CMPE 110 Computer Architecture

Fall 2016, Homework #1

Computer Engineering

UC Santa Cruz

October 3, 2016

Name: VLADOI MARIAN

Email: mvladoi@ucsc.edu

Submission Guidelines:

• **This homework is due on Monday October 17th 2016 11:59 pm.**

• The homework must be submitted to ecommons by 11:59pm.

– Anything later is a late submission

• Please write your name and your UCSC email address

• The homework should be “readable” without too much effort

– The homework must be typed and submitted as a single file in PDF format – Please name your homework file cmpe110-hw1-yourcruzid.pdf – Please keep your responses coherent and organized or you may lose points

• Provide details on how to reach a solution. An answer without explanation gets no credit. Clearly state all assumptions.

• Points: 64 = 16 + 12 + 24 + 12

Question	Part 1	Part 2	Part 3	Part4	Total
1					
2				-	
3			-	-	
4		-	-	-	
Total					

As you work through this homework assignment, you might want to examine the execution of MIPS programs with a known good reference. For this purpose, it will be helpful to learn the basic operation of the SPIM simulator (and its graphical counterpart, xspim). The SPIM simulator and related documentation are available at http://spimsimulator.sourceforge.net. But it is up to you to learn and use SPIM.

Question 1. The MIPS ISA (16 Points)

1.1 Translate a recursive version of the function BitCount into MIPS assembly code. This function counts the number of bits that are set to 1 in an integer. The parameter x is passed to your function in register $a0. Your function should place the return value in register $v0.

```
int BitCount(unsigned x) {
int bit;
 if (x == 0)
   return 0;
 bit = x & 0x1;
    return bit + BitCount(x >> 1);
}
```

Please use the following MIPS calling convention and its register usage (just for your reference, you may not need to use all of these registers):

• The argument x is passed in register $4.

• The result (i.e., c) should be returned in $2.

• $8 to $15 are caller-saved temporary registers.

• $16 to $23 are callee-saved temporary registers.

• $29 is the stack pointer register.

• $31 stores the return address.

A summary of the MIPS ISA is provided at the end of this handout, and a MIPS reference sheet can be found at https://inst.eecs.berkeley.edu/~cs61c/resources/MIPS_Green_Sheet.pdf. The MIPS architecture reference manual can be found at http://www.cs.cornell.edu/courses/cs3410/2015sp/MIPS_Vol1.pdf.

```
countbits: addi $2, $zero, 0          # initialize $2 to 0
           beq  $4, $zero, done       # if (x == 0) return
           bge  $4, $zero, rec        # if (x > 0) rec
           addi $2, $2, 1             # $2 = 1
     rec:  addi $29, $29, -8          # make stack space
           sw   $31, 4($29)           # save return address
           sw   $2, ($29)             # save $2
           sll  $4, $4, 1             # x >> 1
           jal  countbits             # recursive call
           lw   $31, 4($29)           # restore $ ra
           lw   $16, ($29)            # restore $2
           addi $29, $29, 8           # restore stack
           add  $2, $2, $16           # return value in $2
    done:  jr   $ra
```

1.2 Calculate the number of total number of data memory accesses made during execution of

the code.

Data memory is accessed 4 times, because we have 2 sw instructions and 2 lw instruction.

1.3 Calculate the CPI of this program. Assume every load instruction takes 3 cycles, every store instruction takes 2 cycles, and all other instructions take 1 cycle. Also assume that instructions are executed in sequence (one after the other).

We have 14 instructions in total.

2 lw - 3 cycles

2 sw - 2 cycles

10 other - 1 cycle

CPI = (0.142 * 3) + (0.142 * 2) + (0.714 * 1) =1.424

1.4 Assuming a clock period of 500 ns, what is the execution time of this program?

Execution time = instruction count * CPI * clock frequency

= 14 instructions/program * 1.424 cycles/instructions * 5 X 10^{-7} s/cycles

= 9.968 X 10^{-6} s/program

Question 2. Performance Evaluation. (12 Points)

Evaluate the potential performance of two processors, each implementing a different ISA. The evaluation is based on its performance on a particular benchmark. On the processor implementing ISA A, the best compiled code for this benchmark performs at the rate of 10 IPC. That processor has a 1.5 GHz clock. On the processor implementing ISA B, the best compiled code for this benchmark performs at the rate of 2 IPC. That processor has a 1.8 GHz clock. Answer following questions and briefly explain your answers.

2.1 What is the performance in Millions of Instructions per Second (MIPS) of the processor

implementing ISA A?

