CE/CZ3001 Advance Computer Architecture Project

Submitted by: **Adrian Goh Jun Wei**
David Loh Shun Hao

Matriculation Number: **U1721134D**
U1823250B

School of Computer Science and Engineering

2020

Contents

Adrian Goh Jun Wei (U1721134D)
David Loh Shun Hao (U1823250B)

Question 1

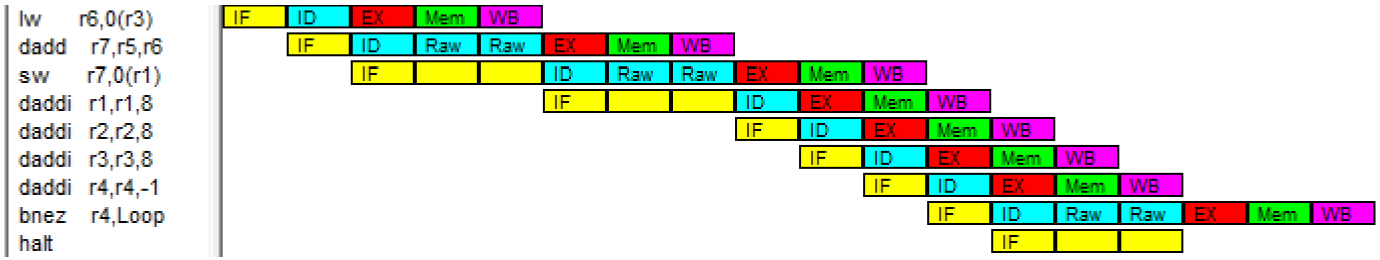
Prediction

	1	2	3	4	5	6	7	8
daddi r1,r0,a	IF	ID	EX	Mem	WB			
daddi r2,r0,b		IF	ID	EX	Mem	WB		
daddi r3,r0,c			IF	ID	EX	Mem	WB	
daddi r4,r0,6				IF	ID	EX	Mem	WB

	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
lw r4,0(r2)	IF	ID	EX	Mem	WB															
lw r6,0(r3)		IF	ID	EX	Mem	WB														
dadd r7,r5,r6			IF	Raw	Raw	ID	EX	Mem	WB											
sw r7,0(r1)						IF	Raw	Raw	ID	EX	Mem	WB								
daddi r1,r1,8									IF	Raw	ID	EX	Mem	WB						
daddi r2,r2,8											IF	ID	EX	Mem	WB					
daddi r3,r3,8												IF	ID	EX	Mem	WB				
daddi r4,r4,-1													IF	ID	EX	Mem	WB			
bnez r4,Loop														IF	Raw	Raw	ID	EX	Mem	WB

Simulation results

This is the first cycle of the simulation



lw r6,0(r3)	IF	ID	EX	Mem	WB													
dadd r7,r5,r6		IF	ID	Raw	Raw	EX	Mem	WB										
sw r7,0(r1)			IF			ID	Raw	Raw	EX	Mem	WB							
daddi r1,r1,8						IF			ID	EX	Mem	WB						
daddi r2,r2,8									IF	ID	EX	Mem	WB					
daddi r3,r3,8										IF	ID	EX	Mem	WB				
daddi r4,r4,-1											IF	ID	EX	Mem	WB			
bnez r4,Loop												IF	ID	Raw	Raw	EX	Mem	WB
halt													IF					

Figure 1

Adrian Goh Jun Wei (U1721134D)
David Loh Shun Hao (U1823250B)

The subsequent cycles have a repetitive pattern, thus only one copy is shown here. Note that instruction [lw (r5, 0(r,2)] starts off at the same cycle as the execution stage of instruction [bnez r4, Loop].

