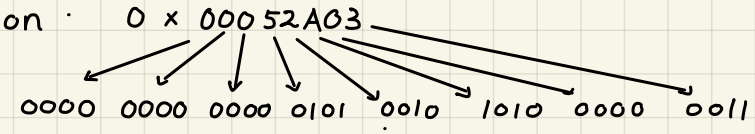
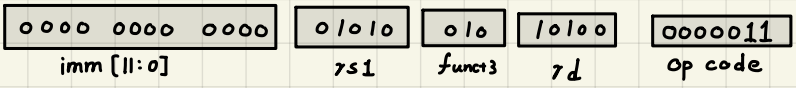
10811235
陳昭維
電機 23

1-1

instruction : 0 x 00052A03





From opcode and funct 3, this is a load word instruction.

This specific instruction loads content of the address which is stored in register x10 to the destination register x20

⇒ lw x20 0(x10)

1-2

the new PC is now at address 0 x 4000 AC0d + 0 x 4 = 0x 4000 AC 11

1-3

There is writeback : Regwrite = 1

The ALU uses register data 1 and immediate : ALUSrc = 1.

There is no branching : Branch = 0

There is no store operation : MemWrite = 0

There is information loaded from Data Memory : MemRead = 1

There is information from Data memory written back : Mem to Reg = 1

1-4

the input of the ALU is Reg [x10] and immediate 0 x 0.

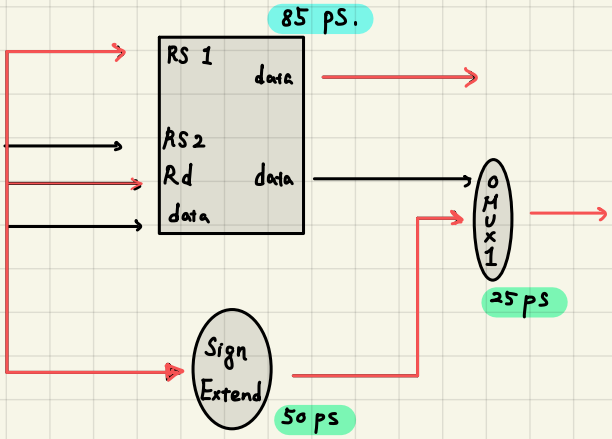
2-1 (R-type)

- PC needs to be read : 20 ps.
- The PC write doesn't matter, since it is a parallel operation, will end before next cycle
- The instruction Memory need accessing and decoding : 250 ps. + 50 ps = 300 ps
- Register read : 85 ps.
- multiplexer select read data 2 as ALU's second input : 25 ps.
- Operation go through ALU : 200 ps
- Multiplexer selects the result from ALU to write back : 25 ps
- Register write : 85 ps.

20 + 300 + 85 + 25 + 200 + 25 + 85 = 740 ps.

2-2 load

- PC needs to be read . 20 ps.
- The PC write doesn't matter, since it is a parallel operation, will end before next cycle
- The instruction Memory need accessing and decoding : 250 ps. + 50 ps = 300 ps



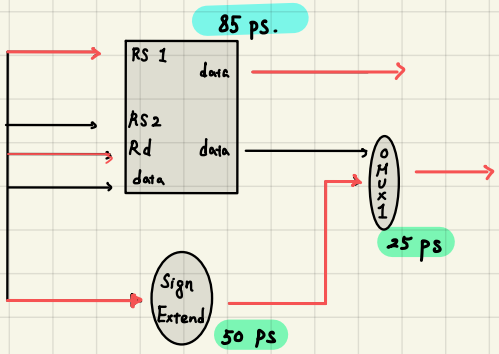
- Register read . 85 ps
- Sign Extension + MUX : 75 ps.

- Operation go through ALU : 200 ps
- Data memory : 250 ps
- Multiplexer selects read data from D-MEM to write back : 25 ps
- Register write : 85 ps

20 + 300 + 85 + 200 + 250 + 25 + 85 = 965 ps

2-3 store

- PC needs to be read . 20 ps.
- The PC write doesn't matter, since it is a parallel operation, will end before next cycle
- The instruction Memory need accessing and decoding : 250 ps. + 50 ps = 300 ps



Register read : 85 ps

Sign Extension + MUX = 50ps + 25 ps = 75 ps

- Operation go through ALU : 200 ps
- Data memory : 250 ps

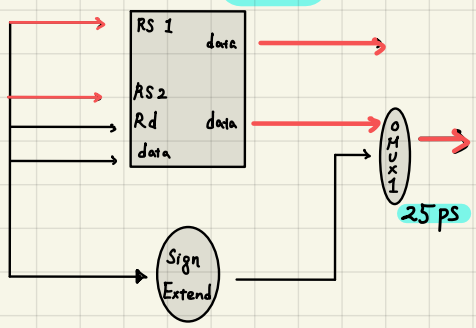
$$20 + 300 + 85 + 200 + 250 = 855\,ps$$

2-4 beq

- PC needs to be read : 20 ps.
- The Instruction Memory needs accessing & decoding : 250 ps. + 50 ps = 300 ps

