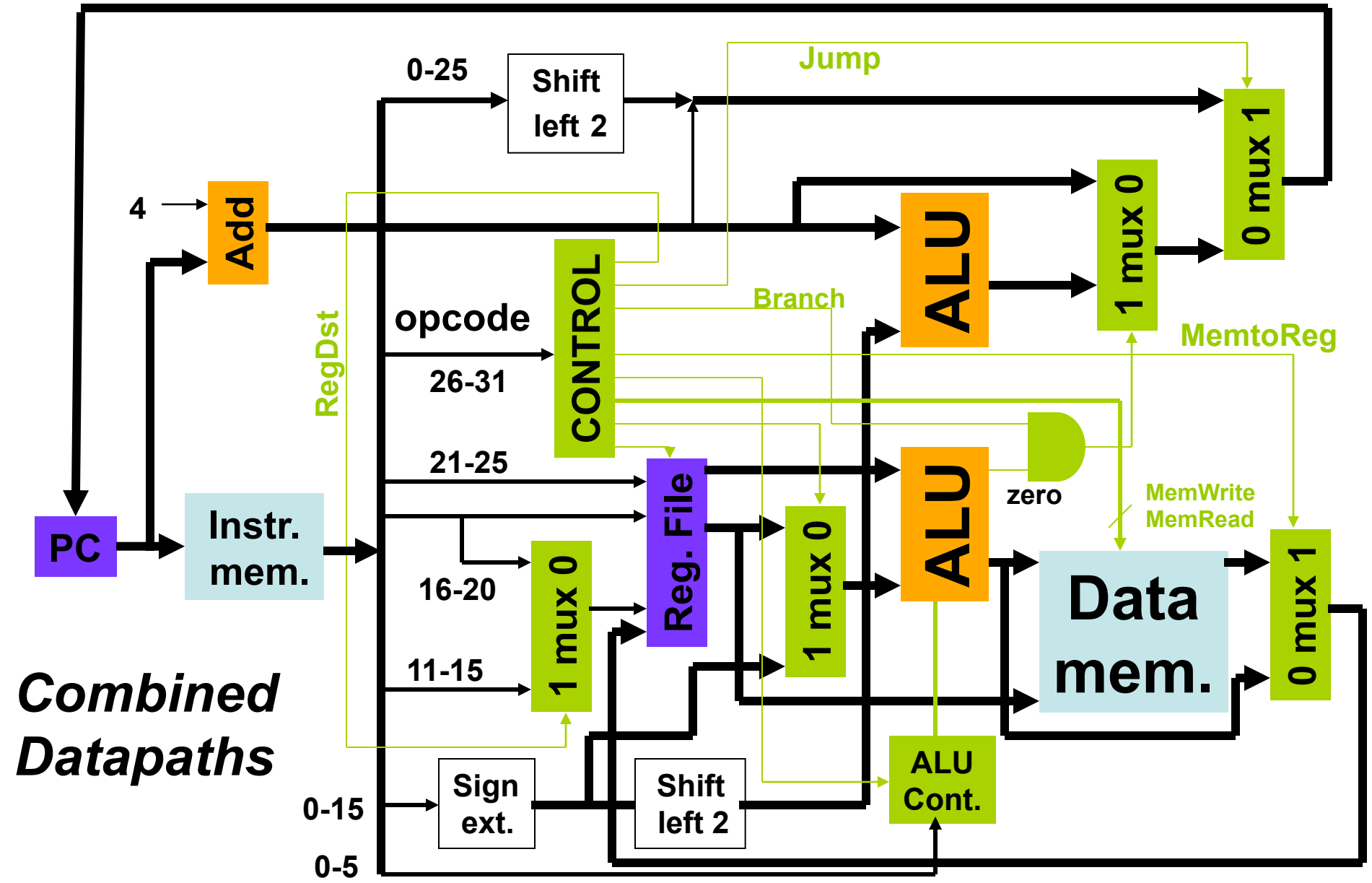
# RISC Design:
## Pipelining

Virendra Singh
Associate Professor
Computer Architecture and Dependable Systems Lab
Department of Electrical Engineering
Indian Institute of Technology Bombay
http://www.ee.iitb.ac.in/~viren/
E-mail: viren@ee.iitb.ac.in

CP-226: Computer Architecture

Lecture 10 (20 Feb 2013)

CADSL


Combined Datapaths
0-25
Shift left 2
Jump
4
Add
ALU
1 mux 0
0 mux 1
RegDst
opcode
26-31
CONTROL
Branch
MemtoReg
21-25
Reg. File
zero
MemWrite
MemRead
PC
Instr. mem.
16-20
1 mux 0
ALU
Data mem.
0 mux 1
11-15
1 mux 0
ALU Cont.
Sign ext.
Shift left 2
0-15
0-5

# Pipelining in a Computer

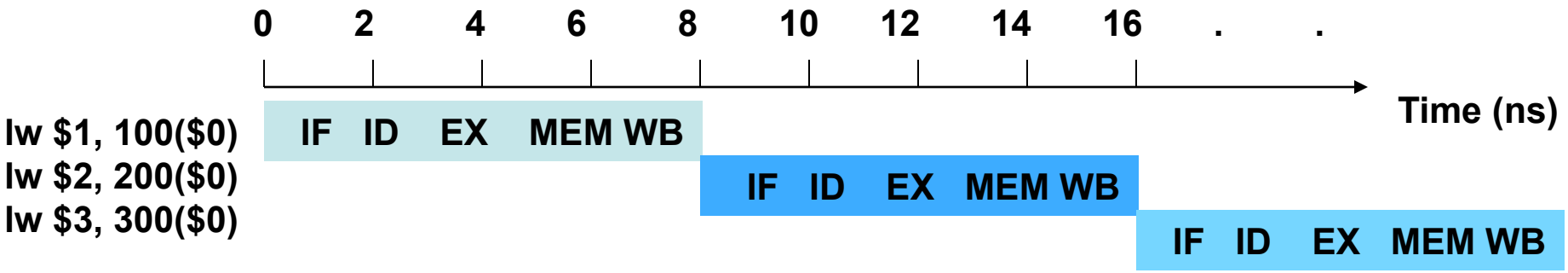
- Divide datapath into nearly equal tasks, to be performed serially and requiring non-overlapping resources.
- Insert registers at task boundaries in the datapath; registers pass the output data from one task as input data to the next task.
- Synchronize tasks with a clock having a cycle time that just exceeds the time required by the longest task.
- Break each instruction down into a fixed number of tasks so that instructions can be executed in a staggered fashion.

