CSC390 (Computer Org. & Arch.)
Exam#2- Spring 2016

SOLUTION

Name:

Student ID:

Q1. Let's assume a 4-bit MIPS instruction set architecture (i.e. all the internal registers are 4-bit registers) is executing the following the lines of codes to detect overflow in a signed arithmetic addition operation ($t0 = $t1 + $t2). Tell me what would be the final value of $t3 after executing the instructions on the following two sets of $t1 and $t2 values. Based on the values of $t3 also decide which case should have overflow.

1st case	2nd case
$t1 = 0100; $t2 = 0010	$t1 = 0101; $t2 = 0111

```
22  addu $t0, $t1, $t2 # $t0 = sum, but don't trap
23  xor $t3, $t1, $t2 # Check if signs differ
24  slt $t3, $t3, $zero # $t3 = 1 if signs differ
25  bne $t3, $zero, No_overflow # $t1, $t2 signs do not match, so no overflow
26
27  xor $t3, $t0, $t1 # signs =; sign of sum match too?
28  # $t3 negative if sum sign different
29  slt $t3, $t3, $zero # $t3 = 1 if sum sign different
30  bne $t3, $zero, Overflow # All 3 signs do not match ; no overflow
```

1st Case :

$t_1 = 0100$

$t_2 = 0010$

$t_0 = t_1 + t_2 = 0110$

$t_1 = 0100$

$t_2 = 0010$

$t_3 = XOR(t_1, t_2) = 0110$

∴ After, slt t_3, t_3, $zero (line 25)

t_3 would be ⇒ 0, Line 25 is not satisfied

Also after executing lines (27, 28, 29)

$t_3 = 0$, Line 30 is not satisfied

The program proceed, i.e no overflow.

2nd Case:

$\$t_1 = 0101$
$\$t_2 = 0111$
$\$t_0 = \$t_1 + \$t_2 = 1100$

$\$t_1 = 0101$
$\$t_2 = 0111$
$\$t_3 = \text{XOR}(\$t_1, \$t_2) = 0010$

After executing (lines, 23, 24), $\$t_3 = 0$

∴ Line 25 is not satisfied

but after executing line 27,

i.e. $\$t_0 = 1100$
$\$t_1 = 0101$
$\$t_3 = \text{XOR}(\$t_0, \$t_1) = 1001$

Thus after line 29, $\$t_3$ becomes 1.

∴ So, line 30 is satisfied (bne is satisfied)

i.e. Overflow is detected

Q2. IEEE 754 binary representation of floating point numbers is widely used in today's computer systems. Consider the decimal number, $(-0.3125)_{10}$, and represent it in a single precision IEEE 754 format. Show all the steps of your calculation.

$-0.3125 = -0.0101$

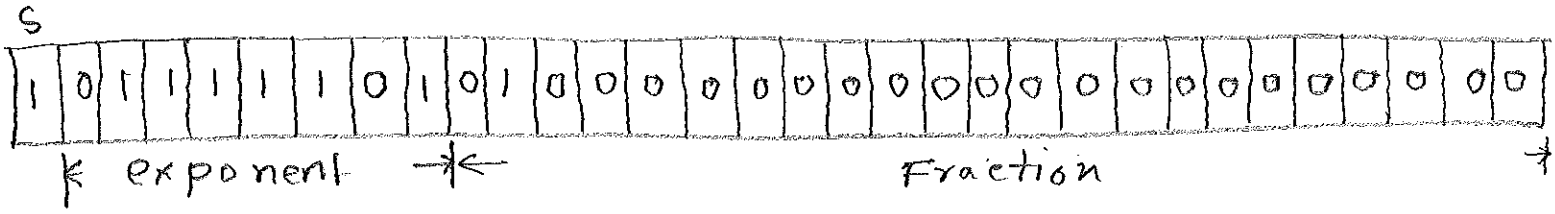
$-0.0101 \xrightarrow[\text{Normalized}]{} 1.01 \times 2^{(-2)}$

