Savitribai Phule Pune University
Second Year of Computer Engineering
(2015 Course)

210257: Microprocessor Lab

Alka Londhe
Pimpri Chinchwad College of Engineering, Pune
alka.londhe@pccoepune.org
+91 92260 94980

Outline

1 : Count +ve & -ve nos

2 : Block transfer

3 : Hex to BCD & BCD to Hex

4 : Multiplication

5 : Far Procedure

6 : Protected Mode Registers

7 : Bubble Sort

1 : Count +ve & -ve nos

Write X86 ALP to count,

- ***positive*** and ***negative*** numbers
- from the ***array***

1 : Count +ve & -ve nos

```
Section   .data
   arr64    dq    -0000000011111111H, 22222222H,
                  -33333333H, 44444444H, 55555555H

   n        equ   5
```

```
Section   .bss
      p_count     resq     1
      n_count     resq     1
```

```
    mov     rsi, arr64
    mov     rcx, n

    mov     rbx,0           ; counter for  +ve nos.
    mov     rdx,0           ; counter for  -ve nos.

next_num:
    mov     rax,[rsi]       ; take no. in RAX
    SHL     rax,1           ; rotate left 1 bit to check for sign bit
    jc      negative

positive:
    inc     rbx             ; no carry, so no. is +ve
    jmp     next

negative:
    inc     rdx             ; carry, so no. is -ve

next:
    add         rsi,8       ; 64 bit nos i.e. 8 bytes
    dec         rcx
    jnz     next_num

    mov     [p_count], rbx     ; store positive count
    mov     [n_count], rdx     ; store negative count
```

2 : Block Transfer

Write X86 ALP to perform,

- ***non-overlapped***
- ***overlapped*** block transfer

- ***with*** and ***without*** string specific instructions

- Block containing data can be defined in the data segment.

2 : Block Transfer

```
Count       equ   5

srcblk      db    11h, 22h, 33h, 44h, 55h

dstblk      times     5  db    0
```

```
Count       equ   5

srcblk      db    11h, 22h, 33h, 44h, 55h

dstblk      times     3  db    0
```

```
print  bfrmsg,bfrmsg_len
print  srcmsg,srcmsg_len   ; Display Source Block
mov    rsi,srcblk
call   display_block

print  dstmsg,dstmsg_len   ; Display Destination Block
mov    rsi,dstblk
call   display_block


call   BT_NO

print  afrmsg,afrmsg_len
print  srcmsg,srcmsg_len   ; Display Source Block
mov    rsi,srcblk
call   display_block

print  dstmsg,dstmsg_len   ; Display Destination Block
mov    rsi,dstblk
call   display_block
```

```
BT_NO:          ; Block Transfer Non-overlapped
    mov     rsi,srcblk
    mov     rdi,dstblk
    mov     rcx,count
repeat:
    mov     al,[rsi]
    mov     [rdi],al
    inc     rsi
    inc     rdi
    loop    repeat

    ret
```

```
BT_NOS:             ; Block Transfer Non-Overlapped
        mov     rsi,srcblk
        mov     rdi,dstblk
        mov     rcx,count
        cld             ; clear direction flag
                        ; (string in normal order)
rep     movsb           ; [rdi]=[rsi] counter is rcx

    ret
```

```
BT_O:          ; Block Transfer overlapped
    mov    rsi,srcblk+4
    mov    rdi,dstblk+2
    mov    rcx,count
repeat:
    mov    al,[rsi]
    mov    [rdi],al
    dec    rsi
    dec    rdi
    loop   repeat

    ret
```

```
BT_NOS:              ; Block Transfer overlapped
       mov    rsi,srcblk+4
       mov    rdi,dstblk+2
       mov    rcx,count
       STD            ; SET direction flag
                      ; (string in reverseorder)
rep    movsb          ; [rdi]=[rsi] counter is rcx

   ret
```

3 : Hex to BCD & BCD to Hex

Write X86 ALP to convert,

- ***HEX to BCD*** (4-digit Hex no into its equivalent BCD no)
- ***BCD to HEX*** (5-digit BCD no into its equivalent Hex no)
- EXIT

Make your program user friendly to **accept the choice from user**

Display proper strings to prompt the user while accepting the input and displaying the result

