Task #1

I. DLX/MIPS pipeline execution

As we can see in the table below, it will take **11 clock cycles** to complete the operations. The reason it is 11 and not 10 is because the write back will begin in clock cycle 10 when carrying out the ADD, R3, R1, R2 instruction but won't complete until the 11th cycle. The values of the registers after execution are:

	| R1=5, R2=3, R3=1
R1=MemContent[R2]	| R1=2, R2=3, R3=1
R1 = R3 - R2	| R1=-2, R2=3, R3=1
R3 = R1 - R2	| R1=-2, R2=-5, R3=1
R2 = R1 + R2	| R1=-2, R2=1, R3=1
R1 = R1 - R2	| R1=-3, R2=1, R3=1
R3 = R1 + R2	| **R1=-3, R2=1, R3=-2**

INSTRUCTION	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8	CC9	CC10	CC11
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	

II. Out-of-order scheduling optimization

Optimization can occur by reordering independent instructions to avoid pipeline stalls. However, every instruction in the MIPS pseudocode is fully dependent on the previous instruction's result, which doesn't allow for out-of-order execution that would reduce the number of cycles.

III. Forwarding deactivated

Without forwarding, it will take **17 clock cycles** to complete, as we will have to wait for writeback to occur. The reason it is 17 and not 16 again is because the write back will begin in clock cycle 10 when carrying out the ADD, R3, R1, R2 instruction but won't complete until the 11th cycle. The final value of R1, R2, and R3 will still be **R1=-3, R2=1, R3=-2**, as deactivating forwarding and waiting in the pipeline will not affect correctness, only efficiency.

INSTRUCTION	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9	CC 10	CC 11	CC 12	CC 13	CC 14	CC 15	CC 16	CC 17
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	STALL	EX	MEM	WB								
ADD R2, R1, R2	-	-	-	IF	ID	STALL	STALL	EX	MEM	WB							
SUB R1, R1, R2	-	-	-	-	IF	ID	STALL	STALL	STALL	STALL	EX	MEM	WB				
ADD	-	-	-	-	-	IF	ID	STAL	STAL	STAL	STAL	STAL	STAL	EX	MEM	WB	

R3, R1, R2								L	L	L	L	L	L				

IV. Ignoring hazards, no stalling

If we have an improperly configured pipeline that does account for the possibilities of hazards by using forwarding and stalling, then it will take 11 clock cycles to execute, but the final values will be incorrect. These final values are: **R1=-5, R2=5, R3=1**. This is because execution occurs before write-back.

1. LW R1, 0(R2): Loads MemContent[3] into R1. Since MemContent[3] is 2, R1 becomes 2 after the write-back stage, which is completed at CC5.

2. SUB R1, R3, R2: Uses the initial values of R3 (1) and R2 (3), as R1's update has not been written back yet. So, R1 = 1 - 3 = -2. This value will be written back at CC6.

3. SUB R3, R1, R2: Uses the initial value of R1(5) as the value from the LW instruction is not returned until CC6 from the WB in CC5(2) since it has been written back, and the initial value of R2 (3). So, R3 = 5 - 3 = 2. This value will be written back at CC7.

4. ADD R2, R1, R2: Uses the updated value of R1 (2) from the LW instruction and the initial value of R2 (3). So, R2 = 2 + 3 = 5. This value will be written back at CC8.

5. SUB R1, R1, R2: Uses the value of R1 from the SUB R1, R3, R2 instruction (-2) since it has been written back, and the and the initial value of R2 (3). So, R1 = -2 - 3 = -5. This value will be written back at CC9.

6. ADD R3, R1, R2: Uses the value of R1 from the SUB R1, R3, R2 instruction (-2), as it has been written back, and the initial value of R2 (3). So, R3 = -2 + 3 = 1. This value will be written back after CC10 in CC11.

Task #2

I.Page Sizes Supported by the MMU

Configuration 8-7-5-12:
- Page Offset: 12 bits (2^{12} bytes per page) = 4096 bytes (4 KB) per page.

Configuration 7-7-5-13:
- Page Offset: 13 bits (2^{13} bytes per page) = 8192 bytes (8 KB) per page.

II. Size of Page Tables in Each Level

Configuration 8-7-5-12:
- Primary Page Level: 2^8 entries × 4 bytes (size of each page table entry) = 1024 bytes (1 KB).
- Secondary Page Level: 2^7 entries × 4 bytes = 512 bytes.
- Tertiary Page Level: 2^5 entries × 4 bytes = 128 bytes.

Configuration 7-7-5-13:

- Primary Page Level: 2^7 entries × 4 bytes = 512 bytes.
- Secondary Page Level: 2^7 entries × 4 bytes = 512 bytes.
- Tertiary Page Level: 2^5 entries × 4 bytes = 128 bytes.

