ECE 341

Lecture # 12

Instructor: Zeshan Chishti
zeshan@ece.pdx.edu

November 10, 2014

Portland State University

Lecture Topics

- Basic Processing Unit
  - Control Signals
  - Hardwired Control
    - Datapath control signals
    - Dealing with memory delay
- Pipelining
  - Basic Concept
  - Pipeline Organization

- References:
  - Chapter 5: Section 5.5 and 5.6 (Pages 172-178 of textbook)
  - Chapter 6: Sections 6.1 and 6.2 (Pages 194-196 of textbook)

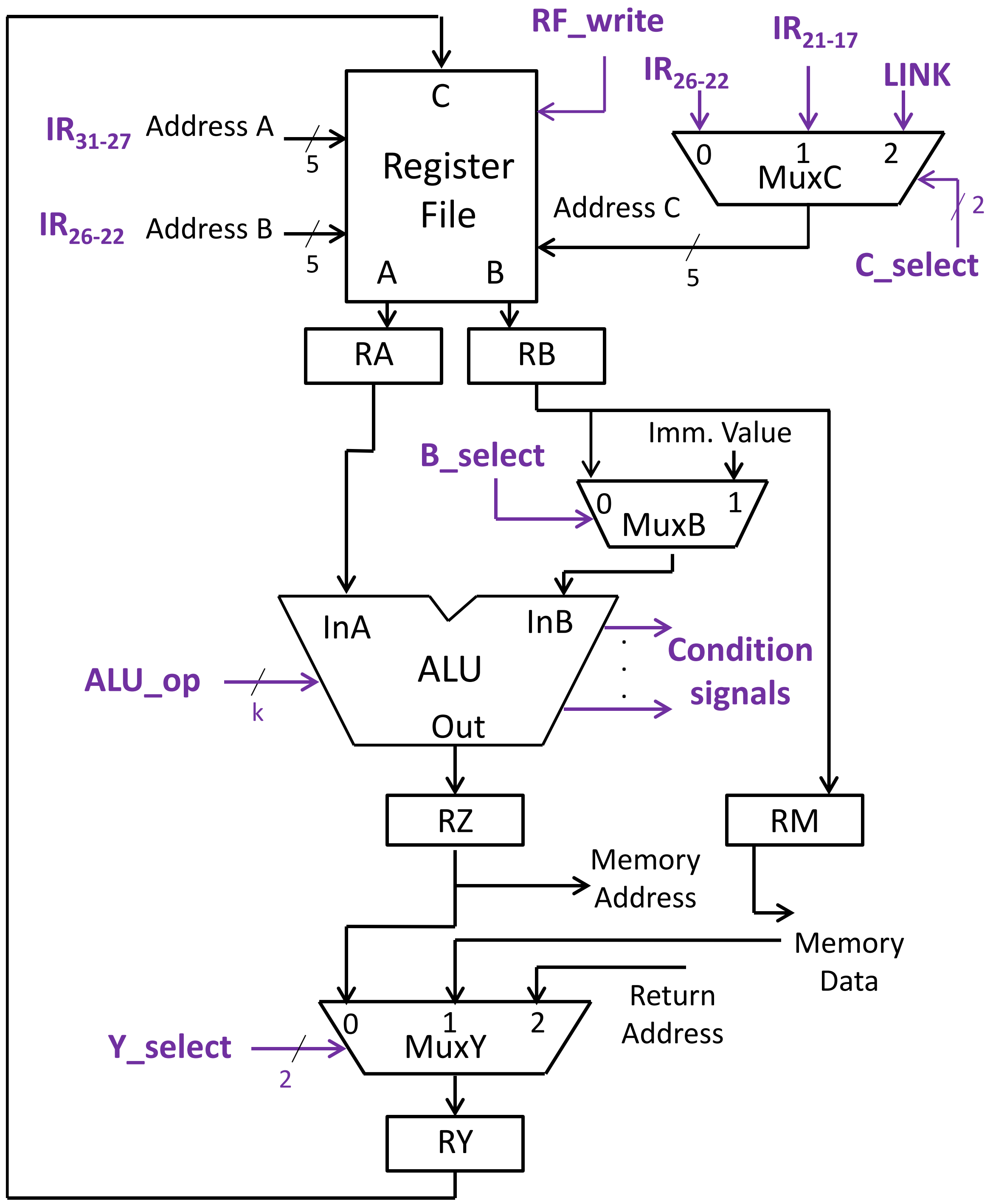
Control Signals

- Control signals govern the operation of a processor's components
- Control circuitry examines the instruction in IR and generates the control signals needed to execute the instruction
- Examples of decisions made by control signals:
  - Which registers (if any) are enabled for writing?
  - Which input is selected by a multiplexer?
  - What operation is performed by the ALU?
- Some control signals depend only on instruction type, while others depend on both the instruction type and current processing step

