Computer Organization

Douglas Comer

Computer Science Department
Purdue University
250 N. University Street
West Lafayette, IN 47907-2066

http://www.cs.purdue.edu/people/comer

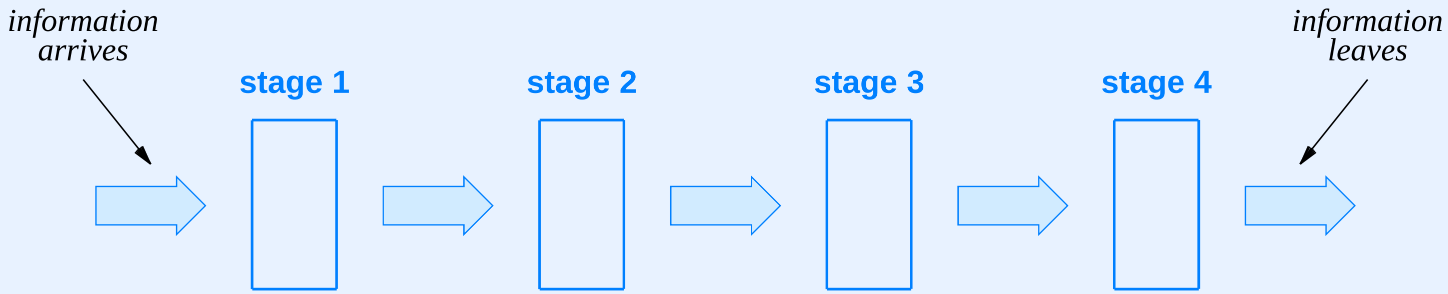
XVIII

Pipelining

Concept Of Pipelining

- One of the two major hardware optimization techniques
- Information flows through a series of stations (processing components)
- Each station can
  - Inspect
  - Interpret
  - Modify

Illustration Of Pipelining



Characteristics Of Pipelines

- Hardware or software implementation
- Large or small scale
- Synchronous or asynchronous flow
- Buffered or unbuffered flow
- Finite chunks or continuous bit streams
- Automatic data feed or manual data feed
- Serial or parallel path
- Homogeneous or heterogeneous stages

Implementation

- Pipeline can be implemented in hardware or software
- Software pipeline
  - Programmer convenience
  - More efficient than intermediate files
- Hardware pipeline
  - Much higher performance

Scale

- Range of scales
- Example of small scale: pipeline within an ALU
- Example of large scale: pipeline composed of programs running on separate computers connected by the Internet

Synchrony

- Synchronous pipeline
  - Operates like an assembly line
  - Items move at exactly the same time
- Asynchronous pipeline
  - Each station forwards whenever it is ready
  - Slow stage may block previous stages

Buffering

- Buffered flow
  - Buffer placed between each pair of stages
  - Useful when processing time per item varies
- Unbuffered flow
  - Stage blocks until next stage can accept item
  - Works best if processing time per stage is constant

Size Of Items

- Finite chunks
  - Discrete items pass through pipeline
  - Example: sequence of Ethernet packets
- Continuous bit stream
  - Stream of bits flows through pipeline
  - Example: video feed

Data Feed Mechanism

- Automatic
  - Built into pipeline
- Manual
  - Separate hardware to move items

Width Of Data Path

- Serial
  - One bit at a time
- Parallel
  - *N* bits at a time

Homogeneity Of Stages

- Homogeneous
  - All stages are the same
  - Example: five identical processors
- Heterogeneous
  - Stages can differ
  - Example: each stage optimized for one function

Software Pipelining

- Popularized by Unix command interpreter (shell)
- User can specify pipeline as a command
- Example

```
cat x | sed 's/friend/partner/g' | more
```

Software Pipeline Performance And Overhead

- Consider the pipeline

  cat x | sed ’s/friend/partner/g’ | sed ’/W/d’ | more

- Substitutes ‘‘partner’’ for ‘‘friend’’
- Deletes lines that contain ‘‘W’’
- Can be optimized (swap sed commands)

Implementation Of Software Pipeline

- Uniprocessor
  - Each stage is a *process* or *task*
- Multiprocessor
  - Each stage executes on separate processor
  - Hardware assist can speed inter-stage data transfer

