CSSE132
Introduction to Computer Systems

15 : Control flow

March 6, 2013

Today: Control flow

- **Jump instructions**
- **If,else examples**
- **Conditional move**
- **Loops**
  - do-while Loop conversion
  - Other loops

Jumping

- **jX Instructions**
  - Jump to different part of code depending on condition codes

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)

Jump targets

- **Unconditional jump**
  - Target can be absolute (direct memory reference)
  - Target can be PC-relative (based on current PC)
  - Target can be indirectly referenced from register/memory

- **Conditional jumps**
  - Target can be absolute (direct memory reference)
  - Target can be PC-relative (based on current PC)

- **In assembler**
  - Place label at target (like `label:`)
  - Jump instruction specifies label
  - Assembler and linker compute final target

Today: Control flow

- Jump instructions
- If,else examples
- Conditional move
- Loops
  - do-while Loop conversion
  - Other loops

Conditional Branch Example

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

```
absdiff:
    pushl   %ebp                     Setup
    movl    %esp, %ebp
    movl    8(%ebp), %edx
    movl    12(%ebp), %eax
    cmpl    %eax, %edx               Body1
    jle     .L6
    subl    %eax, %edx
    movl    %edx, %eax               Body2a
    jmp .L7
.L6:
    subl %edx, %eax                  Body2b
.L7:
    popl %ebp
    ret                              Finish
```

Conditional Branch Example (Cont.)

```
int goto_ad(int x, int y)
{
  int result;
  if (x <= y) goto Else;
  result = x-y;
  goto Exit;
Else:
  result = y-x;
Exit:
  return result;
}
```

- **C allows "goto" as means of transferring control**
  - Closer to machine-level programming style
- **Generally considered bad coding style**

```
absdiff:
    pushl   %ebp                    Setup
    movl    %esp, %ebp
    movl    8(%ebp), %edx
    movl    12(%ebp), %eax
    cmpl    %eax, %edx
    jle     .L6                     Body1
    subl    %eax, %edx
    movl    %edx, %eax              Body2a
    jmp .L7
.L6:
    subl %edx, %eax                 Body2b
.L7:
    popl %ebp
    ret                             Finish
```

Conditional Branch Example (Cont.)

```
int goto_ad(int x, int y)
{
  int result;
  if (x <= y) goto Else;
  result = x-y;
  goto Exit;
Else:
  result = y-x;
Exit:
  return result;
}
```

```
absdiff:
   pushl   %ebp                      Setup
   movl    %esp, %ebp
   movl    8(%ebp), %edx
   movl    12(%ebp), %eax
   cmpl    %eax, %edx
   jle     .L6                       Body1
   subl    %eax, %edx
   movl    %edx, %eax                Body2a
   jmp .L7
.L6:
   subl %edx, %eax                   Body2b
.L7:
   popl %ebp
   ret                               Finish
```

Conditional Branch Example (Cont.)

```
int goto_ad(int x, int y)
{
  int result;
  if (x <= y) goto Else;
  result = x-y;
  goto Exit;
Else:
  result = y-x;
Exit:
  return result;
}
```

```
absdiff:
   pushl   %ebp
   movl    %esp, %ebp              Setup
   movl    8(%ebp), %edx
   movl    12(%ebp), %eax
   cmpl    %eax, %edx
   jle     .L6                     Body1
   subl    %eax, %edx
   movl    %edx, %eax              Body2a
   jmp .L7
.L6:
   subl %edx, %eax                 Body2b
.L7:
   popl %ebp
   ret                             Finish
```

Conditional Branch Example (Cont.)

```
int goto_ad(int x, int y)
{
  int result;
  if (x <= y) goto Else;
  result = x-y;
  goto Exit;
Else:
  result = y-x;
Exit:
  return result;
}
```

```
absdiff:
    pushl   %ebp
    movl    %esp, %ebp             Setup
    movl    8(%ebp), %edx
    movl    12(%ebp), %eax
    cmpl    %eax, %edx
    jle     .L6                    Body1
    subl    %eax, %edx
    movl    %edx, %eax             Body2a
    jmp .L7
.L6:
    subl %edx, %eax                Body2b
.L7:
    popl %ebp
    ret                            Finish
```

General Conditional Expression Translation

C Code

```
val = Test ? Then_Expr : Else_Expr;
```

```
val = x>y ? x-y : y-x;
```

Goto Version

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

- Test is expression returning integer
  - = 0 interpreted as false
  - ≠ 0 interpreted as true
- Create separate code regions for then & else expressions
- Execute appropriate one

