6.004 Computation Structures
Spring 2009

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.
Pipelining
what Seymour Cray taught the laundry industry

"I've got 3 months Worth of laundry To do tonight…"

"Funny, considering that he's only got one outfit…"

Due Thursday: Lab #3

Forget circuits… let's solve a “Real Problem”

**INPUT:**
dirty laundry

**Device:** Washer
**Function:** Fill, Agitate, Spin
**Washer** \( PD = 30 \text{ mins} \)

**Device:** Dryer
**Function:** Heat, Spin
**Dryer** \( PD = 60 \text{ mins} \)

**OUTPUT:**
6 more weeks

One load at a time

Everyone knows that the real reason that MIT students put off doing laundry so long is not because they procrastinate, are lazy, or even have better things to do.

The fact is, doing one load at a time is not smart.

Doing \( N \) loads of laundry

Here’s how they do laundry at Harvard, the “combinational” way.

(Of course, this is just an urban legend. No one at Harvard actually does laundry. The butlers all arrive on Wednesday morning, pick up the dirty laundry and return it all pressed and starched in time for afternoon tea)

\[
\text{Total} = \text{Washer}_PD + \text{Dryer}_PD = \boxed{90} \text{ mins}
\]
Doing N Loads… the MIT way

MIT students “pipeline” the laundry process.

That’s why we wait!

\[
\text{Total} = N \times \text{Max(Washer}_{PD}, \text{Dryer}_{PD})
\]

Actually, it’s more like \( N \times 60 + 30 \) if we account for the startup transient correctly. When doing pipeline analysis, we’re mostly interested in the “steady state” where we assume we have an infinite supply of inputs.

Performance Measures

Latency:
The delay from when an input is established until the output associated with that input becomes valid.

\[
\begin{align*}
\text{(Harvard Laundry} &= \frac{90}{60} \text{ mins}) \\
\text{(MIT Laundry} &= \frac{120}{60} \text{ mins})
\end{align*}
\]

Throughput:
The rate at which inputs or outputs are processed.

\[
\begin{align*}
\text{(Harvard Laundry} &= \frac{1}{90} \text{ outputs/min)} \\
\text{(MIT Laundry} &= \frac{1}{60} \text{ outputs/min)}
\end{align*}
\]

Okay, back to circuits…

For combinational logic:
\[
\text{latency} = t_{PD}, \quad \text{throughput} = \frac{1}{t_{PD}}.
\]

We can’t get the answer faster, but are we making effective use of our hardware at all times?

Pipelined Circuits

use registers to hold H’s input stable!

Now F & G can be working on input \( X_{i+1} \) while H is performing its computation on \( X_i \). We’ve created a 2-stage pipeline if we have a valid input \( X \) during clock cycle \( j \), \( P(X) \) is valid during clock \( j+2 \).

Suppose F, G, H have propagation delays of 15, 20, 25 ns and we are using ideal zero-delay registers:

\[
\begin{align*}
\text{unpipelined} & \quad \text{latency} & \quad \text{throughput} \\
\text{2-stage pipeline} & \quad 45 & \quad 1/45 \\
\text{worse} & \quad 50 & \quad 1/25 \\
\text{better} & \quad 25 & \quad 1/25
\end{align*}
\]
Pipeline diagrams

Clock cycle

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>i+1</th>
<th>i+2</th>
<th>i+3</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>$X_i$</td>
<td>$X_{i+1}$</td>
<td>$X_{i+2}$</td>
<td>$X_{i+3}$</td>
<td></td>
</tr>
<tr>
<td>F Reg</td>
<td>$F(X_i)$</td>
<td>$F(X_{i+1})$</td>
<td>$F(X_{i+2})$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G Reg</td>
<td>$G(X_i)$</td>
<td>$G(X_{i+1})$</td>
<td>$G(X_{i+2})$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H Reg</td>
<td>$H(X_i)$</td>
<td>$H(X_{i+1})$</td>
<td>$H(X_{i+2})$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The results associated with a particular set of input data moves diagonally through the diagram, progressing through one pipeline stage each clock cycle.

Pipeline Conventions

**DEFINITION:**

A **K-Stage Pipeline** ("K-pipeline") is an acyclic circuit having exactly K registers on every path from an input to an output.

A **COMBINATIONAL CIRCUIT** is thus an O-stage pipeline.

**CONVENTION:**

Every pipeline stage, hence every K-Stage pipeline, has a register on its **OUTPUT** (not on its input).

**ALWAYS:**

The CLOCK common to all registers must have a period sufficient to cover propagation over combinational paths PLUS (input) register $t_{PD}$ PLUS (output) register $t_{SETUP}$.

The **LATENCY** of a K-pipeline is K times the period of the clock common to all registers.

The **THROUGHPUT** of a K-pipeline is the frequency of the clock.

Ill-formed pipelines

Consider a BAD job of pipelining:

For what value of K is the following circuit a K-Pipeline? **ANS: none**

**Problem:**

Successive inputs get mixed, e.g., $B(A(X_i), Y)$. This happened because some paths from inputs to outputs have 2 registers, and some have only 1!

This CAN'T HAPPEN on a well-formed K-pipeline!

A pipelining methodology

**Step 1:**

Draw a line that crosses every output in the circuit, and mark the endpoints as terminal points.

**STRATEGY:**

Focus your attention on placing pipelining registers around the slowest circuit elements (BOTTLENECKS).

**Step 2:**

Continue to draw new lines between the terminal points across various circuit connections, ensuring that every connection crosses each line in the same direction. These lines demarcate pipeline stages.

