CMSC 131 Introduction to Computer Organization and Machine-Level Programming
Handout 4: Conditional and Unconditional Transfer

Learning Outcomes

At the end of this session, the student should be able to:

1. understand conditional and unconditional transfer in Assembly; and
2. develop Assembly programs implementing conditional and unconditional transfer

Content

Labels

A program label is a target, or a location to jump to, for control statements. Generally, a label starts with a letter, followed by letters, numbers or symbols (limited to "_"), terminated with a colon (":"). A label must be defined exactly once.

The general form of a label is:

```
<labelName>:
      NEXT SET OF INSTRUCTIONS
```

Unconditional Control Instructions

The unconditional instructions provide an unconditional jump to a specific location in the program denoted with a program label. The target label must be defined exactly once and accessible and within the scope from the originating jump instruction.

The general form of the unconditional control instruction is:

```
jmp <label>
```

EXAMPLE 1:

```
global _start

section .text
_start:
      mov rcx, 10
      jmp exit_here
      add rcx, 50

exit_here:
      mov rax, 60
      xor rdi, rdi
      syscall
```

Conditional Control Instructions

The conditional control instructions provide a conditional jump based on a comparison. This provides the functionality of a basic IF statement. Two steps are required for comparison: the compare instruction and the conditional jump instruction.

The general form of the compare instruction is:

```
cmp <operand1>, <operand2>
```

NOTES:

- Operand1 and operand2 must be of the same size.
- Operands cannot be both from memory.
- Operand1 cannot be an immediate value, but operand2 may be an immediate value.

EXAMPLE 2:

```
cmp rax, 5
cmp ecx, edx
cmp ax, word[wNum]
```

The result of the compare instruction will be the basis of whether the conditional jump instruction will jump or not jump to the provided label. This requires that the compare instruction is immediately followed by the conditional jump instruction.

The general forms of the conditional jump instruction are:

SAME FOR SIGNED & UNSIGNED CONDITIONAL CONTROL INSTRUCTIONS	
je <label>	jump equal; if <op1> == <op2>
jne <label>	jump not equal; if <op1> != <op2>

SIGNED CONDITIONAL CONTROL INSTRUCTIONS	
jl <label>	jump less than; if <op1> < <op2>
jle <label>	jump less than or equal; if <op1> <= <op2>
jg <label>	jump greater than; if <op1> > <op2>
jge <label>	jump greater than or equal; if <op1> >= <op2>

UNSIGNED CONDITIONAL CONTROL INSTRUCTIONS	
jb <label>	jump below than; if <op1> < <op2>
jbe <label>	jump below or equal; if <op1> <= <op2>
ja <label>	jump above than; if <op1> > <op2>
jae <label>	jump above or equal; if <op1> >= <op2>

EXAMPLE 3:

```
global _start

section .data
      SYS_EXIT equ 60
      x db 25
      y db 5
      isEqual db "N"

section .text
_start:

      mov al, byte[x]                  ; x != y
      cmp al, byte[y]
      jne exit_here

      mov byte[isEqual], "Y"           ; isEqual = "Y"

exit_here:
      mov rax, SYS_EXIT
      xor rdi, rdi
      syscall
```

EXAMPLE 4:

```
global _start

section .data
      SYS_EXIT equ 60
      x db 25
      y db 5
      result db 0

section .text
_start:
      mov al, byte[x]                  ; x < y
      cmp al, byte[y]
      jb is_above

      mov al, byte[x]                  ; result = x
      mov byte[result], al

      jmp exit_here

is_above:
      mov al, byte[y]                  ; result = y
      mov byte[result], al

exit_here:
      mov rax, SYS_EXIT
      xor rdi, rdi
      syscall
```

References

Jorgensen, Ed. 2019. x86-64 Assembly Language Programming with Ubuntu. Version 1.1.40.