THE UNIVERSITY OF ARIZONA
UA South

CYBV 471 Assembly Programming for Security Professionals
Week 9
Control Structures: Branching and Looping

Agenda

- **Control Structures**
  - Compare instruction
  - Unconditional jump
  - Conditional jump
  - Test operation with conditional jump
  - if-then-else
  - For loop instruction
  - While loop instruction
  - Do while loop
  - Do while loop vs while loop
  - Repeat string instructions (string encoding)

Compare Instruction: CMP

- CMP instruction subtracts one operand from another but doesn't store the result anywhere

```
cmp    a, b       ; compare a with b,
                  ; which computes a-b
```

- CMP instruction will modify the FLAGS register based on values of a and b
- Which FLAGS?

CMP With Unsigned Integers (+ values)

```
cmp    a, b       ; compare a with b,
                  ; which computes a-b
```

- CMP instruction will modify the values of ZF (Zero Flag) and CF (Carry Flag) based on value of operands a and b
  - ZF: set to 1 if result is 0
  - CF: set if we have overflow, have a carry, or borrow conditions

If a = b: ZF is set (=1), CF is not set (=0)

If a > b: ZF is not set (=0), CF is not set (=0)

If a < b: ZF is not set (=0), CF is set (borrow) (=1)

CMP With Signed Integers (-/+ values)

```
cmp    a, b       ; compare a with b,
                  ; which computes a-b
```

- CMP instruction will modify the values of ZF (Zero Flag), OF (Overflow Flag), and SF (Sign Flag) based on value of operands a and b

  - ZF: set to 1 if result is 0
  - OF: set to 1 if the result overflows or underflows
  - SF: set to 1 if the result is negative

  If a = b: ZF is set (=1), OF is not set (=0), SF is not set (=0)

  If a > b: ZF is not set (=0), and OF = SF

  If a < b: ZF is not set (=0), and OF ≠ SF

Understand the relationship between OF and SF Flags for Signed Integers

```
cmp    a, b       ; compute (a – b)
```

- If a = b: ZF is set (=1), OF is not set (=0), SF is not set (=0)

- If a > b: ZF is not set (=0), and OF = SF
  - If there is no overflow (OF=0), the (correct) result is positive (SF=0)
  - If there is an overflow (OF=1), the (incorrect) result is negative (SF=1)
  - Therefore, in both cases SF = OF

- If a < b: ZF is not set (=0), and OF ≠ SF
  - If there is no overflow (OF=0), the (correct) result is negative (SF=1)
  - If there is overflow (OF=1), the (incorrect) result is positive (SF=0)
    The correct answer should be negative.
  - Therefore, in both cases SF ≠ OF

Examples: Values of OF and SF Flags for Signed Integers

Example: Assume a = 80h (-128d)?

80h = 1000 0000 > 2nd complement -> value =(-128d)

b = 23h > 0010 0011 -> value = 35d

(a < b, one byte)

a - b = a + (-b) = 80h + (2's complement of b) = 80h + DDh (=15Dh)

15Dh = 0001 0101 1101.

15Dh > for two bytes, we have 0101 1101 (Drop 1 because we have one byte) = 5Dh, (SF=0)

-128d – 35d should be negative.

Result is 0101 1101 which is positive. Therefore, OF=1 (incorrect) (SF ≠ OF)

Example: Assume a = F3h (-13d), b = 23h (+35d)

(a < b, one byte)

a - b = a + (-b) = F3h + (2's complement of b) = F3h + DDh = D0h

D0h = 1101 0000 -> value?? (= - 2's complement of D0h) = - 48d (correct)

D0h is negative and we have no overflow (in range). So, SF=1 and OF=0 (SF ≠ OF)

Examples: Values of "OF" and "SF" Flags for Signed Integers

Example: Assume : a = F3h (-13d), b = 82h (-126d)
(a > b, one byte)

a - b = a + (-b) = F3h + 7Eh = 171h

We have overflow condition:

171h = 0001 0111 0001. We have only one byte.

