Computer Architecture Homework 4

Vishalini Laguduva Ramnath

U29903994

1. To compute the basic performance,

Code	Instruction/cycle
Loop: fld f2,0(Rx)	1+3
I0: fmul.d f2,f0,f2	1+4
I1: fdiv.d f8,f2,f0	1+10
I2: fld f4,0(Ry)	1+3
I3: fadd.d f4,f0,f4	1+2
I4: fadd.d f10,f8,f2	1+2
I5: fsd f4,0(Ry)	1+1
I6: addi Rx,Rx,8	1
I7: addi Ry,Ry,8	1
I8: sub x20,x4,Rx	1
I9: bnz x20,Loop	1+1
Cycles per loop iteration	**37**

2. Calculating the cycles per loop iteration with true loop dependency,
   With the true dependency, until the output is ready, no dependent instruction will execute. So here the Load instruction stalls for 3 cycles and so forth.
   The true dependency is seen between, register f2, f8 ,f4.
   The total cycle per loop iteration with the true dependency is 27
   The table explained with the necessary stalls,

Code	Instruction/cycle	Cycle count
Loop: fld f2,0(Rx)	1+3	1
Stall		2
Stall		3
Stall		4
I0: fmul.d f2,f0,f2	1+4	5
Stall on fmuld		6
Stall on fmuld		7
Stall on fmuld		8
Stall on fmuld		9
I1: fdiv.d f8,f2,f0	1+10	10
I2: fld f4,0(Ry)	1+3	11
Stall due to LD		12
Stall due to LD		13
Stall due to LD		14
I3: fadd.d f4,f0,f4	1+2	15
Stall due to fadd.d		16
Stall due to fdiv.d		17
Stall due to fdiv.d		18
Stall due to fdiv.d		19
Stall due to fdiv.d		20
I4: fadd.d f10,f8,f2	1+2	21
I5: fsd f4,0(Ry)	1+1	22
I6: addi Rx,Rx,8	1	23
I7: addi Ry,Ry,8	1	24
I8: sub x20,x4,Rx	1	25
I9: bnz x20, Loop	1+1	26
Stall due to branch delay		27
Cycles per loop iteration	27	

3. The above code should be reordered with preserving the true dependency. For simplicity we can reorder the instructions with no dependencies,

Ins/Stall	Clock cycle
fld f2,(Rx)	1
fld f4,0(Ry)	2
addi Rx,Rx,8	3
sub x20,x4,Rx	4
fmul.d f2,f0,f2	5
fadd.d f4,f0,f4	6
Stall	7
Stall	8
Stall	9
fdiv.d f8,f2,f0	10
fsd f4,0(Ry)	11
Stall	12
Stall	13
Stall	14
Stall	15
Stall	16
Stall	17
Stall	18
Stall	19
fadd.d f10,f8,f2	20
bnz x20,Loop	21
Addi Ry,Ry,8	22

The total number of cycles with reordering is 22.

Alterative metod:

```
Loop: fld f2,0(Rx)
fld f4,0(Ry)
<Stall>
<Stall>
fmul.d f2,f0,f2
fadd.d f4,f0,f4
addi Rx,Rx,8
<Stall>
fsd f4,0(Ry)
fdiv.d f8,f2,f0
sub x20,x4,Rx
<Stall on div(9)>
fadd.d f10,f8,f2
bnz x20, Loop
addi Ry, Ry,8
```

With Reordering the reduced number of cycles per loop iteration is 23 cycles.

There are multiple ways of doing the reordering the instructions to reduce the clock cycle. The lowest number of clock cycles required with reordering is 22 cycles.

4. Loop unrolling:
   With loop unrolling based on the reordered code for two iterations, assume that we can use register renaming to reduce the number of instructions without any further dependencies.

```
Loop: fld f2,0(Rx)
fld f4,0(Ry)
fld f11,0(Rx)
fld f12,0(Ry)
addi Rx,Rx,8
fmuld f2,f0,f2
fmuld f11,f0,f11
fadd.d f4,f0,f4
sub x20,x4,Rx
fdivd f8,f2,f0
fdivd f13,f2,f0
<stall>
fsd f4,0(Ry)
<stall>
<Stall>
<Stall>
<stall>
<stall>
<stall>
<stall>
fadd f10,f8,f2
bnz x20,loop
addi Ry,Ry,8
```

With Loop unrolling there are 23 cycles in total.