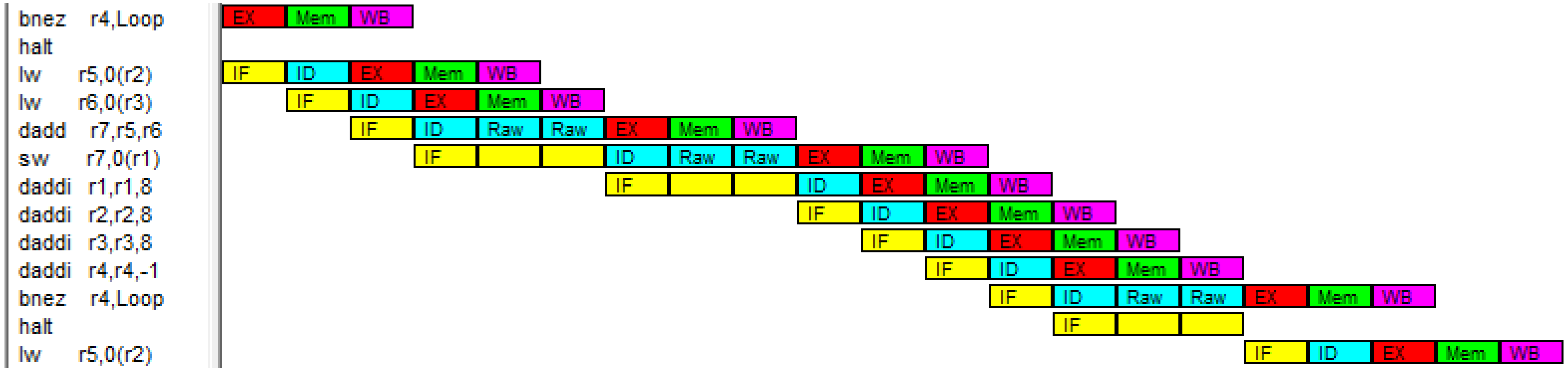


Figure 2

Cause of hazard

Instructions	Cause of hazards
dadd r7, r5, r6	This type of hazard is known as data hazard. The DADD [dadd r7, r5, r6] instruction is used for performing simple addition of binary data in byte, word and doubleword size. In this case, r5 and r6 is the source register which will be used to derive the arithmetic addition. In the previous instruction [lw r6, 0(r3)], LW loads data from the data memory through a specified address, with a possible offset, to the destination register. In this case, r6 is the destination register where data will be written into. Therefore, EX (execution) of DADD can only begin after r6 is updated with the result from the previous LW instruction, which will happen after WB (writeback).
sw r7, 0(r1)	This type of hazard is known as data hazard. The SW instruction stores data to a specified address on the data memory with a possible offset, from a source register. In this case, r7 is the source register address.

	In the previous instruction [dadd r7, r5, r6], DADD instruction is used for performing simple addition of binary data in byte, word and doubleword size. In this case, r7 is the destination register address where the arithmetic result will be written to. Therefore, EX (execution) of SW can only begin after r7 is updated with the result from the previous DADD instruction, which will happen after WB (writeback).
bnez r4, Loop	This type of hazard is known as data hazard. The BNEZ instruction [bnez r4, Loop] branch to Loop if the content of r4 is not equal to 0. In the previous instruction [daddi r4, r4, -1], DADDI instruction is used for performing simple addition of binary data in byte, word and doubleword size. In this case, r4 is both the source and destination register. Therefore, the EX (execution) of BNEZ can only begin after r4 is updated with the result from the previous DADDI instruction, which will happen after WB (writeback).
lw r5, 0(r2)	This type of hazard is known as control hazard. In the previous instruction [bnez r4, Loop], the program control will branch to Loop if the content of r4 is not equal to 0. In the previous HALT instruction, CPU operation is suspended until an interrupt or reset is received. Therefore, the need to wait for BNEZ to finish computation and decide on the branching resulted in the control hazard.

Question 2

CPI estimation

From the Statistics window, we know the following:

Execution	• 104 cycles • 59 Instructions • 1.763 cycles per instruction (CPI)
Stalls	• 36 RAW stalls • 5 Branch taken stalls

Steady-state CPI

To calculate the steady-state CPI using the hotspot method, we will use the last 4 loops.

