EE 108B Review Session #1

Michelle Hewlett

Friday October 7, 2005

5:15 PM – 6:05 PM

Room Skilling 193

EE 108B Review Session Agenda

- Announcements
- MIPS Instruction Set
- Programming Assignment 1
- HW 1 Hints
- Sample Problem

Announcements

- TA Office Hours posted on class website
  - In my OH, I will only answer HW and PA questions. SCPD students can phone in during OH.
  - Drew and Nju are the lab TA's. Ask them lab related questions, but you can also ask them HW questions.
  - Our email is:
    ee108b-fall0506-tas@lists.stanford.edu

Announcements

- Lab Students: Form groups of 2 or work alone
- Homeworks
  - Can work in groups of 2
  - Turn in one HW per group
  - First HW is due Thursday, October 13
- PAs: MUST WORK ALONE!

MIPS Instruction Set

- R, I, and J-Type instructions:
  - R (register) Arithmetic Instruction Format
  - I (immediate) Data Transfer Format
  - J (jump) Jump Format
- Look at the green insert or on pages 77-78 of the book for a comprehensive listing of MIPS instructions.

MIPS Procedure Call Convention

- The MIPS procedural call convention with regards to registers can be summarized as follows: (please follow this guideline when you do MIPS assembly coding for HWs, PAs, and in exams)
- Preserved:
  - Saved Registers ($s0-$s7)
  - Stack Pointer Register ($sp)
  - Return Address Register ($ra)
  - Stack above the stack pointer (make sure the callee doesn't write above the $sp)
- Not Preserved:
  - Temporary Registers ($t0-$t9)
  - Argument Registers ($a0-$a3)
  - Return-Value Registers ($v0, $v1)
  - Stack below the stack pointer

PA1 Logistics

- Due 10/20/05 11:59 PM – one late day can be used per quarter for PAs – write “Use Late Day”
- Learn how to use submit script in advance.
- Notes on grading:
  - Functionality and proper formatting (refer to sample output file)
  - Implement mergesort.c as is (no change of algorithm)
- Tools --use spim and xspim (no support for PC spim).
  - Spim—text based simulator. Quick, but hard to debug.
  - xspim –GUI, can see register values, memory and place breakpoints in code. But, cannot provide request user-input data with xspim, so must hard code data into.
- Finally, check the PA1’s “Answers to common questions” before sending us mail.

PA1

- Use given quicksort.c to determine what the MIPS program must do.
- Also given quicksort.s which is a skeleton of the required assembly program.
- Comments within quicksort.s clearly specify the functionality that needs to be added.

Caller/Callee MIPS Conventions

- Caller
  - Save caller-saved registers $a0-$a3, $t0-$t9
  - Load arguments in $a0-$a3, rest passed on stack
  - Execute jal instruction
- Callee Setup
  - Allocate memory for new frame ($sp = $sp - frame)
  - Save callee-saved registers $s0-$s7, $fp, $ra
  - Set frame pointer ($fp = $sp + frame size - 4)
- Callee Return
  - Place return value in $v0 and $v1
  - Restore any callee-saved registers
  - Pop stack ($sp = $sp + frame size)
  - Return by jr $ra

PA1 Example

```
//power.c
int power(int base, int pow) {
//base case is pow  == 0
if(pow == 0) {
        return 1;
}
return (base * power(base, pow -1));
}
void main(void){
        int base_int;
        int power_int;
        int result_int;
        printf("What is the base integer?\n");
 scanf("%d", &base_int);
 printf("What is the power?\n");
 scanf("%d", &power_int);
 result_int = power(base_int, power_int);
 printf("The answer is: %d\n", result_int);
}
```

