BASIC ASSEMBLY

Structured Branching

Assembly language programming
By xorpd

xorpd.net

OBJECTIVES

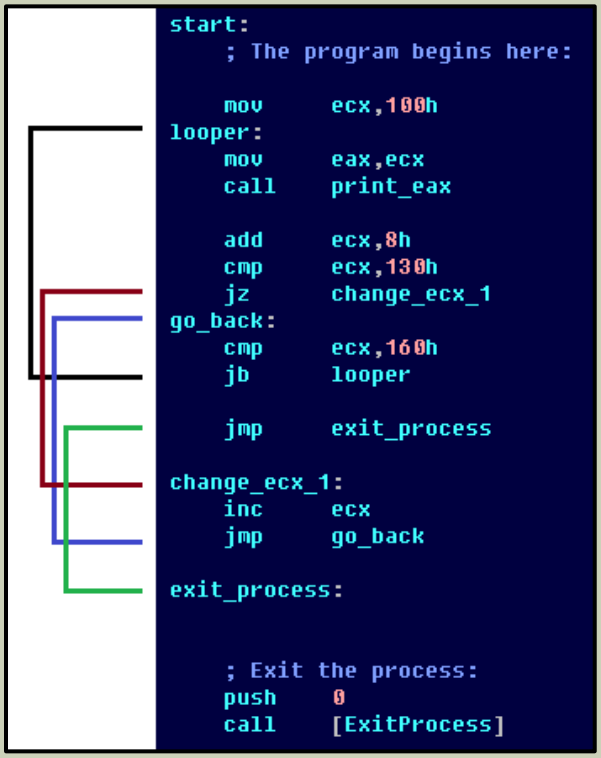
- We will learn about the power and the complexity that JMP style branching can bring.
- We will study a few high level structured branching constructs, and learn how to follow them when branching in our code.
- We will turn a hard to read piece of code into a nice piece of code.

THE POWER OF BRANCHING

- Assembly programming gives the programmer much freedom.
- There are no jump based branches in modern programming languages.
  - Sometimes there are, but their use is usually frowned upon.
- Branching is one of the greatest powers you get as an assembly programmer.
  - “With great powers, comes great responsibility.” (Spiderman, Uncle Ben)
- Unstructured use of branching caused great pain in programming languages of the past.
- Modern programming languages removed much of the branching freedom the programmers used to have.
  - To protect the programmer from himself.

THE CURSE OF BRANCHING

- Conditional branches are very powerful.
- Many jumps could easily create unreadable code.
- Usually referred to as spaghetti code.



```
start:
    ; The program begins here:

    mov     ecx,100h
looper:
    mov     eax,ecx
    call    print_eax

    add     ecx,8h
    cmp     ecx,130h
    jz      change_ecx_1
go_back:
    cmp     ecx,160h
    jb      looper

    jmp     exit_process

change_ecx_1:
    inc     ecx
    jmp     go_back

exit_process:


    ; Exit the process:
    push    0
    call    [ExitProcess]
```

STRUCTURED BRANCHING

- During the last few decades, programmers have developed some rules of thumb and conventions to avoid spaghetti code.
- A few higher level structures of branching were introduced:
  - Conditional execution:
    - **If** statement.
  - Loops:
    - The **For** loop.
    - The **While** loop.
  - Exceptions.
    - We might discuss this construct in the future.
- Limiting yourself to using those constructs–
  - Will make your code readable to other programmers.
  - Will help you to avoid spaghetti code.
- The larger your code becomes, the more important it is to use higher level branching structures.

