



Pipelining the MIPS Datapath

Today's lecture

- **Pipeline implementation**
 - Single-cycle Datapath
 - Pipelining performance
 - Pipelined datapath
 - Example

Single-cycle implementation *Lab*

- So far we have built a **single-cycle implementation** of a subset of the MIPS-based instruction set.
 - We have assumed that instructions execute in the same amount of time; this determines the clock cycle time.
 - We have implemented the datapath and the control unit.

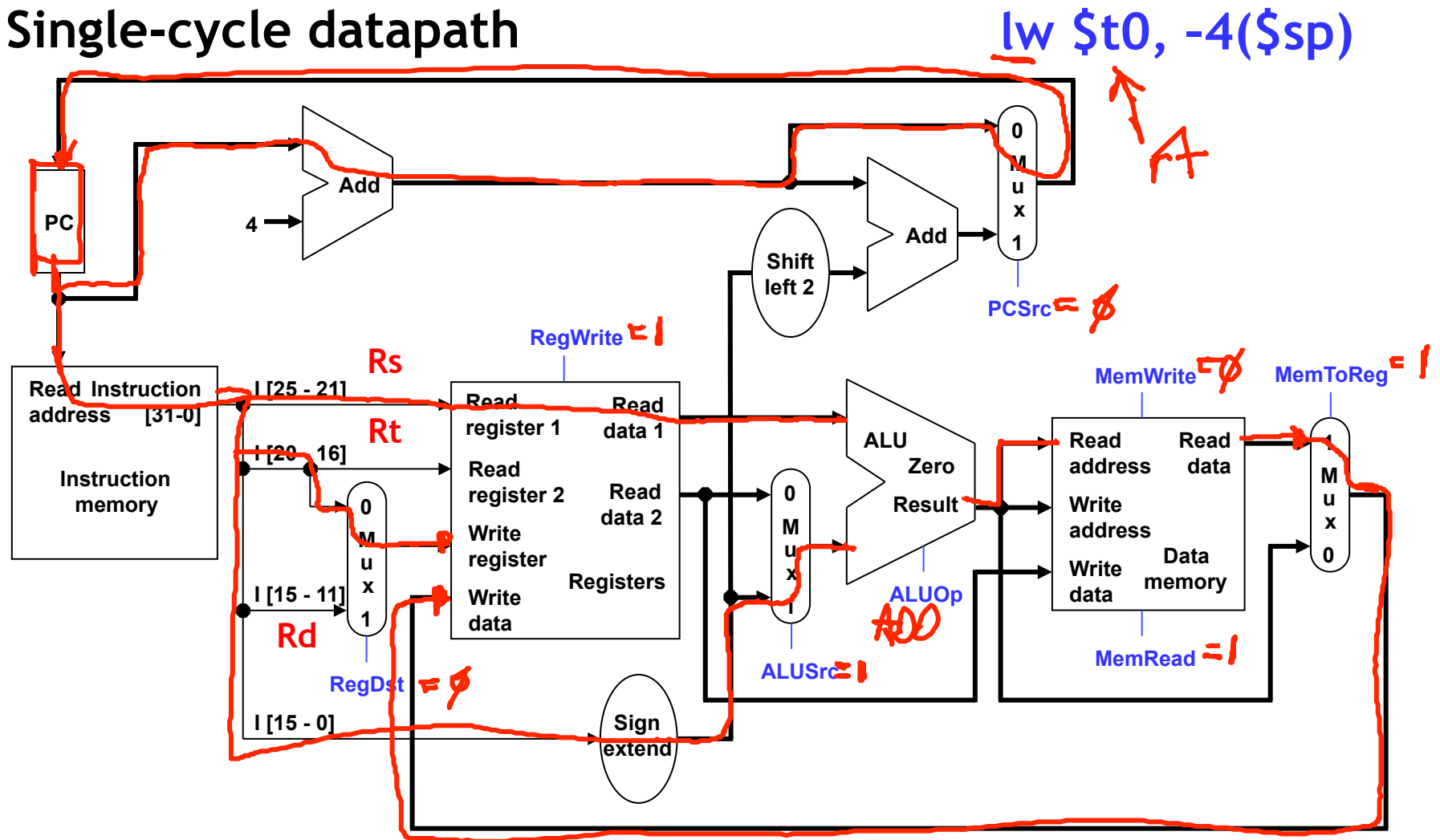


Single-cycle implementation

- For the following lectures, we will use a simpler implementation of the MIPS-based instruction set supporting just the following operations.

| | | | | | |
|----------------|-----|-----|-----|----|-----|
| Arithmetic: | add | sub | and | or | slt |
| Data Transfer: | lw | sw | | | |
| Control: | beq | | | | |