∴ the exponent (8 bits) of (-2) would be represented as

Exponent = bias − 2 = 127 − 2 = 125

exponent = 0 1 1 1 1 1 0 1

thus, The IEEE 754 representation is:



Q3. The following lines of codes which calculate the address (index) of a 4x4 matrix X[i][j], where each element is considered as a double precision floating point number (64 bits, i.e. 8 bytes). The code also loads the indexed element in the register pair $f4. Note that $s0 and $s1 represent the i and j indexes, respectively. Register $a0 contains the base address of X.

```
33  sll  $t2, $s0, 2    # $t2 = i * 4 (size of row of x)
34  addu $t2, $t2, $s1  # $t2 = i * size(row) + j
35  sll  $t2, $t2, 3    # $t2 = byte offset of [i][j]
36  addu $t2, $a0, $t2  # $t2 = byte address of x[i][j]
37  l.d  $f4, 0($t2)    # $f4 = 8 bytes of x[i][j]
```

What changes you would make in the code so that it can calculate the address (index) of the following 8x8 matrix Y[i][j], where each element in the matrix is a single precision floating point number (32 bits, i.e. 4 bytes). With the changed code and the initial values of $s0 = 3 and $s1 = 7, indicate which value of the matrix will be transferred to the register $f4. Show the details of your calculation.

```
17  Y: .float  9.5, 2.5, 3.5, 4.5, 9.5, 2.5, 3.5, 4.5,
18             5.5, 6.5, 7.5, 8.5, 5.5, 6.5, 7.5, 8.5,
19             1.5, 2.5, 3.5, 4.5, 1.5, 2.5, 3.5, 4.5,
20             5.5, 6.5, 7.5, 8.5, 5.5, 6.5, 7.5, 8.5,
21             9.5, 2.5, 3.5, 4.5, 9.5, 2.5, 3.5, 4.5,
22             5.5, 6.5, 7.5, 8.5, 5.5, 6.5, 7.5, 8.5,
23             1.5, 2.5, 3.5, 4.5, 1.5, 2.5, 3.5, 4.5,
24             5.5, 6.5, 7.5, 8.5, 5.5, 6.5, 7.5, 8.5
```

$f4

31-index

memory location = (124 + $a0)

Soln:

```
33.  sll   $t2, $s0, 3     # Row length is 8
34.  addu  $t2, $t2, $s1
35.  sll   $t2, $t2, 2     # 32 bit (4 byte)
36.  addu  $t2, $a0, $t2
37.  l.d   $f4, 0($t2)
```

$\Rightarrow$ $s_0 \rightarrow i$

$s_1 \rightarrow j$

Linear Index = $i \times 8 + j$

$= 3 \times 8 + 7 = 31$

$s_0 = 3 \rightarrow t_2 = 3 \times 8 = 24$

$s_1 = 7 \rightarrow t_2 = 24 + 7 = 31$

sll $t2, $t2, 2 $\Rightarrow$ $31 \times 4 = 124$

memory location = (124 + $a0)

