Computer Architecture and Organization

Chapter 6 Processor Pipelining

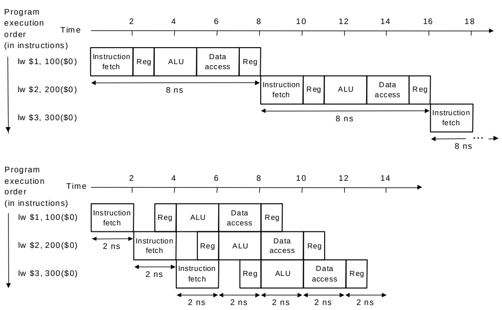
Henk Corporaal / Ben Juurlink
Delft University of Technology
Spring 2000
Email: h.corporaal@et.tudelft.nl
URL : http://cardit.et.tudelft.nl/~heco/courses/et2-010

Topics

- Pipelining overview
- Pipelined datapath
- Pipelined control
- Hazards:
  - Structural
  - Data
  - Control
- Exceptions
- Superscalar:
  - Multi-issue
  - Dynamic scheduling
- PowerPC and Pentium Pro / II / III pipelines

Pipelining

- Improve perfomance by increasing instruction throughput



Pipelining

- Ideal speedup = number of stages
- Do we achieve this?

Pipelining

- What makes it easy
  - all instructions are the same length
  - just a few instruction formats
  - memory operands appear only in loads and stores
- What makes it hard?
  - structural hazards: suppose we had only one memory
  - control hazards: need to worry about branch instructions
  - data hazards: an instruction depends on a previous instruction
- We'll build a simple pipeline and look at these issues
- We'll talk about modern processors and what really makes it hard:
  - exception handling
  - trying to improve performance with out-of-order execution, etc.

Basic Idea

What do we need to add to actually split the datapath into stages?

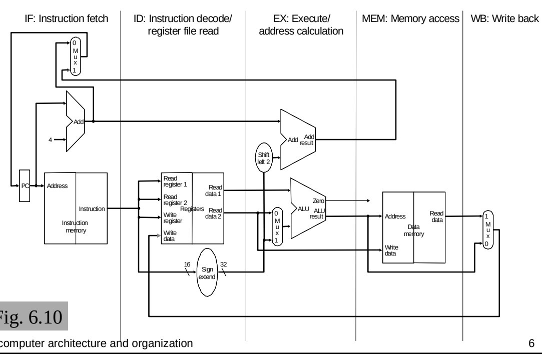


Fig. 6.10

Pipelined Datapath

Can you find a problem even if there are no dependencies?
What instructions can we execute to manifest the problem?

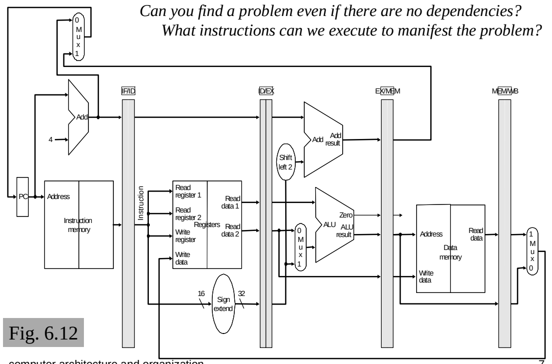


Fig. 6.12

computer architecture and organization 7

Corrected Datapath

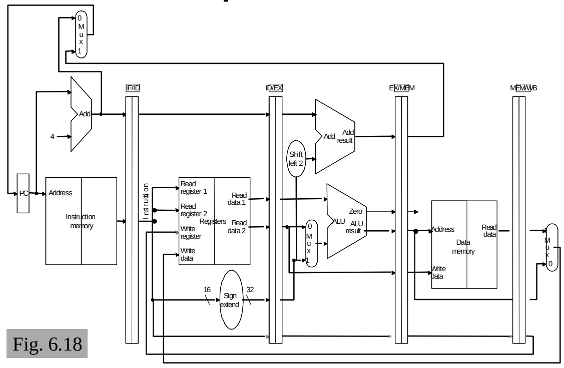
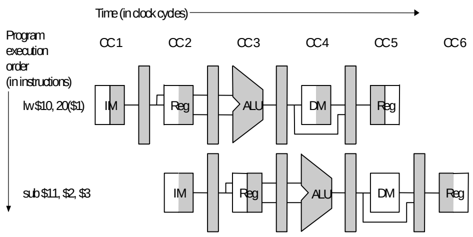


