Advanced Computer Architecture

Homework Assignment 2

Submitted by: Maliha Arif
PID: 4506817
Submitted to: Dr. Dan Marinescu
Submitted on: 9th October 2018

Question 3.15:

[20/20] <3.4, 3.5, 3.7, 3.8> In this exercise, we will look at how variations on Tomasulo's algorithm perform when running the loop from Exercise 3.14.

a. **[20] <3.4–3.5> For this problem use the single-issue Tomasulo MIPS pipeline of Figure 3.6 with the pipeline latencies from the table above. Show the number of stall cycles for each instruction and what clock cycle each instruction begins execution (i.e., enters its first EX cycle) for three iterations of the loop. How many cycles does each loop iteration take?**

Iteration	Instruction	Issues	Memory Access/ Executes	Writes in CDB	Comment
1	fld F2,0(x1)	1	2	3	Waiting for 1st issue command
1	fmul.d F4,F2,F0	2	4	4+15=19	-Waits for load/F2 -In Reservation station, gets F2 at [3], begins execution at 4th clock cycle -Multiply takes 15 clock cycles [5-18]
1	fld F6,0(x2)	3	4	5	-waits for issue command
1	fadd.d F6,F4,F6	4	20	20+10=30	-Waits for F6 to load [5-6] -Waits for F4 RS [5-20] -Addition 10, Add busy [21-29]

1	fsd F6,0(x2)	5	31		-Waits for add to finish RS [6-31]
1	addi x1,x1,8	6	7	8	X1 is available and integer operation takes 1 clock cycle
1	addi x2,x2,8	7	8	9	
1	sltu x3,x1,x4	8	9	10	-checks if loop has to continue
1	bnez x3,foo	9	11		-Waits for sltu to check if loop is needed -waits for x3 register
2	fld F2,0(x1)	10	12	13	-waits for issue to occur and execute for 1 clock cycle before broadcasting in CDB -Executes at12, waits for bnez
2	fmul.d F4,F2,F0	11	19	19+15=34	-FP multiplier unit is busy completing 1st multiplication -RS [12-19] -Uses functional unit [20-33]
2	fld F6,0(x2)	12	13	14	-waits for issue then execute before broadcasting

2	fadd.d F6,F4,F6	13	35	35+10=45	-Waits for fmul to complete to give F6 in RS[14-34] -F6 available -Add use [36-44]
2	fsd F6,0(x2)	14	46		-Waits for add to execute before storing to memory
2	addi x1,x1,8	15	16	17	
2	addi x2,x2,8	16	17	18	
2	sltu x3,x1,x4	17	18	19	-should loop be entered, question?
2	bnez x3,foo	18	20		-nothing to broadcast -waits for sltu command
3	fld F2,0(x1)	19	21	22	-loads after issue and bnez command
3	fmul.d F4,F2,F0	20	34	34+15=49	-FP multiplier busy -waits RS[21-34] -Busy multiply[34-48]
3	fld F6,0(x2)	21	22	23	-Issue load
3	fadd.d F6,F4,F6	22	50	50+10=60	-Waits for fmul to complete -RS[23-49] -Add use[51-59]
3	fsd F6,0(x2)	23	61		-Stores in F6
3	addi x1,x1,8	24	25	26	

3	addi x2,x2,8	25	26	27	
3	sltu x3,x1,x4	26	27	28	
3	bnez x3,foo	27	29		-Waits for sltu

3.15 b

[20] <3.7, 3.8> Repeat part (a) but this time assume a two-issue Tomasulo algorithm and a fully pipelined floating-point unit (FPU).

