Pipelining the MIPS Datapath

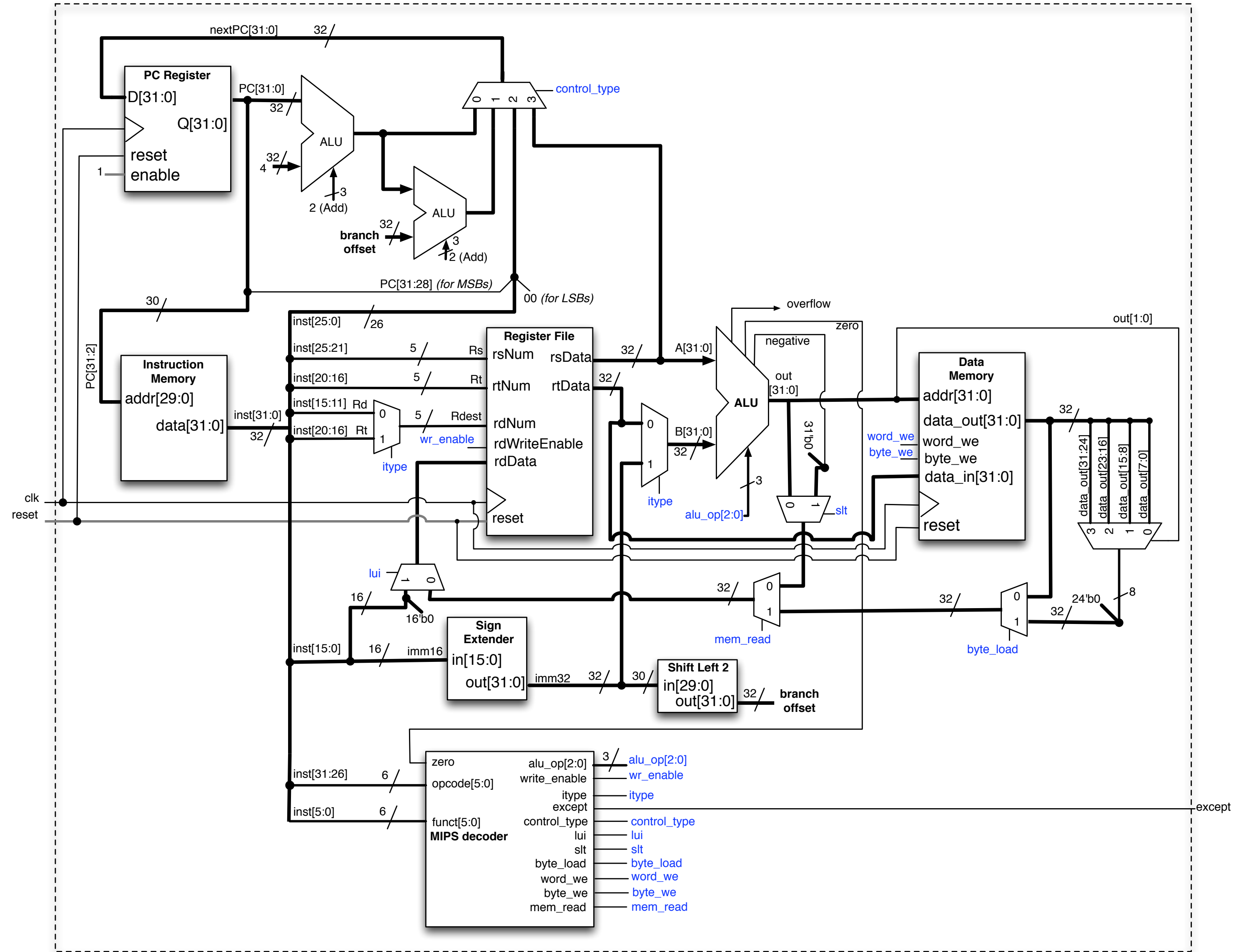
Today's lecture

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

Single-cycle implementation

- **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.

Full Machine Datapath – Lab 6



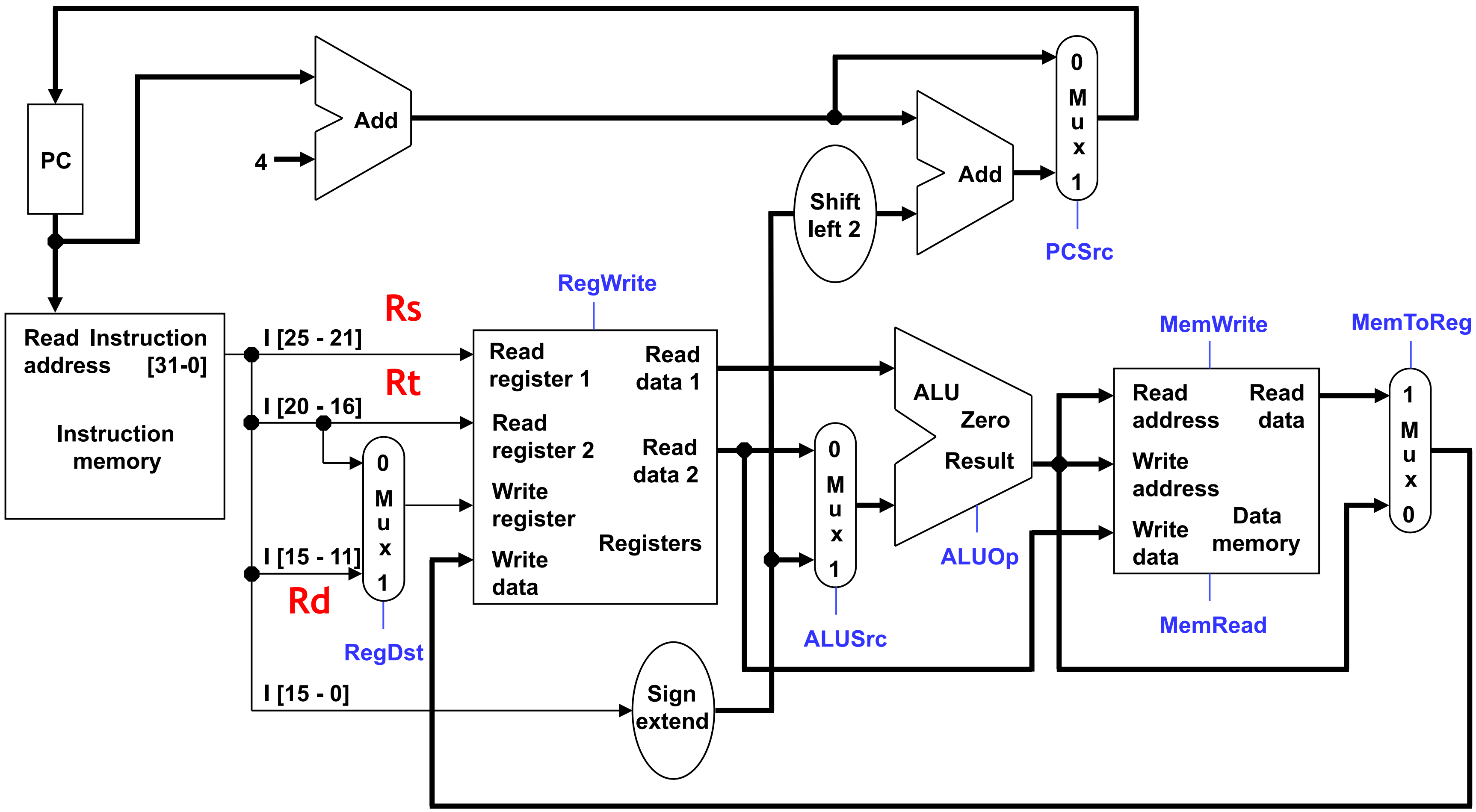
Single-cycle implementation

- **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				

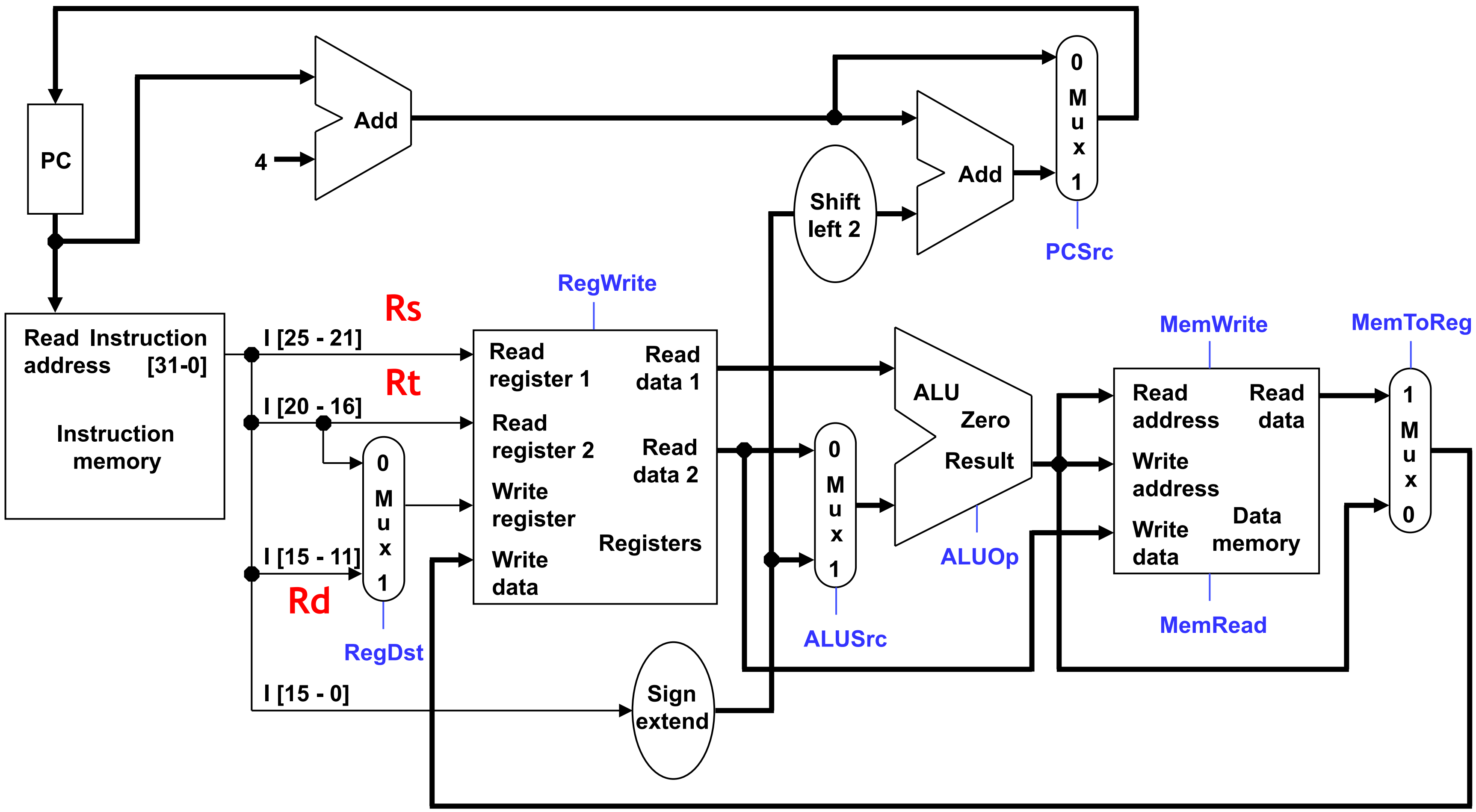
Single-cycle datapath

lw $t0, -4($sp)



