COEN-4730/EECE-5730 Computer Architecture

Lecture 2
Review of Instruction Sets and Pipelines
(Appendix A, C)

Cristinel Ababei
Dept. of Electrical and Computer Engineering
MARQUETTE UNIVERSITY
BE THE DIFFERENCE.

Credits: Slides adapted from presentations of Sudeep Pasricha and others: Kubiatowicz, Patterson, Mutlu, Elsevier

1

Outline

- Instruction Set Principles (Appendix A)
- Pipelining (Appendix C)

2

Arrival to current state-of-the art says that for a new Instruction Set…

- Virtually every new architecture designed after 1980 uses a load-store register architecture
- We should expect the use of general-purpose registers
- Memory addressing modes have the ability to significantly reduce instruction counts. We should expect the support of at least these addressing modes: immediate, displacement, and register indirect
- Should support these data sizes and types: 8-, 16-, 32, and 64-bit integers and 64-bit IEEE 754 floating-point numbers
- Support these simple instructions (they will dominate the number of instructions executed): load, store, add, subtract, move register, and shift
- Support these instructions for Control Flow: compare equal, compare not equal, compare less, branch (with a PC-relative address at least 8 bits long), jump, call, and return

Arrival to current state-of-the art says that for a new Instruction Set…

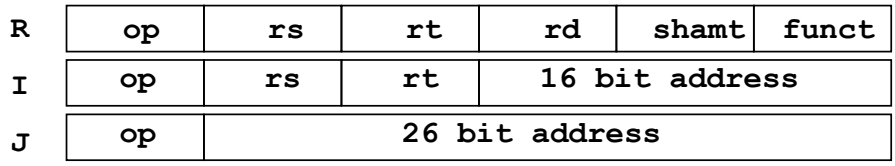
- We should use fixed instruction encoding if interested in performance, and use variable instruction encoding if interested in code size
- Should provide at least 16 general-purpose registers, be sure all addressing modes apply to all data transfer instructions and aim for a minimalist instruction set. This will reduce the complexity of writing a correct compiler (which is a major limitation on the amount of optimization that can be done).
- Before suggesting new instruction set features, look at optimized code first – as a compiler might completely remove the instructions the architect tries to improve…

Example: MIPS, RISC-V follow those recommendations

- MIPS64 has 32 64-bit general purpose registers (GPR)
- Data types: 8-, 16-, 32-, and 64-bit for integers and 32-bit single precision and 64-bit double precision for floating point
- Addressing modes: immediate and displacement, both with 16-bit fields
- Types of instruction format:
  - MIPS – three types: R,I, and J
  - RISC-V – six types: R,I,S,SB,UJ,U
- Supports the list of simple instructions recommended plus a few others
- Control handled through a set of jumps and a set of branches

MIPS Encoding Summary

- Microprocessor without Interlocked Pipeline Stages
- 3 instruction formats: R, I, and J types



