Pipelining the MIPS Datapath

This week + next week –

Pipelining

Bring handout back

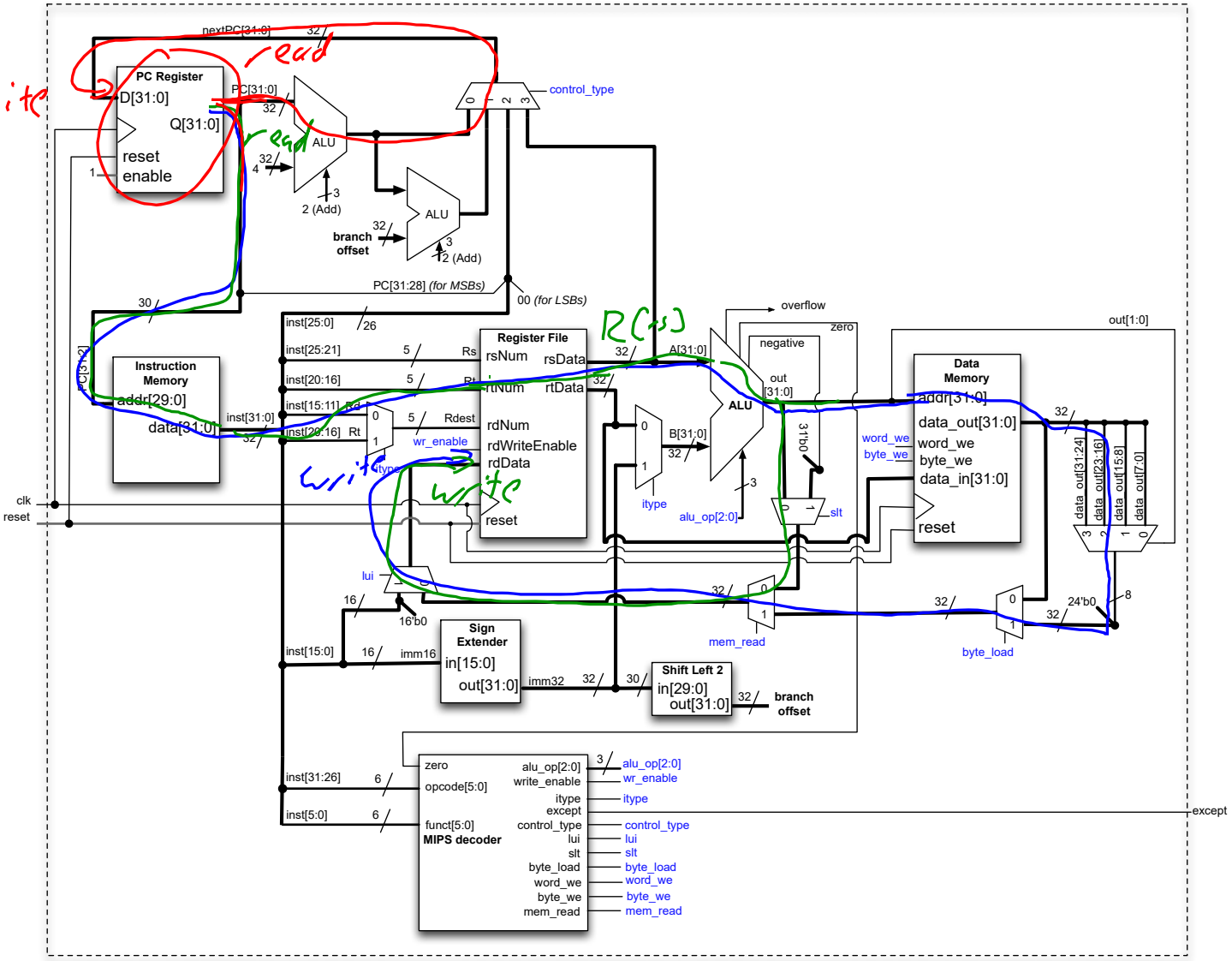
Today's lecture

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

We have built a single-cycle implementation of a subset of the MIPS-based instruction set

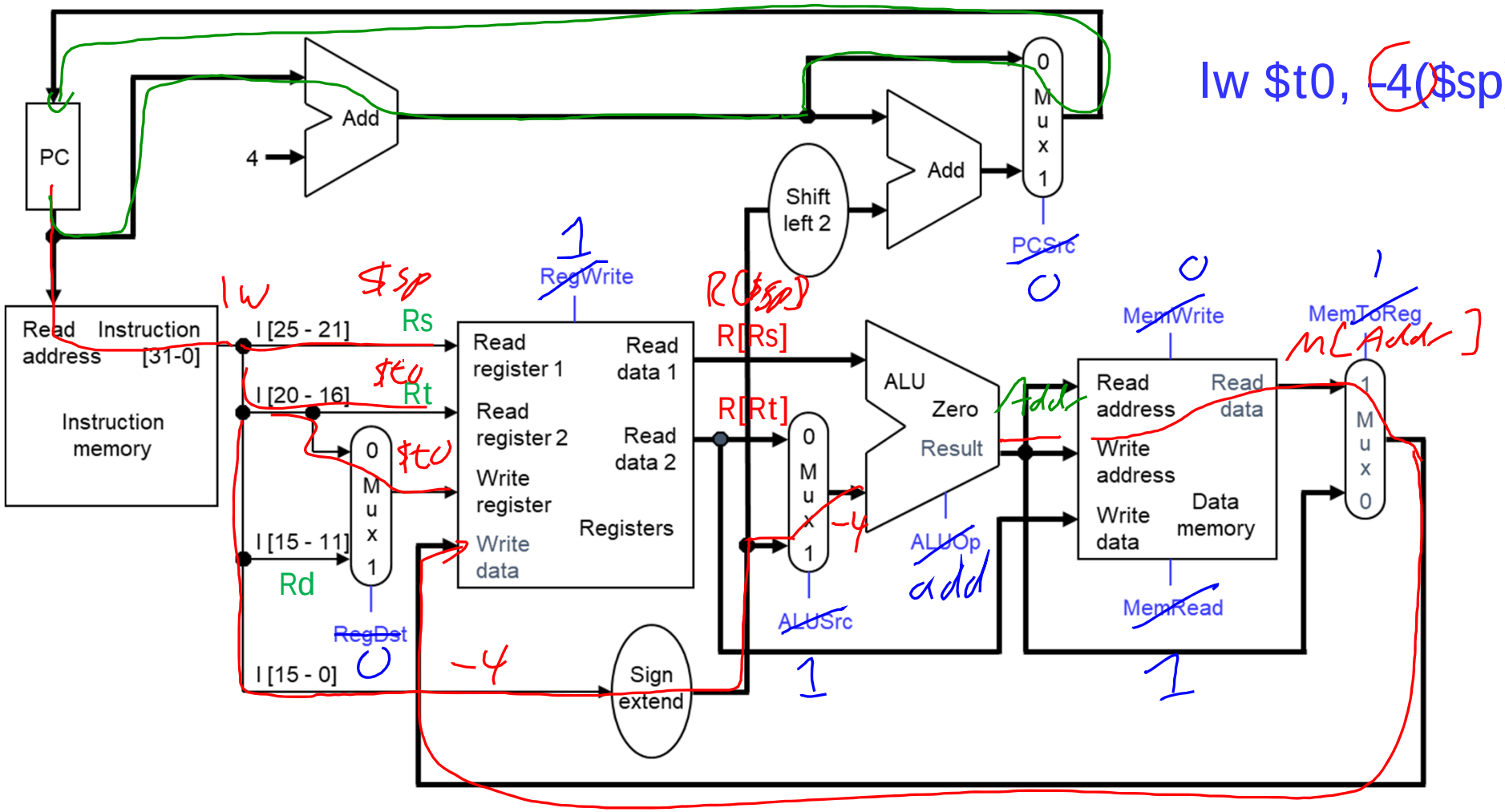
- We have assumed that instructions execute in the same amount of time; this determines the clock cycle time.
- We have implemented the datapath and the control unit.

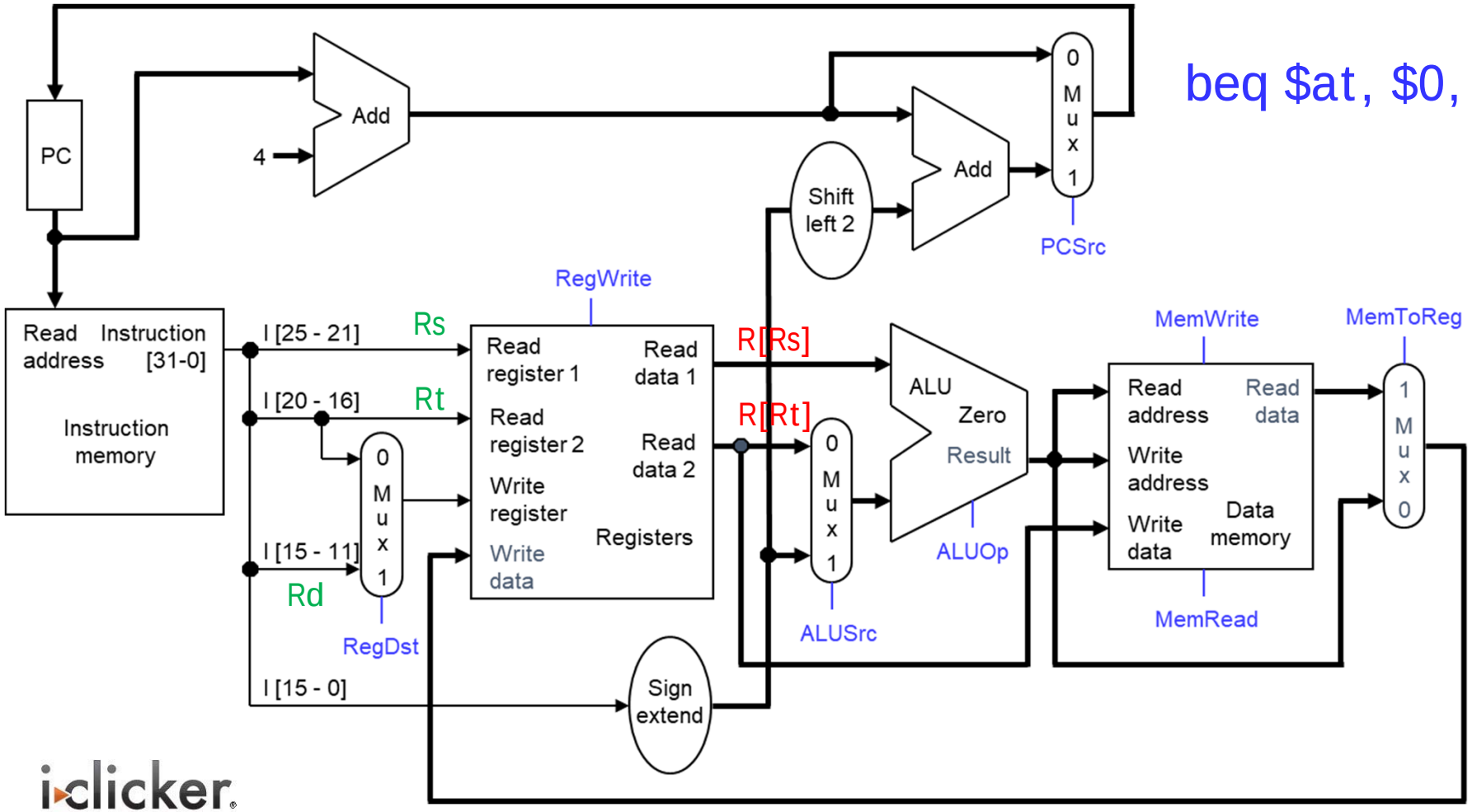
Refresher: The Full Single-cycle Datapath