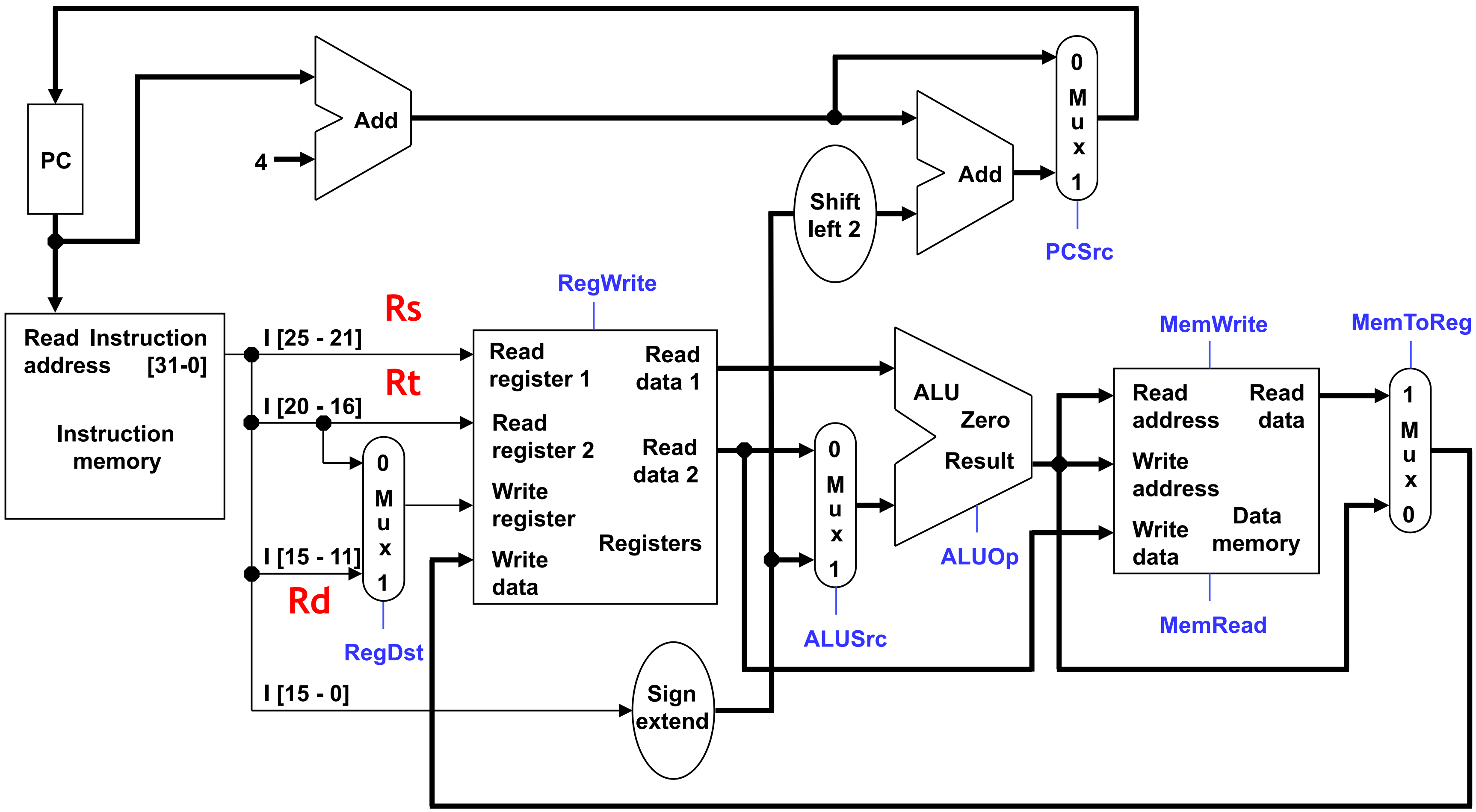
Single-cycle datapath

beq $at, $0, offset

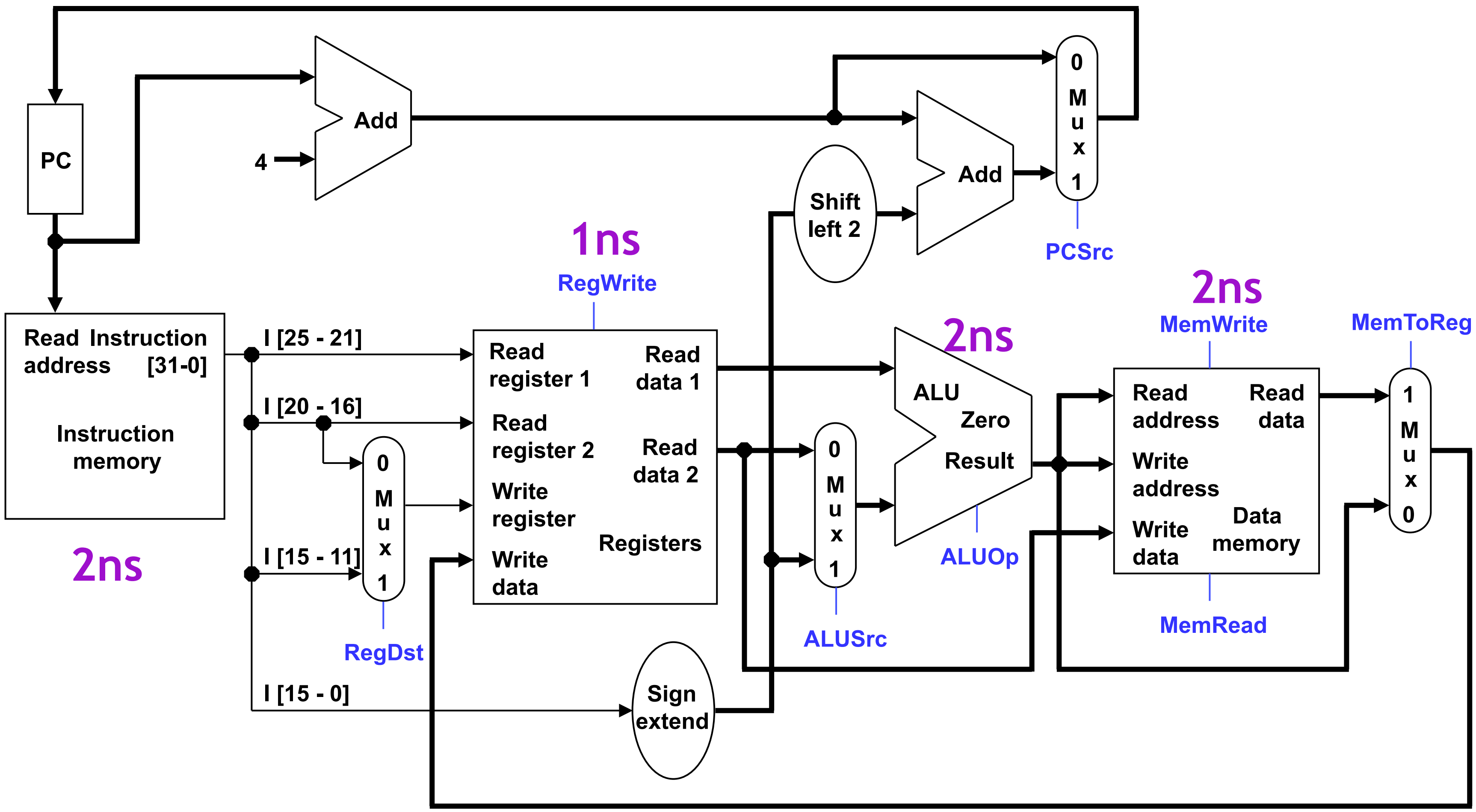


Single-cycle datapath

add $1, $2, $3

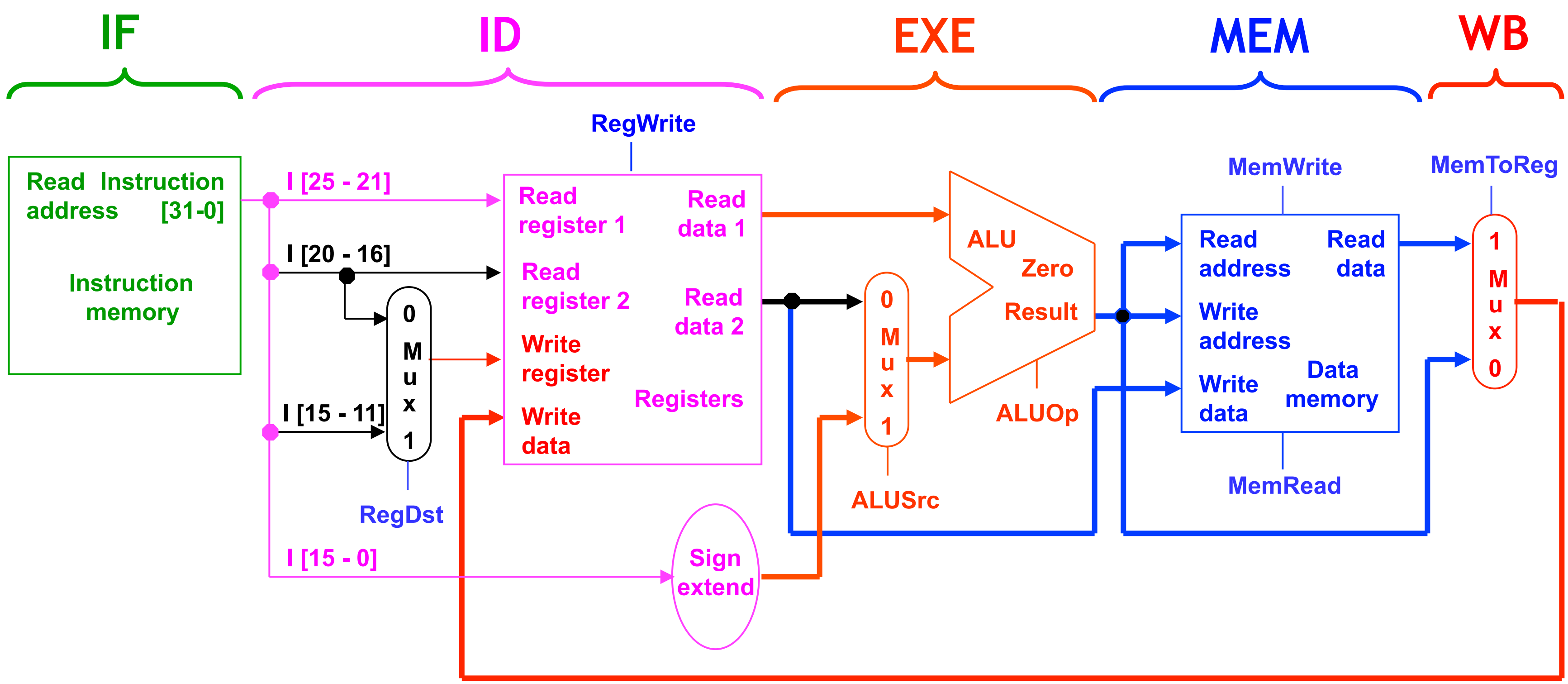


Pipeline Motivation: Single-cycle datapath



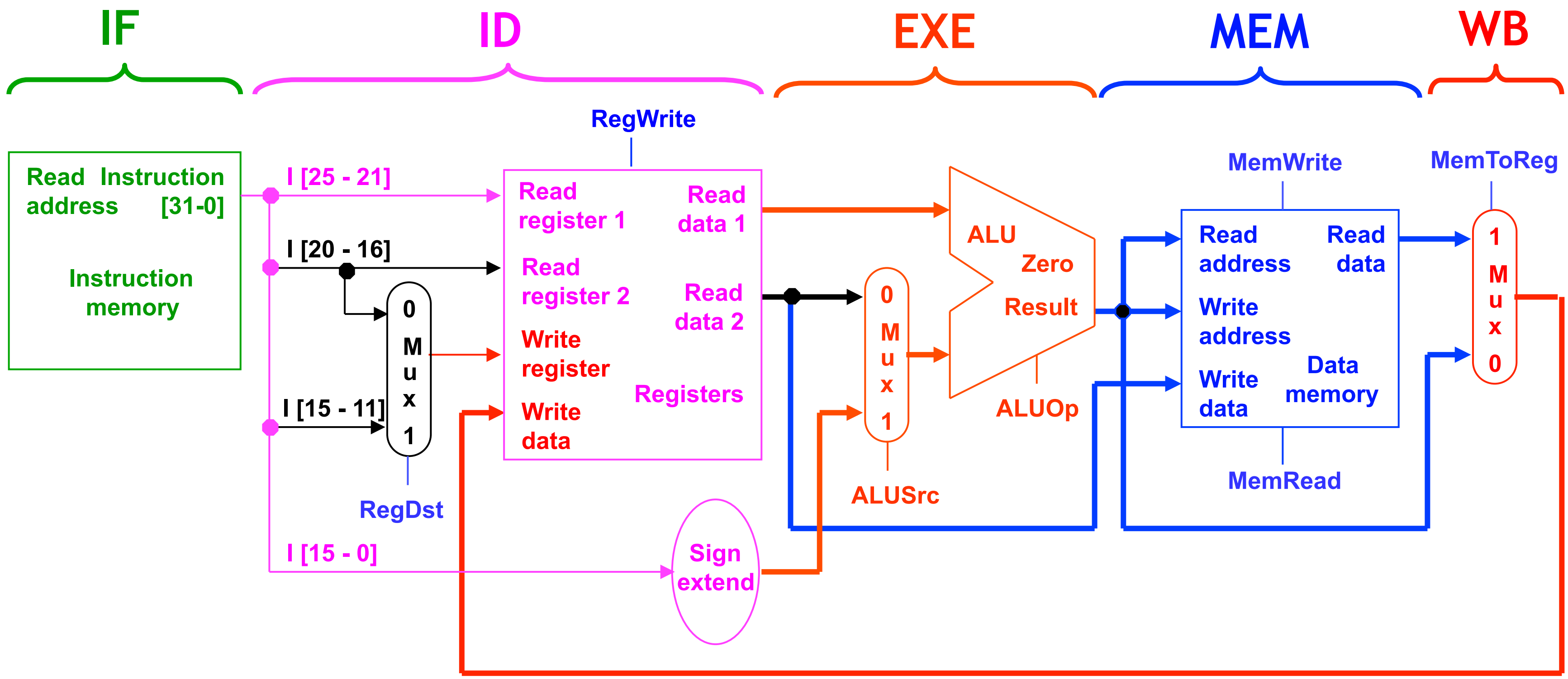
- How long does it take to execute each instruction?

How fast can we clock this datapath



2ns 1ns 2ns 2ns

Break datapath into 5 stages



Clock cycle

	1	2	3	4	5	6	7	8	9
lw $t0, 4($sp)	IF	ID	EX	MEM	WB				
lw $t1, 8($sp)		IF	ID	EX	MEM	WB			
lw $t2, 12($sp)			IF	ID	EX	MEM	WB		
lw $t3, 16($sp)				IF	ID	EX	MEM	WB	
lw $t4, 20($sp)					IF	ID	EX	MEM	WB

Pipelining Performance

Clock cycle

		1	2	3	4	5	6	7	8	9
lw	$t0, 4($sp)	IF	ID	EX	MEM	WB				
lw	$t1, 8($sp)		IF	ID	EX	MEM	WB			
lw	$t2, 12($sp)			IF	ID	EX	MEM	WB		
lw	$t3, 16($sp)				IF	ID	EX	MEM	WB	
lw	$t4, 20($sp)					IF	ID	EX	MEM	WB

filling

- **Execution time on ideal pipeline:**
  - time to fill the pipeline + one cycle per instruction