CONDITIONAL EXECUTION

- The IF-ELSE statement.
  - Useful to take decisions, based on some condition.
  - If the condition is fulfilled, do X. If not, do Y.

```
if eax < edx:

        eax <- eax + 1

else:
        eax <- eax - 1

end if
```

```
        cmp     eax,edx
        jae     else
        inc     eax
        jmp     end_if
else:
        dec     eax

end_if:
```

LOOPS (FOR)

- The FOR loop:
  - Useful for iterating over a range of numbers.
  - **Example** - Sum all the numbers from 0 to 99 (Inclusive):

```
for ecx from 0 to 99 do:
        eax <- eax + ecx




end for
```

```
        mov     ecx,0
for_loop:
        add     eax,ecx

        inc     ecx
        cmp     ecx,100d
        jnz     for_loop
```

LOOPS (WHILE)

- The WHILE loop:
  - Useful to keep going as long as some condition is fulfilled.
  - A FOR loop is a specific type of WHILE loop.
  - **Example**: We sum 0+1+2+3+… until we get to sum of at least 1000:

```
ecx <- 0
eax <- 0
while eax < 1000:


        eax <- eax + ecx
        ecx <- ecx + 1

end while
```

```
        mov     ecx,0
        mov     eax,0
while_loop:
        cmp     eax,1000d
        jae     end_while
        add     eax,ecx
        inc     ecx
        jmp     while_loop
end_while:
```

  - In the end of this WHILE loop, we must have eax ≥1000.

BREAKING FROM LOOPS

- Sometimes you might want to end your loops earlier.
  - We use the "break" higher level construct to exit the loop immediately.
  - Break exits the innermost loop currently running. Works with both FOR and WHILE loops.
- **Example**: We calculate 0+1+2+… and we want to find the first moment where the sum is at least 1000. However, we don't want to sum more than 300 numbers:

```
eax <- 0

for ecx from 0 to 299 do:

          eax <- eax + ecx
          if eax >= 1000:
                    break



end for
```

```
          mov       eax,0
          mov       ecx,0
for_loop:

          add       eax,ecx
          cmp       eax,1000
          jae       end_for

          inc       ecx
          cmp       ecx,300
          jb        for_loop
end_for:
```

- In the end of this loop, either eax is larger than 1000, or we have iterated over all the numbers from 0 to 299.

BRANCHING RULES OF THUMB

- For every jmp instruction that you use, ask yourself:
  - Is this jmp part of an IF statement?
  - Is this jmp part of a FOR loop?
  - Is this jmp part of a WHILE Loop?
  - Is this jmp being used to BREAK from the innermost loop?
- If your jmp is non of these, you should probably not jump.

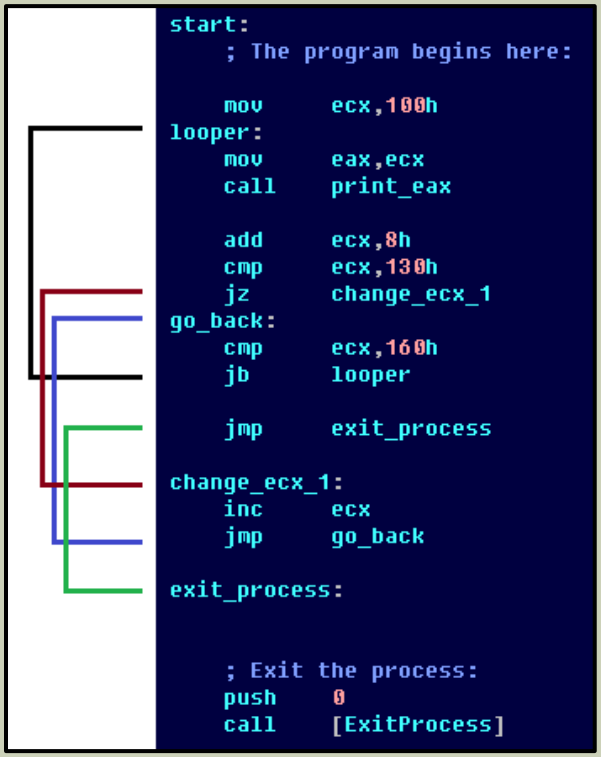
- JMPs by direction:
  - Jumps forward should be IF statements or BREAK statements.
  - Jumps backwards should be FOR or WHILE loops.

