计算机系统

5. 程序的机器级表示-控制结构

华东师范大学 数据科学与工程学院

2021年09月26日

钱卫宁

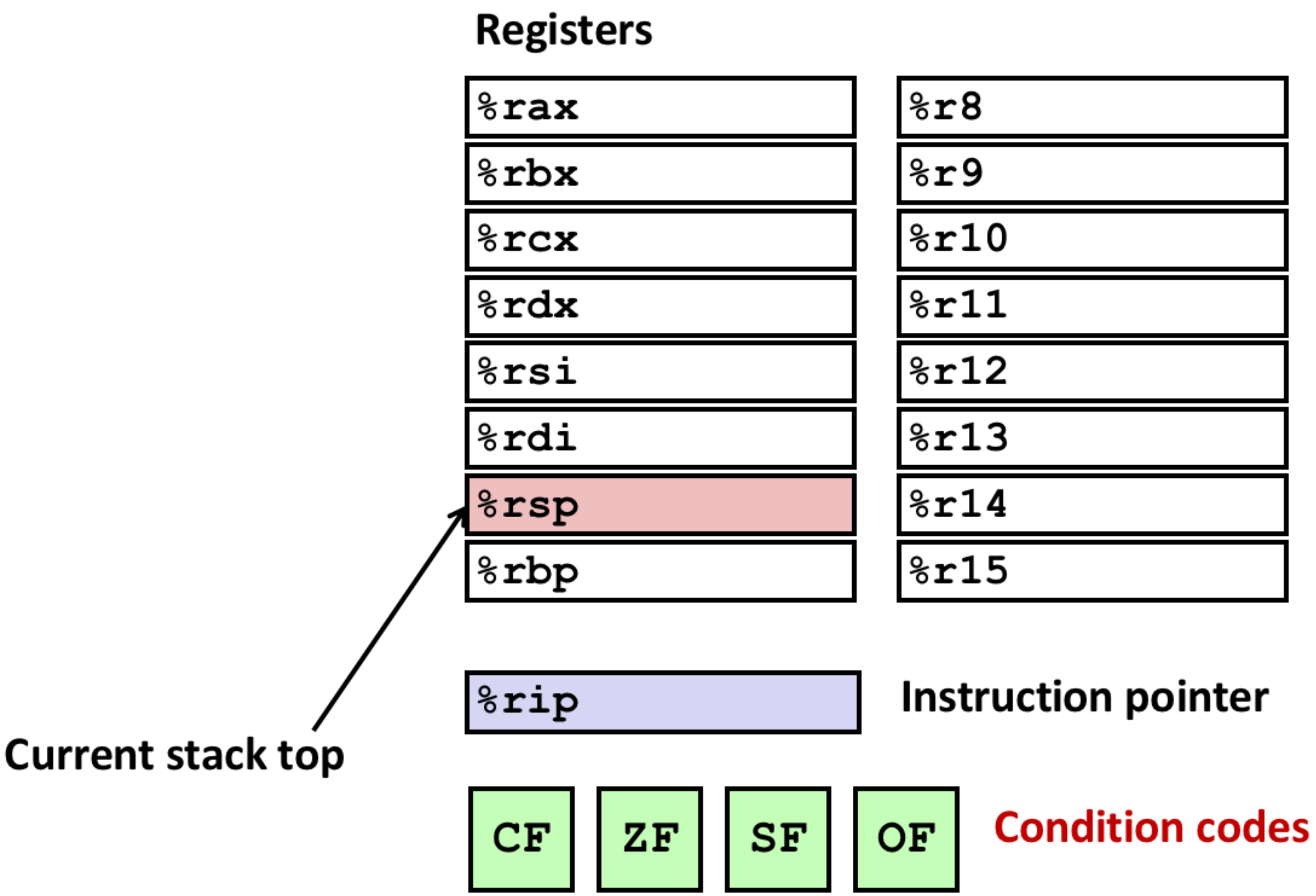
wnqian@dase.ecnu.edu.cn

处理器的状态（x86-64）

当前程序的状态

- 临时数据：如 `%rax, ...`
- 运行时栈：栈顶位置 `%rsp`
- 代码位置：`%rip`
- 最近测试状态：`CF, ZF, SF, OF`

处理器的状态（x86-64）



Registers

%rax	%r8
%rbx	%r9
%rcx	%r10
%rdx	%r11
%rsi	%r12
%rdi	%r13
%rsp	%r14
%rbp	%r15

Current stack top

%rip

Instruction pointer

CF ZF SF OF

Condition codes

条件状态编码

每个编码为一位二进制位寄存器

- CF: Carry Flag (for unsigned)
- ZF: Zero Flag
- SF: Sign Flag (for signed)
- OF: Overflow Flag (for signed)

由算术运算操作*隐式设置（implicitly set）*

注意， `leaq` *不是算术运算操作，不影响条件状态编码*

示例

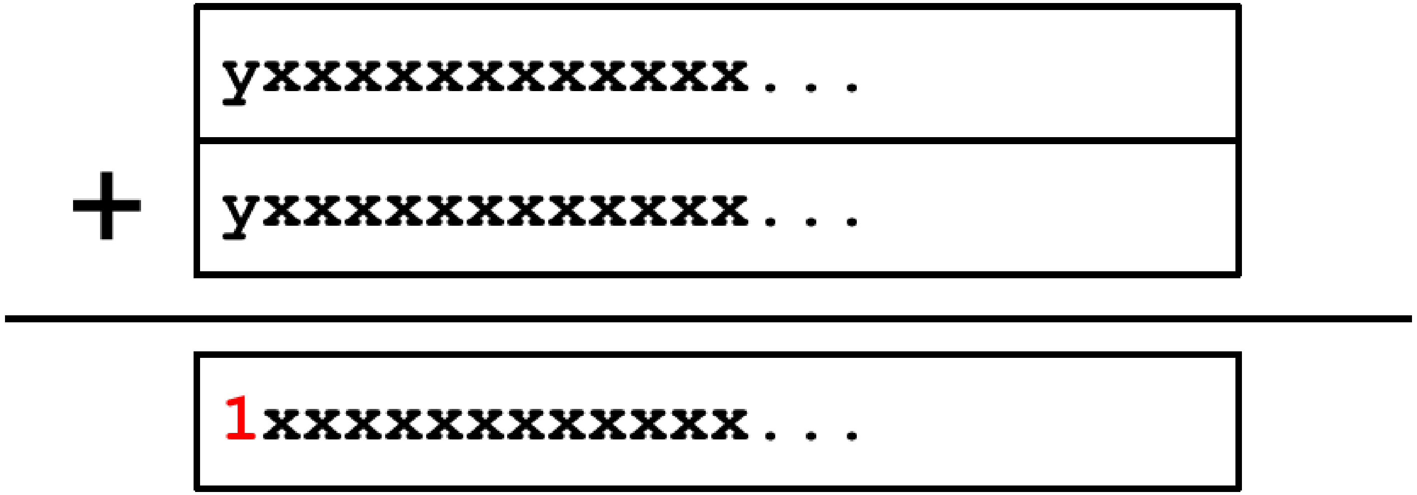
`addq Src,Dest` $(t = a + b)$

- CF set if carry/borrow out from most significant bit (unsigned overflow)
- ZF set if `t == 0`
- SF set if `t < 0` (as signed)
- OF set if two's-complement (signed) overflow `(a>0 && b>0 && t<0) || (a<0 && b<0 && t>=0)`

