CSSE 232 – Computer Architecture I
Rose-Hulman Institute of Technology
Computer Science and Software Engineering Department

Exam 2

Name: PARTIAL KEY Section: 1 2 3 4 5

This exam is **closed book**. You are allowed to use the reference card from the book and one 8.5" × 11" single sided page of hand written notes. You may not use a computer, phone, etc. during the examination.

Write all answers on these pages. Be sure to **show all work** and document your code. Do not use instructions that we have not covered (e.g. no `mul` or `div` but you can use instructions like `slli`, `srl`, etc).

RISC-V code is judged both by its correctness and its efficiency. Unless otherwise stated, you may not use RISC-V pseudoinstructions when writing RISC-V code.

For Pass/Fail problems there will be a redo opportunity for partial credit on a future date. You must submit a good faith effort to qualify for the redo opportunity.

Question	Points	Score
Problem 1	15	
Problem 2	16	
Problem 3	15	
Problem 4	15	
Problem 5	15	
Problem 6	24	
Total:	100	

Problem 1. (15 points) You are the lead designer for a **multi-cycle** implementation of a new RISC-V-like processor. Your team wants an R-type instruction called `decrement and peek memory` or `dpm`. This instruction reads memory at the address specified by `rs1` then subtracts the value of the register specified by `rs2` from the memory output. The result is stored back into memory at the same address and into the register specified by `rd`. The code below demonstrates the new instruction:

```
addi t0, x0, 5
li t1, 0x0100 0000
sw t0, 0(t1)
addi t2, x0, 2
dpm t3, t1, t2
```

After the `dpm` instruction is executed both the `t3` register and memory at location `0x0100 0000` will contain the number 3.

Solution:

Standard MIPS Fetch
Standard MIPS Decode
MDR = Mem[A]
ALUout = MDR - B
Mem[A] = ALUout Reg[IR[11:7]] = ALUout

Modify the Register Transfer Language (RTL) shown in the following table to include the new instruction. You should try and design your RTL such that it would make as few hardware changes to the datapath as possible. While this is a `RISC-V`-like processor, you may need to make major changes to the RTL. **Be sure to make it clear how your solution works**.

R-Type	lw/sw	Branch	New Instruction
IR <= Memory[PC] PC <= PC + 4			
A <= Reg[IR[19:15]] B <= Reg[IR[24:20]] ALUOut <= PC + immediate			
ALUOut <= A op B	ALUOut <= A + immediate	if (A == B) PC <= ALUOut	
Reg[ID[11:7]] <= ALUOut	lw: MDR <= Memory[ALUOut] sw: Memory[ALUOut] <= B		
	lw: Reg[IR[11:7]] <= MDR		

Problem 2. The RTL below describes the operation of a new instruction designed by your team.

