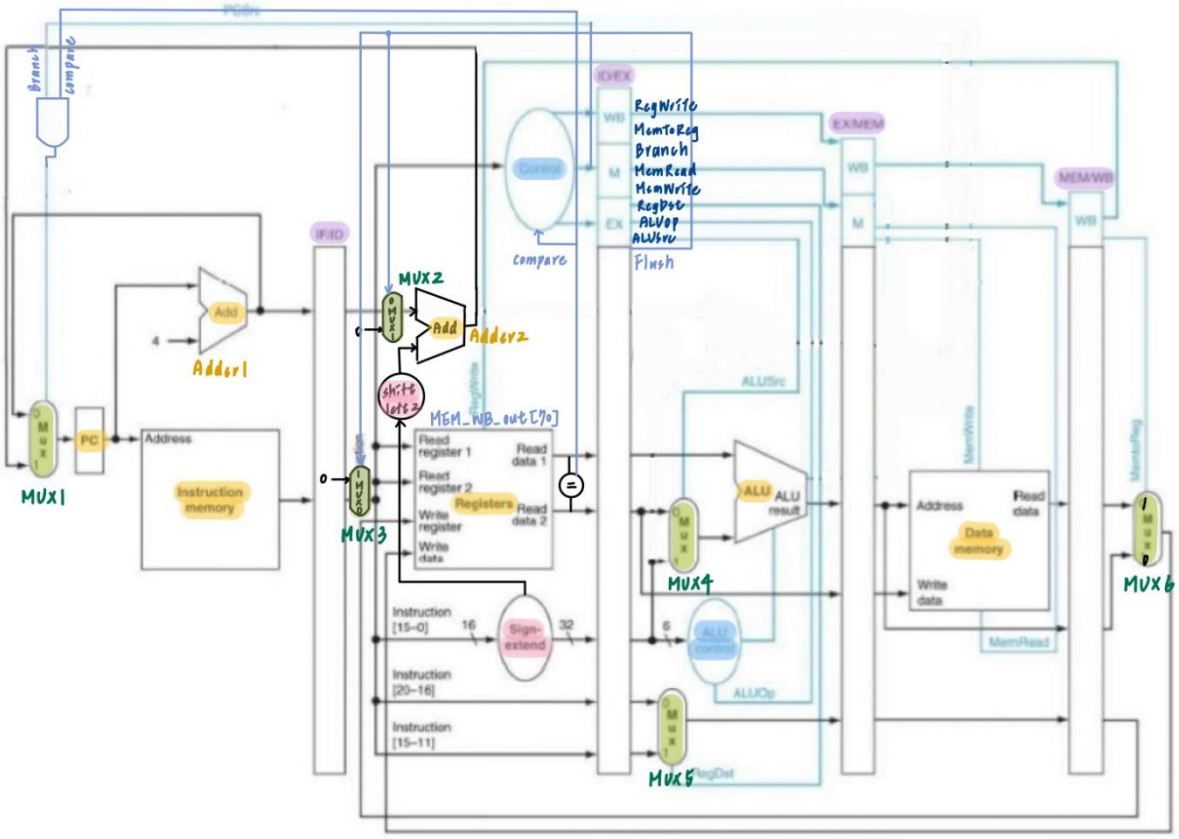
Computer Organization Lab4

Name: 龔祐萱

ID: 0713216

Architecture diagrams:



針對所給的電路架構圖進行了以下的修改，使得 pipeline CPU 架構可以符合各項指令要求:

1. MUX6

為了使得控制訊號的輸出較直覺，因此修改了 MUX6 的輸出結果。如果 MemtoReg=1 時，則 MUX6 會選擇從 Data memory 輸入至 MUX6 的 input；如果 MemtoReg=0 時，則 MUX6 會選擇 ALU_result 輸出至 Register。

2. beq 指令

為了使得 pipeline CPU 可以符合 beq 指令的要求，進行了以下的修改。首先，新增一個 comparator，並輸入從 Register 讀出的兩筆資料做比較，就可以在 stage2 提前知道是否要做 branch。並將原本在 stage3 的 adder2 移至 stage2，將 adder2 算得的 target address 輸入至 MUX1。

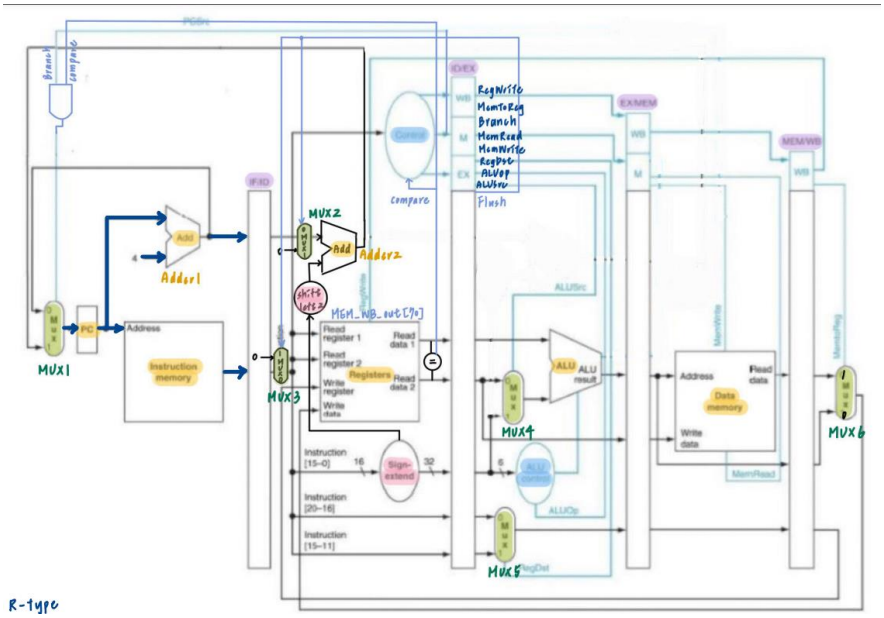
新增兩個控制訊號 Compare 和 Flush，Compare 是由 comparator 輸出並輸入至 Control 中，如果 Compare=1 表示要做 branch，則 Flush=1，使得 MUX2 和 MUX3 輸入 bubble 暫停下一個已經輸入的 sequential 指令並重新抓取正確的指令；如果 Compare=0 表示不做 branch，則 Flush=0，因此 MUX2 和 MUX3 會輸入下一個要執行的 sequential 指令。

Hardware module analysis:

1. R-type instruction (add, sub, and, or, slt, mult) 執行過程

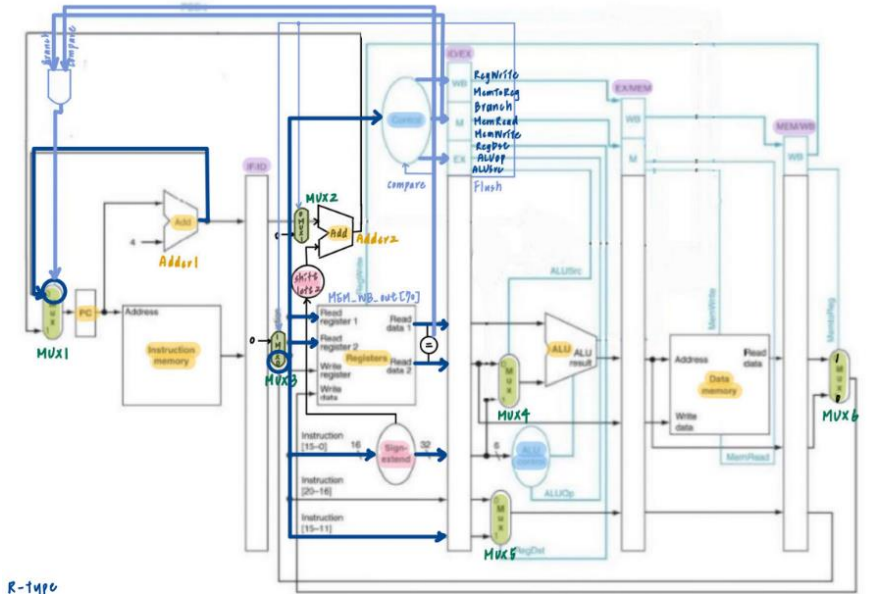
(1) Stage1 (IF)

PC 會指到 instruction memory 中的一個位置，並將該位置的指令輸出。



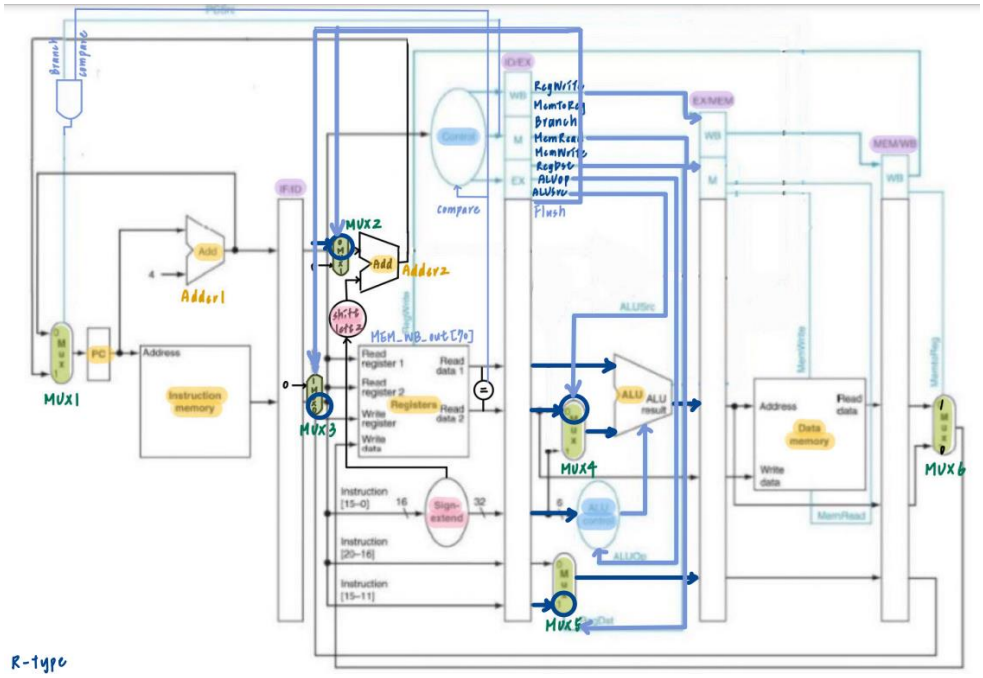
(2) Stage2 (ID)

指令的 31-26 bit 會輸入 control (decoder)進行解碼，並輸出 control signal，指令的 25-16 bit 會輸入 Register 作為 Read Register，並輸出該 register 的資料，指令的 15-11 bit 會作為 Write Register，並和上述資料輸入 ID/EX pipeline register，以及將下一個指令輸入 PC (PC+4)。