ZF

000000000000...00000000000

SF



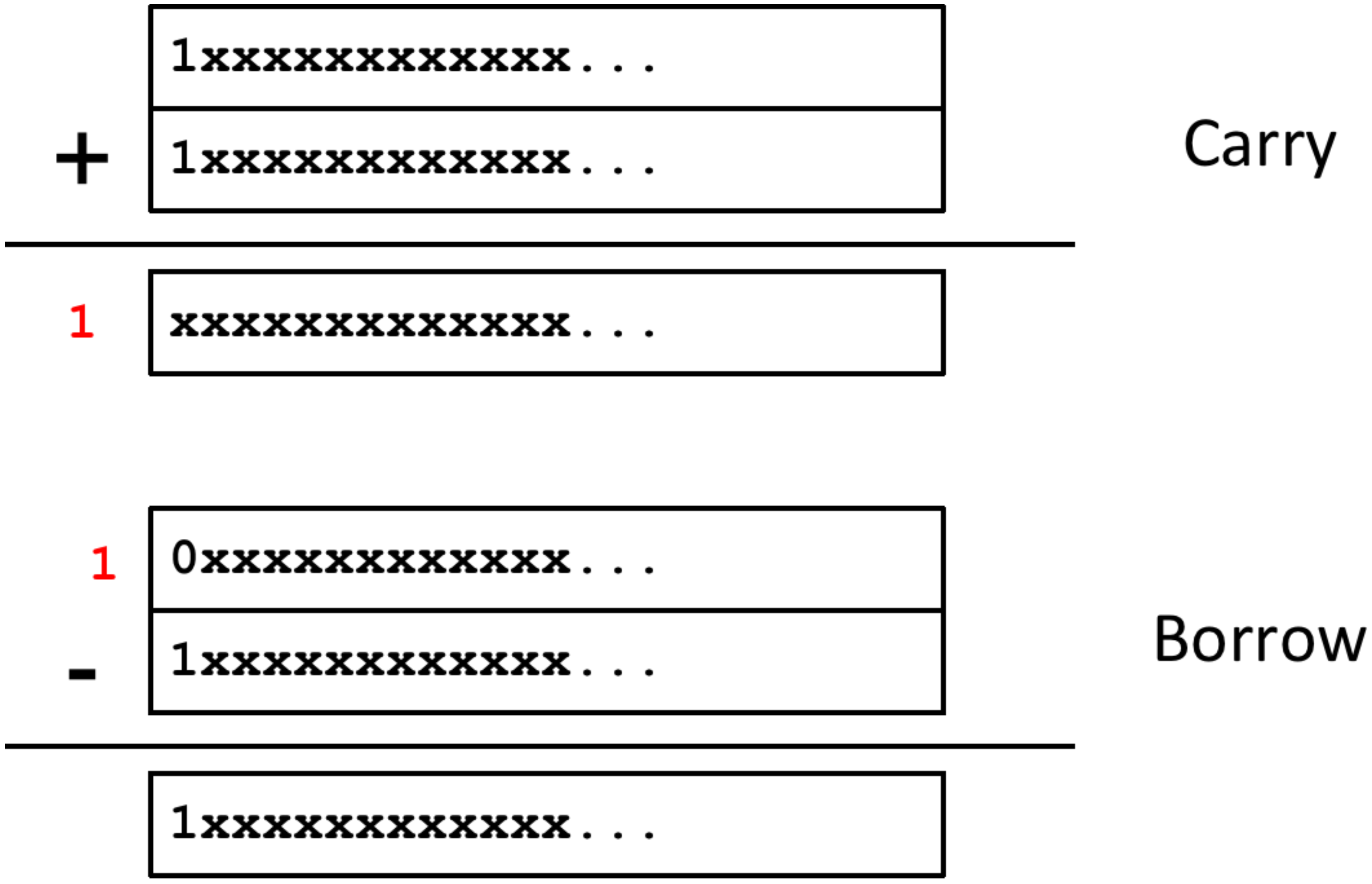
yxxxxxxxxxxxx...

+ yxxxxxxxxxxxx...

1xxxxxxxxxxxx...

For signed arithmetic, this reports when result is a negative number

CF



1xxxxxxxxxxxxx...

+ 1xxxxxxxxxxxxx...

1 xxxxxxxxxxxxxx...

Carry

1 0xxxxxxxxxxxxx...

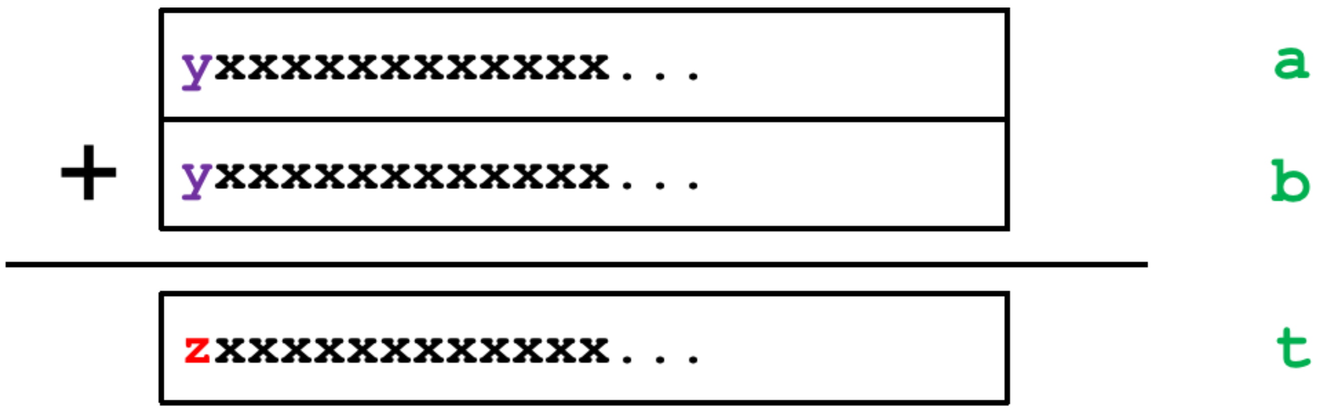
- 1xxxxxxxxxxxxx...

1xxxxxxxxxxxxx...

Borrow

For unsigned arithmetic, this reports overflow

OF



```
  yxxxxxxxxxxxx...      a
+ yxxxxxxxxxxxx...      b
  zxxxxxxxxxxxx...      t
```

$z = \sim y$

```
(a>0 && b>0 && t<0) || (a<0 && b<0 && t>=0)
```

For signed arithmetic, this reports overflow

条件状态编码

由比较指令*显式设置（explicitly set）*

`cmpq Src2, Src1` （`cmpq b, a` 相当于计算$a - b$，但不存储结果）

- CF set if carry/borrow out from most significant bit

  (used for unsigned comparisons)
- ZF set if `a == b`
- SF set if `(a-b) < 0` (as signed)
- OF set if two's-complement (signed) overflow

```
(a>0 && b<0 && (a-b)<0) || (a<0 && b>0 && (a-b)>0
```