10 IPS = 1/10 CPI = 0.1 CPI

1.5 GHz => 1 cycle = 0.67 ns

1 mil instruction takes $0.1 * 10^6 * 0.67$ ns = $6.7 * 10^{-5}$ s

MIPS = $1 / 6.7 * 10^{-5}$ = 14925

2.2 What is the performance in MIPS of the processor implementing ISA B?

2 IPS = ½ CPI = 0.5 CPI

1.8 GHz => 1 cycle = 0.56 ns

1 mil instruction takes $0.5 * 10^6 * 0.56$ ns = $2.8 * 10^{-4}$ s

MIPS = $1 / 2.8 * 10^{-4}$ = 3571

2.3 Which is the higher performance processor: A, B, or don't know?

Performance = 14925 / 3571 = 4.2 Processor A is the higher performance . 318% faster.

Question 3. Amdahl's Law (24 Points)

A vector pipeline processor is an extension to a conventional CPU, which provides registers that can hold an entire vector of floating-point operands (e.g. 64 words). In addition, the machine would have instructions which manipulate vectors. For example, to implement the following loop:

for i := 1 to 64 do

for j := 1 to 64 do

Y[k,j] := a * X[i,j] + Y[k,j];

This could be programmed as follows:

R1 := 0

R2 := 0

R3 := k*N

for i := 1 to 64 do

LV V2, X(R1) ; load row of X (R1 is i*N, where N is row length)

MULTSV V3, a, V2 ; multiply each element of the row vector by scalar 'a'

LV V1, Y(R2) ; load k'th row of Y, where R2 is k*N)

ADDV V4, V3, V1 ; add Y[i] and a*X[i]

SV V4,Y(R3) ; store the vector into array Y

R1 := R1+N

R2 := R2+N

end do

The classic vector pipeline machine is the Cray 1, followed by many later designs from Cray and others (e.g., the NEC SX-6 used in the Earth Simulator). The main advantages of vector registers and vector instructions are that pipelined floating-point arithmetic can be organized very efficiently, and the maximum throughput of a highly interleaved memory system can be exploited.

3.1 Assume that we are considering accelerating a machine by adding a vector mode to it. When a computation is run in vector mode, it is 10 times faster than the normal mode of execution. We call the percentage of time that could be spent using vector mode the percentage of vectorization.

(1) What percentage of vectorization is needed to achieve a speedup of 3?

1 / [(1 - fraction vectorized) + (fraction vectorized / 10)] = 3

fraction vectorized = (3 * 10 - 10) / (3 * 10 - 3) = 74.1 %

(2) What percentage vectorization is needed to achieve one-half the maximum speedup available from using vector mode?

The maximum speedup is 10. The half of the maximum speed up is 5.

1 / [(1 - fraction vectorized) + (fraction vectorized / 10)] = 5

fraction vectorized = (5 * 10 - 10) / (5 * 10 - 5) = 88.8 %

(3) Suppose you have measured the percentage of vectorisation for programs to be 80%. The hardware design group says they can double the speed of the vector mode with a significant engineering investment. You wonder whether the compiler crew could increase the use of vector mode as another approach to increasing performance. How much of an increase in the

percentage of vectorisation (relatitive to current usage) would you need to obtain the same performance gain? Which investment would you recommend?

80% vectorization yields a net speed up of 1/ (0.2 + (0.8 / 10)) = 3.572

Increasing the vector mode speedup to a factor of 20 :

1/ (0.2 + (0.8/20)) = 4.167

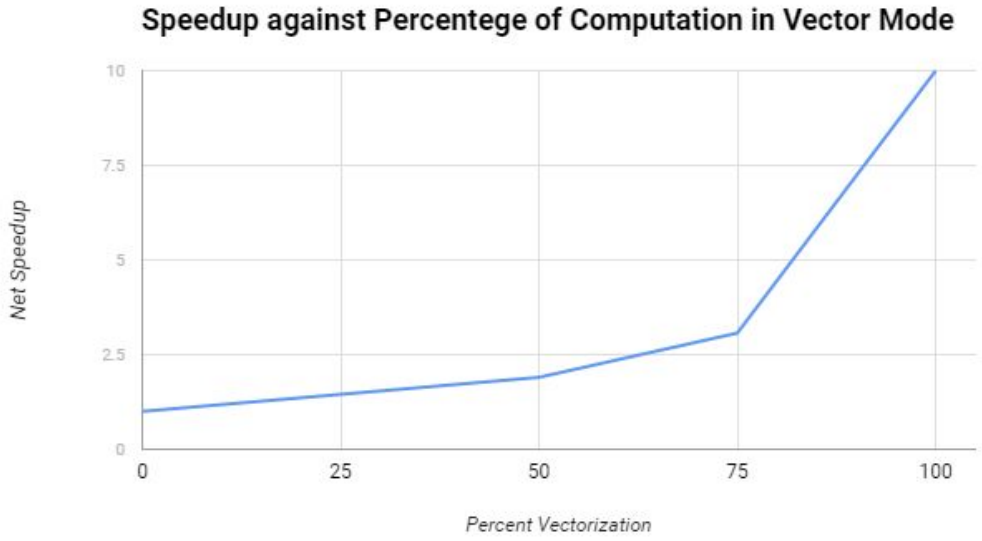
1 / [(1 - fraction vectorized) + (fraction vectorized / 10)] = 4.167

fraction vectorized = (4.167* 10 - 10) / (4.167 * 10 - 4.167) = 84.4 %

Only 10 % better that the original . I would not recommend doubling the speed up.

