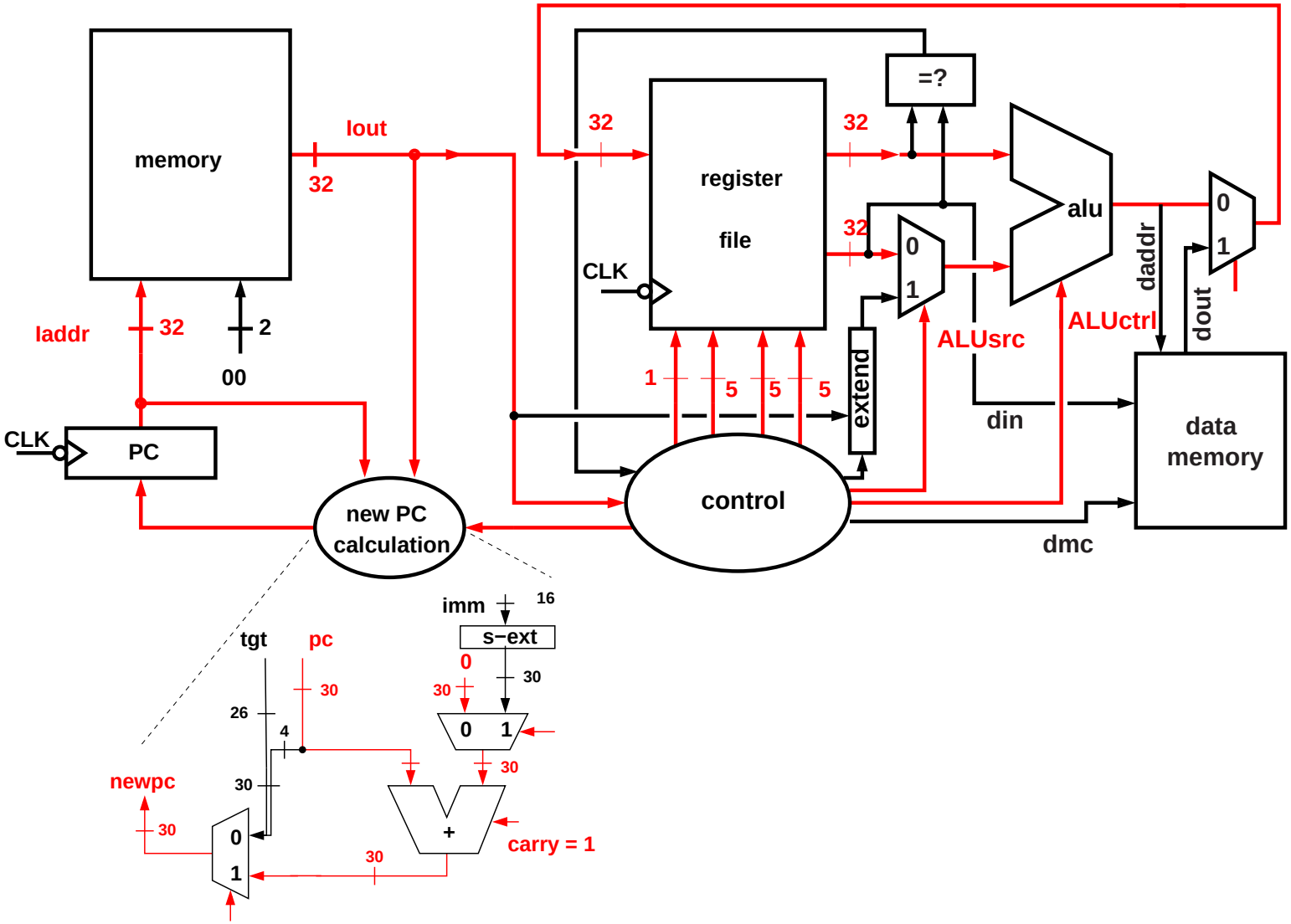
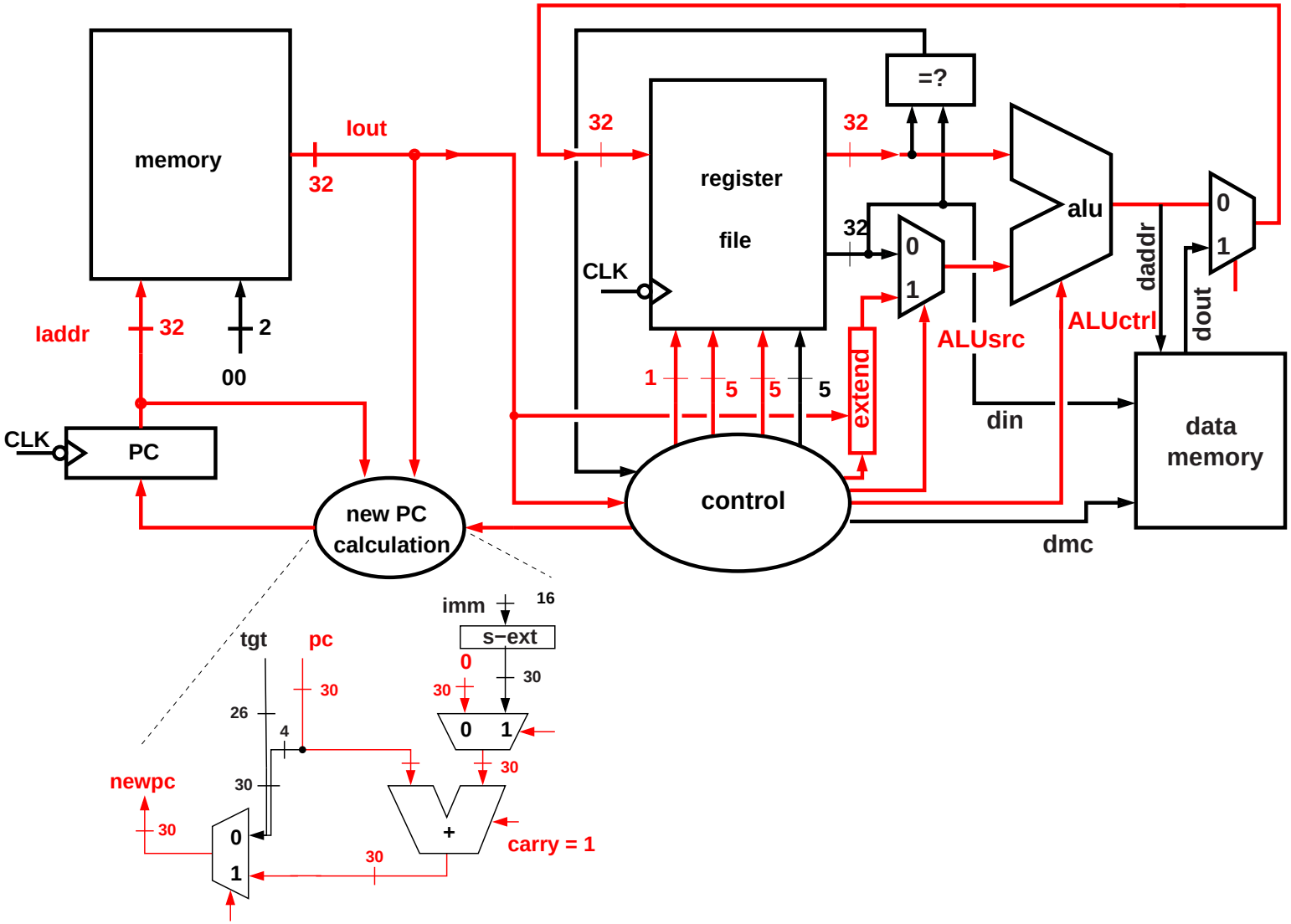
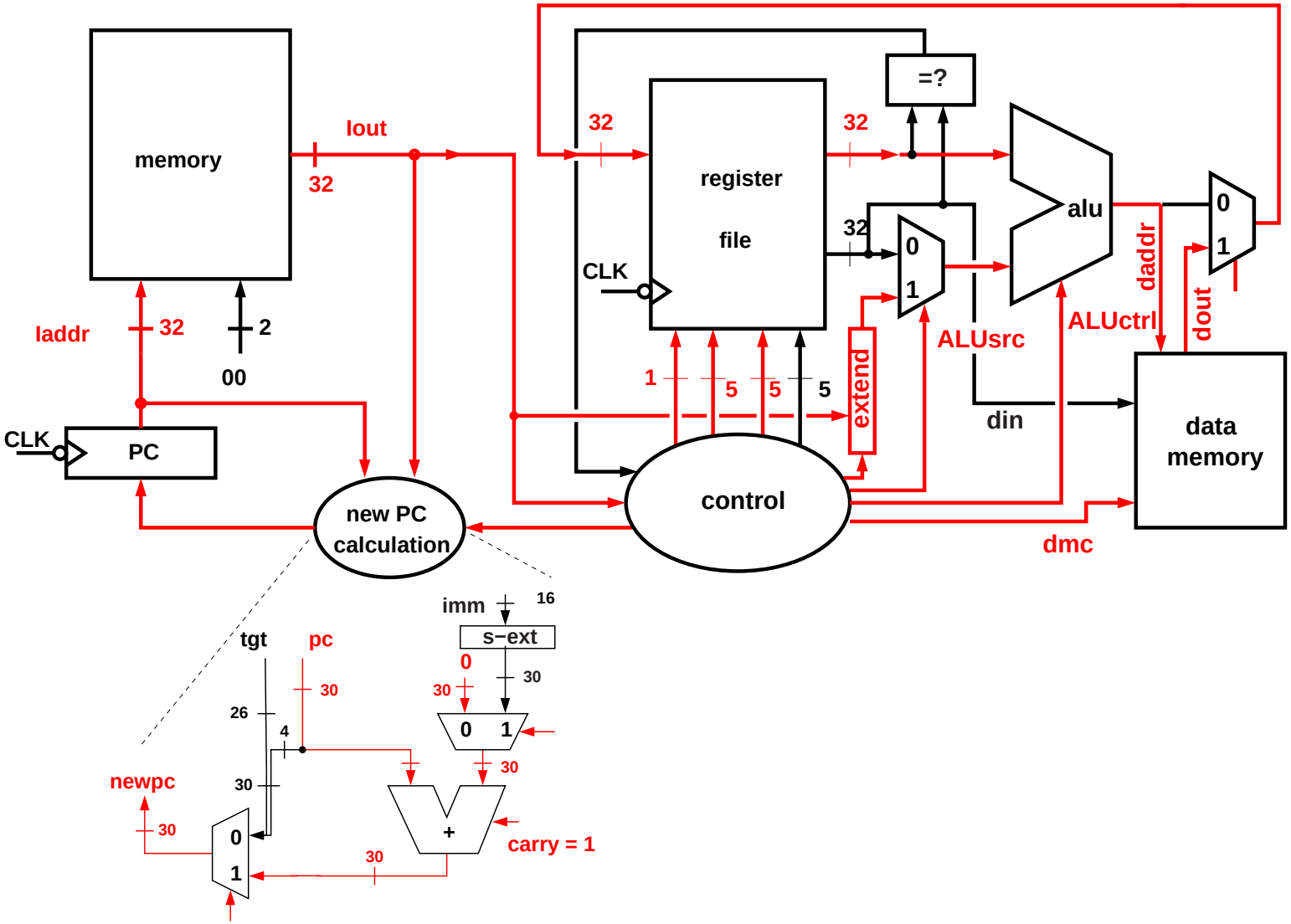
Datapath: addu



