Juan Sebastian Cardona Sanchez

Ingenieria de sistemas

Arquitectura de software

(3.3 Section Review)

What types of files are produced by the assembler?

Assembler produces an object file, a machine-language translation of the program. And optionally, it produces a listing file. (.OBJ) and (.LST) respectively

(True/False): The linker extracts assembled procedures from the link library and inserts them in the executable program.

This is true because the linker copies any required procedures from the link library, combines them with the object file, and produces the executable file.

(True/False): When a program's source code is modified, it must be assembled and linked again before it can be executed with the changes.

Yes, because by modifying the code it could have been modify the numeric address of each instruction, the machine code bytes of each instruction (in hexadecimal), and a symbol table

Which operating system component reads and executes programs?

The operating system loader utility reads the executable file into memory and branches the CPU to the program's starting address, and the program begins to execute

What types of files are produced by the linker?

The executable file (.EXE)

(3.5.5 Section Review)

1. Declare a symbolic constant using the equal-sign directive that contains the ASCII code (08h) for the Backspace key.

BackspaceKey = 08h

2. Declare a symbolic constant named SecondsInDay using the equal-sign directive and assign it an arithmetic expression that calculates the number of seconds in a 24-hour period.

SecondsInDay = (60d*60d*24d)

3. Write a statement that causes the assembler to calculate the number of bytes in the following array, and assign the value to a symbolic constant named ArraySize:

myArray WORD 20 DUP(?)

In this case I would be creating a 30 byte space in which none is initialized, the space is available but has nothing

ArraySize = ($ - myArray)

4. Show how to calculate the number of elements in the following array, and assign the value to a symbolic constant named ArraySize:

myArray DWORD 30 DUP(?)

In this case I would be creating a 30 byte space in which none is initialized, the space is available but has nothing

If the amount of elements we could not but if it is the amount of space yes, and this we would do with ArraySize = ($ - myArray)/4

5. Use a TEXTEQU expression to redefine "proc" as "procedure."

- procedure TEXTEQU <proc>

6. Use TEXTEQU to create a symbol named Sample for a string constant, and then use the symbol when defining a string variable named MyString.

```
.386
.model flat, stdcall
Sample TEXTEQU <"one">
.data
myString = Sample
```

7. Use TEXTEQU to assign the symbol SetupESI to the following line of code: mov esi, OFFSET myArray

```
.386
.model flat, stdcall
SetupESI TEXTEQU <mov esi,OFFSET myArray>
```

Explain why the term assembler language is not quite correct.

When the term assembler is used to refer to the language itself, it appears incorrect as the assembler is the program that translates the code into machine code

Explain the difference between big endian and little endian. Also, look up the origins of this term on the Web.

Big-endian is an order in which the "big end" (most significant value in the sequence) is stored first (at the lowest storage address). Little-endian is an order in which the "little end" (least significant value in the sequence) is stored first.

4.1.10 Section Review

1. What are the three basic types of operands?

There are three basic types of operands:

• Immediate—uses a numeric literal expression

• Register—uses a named register in the CPU

• Memory—references a memory location

2. (True/False): In a MOV instruction, the second operand is known as the destination operand.

The MOV instruction has the next structure

- MOV destination, source

3. (True/False): The EIP register cannot be the destination operand of a MOV instruction.

As the reading says, MOV is very flexible in its use of operands, as long as the following rules are observed:

- The instruction pointer register (IP, EIP, or RIP) cannot be a destination operand

4.2.8 Section Review

Use the following data for Questions 1-5:

```
.data
val1 BYTE 10h
val2 WORD 8000h
val3 DWORD 0FFFFh
val4 WORD 7FFFh
```