# Single-Cycle Datapath

| Instruction class | Instr. fetch (IF) | Instr. Decode (also reg. file read) (ID) | Execution (ALU Operation) (EX) | Data access (MEM) | Write Back (Reg. file write) (WB) | Total time |
|---|---|---|---|---|---|---|
| lw | 2ns | 1ns | 2ns | 2ns | 1ns | 8ns |
| sw | 2ns | 1ns | 2ns | 2ns | | 8ns |
| R-format add, sub, and, or, slt | 2ns | 1ns | 2ns | | 1ns | 8ns |
| B-format, beq | 2ns | 1ns | 2ns | | | 8ns |

**No operation on data; idle time equalizes instruction length to a fixed clock period.**

# Execution Time: Single-Cycle




Clock cycle time = 8 ns

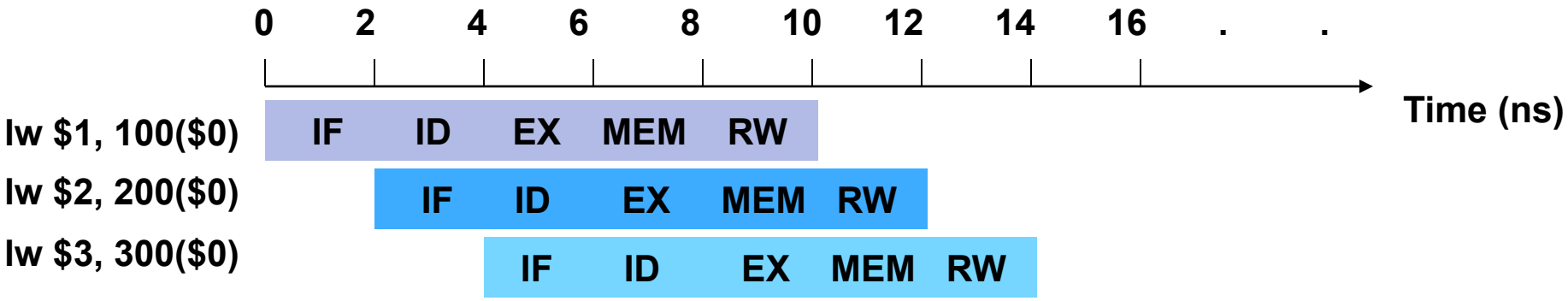
Total time for executing three lw instructions = 24 ns

# Pipelined Datapath

| Instruction class | Instr. fetch (IF) | Instr. Decode (also reg. file read) (ID) | Execu-tion (ALU Opera-tion) (EX) | Data access (MEM) | Write Back (Reg. file write) (WB) | Total time |
|---|---|---|---|---|---|---|
| lw | 2ns | ~~1ns~~ 2ns | 2ns | 2ns | ~~1ns~~ 2ns | 10ns |
| sw | 2ns | ~~1ns~~ 2ns | 2ns | 2ns | ~~1ns~~ 2ns | 10ns |
| R-format: add, sub, and, or, slt | 2ns | ~~1ns~~ 2ns | 2ns | 2ns | ~~1ns~~ 2ns | 10ns |
| B-format: beq | 2ns | ~~1ns~~ 2ns | 2ns | 2ns | ~~1ns~~ 2ns | 10ns |

**No operation on data; idle time inserted to equalize instruction lengths.**

# Execution Time: Pipeline




Clock cycle time = 2 ns, *four times faster than single-cycle clock*

Total time for executing three lw instructions = 14 ns

$$\text{Performance ratio} = \frac{\text{Single-cycle time}}{\text{Pipeline time}} = \frac{24}{14} = 1.7$$

# Pipeline Performance

Clock cycle time = 2 ns

1,003 *lw instructions:*

Total time for executing 1,003 lw instructions = 2,014 ns

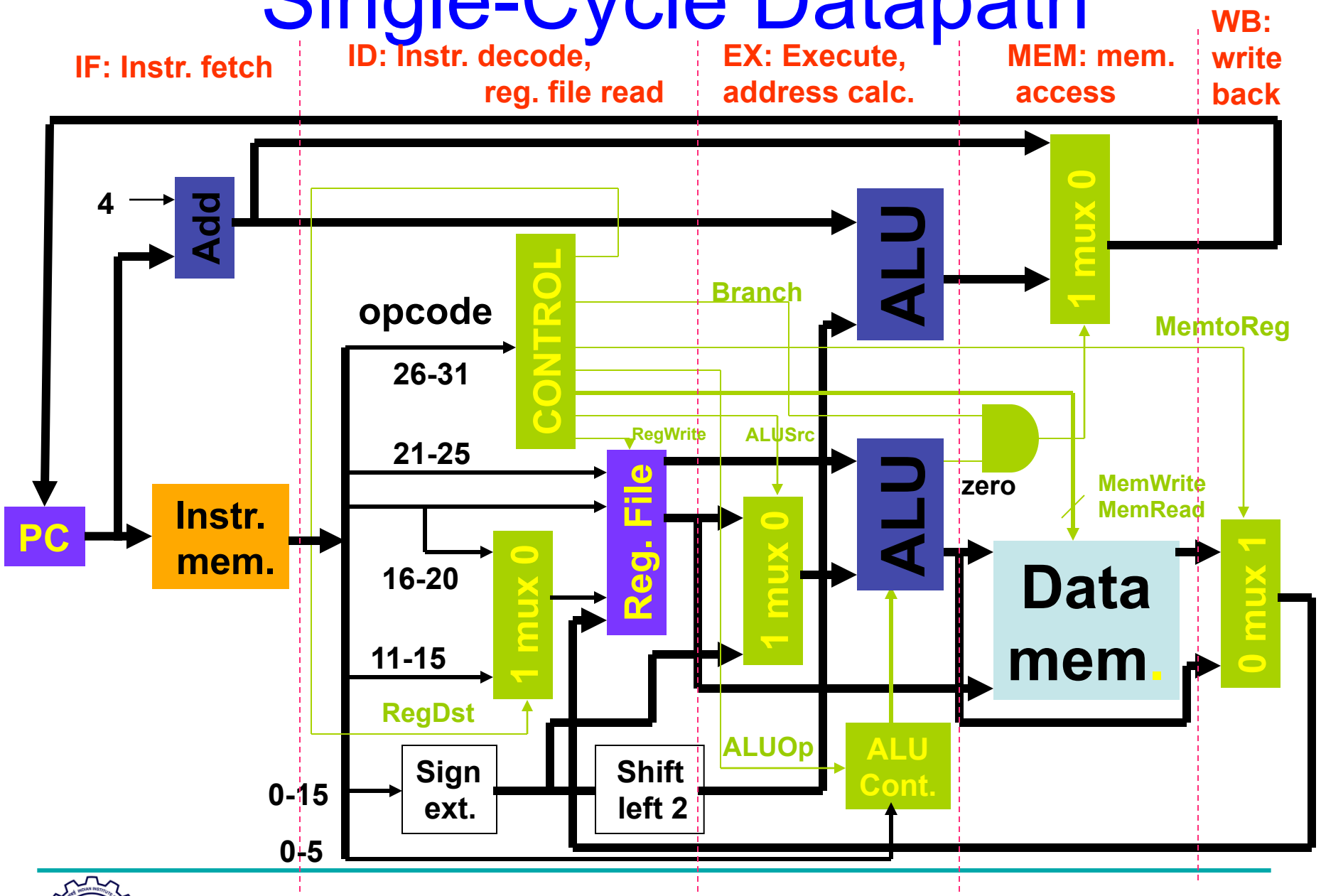
$$\text{Performance ratio} = \frac{\text{Single-cycle time}}{\text{Pipeline time}} = \frac{8{,}024}{2{,}014} = 3.98$$

