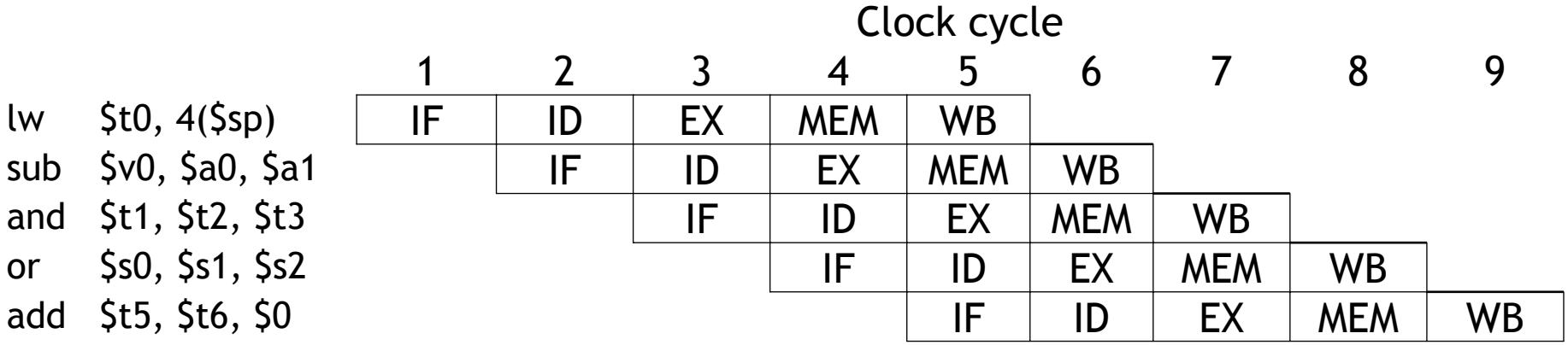
Pipelined datapath and control

- Last time we introduced the main ideas of pipelining.
- Today we'll see a basic implementation of a pipelined processor.
  - The datapath and control unit share similarities with both the single-cycle and multicycle implementations that we already saw.
  - An example execution highlights important pipelining concepts.
- After Spring Break we'll discuss several complications of pipelining that we're hiding from you for now.
- There are still discussion sections this week.

Pipelining concepts

- Single-cycle and multicycle processors allow only one instruction to be in the datapath during any given clock cycle. This results in functional units being idle for much of the time.
- A pipelined processor allows multiple instructions to execute at once, and each instruction uses a different functional unit in the datapath.
- This maximizes the hardware utilization, so programs can run faster.
  - Our example datapath has five stages and up to five instructions can run concurrently, so the ideal speedup is five.
  - One instruction can finish executing on every clock cycle, and simpler stages also lead to shorter cycle times.

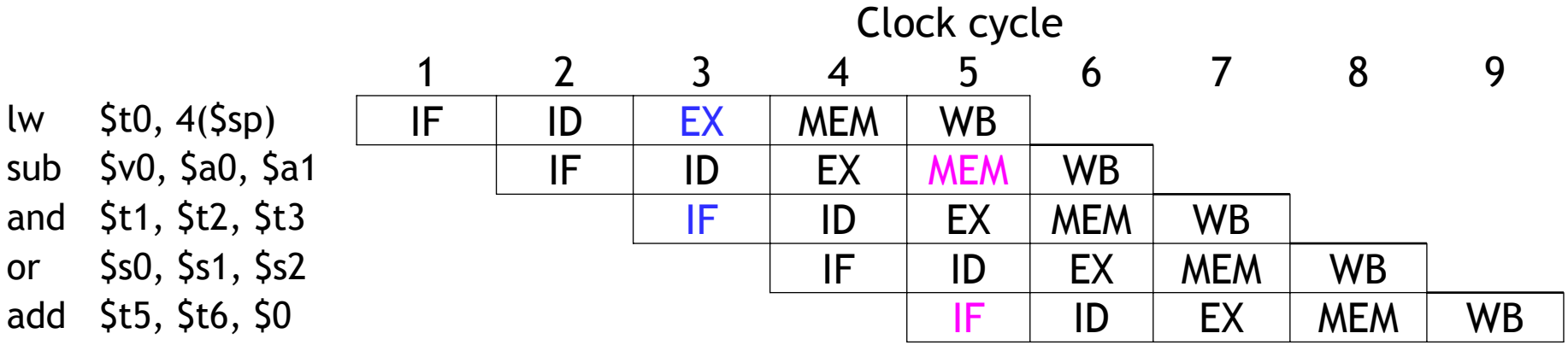


Hardware for single and multicycle CPUs

- In the single-cycle datapath, all instructions execute in one clock cycle.
  - lw and sw need to access memory twice (instruction fetch and data read or write), so two separate memories are needed.
  - All instructions increment the PC and perform some ALU calculation, and beq does another addition to compute a target address. So up to three adders/ALUs are needed.
- A multicycle datapath divides execution into several cycles, or stages.
  - Only one memory is needed. lw and sw access the memory twice, but on different clock cycles, so there is no conflict.
  - Similarly, one ALU is enough. The PC increment, ALU operation, and target address computation are all done on different cycles.
- How much hardware do we need for a pipelined datapath?

Hardware for pipelining

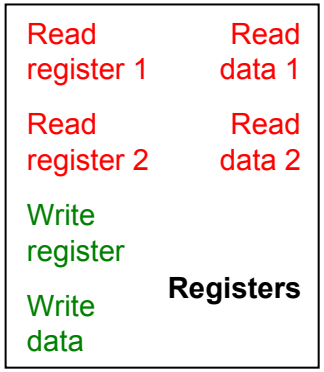
- The whole point of pipelining is to allow multiple instructions to execute at the same time.
- We may need to perform several operations in the same cycle.
  - Increment the PC and add registers at the same time.
  - Fetch one instruction while another one reads or writes data.



- Thus, like the single-cycle datapath, a pipelined processor will need to duplicate hardware elements that are needed several times in the same clock cycle.