1. Write an instruction that increments val2.

Inc val2

2. Write an instruction that subtracts val3 from EAX.

sub eax, val3

3. Write instructions that subtract val4 from val2.

mov eax, val2
sub eax, val4
mov val2, eax

4. If val2 is incremented by 1 using the ADD instruction, what will be the values of the Carry and Sign flags?

CF = 0, SF = 1

5. If val4 is incremented by 1 using the ADD instruction, what will be the values of the Overflow and Sign flags?

OF = 1, SF = 1

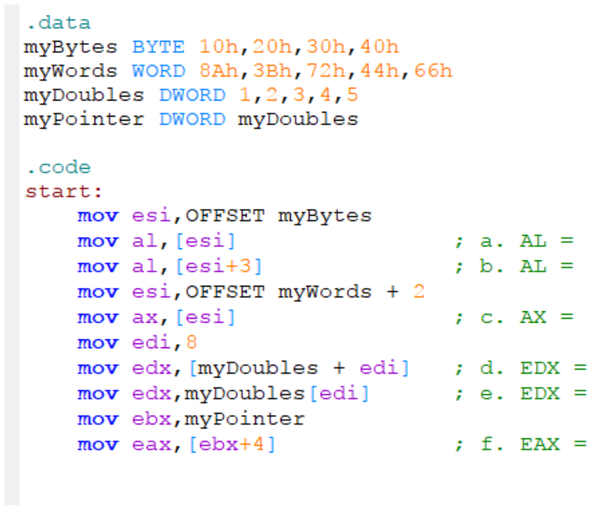
6. Where indicated, write down the values of the Carry, Sign, Zero, and Overflow flags after each instruction has executed:

```
mov add ax,7FF0h
add al,10h    ; a. CF =      SF =     ZF =       OF =
add ah,1      ; b. CF =      SF =     ZF =       OF =
add ax,2      ; c. CF =      SF =     ZF =       OF =
```

a.	CF = 1	SF= 0	ZF = 1	OF= 0
b.	CF = 0	SF= 1	ZF = 0	OF= 1
c.	CF = 0	SF= 1	ZF = 0	OF= 0

4.4.5 Section Review

5. Fill in the requested register values on the right side of the following instruction sequence:



```
.data
myBytes BYTE 10h,20h,30h,40h
myWords WORD 8Ah,3Bh,72h,44h,66h
myDoubles DWORD 1,2,3,4,5
myPointer DWORD myDoubles

.code
start:
    mov esi,OFFSET myBytes
    mov al,[esi]                ; a. AL =
    mov al,[esi+3]              ; b. AL =
    mov esi,OFFSET myWords + 2
    mov ax,[esi]                ; c. AX =
    mov edi,8
    mov edx,[myDoubles + edi]   ; d. EDX =
    mov edx,myDoubles[edi]      ; e. EDX =
    mov ebx,myPointer
    mov eax,[ebx+4]             ; f. EAX =
```