Adding a pipeline register at every point where a separating line crosses a connection will always generate a valid pipeline.
**Pipeline Example**

**OBSERVATIONS:**
- 1-pipeline improves neither L or T.
- T improved by breaking long combinational paths, allowing faster clock.
- Too many stages cost L, don’t improve T.
- Back-to-back registers are often required to keep pipeline well-formed.

<table>
<thead>
<tr>
<th></th>
<th>LATENCY</th>
<th>THROUGHPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-pipe</td>
<td>4</td>
<td>1/4</td>
</tr>
<tr>
<td>1-pipe</td>
<td>4</td>
<td>1/4</td>
</tr>
<tr>
<td>2-pipe</td>
<td>4</td>
<td>1/2</td>
</tr>
<tr>
<td>3-pipe</td>
<td>6</td>
<td>1/2</td>
</tr>
</tbody>
</table>

**Pipelining Summary**

**Advantages:**
- Allows us to increase throughput by breaking up long combinational paths and (hence) increasing clock frequency

**Disadvantages:**
- May increase latency...
- Only as good as the weakest link: slowest step constrains system throughput.

Isn’t there a way around this “weak link” problem?

**Pipelined Components**

Pipelined systems can be hierarchical:
- Replacing a slow combinational component with a k-pipe version may increase clock frequency
- Must account for new pipeline stages in our plan

4-stage pipeline, throughput = 1

**How do 6.004 Aces do Laundry?**

They work around the bottleneck.
First, they find a place with twice as many dryers as washers.

**Throughput** = \(\frac{1}{30}\) loads/min

Latency = 90 mins/load

Figure by MIT OpenCourseware.
Back to our bottleneck...

Recall our earlier example...

- $C$ – the slowest component – limits clock period to $\beta$ ns.
- HENCE throughput limited to $1/\beta$ns.

We could improve throughput by

- Finding a pipelined version of $C$;
- OR...
- interleaving multiple copies of $C$!

Circuit Interleaving

We can simulate a pipelined version of a slow component by replicating the critical element and alternate inputs between the various copies.

This is a simple 2-state FSM that alternates between 0 and 1 on each clock

Circuit Interleaving

We can simulate a pipelined version of a slow component by replicating the critical element and alternate inputs between the various copies.

When $Q$ is 1 the lower path is combinational (the latch is open), yet the output of the upper path will be enabled onto the input of the output register ready for the NEXT clock edge.

Meanwhile, the other latch maintains the input from the last clock.

2-Clock Martinizing

"In by $t_i$, out by $t_{i+2}$”

N-way interleaving is equivalent to

N pipeline Stages...

Latency = 2 clocks

- Clock period 0: $X_i$ presented at input, propagates thru upper latch, $C_0$.
- Clock period 1: $X_i$ presented at input, propagates thru lower latch, $C_1$. $C_0(X_0)$ propagates to register inputs.
- Clock period 2: $X_i$ presented at input, propagates thru upper latch. $C_0(X_0)$ loaded into register, appears at output.
Combining techniques

We can combine interleaving and pipelining. Here, C' interleaves two C elements with a propagation delay of 8nS.

The resulting C' circuit has a throughput of 1/4 nS, and latency of 8 nS. This can be considered as an extra pipelining stage that passes through the middle of the C' module. One of our separation lines must pass through this pipeline stage.

By combining interleaving with pipelining we move the bottleneck from the C element to the F element.

A

B

C'

D

E

F

T = 1/5ns
L = 25ns

And a little parallelism...

We can combine interleaving and pipelining with parallelism.

Throughput = \( \frac{2}{30} = \frac{1}{15} \) load/min
Latency = 90 min

Control Structure Approaches

Synchronous
ALL computation “events” occur at active edges of a periodic clock: time is divided into fixed-size discrete intervals.

Globally Timed
Timing dictated by centralized FSM according to a fixed schedule.

Locally Timed
Each module takes a START signal, generates a FINISHED signal. Timing is dynamic, data dependent.

Asynchronous
Events -- eg the loading of a register -- can happen at at arbitrary times.

Control Structure Alternatives

Synchronous, globally-timed: Control signals (e.g., load enables) From FSM controller

Synchronous, locally-timed: Local circuitry, “handshake” controls flow of data:

Asynchronous, locally-timed system using transition signaling:
Self-timed Example

A glimpse of an asynchronous, locally-time discipline

Elegant, timing-independent design:
- Each component specifies its own time constraints
- Local adaptation to special cases (e.g., multiplication by 0)
- Module performance improvements automatically exploited
- Can be made asynchronous (no clock at all!) or synchronous

Control Structure Taxonomy

Synchronous
- Globally Timed
  - Centralized clocked FSM generates all control signals.
- Locally Timed
  - Start and Finish signals generated by each major subsystem, synchronously with global clock.

Asynchronous
- Central control unit tailors current time slice to current tasks.

Summary

- Latency (L) = time it takes for given input to arrive at output
- Throughput (T) = rate at each new outputs appear
- For combinational circuits: L = t_{PD} of circuit, T = 1/L
- For K-pipelines (K > 0):
  - always have register on output(s)
  - K registers on every path from input to output
  - Inputs available shortly after clock i, outputs available shortly after clock (i+K)
  - T = 1/(t_{PD,REG} + t_{PD} of slowest pipeline stage + t_{SETUP})
    - more throughput → split slowest pipeline stage(s)
    - use replication/interleaving if no further splits possible
  - L = K / T
    - pipelined latency ≥ combinational latency