The value is 0111 0001 = 71h = (+ 113d) (correct answer)

The result is correct answer. Therefore OF =0

71 is positive (SF=0). Therefore, in both cases SF = OF

Example: a = 70h (112d), b = D8h (-40d)
(a > b, one byte)

a - b = a + (-b) = 70h + 28h = 98h, (152 is > 127, overflow)

98h = 1001 1000 = negative. (2^{nd} complement value) = -104d (incorrect)

Therefore OF =1

98h = 1001 1000 -> SF =1

So, SF=1 and OF=1

Summary: CMP for Unsigned and Signed Integers

	cmp a,b	ZF	CF	OF	SF
unsigned	a = b	1	0		
	a < b	0	1		
	a >b	0	0		
signed	a = b	1		0	0
	a < b	0		v v (0 or 1)	!v
	a > b	0		v	v

Jumps and Branches

- In the normal program flow, we execute instructions one by one (next instruction fashion)
- Assembly language has instructions that allow you to modify the order in which instructions are executed.
  - i.e., we not always execute the next instruction
  - Unconditional jumps
  - Conditional jumps

Unconditional Jump
The "JMP" Instruction

- JMP instruction allows you to "jump" to a code label
- Example:

```
        instruction-1
        instruction-2
        jmp label-1          ; then jump to  Label-1
        instruction          ; will never be executed
        instruction          ; will never be executed
        . . .
Label-1:
        instruction-1
        instruction-2
        . . .
```

- JMP instruction will change the content of the EIP register (program counter) to execute sub1 label
- The JMP instruction is an unconditional branch (jump)
- The content of the FLAGS register doesn't affect "JMP" instruction

Conditional Jump Instructions

- These instructions jump to an address in the code segment (i.e., a label) based on the content of the FLAGS register

- Conditional jumps use the flags to determine whether to jump or to proceed to the next instruction.

- FLAGS register is updated by
  - All instructions that perform arithmetic operations
  - The CMP (compare) instruction
  - CMP that sets some flags in the FLAEG register

- Example:

```
        instruction-1
        instruction-2
        Arithmetic instruction or CMP instruction  ; may modify FLAGS register
        Conditional jump label-1    ; jump if condition is satisfied
        instruction                 ; (no jump) continue from here if condition is not satisfied
        instruction
            . . .
label1:
        instruction-1
        instruction-2
```

Conditional Jump Instructions

- There is a large set of conditional jump instructions that act based on bits in the FLAGS register
- The simple jump instructions jumps (or not) depending on the value of one of the following flags:

  - ZF, OF, SF, CF, PF
  - PF: Parity Flag
    - Set to 0 if there is odd number of ones in the lower 8-bit of the "result"
    - Set to 1 if there is even number of ones in the lower 8-bit of the "result"
    - Examples: Result = 11011111 10001000 (we have 2 ones, PF=1)
    - Result = 11011111 10001100 (we have 3 ones, PF=0)

Conditional Jump Instructions

- The simple jump instructions jumps (or not) depending on the value of one of the following flags: ZF, OF, SF, CF, PF

JZ	jumps (branches) if ZF is set (1)
JNZ	branches if ZF is unset
JO	branches if OF is set
JNO	branches if OF is unset
JS	branches is SF is set
JNS	branches is SF is unset
JC	branches if CF is set
JNC	branches if CF is unset
JP	branches if PF is set
JNP	branches if PF is unset

Conditional Jump Based on Unsigned Comparison

Instruction	branches if
JE x, y	x = y
JNE	x != y
JB, JNAE	x < y
JBE, JNA	x <= y
JA, JNBE	x > y
JAE, JNB	x >= y

Mnemonic	Description
JA	Jump if above (if *leftOp* > *rightOp*)
JNBE	Jump if not below or equal (same as JA)
JAE	Jump if above or equal (if *leftOp* >= *rightOp*)
JNB	Jump if not below (same as JAE)
JB	Jump if below (if *leftOp* < *rightOp*)
JNAE	Jump if not above or equal (same as JB)
JBE	Jump if below or equal (if *leftOp* <= *rightOp*)
JNA	Jump if not above (same as JBE)

Conditional Jump Based on Signed Comparison

Instruction	branches if
JE	x = y
JNE	x != y
JL, JNGE	x < y
JLE, JNG	x <= y
JG, JNLE	x > y
JGE, JNL	x >= y