- You may nest IF, FOR and WHILE constructs **inside** each other.

- Try to make sure your jump paths do not cross each other.

EXAMPLE

- We will simplify the spaghetti code from the first slide.



```
start:
    ; The program begins here:

    mov     ecx,100h
looper:
    mov     eax,ecx
    call    print_eax

    add     ecx,8h
    cmp     ecx,130h
    jz      change_ecx_1
go_back:
    cmp     ecx,160h
    jb      looper

    jmp     exit_process

change_ecx_1:
    inc     ecx
    jmp     go_back

exit_process:


    ; Exit the process:
    push    0
    call    [ExitProcess]
```

- Note that many branches intersect each other.

EXAMPLE (ANALYZING THE CODE)

- Take some time to try to understand the code. You may run it to get a better understanding of it.

- The code initiates ecx to be 100h.
- Then it increases it by 8h every time, until ecx = 130h.
- ecx is increased by 1.
- The code keeps increasing ecx by 8h every time, until ecx is at least 160h.
- Finally, the program jumps to exit_process.

- Output:
  100,108,110,118,120,128,131,139,141,149,151,159

spaghetti.asm

```
            mov     ecx,100h
looper:
            mov     eax,ecx
            call    print_eax

            add     ecx,8h
            cmp     ecx,130h
            jz      change_ecx_1
go_back:
            cmp     ecx,160h
            jb      looper

            jmp     exit_process

change_ecx_1:
            inc     ecx
            jmp     go_back

exit_process:
```

EXAMPLE (STRUCTURED BRANCHING)

```
          mov     ecx,100h
looper:
          mov     eax,ecx
          call    print_eax

          add     ecx,8h
          cmp     ecx,130h
          jz      change_ecx_1
go_back:
          cmp     ecx,160h
          jb      looper

          jmp     exit_process

change_ecx_1:
          inc     ecx
          jmp     go_back

exit_process:
```

Structured Branching Pseudo-Code

```
ecx <- 0x100
while ecx < 0x160:
          print(ecx)
          ecx <- ecx + 8
          if ecx == 0x130:
                    ecx <- ecx + 1
          end if
end while
```

- Output: 100,108,110,118,120,128,131,139,141,149,151,159

EXAMPLE (STRUCTURED BRANCHING)

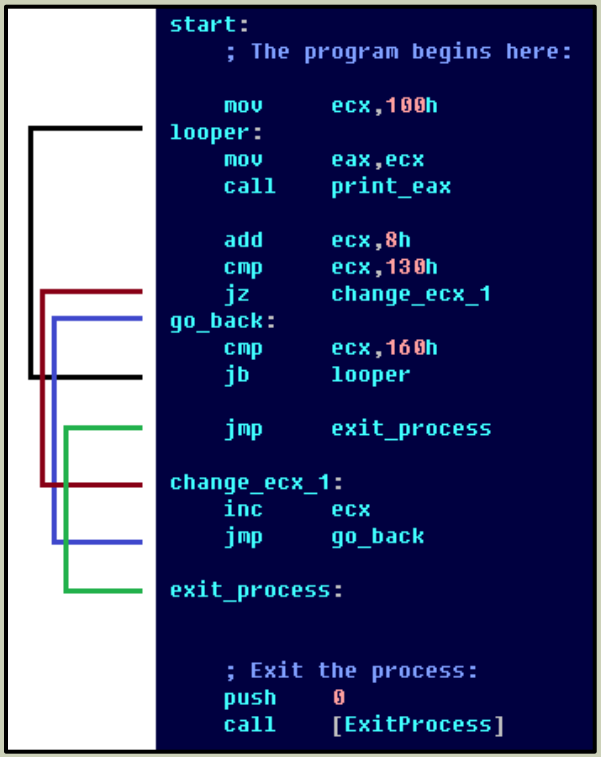
- We write new assembly piece of code, this time using structured branching:

```
            mov     ecx,100h
while_loop:
            mov     eax,ecx
            call    print_eax

            add     ecx,8h

            cmp     ecx,130h
            jnz     end_if
            inc     ecx
end_if:


            cmp     ecx,160h
            jb      while_loop
```

