Assembly

- No class next Thursday

- findChar(&str, bChar); int & (returns address)

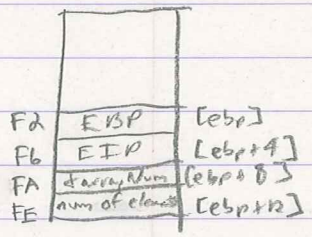
```
char byte 'c'
strMsg byte "John went to town", 0
charaddress dword ?
;charaddress = findChar(&strMsg, char
mov ax, 0
mov al, char
push ax
push offset strMsg
call findChar
add esp, 6
mov char, eax          ; then check for -1

findChar proc Near32
    push ebp           ; preserving base ptr
    mov ebp,esp        ; set new stack ptr
    push ebx; push ecx;   preserve registers
    mov ebx, [ebp+8]   ; ebx points to strMsg
    mov eax, -1        ; assume no char found
    mov cx, [ebp+12]   ; cl = character we are looking for
findCharLoop:
    cmp byte ptr [ebx], 0   ; is this end of string
    je finish          ; leave loop
    cmp [ebx], cl      ; is this the character
    je finish          ; character found
    inc ebx            ; points to next char
    jmp findCharLoop
```

```
found:
    mov eax, ebx        ; eax points to character

finished:
    pop ecx             ; restore register
    pop ebx             ; restore register
    pop ebp             ; restore registers
    ret                 ; return
ENDP

    arrayNum word 50 dup(?)
    dSum  dword ?
    ; dSum = sum (arrayNum, wSize)
    mov ax, 50          ; 50 values to add
    push ax
    push offset arrayNum
    call sum
    mov dSum, eax
    add esp, 6
```




```
Sum proc Near32
    push ebp
    push ebp, esp
    push ebx; push ecx; push edx;
    xor eax, eax        ; total = 0
    mov ebx, [ebp+8]    ; ebx → element in the array
    mov ecx, 0          ; make sure all bytes of ecx are 0
    mov cx, [ebp+12]    ; number of elements
stloop:
    xor edx, edx        ; clear edx
    movsx edx, word ptr [ebx]   ; 2 byte value to 4 byte value
    add eax, edx                ; add
    add ebx, 2                  ; increment pointer
```

```
    loop sLoop

    pop edx
    pop ecx
    pop ebx
    pop ebp
    ret
sum ENDP
```

- putchar (dword) : void
  → display a char on the screen

- getchar ( ) : dword
  → get a character from keyboard (doesn't display)

```
call getchar
mov char, al
```

- getcharecho ( ) : dword
  → same as getchar() and displays the char

```
call getchar
mov char, al
```

Assembly

10.13.11

- get a character
  check to see if a char is = backspace

- to erase a letter, display a backspace and then a space, & then another backspace, reduce number of characters, then get another character

- if the limit has been reached, get another char
  if not, display char then increment count

- if carriage return, done with string
  compare al to 0Dh or 13d (did they hit enter?)

- no leading zeros, can only enter + or - before the number

- Allocating local storage in a method

```
int funct(int iVal, int iNum)
{
    int s, t;
    [code]
    return;
}
```

OR

```
funct proc Near32
    push ebp
    mov ebp, esp
    sub esp, 8
    s equ [ebp-4]
    t equ [ebp-8]
    iVal equ [ebp+8]
    iNum equ [ebp+12]
```

```
mov eax, iVal
add eax, iNum
mov s, eax
. . . . . .
add esp, 8
pop ebp
ret
```

```
int f(int iNum, short sVal)
{
    int t;
    short s;
    int v;
}
```

```
f proc Near32
    push ebp
    mov ebp, esp
    sub esp, 10
    t equ [ebp-4]
    s equ [ebp-6]
    v equ [ebp-10]
    ...
    add esp, 10
    pop ebp;  ret;
f endp
```

AND op1, op2

clears OF & CF

mostly used for turning off bits using a mask (binary)

sets sign flag if MSB is 1, parity is set

for odd parity - 11100101 - PF=0

expect # of bits to be odd 10101010 - PF=1

CA - PF=0
A7 - PF=0
05 - PF=1

and eax, 0FF0E000h

- OR clears OF, CF
  use OR for turning on bits

- XOR clears OF, CF
  op1 XOR op1 = 0

stc - set parity flag
clc - clear parity flag

xor ax, ax is faster than mov ax, 0

- TEST op1, op2
  op1 is not effected but has same result as AND
  set flags

- - - - - - - - -
  ↑
  is it on

jump to bit6ON ; if 1
jump to bit6OFF ; if 0

test room, 01000000b
jz bit6OFF
jnz bit6ON

jump to bit6 or bit2 on

```
test  room, 01000000b
jnz   bit6orbit2on
test  room, 00000100b
jnz   bit6 or bit2on
```