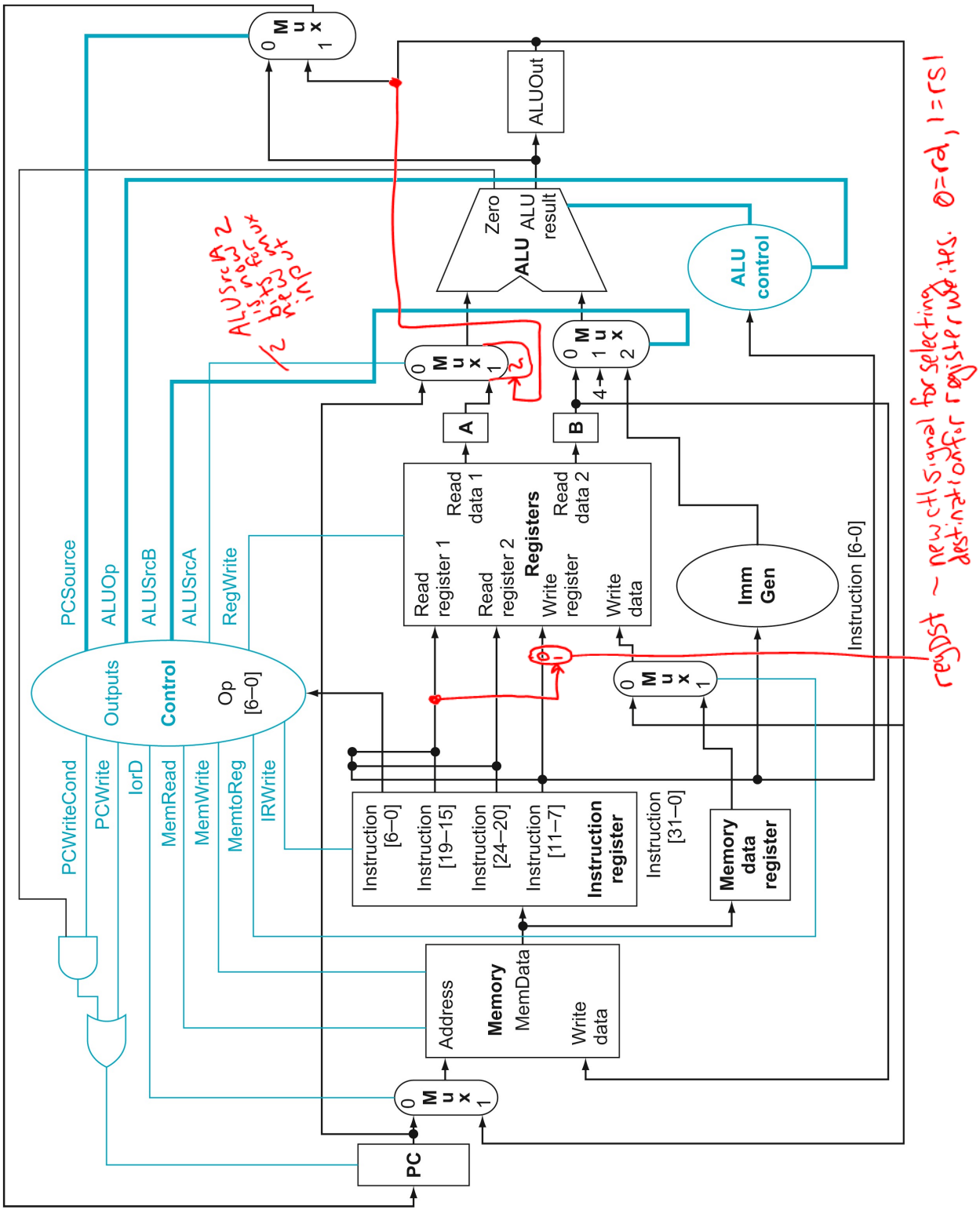
Standard RISC-V Fetch
Standard RISC-V Decode
ALUout = A + SE(IR[31:20])
MDR = Mem[ALUout] ALUout = ALUout + 4
Reg[IR[19:15]] = MDR MDR = Mem[ALUout]
Reg[IR[11:7]] = MDR

Solution: double load word, reads two adjacent words in memory stores them into rs1 and rd.

(a) (4 points) Give this instruction a name and write a brief english description of what it does so that an assembly programmer would understand how to use it (e.g. do not assume the programmer knows the RTL).