10,003 *lw instructions:*

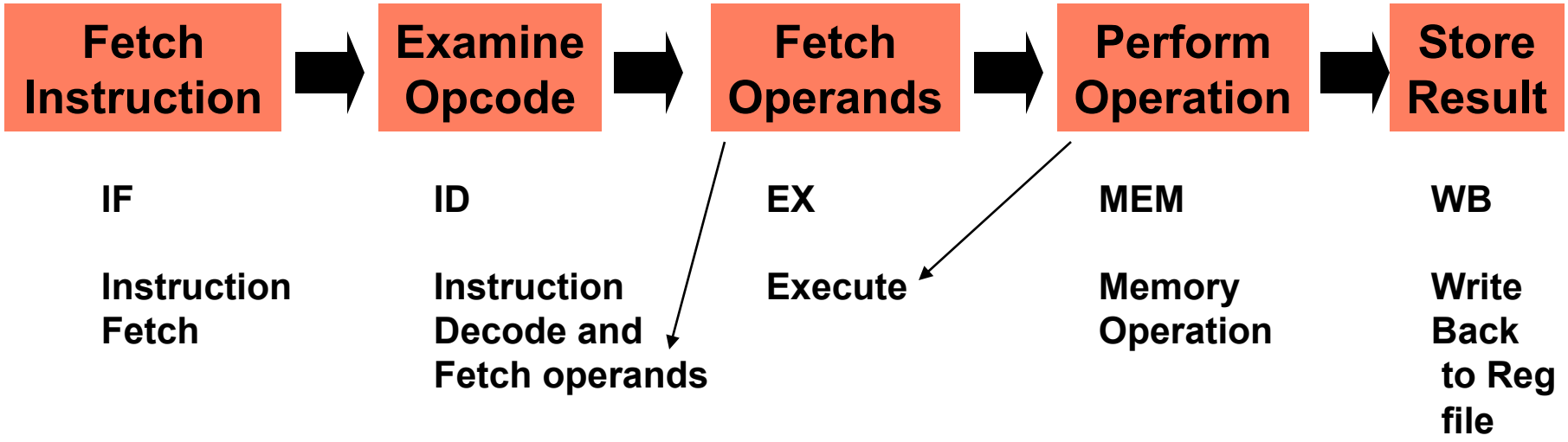
Performance ratio = 80,024 / 20,014 = 3.998 → Clock cycle ratio (4)

Pipeline performance approaches clock-cycle ratio for long programs.