  - How long for N instructions?

- **Compare with other implementations:**
  - Single Cycle: (8ns clock period)

- **How much faster is pipelining for N=1000 ?**

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?**

Clock cycle

		1	2	3	4	5	6	7	8	9
add	$sp, $sp, -4	IF	ID	EX	WB					
sub	$v0, $a0, $a1		IF	ID	EX	WB				
lw	$t0, 4($sp)			IF	ID	EX	MEM	WB		
or	$s0, $s1, $s2				IF	ID	EX	WB		
lw	$t1, 8($sp)					IF	ID	EX	MEM	WB

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

Clock cycle

		1	2	3	4	5	6	7	8	9
add	$sp, $sp, -4	IF	ID	EX	WB					
sub	$v0, $a0, $a1		IF	ID	EX	WB				
lw	$t0, 4($sp)			IF	ID	EX	MEM	WB		
or	$s0, $s1, $s2				IF	ID	EX	WB		
lw	$t1, 8($sp)					IF	ID	EX	MEM	WB

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

R-type	IF	ID	EX	NOP	WB

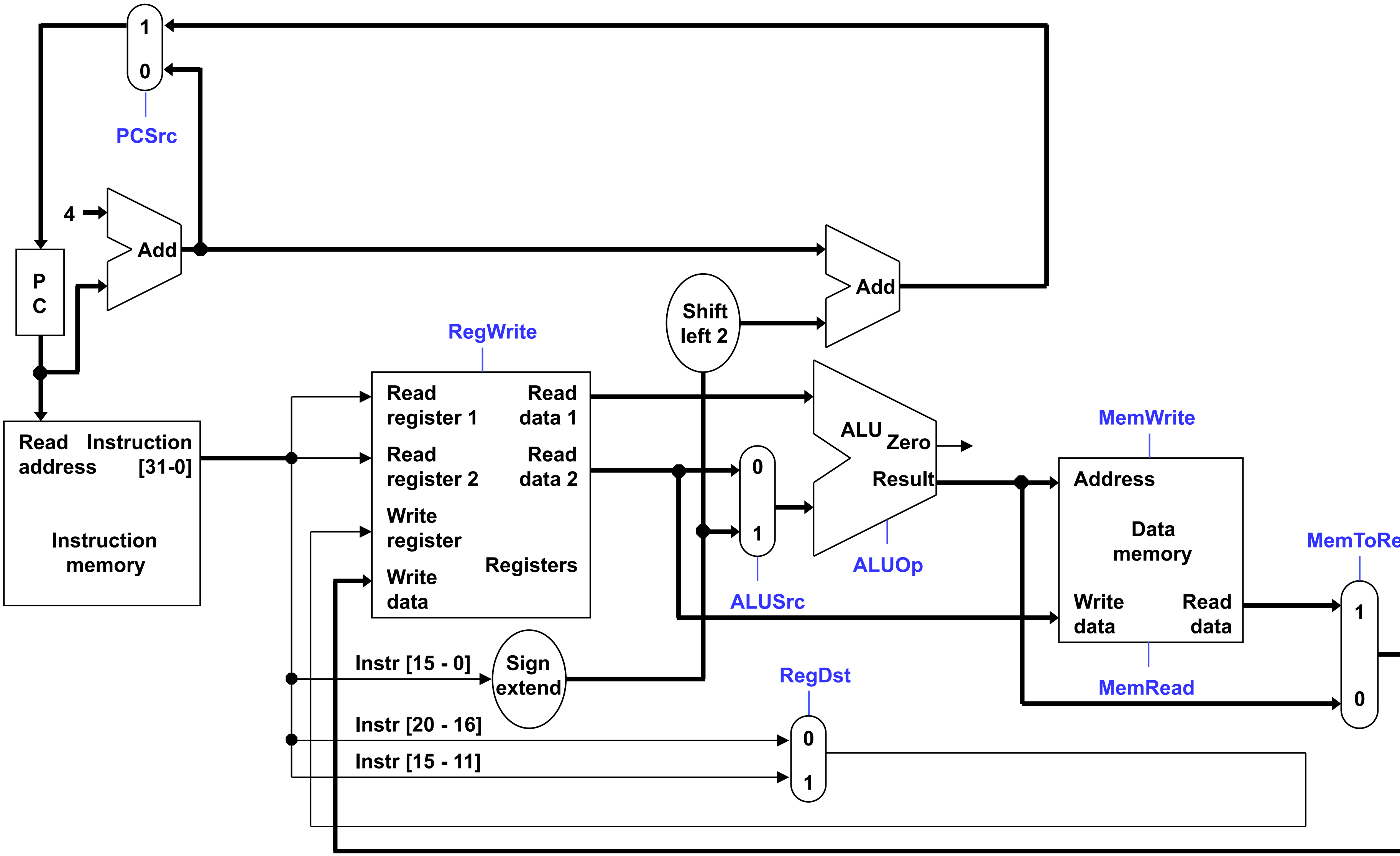
Clock cycle

		1	2	3	4	5	6	7	8	9
add	$sp, $sp, -4	IF	ID	EX	NOP	WB				
sub	$v0, $a0, $a1		IF	ID	EX	NOP	WB			
lw	$t0, 4($sp)			IF	ID	EX	MEM	WB		
or	$s0, $s1, $s2				IF	ID	EX	NOP	WB	
lw	$t1, 8($sp)					IF	ID	EX	MEM	WB