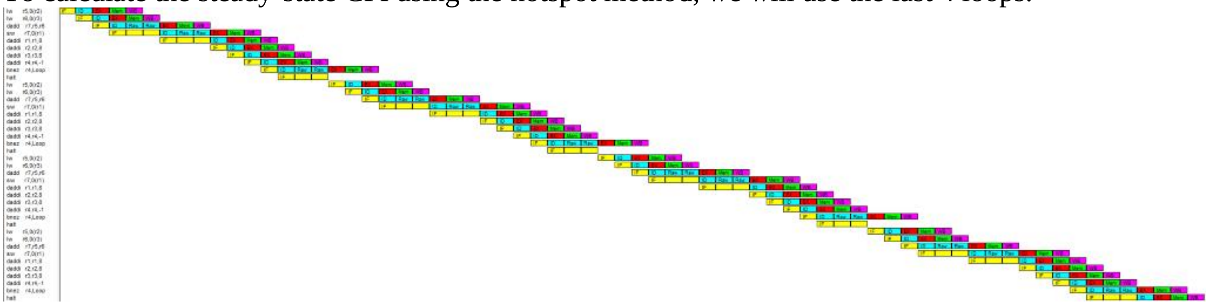


Figure 3

$$\text{Steady-State CPI} = \frac{40 \text{ instructions} + 24 \text{ stall}}{40 \text{ instructions}}$$
$$= 1.6$$

The steady state CPI is 1.6 while the simulation CPI is 1.763. Thus, the steady state CPI is lower.

Why steady-state CPI could be applied

The steady-state CPI could be applied because in practice, the processor will run a lot more instructions than the one used to calculate the CPI during simulation. The number of stalls will then be normalised and "distributed" more evenly across the different set of instructions, instead of being "amplified" and having a larger impact on the CPI during simulation.

Question 3

CPI calculation

To calculate the new steady-state CPI using the hotspot method, we will use the last 4 loops too.

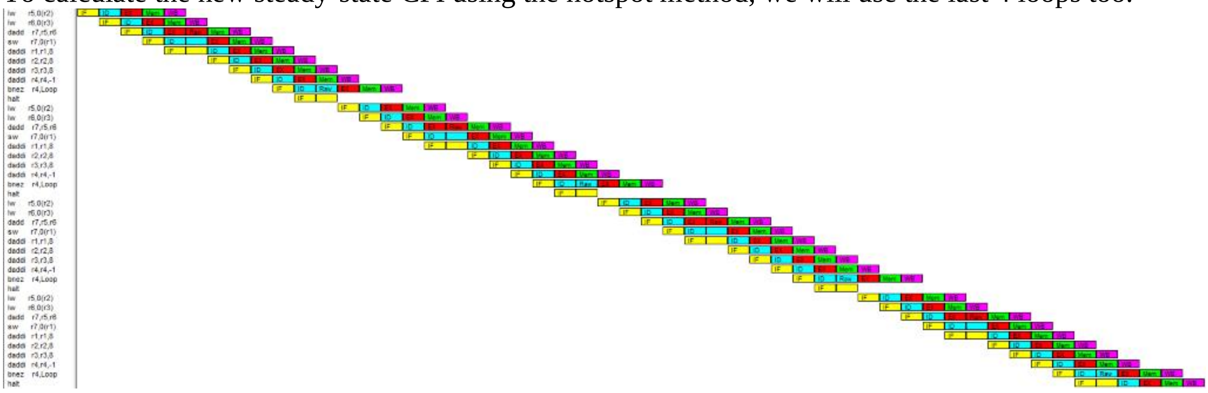
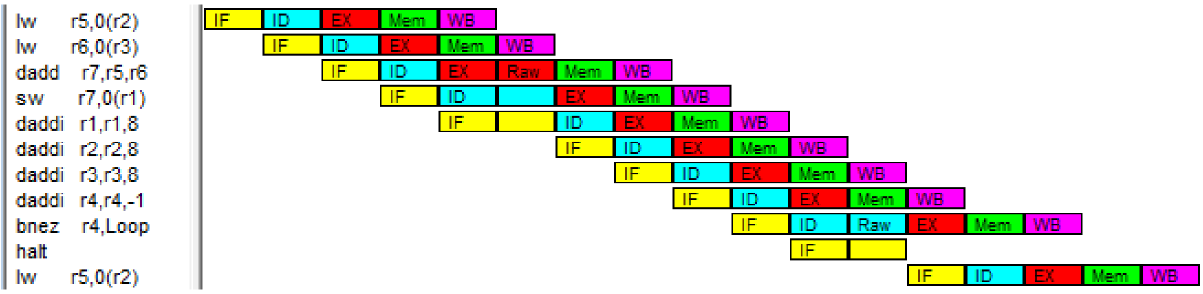