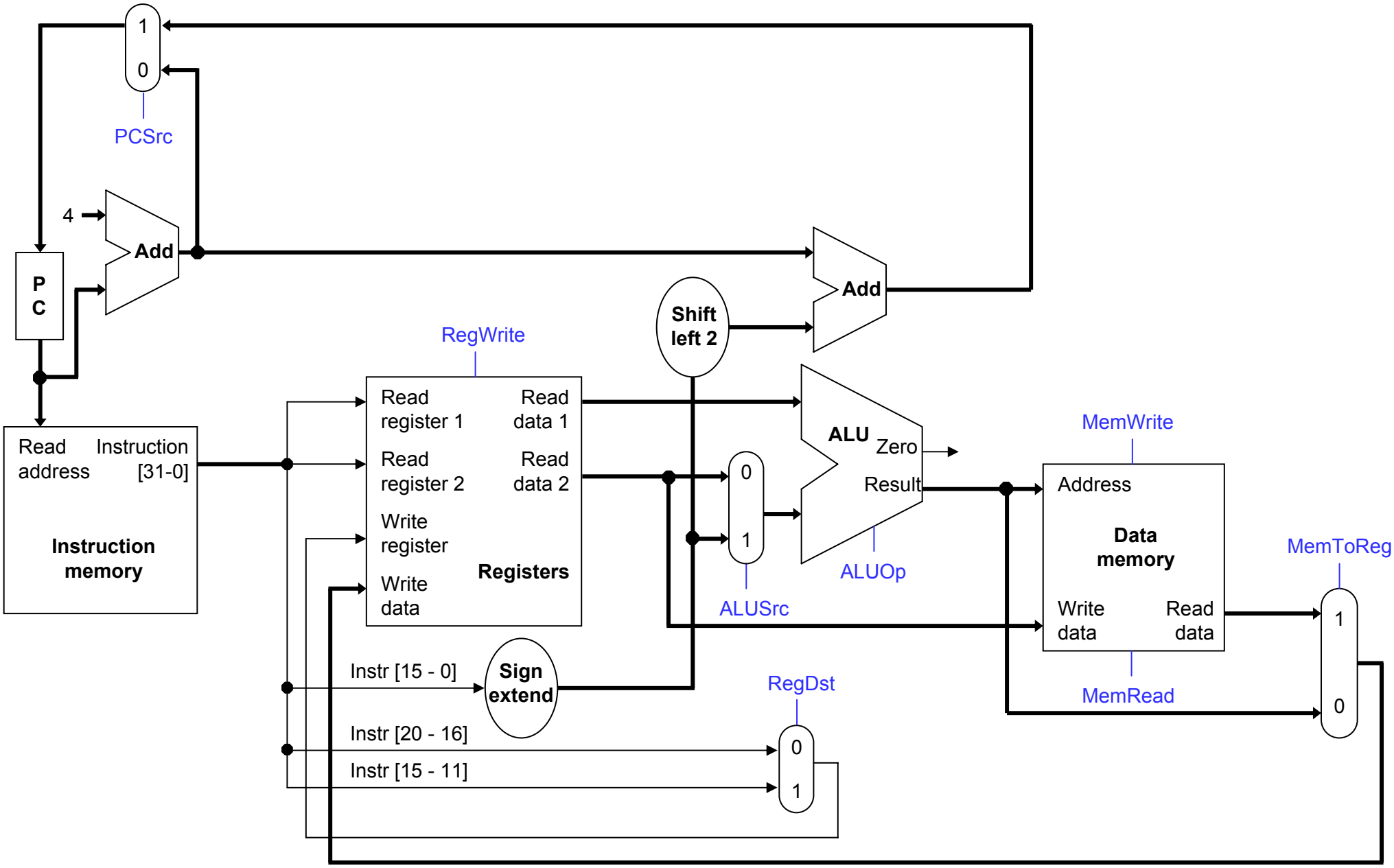
One register file is enough

- We need only one register file to support both the ID and WB stages.



- You can read and write to the register file in the same cycle.
  - Registers are made of edge-triggered flip-flops. Their current values can be read any time, but writes occur only on positive clock edges.
  - Writes complete quickly, so the new register values will be available by the end of the clock cycle.
- We already took advantage of these properties in our single-cycle CPU.

Single-cycle datapath, slightly rearranged



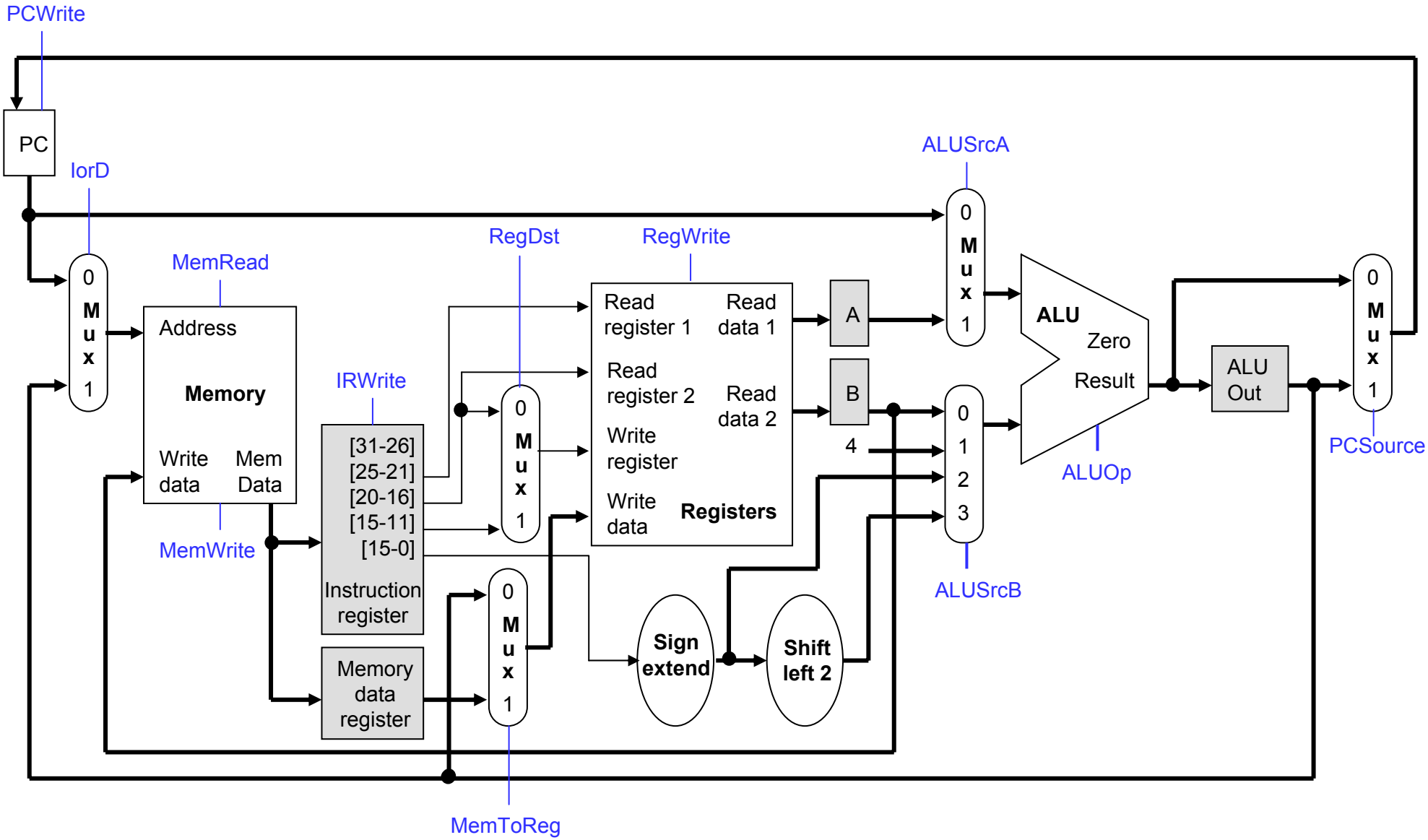
What's been changed?

- Almost nothing! This is equivalent to the original single-cycle datapath.
  - There are separate memories for instructions and data.
  - There are two adders for PC-based computations, and one ALU.
  - The control signals are the same.
- Only some cosmetic changes were made to make the diagram smaller.
  - A few labels are missing, and the muxes are smaller.
  - The data memory has only one Address input. The actual memory operation can be determined from the MemRead and MemWrite control signals.
- The datapath components have also been moved around, to make room for what's coming next.

Multiple cycles

- In pipelining, we also divide instruction execution into multiple cycles.
- Information computed during one cycle may be needed in a later cycle.
  - The instruction read in the IF stage determines which registers are fetched in the ID stage, what constant is used for the EX stage, and what the destination register is for WB.
  - The registers read in ID are used in the EX and/or MEM stages.
  - The ALU output produced in the EX stage is an effective address for the MEM stage or a result for the WB stage.
- We added several intermediate registers to the multicycle datapath to preserve information between stages, as highlighted on the next slide.

