 cld
 back: movb (%esi), %al

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