3 : Hex to BCD & BCD to Hex

```
menu        db      10,"-----------Menu----------"
            db      10,"1. Hex to BCD "
            db      10,"2. BCD to Hex"
            Db      10,"3. Exit "
            Db      10
            db      10,"Enter your choice: "
menu_len    equ     $-menu
```

```
_start:
    print  menu, menu_len
    read   buf,2      ; choice + enter
    mov    al,[buf]

c1: cmp    al,'1'
    jne    c2
    call   hex_bcd
    jmp    _start

c2: cmp    al,'2'
    jne    c3
    call   bcd_hex
    jmp    _start

c3: cmp    al,'3'
    jne    err
    exit

err:   print  emsg,emsg_len
       jmp    _start
```

```
hex_bcd:
    print   h2bmsg, h2bmsg_len
    call    accept_16
    mov     ax,bx

    mov     rbx,10
back:
    xor     rdx,rdx
    div     rbx

    push    dx
    inc     byte[digitcount]

    cmp     rax,0h
    jne     back

    print   bmsg, bmsg_len
print_bcd:
    pop     dx
    add     dl,30h          ; possible digits are 0-9 so add 30H only
    mov     [char_ans],dl   ; store character in char_ans

    print   char_ans,1      ; print on screen in reverse order

    dec     byte[digitcount]
    jnz     print_bcd
    ret
```

```
bcd_hex:
    print   b2hmsg, b2hmsg_len
    read    buf,6                   ; buflen = 5 + 1

    mov     rsi,buf                 ; load bcd pointer
    xor     rax,rax                 ; sum
    mov     rbx,10
    mov     rcx,05                  ; digit_count

back1:  xor rdx,rdx
    mul     ebx             ; previous digit * 10 = ans (rax*rbx = rdx:rax)

    xor     rdx,rdx
    mov     dl,[rsi]            ; Take current digit
    sub     dl,30h              ; accepted digit is Decimal, so Sub 30H only
    add     rax,rdx

    inc     rsi

    dec     rcx
    jnz     back1

    mov     [ans],ax

    print   bmsg, bmsg_len
    mov     ax,[ans]
    call    display_16
    ret
```

4 : Multiplication

Write X86 ALP to perform **multiplication of two 8-bit hexadecimal numbers**.

- successive addition
- add and shift method

use of 64-bit registers is expected

```
SHA:
    mov     rcx,16      ; 16-bit multiplication

    Mov     ebp,0       ; answer=0

    mov     ax,[no1]
    mov     bx,[no2]

back1:
    shl     ebp,1       ; ans = ans*2
    shl     ax,1
    jnc     next1
    add     ebp,ebx

next1:
    dec     rcx
    jnz     back1

    print shamsg,shamsg_len
    mov     eax,ebp         ; display answert
    call    display_32

    ret
```

5 : Far Procedure

Write X86 ALP to find,

a) Number of Blank spaces
b) Number of lines
c) Occurrence of a particular character.

- **Accept the data from the text file.**
- The text file has to be accessed during Program_1 execution and
- write **FAR PROCEDURES** in Program_2 for the rest of the processing.
- Use of **PUBLIC** and **EXTERN** directives is mandatory.

NASM assembly program structure

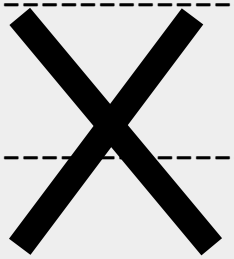
```
;file.asm
;-----------------------------------
section .data

;-----------------------------------
section .bss



;-----------------------------------
section .text
    global _start

_start:


;-----------------------------------
;Procedures
```

NASM assembly program structure: **Near** Procedure

```
;file.asm
;---------------------------------
section .data

;---------------------------------
section .bss

;---------------------------------
section .text
     global _start

_start:

          Call Procedure1
               .
               .
          Call Procedure2

     Exit
;---------------------------------
Procedure1:

     Ret
;---------------------------------
Procedure2:

     Ret
;---------------------------------
```

NASM assembly program structure: **Far** procedure

```
;file.asm
;-----------------------------
section .data

;-----------------------------
section .bss

;-----------------------------
section .text
     global _start

_start:

          Call Procedure1
               .
               .
          Call Procedure2

     Exit
;-----------------------------
Procedure1:

     Ret
;-----------------------------
Procedure2:

     Ret
;-----------------------------
```



```
;file1.asm
;-----------------------------
extern Procedure1, Procedure2

;-----------------------------
section .data

;-----------------------------
section .bss

;-----------------------------
section .text
     global _start

_start:

          Call Procedure1
               .
               .
          Call Procedure2

     Exit
;-----------------------------
```