# Single-Cycle Datapath

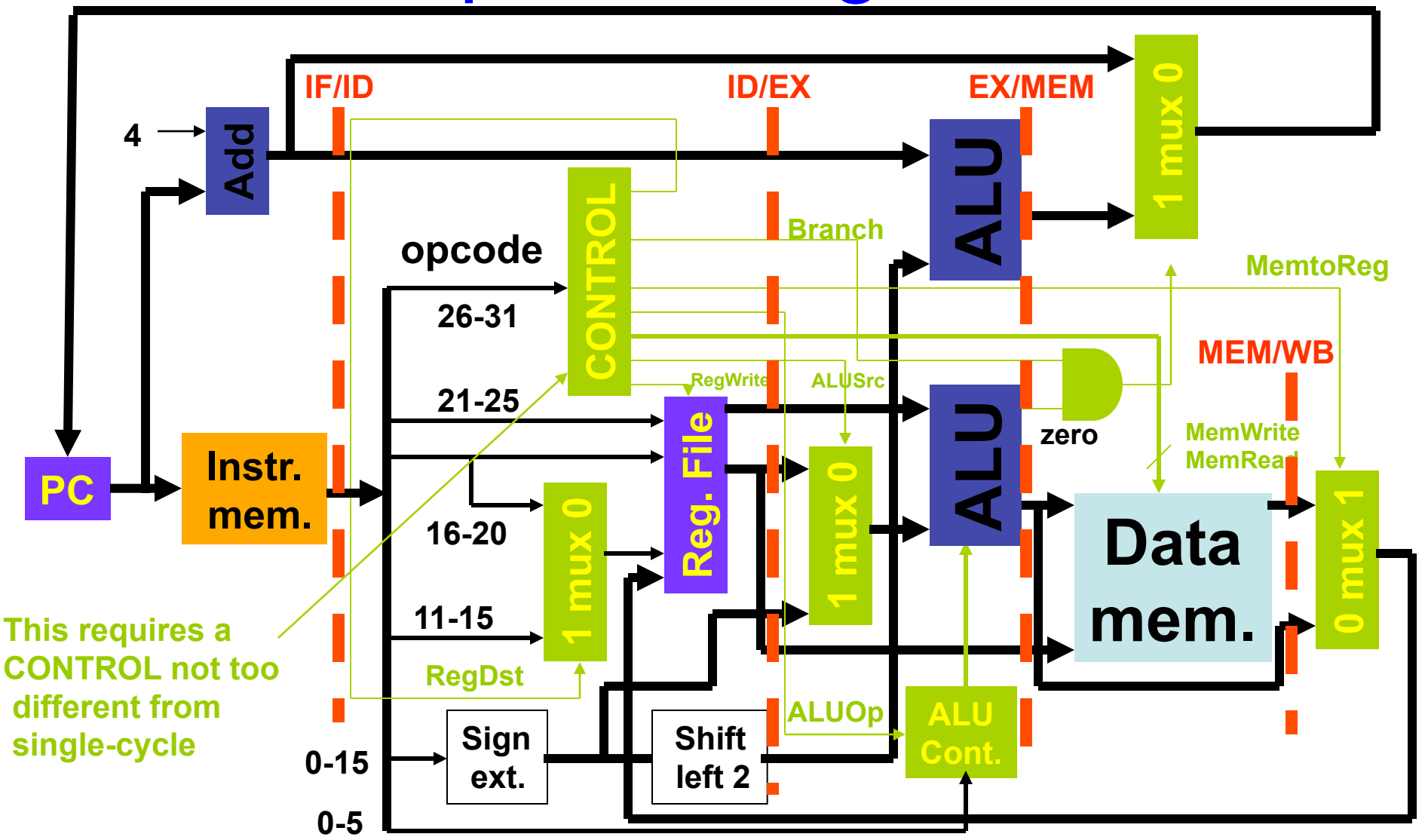


# Pipelining of RISC Instructions




***Although an instruction takes five clock cycles, one instruction is completed every cycle.***

# Pipeline Registers

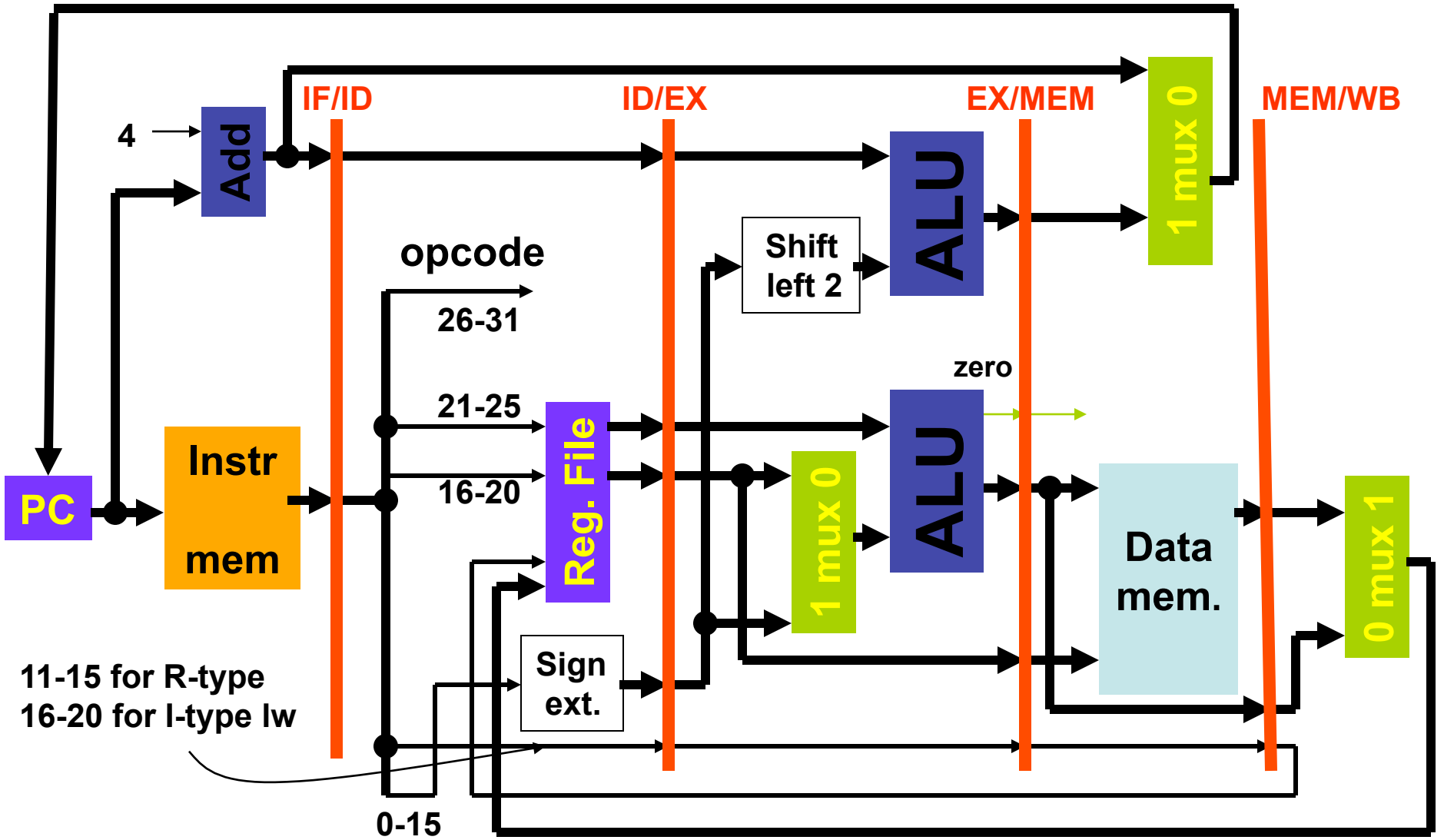


# Pipeline Register Functions

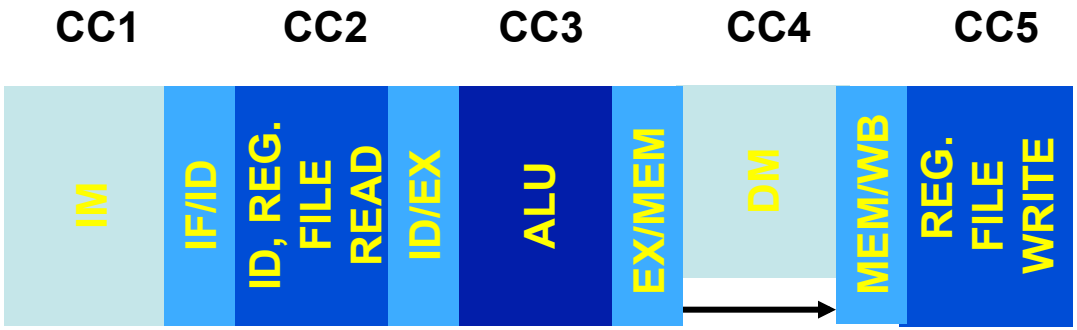
- Four pipeline registers are added:

| Register name | Data held |
|---|---|
| IF/ID | PC+4, Instruction word (IW) |
| ID/EX | PC+4, R1, R2, IW(0-15) sign ext., IW(11-15) |
| EX/MEM | PC+4, zero, ALUResult, R2, IW(11-15) or IW(16-20) |
| MEM/WB | M[ALUResult], ALUResult, IW(11-15) or IW(16-20) |