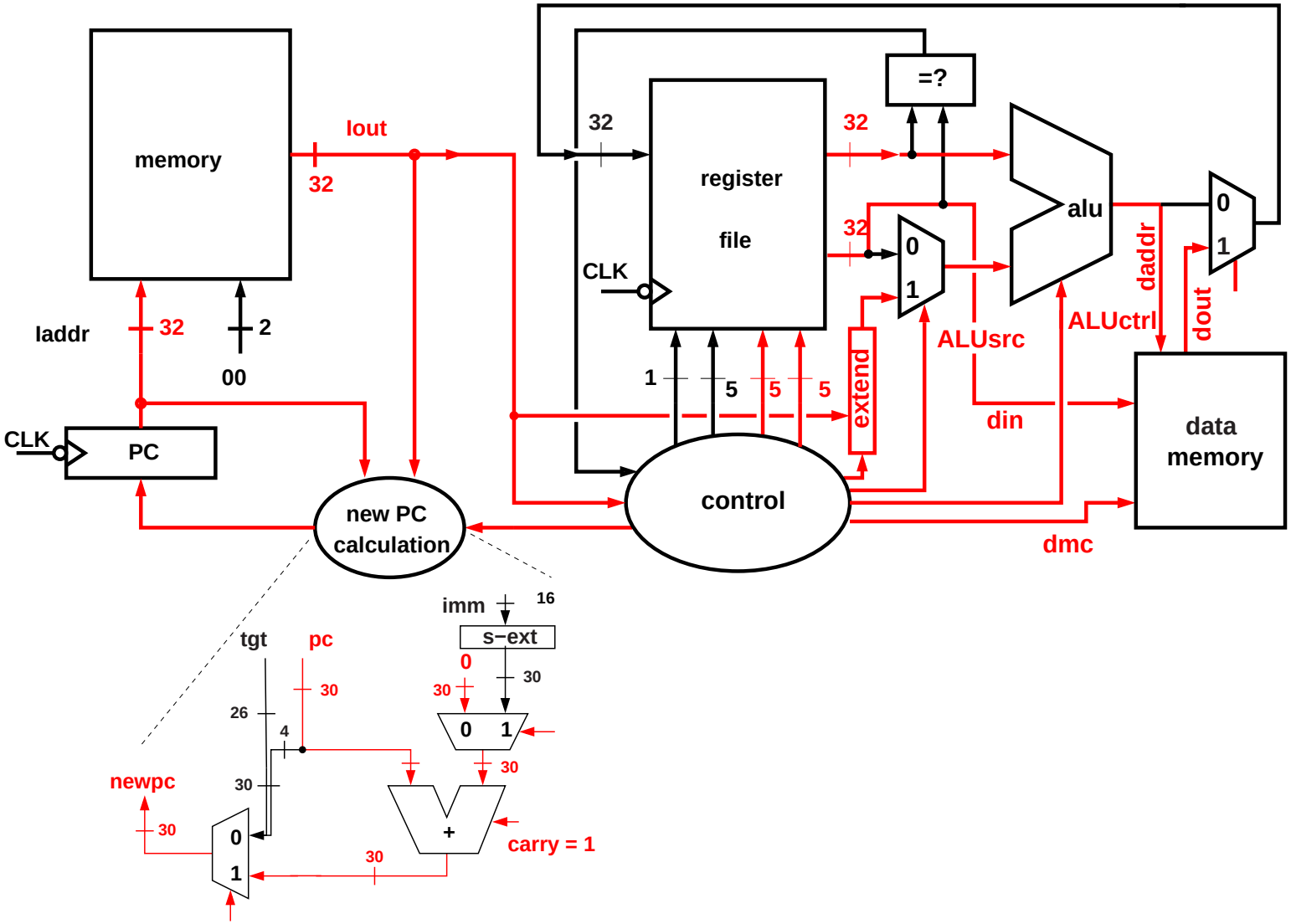
Datapath: addiu



Datapath: lw

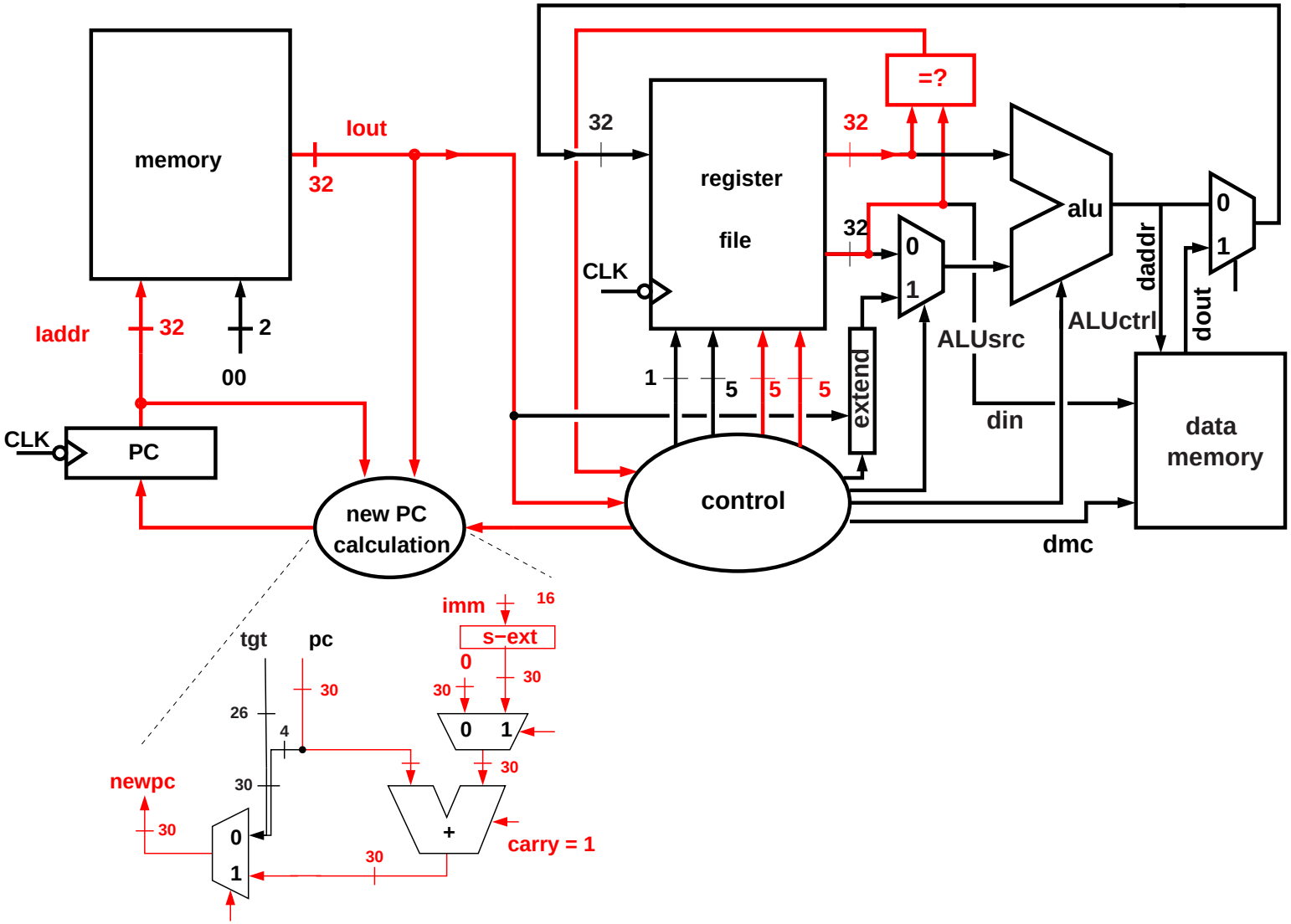

memory
Iout
32
register
file
CLK
=?
alu
dadder
dout
ALUctrl
ALUsrc
din
data
memory
Iaddr
32
2
00
CLK
PC
new PC
calculation
control
dmc
extend
1
5
5
5
imm
16
s-ext
tgt
pc
26
4
30
newpc
carry = 1
+

Datapath: sw


memory
Iout
32
laddr
32
2
00
CLK
PC
new PC
calculation
register
file
CLK
32
32
32
=?
alu
0
1
0
1
dadder
dout
ALUctrl
ALUsrc
din
extend
1
5
5
5
control
data
memory
dmc
imm
16
s-ext
0
30
30
0 1
30
tgt
pc
26
4
30
30
newpc
30
0
1
30
+
carry = 1