Figure 4

$$\text{Steady-state CPI} = \frac{40 \text{ instructions} + 8 \text{ stalls}}{40 \text{ instructions}}$$
$$= 1.2$$

Analysing remaining pipeline stalls



Data hazards still exist, but instead of stalling for 2 cycles each time, the stall is 1 cycle instead. This is because data forwarding is allowed, thus results from WB (writeback) can be passed to EX (execution) of the affected instruction in the same cycle.

Question 4

Performance analysis of NOP stuffing technique

To calculate the steady-state CPI using the hotspot method, we will use the last 3 loops.

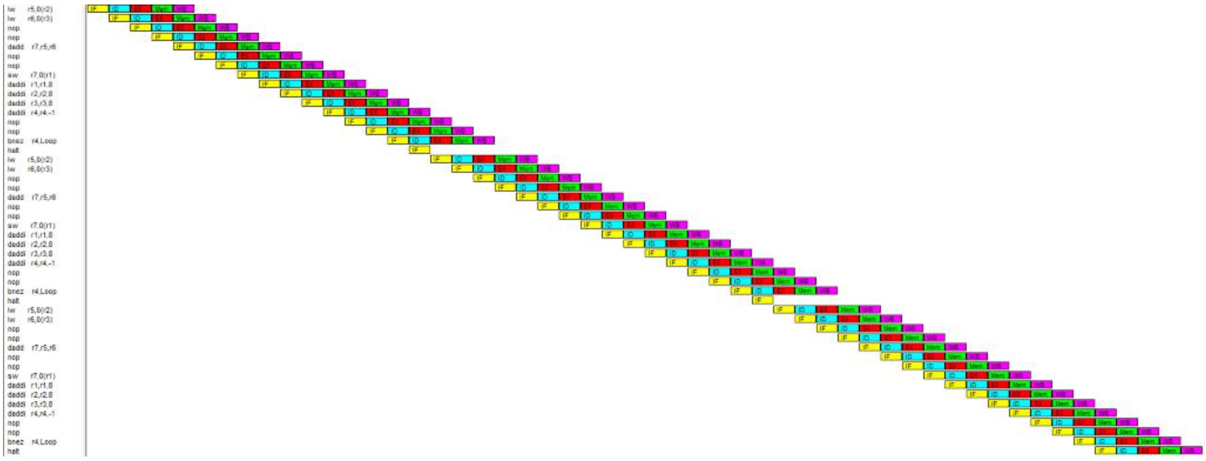


Figure 5

$$\text{Steady-State CPI} = \frac{48 \text{ instructions} + 0 \text{ stalls}}{48 \text{ instructions}}$$
$$= 1$$

The new CPI is 1, which is much lower than the previous 1.7.
Even though the CPI is lower, the performance remains the same. This is because if NOP (software stall) was not manually inserted, the processor does hardware stalls (as seen in Question 2). Therefore, the time taken to execute both set of code is the same

Question 5

Performance analysis of code moving technique

Data before code moving technique applied

```
0000  000000000000000b a: .space 48
0008  000000000000000d
0010  000000000000000f
0018  0000000000000011
0020  0000000000000005
0028  0000000000000007
0030  000000000000000a b: .word 10,11,12,13,0,1
0038  000000000000000b
0040  000000000000000c
0048  000000000000000d
0050  0000000000000000
0058  0000000000000001
0060  0000000000000001 c: .word 1,2,3,4,5,6
0068  0000000000000002
0070  0000000000000003
0078  0000000000000004
0080  0000000000000005
0088  0000000000000006
```