# Pipelined Datapath



# Five-Cycle Pipeline



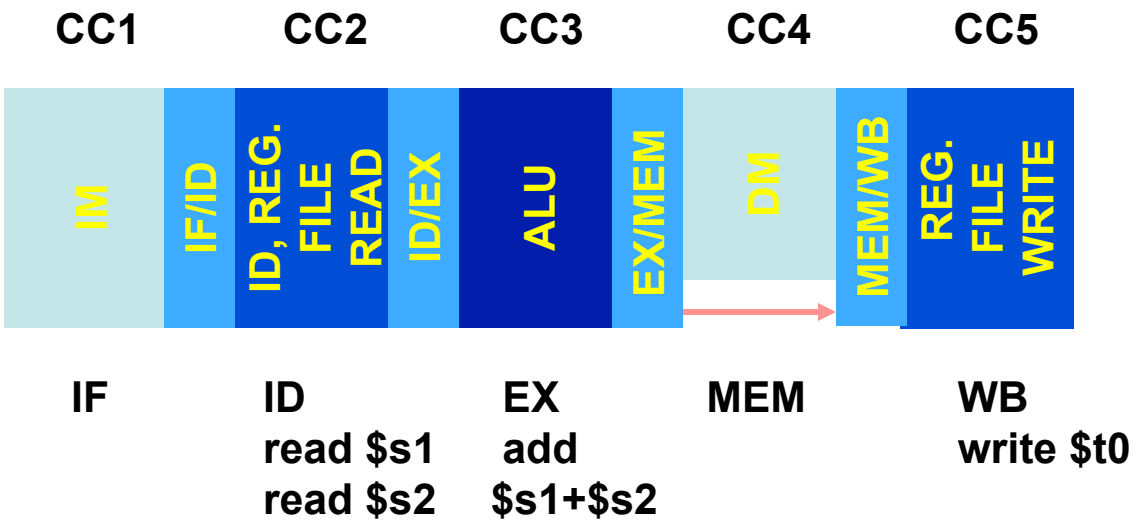
# Add Instruction

- add $t0, $s1, $s2

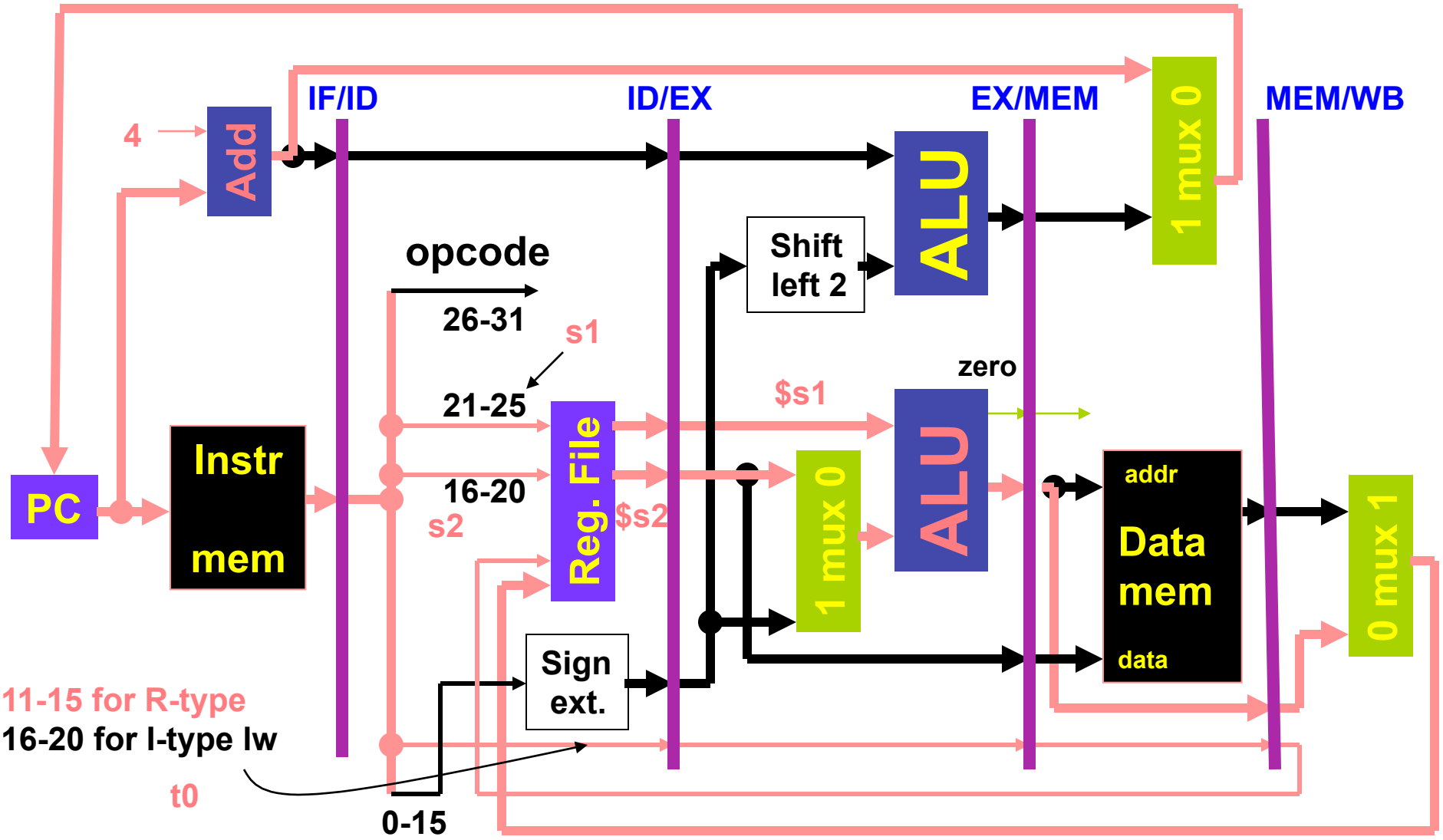
  Machine instruction word

