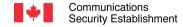
Code Flow 1

If I could change the flow of execution I would be able to write nice applications... Otherwise, I will keep writing small programs...

© Government of Canada

This document is the property of the Government of Canada. It shall not be altered, distributed beyond its intended audience, produced, reproduced or published, in whole or in any substantial part thereof, without the express permission of CSE.





Building IF construct in assembly

Flag	Name	Description
CF	Carry flag	If set on a unsigned mathematical operation, this represent an overflow condition
PF	Parity flag	Indicate if result register contains bad data in math operations
ZF	Zero flag	Flag is set if result of an operation is Zero
SF	Sign flag	Set to most significant bit of the result of an operation
OF	Overflow flag	Used in signed operation in similar way to CF

These flags values change with every instructions being executed!





Setting the flags

The CMP instruction allows to compare two values. It uses two operands:

- cmp eax, ebx

The mechanics involved here go as "EAX - EBX". Flags are set according to result. The values are not modified.

This is also possible:

- cmp eax, 0x01
- cmp eax, [ebp 0x08]

While most instructions will have an impact on the flags values, we will only present CMP at this moment.

Take for granted that every instructions doing any form of computation will have an impact on the flags

For example, "xor eax, eax" would result in the ZF to be set.

Based on what we've seen so far, you should be able to build IF/ELSE type of logic flow construct.

Using the flags

Mnemonic	Description	
JMP	Jump is always taken	
JZ or JE	Jump taken if ZF is set (ZF = 1)	
JL	Jump taken if "less" (SF <> OF)	
JLE	Jump taken if "less or equal" ((ZF = 1) or (SF <> OF))	
JA	Jump taken if "above" (CF = 0 and ZF = 0)	
JAE	Jump taken is "above or equal" (($CF = 0$ and $ZF = 0$) OR ($ZF = 1$))	
	Many more exist See documentation for all variants.	

Various "Jump" instructions are available that allow to jump to a different section of code depending on various conditions.



Putting it together

Let's imagine we are writing a function that receives 2 parameters.

- If the first parameter (C function declaration order) is equal to second parameter, we must return 1
- If both parameters are not equal, we must return 0 How could we write this function in assembly? How could we write this in C?



isEqual (C)

```
int isEqual(int param1, int param2){
    int returnValue;

    if (param1 == param2){
        returnValue = 1;
    }else{
        returnValue = 0;
    }

    return returnValue;
}
```



isEqual (Assembly)

You can always "inspect" the code generated by compilers using objdump # objdump -D -M intel <filename>

```
080483db <isEqual>:
80483db:
                55
                                         push
                                                 ebp
80483dc:
                89 e5
                                                ebp,esp
                                         mov
80483de:
                83 ec 10
                                         sub
                                                esp,0x10
                8b 45 08
                                                eax,DWORD PTR [ebp+0x8]
80483e1:
                                         mov
                3b 45 0c
80483e4:
                                                 eax, DWORD PTR [ebp+0xc]
                                         cmp
80483e7:
                75 09
                                                80483f2 <1sEqual+0x17>
                                         ine
                                                DWORD PTR [ebp-0x4],0x1
                c7 45 fc 01 00 00 00
80483e9:
                                         mov
                                                80483f9 <1sEqual+0x1e>
80483f0:
                eb 07
                                         qmj
                                                -DWORD PTR [ebp-0x4],0x0
80483f2:
                c7 45 fc 00 00 00 00
                8b 45 fc
                                                x,DWORD PTR [ebp-0x4]
80483f9:
                                         mov
80483fc:
                c9
                                         leave
80483fd:
                c3
                                         ret
```





Possible implementation

```
mov ebp, esp
    mov ecx, [ebp + 0xc]; Accessing "second" param
    jmp endIf
notEqual:
endIf:
```

Looking at this, you should have one major conclusion popping in your mind...

Assembly programming is essentially GOTO based...

Could we make this code better?



What about this?

```
isEqual:
        push ebp
        mov ebp, esp
        mov ebx, [ebp + 0x8]
        mov ecx, [esp + 0xc] ; Accessing "first" param
        cmp ebx, ecx
        jne notEqual
    notEqual:
        pop ebp
        ret
```

Could we make it even better?



And this?

```
isEqual:
    mov ebx, [esp + 0x4] ; Accessing "first" param
    mov ecx, [esp + 0x8] ; Accessing "second" param
    xor eax, eax
    cmp ebx, ecx
    jne notEqual
    mov eax, 0x01 ; Both params are equal
    notEqual:
    ret
```

Is this really better?
Can we do even better?