The total number of cycles with loop unrolling is **23 cycles.**
To calculate the speed up,
Cycles without unrolling: 22
Cycles with unrolling: 23 with 2 iterations = 23/2 = 11.5
Speedup = 22/11.5 = 1.91

Alternative method:

```
Loop: fld f2,0(Rx)
fld f4,0(Ry)
fld f11,0(Rx)
fld f12,0(Ry)
addi Rx,Rx,8
fmul.d f2,f0,f2
fmul.d f11,f0,f11
fadd.d f4,f0,f4
sub x20,x4,Rx
fsd f4,0(Ry)
fdiv.d f8,f2,f0
fdiv.d f13,f11,f0
Stall on div(8)
fadd.d f10,f8,f2
bnz x20, Loop
addi Ry, Ry,8
```

5. As given in the question, the instruction will not enter inter into the execution stage until all the operands are ready.

All ops are 1 cycle, sw and lw takes 1+2cycles while branch takes 1+1 cycle.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Lw x1,0(x2)	F	D	E	M	-	-	W													
addi x1,x1,1		F	D	-	-	-	E	M	W											
sw x1,0,(x2)			F	-	-	-	D	-	E	M	-	-	W							
addi x2,x2,4							F	-	D	E	-	-	M	W						
sub x4,x3,x2									F	D	-	-	-	E	M	W				
bnez x4,Loop										F	-	-	-	D	-	E	-	M	W	

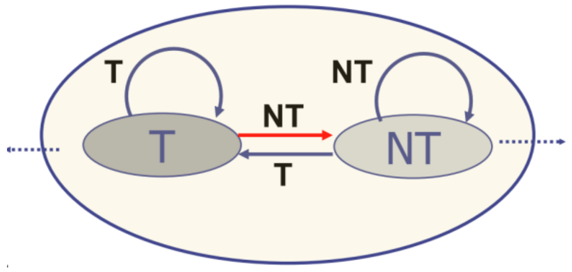
a. The **4 cycles lost to branch overhead**. Here the overhead is for instruction addi,sub and bnez. So the branch will decide at the execute stage so the overhead is between 13 to 17.

b. With the static predictor **2 cycles are lost to branch overhead**, that is the branch is predicted at the decode stage itself. So the overhead is between 13 and 15.

c. With dynamic predictor, **no cycle is lost to the branch overhead**. A dynamic branch predictor remembers that when the branch instruction was fetched in the past, it eventually turned out to be a branch, and this branch was taken. So a "predicted taken" will occur in the same cycle as the branch is fetched, and the next fetch after that will be to the presumed target.

6. The following series of branch outcomes occurs for a single branch in a program. (T means the branch is taken, N means the branch is not taken).

a. 1-bit predictor:
With the help of the 1-bit predictor from the slide, Start the prediction with N.
The mispredictions are given in red.

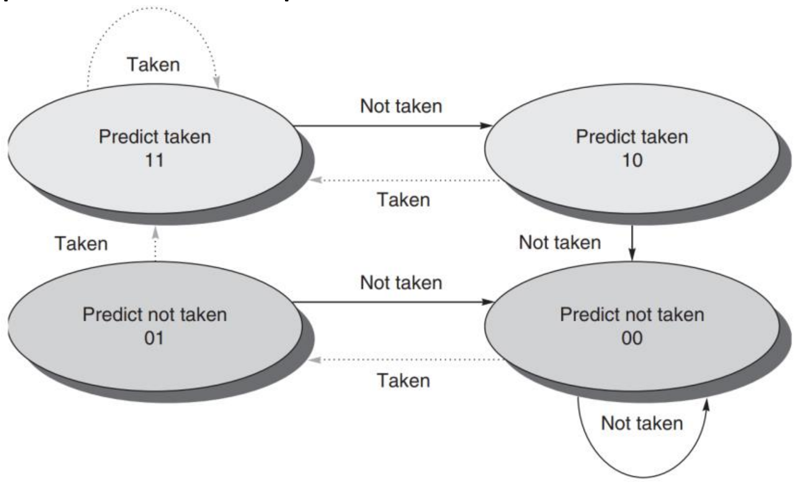


Index	1	2	3	4	5	6	7	8	9	10	11	12	13
Given	T	T	N	T	N	T	T	T	T	N	T	T	N
Predicted Next state	N	T	T	N	T	N	T	T	T	T	N	T	T

There are 8 mispredictions in total.

b. 2-bit predictor:

With the help of the 2-bit predictor from the slide, we can predict the, mis predictions.



Given, start the predictions with 10, 10(N) and 11(T).

The mis predictions are shown in red.

index	predicted	Given outcome	Prediction Result
1	10 (T)	T	Hit
2	11 (T)	T	Hit
3	11 (T)	N	Miss
4	10 (T)	T	Hit
5	11 (T)	N	Miss
6	10 (T)	T	Hit
7	11 (T)	T	Hit
8	11 (T)	T	Hit
9	11 (T)	T	Hit
10	11 (T)	N	Miss
11	10 (T)	T	Hit
12	11 (T)	T	Hit
13	11 (T)	N	Miss

The total mispredictions are 3,5,10,13.