Student Name: Sitare Arslantürk

Student ID: 57677

COMP 303 - Computer Architecture: HW1

Due: Oct 13, 2019, 6pm

Instructor: Didem Unat

Corresponding TAs: Aditya Sasongko

Notes: You may discuss the problems with your peers but the submitted work must be your own work. No late assignment will be accepted. Submit a SOFT copy of your assignment to the blackboard *AND* submit a HARD copy of it to the COMP 303 mailboxes. This assignment is worth 4% of your total grade.

Problem 1

(10 pts)

a) In MIPS assembly, write an assembly language version of the following C code segment. At the beginning of the segment, the only values in registers are the base address of arrays *A* and *B* in registers $*a*0 and $*a*1. To get credit, comment your code.

```
1  int A[40], B[40];
2  for (i=1; i < 40; i++) {
3       B[i] = A[i] + A[i-1];
4       A[i] = 5*B[i] ;
5  }
```

b) Compilers can optimize loops to reduce memory operations (loads and stores) for performance. Implement more optimized version of the above loop in assembly language that uses fewer memory instructions.

```
//writeyourcodefor(a)here

        addi $t0 $zero 1        #i = 0 + 1
        addi $t1 $zero 40       #temp = 0 + 40
Loop:   slt $t2 $t0 $t1         #if(i<temp)
        beq $t2 $zero Exit      #exit if t2==0
        sll $t3 $t0 2           #t3 = i * 4
        add $t3 $t3 $a0         # t3 is the address of A[i]
        lw $t7 -4($t3)          #load t3 = A[i-1]
        lw $t4 0($t3)           #load t4 = A[i]
        add $t5 $t4 $t7         #A[i-1]+A[i]
        sw $t5 0($a1)           #store B[i]
        lw $t5 0($a1)           #load B[i]
        sll $t6 $t5 2           #B[i]*4
        addi $t6 $t6 $t5        # B[i]*4 + B[i]
        sw $t6 0($t3)           #store A[i]
        addi $t0 $t0 1          #i = i + 1
        j Loop                  #jump
Exit:   …                       #exit

//writeyourcodefor(b)here

        addi $t0 $zero 1        #i = 0 + 1
        addi $t1 $zero 40       #temp = 0 + 40
Loop:   slt $t2 $t0 $t1         #if(i<temp)
        beq $t2 $zero Exit      #exit if t2==0
        sll $t3 $t0 2           #t3 = i * 4
        add $t3 $t3 $a0         # t3 is the address of A[i]
        lw $t7 -4($t3)          #load t3 = A[i-1]
        lw $t4 0($t3)           #load t4 = A[i]
        add $t5 $t4 $t7         #A[i-1]+A[i]
        sw $t5 0($a1)           #store B[i]
        sll $t6 $t5 2           #B[i]*4, optimized no need to load B[i] again
        addi $t6 $t6 $t5        # B[i]*4 + B[i]
        sw $t6 0($t3)           #store A[i]
        addi $t0 $t0 1          #i = i + 1
        j Loop                  #jump
Exit:   …                       #exit
```

Problem 2

(6 pts) Suppose that you are asked to design a chip for an embedded system that will be used for gym equipment. Because of the cost and space restrictions, you decided to have 20-bit wide instructions and have only 16 general-purpose registers. Similar to MIPS, this chip uses a three-address ISA, which takes two sources, performs an operation on these sources and stores the result back into a destination register.

a) How will your R-type instructions look like? Assume that there is no opcode extension and shift amount is represented in 3 bits.

16 registers = 2^4

Opcode 5 bits	rs 4 bits	rt 4 bits	rd 4 bits	shamt 3 bits

b) How many different instructions you can encode in this new ISA?

5 bits opcode → $2^5 = 32$

c) How many bits should the immediate field of an I-type instruction be to match the length on an R-type instruction?

In I-type, there is no rd or shamt so, rd+shamt= 4+3 =7 bits

Problem 3

(20 pts) 0-address machine uses a stack, where all operations are done using values stored on the stack. For example,

- PUSH *addr* - pushes the value stored at memory location *addr* onto the stack.
- POP *addr* - pops the stack and stores the value into memory location *addr*
- BOP *addr* - pops two values off the stack, performs the binary operation BOP on the two values, and pushes the result back onto the stack

For example, to compute A + B with 0-address machine, the following sequence of operations are necessary: PUSH A, PUSH B, ADD. After execution of ADD, A and B would no longer be on the stack, but the value A+B would be at the top of the stack.
Make the following assumptions for the next questions and the following code snippet.

- The opcode size is 1 byte (8 bits).
- All register operands are 1 byte (8 bits).
- All memory addresses are 2 bytes (16 bits).
- All data values are 4 bytes (32 bits).
- All instructions are an integral number of bytes in length.

```
1 X = Y +Z;
2 Y = X +Z;
3 T = X -Y;
```

For (a), (b), (c), use the provided table. I-bytes refers to instruction bytes, D-bytes refers to data bytes.

(a) What is the assembly code for the above code for 0-address machine? Assume initially the variables X, Y, Z and T are in memory. Be sure to store the contents of variables back into memory. Do not modify any other values in memory.

