CS222- Lab 4

Assembly language Programming

The goal of this is to familiarize the students with 8086 assembly language features.

Your assignment:
Using Dosbox environment, develop and Test the following (8086 ASM programs).

Develop test programs (2 example each) for each of the addressing modes below and verify the results.

1. **Register Addressing**
2. **Immediate Addressing**
3. **Direct Addressing**
4. **Register Indirect Addressing**
5. **Based Addressing**
6. **Indexed Addressing**
7. **Based Index Addressing**
8. **String Addressing**
9. **Direct I/O port Addressing (no need to test)**
10. **10. Indirect I/O port Addressing (no need to test)**

11. Relative Addressing
12. Implied Addressing

Ans:
Test code 1:

```
.model small
.stack 64

.code
start:  mov ax,08a3h     ;immediate addressing
        mov bx,ax       ;register addressing
        mov ax,[5000]           ;direct addressing
        add al,[bx]      ;register indirect addressing
        jz nxt          ;relative addressing
        mov cx,ax
nxt:    mov al,[bp+0100] ;based addressing
        mov ax,[si+1000] ;indexed addressing
        mov ax,[bp+di]          ;based index addressing
        ;movs            ;string addressing
        ;in [4000], 45h  ;direct i/o addressing
        ;out [dx], ax    ;indirect i/o addressing
        clc              ;implied addressing
        end start
        .end
```

Test code 2:

```
.model small
.stack 64

.code
start:  xor ax,ax         ;register addressing
        add ax,9133h     ;immediate addressing
        mov bx,[5000]            ;direct addressing
        sub al,[bx]      ;register indirect addressing
        jnz nxt          ;relative addressing
        mov cx,ax
nxt:    and al,[bp+0100] ;based addressing
        xor ax,[si+1000] ;indexed addressing
        or ax,[bp+di]   ;based index addressing
        ;movs             ;string addressing
        ;in [4000], 45h  ;direct i/o addressing
        ;out [dx], ax    ;indirect i/o addressing
        clc               ;implied addressing
        end start
        .end
```

P2.

Develop test programs (5 example each) for each of the case below and verify the results.

1. Data Transfer Instructions
2. Arithmetic Instructions
3. Logical Instructions
4. String manipulation Instructions
5. Process Control Instructions
6. Control Transfer Instructions

Test code 1:

```
.model small
.stack 64

.code
start:  xor ax,ax
        mov bx,2342h
        mov cx,3421h
        mov ax,bx       ;data transfer
        add ax,cx       ;arithmetic
        jz ke           ;control transfer
        and ax,bx       ;logical
ke:     nop             ;process control

        end start
        .end
```

Test code 2:

```
.model small
.stack 64

.code
start:  xor ax,ax
        mov bx,2342h
        mov cx,0421h
        mov ax,bx       ;data transfer
        sub ax,cx       ;arithmetic
        jpe ke          ;control transfer
        xor ax,bx       ;logical
ke:     hlt             ;process control

        end start
        .end
```

Test code 3:

```
.model small
.stack 64

.code
start:  xor ax,ax
        mov bx,0342h
        mov cl,02h
        mov ax,bx       ;data transfer
        mul ax,cl       ;arithmetic
        jnc ke          ;control transfer
        or ax,bx        ;logical
ke:     wait            ;process control

        end start
        .end
```

Test code 4:

```
.model small
.stack 64

.code
start:  xor ax,ax
        mov bx,0342h
        mov cl,02h
        mov ax,bx       ;data transfer
        div ax,cl       ;arithmetic
        cmp ax,bx
        je ke           ;control transfer
        or ax,bx        ;logical
ke:     esc             ;process control

        end start
        .end
```

Test code 5:

```
.model small
.stack 64

.code
start:  xor ax,ax
        mov bx,0342h
        mov cl,02h
        mov sp,10ffh
        pop ax          ;data transfer
        div ax,cl       ;arithmetic
        cmp ax,bx
        je ke           ;control transfer
        or ax,bx        ;logical
ke:     cli             ;process control

        end start
        .end
```

P1_3:

Write an assembly language program to find square and cube of a number

Ans here:

```
;final square at cx, final cube at dx after the final interrupt

.model small
.stack 64

.code
start:  mov bl,05h
        mov ax,bl
again:  mul bx
        mov cx,ax
        mul bx
        mov dx,ax
        mov ax,4ch
        int 21h
        end start
        .end
```

P1_4:

Write an assembly language programto find GCD of two numbers

Ans here:

```
;final GCD at bx after the final interrupt

.model small
```

```
.stack 64

.code
start:  mov bx,0ch
        mov ax,09h
again:  cmp ax,bx
        je exit
        jnc nxt
        xchg ax,bx
nxt:    mov dx,0h
        div bx
        cmp dx,0h
        je exit
        mov ax,dx
        jmp again
exit:   mov ah,4ch
        int 21h
        end start
        .end
```

