Loops

Lecture 9
Section 2.7

Robb T. Koether

Hampden-Sydney College

Fri, Sep 13, 2019

Outline

`while` Loops

`while` Loops

```
while (condition)
{
    action
}
```

- The **`while`** statement is the most basic iterative structure in C.
- It is identical to an **`if`** statement except that at the end of the *action*, execution returns to the condition.

do-while Loops

do-while Loops

```
do
{
    action
} while (condition)
```

- The **do-while** structure is identical to the **while** look except that the condition occurs at the end of the *action* rather than the beginning.
- This means that the *action* will be executed at least once, whereas the *action* in a **while** loop may not be executed at all.

`for` Loops

`for` Loop

```
for (i = 0; i < limit; i++)
{
    action
}
```

Equivalent `while` Loop

```
i = 0;
while (i < limit)
{
    action
    i++;
}
```

Outline

Outline

While Loops in MIPS

While Loops in MIPS

```
loop_beg:
        (branch instr)          # Quit if cond is true
        (action)                # Perform action
        j       loop_beg        # Return to condition
loop_end:
```

- The condition is the condition to *quit* the loop.
- Note that the form is the same as for a one-way decision structure except that a jump statement is added at the end of the `action`, which necessitates a label at the top of the loop.
- Note that the label `loop_end` comes *after* the jump.

While Loops in MIPS

Example (While Loop)

Write a MIPS program that will read a series of integers until 0 is read, and then print the total.

Outline

Better While Loops

- One drawback of the previous code is that the loop includes two branch statements (a conditional branch and an unconditional branch), each one executed on every iteration.
- This is unnecessarily inefficient.
- A better design is to place the conditional branch at the bottom of the loop.
- However, it needs to be tested before the first iteration of the loop!
- Precede the loop with a once-executed unconditional jump to the bottom of the loop.

Better While Loops

Better While Loops

```
        j       loop_end    # Jump to condition
loop_beg:
        (action)            # Perform action
loop_end:
        (br to loop_beg)    # Cont loop if cond is true
```

- Note that the condition in the branch is now the condition to stay in the loop.
- Also note that the label `loop_end` comes *before* the branch.

Better While Loops

Example (Better While Loop)

Rewrite the previous MIPS program with the conditional branch at the bottom of the loop.

Outline

Do-While Loops in MIPS

Do-While Loops in MIPS

```
loop_beg:
        (action)                    # Perform action
        (branch to loop_beg)        # Cont loop if true
```

- The condition is placed *after* the `action` has been executed.
- This is identical to the improved **`while`** loop, except without the initial jump to the bottom.
- There is only one label and only one branch. statement.

Do-While Loops in MIPS

Do-While Loops in MIPS

```
loop_beg:
        (action)                    # Perform action

        (branch to loop_beg)        # Cont loop if true
```

- Note that this could be converted to a standard **`while`** loop by adding an initial jump to the branch statement.

Do-While Loops in MIPS

Do-While Loops in MIPS

```
        j       loop_end
loop_beg:
        (action)                # Perform action
loop_end:
        (branch to loop_beg)    # Cont loop if true
```

- Note that this could be converted to a standard **`while`** loop by adding an initial jump to the branch statement.

Outline

For Loops in MIPS

Equivalent `while` Loop

```
i = 0;
while (i < limit)
{
     (action)
     i++;
}
```

- Recall the **`while`** loop equivalent of a **`for`** loop.

For Loops in MIPS

Equivalent Loop in Unstructured C

```
    i = 0;
loop_beg:
    if (i >= limit) goto loop_end;
    {
        (action)
        i++;
        goto loop_beg;
    }
loop_end:
```

- Recall the **`while`** loop equivalent of a **`for`** loop.

For Loops in MIPS

Equivalent Loop in Unstructured C

```
    i = 0;
    goto loop_end;
loop_beg:
    {
        (action)
        i++;
    }
loop_end:
    if (i < limit) goto loop_beg;
```

- The improved **`while`** loop.

For Loops in MIPS

For Loops in MIPS

```
        li      $t0, 0                  # i = 0
        li      $s0, 10                 # limit = 10
        j       loop_end                # Jump to test
loop_beg:
        (action)                        # Perform action
        addi    $t0, $t0, 1             # i = i + 1
loop_end:
        blt     $t0, $s0, loop_beg # Branch if i < 10
```

- Assume that the counter is in `$t0` and the limit is in `$s0`.

For Loops in MIPS

For Loops in MIPS (Count Down)

```
        li      $t0, 10          # i = 10
        j       loop_end         # Jump to test
loop_beg:
        (action)                 # Perform action
        addi    $t0, $t0, -1     # i = i - 1
loop_end:
        bgtz    $t0, loop_beg    # Continue if i > 0
```

- Sometimes it is more convenient to count down instead of up in a **`for`** loop.

For Loops in MIPS

- Examples
  - Run `count_to_10.asm` example.
  - Run `nap_time.asm` example.

Outline

Assignment

Assignment

- Read Section 2.7.