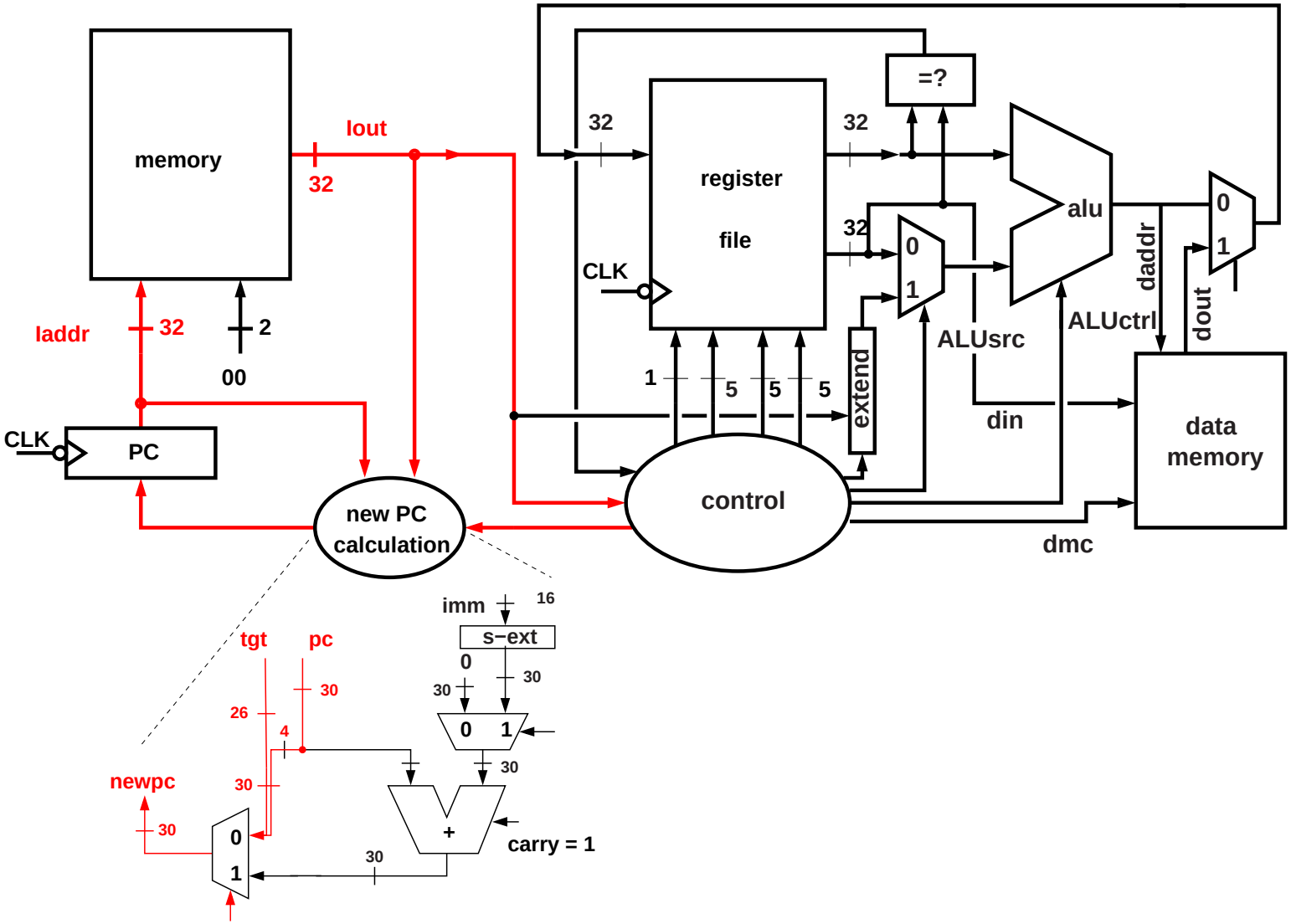

CSL

Datapath: beq



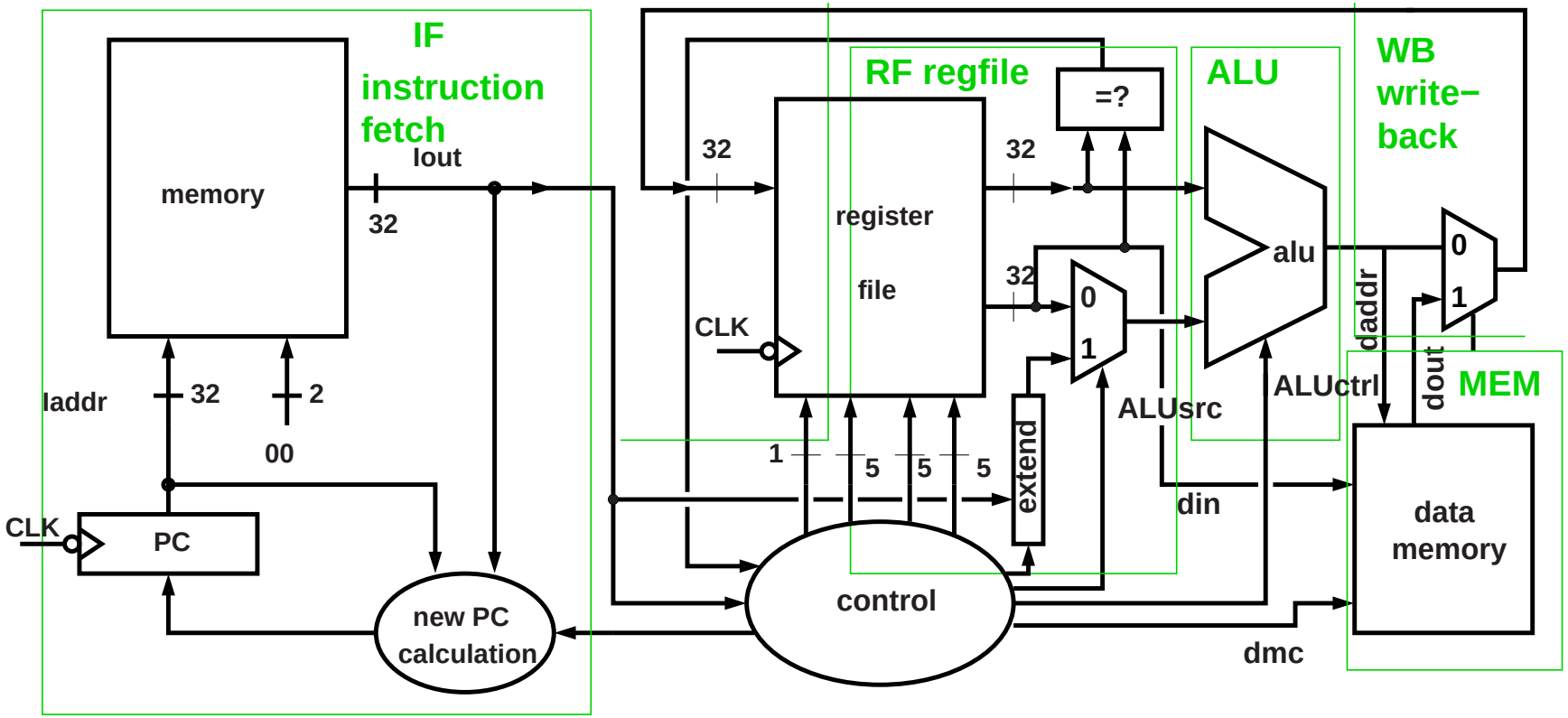
CSL

Datapath: j


memory
Iout
32
laddr
32
2
00
CLK
PC
new PC
calculation
register
file
CLK
32
32
32
=?
alu
0
1
ALUctrl
ALUsrc
dout
daddr
0
1