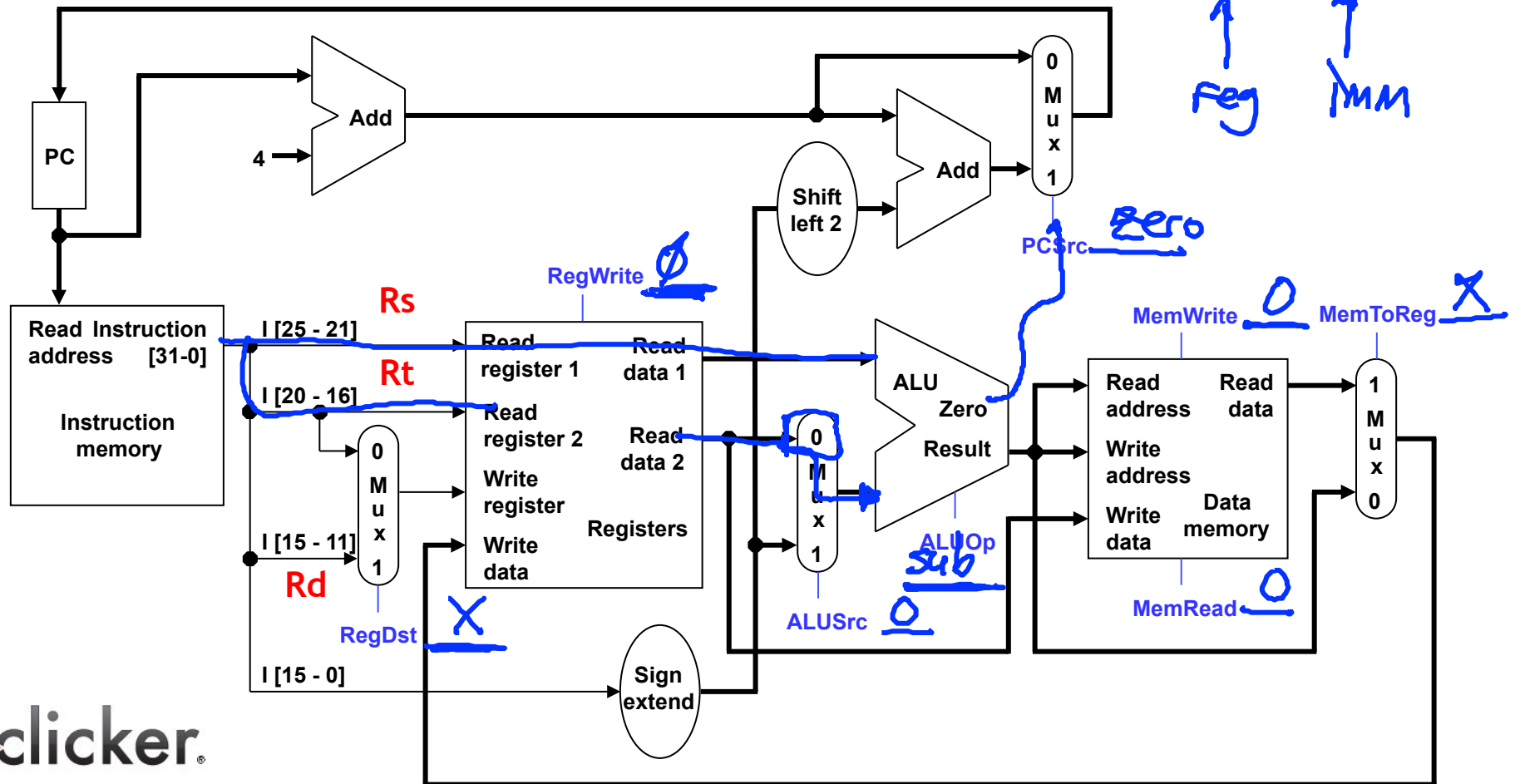
Single-cycle datapath



Single-cycle datapath

beq \$at, \$0, offset

↑
Reg
↑
Imm



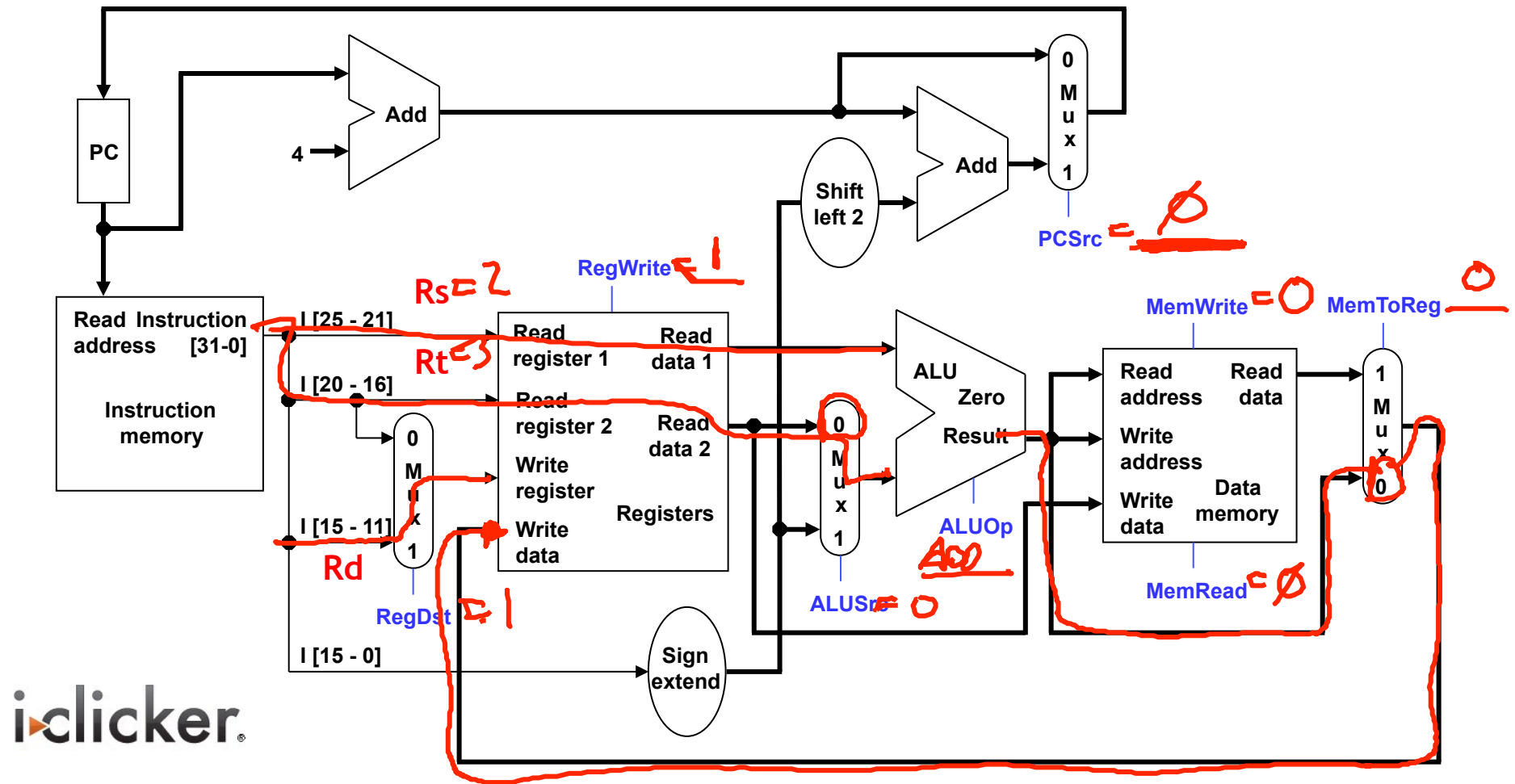
iclicker®

A) ALUSrc=0

B) ALUSrc=1

Single-cycle datapath

add \$1, \$2, \$3



iclicker

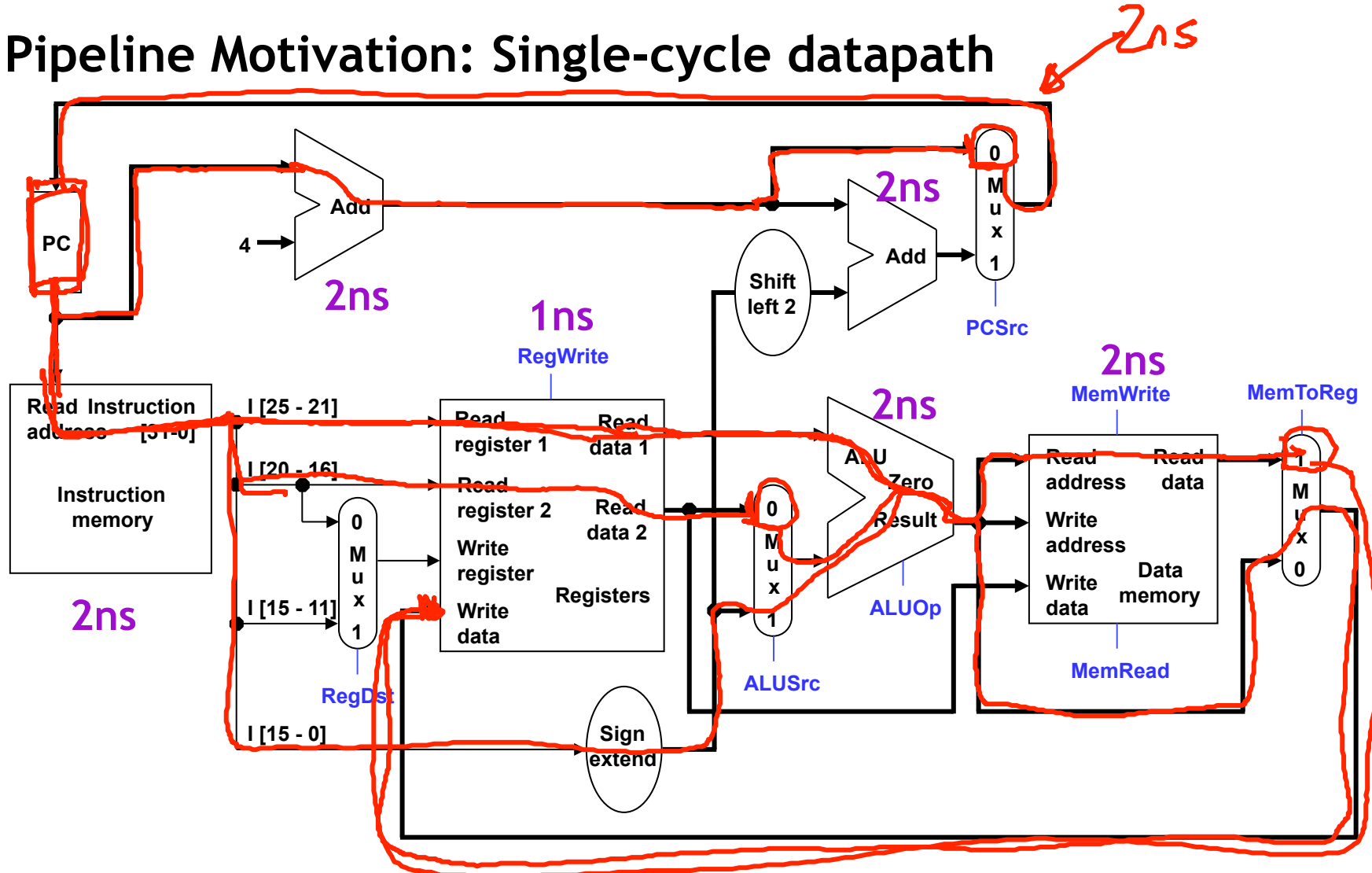
A) ALUSrc=0
RegDst=0

B) ALUSrc=0
RegDst=1

C) ALUSrc=1
RegDst=0

D) ALUSrc=1
RegDst=1

Pipeline Motivation: Single-cycle datapath

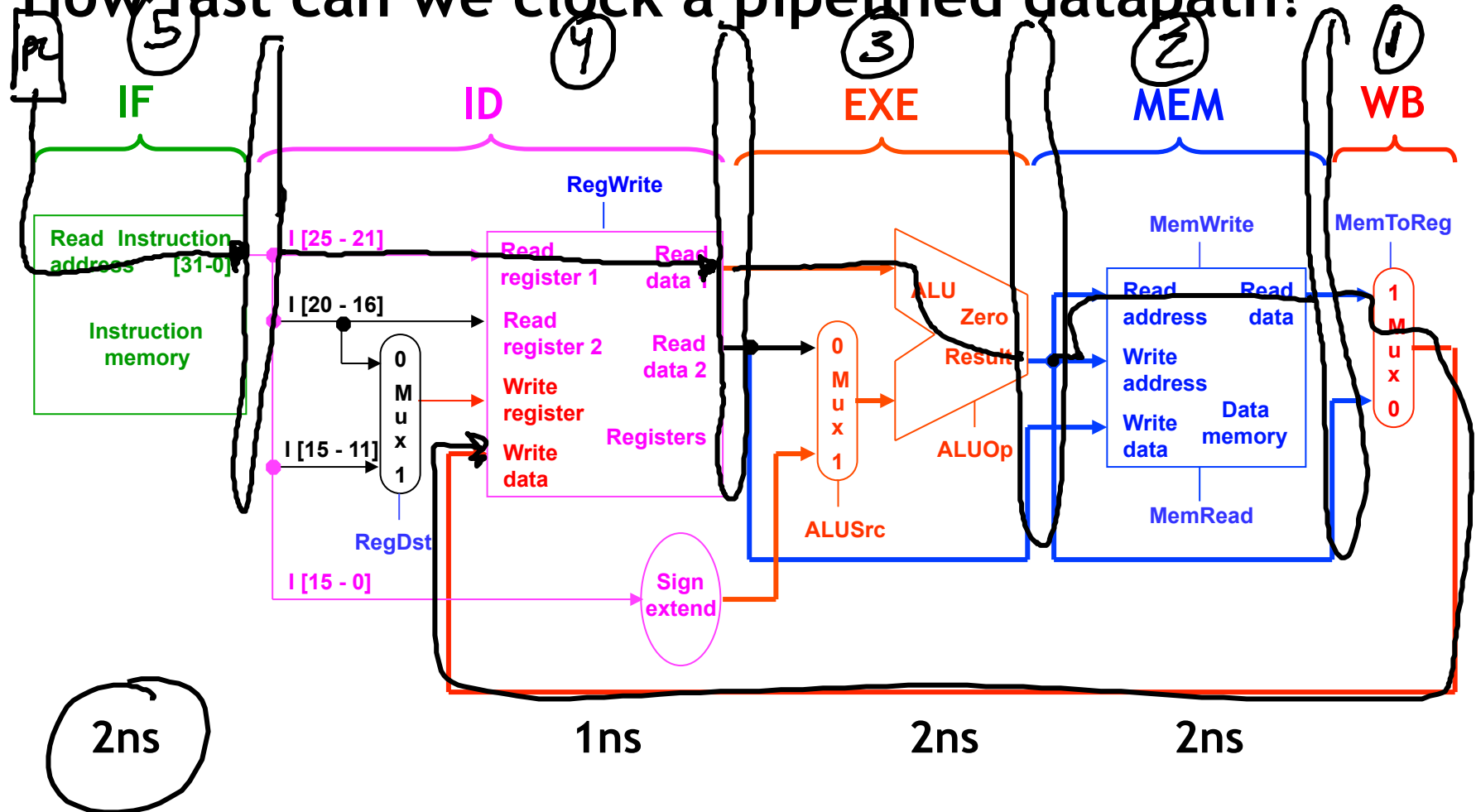


- How long does it take to execute each instruction?

$$\text{Add} = (2 + 1 + 2 + 1) = 6ns$$

$$\text{Load} = (2 + 1 + 2 + 2 + 1) = 8ns$$

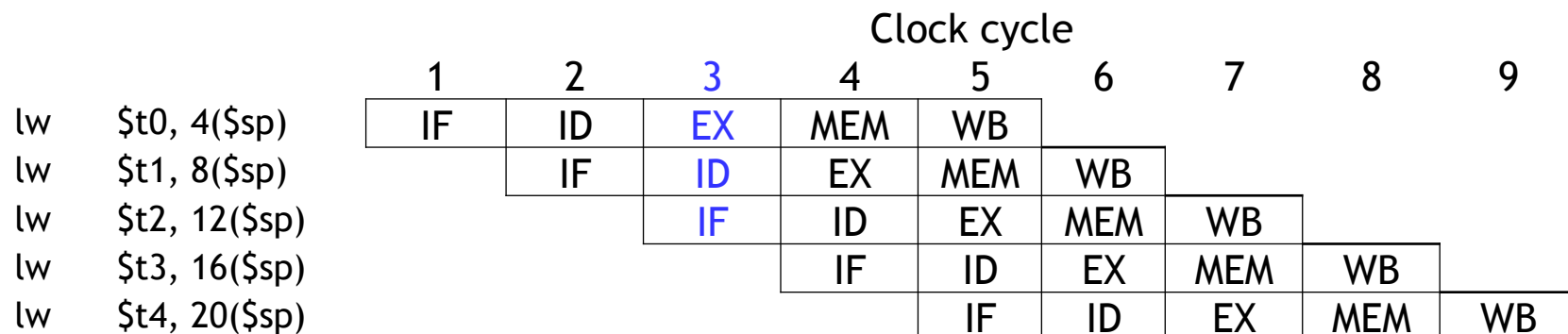
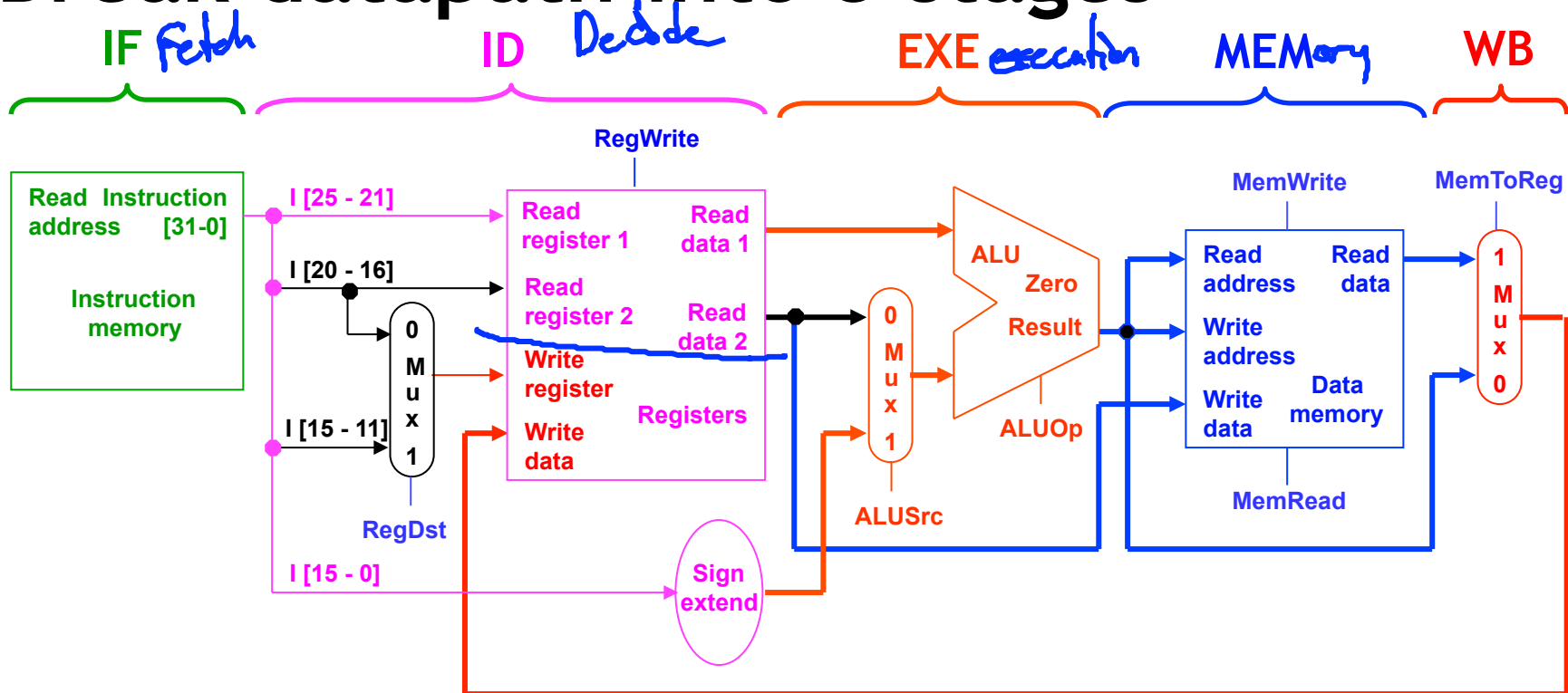
How fast can we clock a pipelined datapath?