- Stores and Branches have NOP stages, too…

store	IF	ID	EX	MEM	NOP
branch	IF	ID	EX	NOP	NOP

Single-cycle datapath, slightly rearranged

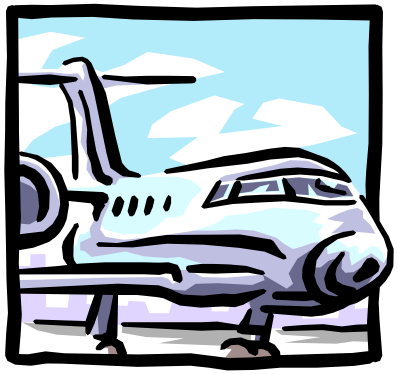


Pipeline registers

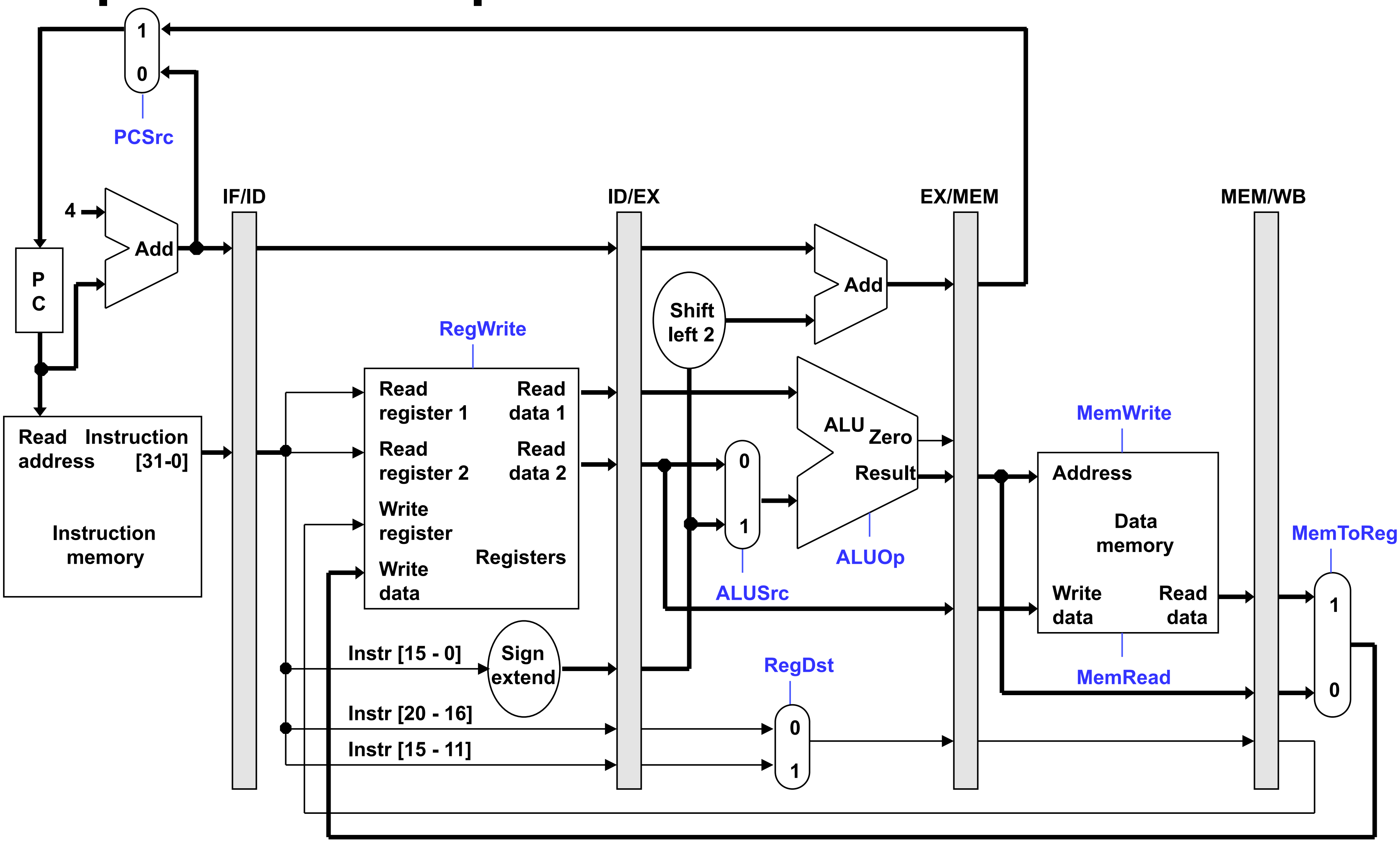
- We'll add intermediate registers to our pipelined datapath.
- There's a lot of information to save, however. We'll simplify our diagrams by drawing just one big pipeline register between each stage.
- The registers are named for the stages they connect.