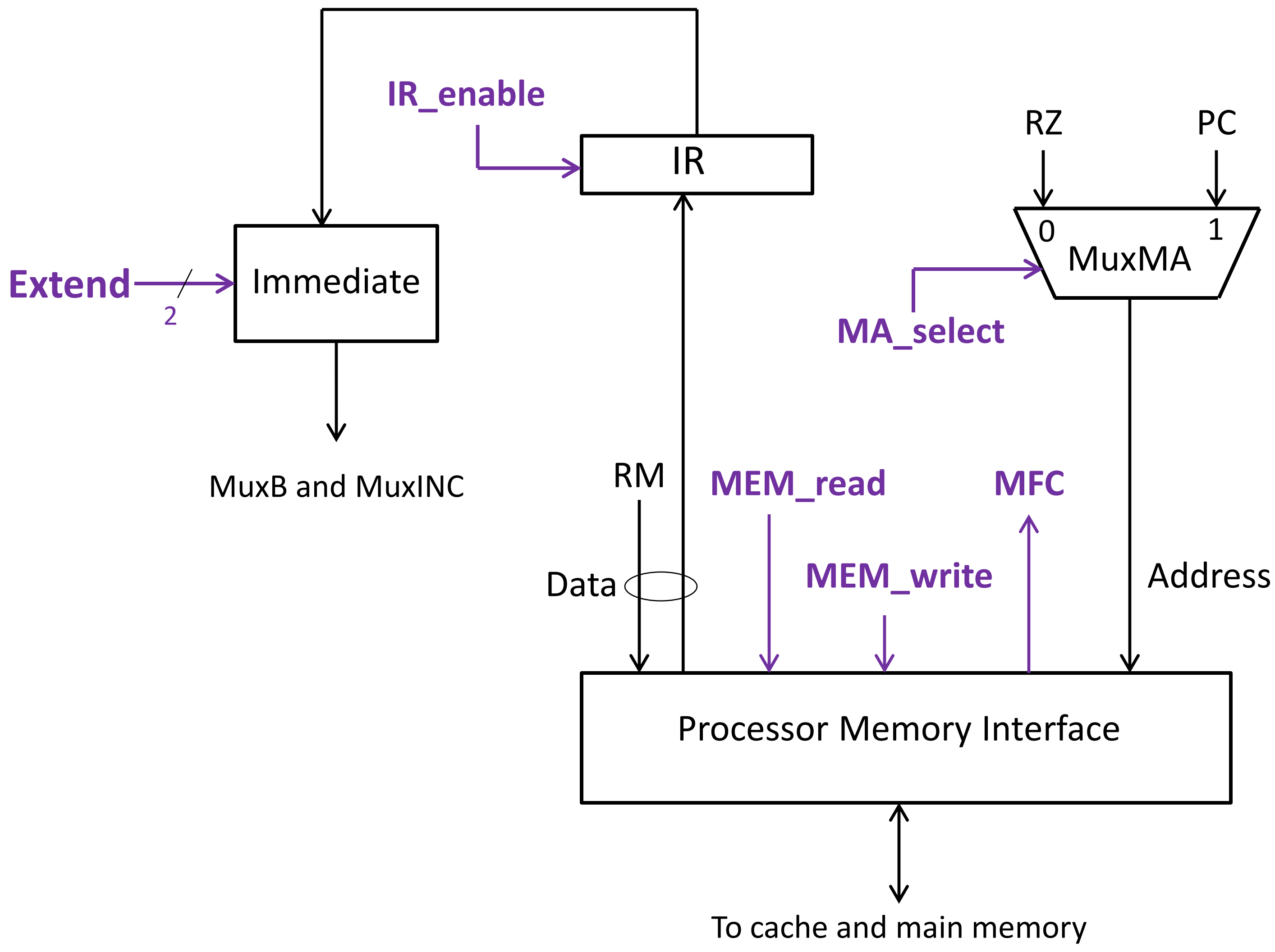
Control Signals for Datapath

Need control signals for register file, multiplexers and ALU

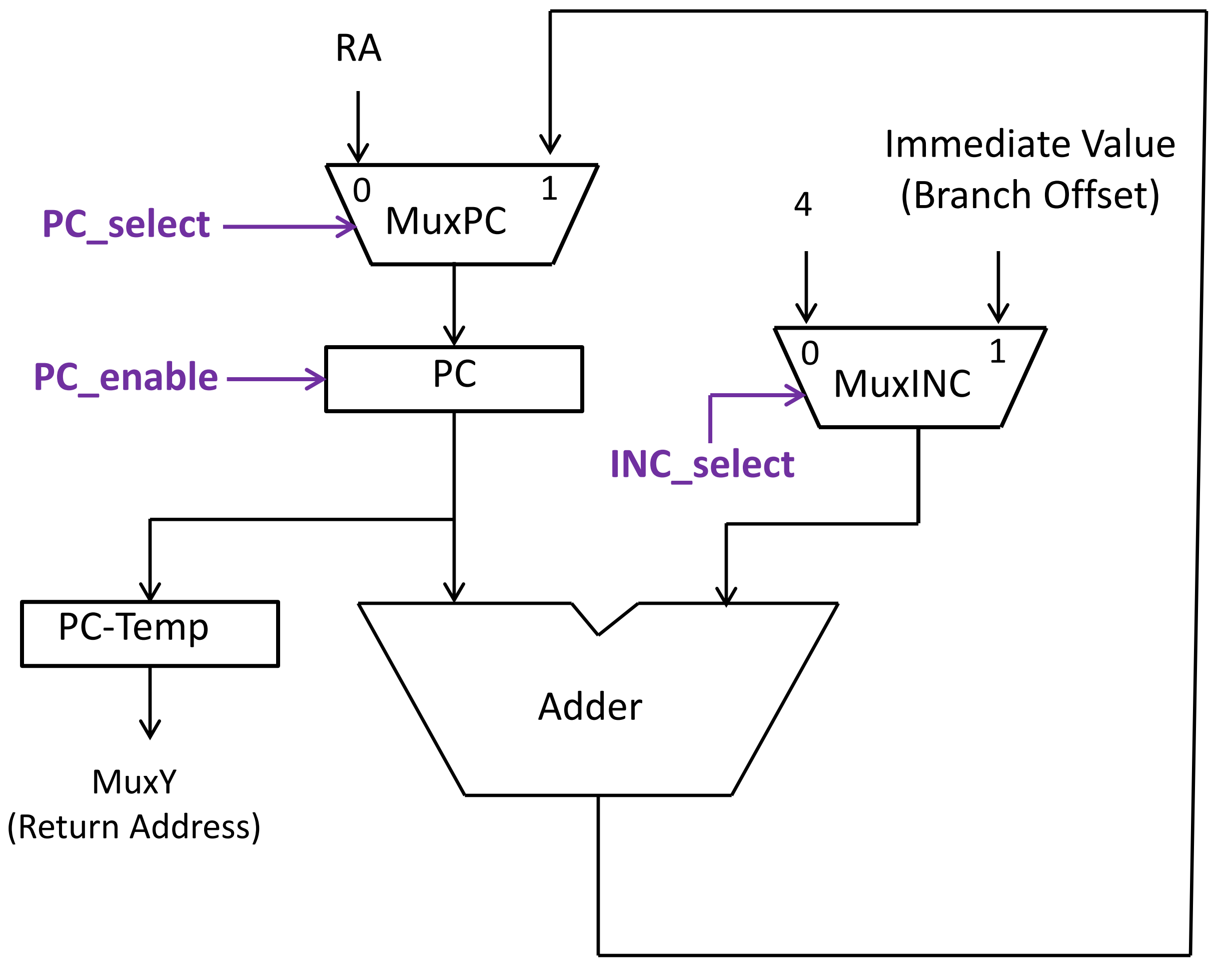
Inter-stage registers transfer data from one stage to the next in ***every*** cycle => **no need** for a control signal, since these registers are always enabled