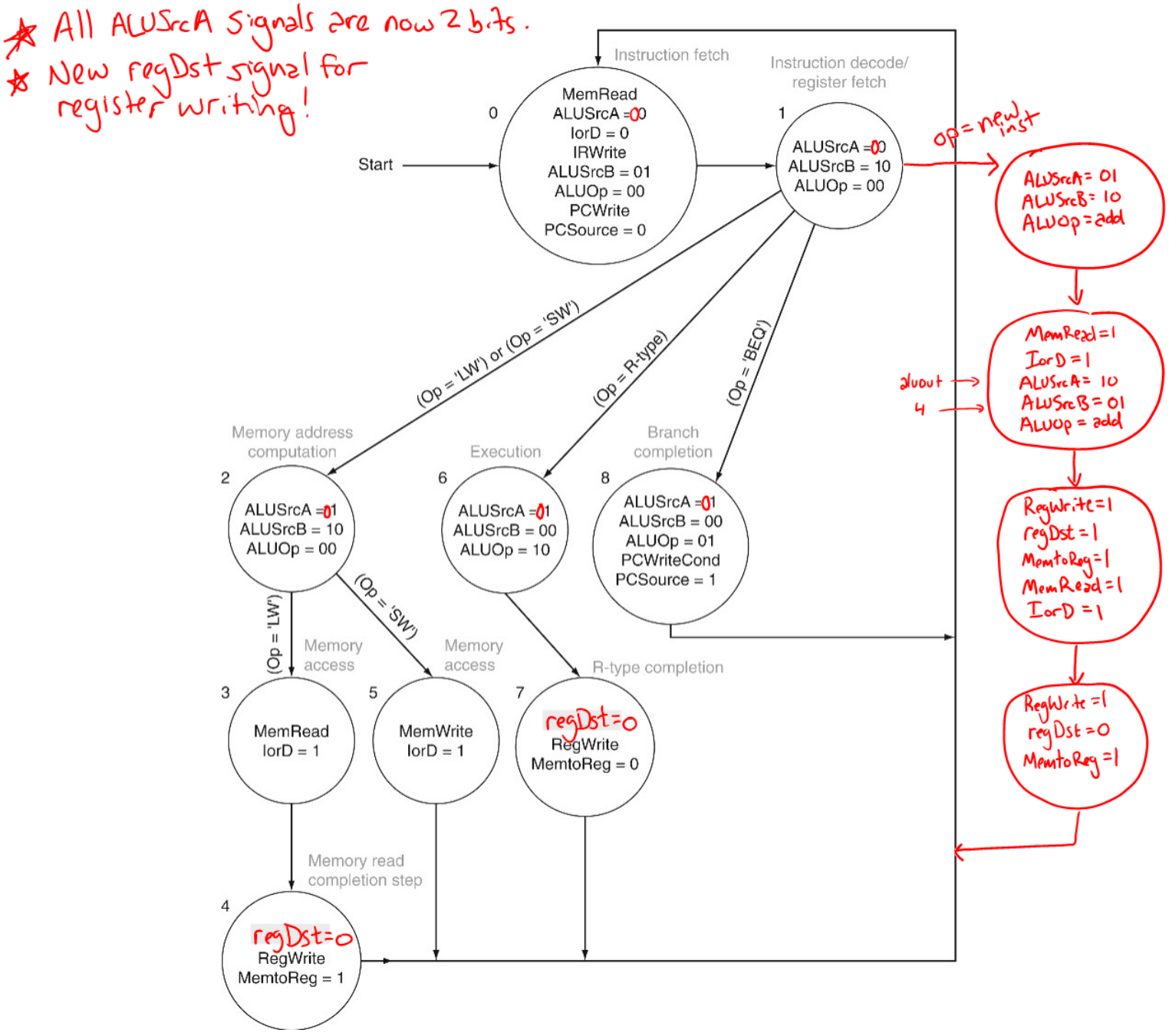
(Continued on next page ...)

(b) (12 points) Modify the multicycle datapath below as necessary to support the new instruction. You should add as little to the datapath as possible. List any additional control signals and their purpose. Be sure it is clear how your changes work.



Problem 3. (15 points) Modify the Multicycle Control Finite State Diagram below as necessary to support the new instruction from **Problem 2**. Be sure these modifications are consistent with the datapath modifications you made in **Problem 2**. In addition, consider the impact upon any existing control states below.

Note: You can simply write `'add'`, `'sub'`, etc. for `ALUOp` values.



Problem 4. (15 points) Adding the instruction from **Problem 2** to the to the RISC-V pipelined datapath we discussed in class is difficult. Three possible approaches to adding this instruction are:

1. Add hardware to existing pipeline stages.
2. Add new pipeline stage(s).
3. Modify control to reuse the existing stages multiple times for this instruction, and stall later instructions.