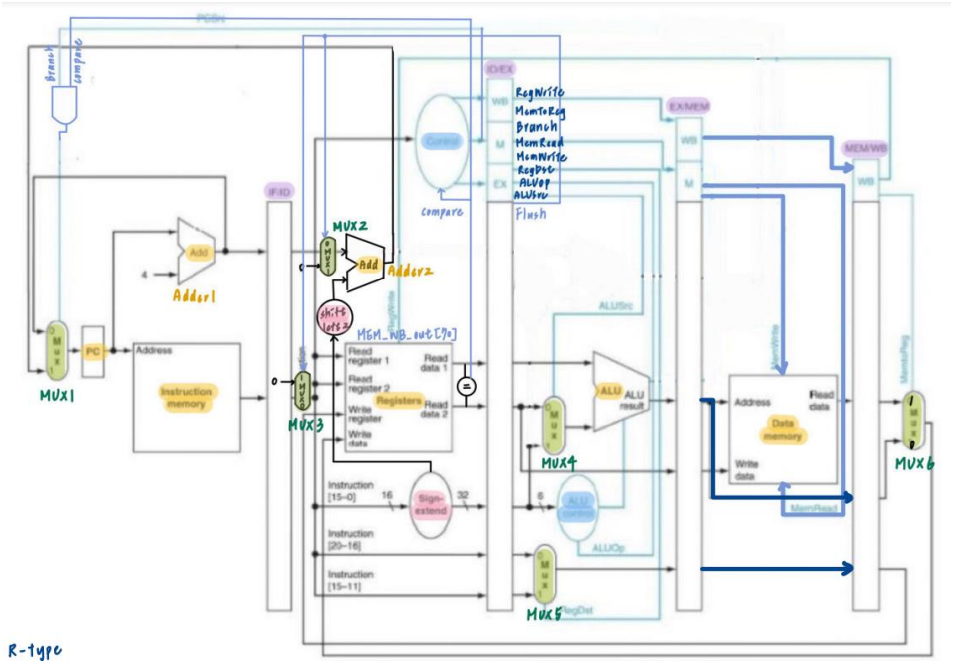
(3) Stage3 (EX)

將從 ID/EX pipeline register 輸出的資料分別輸入 ALU、ALU control 以及 MUX 進行運算，並將結果輸至 EX/MEM pipeline register 儲存。



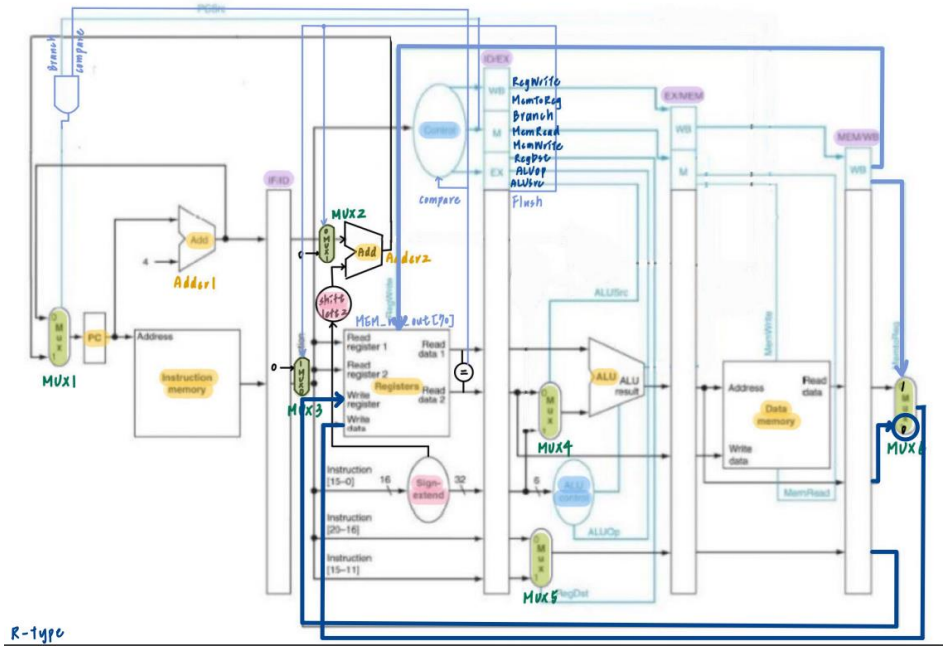
(4) Stage4 (MEM)

將 EX/MEM pipeline register 輸出的資料輸入 MEM/WB pipeline register 儲存。



(5) Stage5 (WB)

將從 MEM/WB pipeline register 輸出的資料分別輸至 Register 的 WiteReg 和 WriteDate 中，更新指定 register 的數值。



(6) 以下是 R-type instruction 的 control signal 整理表格

Compare	Branch
依 data 決定	0

Flush	ALUSrc	ALUOp (3 bits)	RegDst	ALU_control_output (4 bits)
0	0	010	1	add : 0010 sub : 0110 and : 0000 or : 0001 slt : 0111 mult : 1100

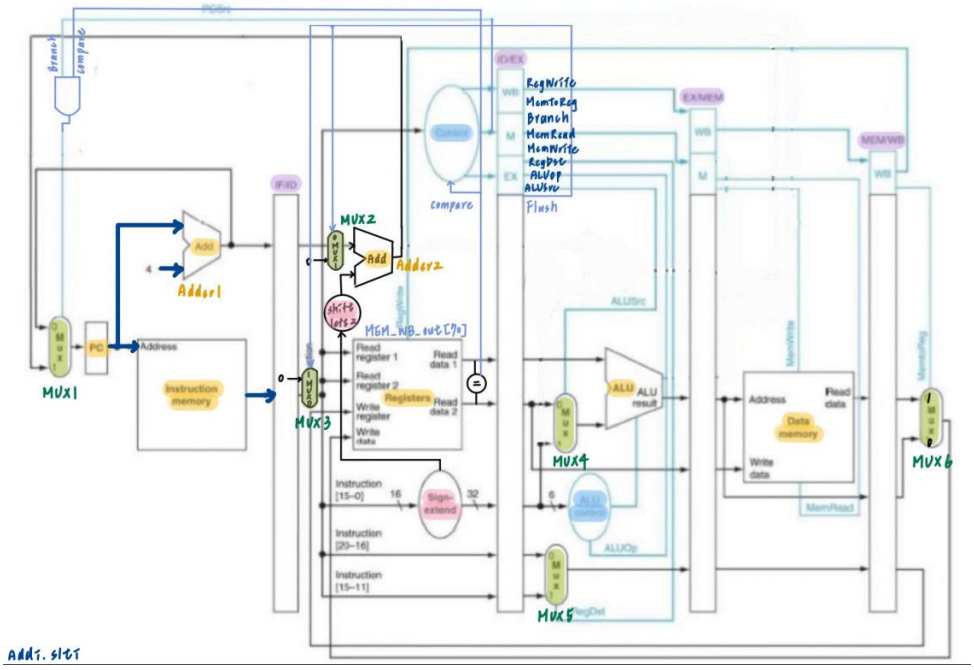
MemWrite	MemRead
0	0

MemtoReg	RegWrite
0	1

2. I-type instruction (addi, slti) 執行過程

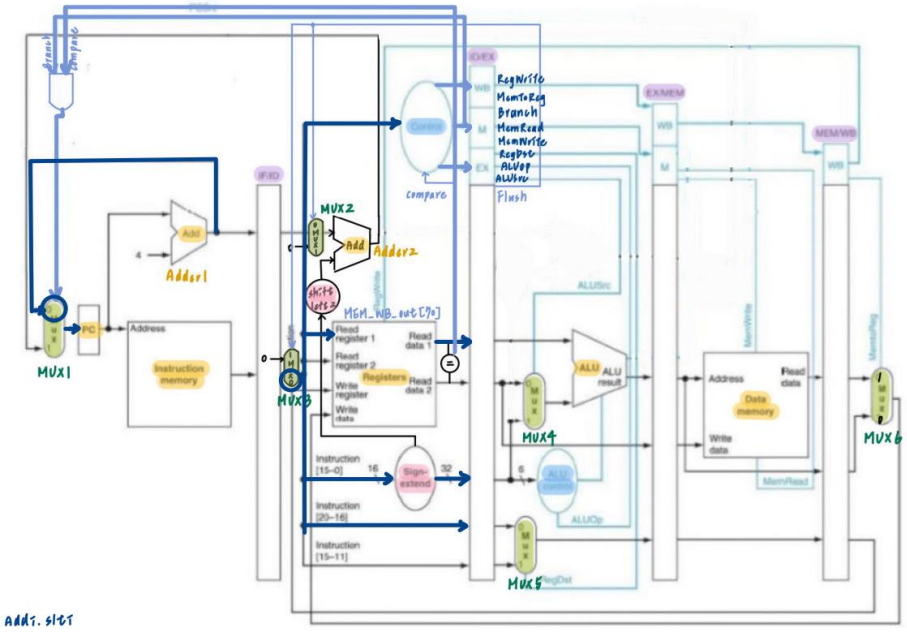
(1) Stage1 (IF)

PC 會指到 instruction memory 中的一個位置，並將該位置的指令輸出。



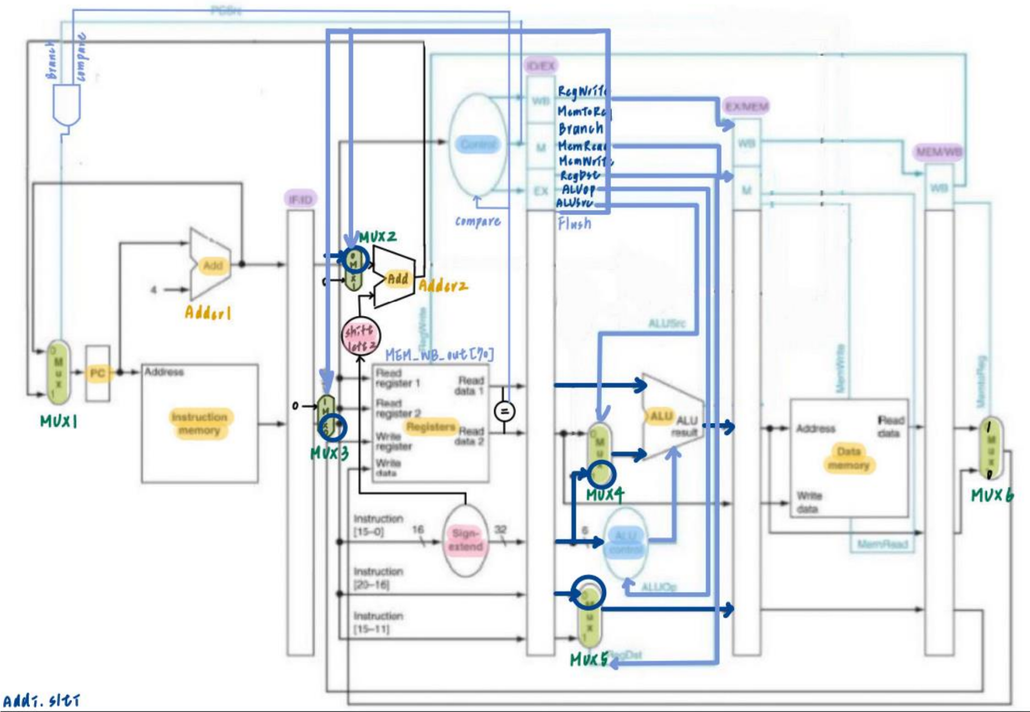
(2) Stage2 (ID)

指令的 31-26 bit 會輸入 control (decoder)進行解碼，並輸出 control signal，指令的 25-21 bit 會輸入 Register 作為 Read Register，並輸出該 register 的資料，指令的 15-0 bit 會進入 sign-extend 進行 sign extension，指令的 15-11 bit 會作為 Write Register，並和上述資料輸入 ID/EX pipeline register，以及將下一個指令輸入 PC (PC+4)。