This is getting short

```
isEqual:
    mov ebx, [esp + 0x4] ; Accessing "first" param
    xor eax, eax
    cmp ebx, [esp + 0x8]
    setz al
    ret
```

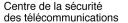
Using conditional move (CMOVcc) and conditional set (SETcc) allows to remove even more branches.

Is this really better?
What are the involvement here?

- Branch prediction unit
- Out of order execution

Can we be sure it's better?
Could we do even better?





Is this better?

с3

80483f8:

```
080483e0 <isEqual>:
                                                 ebx, DWORD PTR [esp+0x4]
80483e0:
                8b 5c 24 04
                                         mov
80483e4:
                31 c0
                                                 eax, eax
                                          xor
80483e6:
                33 5c 24 08
                                                 ebx, DWORD PTR [esp+0x8]
                                          xor
80483ea:
                0f 94 c0
                                         sete
80483ed:
                                          ret
                                   VS
080483e0 <isEqual>:
80483e0:
                8b 5c 24 04
                                          mov
                                                  ebx, DWORD PTR [esp+0x4]
 80483e4:
                31 c0
                                          xor
                                                  eax, eax
 80483e6:
                3b 5c 24 08
                                                  ebx, DWORD PTR [esp+0x8]
                                          cmp
 80483ea:
                0f 94 c0
                                          sete
 80483ed:
                                          ret
                                   VS
080483e0 <isEqual>:
80483e0:
                55
                                          push
                                                 ebp
80483e1:
                89 e5
                                                 ebp,esp
                                          mov
80483e3:
                8b 45 08
                                                 eax, DWORD PTR [ebp+0x8]
                                          mov
80483e6:
                8b 4c 24 0c
                                                 ecx, DWORD PTR [esp+0xc]
                                          mov
80483ea:
                39 c8
                                                 eax,ecx
                                          cmp
80483ec:
                75 07
                                                 80483f5 <notEqual>
                                          jne
80483ee:
                b8 01 00 00 00
                                                 eax.0x1
                                          mov
80483f3:
                eb 02
                                                 80483f7 <endIf>
                                          jmp
080483f5 <notEqual>:
80483f5:
                31 c0
                                          xor
                                                 eax, eax
080483f7 <endIf>:
 80483f7:
                5d
                                                 ebp
                                          pop
```

ret

The first version we did (at the bottom) is 25 bytes long while the two last one are only 14!

- All 3 versions are comparable from a logical standpoint
- Is there something wrong with the latest version?
- Should we chose xor over cmp in this case?



Can we do better than the compiler?

```
080483e0 <isEqual>:
 80483e0:
                 8b 5c 24 04
                                                  ebx, DWORD PTR [esp+0x4]
                                          mov
 80483e4:
                 31 c0
                                           xor
                                                  eax, eax
 80483e6:
                 3b 5c 24 08
                                                  ebx, DWORD PTR [esp+0x8]
                                           cmp
 80483ea:
                 0f 94 c0
                                           sete
                                                  al
 80483ed:
                 с3
                                           ret
```

VS

```
080483e0 <isEqual>:
 80483e0:
                 8b 44 24 04
                                                  eax, DWORD PTR [esp+0x4]
                                           mov
 80483e4:
                3b 44 24 08
                                                  eax, DWORD PTR [esp+0x8]
                                           cmp
 80483e8:
                 0f 94 c0
                                                  al
                                           sete
 80483eb:
                 0f b6 c0
                                                  eax,al
                                          movzx
 80483ee:
                 c3
                                           ret
80483ef:
                 90
                                           nop
```

As part of this training, we will concentrate on the first type of code.

You are free of trying and exploring small optimization but most code we will be showing is not optimized in order to ease code reading

Now, let's come back to Earth...





If with Else

```
int isEqual(int param1, int param2){
                                                 // push ebp
                                                 // mov ebp, esp
        int returnValue;
                                                 // mov eax, [ebp + 0x8]
                                                 // mov ecx, [ebp + 0xc]
        if (param1 == param2){
                                                 // cmp eax, ecx
                                                 // jne notEqual
                returnValue = 1;
                                                 // mov eax, 0x01
                                                 // jmp endIf
        }else{
                                                 // notEqual:
                returnValue = 0;
                                                 // xor eax, eax
                                                 // endIf:
                                                 // pop ebp
        return returnValue;
                                                 // ret
```



Let's write some code!



