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)

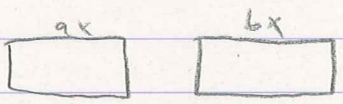
```
.if  condition
     true statements
[ .elseif  conditin
          statements ]
[ .else
          statements ]
.endif
```

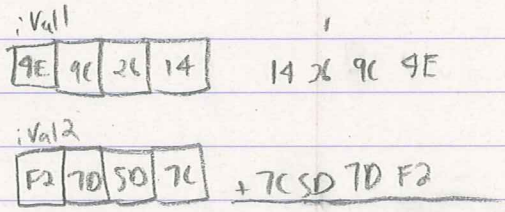
```
:Val1   dword   ?
:Val2   dword   ?

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