Select two of these solutions and list pros and cons for each.

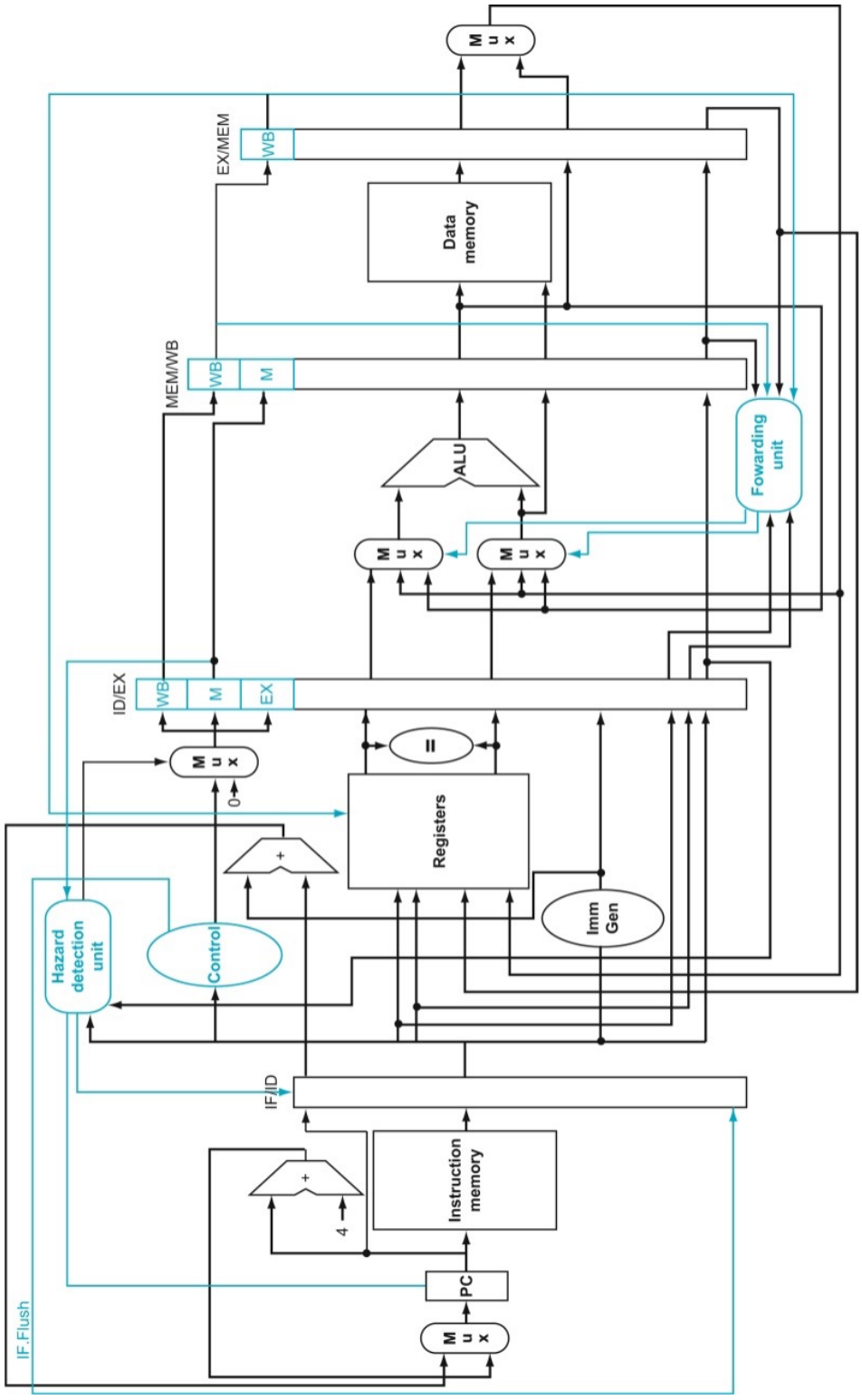
(Note: "it does what the instruction is supposed to do" and "it doesn't break other instructions" are not 'pros' of a design, they are implicit requirements.) The picture on the next page is for your reference. You do not need to modify it.