  000000 10001 10010 01000 00000 100000

  opcode $s1 $s2 $t0 function



# Pipelined Datapath Executing add

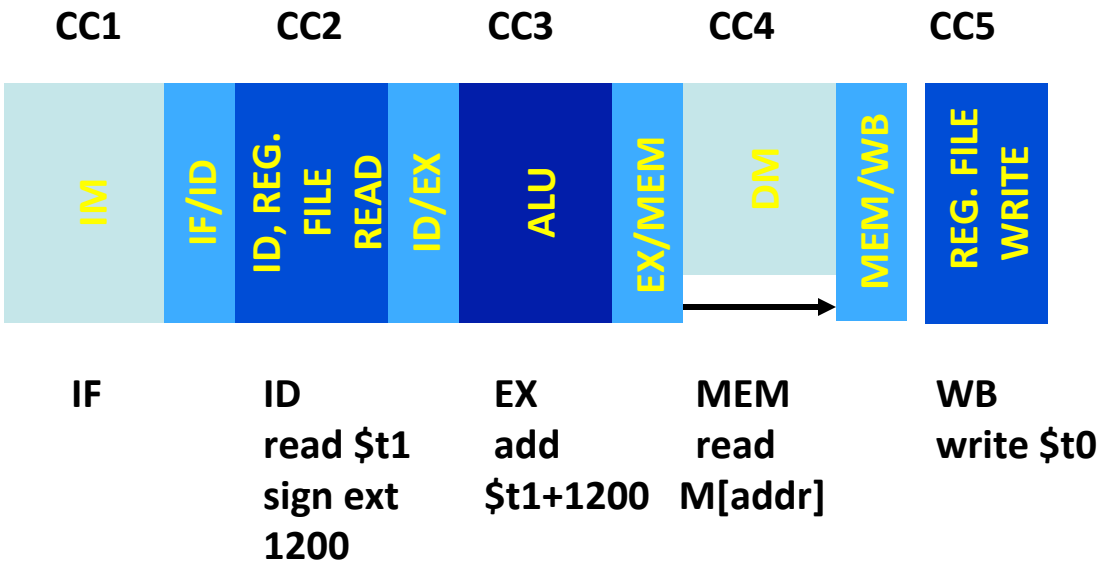


# Load Instruction

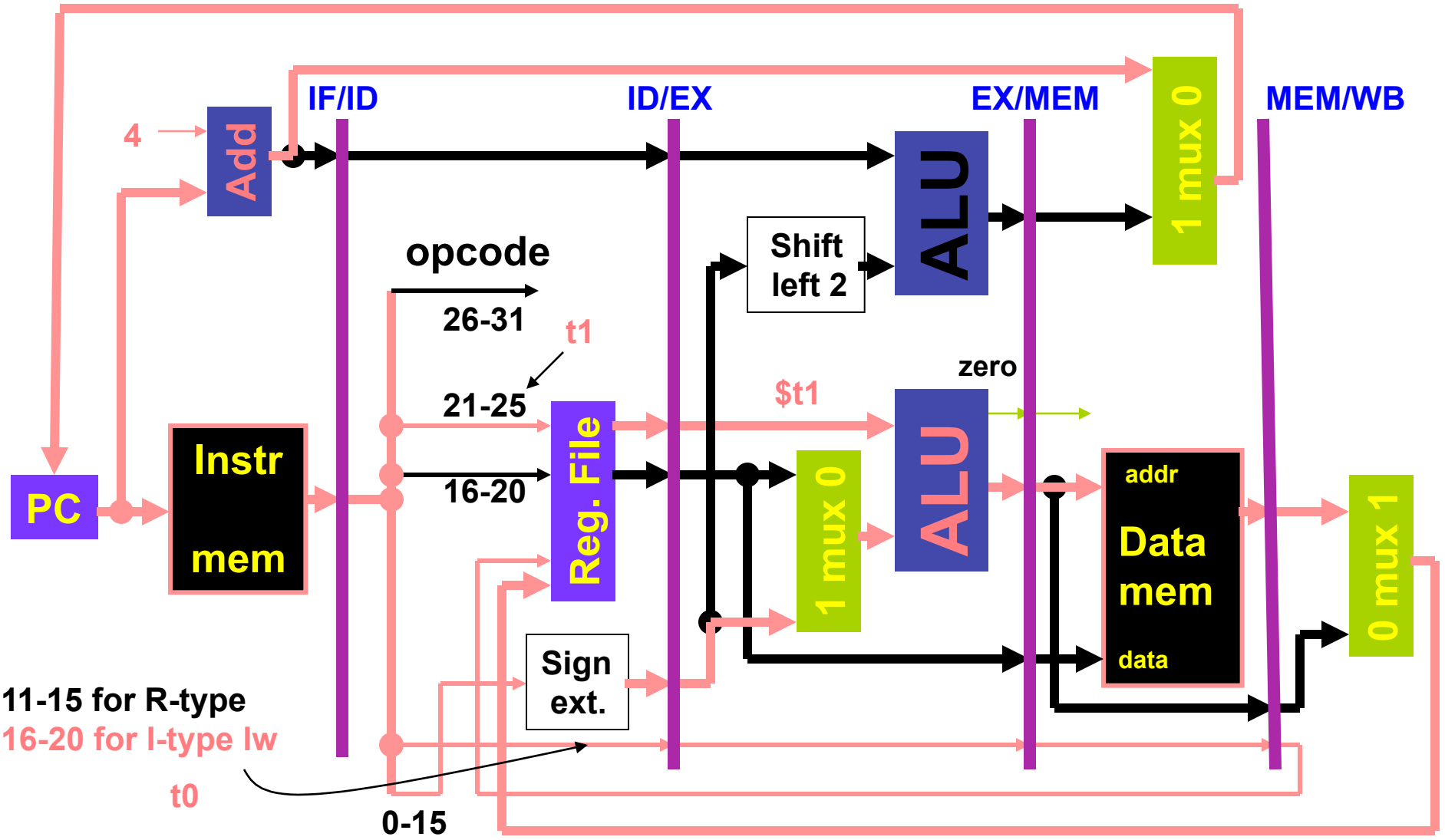
- lw $t0, 1200 ($t1)