IF/ID ID/EX EX/MEM MEM/WB

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



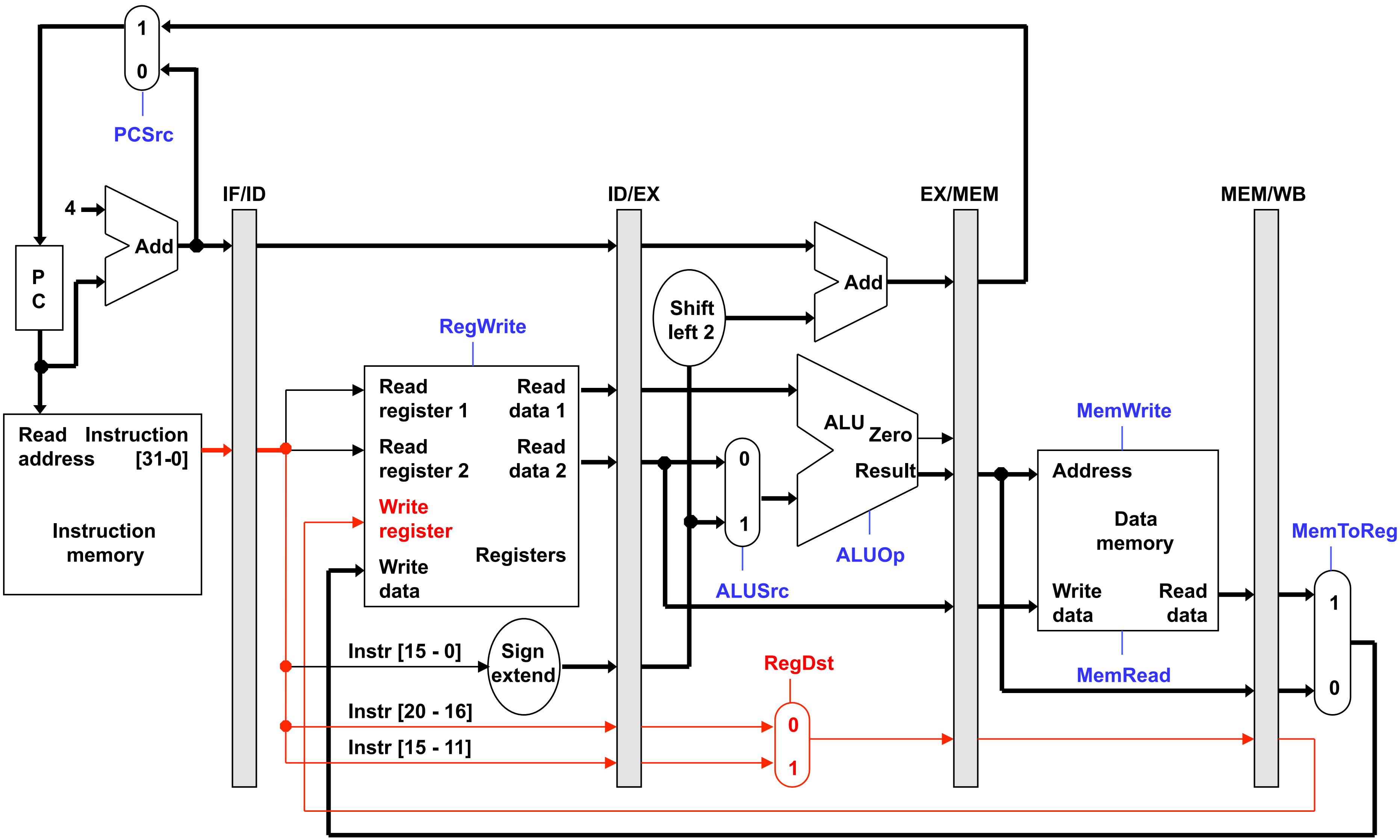
Pipelined datapath



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.**

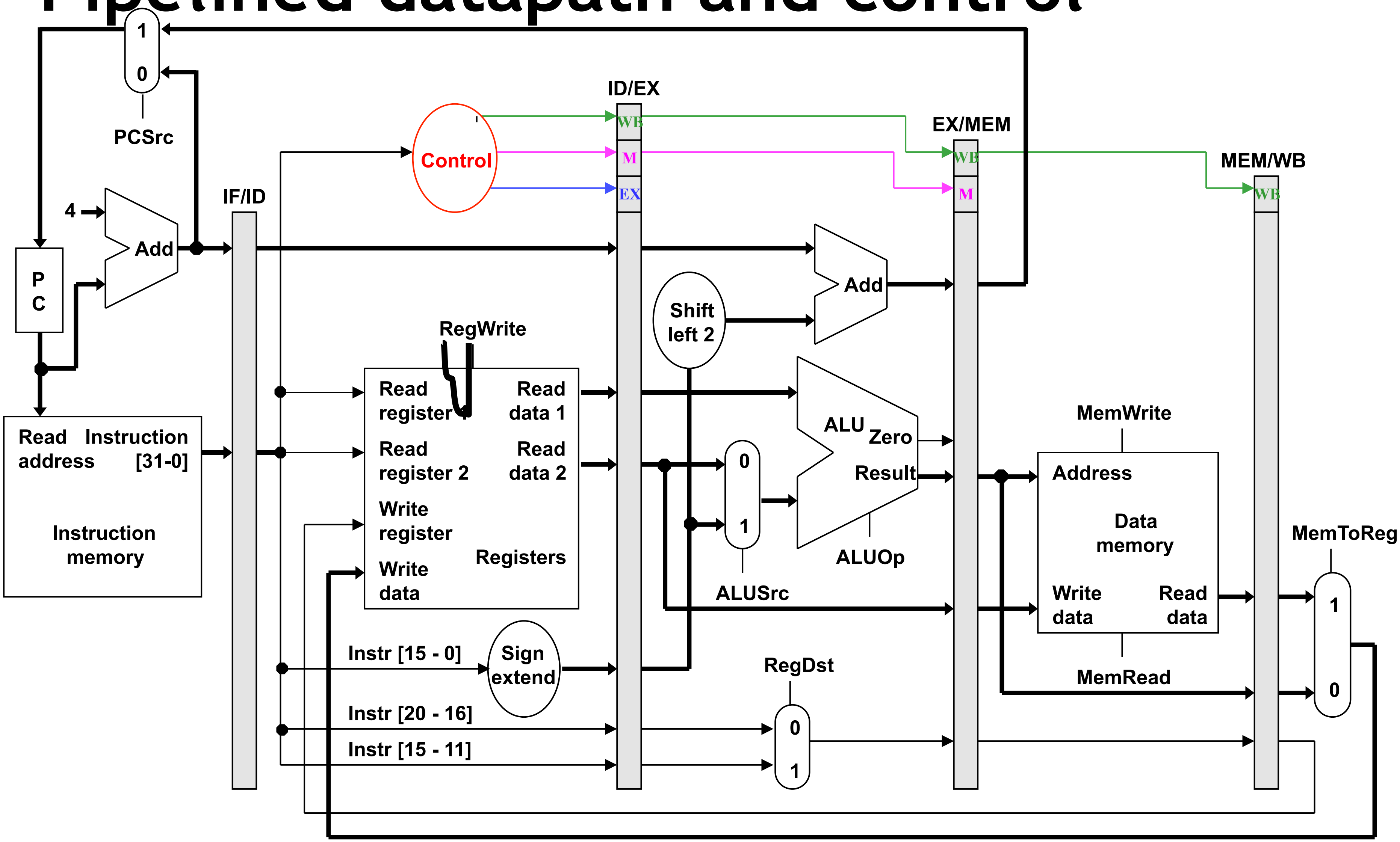
The destination register



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

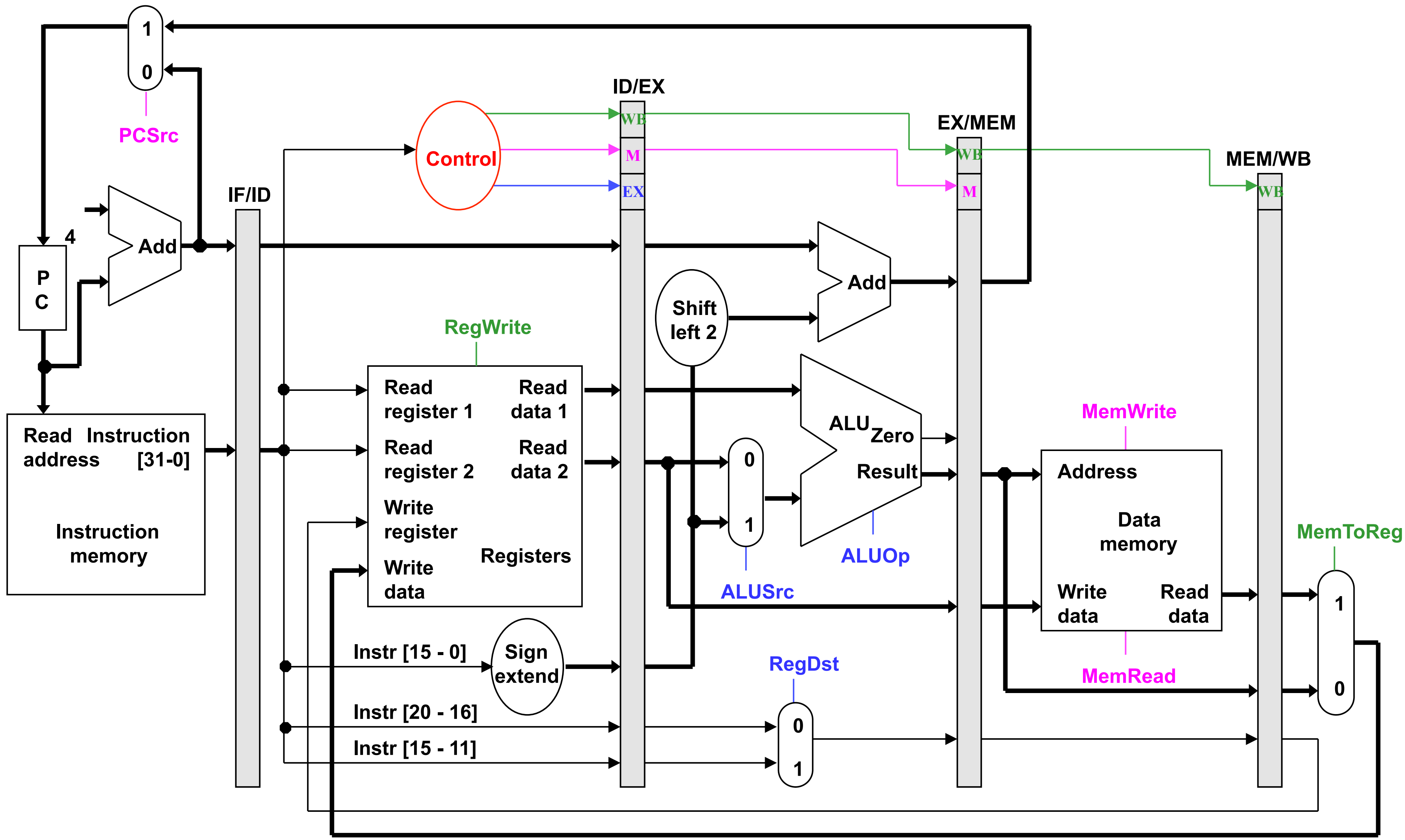


What about control signals?

- 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.

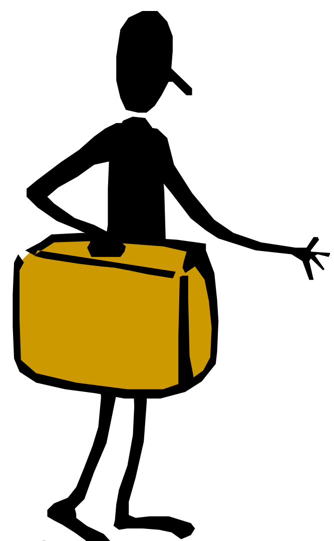
Stage	Control signals needed		
EX	ALUSrc	ALUOp	RegDst
MEM	MemRead	MemWrite	PCSrc
WB	RegWrite	MemToReg	

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.



25

An example execution sequence

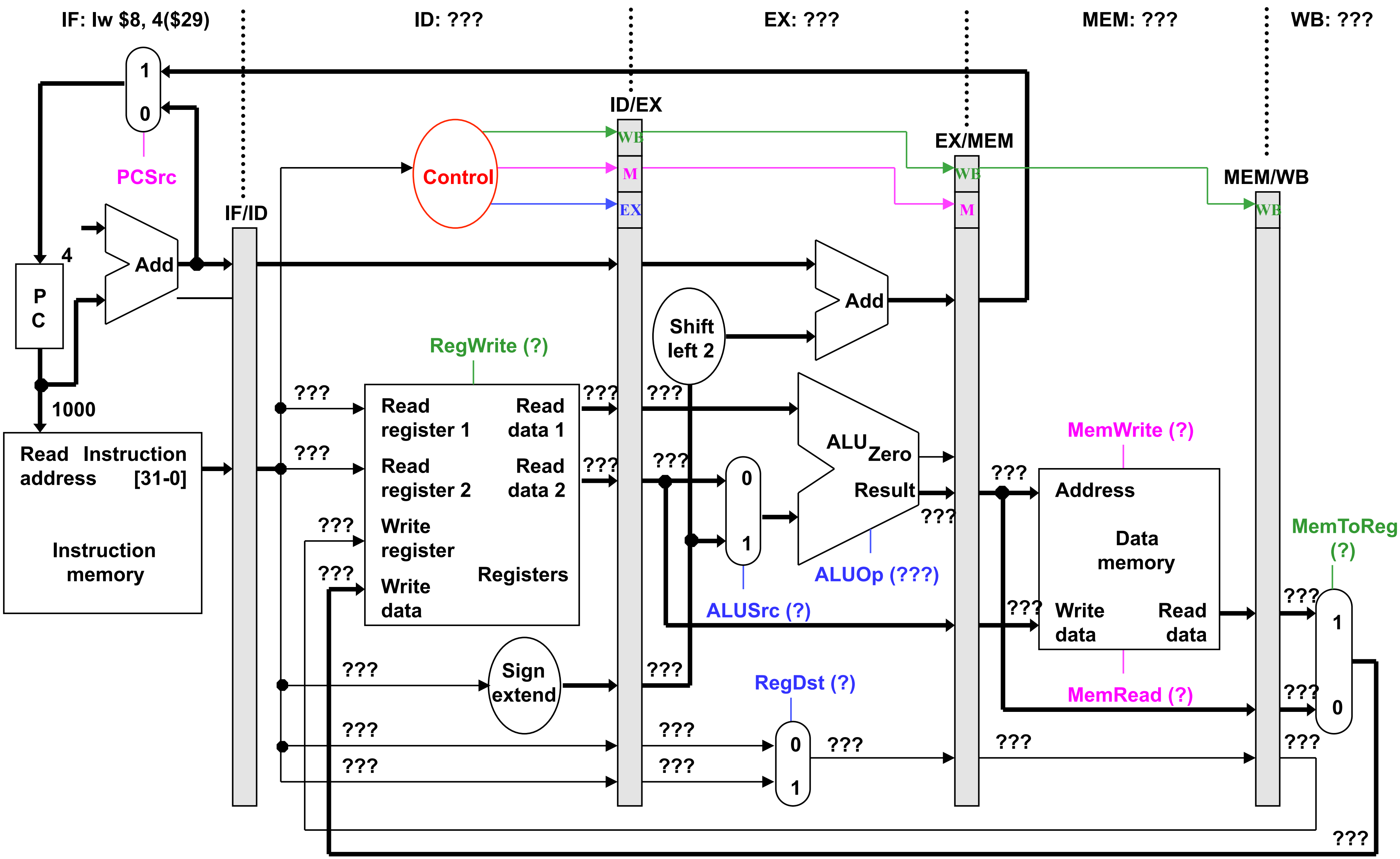
- **Here's a sample sequence of instructions to execute.**