Hardware Pipelining

- Two broad categories
  - Instruction pipeline
  - Data pipeline

Instruction Pipeline

- Covered in Chapter 5
- Recall
  - Instruction processed in stages
  - Exact details and number of stages depend on instruction set and operand types

Data Pipeline

- Data passes through pipeline
- Each stage handles data item and passes item to next stage
- Usually requires programmer to divide code into stages
- Among the most interesting uses of pipelining

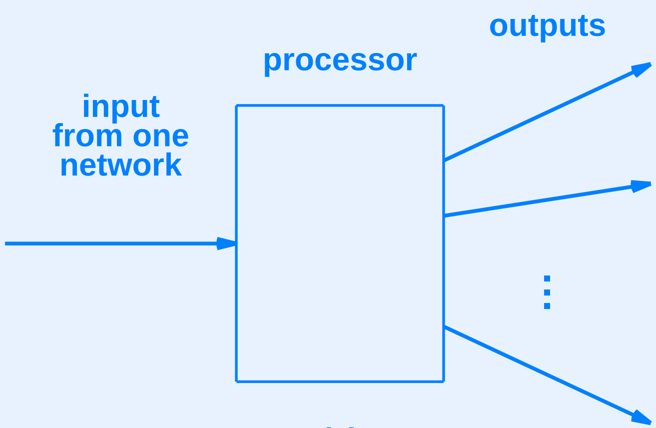
Hardware Pipelining And Performance

- A data pipeline can dramatically increase performance (throughput)
- To see why, consider an example
  - Internet router handles packets
  - Assume
    - Router processes one packet at at time
    - Performs six functions on a packet

Example Of Internet Router Algorithm

1. Receive a packet (i.e., transfer the packet into memory).
2. Verify packet integrity (i.e., verify that no changes occurred between transmission and reception).
3. Check for routing loops (i.e., decrement a value in the header, and reform the header with the new value).
4. Route the packet (i.e., use the destination address field to select one of the possible output networks and a destination on that network).
5. Prepare for transmission (i.e. compute information that will be used to verify packet integrity).
6. Transmit the packet (i.e., transfer the packet to the output device).

Illustration Of A Processor In A Router And The Algorithm Used

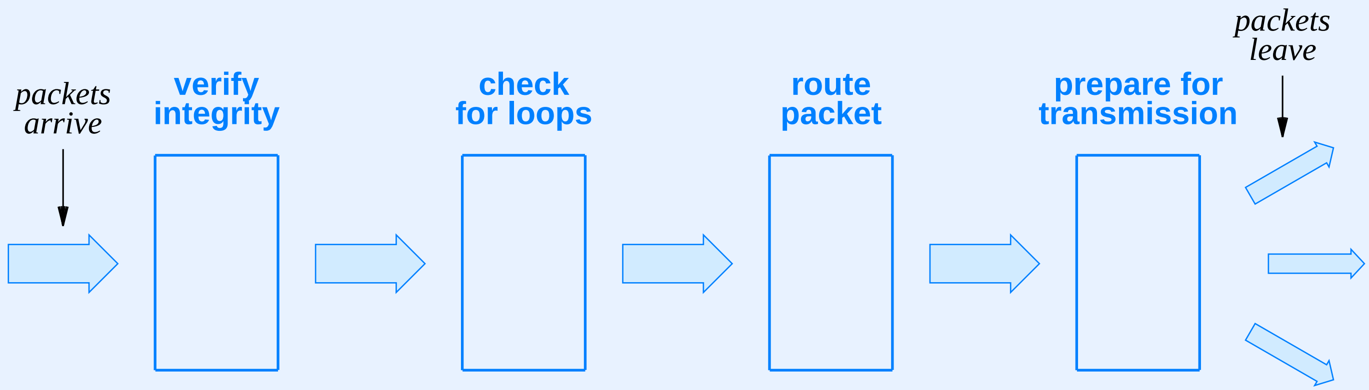


(a)

```
do forever  {
      Wait to receive packet
      Verify integrity
      Check for loops
      Route packet
      Prepare for transmission
      Enqueue packet for output
}
```

(b)

A Pipeline Implementation Of A Processor In A Router



The Bad News

A data pipeline passes data through a series of stages that each examine or modify the data. If it uses the same speed processors as a nonpipeline architecture, a data pipeline will not improve the overall time needed to process a given data item.

The Good News

Even if a data pipeline uses the same speed processors as a nonpipeline architecture, a data pipeline has higher overall throughput (i.e., number of data items processed per second).