条件状态编码

由测试指令*显式设置*（*explicitly set*）

`testq Src2, Src1` （`testq b, a` 相当于计算$a\&b$，但不存储结果）

- ZF set when `a&b == 0`
- SF set when `a&b < 0`

常见用法

- `testq %rax, %rax`
- 两个操作数之一为*掩码*（*mask*）

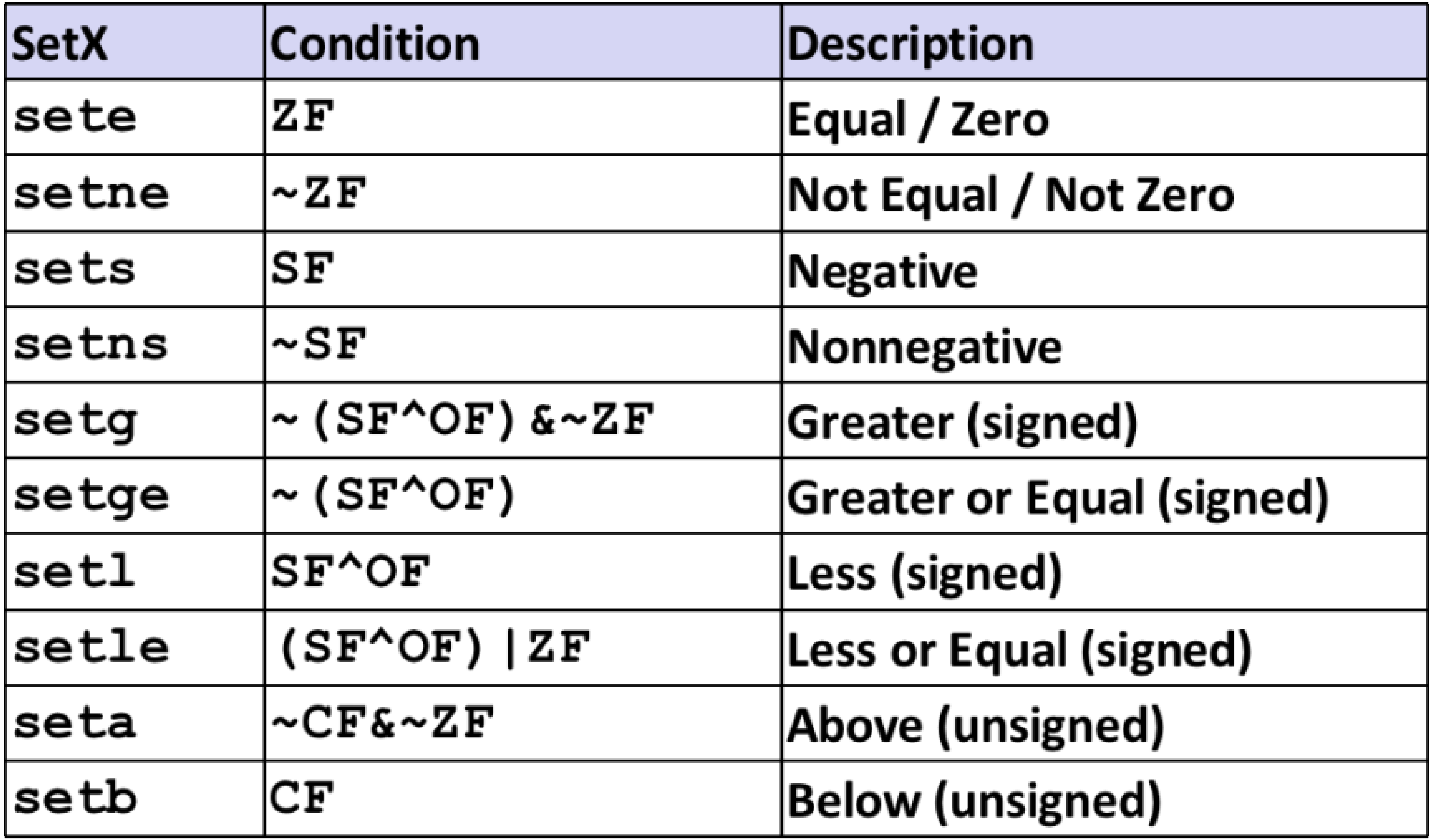
条件状态编码

用设置指令显式读取（explicitly read）

`setX Dest` 根据条件码组合设置 `Dest` 的最低位字节为 0 或者 1

- 并不影响 `Dest` 的剩余 7 个字节

显式读取条件状态编码



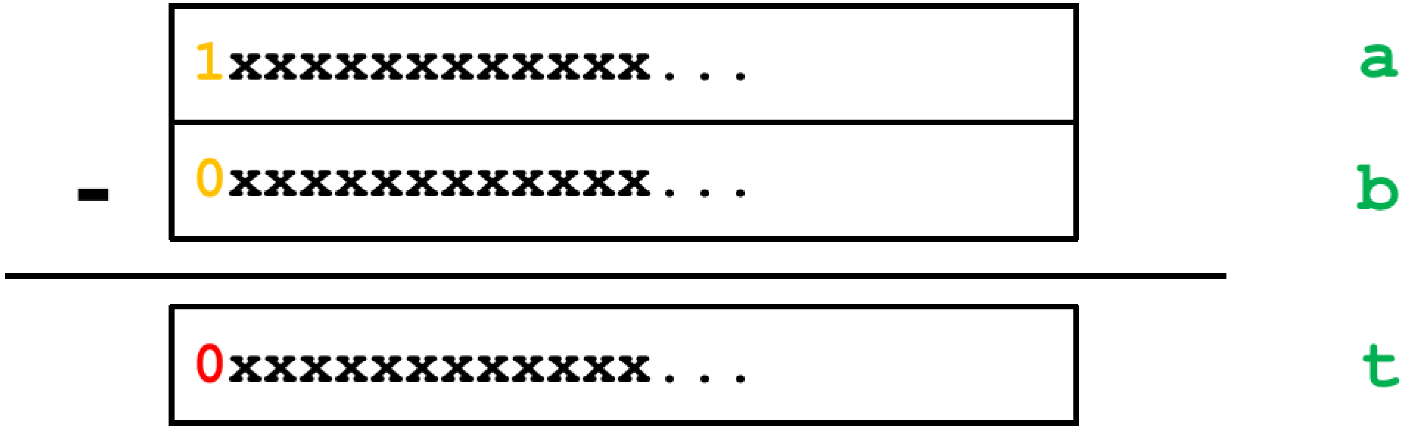
SetX	Condition	Description
sete	ZF	Equal / Zero
setne	~ZF	Not Equal / Not Zero
sets	SF	Negative
setns	~SF	Nonnegative
setg	~(SF^OF)&~ZF	Greater (signed)
setge	~(SF^OF)	Greater or Equal (signed)
setl	SF^OF	Less (signed)
setle	(SF^OF)|ZF	Less or Equal (signed)
seta	~CF&~ZF	Above (unsigned)
setb	CF	Below (unsigned)

setl （signed <） 示例

SF^OF

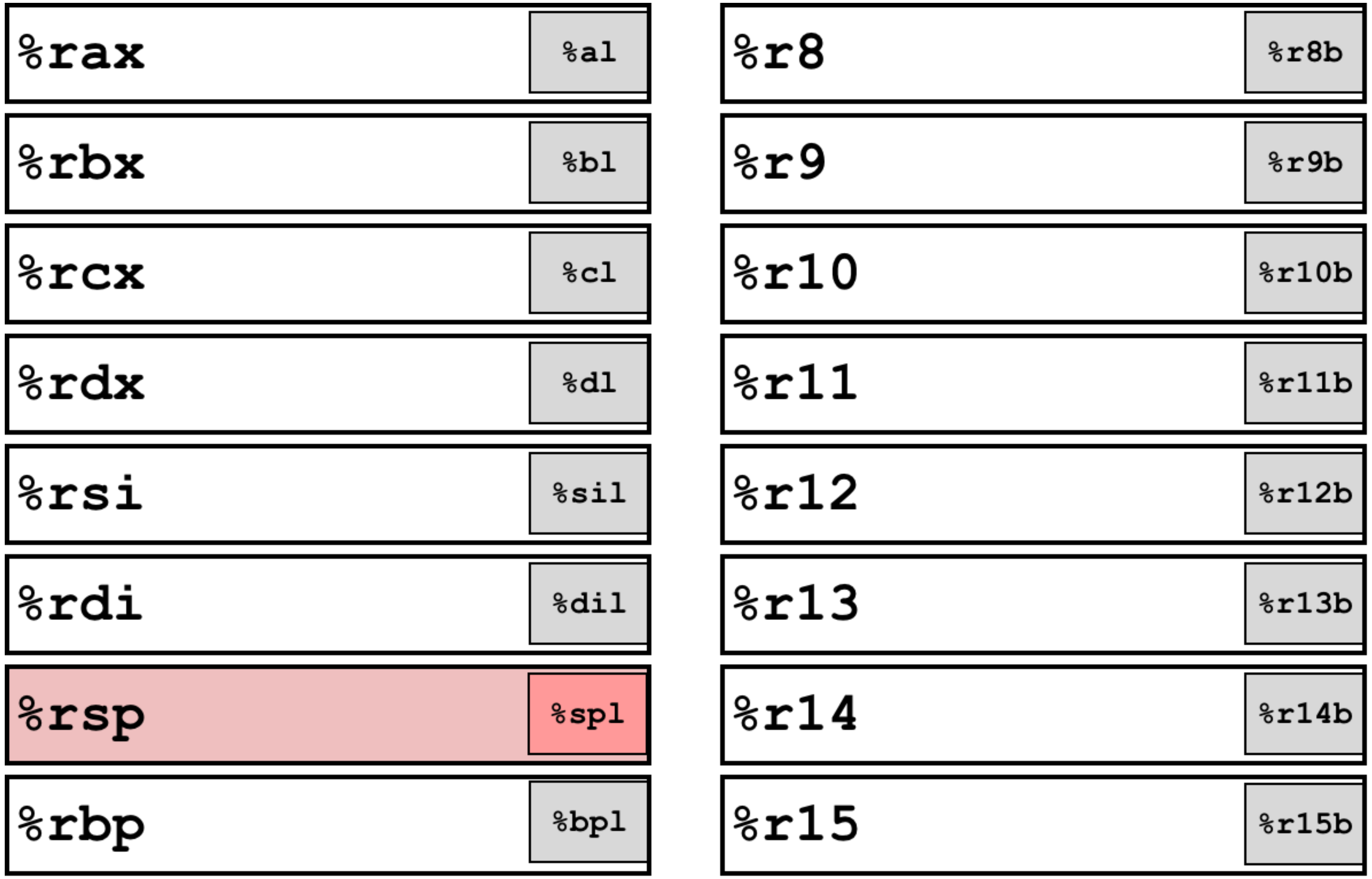
SF	OF	SF ^ OF	Implication
0	0	0	No overflow, so SF implies not <
1	0	1	No overflow, so SF implies <
0	1	1	Overflow, so SF implies negative overflow, i.e. <
1	1	0	Overflow, so SF implies positive overflow, i.e. not <

