

# Control

Young W. Lim

2017-02-02 Thu

## 1 Introduction

- References
- Condition Code
- Accessing the Condition Codes
- Jump Instructions
- Translating Conditional Branches
- Loop Instructions
- Switch

"Self-service Linux: Mastering the Art of Problem Determination", Mark Wilding  
"Computer Architecture: A Programmer's Perspective", Bryant & O'Hallaron

I, the copyright holder of this work, hereby publish it under the following licenses: GNU head Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled GNU Free Documentation License.

CC BY SA This file is licensed under the Creative Commons Attribution ShareAlike 3.0 Unported License. In short: you are free to share and make derivative works of the file under the conditions that you appropriately attribute it, and that you distribute it only under a license compatible with this one.

# Condition Codes

- CF (Carry Flag)
- ZF (Zero Flag)
- SF (Sign Flag)
- OF (Overflow Flag)

# Condition Code Examples

```
addl  
t=a+b
```

```
CF: (unsigned t) < (unsigned a)  
    mag(t) < mag(a) if C=1
```

```
ZF: (t == 0)  
    zero t
```

```
SF: (t < 0)  
    negative t
```

```
OF: (a<0 == b<0) && (t<0 != a<0)  
    sign(a) == sign(b) != sign(t)
```

# Compare and Test

```
cmpb  S2, S1    ; S1 - S2 : compare bytes
testb S2, S1    ; S1 & S2 : test bytes
cmpw  S2, S1    : S1 - S2 : compare words
testw S2, S1    : S1 & S2 : test words
cmpl  S2, S1    : S1 - S2 : compare double words
testl S2, S1    : S1 & S2 : test double words
```

```

sete  D (setz)      ; D <- ZF   (equal / zero)
setne D (setnz)    ; D <- ~ZF  (not equal/ not zero)
sets  D            ; D <- SF    (negative)
setns D           : D <- ~SF   (non-negative)
setg  D (setle)    ; D <- ~(SF^OF)&~ZF (greater, signed >)
setge D (setnl)    ; D <- ~(SF^OF) (greater or equal, signed >=)
setl  D (setnge)   ; D <- SF^OF (less, signed <)
setle D (setng)    ; D <- (SF^OF)|ZF (less or equal, signed <=)
seta  D (setnbe)   ; D <- ~CF&~ZF (above, unsigned >)
setae D (setnb)    ; D <- ~CF  (above or euqal, unsinged >=)
setb  D (setnae)   ; D <- CF   (below, unsigned <)
setbe D (setna)    : D <- CF&~ZF (below or equal, unsigned <=)

```

# Jump instructions

```
jmp Label           ; if (1) direct jump
jmp *Operand       ; if (1) indirect jump
je Label (jz)      ; if (ZF) equal/zero jump
jne Label (jnz)    ; if (~ZF) not equal / non-zero jump
js Label           ; if (SF) negative jump
jns Label          ; if (~SF) non-negative jump
jg Label (jnl)     ; if (~(SF^OF)&~ZF) greater, signed > jump
jge Label (jnl)    ; if (~(SF^OF)) greater or equal, signed >=
jl Label (jnle)    ; if ((SF^OF)) less, signed < jump
jle Label (jnle)   ; if ((SF^OF)|ZF) less or equal, signed <=
ja Label (jnb)     ; if (~CF&~ZF) above, unsigned > jump
jae Label (jnb)    ; if (~CF) above or equal, unsigned >= jump
jb Label (jnae)    ; if (CF) below, unsigned < jump
jbe Label (jna)    ; if (CF&~ZF) below or equal, unsigned <= j
```



# if-else statement

```
if (expr)
  then-statement
else
  else-statement
```

```
t = exorl
if (t)
  goto true;
  // else-statement
  goto done;
true:
  // then-statement
done:
```

## if-else exmple (1)

```
int abs(int x, int y)
{
    if (x<y)
        return y-x;
    else
        return x-y;
}
```

```
int abs_goto(int x, int y)
{
    int val;

    if (x < y)
        goto true;
    val = x - y;
    goto done;
true:
    val = y - x;
done:
    return val;
}
```

## if-else exmple (2)

```
movl 8(%ebp), %edx
movl 12(%ebp), %eax
cmpl %eax, %edx
jl   .L3
subl %eax, %edx
movl %edx, %eax
jmp  .L5
.L3:
subl %edx, %eax
.L5:
```

```
int abs_goto(int x, int y)
{
    int val;

    if (x < y)
        goto true;
    val = x - y;
    goto done;
true:
    val = y - x;
done:
    return val;
}
```

# do-while statement

```
do
    // body-statement
while (expr);
```

```
loop:
    // body-statement
    t = expr;
    if (t)
        goto loop;
```

# while statement (1)

```
while (expr)
{
    // body-statement
}
```

```
loop:
    t = expr;
    if (!t)
        goto done;
    // body-statement
    goto loop;
done:
```

## while statement (2)

```
if (!expr)
    goto done;
do
    // body-statement
while (expr);
done:
```

```
t = expr;
if (!t)
    goto done;
loop:
    // body-statement
    t = expr;
    if (t)
        goto loop;
done;
```

# for statement (1)

```
for (init; test; update)
    // body-statement
```

```
init_expr;
while (test) {
    // body-statement
    update;
}
```

## for statement (2)

```
init_expr;
if (!test)
    goto done;
do {
    // body-statement
    update;
} while (test);
done:
```

```
init_expr;
t = test;
if (!t)
    goto done;
loop:
    // body-statement
    update_expr;
    t = test;
    if (t)
        goto loop;
done;
```



# Switch statement (1)

```
int switch_example(int x)
{
    int result = x;

    switch (x) {
        case 100: result *= 13; break;
        case 102: result += 10;
        case 103: result += 11; break;
        case 104:
        case 105: result *= result; break;
        default: result = 0;
    }

    return result;
}
```

## Switch statement (2)

```
code *jt[7] =  
{loc_A, loc_def, loc_B, loc_C, loc_D, loc_def, loc_D };
```

```
int switch_tanslated(int x)  
{  
    unsigned xi = x - 100;  
    int result = x;  
    if (xi>6) goto loc_def;  
    goto jt[xi];  
loc_A: result *= 13; goto done;  
loc_B: result += 10; goto done;  
loc_C: result += 11; goto done;  
loc_D: result *= result; goto done;  
loc_def: result = 0;  
done: return result;  
}
```