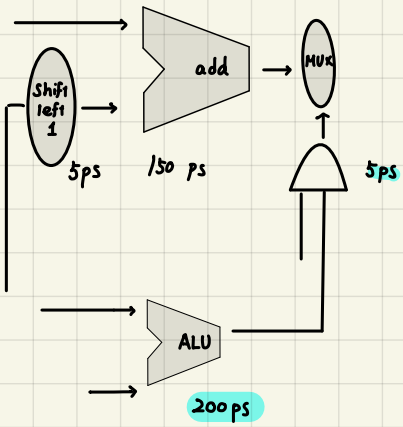
---

- 85 ps



Register Read : 85 ps

Multiplexer : 25 ps

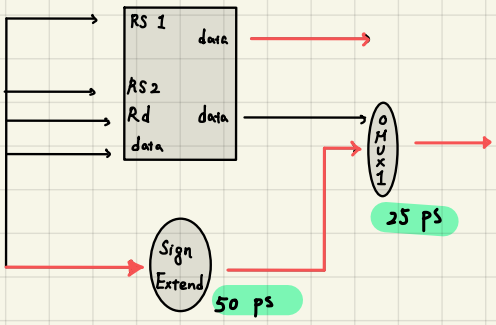


- Multiplexer 25 ps
- PC write : 50 ps

$$20 + 300 + 85 + 25 + 200 + 5 + 25 + 50 = 710 \text{ ps}$$

2-5 I-type

- PC needs to be read . 20 ps.
- The PC write doesn't matter, since it is a parallel operation, will end before next cycle
- The instruction Memory need accessing and decoding : 250 ps. + 50 ps = 300 ps



Register read . 85 ps

Sign Extension + MUX = 50 ps + 25 ps = 75 ps

- Operation go through ALU : 200 ps
- multiplexer : 25 ps.
- Register write : 85 ps

$$20 + 300 + 85 + 200 + 25 + 85 = 715 \text{ ps}$$

2-6.

The clock period is the longest instruction time, which is 965 ps.

2-7

R-type :	740 ps	load	965 ps	store :	855 ps.
I-type :	715 ps.	beq	: 710 ps		

the minimum cycle period has to be larger than the longest latency block.

⇒ 250 ps at minimum

⇒ latency of writing flipfop 50 ps. reading flipflop 20 ps.

So consider a single clock period of 20 + 250 + 50 + 85 + 50 = 455

where the PC read, Instruction memory read, decoding, Register read and temporary register write is in the first stage and then temporary register read, mux, ALU, mux, Register write.

20 + 25 + 200 + 25 + 85 = 355

where the R type, I type and beq can be finished in 2 cycles and load and store can be finished in 3 cycles

455 × 2 × 52% + 455×2 × 12% + 455 × 3 × 36%
= 1073.8 ps.

After some few tries, the fastest performance is 740 ps clock period, the reason is that shorter cycle is designed, more flip-flops has to be inserted into the pathway, thus latency needs consideration

for clock period 740 ps

the R-type, I-type, beq can be finished in 1 clock period

the load and store can be finished in 2 clock period

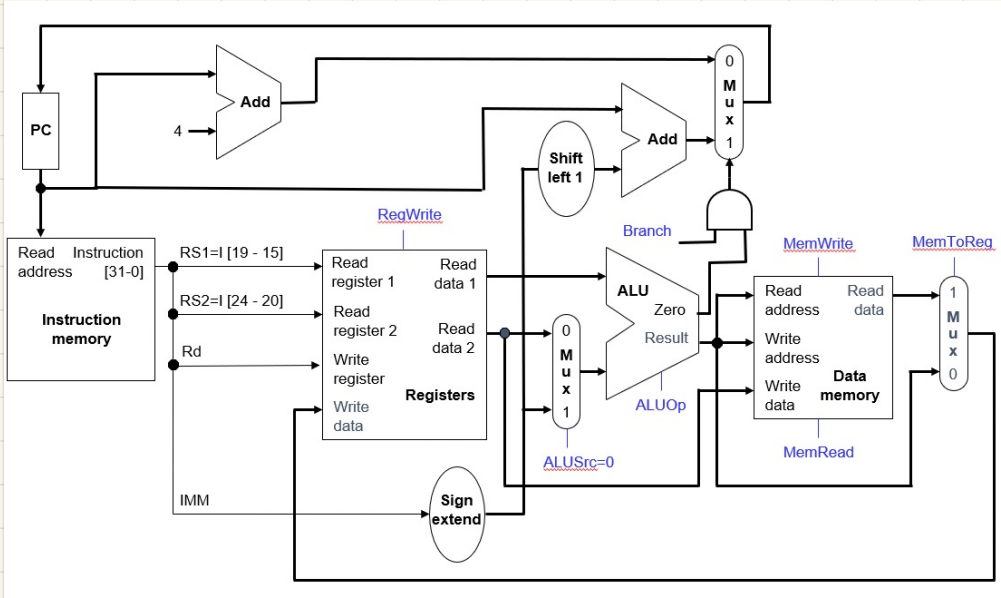
740 × (52% + 12%) + (740 × 2) × (36%)
= 1006.4 ps ⇒ still slower than 965 ps the sychronous timing for slowest instruction

⇒ For the temporary block latency makes it impossible for multicycle design in this case.

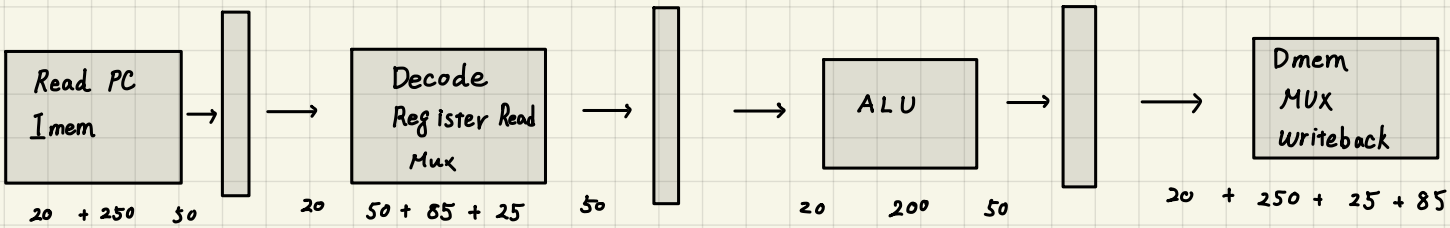
2-8

8. (10 points)[section 4.5] Please design a pipelined processor based on the latency of blocks in the above table.
The design constraints are the same as above multi-cycle processor.
Please also calculate the theoretical performance speedup by the pipelined processor.

