Exam 3

EBP = base pointer or frame pointer

It holds the base address of the base of the stack frame

EBP does not change value during the procedure

EBP must be restored to its original value when a procedure returns

Stack Parameters : more convenient than register parameters

pass argument by value on stack

push argument values on stack

use 32 bit values to keep the stack "aligned"

call the procedure

return value in EAX if any

remove arguments from the stack

pass argument by reference

push the offsets (address) of arguments on the stack

call the procedure

return value in EAX if any

remove arguments from the stack

cannot push 8 bit values on stack
Pushing 16 bit values may cause page fault
or ESP Alignment problem
use movzx or movsx to expand smaller
argument into 32 bit value

Push high order value on the stack first
results in little-endian ordering of data

```
push  EBP              ; preserve the base pointer
mov   EBP, ESP         ; create a stack frame

sub   esp, 8           ; create 2 dword variables

mov   dword [ebp-4], 5 ;  x = 5
mov   dword [ebp-8], 10 ; y = 10

mov   eax, [ebp+8]     ; eax = arg 1
mov   ecx, [ebp+12]    ; ecx = arg 2

; arg 1 = ebp+8 and the last to push
; before calling the procedure
```

pass argument by value vs by reference

arg = value

or

arg = address

LEA instruction

store the address of local variable into a reg instead of its value

```
; [ebp - 4] = 5
mov  edi, [ebp - 4]   ; edi = 5
lea  edi, [ebp - 4]   ; edi = address of
                      ;   [ebp - 4]
```

Enter instruction

create stack frame for a called procedure

push EBP on the stack

set EBP to the base of the stack frame

reserve space for local variables

```
enter 4, 0   =>   push  ebp
                  mov   ebp, esp
                  sub   esp, 4
```

LEAVE instruction

terminate the stack frame for a procedure

```
leave   =>   mov  esp, ebp  ; free local
             pop  ebp       ; space
```

```
;  must  4 * number of arg pushed
;  after  the procedure returns
```

Recursion

A procedure calls itself

Procedure A calls Procedure B then
B calls A

summation of 1 to 10

```
sum:
   push ebp
   mov  ebs , esp

mov   ecx , [ebp + 8]       ; ecx = arg  = 10
mov   eax , 0               ; eax = 0

.if :
 cmp    ecx , 0              ; check value in arg
 je     .endif               ; end if arg = 0
 dec    ecx                  ; decrement value
 push   ecx                  ; push value as parameter
 call   sum                  ; recursive call
 add    eax , [ebp + 8];     add arg to eax
 .endif :

 leave
 ret
```

MOVSB, MOVSW, MOVSD instructions

copy data from memory location pointed to by ESI to the memory location pointed by EDI

copy data from ESI to EDI

ESI and EDI are automatically incremented or decremented

movsb inc/dec by 1

movsw inc/dec by 2

movsd inc/dec by 4

The direction flag controls the inc or dec of ESI and EDI

DF = clear = 0 : inc ESI and EDI

DF = set = 1 : dec ESI and EDI

CLD : clear DF

STD : set DF

REPEAT Prefix

REP can be inserted before MOVSB, MOVSW, MOVSD

ECX controls the repetitions

CMPSB, CMPSW, CMPSD instruction

compare a memory operand pointed to by ESI to a memory operand pointed to by EDI

compare ESI to EDI

SCASB, SCASW, SCASD instructions

compare a value in AL/AX/EAX to a byte, word, dword, respectively, address by EDI

used for:

search for a specific element in a string or array

search for the first element that does not match a given value

eg.

```
str: "ABCDEF", NULL
str_sz: equ $ - str

    mov     edi, str            ; EDI = str
    mov     al, 'F'             ; AL = 'F'
    mov     ecx, str_sz         ; ECX = str_sz
    cld                         ; forward
    repne   scasb               ; repeat while not equal

    dec     edi                 ; EDI points to 'F'
```

STOSB, STOSW, STOSD instructions

store the contents of AL|AX|EAX respectively, In memory at the offset pointed to by EDI
store the value in AL|AX|EAX into EDI

```
str_sz: equ  255
str:    resb  str_sz    ; fill the string with NULL

mov   al, NULL          ; AL = NULL
mov   edi, str          ; EDI = str
mov   ecx, str_sz       ; ECX = str_sz
cld                     ; forward
rep   stosb             ; fill str with NULL
```

LODSB, LODSW, LODSD instructions

load a byte, word, dword from memory at ESI into AL, AX, EAX, respectively

eg.
```
array:      db   1, 2, 3, 4, 5
array_sz:  equ $ - array

  mov     esi , array          ; ESI = array
  mov     edi , array          ; EDI = array
  mov     ecx, array_sz        ; ECX = array_sz
  mov     dl , 5               ; DL = 5

  cld

  .while
  lodsb                        ; copy ESI into AL
  mul      dl                  ; AL multiple by DL
  stosb                        ; store AL into EDI
  loop   .while                ; loop while ECX > 0
  .wend
```

struct is a class
a design element
not an object

struct
In NASM, struc is a preprocessor macro.
a collection of data types

string
is an array of char
NULL terminate string
the NULL marks the end of the string
can search for the NULL to find out the
length of the string

Binary Multiplication

128 1000 0000 000000

33 000~~10~~ ~~0001~~

0000 0000 0000 0000

0000 0000 1000 0000

0001 0000 0000 0000

0001 0000 1000 0000

4096 128

$$\begin{array}{r} 4096 \\ +\ 128 \\ \hline 4224 \end{array}$$