Iteration	**Instruction**	**Issues**	**Memory Access/ Executes**	**Writes in CDB**	**Comment**
1	fld F2,0(x1)	1	2	3	Waiting for 1st issue command then writes in bus
1	fmul.d F4,F2,F0	1	4	4+15=19	-Waits for F2 -Multiply RS [2:4] -Multiply use at [5] till [5-18]
1	fld F6,0(x2)	2	3	4	-loading takes 1 clock cycle
1	fadd.d F6,F4,F6	2	20	20+10=30	-Gets F6 -waits for F4 -Add rs[3-19] Add busy [21] or Add busy[21-29] pipelined so
1	fsd F6,0(x2)	3	31		-Waits for F6 before saving -stbuf [4-30]
1	addi x1,x1,8	3	4	5	X1 is available and

					integer operation takes 1 clock cycle -immediate hence no wait
1	addi x2,x2,8	4	5	6	
1	sltu x3,x1,x4	4	6	7	-gets x1, compares -INTEGER unit busy -INT rs[5-6]
1	bnez x3,foo	5	7		-INT is busy hence INT rs[6-7]
2	fld F2,0(x1)	6	8	9	Waits for bnez to finish execution
2	fmul.d F4,F2,F0	6	10	10+15=25	-waits for F2 value -multiply rs[7-10] multiply use[11-24] (occurs in parallel)
2	fld F6,0(x2)	7	9	10	-waits for INT -INT busy -INT rs[8-9]
2	fadd.d F6,F4,F6	7	26	26+10=36	-waits for F4 -Add RS[8-25] -Add busy [26-35] or [27] (pipelined , parallel)
2	fsd F6,0(x2)	8	37		-Waits for R6

2	addi x1,x1,8	8	10	11	-INT was busy -INT rs[10-11]
2	addi x2,x2,8	9	11	12	
2	sltu x3,x1,x4	9	12	13	-INT busy -INT rs[10-12]
2	bnez x3,foo	10	14		-wait for x3
3	fld F2,0(x1)	11	15	26	-waits for bnez
3	fmul.d F4,F2,F0	11	17	17+15=32	-waits for F2 -Multiply rs[12-16] Busy [17]
3	fld F6,0(x2)	12	16	17	-INT busy rs[13-16]
3	fadd.d F6,F4,F6	12	33	33+10=43	-waits for F4 -Add rs[12-33] -Add use[33]
3	fsd F6,0(x2)	14	44		-wait for F6 -INT has 5 rs, full now
3	addi x1,x1,8	15	17		-INT unit is busy and full now - INT rs[17]
3	addi x2,x2,8	16	18		-INT unit is busy and full now - INT rs[18]
3	sltu x3,x1,x4	20	21		-INT unit or INT rs is full
3	bnez x3,foo	21			-INT unit or INT rs is still full

Question 3.16:

[10] <3.4> Tomasulo's algorithm has a disadvantage: Only one result can compute per clock per CDB. Use the hardware configuration and latencies from the previous question and find a code sequence of no more than 10 instructions where Tomasulo's algorithm must stall due to CDB contention. Indicate where this occurs in your sequence.

Solution:

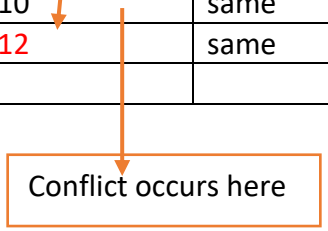
I have 2 solutions to this problem, one using single issue Tomasalo's algorithm and one using 2 issue Tomasulo's algorithm

2-way

	Instruction	Issued at	Access	CDB
1.	fld F2,0(x1)	1	2	3
2.	Addi x1,x1,8	1	2	3

1-way

	Instruction	Issued at	Access	CDB	Comments
1.	fadd.d F3,F2,F3	1	2	2+10= 12	Add instruction takes 10 cycles to execute
2.	addi x3,x4,x3	2	3	4	Integer operation takes 1 cycle to complete
3.	addi x3,x4,x3	3	5	6	Same
4.	addi x3,x4,x3	4	7	8	same
5.	addi x3,x4,x3	5	9	10	same
6.	addi x3,x4,x3	6	11	12	same



Questions 3.17 :

[20] <3.3> An (m,n) correlating branch predictor uses the behavior of the most recent m executed branches to choose from 2 m predictors, each of which is an nbit predictor. A two-level local predictor works in a similar fashion, but only keeps track of the past behavior of each individual branch to predict future behavior.