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
```

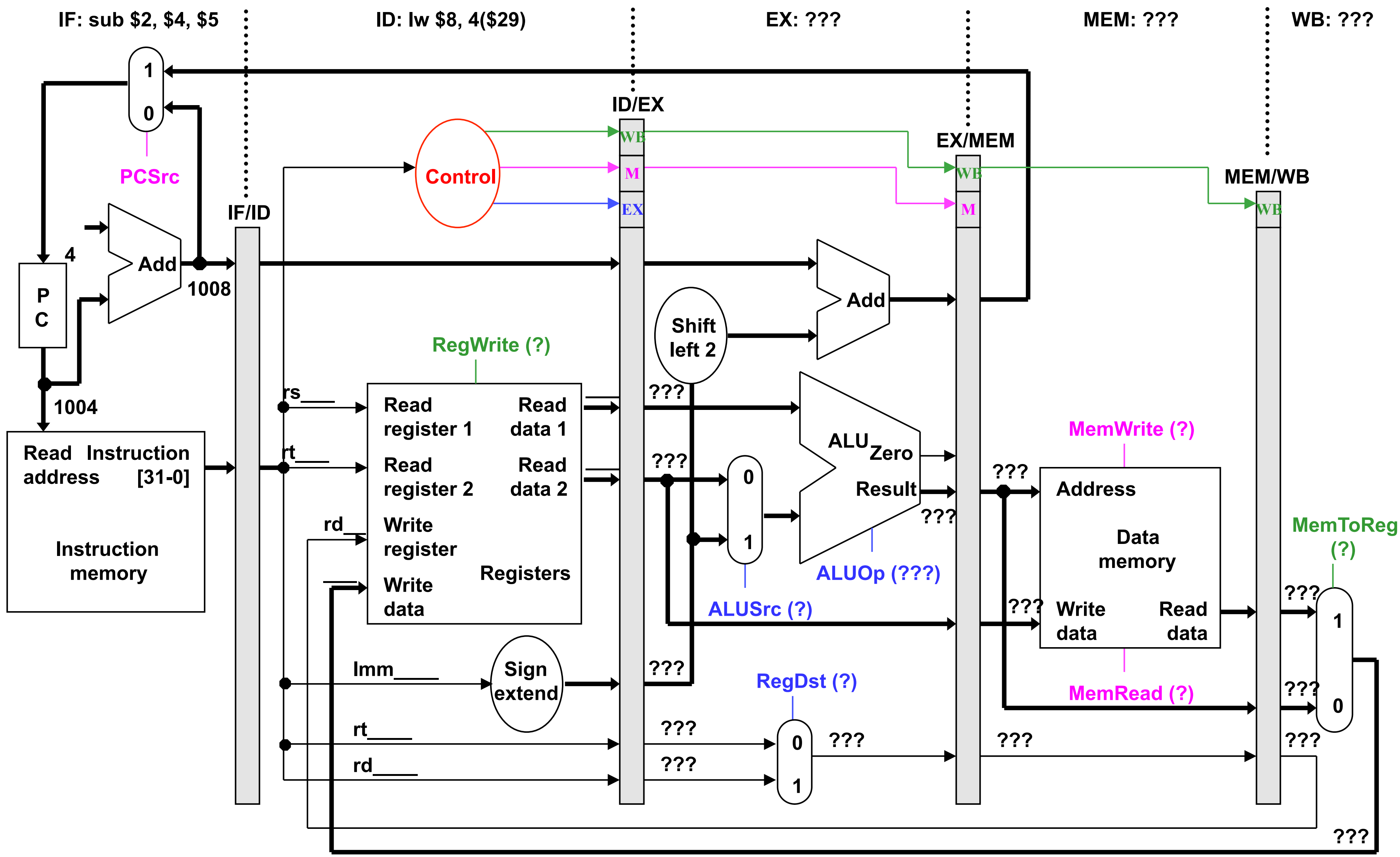
- **We'll make some assumptions, just so we can show actual data values.**
  - Each register contains its number plus 100. For instance, register $8 contains 108, register $29 contains 129, and so forth.
  - Every data memory location contains 99.
- **Our pipeline diagrams will follow some conventions.**
  - 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)

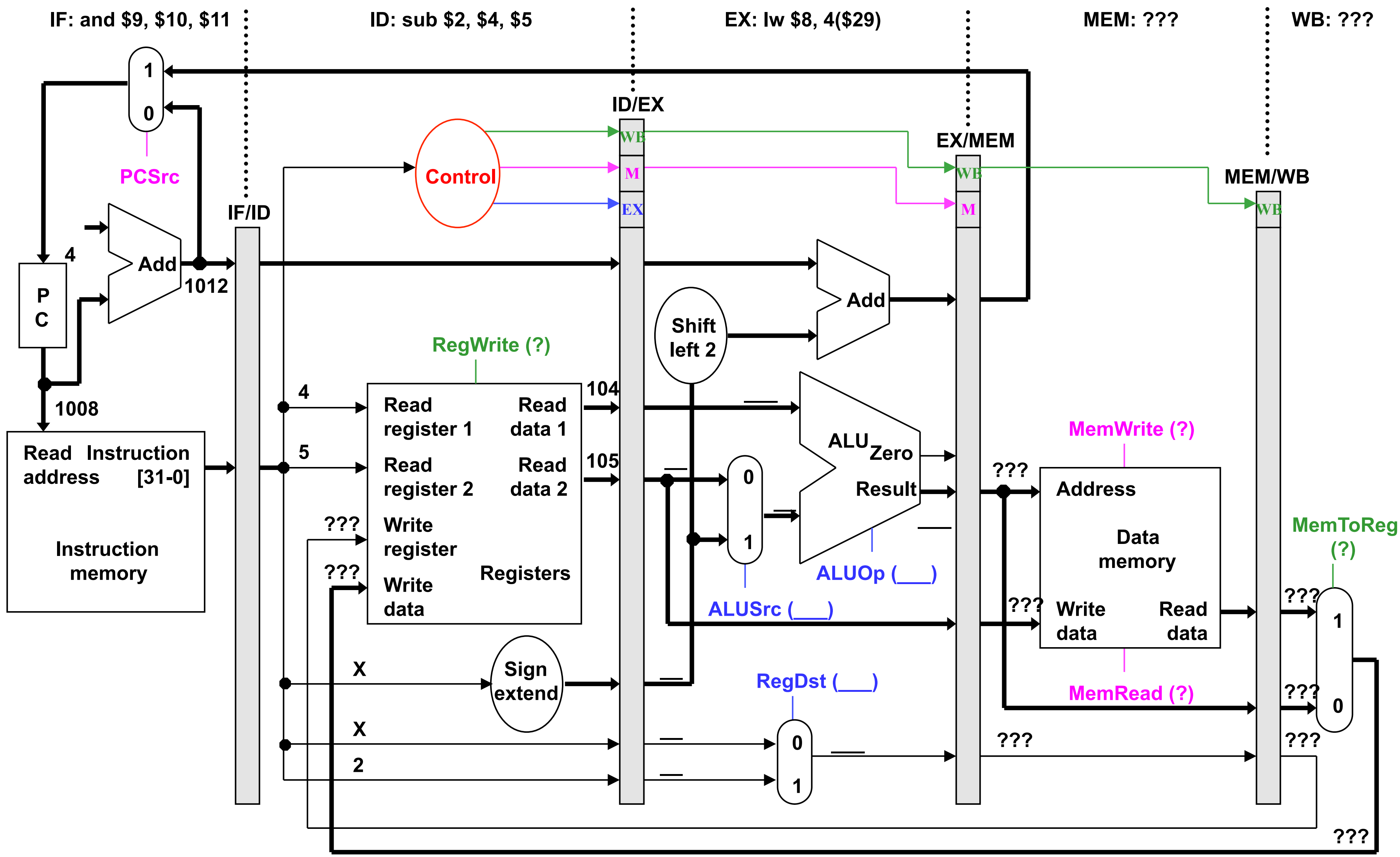


27

Cycle 2

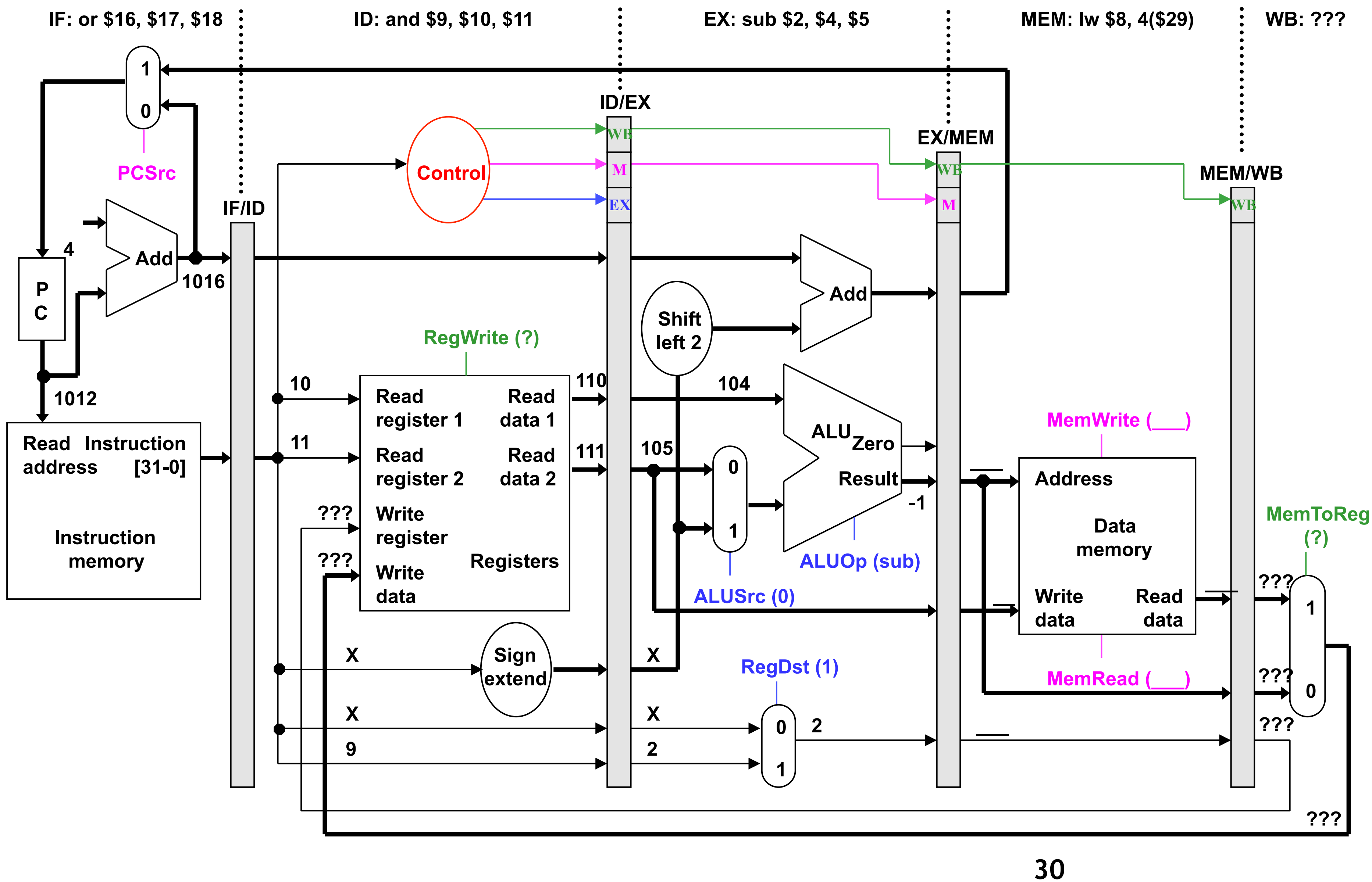

IF: sub $2, $4, $5
ID: lw $8, 4($29)
EX: ???
MEM: ???
WB: ???
1
0
PCSrc
IF/ID
ID/EX
EX/MEM
MEM/WB
Control
WB
M
EX
WB
M
WB
4
Add
1008
P
C
1004
Read address
Instruction [31-0]
Instruction memory
Add
Shift left 2
RegWrite (?)
rs
rt
rd
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
???
ALU
Zero
Result
0
1
ALUSrc (?)
ALUOp (???)
MemWrite (?)
Address
Data memory
Write data
Read data
MemRead (?)
MemToReg (?)
Imm
Sign extend
RegDst (?)
rt
rd
???

Cycle 3



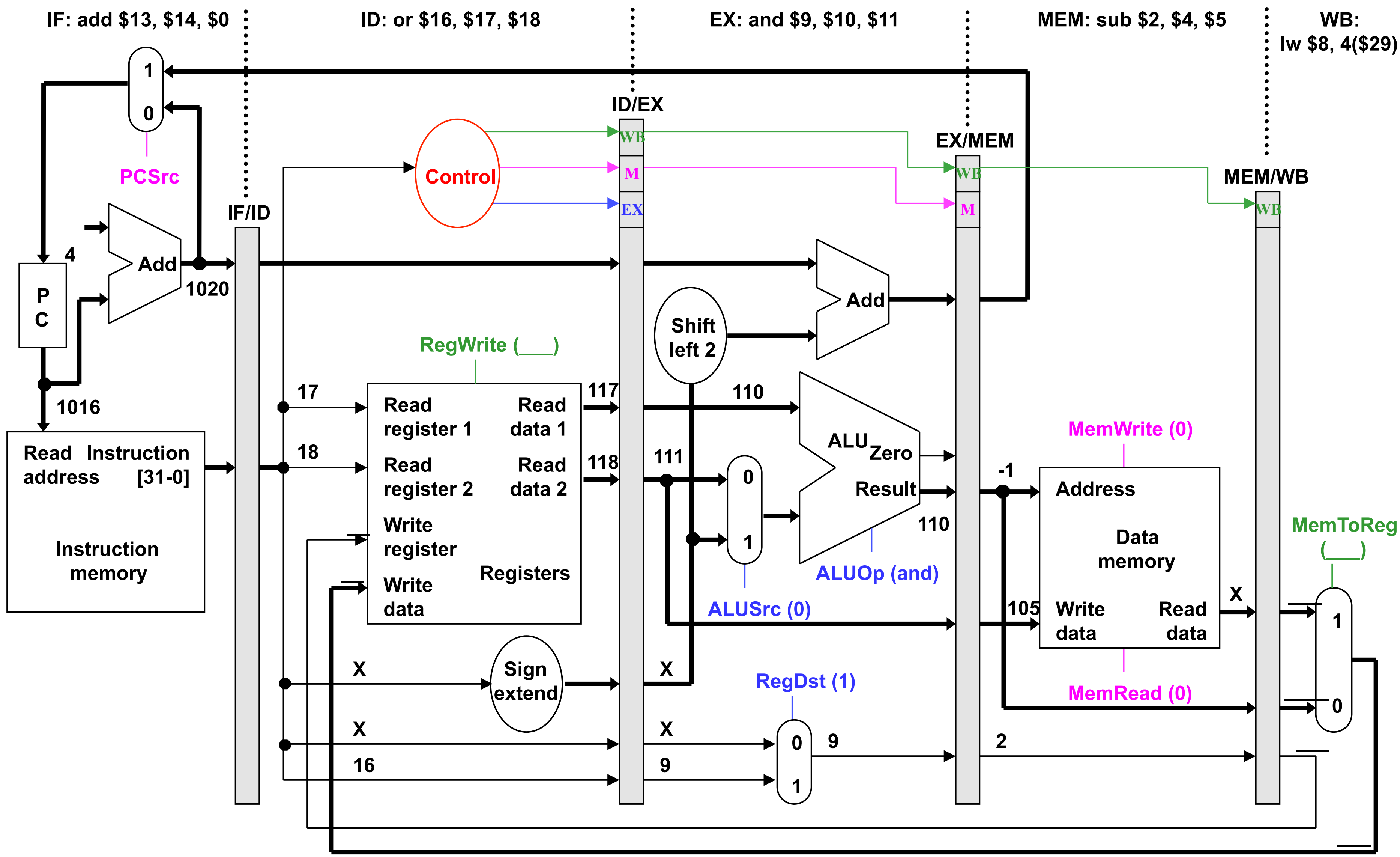
IF: and $9, $10, $11
ID: sub $2, $4, $5
EX: lw $8, 4($29)
MEM: ???
WB: ???
1
0
PCSrc
Control
ID/EX
WB
M
EX
EX/MEM
WB
M
MEM/WB
WB
IF/ID
4
Add
1012
P
C
1008
Read address
Instruction [31-0]
Instruction memory
Add
Shift left 2
RegWrite (?)
4
5
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
104
105
???
???
ALU
Zero
Result
0
1
