Designing a Pipelined Processor

In this Chapter, we will study

1. Pipelined datapath
2. Pipelined control
3. Data Hazards
4. Forwarding
5. Branch Hazards
6. Performance

Pipelining is Natural

Car Wash Station (non-pipelined)

Vacuum  Wash  Dry
Brian  Brian  Brian
Steve  Brian  Brian
Steve  Brian  Brian
Bob    Steve  Brian

Vacuum, Wash, Dry each takes 5 minutes
Nonpipelined Car Wash Station serving one customer per 15 minutes

Car Wash Station (pipelined)

Vacuum  Wash  Dry
Brian  Steve  Bob
Bob    Steve  Mike
Mike   Bob    Gorge
Gorge  Bob    Steve

Pipelined Car Wash Station after the first customer (it takes 15 minutes), then every 5 minutes finishing service for a new customer

What we learn from car wash station

• Pipelining does not help latency of a single task, but it helps throughput of entire workload
• Pipeline rate is limited by the slowest pipeline stage
• Multiple tasks operating simultaneously using different resources
• Potential speedup = Number of stages
• Unbalanced lengths of pipe stages reduces speedup
• Time to fill the pipe and time to drain the pipe → reduce speedup
• Stall the pipe for dependences

Performance

• Suppose we execute 100 instruction
• single cycle implementation
  – 50 ns/cycle × 1 CPI × 100 inst. = 5000 ns
• multiple cycle implementation
  – 10 ns/cycle × 4.6 CPI (instr. mix) × 100 inst = 4600 ns
• ideal pipelined processor
  – 10 ns/cycle × (1 CPI × 100 inst + 4 cycle to fill up the pipe) = 1040 ns
  – here we assume that 5 stages in the pipeline
  – 5 stages : IF, ID, EX, MEM, WB
Why pipeline?

Because the resources are there! Why waste!

Start fetching and executing the next instruction before the current one has completed.

Remember the performance equation: CPU time = CPI * CC * IC

Fetch (and execute) more than one instruction at a time.

Superscalar processing – stay tuned.

A Pipelined MIPS Processor

- Start the next instruction before the current one has completed.
  - Improves throughput - total amount of work done in a given time.
  - Instruction latency (execution time, delay time, response time - time from the start of an instruction to its completion) is not reduced.

Cycle 1  Cycle 2  Cycle 3  Cycle 4  Cycle 5  Cycle 6  Cycle 7  Cycle 8

lw  sw  R-type

IFetch Dec Exec Mem WB

Clock cycle (pipeline stage time) is limited by the slowest stage:
  - for some instructions, some stages are wasted cycles.
Pipelining the MIPS ISA

• What makes it easy
  – all instructions are the same length (32 bits)
    » can fetch in the 1st stage and decode in the 2nd stage
  – few instruction formats (three) with symmetry across formats
    » can begin reading register file in 2nd stage
  – memory operations can occur only in loads and stores
    » can use the execute stage to calculate memory addresses
  – each MIPS instruction writes at most one result (i.e., changes
    the machine state) and does so near the end of the pipeline
    (MEM and WB)

• What makes it hard
  – structural hazards: what if we had only one memory?
  – control hazards: what about branches?
  – data hazards: what if an instruction’s input operands depend
    on the output of a previous instruction?

Can pipelining cause us trouble?

• Pipeline Hazards:
  – structural hazards: attempt to use the same resource by
    two different instructions
  – data hazards: attempt to use data before it is valid. Such
    as an instruction depends on results of prior instructions
    still in the pipeline
  – control hazards: attempt to execute branch before
    condition is computed by prior instructions

• Resolve Pipeline Hazards:
  – simplest solution: wait until hazards are gone
  – take action to resolve hazards, the earlier the hazards are
    solved, the higher the performance the machine can
    achieve.

