

Computer Architecture II Assignment 3

Gerard Moylan 21364007

Q1:

I:

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10
LW R1, 0(R2)	IF	ID	EX	MEM	WB					
SUB R1, R3, R2		IF	ID	EX	MEM	WB				
SUB R3, R1, R2			IF	ID	EX	MEM	WB			
ADD R2, R1, R2				IF	ID	EX	MEM	WB		
SUB R1, R1, R2					IF	ID	EX	MEM	WB	
ADD R3, R1, R2						IF	ID	EX	MEM	WB

Where:

The value for R1 calculated in EX in CC4 is forwarded to EX in CC5 and EX in CC6.

The value for R2 calculated in EX in CC6 is forwarded to EX in CC7 and CC8.

The value for R1 calculated in EX in CC7 is forwarded to EX in CC8.

It would take 10 clock cycles as shown in the table above.

There would be no stalls. This is because after the first instruction where $R1 = \text{MemContent}[R2]$, R1 is immediately overwritten to be R3-R2.

Lines 3 and 4 which use R1 as a parameter get this R1 value from line 2 via forwarding, and by the time any subsequent lines need R1, the correct value for R1 is already finished being written into memory by line 2.

To calculate the final values of R1, R2 and R3, we can just calculate the code line by line.

Doing it out manually, showing the values of each register after each step:

$R1 = \text{MemContent}[R2] \quad \rightarrow R1=2, R2=3, R3=1$
 $R1 = R3 - R2 \quad \rightarrow R1=-2, R2=3, R3=1$
 $R3 = R1 - R2 \quad \rightarrow R1=-2, R2=3, R3=-5$
 $R2 = R1 + R2 \quad \rightarrow R1=-2, R2=1, R3=-5$
 $R1 = R1 - R2 \quad \rightarrow R1=-3, R2=1, R3=-5$
 $R3 = R1 + R2 \quad \rightarrow R1=-3, R2=1, R3=-2$

Final answer: $R1 = -3, R2 = 1, R3 = -2$

II:

No, the number of clock cycles cannot be optimised by employing out-of-order instruction scheduling because there are no stalls.

III:

It would now take 13 clock cycles.

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10	CC11	CC12	CC13
LW R1, 0(R2)	IF	ID	EX	MEM	WB								
SUB R1, R3, R2		IF	ID	EX	MEM	WB							
SUB R3, R1, R2			IF	ID	STALL	EX	MEM	WB					
ADD R2, R1, R2				IF	ID	STALL	EX	MEM	WB				
SUB R1, R1, R2					IF	ID	STALL	STALL	EX	MEM	WB		
ADD R3, R1, R2						IF	ID	STALL	STALL	STALL	EX	MEM	WB

The stall on line 3 is due to the fact that it is waiting for line 2 to finish writing the answer to R1.

The stall on line 4 is because it can't perform EX at the same time as EX on line 3.

The stalls on line 5 are because it is waiting for line 4 to finish writing the value for R2

The stalls on line 6 are because it is waiting for line 5 to finish writing the value for R1.

The final values for R1, R2 and R3 would be the same as part 1. The whole reason it is stalling is to ensure that it is taking the correct values for registers AFTER they are finished being written.

The reason there is no stall when IF and MEM are on the same cycle is because it is assumed there are caches for data and instructions.

IV:

In this case the pipeline table would look exactly the same as for part I except without any forwarding whatsoever. So the number of clock cycles would be 10 but the values of R1, R2, and R3 would be different.

The execution in pseudocode would be as follows

$R1 = \text{MemContent}[R2]$

$R1 = R3 - R2 \quad (1 - 3 = -2)$

$R3 = R1 - R2 \quad (2 - 3 = -1)$

$R2 = R1 + R2 \quad (-2 + 3 = 1)$

$R1 = R1 - R2 \quad (-2 - 3 = -5)$

$R3 = R1 + R2 \quad (-2 + 1 = -1)$

So:

$R1 = -5, R2 = 1, R3 = -1$

Where the brackets correspond to the values that are being used for each operand respectively, and the result.

For line 3, R1 is being taken as 2 because that is the value written on line 1.

For line 5, R2 is taken as 3 because the operation on line 4 has not yet been written.

For line 6, R1 is taken as -2 because the operation on line 5 has not yet been written.

Task 2:

I:

For the 1st configuration, since the offset is 12 bits long the pages are $2^{12} = 4\text{kB}$

Similarly, the 2nd configuration is $2^{13} = 8\text{kB}$

II:

The following assumes each table entry is 4 bytes.