extend
1
5
5
5
din
control
data
memory
dmc
imm
16
s−ext
0
30
30
0 1
30
+
carry = 1
tgt
pc
30
26
4
30
newpc
30
0
1
30

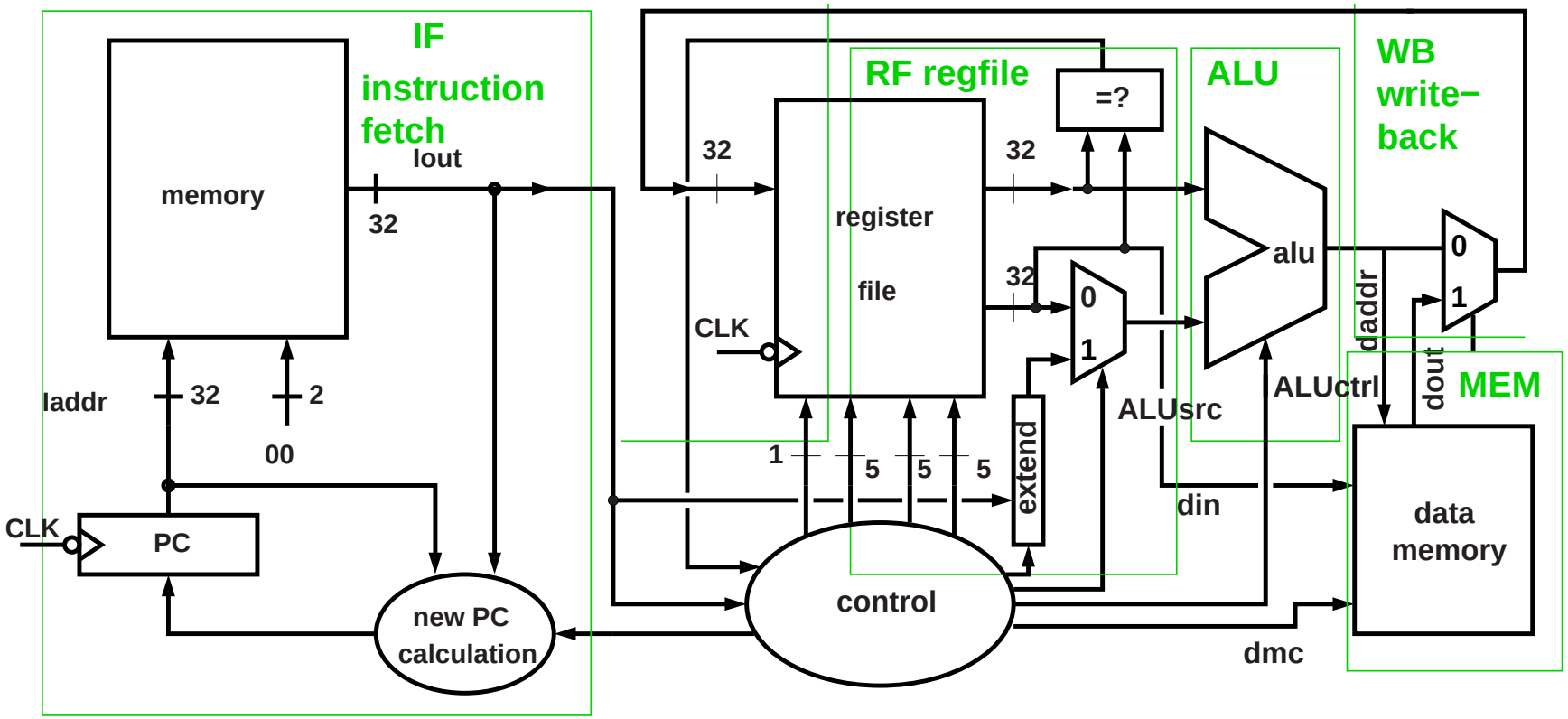

CSL

Performance



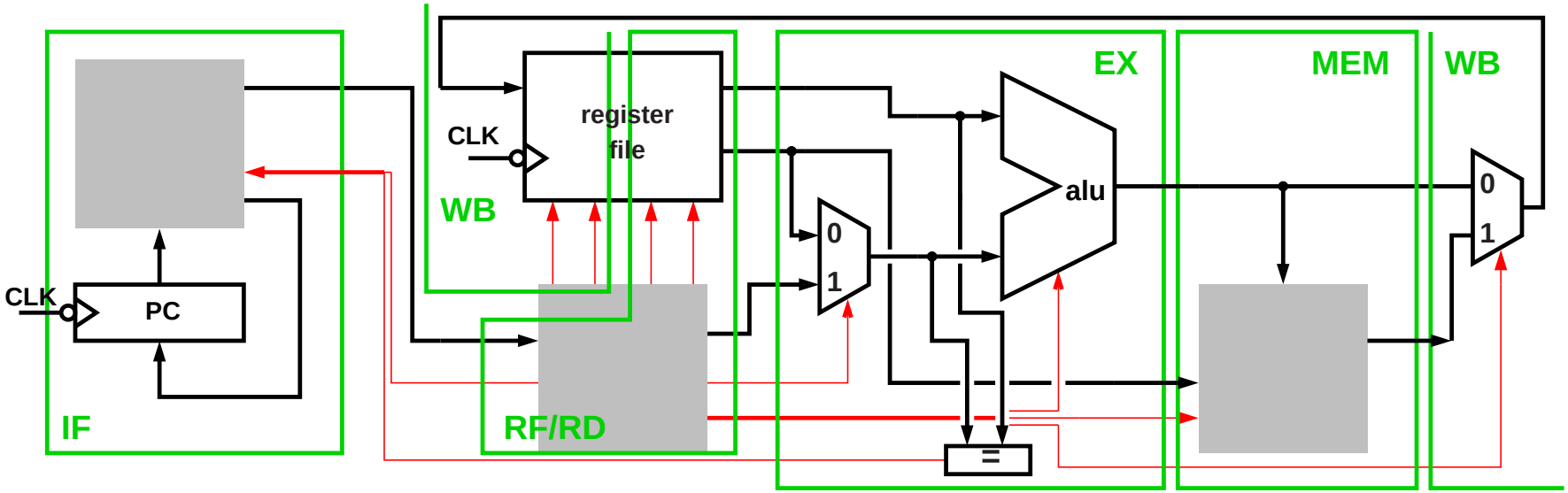
How fast can we run the clock?