Control Signals for Memory Interface and IR



Control Signals for Instruction Address Generator

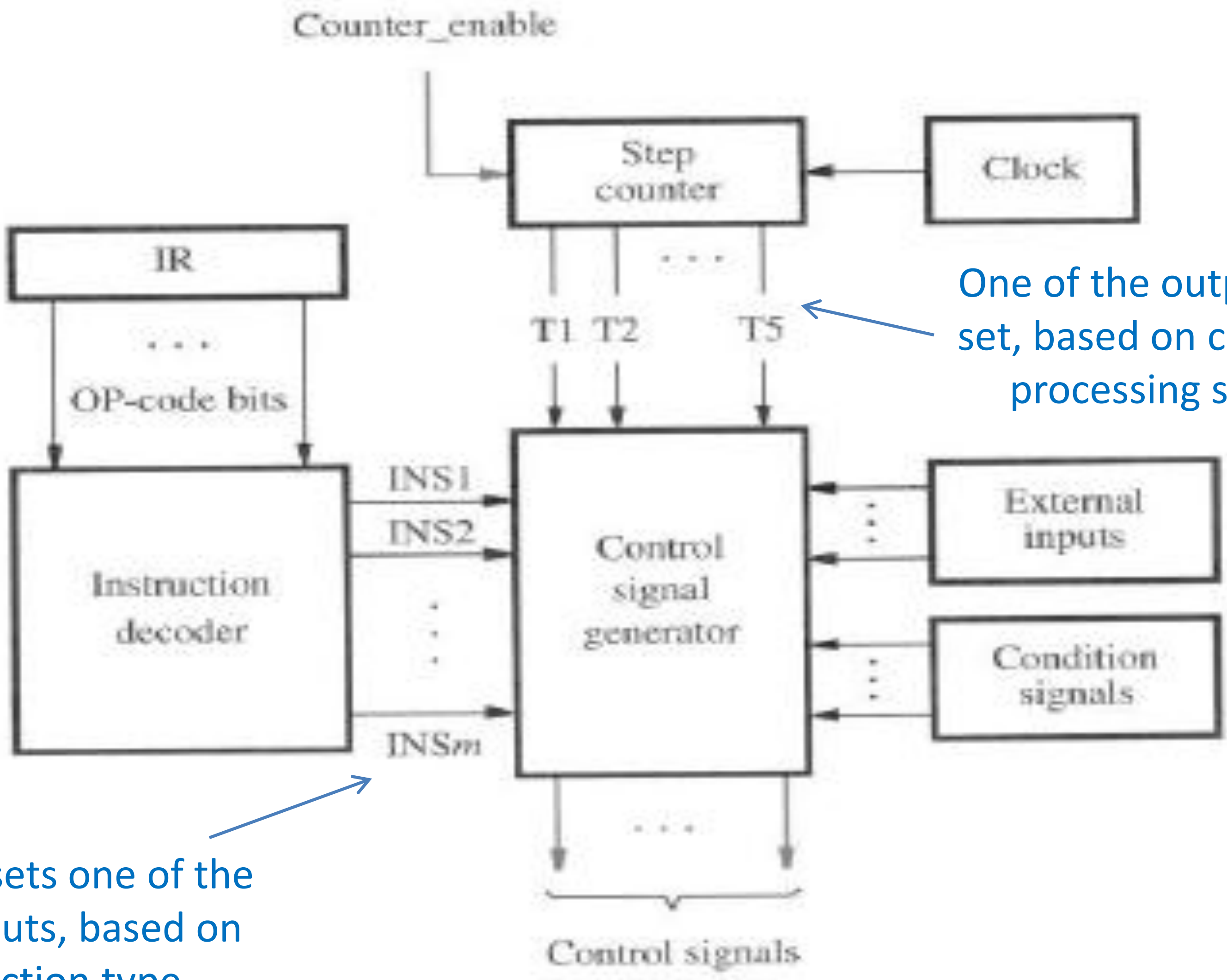


Hardwired Control

- How does the processor ensure that the control signals needed to execute an instruction are generated in the correct sequence and at the right time?
- Two basic approaches: (1) Hardwired control, (ii) Microprogrammed control

- The control signals depend on the current processing step for an instruction
- Question: How do we keep track of the current step of an instruction?
- Answer: use a step counter
- Question: What other factors do the control signals depend on?
- Answer:
  - Contents of instruction register
  - The result of a computation (or comparison) operation
  - External input signals, such as interrupt requests

Generation of Control Signals



One of the outputs is set, based on current processing step

Decoder sets one of the INS*i* outputs, based on instruction type

Generation of Control Signals (cont.)