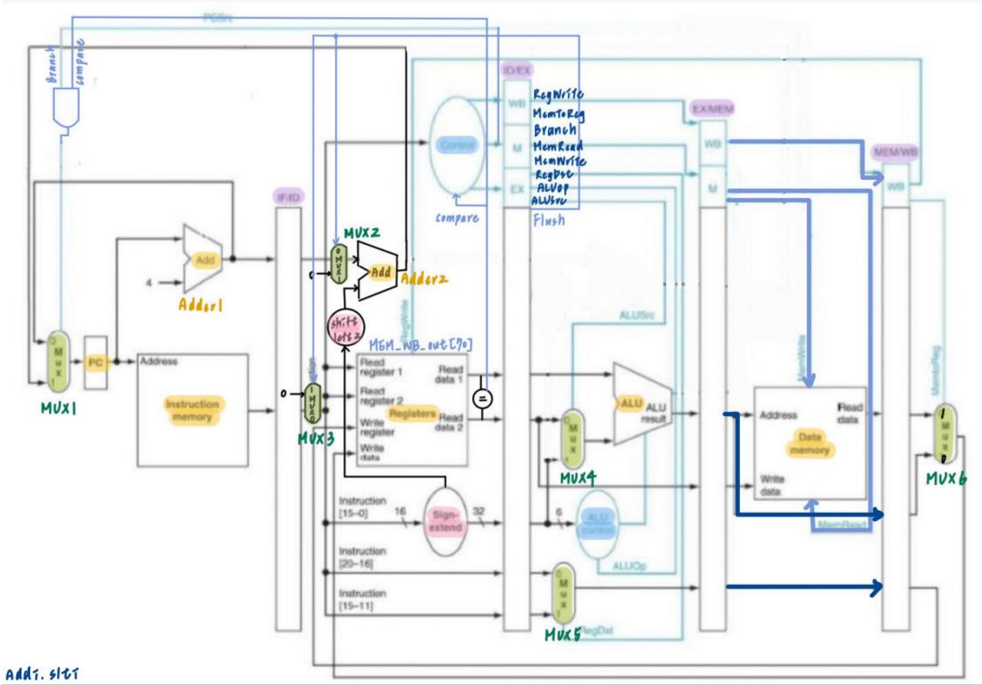
(3) Stage3 (EX)

將從 ID/EX pipeline register 輸出的資料分別輸入 ALU、ALU control 以及 MUX 進行運算，並將結果輸至 EX/MEM pipeline register 儲存。



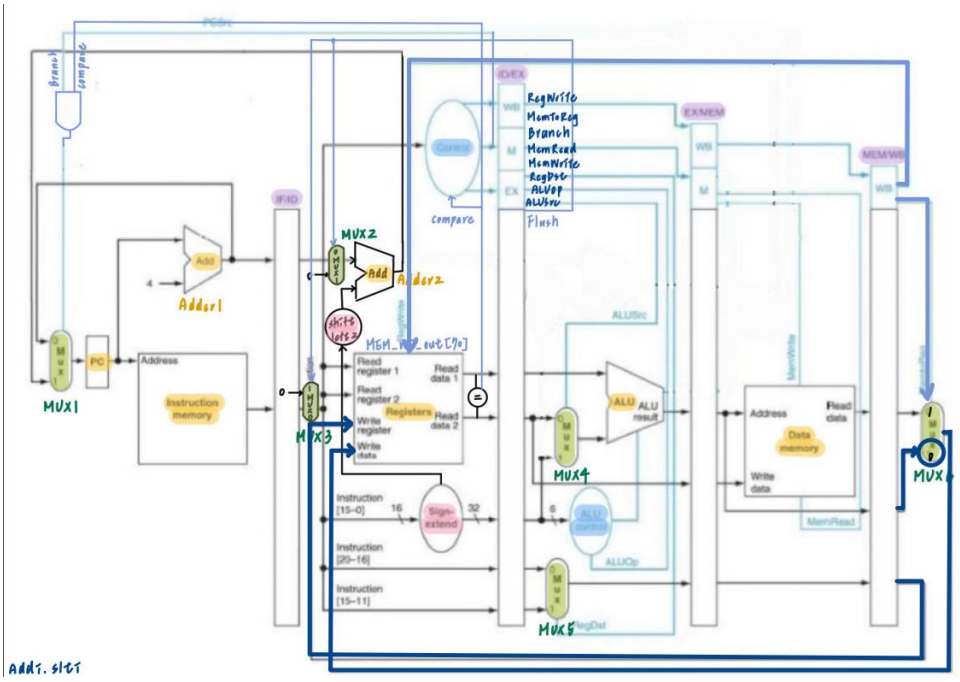
(4) Stage4 (MEM)

將 EX/MEM pipeline register 輸出的資料輸入 MEM/WB pipeline register 儲存。



(5) Stage5 (WB)

將從 MEM/WB pipeline register 輸出的資料分別輸至 Register 的 WiteReg 和 WriteDate 中，更新指定 register 的數值。



(6) 以下是 I-type instruction 的 control signal 整理表格

Compare	Branch
依 data 決定	0

Flush	ALUSrc	ALUOp (3 bits)	RegDst	ALU_control_output (4 bits)
0	1	addi : 000 slti : 011	0	addi : 1000 slti : 0101

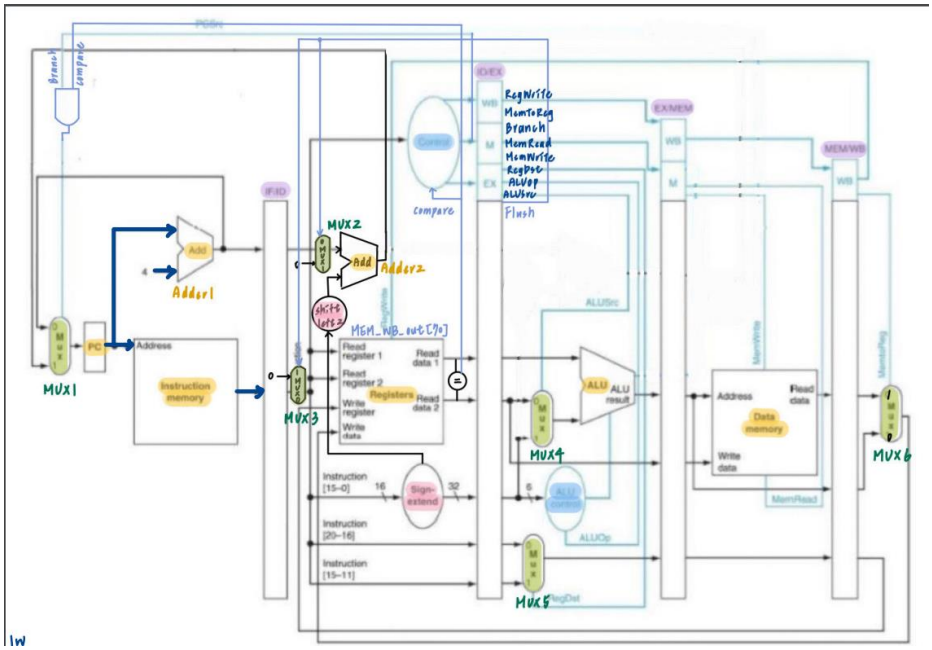
MemWrite	MemRead
0	0

MemtoReg	RegWrite
0	1

3. lw instruction 執行過程

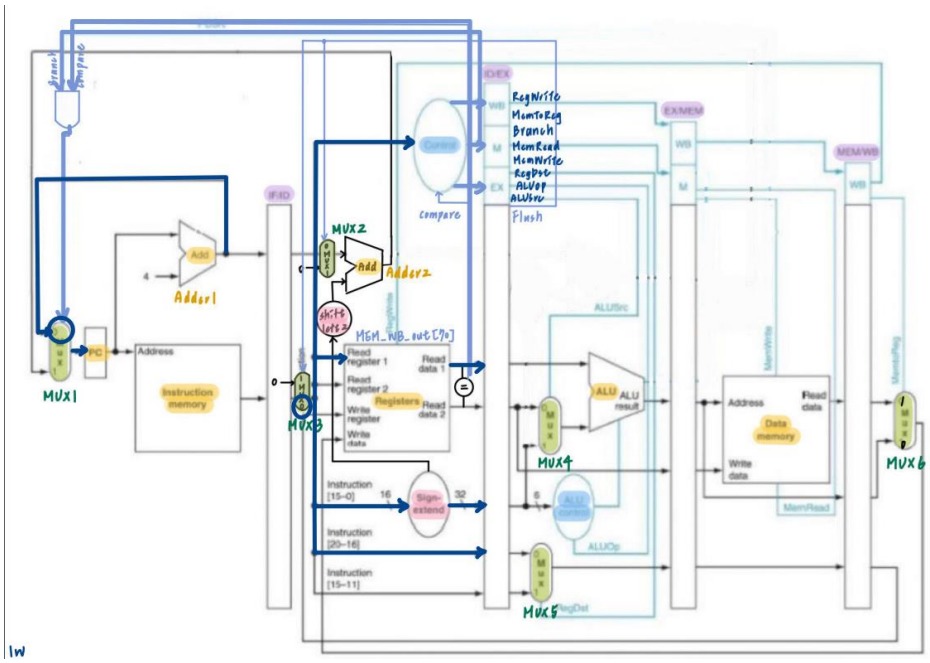
(1) Stage1 (IF)

PC 會指到 instruction memory 中的一個位置，並將該位置的指令輸出。



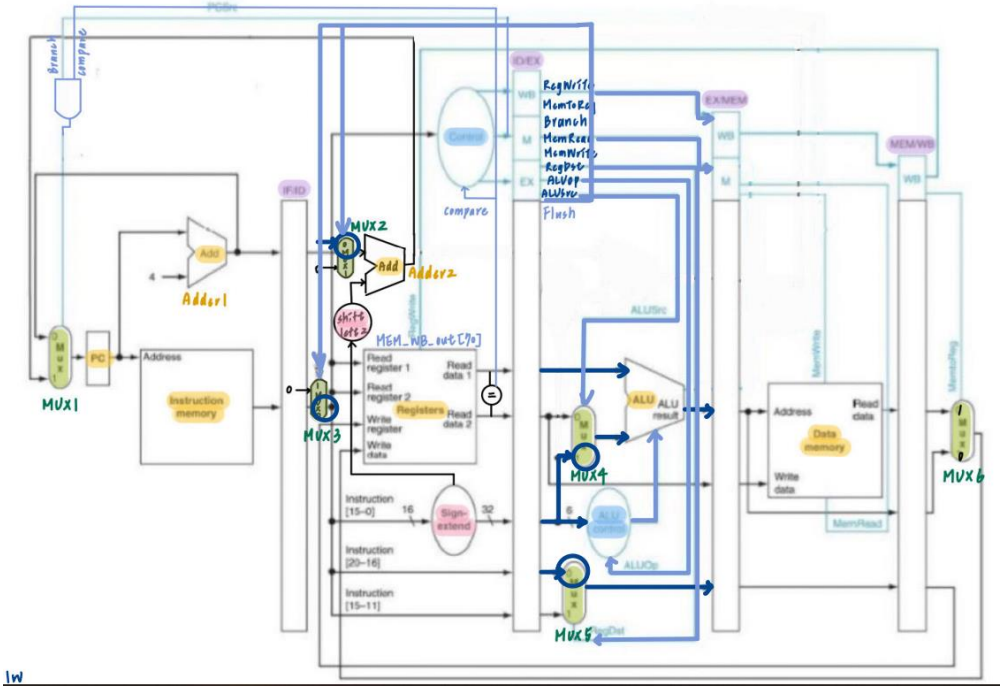
(2) Stage2 (ID)

指令的 31-26 bit 會輸入 control (decoder)進行解碼，並輸出 control signal，指令的 25-21 bit 會輸入 Register 作為 Read Register，並輸出該 register 的資料，指令的 15-0 bit 會進入 sign-extend 進行 sign extension，指令的 20-16 bit 會作為 Write Register，並和上述資料輸入 ID/EX pipeline register，以及將下一個指令輸入 PC (PC+4)。