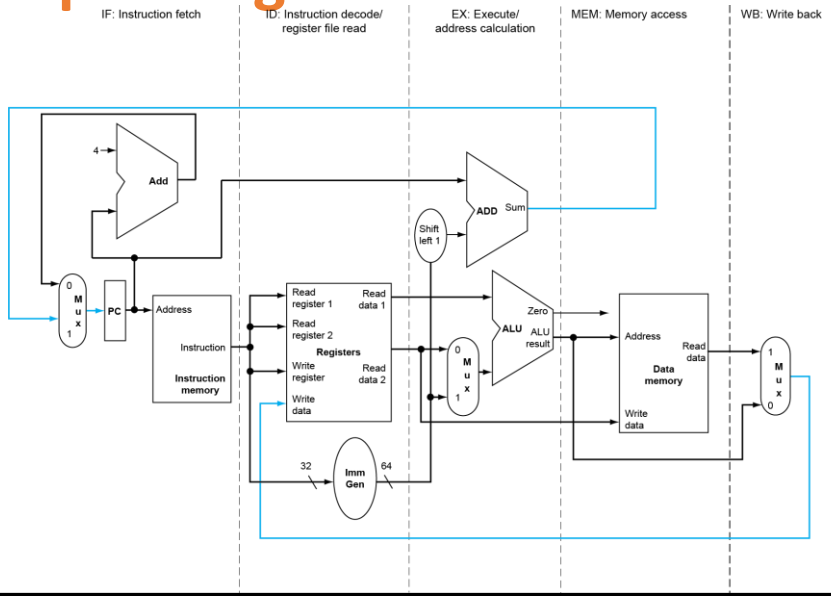
RISC-V Encoding Summary

Name (Field Size)	Field 7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	Comments
R-type	funct7	rs2	rs1	funct3	rd	opcode	Arithmetic instruction format
I-type	immediate[11:0]		rs1	funct3	rd	opcode	Loads & immediate arithmetic
S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode	Stores
SB-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode	Conditional branch format
UJ-type	immediate[20,10:1,11,19:12]				rd	opcode	Unconditional jump format
U-type	immediate[31:12]				rd	opcode	Upper immediate format

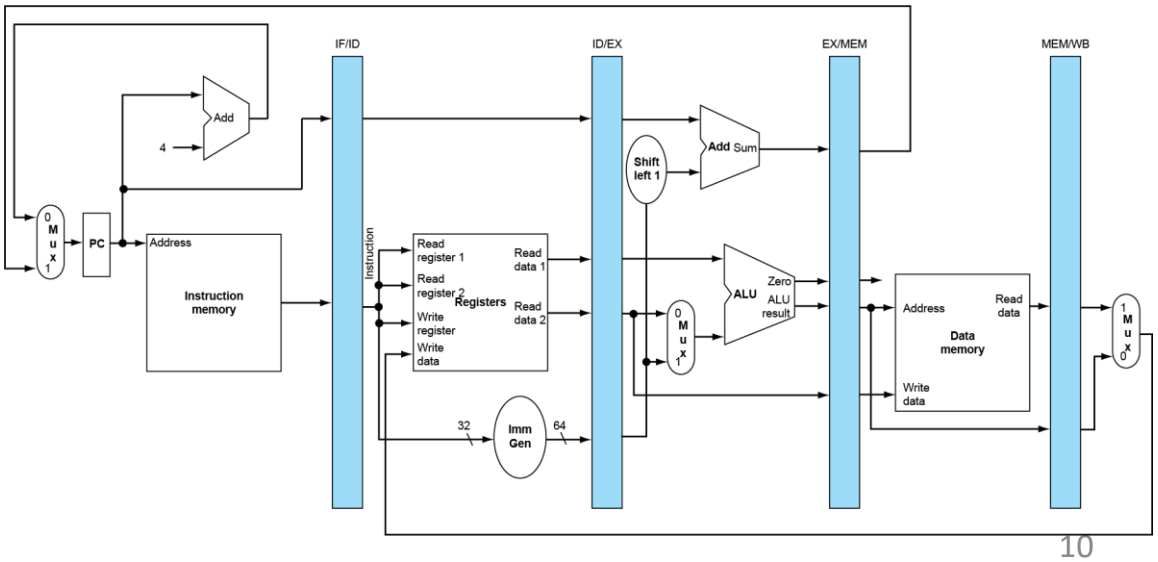
Outline

- Instruction Set Principles (Appendix A)
- Pipelining (Appendix C)

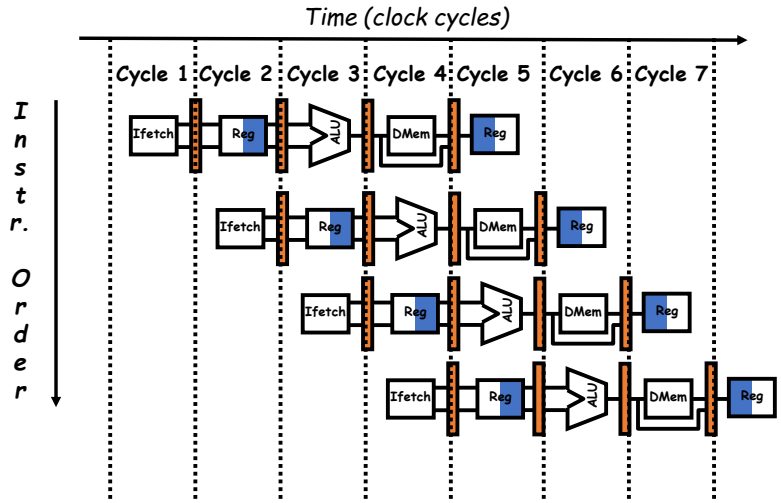
Pipelining – how is it done?