negative overflow case



```
  1xxxxxxxxxxxxx...   a
- 0xxxxxxxxxxxxx...   b
---------------------
  0xxxxxxxxxxxxx...   t
```

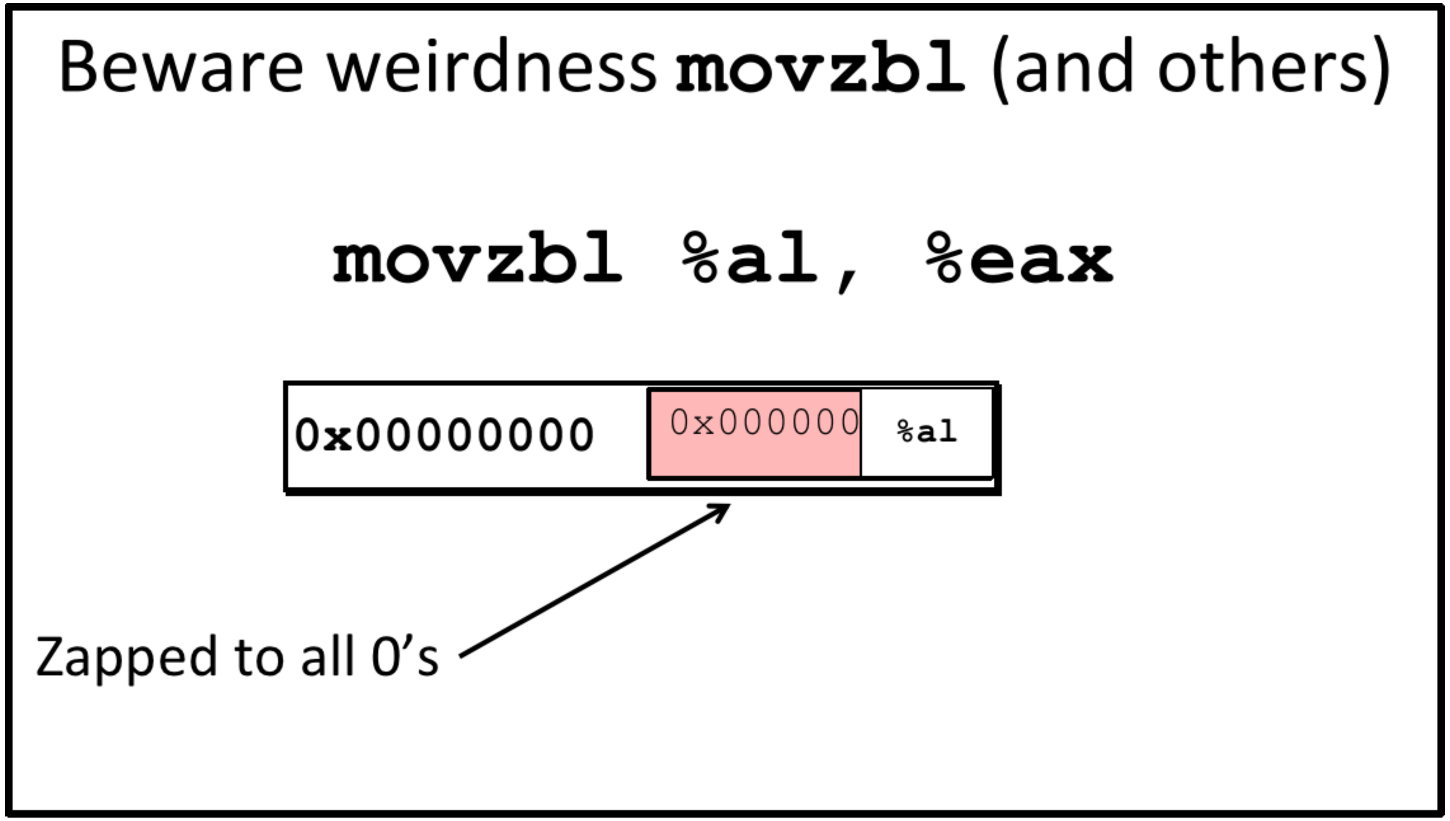
寄存器的最低位字节



%rax	%al	%r8	%r8b
%rbx	%bl	%r9	%r9b
%rcx	%cl	%r10	%r10b
%rdx	%dl	%r11	%r11b
%rsi	%sil	%r12	%r12b
%rdi	%dil	%r13	%r13b
%rsp	%spl	%r14	%r14b
%rbp	%bpl	%r15	%r15b

