University of Arizona

CSc 340

Foundations of Computer Systems

Christian Collberg

February 6, 2001

Simple Control Flow

Control Flow Constructs

- The order in which a program executes instructions is called its control flow. Most instructions simply increment the PC, so that the instructions are executed in-order. This is called sequential control flow.
- High-level languages (HLLs) have a variety of control flow constructs that modify the control flow, such as while loops, for loops, do-while loops, repeat-until loops, if-then-else, switch-case, try-catch, procedure/subroutine/function calls, etc.
- The MIPS has none of these. It has:

MIPS Control Flow

1. Unconditional branch:
   - `PC := PC +` 16-bit constant
   - I.e., move forward/backward a constant number of instructions.
   - The HLL equivalent (not in Java) is

```
goto label
...
label:
```

   - Note that once you goto somewhere, you never come back unless you goto back to where you came from (it's not a procedure call).

MIPS Control Flow ...

2. Conditional branch:
   - `if (cond) PC := PC +` 16-bit constant
   - I.e., if a condition is true, change the PC by a constant number of instructions. If not, go to the next instruction.
   - The HLL equivalent (not in Java) is

```
if (cond) goto label
...
label:
```

   This is called a conditional goto.

Slide 7–4

MIPS Control Flow . . .

3. Unconditional jump: set the PC to an address (26-bit field). This is subtly different from an unconditional branch.
4. ”and link” variations:
   - These are variations of branch and jump instructions that can be used to implement procedure calls.
   - They differ from the standard versions in that they allow you to come back to where you left off when the procedure was invoked. More on this later.

Slide 7–5

Conditions

- By ”condition” we mean a simple expression (no ”and” or ”or”) based on the value in one or more registers, e.g.

```
r4 > 0
r2 == r3
```

  A simple expression is true if its value is non-zero.

- Fortunately, with a little cleverness we can construct the familiar HLL constructs using what the MIPS gives us. This is essentially what a HLL compiler does.

Slide 7–6

If-then

- Suppose you have an if-then construct in a HLL:

```
if (C) then
   T
A
```

- `C` is a simple condition, `T` is the code to execute if `C` is true, and `A` is the code to execute after the if-then.
- To implement this in MIPS assembly code we have to use conditional branches, so it will look a bit like this:

```
if (not C) then goto Alab
T
Alab:  A
```

Slide 7–7

If-then . . .

- The following also works, but is a bit more convoluted:

```
if (C) then goto Tlab
goto Alab
Tlab:  T
Alab:  A
```

- Example:

```
if (a == 0) then
   b = 1
a = 0
```

Slide 7–8

If-then . . .

- Converting to conditional gotos:

```
if (a != 0) then goto A
b = 1
A: a = 0
```

- Assume `a` is in `s0` and `b` is in `s1`. The corresponding MIPS assembly is:

```
      bne   $s0, $zero, A
      li    $s1, 1
A:    li    $s0, 0
```

Slide 7–9

If-then-else

- HLL:

```
if (C) then
   T
else
   F
A
```

- Conditional gotos:

```
if (not C) then goto Flab
T
goto Alab
Flab:  F
Alab:
```

Slide 7–10

If-then-else — Example

- HLL:

```
if (a == 0) then b = 1
else b = 0
a = 0
```

Conditional gotos	MIPS
`if (a != 0) then goto F`	`bne   $s0, $zero, F`
`b = 1`	`li    $s1, 1`
`goto A`	`b     A`
`F: b = 0`	`F: li    $s1, 0`
`A: a = 0`	`A: li    $s0, 0`

Slide 7–11

While loops

- HLL:

```
while (C)
   B
A
```

- Conditional gotos:

```
top:  if (not C) goto Alab
B
goto top
Alab:  A
```

Slide 7–12

While loops — Example

- HLL:

```
a = 10
while (a < 100)
   a = a + 10
```

Conditional gotos	MIPS
`a = 10`	`      li    $s0, 10`
`top:  if (a>=100) goto A`	`top:  bge   $s0, 100, A`
`a = a + 10`	`      add   $s0, $s0, 10`
`goto top`	`      b     top`
`A:`	`A:`

Slide 7–13

While loops — Example ...