We will use a simplified implementation of MIPS to create a pipelined version

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

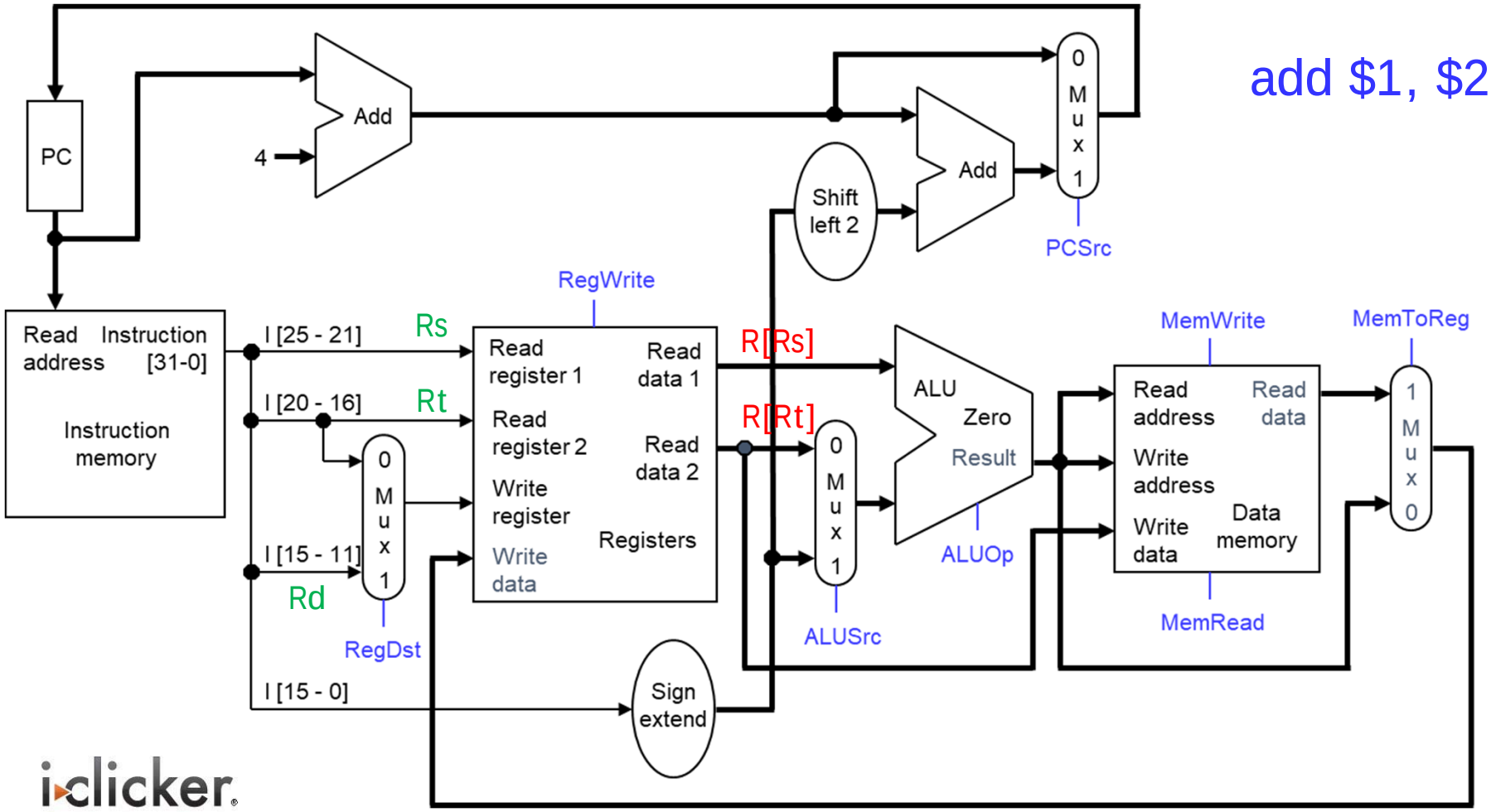

lw $t0, -4($sp)
PC
Add
4
Add
Shift left 2
0 M u x 1
PCSrc
RegWrite
Instruction address
Instruction [31-0]
Instruction memory
I [25 - 21]
I [20 - 16]
I [15 - 11]
I [15 - 0]
Rs
Rt
Rd
0 M u x 1
RegDst
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
Sign extend
ALU
Zero
Result
ALUOp
add
0 M u x 1
ALUSrc
MemWrite
MemRead
Read address
Write address
Write data
Read data
Data memory
MemToReg
1 M u x 0
R[$sp]
R[Rs]
R[Rt]
-4
M[Addr]



beq $at, $0, offset

i>clicker

A) ALUSrc=0 B) ALUSrc=1



i-clicker

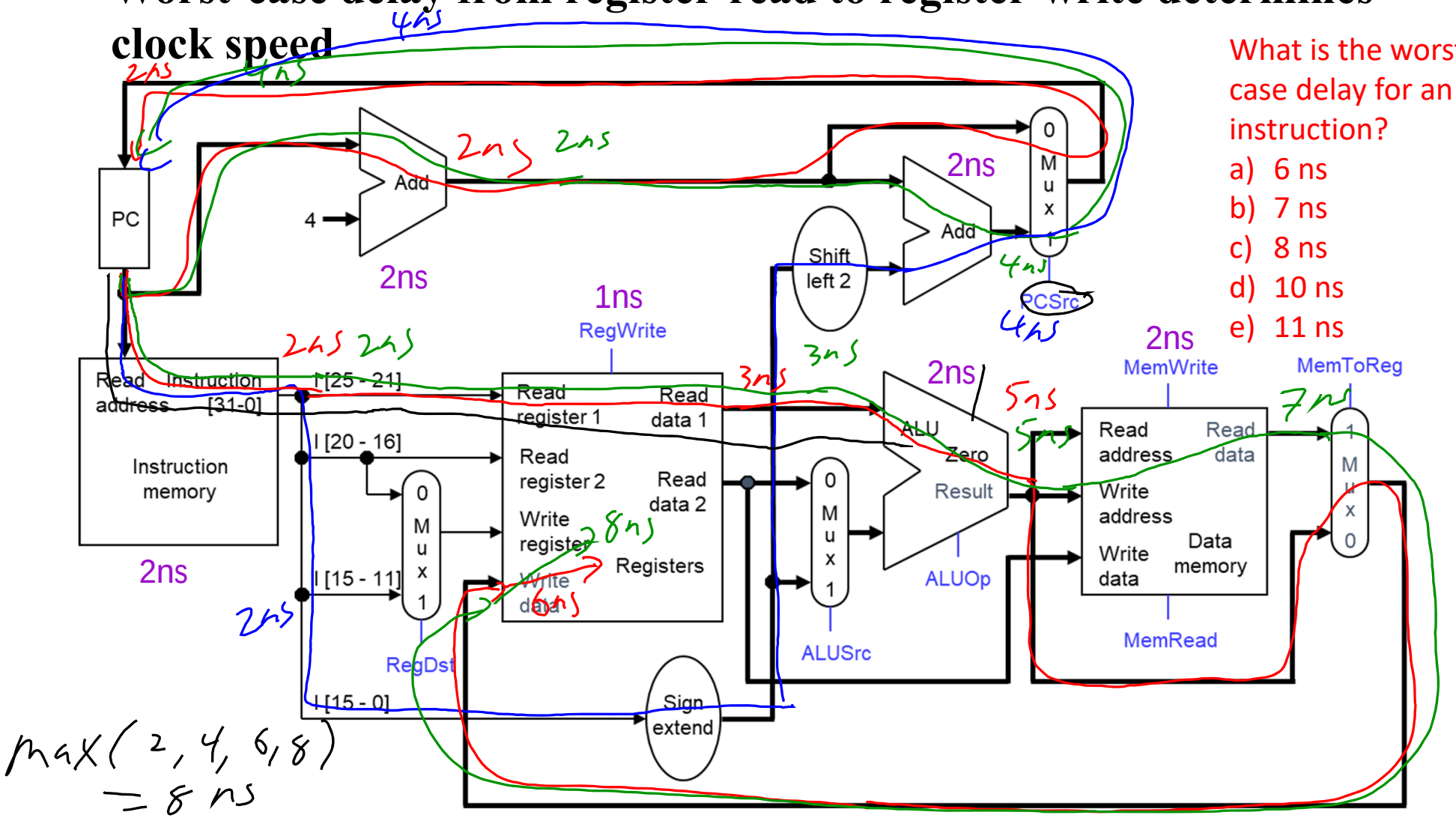
A) ALUSrc=0 RegDst=0

B) ALUSrc=0 RegDst=1

C) ALUSrc=1 RegDst=0

D) ALUSrc=1 RegDst=1

Worst-case delay from register-read to register-write determines clock speed

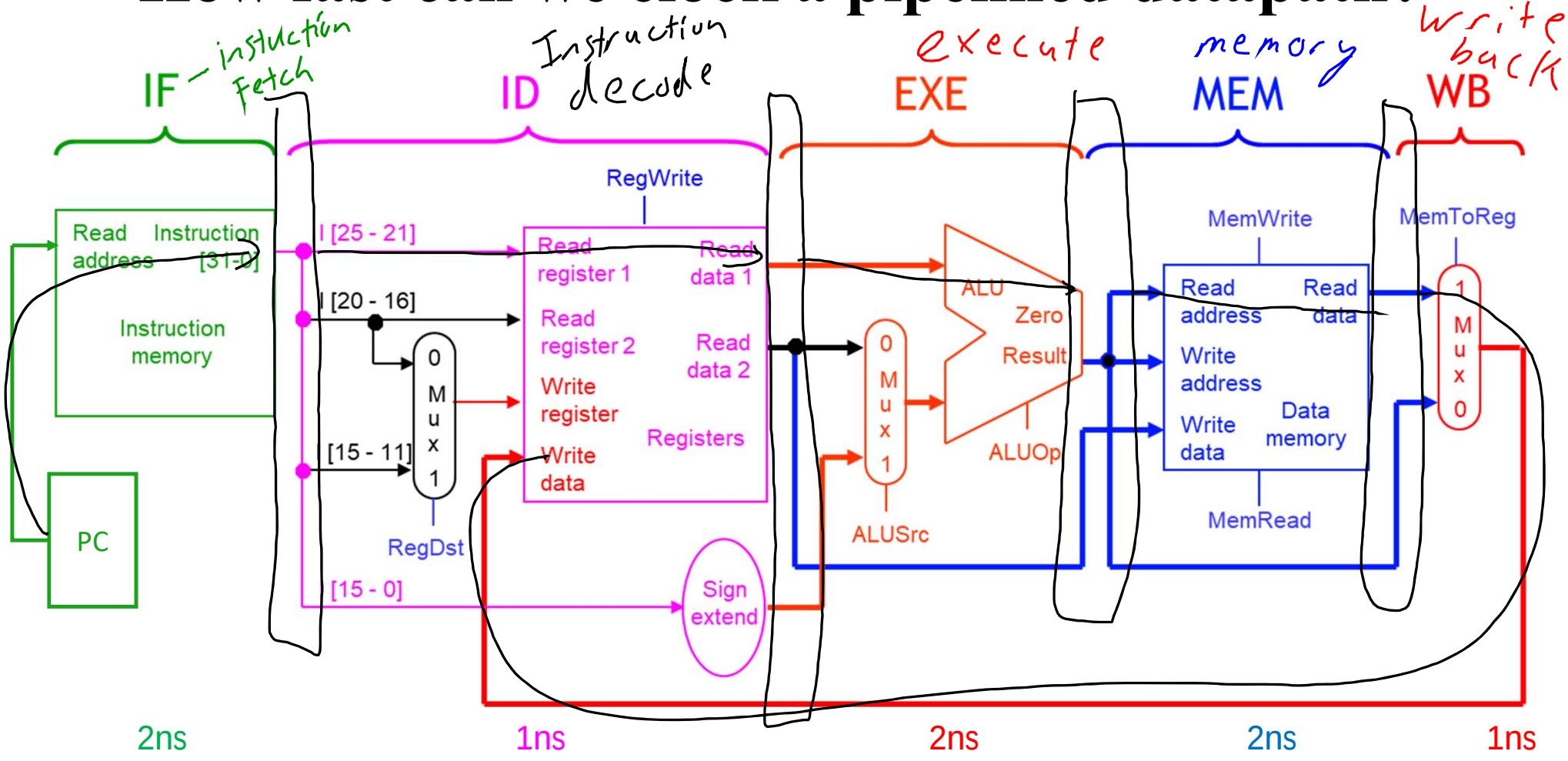


What is the worst-case delay for an instruction?

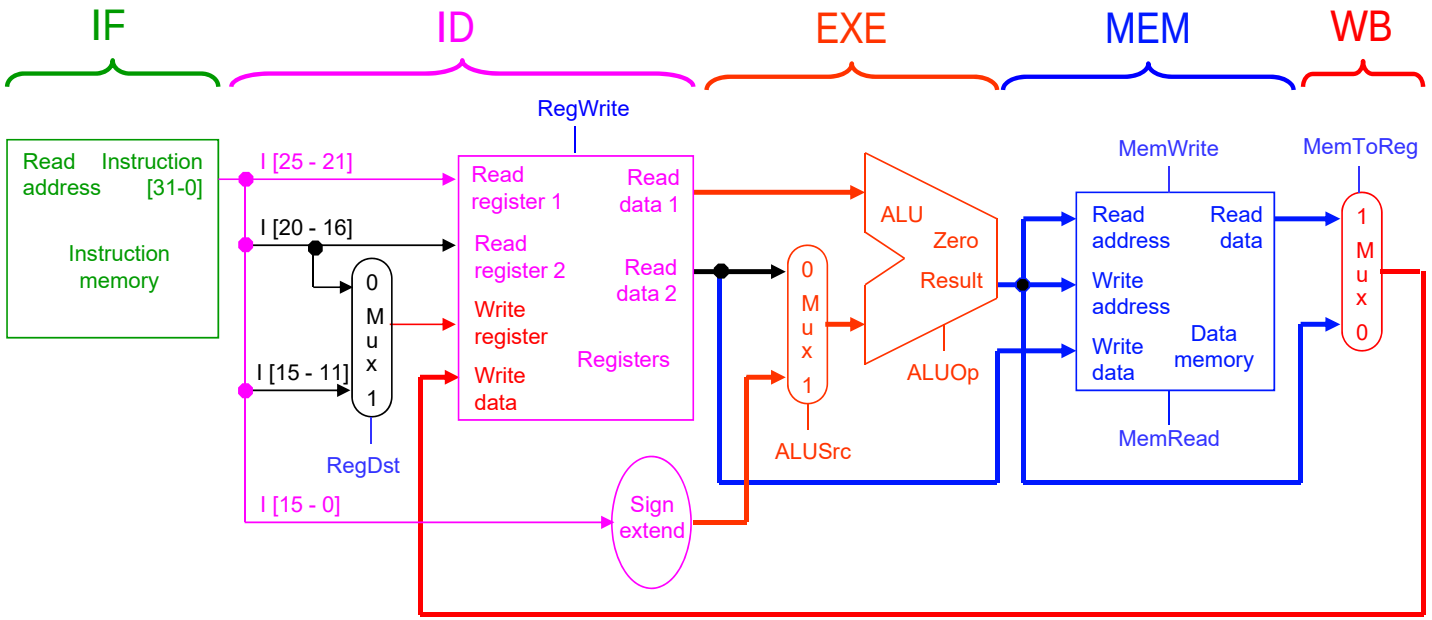
a) 6 ns
b) 7 ns
c) 8 ns
d) 10 ns
e) 11 ns

max(2, 4, 6, 8) = 8 ns

How fast can we clock a pipelined datapath?