P1_5:
Write an assembly language program to find largest and smallest number from a given set of numbers

Ans here:

```
;final largest number in cx, smallest number in dx after the final interrupt

.model small
.stack 64

.data
a db 04h,06h,01h,0eh,0ah

.code
start:  mov ax,@data
        mov ds,ax
        mov cl,04h
        lea si,a
        mov al,[si]
        mov ah,00h
        mov dx,ax
again:  inc si
        mov bl,[si]
        cmp al,bl
        jnc cont
        mov al,bl
cont:   cmp dl,bl
        jc nxt
        mov dl,bl
nxt:    dec cl
        jnz again
exit:   mov cx,ax
        mov ah,4ch
        int 21h
        end start
        .end
```

Part 2: Machine-Level Representation of Programs 32/64 bit systems

Suppose we write a C code file code.c containing the following procedure definition:

Assume file name: code.c

```
int accum = 0;

 int sum(int x, int y)
 {
        int t = x + y;
        accum += t;
        return t;
 }
```

To see the assembly code generated by the C compiler, we can use the “-S” option on the command line:

unix> ***gcc -O1 -S code.c***

This will cause gcc to run the compiler, generating an assembly file code.s, and go no further. (Normally it would then invoke the assembler to generate an objectcode file.)

The assembly-code file contains various declarations including the set of lines:

```
sum:
        pushl %ebp
        movl %esp, %ebp
        movl 12(%ebp), %eax
        addl 8(%ebp), %eax
        addl %eax, accum
        popl %ebp
        ret
```

If we use the ‘-c’ command-line option, gcc will generate both compile and assemble the code:

unix> ***gcc -O1 -c code.c***

This will generate an object-code file code.o that is in binary format and hence cannot be viewed directly. Embedded within the 800 bytes of the file code.o is a 17-byte sequence having hexadecimal representation

```
55 89 e5 8b 45 0c 03 45 08 01 05 00 00 00 00 5d c3
```

To inspect the contents of machine-code files, a class of programs known as *disassemblers* can be used. These programs generate a format similar to assembly code from the machine code.With Linux systems, the program objdump (for “object dump”) can serve this role given the ‘-d’ command-line flag:
unix> *objdump -d code.o*

Suppose in file main.c we had the following function:

```
 int main()
 {
 return sum(1, 3);
 }
```

Then, we could generate an executable program prog as follows:

unix> *gcc -O1 -o prog code.o main.c*

We can also disassemble the file prog:
unix> *objdump -d prog*

P2_1: Compare the code generate with
unix> *gcc -O1 -S -masm=intel code.c*
Ans. here:

:
P2_2: Test your own functions here:

```
 objdump -d prog

0000000000001119 <sum>:
1119: 8d 04 37  lea  (%rdi,%rsi,1),%eax
111c: 01 05 0a 2f 00 00   add  %eax,0x2f0a(%rip)  # 402c <accum>
 1122: c3                               retq

0000000000001123 <main>:
   1123: 48 83 ec 08          sub    $0x8,%rsp
   1127: be 03 00 00 00       mov    $0x3,%esi
   112c: bf 01 00 00 00       mov    $0x1,%edi
   1131: b8 00 00 00 00       mov    $0x0,%eax
   1136: e8 de ff ff ff      callq  1119 <sum>
   113b: 48 83 c4 08          add    $0x8,%rsp
   113f: c3                retq

gcc -O1 -S -masm=intel code.c

lea eax, [rdi+rsi]
add DWORD PTR accum[rip], eax
ret
```

P2_2: Compile simple C programs and look at asm content:

Ans:

```
int gcd(int x, int y) {
   return y ? gcd(y, x % y): x;
}
```

ASM code:

```
gcd:
.LFB0:
.cfi_startproc
mov eax, edi
test esi, esi
jne .L7
ret
.L7:
sub rsp, 8
.cfi_def_cfa_offset 16
mov edi, esi
cdq
idiv esi
mov esi, edx
call gcd
add rsp, 8
.cfi_def_cfa_offset 8
ret
.cfi_endproc
```

C program:

```
int main() {
   int ans = 0;
   for(int i = 0; i < 10; i++) {
       ans += i;
   }
   return ans;
}
```

ASM content

```
.file "main.c"
.intel_syntax noprefix
.text
.globl main
.type main, @function
```

```
main:
.LFB0:
.cfi_startproc
mov eax, 45
ret
.cfi_endproc
.LFE0:
.size main, .-main
.ident "GCC: (GNU) 8.2.0"
.section .note.GNU-stack,"",@progbits
```

Submission :

Submit single doc/pdf file with above answers. Course work submission through cs322.iitp@gmail.com with subject: YourrollNo_Lab4. **Due on** 31st August 2018, 5PM**.**