Computer Science 605.611

Mid Term Exam

1. A 16-bit system used to represent floating-point numbers has the following format:

|  |  |  |
| --- | --- | --- |
| S | Characteristic | Fraction |
| 15 | 14 10 | 9 0 |

The MSB (S) is the sign bit (1 for -, 0 for +). Bits 14 through 10 are the 5-bit excess-15 representation of the signed exponent. Bits 9 through 0 are the 10-bit fraction. There is a hidden assumed 1 to the left of the fraction and the binary point is between the hidden 1 and the fraction. This system has no representation for Nans, ±infinity or denormalized values.

a) (5) What decimal value does the pattern 0xE7FF represent in this system when the pattern is interpreted as a floating-point number?

This becomes 1110 0111 1111 1111, so sign 1 exponent 11001 and fraction 1111111111. So number is negative and 11001 = 25, we need to remove the bias 25-15 = 10. So 1.1111111111 \* 2^10 = - 11111111111 = **-2047**

b) (5) What is the largest positive value that can be represented in this system? Express your answer as a decimal number.

So the largest number in binary would be 0111 1111 1111 1111 (0 since it needs to be positive). Exponent now equal 11111 = 31, remove 15 for bias so 16. 1.1111111111 \* 2^16, we already know 11111111111 = 2047, and the remaining values would be 64 (2^6), so 2047 \* 64 = **131008**.

2. (5) Initially register $8 contains the 32-bit pattern 0xABCDEF12. Assume that rol (rotate left) and ror (rotate right) pseudo-instructions are builtin true-op instructions so that no substitutions have to be made for them by the assembler. Use 8 hex digits to show the final 32-bit pattern in $8 after the following instructions are executed:

lui $2, 255

ori $2, $2, 255

sll $3, $2, 8

rol $1, $8, 8

and $1, $2, $1

ror $8, $8, 8

and $8, $3, $8

or $8, $1, $8

Lets just go through what it does:

Lui loads the immediate value 255 into register $2, ie placing 0x000000FF into register $2.

Ori does the bitwise or operation on $2 and 255, since they are the same number it remains 255 in $2.

Sll then shifts the value in $2 by 8 bits and stores it in $3. Since its just left shift and all values besides the first 2 are 0, we can move the FF without issue, 0x0000FF00 in $3

Rol then rotates left our initial value 0xABCDEF12 and sores it in $1. Rotations are nice and easy making $1 = 0xCDEF12AB.

And does the obvious of anding $1 and $2 and placing it in $1.$2 still has 0x000000FF and $1 has our now rotated number 0xCDEF12AB. Luckily the first 24 values will be 0 and the last 8 will be whatever is in $1 due to $2. $1 now holds 0x000000AB.

Ror does a rotate right of $8 by 8 bits and stores it into $8. $8 still contains our original value 0xABCDEF12, and after a rotate it is 0x12ABCDEF.

And once again performs the and function on $3 and $8 and stores it in $8. $8 contains 0x12ABCDEF and $3 contains 0x0000FF00. Based on the previous method we can quickly get $8 contains 0x0000CD00.

Or is the final operation performed between $1 and $8. $1 contains 0x000000AB and $8 contains 0x0000CD00. The or gives us **0x0000CDAB** stored in $8.

3. (5) CPU register $7 contains the 32-bit IEEE-754 floating representation of -11.5. What floating point value is represented by the pattern that is produced in register $7 by the MIPS instruction: sra $7, $7, 2. Express your answer in the format ±d.dddE±dd where each d is a decimal digit.

In 32 bit IEEE 754 format, 1 bit is the sign bit, 8 bits are exponent, the final 23 bits are the fractional parts of the number.

So 1 is the sign bit, 10000010 is the exponent (needs to be -2 and the bias is 127 in decimal), and 01110000000000000000000 for the fraction. Then we can perform the sra (shift right arithmetic) and it is shifting the value 2 bits. Since its an arithmetic shift we fill based on sign bit. So

11000001001110000000000000000000 = 11110000010011100000000000000000. So sign is still 1, exponent is now 11100000 and 10011100000000000000000 finishes the rest. So our final value is **-2.559E+29**.