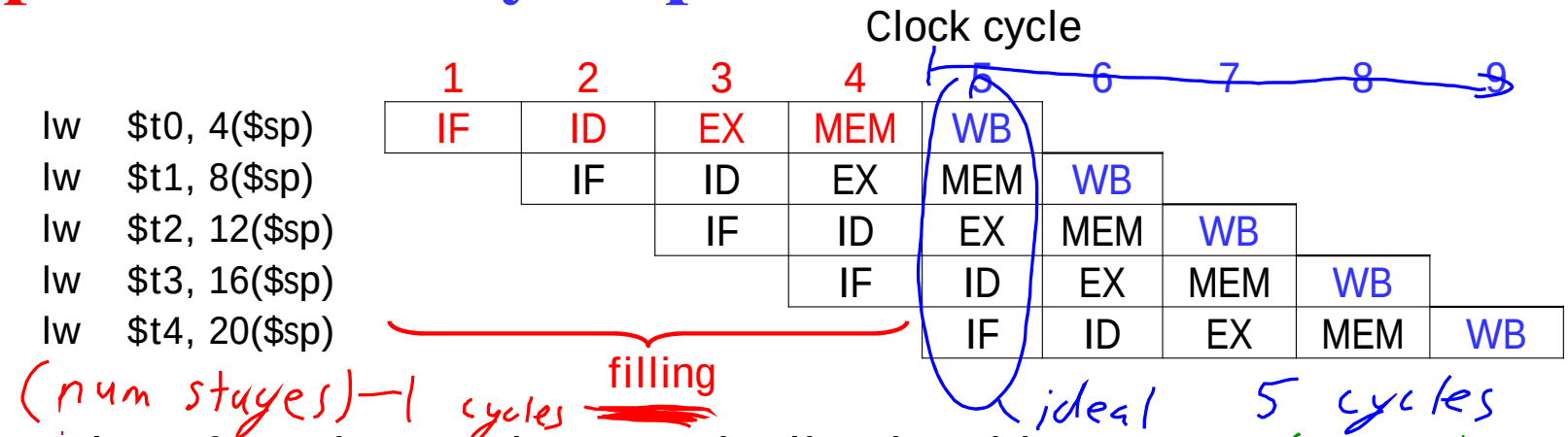
Break datapath into 5 stages



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

Ideal pipeline performance is time to fill the pipeline + one cycle per instruction



- How long for N instructions on pipelined architecture?

$(4+N)\cdot 2ns = 8+2N$ ns

$(4+5)\cdot 2ns = 18ns$

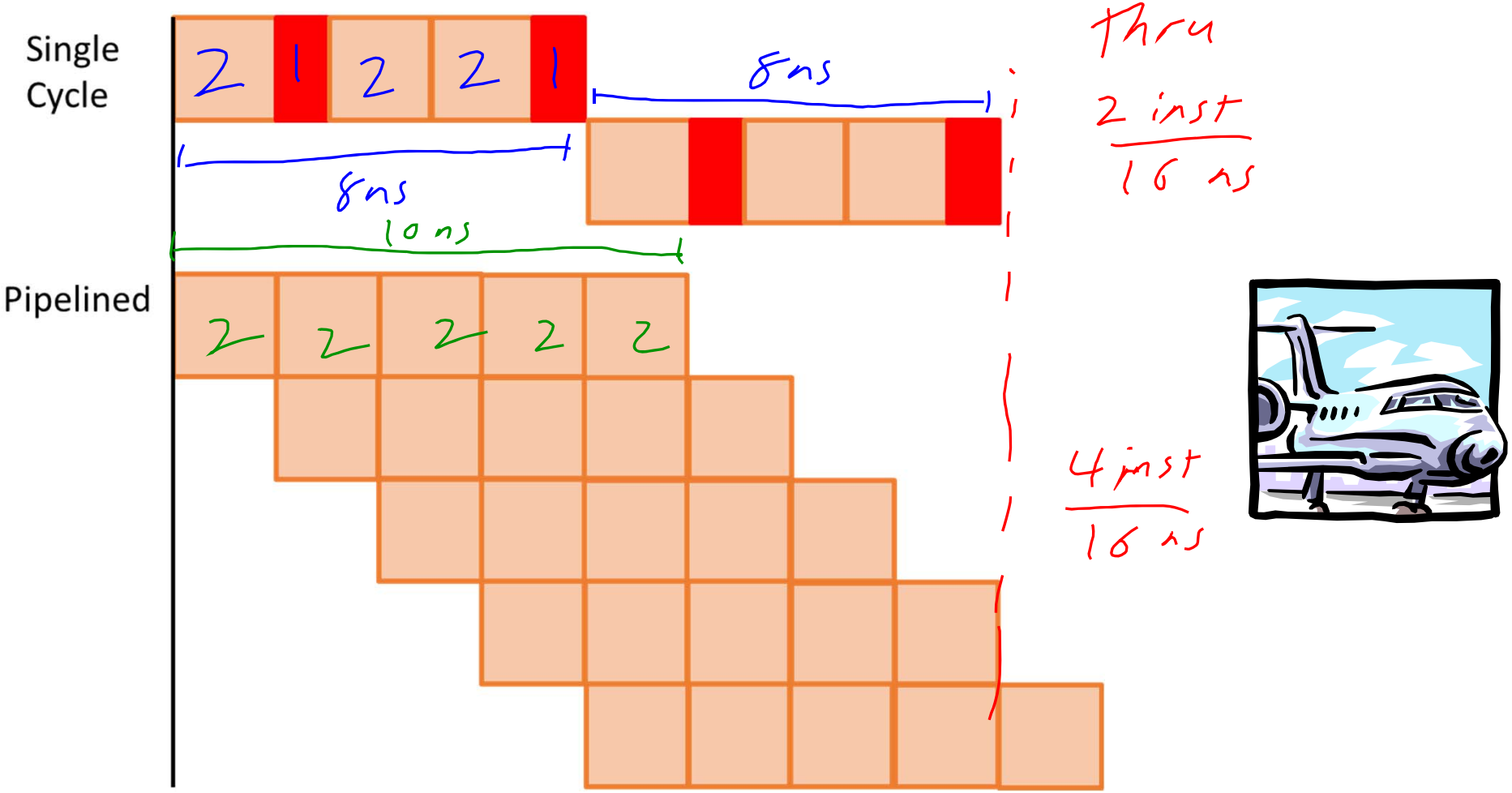
- How long for N instructions on single-cycle (8ns clock period)?

$N \cdot 8ns = 8N$ ns

- How much faster is pipelining for N=1000 ?

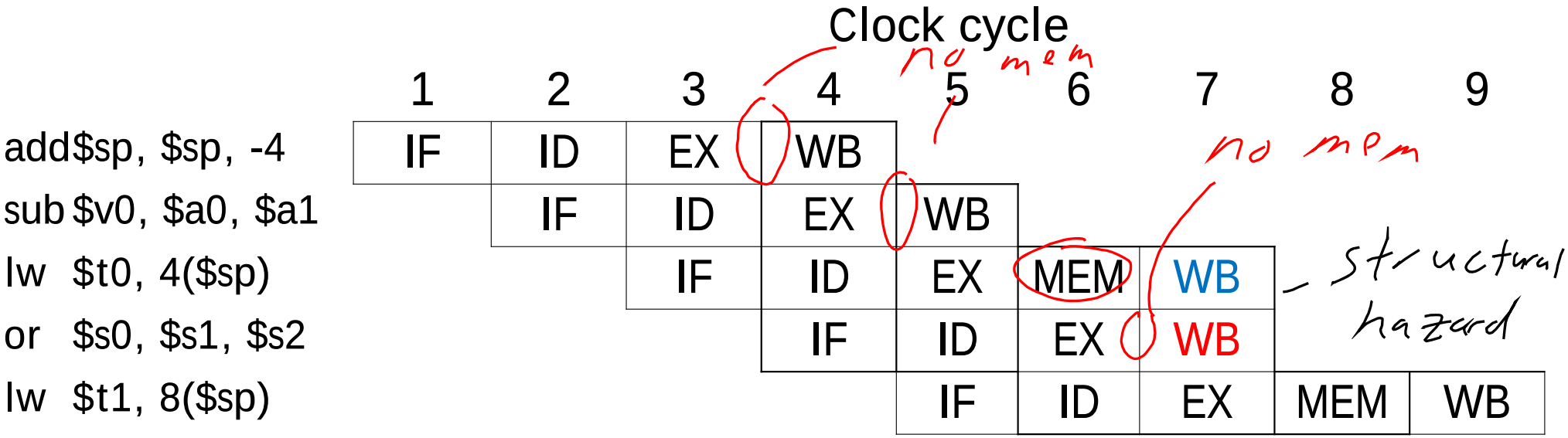
$$\frac{8+2n}{8N} \approx \frac{2000}{8000} \approx \frac{1}{4}$$

Pipelining improves throughput at the cost of increased latency



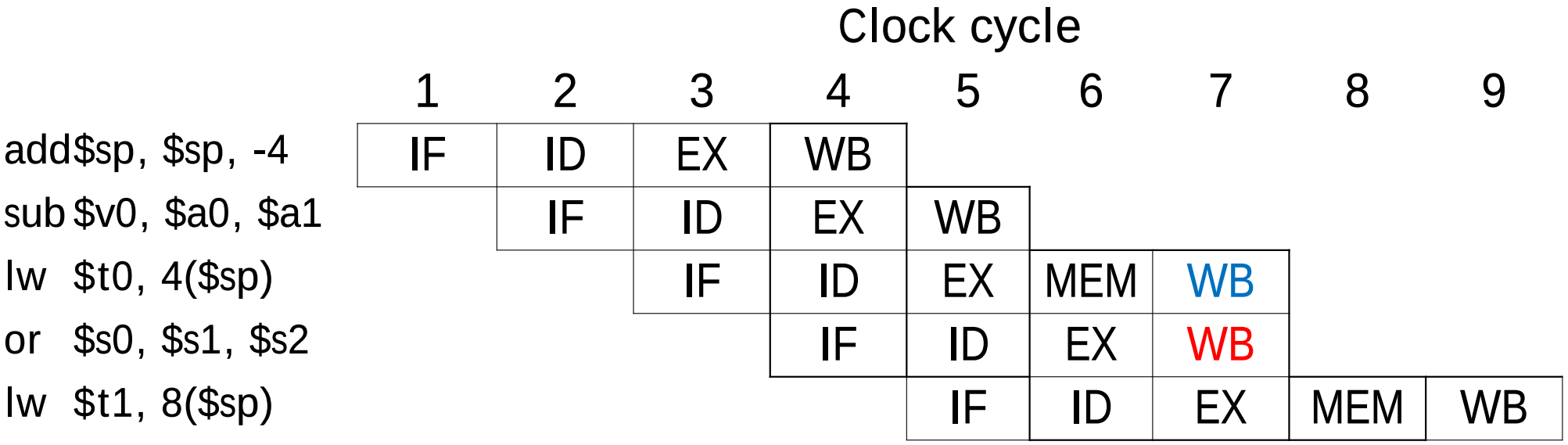
Some instructions do not require all five stages, can we skip stages?

- Example: 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?