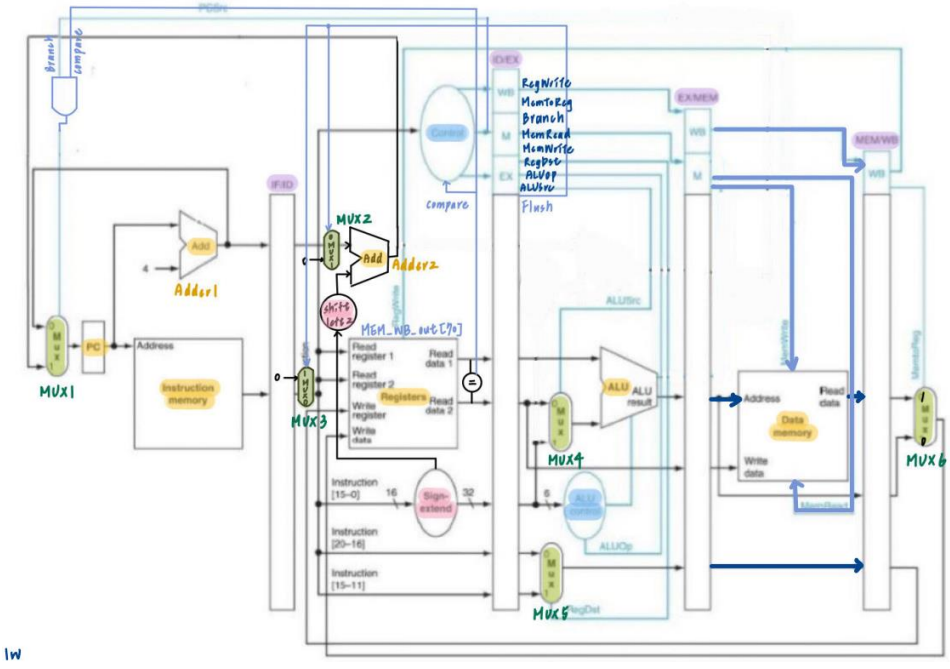
(3) Stage3 (EX)

將從 ID/EX pipeline register 輸出的資料分別輸入 ALU、ALU control 以及 MUX 進行運算，並將結果輸至 EX/MEM pipeline register 儲存。



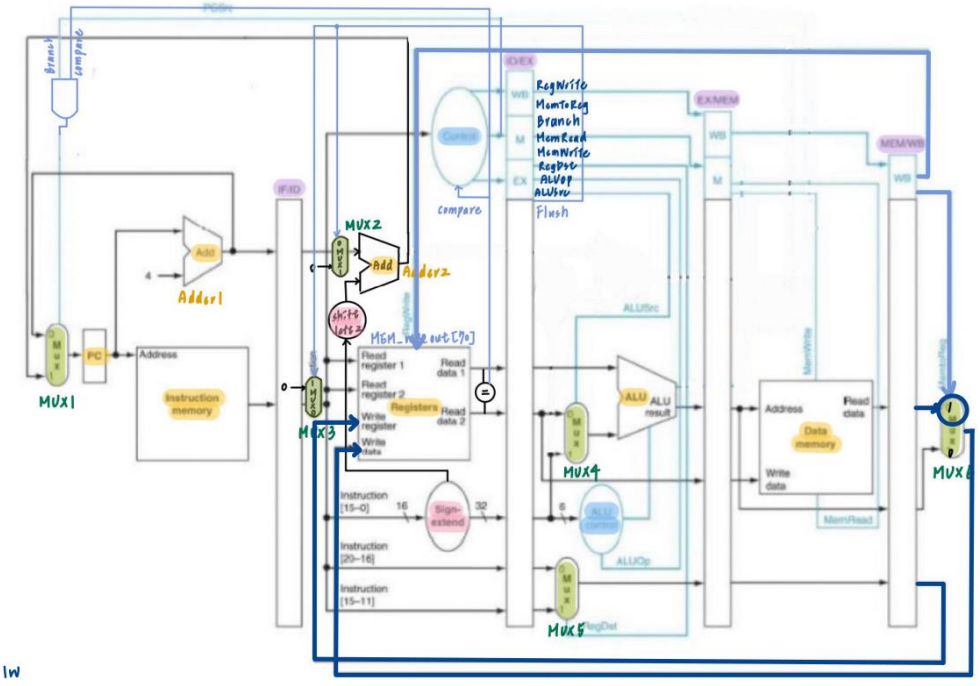
(4) Stage4 (MEM)

將 EX/MEM pipeline register 輸出的資料輸入 Data Memory 的 address 讀取資料並輸出至 MEM/WB pipeline register 儲存。



(5) Stage5 (WB)

將從 MEM/WB pipeline register 輸出的資料分別輸至 Register 的 WiteReg 和 WriteDate 中，更新指定 register 的數值。



(6) 以下是 lw instruction 的 control signal 整理表格

Compare	Branch
依 data 決定	0

Flush	ALUSrc	ALUOp (3 bits)	RegDst	ALU_control_output (4 bits)
0	1	100	0	0010

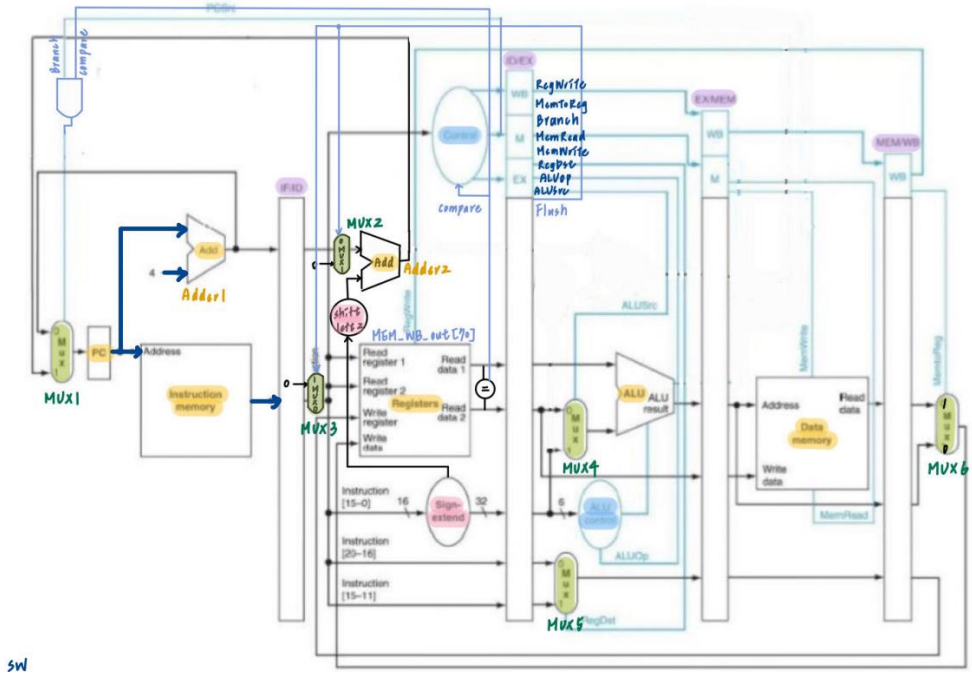
MemWrite	MemRead
0	1

MemtoReg	RegWrite
0	1

4. sw instruction 執行過程

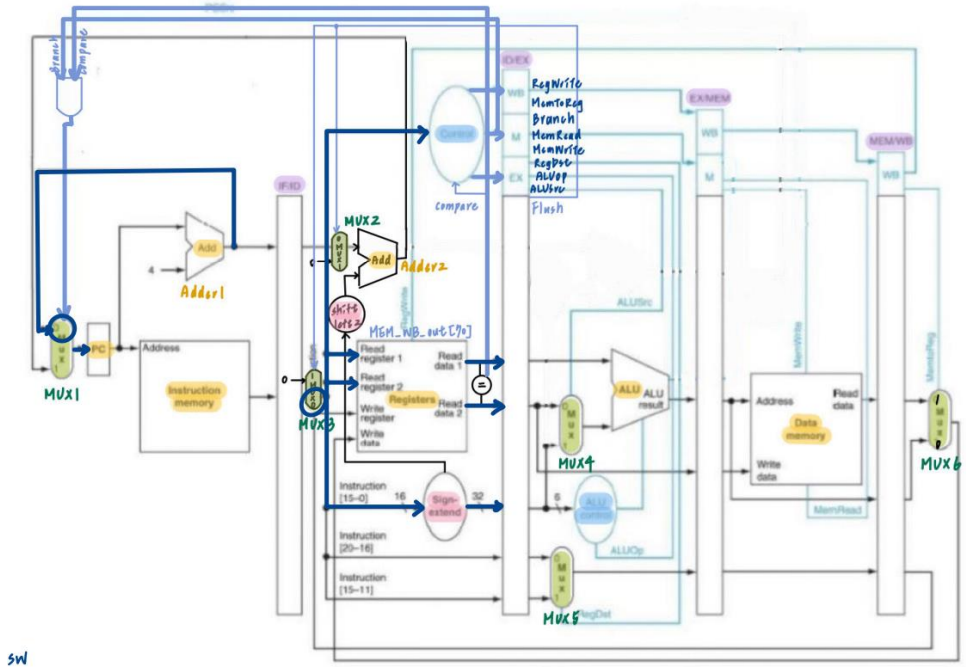
(1) Stage1 (IF)

PC 會指到 instruction memory 中的一個位置，並將該位置的指令輸出。



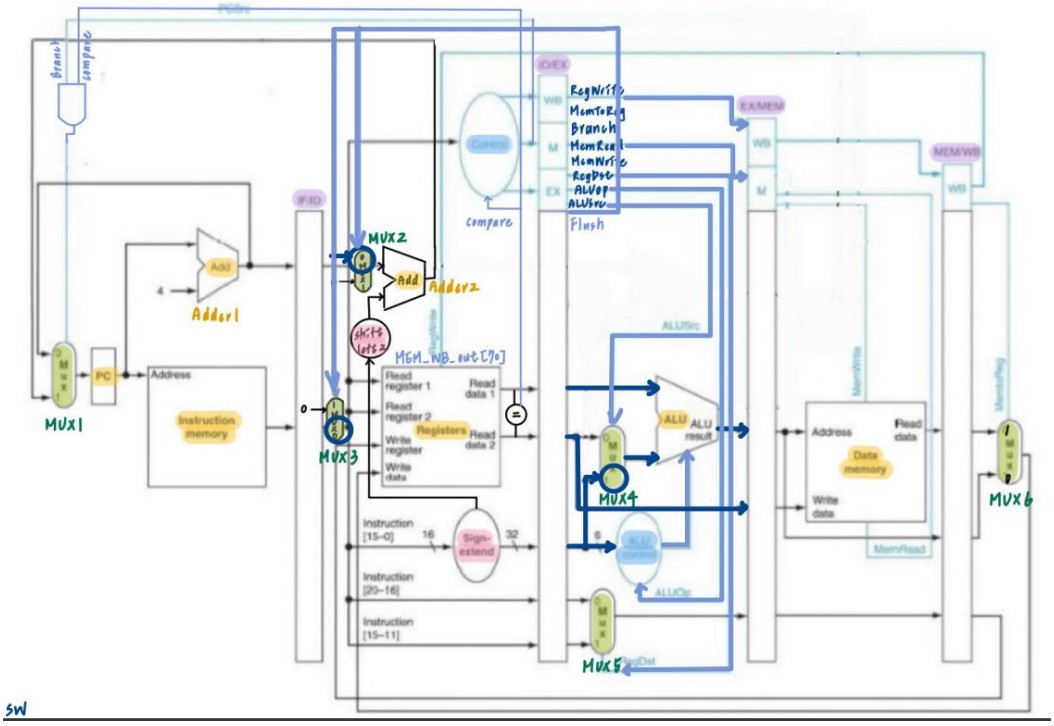
(2) Stage2 (ID)