4. a) (3) Is the single-cycle data path an example of a Princeton (i.e., von Neumann) architecture or is it an example of a Harvard architecture? Explain your answer.

If we are discussing our single-cycle data path we are discussing a Harvard architecture. This is because we need separate instruction and data memories.

b) (3) Is the multi-cycle data path an example of a Princeton (i.e., von Neumann) architecture or is it an example of a Harvard architecture. Explain your answer.

This question is a little more complex because a multi-cycle datapath could be an example of either of the architectures. However, if we are discussing our specific data path it would most closely represent the Princeton architecture. In module 5 it is discussed that we would use a single memory that contains both instructions and data, removing the need for multiple buses.

c) (3) Our MIPS core instruction subset was defined and used multiple times on earlier examples and assignments. Based on the MIPS core instruction subset, what is the theoretical peak MIPS rating for a program that runs on the multi-cycle data path with a 2 GHz clock rate?

Lets assume 1 million

Peak MIPS = clock\_rate/CPImin \* 10^6 = 2GHz/1 \* 10^6 = **2000**

5. a) (3) A sw instruction and a lw instruction that immediately follows it in the pipeline both access that same word in memory. Can the sw instruction cause a data hazard for the lw instruction? Explain why or why not.

Sw cannot cause a data hazard. This is because sw does not write a result, so lw could not access a variable that is currently being written to.

b) (3) Can a sw instruction in stage 4 of the pipeline cause a data hazard for a lw instruction that is in stage 2 by overwriting the memory word read by the lw instruction? Explain why or why not.

This could cause a data hazard. This is because, while under normal circumstances sw cannot cause a data hazard, this is not a normal circumstance. The sw instruction will be writing to the word at the same time as the lw instruction is reading it. This can cause the lw instruction to read the data that hasn’t been changed yet meaning the overall result may be incorrect.

6. Assume that on the multi-cycle datapath all instructions except for the lw instruction require 4 cycles to execute. The lw instruction takes 5 cycles on the multi-cycle datapath. Consider the following instruction sequence:

lui $2, 0x4004

ori $5, $0, 50

lui $3, 0x400C

lw $4, 80($2)

addu $6, $5, $5

lw $7, 0($3)

addiu $3, $3, 4

addu $9, $4, $5

nop

sw $7, 0($2)

a) (3) How many nano-seconds will this instruction sequence take to execute on the multi-cycle datapath if the clock cycle time is 4 nano-seconds?

So, lets just do a calculation, we have 8 instructions that require 4 cycles and 2 lw instructions that require 5 cycles. So 4 \* 8 + 2 \* 5 = 42 cycles total, meaning 4 ns \* 42 cycles = **168 ns to complete all the instructions**.

b) (5) How many nano-seconds will this instruction sequence take to execute on our 5-stage MIPS pipelined system if the clock cycle time is 4 nano-seconds? Assume that the pipelined system includes a data hazard unit as well as a data forwarding unit.

Since we added the data hazard unit we can assume that lw now only takes 4 cycles to execute like the other instructions. Thus we now have 10 instructions that take 4 cycles each, meaning 40 cycles total. 4 ns \* 40 cycles = **160 ns to complete all instructions**.

7. (5) The contents of register $4 is set to 0xC0BE412C and the contents of register $3 is set to 0xFFFFFF00 before each of the two instructions listed below is executed. These patterns are the initial contents of $4 and $3 for both instructions. Identify all result registers written by each instruction and use 8 hex digits to show the contents of each register that is written.

mult $4, $3 register written is \_$hi\_\_\_ contents of written register = \_0xC0BE406B\_\_

register written is \_$lo\_\_\_ contents of written register = \_0x41BED000\_\_

mul $4, $4, $3 register written is \_$4\_\_\_ contents of written register = \_0x41BED000\_\_

8. (3) The machine instruction 0x1000FFFF is located in memory at address 0x400400C0. What value will the PC register contain immediately after this machine instruction is executed on the single-cycle datapath?

I believe that the machine instruction is just a normal instruction and not a jump or branch. Since it isn’t that the PC would simply add 4 to the current location. So **0x400400C4** should be in the PC register.

9. a) (2) Show how a single “or” instruction can be used to set register $7 to zero.