- If we care about speed we can reduce the number of instructions inside the loop:

```
      li    $s0, 10
      bge   $s0, 100, A
top:  add   $s0, $s0, 10
      blt   $s0, 100, top
A:
```

- Testing the conditional is moved to the end of the previous iteration, rather than the top of the current. An additional test is needed before the loop for the first iteration.

Slide 7–14

While loops — Example ...

- In conditional gotos this is:

```
a = 10
if (a >= 100) goto A
top:  a = a + 10
if (a < 100) goto top
A:
```

Slide 7–15

Do-while Loops

- HLL:

```
a = 10
do
   a = a + 10
while (a < 100)
```

Conditional gotos	MIPS
`      a = 10`	`      li    $s0, 10`
`top:  a = a + 10`	`top:  add   $s0, $s0, 10`
`      if (a<100) goto top`	`      blt   $s0, 100, top`

Slide 7–16

For-Loops

- A for loop header has three parts: a prelude, a condition, and an increment. The prelude is executed before the first iteration, the condition before each iteration, and the increment after each iteration.

HLL	Conditional gotos
	`P`
	`if (not C) goto Alab`
`for (P; C; I)`	`top:  B`
`   B;`	`I`
`A`	`if (C) goto top`
	`Alab:`

Slide 7–17

For-Loops Example

- HLL: `for (a = 10; a < 100; a = a + 10) c = c + a`

Conditional gotos	MIPS
`a = 10`	`      li   $s0, 10`
`if (a >= 100) goto Alab`	`      bge  $s0, 100, A`
`top:  c = c + a`	`top:  add  $s1, $s1, $s0`
`a = a + 10`	`      add  $s0, $s0, 10`
`if (a < 100) goto top`	`      blt  $s0, 100, top`
`Alab:`	`A:`

Slide 7–18

Loop-and-a-half

- Some things are easy to do in assembly (because there is no structure), and somewhat awkward in HLLs, such as the loop-and-a-half, in which you want to break out of a loop body halfway through it:

HLL	Conditional gotos
`while(C1)`	`if (not C1) goto Alab`
`   B1`	`top:  B1`
`   if (C2) then`	`if (C2) goto Alab`
`      break`	`B2`
`   B2`	`if (C1) goto top`
`A`	`Alab:  A`

Slide 7–19

Complex Conditions

- Conditions that include "and" or "or" are much more difficult to implement. One solution is positional evaluation – you never actually compute the value of the expression, instead you branch from simple condition to simple condition until the overall condition is determined to be true or false.
- I'll do all examples with if-then statements, but the principles work for all HLL control flow constructs.

Slide 7–20

Complex Conditions — And

HLL	Conditional gotos

`if ((a>0)&&(a<100)) then`
`   T`
`A`
	`if (a<=0) then goto Alab`
	`if (a>=100) then goto Alab`
	`T`
	`Alab:  A`

- Each condition is tested in turn, and the program branches out when one is found to be false. This is an example of short-circuited evaluation – once one simple condition is false, the entire condition must be false and the rest of the conditions are skipped. This is how C does it (probably because it's simple), but it depends on the programming language.

Slide 7–21

Complex Conditions — Or

HLL	Conditional gotos

`if ((a>0)||(a<100)) then`
`   T`
`A`
	`if (a > 0) then goto Tlab`
	`if (a >= 100) then goto Alab`
	`Tlab:  T`
	`Alab:  A`

- Basically, branch to the "true" code (T) if any of the first N-1 conditions are true, skip the "true" code if the last condition is false.

Slide 7–22

Readings and References

- Read Maccabe, Section 4.5