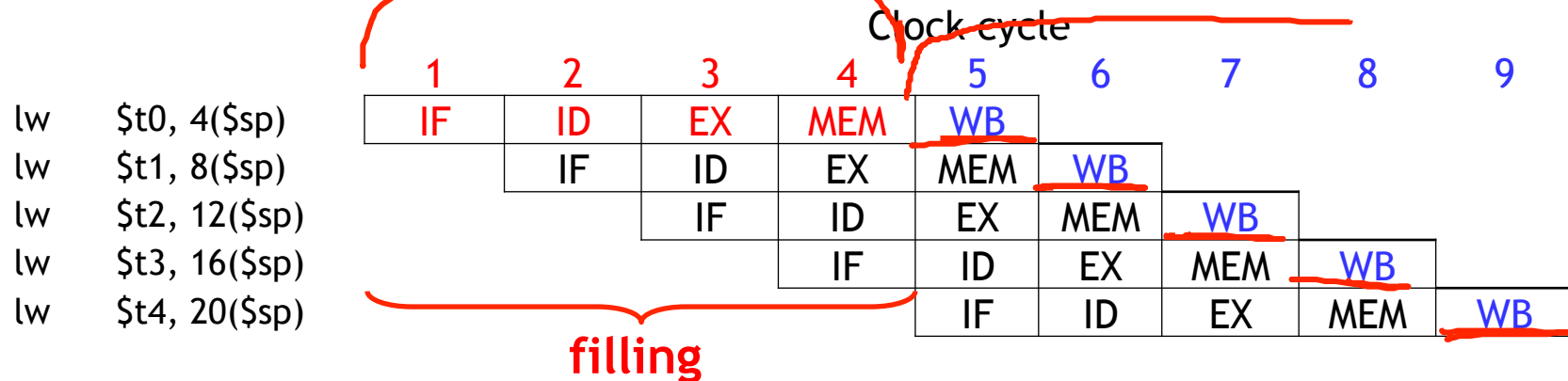
$$\max(2ns, 1ns, 2ns, 2ns, 1ns) = 2ns$$

Break datapath into 5 stages

write back



Pipelining Performance



■ Execution time on ideal pipeline:

- time to fill the pipeline + one cycle per instruction

- How long for N instructions?

$$- p \text{ stage} \quad ((p-1) + N) 2ns$$

■ Compare with other implementations:

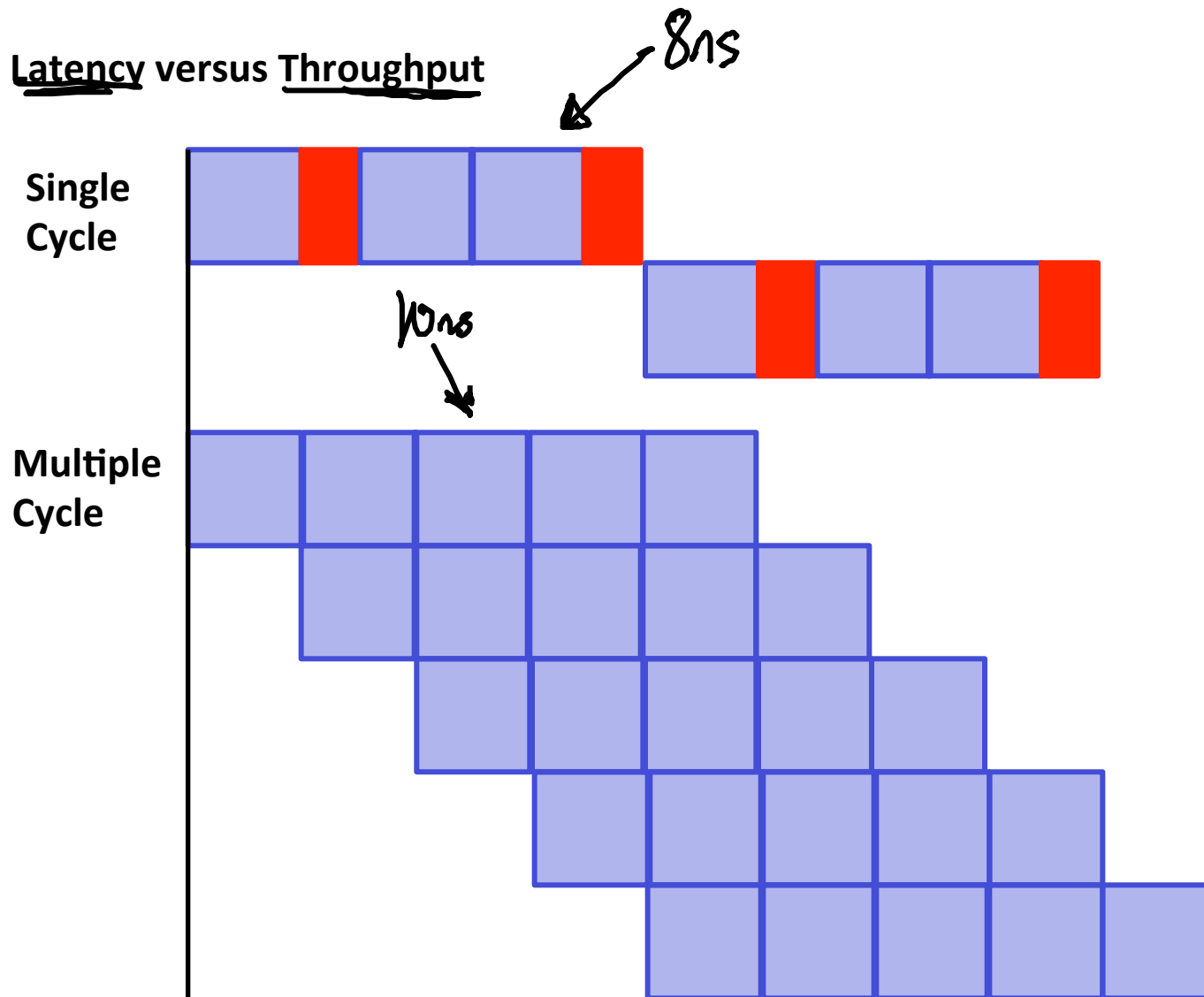
- Single Cycle: (8ns clock period)

$$8Nns$$

- How much faster is pipelining for N=1000 ?

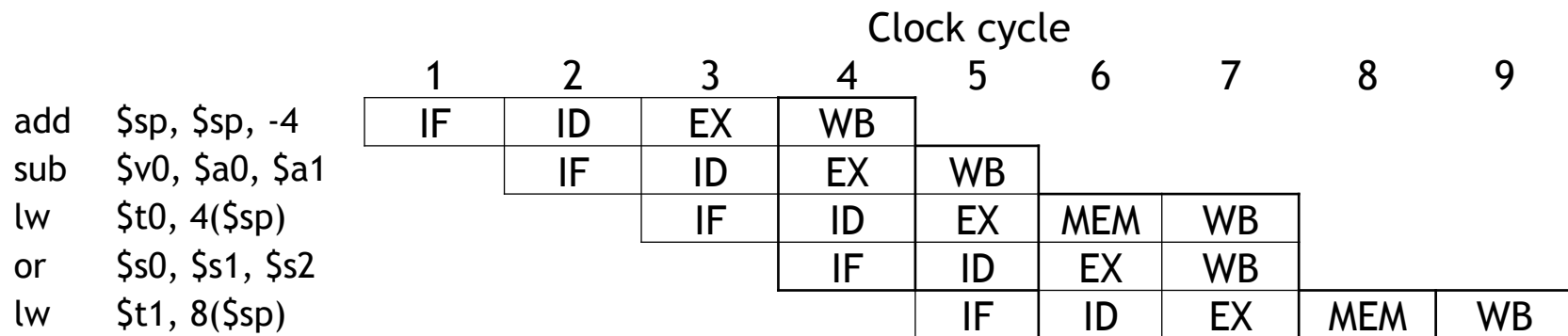
$$\frac{8N}{2((p-1) + N)} = \frac{8N}{2N} = 4$$

Pipeline versus Single Cycle implementation



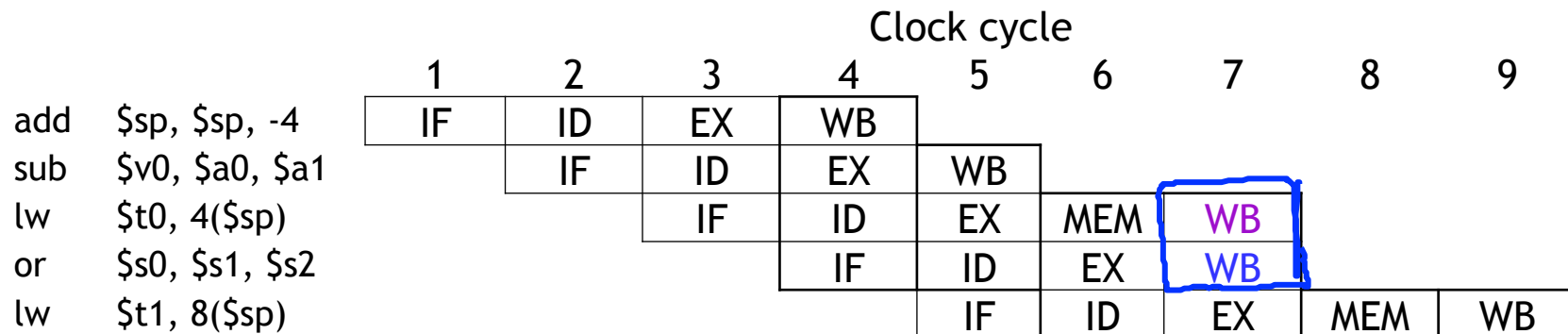
Pipelining other instruction types

- R-type instructions only require 4 stages: IF, ID, EX, and WB
 - We don't need the MEM stage
- What happens if we try to pipeline loads with R-type instructions?



Important Observation

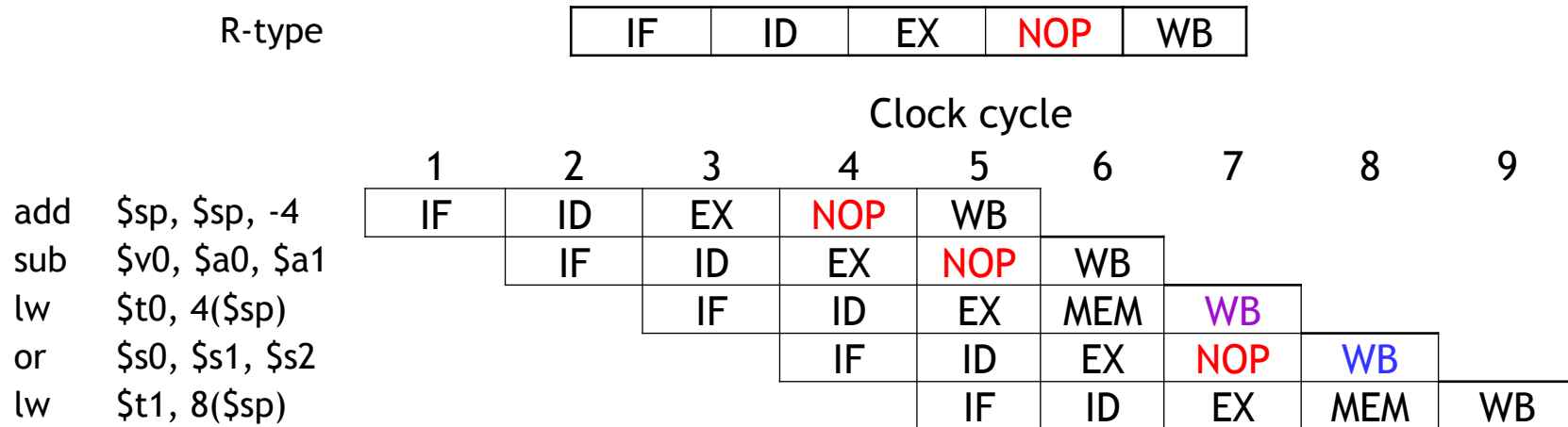
- Each functional unit can only be used **once** per instruction
- Each functional unit must be used at the **same** stage for all instructions:
 - Load uses Register File's Write Port during its **5th** stage
 - R-type uses Register File's Write Port during its **4th** stage



A solution: Insert NOP stages

- Enforce uniformity

- Make all instructions take 5 cycles.
- Make them have the same stages, in the same order
 - Some stages will **do nothing** for some instructions

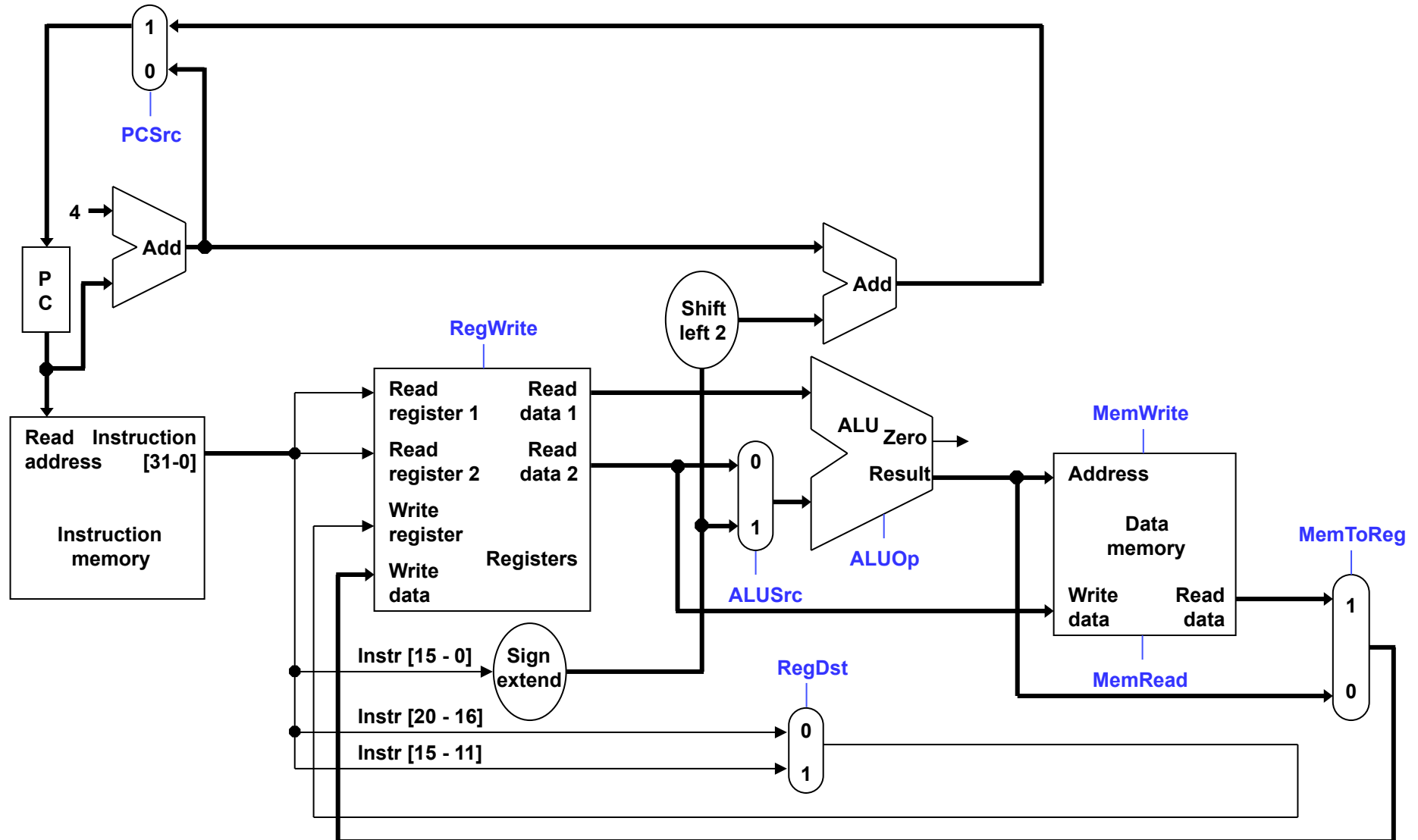


- Stores and Branches have **NOP** stages, too...