ALUOp (___)
ALUSrc (___)
MemWrite (?)
Address
Data memory
Write data
Read data
MemRead (?)
MemToReg (?)
X
Sign extend
X
2
RegDst (___)
0
1
???

Cycle 4



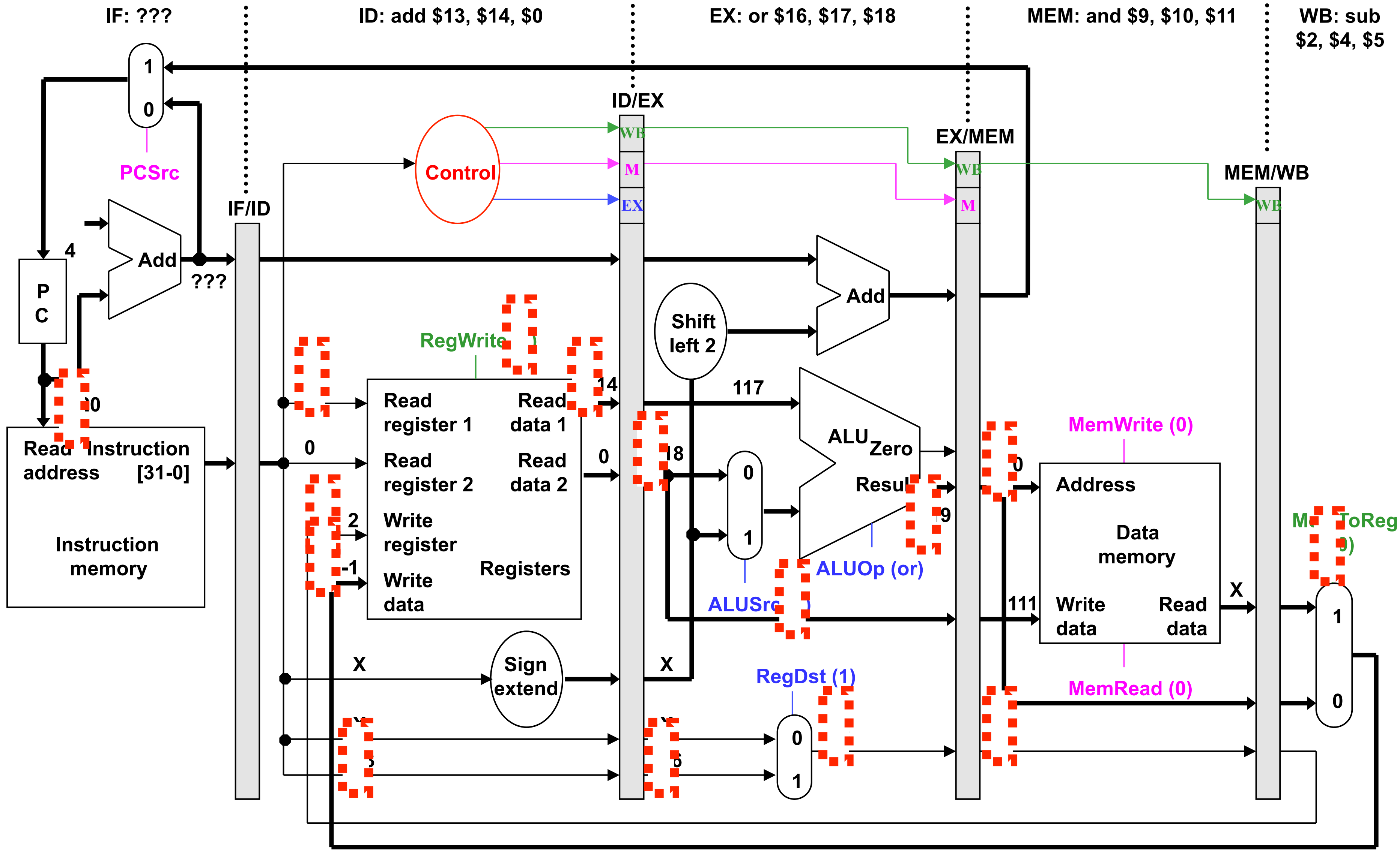
IF: or $16, $17, $18
ID: and $9, $10, $11
EX: sub $2, $4, $5
MEM: lw $8, 4($29)
WB: ???
PCSrc
1
0
IF/ID
ID/EX
EX/MEM
MEM/WB
Control
WB
M
EX
Add
4
P C
1016
1012
Read address
Instruction [31-0]
Instruction memory
RegWrite (?)
10
11
???
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
110
111
Shift left 2
104
105
ALU
Zero
Result
-1
ALUSrc (0)
ALUOp (sub)
MemWrite (___)
MemRead (___)
Address
Data memory
Write data
Read data
MemToReg (?)
X
9
2
Sign extend
RegDst (1)
30

Cycle 5 (full)



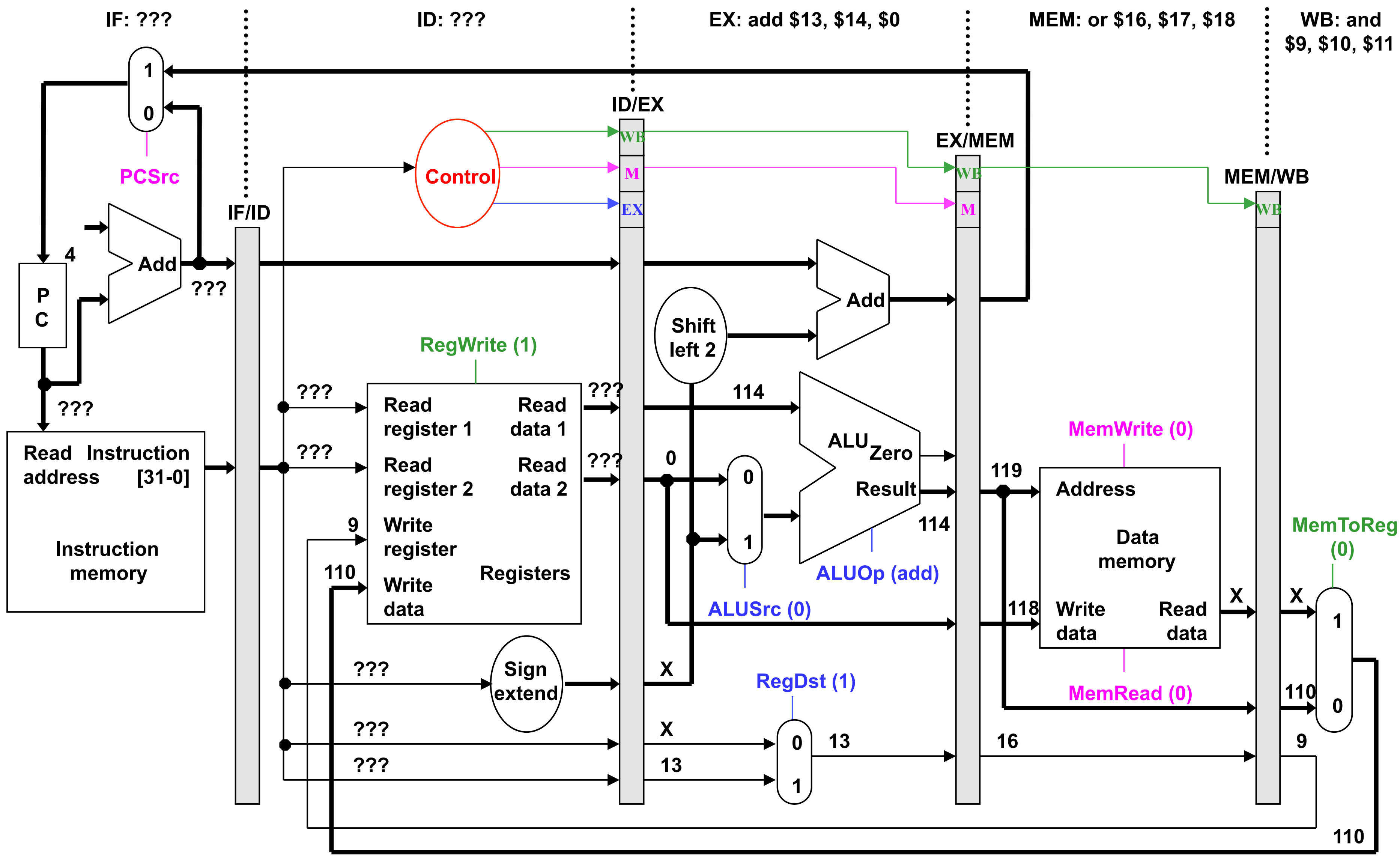
31

Cycle 6 (emptying)



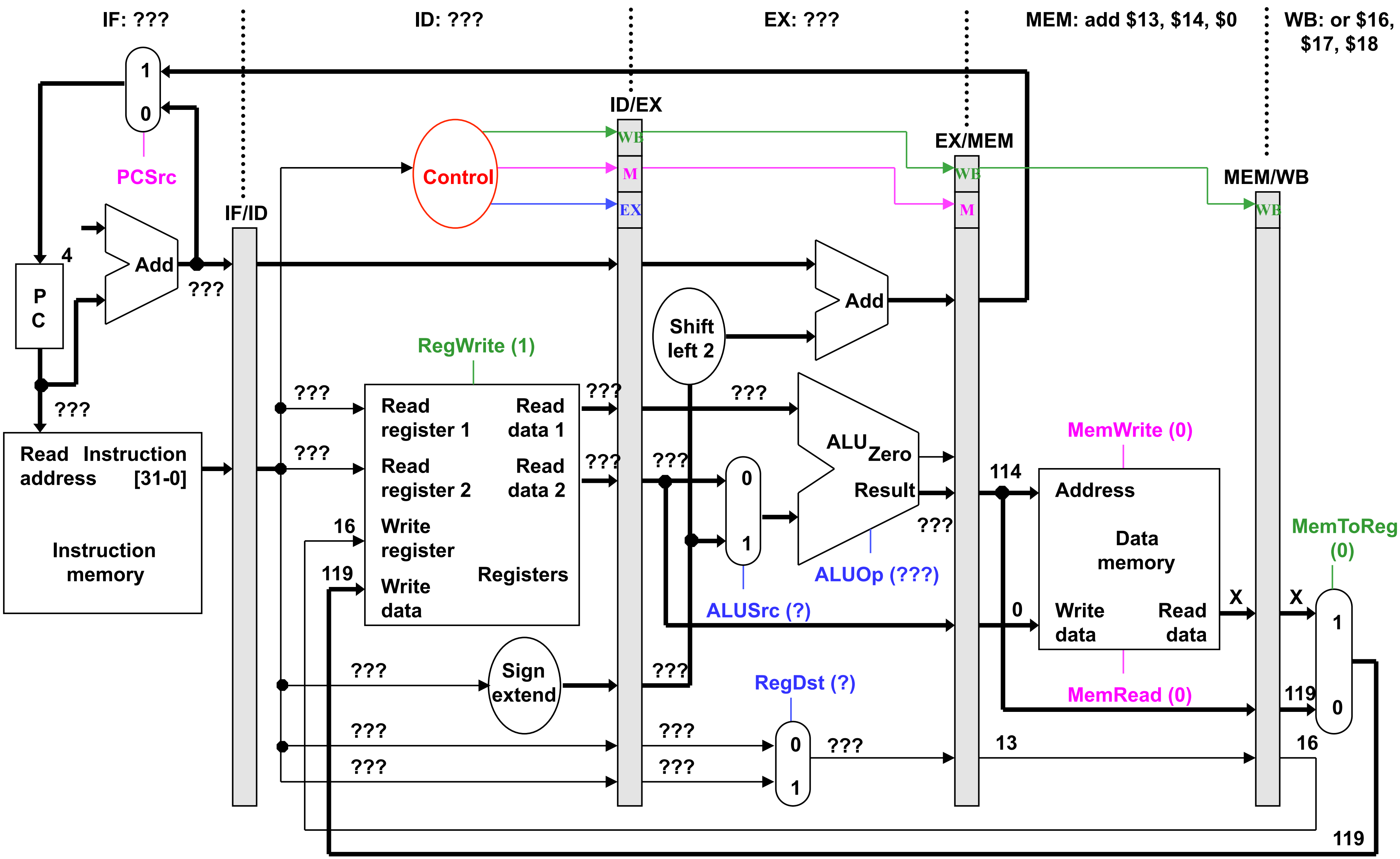
32

Cycle 7



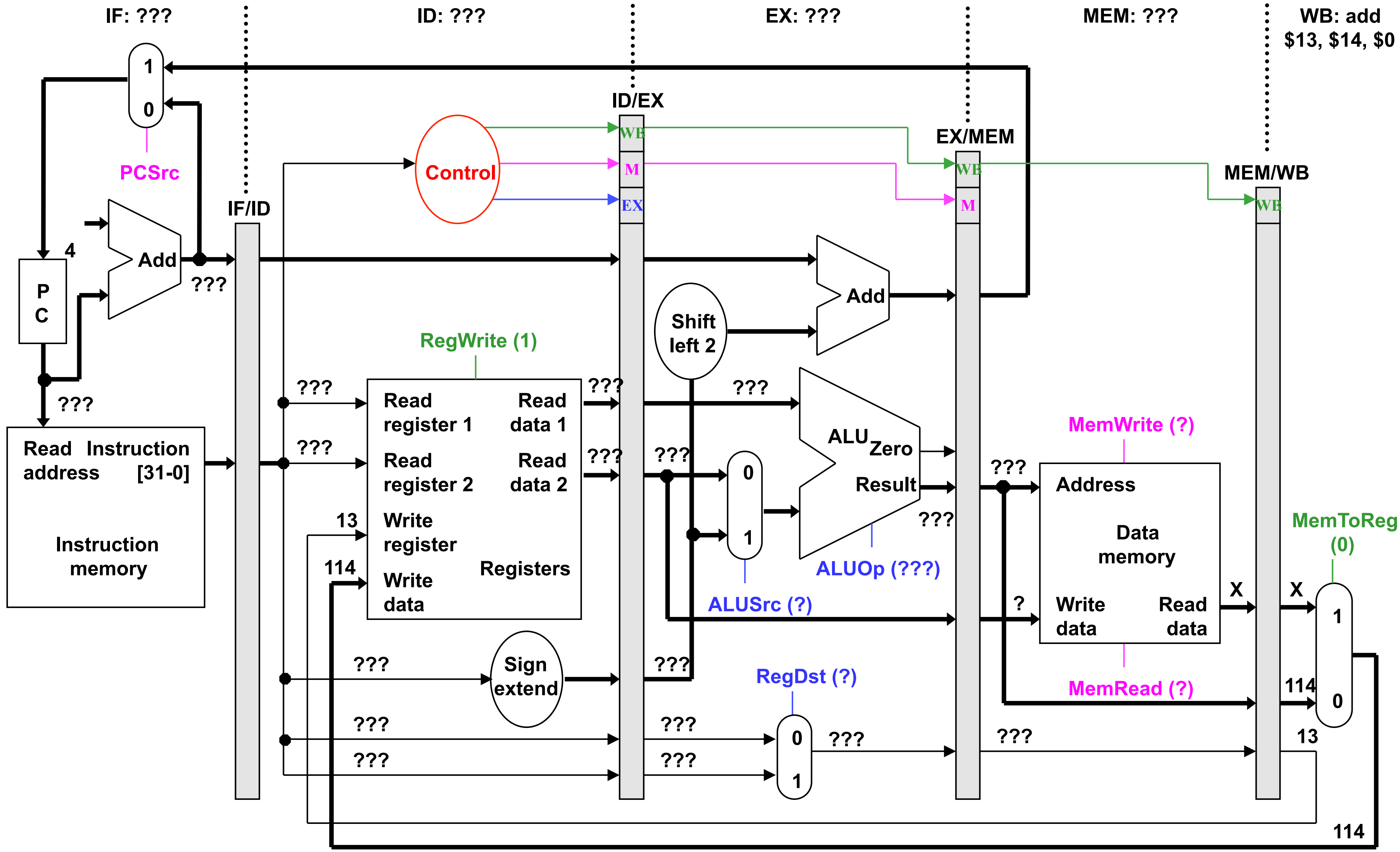
IF: ???
ID: ???
EX: add $13, $14, $0
MEM: or $16, $17, $18
WB: and $9, $10, $11
1
0
PCSrc
IF/ID
ID/EX
EX/MEM
MEM/WB
Control
WB
M
EX
WB
M
WB
4
P C
Add
???
Add
Shift left 2
RegWrite (1)
???
???
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
???
???
9
110
114
0
0
1
ALU
Zero
Result
114
ALUOp (add)
ALUSrc (0)
119
MemWrite (0)
Address
Data memory
Write data
Read data
118
MemRead (0)
X
X
MemToReg (0)
1
0
110
???
Read address
Instruction [31-0]
Instruction memory
???
Sign extend
X
???
X
???
13
RegDst (1)
0
1
13
16
9
110

Cycle 8



IF: ???