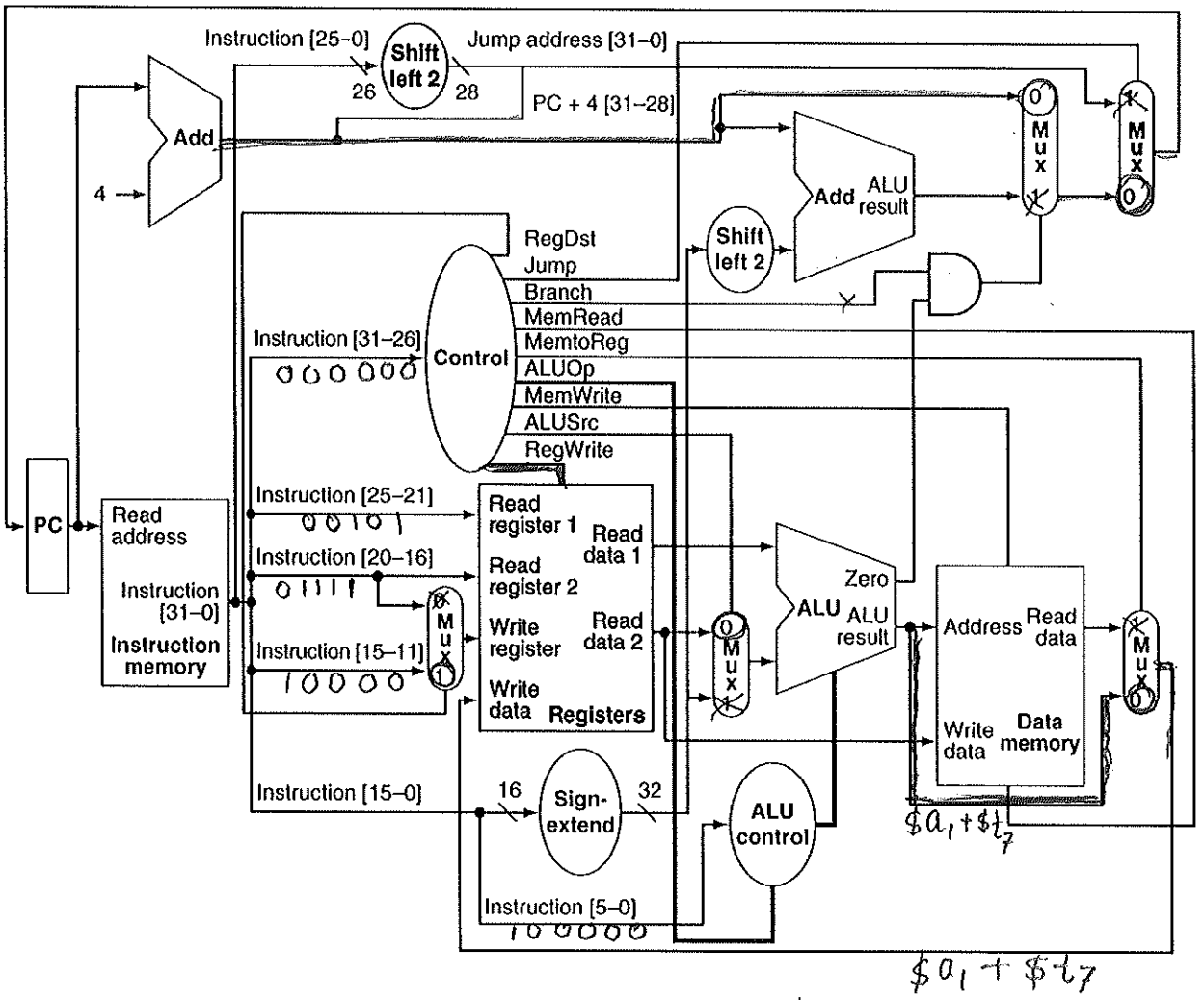
Q4. Consider the following R-type instruction, **add $s0, $a1, $t7**, and its corresponding machine code:

000000 (opcode) 00101 (Rs) 01111 (Rt) 10000 (Rd) 00000 100000 (Function)

The machine code consists of mainly Opcode, Source registers, destination registers, and Function code etc.

add Rd, Rs, Rt

Now suppose the instruction is executed in the MIPS processor as shown in the following figure. Highlight the different control signals and multiplexer inputs that will be activated during the execution process of the above **add** instruction. Also indicate the buses where different portions of the machine code and the result of $a1+$t7 can be found.



Q5. Pipelining is an implementation technique in which multiple instructions are overlapped in execution to improve the performance by increasing the instruction throughput of the system. However, there are situations in pipelining when the next instruction cannot execute in the following clock cycle. These events are called hazards. There are three different types of hazards, such as structural, data, and control hazards. Consider the following lines of codes and determine where you could have possible hazards. Also indicate the types of Hazard.

```
and $t2, $s1, $s2
sub $s4, $t2, $s3
or $s7, $s6, $t2
add $t7, $t2, $t5
beq $t2, $s8, exit
sw $s9, 100($t2)
```