- **Example:** Consider the fetch stage (stage 1) of the five-stage hardware

- Step counter asserts the signal T1
- Control circuitry:
  - sets *MA_select* signal to 1 to select PC contents as memory address
  - activates *Mem_Read* to initiate a memory read operation
  - activates *IR_enable* to load the data returned from memory into *IR,* when *MFC* is asserted
  - sets *Inc_Select* to 0, *PC_select* to 1 and asserts *PC_enable* to increment PC by 4 at the end of step T1

Datapath Control Signals

- Setting of control signals can be determined by examining the actions taken in each execution step of every instruction

- Example 1: *RF_write* signal is set to 1 in step T5 during an instruction that writes data into the register file:

  $$RF\_write = T5.(ALU + Load + Call)$$

  where *ALU*, *Load* and *Call* stand for arithmetic/logic instructions, load instructions and subroutine call instructions respectively
  - *RF_write* is a function of **both** the timing and instruction signals

- Example 2: The multiplexer *B_select* is a function of **only** the instruction and does not need to change from one timing step to the other

  $$B\_select = Immediate$$

  where *Immediate* stands for all instructions that use an immediate operand

Dealing with Memory Delay

- The step counter is usually incremented at the end of every clock cycle
- However, a step in which a *Mem_Read* or *Mem_Write* is issued does not end until the *MFC* signal is asserted
- Step counter should not be incremented until the MFC signal is asserted
- Counter_enable signal controls whether the step counter is incremented
- Let WMFC be a control signal that represents the need to wait for memory
  - WMFC is activated only in those steps in which the *Wait_for_MFC* command is issued

Counter_enable = NOT(WMFC) + MFC

- We must also ensure that PC is incremented only once when the instruction fetch step is extended for more than one clock cycle
  - PC_enable signal controls if the PC is incremented or not

PC_enable = T1.MFC + T3.BR

where BR stands for all the branch instructions

Pipelining

Overview

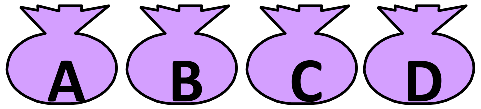
- So far, we have assumed that only *one* instruction is being processed by the multi-stage hardware at any point of time
- How do we decrease the execution time of a program?
- One possibility is to use faster circuits to implement the processor
  - This approach will decrease the execution time of each instruction
- Another possibility is to arrange the processor hardware in such a way that *multiple* instructions can be processed at the same time. This approach is called **pipelining**

Pipelining does not change the time needed to perform a single instruction, but it increases the number of instructions performed per second (*instruction completion rate* or *throughput*)

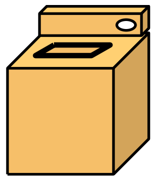
Traditional Pipelining Concepts

- Laundry Example

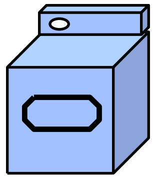
- Four loads of laundry need
to be washed, dried and folded