The easiest way to do so is simply:  
**or $7 $0 $0**

This ors $0 to $0 which is all 0’s in mips, making $7 zero.

b) (3) Assume that our pipeline runs at a 2 GHz clock rate and is initially empty. The pipeline includes a data hazard unit but it does not include a data forwarding unit.

If the first instruction in the sequence shown below is fetched at time t=0, and only necessary stalls are performed, at what time will the sw instruction exit the pipeline?

nop

sll $0, $0, 0

add $0, $0, $0

sw $0, 400($0)

First instruction is a no op instruction meaning 5 cycles taken but nothing was done. Sll is only using the 0 register and 0 which more or less also makes it a no op instruction. add is in the same instance as sll. Since sw wont have any data hazards it should go through pretty quickly. Sw enters the pipeline at 4 cycles and will complete at **t=9.**

10. Use hex to show the 32-bit MIPS machine instructions that correspond to a workable implementation of each of the following assembly language statements. Each implementation should contain no more than two machine instructions. Use 8 hex digits to show each machine instruction.

a) (3) addiu $4, $6, -13634

since that number is actually within range of addiu, we can simply perform the operation.

001001 00100 00110 1100101000111110 is the twos complement binary representation of each part

**0x2486CA3E**

b) (3) addiu $5, $7, 51902

This one cannot be done in 1 step as 51902 is outside the range of addiu (-32768, 32767) since those are the maximum numbers that fit inside the 16 bit immediate. Thus

**0x3401CABE ori $1, $0, 0xCABE (or 51902 into register $1)**

**0x00A13821 addu $5, $7, $1 (add the values of $1 and $7 and store it in $5)**

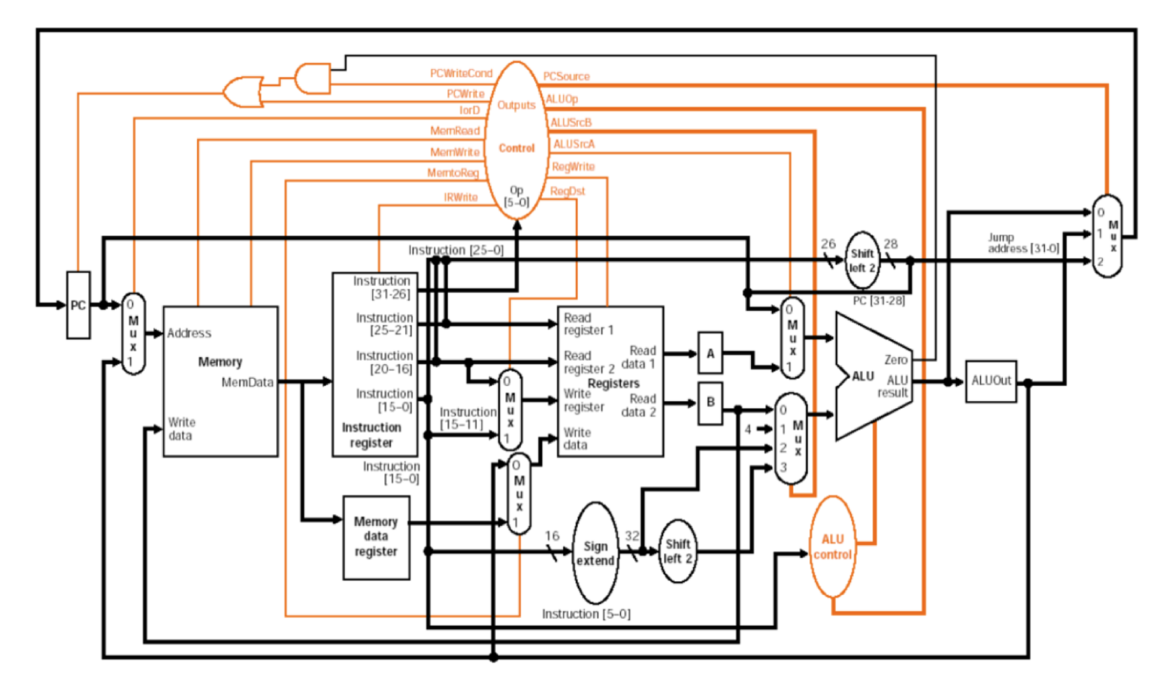
c) (3) div $8, $9

In binary this would be 000000 01000 01001 0000000000 011010, turning this into binary makes it

**0x0109001A**

11. (5) Consider the instruction: ori $4, $0, -2

Explain whether the multi-cycle data path shown in the diagram below DOES or DOES NOT support this instruction.