Configuration 1:

1st level: $(2^8) * 4 = 1\text{kB}$

2nd level: $(2^7) * 4 = 512\text{B}$

3rd level: $(2^5) * 4 = 128\text{B}$

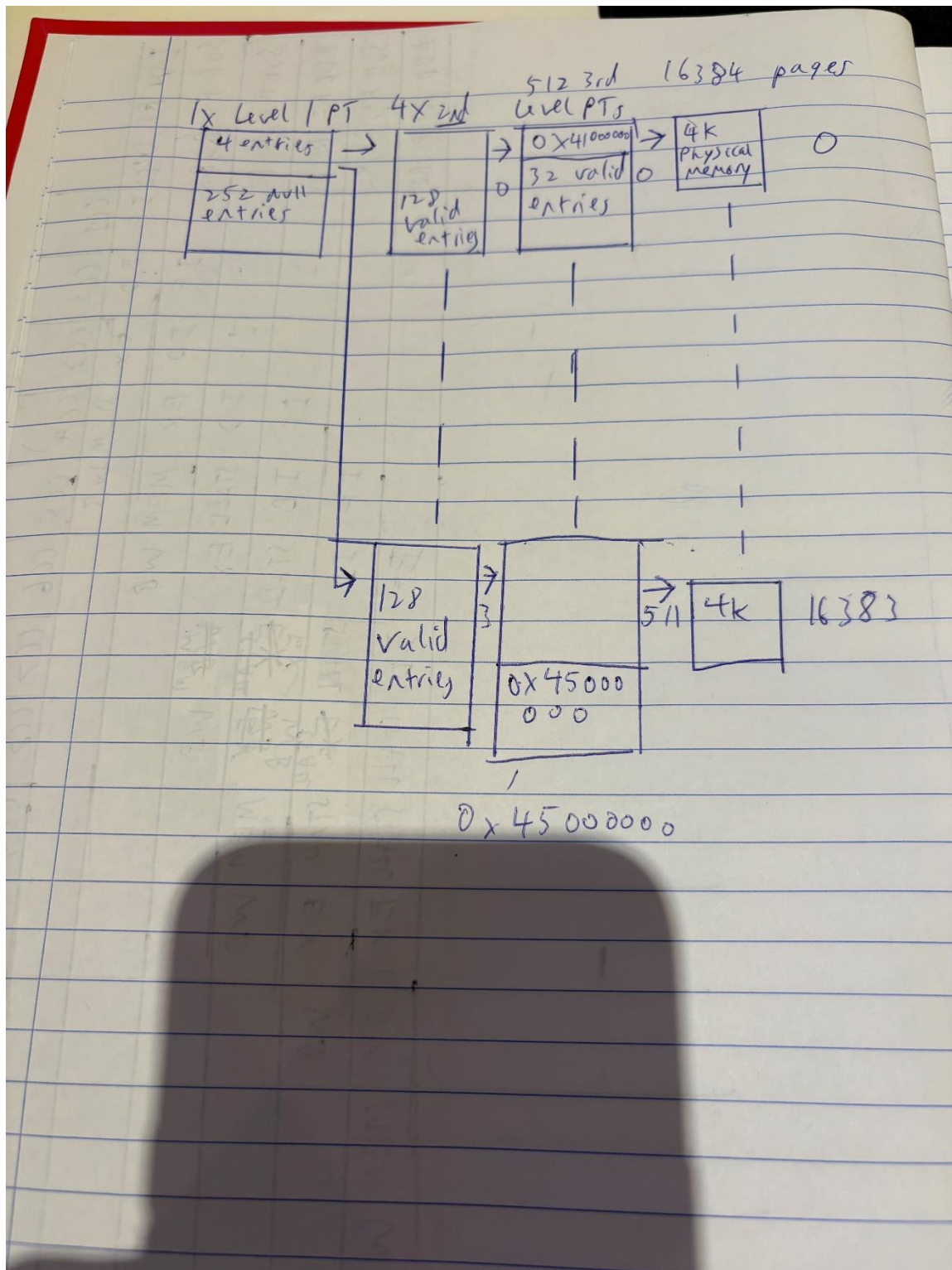
Configuration 2:

1st level: $(2^7) * 4 = 512\text{B}$

2nd level: $(2^7) * 4 = 512\text{B}$

3rd level: $(2^5) * 4 = 128\text{B}$

III:



Since there is 64MB of physical memory and each page is 4kB, that means there are $67108864 / 4096 = 16384$ pages to be mapped.

