Computer architecture:

HW 3

Roll no: - CS18BTECH11039
NAME: - RAJ PATIL
Date: - 4/10/2019

Question 1

weight vector (w) : - [20, -14, 29, -12, 4], (last one for bias)

branch history register(bh) : - [1, -1, 1, 1] (1 for bias) i.e. [1, -1, 1, 1,1]

prediction = (w.bh)

20+14+29-12+4 = 55 i.e. taken

Given the prediction is correct (outcome is in fact taken):-

$w^{new} = w^{old} + bh = [20+1,-14-1,29+1,4+1] = [21,-15,30,-11,5]$

second prediction(assuming similar branch history) = w^{new}.bh =

21+15+30-11+5 = 60 i.e. taken

Question 2

The following sequence of instructions will exhibit a WAR hazard without control: -

cmp r0,r1

mov r0,r2

Question 3

The following sequence of instructions will exhibit a RAW hazard without control:-

beq. foo

add r0,r1,r2 /* unnecessarily fetched if equality flag is 1*/

.foo:

mov r0,3

add r0,r0,2

Question 4

[1]:add r11, r11, 5

[2]: mul r2, r11, 90

[3]:add r8, r8, r2

[4]:sub r9, r9, 5

[5]:ld r0, 55(r9)

Showing pipeline diagram for the above instructions In these cases

CASE 1 :- simple pipelining:-

Instruction							
[1] (add)	Fetch	Decode	**Add**				
[2] (mul)		Fetch	Decode	**Multiply**			
[3] (add)			Fetch	Decode	**Add**		
[4] (sub)				Fetch	Decode	**Subtract**	
[5] ld					Fetch	Decode	**Load**

CASE 2 :- superscalar execution:-

Instruction						
[1] (add)	Fetch	Decode	**Add**			
[2] (mul)		Fetch	Decode	**Multiply**		
[3] (add)			Fetch	Decode	**Add**	
[4] (sub)			Fetch	Decode	**Subtract**	
[5] ld				Fetch	Decode	**Load**

CASE 3:- out of order execution:-

Instruction					
[1] (add)	Fetch	Decode	**Add**		
[2] (mul)		Fetch	Decode	**Multiply**	
[3] (add)			Fetch	Decode	**Add**
[4] (sub)	Fetch	Decode	**Subtract**		
[5] ld		Fetch	Decode	**Load**	

Question 5

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

B1(biased)]

for B1 to be a biased branch (taken and not taken for unequal number of times), we consider the case when it is taken and not taken for equal number of times which happens only for N=1.

Hence for N<1 it's biased for not taken

And for N>1 it's biased for taken

And the array values do not affect the outcome of this branch(B1)

Hence, For B1 to be biased N != 1 and it's independent of the array values.

B2(unbiased)]

For B2 to be unbiased, val%20 should be 0 for half the time. So the values of array should be in such a way they have the following distribution:-

There is 50% probability that val will a multiple of 20;

And 50% probability that it won't.

There are infinite such distributions and the most basic one would be when the values are random integral multiples of 10.

Note that N being even or odd does not matter because using profiling, we'll be looping through the code multiple times and it only depends on the distribution of the values in the array.

Hence for B2 to be unbiased, values from the array should be distributed as stated above and it is independent of N.

Question 6

Format	Total bits	Exponent	Mantissa	bias
FP32	32	8	23	127
FP64	64	11	52	1023

Analyzing the storage of the number:- 6.6979 in these formats.

6.6979 = 4* (1 + 0.674475)

Hence unbiased exponent =$\log_2 4$ = 2;

exp - bias = 2;

actual mantissa = 0.674475

FP32 (single precision)

truncating mantissa to 23 bits:-

0.674475 = 0.10101100101010100110010

this in decimal is

0.10101100101010100110010 = 0.674474954605102539062 5

resulting number(stored) :-

(1 + 0.674474954605102539062 5)*4 = **6.69789981842041015625**

difference between actual and this:-

0.00000018157958984375 (actual will be larger because we're truncating the mantissa)

Representation

number representation:

sign:- 0
exp:- 10000001
mantissa:- 10101100101010100110010

0 10000001 10101100101010100110010

FP64 (Double precision)

truncating mantissa to 52 bits:-

0.674455 = 0.1010110010101010011001001100001011111000001101111011

this in decimal is

0.1010110010101010011001001100001011111000001101111011 =

0.674474999999999935695882413710933178663253784179687 5

resulting number(stored) :-

(1 + 0.674474999999999935695882413710933178663253784179687 5)*4=

6.697899999999999742783529654843732714653015136718 75

difference between original and this =

0.000000000000000257216470345156267285346984863281 25

Representation

number representation:-
sign:- 0
exp:- 10000000001
mantissa:- 1010110010101010011001001100001011111000001101111011

0 10000000001 1010110010101010011001001100001011111000001101111011

The difference is smaller in the second case(FP64) because we're truncating lesser number of bits from the actual binary representation of the mantissa

Question 7

Old code (Factorial)

```
.factorial:
        cmp r0, 1
        beq .initR1
        bgt .continue
        b .initR1

.continue:
        sub sp, sp, 8
        st r0, [sp]
        st ra, 4[sp]
        sub r0, r0, 1
        call .factorial
        ld r0, [sp]
        ld ra, 4[sp]
        mul r1, r0, r1
        add sp, sp, 8
        ret

.initR1:
        mov r1, 1
        ret

.main:
        mov r0, <param as imm>
        call .factorial
```

New code (Fibonacci)

```
.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, <param as imm>
        call .fib
```

Explanation of changes on next page

In the second code the registers are as follows:-

R0:- counter (similar to that of the first code)

R1:- stores the n-1th Fibonacci number

R2:- stores the nth Fibonacci number

R3:- a temporary buffer needed when updating R1 and R2

Sp :- the stack pointer

Note that the series corresponding to this code is 1,1,2,3,5… (indexes start from 0) i.e. f(0)=1 and f(1)=1 are the sequence's base definition.