- Washer takes 30 minutes

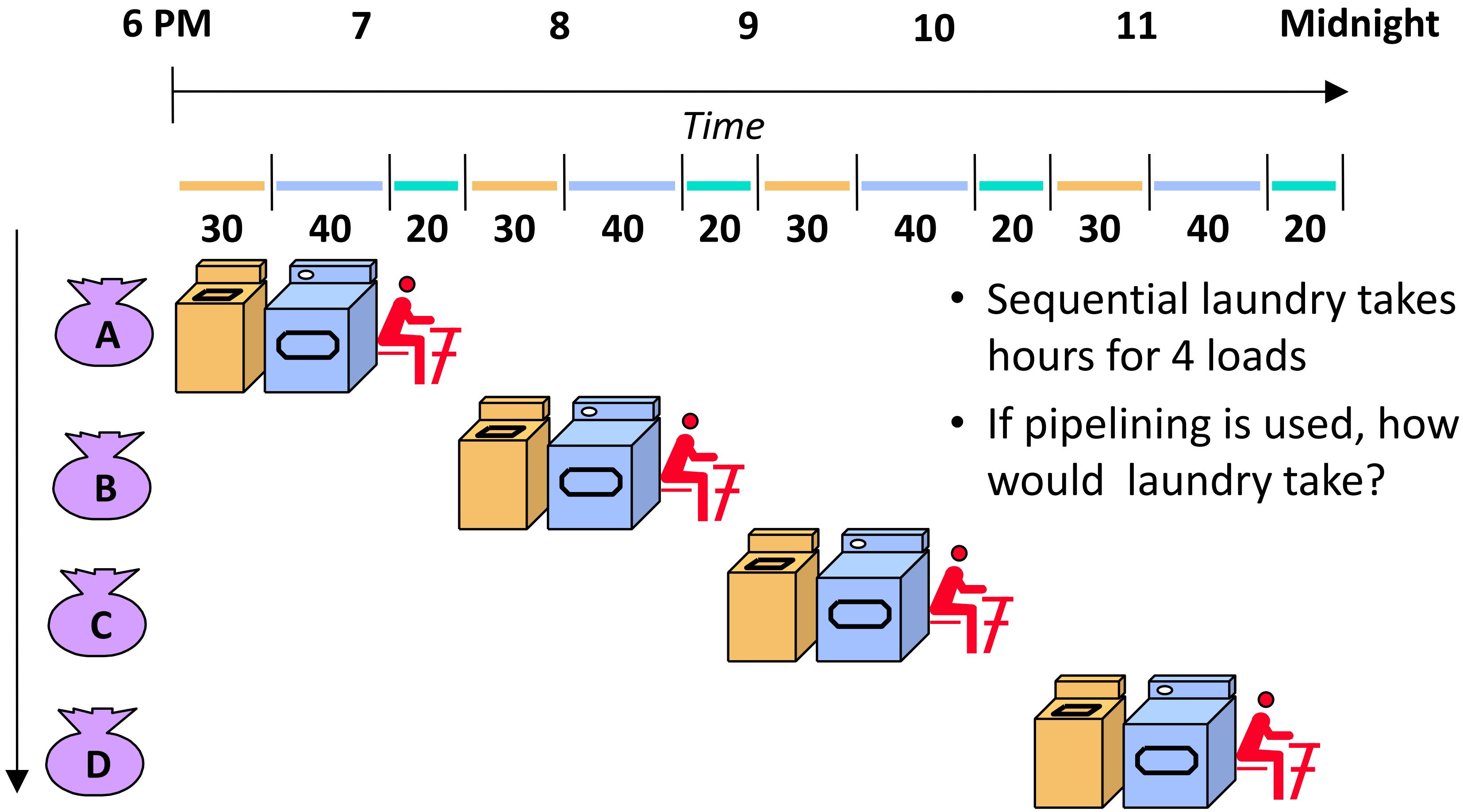


- Dryer takes 40 minutes



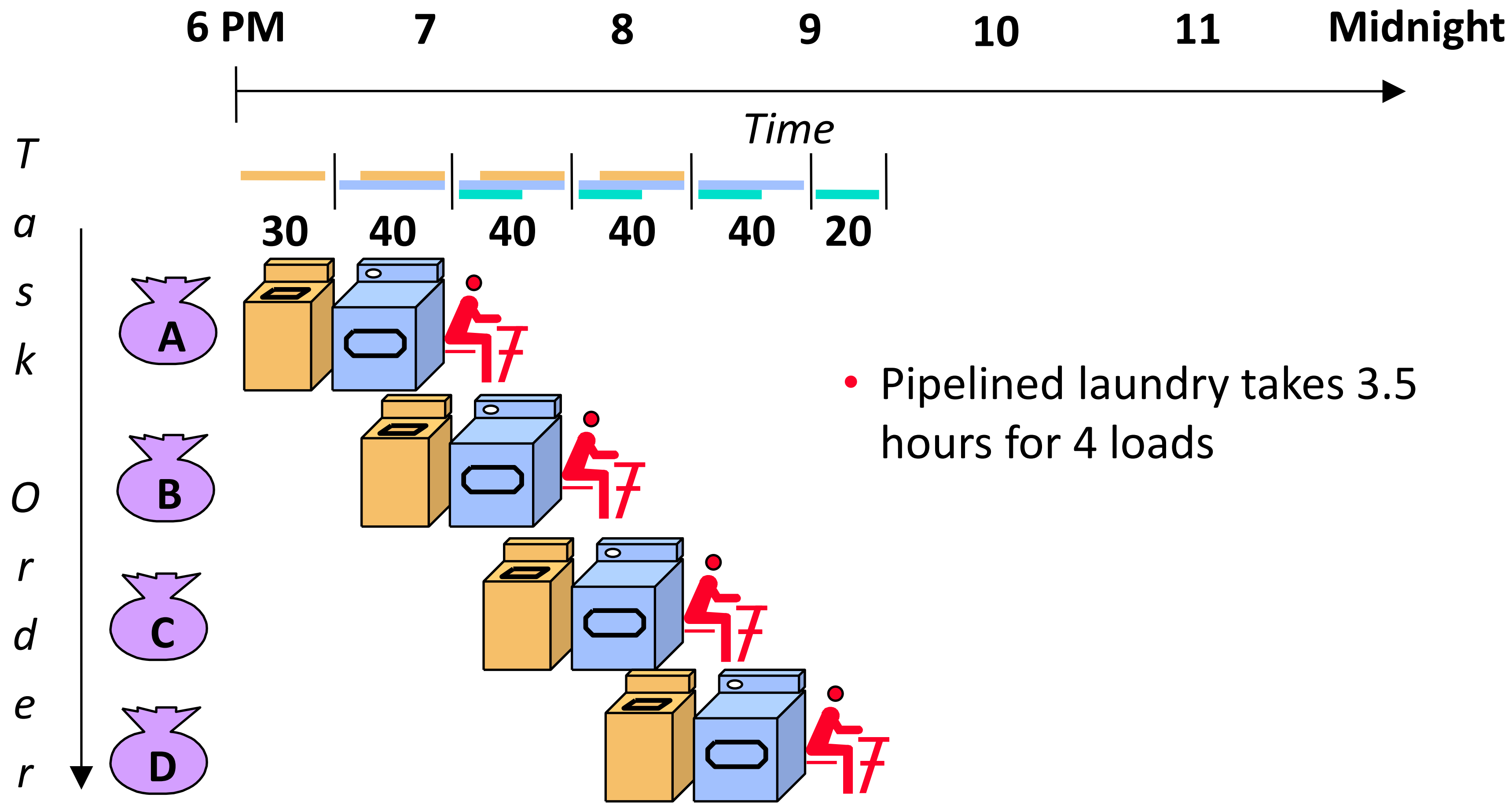
- “Folder” takes 20 minutes

Traditional Pipelining Concepts (cont.)



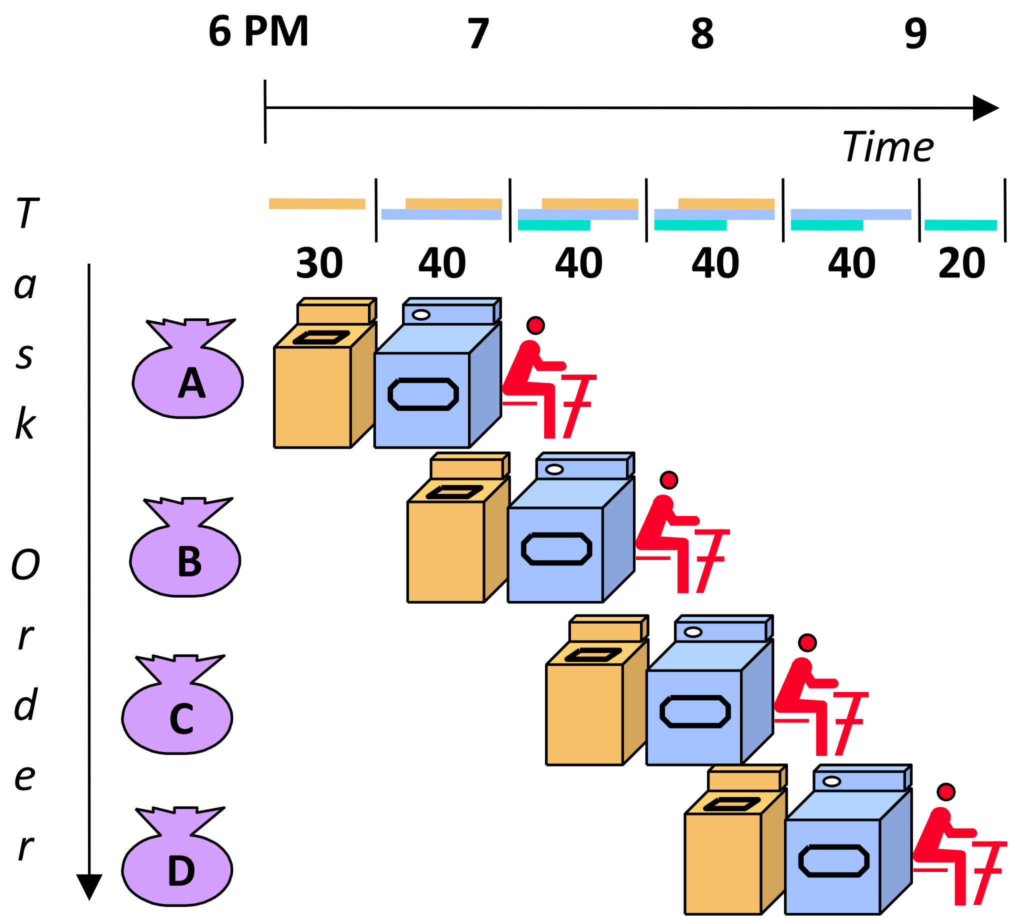
- Sequential laundry takes 6 hours for 4 loads
- If pipelining is used, how long would laundry take?

Traditional Pipelining Concepts (cont.)