Assembly

- "above" "below"

ja , jb (unsigned)
jnb (unsigned)

jg , jl - signed

jc here (here = 2 bytes)

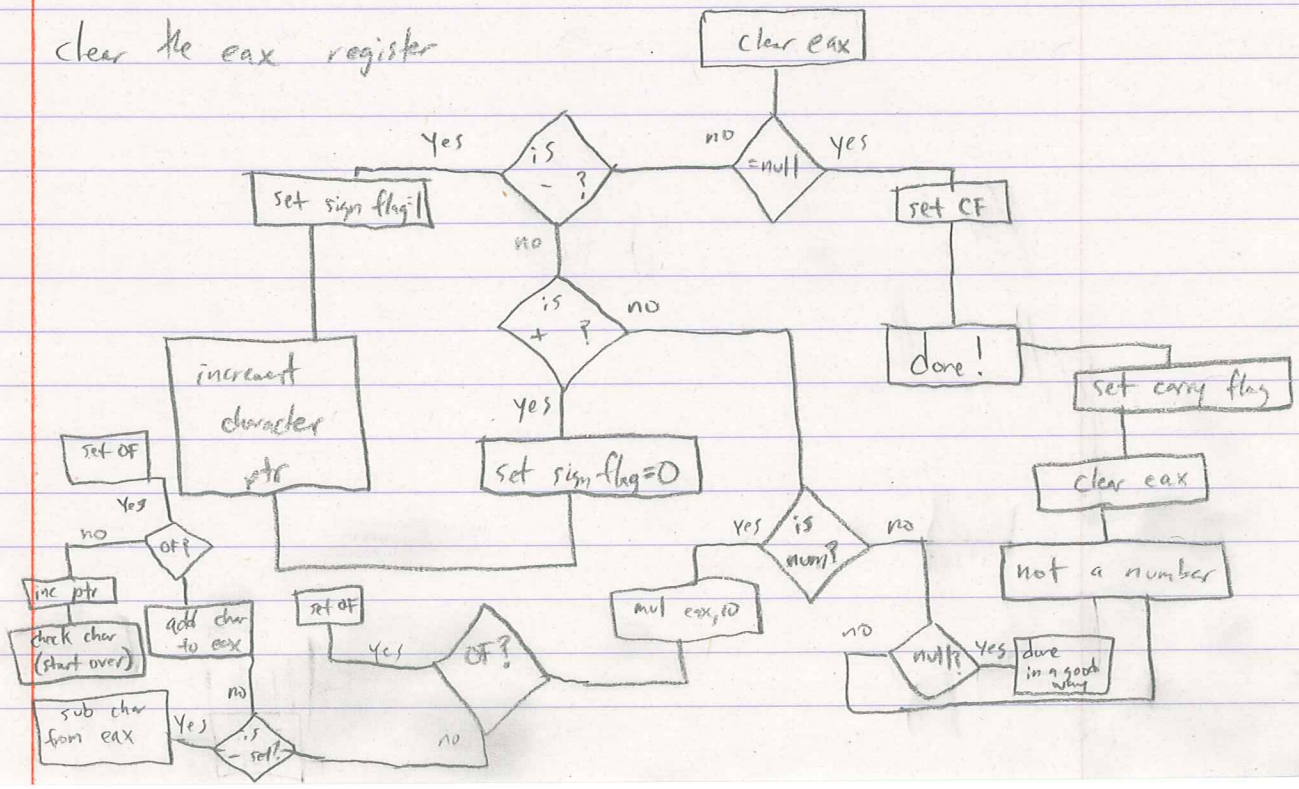
here contains a 2 byte value added to the current memory location
here is signed, so offset can be subtracted from current location