(b) What is the assembly code for the above code for MIPS?

(c) Calculate the instruction bytes fetched and the memory-data bytes transferred (read or written) for 0-address machine and for MIPS.

(d) Which ISA is most efficient as measured by code size? Why?

(e) Which ISA is most efficient as measured by total memory traffic (code + data)? Why?

Instruction Set Architecture	Opcode (e.g. Load, Store)	Operands (e.g. X, Y, Z, T)	I-bytes	D-bytes	Total Bytes
0-address Machine	PUSH Y PUSH Z ADD POP X PUSH X PUSH Z ADD POP Y PUSH Y PUSH X SUBTRACT POP T		3 bytes 3 bytes 1 byte 3 bytes 3 bytes 3 bytes 1 byte 3 bytes 3 bytes 3 bytes 1 byte 3 bytes	4 bytes 4 bytes 0 byte 4 bytes 4 bytes 4 bytes 0 byte 4 bytes 4 bytes 4 bytes 0 byte 4 bytes	7 bytes 7 bytes 1 byte 7 bytes 7 bytes 7 bytes 1 byte 7 bytes 7 bytes 7 bytes 1 byte 7 bytes
Total	8 opcode used	9 times used	30 bytes	36 bytes	66 bytes
MIPS	lw $t0 Z lw $s0 Y add $s1 $t0 $s1 sw $s1 X add $s0 $s1 $t0 sw $s0 Y sub $s2 $s1 $s0 sw $s2 T		4 bytes 4 bytes 4 bytes 4 bytes 4 bytes 4 bytes 4 bytes 4 bytes	4 bytes 4 bytes 0 byte 4 bytes 0 byte 4 bytes 0 byte 4 bytes	8 bytes 8 bytes 4 bytes 8 bytes 4 bytes 8 bytes 4 bytes 8 bytes
Total	6 opcode used	5 times used	32 bytes	20 bytes	52 bytes

d) MIPS requires less instructions thus it is more efficient.

e) MIPS is more efficient because it takes up less memory traffic by entering and storing to the memory less.

Problem 4

(10 pts) MIPS is just one of the Instruction Set Architectures and it is a simple one. Consider a more complex instruction set, called MAPS. One instruction in MAPS can perform the function of many instructions in MIPS. In this question, you are ask to implement the MIPS equivalent for a single instruction in MAPS, namely *short_int_CP Y X*, which is defined as follows:

This instruction copies short integers from one address to another. The instruction uses three registers: CNT (count), SRC (source), and DST (destination). The CNT indicates the number of iterations. Each iteration, the instruction moves a short integer (two bytes data) from memory at address SRC to memory at address DST, and then increments both address pointers by two bytes. Thus, the instruction copies in total 2*CNT many bytes from source to destination.

(a) Write the corresponding MIPS code for *short_int_CP Y X* that performs the same function. Assume $$s1 = CN T,$ $s2 = SRC,$ $s3 = DST$. Try to minimize code size as much as possible.

```
1  . Loop:lh $t0, 0($s2)
2  .      sh $t0, 0($s3)
3  .      addi $s3, $s3, 2
4  .      addi $s2, $s2, 2
5  .      addi $s1, $s1, -1
6  .      bne $s1, $zero, Loop
7  .
8  .
9  .
10 .
11 .
12 .
13 .
14 .
15 .
16 .
17 .
18 .
```

b) Compare the code sizes of MIPS and MAPS for the same function in bytes. Which one is shorter?

MAPS is shorter because it is able to perform all the instructions written above which is about 6 lines of instruction in MIPS in only 1-line instruction execution in MAPS.

Problem 5

4 pts (This topic will be covered in the class soon)

Instruction Type	Machine M1 Cycles/Instruction	Machine M2 Cycles/Instruction	Instruction Frequency
Loads/Stores	4	4	30%
ALU Operations	1	2	60%
Branches	2	1	10%

Consider two different implementations, Machine M1 and M2, of the same instruction set. There are three types of instructions (loads/stores, ALU operations and branches) in the instruction set. M1 has a clock rate of 800 MHz and M2 has a clock rate of 1.25 GHz. The average number of cycles for each instruction type and their frequencies for an image processing algorithm are provided in the table above.

a) Calculate the average CPI for each machine, M1 and M2.

For M1,

$$\%30 * 4 + \%60 * 1 + \%10 * 2 = 2$$

For M2,

$$\%30 * 4 + \%60 * 2 + \%10 * 1 = 2.5$$

b) Compare the CPU time of each machine for the image processing algorithm.

$$CPU = \frac{Instruction\ count * CPI}{Clock\_rate}$$

For M1,

$$CPU = \frac{I * 2}{800\ MHz} = I * 0{,}0025$$

For M2,
1 GHz = 1000 MHz

$$CPU = \frac{I * 2.5}{1.25\ GHz} = \frac{I * 2.5}{1250\ MHz} = I * 0{,}0020$$

M2 is faster by $(I * 0{,}0025 / I * 0{,}0020) = 1.25$