More on MIPS Assembly and Memory

CS 64: Computer Organization and Design Logic
Lecture #9

Ziad Matni
Dept. of Computer Science, UCSB


AN ENGINEER HELPING A DESIGNER
NO PROBLEM, WE CAN FIX THIS ON THE TERMINAL
CLACK
CLACK
WHOA, YOU'RE A *HACKER!*
NO, IT'S JUST THE *TERMINAL.*
WHERE ARE ALL THE BUTTONS, ICONS, AND DROPDOWN MENUS?
IS THIS...
THE *MATRIX?*
YES.
PABLO STANLEY

Lecture Outline

- Midterm Exam: Sample Qs
- Shifting bits
- Stdin process
- How to implement arrays
- Intro to MIPS Calling Convention
  - Functions

MIDTERM IS COMING!

- **Thursday, 2/15** in this classroom
- **Starts at 3:30pm **SHARP****
  - Please start arriving 5-10 minutes before class
- **I may ask you to change seats**
- Please bring your UCSB IDs with you

- **Closed book**: no calculators, no phones, no computers
- ***Only*** the MIPS Reference Card is allowed
- **You will write your answers on the exam sheet itself.**

What's on the Midterm?? 1/2

- Data Representation
  - Convert bin ←→ hex ←→ decimal ←→ bin
  - Signed and unsigned binaries
- Logic and Arithmetic
  - Binary addition, subtraction
    - Carry and Overflow
  - Bitwise AND, OR, NOT, XOR
  - General rules of AND, OR, XOR, using NOR as NOT
- All demos done in class
- Lab assignments 1, 2 , 3 and 4

What's on the Midterm?? 2/2

Assembly

- Core components of a CPU
  - How instructions work
- Registers ($t, $s, $a, $v)
- Arithmetic in assembly (add, subtract, multiply, divide)
  - What's the difference between add, addi, addu, addui, etc…
- Conditionals and loops in assembly
- Conversion to and from Assembly and C/C++
- syscall and its various uses (printing output, taking input, ending program)
- **.data** and **.text** declarations
- Memory in MIPS
- Big Endian vs Little Endian
- R-type and I-type instructions
- Pseudo instructions

About the Midterm Exam

- Made up of Multiple Choice & Short Answers/Coding.

EXAMPLES:

Complete the following MIPS assembly code that is supposed to add the number in register $t0 to 15:

```
li $t0, 12
______________
```

```
A. add $t2, $t1, $t2
B. addu $t2, $t1, $t2
C. addi $t2, $t0, F
D. addi $t2, $t0, 0x15
E. addui $t2, $t0, 0xF
```

What is the 2's complement of 0x5EC?

```
A. 1x5EC
B. 0x5EC
C. 0xA13
D. 0xA14
E. 0xA15
```

Sample Questions

Translate this C-style code into 4 lines of MIPS assembly code:

```
int t1 = 10, t2= 3;
int t3= t1 + 2 * t2
```

```
li $t1, 10
li $t2, 3
sll $t2, $t2, 1
add $t3, $t2, $t1
```

What is the result of these operations?

`0xF2 & ~(0x55)` **0xA2**

`0x6701 | 0x1076` **0x7777**

`0x102A99D8 ^ 0xABA11CAB`

```
  0001 0000 0010 1010 1001 1001 1101 1000
^ 1010 1011 1010 0001 0001 1100 1010 1011
= 1011 1011 1000 1011 1000 0101 0111 0011
= 0xBB8B8573
```

Sample Questions

Translate this MIPS assembly code into pseudo-code (C or C++ accepted):

```
li $s0, 2
li $s1, 6
li $t0, 2
add $s2, $s1, $s0
sll $s2, 3
mult $s2, $t0
mflo $s3
```

Translate this C code into MIPS:

```
int t0 = 3;
if (t0 < 7)
{
   t1 = 1;
   printf (t1);
}
else
  t1 = t0 + t0;
```

`srl` vs `sra`

- `srl` replaces the "lost" MSBs with 0s
- `sra` replaces the "lost" MSBs with
  either 0s (if number is +ve) *or* 1s (if number is –ve)

IMPLICATIONS:

shiftDemo.asm

- `srl` should NOT be used for negative numbers
- `sra` use for positive numbers is redundant
- When using negative numbers, use `sra`

More on I/O Instructions

- Recall: to call an output, we use `syscall` which looks at what's in `$v0`
  - If `$v0` = 1, `syscall` will print (output) an integer
  - If `$v0` = 4, `syscall` will print (output) a string

- Recall: we can also execute inputs with the same method
  - If `$v0` = 5, `syscall` will get (input) an integer
  - The integer is returned in `$v0`

StdIn: Using the **$v0 = 5** syscall

- If `$v0` = 5, `syscall` will get (input) an integer
- The integer is returned in `$v0`

EXAMPLE:

```
li $v0, 5
syscall
move $t0, $v0
…
li $v0, 1
move $a0, $t0
syscall
```

inputDemo.asm

Arrays

- Question:

As far as memory is concerned, what is the *major difference* between an **array** and a **global variable**?