RISC-V Datapath: 5 Stages



Visualizing Pipelining



11

11

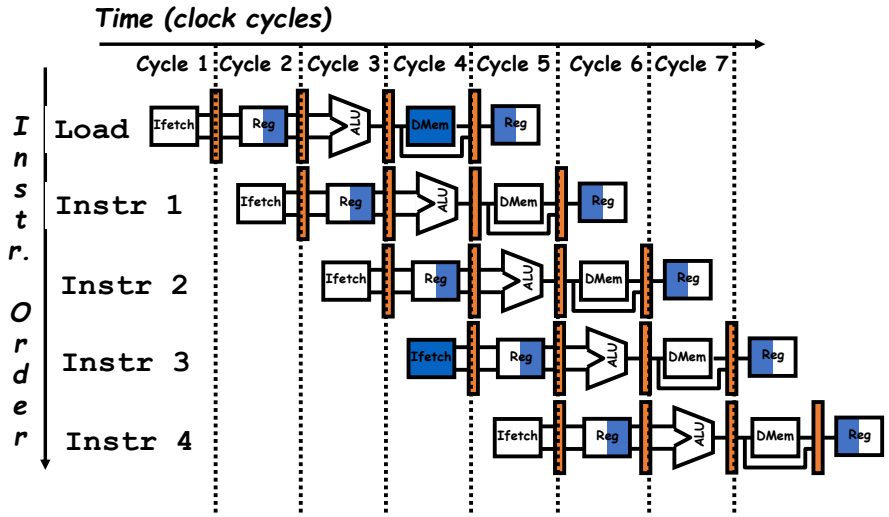
Pipelining is not quite that easy!

- Limits to pipelining: hazards prevent next instruction from executing during its designated clock cycle
  1. Structural hazards: HW cannot support this combination of instructions
  2. Data hazards: Instruction depends on result of prior instruction still in the pipeline
  3. Control hazards: Caused by delay between the fetching of instructions and decisions about changes in control flow (branches and jumps)

12

12

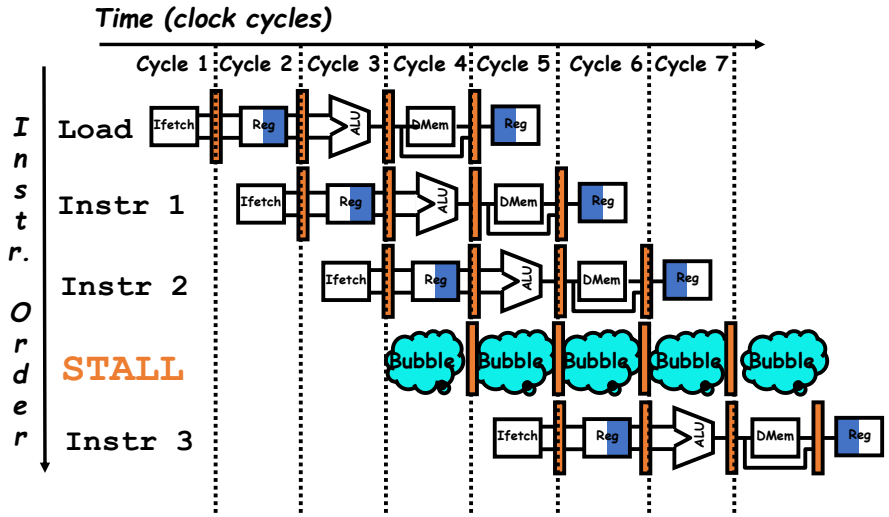
1. One Memory Port/Structural Hazards
Instruction and Data memories are the same memory unit