100011 01001 01000 0000 0100 1000 0000

opcode $t1 $t0 1200



# Pipelined Datapath Executing lw

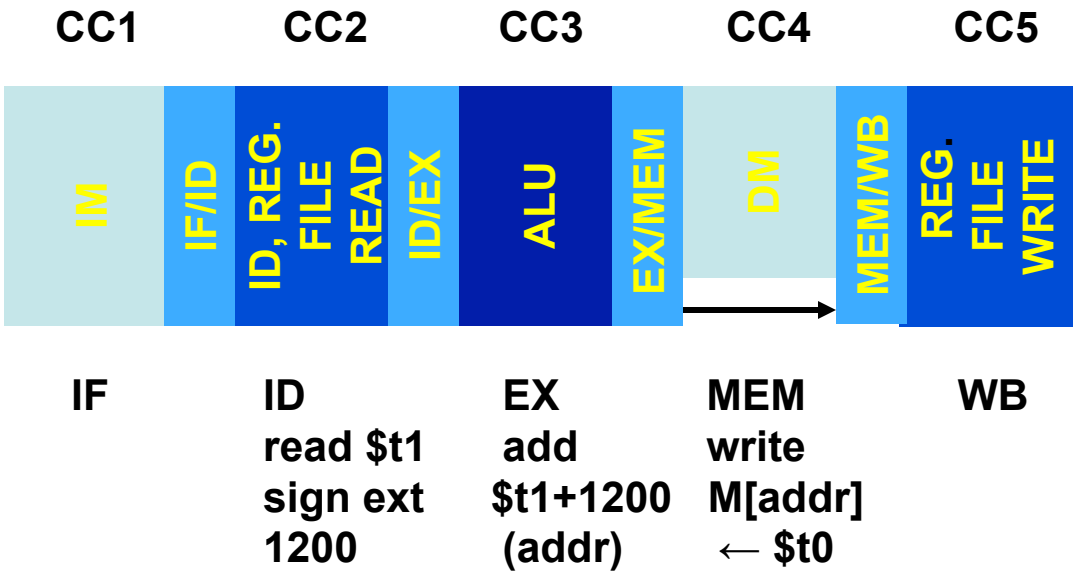


# Store Instruction

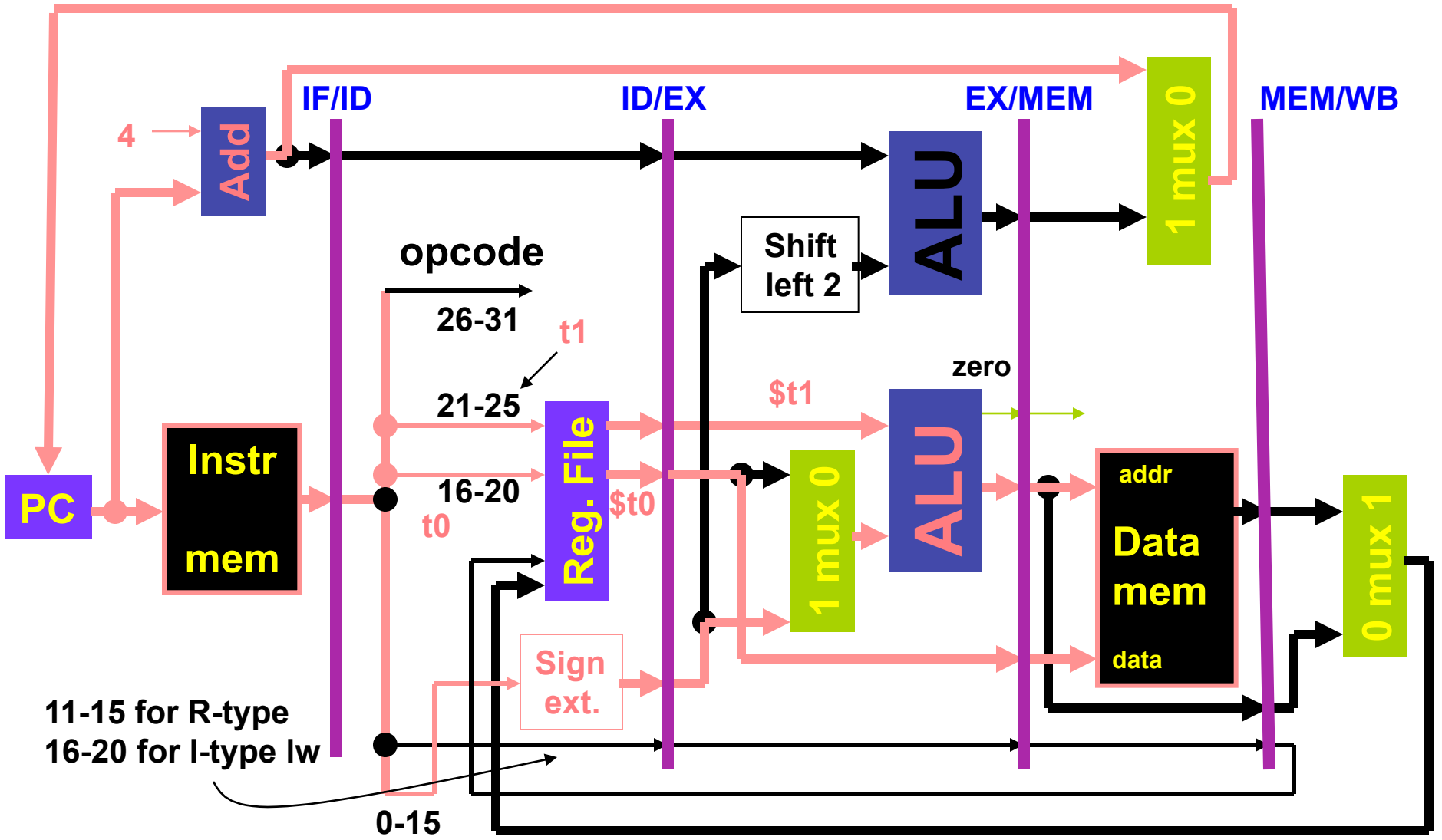
- sw $t0, 1200 ($t1)