I-Mem/D-Mem	Register Read	Register Write	PC Read	PC Write	Mux	ALU	Adder	Single gate/shift	Sign extension	Decoder Contrl
250ps	85ps	85ps	20ps	50ps	25ps	200ps	150ps	5ps	50ps	50ps



Considered a four stage pipeline.



20 + 250 + 50 = 320 ps

20 + 50 + 85 + 25 = 230 ps

50

20 + 200 + 50 = 270 ps

20 + 250 + 25 + 85 = 380 ps

⇒ each stage is 380 ps.

Theoretical speed up : $\frac{965\ N}{380 \times 4 + (N-1) \times 380}$

for large N ⇒ $\lim_{N \to \infty} \frac{965\ N}{380 \times 4 + (N-1) \times 380} = \frac{965}{380} \approx 2.6$

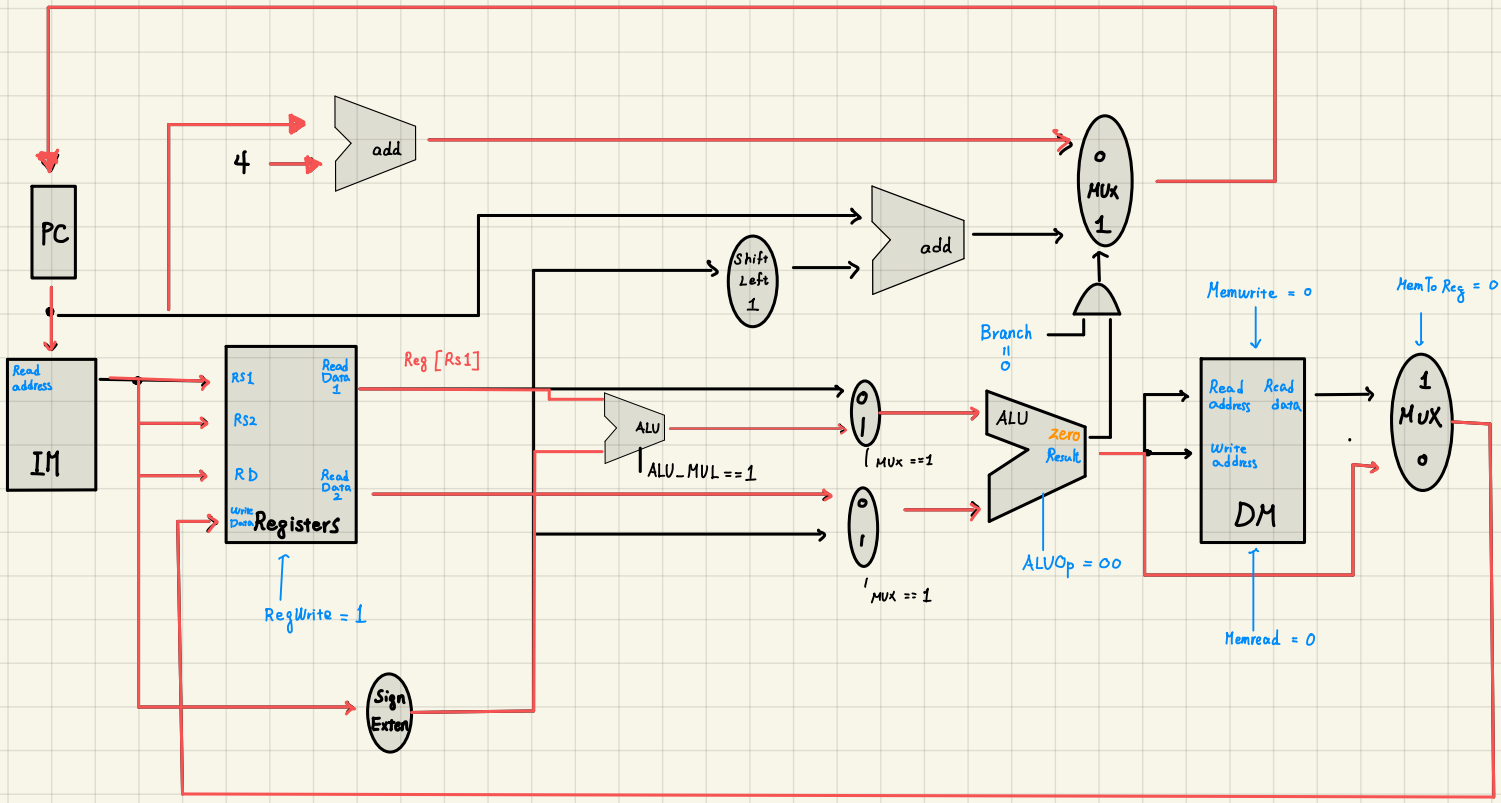
2.6 times faster

3-1

3. (50 points) [section 4.4 and 4.5] For the following problems, you may use data path files in pptx to draw the new ones.
   1. (20 points) Please draw a new data path in the single-cycle processor to support a new maci instruction: maci rd, rs1, rs2, imm.
      The function of the instruction can be represented as follows: Reg[rd]=Reg[rs2]+Reg[rs1]ximm

   2. (10 points) With the block latency table in Problem 2, please calculate the latency for the new instruction.

   3. (20 points) How do the pipeline design in Problem 2 change by including the new instruction?