Intermediate registers in a multicycle datapath



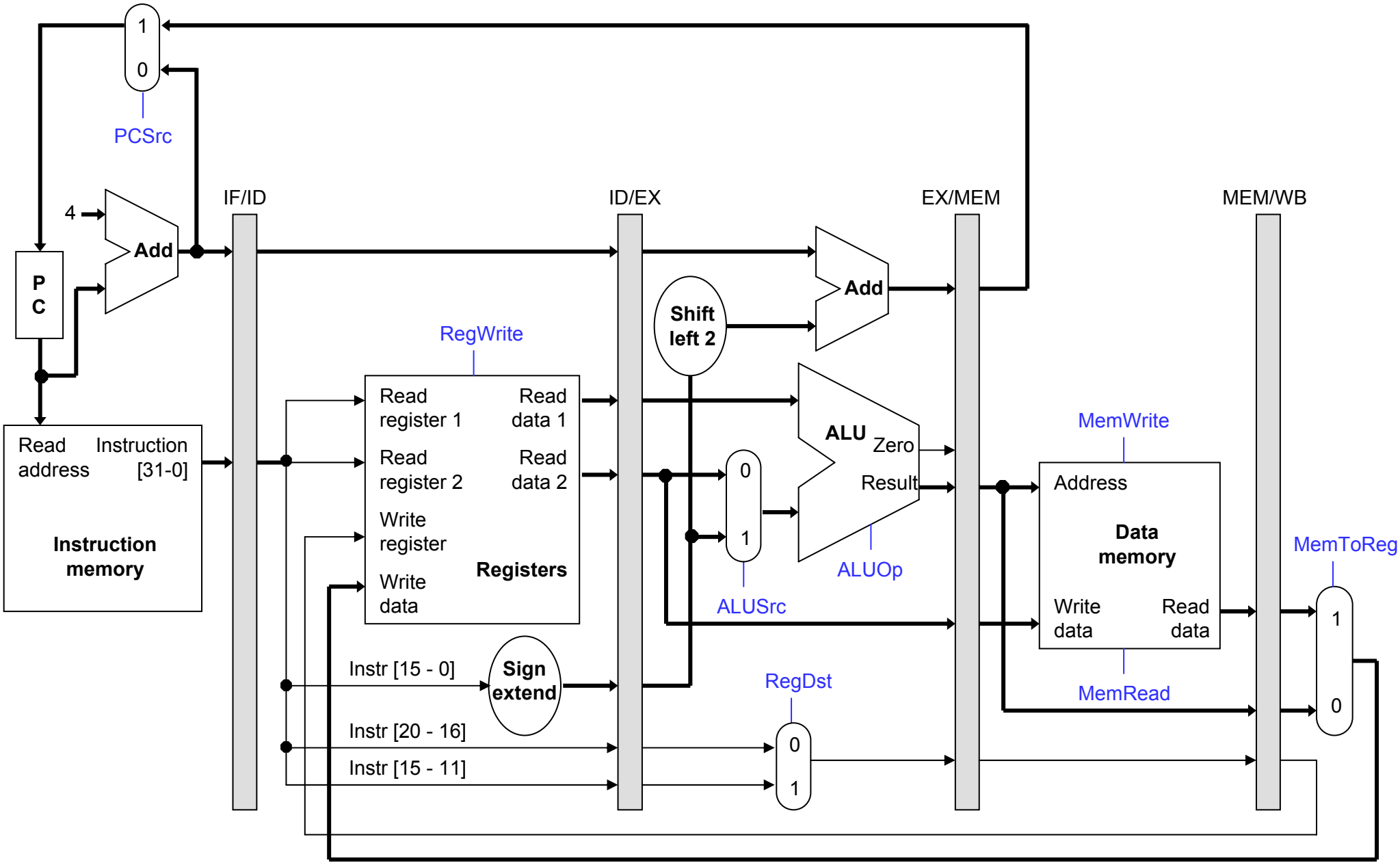
Pipeline registers

- We'll add intermediate registers to our pipelined datapath too.
- 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 the register file itself will act as the "pipeline register" between instructions.

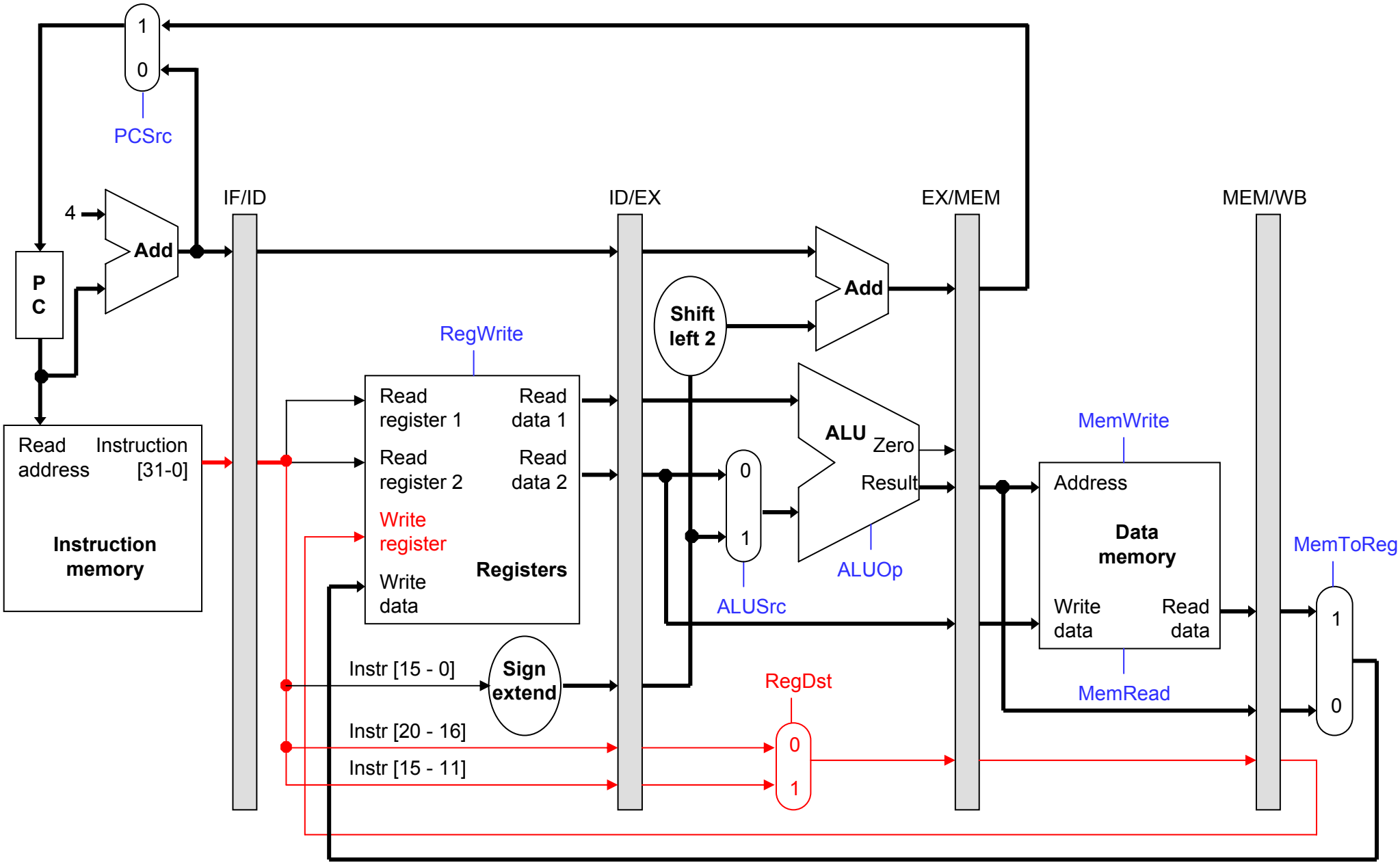
Pipelined datapath



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 an "instruction register" like we did before in the multicycle datapath, because the pipelined machine needs to fetch a new instruction every clock cycle.

