

Making decisions

Lecture B12



Lecture notes section B1 2

2002-01-29

CSE1 303 Part B lecture notes

1

Last time

- Memory diagrams
- System stack
 - pushing and popping
 - \$sp and \$fp
- Local variables
 - stored on stack
 - accessed with negative offset from \$fp
- Addressing modes
 - register + constant

2002-01-29

CSE1 303 Part B lecture notes

2

In this lecture

- C goto
- MIPS branch and jump instructions
- Selection
 - if-else
- Iteration (loops)
 - while
 - for
 - do-while

2002-01-29

CSE1 303 Part B lecture notes

3

C goto statement

/* Don't ever write code like this! */

```
int main()
{
    printf("one");
    goto apple;
orange:
    printf("three");
    goto pomegranate;
apple:
    printf("two");
    goto orange;
pomegranate:
    printf("four");
    exit(0);
}
```

goto is a dangerous keyword to use in C, as it promotes bad code design, as in this example ...

... but in MIPS, the equivalent (jump) instruction is all we've got

2002-01-29

CSE1 303 Part B lecture notes

4

MIPS jump instruction

- Jump causes program to immediately continue from specified label
 - copies address of label into PC (program counter) register
- Jumps are unconditional
 - always happen each time they are executed

2002-01-29

CSE1 303 Part B lecture notes

5

MIPS jump instruction

```
.data
str1: .asciiz "one"
str2: .asciiz "two"
str3: .asciiz "three"
str4: .asciiz "four"

.text
main: li $v0, 4 # print one
      la $a0, str1
      syscall
      j apple
apple: li $v0, 4 # print two
      la $a0, str2
      syscall
orange: li $v0, 4 # print three
      la $a0, str3
      syscall
      j pomegranate
pomegranate: li $v0, 4 # print four
             li $a0, str4
             syscall
             li $v0, 10
             syscall
# rest of program continued
# at right ...
```

2002-01-29

CSE1 303 Part B lecture notes

6

Selection

- Selection is how programs make choices
 - in C, with if, if-else, switch
- Achieved by selectively not executing some lines of code

2002-01-29

CSEI 303 Part B lecture notes

7

Selection: if

```

/* Sane people write C like this. */
int main()
{
    int i;
    scanf("%d", &i);
    if (i < 0)
    {
        printf("negative");
    }
    exit(0);
}

/* You could write it like this too (ugh). */
int main()
{
    int i;
    scanf("%d", &i);
    /* Skip if i not neg. */
    if (i >= 0) goto skip;
    printf("negative");
skip:
    exit(0);
}

```

inverting condition (here, < to >=) makes code neater

2002-01-29

CSEI 303 Part B lecture notes

8

MIPS branch instructions

- Branch instruction transfers control to specified label depending on a condition
 - less than (or equal)
 - greater than (or equal)
 - (not) equal
- If condition is true, go to label
- If condition is false, do nothing
 - and go to following instruction as normal
- Branch is conditional

2002-01-29

CSEI 303 Part B lecture notes

9

if in MIPS

```

.data
str1: .ascii "negative"

.text
main: # Make one local var.
      move $fp, $sp
      subu $sp, $sp, 4

      # Read i.
      li $v0, 5 # read int
      syscall
      sw $v0, -4($fp) # i

      # Comparison part:
      lw $t0, -4($fp) # i
      # if (i >= 0) goto skip
      bge $t0, 0, skip
      # ... else fall through
      # to here.

      # Print out string.
      li $v0, 4 # print str
      la $a0, str1
      syscall

      # Join up here.
      li $v0, 10 # Exit
      syscall

# ... Continued from left column.
skip: # Continued next column ...

