

MIPS programming

Lecture B10



Lecture notes section B10

2002-01-18

CSEI 303 Part B lecture notes

1

Last time

- MIPS R2000 instructions
 - instruction set
 - format
 - encoding
 - pseudoinstructions
- Sign
 - signed vs unsigned instructions
 - sign/zero extending

2002-01-18

CSEI 303 Part B lecture notes

2

In this lecture

- Simple MIPS programs
 - arithmetic
 - string handling
- Input/Output
 - using system calls
- Using data
 - data segment
- Assembler directives

2002-01-18

CSEI 303 Part B lecture notes

3

Arithmetic

- To perform arithmetic in MIPS, must break expression down into simple operations
 - add, subtract, multiply, divide, remainder, negate
 - these are the arithmetic operations supported by MIPS instruction set
 - perform operations in logical order
 - determined by precedence of operators
 - use temporary registers (\$t0 to \$t9) to store intermediate results
 - use load to fetch variable's value
 - use store to assign value to variable

2002-01-18

CSEI 303 Part B lecture notes

4

Arithmetic

$$C = 5 \times (F - 32) \div 9$$

```
li $t0, 5           # put 5 in $t0
lw $t1, F           # fetch F, put in $t1
li $t2, 32          # put 32 in $t2
sub $t1, $t1, $t2   # compute difference
mul $t0, $t0, $t1   # multiply 5 by result
li $t1, 9           # put 9 in $t1
div $t0, $t0, $t1   # divide result by 9
sw $t0, C           # store result in C
```

2002-01-18

CSEI 303 Part B lecture notes

5

Arithmetic

even easier with reverse Polish notation: just work left to right

5 get-F 32 - × 9 ÷ put-C

```
li $t0, 5           # put 5 in $t0
lw $t1, F           # fetch F, put in $t1
li $t2, 32          # put 32 in $t2
sub $t1, $t1, $t2   # compute difference
mul $t0, $t0, $t1   # multiply 5 by result
li $t1, 9           # put 9 in $t1
div $t0, $t0, $t1   # divide result by 9
sw $t0, C           # store result in C
```

2002-01-18

CSEI 303 Part B lecture notes

6

Arithmetic

$$C = 5 \times (F - 32) \div 9$$

```
li $t0, 5
lw $t1, F
li $t2, 32
sub $t1, $t1, $t2
mul $t0, $t0, $t1
li $t1, 9
div $t0, $t0, $t1
sw $t0, C
```



```
li $t0, 5
lw $t1, F
sub $t1, $t1, 32
mul $t0, $t0, $t1
div $t0, $t0, 9
sw $t0, C
```

can sometimes abbreviate steps using immediate instructions

2002-01-18

CSE1 303 Part B lecture notes

7

Global variables

- Exist for duration of program
- Accessible from any part of program
 - registers too volatile
- Use memory
 - data segment

```
# ASSEMBLY PROGRAM
# Global variables
# follow.
.data
```

2002-01-18

CSE1 303 Part B lecture notes

8

Global variables

- Initialized global variables are declared with initial value
 - int (32 bits) with `.word` and value
 - other types according to size

```
/* C PROGRAM */
/* declare var = -42 */
int var = -42;
```

```
# ASSEMBLY PROGRAM
# Global variables
# follow.
.data
var: .word -42
```

`.word` allocates 4-byte value

value to store

2002-01-18

CSE1 303 Part B lecture notes

10

Global variables

- Uninitialized global variables are allocated by specifying size only
 - use `.space` directive with bytes

```
/* C PROGRAM */
/* declare var = -42 */
int var = -42;
/* empty has no
initial value. */
int empty;
```

```
# ASSEMBLY PROGRAM
# Global variables
# follow.
.data
var: .word -42
empty: .space 4
```

`.space` allocates empty space

number of bytes

2002-01-18

CSE1 303 Part B lecture notes

11

Assembler directives

- Instruct assembler to allocate space/data or switch modes
- Do not assemble to machine language instructions
- Always start with `.` (dot)

2002-01-18

CSE1 303 Part B lecture notes

11

Assembler directives

- Change mode
 - `.data`
 - assemble into data segment
 - `.text`
 - assemble into text segment (code)
- Allocate space
 - `.space n`
 - allocate *n* bytes, store nothing

2002-01-18

CSE1 303 Part B lecture notes

12

Assembler directives

- Allocate data
 - `.word w1[, w2, w3, ...]`
 - allocate 4-byte word(s) and store value(s)
 - `.half h1[, h2, h3, ...]`
 - allocate 2-byte halfword(s) and store value(s)
 - `.byte b1[, b2, b3, ...]`
 - allocate single byte(s) and store value(s)
 - `.ascii "string"`
 - allocate sequence of bytes, store ASCII values
 - `.asciiz "string"`
 - allocate sequence of bytes, store ASCII values, terminate string with 0 byte (end of string)
- Other types: `.float`, `.double`