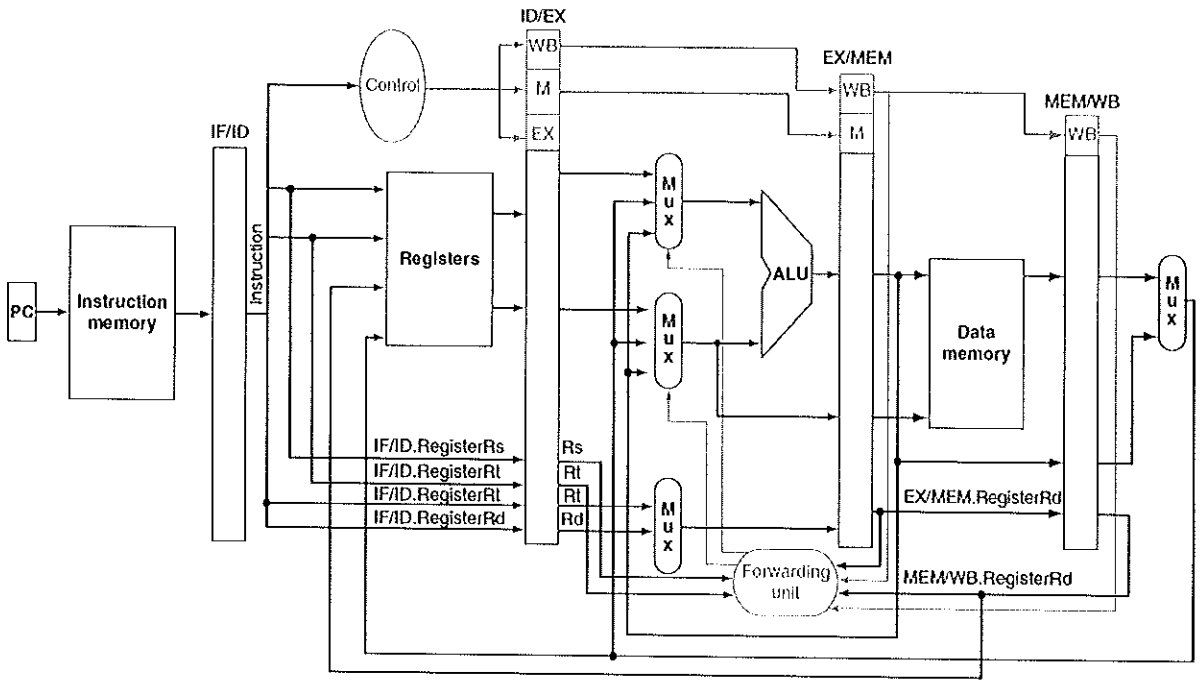
data Hazards

Control hazard

In MIPS, structural hazard is avoided by using two seperate memories.

To support the pipelining operation, MIPS processors use **pipeline-register,** in between the pipeline stages. There are four pipeline registers (IF/ID, ID/EX, EX/MEM and MEM/WB registers) in the MIPS architecture (as shown in the following figure). Pipeline registers temporarily hold the control signals and register numbers to facilitate the data forwarding or bypassing, an efficient way of overcoming the data hazards.

Write a short notes on how you can detect the above data hazards by comparing the pipeline registers (such as EX/MEM.RegisterRd, ID/EX.RegisterRs and ID/EX.RegisterRt registers) and using the pipeline-register control signals (such as EX/MEM.RegWrite, MEM/WB.RegWrite).



The data hazard in between 1st and 2nd Instructions can be detected by Comparing:

If (EX/MEM.RegWrite
and (EX/MEM.Register $R_d \neq 0$)
and (EX/MEM.Register R_d = ID/EX.Register R_s))

Instruction: and (t_2), S_1, S_2 ; sub S_4, (t_2), S_3

The data hazard in between 1st and 3rd instruction can be detected by Comparing:

If (MEM/WB.RegWrite
and (MEM/WB.Register $R_d \neq 0$)
and (MEM/WB.Register R_d = ID/EX.Register R_t)

Instraction: and (t_2), S_1, S_2 ; Or S_7, S_6, (t_2)