3-2

PC read : 20
Instruction memory : 250
Decoding : 50

Register read : 85

ALU for reg[rs1] × imm : 200

Mux : 25

ALU for reg [rs2] + reg [rs1] × imm : 200

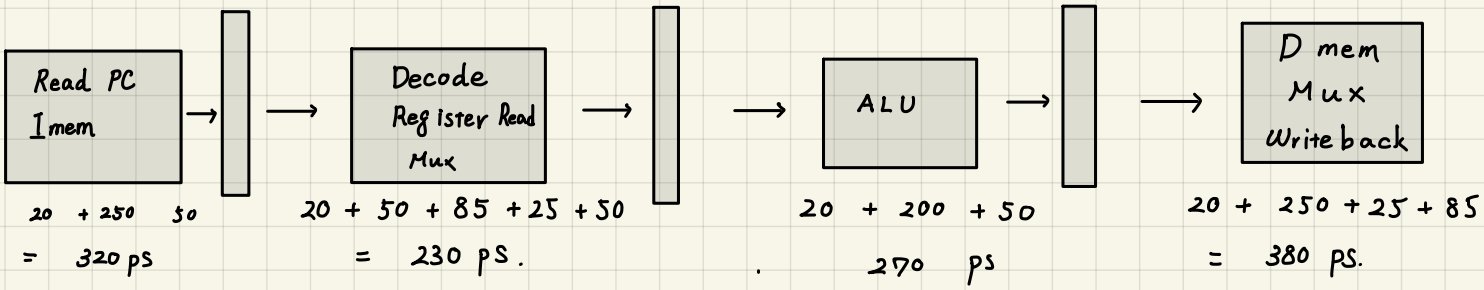
MUX : 25
register write back : 85

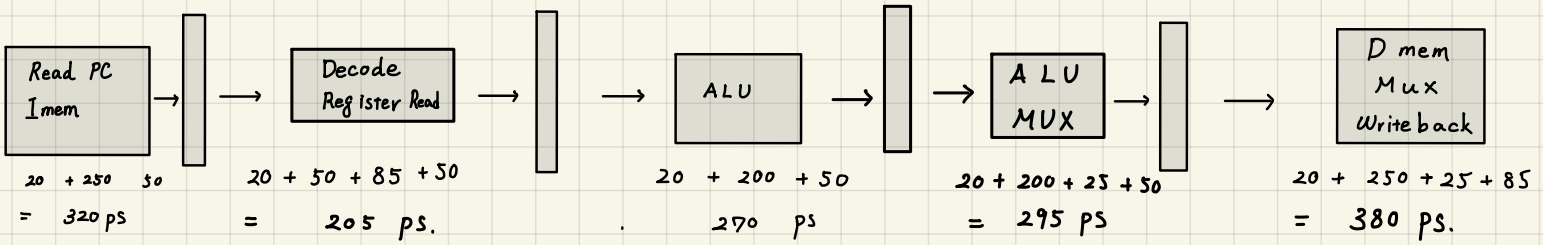
20 + 250 + 50 + 85 + 200 + 25 + 200 + 25 + 85

= 940 ps

3-3

An additional ALU is inserted for Imm x Reg[rs1]





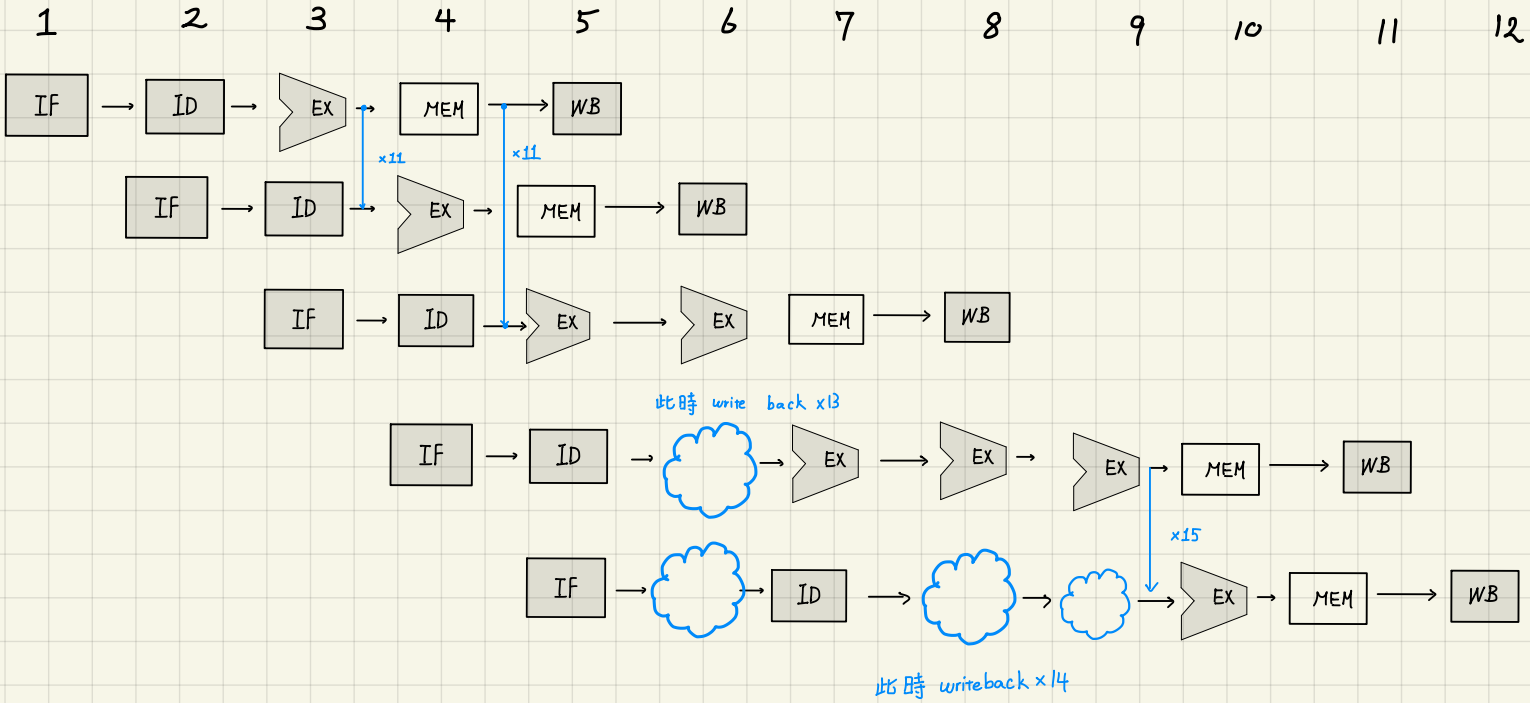
4.

