**EE660: Computer Architecture**

**Spring, 2017**

Assignment No. 3 (Due: April 18, 2017)

Score: /100

*Student: Hualiang Li hualiang@hawaii.edu Date: April 10, 2017*

**31 (20/100).** Assume a computer architecture with a 1024-entry Branch History Table (BHT). Each BHT entry is a two bit predictor that implements a two-bit saturating counter that starts in the strong not-taken state. After the following code execution, what is the state of the BHT? What is the prediction accuracy of each branch?

p 4 :

// Assume that at address 0 x1000 , the re i s an array o f word s i z e d i n t e g e r s

// with the f o l l o w i n g data [ 0 x0 , 0 x0 , 0 x0 , 0 x0 , 0 x0 , 0 x1 , 0 x0 , 0 x1 ] ADDI R7 , R0 , 7

ADDI R11 , 0 x1000

loop :

SUBI R7 , R7 , 1 ANDI R8 , R7 , 1

b 0 :

BEQZ R8 , l 1 LW R12 , 0 ( R11 )

b 1 :

BEQZ R12 , l 2

l 1 :

ADD R9 , R9 , R16

l 2 :

ADDI R11 , R11 , 4

b 3 :

BNEZ R7 , loop

Iteration b\_0 state b\_1 state b\_3 state

1 Taken NT SNT Taken NT

2 Ntaken SNT Taken NT Taken T

3 Taken NT NT Taken ST

4 Ntaken SNT Taken T Taken ST

5 Taken NT T Taken ST

6 Ntaken SNT Ntaken NT Taken ST

7 Taken NT NT Ntaken T

For b\_0, the accuracy is 42.9%. For b\_1, the accuracy is 0%. For b\_3, the accuracy is 57.1%

**32 (20/100).** [[PH11],](#_bookmark0) page 137, problem 2.9-a,b,c,d.

1. From 2.8 c, the memory access time is 1.69ns for 4-way 64KB. The direct mapped cache access time is 0.86ns.

0.0033\*20+(0.8\*2+(1-0.8)\*3)\*(1-0.0033) = 2.3 cycles. So, 2.3\*0.5 = 1.15ns

b. 0.83/0.5 = 1.66. So 66% faster.

c. 0.0033\*20+(0.8\*2+(1-0.8)\*15)\*(1-0.0033) = 4.65 cycles. So 4.65\*0.5 = 2.33ns

d. 2.4/1.59 = 1.509. So 50.9% faster.

**33 (20/100).** You are executing on a VMIPS (as described on page 266 of [[PH11])](#_bookmark0) architecture the code below. Assume that the maximum vector length of the architecture is 128. Draw the pipeline diagram of this code executing on a single-lane architecture which has an independent load unit, store unit, multiply unit, and ALU unit. Loads have a latency of three cycles (L0, L1, L2), stores take two cycles to occur (S0, S1), multiplies take 5 cycles (Y0, Y1, Y2, Y3, Y4), and ALU operations take two cycles (X0, X1). Assume full pipelining of the functional units. Assume that the pipeline has a dedicated register read stage and a single write-stage. For the first part of this problem, assume that the architecture support chaining through the register file, but only has two read ports and one write port on the register file.

C Code :

f o r ( i = 0 ; i *<* 6 ; i ++)

*{*

a [ i ] = ( b [ i ] *∗* 7 ) + c [ i ] ;

*}*

3.1

*Assignment No. 3 (Due: April 18, 2017)* 3.2

VMIPS:

ADDI R7 , R0 , 7

CVT.W.D F2 , R7 ADDI R1 , R0 , 6

MTC1 VLR, R1 LV V1 , R4

MULVS.D V3 , V1 , F2 LV V2 , R5 ADD V4 , V3 , V2 SV R6 , V4

LV F D R L0 L1 L2 W

R L0 L1 L2 W

R L0 L1 L2 W

R L0 L1 L2 W

R L0 L1 L2 W

R L0 L1 L2 W

MULVS.D F D D D D D D R Y0 Y1 Y2 Y3 Y4 W

R Y0 Y1 Y2 Y3 Y4 W

R Y0 Y1 Y2 Y3 Y4 W

R Y0 Y1 Y2 Y3 Y4 W

R Y0 Y1 Y2 Y3 Y4 W

R Y0 Y1 Y2 Y3 Y4 W

LV F F F F F F D D D D D D D D R L0 L1 L2 W

R L0 L1 L2 W

R L0 L1 L2 W

R L0 L1 L2 W

R L0 L1 L2 W

R L0 L1 L2 W

ADD F F F F F F F F D D D D D D D R X0 X1 W

R X0 X1 W

R X0 X1 W

R X0 X1 W

R X0 X1 W

R X0 X1 W

SV F F F F F F F D D D D D D R S0 S1 W

R S0 S1 W

R S0 S1 W

R S0 S1 W

R S0 S1 W

R S0 S1 W

**34 (20/100).** [[PH11],](#_bookmark0) page 254, 3.13.

Processor A:

Time

1 T1 LD R1 T2 LD R1

2 T3 LD R1 T4 LD R1

3 T1 LD R2 T2 LD R2

4 T3 LD R2 T4 LD R2

.

.

.

16 T3 LD R8 T4 LD R8

17 T1 BEQ T2 BEQ

.

.

.

48 T3 BEQ T3 BEQ

49 T1 DADDIU T2 DADDIU

50 T3 DADDIU T4 DADDIU

51 T1 BCT T2 BLT

.

.

.

54

So, for processor A, 54\* 2 = 108cycles

Processor B:

Time

1 T1 LDR1 LDR2 LDR3 LDR4

2 T1 LDR5 LDR6 LDR7 LDR8

3 T1 BEQ

4 T2 LDR1 LDR2 LDR3 LDR4

5 T2 LDR5 LDR6 LDR7 LDR8

6 T2 BEQ

.