The destination register

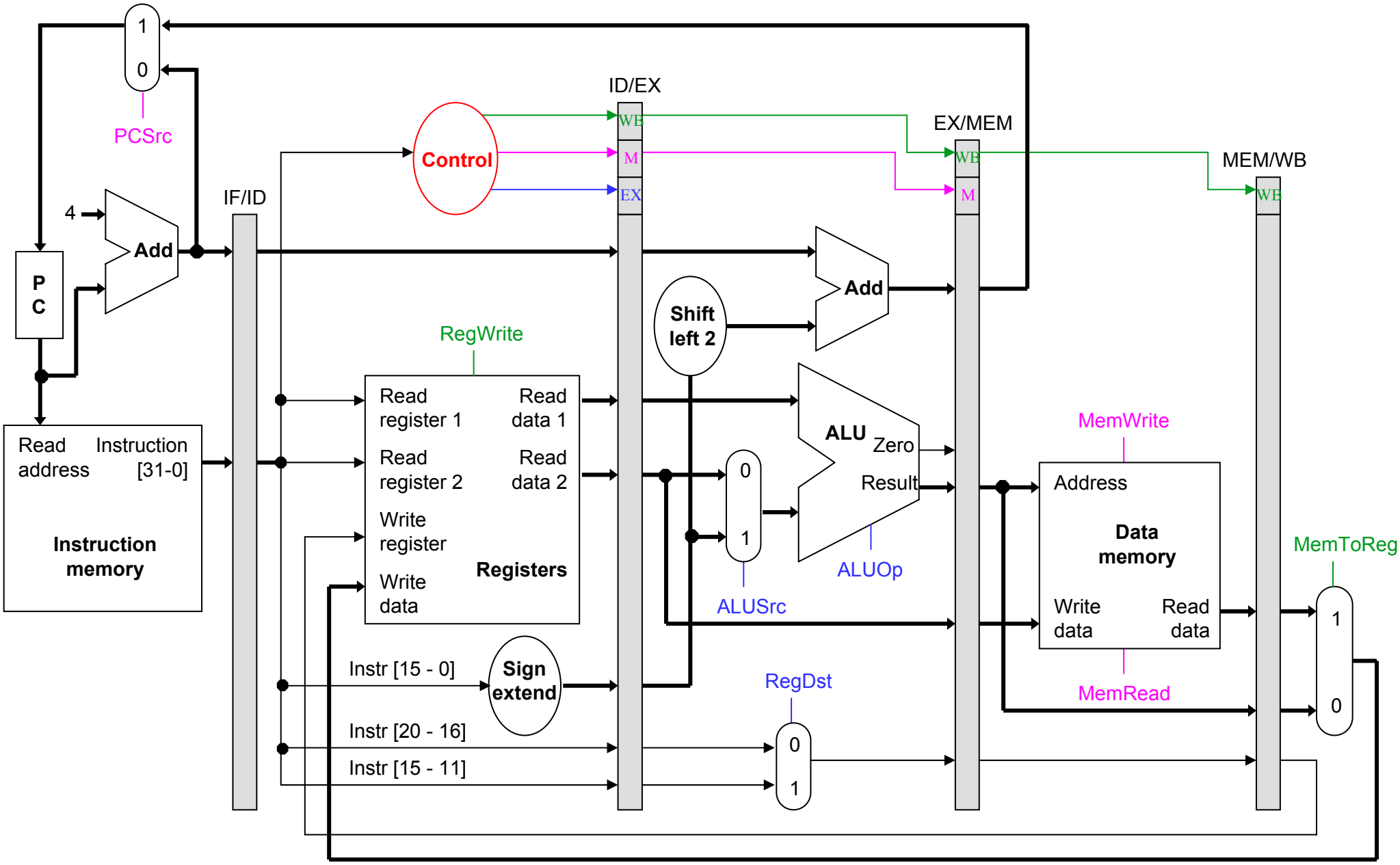


What about control signals?

- 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.

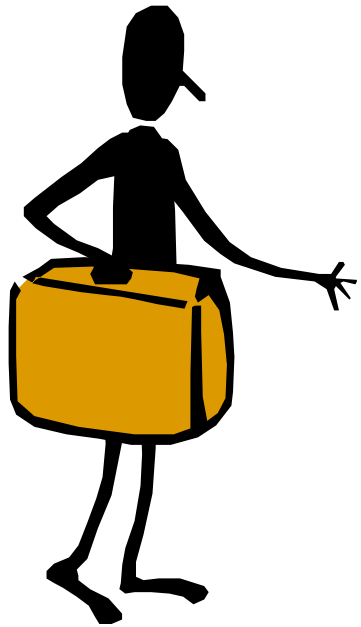
Stage	Control signals needed		
EX	ALUSrc	ALUOp	RegDst
MEM	MemRead	MemWrite	PCSrc
WB	RegWrite	MemToReg	

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.



Instruction set architectures and pipelining

- The MIPS instruction set was designed especially for easy pipelining.
  - All instructions are 32-bits long, so the instruction fetch stage just needs to read one word on every clock cycle.
  - Fields are in the same position in different instruction formats—the opcode is always the first six bits, rs is the next five bits, etc. This makes things easy for the ID stage.
  - MIPS is a register-to-register architecture, so arithmetic operations cannot contain memory references. This keeps the pipeline shorter and simpler.
- Pipelining is harder for older, more complex instruction sets.
  - If different instructions had different lengths or formats, the fetch and decode stages would need extra time to determine the actual length of each instruction and the position of the fields.
  - With memory-to-memory instructions, additional pipeline stages may be needed to compute effective addresses and read memory *before* the EX stage.