III. 8-7-5-12 Configuration Page Table Structure

1. The primary page table will have 256 entries and since the page size is 4KB we will use 64MB/4KB = 16384 tertiary page tables
2. Each secondary table has 128 entries equating to 16384/128 = 128 secondary page tables
3. There will be 128/256 = 0.5 page tables but since we can't half a page table this will round up to 1 primary page table

Resulting structure would be:

- 1 primary page table with entries pointing to 128 secondary page tables
- Each secondary page table has entries for 128 tertiary page tables
- Each tertiary page table maps 32 pages, covering the 64MB of space

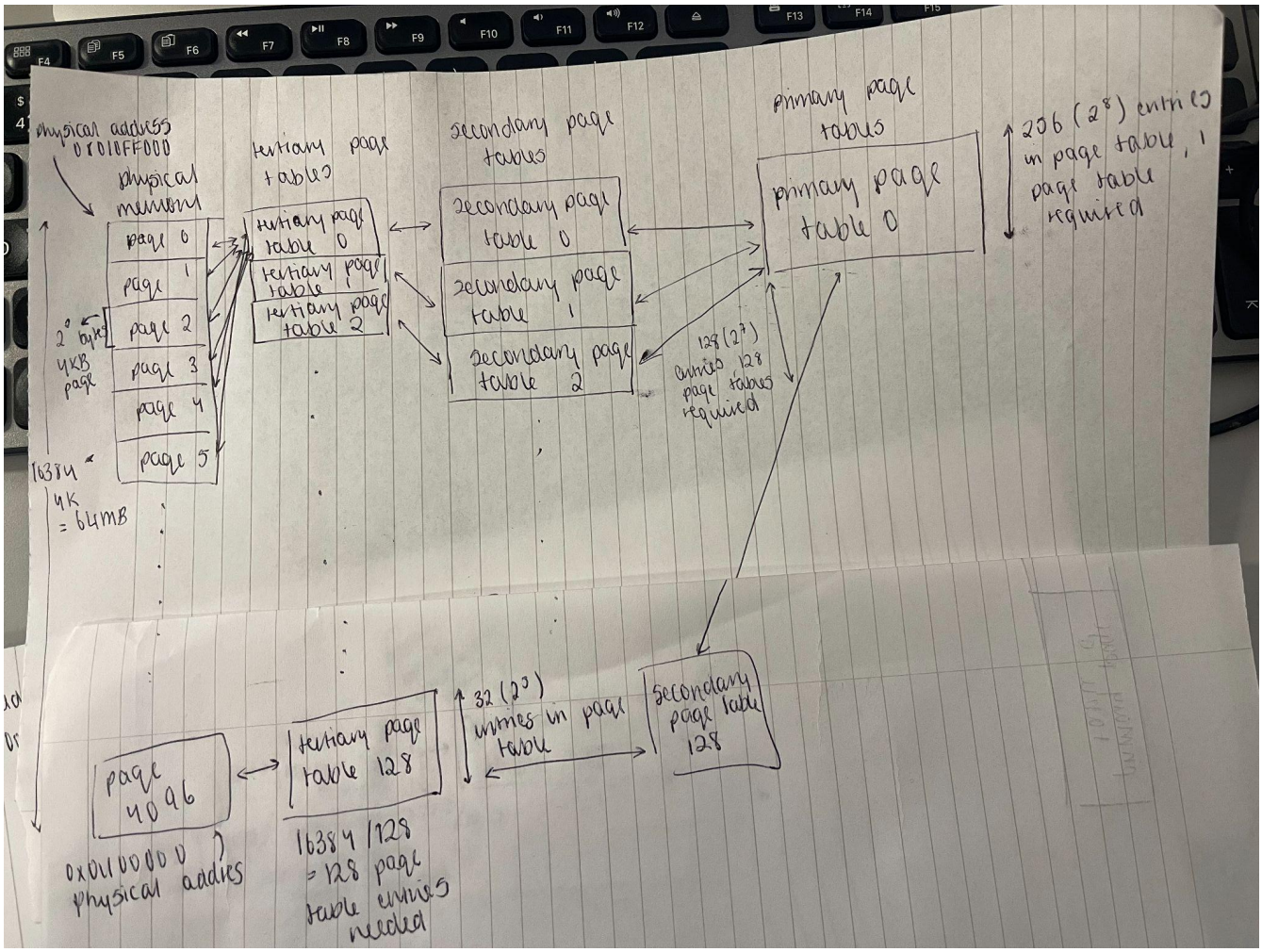
Assumptions:

- Kernel virtual space starts at 0x0F000000.
- Kernel space size is 64MB (64 × 2^20 bytes).
- Physical address space starts at 0x41000000.

Page Table Structure:

- The 8-7-5-12 split means that the page table is a three-level hierarchy with each level having 2^8, 2^7, and 2^5 entries, respectively.

Visual Representation:



IV. 7-7-5-13 Configuration Page Table Structure

1. The primary table will have 128 entries and since the page size is 8KB we will use 8mb/8kb = 1024 tertiary page tables
2. Each secondary page table has 128 entries, equating to 1024/128 = 8 secondary page tables
3. There will be 8/128 = 0.0625 primary page tables, again rounding up we need 1 primary page table.

Resulting structure would be:

- 1 primary page table with entries pointing to 8 secondary page tables
- Each secondary page table has entries for 128 tertiary page tables
- Each tertiary page table maps 32 pages covering the 8MB of space. Since we have 12 extra bytes we need an additional entry to cover this, making a total of 1025 entries at tertiary level

Assumptions:

- Kernel virtual space starts at 0x00000000.
- Kernel space size is 8MB plus 12B.
- Physical address space starts at 0x010FF000.

Page Table Structure:

- The 7-7-5-13 split implies a three-level hierarchy with each level having 2^7, 2^7, and 2^5 entries, respectively.

Visual Representation: