Enhancing Performance by Pipelining

COE608: Computer Organization and Architecture

Dr. Gul N. Khan
http://www.ee.ryerson.ca/~gnkhan
Electrical and Computer Engineering
Ryerson University

Overview

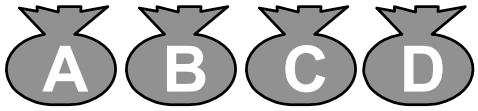
- Introduction to Pipelining
  - Laundry Example
- Pipelining and CPU Performance
  - Pipelining Hazards
  - Hazard Solutions
- Pipelined Datapath and Control
  - Stalling, Forwarding and Flushing

Chapter 4 (Sections 4.5, 4.6, 4.7, 4.8 and part of 4.10/4.11) of Text

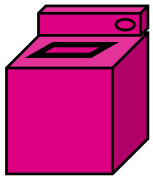
Introduction

Laundry Example

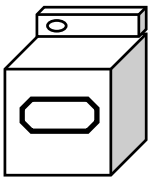
Consider Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold



Washer takes 30 minutes



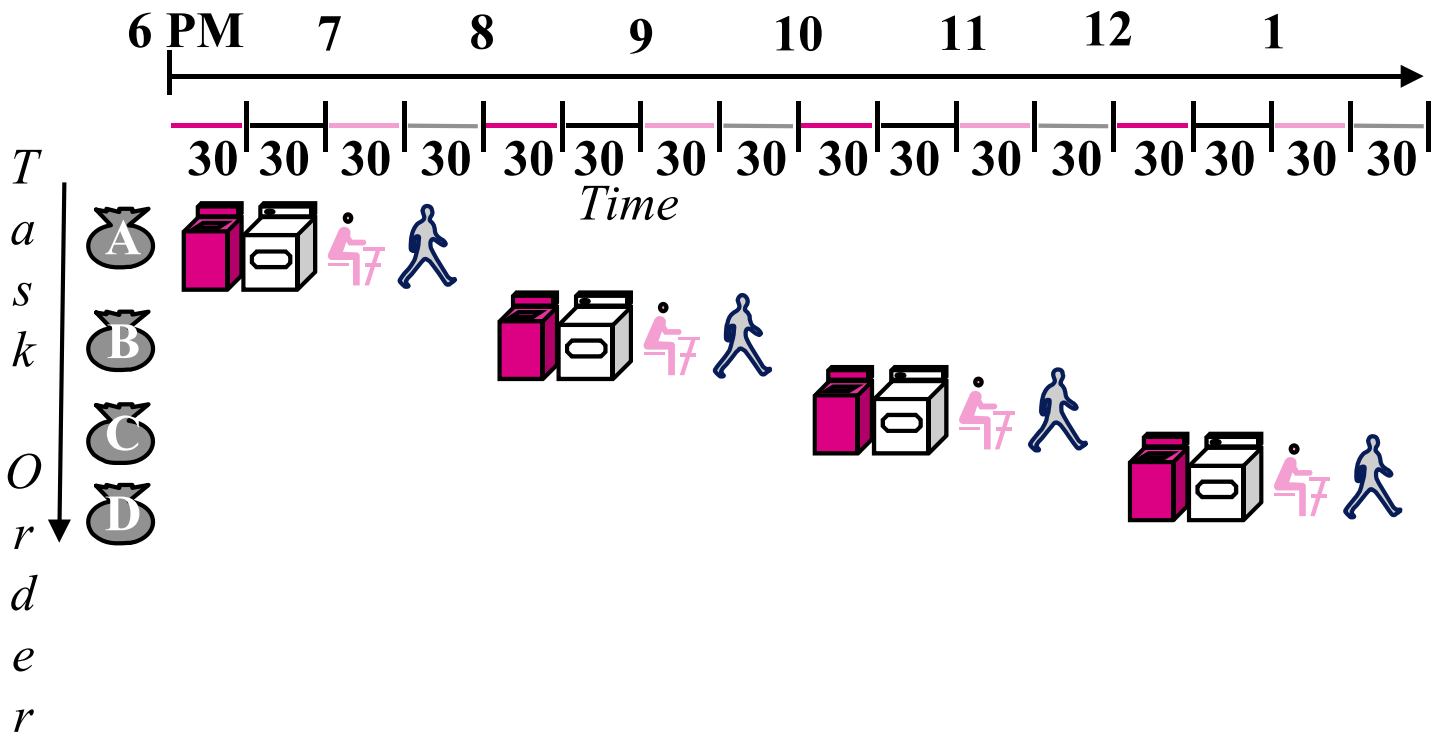
Dryer takes 30 minutes



"Folder" takes 30 minutes

"Stasher" takes 30 minutes to put clothes into drawers

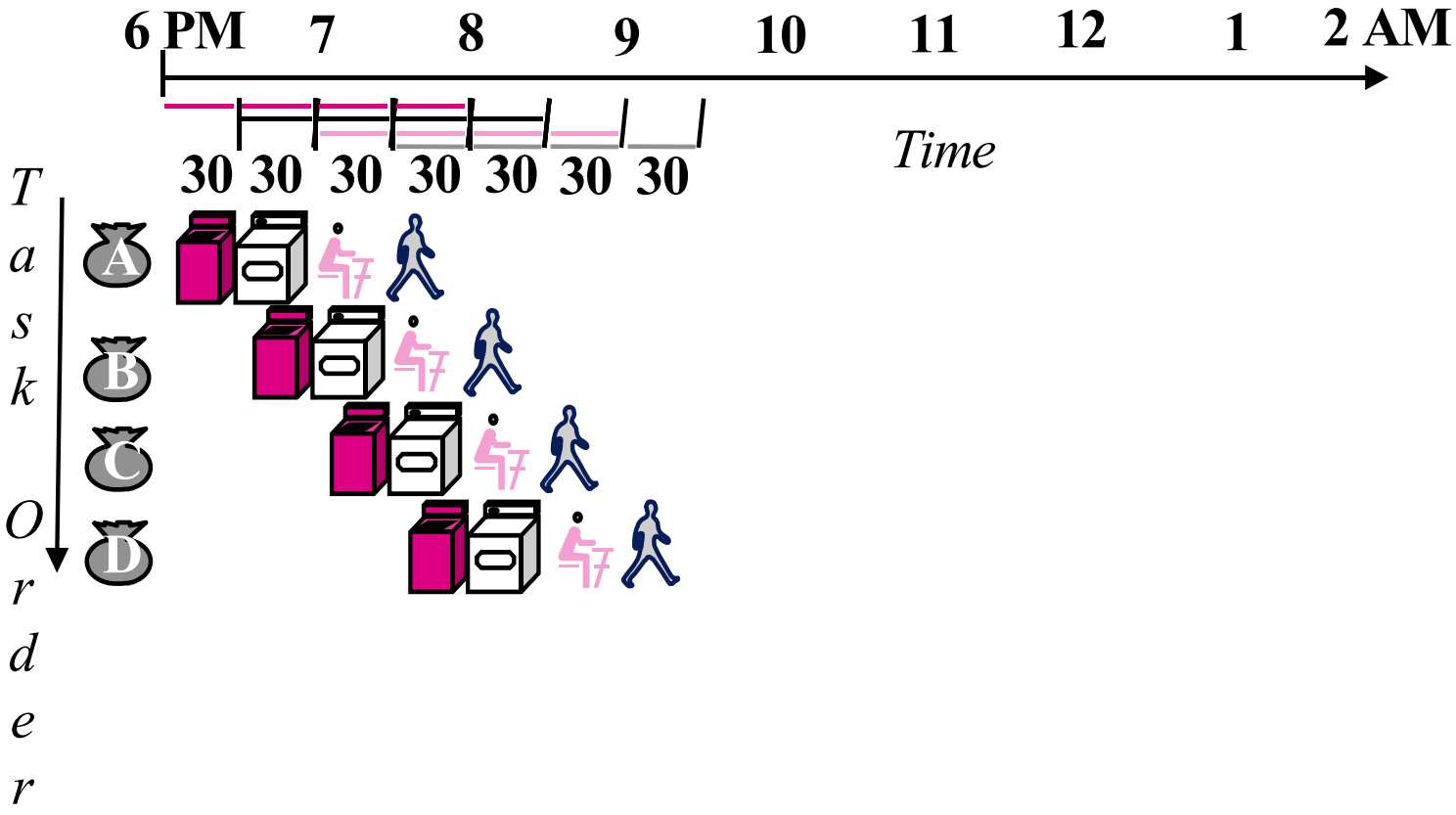
Sequential Laundry



Sequential laundry takes 8 hours for 4 loads

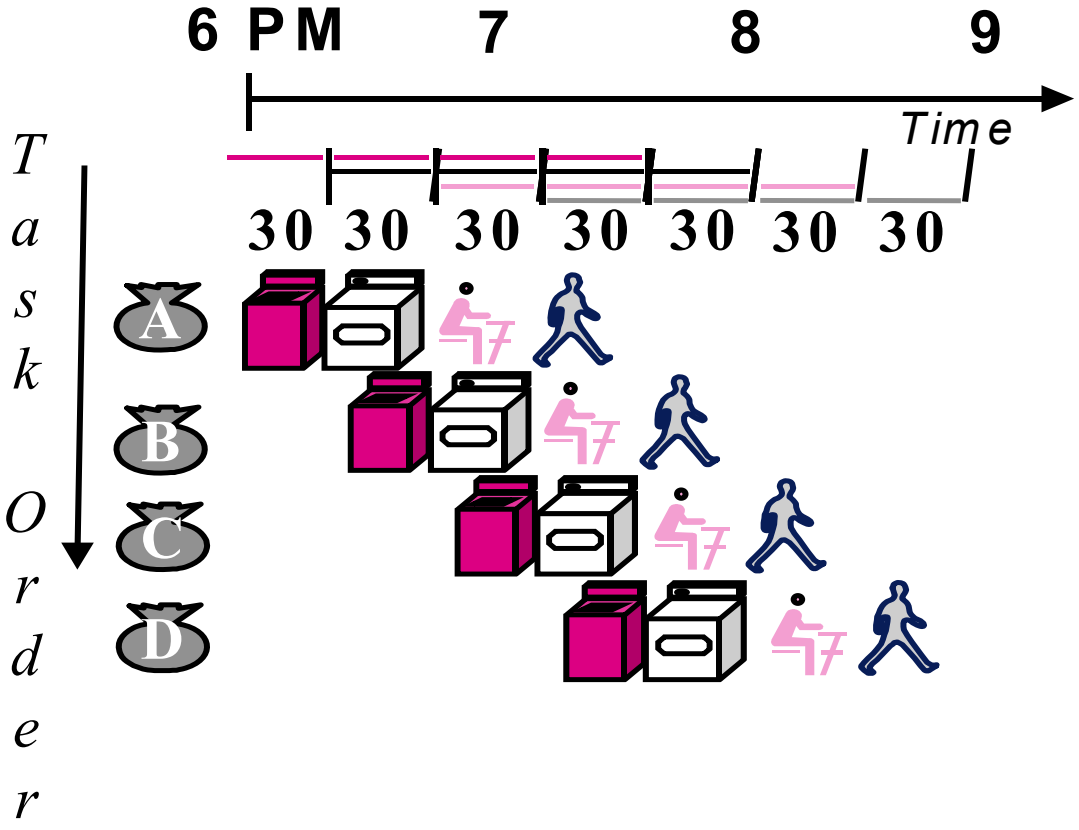
Pipelined Laundry

Start work ASAP



- **Multiple tasks operating simultaneously using different resources.**

Pipelined Laundry

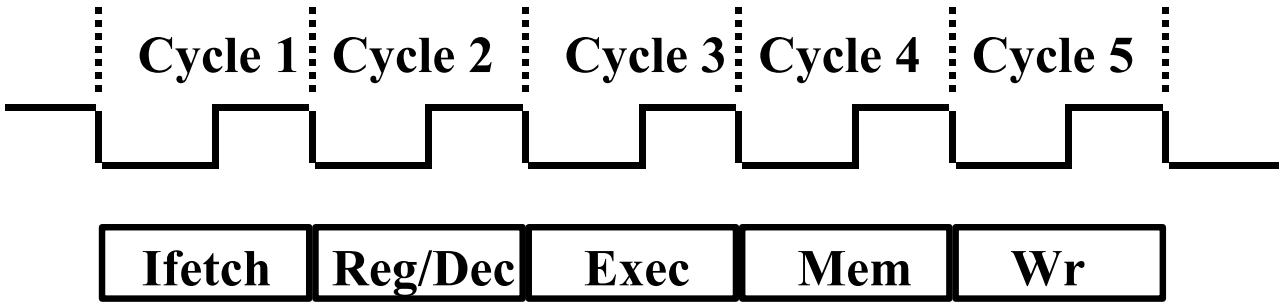


- Pipelining doesn't reduce latency of a single task. It improves throughput of the entire workload
- Potential speedup = Number of (pipe) stages
- Pipeline rate limited by the slowest pipeline stage
- Unbalanced lengths of pipe stages reduce the speedup
- Time to "fill" pipeline and time to "drain" it reduces the speedup.
- Stall due to dependencies??

Pipelining in the CPU

Improve performance by increasing instruction throughput.

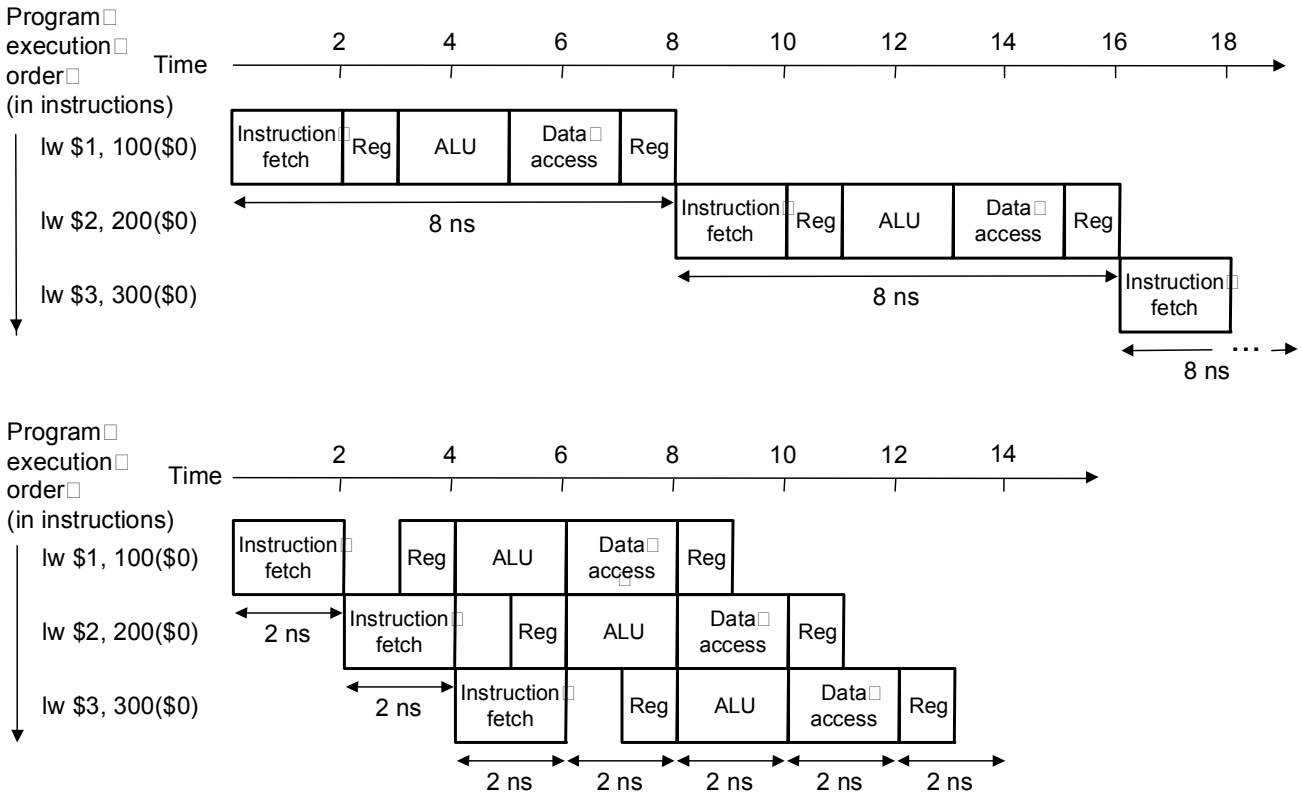
Five Natural Stages in an Instruction Execution



Load Word Instruction Stages

- **Ifetch:** Instruction Fetch
  - Fetch instruction from Instruction Memory
- **Reg/Dec:** Registers Fetch and Instruction Decode
- **Exec:** Calculate the memory address
- **Mem:** Read data from the Data Memory
- **Wr:** Write data back to the register file

CPU Pipelining



Ideal speedup is number of stages in the pipeline.

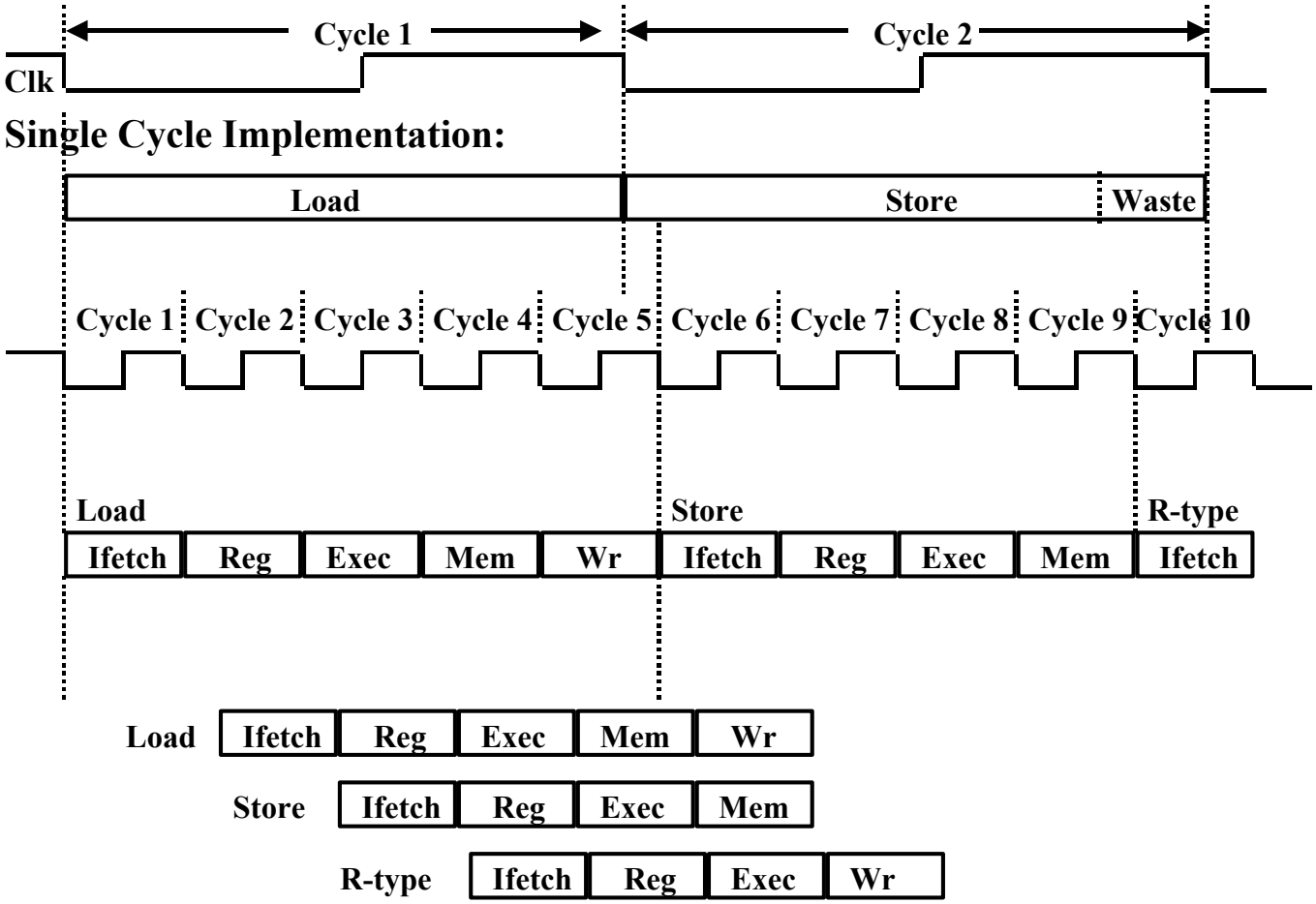
What makes pipelining easy?

- When all instructions are of the same length.
- Few instruction formats.
- Memory operands appear only in loads and stores.

What makes pipelining hard?

- **Structural Hazards:**
- **Control Hazards:**
- **Data Hazards:** An instruction depends on a previous instruction.

Single Cycle, Multiple Cycle vs. Pipeline



Suppose we execute 100 instructions

- **Single Cycle Machine**

  45 ns/cycle x 1 CPI x 100 inst = 4500 ns

- **Multicycle Machine**

  10 ns/cycle x 4.6 CPI (mix of inst) x 100 inst = 4600 ns

- **Ideal Pipelined Machine**

  10 ns/cycle x (1 CPI x 100 inst + 4 cycle drain) = 1040 ns

Pipeline Hazards

- Structural Hazards: attempt to use the same resource two different ways at the same time

- Data Hazards: attempt to use an item before it is ready

  Instruction depends on the results of a prior instruction still in the pipeline

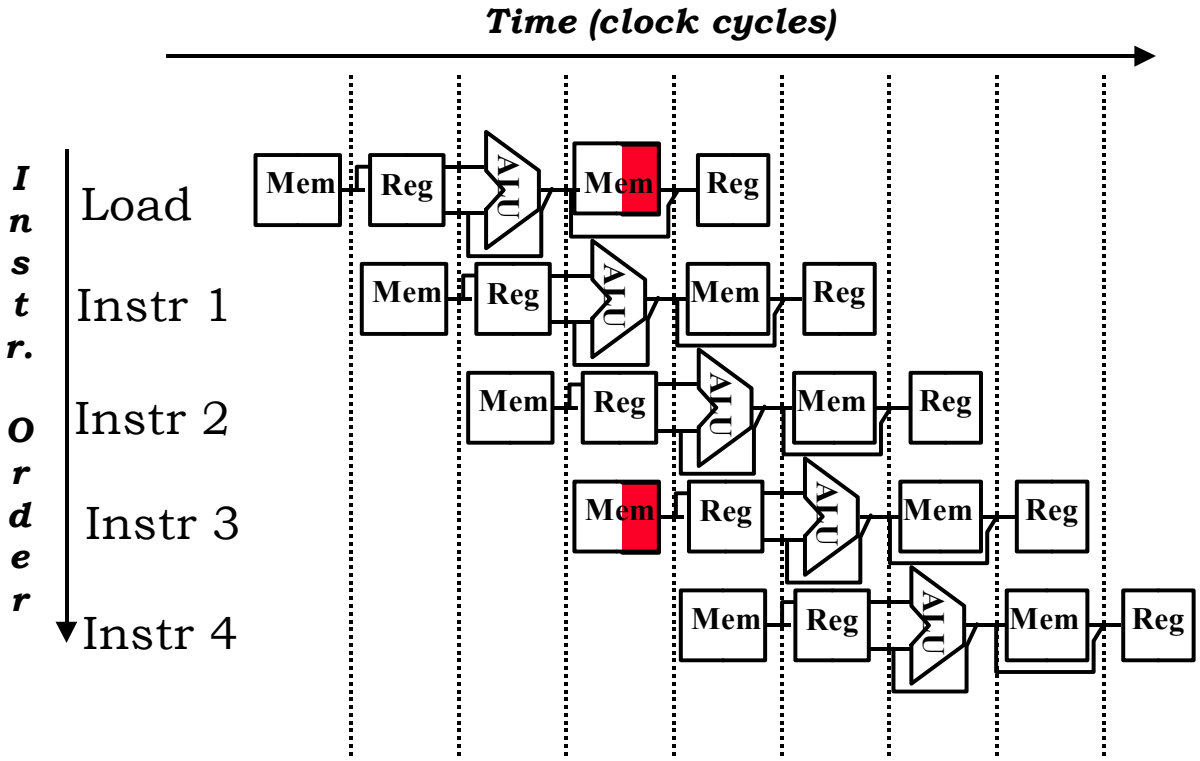
- Control Hazards: attempt to make a decision before condition is evaluated

  Branch instructions

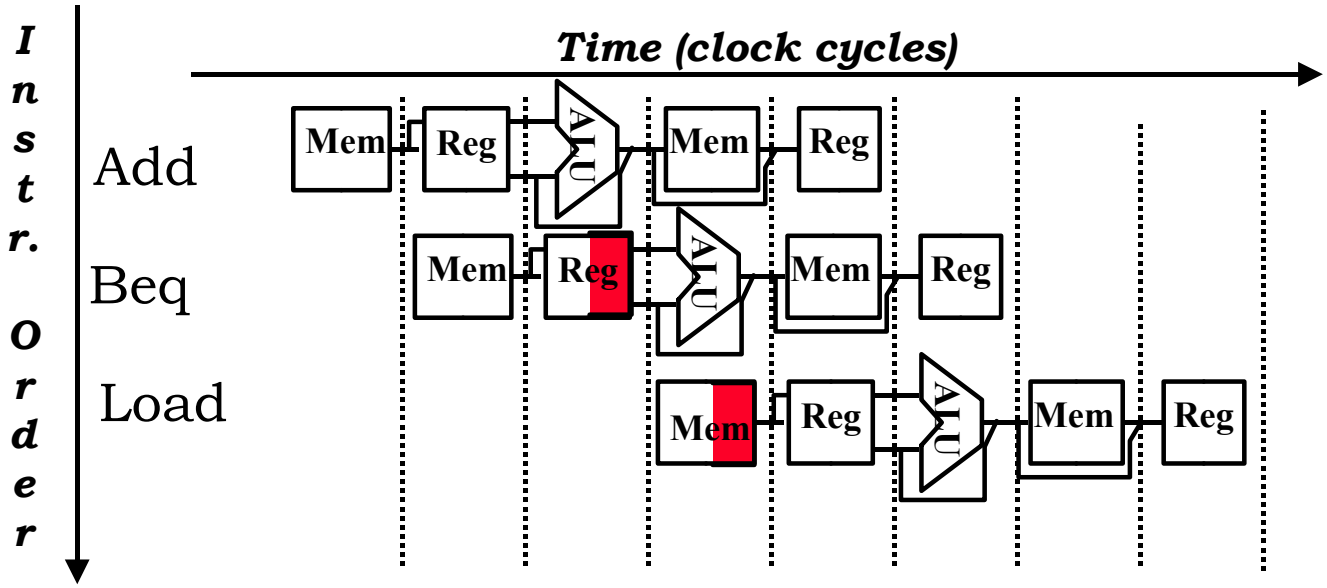
The hazards can always be resolved by waiting. Pipeline control must detect the hazards and take actions (or delay action) to resolve hazards.

Hazards

Structural Hazard: Single Memory



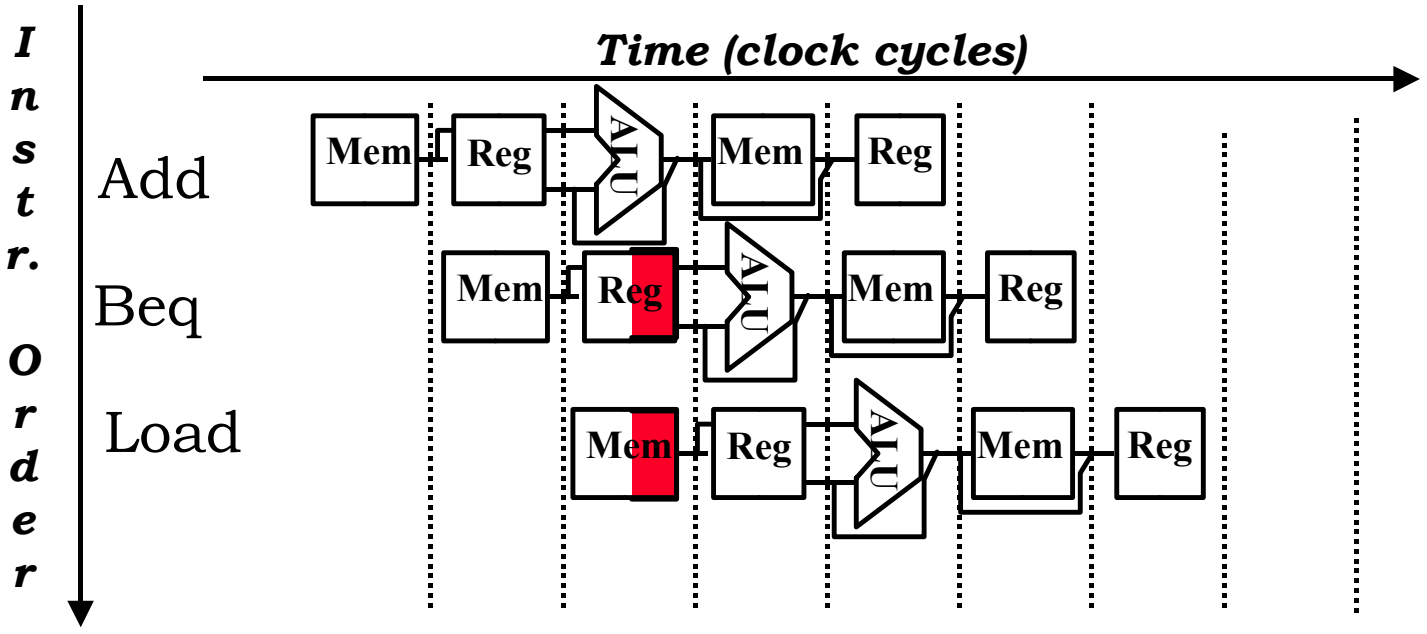
Control Hazard



Control Hazard Solutions

Stall

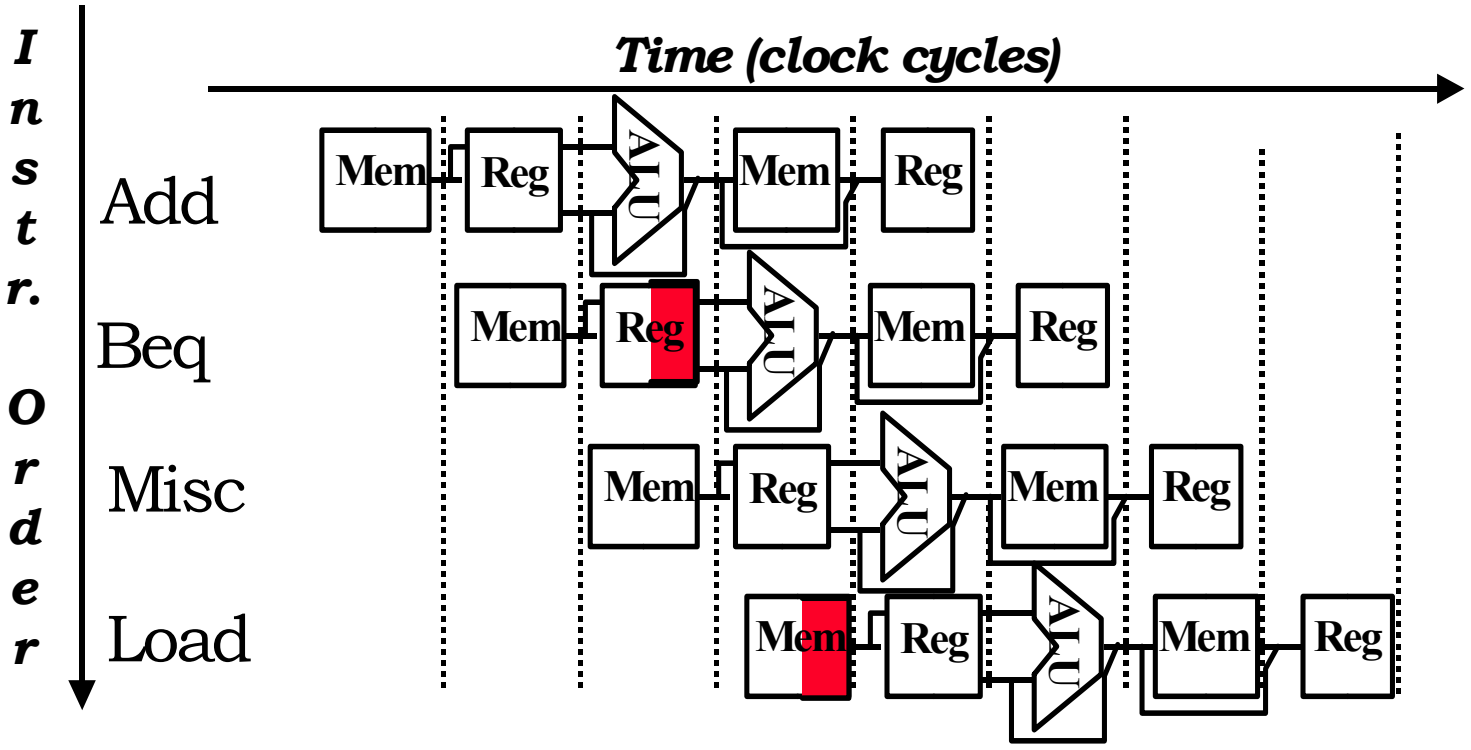
It is possible to move the decision upward to 2nd stage by adding hardware that checks the registers as being read.



- Predict: guess one branch direction then backup if wrong.

Control Hazard Solutions

Redefine branch behavior (takes place after next instruction) “delayed branch”

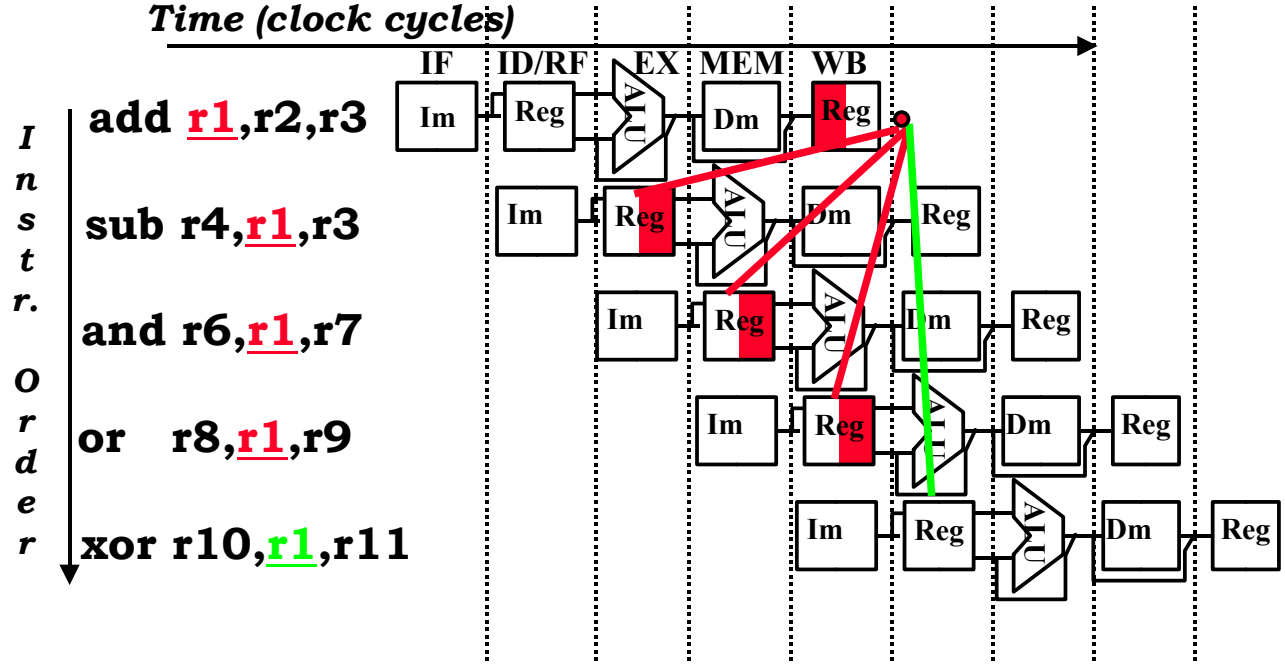


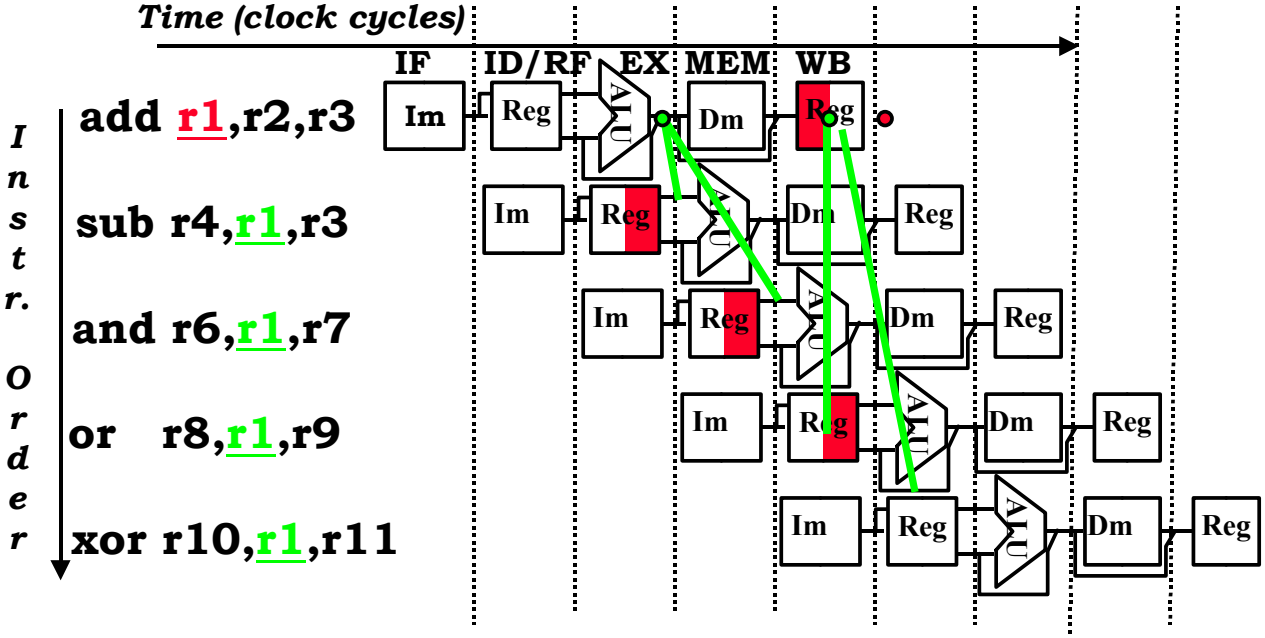
- Impact:
  No clock cycle is wasted if one can find an instruction to put in “slot”

Data Hazard

Data Hazard in r1

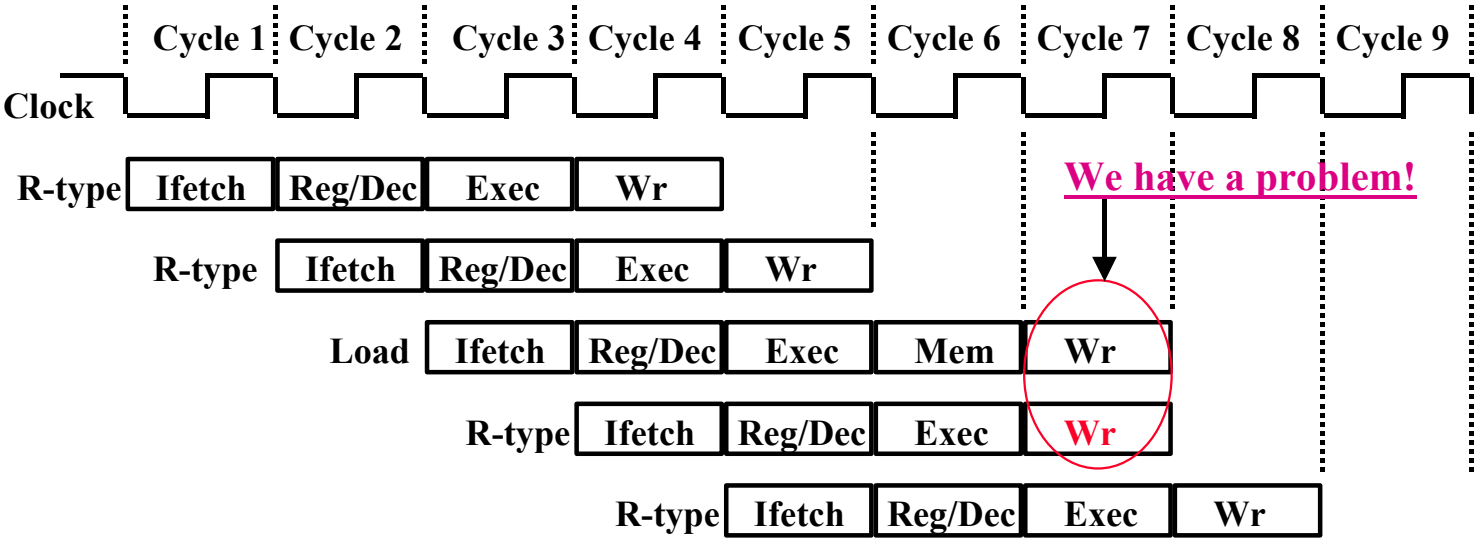
Dependencies backwards in time are hazards





"Forward" result from one stage to another.

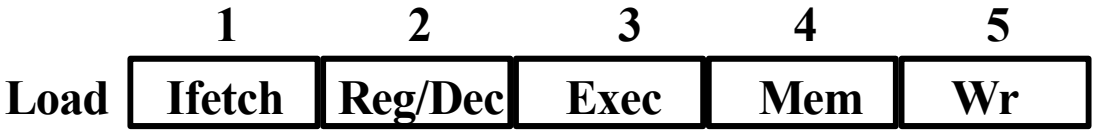
Pipelining R-type and Load



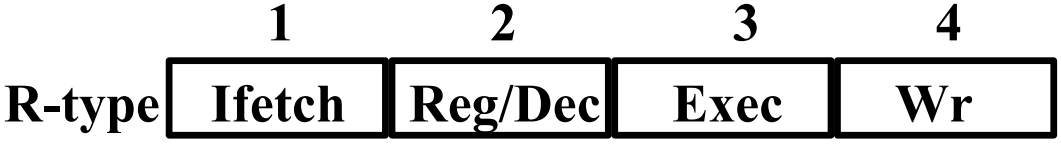
Structural hazard:

- Two instructions try to write to the register file at the same time.

Load uses Register File's Write Port in 5th stage.



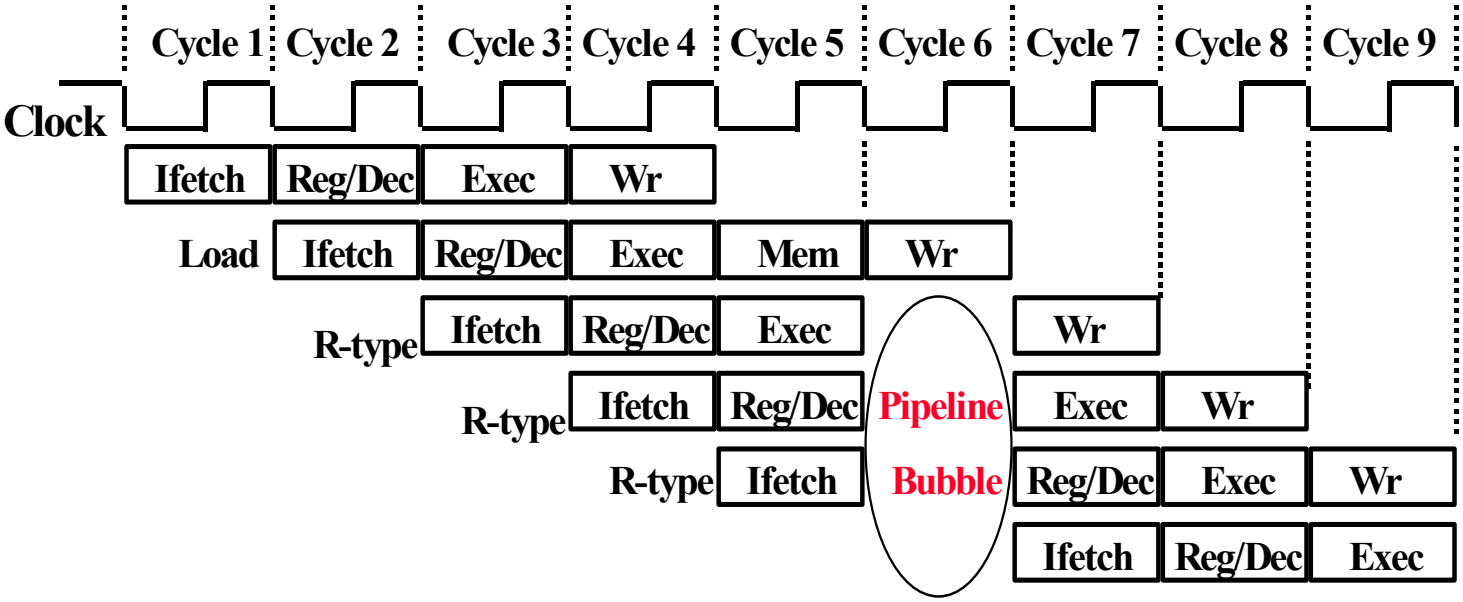
R-type uses Register File's Write Port in 4th stage.



Solution ???

Pipelining R-type and Load

Insert "Bubble" into the Pipeline



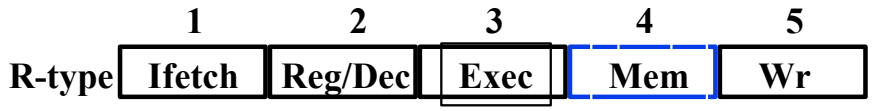
Insert a "bubble" into the pipeline to prevent 2 writes at the same cycle.

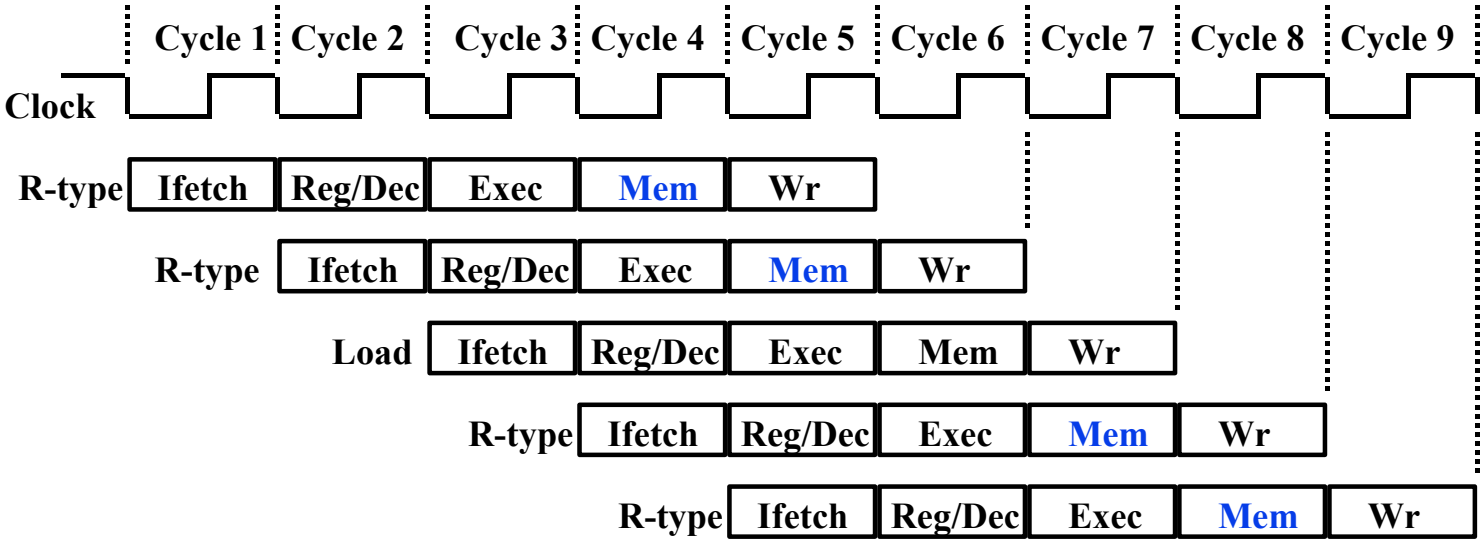
- The control logic can be complex.
- Lose instruction fetch and issue opportunity.

Pipelining R-type and Load

Delay R-type's register-write by one cycle:

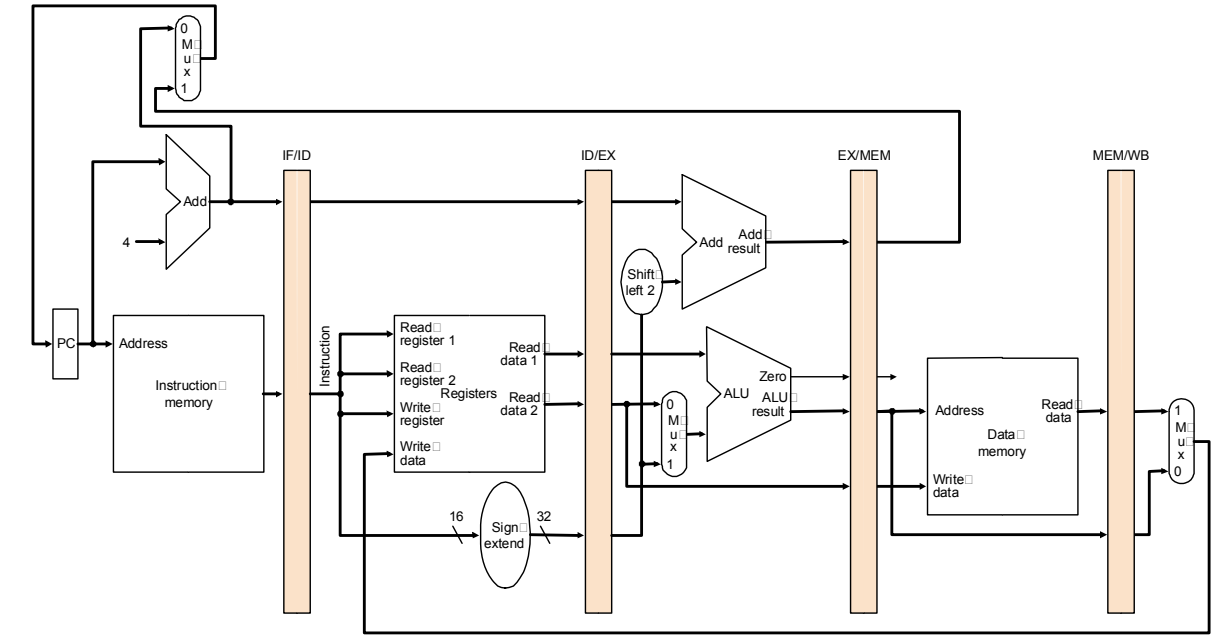
- Now R-type instructions also use Reg File's write port at Stage 5.
- Mem stage is a NOOP stage: nothing is being done.

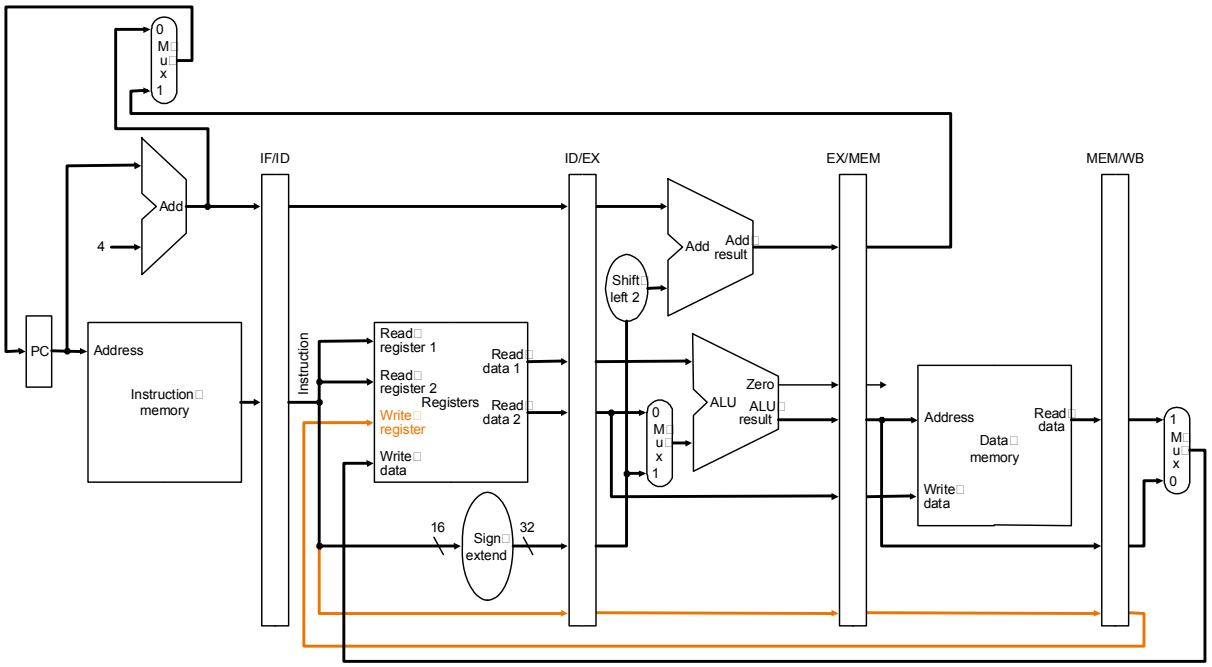




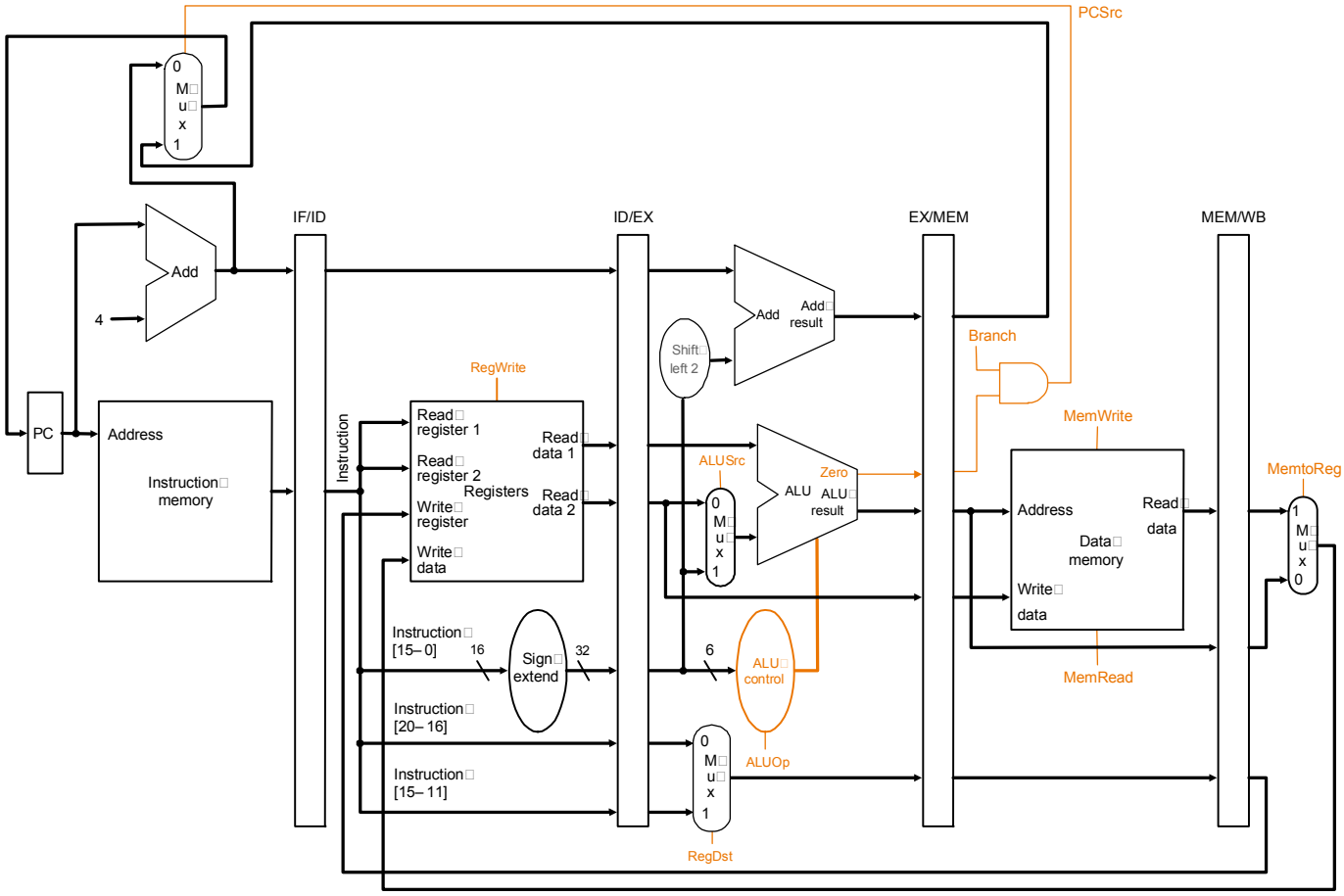
Stages in the Datapath

What do we need to add for splitting a datapath into stages?





Pipelined Datapath and Control



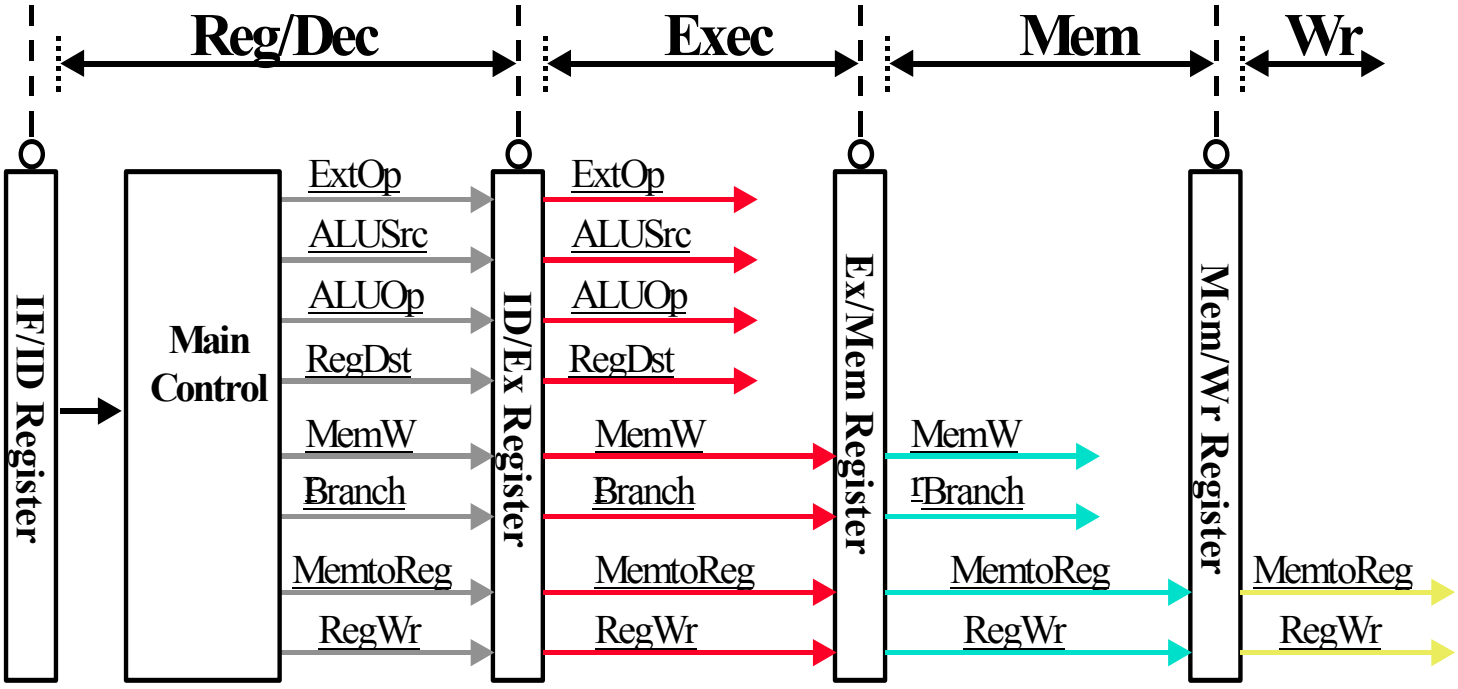
Pipeline Control

Instruction	Execution/Address Calculation stage control lines				Memory access stage control lines			stage control lines	
	Reg Dst	ALU Op1	ALU Op0	ALU Src	Branch	Mem Read	Mem Write	Reg write	Mem to Reg
R-format	1	1	0	0	0	0	0	1	0
`lw`	0	0	0	1	0	1	0	1	1
`sw`	X	0	0	1	0	0	1	0	X
`beq`	X	0	1	0	1	0	0	0	X

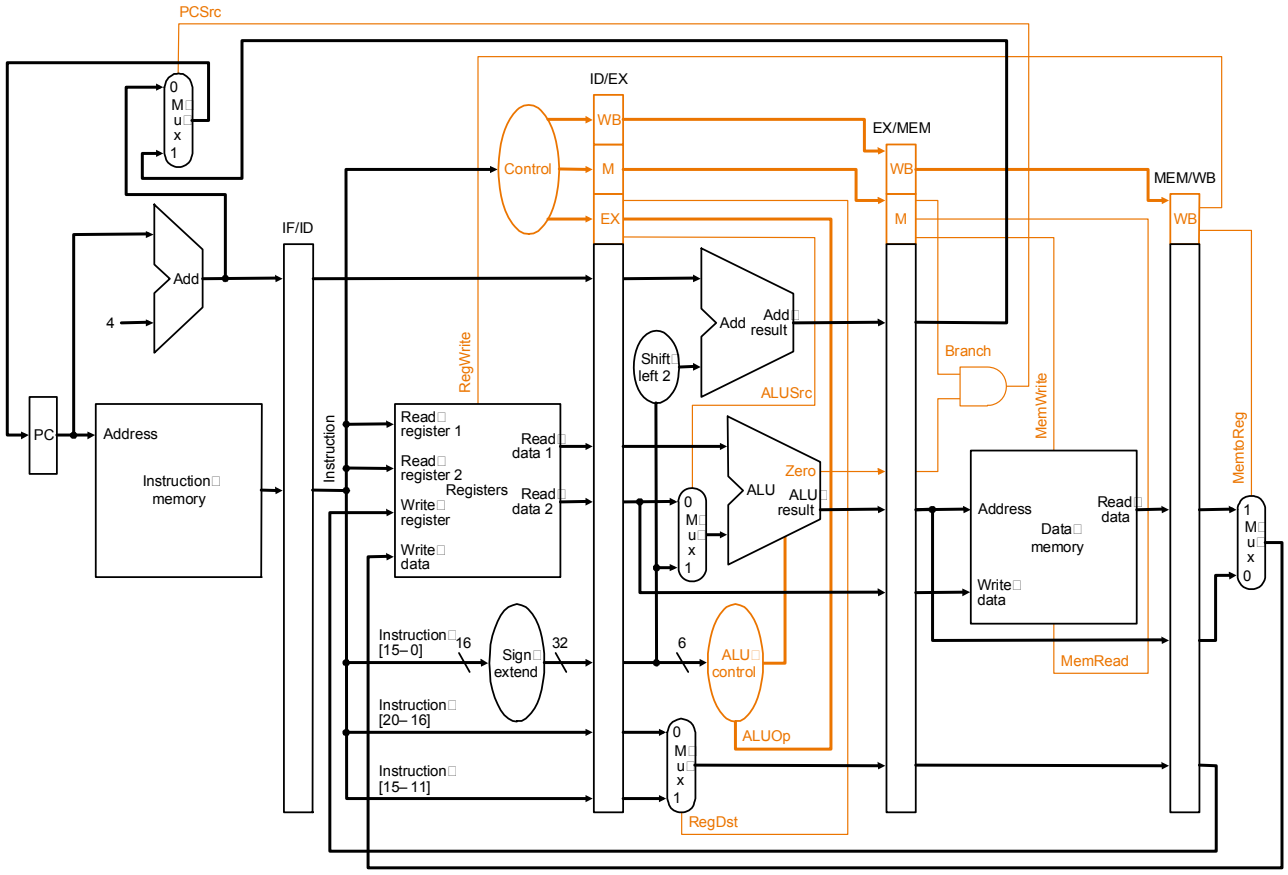
Main Control generates the control signals during Instruction Decode

- Exec (ExtOp, ALUSrc, ...) control signals are used 1 cycle later.
- Mem (MemWr Branch) control signals are used 2 cycles later.
- Wr (MemtoReg MemWr) control signals are used 3 cycles later.

Pass control signals along just like the data



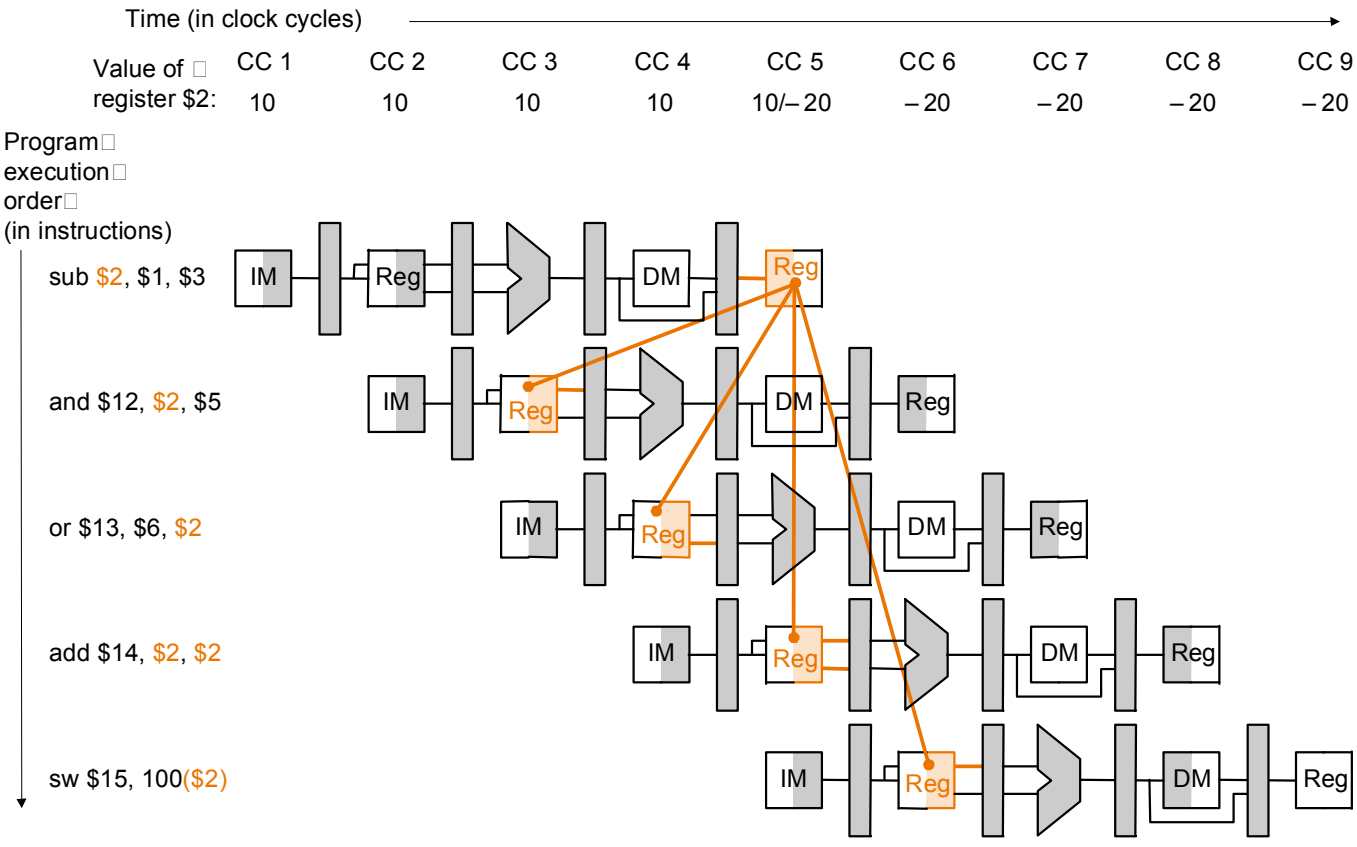
Datapath with Control



Dependencies

Problem: When starting next instruction before the first is completed.

Dependencies that "go backward in time" cause data hazards



Software Solution

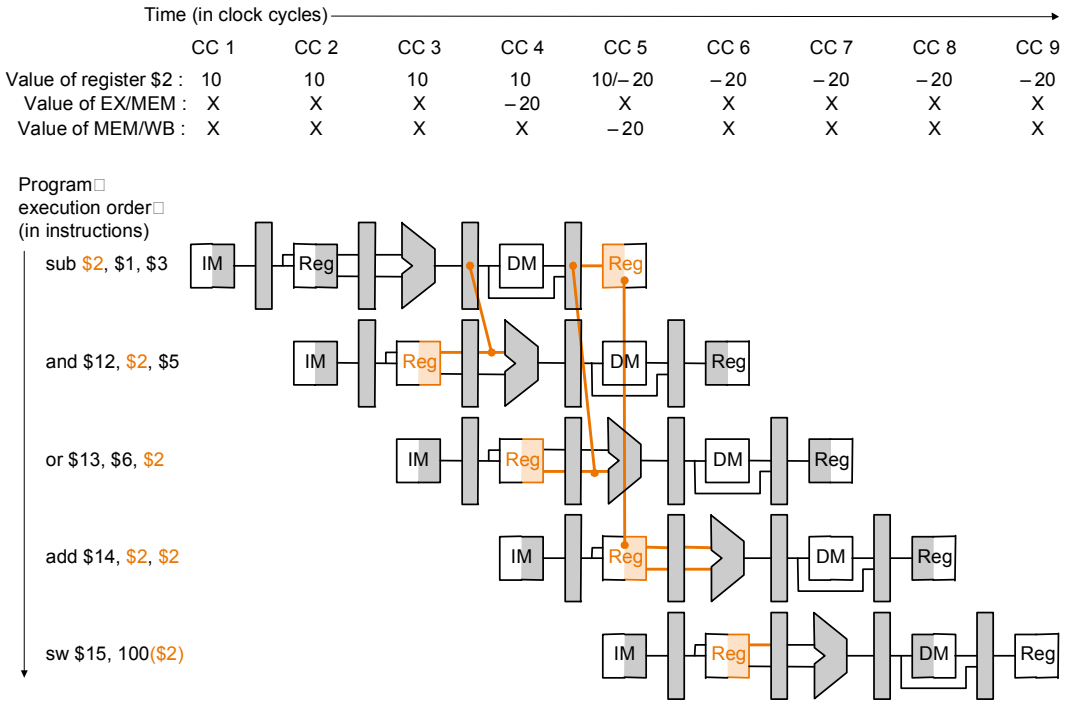
Have compiler guarantee no hazards
Where do we insert the "noops" ?

```
sub    $2, $1, $3
and    $12, $2, $5
or     $13, $6, $2
add    $14, $2, $2
sw     $15, 100($2)
```

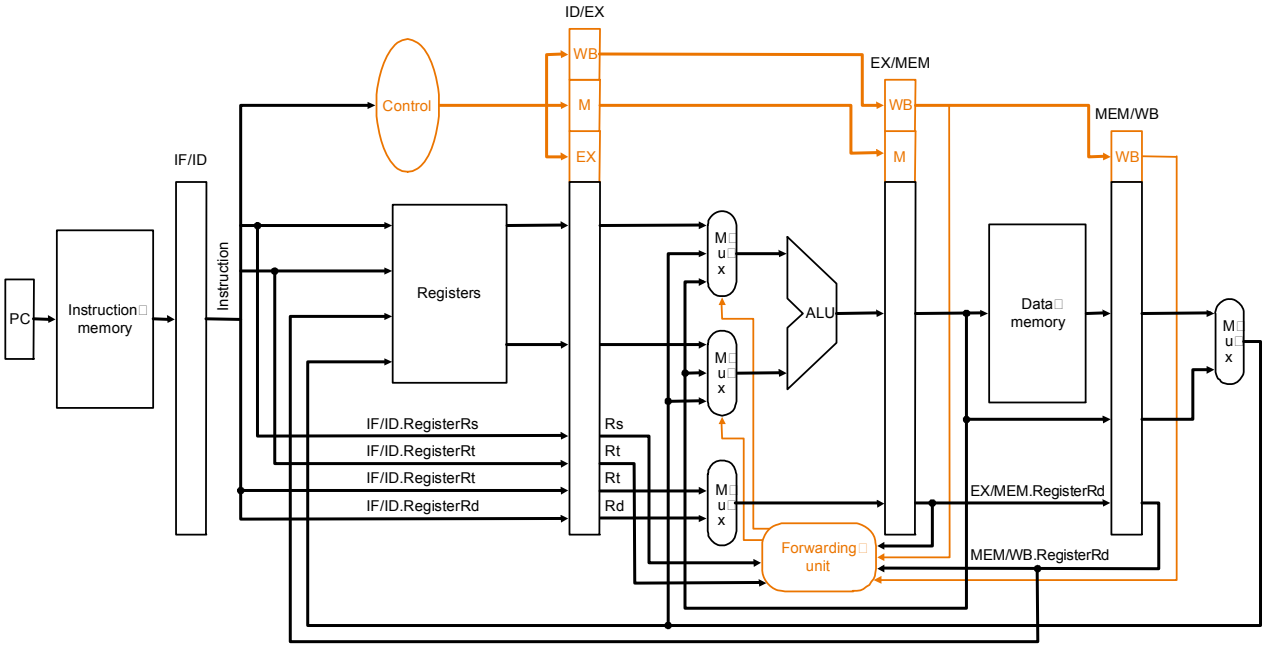
Problem: this really slows down the application!

Use temporary results and don't wait for them to be written into register file
Forwarding to handle read/write to same register

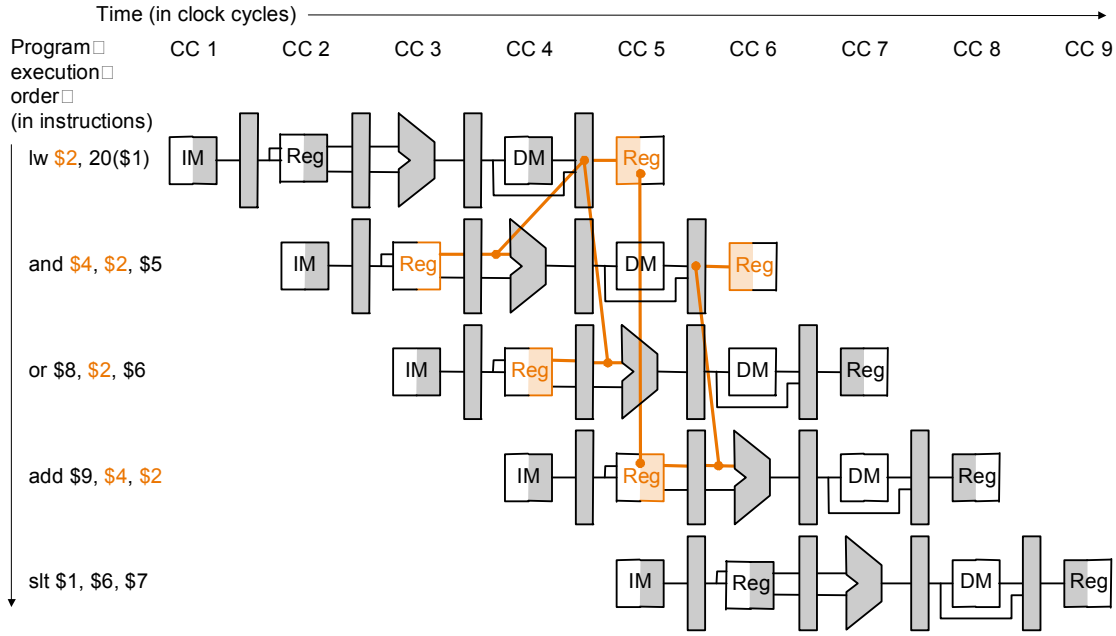
- ALU forwarding



Forwarding



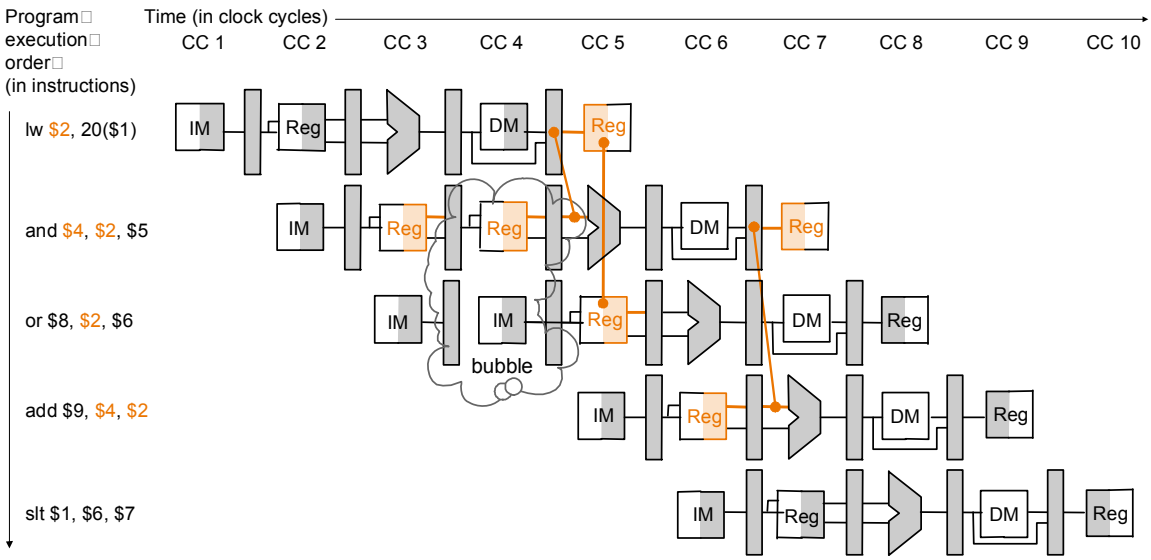
Can't always forward

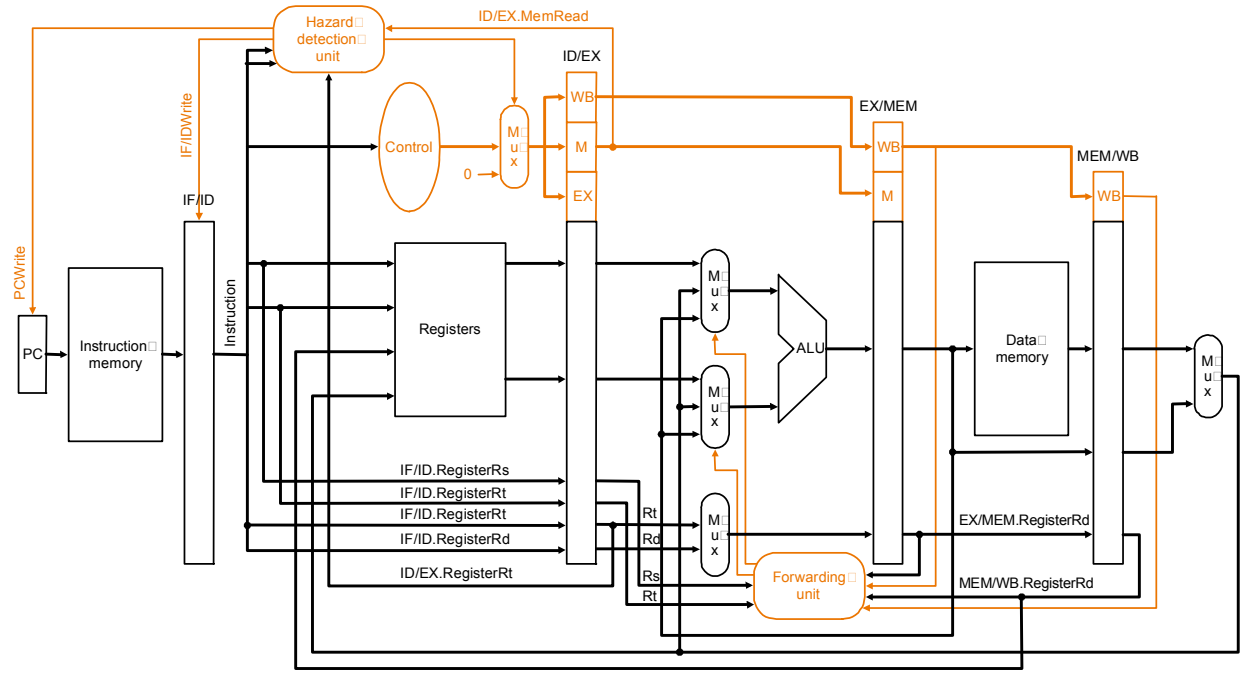


An instruction tries to read a register following a load instruction that writes to the same register.

Stalling

A hazard detection unit to “stall” load instruction
Stall the pipeline by keeping an instruction in the same stage

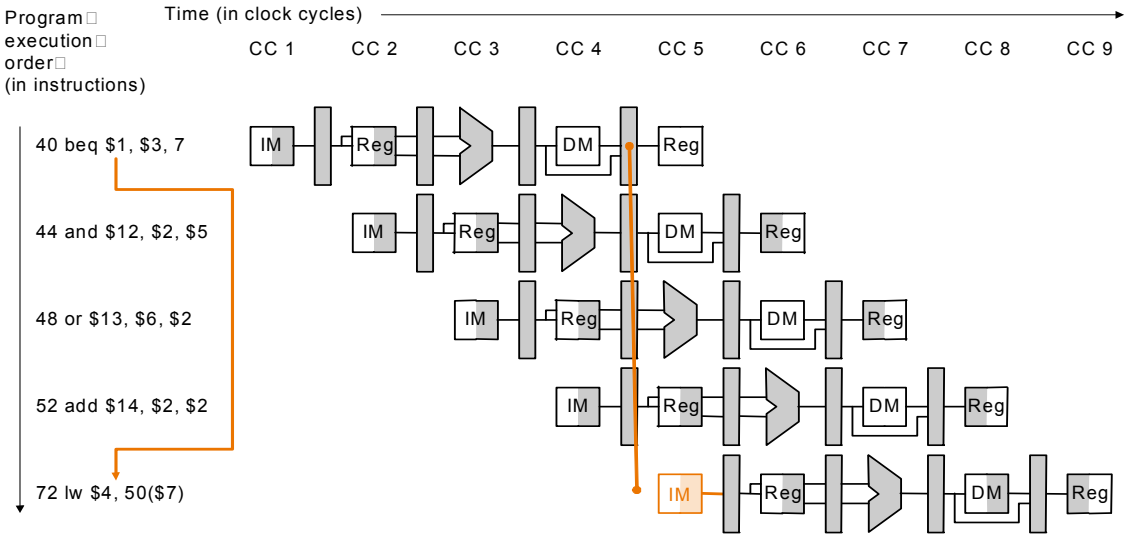


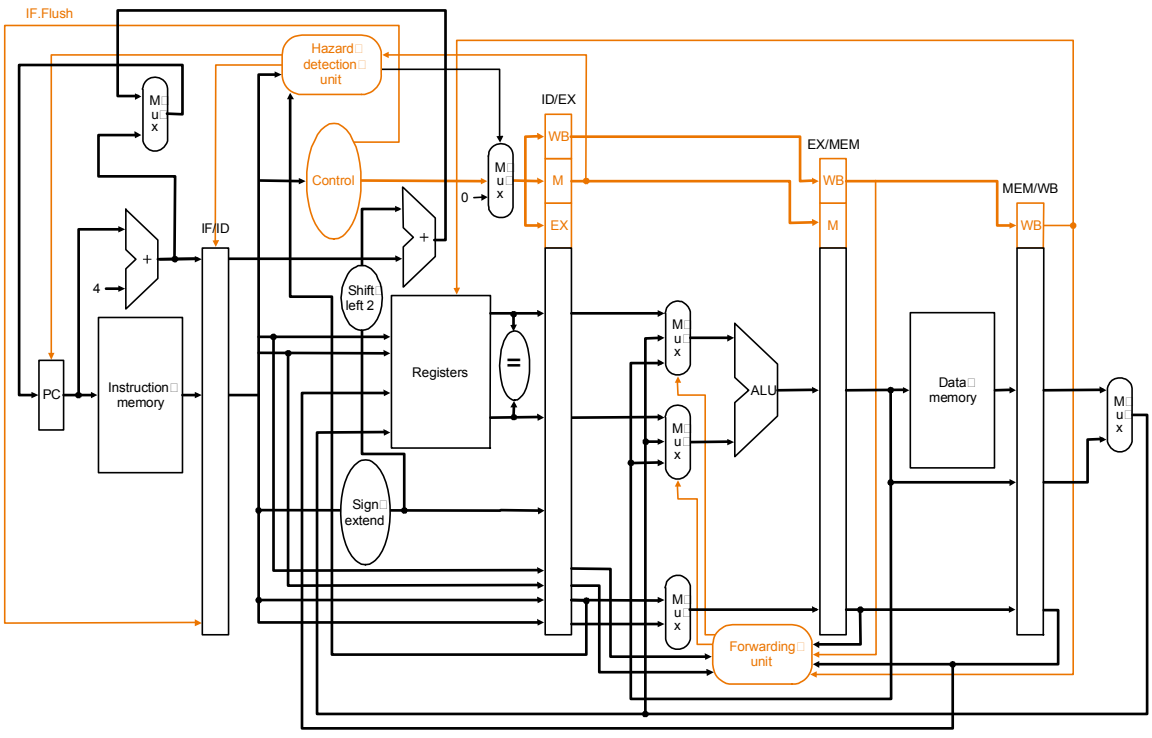


Branch Hazards

When decide to branch, instructions are in pipeline
We predict “branch not taken”

Need to add hardware for flushing instructions if we are wrong.





Improving Performance

Try and avoid stalls. e.g. reorder instructions.

```
lw $t0, 0($t1)
lw $t2, 4($t1)
sw $t2, 0($t1)
sw $t0, 4($t1)
```

Add a "branch delay slot"

- Always execute next instruction after a branch.
- Rely on compiler to "fill" the slot with something useful rather than NOOPS

Dynamic Scheduling

The hardware performs the "scheduling"

- Hardware tries to find instructions to execute
- Out of order execution is possible
- Speculative execution and dynamic branch prediction

Modern processors are very complicated

- Alpha 21264: 9 stage pipeline, 6 instruction issue
- PowerPC and Pentium: Keeps branch history table.

Compiler technology important

Superscalar Architecture

Start more than one instruction in the same cycle

Nondelayed vs. Delayed Branch

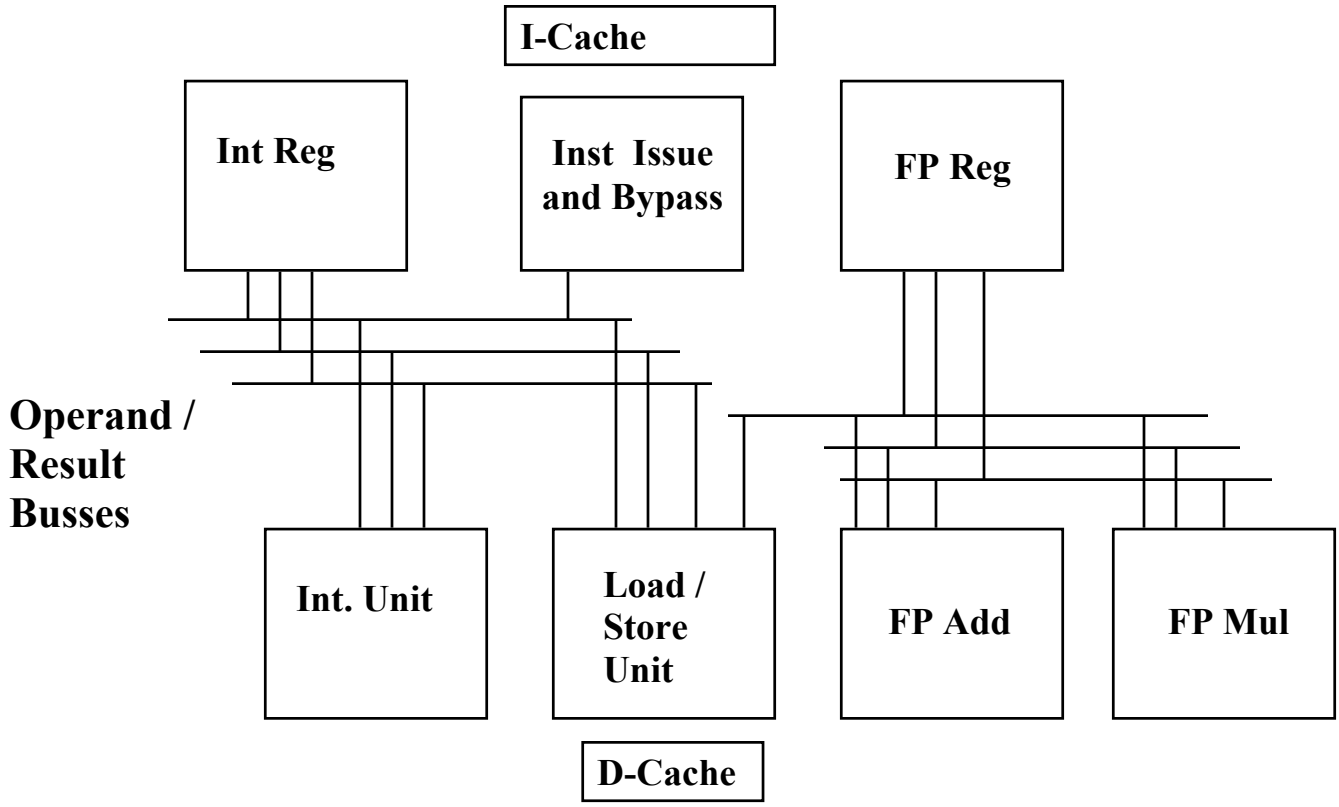
```
            or   $8, $9 ,$10
            add $1 ,$2,$3
            sub $4, $5,$6
            beq $1, $4, Exit
            xor $10, $1,$11
      Exit:
```

Fill Slot

```
            add $1 ,$2,$3
            sub $4, $5,$6
            beq $1, $4, Exit
            or   $8, $9 ,$10
            xor $10, $1,$11
      Exit:
```

Superscalar Architecture

An Example: Independent integer and FP issue to separate pipelines

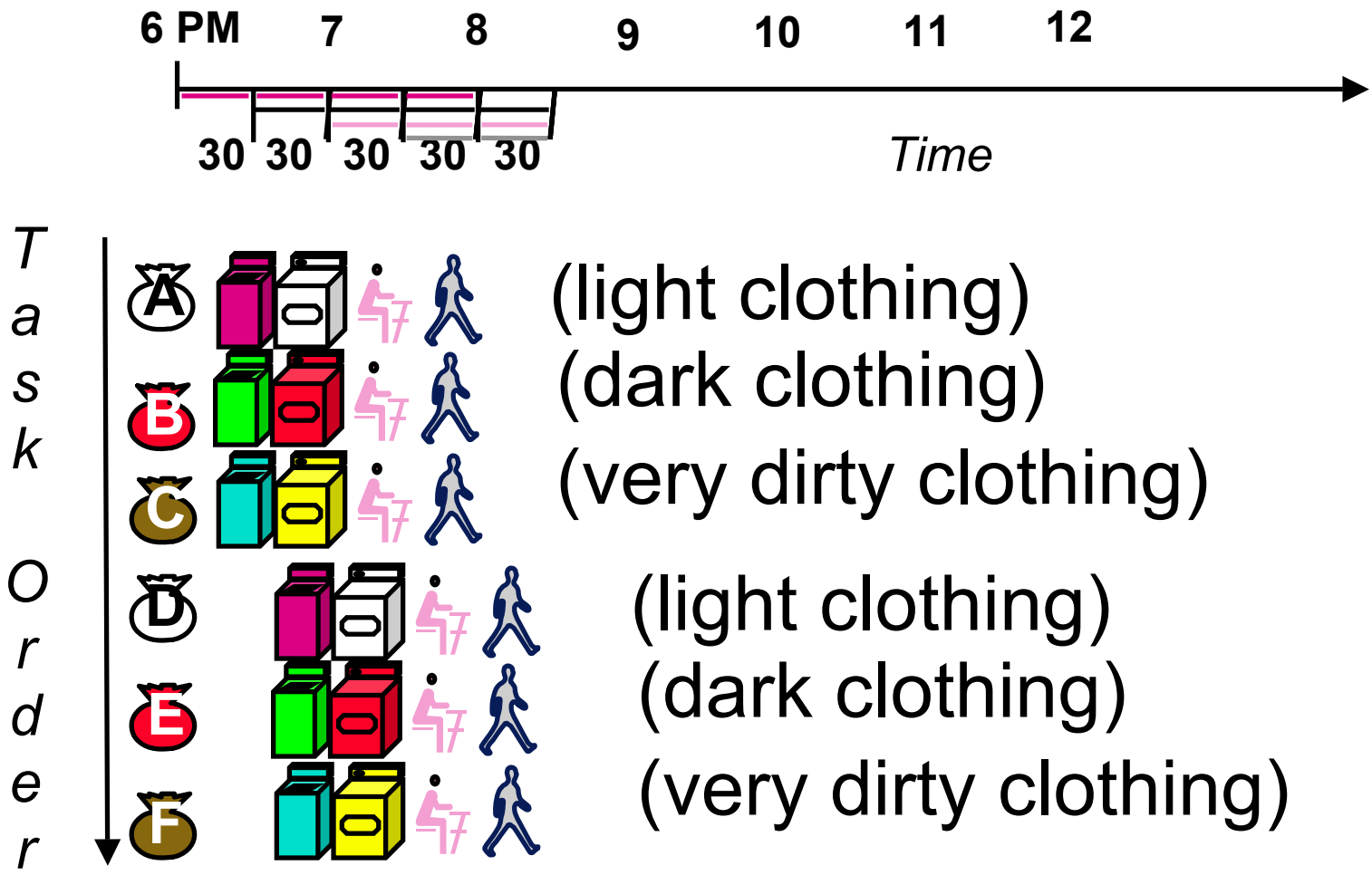


Single Issue Total Time =

Int Exec Time + FP Exec Time

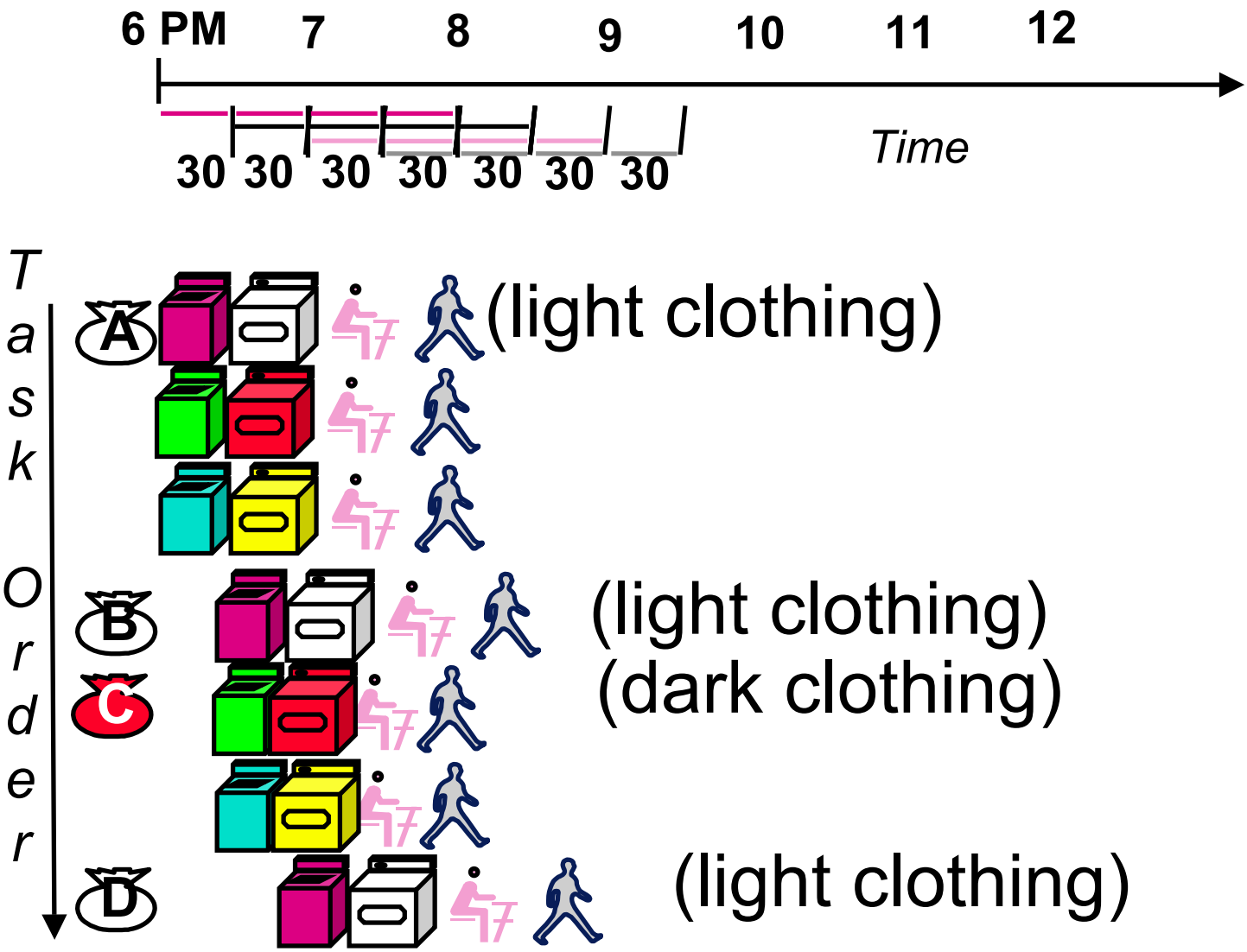
Max Speedup: $\frac{\text{Total Time}}{\text{MAX(Int Exec Time, FP Exec Time)}}$