store
branch

| | | | | |
|----|----|----|-----|-----|
| IF | ID | EX | MEM | NOP |
| IF | ID | EX | NOP | NOP |

Single-cycle datapath, slightly rearranged



March 27, 2013

Pipelined datapath and control

17

Pipeline registers

- We'll add intermediate registers to our pipelined datapath.
- There's a lot of information to save, however. We'll simplify our diagrams by drawing just one big **pipeline register** between each stage.
- The registers are named for the stages they connect.

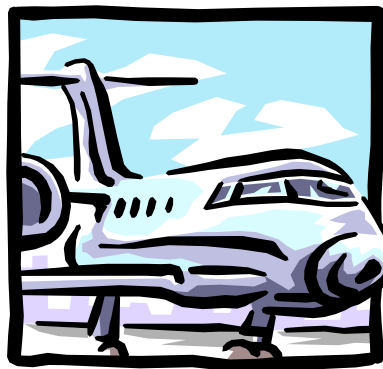
IF/ID

ID/EX

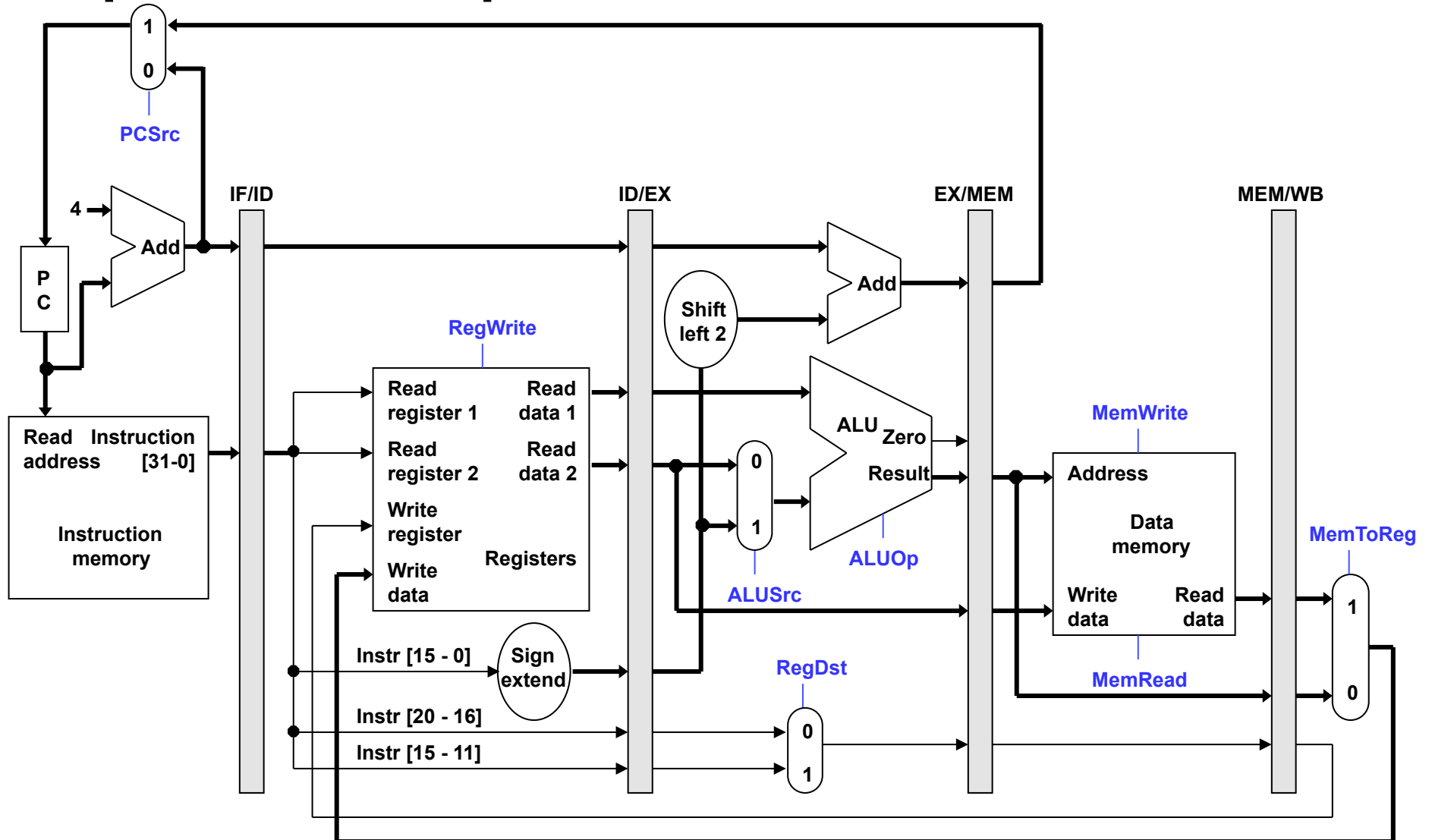
EX/MEM

MEM/WB

- No register is needed after the WB stage, because after WB the instruction is done.



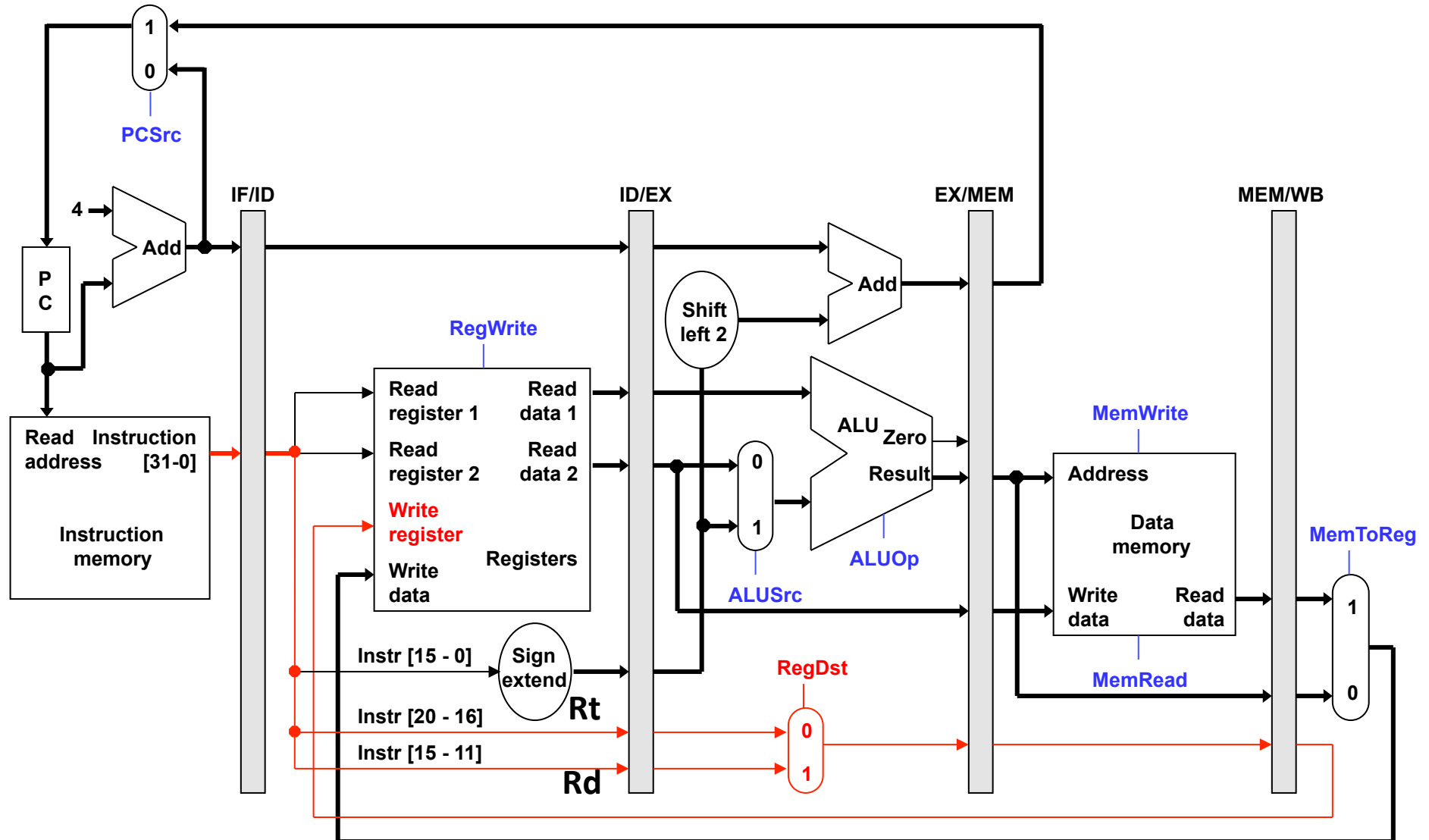
Pipelined datapath



Propagating values forward

- Any data values required in later stages must be propagated through the pipeline registers.
- The most extreme example is the destination register.
 - The rd field of the instruction word, retrieved in the first stage (IF), determines the destination register. But that register isn't updated until the *fifth* stage (WB).
 - Thus, the rd field must be passed through all of the pipeline stages, as shown in red on the next slide.
- Notice that we can't keep a single "instruction register," because the pipelined machine needs to fetch a new instruction every clock cycle.

The destination register



March 27, 2013

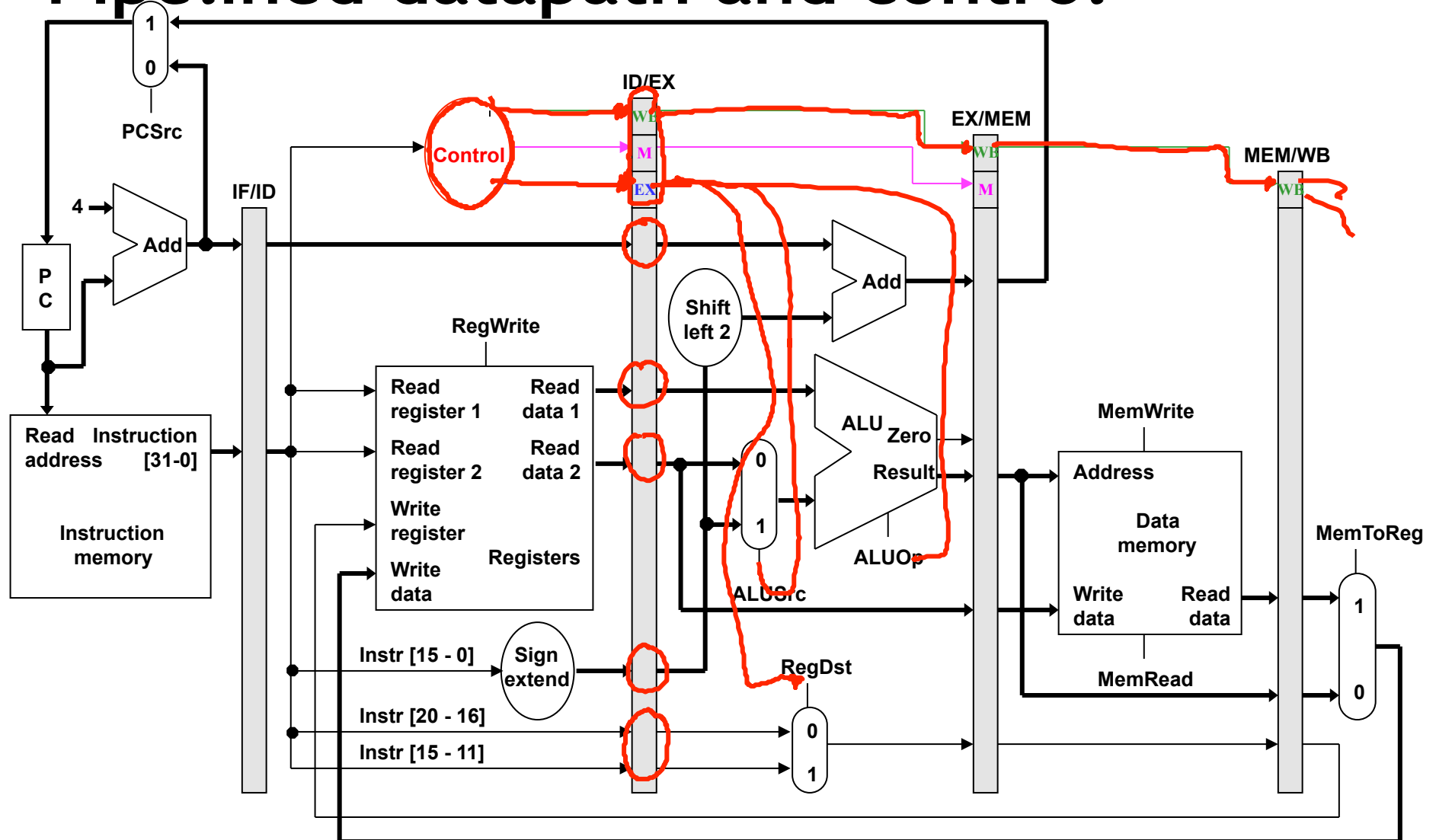
Pipelined datapath and control

21

What about control signals?

