CS2323 Computer Architecture 2019

Homework 3

Important: Your submission should be named as RollNumber_CA_HW3.pdf. For example, if your roll number is cs16mtech11075, then your submission should be cs16mtech11075_CA_HW3.pdf. Except pdf, no other format is acceptable. **10 marks will be deducted for not following these instructions or if you submit a zipped file.**

To reduce TA efforts, please answer the questions in the order in which they are given. There is no latex bonus.

===

1. (4 mark) Assume a neural predictor entry has the weight vector as [20, -14, 29, -12, 4] (bias weight is the last one) and branch history register as [1, -1, 1, 1]. Find its prediction. Assume that the outcome is taken, show updated weights and second prediction.

y = 4+(20+14+29-12) = 55 > 0 => Taken

New weight : [21, -15, 30, -11, 5]. y = 5 + 21+15+30-11 = 60 > 0 => Taken

2. (1 mark) Write a minimal sequence of instructions which shows WAR hazard.

add r3, r1, r2

add r1, r4, r5 [Anything where write(2nd instruction) is after reading (first inst).

3. (1 mark) Write a minimal sequence of instructions which shows control hazard.

beq .a

add r1, r2, r3

add r2, r3, r1… [Any two instructions after a branch - control hazard]

4. (3 marks) Consider the following instructions:

add r11, r11, 5

mul r2, r11, 90

add r8, r8, r2

sub r9, r9, 5

ld r0, 55(r9)

Assume the pipeline has only 3 stages: fetch, decode and execute (as assumed in the lecture on superscalar execution). Show the pipeline diagram assuming (1) simple pipelining (2) superscalar execution and (3) out-of-order execution.

[Table could be in other form, where 5 instructions would be in rows and *fetch, decode* and *execute* would be the entries.

Simple Pipeline

Clock	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Fetch	*1*	*2*	*3*	*4*	*5*		
Decode		*1*	*2*	*3*	*4*	*5*	
Execute			*1*	*2*	*3*	*4*	*5*

Super Scalar

Clock	**1**	**2**	**3**	**4**	**5**	**6**
Fetch	*1*	*2*	*3, 4*	*5*		
Decode		*1*	*2*	*3, 4*	*5*	
Execute			*1*	*2*	*3,4*	*5*

Out Of Order

Clock	**1**	**2**	**3**	**4**	**5**
Fetch	*1, 4*	*2, 5*	*3*		
Decode		*1, 4*	*2, 5*	*3*	
Execute			*1, 4*	*2, 5*	*3*

5. (2 marks) Consider the code below and answer the following two questions:

```
for ( int i=0; i<N; i++) { /* B1 */
    val = array [ i ] ;
    if ( val % 20 == 0) { /* B2 */
```

```
        sum += val ;
    }}
```

For branch B1 to be a biased branch, what should be the condition on the value of N and what should be the condition on values of array[]. If no condition is required, just say so.

For branch B2 to be an un-biased branch, what should be a condition on the value of N and and what should be the condition on values of array[]. If no condition is required, just say so.

For B1 to be biased, N should be large or N<=0. If someone has just written “N should be large”, that will fetch full marks. No condition on arr values.

For B2 to be unbiased, nearly half the values in N must be either divisible or indivisible by 20. No condition on N.

6. (2 marks) Represent 6.6979 in single-precision and double-precision floating point notation (you may use online tool for this). Also, find the difference between the number (i.e., 6.6979) and what is stored for both single and double precision-numbers. In which case, is the error (difference) smaller?

Single precision:

Error = 6.697899818420410015625 - 6.6979 = 1.82 * 10^{-7}

Double precision:

Error = 6.69789999999999974278352965484.. - 6.6979 ~ 2.58 * 10^{-16}

Smaller in double precision

7. (2 marks) Think of a recursive function to compute fibonnaci series in SimpleRISC. In your answer, you may either only write the changes in the code for fibonnaci series compared to that shown for factorial in L09, or you can write the whole new code.

```
.fib:
        cmp r0, 1
        beq .initR1_R2
        bgt .continue
        b .initR1_R2
```

```
.continue:
        sub sp, sp, 8
        st r0, [sp]
        st ra, 4[sp]
        sub r0, r0, 1
        call .fib
        ld r0, [sp]
        ld ra, 4[sp]
        mov r3,r2
        add r2,r2,r1
        mov r1,r3
        add sp, sp, 8
        ret
.initR1_R2:
        mov r1, 1
        mov r2, 1
        ret
.main:
        mov r0, 7 //number
        call .fib
```