```

2002-01-29

CSEI 303 Part B lecture notes

10

Selection: if-else

```

int main()
{
    int i;
    scanf("%d", &i);
    if (i < 0)
    {
        printf("negative");
    }
    else
    {
        printf("0/positive");
    }
    exit(0);
}

int main()
{
    int i;
    scanf("%d", &i);
    /* Branch based on cond. */
    if (i >= 0) goto false;
    /* original cond true. */
    printf("negative");
    /* Skip false case. */
    goto endif;
false:
    /* original cond false. */
    printf("0/positive");
endif:
    exit(0);
}

```

2002-01-29

CSEI 303 Part B lecture notes

11

if-else in MIPS

```

.data
str1: .ascii "negative"
str2: .ascii "0/positive"

.text
main: # Make one local var.
      move $fp, $sp
      subu $sp, $sp, 4

      # Read i.
      li $v0, 5 # read int
      syscall
      sw $v0, -4($fp) # i

      # Comparison part:
      lw $t0, -4($fp) # i
      # if (i >= 0) goto false
      bge $t0, 0, false
      # ... else fall through.

      # Print out -ve string.
      li $v0, 4 # print str
      la $a0, str1
      syscall

      # skip over false code.
      j endif

false: # Print out +ve string.
      li $v0, 4 # print str
      la $a0, str2
      syscall

      # Join up here.
      li $v0, 10 # Exit
      syscall

# ... Continued from left column.
endif: # Continued next column ...

```

2002-01-29

CSEI 303 Part B lecture notes

12

Iteration

- Iteration is the repetition of a section of code
 - in C, with `while`, `do-while`, `for`
 - `while` tests condition before loop entry
 - "pre-tested" loop
 - `do-while` tests condition after loop is run once
 - "post-tested" loop
 - `for` is shorthand for `while`
- Achieved by sending control from end of loop back to beginning
 - test some condition to prevent infinite loop

2002-01-29

CSEI 303 Part B lecture notes

13

Iteration: while

```

/* while loops written like this ... */
main()
{
    int n;
    int f = 1;
    scanf("%d", &n);
    /* compute factorial */
    while (n > 0)
    {
        f = f * n;
        n--;
    }
    printf("%d", f);
}

/* ... could also be written like this. */
main()
{
    int n; int f = 1;
    scanf("%d", &n);
    /* Exit if while condition stops being true. */
    loop:
    if (n <= 0) goto end;
    /* Body of loop. */
    f = f * n;
    n--;
    /* Repeat loop. */
    goto loop;
end:
    printf("%d", f);
}
    
```

note inversion of condition

2002-01-29

CSEI 303 Part B lecture notes

14

while in MIPS

```

-text
main: # Two local variables. # ... Continued from left
      move $fp, $sp
      subu $sp, $sp, 8
      # Initialization
      li $t0, 1
      sw $t0, -4($fp) # f
      # Read n.
      li $v0, 5 # read int
      syscall
      sw $v0, -8($fp) # n
      # Now comes the loop.
      # Exit loop if n <= 0.
      lw $t0, -8($fp)
      ble $t0, 0, end
      # ... else fall through.
      # Continued at right ...

      # Body of while loop.
      # f = f * n
      lw $t0, -4($fp) # f
      lw $t1, -8($fp) # n
      mul $t0, $t0, $t1
      sw $t0, -4($fp) # f
      # n-- (n = n - 1)
      lw $t0, -8($fp) # n
      sub $t0, $t0, 1
      sw $t0, -8($fp) # n
      # End of loop, go back
      j loop

end: # Print result
     li $v0, 1 # print int
     lw $a0, -4($fp) # f
     syscall
     li $v0, 10 # exit
     syscall
    
```

2002-01-29

CSEI 303 Part B lecture notes

15

Iteration: do-while

```

/* do-while loops written like this ... */
int v;
main()
{
    /* Keep prompting until user enters valid value. */
    do {
        printf("type num <= 31");
        scanf("%d", &v);
    } while (v > 31);
    /* output 2 to power v. */
    printf("%d", 1 << v);
}

/* ... could also be written like this. */
int v;
main()
{
    /* Keep prompting until user enters valid value. */
    loop:
    printf("type num <= 31");
    scanf("%d", &v);
    /* Repeat loop if condition is still satisfied. */
    if (v > 31) goto loop;
    /* End of loop. */
    /* Program continues. */
    printf("%d", 1 << v);
}
    
```