Today: Control flow

- **Jump instructions**
- **If,else examples**
- **Conditional move**
- **Loops**
  - do-while Loop conversion
  - Other loops

Using Conditional Moves

- **Conditional Move Instructions**
  - Instruction supports:

    if (Test) Dest ← Src
  - Supported in post-1995 x86 processors
  - GCC does not always use them
    - Wants to preserve compatibility with ancient processors
    - Enabled for x86-64
    - Use switch `-march=686` for IA32
- **Why?**
  - Branches are very disruptive to instruction flow through pipelines
  - Conditional move do not require control transfer

C Code

```
val = Test
   ? Then_Expr
   : Else_Expr;
```

Goto Version

```
  tval = Then_Expr;
  result = Else_Expr;
  t = Test;
  if (t) result = tval;
  return result;
```

Conditional Move Example: x86-64

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

x in %edi

y in %esi

```
absdiff:
  movl    %edi, %edx
  subl    %esi, %edx   # tval = x-y
  movl    %esi, %eax
  subl    %edi, %eax   # result = y-x
  cmpl    %esi, %edi   # Compare x:y
  cmovg   %edx, %eax   # If >, result = tval
  ret
```

Bad Cases for Conditional Move

Expensive Computations

```
val = Test(x) ? Hard1(x) : Hard2(x);
```

- **Both values get computed**
- **Only makes sense when computations are very simple**

Risky Computations

```
val = p ? *p : 0;
```

- Both values get computed
- May have undesirable effects

Computations with side effects

```
val = x > 0 ? x*=7 : x+=3;
```

- Both values get computed
- Must be side-effect free

Today: Control flow

- **Jump instructions**
- **If,else examples**
- **Conditional move**
- **Loops**
  - do-while Loop conversion
  - Other loops

“Do-While” Loop Example

C Code

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

Goto Version

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

- **Count number of 1’s in argument x (“popcount”)**
- **Use conditional branch to either continue looping or to exit loop**

"Do-While" Loop Compilation

Goto Version

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

- **Registers:**

%edx	x
%ecx	result

```
  movl   $0, %ecx       #   result = 0
.L2:                    # loop:
  movl   %edx, %eax
  andl   $1, %eax       #   t = x & 1
  addl   %eax, %ecx     #   result += t
  shrl   %edx           #   x >>= 1
  jne    .L2            #   If !0, goto loop
```

General “Do-While” Translation

C Code

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

Goto Version

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

- **Body:** {

  $\texttt{Statement}_1$;
  $\texttt{Statement}_2$;
  …
  $\texttt{Statement}_n$;
  }

- **Test returns integer**
  - = 0 interpreted as false
  - ≠ 0 interpreted as true

Today: Control flow

- **Jump instructions**
- **If,else examples**
- **Conditional move**
- **Loops**
  - do-while Loop conversion
  - Other loops