a. AL = 10h

b. AL = 40h

c. AX = 003Bh

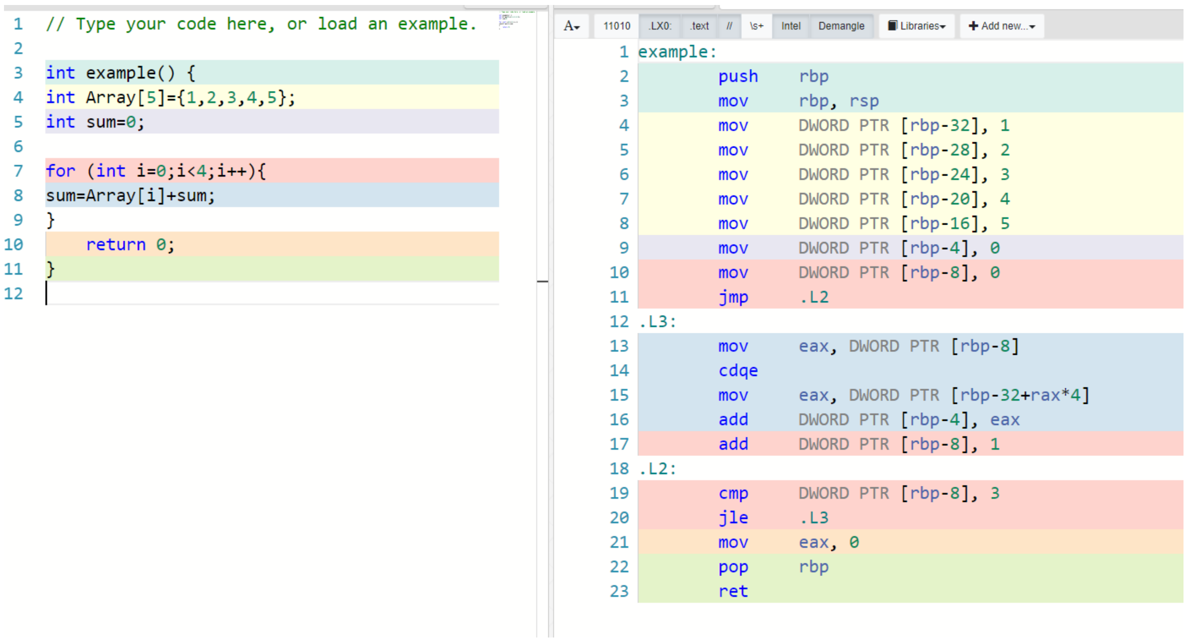
d. EDX = 3

e EDX = 3

f. EAX = 2

Ejercicio Semana 11

https://godbolt.org/



```
// Type your code here, or load an example.

int example() {
int Array[5]={1,2,3,4,5};
int sum=0;

for (int i=0;i<4;i++){
sum=Array[i]+sum;
}
    return 0;
}

```

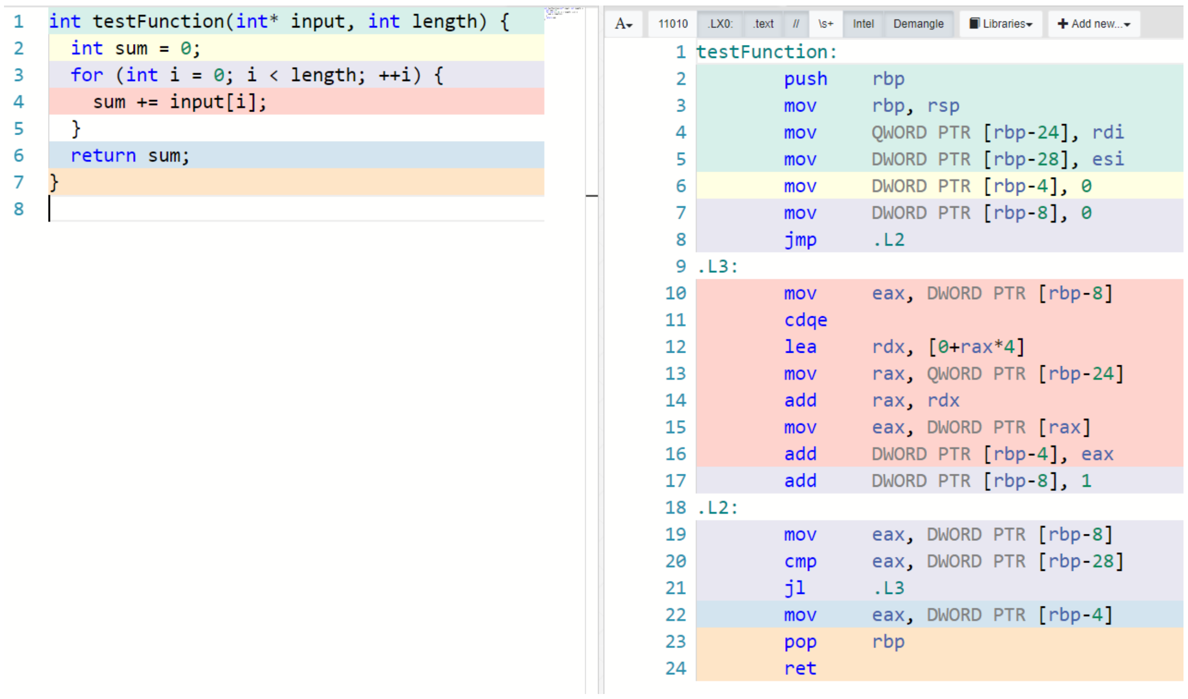
```
example:
        push    rbp
        mov     rbp, rsp
        mov     DWORD PTR [rbp-32], 1
        mov     DWORD PTR [rbp-28], 2
        mov     DWORD PTR [rbp-24], 3
        mov     DWORD PTR [rbp-20], 4
        mov     DWORD PTR [rbp-16], 5
        mov     DWORD PTR [rbp-4], 0
        mov     DWORD PTR [rbp-8], 0
        jmp     .L2
.L3:
        mov     eax, DWORD PTR [rbp-8]
        cdqe
        mov     eax, DWORD PTR [rbp-32+rax*4]
        add     DWORD PTR [rbp-4], eax
        add     DWORD PTR [rbp-8], 1
.L2:
        cmp     DWORD PTR [rbp-8], 3
        jle     .L3
        mov     eax, 0
        pop     rbp
        ret
```