指令的 31-26 bit 會輸入 control (decoder)進行解碼，並輸出 control signal，指令的 25-16 bit 會輸入 Register 作為 Read Register，並輸出該 register 的資料，指令的 15-0 bit 會進入 sign-extend 進行 sign extension，並和上述資料輸入 ID/EX pipeline register，以及將下一個指令輸入 PC (PC+4)。



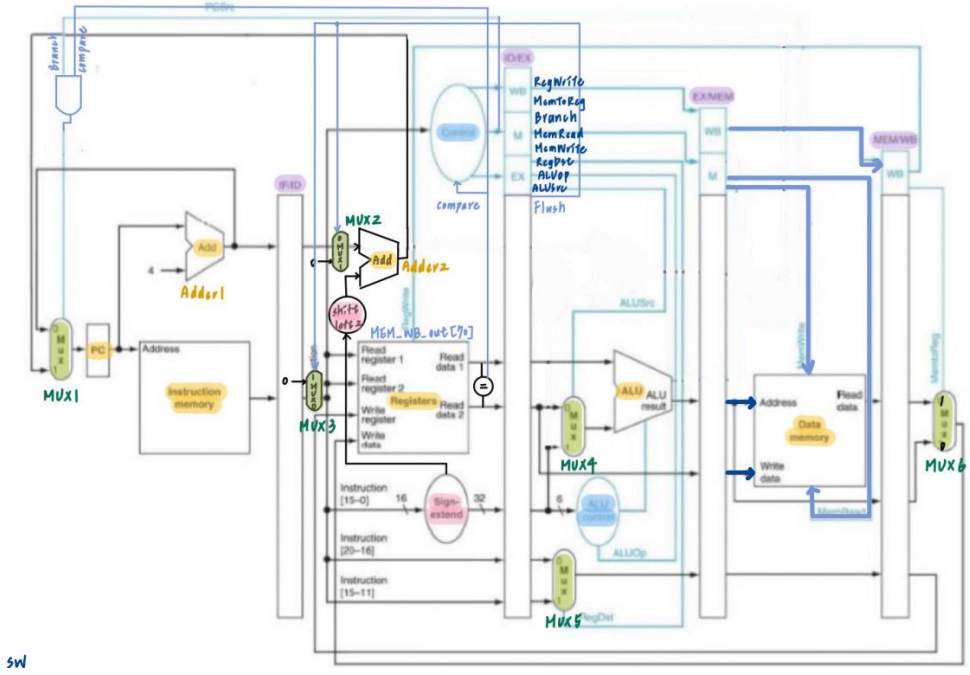
(3) Stage3 (EX)

將從 ID/EX pipeline register 輸出的資料分別輸入 ALU、ALU control 以及 MUX 進行運算，並將結果輸至 EX/MEM pipeline register 儲存。



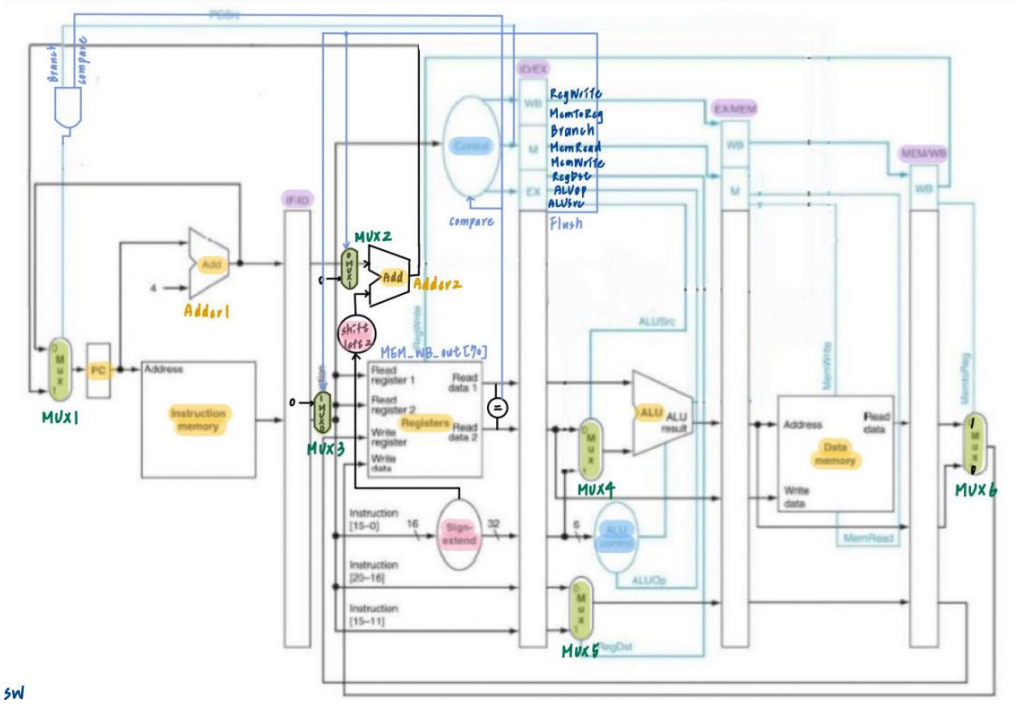
(4) Stage4 (MEM)

將 EX/MEM pipeline register 輸出的資料輸入 Data Memory 的 address 儲存，更新指定 memory address 的資料。



(5) Stage5 (WB)

sw 指令不會執行 write back 的程序。



(6) 以下是 sw instruction 的 control signal 整理表格

Compare	Branch
依 data 決定	0

Flush	ALUSrc	ALUOp (3 bits)	RegDst	ALU_control_output (4 bits)
0	1	101	0	0010

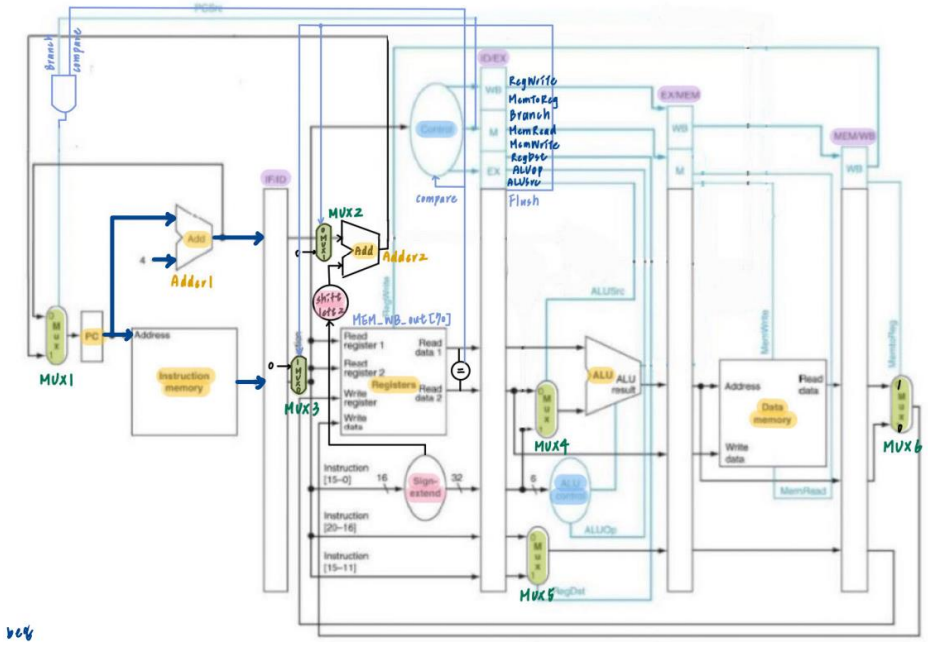
MemWrite	MemRead
1	0

MemtoReg	RegWrite
0	0

5. beq instruction 執行過程

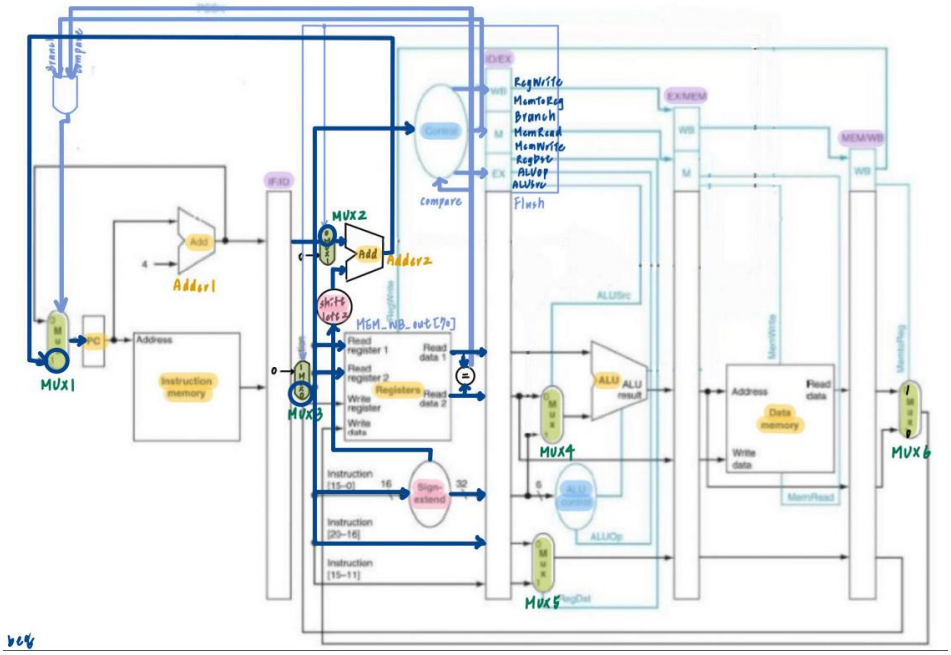
(1) Stage1 (IF)

PC 會指到 instruction memory 中的一個位置，並將該位置的指令輸出。



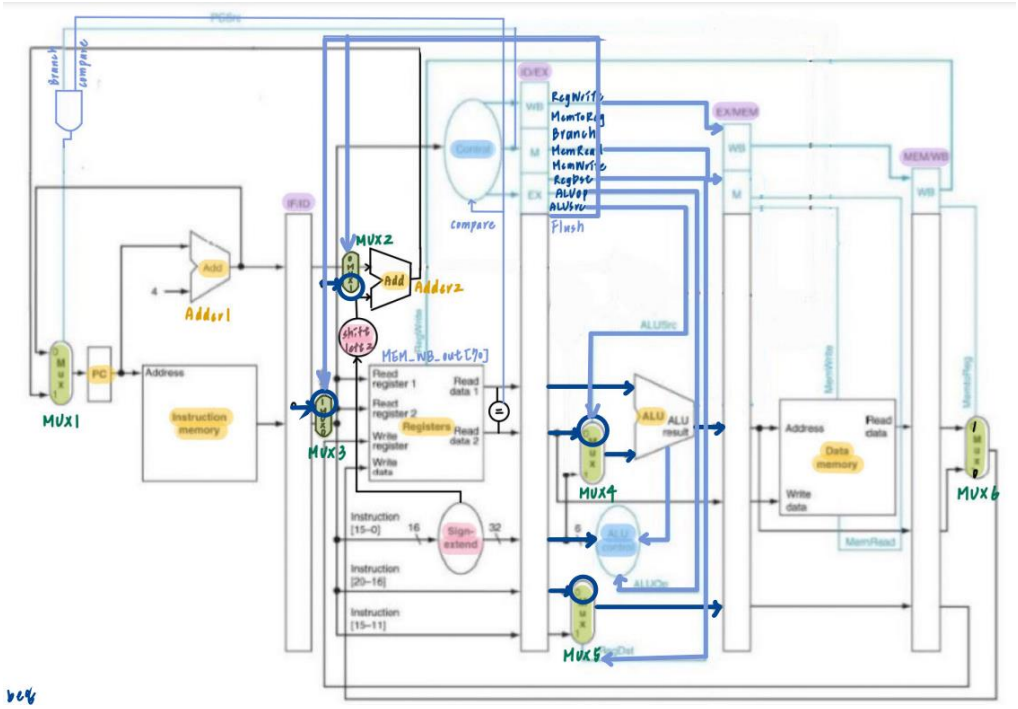
(2) Stage2 (ID)