- Conversion Routines

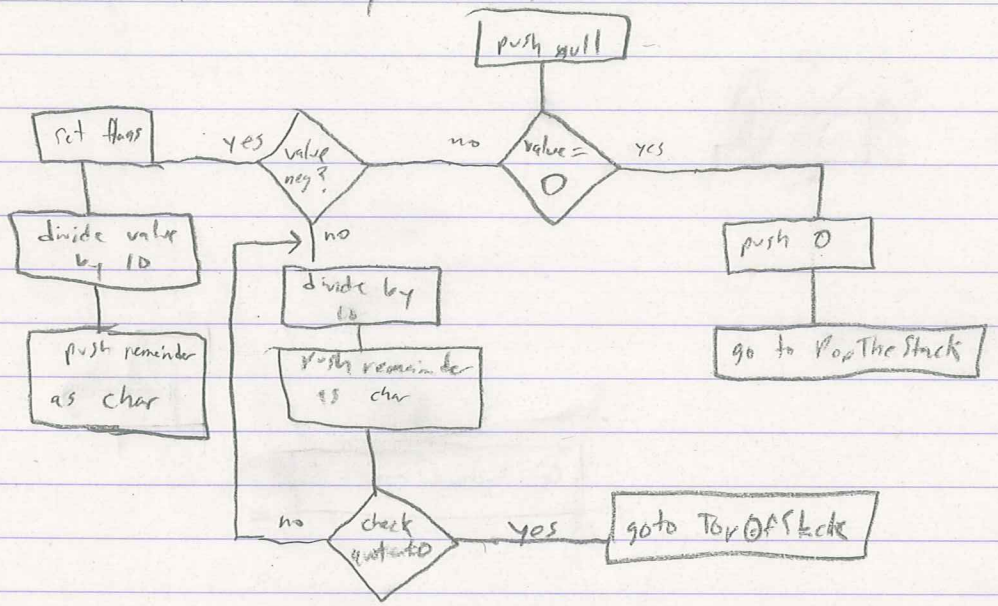
getstring

for a number 324, the string is 33323400

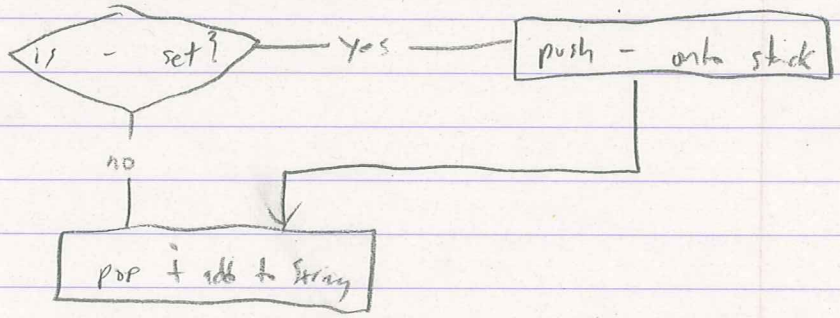
clear the eax register



- convert from binary to ASCII



- PopTheStack



Assembly

- ascint32

```
push offset strNum
call ascint32
add esp, 4
mov iVal, eax
```

- convasc macro strNum, iVal

```
push eax
push offset strNum
call ascint
add esp, 4
mov iVal, eax
pop eax
endm
```

-
```
iVal1 word ?
iVal2 word ?
strInput1 byte 10 dup(?)
strInput2 byte 10 dup(?)

convasc strInput1, iVal1
convasc strInput2, iVal2
```

- macro must appear somewhere before it is called

```
exitProcess ...
```

MACROS

```
.data
.code
```

- write a putc macro

```
putc    macro    bChar

        push eax
        xor eax, eax;
        mov al bChar
        push eax
        call putchar
        add esp, 4
        pop eax
endm
```

- write a getch macro

```
getch   macro   bChar

        push eax
        xor eax, eax
        call getchar
        mov bChar, al
        pop eax
endm
```

- write a getche macro

```
getche  macro   bChar

        push eax
        xor eax, eax
        call getcharecho
        mov bChar, al
        pop eax
endm
```

- write puts macro

```
.puts macro strMsg

  push eax
  push ecx → mov eax, offset strMsg
  loop:
    mov al, eax + ecx
    cmp al, 0
    je escapeloop
    putc al
    inc ecx
    jmp loop
  escapeloop:
  pop ecx
  pop eax
endm
```

- write code expansion for puts strPrompt1 + puts strPrompt2

the macro expands the labels - local

use:

```
puts macro strMsg
      local loop, escapeLoop
```

- addbytes macro x, y, z

```
push eax
mov al, x
add al, y
add al, z
pop eax
endm
```

- addshorts macro x, y, z

```
push eax
mov ax, x
add ax, y
add ax, z
pop eax
endm
```