R2 is where the nth Fibonacci number(the final result) is stored and R0 is a parameter of the function.

<param as imm> signifies the parameter inserted as an immediate.

Question 8

Showing simple 32-bit RISC encoding for the following instructions: -

A> **ret**

10100 <27 bits set to 0> =

10100 00000000000000000000000000

B> **call. factorial**

offset = (address of branch – PC)>>>2 = (110101 – 101)>>2 = 1100

10011 <23 bits set to 0> 1100 =

10011 00000000000000000000 1100

C> **st ra,4[sp] (is an exception)**

01111 1 1111 1110 00 <13 zeros> 100 =

01111 1 1111 1110 00 0000000000000 100

D> **b. continue**

offset = (address of branch – PC)>>>2 = (111001 - 1011)>>2 = 11010

10010 <22 bits set to 0> 11010 =

10010 00000000000000000000 11010

E> sub r1 r9 8

00001 1 0001 1001 00<12 bits set to 0>1000

00001 1 0001 1001 00000000000001000

F> Lsl r5 r8 r9

01010 0 0101 1000 1001 <remaining 14 bits set to 0>

01010 0 0101 1000 1001 00000000000000

Question 9

Showing the pipeline diagram for the following instructions

[1] add r1, r2, r3

[2] mul r7, r9, r10

[3] sub r4, r1, r5

Note :- a bubble is represented as {o} in the pipeline diagram

The five stages are as follows:-

IF :- instruction fetch
OF:-operand fetch
EX:-execution
MA:-memory access
RW:- register write

[not using forwarding]

Instruction stage									
IF	[1]	[2]	[3]						
OF		[1]	[2]	[3]	[3]	[3]			
EX			[1]	[2]	{o}	{o}	[3]		
MA				[1]	[2]	{o}	{o}	[3]	
RW					[1]	[2]	{o}	{o}	[3]

Stalling the OF stage of [3] for 2 cycles due to interdependence occurring due to r1(avoiding a RAW hazard)

Question 10

IsLd and IsSt are inputs to adder unit in the ALU because we compute branch offsets from immediates occurring in their instruction encodings.

Question 11

Given that we have to look at the past 3 local histories, we have 8 possible histories that can occur for a bidirectional branch but only 5 of them occur (101,010 and 000 never occur), hence keeping a tab of only these 5 histories when going through a cycle:-

(initialising the two bit saturation counters for all as 01(weakly taken))

Cycle 1

History	Saturated counter change
001	01→10
011	01→10
111	01→10→11
110	01
100	01

Next outcome	**0**	**0**	**1**	**1**	**1**	**1**	**1**
Prediction	-	-	-	0	0	0	1
Corresponding history	-	-	-	001	011	111	111
feedback	Not enough history	Not enough history	Not enough history	Incorrect	Incorrect	Incorrect	Correct
Change in saturation counter	-	-	-	+1	+1	+1	+1

Cycle 2

History	Saturated counter change
001	10→11
011	10→11
111	11→10→11
110	01→00
100	01→10

Next outcome	**0**	**0**	**1**	**1**	**1**	**1**	**1**
Prediction	1	0	0	1	1	1	1
Corresponding history	111	110	100	001	011	111	111
feedback	Incorrect	Correct	Incorrect	Correct	Correct	Correct	Correct
Change in saturation counter	-1	-1	+1	+1	+1	+1	-

Cycle 3

History	Saturated counter change
001	11
011	11
111	11→10→11
110	00
100	10→1

Next outcome	**0**	**0**	**1**	**1**	**1**	**1**	**1**
Prediction	1	0	1	1	1	1	1
Corresponding history	111	110	100	001	011	111	111
feedback	Incorrect	Correct	Correct	correct	Correct	Correct	Correct
Change in saturation counter	-1	-	+1	-	-	+1	-

Note that after this cycle, the number of correct and incorrect predictions stay the same and only the saturated counter corresponding to the history 111 switches twice in a cycle (11 to 10 to 11 again) and every thing else stays the same(history log of 111 highlighted in yellow)

These many cycles are enough for us and hence the number of times we get a correct prediction out of 7 is 6 times.

Accuracy = 6/7

Question 12

For 1000 inputs.

The weights ordered as corresponding to [bias ,input 1, input 2] are as follows:-

a) Input 1 = modulo 2==0
   input 2 = modulo 9==0 (correlated branch)
   outcome monitored = modulo 18==0
   weight vector = [-896,88,884]

b) input 1 = modulo 2==0
   input 2 = modulo 7==0 (uncorrelated branch)
   outcome monitored = modulo 18==0
   weight vector = [-896,88,656]

c) input 1 = modulo 2==0
   input 2 = isPositive() (highly biased)
   output monitored = modulo 18==0
   weight vector = [-896,88,-896]

d) input 1 = modulo 2==0
   input 2 = not taken (equivalent to being set as 1 always (as in above))
   output monitored = modulo 18==0
   weight vector = [-896,88] (corresponding to bias and input 1)