指令的 31-26 bit 會輸入 control (decoder)進行解碼，並輸出 control signal，指令的 25-21 bit 會輸入 Register 作為 Read Register，並輸出該 register 的資料，指令的 15-0 bit 會進入 sign-extend 進行 sign extension，指令的 20-16 bit 會作為 Write Register，並和上述資料輸入 ID/EX pipeline register，以及依照是否會做 branch 的結果將下一個指令輸入 PC (PC+4/target address)。



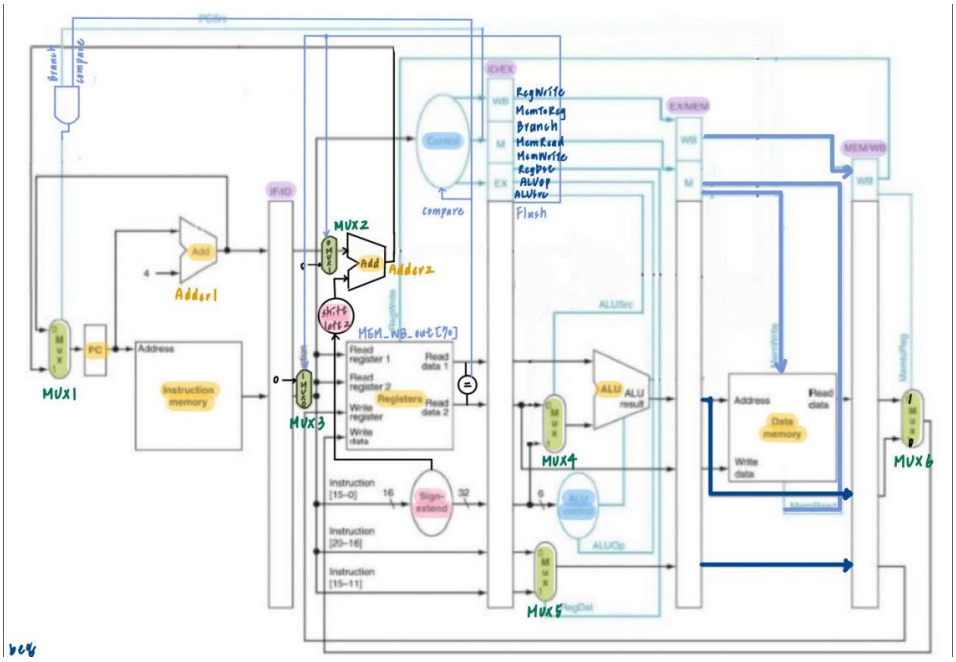
(3) Stage3 (EX)

將從 ID/EX pipeline register 輸出的資料分別輸入 ALU、ALU control 以及 MUX 進行運算，並將結果輸至 EX/MEM pipeline register 儲存。



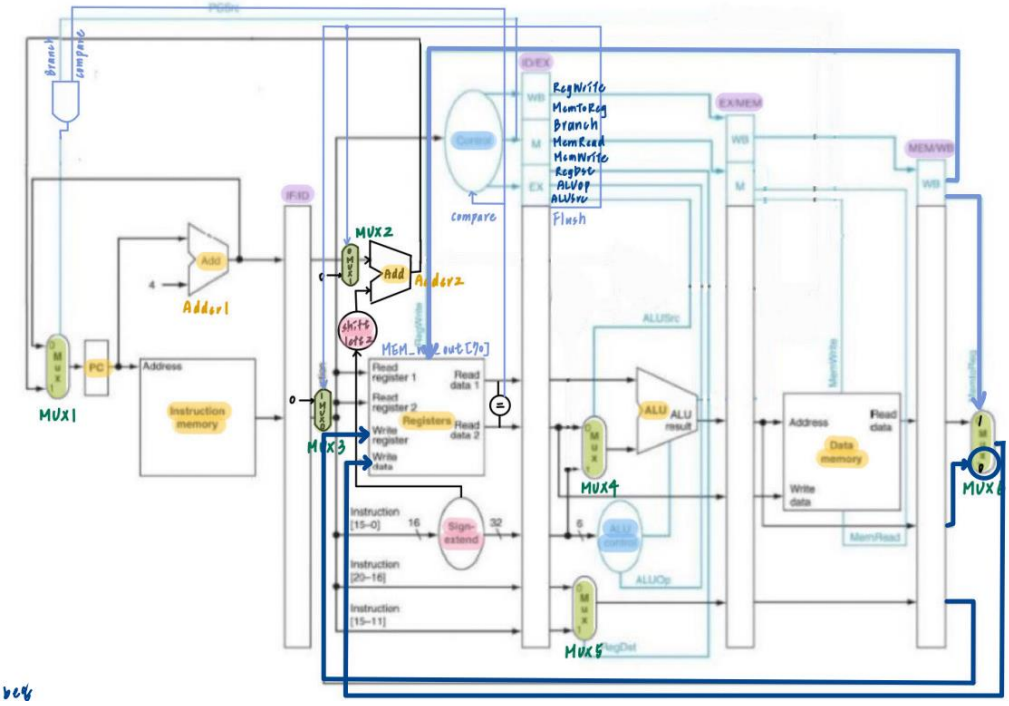
(4) Stage4 (MEM)

將 EX/MEM pipeline register 輸出的資料輸入 Data Memory 的 address 讀取資料並輸出至 MEM/WB pipeline register 儲存。



(5) Stage5 (WB)

將從 MEM/WB pipeline register 輸出的資料分別輸至 Register 的 WiteReg 和 WriteDate 中，更新指定 register 的數值。



(6) 以下是 beq instruction 的 control signal 整理表格

Compare	Branch
依 data 決定	1

Flush	ALUSrc	ALUOp (3 bits)	RegDst	ALU_control_output (4 bits)
依 data 決定	0	001	0	1010

MemWrite	MemRead
0	0

MemtoReg	RegWrite
0	0

6. 優缺點

因為 pipeline CPU 加入了 pipeline register，所以使得 pipeline CPU 可以被分成多個部分，可以分別執行 instruction fetch、decoding、execution、memory store and load、write back 等五個 stage，而每個 stage 都可以分別執行一個指令，所以在每個 clock cycle，都可以丟入一個指令進入 pipeline CPU 執行，因此每個 clock cycle 可以同時執行多個指令，使得 CPU 的整體執行效率提升。

也因為 pipeline CPU 的特殊架構，所以一個指令從丟入到執行完成需要五個 clock cycle 才會完成，也表示需要五個 clock cycle pipeline CPU 中的 register 或 data memory 中的資料才會做更新。因此如果前後兩個指令具有 data dependency 的關係時，兩個指令如果一前一後丟入 pipeline CPU 執行，會因為前一個指令對於 register 或 memory 的資料未做更新，而導致後一個指令的執行結果錯誤，所以就需要依賴 complier 將指令順序做 reorder 或是對於 data hazard 做硬體上的調整。

此外，因為 pipeline CPU 是每個 clock cycle 都會丟入一個指令執行，因此可能會出現 beq 指令需要做 branch，而下一個 sequential 指令已經進入 pipeline CPU 的情況，這個時候就需要做 flush 或 stall sequential 指令，並輸入正確的指令做修正，但如果輸入 stall 就表示 pipeline CPU 中會有一個 stage 是 no operation，因此會降低 pipeline CPU 的執行效率。

Finished part:

以下是測資的執行指令、指令執行過程、執行結果以及程式執行結果截圖:

1. Case 1

執行指令	指令執行過程	執行結果 (register)	執行結果 (memory)
(1) begin: (2) addi $1, $0, 3 (3) addi $2, $0, 4 (4) addi $3, $0, 1 (5) sw $1, 4($0) (6) add $4, $1, $1 (7) or $6, $1, $2 (8) and $7, $1, $3 (9) sub $5, $4, $2 (10) slt $8, $1, $2 (11) beq $1, $2, begin (12) lw $10, 4($0)	(1) r1 = r0+3 = 0+3 = 3 (2) r2 = r0+4 = 0+4 =4 (3) r3 = r0+1 = 0+1 =1 (4) Store r1=3 to m1, M1 = 3 (5) r4 = r1+r1 = 3+3 =6 (6) r6 = r1 OR r2 =7 (7) r7 = r1 AND r3 = 1 (8) r5 = r4 – r2 = 2 (9) (r1=3) < (r2=4), r8 = 1 (10) (r1=3) ≠ (r2=4), no branch (11) Load m1=3 to r10, r10 = 3	R0 = 0 R1 = 3 R2 = 4 R3 = 1 R4 = 6 R5 = 2 R6 = 7 R7 = 1 R8 = 1 R9 = 0 R10 = 3 R11~R31 = 0	M0 = 0 M1 = 3 M2~M31 = 0

程式執行結果截圖

```
Register=======================================================

r0=         0, r1=         3, r2=         4, r3=         1, r4=         6, r5=         2, r6=         7, r7=         1

r8=         1, r9=         0, r10=        3, r11=        0, r12=        0, r13=        0, r14=        0, r15=        0

r16=        0, r17=        0, r18=        0, r19=        0, r20=        0, r21=        0, r22=        0, r23=        0

r24=        0, r25=        0, r26=        0, r27=        0, r28=        0, r29=        0, r30=        0, r31=        0


Memory=========================================================

m0=         0, m1=         3, m2=         0, m3=         0, m4=         0, m5=         0, m6=         0, m7=         0

m8=         0, m9=         0, m10=        0, m11=        0, m12=        0, m13=        0, m14=        0, m15=        0

r16=        0, m17=        0, m18=        0, m19=        0, m20=        0, m21=        0, m22=        0, m23=        0

m24=        0, m25=        0, m26=        0, m27=        0, m28=        0, m29=        0, m30=        0, m31=        0
```