.

.

10 T3 LDR1 LDR2 LDR3 LDR4

11 T3 LDR5 LDR6 LDR7 LDR8

12 T3 BEQ

13 T1 BEQ

14 T2 BEQ

15 T3 BEQ

16 T4 BEQ

.

.

.

37 T1 BEQ

38 T2 BEQ

39 T3 BEQ

40 T4 BEQ

41 T1 DADDIU

42 T2 DADDIU

43 T3 DADDIU

44 T4 DADDIU

45 T1 BLT

46 T2 BLT

47 T3 BLT

48 T4 BLT

So, for processor B, 48\*2 = 96 cycles

Processor C

Time

1 T1 LDR1 LDR2 LDR3 LDR4 LDR5 LDR6 LDR7 LDR8

2 T2 LDR1 LDR2 LDR3 LDR4 LDR5 LDR6 LDR7 LDR8

3 T3 LDR1 LDR2 LDR3 LDR4 LDR5 LDR6 LDR7 LDR8

4 T4 LDR1 LDR2 LDR3 LDR4 LDR5 LDR6 LDR7 LDR8

5 T1 BEQ

6 T2 BEQ

7 T3 BEQ

8 T4 BEQ

.

.

.

36 T4 BEQ

37 T1 DADDIU

38 T1 BLT

39 T2 BLT

40 T3 BLT

41 T4 BLT

So, for processor c, 41\*2)+1 = 83 cycles, 83\*4 = 332 cycles.

**35 (20/100).** You are writing a multi-threaded program that will count the number of occurrences of a value in an array. The values in the array are between 0 and 1023. In effect, you will be building a histogram. Assume that the list of numbers is very large, on the order of gigabytes large. Extend the following program such that 100 threads (processors) can execute on the program concurrently. Assume a sequentially consistent memory model. Add P() and V() semaphores where appropriate and add any storage needed for the semaphores. Explain why the speedup of such a solution may not be 100x. Note that the output lock array is assumed to be initialized to 1 (this allows for a mutex).

// S e q ue nti al code , assume that the input and output arrays are c re ate d

// o uts i de o f the f unc t i o n

#d e f i n e MAX VALUE 1023

f unc t i o n ( i n t i n p u t a r r a y s i z e , i n t *∗* input array , i n t *∗* output array )

*{*

i n t counter ;

f o r ( counter = 0 ; counter *<* i n p u t a r r a y s i z e ; counter++)

*{*

a s s e r t ( i nput array [ counter ] *<*= MAX VALUE) ; a s s e r t ( i nput array [ counter ] *>*= 0 ) ;

output array [ i nput array [ counter ] ]++;

*} }*

*#include <thread.h>*

*#include <stdlib.h>*

*#define MAX 1023*

*#define MIN 0*

*#define NUM\_THREADS 100*

*struct data{*

*int intput\_size;*

*int\* input\_array;*

*int\* output\_array;*

*int\* output\_lock\_array;*

*}*

*struct thread\_data thread\_data\_array[NUM\_THREADS];*

*void function(void \* thread\_args)*

*{*

*struct thread\_data \* temp\_data;*

*temp\_data = (struct thread\_data \*) thread\_args;*

*int input\_array\_size = temp\_data‐>input\_size;*

*int \* input\_array = temp\_data->input\_array;*

*int \* output\_array = temp\_data->output\_array;*

*int \* output\_lock\_array = temp\_data->output\_lock\_array;*

*int counter;*

*for(counter = 0; counter < input\_array\_size; counter++)*

*{*

*assert(input\_array[counter] <= MAX);*

*assert(input\_array[counter] >= MIN);*

*P(&output\_lock\_array[counter]);*

*output\_array[input\_array[counter]]++;*

*V(&output\_lock\_array[counter]);*

*}*

*pthread\_exit();*

*}*

*int main() {*

*int counter;*

*pthread\_t threads[NUM\_THREADS];*

*int \* input\_array;*

*input\_array = malloc(INPUT\_ARRAY\_ELEMENTS\*sizeof(int));*

*int \* output\_array = malloc((MAX + 1) \* sizeof(int));*

*int \* output\_lock\_array = malloc((MAX + 1) \* sizeof(int));*

*for(counter = 0; counter <= MAX; counter++);*

*{*

*output\_array = 0;*

*output\_lock\_array = 1;*

*}*

*for(counter = 0; counter < 100; counter++)*

*{*

*thread\_data\_array[counter].input\_array\_size = INPUT\_ARRAY\_ELEMENTS /*

*NUM\_THREADS;*

*thread\_data\_array[counter].input\_array =*

*&input\_array[INPUT\_ARRAY\_ELEMENTS / NUM\_THREADS \* counter];*

*thread\_data\_array[counter].output\_array = output\_array;*

*thread\_data\_array[counter].output\_lock\_array = output\_lock\_array;*

*pthread\_create(&threads[counter], NULL, function, (void*

*\*)thread\_data\_array[counter]);*

*}*

*for(counter = 0; counter < 100; counter++)*

*{*

*pthread\_join(threads[counter], NULL);*

*}*

*}*

*No. It will be less than 100 faster. This is because of 1) the threads overhead, 2) lock overhead 3) cach miss overhead.*

**References**

[PH11] David A. Patterson and John L. Hennessy. *Computer Architecture: A Quantitative Approach*. 5th ed. Morgan Kaufmann Publishers Inc., 2011. isbn: 978-0-12-383872-8. url: [http://booksite.](http://booksite.elsevier.com/9780123838728) [elsevier.com/9780123838728](http://booksite.elsevier.com/9780123838728).