Mnemonic	Description
JG	Jump if greater (if *leftOp* > *rightOp*)
JNLE	Jump if not less than or equal (same as JG)
JGE	Jump if greater than or equal (if *leftOp* >= *rightOp*)
JNL	Jump if not less (same as JGE)
JL	Jump if less (if *leftOp* < *rightOp*)
JNGE	Jump if not greater than or equal (same as JL)
JLE	Jump if less than or equal (if *leftOp* <= *rightOp*)
JNG	Jump if not greater (same as JLE)

Conditional Jump for Unsigned/Signed Comparison

cmp x, y			
signed		Unsigned	
Instruction	branches if	Instruction	branches if
JE	x = y	JE	x = y
JNE	x != y	JNE	x != y
JL, JNGE	x < y	JB, JNAE	x < y
JLE, JNG	x <= y	JBE, JNA	x <= y
JG, JNLE	x > y	JA, JNBE	x > y
JGE, JNL	x >= y	JAE, JNB	x >= y

Summary Conditional Jump Instructions

- Here are the most common conditional jump instructions

Mnemonic	Description	Flags
JZ	Jump if zero	ZF = 1
JNZ	Jump if not zero	ZF = 0
JC	Jump if carry	CF = 1
JNC	Jump if not carry	CF = 0
JO	Jump if overflow	OF = 1
JNO	Jump if not overflow	OF = 0
JS	Jump if signed	SF = 1
JNS	Jump if not signed	SF = 0
JP	Jump if parity (even)	PF = 1
JNP	Jump if not parity (odd)	PF = 0

Test operation with conditional jump

- Test instruction performs a nondestructive AND operation between each pair of matching bits in two operands
- No operands are modified, but the Zero flag (ZF) is affected.
- Example-1: jump to a label-1 if either bit 0 or bit 1 in AL is set (1)

```
test al,00000011b    ; zf = 1  if al has xxxxxx00
                     ; zf = 0 if al has xxxxxx01 (or 10 or 11)
jnz label-1          ; jump if zf is not set (zf=0)
```

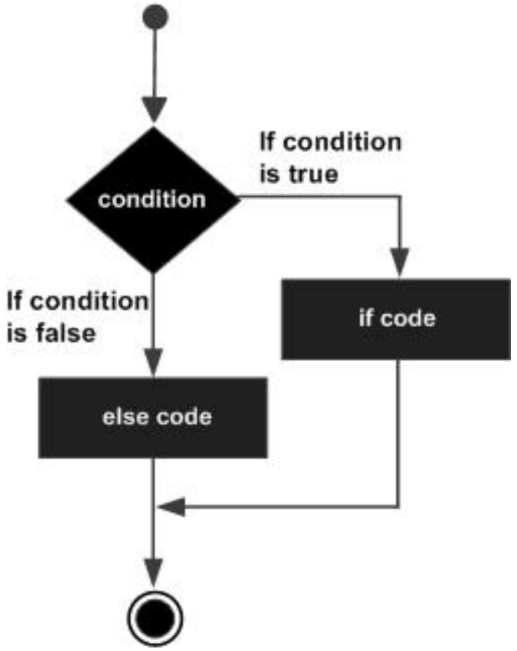
Example-2: jump to a label-1 if neither bit 0 nor bit 1 in AL is set (1)

```
test al,00000011b    ; zf = 1 if al has xxxxxx00
                     ; zf =  0 if al has xxxxxx01 (or 10 or 11)
jz  label-1          ; jump if zf = 1
                     ; no jump if zf is 0
```

Control Structures: if-then-else

A generic if-then-else construct:

```
    if (condition) then
       {
        execute block-code-1    ; if  condition is true
        }
    else
       {
         execute else-block-code-2  ; if condition is false
       }
        next instruction
```



Translation into x86 assembly:

```
                ; instructions to set flags (e.g., cmp ...)
                 jyy else-block        ; jyy branches to else-block  if-condition is false
 if-block:
                ; code of block-code-1
                 jmp endif
else-block:
               ; code of block-code2
     endif:
            next instruction
```