显式读取条件状态编码典型用法

`setX` 与 `movzbl` 常配合使用（32 位指令，但同时会设置高 32 位为 0）

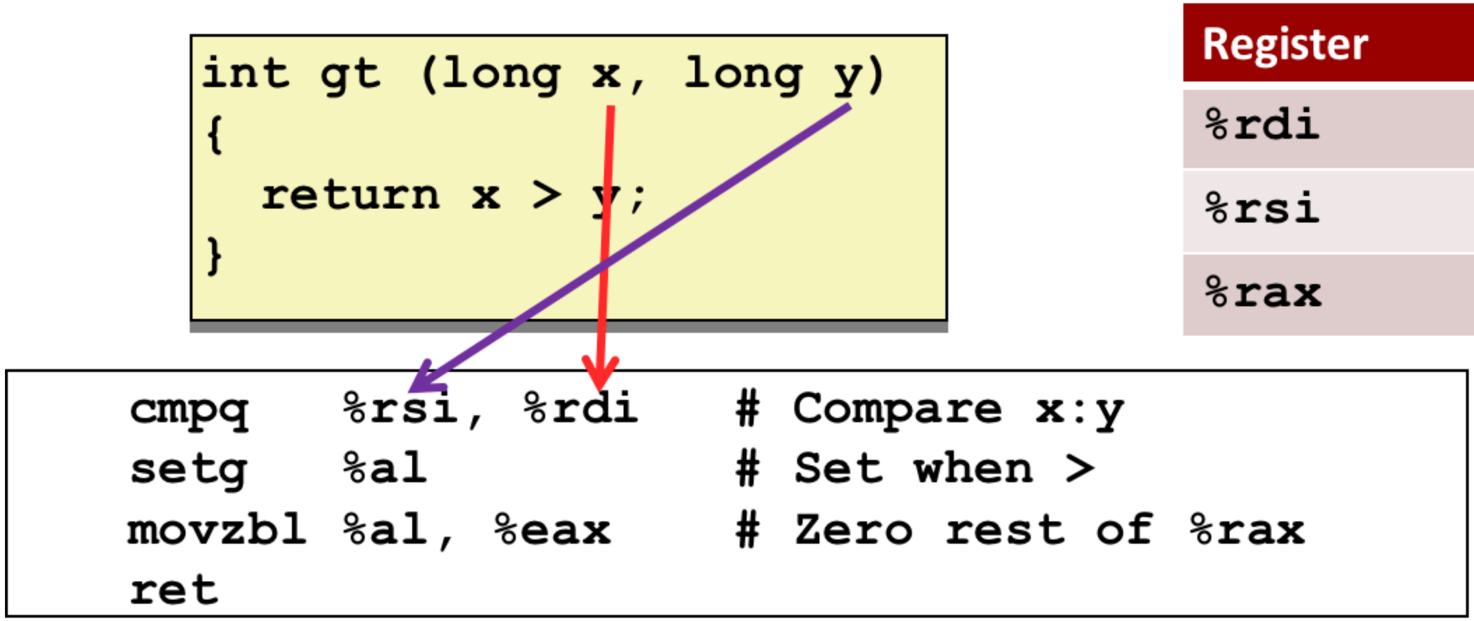


Beware weirdness **movzbl** (and others)

```
movzbl %al, %eax
```

Zapped to all 0’s

显式读取条件状态编码典型用法



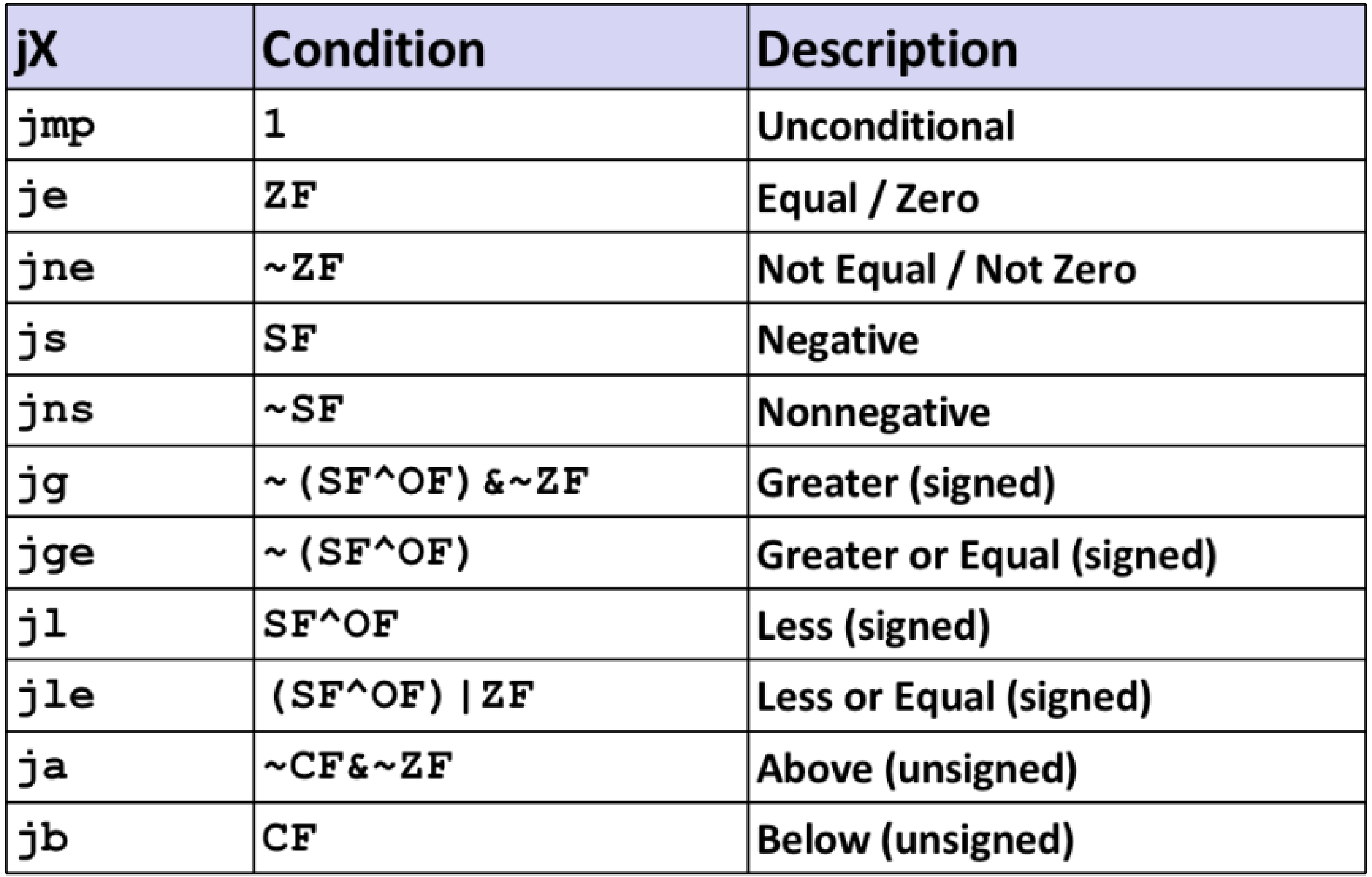
```
int gt (long x, long y)
{
  return x > y;
}
```

```
    cmpq   %rsi, %rdi    # Compare x:y
    setg   %al           # Set when >
    movzbl %al, %eax     # Zero rest of %rax
    ret
```

Register	Use(s)
%rdi	Argument **x**
%rsi	Argument **y**
%rax	Return value

条件跳转

隐式读取条件码，根据条件码组合决定下一条指令跳转的位置



jX	Condition	Description
jmp	1	Unconditional
je	ZF	Equal / Zero
jne	~ZF	Not Equal / Not Zero
js	SF	Negative
jns	~SF	Nonnegative
jg	~(SF^OF)&~ZF	Greater (signed)
jge	~(SF^OF)	Greater or Equal (signed)
jl	SF^OF	Less (signed)
jle	(SF^OF)|ZF	Less or Equal (signed)
ja	~CF&~ZF	Above (unsigned)
jb	CF	Below (unsigned)

条件分枝示例

```
gcc –Og -S –fno-if-conversion control.c
```

```
long absdiff
  (long x, long y)
{
  long result;
  if (x > y)
    result = x-y;
  else
    result = y-x;
  return result;
}
```

```
absdiff:
   cmpq    %rsi, %rdi  # x:y
   jle     .L4
   movq    %rdi, %rax
   subq    %rsi, %rax
   ret
.L4:       # x <= y
   movq    %rsi, %rax
   subq    %rdi, %rax
   ret
```