4. (20 points) [section 4.5]
Assume we have designed a ALU for a 5-stage RV32I RISC-V core to support the following instructions (in additional to RV32I):
All RV32I instructions use 1 clock cycle ALU. This is a multi-cycle pipelined processor that mul, div and sqrt need extra cycles to complete.

Instruction	Clock cycle	Format	Function
mul	2	mul rd, rs1, rs2	Multiplication rd=rs1xrs2
div	3	div rd, rs1, rs2	Division rd=rs1/rs2

Please draw the pipeline diagram for the following assembly program. Assume full forwarding support.
What is the total execution cycle?

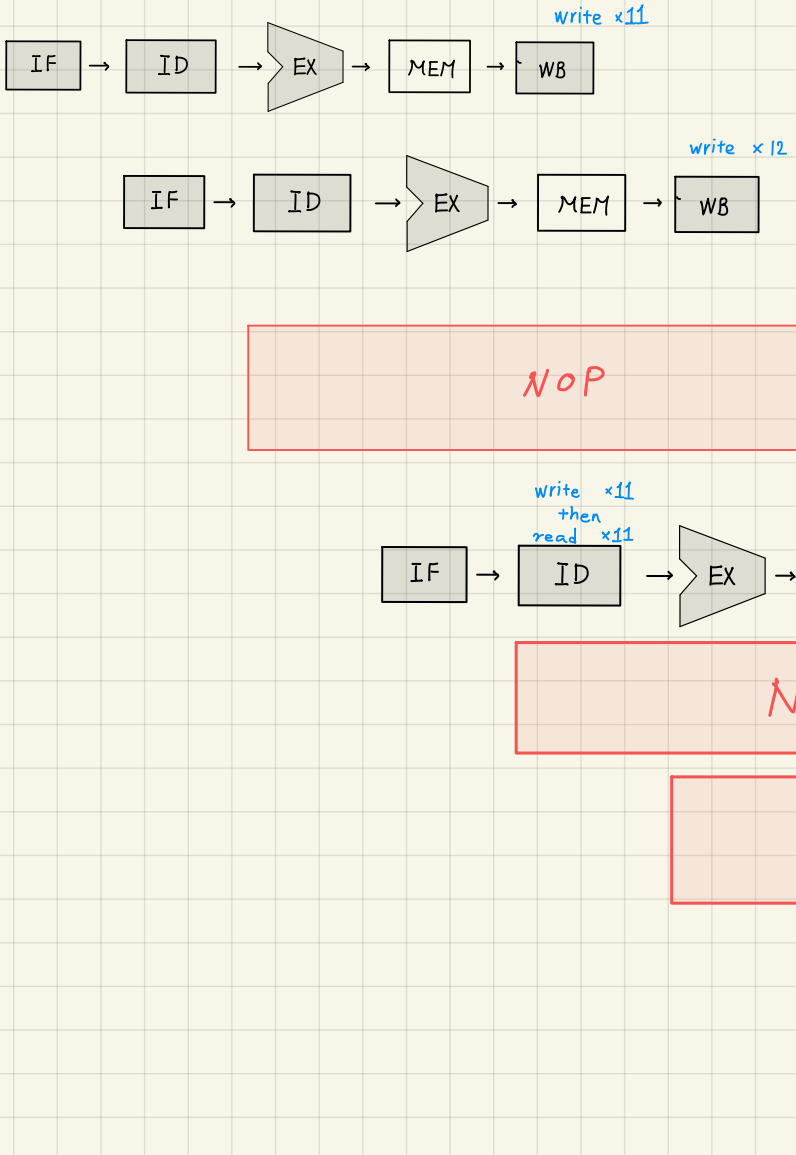
```
addi x11, x12, 5
add x13, x11, x12
mul x14, x11, 15
div x15, x13, x12
add x15, x14, x15
```



A: 12 cycles is needed.