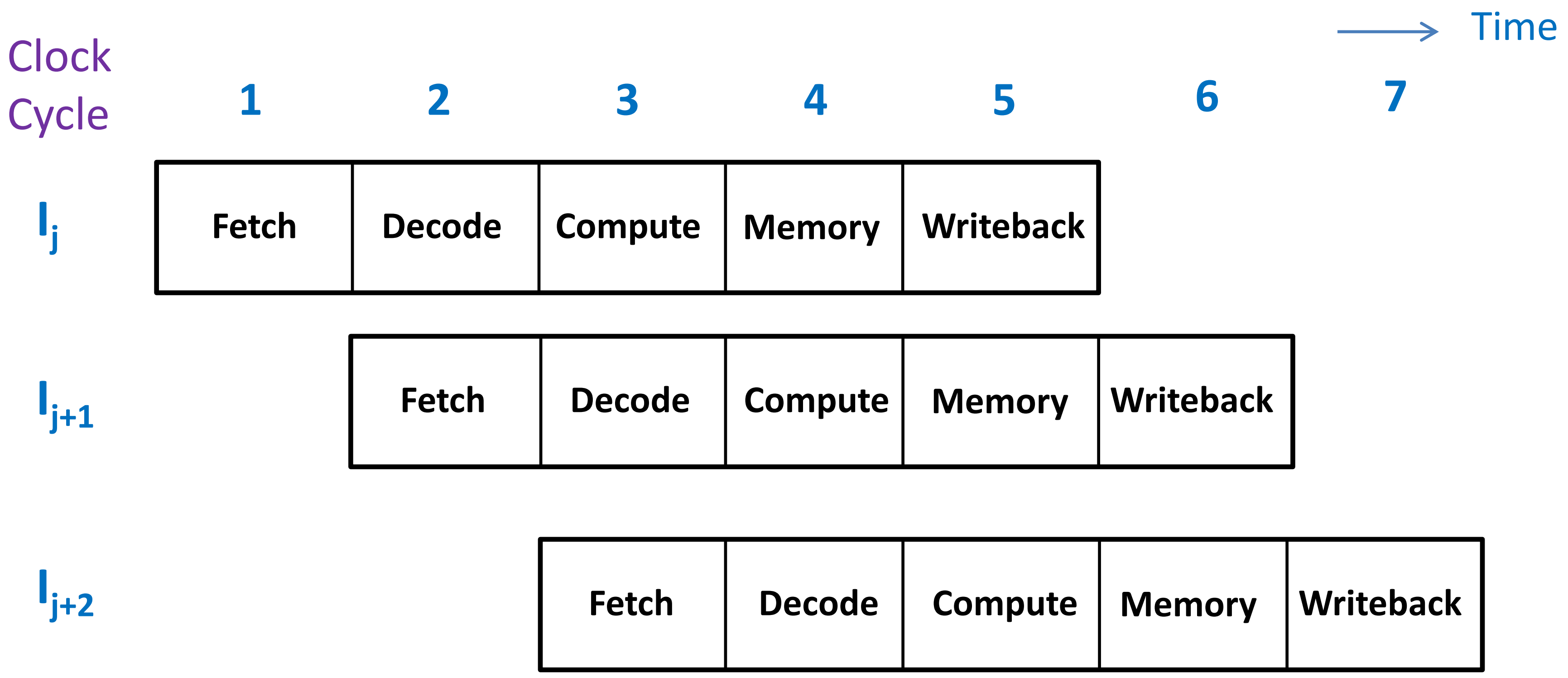
- Pipelined laundry takes 3.5 hours for 4 loads

Traditional Pipelining Concepts (cont.)



- Pipelining doesn't reduce the time taken by an individual task, it improves the throughput of entire workload
- Task completion rate limited by slowest pipeline stage
- Potential speedup = Number of pipeline stages
- Unbalanced lengths of pipeline stages reduces speedup
- Time to "fill" pipeline and time to "drain" it further reduces speedup

Pipelining in a 5-stage Processor



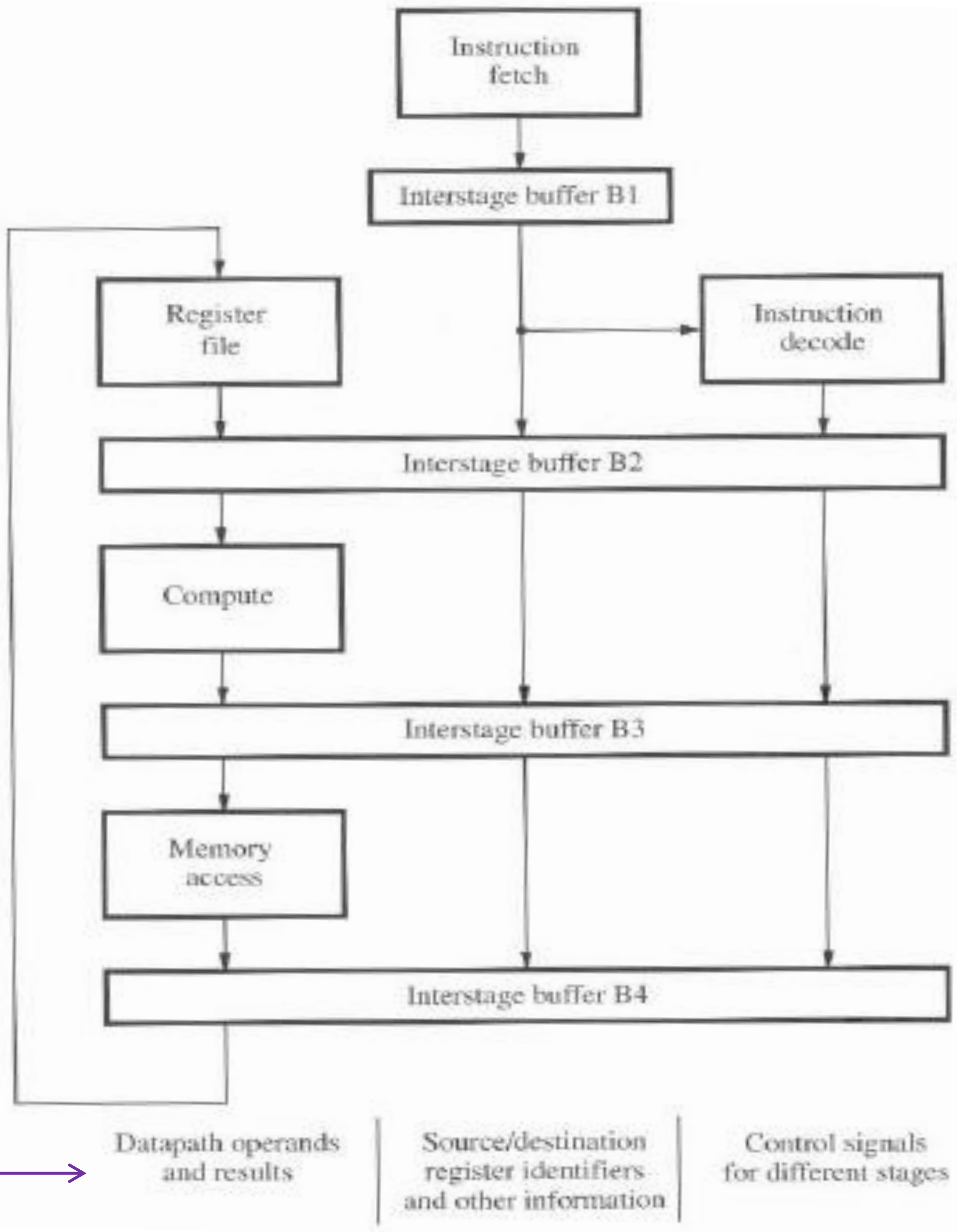
At any given time, a different instruction is being processed by each pipeline stage

How do we ensure that each stage has the correct inputs that it needs to process a particular instruction?