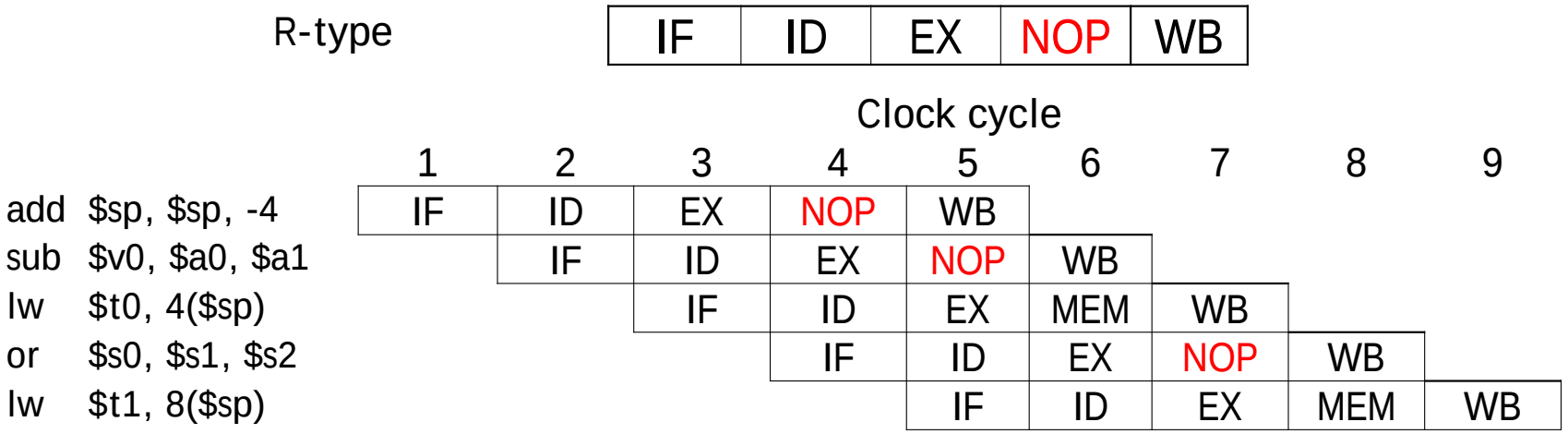
Trying to use the single stage for multiple instructions creates a structural hazard

- 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



Insert NOP stages to avoid structural hazards

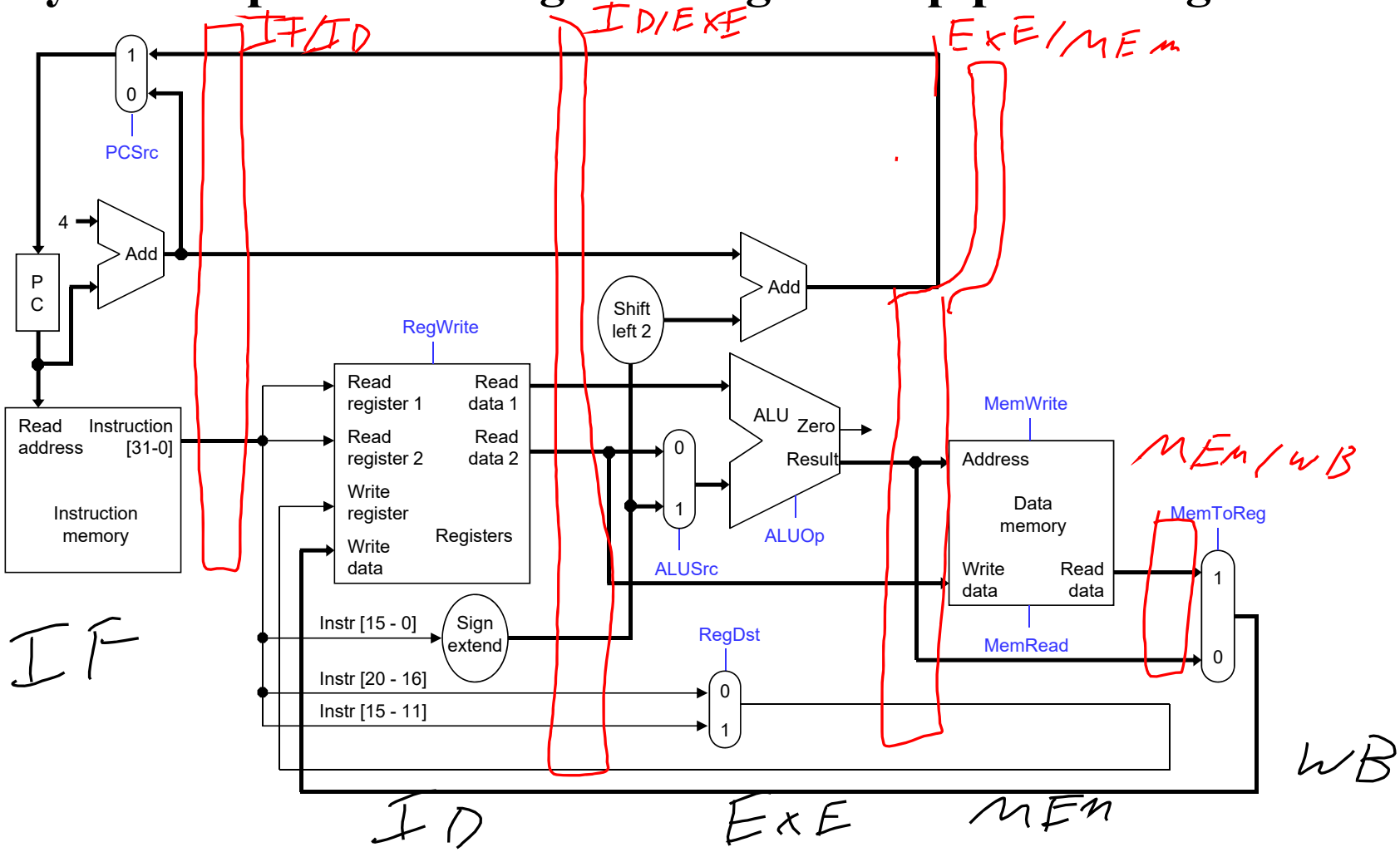
- All instructions take 5 cycles with the same stages in the same order
  - Some stages will do nothing for some instructions



  - Stores and Branches have NOP stages, too…

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

Single-cycle datapath rearranged to align with pipeline stages



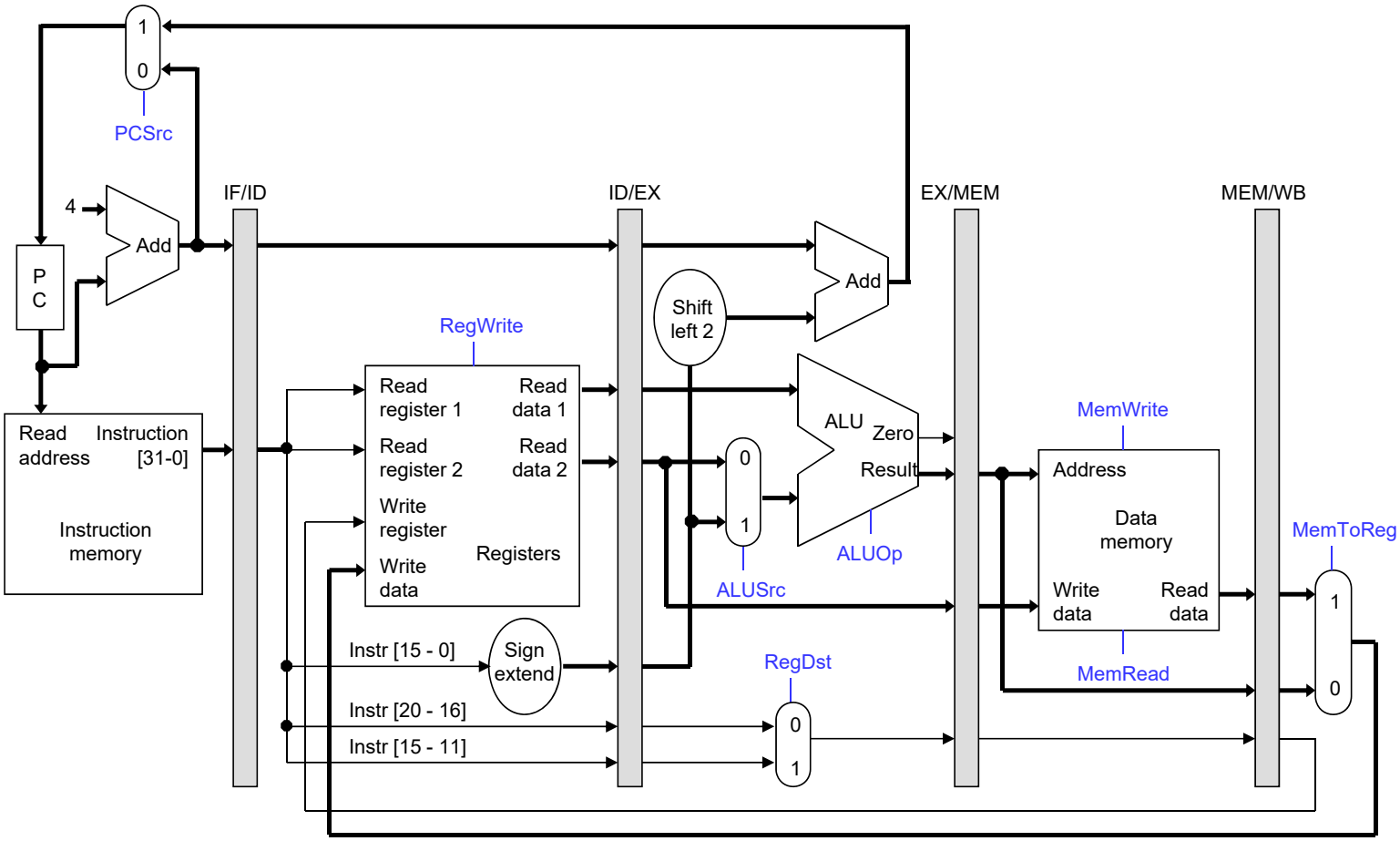
Add pipeline registers in between stages

- 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

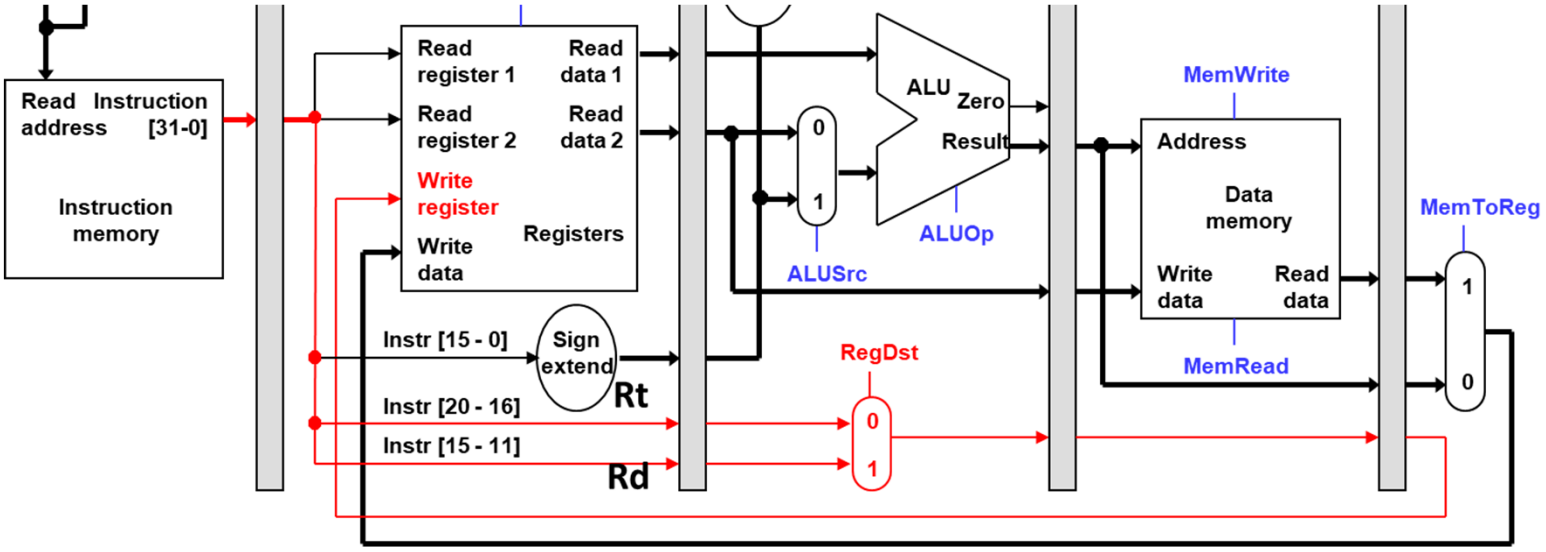
- No register is needed after the WB stage, because after WB the instruction is done.

