CS2100 TERM TEST #2 ANSWER SHEET *AY2011/2 Semester 2*

NAME:

/ 30

TOTAL SCORE

MATRIC. NO.:

U	0							
A	0	0						

TUTORIAL GROUP:

Explanations are given for some selected questions at the end of this form. Please remember to take a look!

SECTION A (2 marks each)

1.	2.	3.	4.	5.	6.
D	E	C	C	A	B

SECTION B (1 mark)

7. C. Correct association is "Aragorn is like the MUX(PCSrc)" [1]

SECTION C (18 marks)

8a. beq, sw [1]

8b. add, lw, sw [1]

8c. beq [1]

9a. 3069 (decimal) or 101 1111 1101 (binary) or 0x5FD (hexadecimal) [2]

9b. $s2 = $s0 * $s1 (multiplication) [2]

9c. C > 32*7 + 4 = 228 [2]

Please submit only this answer sheet.

Please turn over…

10a. [2]

```
if ( r5 == r1 )
    ID/EXE.O1 = EXE/MEM.RES
```

10b. [2]

```
if ( r8 == r1 )
    ID/EXE.O1 = MEM/WB.RES
```

10c. [2]

```
if ( r8 == r4 )
     ID/EXE.O1 = EXE/MEM.RES
else if ( r8 == r1 )
     ID/EXE.O1 = MEM/WB.RES
```

11a. [1]

```
1 / ( (0.5/3) + (0.5/5) )
```

11b. [2]

```
1/ ( (0.1/4) + (0.9/5) )
```

MCQ 1:

Original Ave. CPI 4.2 = 20%*5 + 80%*X

X is 4 (i.e. the combined contribution of the non-load instruction is 4.0)

New Ave.CPI

= 20%*4 (improved) + 80%*X

= 4.0

MCQ 4:

With no forwarding, the worst case scenario is that all instructions, except the first, is RAW dependent on the instruction before it. This causes a 2 cycle stalls for pipeline processor with no forwarding path.

Best case total cycles = 5 + 9 = **14** cycles

Worst case total cycles = 5 + 3*9 = **32** cycles (1^{st} instruction takes 5, the rest takes 3 cycles each to reach the WB stage).

MCQ5:

Option (i): When "**`j`**" happens to be the last instruction in a 256MB region, then all addresses that can be specified are before the **`j`** ➔ can only jump backward.

Option (iii): As the "**`j`**" and its target must have the same 4 MSBits, if the original "**`j`**" fails, so would all the "j" in the chain.

Q8c:

The calculation result is not used (utilized). Only "**`beq`**" has this behavior, as it uses only the "**`isZero`**" signal coming out from the ALU unit. All other instructions uses the result in some way (R-Format: to be written into destination register; **`lw`**/**`sw`**: used as memory address).

Q9:

Observe that the binary representation of a number X, say 1010_2 (10_{10}). Each occurrences of "1" is weighted, i.e.

$1010_2 = 1*2^3 + 1*2^1$

The code looks for "1" in $s0, then shift $s1 by the bit-position to the left, i.e. multiply $s1 by 2^b, where ***b*** is the position of the "1" in $s0. The shifted amount is then accumulated (summed) in $s2.

Let's use an example:

$s0 = $1010_2 = 1*2^3 + 1*2^1$

$s1 = 111_2

The code calculates $s2 as

$s2

= $(2^3 * \$s1) + (2^1 * \$s1)$

= $(1*2^3 + 1*2^1) * \$s1$

= $s0 * $s1

The code is the "shift-and-add" operations which transform multiplication into a series of shifting and addition.

Comment: This question tests your ability to build relationship between operation, rather than just the operations themselves. The "shift-and-add", although quite slow in the version shown, can be much faster than multiplication if we only include the instruction to process those bits with "1". You can imagine the code can be used by a compiler to generate code for the actual shift-and-add transformation.

Q11:

Amdahl's Law states that when you improve (speed up) a certain "component" in the system, the overall improvement is determined by the contribution (or proportion) of the improved component.

So, in general:

Time = (AffectedPortion/ SpeedupOfAffectedPortion) + UnaffectedPortion

When we look at this in the pipeline context:

- Affected Portion:
    - Instructions that can be executed by pipeline
- Speedup gained:
    - Potentially N (the number of stages in pipeline processor)

However, different type of instructions may see different amount of speedup in the pipeline. For example, instructions with no hazards can approaches N, while instructions with dependency (e.g. data dependency) may introduces additional stall => reduce the potential speedup.

So, the question introduces a way to approximate the speedup gained by different types of instruction.

Potential speedup = N

If instruction introduces X cycle of stall = (N-X) speedup.

Now, we can plug in the information into Amdahl's law.

Given an original set of instructions:

a. 50% has no data dependency (i.e. full speedup of N)

b. 50% has data dependency

- 20% of the 50% are memory load instruction

So, for (a):

No forwarding paths:

Number of stall = 2 cycles (need to delay the instruction until the writeback of the producer instruction).

This gives us:

Time = 0.5 / 5 + 0.5 / 3

This is the improved time compared to a non-pipeline processor. So, to find the speedup:

Speedup = Time(original) / Time (improved)

= 1 / (0.5/5 + 0.5/3)

You can use similar understanding to calculate for part (b).