Information needed by an instruction is *carried through the pipeline*, as the instruction proceeds from one stage to the next

This information is held in **inter-stage buffers**

Read the details of each inter-stage buffer in Section 6.2



Information needed to process the instruction $\longrightarrow$

Pipeline Performance

- **Example:** A program consisting of 500 instructions is executed on a 5-stage processor. How many cycles would be required to complete the program, (i) without pipelining, (ii) with pipelining? Assume *ideal* overlap in case of pipelining.

- **Solution:**

  Without pipelining: Each instruction will require 5 cycles. There will be no overlap amongst successive instructions.

  Number of cycles = 500 * 5 = 2500

  With pipelining: Each pipeline stage will process a different instruction every cycle. First instruction will complete in 5 cycles, then one instruction will complete in every cycle, due to ideal overlap.

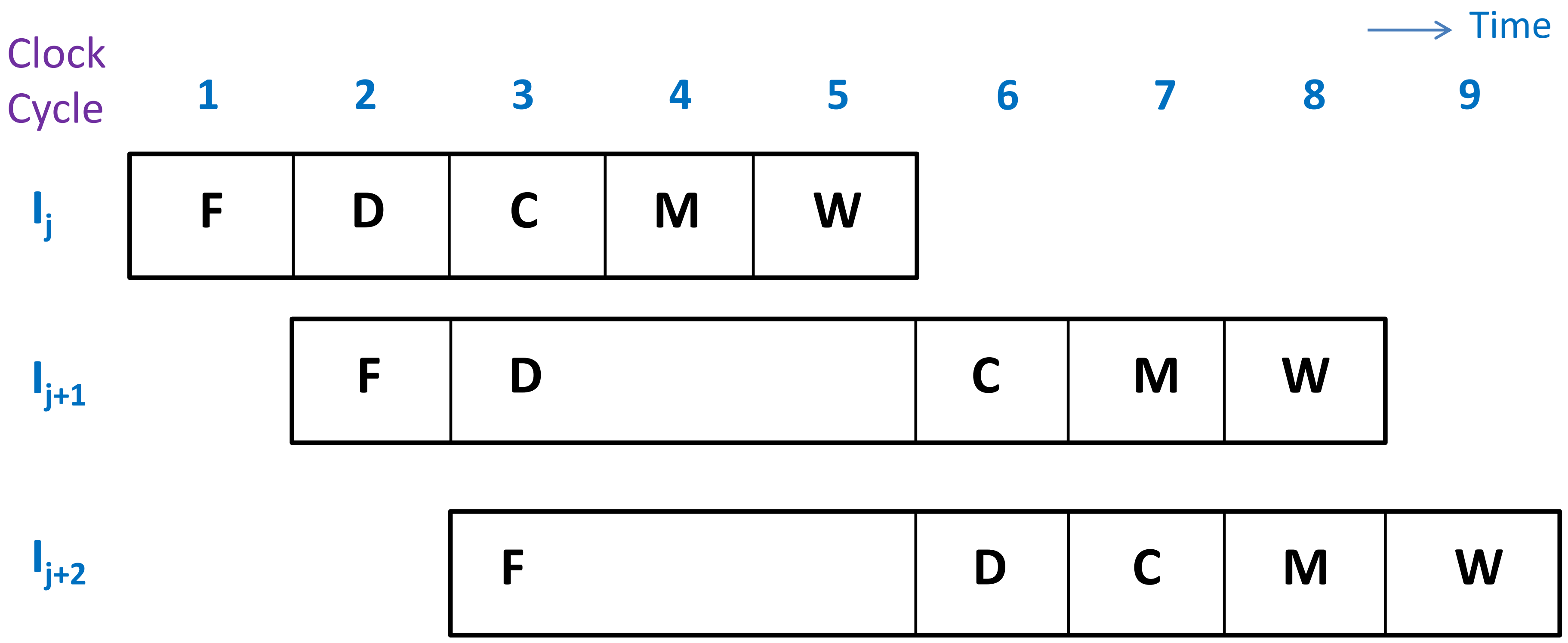
  Number of cycles = 5 + ((500-1)*1) = 504

- **Speedup for ideal pipelining = 2500/504 = 4.96 (or approx. 5)**

Pipeline Performance (cont.)

- The potential increase in performance resulting from pipelining is proportional to the number of pipeline stages
- However, this increase would be achieved only if
  - all pipeline stages require the same time to complete, and
  - there is no interruption throughout program execution
- Unfortunately, this is not true
  - there are times when an instruction cannot proceed from one stage to the next in every clock cycle

Pipeline Performance (cont.)



- Assume that Instruction I_{j+1} is *stalled* in the decode stage for two extra cycles
- This will cause I_{j+2} to be stalled in the fetch stage, until I_{j+1} proceeds
- New instructions cannot enter the pipeline until I_{j+2} proceeds past the fetch stage after cycle 5 => execution time increases by two cycles