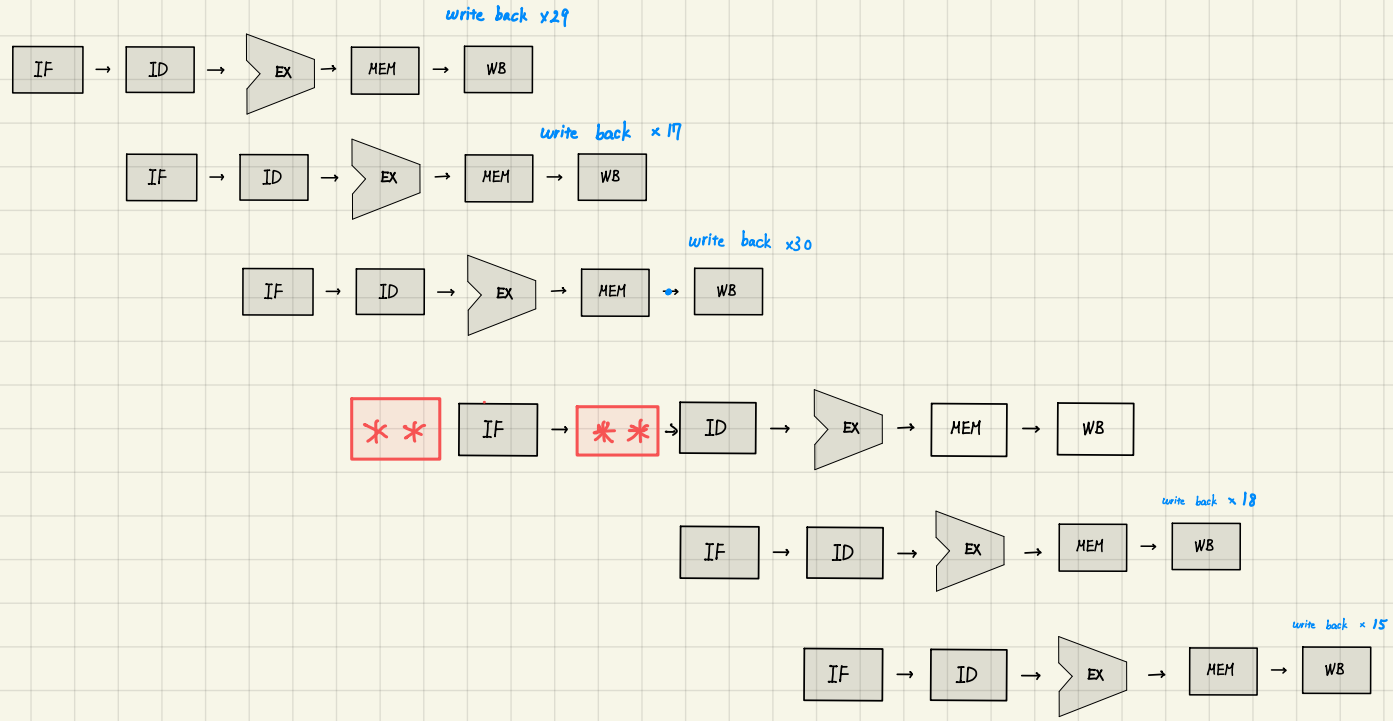
5. (20 points) [section 4.5]
Please add `nop` instruction to the following codes such that the results will be correct for a pipeline processor without forwarding support.
Please draw the pipeline diagram for the 5-stage pipelined design.

```
addi x11, x9, 5
add  x12, x10, x10
addi x13, x11, 15
add  x14, x13, x12
```



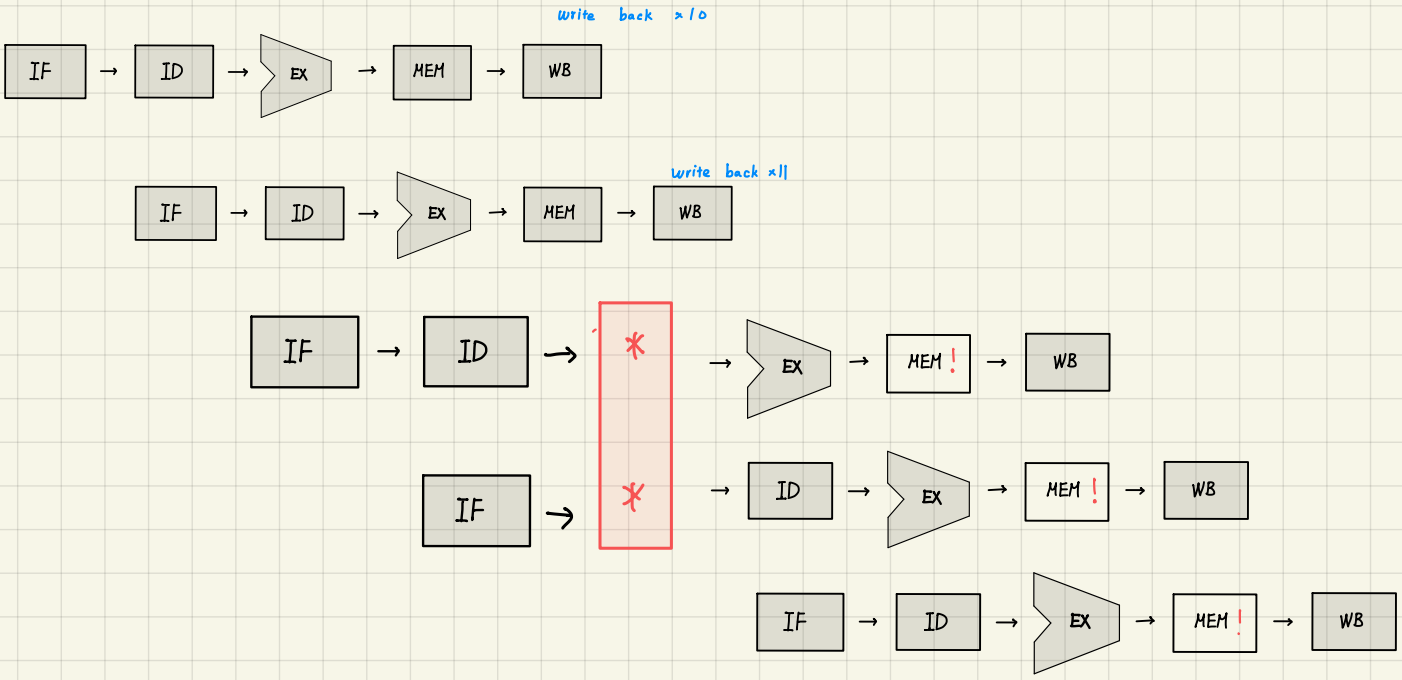
6. (10 points) [section 4.5]
Assume we have only one memory for both instruction and data. Please draw the pipeline diagram (5-stage processor and stages are IF, ID, EX, MEM, WB) for the following codes.
Please specify the stall cycle by `**`. Note that for this processor with structural hazard, IF will have to stall for MEM if the instruction actually access the memory for data.

```
ld  x29, 8(x10)
sub x17, x15, x14
ld  x30, 12(x10)
beq x29, x30, L1
add x18, x16, x17
sub x15,x30,x17
```



7. (20 points) [section 4.7]
   Please draw the pipeline diagram (5-stage processor) for the following codes.
   The processor has a full forwarding support.
   Please specify the stall cycle by `**`. Also mark the stage without useful work with `!`.

