Processor Design — Pipelined Processor

Hung-Wei Tseng
Outline

- Pipelining
- Designing a 5-stage pipeline processor for MIPS
**Drawbacks of a single-cycle processor**

- The cycle time is determined by the longest instruction
  - Could be very long, thinking about fetch data from DRAM
- Hardware is mostly idle
Pipelining
Pipelining

• Break up the logic with “pipeline registers” into pipeline stages
• Each stage can act on different instruction/data
• States/Control Signals of instructions are hold in pipeline registers (latches)
Pipelining

cycle #1

cycle #2

cycle #3

cycle #4

cycle #5
Single-cycle v.s. pipeline v.s.
Performance of a pipeline processor

- If we have 500 instructions, what's the speedup of a pipeline processor with 2 ns cycle time v.s. a single-cycle processor with 10 ns cycle time?

A. 5
B. 4.96
C. 2.78
D. 1
E. None of the above
Cycle time of a pipeline processor

- 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.
Limitations of pipelining

• How many of the following descriptions about pipelining is correct?
  • You can always divide stages into short stages with latches
  • Pipeline registers incur overhead for each pipeline stage
  • The latency of executing an instruction in a pipeline processor is longer than a single-cycle processor
  • The throughput of a pipeline processor is usually better than a single-cycle processor
  • Pipelining a stage can always improve cycle time

A. 1
B. 2
C. 3
D. 4
E. 5
Designing a 5-stage pipeline processor for MIPS
Basic steps of execution

• Instruction fetch: fetch an instruction from memory
• Decode:
  • What’s the instruction?
  • Where are the operands?
• Execute
• Memory access
  • Where is my data? (The data memory address)
• Write back
  • Where to put the result
• Determine the next PC
Pipeline a MIPS processor

1. Instruction Fetch
   - Read the instruction
2. Decode
   - Figure out the incoming instruction?
   - Fetch the operands from the register file
3. Execution: ALU
   - Perform ALU functions
4. Memory access
   - Read/write data memory
5. Write back results to registers
   - Write to register file
From single-cycle to pipeline

Instruction Fetch
- PCSrc = Branch & Zero

Instruction Decode
- Control

Execution
- ALU
- RegWrite
- Shift left 2

Memory Access
- Data Memory
- ALUSrc
- Zero

Write Back
- MemWrite
- MemtoReg

Will this work?
Pipelined processor

add $1, $2, $3
lw  $4, 0($5)
sub $6, $7, $8
sub $9,$10,$11
sw  $1, 0($12)
Pipelined processor

add $1, $2, $3
lw $4, 0($5)
sub $6, $7, $8
sub $9,$10,$11
sw $1, 0($12)
Pipelined processor

Where can I find these?

add $1, $2, $3
lw  $4, 0($5)
sub $6, $7, $8
sub $9,$10,$11
sw  $1, 0($12)
Pipelined processor

add $1, $2, $3
lw  $4, 0($5)
sub $6, $7, $8
sub $9,$10,$11
sw  $1, 0($12)
Pipelined processor

Is this right?

add $1, $2, $3
lw $4, 0($5)
sub $6, $7, $8
sub $9,$10,$11
sw $1, 0($12)
Pipelined processor
5-stage pipelined processor
Simplified pipeline diagram

- Use symbols to represent the physical resources with the abbreviations for pipeline stages.
  - IF, ID, EXE, MEM, WB
- Horizontal axis represent the timeline, vertical axis for the instruction stream
- Example:
  
  ```
  add $1, $2, $3  
  lw  $4, 0($5)  
  sub $6, $7, $8  
  sub $9,$10,$11  
  sw  $1, 0($12)  
  ```
What if ...

Will this work?

Pipeline hazards

add $1, $2, $3
lw $4, 0($1)
sub $6, $7, $8
sub $9,$10,$11
sw $1, 0($12)
Pipeline hazards
Pipeline hazards

• Even though we perfectly divide pipeline stages, it’s still hard to achieve CPI == 1.

• Pipeline hazards:
  • Structural hazard
    • The hardware does not allow two pipeline stages to work concurrently
  • Data hazard
    • A later instruction in a pipeline stage depends on the outcome of an earlier instruction in the pipeline
  • Control hazard
    • The processor is not clear about what’s the next instruction to fetch
Structural hazard
Structural hazard

• What just happened here is problematic if we change one of the source register of the 2nd sub instruction?

```
add $1, $2, $3
lw  $4, 0($5)
sub $6, $7, $8
sub $9,$10, $1
sw  $1, 0($12)
```

A. The register file is trying to read and write at the same cycle
B. The ALU and data memory are both active at the same cycle
C. A value is used before it’s produced
D. Both A and B
E. Both A and C
**Structural hazard**

- The hardware cannot support the combination of instructions that we want to execute at the same cycle.
- The original pipeline incurs structural hazard when two instructions competing the same register.
- **Solution: write early, read late**
  - Writes occur at the clock edge and complete long enough before the end of the clock cycle.
  - This leaves enough time for outputs to settle for reads.
  - The revised register file is the default one from now!

```assembly
add $1, $2, $3
lw  $4, 0($5)
sub $6, $7, $8
sub $9,$10, $1
sw  $1, 0($12)
```
Structural hazard

• What just happened here is problematic if we only have a combined memory unit (instruction/data) that accepts only one request at each cycle?

```
add $1, $2, $3
lw  $4, 0($5)
sub $6, $7, $8
sub $9,$10,$11
sw  $1, 0($12)
```

A. The register file and memory are both active at the same cycle
B. The ALU and memory are both active at the same cycle
C. The processor needs to fetch an instruction and access memory at the same cycle
D. Both A and B
E. Both A and C
Structural hazard

- The design of hardware cause structural hazard
- We need to modify the hardware design to avoid structural hazard