- Recursion

$f(0) = 0$

$f(n) = n + f(n-1)$

C code:

```
int f(n)
{
    if(n==0)
        return 0;
    else
        return (n+ f(n));
}
```

```
int f(int n)
{
    if(n==0)
        return 0;
    else
    {
        t=f(n-1);
        v= n+t;
        return v;
    }
}
```

```
y=g(;X);
assembly =      push ;X
                call g
                add esp,4
                mov y, eax
```

```
f proc Near32
push ebp
mov ebp, esp
sub esp, 8          <-------- push ebx
t EQU [ebp-4]
v EQU [ebp-8]
mov eax, 0
cmp dword ptr [ebp+8], n
je done
```

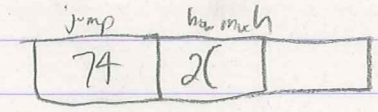
```
    mov ebx, n
    dec ebx
    push ebx
    call f
    add esp, 4
    mov t, eax

    add eax, n
finish:
    pop ebx
    add esp, 8
    pop ebp
    ret

f proc Near32
    push ebp
    mov ebp, esp
    n equ [ebp+8]
    push ebx
    mov eax, 0
    cmp dword ptr n, 0
    je done
    mov ebx, n              ; get to N-1
    dec ebx
    push ebx                ; get ready to call function
    call f
    add esp, 4
    add eax, n
done:
    pop ebx
    pop ebp
    ret
```

Assembly

- je DONE 04004040



EIP = 04004042 before exec

74 is op code for jmp

04004042
+ 2C
DONE: 04004046E

- je DONE 04004040

EIP = 04004042
+ FFFFFFCE
DONE: (1) 04004010

- TEST
- no flags are set for the NOT instruction
- Recursion Review

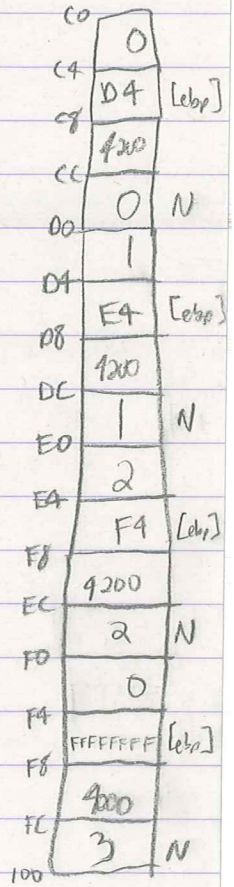
```
; f(n) = n + f(n-1)                    ; 00401100
   f proc near 32
      push ebp
      mov ebp, esp
      n equ [ebp+8]
      push ebx
      mov eax, 0
      cmp dword ptr n, 0                ; N=0 ?
      je done
      mov ebx, n
      dec ebx                           done:
      push ebx                                pop ebx
      call f                                  pop ebp
      add esp, 4                              ret
      add eax, n                        f endp
```

- 1 + 2 + 3

```
n dword 3
; sum = f(n),
push n
call sum
add esp, 4                    ; 0x00401000
mov sum, eax
```

1. esp = 100h esp=00401100 ebp = F4 ebx = 0 eax = 0

FC	DC	4200	E4	3	0
F8	E0	4100	D4	2	0
F4	E4	-	C4	2	0
F0	E8	-	D4	1	1
EC	EC	4200	E4	1	3
E8	F0	4100	F4	0	6
E4	F4	-	-1	0	
E0	F8	4200		1	
DC	FC	4100		1	
D8	100	4200		0	
D4		-			
D0		-			
CC		4200			
C8		4000			
C4					
C0					
C4					
C8					
CC					
D0					
D4					
D8					



```
; f(n) = n * f(n-1)
f proc near 32
```

Assembly

-- Quiz 11 Review

- g, l = signed a, b = unsigned
  ja = jump when above (unsigned)

- next exam - Thursday before Thanksgiving - Nov. 17

-- Lecture

.if - allows comparisons (still cannot do mem to mem)

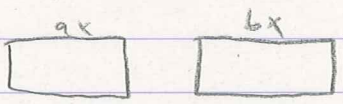
```
.if  condition
     true statements
[ .elseif  conditin
          statements ]
[ .else
          statements ]
.endif
```