   ```
   ld x10, 0(x13)
   ld x11, 8(x13)
   add x12, x10, x11
   sub x14, x10, x11
   addi x13, x13, -16
   ```

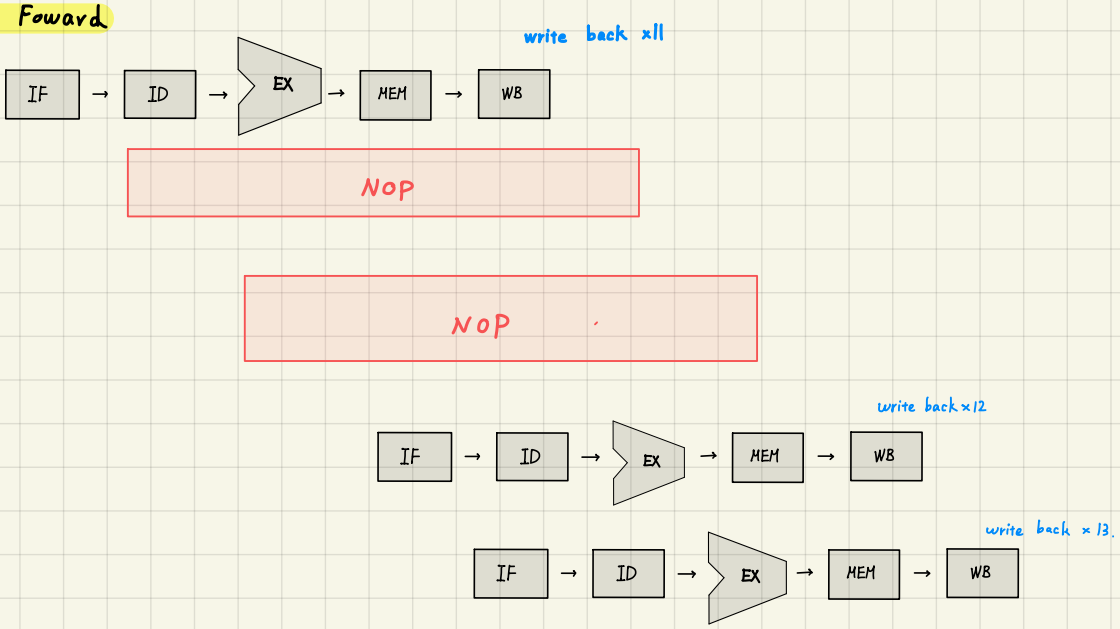


8. (20 points) [section 4.7]
For all the following code segments, please fix the data dependency with `nop` for the following 4 different processors:
(1) Add `nop` so that for processors without forwarding , the execution results are correct.
(2) Add `nop` so that for processors with forwarding from EX/MEM only, the execution results are correct.
(3) Add `nop` so that for processors with forwarding from MEM/WB only, the execution results are correct.
(4) Add `nop` so that for processors with full forwarding (both EX/MEM and MEM/WB), the execution results are correct.

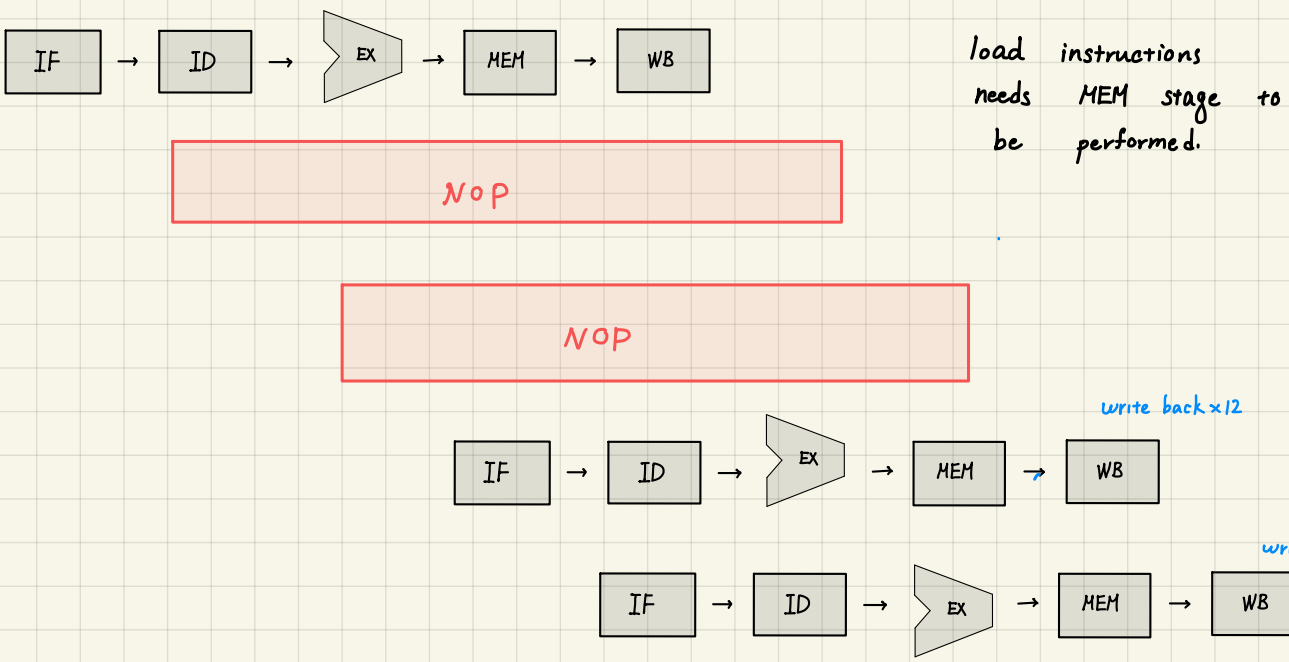
Case with dependency from data memory to the following 1st instruction.

```
ld x11, 0(x10)
addi x12, x11, 4
addi x13, x11, 8
```