<u>Código en MASM:</u>

```
.386
.model flat, stdcall
option casemap : none
.code
start:
        push ebp
        mov ebp, esp
        mov DWORD PTR [ebp-32], 1
        mov DWORD PTR [ebp-28], 2
        mov DWORD PTR [ebp-24], 3
        mov DWORD PTR [ebp-20], 4
        mov DWORD PTR [ebp-16], 5
        mov DWORD PTR [ebp-4], 0
        mov DWORD PTR [ebp-8], 0
        jmp L2

L3:
        mov eax, DWORD PTR [ebp-8]
        mov eax, DWORD PTR [ebp-32+eax*4]
        add DWORD PTR [ebp-4], eax
        add DWORD PTR [ebp-8], 1

L2:
        cmp DWORD PTR [ebp-8], 3
        jle L3
        mov eax, 0
        pop ebp
        xor eax, eax
        ret
        end start
```



```
int testFunction(int* input, int length) {
  int sum = 0;
  for (int i = 0; i < length; ++i) {
    sum += input[i];
  }
  return sum;
}
```

```
testFunction:
        push    rbp
        mov     rbp, rsp
        mov     QWORD PTR [rbp-24], rdi
        mov     DWORD PTR [rbp-28], esi
        mov     DWORD PTR [rbp-4], 0
        mov     DWORD PTR [rbp-8], 0
        jmp     .L2
.L3:
        mov     eax, DWORD PTR [rbp-8]
        cdqe
        lea     rdx, [0+rax*4]
        mov     rax, QWORD PTR [rbp-24]
        add     rax, rdx
        mov     eax, DWORD PTR [rax]
        add     DWORD PTR [rbp-4], eax
        add     DWORD PTR [rbp-8], 1
.L2:
        mov     eax, DWORD PTR [rbp-8]
        cmp     eax, DWORD PTR [rbp-28]
        jl      .L3
        mov     eax, DWORD PTR [rbp-4]
        pop     rbp
        ret
```

Código en MASM:

```
.386
.model flat, stdcall
option casemap : none
.code
start:
        push ebp
        mov ebp, esp
        mov DWORD PTR [ebp-24], edi
        mov DWORD PTR [ebp-28], esi
        mov DWORD PTR [ebp-4], 0
        mov DWORD PTR [ebp-8], 0
        jmp L2
L3:
        mov eax, DWORD PTR [ebp-8]
        lea edx, [0+eax*4]
        mov eax, DWORD PTR [ebp-24]
        add eax, edx
        mov eax, DWORD PTR [eax]
        add DWORD PTR [ebp-4], eax
        add DWORD PTR [ebp-8], 1
```

```
L2:
        mov eax, DWORD PTR [ebp-8]
        cmp eax, DWORD PTR [ebp-28]
        jl L3
        mov eax, DWORD PTR [ebp-4]
        pop ebp
        xor eax, eax
        ret
        end start
```