There are many valid answers to this question. The difficulty comes from the order of steps in this instruction that do not match up with our 5-stage pipeline.

Normal: F → D → X → M → W

new inst.: F → D → X → M + X → W + M

Your answer will explain the benefits (pros) and drawbacks (cons) of two approaches to adding this instruction.


M u x
EX/MEM
WB
Data memory
MEM/WB
WB
M
ALU
Fowarding unit
M u x
M u x
ID/EX
WB
M
EX
M u x
0
=
Registers
+
Hazard detection unit
Control
Imm Gen
IF/ID
+
4
Instruction memory
PC
M u x
IF.Flush

Problem 5. Consider the following piece RISC-V code that makes a system call to the operating system. A system call is an exception that is initiated by the user, which causes the operating system to perform a service on behalf of that user. To make a system call, the user has to put the system call number (predefined by the operating system) into the a7 register, and then issue the ecall instruction.

```
li    a7, 5              # load the system call number 5 into a7
ecall                    # issue the ecall instruction
addi  t0, a0, 0          # save the return value into t0
beq   t0, x0, DONE       # branch to DONE if equal to 0
jal   x0, ERROR          # jump to ERROR otherwise
...
```

The ecall instruction **acts in the same way as an exception.** It causes the CPU to transfer control to the operating system exception handler, which will then determine it is a system call and execute the requested service on behalf of the user.

(a) (5 points) Assume this code runs on a single-cycle RISC-V processor. Describe the steps taken by the processor when the ecall instruction is executed.

Solution:

1. Save the address of ecall into the sepc register.
2. Save the cause of the exception into the scause register.
3. Jump to the address saved into the stvec register by loading it directly into PC.

(b) (2 points) As part of the exception handler, the operating system executes the following C code:

```
p->trapframe->sepc += 4;
```

Why is the operating system incrementing the value stored in the sepc register?

Solution: To cause the `eret` instruction to jump back to the next instruction after the `ecall`.

(c) Consider now that we are dealing with pipelined `RISC-V` processor with a five-stage pipeline similar to the one we discussed in class.

i. (3 points) At which stage of the pipeline would the processor realize the it is executing an `ecall` instruction? *Explain your reasoning.*

Solution: Start of `EX` stage or end of `D` stage.

ii. (5 points) When the processor realizes that it is supposed to jump to the operating system, the pipeline will contain instructions that have already been fetched. Why is that a problem? Describe what the processor does to handle this particular hazard.

Solution: Because they must not alter the state of the processor. So the processor will flush those instructions by replacing their control bits with 0 and flushing the ID/EX register.

Problem 6. (24 points) (Pass/Fail)

Consider the following code running on a RISC-V processor with an advanced 5-stage pipeline that resolves hazards with forwarding wherever possible and stalls whenever necessary.

```
L: lw   t1, 0(a0)
   add  t2, a1, a2
   beq  t1, t2, L
   addi a0, a0, 4
```

BDS

Solution: 2 forward (add to beq, addi to lw after loop)
1 stall (beq)
1 write then read (lw to beq)

Draw the pipeline diagram for one pass through the code plus the first instruction that is executed after the branch indicated with a $*$ is taken (i.e. the `L` instruction will be the last one in your diagram). **Be sure to indicate any data forwarding and/or pipeline stalls that occur.** Remember that the final pipelined datapath developed in class has the branch logic in the **ID** (decode) stage and a branch delay slot. This optimization also applies to jumps (e.g. `jal`, `jalr`).

Fit your diagram in the table below.

instruction	pipeline stage															
lw	F	D	X	M	[W] t1											
add		F	D	X t2	M	W										
beq			F	{D}	[D]	X	M	W				STALL				
addi				{F}	F	D	X t0	M	W			BRANCH DELAY SLOT				
lw						F	D	X	M	W						