Pipelining Can Only Be Used If

- It is possible to partition processing into independent stages
- Overhead required to move data from one stage to another is insignificant
- The slowest stage of the pipeline is faster than a single processor

Understanding Pipeline Speed

- Assume
  - The task is packet processing
  - Processing a packet requires exactly 500 instructions
  - A processor executes 10 instructions per μsec
- Total time required for one packet is:

$$time = \frac{500\ instructions}{10\ instr.\ per\ \mu sec} = 50\ \mu sec$$

- Throughput for a non-piplined system is:

$$T_{np} = \frac{1\ packet}{50\ \mu sec} = \frac{1\ packet \times 10^6}{50\ sec} = 20{,}000\ packets\ per\ second$$

Understanding Pipeline Speed
(continued)

- Suppose the problem can be divided into four stages and that the stages require:
  - 50 instructions
  - 100 instructions
  - 200 instructions
  - 150 instructions
- The slowest stage takes 200 instructions
- So, the time required for the slowest stage is:

$$\textit{total time} = \frac{200 \textit{ inst}}{10 \textit{ inst} / \mu \textit{sec}} = 20 \ \mu \textit{sec}$$

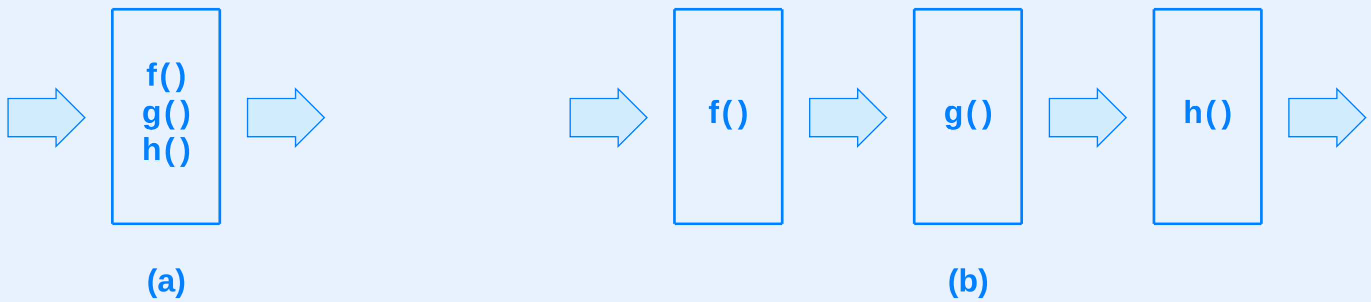
Understanding Pipeline Speed
(continued)

- Throughput of the pipeline is limited by the slowest stage
- Overall throughput can be calculated:

$$T_p = \frac{1\ packet}{20\ \mu sec} = \frac{1\ packet \times 10^6}{20\ sec} = 50,000\ packets\ per\ second$$

- Note: throughput of pipelined version is 150% greater than throughput of the non-pipelined version!

Conceptual Division Of Processing



- (a) shows a single processor handling all functions
- (b) shows processing divided into a pipeline

Pipeline Architectures

- Refer to architectures that are primarily formed around pipelining
- Most often used for special-purpose systems
- Less relevant to general-purpose computers

Pipeline Setup, Stall, And Flush

- Setup time
  - Refers to time required to start the pipeline initially
- Stall time
  - Refers to time required to restart the pipeline after a stage blocks to wait for a previous stage
- Flush time
  - Refers to time that elapses between the cessation of input and the final data item emerging from the pipeline (i.e., the time required to shut down the pipeline)

Superpipelining

- Most often used with instruction pipelining
- Subdivides a stage into smaller stages
- Example: subdivide operand processing into
  - Operand decode
  - Fetch immediate value or value from register
  - Fetch value from memory
  - Fetch indirect operand

Summary

- Pipelining
  - Broad, fundamental concept
  - Can be used with hardware or software
  - Applies to instructions or data
  - Can be synchronous or asynchronous
  - Can be buffered or unbuffered

Summary
(continued)

- Pipeline performance
  - Unless faster processors are used, data pipelining does not decrease the overall time required to process a single data item
  - Using a pipeline does increase the overall throughput (items processed per second)
  - The stage of a pipeline that requires the most time to process an item limits the throughput of the pipeline

Questions?