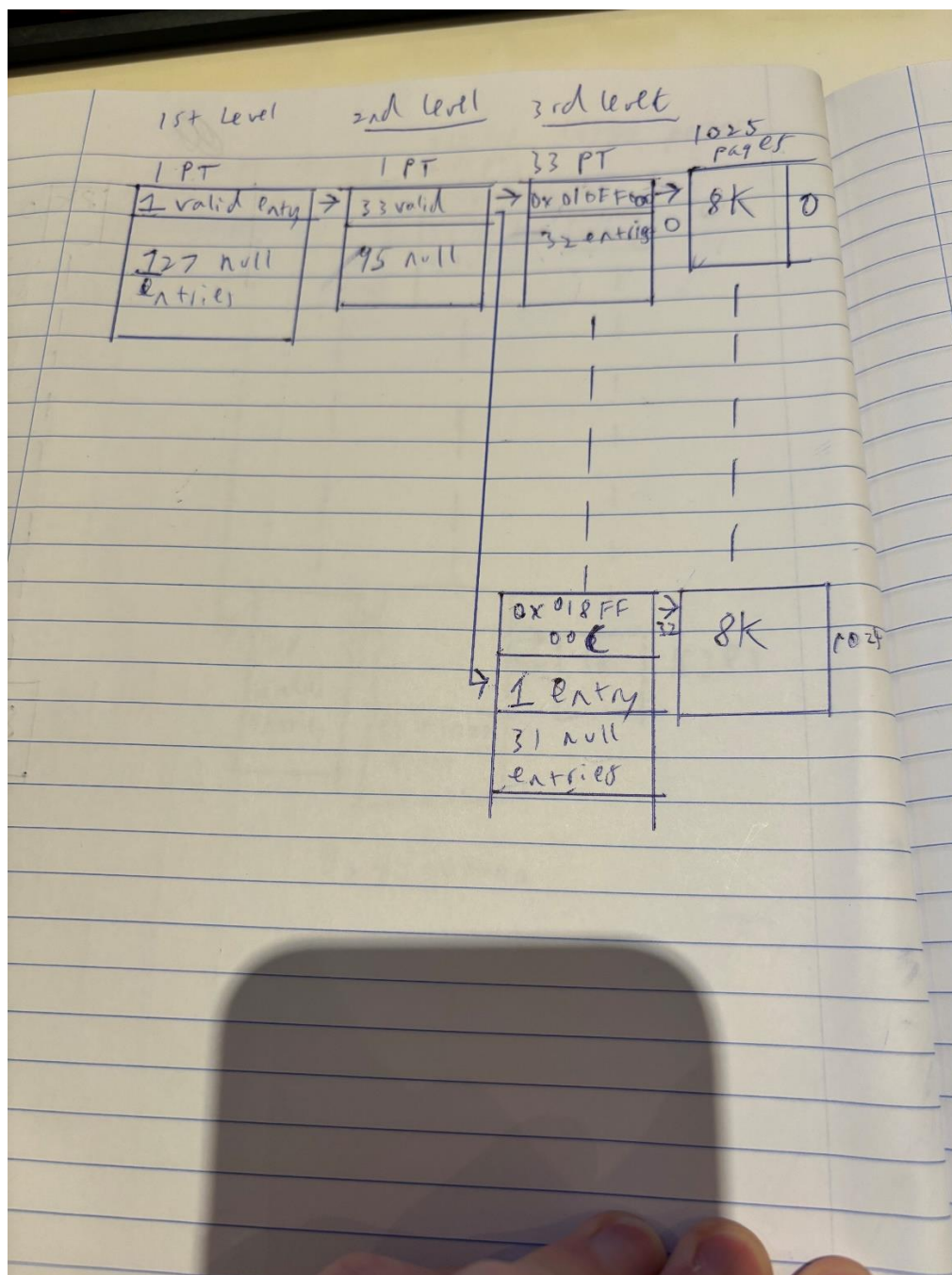
Since each level 3 page table has $2^5 = 32$ entries, it takes $16384 / 32 = 512$ level 3 page tables to map all the pages.

Since each level 2 page table has 128 entries, it takes $512 / 128 = 4$ level 2 page tables.

The level 1 page table has 256 entries and it takes 4 entries to map the 4 level 2 page tables.

Since it starts at address $0x41000000$, the 64MB spans until address $0x45000000$

IV:



So since each page is 8192B, the number of pages is

$8\text{MB} / 8192\text{B} = 1024 + 1$ page to account for the 12B so in total 1025 pages.

3rd level:

There are $1024/32 = 32$ full page tables + 1 page table with 1 valid entry pointing to the 1025th page.

So in total there are 33 pages.

2nd level:

There is 1 page table with 33 valid entries that points to the 33 3rd level page tables and $128-33 = 95$ null entries.

1st level:

There is 1 page table with 1 valid entry pointing to the 1 2nd level page table and $128-1 = 127$ null entries.