Control Structures: if-then-else

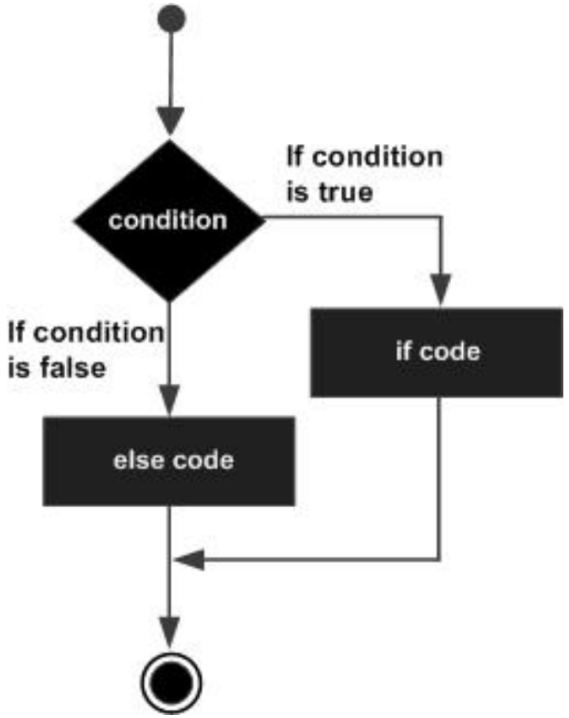
Example:

```
if (val1 = = val2)
    x = 1;
else
    x = 2
  Next instruction
```

Translation into x86 assembly:

```
        mov eax, var1
        cmp eax, var2
        jne  L1          ; condition false
        mov x, 1         ; condition true
        jmp L2           ; end of if-else

L1: mov x, 2
L2:
   Next instruction
```



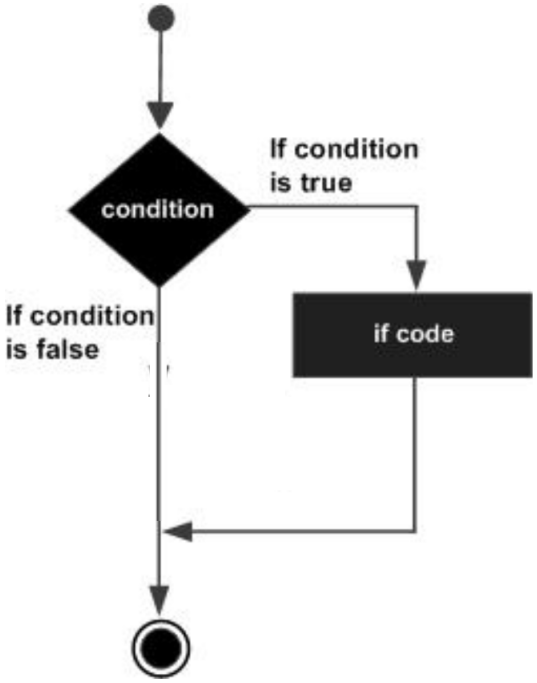
Control Structures: if-then (no else)

A generic if-then-else construct:

```
        if (condition) then
            {
            execute block-code-1     ; if  condition is true
            }
            next instruction
```

Translation into x86 assembly:

```
      ; instructions to set flags (e.g., cmp ...)
       jxx endif    ; jxx branches to endif  if condition is false
if-block:
            ; code of block-code-1
 endif:
            next instruction
```



Control Structures: if-then (no else)

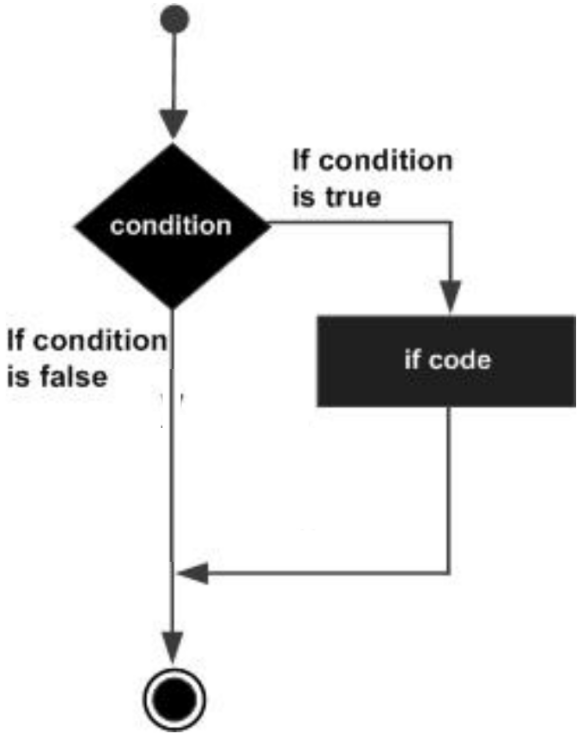
Example:

```
if (val1 = = val2)
    x = 1;
Next instruction
```