“While” Loop Example

C Code

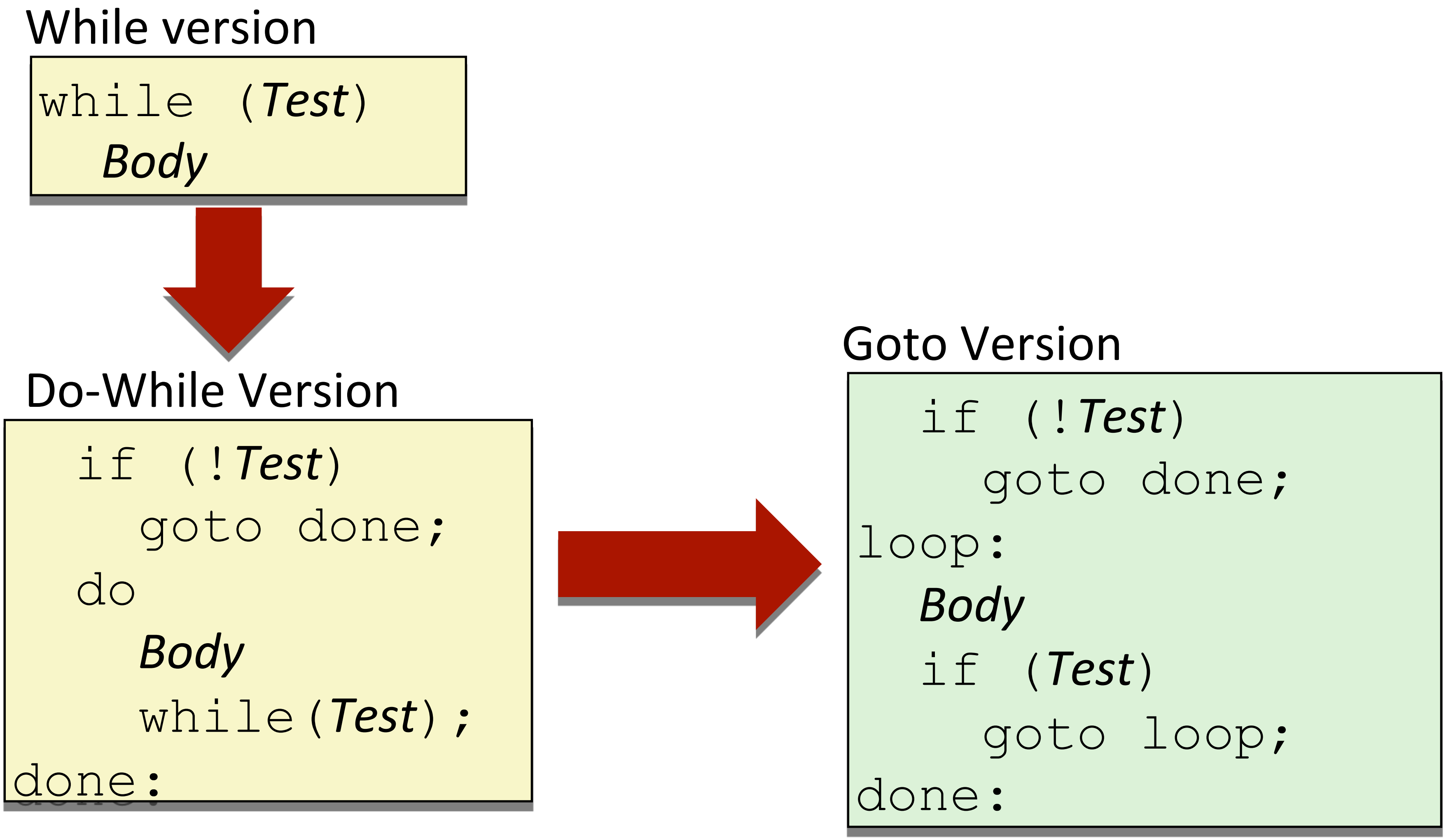
```
int pcount_while(unsigned x)  {
  int result = 0;
  while (x)  {
    result += x & 0x1;
    x >>= 1;
  }
  return result;
}
```

Goto Version

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

- **Is this code equivalent to the do-while version?**

General “While” Translation



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:
```

“For” Loop Example

C Code

```
#define WSIZE 8*sizeof(int)
int pcount_for(unsigned x) {
  int i;
  int result = 0;
  for (i = 0; i < WSIZE; i++) {
    unsigned mask = 1 << i;
    result += (x & mask) != 0;
  }
  return result;
}
```

- Is this code equivalent to other versions?

“For” Loop Form

General Form

```
for (Init; Test; Update )

    Body
```

```
for (i = 0; i < WSIZE; i++) {
    unsigned mask = 1 << i;
    result += (x & mask) != 0;
  }
```

Init

```
i = 0
```

Test

```
i < WSIZE
```

Update

```
i++
```

Body

```
{
  unsigned mask = 1 << i;
  result += (x & mask) != 0;
}
```

“For” Loop → While Loop

For Version

```
for (Init; Test; Update )

          Body
```

While Version

```
Init;

while (Test) {

    Body

    Update;

}
```

“For” Loop → … → Goto

For Version

```
for  (Init;  Test;  Update )

      Body
```

While Version

```
Init;

while  (Test )  {

      Body

      Update;

}
```

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

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

“For” Loop Conversion Example

C Code

```
#define WSIZE 8*sizeof(int)
int pcount_for(unsigned x) {
  int i;
  int result = 0;
  for (i = 0; i < WSIZE; i++) {
    unsigned mask = 1 << i;
    result += (x & mask) != 0;
  }
  return result;
}
```

- Initial test can be optimized away

Goto Version

```
int pcount_for_gt(unsigned x) {
  int i;
  int result = 0;
  i = 0;                         Init
  ~~if (!(i < WSIZE))~~          ! Test
    ~~goto done;~~
 loop:
  {                              Body
    unsigned mask = 1 << i;
    result += (x & mask) != 0;
  }
  i++;   Update
  if (i < WSIZE)   Test
    goto loop;
 done:
  return result;
}
```