Multi-Cycle Datapath



Observation: we can run the clock faster by partitioning the execution into multiple *stages*.

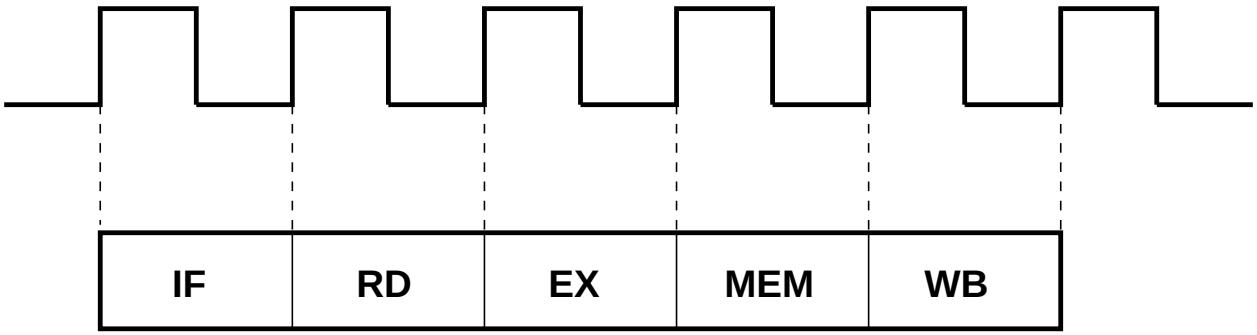
Multi-Cycle Datapath

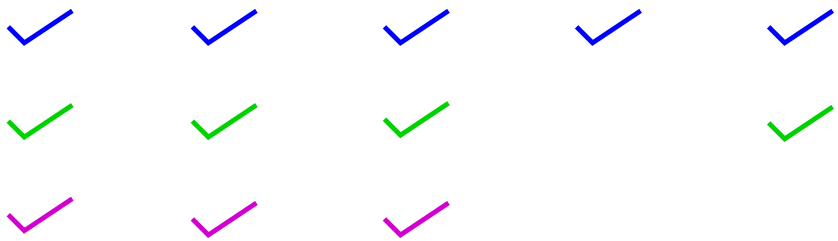


Stages: instruction fetch, register fetch/decode, execute, memory, writeback.

Idea: break down single instruction execution into smaller parts, some instructions skip parts of execution.

Multi-Cycle Datapath





```
lw $1,-4($3)
add $1,$2,$3
beq $1,$2,L
```

- PC needs a write-enable signal
- Design FSM for control

(textbook has details: sections 5.4, 5.5)

Performance

Suppose we had:

- IF: 5ns, RD: 3ns, ALU: 6ns, MEM: 5ns, WB: 4ns

Single-cycle datapath cycle time: 23ns

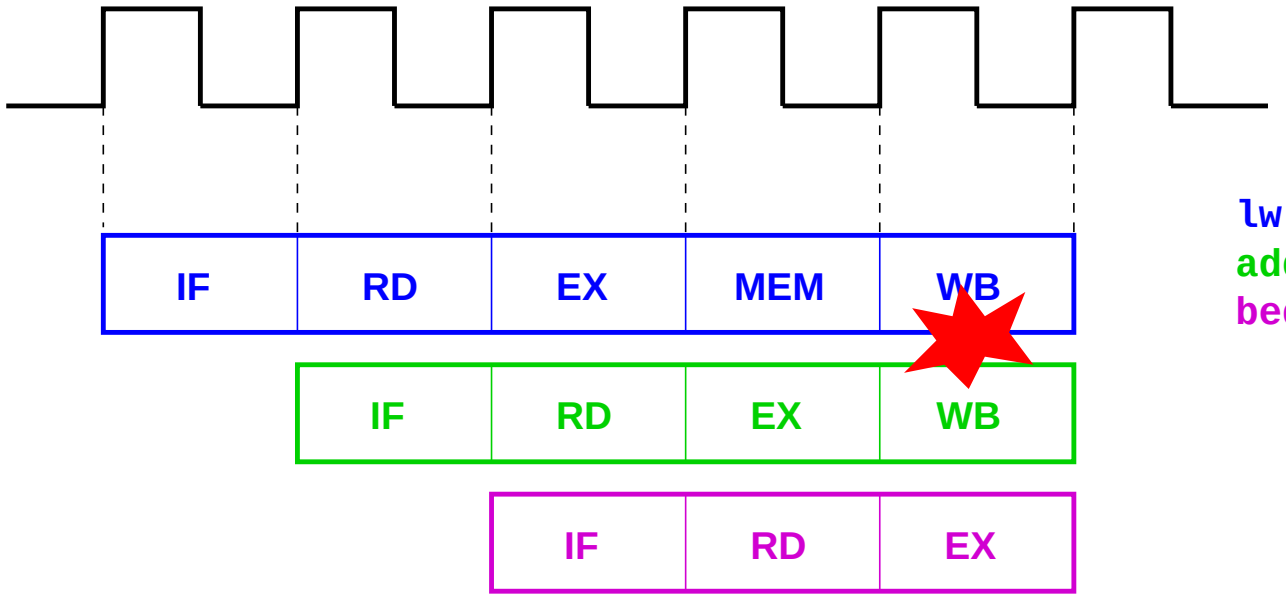
Multi-cycle datapath:

- lw: 30ns
- add: 24ns
- beq: 18ns

Is this better? Depends on the instruction mix!
(should have *balanced* pipeline stages)

Pipelining

Why not start executing the next instruction as soon as the first one has been fetched?

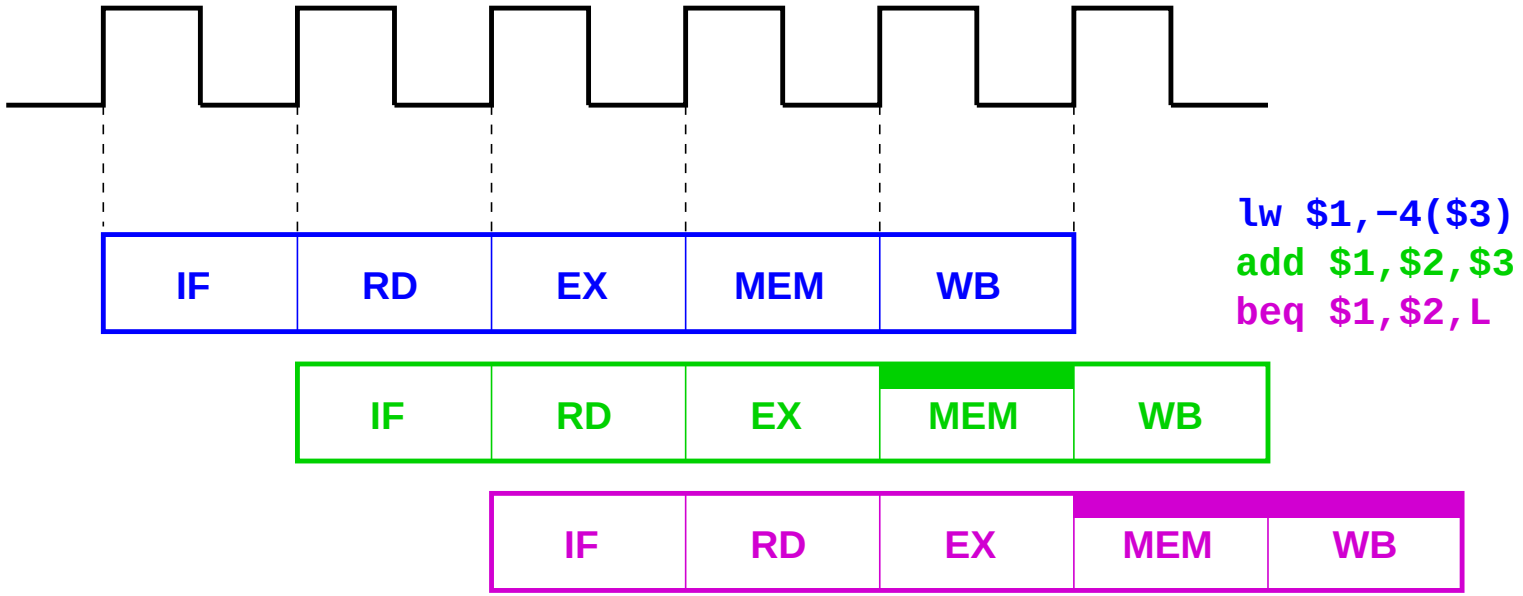



```
lw $1,-4($3)
add $1,$2,$3
beq $1,$2,L
```

Hardware resource conflict...