(4) Draw (sketch) a graph that plots the speedup as a percentage of the computation performed in vector mode. Label the y-axis “Net Speedup”, and label the x-axis “Percent Vectorization”.

- when the percentage vectorised is 0%, net speedup is 1.
- when the percentage vectorised is 100%, net speedup is 10.
- when the percentage vectorised is 50%, net speedup is 1.9.
- when the percentage vectorised is 75%, net speedup is 3.07.



3.2 Assume that we make an enhancement to a computer that improves some mode of execution by a factor of 5. Enhanced mode is used 75% of the time, measured as a percentage of the execution time when the enhanced mode is in use (rather than defined as the percentage of the running time without the enhancement is used)

(1) What is the speedup we have obtained from fast mode?

Accelerated execution time consisted of two halves: accelerated phase 75% and unaccelerated phase 25%.

So the relative execution time without the enhancement would be 25% + (75% * 5) = 400%

Thus the overall speedup is: ex unaccelerated/ ex accelerated = 400% / 100% = 4

(2) What percentage of the original execution time has been converted to fast mode?

Fraction Vectorised = (speedup overall * speedup accelerated - speedup accelerated)/

Speedup overall * speedup accelerated - speedup overall)

= (4 * 5 - 5) / (4 * 5 - 4) = 93%

Question 4. Addressing Modes (12 Points)

We covered the following addressing modes in class:

• Absolute

• Register indirect

• Based (base + displacement)

• Scale indexed (base + index × constant)

• Memory indirect

• Auto increment/decrement (by 1 byte)

Consider the following high-level programs:

(1) int a = 0; uint8_t D[100]; // D is allocated in memory While (a < 100) { D[a] = a + 5; a += 1; }

(2) int a = 0; int D[100]; // D is allocated in memory While (a < 100) { D[a] = a + 5; a += 1; }

(3) int *p; // *p is allocated in memory *p = 100;

(4) int **p; // *p and **p are allocated in memory **p = 100;

Assume that in the first two programs, a register contains the address of the start of the array, and in the last two programs, a register contains the value of p.

For each of the above four programs, which of the addressing modes, do you think, would lead to the minimum number of instructions? (Note that no addressing mode fits perfectly. You might require other instructions for address computation.)

(1) Auto increment < (2) Scale indexed <(3) Register indirect <(4) Memory indirect

MIPS Instruction Summary

Opcode Example Assembly Semantics add add $1, $2, $3 $1 = $2 + $3

sub sub $1, $2, $3 $1 = $2 - $3

add immediate addi $1, $2, 100 $1 = $2 + 100

add unsigned addu $1, $2, $3 $1 = $2 + $3

subtract unsigned subu $1, $2, $3 $1 = $2 - $3 add immediate unsigned addiu $1, $2, 100 $1 = $2 + 100
multiply mult $2, $3 hi, lo = $2 * $3

multiply unsigned multu $2, $3 hi, lo = $2 * $3

divide div $2, $3 lo = $2/$3, hi = $2 mod $3

divide unsigned divu $2, $3 lo = $2/$3, hi = $2 mod $3

move from hi mfhi $1 $1 = hi

move from low mflo $1 $1 = lo

and and $1, $2, $3 $1 = $2 & $3

or or $1, $2, $3 $1 = $2 | $3

and immediate andi $1, $2, 100 $1 = $2 & 100

or immediate ori $1, $2, 100 $1 = $2 | 100

shift left logical sll $1, $2, 10 $1 = $2 << 10

shift right logical srl $1, $2, 10 $1 = $2 >> 10

load word lw $1, 100($2) $1 = memory[$2 + 100]

store word sw $1, 100($2) memory[$2 + 100] = $1

load upper immediate lui $1, 100 $1 = 100 << 16

branch on equal beq $1, $2, label if ($1 == $2) goto label

branch on not equal bne $1, $2, label if ($1 != $2) goto label

set on less than slt $1, $2, $3
if ($2 < $3) $1 = 1 else $1 = set on less than immediate slti $1, $2, 100
if ($2 < 100) $1 = 1 else $1 = set on less than unsigned sltu $1, $2, $3
if ($2 < $3) $1 = 1 else $1 = set on less than immediate sltui $1, $2, 100 if ($2 < 100) $1 = 1 else $1 = 0

jump j label goto label

jump register jr $31 goto $31

jump and link jal label $31 = PC + 4; goto label