Register	Use(s)
%rdi	Argument **x**
%rsi	Argument **y**
%rax	Return value

条件分枝示例

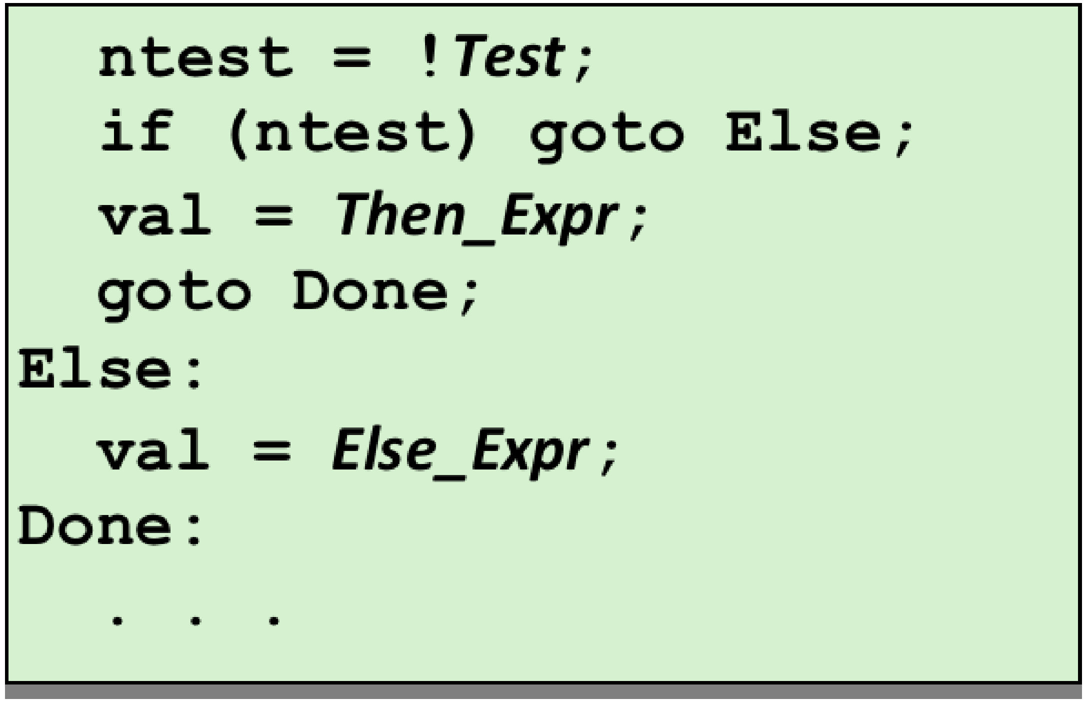
```
long absdiff
  (long x, long y)
{
    long result;
    if (x > y)
        result = x-y;
    else
        result = y-x;
    return result;
}
```

```
long absdiff_j
  (long x, long y)
{
    long result;
    int ntest = x <= y;
    if (ntest) goto Else;
    result = x-y;
    goto Done;
 Else:
    result = y-x;
 Done:
    return result;
}
```

用分枝翻译条件表达式

`val = Test ? Then_Expr : Else_Expr;` （`val = x>y ? x-y : y-x;`）

为then和else表达式创建单独代码块，根据状态编码决定执行哪一部分



```
   ntest = !Test;
   if (ntest) goto Else;
   val = Then_Expr;
   goto Done;
Else:
   val = Else_Expr;
Done:
   . . .
```

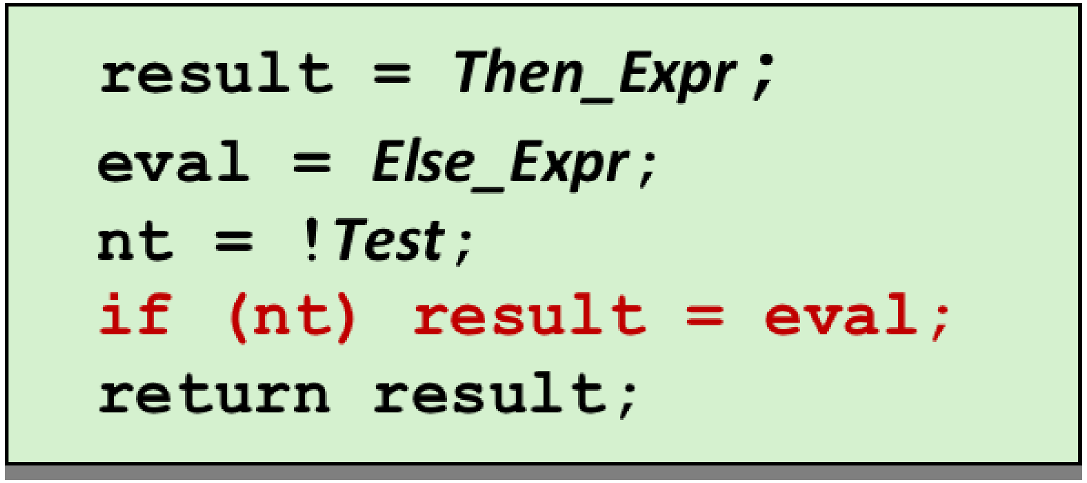
用条件移动指令翻译条件表达式

条件移动（conditional move）指令，支持：`if (Test) Dest = Src;`

- 1995 年后的x86处理器支持
- GCC 在明确安全的情况下会尽量使用该指令
- 分枝会影响流水线执行的指令流
- 条件移动不需要破坏控制流

用条件移动指令翻译条件表达式

`val = Test ? Then_Expr : Else_Expr;` （`val = x>y ? x-y : y-x;`）



```
result = Then_Expr;
eval = Else_Expr;
nt = !Test;
if (nt) result = eval;
return result;
```

用条件移动指令翻译条件表达式示例

```
long absdiff
  (long x, long y)
{
    long result;
    if (x > y)
        result = x-y;
    else
        result = y-x;
    return result;
}
```

Register	Use(s)
%rdi	Argument **x**
%rsi	Argument **y**
%rax	Return value

```
absdiff:
   movq     %rdi, %rax  # x
   subq     %rsi, %rax  # result = x-y
   movq     %rsi, %rdx
   subq     %rdi, %rdx  # eval = y-x
   cmpq     %rsi, %rdi  # x:y
   cmovle   %rdx, %rax  # if <=, result = eval
   ret
```

When is
this bad?

条件移动的问题

- 计算代价高： `val = Test(x) ? Hard1(x) : Hard2(x);`
    - 两个表达式都需要计算，只适用于两个表达式都很简单的情况
- 不安全的计算： `val = p ? *p : 0;`
    - 可能会有非预期的结果
- 副作用（错误的计算结果）： `val = x > 0 ? x*=7 : x+=3;`
    - 不应有副作用（两个表达式相互干扰）

do-while 循环

C Code

```
long pcount_do
   (unsigned long x) {
   long result = 0;
   do {
      result += x & 0x1;
      x >>= 1;
   } while (x);
   return result;
}
```

Goto Version

```
long pcount_goto
   (unsigned long x) {
   long result = 0;
 loop:
   result += x & 0x1;
   x >>= 1;
   if(x) goto loop;
   return result;
}
```

do-while 循环

```
long pcount_goto
  (unsigned long x) {
  long result = 0;
 loop:
  result += x & 0x1;
  x >>= 1;
  if(x) goto loop;
  return result;
}
```

Register	Use(s)
%rdi	Argument **x**
%rax	**result**

```
    movl    $0, %eax     #   result = 0
 .L2:                        # loop:
    movq    %rdi, %rdx
    andl    $1, %edx     #   t = x & 0x1
    addq    %rdx, %rax   #   result += t
    shrq    %rdi         #   x >>= 1
    jne     .L2          #   if(x) goto loop
    rep; ret
```