Paths from register-read to register-write are now shorter

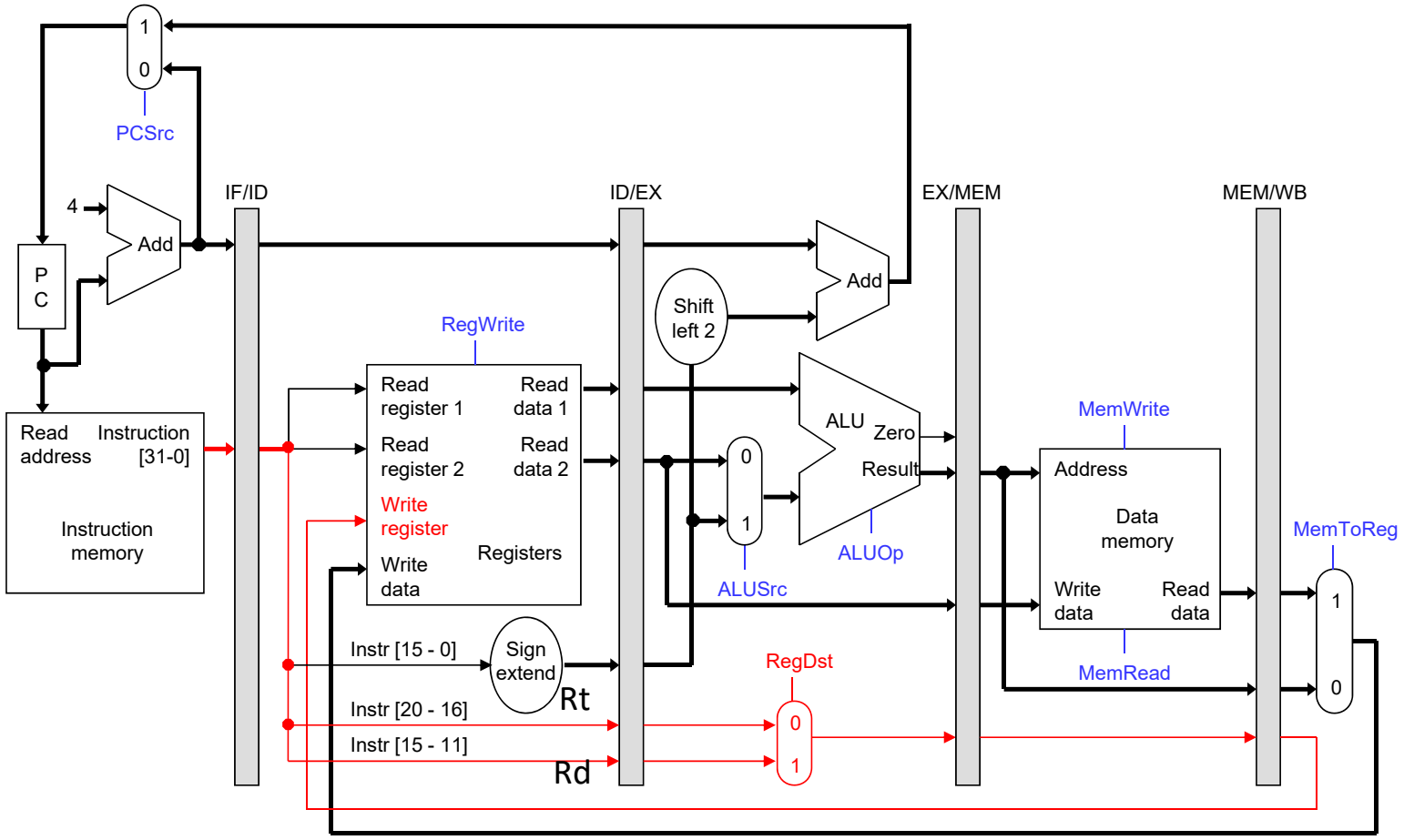


Data values required in later stages must be propagated forward through the pipeline registers.

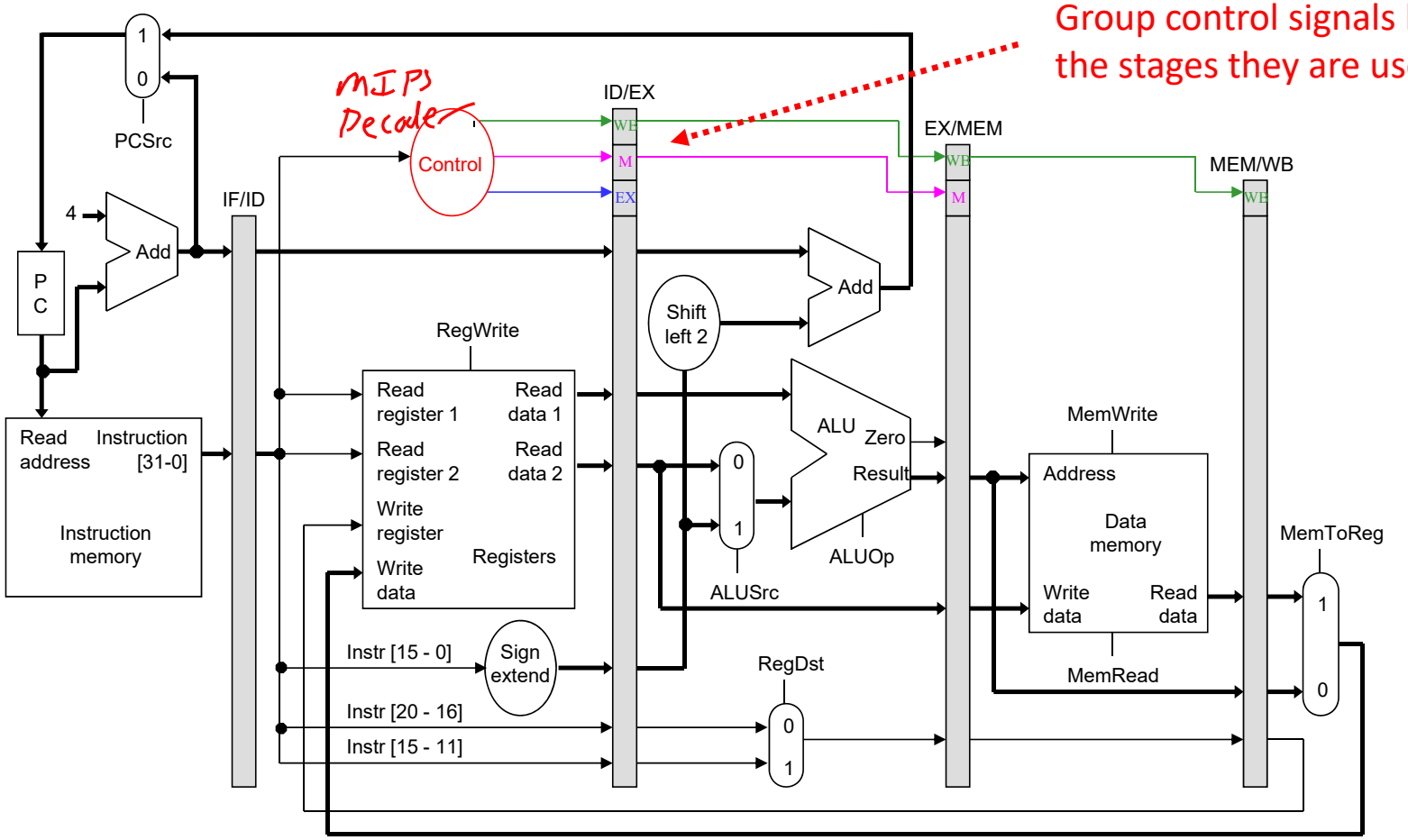
- Example
  - Destination register (`rd`) is determined during the *first* stage (IF)
  - We store into the destination register during the *fifth* stage (WB)
  - `rd` must be passed through all of the pipeline stages



Note – We cannot keep values like destination register in the "instruction register"



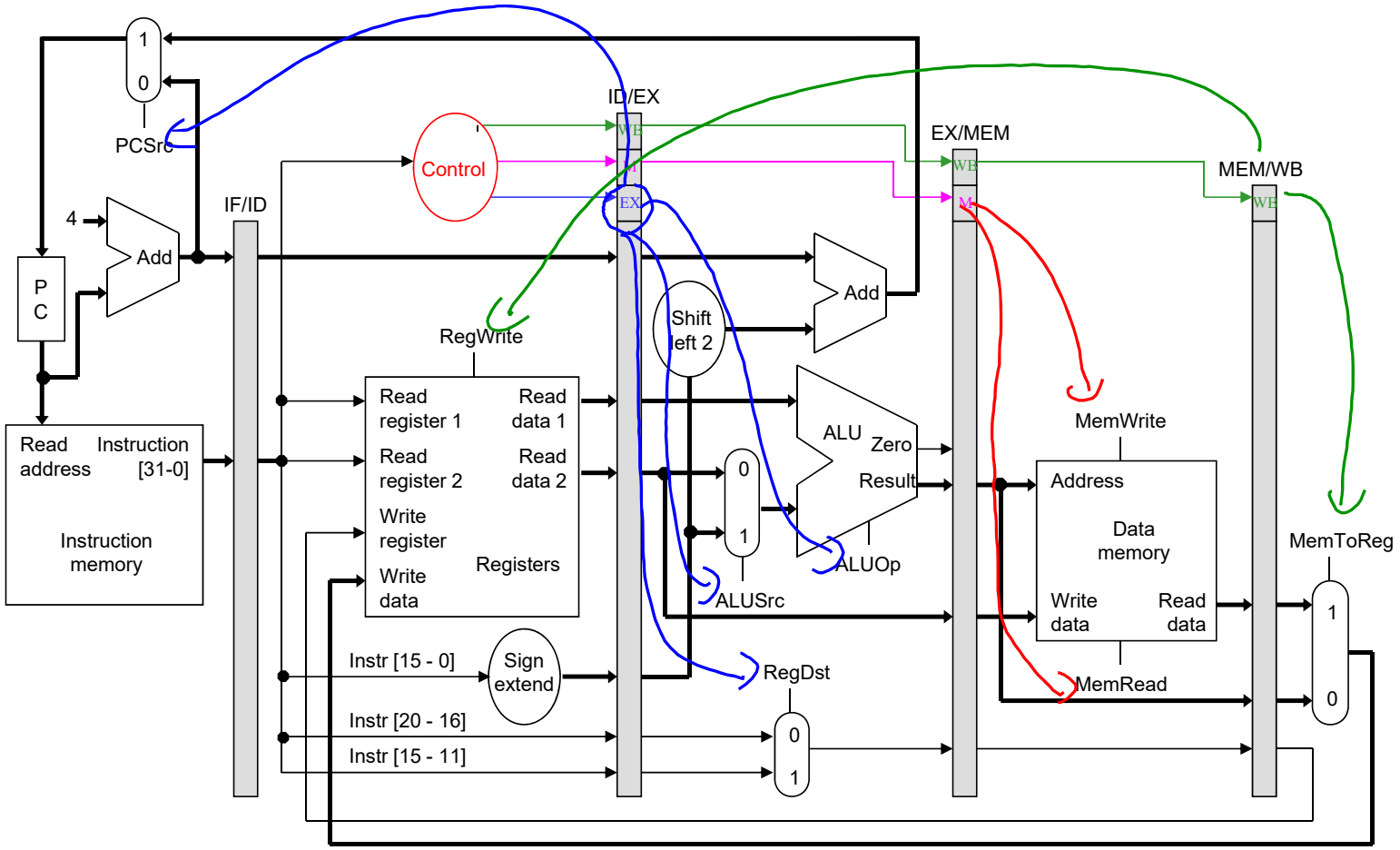
Control signals are generated in the decode stage and are propagated across stages



Categorize control signals by the pipeline stage that uses them

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

The pipeline registers and program counter update every clock cycle, so they do not have write enable controls