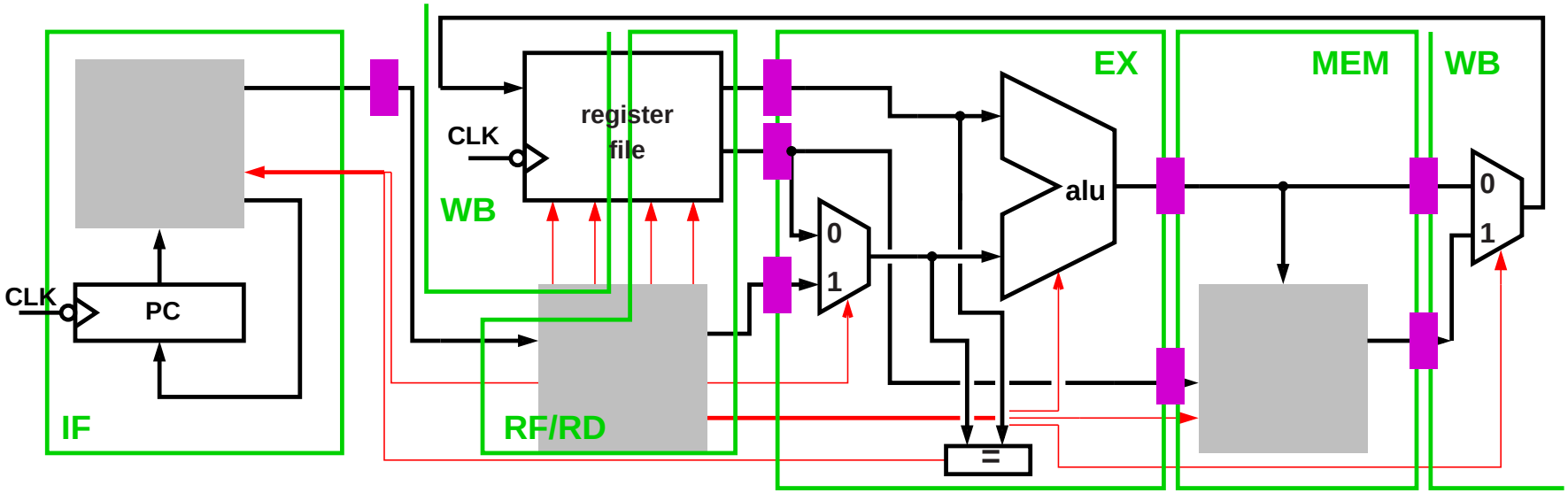
Pipelining

Simple fix: make *all instructions* take the same number of cycles.



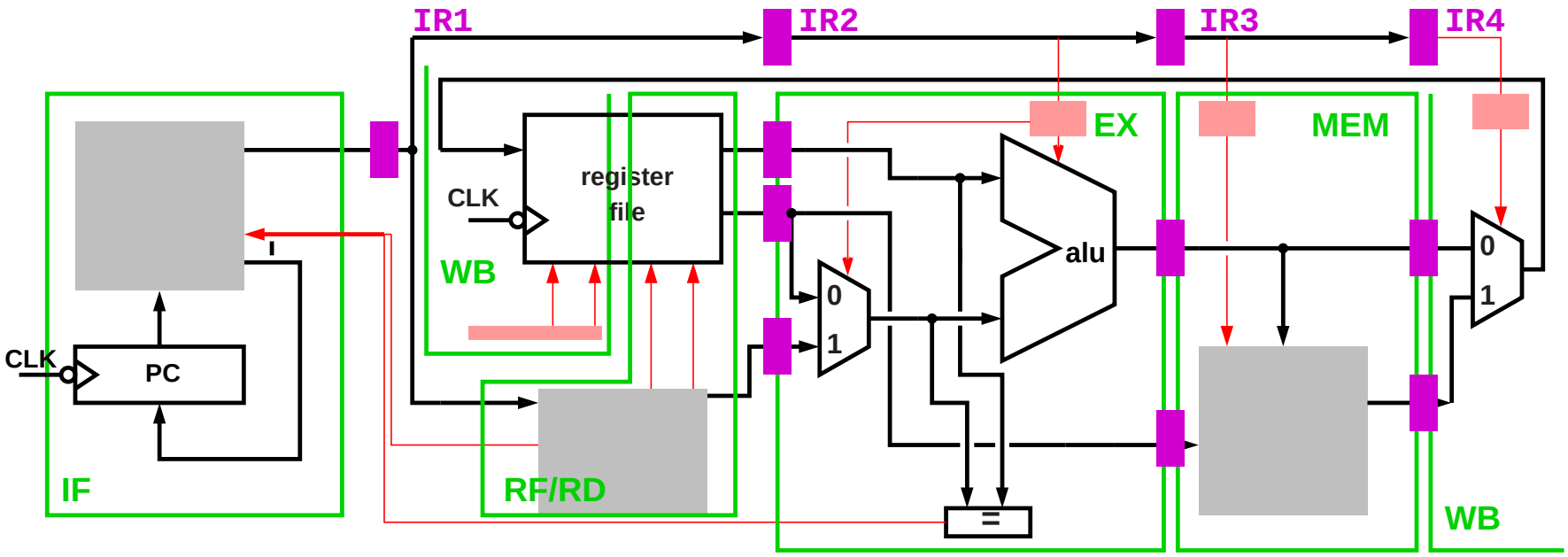
Each instruction takes 30ns, but we complete an instruction every 6ns (caveat: flop/latch overhead).

Pipelined Datapath



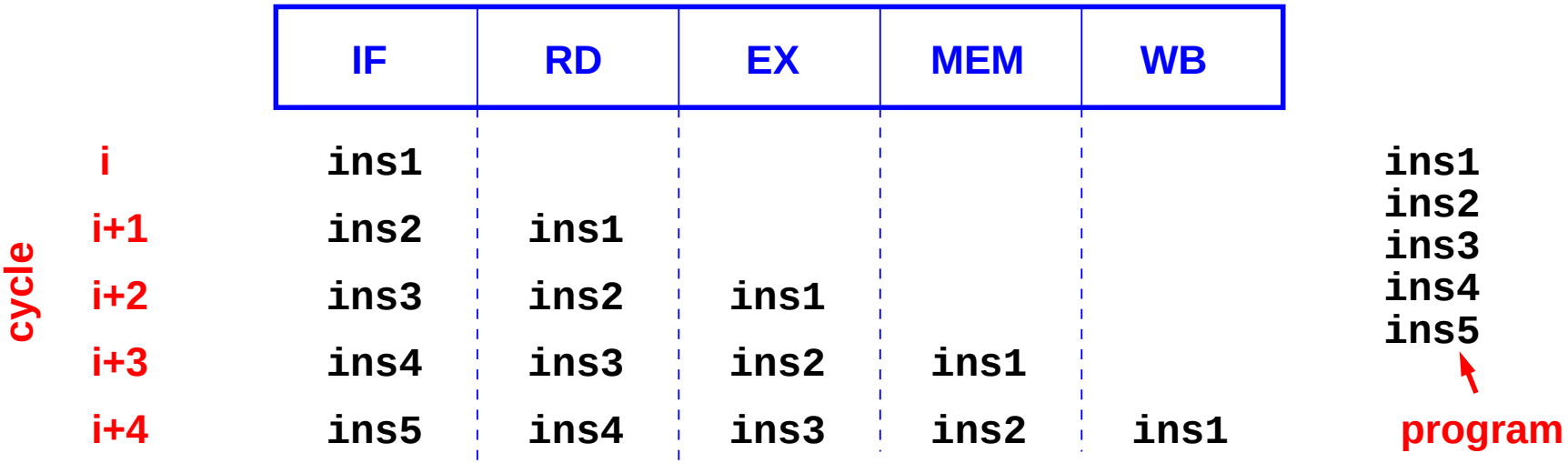
Introduce positive-edge triggered flip-flops between pipeline stages to hold intermediate values.

Pipelined Datapath: Control



Basic idea: copy the instruction, and generate local control in each pipeline stage.