- The control signals are generated in the same way as in the single-cycle processor—after an instruction is fetched, the processor decodes it and produces the appropriate control values.
- But just like before, some of the control signals will not be needed until some later stage and clock cycle.
- These signals must be propagated through the pipeline until they reach the appropriate stage. We can just pass them in the pipeline registers, along with the other data.
- Control signals can be categorized by the pipeline stage that uses them.

Pipelined datapath and control



March 27, 2013

Pipelined datapath and control

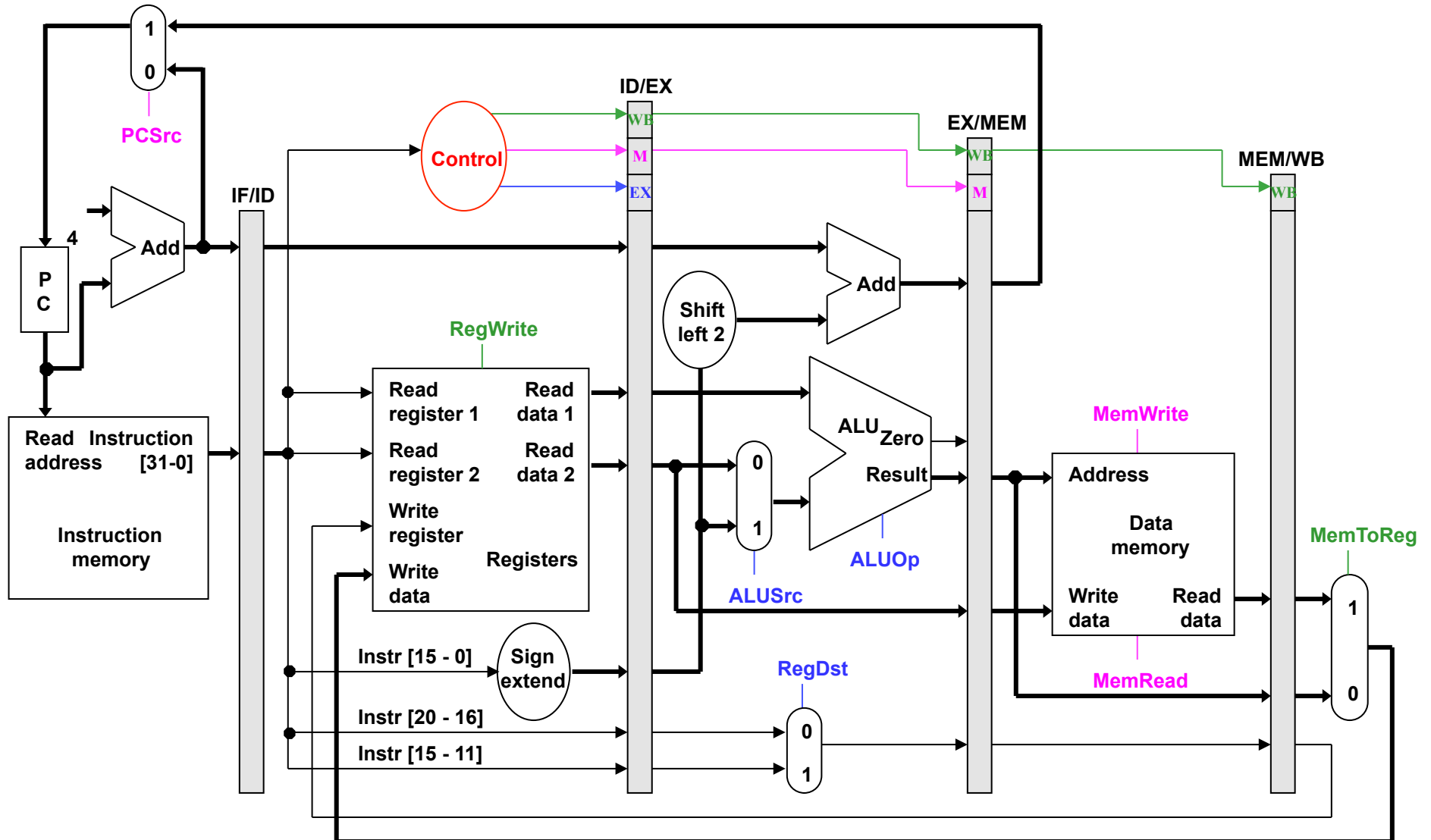
23

What about control signals?

- The control signals are generated in the same way as in the single-cycle processor—after an instruction is fetched, the processor decodes it and produces the appropriate control values.
- But, some of the control signals will not be needed until some later stage and clock cycle.
- These signals must be propagated through the pipeline until they reach the appropriate stage. We can just pass them in the pipeline registers, along with the other data.
- Control signals can be categorized by the pipeline stage that uses them.

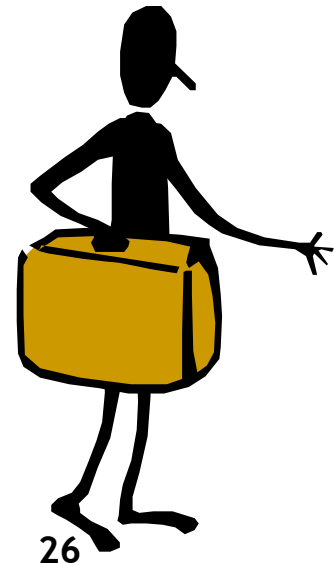
| Stage | Control signals needed | | |
|-------|------------------------|----------|--------|
| EX | ALUSrc | ALUOp | RegDst |
| MEM | MemRead | MemWrite | PCSrc |
| WB | RegWrite | MemToReg | |

Pipelined datapath and control



Notes about the diagram

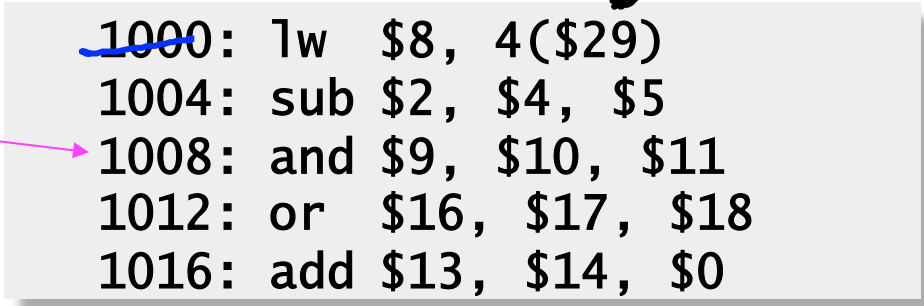
- The control signals are grouped together in the pipeline registers, just to make the diagram a little clearer.
- Not all of the registers have a write enable signal.
 - Because the datapath fetches one instruction per cycle, the PC must also be updated on each clock cycle. Including a write enable for the PC would be redundant.
 - Similarly, the pipeline registers are also written on every cycle, so no explicit write signals are needed.



An example execution sequence

- Here's a sample sequence of instructions to execute.

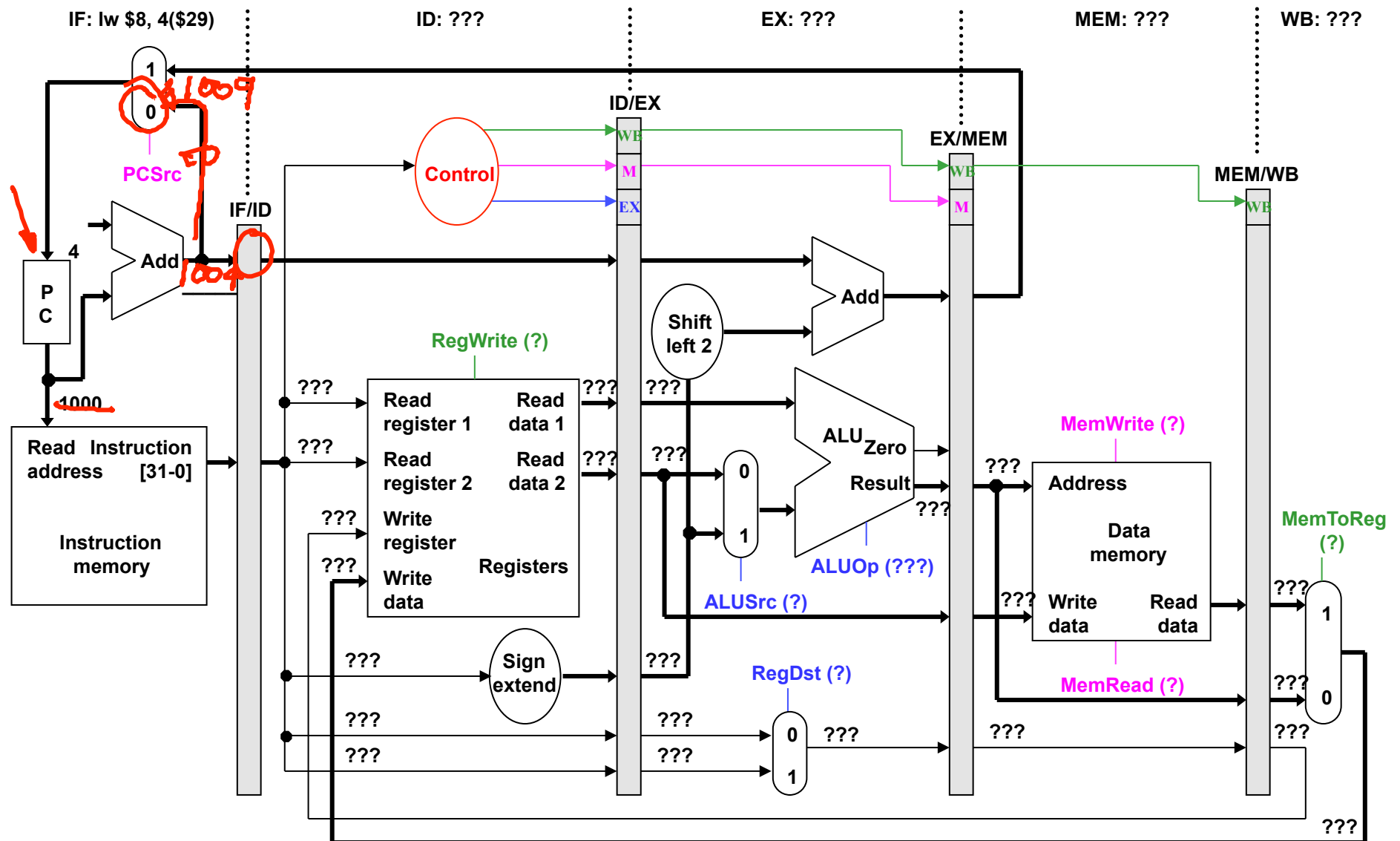
addresses in
decimal



```
1000: lw $8, 4($29)
1004: sub $2, $4, $5
1008: and $9, $10, $11
1012: or $16, $17, $18
1016: add $13, $14, $0
```