2002-01-18

CSE1 303 Part B lecture notes

13

Input/Output

- On real machines (including real MIPS computers), I/O is very complicated
 - usually handled by operating system
- In SPIM, I/O is managed by system calls
 - `syscall` instruction
 - SPIM suspends simulation to perform I/O, then resumes simulation

2002-01-18

CSE1 303 Part B lecture notes

14

System calls

- To make a system call
 - determine which service you want
 - put service's call code in register `$v0`
 - put arguments (if any) in registers `$a0`, `$a1`, ...
 - perform the `syscall` instruction
 - SPIM performs service
 - results (if any) in registers `$v0`, `$v1`

2002-01-18

CSE1 303 Part B lecture notes

15

System calls

Service	Call code (\$v0)	Arguments	On return	Notes
print integer	1	\$a0 = value to print		
print string	4	\$a0 = address of start of string		string terminated with '\0'
read integer	5		\$v0 = entered integer	
read string	8	\$a0 = address to write string, \$a1 = maximum length	entered string is located in memory	ends at maximum length or '\n', string terminated with '\0'
exit	10			stop simulator

2002-01-18

CSE1 303 Part B lecture notes

16

System calls

```

/* syscall acts like this almost-C function. */
void syscall(void)
{
    switch ($v0) {
        case 1: /* print integer in $a0 */
            printf("%d", $a0); break;
        case 4: /* print string pointed to by $a0 */
            puts($a0); break;
        case 5: /* read integer, return in $v0 */
            scanf("%d", &$v0); break;
        case 8: /* read string into $a0[], len $a1 */
            fgets($a0, $a1, stdin); return;
        case 10: /* exit */
            exit(0);
        /* other syscalls exist ... */
    }
}

```

2002-01-18

CSE1 303 Part B lecture notes

17

Input/Output: numbers

```

#include <stdio.h>
#include <stdlib.h>

/* For storing user input. */
int val;

int main()
{
    /* Read an int. */
    scanf("%d", &val);

    /* Print it out squared. */
    printf("%d", val * val);

    exit(0);
}

```

```

.data
# User input.
val: .space 4

.text
main:
# Syscall 5: read int
li $v0, 5
syscall
# Number is in $v0
sw $v0, val

# Syscall 1: print int
li $v0, 1
li $t0, val
# $a0 is what to print
mul $a0, $t0, $t0
syscall

# Syscall 10: exit
program
li $v0, 10
syscall

```

2002-01-18

CSE1 303 Part B lecture notes

18

Input/Output: numbers

```
#include <stdio.h>
#include <stdlib.h>

/* For storing user input. */
int val;

int main()
{
    /* Read an int. */
    scanf("%d", &val);

    /* Print it out squared. */
    printf("%d", val * val);

    exit(0);
}
```

```
.data
# User input.
val: .space 4

.text
# Syscall 5: read int
li $v0, 5
syscall
# Number is in $v0
sw $v0, val

# Syscall 1: print int
li $v0, 1
li $t0, val
# $a0 is what to print
mul $a0, $t0, $t0
syscall

# Syscall 10: exit
program
li $v0, 10
syscall
```

2002-01-18

CSEI 303 Part B lecture notes

19

Input/Output: strings

```
#include <stdio.h>
#include <stdlib.h>

/* Prompt and buffer. */
char prom[] = "Type a string: ";
char buf[20];

int main()
{
    /* Print prompt */
    puts(prom);

    /* Read a string. */
    fgets(buf, 20, stdin);

    /* Print it back out. */
    puts(buf);

    exit(0);
}
```

```
.data
prom: .asciz "Type a string: "
buf: .space 20
.text
main:
# Syscall 4: print str
li $v0, 4
# $a0: address of string
la $a0, prom
syscall

# syscall 8: read string
li $v0, 8
la $a0, buf
# $a1: max length of str
li $a1, 20
syscall

li $v0, 4
la $a0, buf
syscall

li $v0, 10 # syscall 10
syscall
```

2002-01-18

CSEI 303 Part B lecture notes

20

Input/Output: strings

```
#include <stdio.h>
#include <stdlib.h>

/* Prompt and buffer. */
char prom[] = "Type a string: ";
char buf[20];

int main()
{
    /* Print prompt */
    puts(prom);

    /* Read a string. */
    fgets(buf, 20, stdin);

    /* Print it back out. */
    puts(buf);

    exit(0);
}
```

```
.data
prom: .asciz "Type a string: "
buf: .space 20
.text
main:
# Syscall 4: print str
li $v0, 4
# $a0: address of string
la $a0, prom
syscall

# syscall 8: read string
li $v0, 8
la $a0, buf
# $a1: max length of str
li $a1, 20
syscall

li $v0, 4
la $a0, buf
syscall

li $v0, 10 # syscall 10
syscall
```

2002-01-18

CSEI 303 Part B lecture notes

21

Translation

- Compilers translate C into assembly language
 - by generating assembly language code that is functionally equivalent to the C
- Compilers do not "understand" what a program does
 - translation is done blindly according to a set of rules