condition does not need inverting here

2002-01-29

CSEI 303 Part B lecture notes

16

do-while in MIPS

```

.data
v: .space 4
prom: .asciiz "type num <= 31"

-text
main: # Begin loop. # ... Continued from left
loop: # Print prompt
     li $v0, 4 # print str
     la $a0, str
     syscall
     # Read v.
     li $v0, 5 # read int
     syscall
     sw $v0, v
     # Continued at right ...

     # Repeat if v > 31
     lw $t0, v
     bgt $t0, 31, loop
     # Loop ends here.
     # Program continues.
     # Print 1 << v
     li $v0, 1 # print int
     lw $t0, 1
     li $t1, v
     sll $a0, $t0, $t1
     syscall
     li $v0, 10 # exit
     syscall
    
```

2002-01-29

CSEI 303 Part B lecture notes

17

Iteration: for

- `for` loops are essentially a neater way of writing `while` loops
 - initialization, condition and increment code all in one place
- To implement `for` loop, write as `while` loop

```

for (init; cond; inc)
{
    body;
}
    
```

→

```

init;
while (cond)
{
    body;
    inc;
}
    
```

2002-01-29

CSEI 303 Part B lecture notes

18

Iteration: for

```

/* turn this for loop... */          /* ... into this while loop. */
main()                               main()
{                                     {
  /* An old drinking                int bottles;
  song, abbreviated. */
  int bottles;
  /* Count down bottles. */         bottles = 10;
  for (bottles = 10;                while (bottles != 1)
      bottles = 1;
      bottles--;)
  {
    printf("%d green bottles\n",    printf(
      bottles);                    "%d green bottles\n",
    }                               bottles);
  /* Special case, no plural. */     /* Special case, no plural. */
  printf("1 green bottle\n");       printf("1 green bottle\n");
  /* End of song. */               /* End of song. */
  printf("no green bottles\n");     printf("no green bottles\n");
}                                     }

```

note re-ordering of parts

2002-01-29

CSEI 303 Part B lecture notes

19

for in MIPS

```

.data
botn: .asciiz "green bottles\n"      # ... continued from left
bot1: .asciiz "one green bottle\n"
bot0: .asciiz "no green bottles\n"

main:
# One local var.
move $fp, $sp
subu $sp, $sp, 4

# For loop: initialize.
li $t0, 10
sw $t0, -4($fp) # bottles

# For loop: condition.
lw $t0, -4($fp) # bottles
# End when bottles == 1.
bge $t0, 1, end

# For loop: body.
li $v0, 1 # print int
lw $a0, -4($fp) # bottles
syscall

# continued at right ...

# ... continued from left
li $v0, 4 # print str
la $a0, botn
syscall

# For loop: increment.
lw $t0, -4($fp) # bottles
sub $t0, $t0, 1
sw $t0, -4($fp) # bottles

# Return to loop start.
j loop

# Program continues
li $v0, 4 # print str
la $a0, bot1
syscall

li $v0, 4 # print str
la $a0, bot0
syscall

li $v0, 10 # exit
syscall

```

2002-01-29

CSEI 303 Part B lecture notes

20

Covered in this lecture

- C goto
- MIPS branch and jump instructions
- Selection
 - if-else
- Iteration (loops)
 - while
 - for
 - do-while

2002-01-29

CSEI 303 Part B lecture notes

21

Going further

- C switch statement
 - efficiently selecting one of many options
 - sometimes implemented with jump table (array of target addresses)
- MIPS jump/branch delay slots
 - related to the instruction pipeline
 - executing the instruction following a branch/jump because it has already started

2002-01-29

CSEI 303 Part B lecture notes

22

Next time

- Pointers
- Arrays



Reading:

Lecture notes section B1.3

2002-01-29

CSEI 303 Part B lecture notes

23

Copyright

Copyright © 2001 Deborah Pickett.
No part of this presentation may be duplicated without permission from the author.

2002-01-29

CSEI 303 Part B lecture notes

24