ID: ???
EX: ???
MEM: add $13, $14, $0
WB: or $16, $17, $18
IF/ID
ID/EX
EX/MEM
MEM/WB
PCSrc
Control
WB
M
EX
Add
4
P C
???
Read address
Instruction [31-0]
Instruction memory
RegWrite (1)
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
16
119
Sign extend
Shift left 2
ALU
Zero
Result
ALUOp (???)
ALUSrc (?)
RegDst (?)
MemWrite (0)
MemRead (0)
114
0
13
Address
Data memory
Write data
Read data
X
MemToReg (0)
119
16

Cycle 9


IF: ???
ID: ???
EX: ???
MEM: ???
WB: add $13, $14, $0
1
0
PCSrc
IF/ID
ID/EX
EX/MEM
MEM/WB
Control
WB
M
EX
WB
M
WB
4
P C
Add
???
???
Read address
Instruction [31-0]
Instruction memory
RegWrite (1)
???
???
Read register 1
Read register 2
13
Write register
114
Write data
Read data 1
Read data 2
Registers
???
???
Shift left 2
Add
???
???
0
1
ALU
Zero
Result
ALUOp (???)
ALUSrc (?)
???
???
MemWrite (?)
Address
Data memory
Write data
Read data
?
MemRead (?)
X
X
MemToReg (0)
1
0
114
13
???
Sign extend
???
???
???
???
???
RegDst (?)
0
1
???
???
114

35

That's a lot of diagrams there

Clock cycle

		1	2	3	4	5	6	7	8	9
lw	$t0, 4($sp)	IF	ID	EX	MEM	WB				
sub	$v0, $a0, $a1		IF	ID	EX	MEM	WB			
and	$t1, $t2, $t3			IF	ID	EX	MEM	WB		
or	$s0, $s1, $s2				IF	ID	EX	MEM	WB	
add	$t5, $t6, $0					IF	ID	EX	MEM	WB

- **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.**

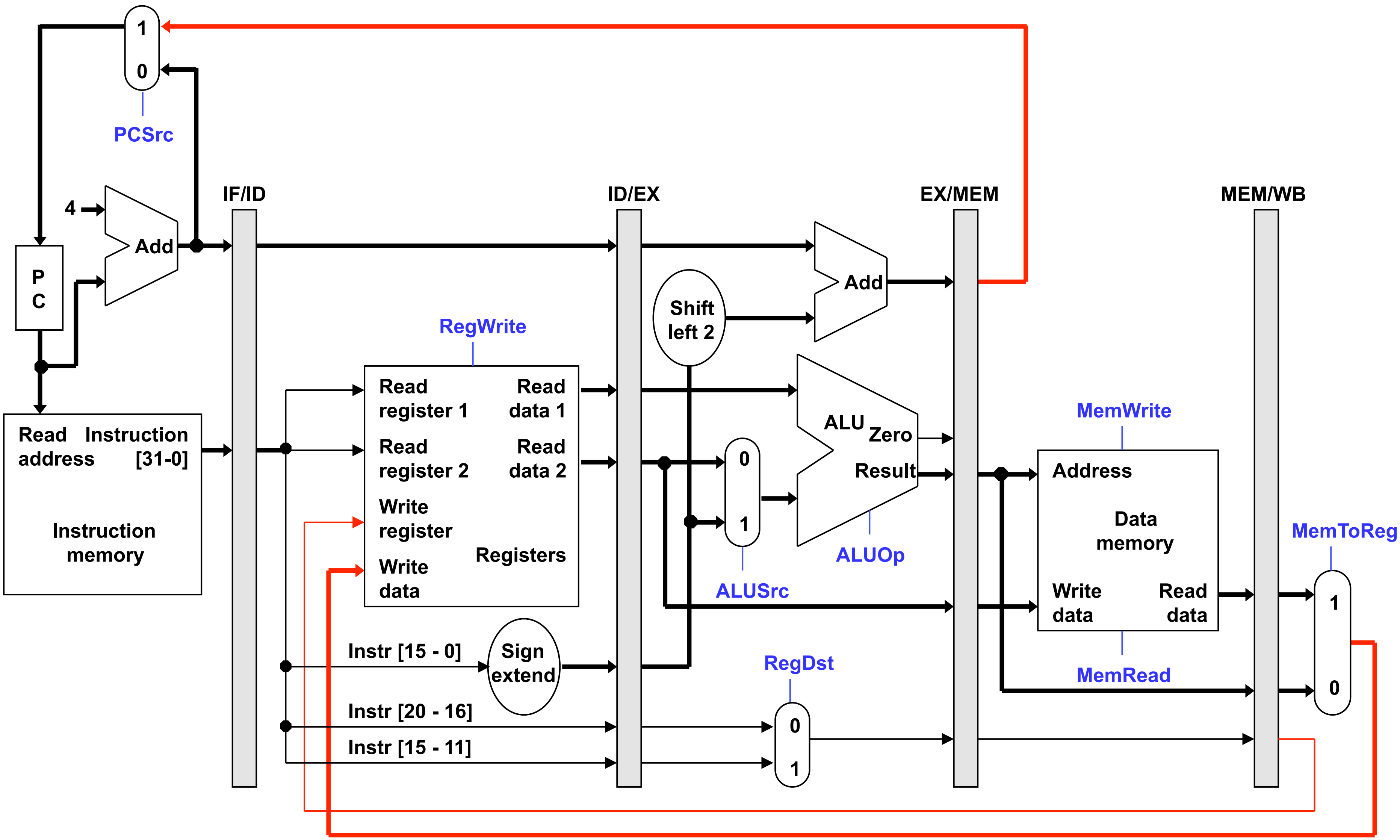
36

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)