Figure 6

Data after code moving technique applied

```
0000  000000000000000b a: .space 48
0008  000000000000000d
0010  000000000000000f
0018  0000000000000011
0020  0000000000000005
0028  0000000000000007
0030  000000000000000a b: .word 10,11,12,13,0,1
0038  000000000000000b
0040  000000000000000c
0048  000000000000000d
0050  0000000000000000
0058  0000000000000001
0060  0000000000000001 c: .word 1,2,3,4,5,6
0068  0000000000000002
0070  0000000000000003
0078  0000000000000004
0080  0000000000000005
0088  0000000000000006
```

Figure 7

The simulation result is the same after the instructions reordering.

CPI calculation

To calculate the steady-state CPI using the hotspot method, we will use the last 4 loops.

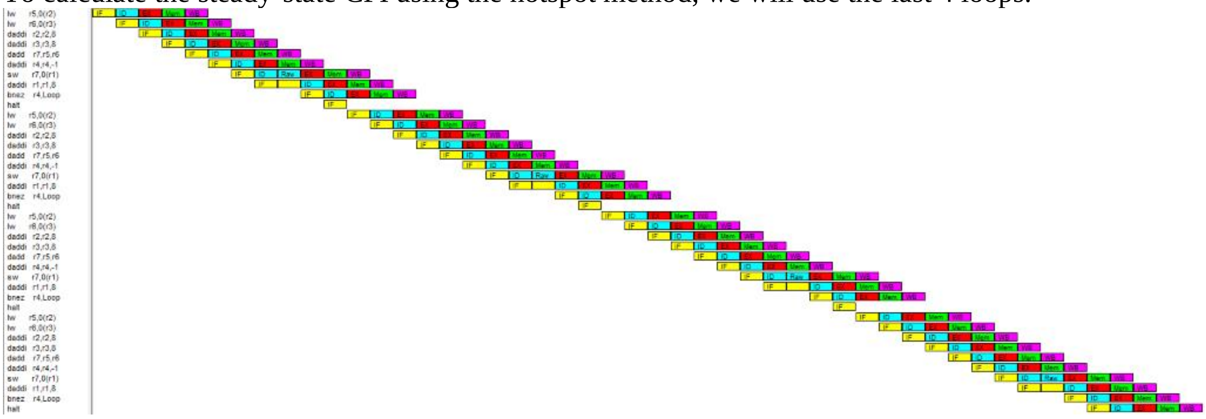


Figure 8

$$\text{Steady-state CPI} = \frac{40 \text{ instructions} + 4 \text{ stalls}}{40 \text{ instructions}}$$
$$= 1.1$$

Question 6

CPI calculation

To calculate the steady-state CPI using the hotspot method, we will use the last 4 loops.