  - Arrays contain multiple elements

- Let's take a look at:
  - print_array1.asm
  - print_array2.asm
  - print_array3.asm

print_array1.asm

```
int myArray[]
   = {5, 32, 87, 95, 286, 386};
int myArrayLength = 6;
int x;

for (x = 0; x < myArrayLength; x++)
{
   print(myArray[x]);
   print("\n");
}
```

```
# C code:
# int myArray[] =
#     {5, 32, 87, 95, 286, 386}
# int myArrayLength = 6
# for (x = 0; x < myArrayLength; x++) {
#   print(myArray[x])
#   print("\n") }
.data
newline: .asciiz "\n"
myArray: .word 5 32 87 95 286 386
myArrayLength: .word 6

.text
main:
     # t0: x
     # initialize x
     li $t0, 0
loop:
     # get myArrayLength, put result in $t2
     # $t1 = &myArrayLength
     la $t1, myArrayLength
     lw $t2, 0($t1)

     # see if x < myArrayLength
     # put result in $t3
     slt $t3, $t0, $t2
     # jump out if not true
     beq $t3, $zero, end_main

     # get the base of myArray
     la $t4, myArray

     # figure out where in the array we need
     # to read from. This is going to be the array
     # address + (index << 2). The shift is a
     # multiplication by four to index bytes
     # as opposed to words.
     # Ultimately, the result is put in $t7
     sll $t5, $t0, 2
     add $t6, $t5, $t4
     lw $t7, 0($t6)

     # print it out, with a newline
     li $v0, 1
     move $a0, $t7
     syscall
     li $v0, 4
     la $a0, newline
     syscall

     # increment index
     addi $t0, $t0, 1

     # restart loop
     j loop

end_main:
     # exit the program
     li $v0, 10
     syscall
```

print_array3.asm

```
int myArray[]
   = {5, 32, 87, 95, 286, 386};
int myArrayLength = 6;
int* p;

for ( p = myArray; p < myArray + myArrayLength; p++)
{
   print(*p);
   print("\n");
}
```

This is the end of the material of what is on your midterm exam

MIPS Function Calling Convention

(not on the midterm exam)

Functions

- Up until this point, we have not discussed **functions**
- Why not?
  - Memory management is a <u>must</u> for the call stack ...though we can make some progress without it
- Think of recursion…
  - How many variables are we going to need ahead of time?
  - What memory do we end up using in recursive functions?

Implementing Functions

What capabilities do we need for functions?

1. Ability to execute code elsewhere
   – Branches and jumps

2. Way to pass arguments
   – There a way (convention) to do that…

3. Way to return values
   – Registers

Jumping to Code

```
void foo() {
  bar();
  baz();
}
```

```
void bar() {
  ...
}
```

```
void baz() {
  ...
}
```

- We have ways to jump to code (`j` instruction)
- But what about jumping back?
  - We'll need a way to *save* where we were (so we can "jump" back)
- **Q**: What do need so that we can do this on MIPS?
  - **A**: A way to store the program counter ($PC) (to tell us where the *next* instruction is so that we know *where* to return!)

Calling Functions on MIPS

- Two crucial instructions: `jal` and `jr`
- One specialized register: **`$ra`**

- `jal` (**jump-and-link**)
  - Simultaneously **jump to an address**, and **store the location of the next instruction** in register **`$ra`**

- `jr` (**jump-register**)
  - **Jump to the address stored in a register**, often **`$ra`**

Simple Call Example

- See program: **simple_call.asm**

```
# Calls a function (test) which immediately returns
.text
test:
        # return to whoever made the call
        jr $ra

main:
        # call test
        jal test

        # exit
        li $v0, 10
        syscall
```

Passing and Returning Values

- We want to be able to call arbitrary functions without knowing the implementation details

- How might we achieve this?
  - Designate specific registers for **arguments** and **return values**

Passing and Returning Values in MIPS

- Registers **`$a0`** thru **`$a3`**
  - **Argument registers**, for passing function arguments
- Registers **`$v0`** and **`$v1`**
  - **Return registers**, for passing return values

Passing and Returning Values in MIPS

Demos

- ***print_int.asm***
  - Illustrates the use of a printing sub-routine (i.e. like a simple function)
- ***add_ints.asm***
  - Illustrates the use of an adding sub-routine (i.e. like a simple function that returns a value)

YOUR TO-DOs

- Assignment/Lab #5 is <u>next</u> week
- **No lab this week**

- **Study for your midterm exam!! ☺**

- **<u>IMPORTANT</u>**:
  Read the ***MIPS Calling Convention*** handout for next week (after the exam)
  - Handout is on the class website

</LECTURE>