do-while 循环的一般翻译

C Code

```
do
   Body
   while (Test);
```

Goto Version

```
loop:
   Body
   if (Test)
      goto loop
```

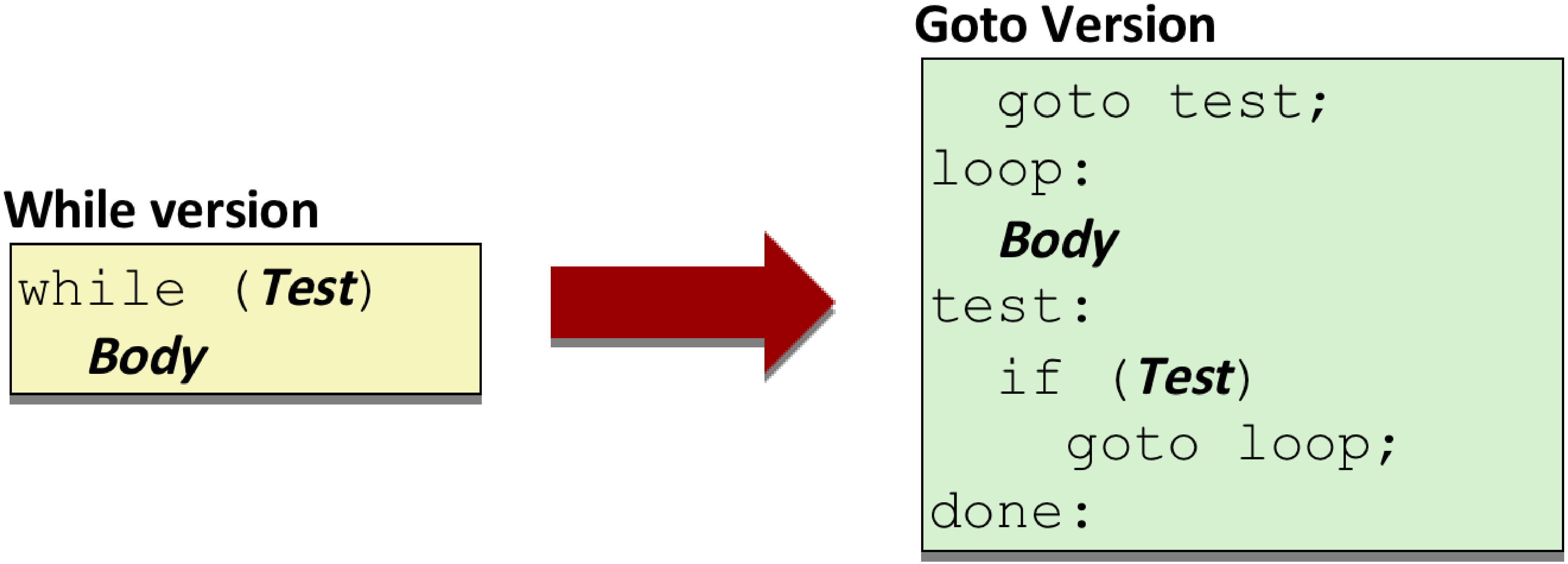
- **Body:** {

  Statement$_1$;
  Statement$_2$;
  …
  Statement$_n$;

  }

while 循环的一般翻译

Jump-to-middle，如果编译时使用 `-Og` 会采用这种翻译



While version

```
while (Test)
    Body
```

Goto Version

```
  goto test;
loop:
  Body
test:
  if (Test)
    goto loop;
done:
```

Jump-to-middle 示例

C Code

```
long pcount_while
   (unsigned long x) {
   long result = 0;
   while (x) {
      result += x & 0x1;
      x >>= 1;
   }
   return result;
}
```

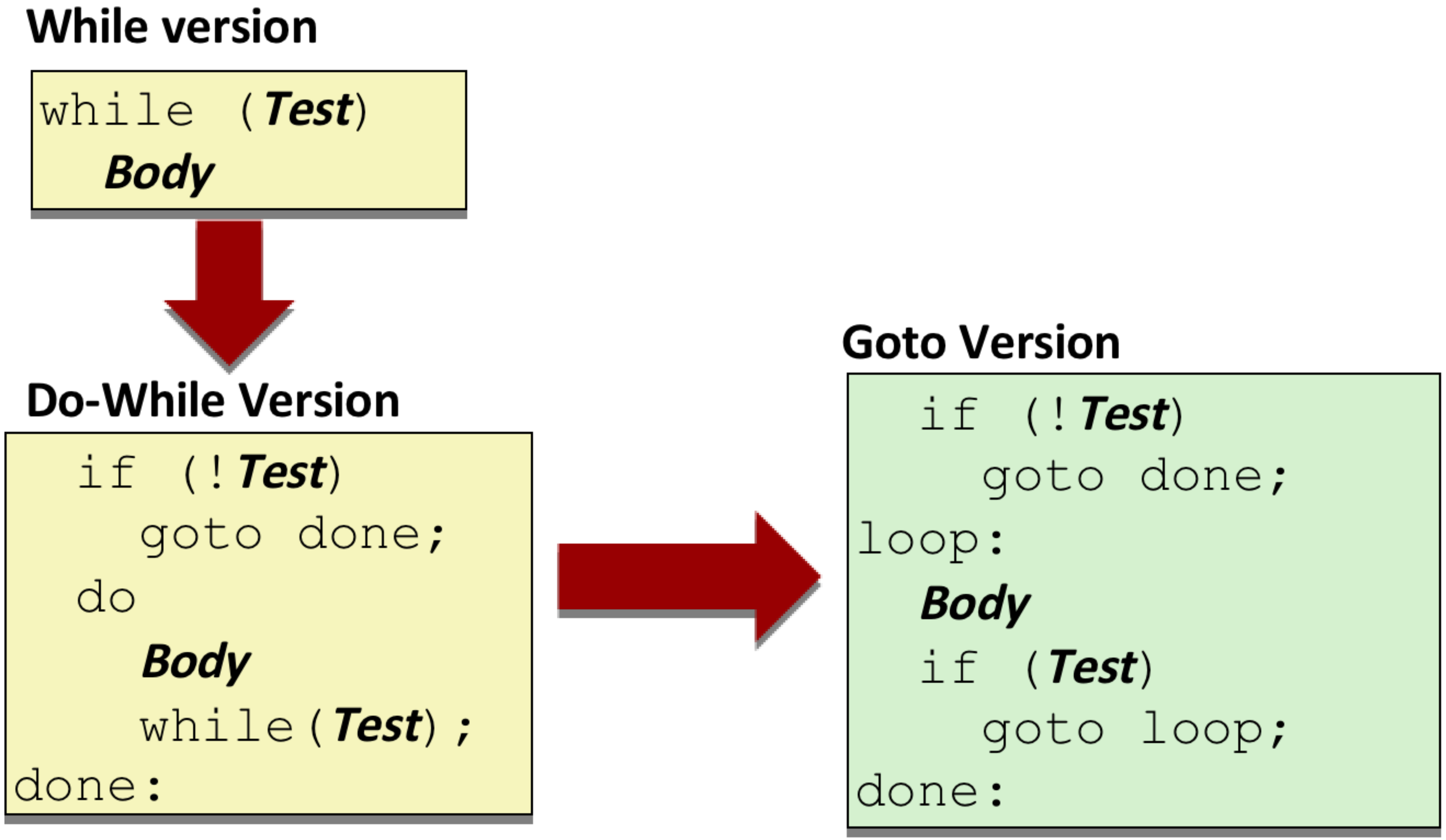
Jump to Middle

```
long pcount_goto_jtm
   (unsigned long x) {
   long result = 0;
   goto test;
 loop:
   result += x & 0x1;
   x >>= 1;
 test:
   if(x) goto loop;
   return result;
}
```

第一次循环会先跳转到 `test`

while 循环的一般翻译

*do-while*转换，如果编译时使用 `-01` 会采用这种翻译



While version

```
while (Test)
  Body
```

Do-While Version

```
  if (!Test)
    goto done;
  do
    Body
    while(Test);
done:
```

Goto Version

```
  if (!Test)
    goto done;
loop:
  Body
  if (Test)
    goto loop;
done:
```

Do-while 转换示例

C Code

```
long pcount_while
  (unsigned long x) {
  long result = 0;
  while (x) {
    result += x & 0x1;
    x >>= 1;
  }
  return result;
}
```

Do-While Version

```
long pcount_goto_dw
  (unsigned long x) {
  long result = 0;
  if (!x) goto done;
 loop:
  result += x & 0x1;
  x >>= 1;
  if(x) goto loop;
 done:
  return result;
}
```

