Bionic Eyes let blind see! ⇒

Johns Hopkins researchers have announced they have invented a “bionic eye” with a computer chip on the back of the eye and a small wireless video camera in a pair of glasses. Face recognition? Not yet, but soon!

news.bbc.co.uk/1/hi/health/4411591.stm
5 steps to design a processor

1. Analyze instruction set => datapath requirements
2. Select set of datapath components & establish clock methodology
3. Assemble datapath meeting the requirements
4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
5. Assemble the control logic

Control is the hard part

MIPS makes that easier

- Instructions same size
- Source registers always in same place
- Immediates same size, location
- Operations always on registers/immediates
Review Datapath

Datapath is the hardware that performs operations necessary to execute programs.

Control instructs datapath on what to do next.

Datapath needs:
  • access to storage (general purpose registers and memory)
  • computational ability (ALU)
  • helper hardware (local registers and PC)
Review Datapath (2/3)

- Five stages of datapath (executing an instruction):
  1. Instruction Fetch (Increment PC)
  2. Instruction Decode (Read Registers)
  3. ALU (Computation)
  4. Memory Access
  5. Write to Registers

- ALL instructions must go through ALL five stages.
Review Datapath

1. Instruction Fetch
2. Decode/Register Read
3. Execute
4. Memory
5. Write Back
Gotta Do Laundry

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

- 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

- Pipelined laundry takes 3.5 hours for 4 loads!
General Definitions

- **Latency**: time to completely execute a certain task
  - for example, time to read a sector from disk is disk access time or disk latency

- **Throughput**: amount of work that can be done over a period of time
Pipelining Lessons
(1/2)

• Pipelining doesn’t help latency of single task, it helps throughput of entire workload

• Multiple tasks operating simultaneously using different resources

• Potential speedup = Number pipe stages

• Time to “fill” pipeline and time to “drain” it reduces speedup: 2.3X v. 4X in this example

6 PM 7 8 9

Task Order

Time

30 30 30 30 30 30 30

T a s k

O r d e r

B

C

D
Pipelining Lessons (2/2)

- Suppose new Washer takes 20 minutes, new Stasher takes 20 minutes. How much faster is pipeline?

- Pipeline rate limited by slowest pipeline stage

- Unbalanced lengths of pipe stages also reduces speedup
Steps in Executing MIPS

1) **IFetch**: Fetch Instruction, Increment PC

2) **Decode** Instruction, Read Registers

3) **Execute**:
   - Mem-ref: Calculate Address
   - Arith-log: Perform Operation

4) **Memory**:
   - Load: Read Data from Memory
   - Store: Write Data to Memory

5) **Write Back**: Write Data to Register
Pipelined Execution Representation

- Every instruction must take the same number of steps, also called pipeline "stages", so some will go idle sometimes.
Review: Datapath for MIPS

1. Instruction Fetch
2. Decode/Register Read
3. Execute
4. Memory
5. Write Back

- Use datapath figure to represent pipeline
Graphical Pipeline Representation
(In Reg, right half highlight read, left half write)

Time (clock cycles)

Instr. Order

Load
Add
Store
Sub
Or
Example

• Suppose 2 ns for memory access, 2 ns for ALU operation, and 1 ns for register file read or write; compute instr rate

• Nonpipelined Execution:
  • lw : IF + Read Reg + ALU + Memory + Write Reg = 2 + 1 + 2 + 2 + 1 = 8 ns
  • add: IF + Read Reg + ALU + Write Reg = 2 + 1 + 2 + 1 = 6 ns

• Pipelined Execution:
  • Max(IF,Read Reg,ALU,Memory,Write Reg) = 2 ns
Pipeline Hazard: Matching socks in later load

A depends on D; **stall** since folder tied up
Administrivia

• Any administration?
Problems for Computers

• Limits to pipelining: **Hazards** prevent next instruction from executing during its designated clock cycle

  • **Structural hazards**: HW cannot support this combination of instructions (single person to fold and put clothes away)

  • **Control hazards**: Pipelining of branches & other instructions **stall** the pipeline until the hazard; “**bubbles**” in the pipeline

  • **Data hazards**: Instruction depends on result of prior instruction still in the pipeline (missing sock)
Structural Hazard #1: Single Memory

Read same memory twice in same clock cycle
Structural Hazard #1: Single Memory (2/2)

• Solution:
  • infeasible and inefficient to create second memory
  • (We’ll learn about this more next week)
  • so simulate this by having two Level 1 Caches (a temporary smaller [of usually most recently used] copy of memory)
  • have both an L1 Instruction Cache and an L1 Data Cache
  • need more complex hardware to control when both caches miss
Structural Hazard #2: Registers

Can’t read and write to registers simultaneously
Structural Hazard #2: Registers (2/2)

• Fact: Register access is *VERY* fast: takes less than half the time of ALU stage

• Solution: introduce convention
  • always Write to Registers during first half of each clock cycle
  • always Read from Registers during second half of each clock cycle

• Result: can perform Read and Write during same clock cycle
A. Thanks to pipelining, I have **reduced the time** it took me to wash my shirt.

B. Longer pipelines are **always a win** (since less work per stage & a faster clock).

C. We can **rely on compilers** to help us avoid data hazards by reordering instrs.

<table>
<thead>
<tr>
<th></th>
<th>ABC</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FFF</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>FFT</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>FTF</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>FTT</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>TFF</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>TFT</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>TTF</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>TTT</td>
<td></td>
</tr>
</tbody>
</table>
A. Throughput better, not execution time

B. “...longer pipelines do usually mean faster clock, but branches cause problems!”

C. “they happen too often & delay too long.”

Forwarding! (e.g, Mem ⇒ ALU)

A. Thanks to pipelining, I have reduced the time it took me to wash my shirt. FALSE

B. Longer pipelines are always a win (since less work per stage & a faster clock). FALSE

C. We can rely on compilers to help us avoid data hazards by reordering instrs. FALSE
Things to Remember

• Optimal Pipeline
  • Each stage is executing part of an instruction each clock cycle.
  • One instruction finishes during each clock cycle.
  • On average, execute far more quickly.

• What makes this work?
  • Similarities between instructions allow us to use same stages for all instructions (generally).
  • Each stage takes about the same amount of time as all others: little wasted time.