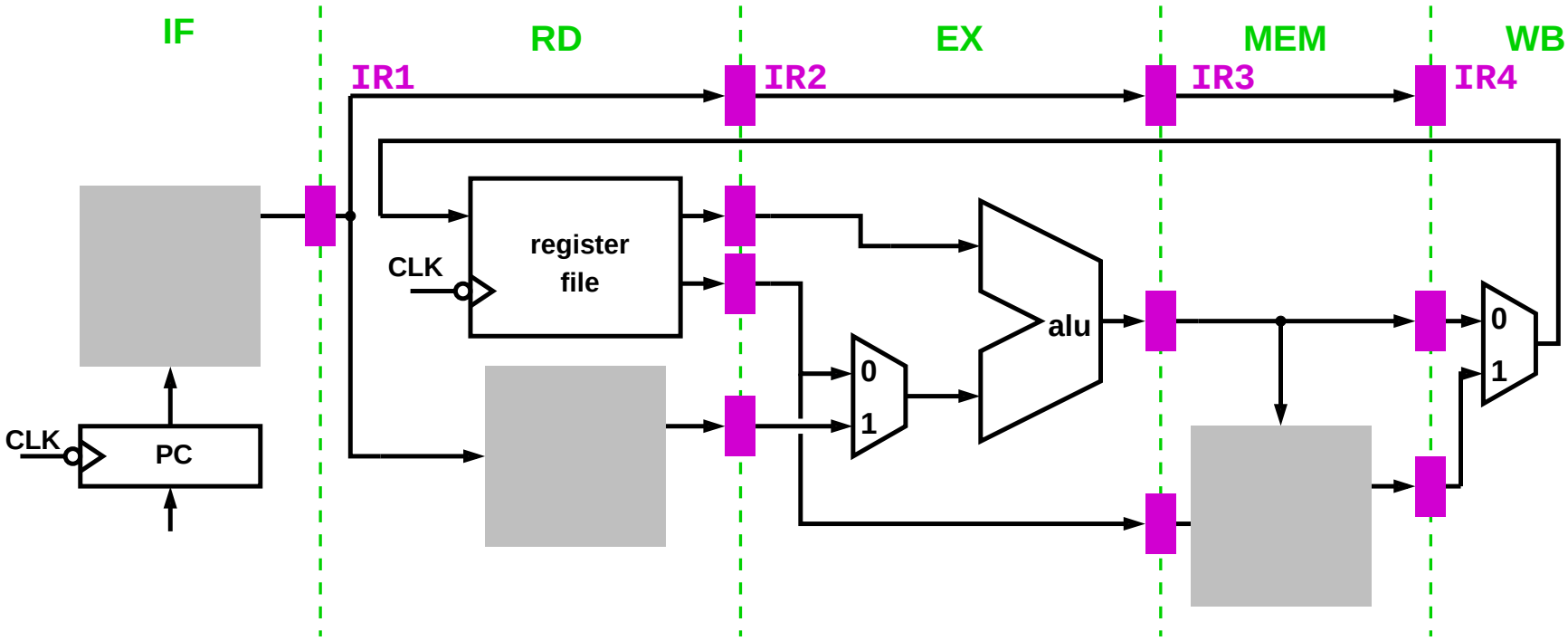
Pipelined Datapath: Timing



Instructions flow through the datapath, advancing every cycle.