2. Case 2

執行指令	指令執行過程	執行結果 (register)	執行結果 (memory)
(1) begin: (2) addi $1, $2, 4 (3) addi $2, $0, 4 (4) addi $3, $0, 1 (5) sw $1, 4($0) (6) add $4, $1, $1 (7) or $6, $1, $2 (8) and $7, $1, $3 (9) sub $5, $4, $2 (10) slt $8, $1, $2 (11) beq $1, $2, begin (12) lw $10, 4($0)	(1) r1 = r2+4 = 0+4 = 4 (2) r2 = r0+4 = 0+4 =4 (3) r3 = r0+1 = 0+1 =1 (4) Store r1=4 to m1, M1 = 4 (5) r4 = r1+r1 = 4+4 =8 (6) r6 = r1 OR r2 =4 (7) r7 = r1 AND r3 = 0 (8) r5 = r4 – r2 = 4 (9) (r1=4) ≮ (r2=4), r8 = 0 (10) (r1=4) = (r2=4), go to begin (11) r1 = r2+4 = 4+4 = 8 (12) r2 = r0+4 = 0+4 =4 (13) r3 = r0+1 = 0+1 =1 (14) Store r1=8 to m1, M1 = 8 (15) r4 = r1+r1 = 8+8 =16 (16) r6 = r1 OR r2 =12 (17) r7 = r1 AND r3 = 0 (18) r5 = r4 – r2 = 12 (19) (r1=8) ≮ (r2=4), r8 = 0 (20) (r1=8) ≠ (r2=4), no branch (21) Load m1=8 to r10, r10 = 8	R0 = 0 R1 = 8 R2 = 4 R3 = 1 R4 = 16 R5 = 12 R6 = 12 R7 = 0 R8 = 0 R9 = 0 R10 = 8 R11~R31 = 0	M0 = 0 M1 = 8 M2~M31 = 0
程式執行結果截圖			

```
Register==========================================================

r0=         0, r1=         8, r2=         4, r3=         1, r4=        16, r5=        12, r6=        12, r7=         0

r8=         0, r9=         0, r10=        8, r11=        0, r12=        0, r13=        0, r14=        0, r15=         0

r16=        0, r17=        0, r18=        0, r19=        0, r20=        0, r21=        0, r22=        0, r23=         0

r24=        0, r25=        0, r26=        0, r27=        0, r28=        0, r29=        0, r30=        0, r31=         0

Memory============================================================

m0=         0, m1=         8, m2=         0, m3=         0, m4=         0, m5=         0, m6=         0, m7=         0

m8=         0, m9=         0, m10=        0, m11=        0, m12=        0, m13=        0, m14=        0, m15=         0

r16=        0, m17=        0, m18=        0, m19=        0, m20=        0, m21=        0, m22=        0, m23=         0

m24=        0, m25=        0, m26=        0, m27=        0, m28=        0, m29=        0, m30=        0, m31=         0
```

3. Case3

執行指令	指令執行過程	執行結果 (register)	執行結果 (memory)
(1) addi r1, r0, 10 (2) addi r2, r0, 4 (3) NOP (4) NOP (5) NOP (6) NOP (7) slt r3, r1, r2 (8) NOP (9) NOP (10) NOP (11) NOP (12) beq r3, r0, 1 (13) add r4, r1, r2 (14) sub r5, r1, r2	(1) r1=r0+10=0+10=10 (2) r2=r0+4=0+4=4 (3) no operation (4) no operation (5) no operation (6) no operation (7) (r1=10)≮(r2=4), r3=0 (8) no operation (9) no operation (10) no operation (11) no operation (12) (r3=0)=(r0=0) goto PC+4 (13) r5=r1-r2=10-4=6	R0 = 0 R1 = 10 R2 = 4 R3 = 0 R4 = 0 R5 = 6 R6~31 = 0	M1~M31 = 0

程式執行結果截圖

```
Register=============================================================

r0=         0, r1=        10, r2=         4, r3=         0, r4=         0, r5=         6, r6=         0, r7=         0

r8=         0, r9=         0, r10=        0, r11=        0, r12=        0, r13=        0, r14=        0, r15=        0

r16=        0, r17=        0, r18=        0, r19=        0, r20=        0, r21=        0, r22=        0, r23=        0

r24=        0, r25=        0, r26=        0, r27=        0, r28=        0, r29=        0, r30=        0, r31=        0


Memory=============================================================

m0=         0, m1=         0, m2=         0, m3=         0, m4=         0, m5=         0, m6=         0, m7=         0

m8=         0, m9=         0, m10=        0, m11=        0, m12=        0, m13=        0, m14=        0, m15=        0

r16=        0, m17=        0, m18=        0, m19=        0, m20=        0, m21=        0, m22=        0, m23=        0

m24=        0, m25=        0, m26=        0, m27=        0, m28=        0, m29=        0, m30=        0, m31=        0
```

4. Case4

執行指令	指令執行過程	執行結果 (register)	執行結果 (memory)
(1) addi r1, r0, 2 (2) addi r2, r0, 4 (3) NOP (4) NOP (5) NOP (6) NOP (7) slt r3, r1, r2 (8) NOP (9) NOP (10) NOP (11) NOP (12) beq r3, r0, 1 (13) add r4, r1, r2 (14) sub r5, r1, r2	(1) r1=r0+2=0+2=2 (2) r2=r0+4=0+4=4 (3) no operation (4) no operation (5) no operation (6) no operation (7) (r1=2)<(r2=4), r3=1 (8) no operation (9) no operation (10) no operation (11) no operation (12) (r3=1) ≠ (r0=0), no branch (13) r4=r1+r2=2+4=6 (14) r5=r1-r2=2-4=-2	R0 = 0 R1 = 2 R2 = 4 R3 = 1 R4 = 6 R5 = -2 R6~R31 = 0	M1~M31 = 0

程式執行結果截圖

```
Register=============================================================

r0=         0, r1=         2, r2=         4, r3=         1, r4=         6, r5=        -2, r6=         0, r7=         0

r8=         0, r9=         0, r10=        0, r11=        0, r12=        0, r13=        0, r14=        0, r15=        0

r16=        0, r17=        0, r18=        0, r19=        0, r20=        0, r21=        0, r22=        0, r23=        0

r24=        0, r25=        0, r26=        0, r27=        0, r28=        0, r29=        0, r30=        0, r31=        0


Memory=============================================================

m0=         0, m1=         0, m2=         0, m3=         0, m4=         0, m5=         0, m6=         0, m7=         0

m8=         0, m9=         0, m10=        0, m11=        0, m12=        0, m13=        0, m14=        0, m15=        0

r16=        0, m17=        0, m18=        0, m19=        0, m20=        0, m21=        0, m22=        0, m23=        0

m24=        0, m25=        0, m26=        0, m27=        0, m28=        0, m29=        0, m30=        0, m31=        0
```

5. Case5

執行指令	指令執行過程	執行結果 (register)	執行結果 (memory)
(1) addi r6, r0, 2 (2) NOP (3) NOP (4) NOP (5) NOP (6) slti r1, r6, 1	(1) r6=r0+2=0+2=2 (2) no operation (3) no operation (4) no operation (5) no operation (6) (r6=2) ≮ (1), r1=0	R0 = 0 R1 = 0 R2 = 0 R3 = 0 R4 = 0 R5 = 0 R6 = 2 R7~R31 = 0	M1~M31 = 0

程式執行結果截圖

```
Register=============================================================

r0=         0, r1=         0, r2=         0, r3=         0, r4=         0, r5=         0, r6=         2, r7=         0

r8=         0, r9=         0, r10=        0, r11=        0, r12=        0, r13=        0, r14=        0, r15=        0

r16=        0, r17=        0, r18=        0, r19=        0, r20=        0, r21=        0, r22=        0, r23=        0

r24=        0, r25=        0, r26=        0, r27=        0, r28=        0, r29=        0, r30=        0, r31=        0


Memory=============================================================

m0=         0, m1=         0, m2=         0, m3=         0, m4=         0, m5=         0, m6=         0, m7=         0

m8=         0, m9=         0, m10=        0, m11=        0, m12=        0, m13=        0, m14=        0, m15=        0

r16=        0, m17=        0, m18=        0, m19=        0, m20=        0, m21=        0, m22=        0, m23=        0

m24=        0, m25=        0, m26=        0, m27=        0, m28=        0, m29=        0, m30=        0, m31=        0
```

6. Case6

執行指令	指令執行過程	執行結果 (register)	執行結果 (memory)
(1) addi r6, r0, 2 (2) NOP (3) NOP (4) NOP (5) NOP (6) slti r1, r6, 14	(1) r6=r0+2=0+2=2 (2) no operation (3) no operation (4) no operation (5) no operation (6) (r6=2) < (4), r1=1	R0 = 0 R1 = 1 R2 = 0 R3 = 0 R4 = 0 R5 = 0 R6 = 2 R7~R31 = 0	M1~M31 = 0

程式執行結果截圖

```
Register=============================================================

r0=         0, r1=         1, r2=         0, r3=         0, r4=         0, r5=         0, r6=         2, r7=         0

r8=         0, r9=         0, r10=        0, r11=        0, r12=        0, r13=        0, r14=        0, r15=        0

r16=        0, r17=        0, r18=        0, r19=        0, r20=        0, r21=        0, r22=        0, r23=        0

r24=        0, r25=        0, r26=        0, r27=        0, r28=        0, r29=        0, r30=        0, r31=        0


Memory=============================================================

m0=         0, m1=         0, m2=         0, m3=         0, m4=         0, m5=         0, m6=         0, m7=         0

m8=         0, m9=         0, m10=        0, m11=        0, m12=        0, m13=        0, m14=        0, m15=        0

r16=        0, m17=        0, m18=        0, m19=        0, m20=        0, m21=        0, m22=        0, m23=        0

m24=        0, m25=        0, m26=        0, m27=        0, m28=        0, m29=        0, m30=        0, m31=        0
```

Problems you met and solutions:

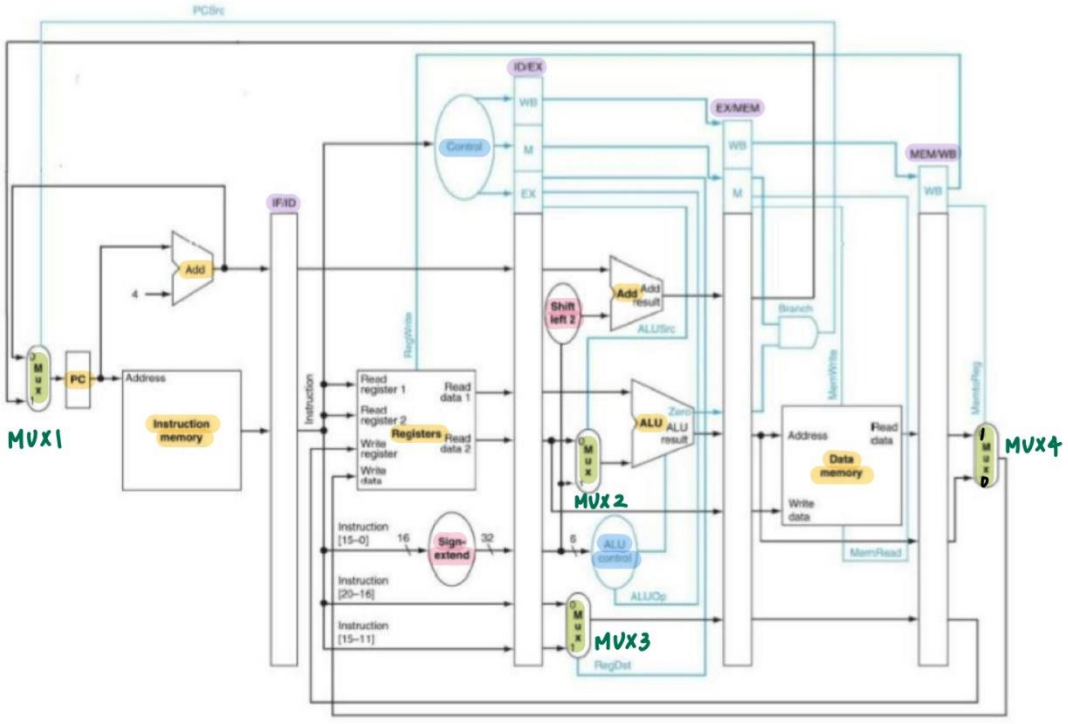
一開始進行 pipeline CPU 實作時沒有考慮到太多的東西，就依照所給的電路圖開始進行程式的實作，而實作出來的程式也可以跑過第一個測資，但當我使用之前的測資來跑時，發現 beq 的 branch 問題。