Superscalar Laundry: Parallel per stage



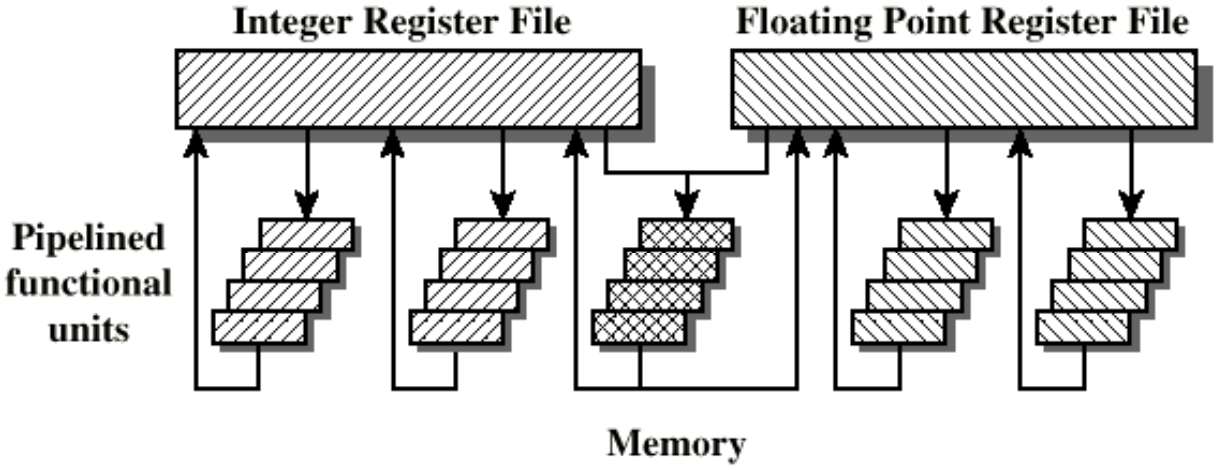
More resources, HW to match mix of parallel tasks?

Superscalar Laundry: Mismatch Mix



Task mix underutilizes extra resources

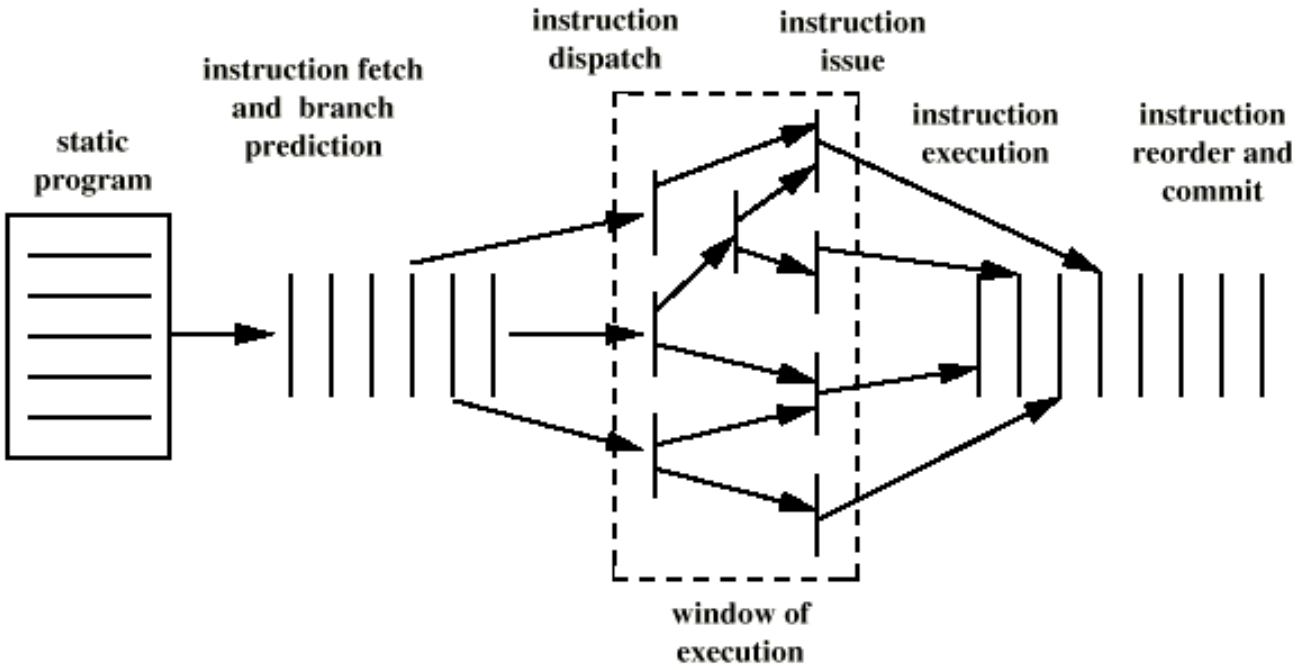
General Superscalar Organization



Super-Pipelined

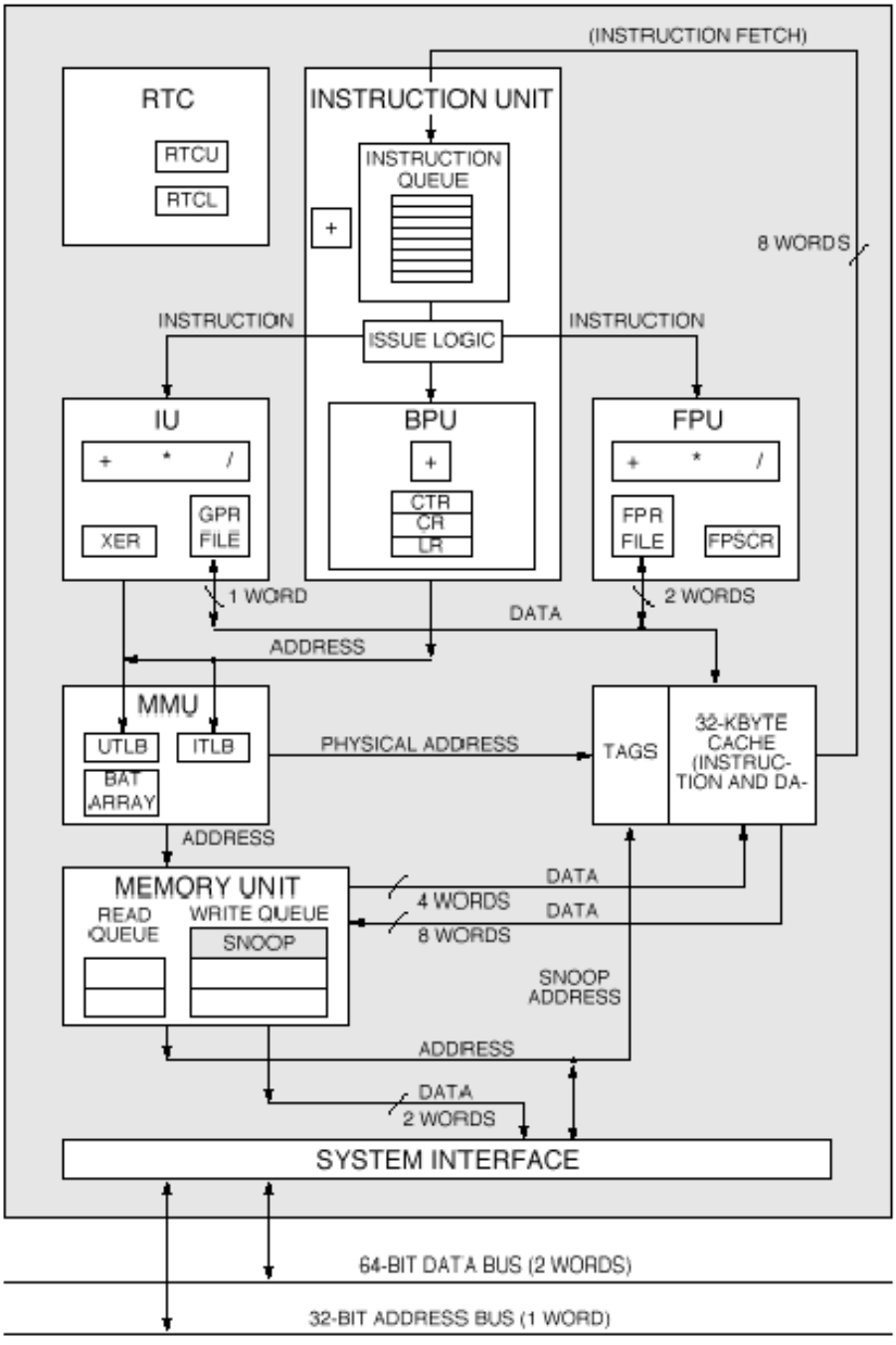
- Many pipeline stages need less than half a clock cycle.
- Double internal clock speed gets two tasks per external clock cycle
- Superscalar allows parallel fetch and execute

Superscalar Execution

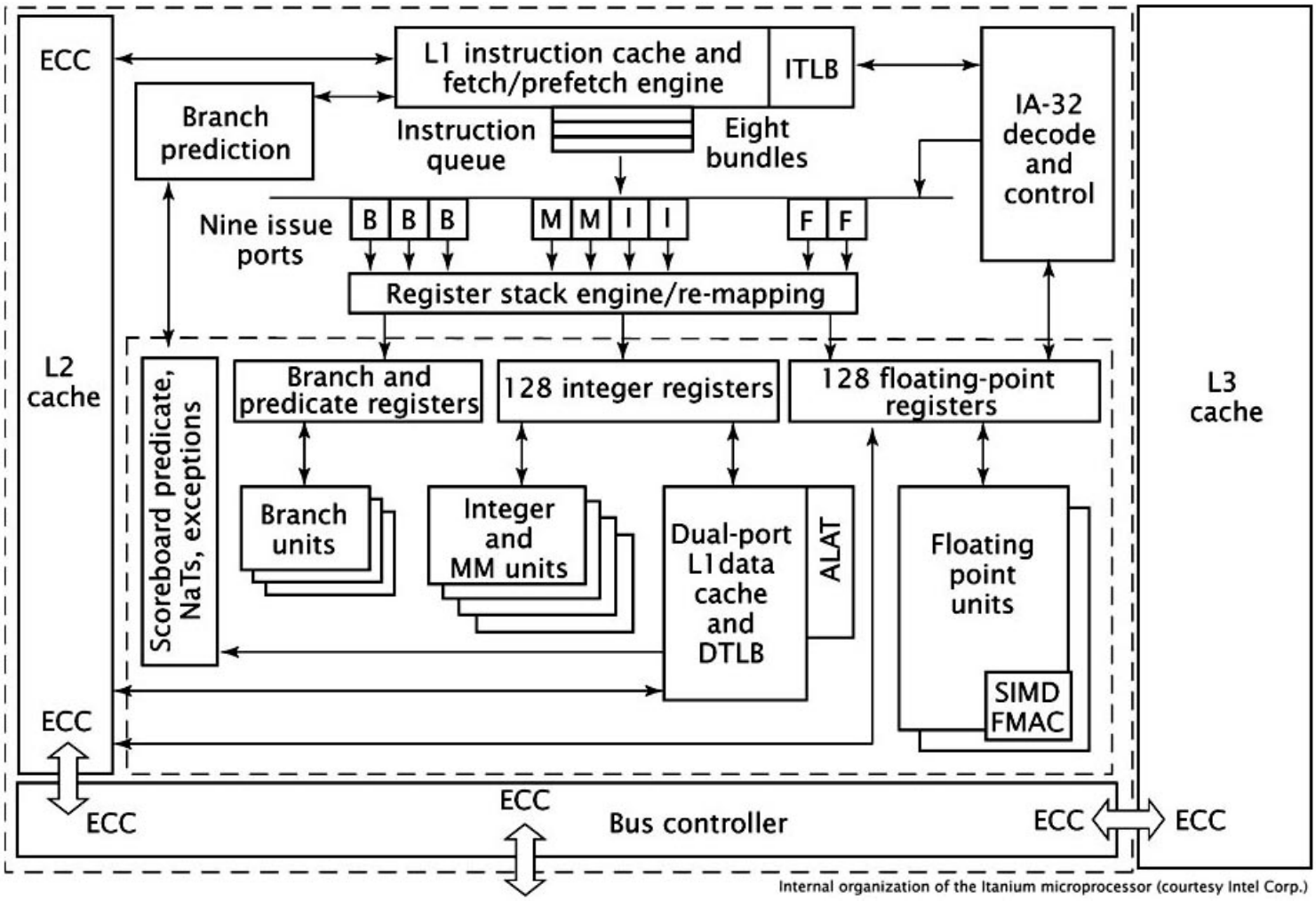


- Simultaneously fetch multiple instructions
- Logic to determine true dependencies involving register values.
- Mechanisms to communicate these values.
- Mechanisms to initiate multiple instructions in parallel.
- Resources for parallel execution of multiple instructions.
- Mechanisms for committing process state in correct order.

The PowerPC 601 Architecture



Intel Itanium Microprocessor



Internal organization of the Itanium microprocessor (courtesy Intel Corp.)