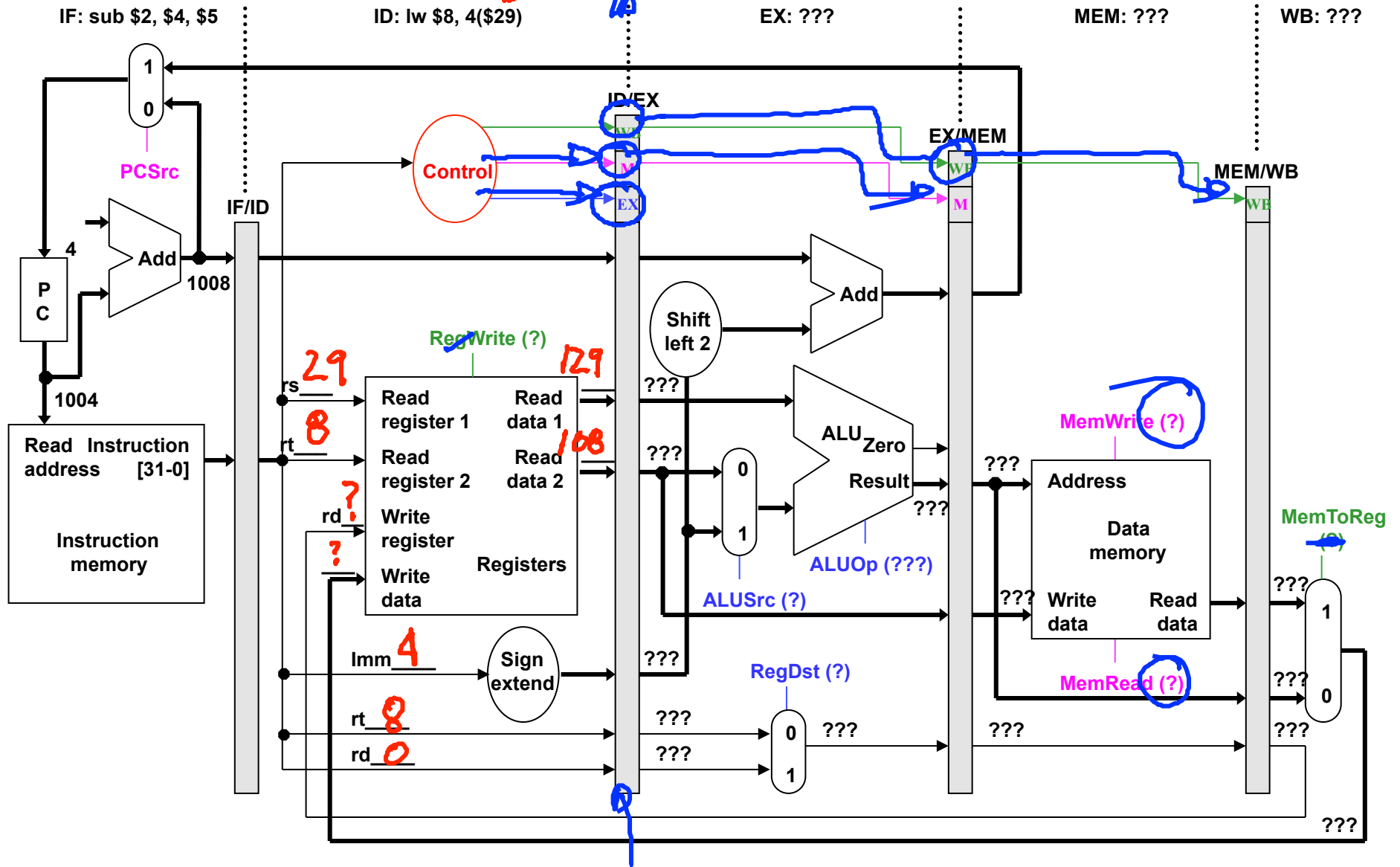
- We'll make some assumptions, just so we can show actual data values.
 - Each register contains its number plus 100. For instance, register \$8 contains 108, register \$29 contains 129, and so forth.
 - Every data memory location contains 99.
- Our pipeline diagrams will follow some conventions.
 - An X indicates values that aren't important, like the constant field of an R-type instruction.
 - Question marks ??? indicate values we don't know, usually resulting from instructions coming before and after the ones in our example.

Cycle 1 (filling)

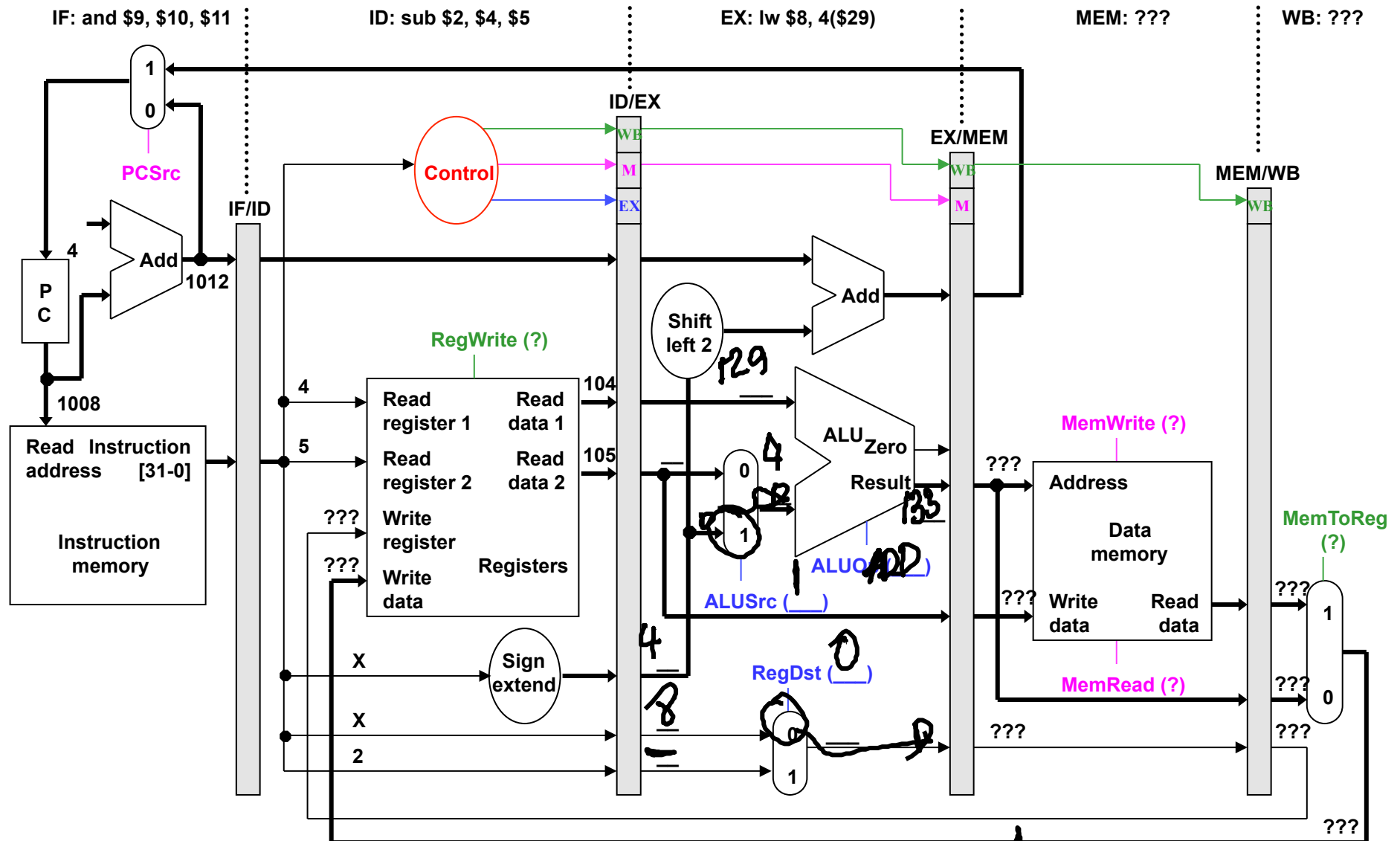


Cycle 2

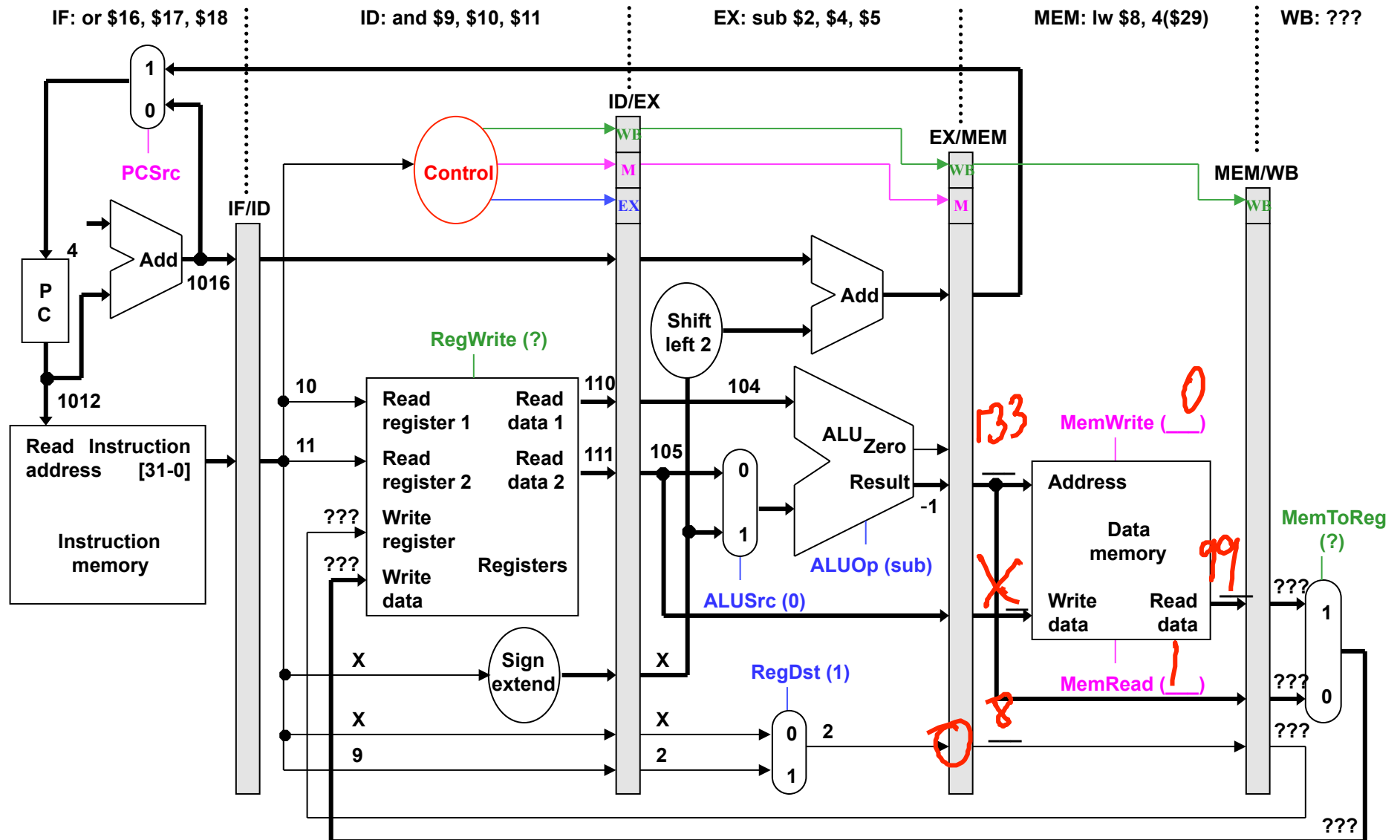
Handwritten notes:
 Pipeline Registers
 ID: lw \$8, 4(\$29)