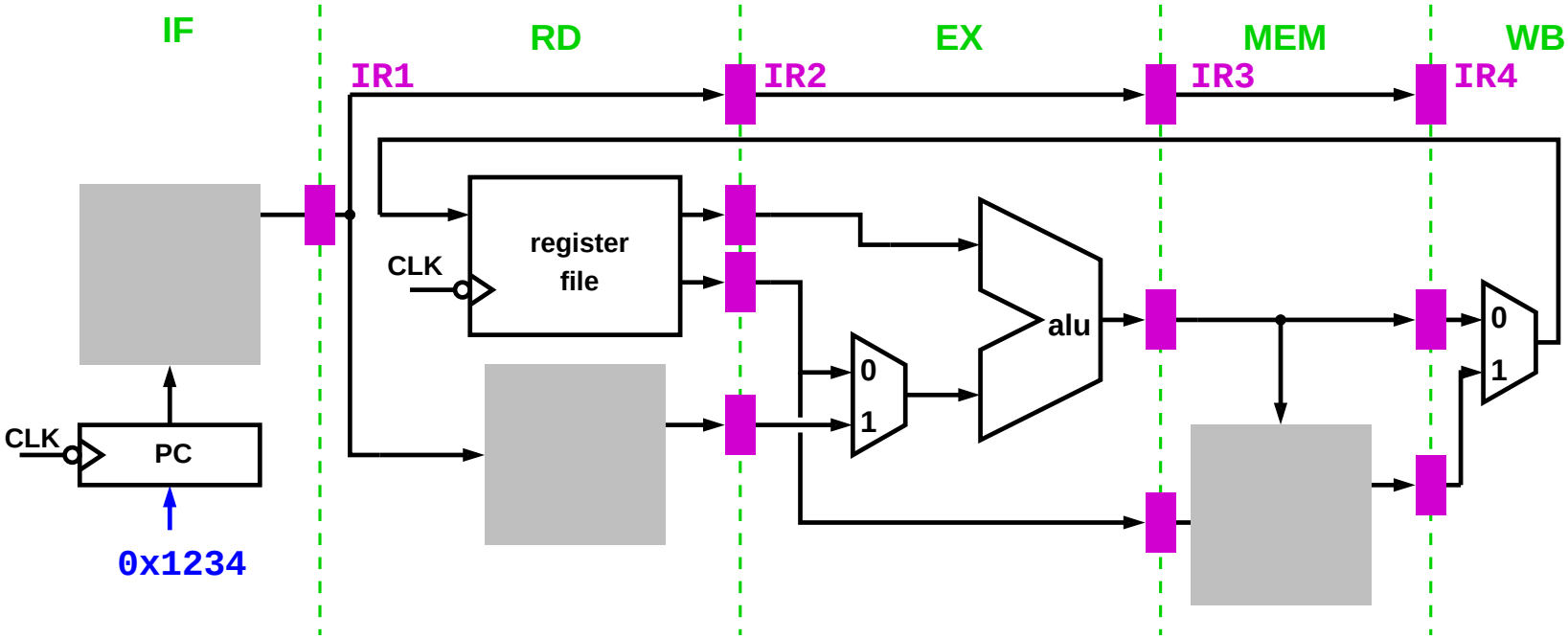
CSL

Pipelined Datapath: Timing

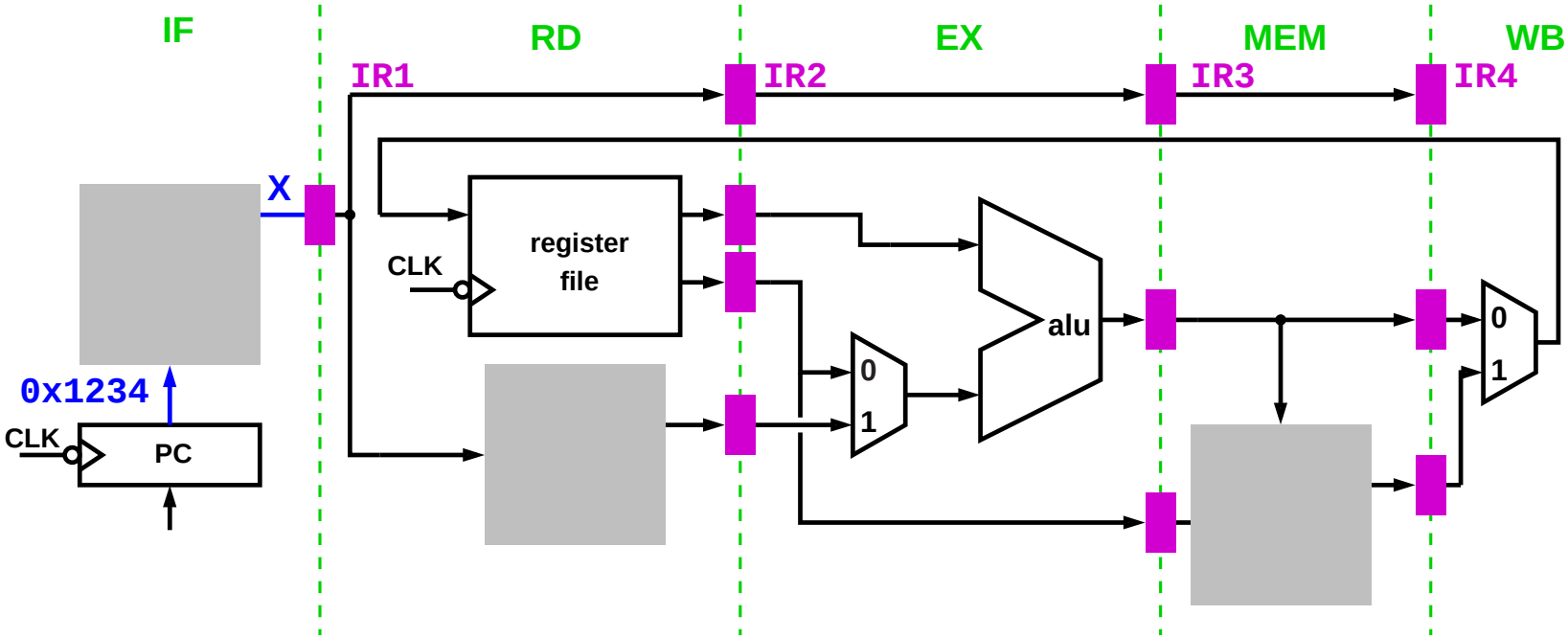


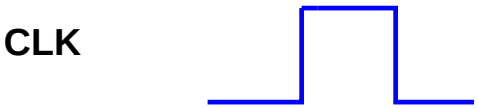
- Flops between stages: positive-edge triggered
- PC, register file: negative-edge triggered

Single Instruction Execution

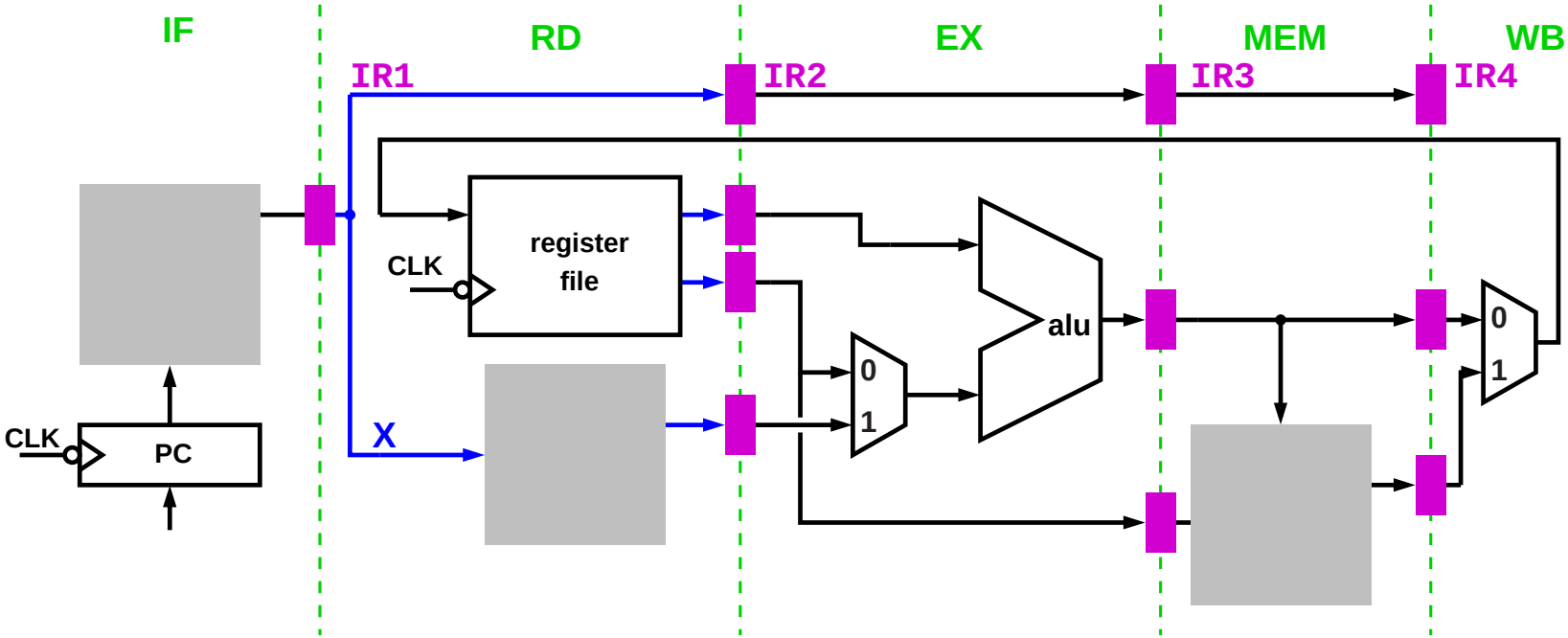


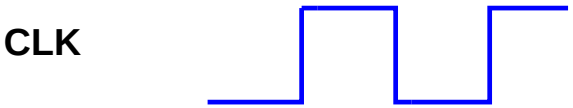
Single Instruction Execution



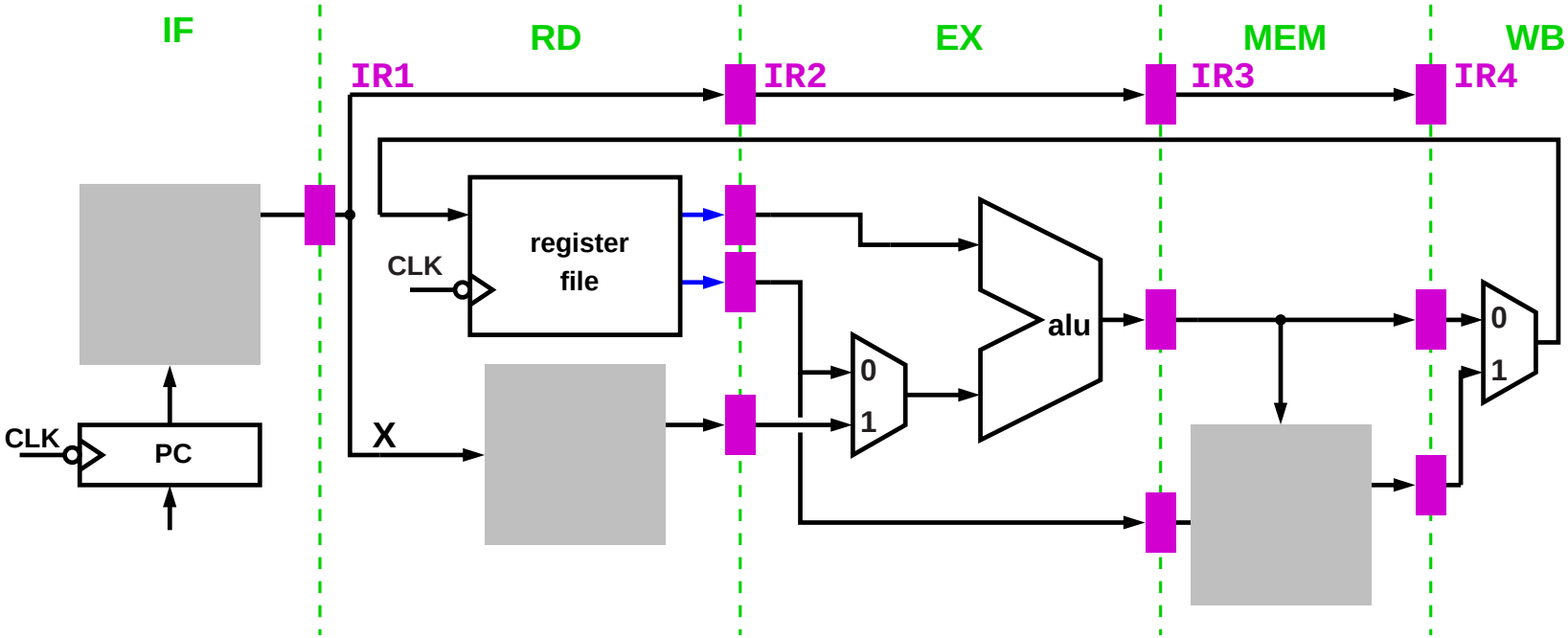


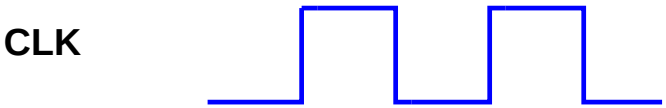
Single Instruction Execution





Single Instruction Execution





Register file output can change!