Fig. 6.18

computer architecture and organization 8

Graphically Representing Pipelines



- Can help with answering questions like:
  - how many cycles does it take to execute this code?
  - what is the ALU doing during cycle 4?
  - use this representation to help understand datapaths

computer architecture and organization 9

Pipeline Control

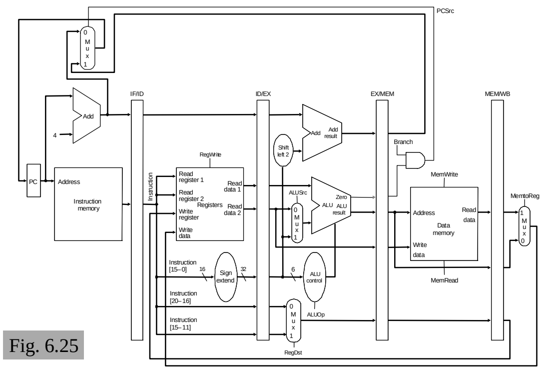


Fig. 6.25

computer architecture and organization 10

Pipeline control

- We have 5 stages. What needs to be controlled in each stage?
  - Instruction Fetch and PC Increment
  - Instruction Decode / Register Fetch
  - Execution
  - Memory Stage
  - Write Back
- How would control be handled in an automobile plant?
  - a fancy control center telling everyone what to do?
  - should we use a finite state machine?

computer architecture and organization 11

Pipeline Control

- Pass control signals along just like the data

	Execution/Address Calculation stage control lines				Memory access stage control lines			stage control lines	
Instruction	Reg Dst	ALU Op1	ALU Op0	ALU Src	Branch	Mem Read	Mem Write	Reg write	Mem to Reg
R-format	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

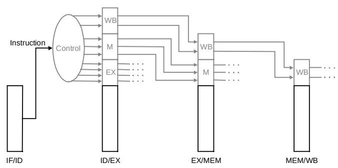


Fig. 6.29

computer architecture and organization 12

Datapath with Control

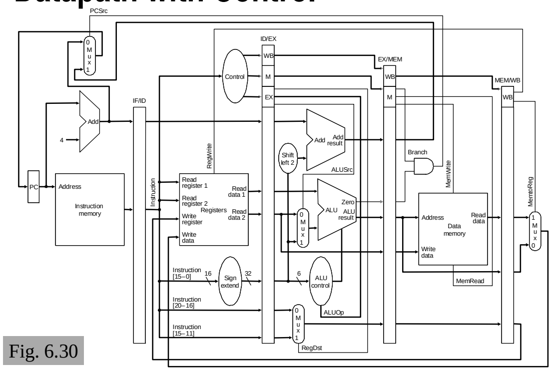


Fig. 6.30

computer architecture and organization 13

Dependencies

- Problem with starting next instruction before first is finished
  - dependencies that “go backward in time” are data hazards

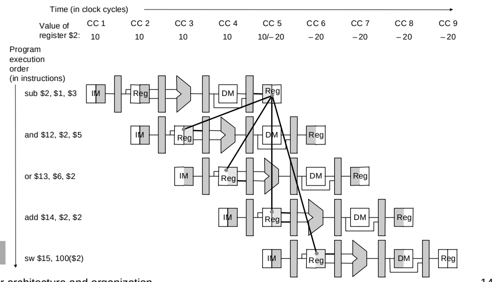


Fig. 6.36

computer architecture and organization 14

Software Solution

- Have compiler guarantee no hazards
- Where do we insert the “NOPs” ?

```
sub  $2, $1, $3
and  $12, $2, $5
or   $13, $6, $2
add  $14, $2, $2
sw   $15, 100($2)
```

- Problem: this really slows us down!

computer architecture and organization 15

Forwarding

- Use temporary results, don’t wait for them to be written
  - register file forwarding to handle read/write to same register
  - ALU forwarding

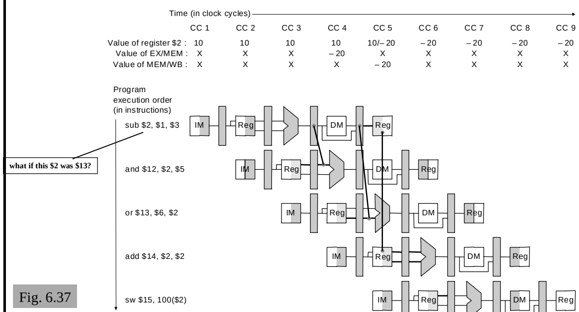


Fig. 6.37

computer architecture and organization 16

Forwarding

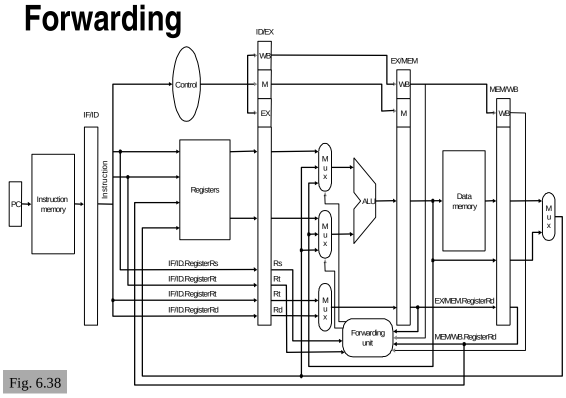


Fig. 6.38

computer architecture and organization 17

Can’t always forward

- Load word can still cause a hazard:
  - an instruction tries to read a register following a load instruction that writes to the same register.