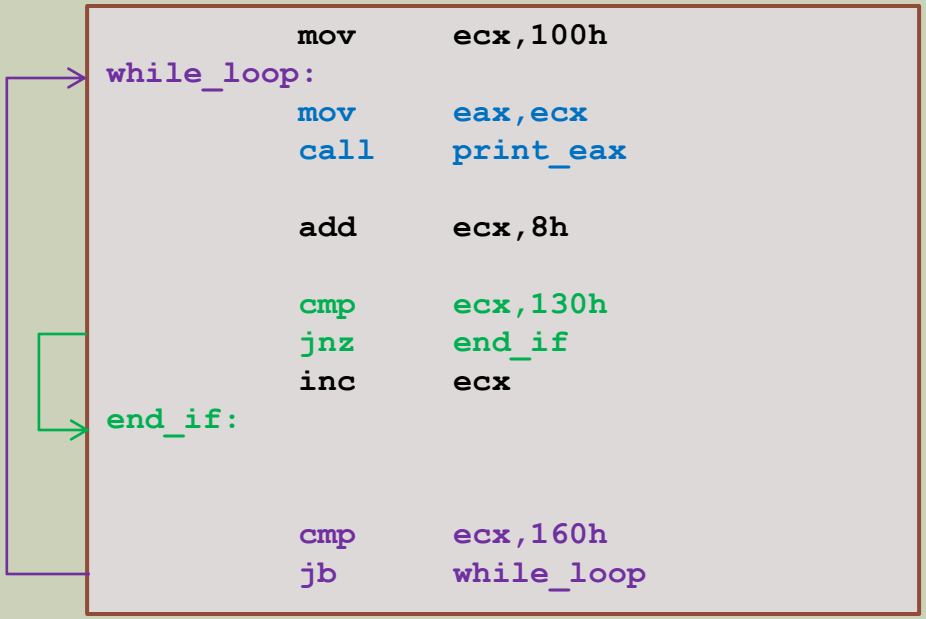
```
ecx <- 0x100
while ecx < 0x160:
            print(ecx)


            ecx <- ecx + 8

            if ecx == 0x130:

                        ecx <- ecx + 1
            end if



end while
```

EXAMPLE (STRUCTURED BRANCHING)

- This time, the branches do not intersect:



```
start:
    ; The program begins here:

    mov     ecx,100h
looper:
    mov     eax,ecx
    call    print_eax

    add     ecx,8h
    cmp     ecx,130h
    jz      change_ecx_1
go_back:
    cmp     ecx,160h
    jb      looper

    jmp     exit_process

change_ecx_1:
    inc     ecx
    jmp     go_back

exit_process:


    ; Exit the process:
    push    0
    call    [ExitProcess]
```



```
            mov     ecx,100h
while_loop:
            mov     eax,ecx
            call    print_eax

            add     ecx,8h

            cmp     ecx,130h
            jnz     end_if
            inc     ecx
end_if:


            cmp     ecx,160h
            jb      while_loop
```

EXAMPLE (FURTHER IMPROVEMENT)

- In the end of the lecture, take a look at:
  - **spaghetti.asm** – The original code.
  - **structured1.asm** – Our first structured alternative to the original code.
  - **structured2.asm** – Another structured alternative to the original code.
- Compare the two structured alternatives.
  - Note that in both structured1.asm and structured2.asm, branches do not intersect.

SUMMARY

- JMPs give the programmer great power.
  - But also great pain, if used without care.
- When using JMPs, we follow structured branching constructs: IF, FOR, WHILE and BREAK.
  - To make our code easier to read, for us and for our coworkers.
  - To make our code easier to maintain.

EXERCISES

- Some code reading.
  - Examples of using structured style branches.
- Code writing.
  - Writing some loops, conditionals and nesting those inside each other.
- Fixing spaghetti code.