

# Pipelining (Introduction)

CIT 595  
Spring 2008

## Laundry Example

- Ann, Brian, Cathy and Dave each have one load of clothes to wash,



- Washer takes 30 mins



- Dryer takes 40 mins



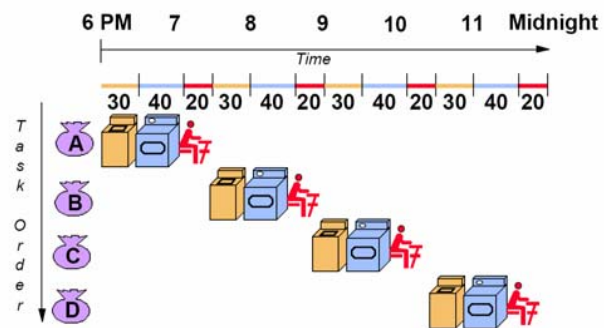
- "Folder" takes 20 mins



CIT 595

2

## Sequential Laundry

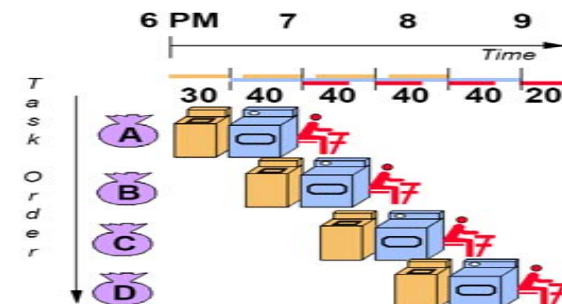


- Entire workload takes 6 hours to complete

CIT 595

3

## Pipelined Laundry



- Pipelined Laundry takes only 3.5 hours
- Speedup =  $6/3.5 = 1.7$
- Pipelining did not reduce completion time for one task but it helps the throughput of the entire workload in turn decreasing the completion time

CIT 595

4

## Instruction Level Pipelining

- In instruction processing, each instruction goes through
  - F->D->EA->OP->EX->S cycle
- The instruction cycle is divided into stages
  - One stage could contain more than one phase of the instruction cycle or
  - one phase can be divided into two stages
- If an instruction is in a particular stage of the cycle, the rest of the stages are *idle*
  - We exploit this idleness to allow instructions to be executed in parallel
  - Such parallel execution is called *instruction-level pipelining*

CIT 595

5

## Instruction Level Pipelining: Big Picture

- Each stage of the Instruction Processing Cycle takes 1 clock cycle
  - 1 clock cycle = x time units per stage
- For each stage, one phase of instruction is carried out, and the stages are overlapped



S1. Fetch instruction  
S2. Decode opcode  
S3. Evaluate Address

S4. Fetch operands  
S5. Execute  
S6. Store result

CIT 595

6

## Theoretical Speedup due to Pipelining

- The theoretical speedup offered by a pipeline can be determined as follows:
  - Let  $k$  be total number of stages and  $t_p$  be the time per stage
  - Let  $n$  be the total number of tasks
  - The first task (instruction) requires  $k \times t_p$  time to complete in a  $k$ -stage pipeline.
  - The remaining  $(n - 1)$  tasks emerge from the pipeline *one per cycle*
    - So the total time to complete the remaining tasks is  $(n - 1)t_p$
  - Thus, to complete  $n$  tasks using a  $k$ -stage pipeline requires:
 
$$(k \times t_p) + (n - 1)t_p = (k + n - 1)t_p$$

CIT 595

7

## Theoretical Speedup due to Pipelining

- If we take the time required to complete  $n$  tasks without a pipeline and divide it by the time it takes to complete  $n$  tasks using a pipeline, we find:

$$\text{Speedup } S = \frac{n t_n}{(k + n - 1) t_p} \longrightarrow t_n = k \times t_p$$

- If we take the limit as  $n$  approaches infinity,  $(k + n - 1)$  approaches  $n$ , which results in a theoretical speedup of:

$$\text{Speedup } S = \frac{k t_p}{t_p} = k$$

CIT 595

8

# LC3 Hardwired Control (modified)

The diagram illustrates the LC3 Hardwired Control (modified) architecture. It shows the following components and their interconnections:

- PC (Program Counter):** A 16-bit register that provides the address to the Memory and the instruction to the Register File.
- Memory (2<sup>nd</sup> by 16 bit):** A 16-bit memory unit that provides the instruction to the Register File and the next PC value to the PC.
- Register File:** A 16-bit register that provides the instruction to the ALU and the next PC value to the PC.
- ALU:** A 16-bit ALU that performs operations on the instruction and the next PC value. It includes a 16-bit adder (+) and a 16-bit subtracter (-).
- Control Units:** Several control units are shown, including a 16-bit adder (+1), a 16-bit subtracter (-1), and a 16-bit register (PC).

The diagram shows the flow of data and control signals between these components. The PC provides a 16-bit address to the Memory and a 16-bit instruction to the Register File. The Memory provides a 16-bit instruction to the Register File and a 16-bit next PC value to the PC. The Register File provides a 16-bit instruction to the ALU and a 16-bit next PC value to the PC. The ALU performs operations on the instruction and the next PC value. The control units provide control signals to the ALU and the PC.

Note: In this example only instructions that can be performed are the ones that update PC by 1

CIT 595

## How Pipelining actually Implemented?

- Since we are overlapping stages
- All the control information plus data must be remembered per instruction and must be carried through each stage
- This is achieved by placing a n-bit register that can hold the control/data information in between each stage

CIT 595 10

## LC3 Pipelined Implementation

- Evaluate Address and Execute are combined as they both use ALU
- Operand Fetch is separated into Register Fetch and Memory Access
- Store consists of only register writes
- Memory Write is part of Memory Access
- Thus we have a total of 6 stages

CIT 595 11

### LC3 Pipelined Implementation

The diagram illustrates the LC3 Pipelined Implementation, showing the flow of data and control signals through five pipeline stages: S1/S2, S2/S3, S3/S4, S4/S5, and S5/S6. The stages are separated by pipeline registers, which are represented by vertical bars with green segments indicating control signals.