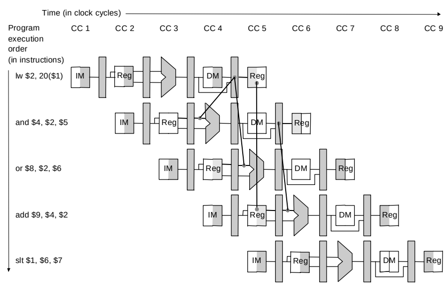


Fig. 6.44

- Thus, we need a hazard detection unit to “stall” the load instruction

computer architecture and organization 18

Stalling

- We can stall the pipeline by keeping an instruction in the same stage

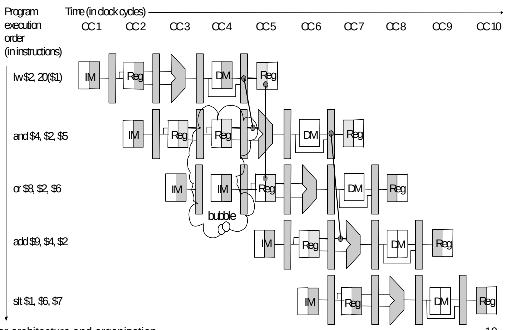


Fig. 6.45

Hazard Detection Unit

- Stall by letting an instruction that won't write anything go forward

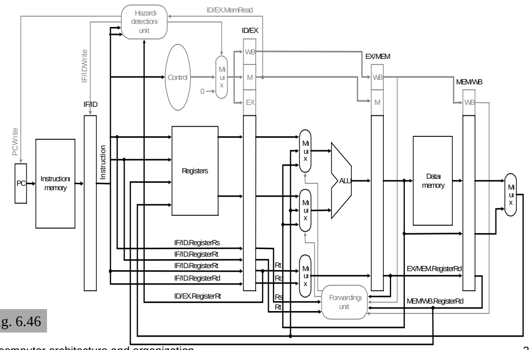


Fig. 6.46

Branch Hazards

- When we decide to branch, other instructions are in the pipeline!

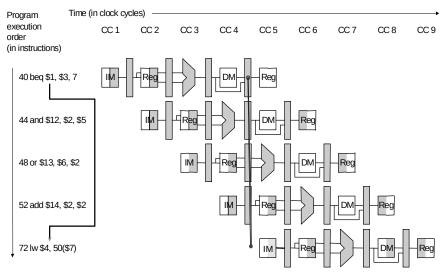


Fig. 6.50

- We are predicting "branch not taken"
  - need to add hardware for flushing instructions if we are wrong

Flushing Instructions

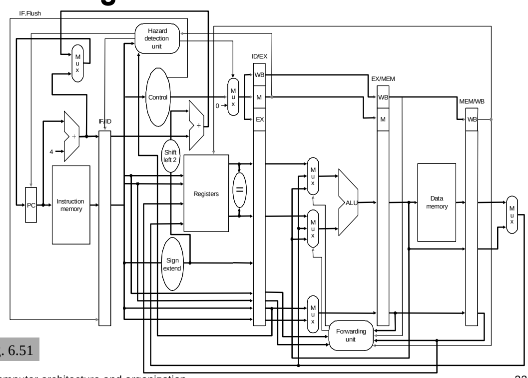


Fig. 6.51

Improving Performance

- Try and avoid stalls! E.g., reorder these instructions:

```
lw $t0, 0($t1)
lw $t2, 4($t1)
sw $t2, 0($t1)
sw $t0, 4($t1)
```

- Add a "branch delay slot"
  - the next instruction after a branch is always executed
  - rely on compiler to "fill" the slot with something useful
- Superscalar: start more than one instruction in the same cycle

Dynamic Scheduling

- The hardware performs the "scheduling"
  - hardware tries to find instructions to execute
  - out of order execution is possible
  - speculative execution and dynamic branch prediction
- All modern processors are very complicated
  - DEC Alpha 21264: 9 stage pipeline, 6 instruction issue
  - PowerPC and Pentium: branch history table
  - Compiler technology important

Exercises

From Chapter six:

- 6.1, 6.2
- 6.4
- 6.11 - 6.15
- 6.19
- 6.21, 6.22

computer architecture and organization 25