• Pipeline Overhead:
  – need pipeline registers between stages

Why Pipeline? For Performance!

| Inst 0 | Inst 1 | Inst 2 | Inst 3 | Inst 4 |
|发明| Reg| ALU| Reg| ALU|
| Reg| Reg| Reg| Reg| Reg|

Once the pipeline is full, one instruction is completed every cycle, so CPI = 1

Structural Hazard Example

| Instruction 1 | Mem | Reg | ALU | Mem | Reg |
| Instruction 2 | Mem | Reg | ALU | Mem | Reg |
| Instruction 3 | Mem | Reg | ALU | Mem | Reg |
| Instruction 4 | Mem | Reg | ALU | Mem | Reg |

Two instructions want to access the MEM at the same time
Structural Hazard!
How to resolve this hazard?
A Single Memory Would Be a Structural Hazard

- Fix with separate instr and data memories (I$ and D$)

How About Register File Access?

Fix register file access hazard by doing reads in the second half of the cycle and writes in the first half

Register Usage Can Cause Data Hazards

- Dependencies backward in time cause hazards

- Read before write data hazard
Register Usage Can Cause Data Hazards

- Dependencies backward in time cause hazards

```
add $1, IM Reg DM Reg
sub $4,$1,$5
and $6,$1,$7
or $8,$1,$9
xor $4,$1,$5
```

- Read before write data hazard

Loads Can Cause Data Hazards

- Dependencies backward in time cause hazards

```
lw $1,4($2)
sub $4,$1,$5
and $6,$1,$7
or $8,$1,$9
xor $4,$1,$5
```

- Load-use data hazard

One Way to “Fix” a Data Hazard

```
add $1, IM Reg DM Reg
sub $4,$1,$5
and $6,$1,$7
```

Can fix data hazard by waiting – stall – but impacts CPI

Another Way to “Fix” a Data Hazard

```
add $1, IM Reg DM Reg
sub $4,$1,$5
and $6,$1,$7
or $8,$1,$9
xor $4,$1,$5
```

Fix data hazards by forwarding results as soon as they are available to where they are needed
Another Way to “Fix” a Data Hazard

Fix data hazards by forwarding results as soon as they are available to where they are needed.

add $1, IM Reg
sub $4,$1,$5 DM Reg
and $6,$1,$7 DM Reg
or $8,$1,$9 DM Reg
xor $4,$1,$5 DM Reg

Forwarding with Load-use Data Hazards

lw $1,4($2) IM Reg
sub $4,$1,$5 IM Reg
and $6,$1,$7 DM Reg
or $8,$1,$9 DM Reg
xor $4,$1,$5 IM Reg

• Will still need one stall cycle even with forwarding

Branch Instructions Cause Control Hazards

• Dependencies backward in time cause hazards

lw Inst 4
beq Inst 3
Inst 4
One Way to “Fix” a Control Hazard

Fix branch hazard by waiting — but affects CPI

Corrected Datapath to Save RegWrite Addr

• Need to preserve the destination register address in the pipeline state registers

Corrected Datapath to Save RegWrite Addr

• Need to preserve the destination register address in the pipeline state registers

MIPS Pipeline Control Path Modifications

• All control signals can be determined during Decode — and held in the state registers between pipeline stages
Other Pipeline Structures Are Possible

• What about the (slow) multiply operation?
  – Make the clock twice as slow or ...
  – let it take two cycles (since it doesn’t use the DM stage)

• What if the data memory access is twice as slow as the instruction memory?
  – make the clock twice as slow or ...
  – let data memory access take two cycles (and keep the same clock rate)

Sample Pipeline Alternatives

• ARM7

• StrongARM-1

• XScale

Summary

• All modern day processors use pipelining
• Pipelining doesn’t help latency of single task, it helps throughput of entire workload
• Potential speedup: a CPI of 1 and fast a CC
• Pipeline rate limited by slowest pipeline stage
  – Unbalanced pipe stages makes for inefficiencies
  – The time to “fill” pipeline and time to “drain” it can impact speedup for deep pipelines and short code runs
• Must detect and resolve hazards
  – Stalling negatively affects CPI (makes CPI less than the ideal of 1)