101011 01001 01000 0000 0100 1000 0000

opcode $t1 $t0 1200



# Pipelined Datapath Executing sw

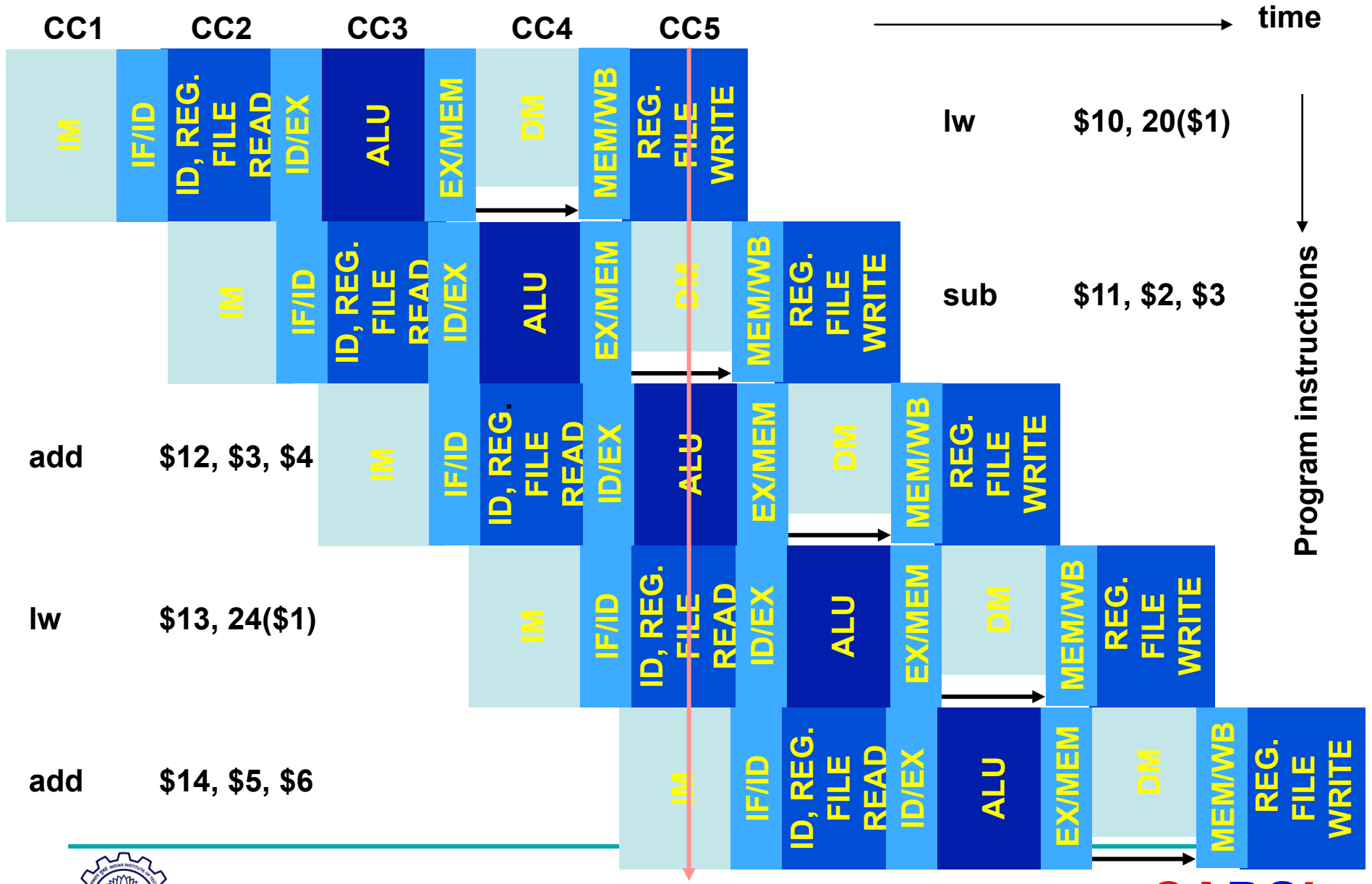


# Executing a Program

*Consider a five-instruction segment:*

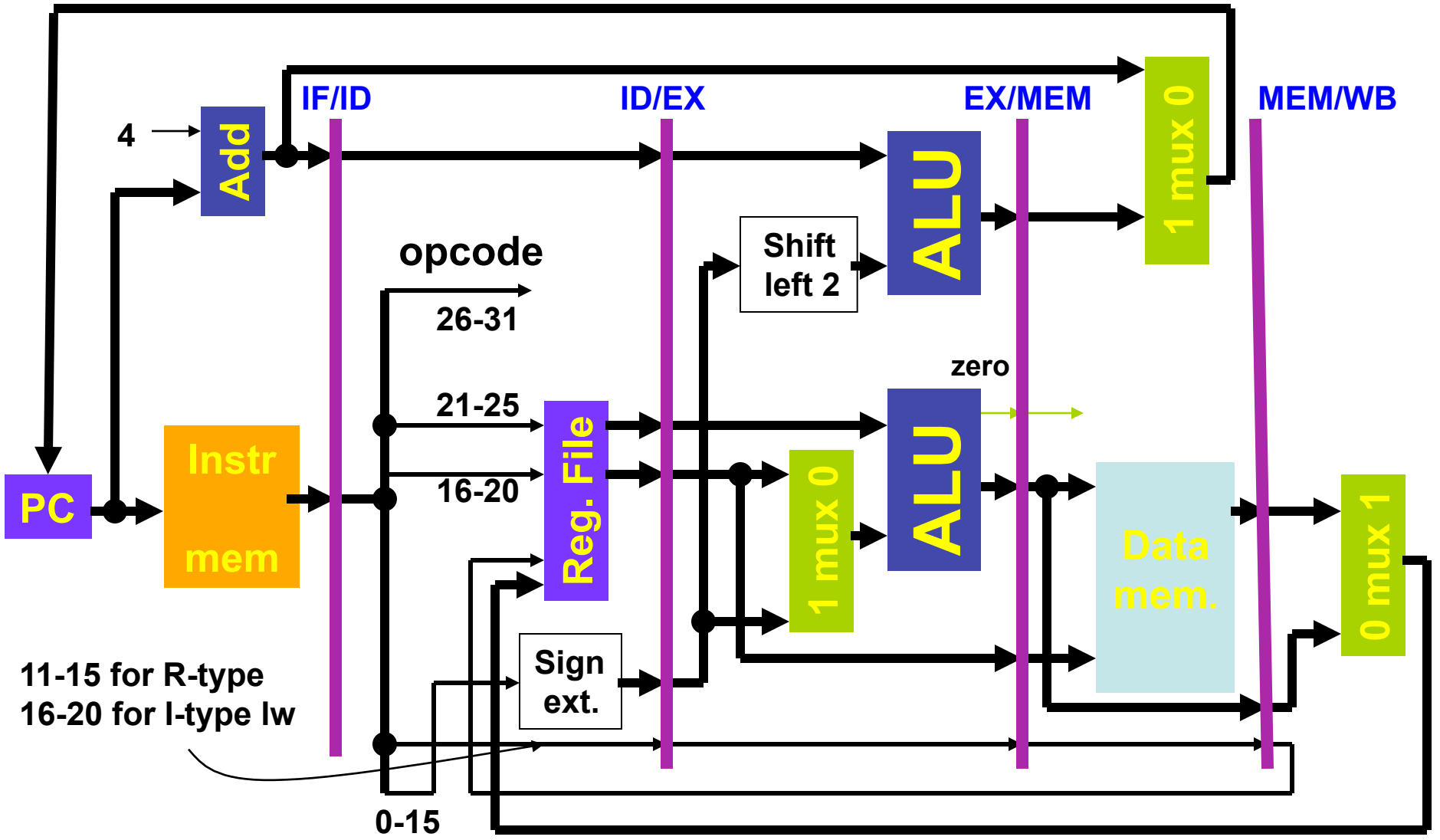
```
lw    $10, 20($1)
sub   $11, $2, $3
add   $12, $3, $4
lw    $13, 24($1)
add   $14, $5, $6
```

# Program Execution



# CC5

IF: add $14, $5, $6 | ID: lw $13, 24($1) | EX: add $12, $3, $4 | MEM: sub $11, $2, $3 | WB: lw $10, 20($1)



# Advantages of Pipeline

- After the fifth cycle (CC5), one instruction is completed each cycle; CPI ≈ 1, neglecting the initial pipeline latency of 5 cycles.
  - *Pipeline latency is defined as the number of stages in the pipeline, or*
  - *The number of clock cycles after which the first instruction is completed.*
- The clock cycle time is about four times shorter than that of single-cycle datapath and about the same as that of multicycle datapath.
- For multicycle datapath, CPI = 3. ….
- So, pipelined execution is faster, but . . .

# Thank You