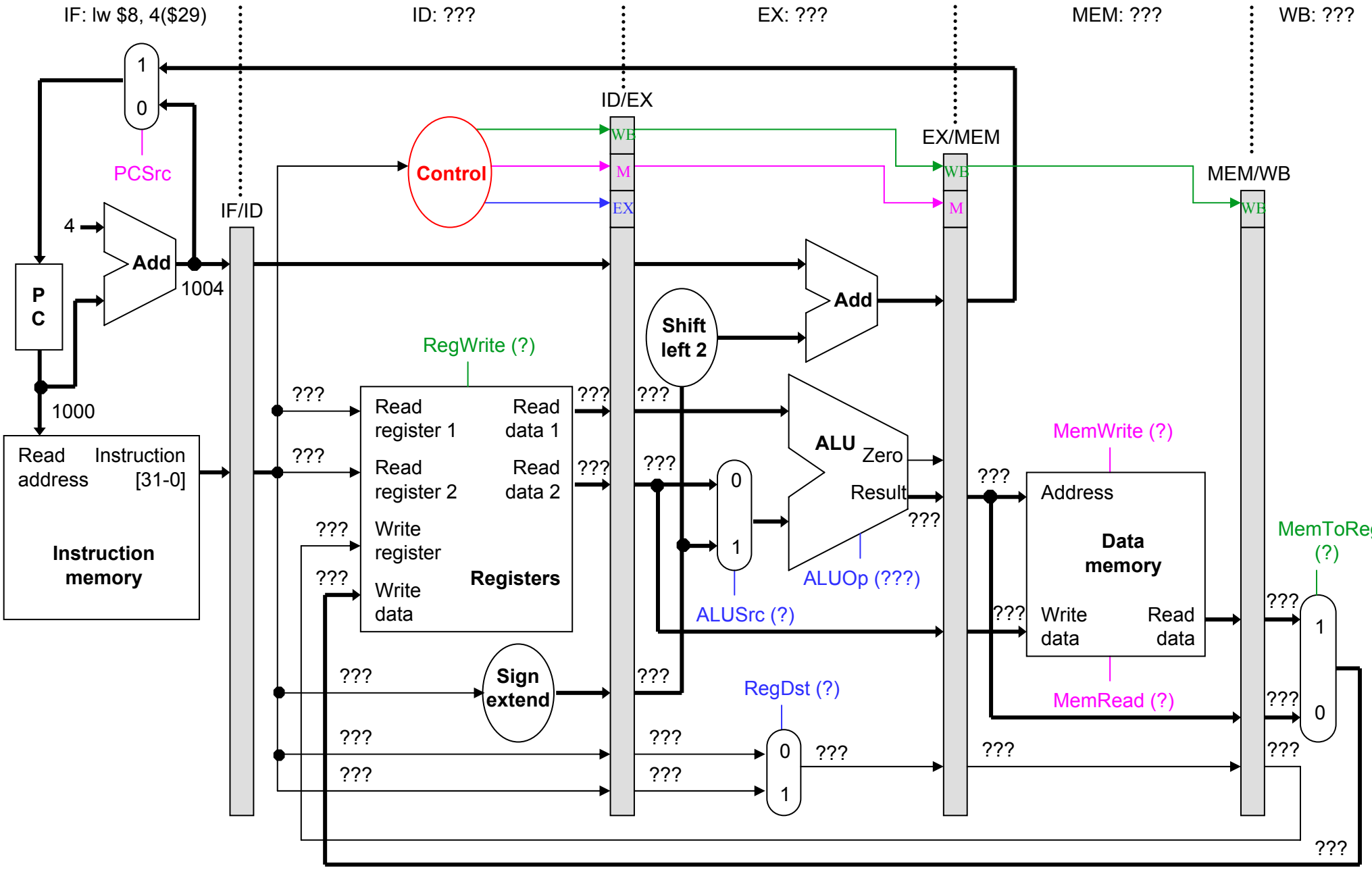
An example execution sequence

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

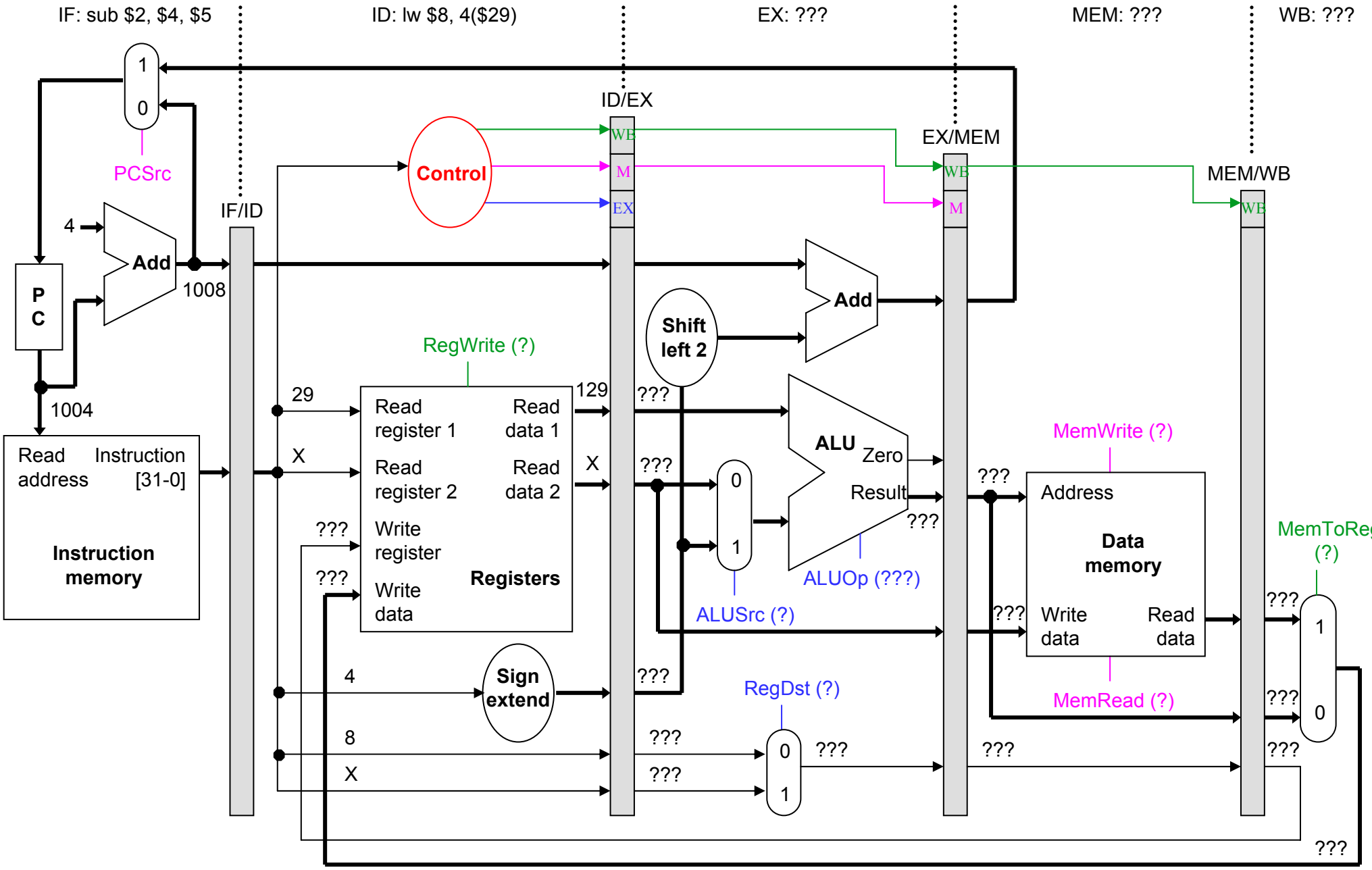
```
1000:  lw  $8, 4($29)
1004:  sub $2, $4, $5
1008:  and $9, $10, $11
1012:  or  $16, $17, $18
1016:  add $13, $14, $0
```

- 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.

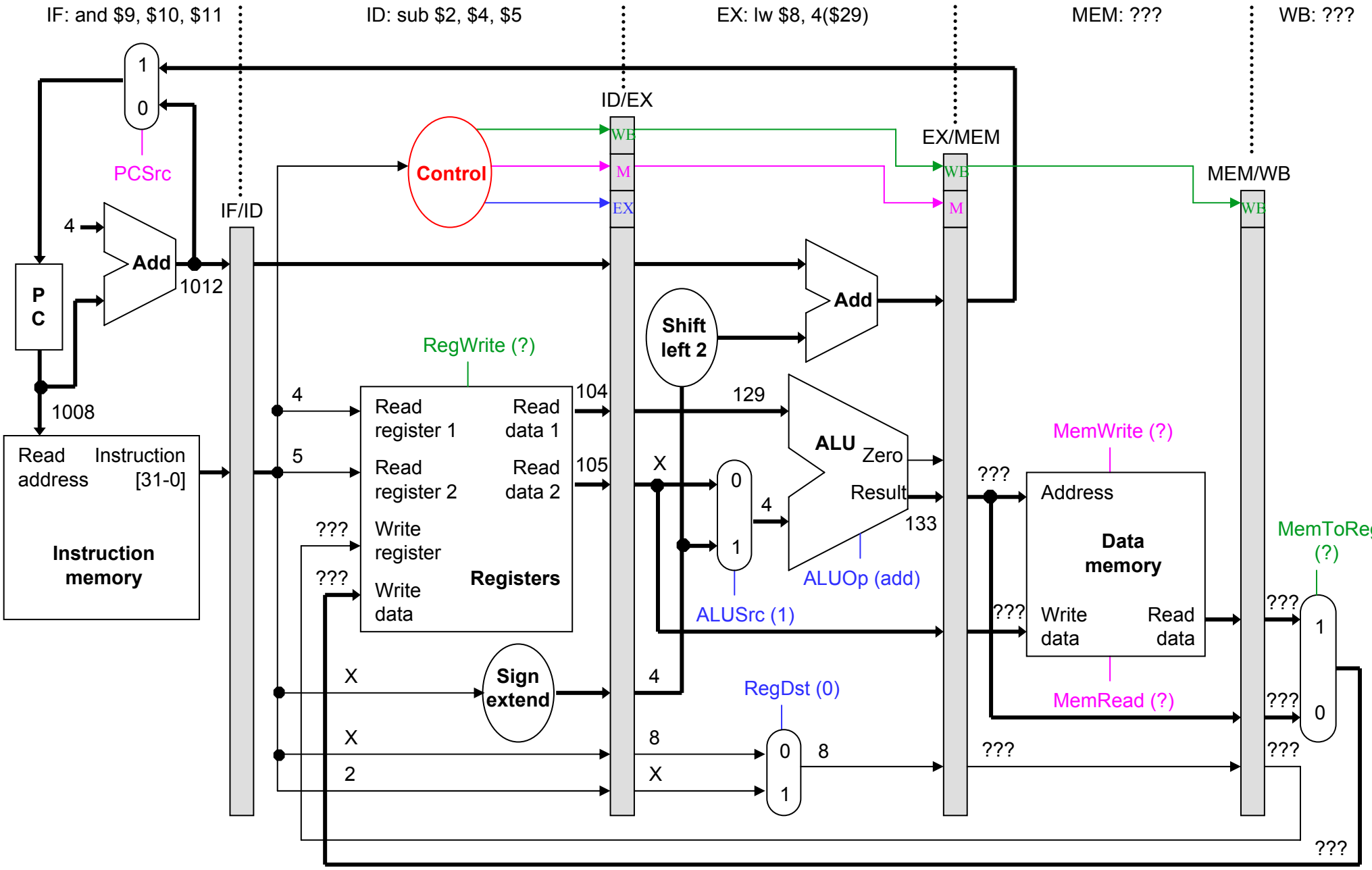
Cycle 1 (filling)