Translation into x86 assembly:

```
    mov eax, var1
    cmp eax, var2
    jne  endif        ; condition false

 L1: mov x, 1
endif:
    Next instruction
```



For Loop

Assume we have would like to implement the following c-code

```
sum = 0
for (i = 0;  i <=10 ; i++)
        sum = sum + i
```

Translation into x86 assembly:

```
        mov eax, 0        ;  store sum value in eax
        mov ecx, 0        ;  store i value in ecx

loop_start:
        cmp ecx, 10       ; compare i and 10
        jg loop_end       ; if (i>10) go loop_end
        add eax, ecx      ; if i < 10,  sum = sum + i
        inc ecx           ; i = i + 1
        jmp loop_start    ; start over, go to loop_start
loop_end:
```

For Loop

Assume we have would like to implement the following c-code

```
sum = 0
for (i = 0;  i <=10 ; i++)
        {
          sum = sum + i
           several code lines
        }
```

Translation into x86 assembly:

```
        mov eax, 0  ;  store sum value in eax
        mov ecx, 0  ;  store i ( loop index) value in ecx
       ; use other registers for other variables as needed

loop1_start:
        cmp ebx, 10          ; compare i and 10;
        jg loop1_end          ; jump greater, if (I >10) go loop_end
        add eax, ebx         ; if i <= 10,  sum = sum + i
        inc ecx              ; i = i + 1
        several code lines   ; use other registers
        jmp loop1_start       ; start over, goto loop_start
loop1_end:
```

For Loop

The previous loop can be written in a reverse

```
sum = 0
for (i = 10;  i >= 0 ; i--)
        sum = sum + i
```

Translation into x86 assembly:

```
        mov eax, 0  ;  store sum value in eax
        mov ecx, 10  ;  store i (loop index) value in ecx

loop1_start:
        cmp ecx, 0          ; compare i and 0   (LOOP CONDITION)
        jl loop1_end        ;  if (i < 0) go loop_end  (CHECK CONDITION)
        add eax, ecx        ; if i > 0,  sum = sum + i
        dec ecx             ; i = i – 1  (LOOP CONTRO)
        jmp loop1_start     ; start over, goto loop_start
loop1_end:                  ; END
```

Using the loop Instruction

The previous loop can be written in a reverse

```
sum = 0
for (i = 10;  i > 10 ; i--)
        sum = sum + i
        other codes
```

Translation into x86 assembly:

```
        mov eax, 0  ;  store sum value in eax
        mov ecx, 10  ;  store i (loop index) value in ecx

loop1_start:
        add eax, ecx
        other codes
        loop  loop1_end:     ; check the required conditions and decrement the index
```

NOTE-1: Store loop index in ecx

Make sure you have the reversed loop (decrement i)

Do While loop

A generic do while loop construct:

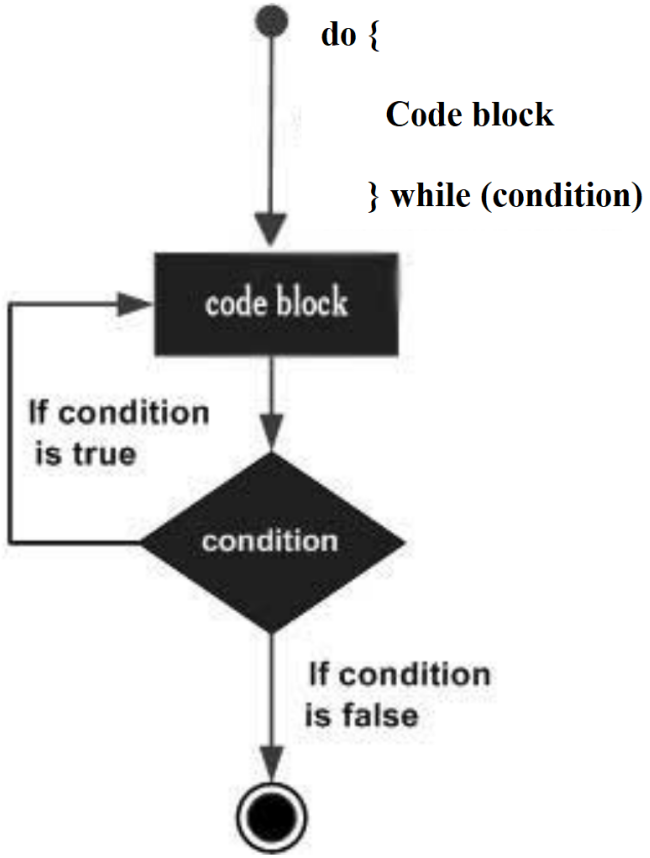
```
do
 {
    execute block-code      ; if  condition is true
  } while (condition)

            next instruction
```

Translation into x86 assembly:

```
do:
    block-code
    ; instructions to set flags (e.g., cmp ...)
     jxx  do          ; branches if condition is true

    next instruction
```



Do While loop

A generic do while loop construct:

```
do
  {
    execute block-code        ; if  condition is true
  } while (condition)

  next instruction
```

The block-code will be executed at least one time

The code will be repeated if the condition is true

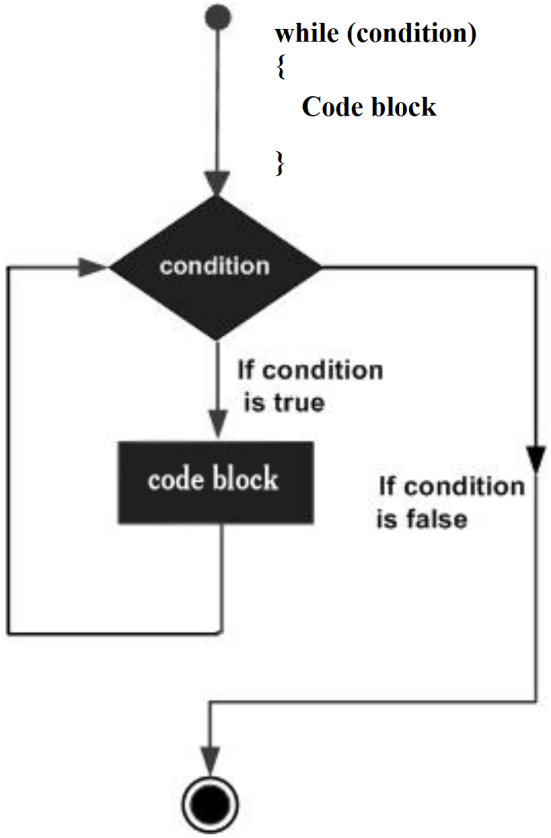
While loop

A generic while loop construct:

```
while (condition)
{
    execute block-code   ; if  condition is true
}
 next instruction
```

Translation into x86 assembly:

```
while:
        ; instructions to set flags (e.g., cmp ...)
        jxx  end_while      ; branches if condition is false
        block-code
        jmp while
end_while:
                next instruction
```



Do While loop vs While loop

A generic do while loop construct:

```
do
  {
    execute block-code        ; if  condition is true
  } while (condition)

  next instruction
```

The block-code will be executed at least one time
The code will be repeated if the condition is true

```
while (condition)
  {
    execute block-code        ; if  condition is true
  }
```

The block-code will be executed only if condition is true

Putting It All Together

You should know:

- **Control Structures**
  - Compare instruction
  - Unconditional jump
  - Conditional jump
  - Test operation with conditional jump
  - if-then-else
  - Loop instructions
  - Repeat (while)string instructions

Questions?

Coming Next Week
C-Stream I/O

Week 9 Assignments

- **Learning Materials**

  1- Week 9 Presentation

  2- Read pages 298-303 (ch. 10): Duntermann, Jeff. Assembly Language Step by Step, Programming with Linux

- **Assignment**

  1- Complete “Lab 9” by coming Sunday 11:59 PM.