2002-01-18

CSEI 303 Part B lecture notes

22

Translation

- Compilers can optimize code
 - by changing code to perform simpler but equivalent instructions
 - to make code run faster
 - to make code use less memory
- Optimization is difficult
 - need to prove that original version is equivalent to optimized one in all cases

2002-01-18

CSEI 303 Part B lecture notes

23

Translation

- Simple optimizations
 - replace multiplication/division by power of two with shift
 - $x = y * 8 \Leftrightarrow x = y \ll 3$
 - re-ordering expressions to use fewer registers
 - $x = 5 + 6 * y \Leftrightarrow x = 6 * y + 5$
 - constant folding
 - $s = 60 * 60 * 24 \Leftrightarrow s = 86400$

2002-01-18

CSEI 303 Part B lecture notes

24

Translation

- Complex optimization
 - keeping and re-using values in registers
 - extracting invariant expressions from loop and evaluating once before loop entry
 - while (i < strlen(s)) { } ⇔
 - n = strlen(s); while (i < n) { }
 - removing redundant variables
 - b = c; a = b + 3 ⇔ a = c + 3
 - introducing other variables
 - x = hard() + 5; y = hard() - 5 ⇔
 - temp = hard(); x = temp + 5; y = temp - 5;
 - changing loop exit conditions
 - while (i < 10) { } ⇔ while (i != 10) { }

2002-01-18

CSE1 303 Part B lecture notes

25

Translation

- In CSE1 303, we will permit simple optimizations and avoid complex ones
 - called “faithful translation”
 - makes task of translating easier since no “action at a distance” effects
 - each line of C largely translated independently of others
 - at slight space/speed cost

2002-01-18

CSE1 303 Part B lecture notes

26

```
#include <stdio.h>
#include <stdlib.h>

/* Read a number n, then print
(n * (n-1))/2 (a combinatorial
function). */

int n;
char prom[] = "Type a number: ";
char nl[] = "\n";

int main()
{
    /* Print prompt. */
    puts(prom);

    /* Read the number. */
    scanf("%d", &n);

    /* Print result then newline.
    */
    printf("%d", (n * (n-1)) / 2);
    puts(nl);

    /* Exit. */
    exit(0);
}
```

```
.data
n:      .space 4
prom:   .asciiz "Type a number: "
nl:     .asciiz "\n"

.text
main:   li $v0, 4      # print str
        la $a0, prom  # at prom
        syscall

        li $v0, 5      # read int
        syscall
        sw $v0, n      # store in
                    n

        li $v0, 1      # print int
        lw $t0, n      # n
        sub $t1, $t0, 1 # n-1
        mul $t0, $t0, $t1 # *
        sra $a0, $t0, 1 # /2
        syscall

        li $v0, 4      # print str
        la $a0, nl     # at nl
        syscall

        li $v0, 10     # exit
        syscall
```

2002-01-18

CSE1 303 Part B lecture notes

27

```
#include <stdio.h>
#include <stdlib.h>

/* Read a number n, then print
(n * (n-1))/2 (a combinatorial
function). */

int n;
char prom[] = "Type a number: ";
char nl[] = "\n";

int main()
{
    /* Print prompt. */
    puts(prom);

    /* Read the number. */
    scanf("%d", &n);

    /* Print result then newline.
    */
    printf("%d", (n * (n-1)) / 2);
    puts(nl);

    /* Exit. */
    exit(0);
}
```

```
.data
n:      .space 4
prom:   .asciiz "Type a number: "
nl:     .asciiz "\n"

.text
main:   li $v0, 4      # print str
        la $a0, prom  # at prom
        syscall

        li $v0, 5      # read int
        syscall
        sw $v0, n      # store in
                    n

        li $v0, 1      # print int
        lw $t0, n      # n
        sub $t1, $t0, 1 # n-1
        mul $t0, $t0, $t1 # *
        sra $a0, $t0, 1 # /2
        syscall

        li $v0, 4      # print str
        la $a0, nl     # at nl
        syscall

        li $v0, 10     # exit
        syscall
```

2002-01-18

CSE1 303 Part B lecture notes

28

Covered in this lecture

- Simple MIPS programs
 - arithmetic
 - string handling
- Input/Output
 - using system calls
- Using data
 - data segment
- Assembler directives

2002-01-18

CSE1 303 Part B lecture notes

29

Going further

- Compiler optimization
- Other SPIM system calls
 - read/write character
 - read/write float/double
 - file open/close and read/write
 - allocate memory from heap

2002-01-18

CSE1 303 Part B lecture notes

30

Next time

- Working with memory
- Local variables
- System stack
 - stack frames
 - using \$sp and \$fp
- Accessing memory in MIPS
 - addressing modes



Reading:
Lecture notes section B1.1

2002-01-18

CSE1303 Part B lecture notes

31

Copyright

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

2002-01-18

CSE1303 Part B lecture notes

32