CS 402 Homework 3

Muhammad Umar

2.12 Assume that registers $s0 and $s1 hold the values 0x80000000 and 0xD0000000, respectively.

2.12.1 What is the value of $t0 for the following assembly code?

add $t0, $s0, $s1

0x8000 0000 + 0xD000 0000 = 0x1 5000 0000

Since the answer is beyond 32-bits or 8 bytes, **only 0x5000 0000 will be stored in $t0** due to overflow

2.12.2 Is the result in $t0 the desired result, or has there been overflow?

No, it is not the desired result as there has been a overflow.

2.12.3 For the contents of registers $s0 and $s1 as specified above, what is the value of $t0 for the following assembly code?

sub $t0, $s0, $s1

Subtraction is equivalent to adding a two's complement number in assembly.

Hence,

0xD000 0000

One's complement = 0010 1111 1111 1111 1111 1111 1111 1111

Two's complement (+1) = 0011 0000 0000 0000 0000 0000 0000 0000 = 0x3000 0000

0x8000 0000 - 0xD000 0000 = 0x8000 0000 + 0x3000 0000 = 0xB000 0000

2.12.4 Is the result in $t0 the desired result, or has there been overflow?

It is the desired result, there is no overflow.

2.12.5 For the contents of registers $s0 and $s1 as specified above, what is the value of $t0 for the following assembly code?

add $t0, $s0, $s1

add $t0, $t0, $s0

0x8000 0000 + 0xD000 0000 = 0x1 5000 0000

0x1 5000 0000 + 0x8000 0000 = 0x1 D000 0000

$t0 = 0xD000 0000 due to overflow

2.12.6 Is the result in $t0 the desired result, or has there been overflow?

There has been overflow.

2.19 Assume the following register contents:

$t0 = 0xAAAAAAAA, $t1 = 0x12345678

2.19.1 For the register values shown above, what is the value of $t2 for the following sequence of instructions?

sll $t2, $t0, 44 or $t2, $t2, $t1

0x12345678 = 1010 1010 1010 1010 1010 1010 1010 1010

On shifting $t0 left 44 times, we get

$t2 = 0000 0000 0000 0000 0000 0000 0000 0000

Hence, after doing a logical OR with 0, we will get

$t2 = 0x12345678

2.19.2 For the register values shown above, what is the value of $t2 for the following sequence of instructions?

sll $t2, $t0, 4

andi $t2, $t2, -1

0x12345678 = 1010 1010 1010 1010 1010 1010 1010 1010

On shifting $t0 left 44 times, we get

$t2 = 1010 1010 1010 1010 1010 1010 1010 0000 => 0xAAAA AAA0

-1 = 1111 1111 1111 1111 1111 1111 1111 1111

Hence, the AND operation has no effect and

$t2 = 0xAAAA AAA0

2.19.3 For the register values shown above, what is the value of $t2 for the following sequence of instructions?

srl $t2, $t0, 3

andi $t2, $t2, 0xFFEF

0x12345678 = 1010 1010 1010 1010 1010 1010 1010 1010

On shifting $t0 right 3 times, we get

$t2 = 0001 0101 0101 0101 0101 0101 0101 0101

0xFFEF = 0000 0000 0000 0000 1111 1111 1110 1111

Applying AND operation

$t2 = 0000 0000 0000 0000 0101 0101 0100 0101 => 0x0000 5545

2.20 Find the shortest sequence of MIPS instructions that extracts bits 16 down to 11 from register $t0 and uses the value of this field to replace bits 31 down to 26 in register $t1 without changing the other 26 bits of register $t1

Prepare $t0 for extraction

srl $t0, $t0, 11 # t0 = t0 » 11

sll $t0, $t0, 26 # t0 = t0 « 26

Remove bits from 31 to 26 = 6 bits

sll $t1, $t1, 6

srl $t1, $t1, 6

Add $t0 bits to $t1 and store in $t1

or $t1, $t1, $t0

2.21 Provide a minimal set of MIPS instructions that may be used to implement the following pseudo instruction:

not $t1, $t2 // bit-wise invert

MIPS uses three operands and hence, nor is used with zero instead of not with two operands.

nor $t1, $t2, $0

2.22 For the following C statement, write a minimal sequence of MIPS assembly instructions that does the identical operation. Assume $t1 = A, $t2 = B, and $s1 is the base address of C.

A = C[0] « 4;

lw $t1, 0(s1)$ # load C in A sll $t1, $t1, 4 # Shift A by 4

2.23 Assume $t0 holds the value 0x00101000. What is the value of $t2 after the following instructions?

slt $t2, $0, $t0

bne $t2, $0, ELSE

j DONE

ELSE: addi $t2, $t2, 2

DONE:

slt sets $t2 to 1 as $0 < $t0 (0x0010 1000)

The execution jumps to ELSE as $t0 != 0 is true

Finally, add 2 to $t2's current value (1), we get

$t2 = 3

2.26 Consider the following MIPS loop:

LOOP: slt $t2, $0, $t1

beq $t2, $0, DONE

subi $t1, $t1, 1

addi $s2, $s2, 2

j LOOP

DONE:

2.26.1 Assume that the register $t1 is initialized to the value 10. What is the value in register $s2 assuming $s2 is initially zero?

The loop will run 10 times, adding 2 to $s2 each time.

Hence, if $s0 is 0 at the beginning, the value stored will be 20 by the end.

2.26.2 For each of the loops above, write the equivalent C code routine. Assume that the registers $s1, $s2, $t1, and $t2 are integers A,B, I, and temp, respectively.

This code assumes i is initialized by some value (int i = 10 for part 1 for example)

for (; 0 < i; i–){ B+=2; }

2.26.3 For the loops written in MIP assembly above, assume that the register $t1 is initialized to the value N. How many MIPS instructions are executed?