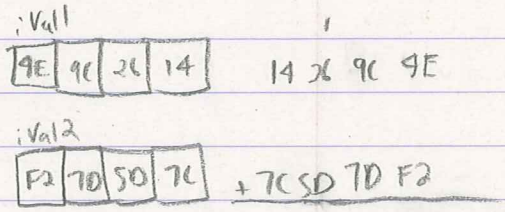
```
:Val1   dword   ?
:Val2   dword   ?

mov. eax, .Val1
.if  eax, :Val2
     sub eax, :Val2

.else
     add eax, :Val2
.endif
mov :Sum, eax
```

- .WHILE condition
  .ENDW
- .BREAK (leave loop)
- .CONTINUE (stops loop and goes back to start of loop)
- .REPEAT
  .UNTIL condition (do while)
- .if eax > ;Val1
  mov ebx, 5
  .else
  mov ebx, 10
  .endif
- .if using .if, else, etc, use sword, sdword, sqword so that they are signed
- when doing registers comparisons, it always does unsigned comparisons
- ;Sum = ;Val1 + ;Val2






```
        mov ax, word ptr ;Val1      ; ax = 9C 9E
        mov bx, word ptr ;Val2      ; bx = 7D F2
1.      add ax, bx
change dx to cx   mov dx, word ptr ;Val1+2   ; dx = 14 26
change cx to bx   mov cx, word ptr ;Val2+2   ; cx = 7C 5D
1.      adc ax, bx
!1      add ax, bx                  ; ax = 1A40
!1      adc dx, cx                  ; dx = dx + cx + CF (9084)
                                    ; dx:ax = 9084:1A40
```

- 1. add also only use ax, bx

```
num1    word   20   dup(?)
num2    word   20   dup(?)
num3    word   20   dup(?)
 clc
xor   cx, cx                    ;clear cx
.WHILE    cx < lengthof num1    ; while not done
    mov  ax, word ptr num1+cx
    mov  bx, word ptr num2+cx
    adc  ax, bx
    mov  num3+cx, ax
    add  cx, type num1
.ENDW
```

- Structures
- used to define our own storage (end with name ends)

```
Employee  struct
    id      byte  9  dup(?)
    EmpName byte  20 dup(?)
    age     word  ?
    salary  dword ?
ends
```

```
myEmp    Employee   <>                ; declare myEmp with no initial vars
myEmp2   Employee   <"111223456">     ; declare myEmp2 with id param
myEmp3   Employee   <,,25>            ; declare myEmp3 with age param
Employees Employee  20 dup(<>)        ; array of 20 with no params
```

- ALIGN dword

```
myEmp Employee <>          ; aligns myEmp on dword boundary
```

- "Aligned Employee"

```
Employee struct
    id byte 9 dup(?)
    EmpName byte 20 dup(?)
    ALIGN word
    age word ?
    ALIGN dword
    salary dword ?
Employee ends
```

- `mov ax, myEmpl.age`
- `mov eax, myEmpl.salary`
- `Employees Employee 20 dup(< >)`

```
mov esi, 0              ; esi points to each structure in array
xor eax, eax            ; clear eax, eax = sum
stloop:
    add eax, Employees[esi].salary      ; add salary
    add esi, type Employee              ; go to next employee
    loop stLoop
```

- with in a method

```
mov ebx, offset Employees       ; ebx → array element 0
xor eax, eax
stloop:
    add eax, (Employee ptr [ebx]).salary
    add esi, type Employee
    loop stloop
```

- node structure
  left dword ?
  value dword ?
  right dword ?
node ends

bTree label byte
  node <10, 20, 17>
  node <36, 25, 44>
  node <20, 19, 30>

- Quiz on Tuesday

Assembly

- Employee struct
  - sName byte 10 dup(?)
  - age word 18
  - salary dword ?
    - dword ?
    - dword ?
    - dword ?
    - dword ?

  or salary dword 5 dup(?)

  ends

lengthof Employee.salary = 1 lengthof Employee.salary = 5

- find average

```
tom employee <>
xor eax, eax                ; total of all salaries
mov ecx, lengthof tom.salary
stLoop:
    add eax, tom.salary[esi]
    add esi, type tom.salary
    loop stLoop
```

xor esi, esi

- tom - send to method

```
mov ebx, offset tom         ; ebx -> tom
xor eax, eax
mov ecx, lengthof tom.salary
stLoop
    add eax, (employee ptr [ebx]).salary[esi]
    add esi, 4
    loop stLoop
```

```
node  struct
    left  dword  ?
    value dword  ?
    right dword  ?
ends
```