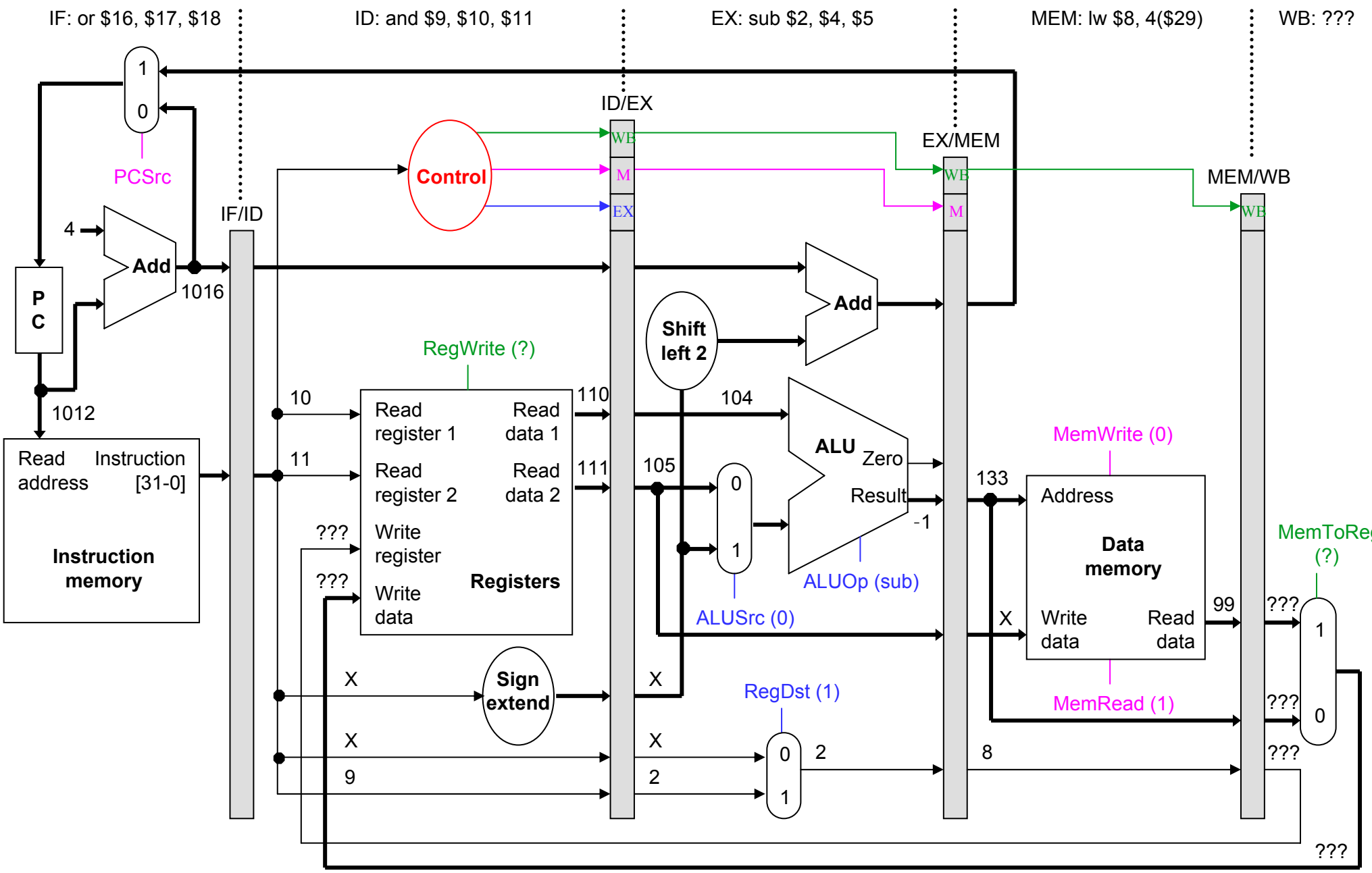
Cycle 2


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

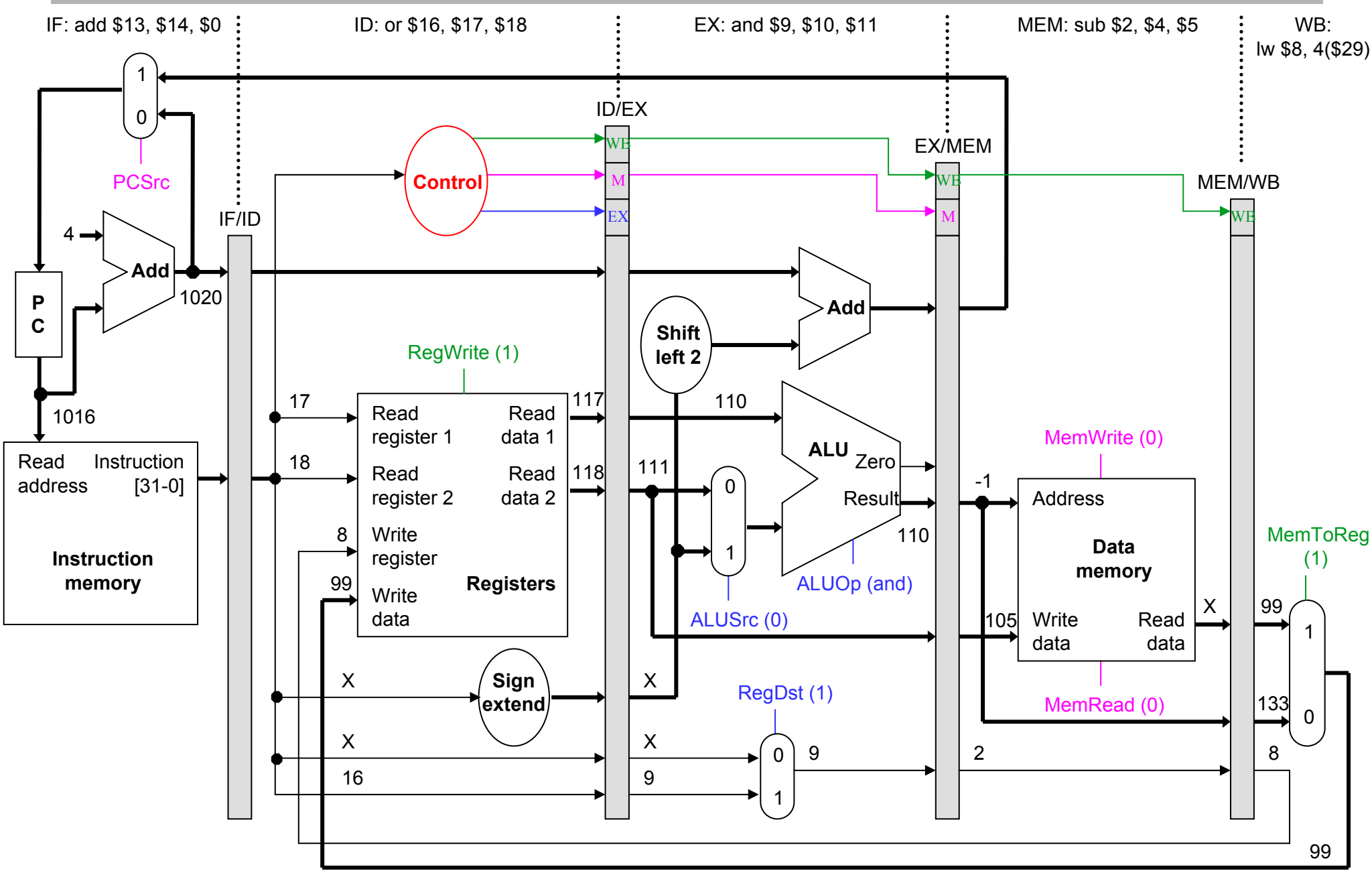
Cycle 3


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

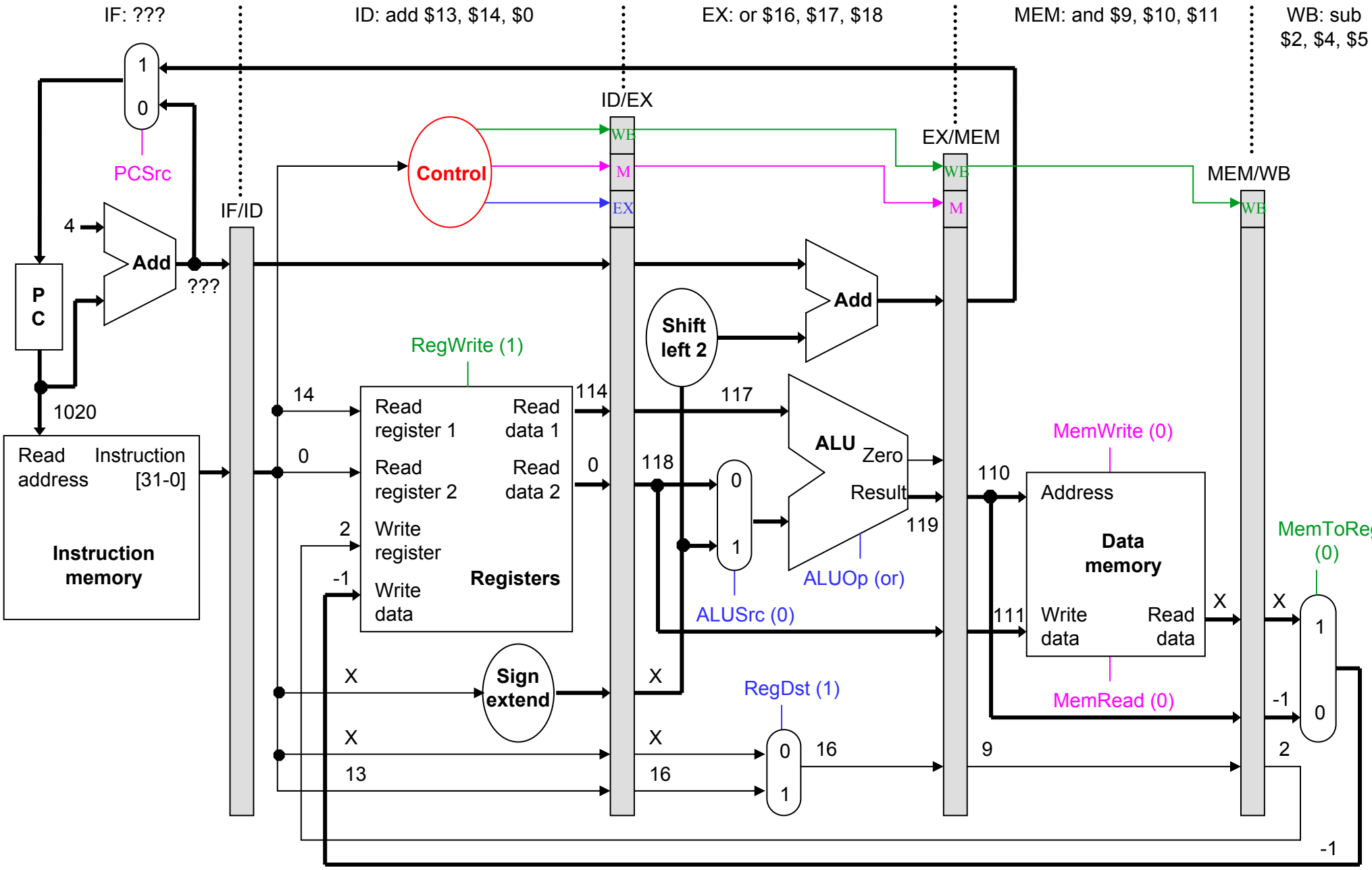
Cycle 4


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

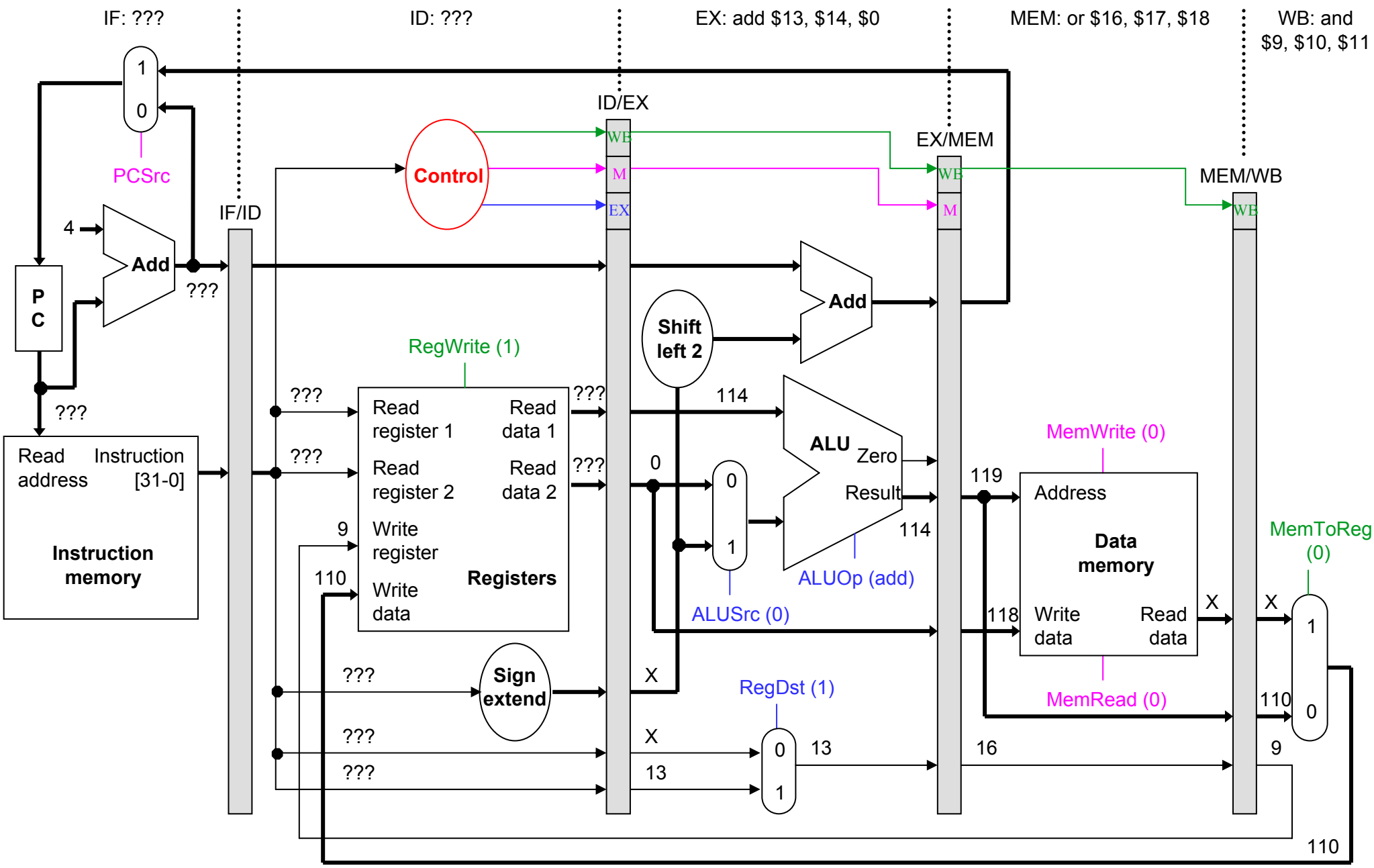
Cycle 5 (full)


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)
ID/EX
EX/MEM
MEM/WB
IF/ID
PCSrc
Control
WB
M
EX
Add
4
1020
P C
1016
Shift left 2
RegWrite (1)
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
17
18
8
99
117
118
110
111
ALU
Zero
Result
ALUOp (and)
ALUSrc (0)
MemWrite (0)
Address
Data memory
Write data
Read data
MemRead (0)
MemToReg (1)
-1
105
X
99
133
RegDst (1)
Sign extend
16
9
2
Instruction address
Instruction [31-0]
Instruction memory