8. (3 mark) Show the 32-bit SimpleRISC encoding of each of the following instructions:

* ret - $[10100]_5[00\ldots0]_{27}$

* call .factorial //address of factorial in binary is 110101, PC is 101

$[10011]_5[00..01100]_{27}$

* st ra, 4[sp]

[01111] [1] [1111] [1110] [00-0000000000000100]

* b .continue //address of this instruction (PC) is 001011 and address of

.continue is 01110011

$[10010]_5$ $[00...011010]_{27}$

* sub r1 r9 8

[00001] [1] [0001] [1001] [000000000000001000]

* lsl r5 r8 r9

[01010] [0] [0101] [1000] [1001] $[000...0]_{14}$

9. [2 mark] Show pipeline diagram for the following instructions assuming interlocking technique is used. You need not explain the diagram.

add r1, r2, r3

mul r7, r9, r10

sub r4, r1, r5

	1	2	3	4	5	6	7	8	9
IF	1	2	3						
OF		1	2	3	3	3			
EX			1	2	*(B)	*(B)	3		
MA				1	2	*(B)	*(B)	3	
RW					1	2	*(B)	*(B)	3

10. [1 mark] Write clearly why isLd is an input to the adder in ALU even though load instruction loads data from memory. [refer L10]

For load instruction, we need to compute the address of the memory location from where we need to read (load) the data. When address is specified in base-offset format (e.g., V ← [r1+offset]), then addition is happening and hence, we need to access the adder in the ALU.

11. [6 marks] Consider a branch which shows "TAKEN" outcome for five times consecutively and "NOT-TAKEN" twice consecutively. This pattern is repeated indefinitely, i.e., 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, and so on.

We use a two-level local history predictor where we remember the history of past three outcomes of this branch. Find out the accuracy of this branch predictor as p/q (where q=7). Ignore the accuracy for first five rounds (i.e., first 35 accesses) because that is the warmup period. (Hint: you may write a C/C++ code to find this but you don't have to upload the code. You can also answer without writing code, just by reasoning.)

There are two possible solutions:

Solution 1: If someone has not specified whether second-level predictor is 1 or 2 bit, then, we assume that it is 2 bit (as assumed in the slides). Then, the answer is as follows:

Accuracy is 6/7

Past 3 outcomes	Prediction	Actual
100	1	1
001	1	1
011	1	1
111	1	1
111	1	1
111	1	0
110	0	0

Solution 2:

If someone assumed that the second level predictor is 1-bit, then, accuracy is 5/7. The student must have clearly written that he/she is assuming that second level predictor is 1-bit

Past 3 outcomes	Prediction	Actual
100	1	1

001	1	1
011	1	1
111	0 (this is different from previous case)	1
111	1	1
111	1	0
110	0	0

12. Use the code uploaded on google-classroom. Write the value of [bias, weight1, weight2] after 1000 predictions for following cases and accuracy. (Do not use any seed for random number generation. Due to random number generation, the answer of different students may not match, which is OK.)

[This answer depends on the students random number generation. But, this is a fairly common one.]

(a) [1 mark]

n =rand();

if (n is divisible by 2?) //B1

if (n is divisible by 9?) //B2 (correlated branch)

if(n is divisible by 18?) we use neural branch predictor for this branch

[-896, 88, 884], Accuracy = 0.974

(b) [1 mark]

n =rand();

if (n is divisible by 2?) //B1

if (n is divisible by 7?) //B2 (uncorrelated branch)

if(n is divisible by 18?) we use neural branch predictor for this branch

[-896, 88, 656], Accuracy = 0.943

(c) [1 mark]

```
n =rand();
if (n is divisible by 2?) //B1
if (n >0) //B2 (highly biased branch)
if(n is divisible by 18?)  we use neural branch predictor for this branch
```

[-896, 88, -896], Accuracy = 0.947

(d) [1mark] We now remove the biased branch and find correlation with only one branch. You need to write only bias and weight1 for this case

```
n =rand();
if (n is divisible by 2?) //B1
if(n is divisible by 18?)  we use neural branch predictor for this branch
```

[-896, 88], Accuracy = 0.947