CMSC 411 : Assignment #1

Due on Tuesday, February 17, 2015

3:30pm

Anwar Mamat

Problem 1: Big vs Little Endian Addressing (10 points)

Consider the 32-bit hexadecimal number 0xab8fd6a1.

- 1. What is the binary representation of this number in big endian format? Please clearly mark the bytes and number them from low (0) to high (3).
- 2. What is the binary representation of this number in little endian format? Please clearly mark the bytes and number them from low (0) to high (3).
- Hint: You may find the discussion on page 24 of the MIPS R4000 User's Manual at http://cs.umd.edu/class/spring2015/cmsc411-0201/resources/mips_r4000_users_manual.pdf helpful.

Problem 2

MIPS code optimization (70 points)
Consider the following fragment of C code:

Listing 1: C code fragment

```
     printf("Old Array\n");
     for(int i=0; i< 10; i++) printf("%d,",a[i])
     for(int i=0; i< 10; i++) b[i]=a[i]+1
     printf("Old Array\n");
     for(int i=0; i< 10; i++) printf("%d,",b[i])
```

Assume that a and b are arrays of 32-bit integers, here is the MIPS implementation.

Listing 2: MIPS code

```
.data
     a1: .word 65:10
     b1: .word 0:10
     length: .word 10
     index: .word 0
     msg1: .asciiz "Old Array:\n"
     msg2: .asciiz "New Array:\n"
     cr: .asciiz "\n"
     sp: .asciiz ","
.text
main:
     la $a0, msg1
     li $v0, 4
     syscall
     la $t6, a1
     li $t0,10

print0:
     lw $a0, ($t6)
     li $v0, 1
     syscall
     la $a0, sp
     li $v0, 4
     syscall
     add $t6, $t6, 4
```

Problem 2 continued on next page...

```
     sub $t0,$t0,1
     bgt $t0,$zero, print0

update:
     la $t1, index  #load index
     lw $t3, ($t1)
     la $t5, a1          #load a1
     add $t5,$t5,$t3
     lw $t3, ($t5)  #load a1[0]
     add $t7,$t3, 1

     la $t1, index  #load index
     lw $t3, ($t1)
     la $t6, b1          #load b1
     add $t6,$t6,$t3
     sw $t7,($t6)   #store to b[0]

     la $t1, index  #load index
     lw $t3, ($t1)
     add $t3, $t3,4
     sw $t3, ($t1)

     la $t0,length  #load length
     lw $t3, ($t0)
     sub $t3, $t3,1
     sw $t3, ($t0)
     lw $t3, ($t0)
     bgt $t3,$zero, update

     la $a0, cr
     li $v0, 4
     syscall
     la $a0, msg2
     li $v0, 4
     syscall

     la $t6, b1
     li $t0,10
print2:
     lw $a0, ($t6)
     li $v0, 1
     syscall
     la $a0, sp
     li $v0, 4
     syscall
     add $t6, $t6, 4
     sub $t0,$t0,1
     bgt $t0,$zero, print2

     li $v0, 10
     syscall
```

- A) The MIPS code fragment from line 29 to line 54 reads the array a, adds 1, and stores the updated

data to array b. Answer following question only using the code fragment in lines 29-54:

1. How many instructions are required dynamically? [10 points]
2. How many memory-data references will be executed? [10 points]
3. what is the code size? [10 points]

- B): The code fragment in lines 29-54 does not keep memory data in the register for reuse. It loads a data from memory, and loads again from memory when it needs the same data later . For example: "index" is loaded multiple times.

1. Optimize the code fragment in lines 29-54, and minimize the number of memory access [25 points]
2. How many instructions are required dynamically in your optimized code? [5 points]
3. How many memory-data references will be executed in your optimized code? ? [5 points]
4. what is the code size of your optimized code? [5 points]

You may want to examine the execution of your MIPS programs you write. For this purpose, it will be helpful to learn the basic operation of the SPIM simulator (and its graphical counterpart, QtSpim). You can download spim simulator at http://pages.cs.wisc.edu/ larus/spim.html and spim tutorials at http://pages.cs.wisc.edu/ larus/spim.html#information

Problem 3: ISA (20 points)

Stack	Accumulator	Register (register-memory)	Register (load-store)
`Push A`	`Load A`	`Load R1,A`	`Load R1,A`
`Push B`	`Add B`	`Add R3,R1,B`	`Load R2,B`
`Add`	`Store C`	`Store R3,C`	`Add R3,R1,R2`
`Pop C`			`Store R3,C`

The code sequence for `C = A + B` for four classes of instruction sets. Note that the **`Add`** instruction has implicit operands for stack and accumulator architectures, and explicit operands for register architectures. It is assumed that A, B, and C all belong in memory and that the values of A and B cannot be destroyed. Figure B.1 shows the **`Add`** operation for each class of architecture.

Figure 1: Code sequence for C=A+B for four classes of instruction sets

Assume A,B, and C reside in memory. Also assume that instruction operation codes are represented in 8 bits, memory addresses are 64 bits, and register addresses are 6 bits. For each instruction set architecture in Figure 1, calculate the total code size for the code to compute C = A + B.

ISA	Total code size	Calculation details	points
Stack			5 points
Accumulator			5 points
Register-memory			5 points
Register-register			5 points

What to submit

Submit hardcopy of your solution before class starts at 3:30pm on 02/17/2015.