- 循环前进行条件判断确认是否进入循环
- 与*jump-to-middle*孰优孰劣？ （when & why?）

for 循环的翻译

```
for (Init; Test; Update)
  Body;
```

……

```
#define WSIZE 8*sizeof(int)
long pcount_for
  (unsigned long x)
{
  size_t i;
  long result = 0;
  for (i = 0; i < WSIZE; i++)
  {
    unsigned bit =
      (x >> i) & 0x1;
    result += bit;
  }
  return result;
}
```

Init

```
i = 0
```

Test

```
i < WSIZE
```

Update

```
i++
```

Body

```
{
  unsigned bit =
      (x >> i) & 0x1;
  result += bit;
}
```

for 循环的翻译

```
for (Init; Test; Update)
  Body;
```

等价于

```
Init;
while (Test) {
  Body;
  Update;
}
```

for 循环翻译示例

```
#define WSIZE 8*sizeof(int)
long pcount_for
   (unsigned long x)
{
  size_t i;
  long result = 0;
  for (i = 0; i < WSIZE; i++)
  {
    unsigned bit =
       (x >> i) & 0x1;
    result += bit;
  }
  return result;
}
```

Init

```
i = 0
```

Test

```
i < WSIZE
```

Update

```
i++
```

Body

```
{
  unsigned bit =
      (x >> i) & 0x1;
  result += bit;
}
```

```
long pcount_for_while
   (unsigned long x)
{
  size_t i;
  long result = 0;
  i = 0;
  while (i < WSIZE)
  {
    unsigned bit =
       (x >> i) & 0x1;
    result += bit;
    i++;
  }
  return result;
}
```

for 循环的进一步展开

C Code

```
long pcount_for
  (unsigned long x)
{
  size_t i;
  long result = 0;
  for (i = 0; i < WSIZE; i++)
  {
    unsigned bit =
      (x >> i) & 0x1;
    result += bit;
  }
  return result;
}
```

Goto Version

```
long pcount_for_goto_dw
  (unsigned long x) {
  size_t i;
  long result = 0;
  i = 0;                    Init
  ~~if (!(i < WSIZE))~~
    ~~goto done;~~          ! Test
 loop:
  {
    unsigned bit =
      (x >> i) & 0x1;       Body
    result += bit;
  }
  i++;  Update
  if (i < WSIZE)            Test
    goto loop;
 done:
  return result;
}
```

switch 分枝

```
long my_switch
   (long x, long y, long z)
{
    long w = 1;
    switch(x) {
    case 1:
        w = y*z;
        break;
    case 2:
        w = y/z;
        /* Fall Through */
    case 3:
        w += z;
        break;
    case 5:
    case 6:
        w -= z;
        break;
    default:
        w = 2;
    }
    return w;
}
```

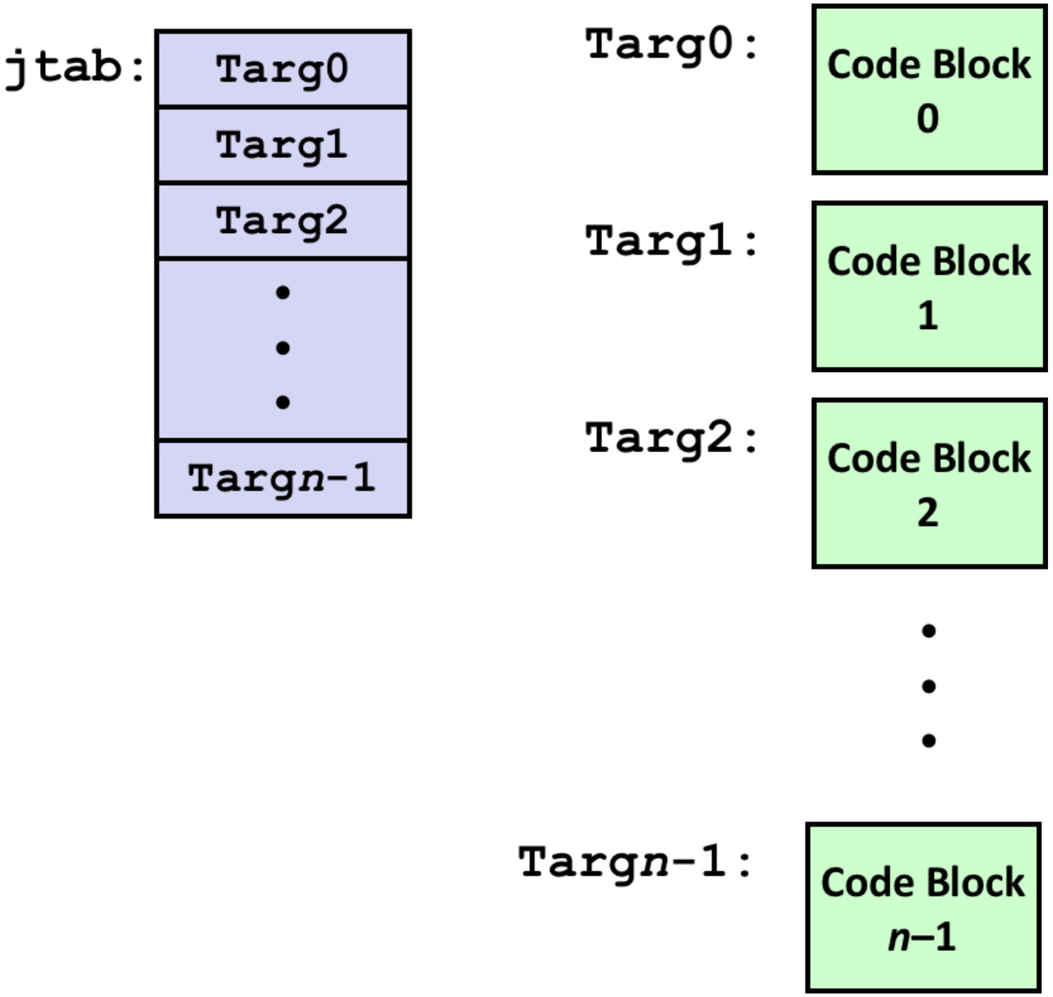
跳转表（Jump Table）

Switch Form

```
switch(x) {
  case val_0:
    Block 0
  case val_1:
    Block 1
    • • •
  case val_n-1:
    Block n–1
}
```

Translation (Extended C)

```
goto *JTab[x];
```



jtab:

Targ0
Targ1
Targ2
• • •
Targ*n*-1

Targ0: Code Block 0

Targ1: Code Block 1

Targ2: Code Block 2

•
•
•

Targ*n*-1: Code Block *n*–1

switch 语句的翻译

```
long my_switch(long x, long y, long z)
{
    long w = 1;
    switch(x) { . . . }
    return w;
}
```

Setup

```
my_switch:
    movq    %rdx, %rcx
    cmpq    $6, %rdi    # x:6
    ja      .L8
    jmp     *.L4(,%rdi,8)
```

What range of values takes default?

Register	Use(s)
`%rdi`	Argument **`x`**
`%rsi`	Argument **`y`**
`%rdx`	Argument **`z`**
`%rax`	Return value

Note that **`w`** not initialized here

switch 语句的翻译

```
long my_switch(long x, long y, long z)
{
    long w = 1;
    switch(x) { . . . }
    return w;
}
```

Setup

```
my_switch:
    movq    %rdx, %rcx
    cmpq    $6, %rdi      # x:6
    ja      .L8           # use default
    jmp     *.L4(,%rdi,8)  # goto *Jtab[x]
```

Indirect jump

Jump table

```
.section    .rodata
  .align 8
.L4:
  .quad     .L8    # x = 0
  .quad     .L3    # x = 1
  .quad     .L5    # x = 2
  .quad     .L9    # x = 3
  .quad     .L8    # x = 4
  .quad     .L7    # x = 5
  .quad     .L7    # x = 6
```

预习要求

阅读至3.7结束

抽时间仔细/反复阅读第一章