NASM assembly program structure: **Far** procedure

```
;file1.asm
;-------------------------------
extern Procedure1, Procedure2

;-------------------------------
section .data

;-------------------------------
section .bss

;-------------------------------
section .text
     global _start
_start:

          Call Procedure1
               .
               .
          Call Procedure2


     Exit
;-------------------------------
```

```
;file2.asm
;-------------------------------
global Procedure1, Procedure2

;-------------------------------
section .data

;-------------------------------
section .bss

;-------------------------------
section .text
     global xyz
xyz:

Procedure1:

     Ret
;-------------------------------
Procedure2:

     Ret
;-------------------------------
```

NASM assembly program structure: **Far** procedure

```
$ nasm -f elf64 file1.asm

$ nasm -f elf64 file2.asm

$ ld  file1.o  file2.o  -o file

$ ./file
```

NASM assembly program structure: **Far** procedure

```
alka@ubuntu:~/ma-code/A5_Far$ nasm -f elf64 A5_file1.asm
alka@ubuntu:~/ma-code/A5_Far$ nasm -f elf64 A5_file2.asm
alka@ubuntu:~/ma-code/A5_Far$ ld A5_file1.o A5_file2.o -o A5_file
alka@ubuntu:~/ma-code/A5_Far$ ./A5_file



ML assignment 05 :- String Operation using Far Procedure
------------------------------------------------

Enter filename for string operation     : myfile.txt

Enter character to search       : !

No. of spaces are       : 0000
No. of lines are        : 0004
No. of character occurances are : 0003

Exit from program...


alka@ubuntu:~/ma-code/A5_Far$
```

Assignment 5 : Far Procedure

Write X86 ALP to find,

a) Number of Blank spaces
b) Number of lines
c) Occurrence of a particular character.

- **Accept the data from the text file.**
- The text file has to be accessed during Program_1 execution and
- write **FAR PROCEDURES** in Program_2 for the rest of the processing.
- Use of **PUBLIC** and **EXTERN** directives is mandatory.

File operations : Open & Close

```
mov   rax,2                ;open
mov   rdi,filename         ;terminated with '\0'
mov   rsi,mode             ;0-R, 1-W, 2-RW
mov   rdx,permissions      ;-rwxrwxrwxo (octal)
Syscall
```

Returns : rax = filehandle (on success)
rax = -1 (on failure)

```
mov   rax,3                ;close
mov   rdi,filehandle       ;file handle
syscall
```

File operations : Read & Write

```
mov   rax,0                    ;read
mov   rdi,filehandle           ;from file
mov   rsi,buf                  ;store in buffer
mov   rdx,buf_len
Syscall
```

```
mov   rax,1                    ;Write
mov   rdi,filehandle           ;into file
mov   rsi,buf                  ;from buffer
mov   rdx,buf_len
syscall
```

6 : Protected Mode Registers

Write X86 ALP to

- switch from real mode to protected mode
- display the values of
  - GDTR
  - LDTR
  - IDTR
  - TR
  - MSW Registers.

```
Section   .bss
    GDTR        resw   3       ; 48 bits, so 3 words
    IDTR        resw   3

    LDTR        resw   1       ; 16 bits, so 1 word
    TR          resw   1
    MSW         resw   1
```

```
    SMSW        [MSW]

    mov     rax,[MSW]
    BT      rax,0           ; Check PE bit, if 1=Protected Mode,
                            ; else Real Mode
    jc      p_mode

    Print   rmsg,rmsg_len
    jmp     next

p_mode:
    Print   pmsg,pmsg_len

next:
    SGDT        [GDTR]
    SIDT        [IDTR]
    SLDT        [LDTR]
    STR         [TR]
    SMSW        [MSW]
```

```
Print  gmsg, gmsg_len          ;GDTR
                               ; LITTLE ENDIAN SO TAKE LAST WORD FIRST

   mov    ax,[GDTR+4]          ; load value of GDTR[4,5] in ax
   call   disp16_proc          ; display GDTR contents

   mov    ax,[GDTR+2]          ; load value of GDTR[2,3] in ax
   call   disp16_proc          ; display GDTR contents

   Print  colon,1

   mov    ax,[GDTR+0]          ; load value of GDTR[0,1] in ax
   call   disp16_proc          ; display GDTR contents
```

```
   Print  tmsg, tmsg_len          ;TR (Task Register)
   mov    ax,[TR]
   call   disp16_proc
```

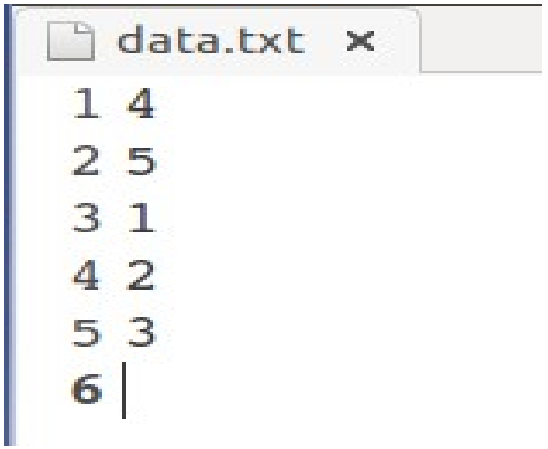
Assignment 7 : Bubble Sort

Write X86 ALP -

- To sort the list of integers in **ascending** / descending order using **bubble sort**.
- Read the input from the **text file**
- and
- write the sorted data back to the same text file.