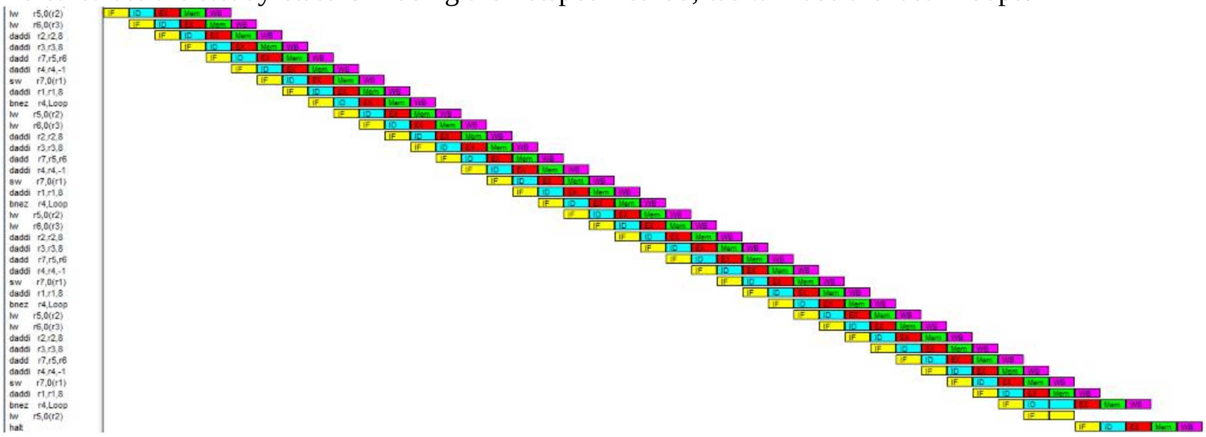


Figure 9

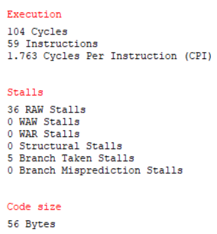
$$\text{Steady-state CPI} = \frac{38 \text{ instructions} + 1 \text{ stall}}{38 \text{ instructions}}$$
$$= 1.026$$

Performance analysis

With forwarding, the CPI is 1.2; With instruction rescheduling, the CPI is 1.
With both in place, the CPI is reduced to 1.026. Performance has been improved as the instruction "HALT" is negligible between the loops.

Question 7

With six additional inputs the CPI is expected to increase as more instructions will be executed.

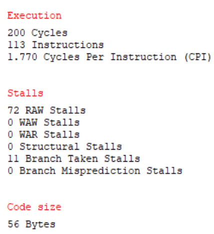


Execution
104 Cycles
59 Instructions
1.763 Cycles Per Instruction (CPI)

Stalls
36 RAW Stalls
0 WAW Stalls
0 WAR Stalls
0 Structural Stalls
5 Branch Taken Stalls
0 Branch Misprediction Stalls

Code size
56 Bytes

Figure 10



Execution
200 Cycles
113 Instructions
1.770 Cycles Per Instruction (CPI)

Stalls
72 RAW Stalls
0 WAW Stalls
0 WAR Stalls
0 Structural Stalls
11 Branch Taken Stalls
0 Branch Misprediction Stalls

Code size
56 Bytes

Figure 11

Original normal CPI =1.773

$$\text{Original steady-state CPI} = \frac{59\ + 41}{59}$$
$$= 1.695$$

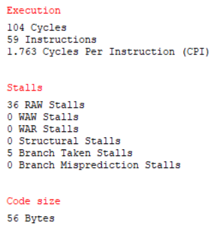
New normal CPI = 1.770

$$\text{New steady-stead CPI} = \frac{113 + 83}{113}$$
$$= 1.735$$

The addition of six input values doubled the number of instructions within the loop and the number of stalls cycles also increased, therefore the new steady-state CPI has also gone up from 1.695 to 1.735.

Question 8

$$\text{Steady-state CPI (no forwarding)} = \frac{56 \text{ instructions} + 35 \text{ stalls}}{56 \text{ instructions}}$$
$$= 1.625$$



Execution
104 Cycles
59 Instructions
1.763 Cycles Per Instruction (CPI)

Stalls
36 RAW Stalls
0 WAW Stalls
0 WAR Stalls
0 Structural Stalls
5 Branch Taken Stalls
0 Branch Misprediction Stalls

Code size
56 Bytes

Figure 12

As shown in figure 12, by unrolling the loop by a factor of 2, the number of RAW stall cycles reduced by half and the number of branch taken stall cycles decreased from 11 to 5. This results in the improvement in steady-state CPI, reducing it from 1.735 to 1.625.

Question 9

$$\text{Steady-state CPI (no forwarding)} = \frac{53 \text{ instructions} + 8 \text{ stalls}}{53 \text{ instructions}}$$
$$= 1.151$$

With instruction rescheduling applied there is a better performance with the CPI decreasing from 1.625 to 1.151.

Question 10

Acknowledgement

Student Name: Adrian Goh Jun Wei **Student's Signature:**

Student Name: David Loh Shun Hao **Student's Signature:**