There is a design trade-off involved with such predictors: Correlating predictors require little memory for history which allows them to maintain 2-bit predictors for a large number of individual branches (reducing the probability of branch instructions reusing the same predictor), while local predictors require substantially more memory to keep history and are thus limited to tracking a relatively small number of branch instructions. For this exercise, consider a (1,2) correlating predictor that can track four branches (requiring 16 bits) versus a (1,2) local predictor that can track two branches using the same amount of memory. For the following branch outcomes, provide each prediction, the table entry used to make the prediction, any updates to the table as a result of the prediction, and the final misprediction rate of each predictor. Assume that all branches up to this point have been taken.

Correlating Predictor:

Here we take mod as 4 and we should know that modulo of address gives us the branch That it will take. We use the value to find branch in the predictor table and assume last outcome as taken as given in the question. We look at the prediction column and see if it matches with the outcome as in the address table given in the question (3rd one).

Example:

454 mod 4= 2
543 mod 4 =3
777 mod 4= 1

for the first one, the branch is 2 hence in the predictor table, we will look at branch column, we see taken, not taken. We will look at the entry with taken option as the question tells us to assume that. The prediction corresponding is our prediction.

As in this case, its 2 and final prediction is taken.

Correlating predictor			
Entry	**Branch**	**Last outcome**	**Prediction**
0	0	T	T with one misprediction
1	0	NT	NT
2	1	T	NT
3	1	NT	T
4	2	T	T
5	2	NT	T

Table with all values calculated as explained above:

Branch PC (word address)	Outcome	Prediction	Table entry used to make the prediction	Updates to Predictor table
454	T	T	4	No change
543	NT	NT	6	Change to NT
777	NT	NT	2	No change
543	NT	NT	7	No change
777	NT	T	3	Change to T with one misprediction
454	T	T	4	No change
777	NT	T	3	Change to NT
454	T	T	4	No change
543	T	NT	7	Change to T with one misprediction

Misprediction rate:

We observe a total of 3 mispredictions out of a total of 9 ,
hence misprediction rate = 3/9 = 0.33

Local Predictor:

Branch PC (word address)	Outcome	Prediction	Table entry used to make the prediction	Updates to Predictor table
454	T	T	0	Change to T
543	NT	T	4	Change to T with one misprediction
777	NT	NT	1	No change
543	NT	T	3	Change to T with one misprediction
777	NT	T	3	Change to NT
454	T	T	0	No change
777	NT	NT	3	No change
454	T	T	0	No change
543	T	T	5	Change to T

Misprediction rate:

We observe a total of 3 mispredictions out of a total of 9 ,
hence misprediction rate = 3/9 = 0.33

Question 3.18:

[10] <3.9> Suppose we have a deeply pipelined processor, for which we implement a branch-target buffer for the conditional branches only. Assume that the misprediction penalty is always four cycles and the buffer miss penalty is always three cycles. Assume a 90% hit rate, 90% accuracy, and 15% branch frequency. How much faster is the processor with the branch-target buffer versus a processor that has a fixed two-cycle branch penalty? Assume a base clock cycle per instruction (CPI) without branch stalls of one.

In order to calculate the speed up, we need to calculate the CPI (clock cycles per instruction) with and without Branch target buffer (BTB)

Speed up would be the difference or ratio of with BTB and without BTB, written as follows:

Speed up= CPI without BTB
.. CPI with BTB

CPI without BTB:
stalls = 15% branch frequency X fixed 2 cycle branch penalty = 0.3

CPI without BTB = CPI (base) + stalls = 1 + .15 *2. = 1 + 0.3 = 1.3

CPI with BTB:

BTB result	Frequency per instruction	Penalty
Buffer miss	15% X 10%=1.5 %	3
Hit – not correct, as a miss	15% × 90% × 10% = 1.3%	4
Hit – correct	15% × 90% × 90% = 12.1%	0

Stalls = (1.5%*3) + (1.3% 4) +(12.1%0) =0.097

CPI with BTB= CPI (base) + stalls=1 +0.097 = 1.097

Speed up= CPI without BTB. = 1.3 =. 1.185 ~ 1.2 is the speed up
..CPI with BTB 1.097