**Legend:**

- █ Data
- █ Control

**Components and Signals:**

- PC (Program Counter):** Outputs a 16-bit address to the Memory and a 16-bit value to the S1/S2 stage.
- Memory (2<sup>11</sup> by 16 bit):** Receives a 16-bit address and outputs a 16-bit data value to the S1/S2 stage.
- Register File:** Receives a 16-bit address and outputs a 16-bit data value to the S3/S4 stage. It also receives control signals (3, 3, 3, 3) from the S2/S3 stage.
- ALU:** Receives two 16-bit data values and a 16-bit control signal, and outputs a 16-bit result to the S4/S5 stage.
- Multiplexers:** Select between different data sources (e.g., PC, Memory, Register File, ALU) to update the PC and other registers.
- Control Signals:** Indicated by green lines and green segments in the pipeline registers, showing the flow of control signals between stages.

**Stages and Data Flow:**

- S1/S2:** Receives data from the PC and Memory. It outputs a 16-bit value to the S2/S3 stage.
- S2/S3:** Receives data from the S1/S2 stage and the Register File. It outputs a 16-bit value to the S3/S4 stage.
- S3/S4:** Receives data from the S2/S3 stage and the Register File. It outputs a 16-bit value to the S4/S5 stage.
- S4/S5:** Receives data from the S3/S4 stage and the ALU. It outputs a 16-bit value to the S5/S6 stage.
- S5/S6:** Receives data from the S4/S5 stage and the ALU. It outputs a 16-bit value to the S6/S7 stage.

The diagram shows the internal structure of the processor, including the Register File, ALU, and Memory, and the flow of data and control signals through the pipeline stages.

## Impact on Clock Cycle due to Pipelining

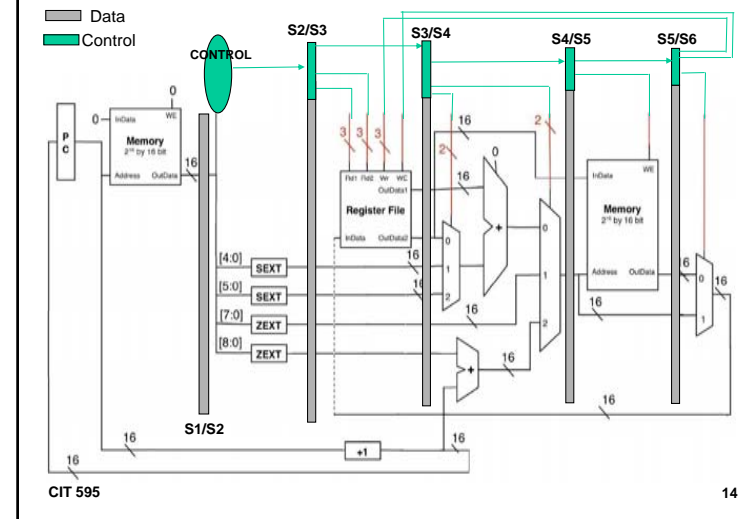
- The *clock* is sequencing the stages (instructions move in lock step fashion)
- Make sure that all work done in one stage gets done on time before it moves to next stage
- Hence, the clock cycle time should be as long as time it takes through the longest pipe stage
  - This also includes the time for capturing data into registers in between stages

$$\text{Clock cycle time} = \max(t_1, t_2, t_3, t_4, t_5, t_6)$$

CIT 595

13

## LC3 Pipelined Implementation



CIT 595

14

## Impact on Clock cycle time due to Pipelining

- Recall

$$\text{CPU Time} = \frac{\text{time}}{\text{program}} = \frac{\text{time}}{\text{cycle}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{instructions}}{\text{program}}$$

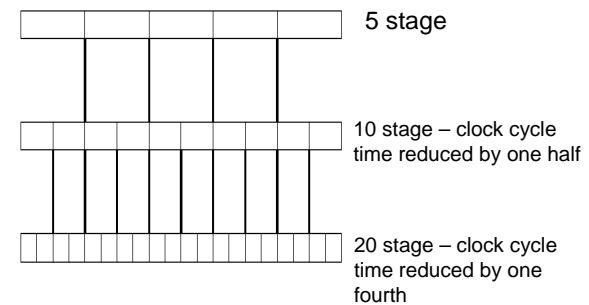
Clock Cycle time
CPI
Instruction Count

- If we lower the time per cycle, this will lower the program execution time and hence improve performance
- This implies that we if we shorten the time per pipeline stages, we will lower clock cycle time
  - This can be achieved by adding more pipe stages of shorter duration

CIT 595

15

## Impact on Clock cycle time due to Pipelining

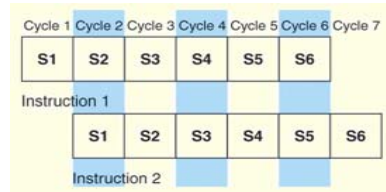


CIT 595

16

### Cycles Per Instruction (CPI) with Pipelining

- In pipelining, one instruction is in each stage
- Since one instruction will be fetched (or finish) each cycle, the average CPI will equal 1 (obviously we are ignoring the very first instruction – cold start)



- However, CPI = 1 is barely achieved

CIT 595

17

### Why CPI is not always 1?

- We assume that the pipeline can be kept filled at all times
- However, this is not always the case
- The situations that cause pipeline not to be filled at all times arises due to what is known as *Pipeline Hazards*

CIT 595

18