This does allow for the ori instruction above. The ori instruction needs a few things to work. The main ones being the ability to read registers and then write to a register, both of which this can do. But also an important item is the sign extend so you can extend the immediate to 32 bits. Thus ori should be able to be run in this multi-cycle datapath.

12. a) (4) Consider a 32-bit signed integer M in two’s complement form. Does the 32-bit result (2^32) – M match the 32-bit two’s complement representation of -M (i.e., negative M)?

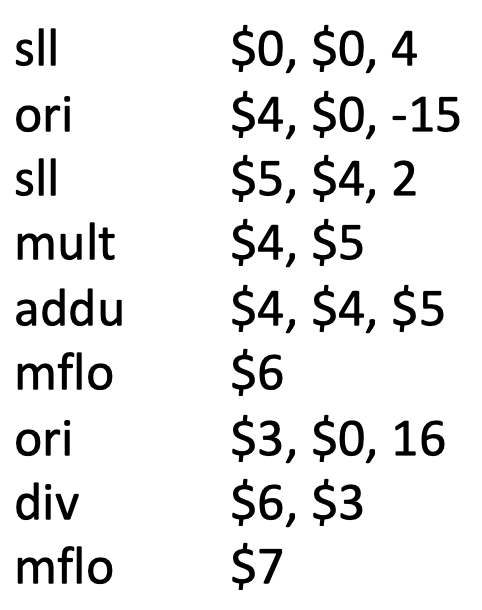
Explain your answer.

Yes it is. This is because 2^32 is equivalent to a 34 bit twos complement number 0100000000000000000000000000000000. When you subtract M from it it ends up giving you -M do to the zero’s being in the same place as the entirety of M, with subtracting you more or less just invert the binary giving you -M.

b) (4) Show that the 32-bit excess-4294967296 representation of -N is the same bit pattern as the 32-bit two’s complement representation of -N, where -N = decimal -332.

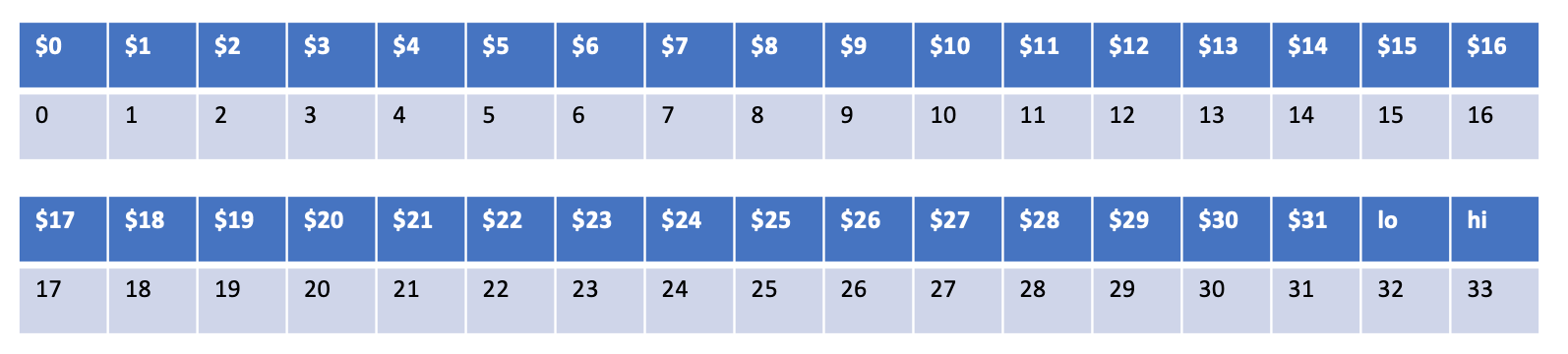
In 32 bit twos complement -332 is represented by 11111111111111111111111010110100. If we then convert that into normal decimal representation it leaves us at 4294966964. We then would need to subtract the excess of 4294967296. 4294966964-4294967296 = -332, our original number.