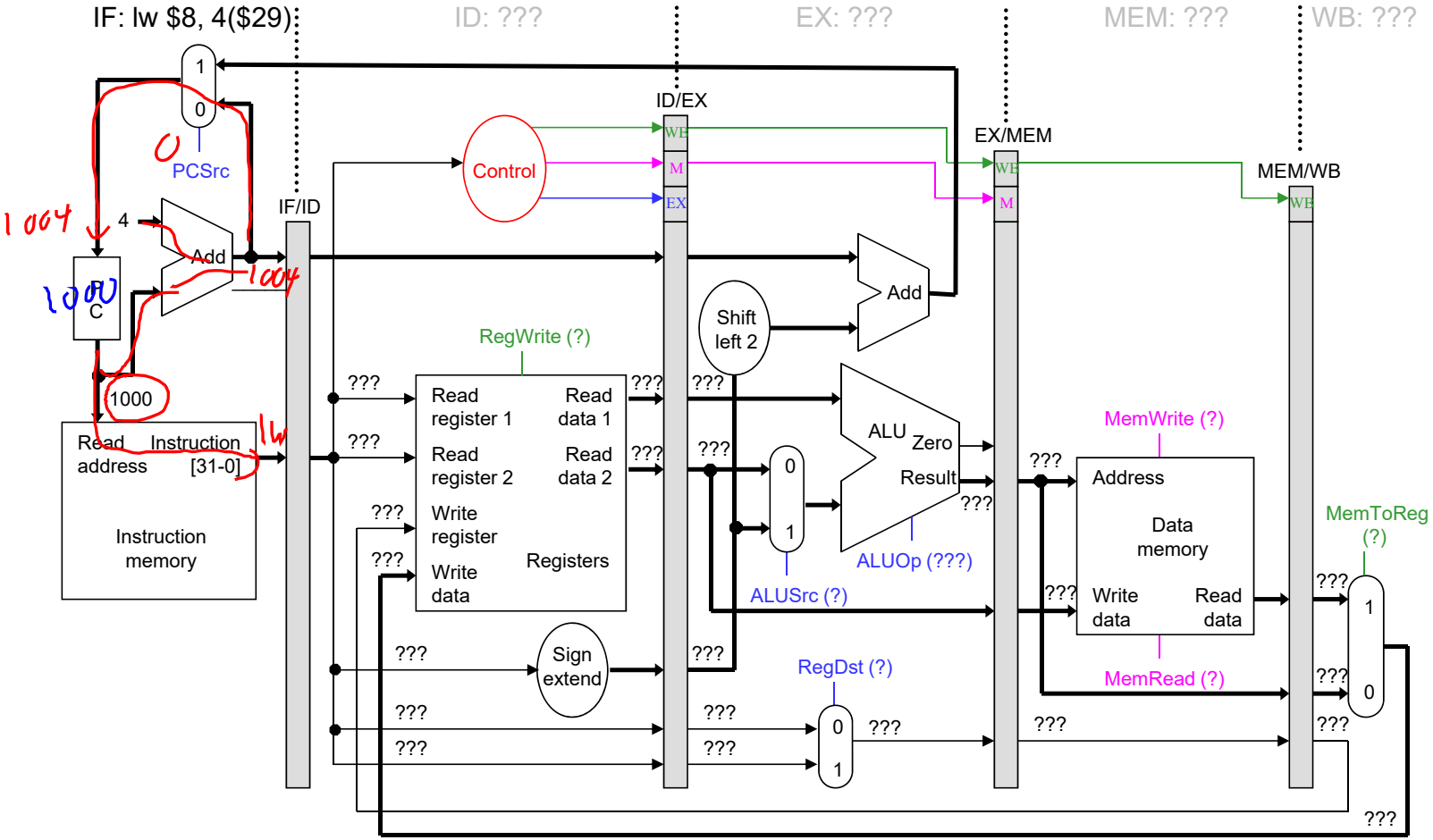
An example execution sequence

addresses in decimal

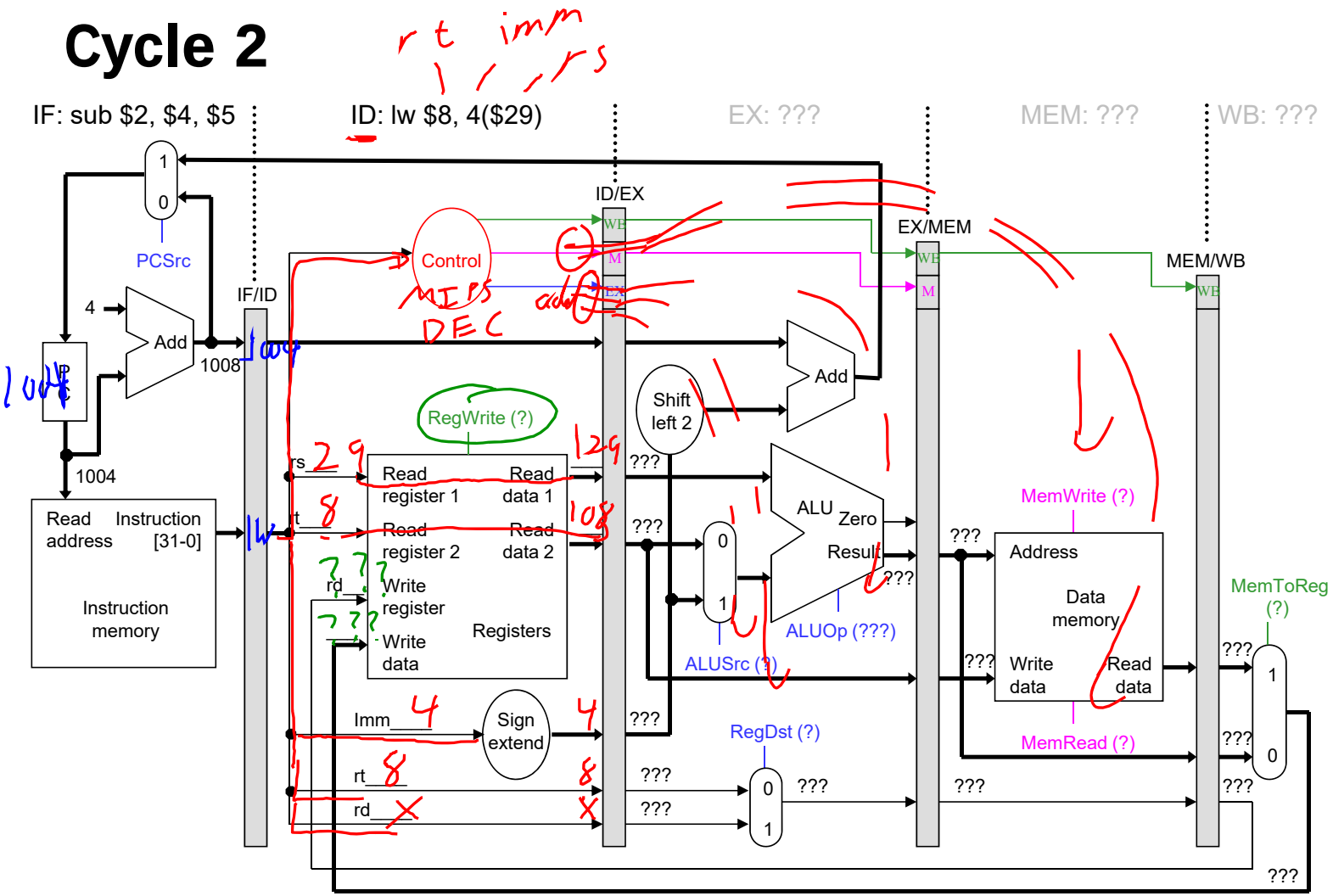
```
1000:   lw      $8, 4($29)
1004:   sub     $2, $4, $5
1008:   and     $9, $10, $11
1012:   or      $16, $17, $18
1016:   add     $13, $14, $0
```

- **ASSUMPTIONS**
  - Each register contains its number plus 100. Example: R[8] == 108, R[29] == 129
  - Every data memory location contains 99. Example: M[8] == 99, M[29] == 99
- **CONVENTIONS**
  - X indicates values that are not important, Example: Imm16 for R-type.
  - Question marks ??? indicate values we do not know, usually resulting from instructions coming before and after the ones in our example.

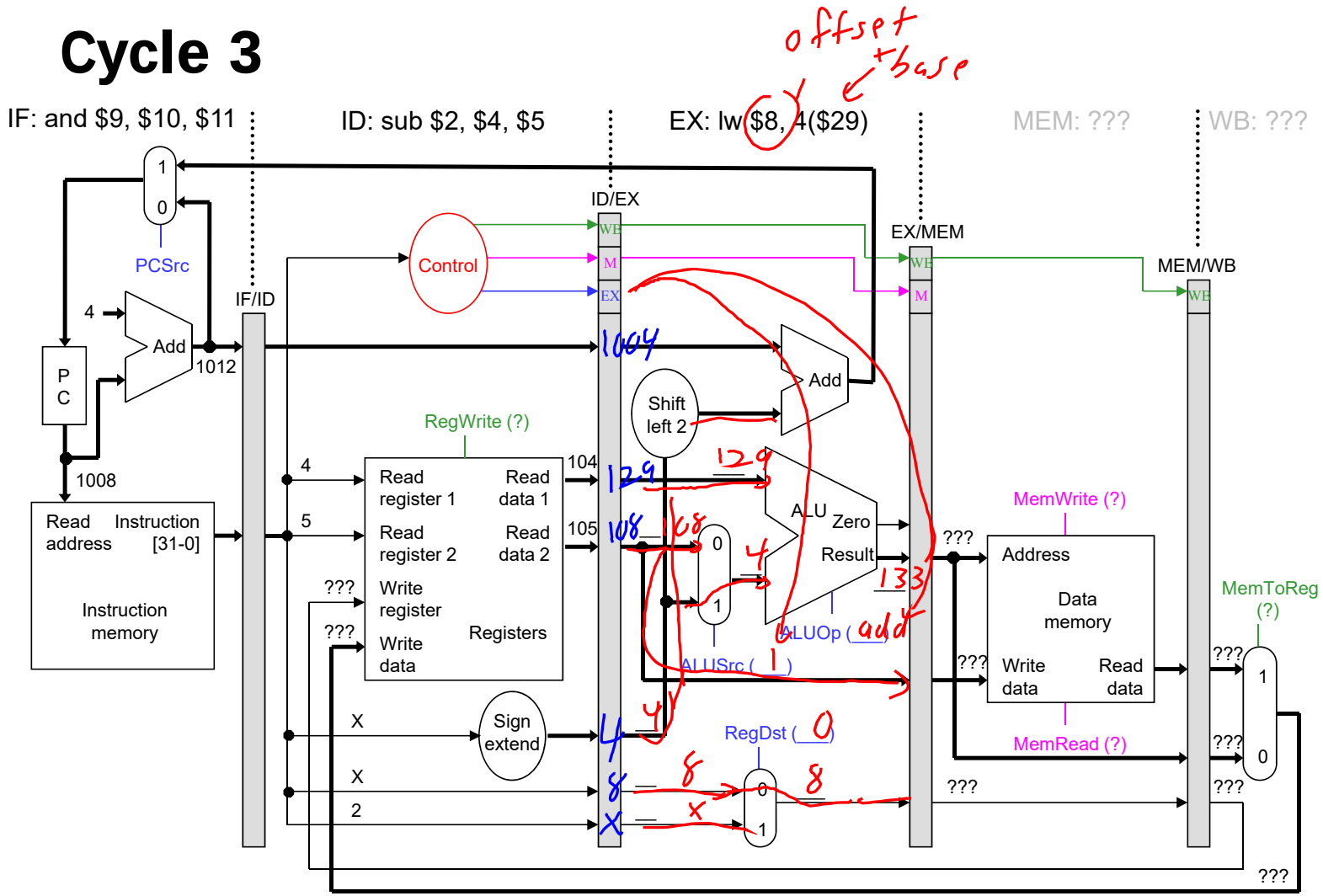
Cycle 1 (filling)