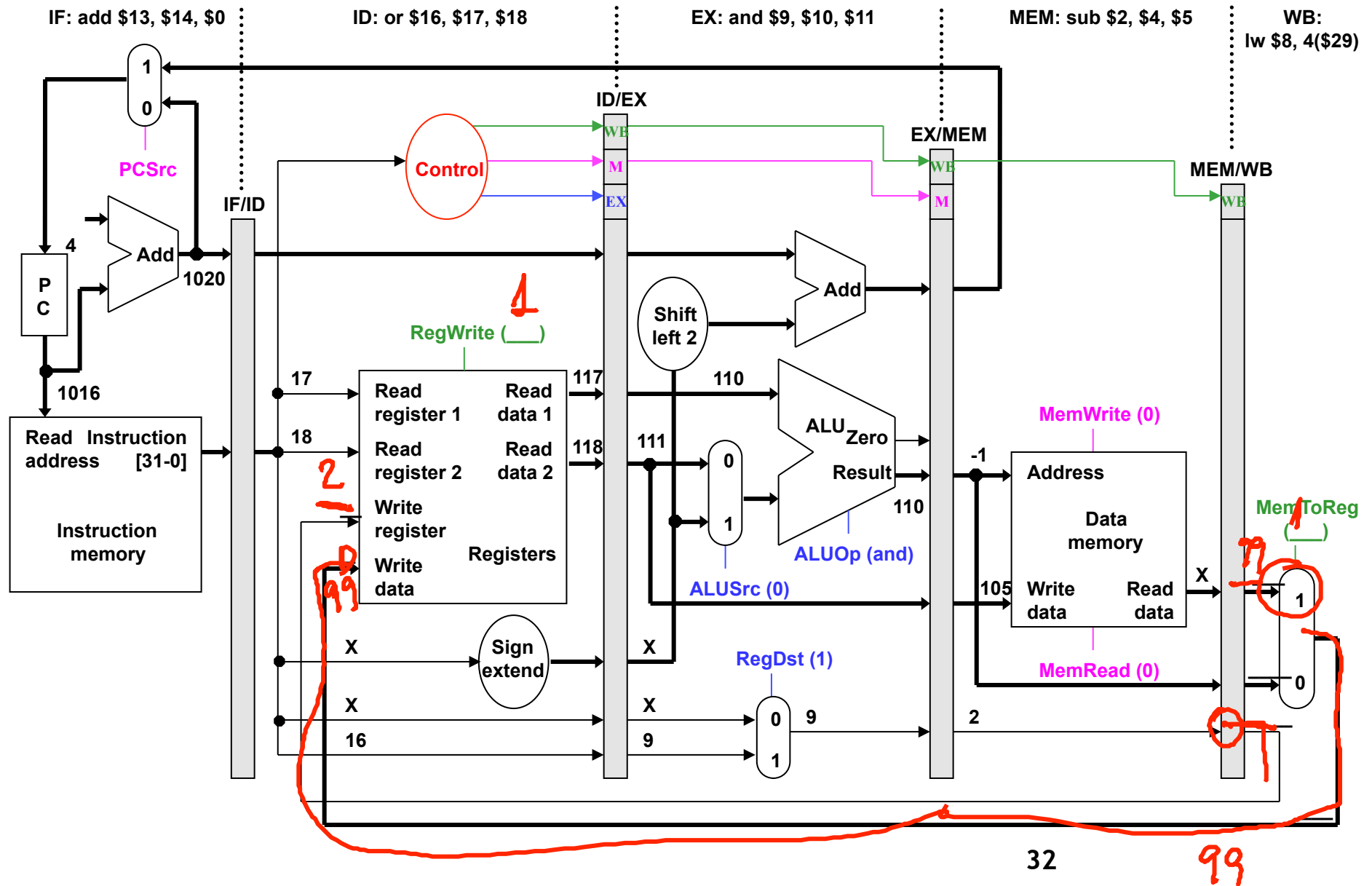
Cycle 3

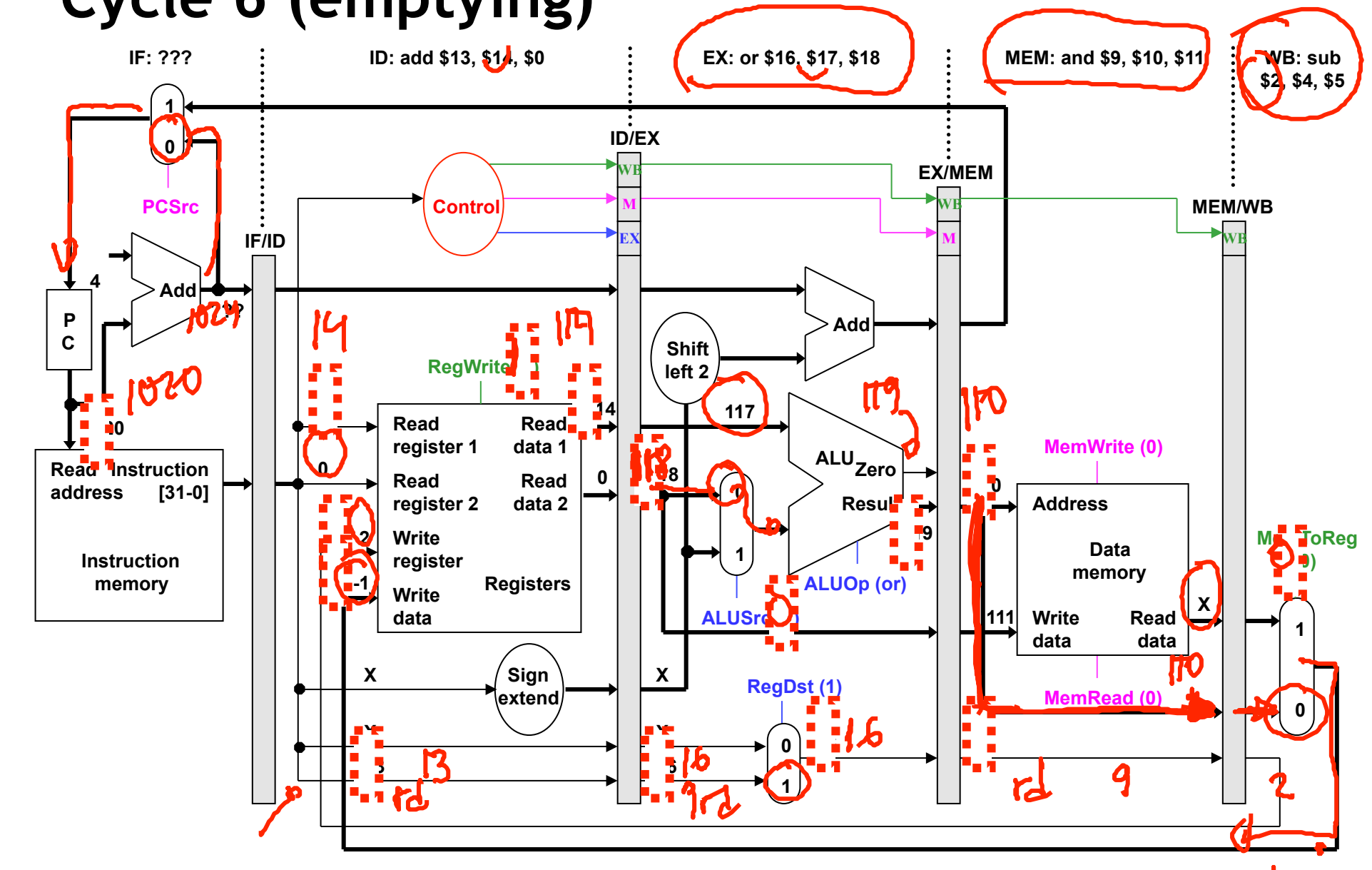


Cycle 4

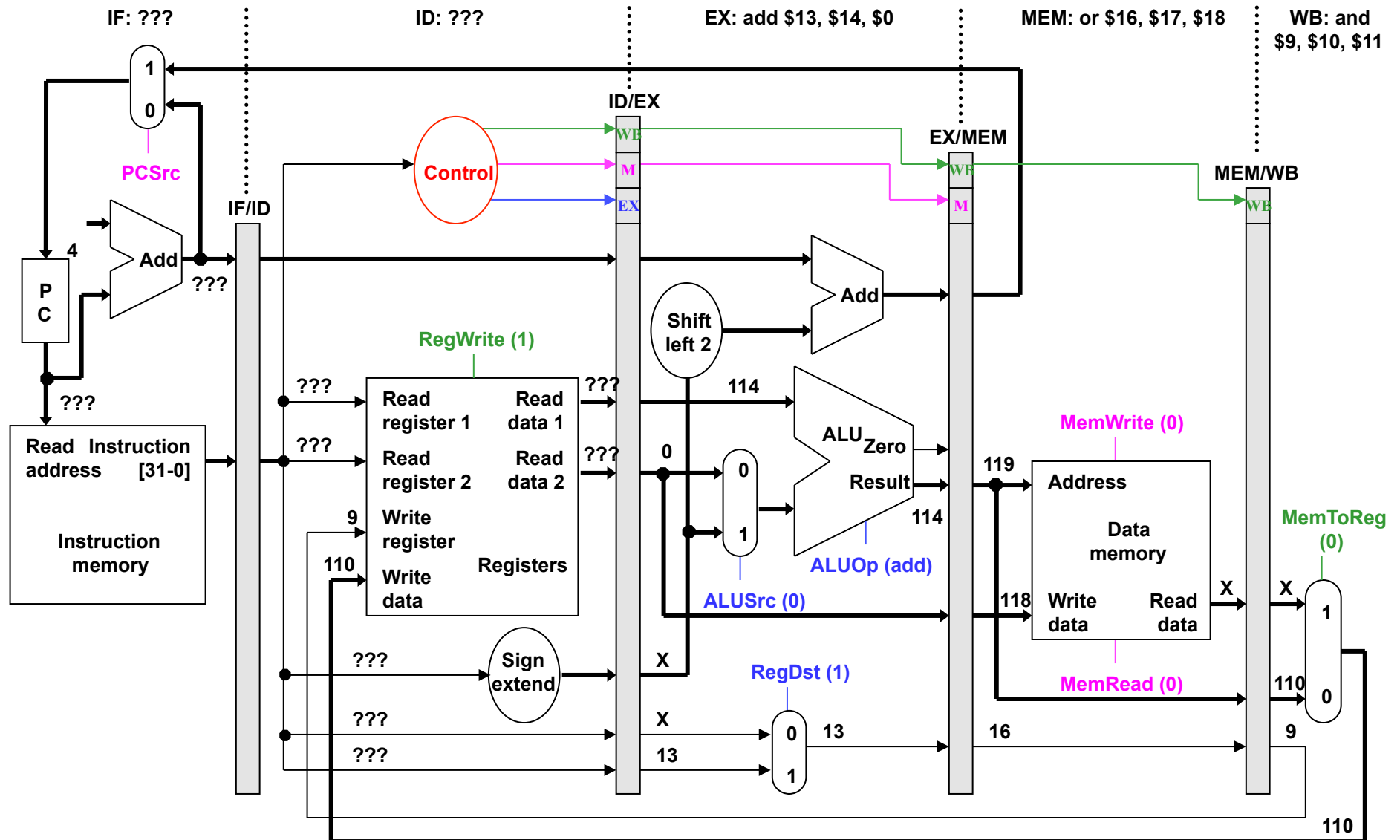


Cycle 5 (full)