PA1 Assembly Example

```
 # FILE:        power.s
.data
BASE_NUM:     .asciiz "What is the base integer?"
POWER_NUM:    .asciiz "What is the power?"
ANS:          .asciiz "The answer is: "
SPACE:        .asciiz " “              # Space
EOL:          .asciiz "\n"            # Return character
.text
.globl main
main:
              # Register Definitions
              # $s0 - base integer
              # $s1 - power to raise the base to
              # $s2 - final result
```

```
#---- Store the old values into stack
                addiu   $sp, $sp, -32     # Stack frame 32 bytes long
                sw      $ra, 28($sp)      # Store return address
                sw      $fp, 24($sp)      # Store frame pointer
                sw      $s0, 20($sp)
                sw      $s1, 16($sp)
                sw      $s2, 12($sp)
                addiu   $fp, $sp, 28      # Set up new frame pointer
#---- Prompt user for base integer
                li      $v0, 4            # print_string
                la      $a0, BASE_NUM     # text to be displayed
                syscall
                li      $v0, 4
                la      $a0, EOL
                syscall                   # print "\n"
                li      $v0, 5            # read_int
                syscall
                move    $s0, $v0          # save base integer
```

```
#---- Prompt user for power integer
                li      $v0, 4                  # print_string
                la      $a0, POWER_NUM          # text to be displayed
                syscall
                li      $v0, 4
                la      $a0, EOL
                syscall                         # print "\n"
                li      $v0, 5                  # read_int
                syscall
                move    $s1, $v0                # save power integer
#---- Call POWER
#
#  Make sure that you pass parameters according to the
#  established convention !!!!
#
                move    $a0, $s0                # $a0 <= the base integer
```

```
                move    $a1, $s1          # $a1 <= the power integer
                jal     POWER
                move    $s2, $v0          # $s2 <= the result
#---- Print result
                li      $v0, 4            # print_string
                la      $a0, ANS          # text: "The answer is: "
                syscall
                li      $v0, 1            # print_int
                move    $a0, $s2          # the final result from the function
                syscall
                li      $v0, 4            # print_string
                la      $a0, EOL          # text
                syscall

#---- Restore the old values from the stack
                lw      $s2, 12($sp)
                lw      $s1, 16($sp)
                lw      $s0, 20($sp)
```

```
                lw      $fp, 24($sp)
                lw      $ra, 28($sp)
                addiu   $sp, $sp, 32      # stack frame 32 bytes long
#---- Exit
                jr      $ra
POWER:

# Remember to use the correct calling conventions!
# $a0 is base in the C code
# $a1 is pow in the C code

#---- Store the old values into stack
                addiu   $sp, $sp, -32     # Stack frame 32 bytes long
                sw      $ra, 28($sp)      # Store return address
                sw      $fp, 24($sp)      # Store frame pointer
                sw      $s0, 20($sp)      # Store $s0
                addiu   $fp, $sp, 28      # Set up new frame pointer
#---- Check for base case
                bne     $a1, $zero, skip    # if base!=0, skip base case
```

```
                li      $v0, 1          # base==0, so return 1
                j       EXIT
skip:           move    $s0, $a0        # store the value of base – why??
                addi    $a1, $a1, -1    #subtract 1 from num
                jal     POWER

                mul     $v0, $v0, $s0   # multiply num the result of
                                        # the recursive call to set
                                        # up our return register
EXIT:           #---- Exit
                #---- Restore the old values from the stack
                lw      $s0, 20($sp)
                lw      $fp, 24($sp)
                lw      $ra, 28($sp)
                addiu   $sp, $sp, 32    # stack frame 32 bytes long

                jr      $ra
```

PA1 Debugging Hints

- Comment your code!
  - Assembly code is hard to read, so comment thoroughly (refer to power.s)
- With recursive procedures, make sure base case works (ex. 1-element array)
- After testing base case, move to 2 elements, 3, etc.
- Take advantage of xspim GUI – can see state of register and even memory values. Insert break points and step through your code, while checking the state (the registers)
- Read FAQ!
- Start Early!!

HW 1 Hints

- Problem 1:
  - Part a: Try to interpret the assembly code into a C program
  - Part c & d: Total Execution Time = (total clock cycles)/(clock cycle rate)
  - Part d: How does new instruction change the inner loop? Reduces instruction count, but cycle time has increased,…

HW 1 Hints

- Problem 2
  - Tail recursion is a special form of recursion where the last operation of a function is a recursive call (nothing has to be done after the call returns)
- Problem 3
  - Small has 16 bits and Big has 32 bits
  - Break big into upper and lower half (upper(big) and lower(big))

Sample Problem

As the new director of technology at XYZ Corporation, you have been asked to evaluate two different systems.

Instruction Type	CPI	
	M1	M2
A	2	6
B	3	1
C	3	1
D	2	6

The first machine operates at 500 MHz and the second machine operates at 600 MHz.

1. Assuming an even workload, what is the CPI of these 2 machines?

2. Consider the case where machine M2 is running 800,000 instructions, evenly split across the four instruction types. Your group wants to tell marketing that you have achieved a speedup of M2 by 1.17 of what it was previously. If you can only improve performance of the A instructions, what must M2's new CPI be for Instruction Type A to achieve those results?

3. The manufacturer for M2 considers the instructions B and C to be in the same category and A and D to be in the same category. They report that their machine is 1.5 times faster than machine M1. The same instruction mix was used for both machines in arriving at that number. What instruction mix (for AD and BC) must they have been using to achieve that result?