Assignment 7 : Bubble Sort

```
buf_array:

    mov rcx,[abuf_len]
    mov rsi,buf
    mov rdi,array

next_num:
    mov al,[rsi]
    mov [rdi],al

    inc rsi      ; number
    inc rsi      ; newline
    inc rdi

    inc byte[n] ; counter

    dec rcx      ; number
    dec rcx      ; newline
    jnz next_num
ret
```



Assignment 7 : Bubble Sort

```
function bubbleSort(array a)
{
   for (i = 0 to n-1)
   {
      for (j = 0 to n-i)
      {
         if (a[j] > a[j+1])
         {
            swap(a[j], a[j+1]);
         }
      }
   }
}
```

Assignment 7 : Bubble Sort

```
function bsort(array)
{
   for (RCX = 0 to RBP-1)
   {
      for (RBX = 0 to RBP-i)
      {
         if ([RSI] > [RDI])
         {
            swap(a[j], a[j+1]);
         }
      }
   }
}
```

Assignment 7 : Bubble Sort

```
        Mov  rbp,[n]
        Dec  rbp
        mov  rcx,0                  ; i=0

oloop:  mov  rbx,0                  ; j=0
        mov  rsi,array              ; a[j]

iloop:  mov  rdi,rsi                ; a[j+1]
        inc  rdi

        mov  al,[rsi]
        cmp  al,[rdi]
        jbe  next

        mov  dl,[rdi]               ; swap
        mov  [rdi],al
        mov  [rsi],dl

next:   inc  rsi
        inc  rbx                    ; j++
        cmp  rbx,rbp
        jb   iloop

        inc  rcx
        cmp  rcx,rbp
        jb   oloop
```

data.txt ×

```
1 4
2 5
3 1
4 2
5 3
6
7
8 Sorting using bubble sort Operation s
9 Output stored in same file...
10
11 12345
```

```
accept_16:
     read buf,5          ; buflen = 4 + 1

     xor  bx,bx
     mov  rcx,4
     mov  rsi,buf
next_digit:
     shl  bx,04
     mov  al,[rsi]
        cmp       al,"0"        ; "0" = 30h or 48d
        jb        error         ; jump if below "0" to error
        cmp       al,"9"
        jbe       sub30         ; subtract 30h if no is in the range "0"-"9"

        cmp       al,"A"        ; "A" = 41h or 65d
        jb        error         ; jump if below "A" to error
        cmp       al,"F"
        jbe       sub37         ; subtract 37h if no is in the range "A"-"F"

        cmp       al,"a"        ; "a" = 61h or 97d
        jb        error         ; jump if below "a" to error
        cmp       al,"f"
        jbe       sub57         ; subtract 57h if no is in the range "a"-"f"

error:  print emsg,emsg_len     ; "You entered Invalid Data!!!"
        exit

sub57:  sub       al,20h        ; subtract 57h if no is in the range "a"-"f"
sub37:  sub       al,07h        ; subtract 37h if no is in the range "A"-"F"
sub30:  sub       al,30h        ; subtract 30h if no is in the range "0"-"9"

     add  bx,ax              ; prepare number
     inc  rsi                ; point to next digit
     loop next_digit
ret
```

```
display_16:
    mov  rsi,char_ans+3    ; load last byte address of char_ans in rsi
    mov  rcx,4             ; number of digits
    mov  rbx,16            ; divisor=16 for hex

cnt:mov  rdx,0             ; make rdx=0 (as in div instruction rdx:rax/rbx)

    div      rbx

    cmp      dl, 09h       ; check for remainder in RDX
    jbe      add30
    add      dl, 07h
add30:
    add      dl,30h        ; calculate ASCII code
    mov      [rsi],dl      ; store it in buffer
    dec      rsi           ; point to one byte back

    dec      rcx           ; decrement count
    jnz      cnt           ; if not zero repeat

    print    char_ans,4    ; display result on screen
ret
```

Have a Nice Day!