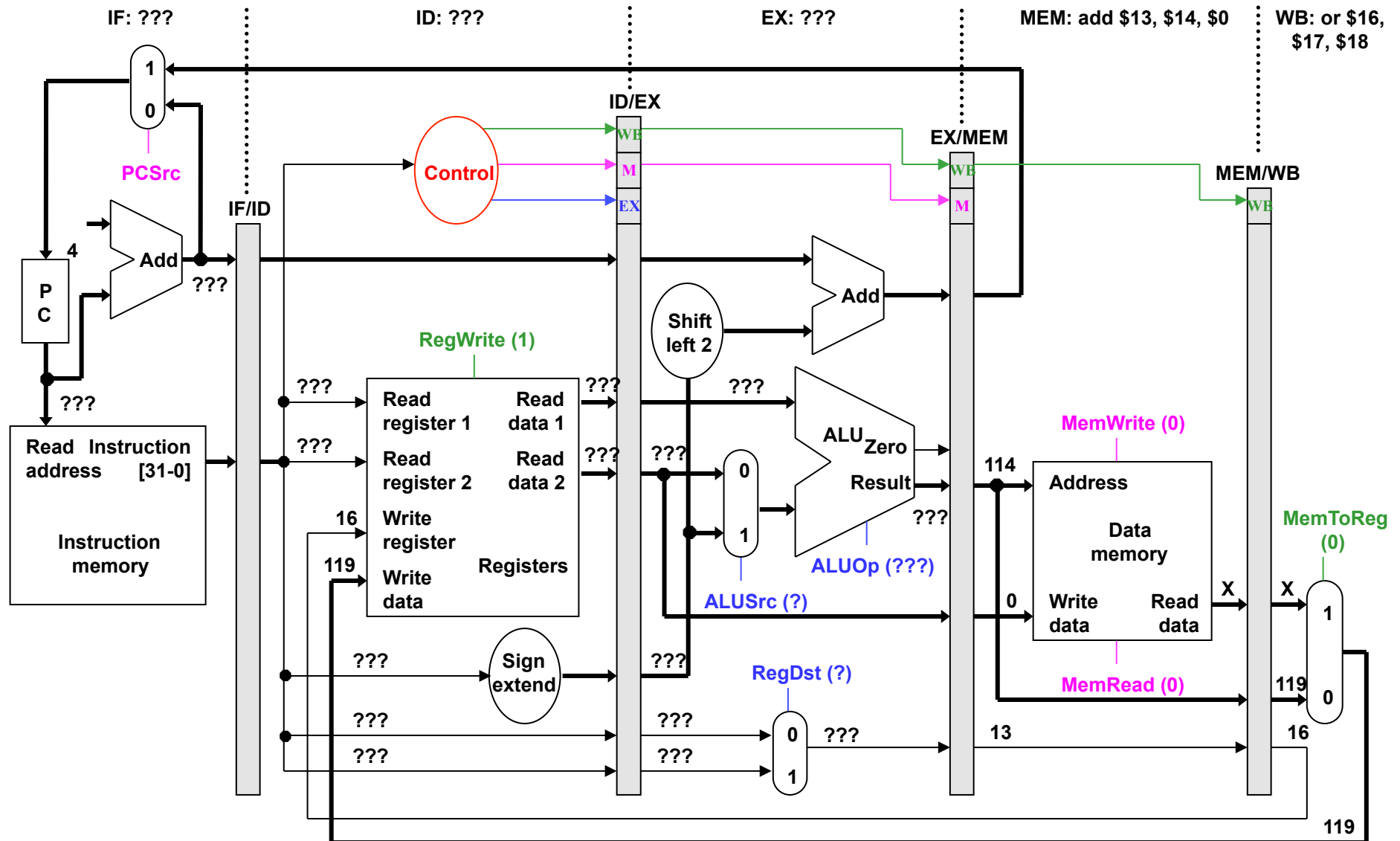




Cycle 7



Cycle 8

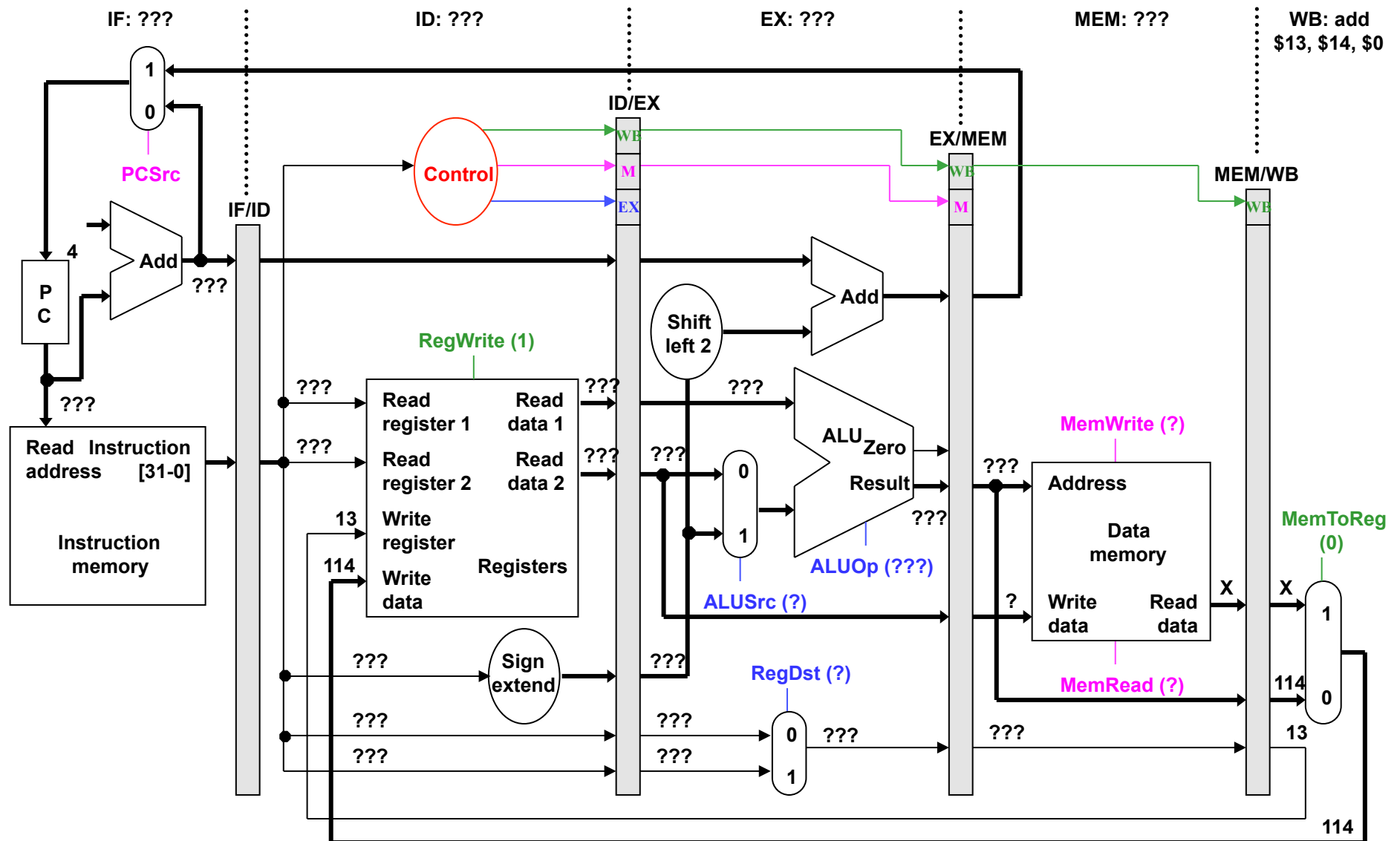


March 27, 2013

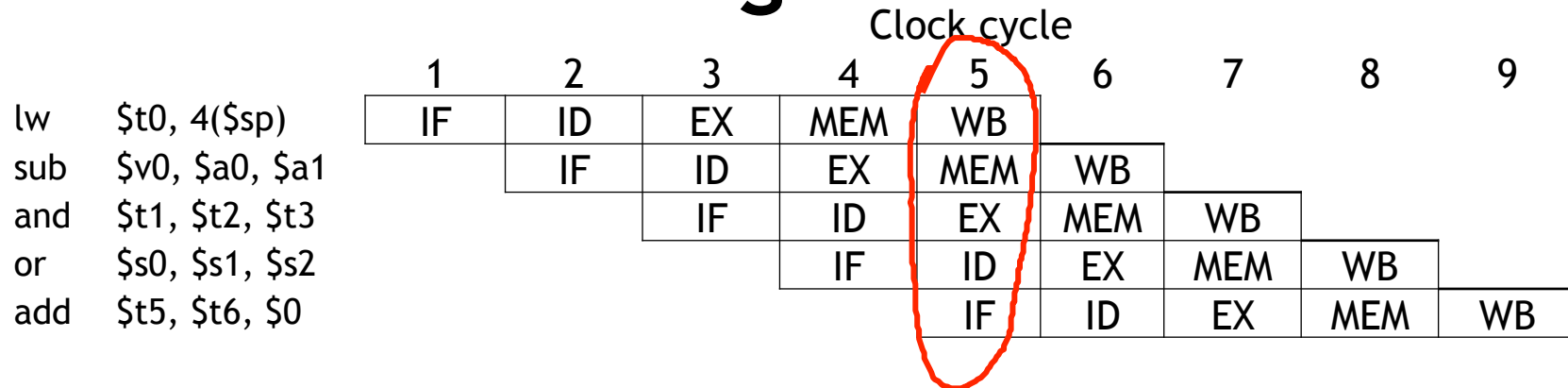
Pipelined datapath and control

35

Cycle 9



That's a lot of diagrams there



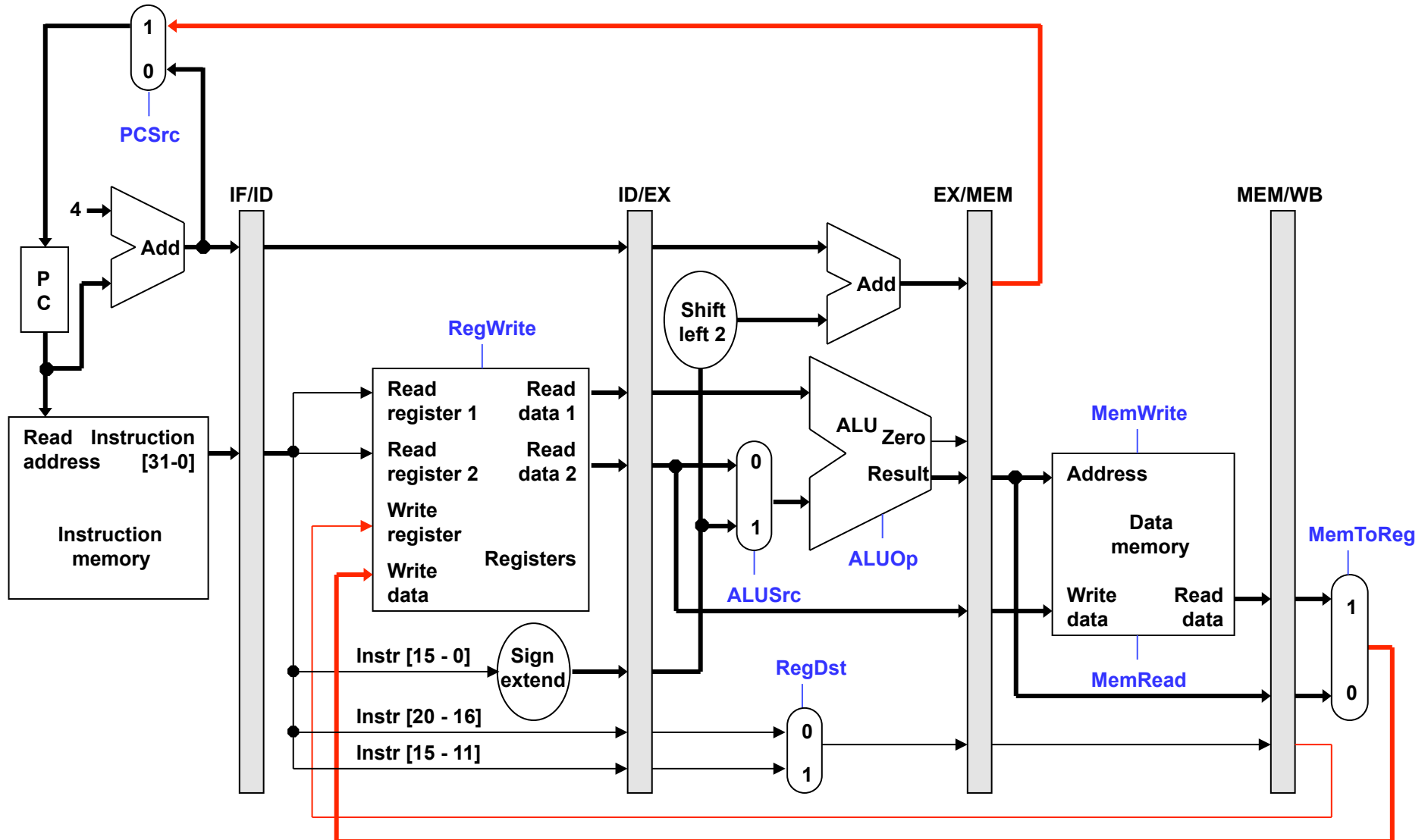
- Compare the last nine slides with the pipeline diagram above.
 - You can see how instruction executions are overlapped.
 - Each functional unit is used by a *different* instruction in each cycle.
 - The pipeline registers save control and data values generated in previous clock cycles for later use.
 - When the pipeline is full in clock cycle 5, all of the hardware units are utilized. This is the ideal situation, and what makes pipelined processors so fast.
- Try to understand this example or the similar one in the book at the end of Section 6.3.

Summary

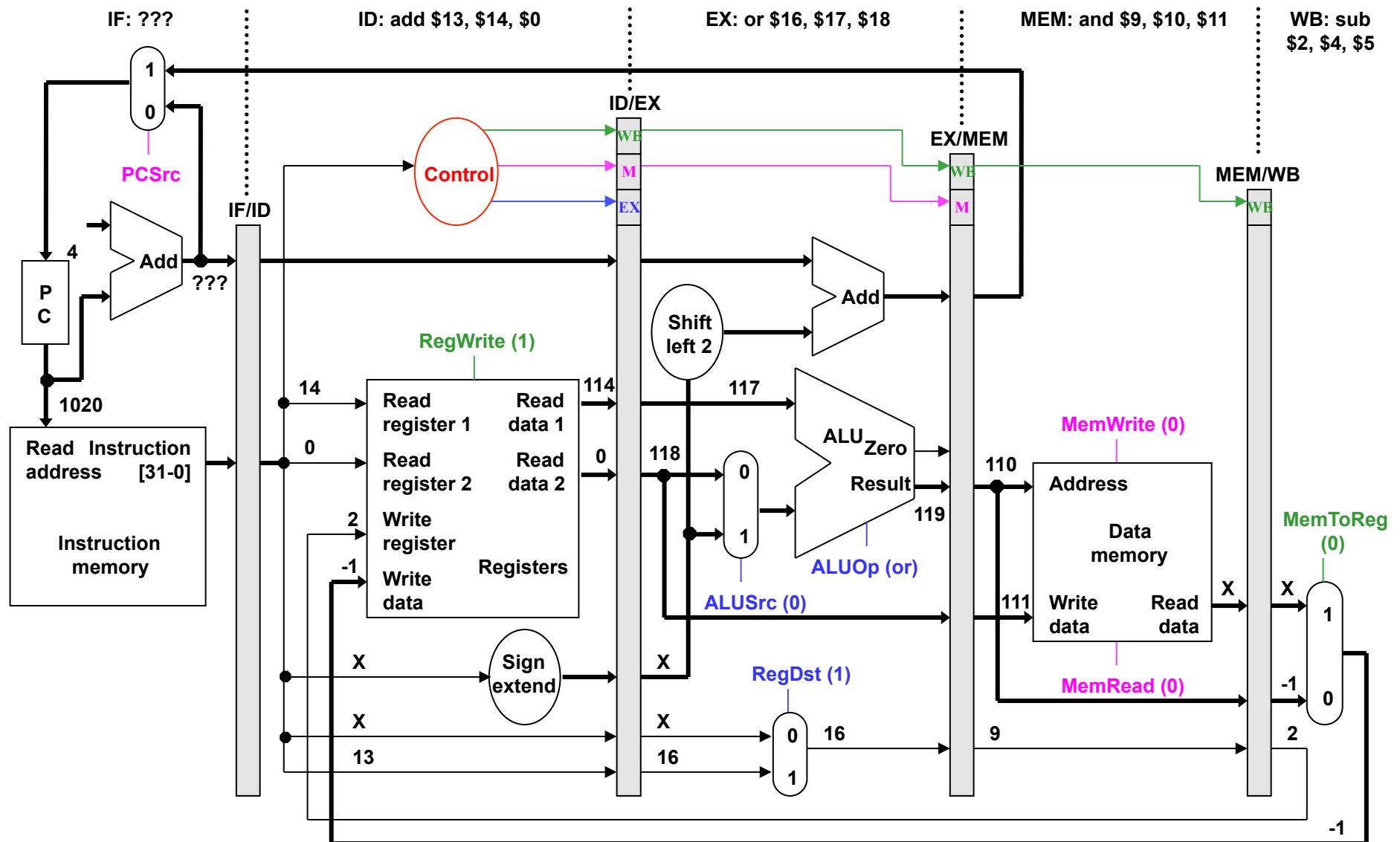
- The **pipelined datapath** extends the single-cycle processor that we saw earlier to improve instruction throughput.
 - Instruction execution is split into several stages.
 - Multiple instructions flow through the pipeline simultaneously.
- **Pipeline registers** propagate data and control values to later stages.
- The MIPS instruction set architecture supports pipelining with uniform instruction formats and simple addressing modes.
- Next lecture, we'll start talking about **Hazards**.



Note how everything goes left to right, except ...



Cycle 6 (emptying)



March 27, 2013

Pipelined datapath and control

40