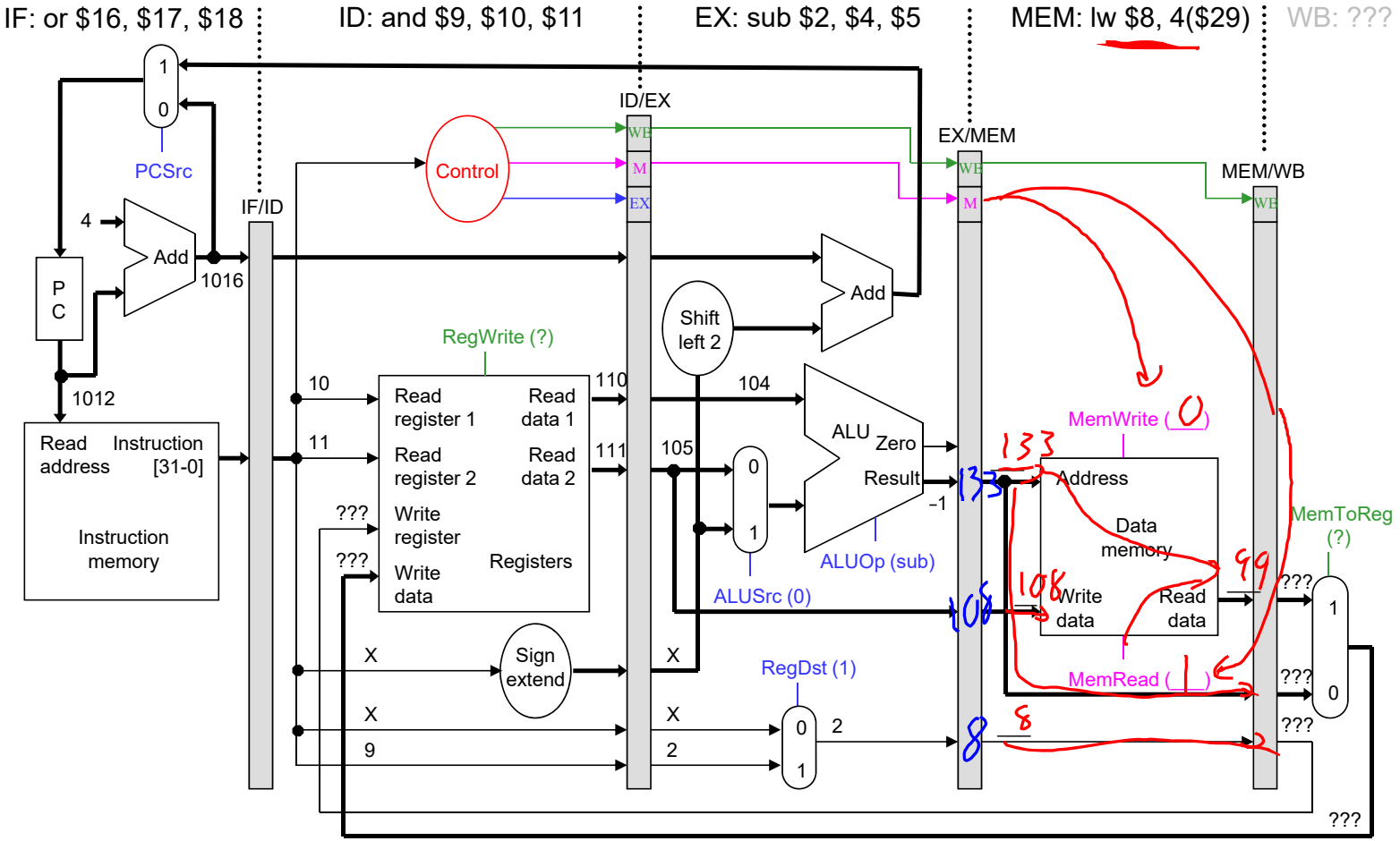
Cycle 2


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

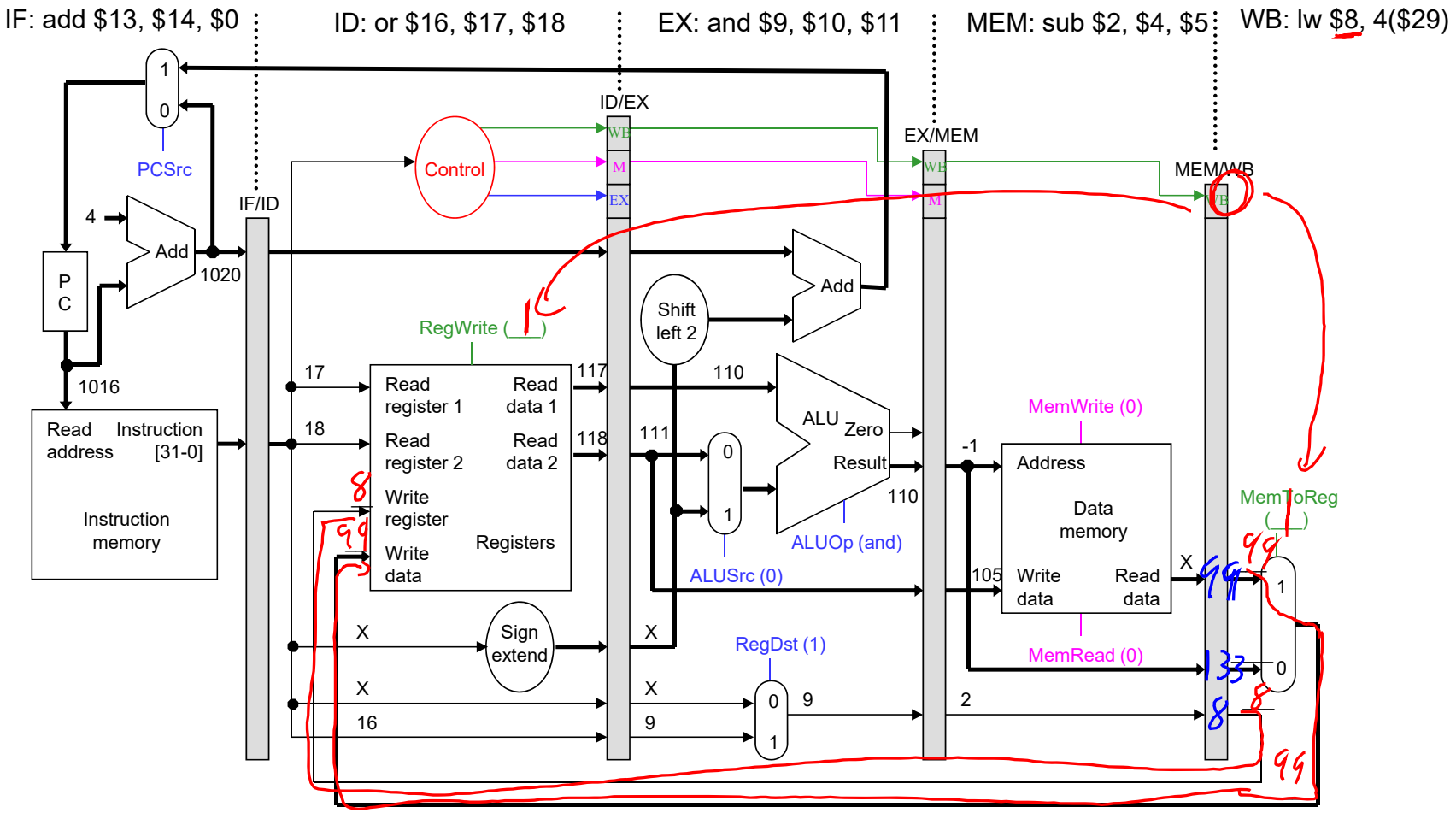
Cycle 3


IF: and $9, $10, $11
ID: sub $2, $4, $5
EX: lw $8, 4($29)
MEM: ???
WB: ???
offset
+base
PCSrc
Add
4
1012
1008
P C
Read address
Instruction [31-0]
Instruction memory
IF/ID
Control
ID/EX
EX/MEM
MEM/WB
WB
M
EX
RegWrite (?)
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
104
105
Shift left 2
Add
ALU
Zero
Result
ALUOp (add)
ALUSrc (1)
RegDst (0)
Sign extend
X
2
MemWrite (?)
Address
Data memory
Write data
Read data
MemRead (?)
MemToReg (?)
???
1004
129
108
133
4
8

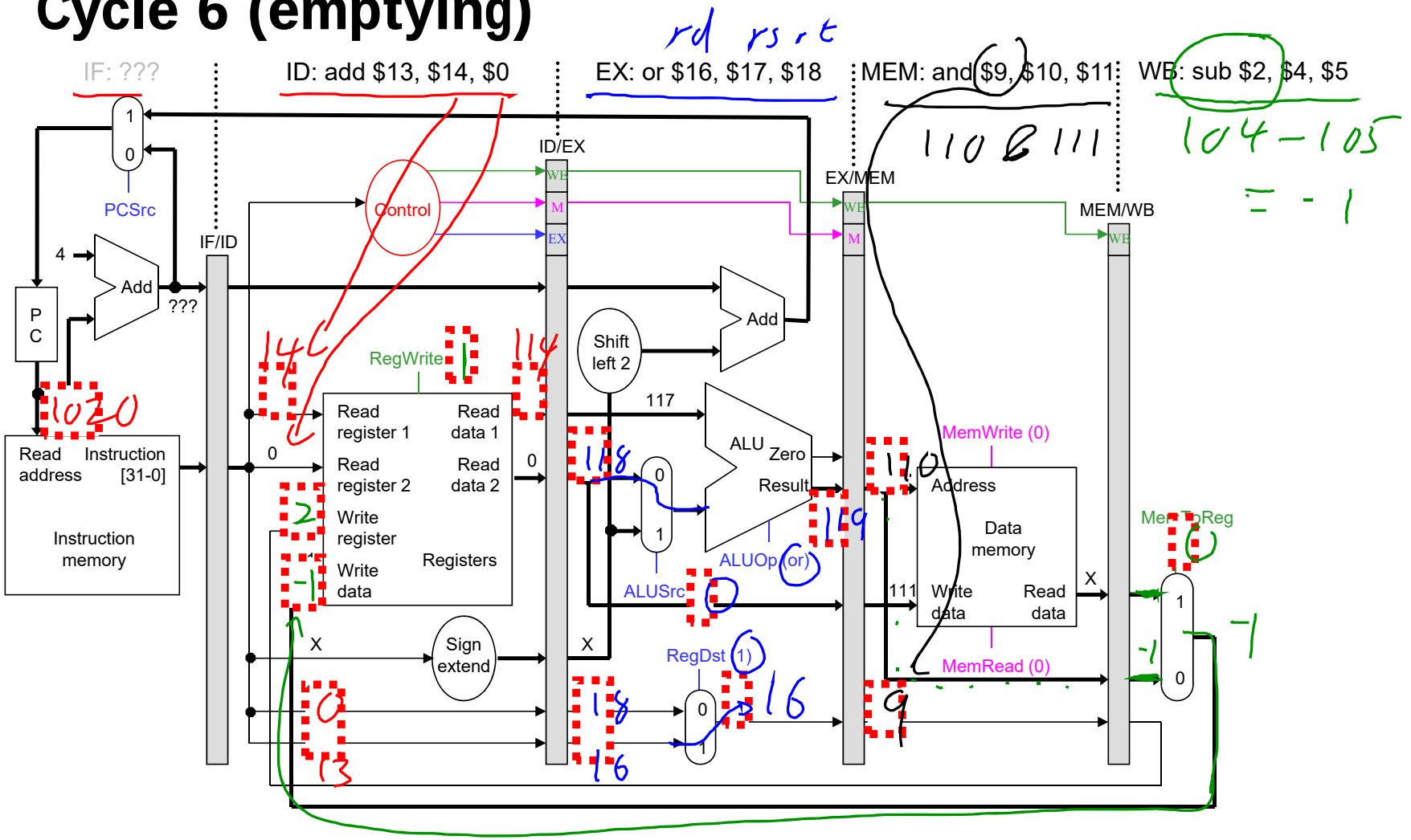
Cycle 4


IF: or $16, $17, $18
ID: and $9, $10, $11
EX: sub $2, $4, $5
MEM: lw $8, 4($29)
WB: ???
PCSrc
Add
4
1016
P C
1012
Read address
Instruction [31-0]
Instruction memory
IF/ID
Control
ID/EX
EX/MEM
MEM/WB
WB
M
EX
Shift left 2
Add
RegWrite (?)
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
10
11
???
???
110
111
104
105
ALU
Zero
Result
-1
ALUOp (sub)
ALUSrc (0)
Sign extend
X
X
X
9
2
2
RegDst (1)
MemWrite (0)
Address
Data memory
Write data
Read data
MemRead (1)
MemToReg (?)
133
133
108
108
99
8
8
???
???
???
???
???

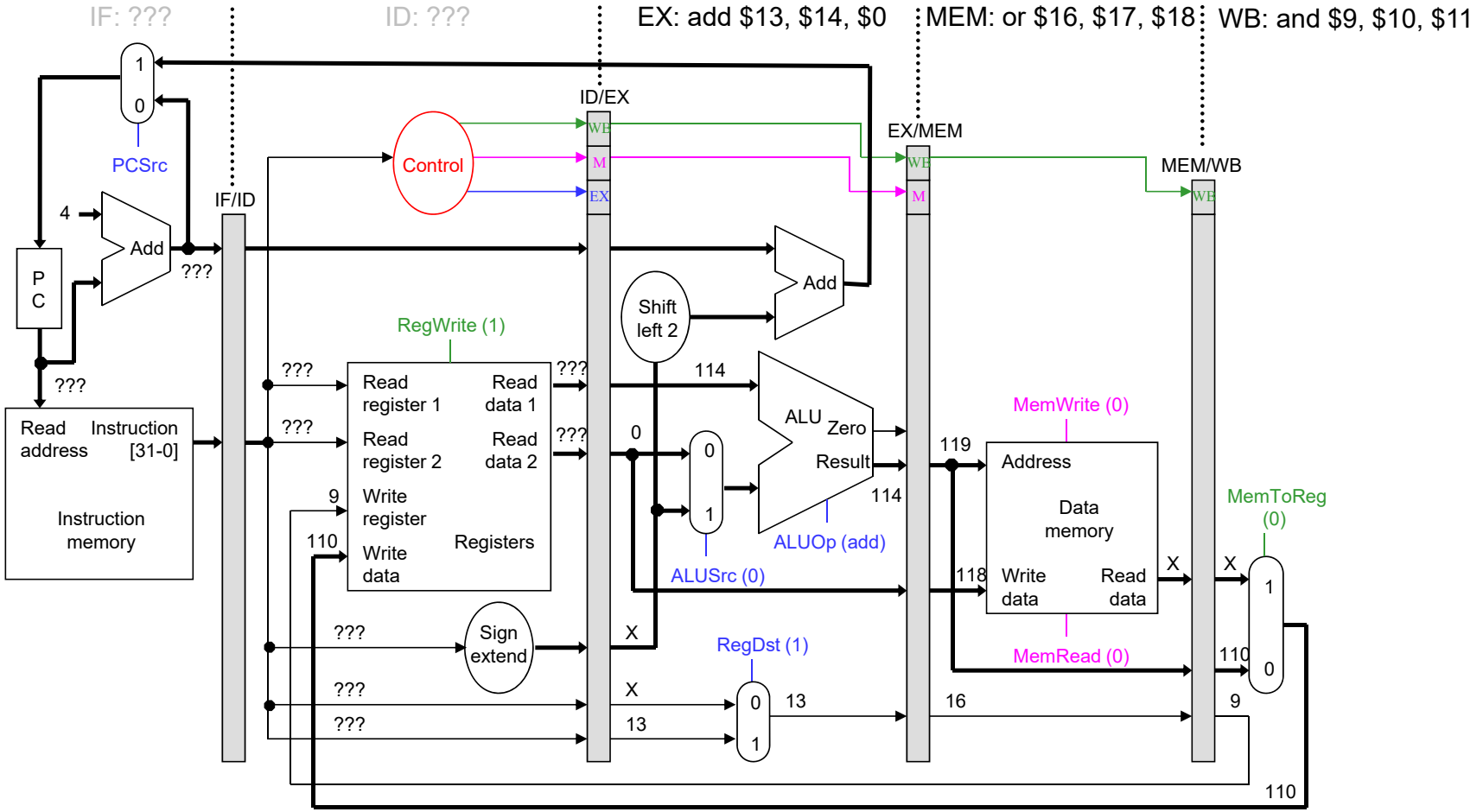
Cycle 5 (full)


IF: add $13, $14, $0
ID: or $16, $17, $18
EX: and $9, $10, $11
MEM: sub $2, $4, $5
WB: lw $8, 4($29)
IF/ID
ID/EX
EX/MEM
MEM/WB
PCSrc
Control
WB
M
EX
Add
4
1020
P C
1016
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
17
18
117
118
Shift left 2
110
111
ALU
Zero
Result
ALUOp (and)
ALUSrc (0)
MemWrite (0)
MemRead (0)
MemToReg
Address
Data memory
Write data
Read data
105
-1
X
Sign extend
16
9
RegDst (1)
2
133
99
8

