Pipelining
Pipelining

What’s the latency for one unit of work?

What’s the throughput?
Pipelining

1. Break up the logic with latches into “pipeline stages”
2. Each stage can act on different data
3. Latches hold the inputs to their stage
4. Every clock cycle data transfers from one pipe stage to the next
What's the latency for one unit of work? What's the throughput?
Critical path review

- Critical path is the longest possible delay between two registers in a design.
- The critical path sets the cycle time, since the cycle time must be long enough for a signal to traverse the critical path.
- Lengthening or shortening non-critical paths does not change performance.
- Ideally, all paths are about the same length.
Pipelining and Logic

• Hopefully, critical path reduced by 1/3
Limitations of Pipelining

- You cannot pipeline forever
  - Some logic cannot be pipelined arbitrarily -- Memories
  - Some logic is inconvenient to pipeline.
  - How do you insert a register in the middle of an multiplier?
- Registers have a cost
  - They cost area -- choose “narrow points” in the logic
  - They cost energy -- latches don’t do any useful work
  - They cost time
    - Extra logic delay
    - Set-up and hold times.
- Pipelining may not affect the critical path as you expect
You can’t always pipeline how you would like
The critical path only went down “fast logic”
How to pipeline a processor

- The “classic 5-stage MIPS pipeline”
  - Fetch -- read the instruction
  - Decode -- decode and read from the register file
  - Execute -- Perform arithmetic ops and address calculations
  - Memory -- access data memory.
  - Write back-- Store results in the register file.
Pipelining a processor
Pipelined Datapath

Instruction Memory
- Read Address
- Read Addr 1
- Read Addr 2
- Write Addr
- Write Data

Register File
- Read Data 1
- Read Data 2

ALU
- Shift left 2
- Add

Data Memory
- Address
- Read Data
- Write Data

Sign Extend
- 16
- 32
Pipelined Datapath

Instruction Memory
- Read Address
- Read Addr 1
- Read Addr 2
- Write Addr
- Write Data

Register File
- Read Data 1
- Read Data 2

Data Memory
- Address
- Read Data
- Write Data

ALU
- Add
- Shift left 2

PC

Immediate/Dec

Dec/Exec

Exec/Mem

Mem/WB
Pipelined Datapath

Instruction Memory

Register File

ALU

Data Memory

- Read Address
- Read Addr 1
- Read Addr 2
- Write Addr
- Write Data
- Read Data 1
- Read Data 2
- Add
- Shift left 2
- Add

- Address
- Read Data
- Write Data

- Add ...
- lw ...
- Sub ...
- Sub ....
- Add ...
- Add ...
Pipelined Datapath

Instruction Memory
  Read Address

Register File
  Read Addr 1
  Read Addr 2
  Write Addr
  Write Data

ALU
  Read Data 1
  Read Data 2
  Add
  Shift left 2
  Add

Data Memory
  Address
  Read Data
  Write Data

Add
  4

Sign Extend
  16
  32

add ...

lw ...

Sub ...

Sub ....

Add ...

Add ...
Pipelined Datapath

Instruction Memory

Register

File

Data Memory

Read Address

Write Addr

Write Data

Read Addr 1

Read Addr 2

Read Data 1

Read Data 2

Add

Shift left 2

Add

ALU

Address

Read Data

Write Data

16

32

Sign Extend

add ...
lw ...
Sub...
Sub ....
Add ...
Add ...
Pipelined Datapath

- Instruction Memory
  - Read Address
- Read Addr 1
- Read Addr 2
  - File
  - Read Data 1
  - Read Data 2
- Write Addr
- Write Data

- ALU
  - Add
  - Shift left 2
- Data Memory
  - Address
  - Read Data
- Write Data

- Sign
- Extend

- 16
- 32

- add ...
- lw ...
- Subi ...
- Sub ....
- Add ...
- Add ...
Pipelining is Tricky

- If all the data flows in one direction, pipelining is relatively easy.
- Not so, for processors.
  - Decode and write back both access the register file.
  - Branch instructions affect the next PC
  - Instructions need values computed by previous instructions
Not just tricky, Hazardous!

• Hazards are situations where pipelining does not work as elegantly as we would like
  • Caused by backward flowing signals
  • Or by lack of available hardware

• Three kinds
  • Data hazards -- an input is not available on the cycle it is needed
  • Control hazards -- the next instruction is not known
  • Structural hazards -- we have run out of a hardware resource

• Detecting, avoiding, and recovering from these hazards is what makes processor design hard.
  • That, and the Xilinx tools ;-)