13. (10) Assume that the hi and lo registers, used by the mult and div instructions, are read in pipeline stage 2 and written in pipeline stage 5, like the other CPU registers. Complete the table below to show how the following instructions flow through our MIPS 5-stage pipeline with a data hazard unit but without a data forwarding unit. Insert pipeline bubbles only if and where they are needed. The instructions should be executed in the same order as they appear in the sequence. Include as many additional rows in the table as you need.



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Clock cycle | Fetch | Decode | Execute | Memory | Write-back |
| 1 | Sll $0 $0 4 |  |  |  |  |
| 2 | Ori $4 $0 -15 | Sll $0 $0 4 |  |  |  |
| 3 | Sll $5 $4 2 | Ori $4 $0 -15 | Sll $0 $0 4 |  |  |
| 4 | Mult $4 $5 | Sll $5 $4 2 | Ori $4 $0 -15 | Sll $0 $0 4 |  |
| 5 | Addu $4 $4 $5 | Mult $4 $5 | Sll $5 $4 2 | Ori $4 $0 -15 | Sll $0 $0 4 |
| 6 |  | Addu $4 $4 $5 | Mult $4 $5 | Sll $5 $4 2 | Ori $4 $0 -15 |
| 7 | Mflo $6 |  | Addu $4 $4 $5 | Mult $4 $5 | Sll $5 $4 2 |
| 8 | Ori $3 $0 16 | Mflo $6 |  | Addu $4 $4 $5 | Mult $4 $5 |
| 9 | Div $6 $3 | Ori $3 $0 16 | Mflo $6 |  | Addu $4 $4 $5 |
| 10 | Mflo $7 | Div $6 $3 | Ori $3 $0 16 | Mflo $6 |  |
| 11 |  | Mflo $7 | Div $6 $3 | Ori $3 $0 16 | Mflo $6 |
| 12 |  |  | Mflo $7 | Div $6 $3 | Ori $3 $0 16 |
| 13 |  |  |  | Mflo $7 | Div $6 $3 |
| 14 |  |  |  |  | Mflo $7 |

14. Recall that the MIPS system contains a PC register used to fetch instructions, 32 CPU registers $0 through $31 as well as the hi and lo registers used by the mult and div instructions. The tables below show the initial decimal contents of the hi, lo and each CPU register:



a) (3) Show the hex contents of any register that is written by the machine instruction: 0x00002012.

This in decimal is 00000000000000000010000000010010. The machine instruction that this is calling is mflo. This is split by 000000 0000000000 00100 00000 010010. This means that it is mflo $4 since the middle 5 is rd and the binary converted is 4. So the hex contents of lo would be moved into $4. So **$4 now contains 0x20**.

b) (3) Show the hex contents of any register that is written by the machine instruction: 0x00E00011.

This converts to 00000000111000000000000000010001. This is calling the mthi instruction from MIPS. This is in a similar format to the instruction above but with small differences, it is split like: 000000 00111 000000000000000 010001. The register that is being moved to hi is 00111 or $7. Register **hi now contains 0x07**.

15. Module 3 describes multiple logic devices including decoders, encoders, demultiplexers and selectors. In the two circuits shown below, C is a control line and the remaining inputs are all data inputs. Which one of these four terms (decoder, encoder, demultiplexer or selector) corresponds best to each of the two circuits?

Circuit 1 Circuit 2

A diagram of a diagram

Description automatically generated

\_\_\_decoder\_\_\_\_ \_\_Selector\_\_\_

16. (3) Suppose our 5-stage MIPS pipeline contains a data hazard unit and the instruction in a load delay slot uses the register written by the lw instruction. What is the MAXIMUM number of stall cycles required for the instruction in the load delay slot WITH a data forwarding unit and WITHOUT a data forwarding unit? Explain your answer.

Lets look at each of these instances separately.  
WITH a data forwarding unit the data can be passed from the mem stage straight into the execution stage for the delayed instruction meaning **no stall would be necessary**.

WITHOUT a data forwarding unit the data would not be able to be passed like I just mentioned in the with a data forwarding unit. Due to this the pipeline does need to stall until the lw instruction gets to the write back stage, thus it needs to **be stalled for two cycles**.