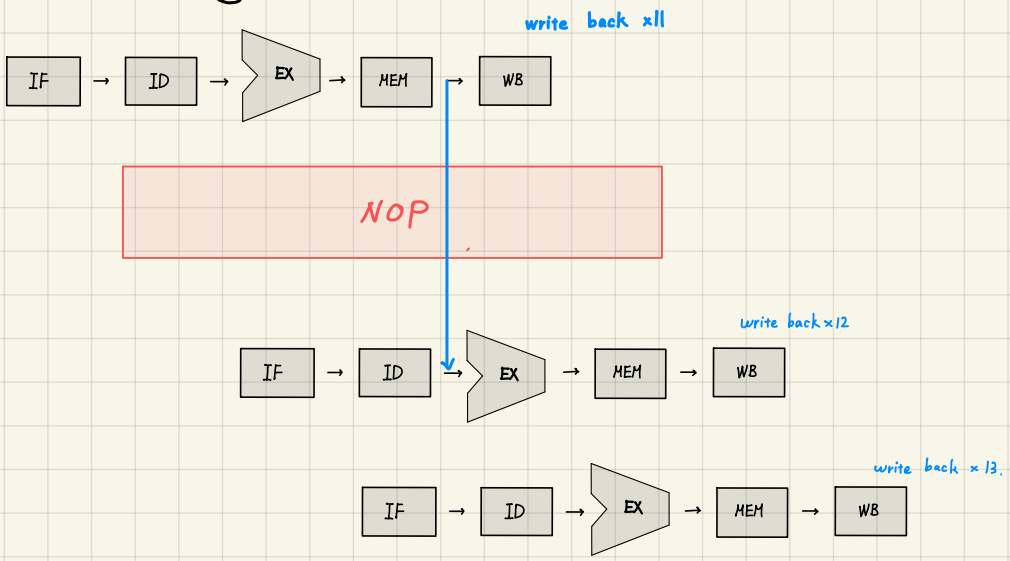
(1) No Foward



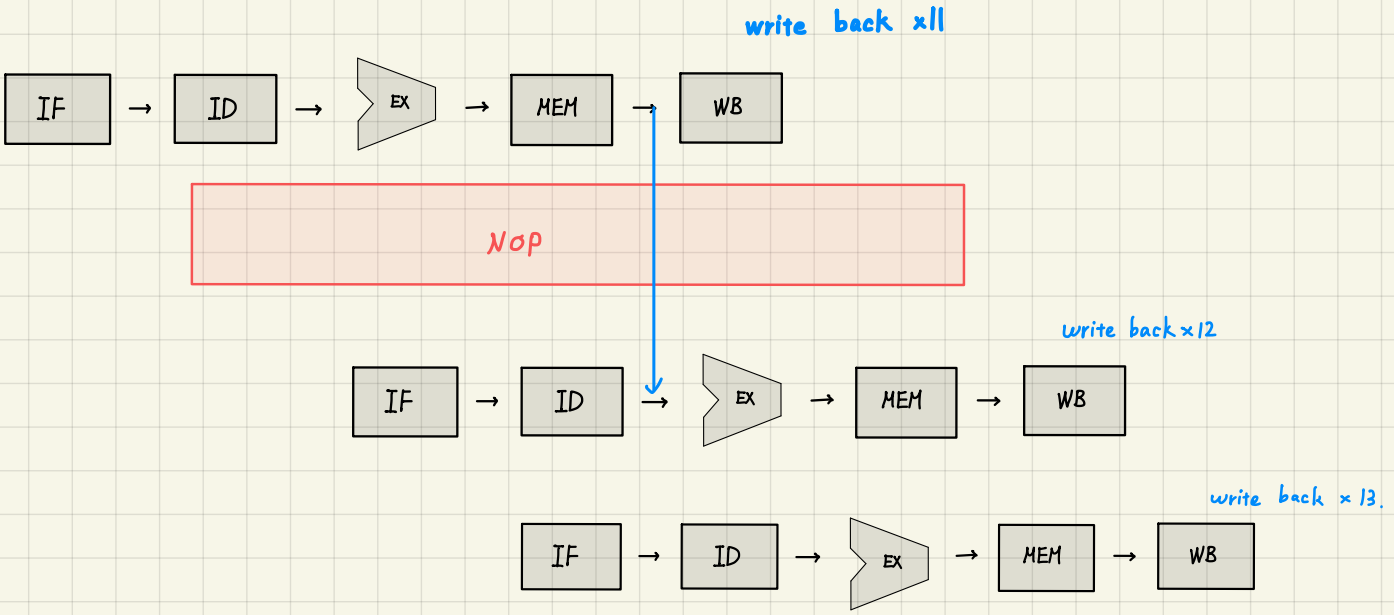
(2) EX/MEM ONLY



(3) MEM / WB Only



(4) ALL Forward



9. (20 points) [section 4.7]
For the following code segment, please list variables to check and generate the ForwardA and ForwardB selection conditions,
and also stall conditions.
For example, x10 is written by 1st add instruction and read by 2nd ld instruction. When "addi" is at MEM stage, we know that RegWrite==1,
and EX/MEM.rd == ID/EX.RS1 == x10, such that ForwardA=2 (forwarding from EX/MEM.ALU_Result to 1st input of ALU).

```
addi x10, x11, 4
ld   x12, 4(x10)
addi x13, x12, 8
```