In every iteration, 5 instructions are executed except for the last iteration when the beq is true where only 2 instructions are executed. Hence,

Instructions count = 5 x (N-1) + 2

Instructions count = 5N - 3

2.27 Translate the following C code to MIPS assembly code. Use a minimum number of instructions. Assume that the values of a, b, i, and j are in registers $S0, $S1, $t0, and $t1, respectively. Also, assume that register $S2 holds the base address of the array D.

for (i = 0; i < a; i++)

for (j = 0; j < b; j++)

```
D[4*j] = i + j;
```

Ans:

and $t0, $t0, 0 # int i = 0

// s0 = i limit

// s1 = j limit

OUTER: beq $t0, $s0, DONE and $t1, $t1, $0 # set j = 0

INNER: beq $t1, $s1, INNER_END addu $t9, $t0, $t1 # t9 = i+j sll $k0, $t1, 4 # Multiply j by 4 and store in $k0 #sw $t9, 0($k0) # Store i+j to address D[j*4]

```
addiu $t1, $t1, 1   # j++
j INNER
```

INNER_END: addiu $t0, $t0, 1 #i++ j OUTER

DONE:

2.38 Consider the following code:

lbu $t0, 0($t1) sw $t0, 0($t2)

Assume that the register $t1 contains the address 0x1000 0000 and the register $t2 contains the address 0x1000 0010. Note the MIPS architecture utilizes big-endian addressing. Assume that the data (in hexadecimal) at address 0x1000 0000 is: 0x11223344. What value is stored at the address pointed to by register $t2?

MIPS processor uses Big Endian addressing. hence lbu will load the most significant byte (0x11).

Hence, 0x0000 0011 is stored at address pointed to by register $t2.

2.39 Write the MIPS assembly code that creates the 32-bit constant 0010 0000 0000 0001 0100 1001 0010 0100 two and stores that value to register $t1.

0010 0000 0000 0001 0100 1001 0010 0100 = 0x2001 4924

lui $t1, 0x2001 ori $t1, 0x4924

2.46 Assume for a given processor the CPI of arithmetic instructions is 1, the CPI of load/store instructions is 10, and the CPI of branch instructions is 3. Assume a program has the following instruction breakdowns: 500 million arithmetic instructions, 300 million load/store instructions, 100 million branch instructions.

Execution time = (Instructions * CPI)/Clock Rate or Cycles/Clock Rate

Instructions = (500 * 1) + (300 * 10) + (100 * 3) = 3800

Execution time = 3800 * 10^6 * Clock Cycle Time

2.46.1 Suppose that new, more powerful arithmetic instructions are added to the instruction set. On average, through the use of these more powerful arithmetic instructions, we can reduce the number of arithmetic instructions needed to executea program by 25%, and the cost of increasing the clock cycle time by only 10%. Is this a good design choice? Why?

Execution time = (Instructions * CPI)/Clock Rate or Cycles/Clock Rate

New Instructions = (0.75 * 500 * 1) + (300 * 10) + (100 * 3) = 3675

New Execution Time = 3675 * 1.1 * 10^6 * Clock Cycle Time.

New Execution Time = 4.0425×10^9 * Clock Cycle Time.

The execution time or CPU time has overall increased. Hence this is a bad design choice.

2.46.2 Suppose that we find a way to double the performance of arithmetic instructions. What is the overall speedup of our machine? What if we find a way to improve the performance of arithmetic instructions by 10 times?

Doubling would mean the instructions counts for arithmetic would half.

Execution time = (Instructions * CPI)/Clock Rate or Cycles/Clock Rate

New Instructions = (0.5 * 500 * 1) + (300 * 10) + (100 * 3) = 3550

New Execution Time = 3550 * 10^6 * Clock Cycle Time.

Speedup = 3550/3800 = 0.9342105

Hence, the overall improvement in execution time is 6.5789474%

Improving by 10 times would mean the instructions counts for arithmetic would only be 10% of the original.

Execution time = (Instructions * CPI)/Clock Rate or Cycles/Clock Rate

New Instructions = (0.1 * 500 * 1) + (300 * 10) + (100 * 3) = 3350

New Execution Time = 3350 * 10^6 * Clock Cycle Time.

Speedup = 3350/3800 = 0.8815789

Hence, the overall improvement in execution time is just 11.8421053% despite the massive decrease in arithmetic instructions.

2.47 Assume that for a given program 70% of the executed instructions are arithmetic, 10% are load/store, and 20% are branch.

2.47.1 Given this instruction mix and the assumption that an arithmetic instruction requires 2 cycles, a load/store instruction takes 6 cycles, and a branch instruction takes 3 cycles, find the average CPI.

Average CPI = (0.7 * 2) + (0.1 * 6) + (0.2 * 3) = 2.6

2.47.2 For a 25% improvement in performance, how many cycles, on average, may an arithmetic instruction take if load/store and branch instructions are not improved at all?

New Average CPI = 0.75 * 2.6 = 1.95

1.95 = (0.7 * New Arithmetic Cycles) + (0.1 * 6) + (0.2 * 3)

New Arithmetic Cycles = (1.95 - 1.2)/0.7

New Arithmetic Cycles = 1.0714286 cycles or 1.07 cycles.

2.47.3 For a 50% improvement in performance, how many cycles, on average, may an arithmetic instruction take if load/store and branch instructions are not improved at all?

New Average CPI = 0.5 * 2.6 = 1.3

1.3 = (0.7 * New Arithmetic Cycles) + (0.1 * 6) + (0.2 * 3)

New Arithmetic Cycles = (1.3 - 1.2)/0.7

New Arithmetic Cycles = 0.1428571 cycles or 0.14 cycles.