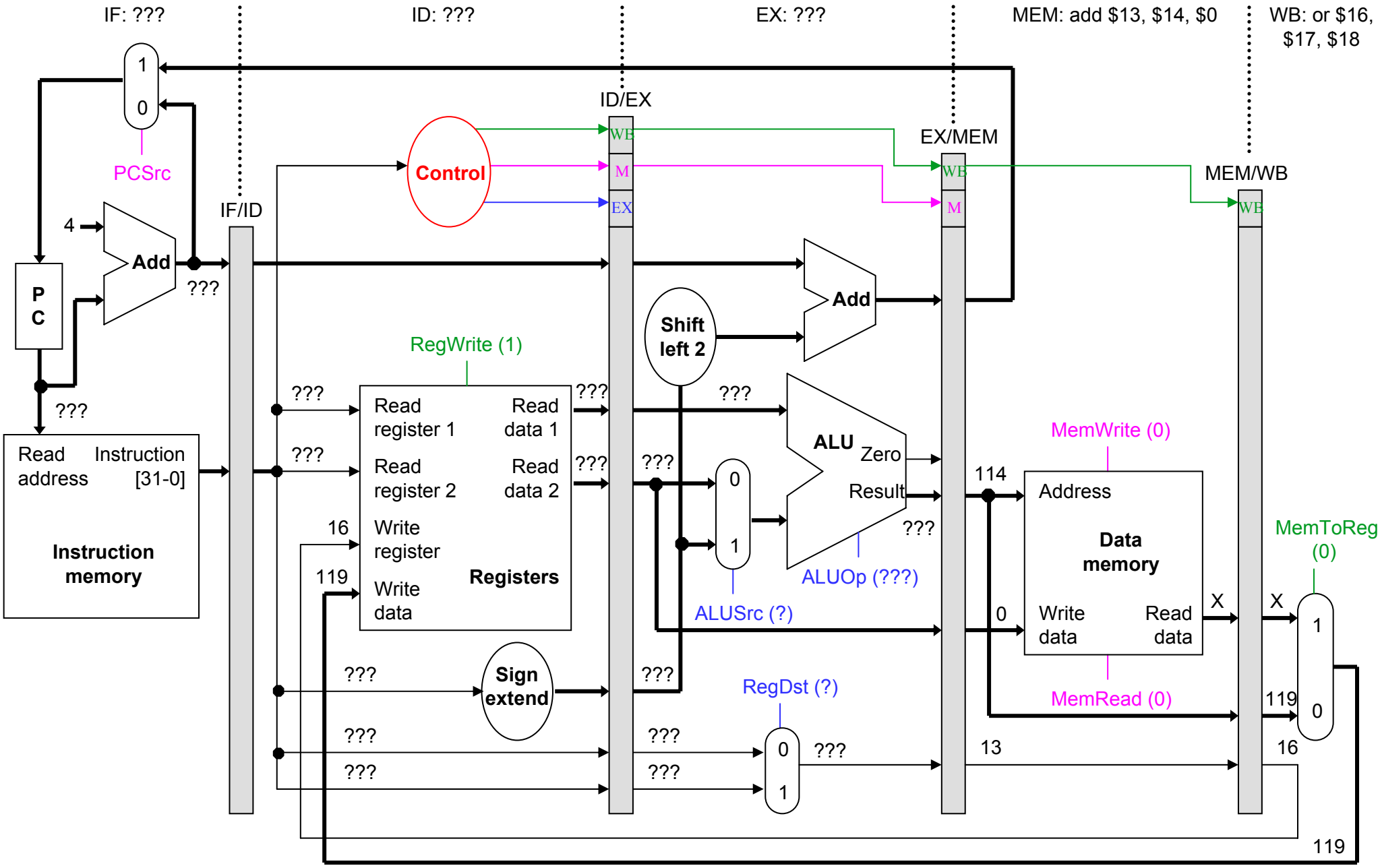
Cycle 6 (emptying)



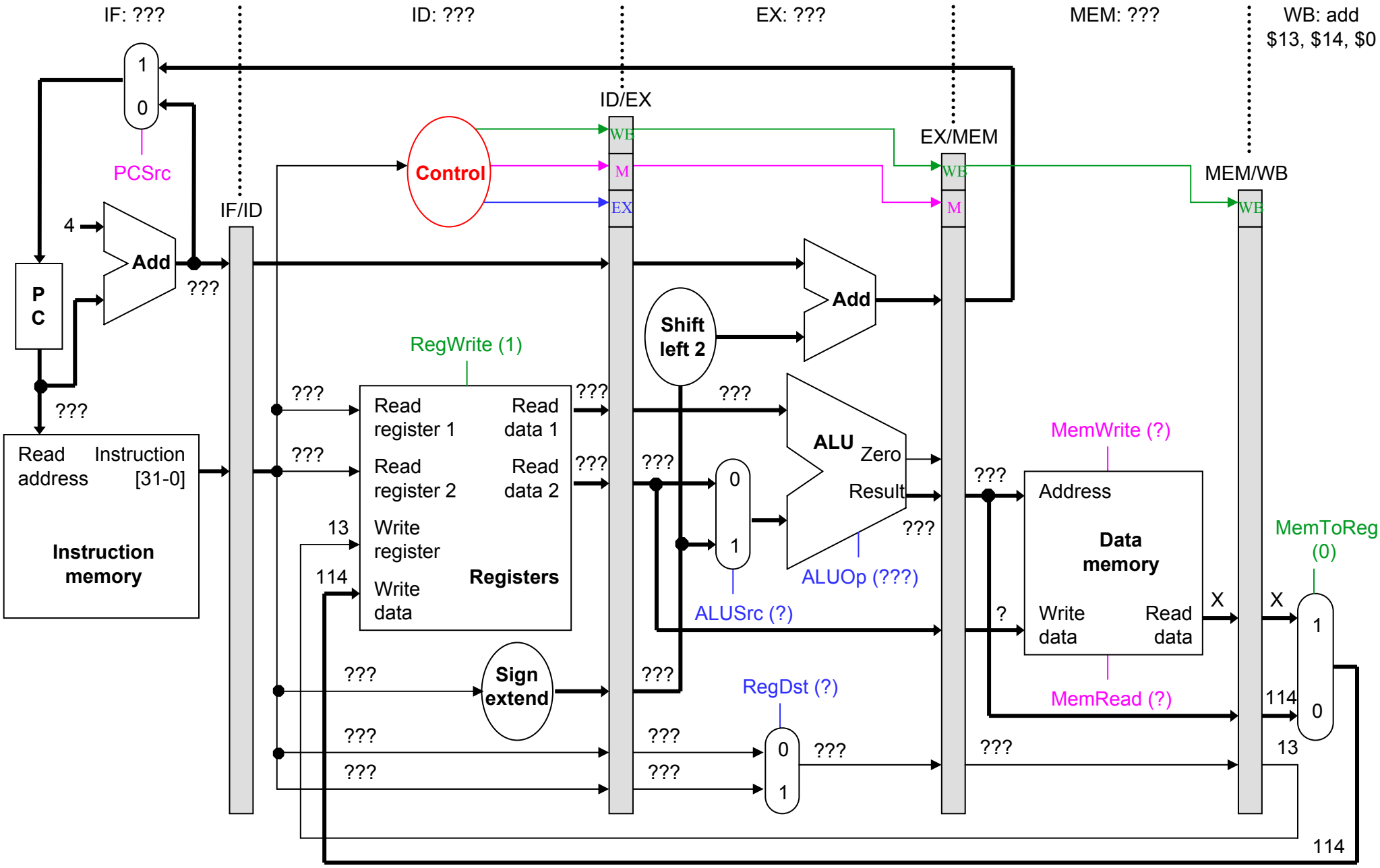
Cycle 7


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
WB
M
EX
Control
PCSrc
Add
4
???
P C
Shift left 2
RegWrite (1)
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
Read address
Instruction [31-0]
Instruction memory
ALU
Zero
Result
ALUOp (add)
ALUSrc (0)
MemWrite (0)
MemRead (0)
MemToReg (0)
Address
Data memory
Write data
Read data
RegDst (1)
Sign extend
114
119
118
16
13
9
110
X
0
1

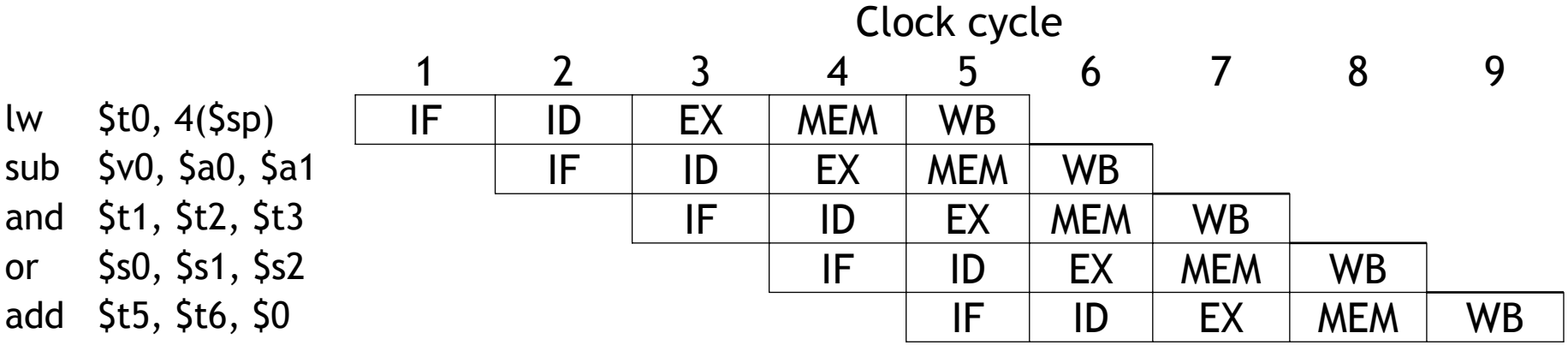
Cycle 8


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

Cycle 9


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

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 combines ideas from the single and multicycle processors that we saw earlier.
  - It uses multiple memories and ALUs.
  - Instruction execution is split into several stages.
- 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.
- Have a great Spring Break!