Assembly

- type - # of bytes - the type that is assigned
  lengthof - # of different values
  sizeof - total # of bytes used

bVal Byte 20 dup(0) type=1, length=20, sizeof=type x lengthof

mov ax, type bVal ax: 00 01
mov ax, lengthof bVal ax: 00 14
mov ax, sizeof bVal ax = 00 14

- label - doesn't have location etc., give a location a different type

dT dword ? loc: 00 4C

- jump [label] - unconditional jump - "go to"

```
mov eax
add eax, dVal
jump end_of_program
```

- loop - works with ecx
  1) ecx is decremented by 1
  2) if (ecx != 0), then jump to the label

- wVal word 15, 20, 32, -1, 6

```
mov ecx, lengthof wVal;      ecx = 5
mov eax, 0;                  set sum to 0;
mov ebx, offset wVal;        ebx = beginning of array;
```

```
stloop:
    add  eax, word ptr [ebx];      sum += element at ebx
    add  ebx, 2;                   ebx points to next element
    loop stloop;
```

-
```
strOne byte "Hello, there!", 0
strTwo byte 40 dup(?)

mov ecx, sizeof strOne;
mov eax, 0;
mov ebx, offset strOne;        mov edx, offset strTwo;
stloop:
    mov al, byte ptr [ebx];
    mov strTwo, al;
    add ebx, type strOne;
    add edx, type strTwo;
loop stloop
```

- use ebx, esi, & edi for pointer registers
- inc & dec is a little faster than add

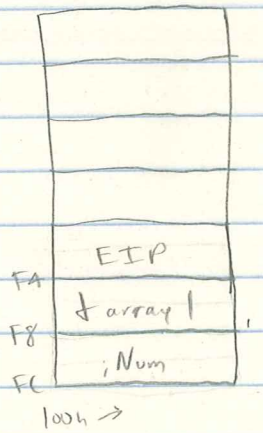
- [label] proc
    push ebp
    move ebp, esp

    pop ebp
    ret
    end proc
; this is the shortest proc, in Java, this is { }

; short function (short, short)
  wVal1 word ?
  wVal2 word ?
  wVal3 word ?
; wVal3 = function (wVal1, wVal2)
  push wVal2
  push wVal1
  call function
  mov wVal3, ax ; or mov wVal3, word ptr eax
  add esp, 4
; sVal = f1(iVal, sVal1, sVal2)
  push sVal2
  push sVal1
  push iVal
  call f1
  mov sVal, ax
  add esp, 8

- ;Total= SumArray(&array1, ;Num) ; C syntax
  push ;Num
  push offset array1
  call SumArray
  mov ;Total, eax
  add esp, 8

1) EIP is pushed onto stack
2) EIP is loaded with address of procedure
   when the function is entered, the old EIP value is on the top of the stack
3) when the function is entered, ebp is pushed onto stack
   - the programmer must push & pop the stack equal times by the end of the method
4) ret copies what is on top of stack to EIP



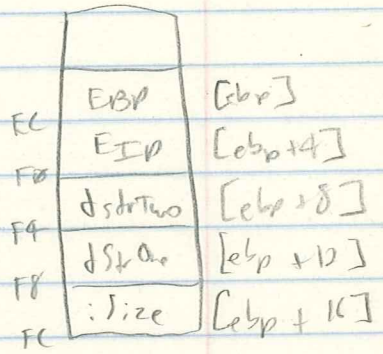
- sZ = addshorts(sX, sY)

```
wX  word  27
wY  word  16
wZ  word  ?
; wZ = addshorts(wX, wY)
    push wY
    push wX
    call addshorts
    mov wZ, eax
    add esp, 4
addshorts proc
    push ebp
    mov ebp,esp
    mov ax, [ebp+10]
    add ax, [ebp+8]
    pop ebp
    ret
```

- strCopy (dstOne, dstTwo, iSize)

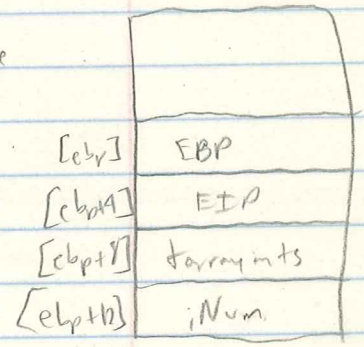
```
    push iSize
    push offset strOne
    push offset strTwo
    call strCopy

    add esp, 8;          ; restore stack pointer
```

```
strCopy proc
    push ebp            ; preserve base pointer
    mov ebp, esp        ; new stack frame
    push eax; push ecx; push edi; push esi;   preserve registers
    mov ecx, [ebp+16]   ; ecx = # of bytes copied
    mov esi, [ebp+12]   ; esi → source str
    mov edi, [ebp+8]    ; edi → dest str
stLoop:
    mov al, [esi]
    mov [edi], al
    inc esi
    inc edi
    loop stLoop;
```



- int AddsUp(&arrayints, iNum) ; write procedure

```
AddsUp proc
✓   push ebp
✓   mov ebp, esp
✓   mov ecx, [ebp+12]
    mov esi, [ebp+8]   ; ✓mov eax, 0;
stLoop:
    mov ebx, [esi]      ; move source into ebx
    add eax, ebx        ; add eax & ebx
    add esi, type arrayints   ; increment esi by # of bytes in type
    loop stLoop         ; restart loop if ecx ≠ 0
```



would also want to preserve ebx & ecx after AddsUp proc

- Compare

cmp op1, op2

what happens is op1 - op2 so that flags get set

still cannot compare memory to memory

```
strName byte "Hello, there!", 0
  mov ebx, offset strName   ; ebx -> array

  cmp byte ptr [ebx], 0
  je finished
```

(je = jump equal; jne = jump not equal; jl = jump less than)

je tests the zero flag

-

```
sub count, 1
jz done
```

jz tests the zero flag

je = jz but use which one makes sense