- preorder, in order, postorder
  - pre - get value, process left, process right
  - in - process left, get value, process right
  - post - go left, go right, get value

```
nodes  struct  100 dup(< >)
start  dword  -1
avail  dword  nodes
```

	left	value	right
100			10C
10C			118
118			124
124			130
130			

```
mov ecx, 99
mov ebx, offset nodes
stLoop:
mov (node struct ptr [ebx]).right, esi
add ebx, 12
add esi, 12
loop stLoop
mov (node struct ptr [ebx]).right, -1
```

```
mov esi, ebx
add esi, 12
```

- as each node is filled, right is copied to avail

```
dataNode struct
info  byte  dup(81)
next  dword (?)
ends
```

dataNodes struct 100 (< >)

	info	next
100	null terminated strings	700
200		800
300		
400		600
500		
600		100
700		200
800		-1

```
start   dword 400

        mov ebx, start
stloop: cmp ebx, -1  je done
        puts ebx
        puts offset CRLF
        mov ebx, (dataNode ptr [ebx]).info
        jmp stloop
done:
```

- Shifts
  - SHL - shift left
  - SHR - shift right

destination, count

count can be immediate or use CL

SHL al, 1 CF=0 AL=10101010

result is CF=1 AL=01010100

OF bit means only if shifting 1 bit
in this case, OF was set, but shifting 2 doesn't matter

if AL = 11001010, then 1 shift CF=1, OF=0

- SHR

```
x   word  4CDAh
shl x, 1        0100110001010100 → 1001100010101000
                                   9844
```

sal - shift arithmetic left (same as shl)
sar - shift arithmetic right (not same as shr)
sar saves sign bit when moved, shr doesn't

- SHR doesn't affect OF

Assembly

- No class on Tuesday before Thanksgiving

- SHL - packing data
  - ax = 0F 0C

```
shl ah, 4     ax=F0 0C
or al, ah     al=FC
```

- SHR al=FC unpacking

```
mov ah, al
SHR ax, 4
SHR al, 4
```

ax

FC	FC
0F	CF
0F	0C

- SAL, SHL — same, multiplies by 2 ax = 00 0F (15) shl = 00 1E (30)

- mov ax, -15 ax = FF F1 (sar means sign is kept)
  sar ax, 1 ax = FF F8

- mov ax, -15
  mov bl, 2
  idiv bl ah = FF al, F9
- sar is going to be larger than idiv if its not equal
- shifting affects all flags

Assembly

11.15.11

-- Review for Exam 2

7. create two dimensional table and return the value

```
; int findRecord( int*, int )
findRecord proc Near32
    push ebp
    mov ebp, esp
    push ebx   push esi
    mov esi, [ebp+12]
    mov ebx, [ebp+8]        ; points to table
    mov eax, -1
stloop:
    cmp dword ptr [ebx], -1     ; end of table?
    je done
    cmp [ebx], esi              ; find key?
    je found
    add ebx, 8                  ; ebx points to next record
    jmp stLoop

found:
    mov eax, [ebx+4]
done:
    pop ebx
    pop esi
    pop ebp
    ret
```

8.
```
push iValue
push offset iTableIndex
call findRecord
add esp, 8
cmp eax, -1
jg displayfound
puts strError
jmp around

displayfound:
puts strFound
```

10.
```
mov ax, word ptr iD
mov dx, word ptr iD+2
idiv sB
cwde
cdq
mov ebx, offset qA
mov [ebx], eax
mov [ebx+4], edx
```

15. ; int myFun(int N)

```
myFun PROC
    push ebp
    mov ebp,esp
    sub esp,12
    mov eax,1                ; assume N is 1
    N equ [ebp+8]            ; N
    cmp dword ptr N,0        ; if =, done
    je done                  ; return 1
    cmp dword ptr N,1
    je returnTwo             ; return 2

    mov esi, N
    dec esi                  ; esi = N-1

    t equ [ebp-4]
    v equ [ebp-8]
    z equ [ebp-12]

    push esi
    call myFun
    add esp,4
    mov t, eax
    dec esi
    push esi
    call myFun
    add esp,4
    mov v, eax
    add eax, T
    jmp done

returnTwo:
    mov eax, 2

done:
    add esp,12
    pop ebp
    ret
```

- mov ebx, offset strOut =
  lea ebx, strOut

21. ① 00 1004CD CS
+ FFFFFFCS
00100492