由於第一個測資中的 beq 是不會進行 branch 的，所以沒有將原本電路圖的問題顯現出來。因為 pipeline CPU 同時會執行多個指令的運算，指令會在每個 clock cycle 不斷丟入 CPU 中，而如果依照原本的電路圖實作，beq 指令需要到 stage3 才會知道是否要 branch，且在 stage4 時才會把 target address 送回 PC，因此會需要修正之後丟入 CPU 的指令。

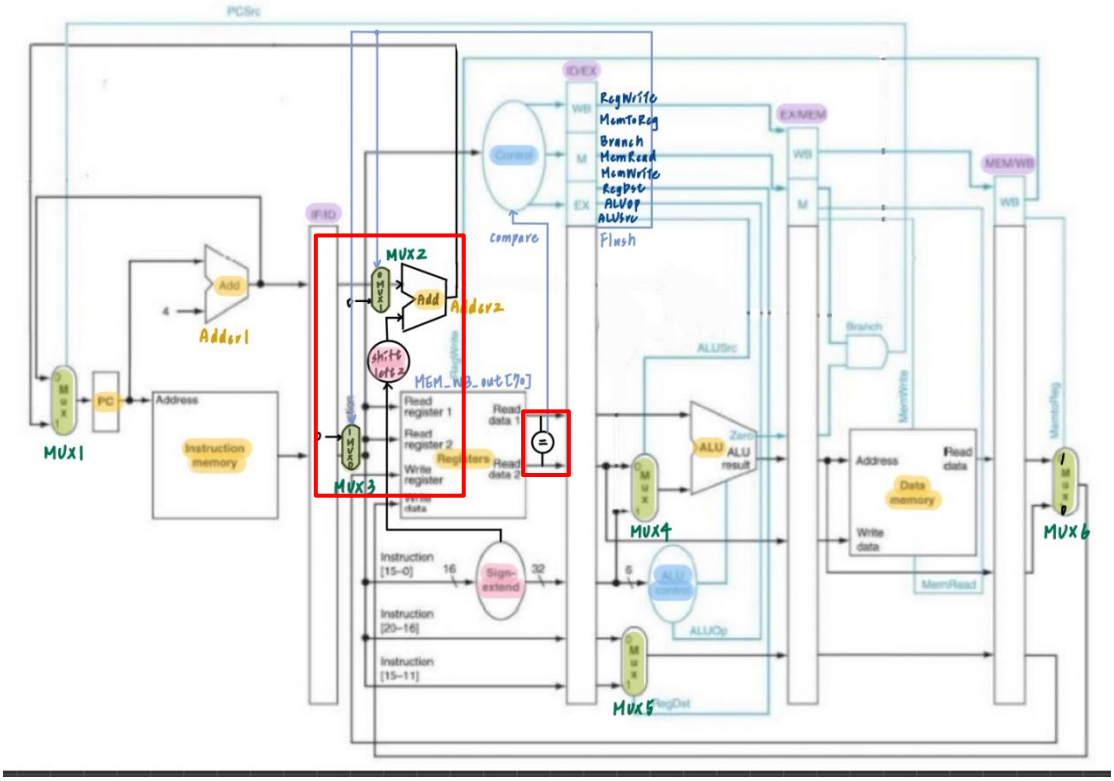
為了增加 pipeline CPU 的效能，我將計算 target address 的 adder 從 stage3 移至 stage2，並新增 comparator，使得在 stage2 就可以知道是否要 branch，再透過 MUX 和 control signal 來 flush 掉錯誤的指令，並重新輸入正確的指令執行。

之後，我將第一個測資修改成需要 branch 的 data，發現修改過後的電路圖仍存在瑕疵，因未考慮到控制 MUX1 的 control signal 的進入時間點，因此發現 branch 到的指令是錯誤的，所以我將控制 MUX1 的 control signal branch AND compare 在 stage2 時就輸入 MUX1，使得下一個 target address 可以在下一個 clock cycle 就進入 PC。

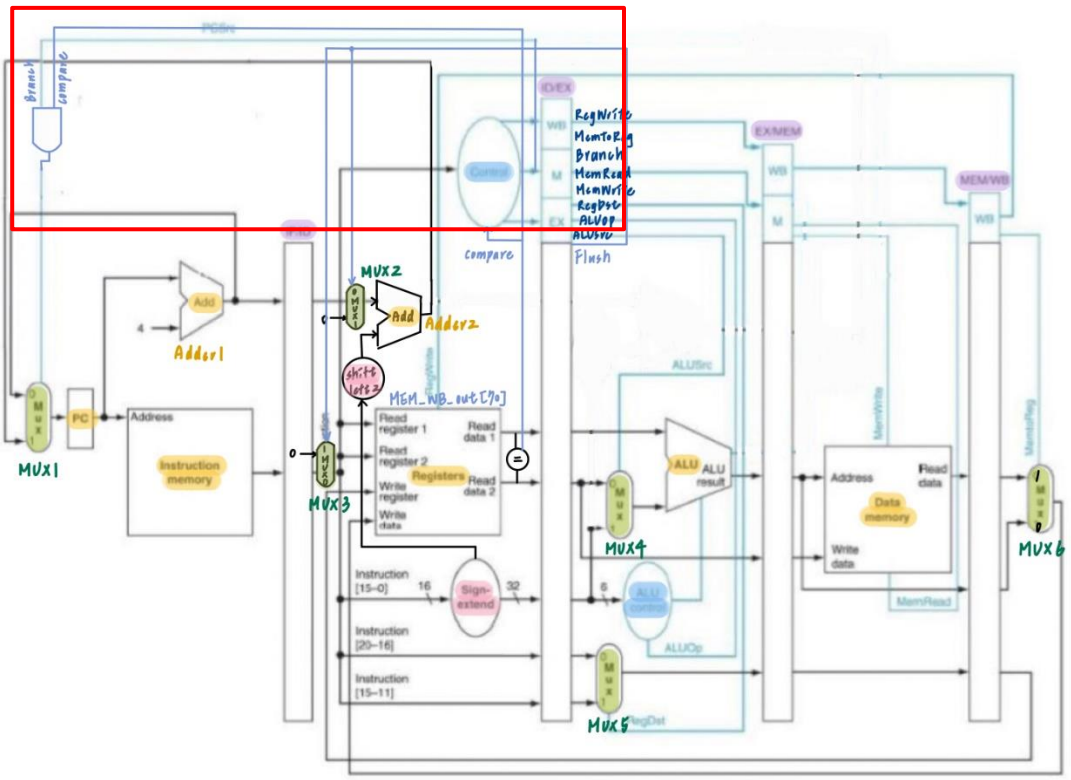
以下為修正前後的 pipeline CPU 電路架構圖:



未修正 pipeline CPU 架構圖



修正 pipeline CPU 電路架構圖-1



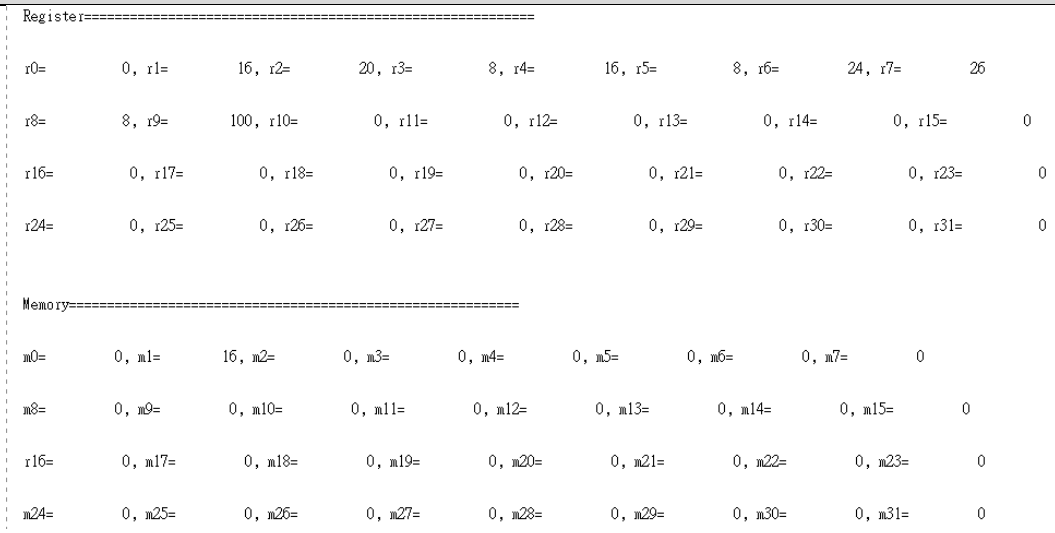
修正 pipeline CPU 電路架構圖-2

Bonus (optional):

因為 pipeline CPU 在每個 stage 會處理不同的指令，所以當兩個指令具有因果關係時，會出現 data dependency。從 pipeline CPU 的架構可知，有 data dependency 的兩個指令，後面的指令需要等前面指令先完成存取之後才能進行運算，所以兩個指令中間至少需要間隔四個指令。以下為原始指令、reorder 指令以及執行結果的整理:

原始指令	Reorder 指令
I1: addi $1, $0, 16 I2: addi $2, $1, 4 I3: addi $3, $0, 8 I4: sw $1, 4($0) I5: lw $4, 4($0) I6: sub $5, $4, $3 I7: add $6, $3, $1 I8: addi, $7, $1, 10 I9: and $8, $7, $3 I10: addi $9, $0, 100	I1: addi $1, $0, 16 I3: addi $3, $0, 8 NOP I4: sw $1, 4($0) I5: lw $4, 4($0) I2: addi $2, $1, 4 I8: addi, $7, $1, 10 I7: add $6, $3, $1 I6: sub $5, $4, $3 I10: addi $9, $0, 100 I9: and $8, $7, $3
執行結果 (register)	執行結果 (memory)
R0 = 0, R1 = 16, R2 = 20, R3 = 8, R4 = 16, R5 = 8, R6 = 24, R7 = 26, R8 = 8, R9 = 100, R10~R31 = 0	M0 = 0, M1 = 16 M2~M31 = 0

程式執行結果截圖



```
Register=======================================================

r0=  0, r1=  16, r2=  20, r3=  8, r4=  16, r5=  8, r6=  24, r7=  26

r8=  8, r9=  100, r10=  0, r11=  0, r12=  0, r13=  0, r14=  0, r15=  0

r16=  0, r17=  0, r18=  0, r19=  0, r20=  0, r21=  0, r22=  0, r23=  0

r24=  0, r25=  0, r26=  0, r27=  0, r28=  0, r29=  0, r30=  0, r31=  0


Memory=========================================================

m0=  0, m1=  16, m2=  0, m3=  0, m4=  0, m5=  0, m6=  0, m7=  0

m8=  0, m9=  0, m10=  0, m11=  0, m12=  0, m13=  0, m14=  0, m15=  0

r16=  0, m17=  0, m18=  0, m19=  0, m20=  0, m21=  0, m22=  0, m23=  0

m24=  0, m25=  0, m26=  0, m27=  0, m28=  0, m29=  0, m30=  0, m31=  0
```

Summary:

透過這次 lab 可以更加認識 pipeline CPU 各個指令的運作方式，並且以 lab3 為基礎，進行電路結構上的調整，使得 CPU 可以同時執行多個指令，增加 CPU 的效能。這次的 pipeline CPU 需要考慮較多的情況，像是 beq 不 branch 的電路處理等等，需要花較久的時間在觀念上的釐清，才能找到更好的解決或實作方案。