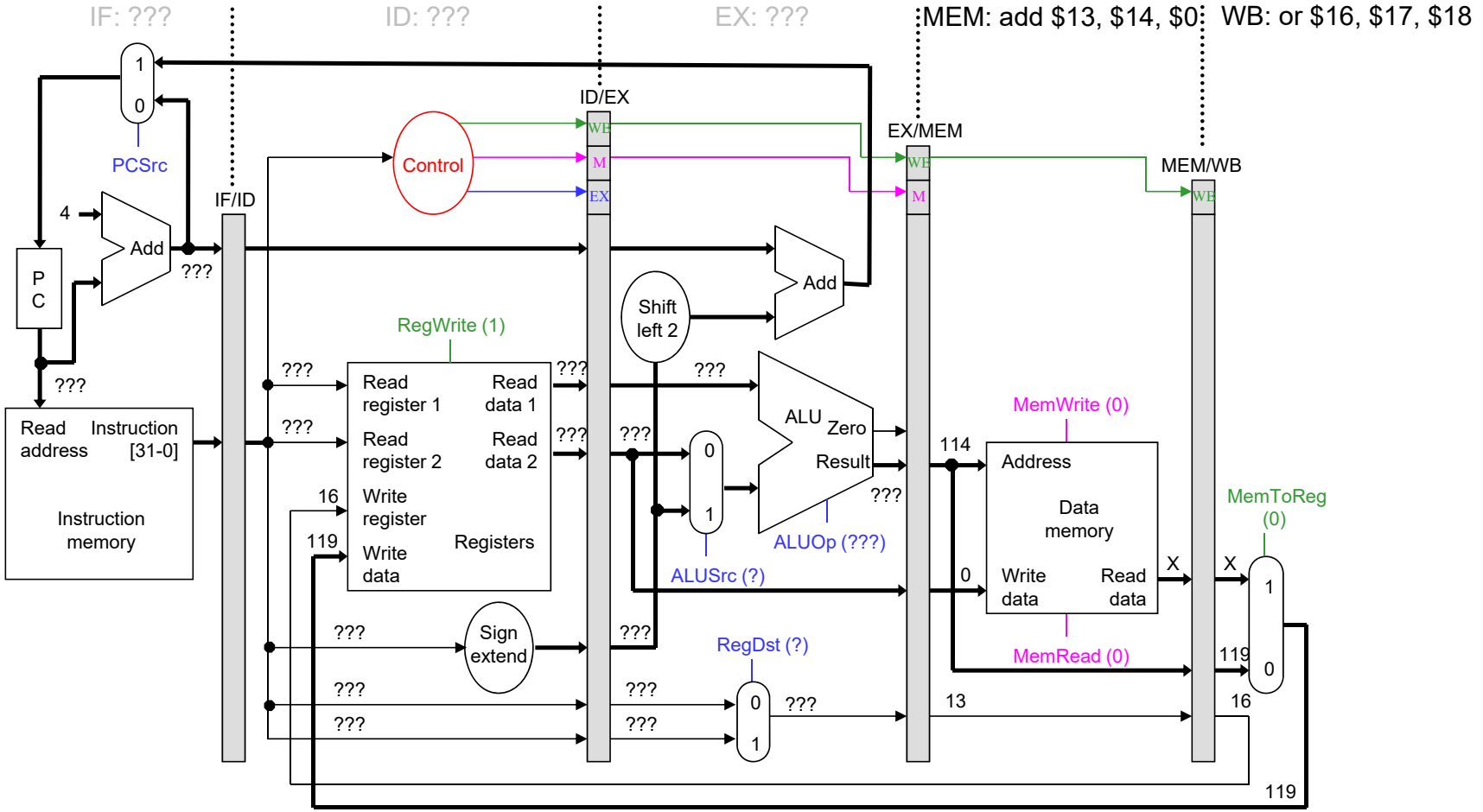
Cycle 6 (emptying)



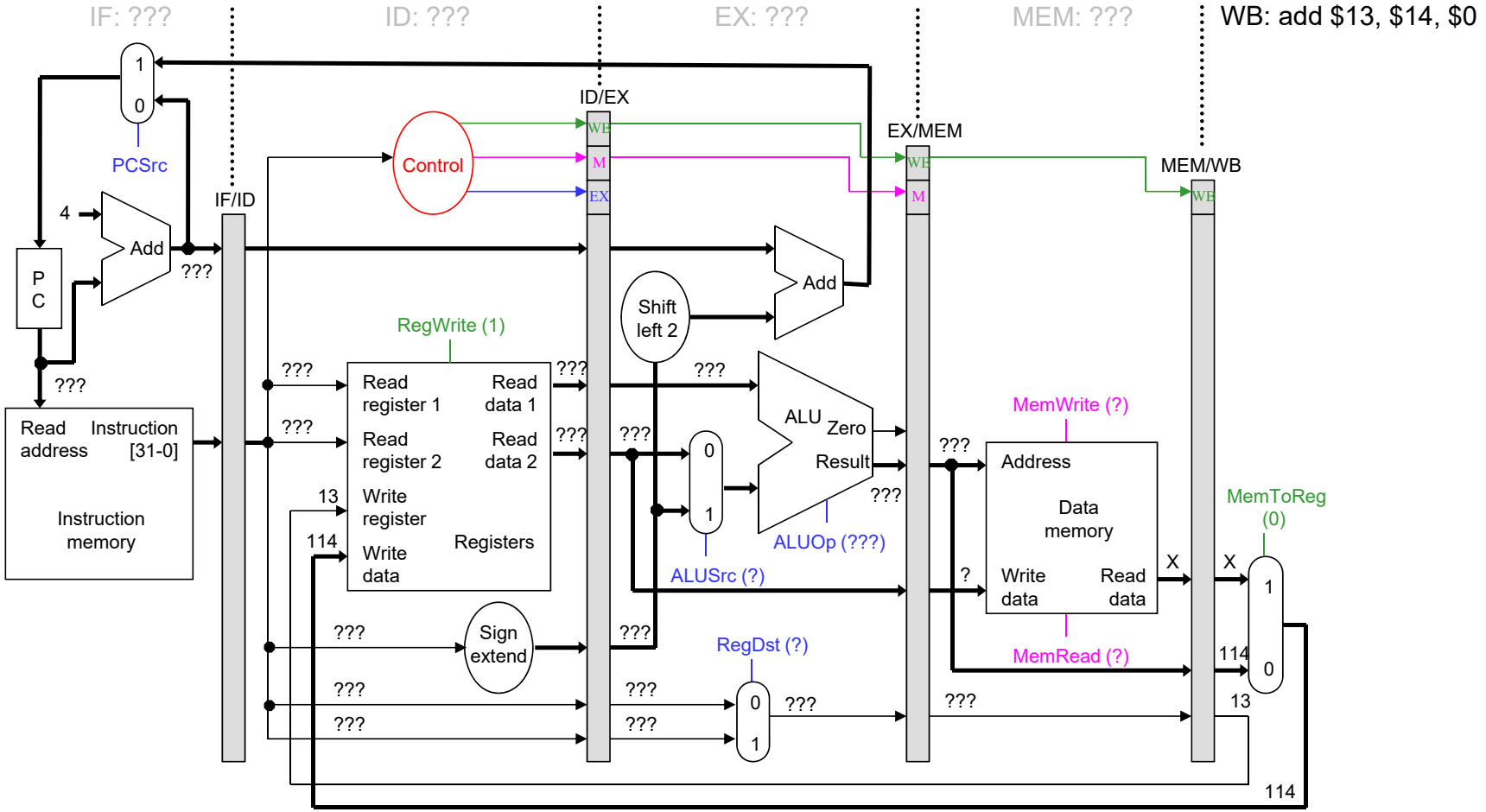
Cycle 7


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

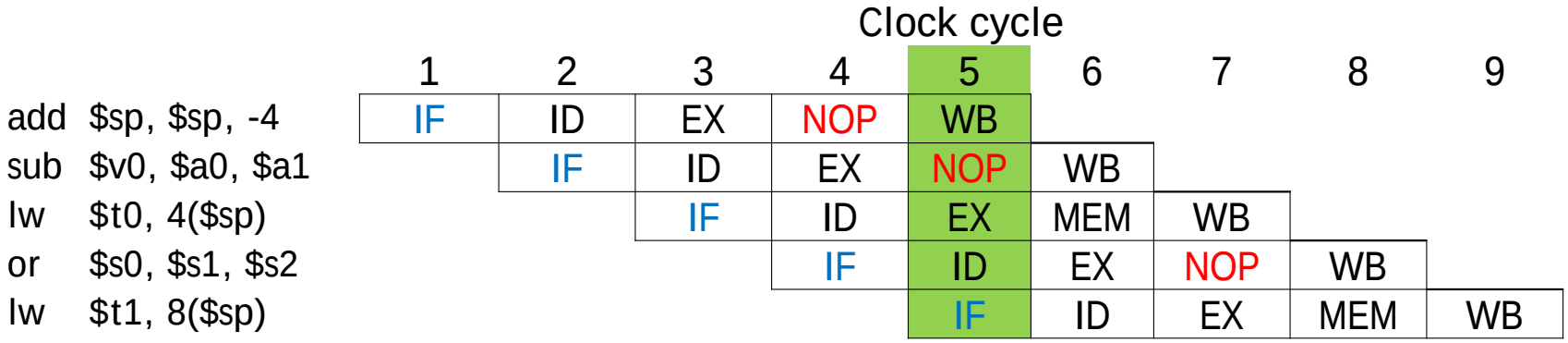
Cycle 8


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

Cycle 9


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

Things to notice from the last nine slides

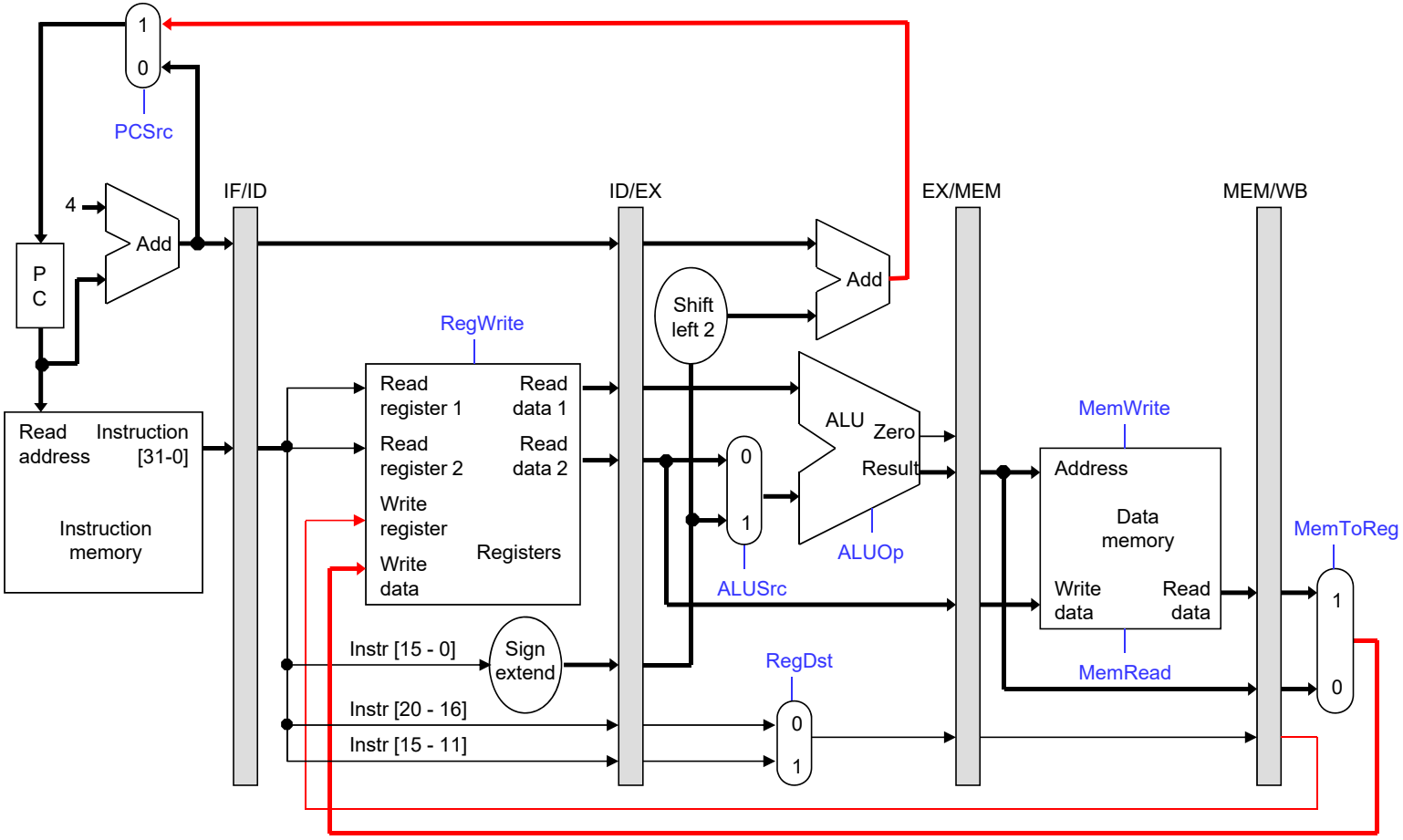


- Instruction executions overlap
- Each functional unit is used by a *different* instruction in each cycle.
- In clock cycle 5, all of the hardware units are used (the pipeline is full). This is the ideal situation, and what makes pipelined processors so fast
- Similar example in the book available at the end of Section 6.3.

MIPs ISA makes pipelining “easy”

- Instruction formats are the same length and uniform
- Addressing modes are simple
- Each instruction takes only one cycle

Note how everything goes left to right, except …



Next time: We will discuss Data Hazards

Cycle 6 (emptying)