13

13

Structural Hazards

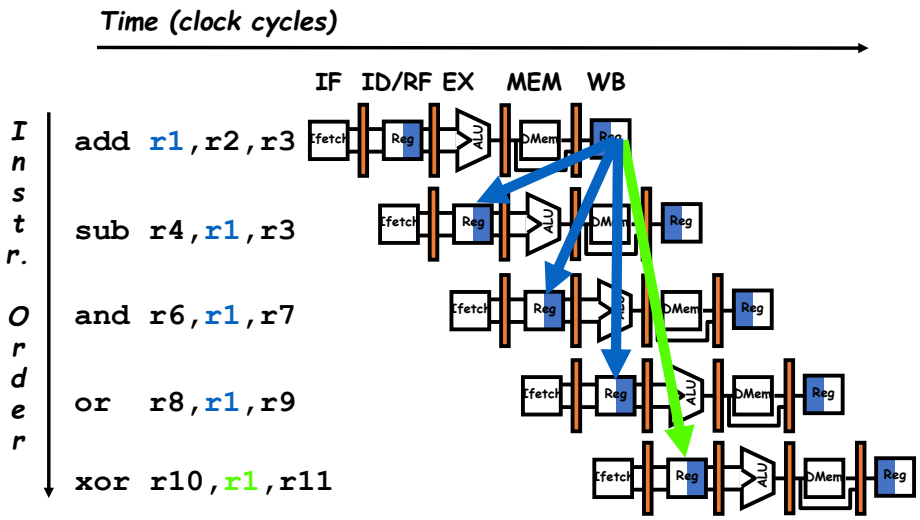


How do you "bubble" the pipe? De-assert all control lines in appropriate pipe stages.

14

14

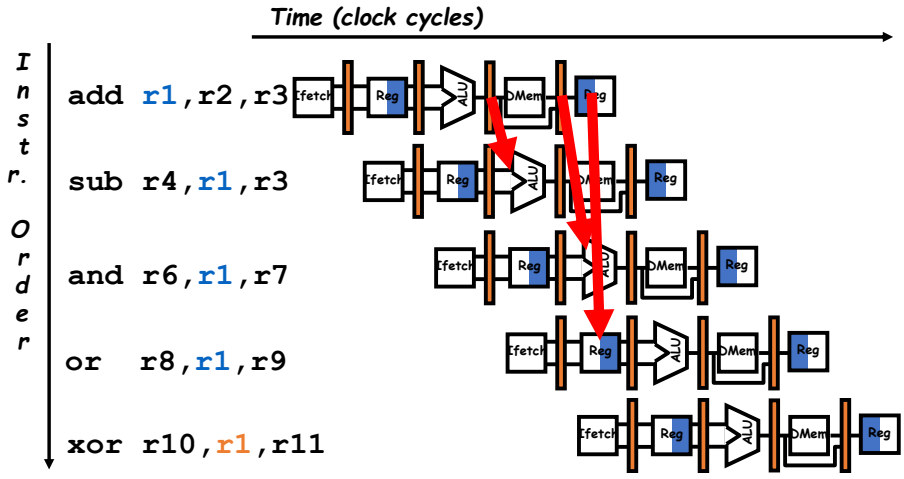
2. Data Hazard on R1



15

15

Forwarding to Avoid Data Hazard

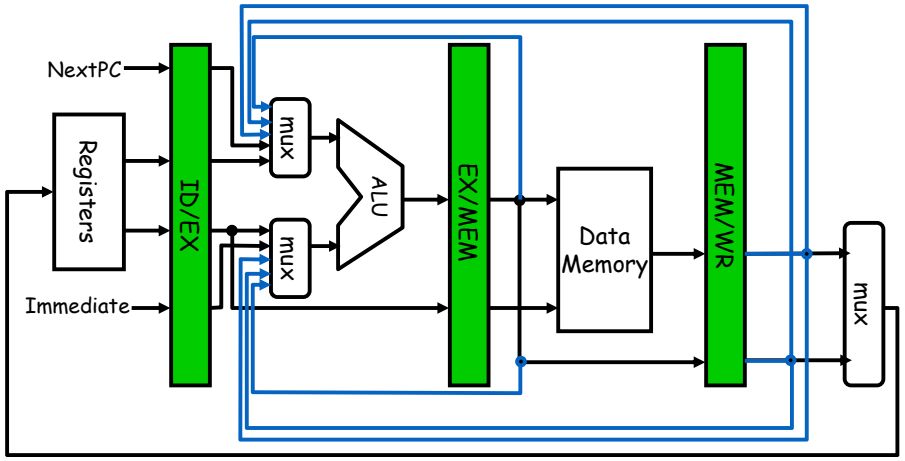


Take result from where it's 1st time available and move it to where it's/could-be needed

16

16

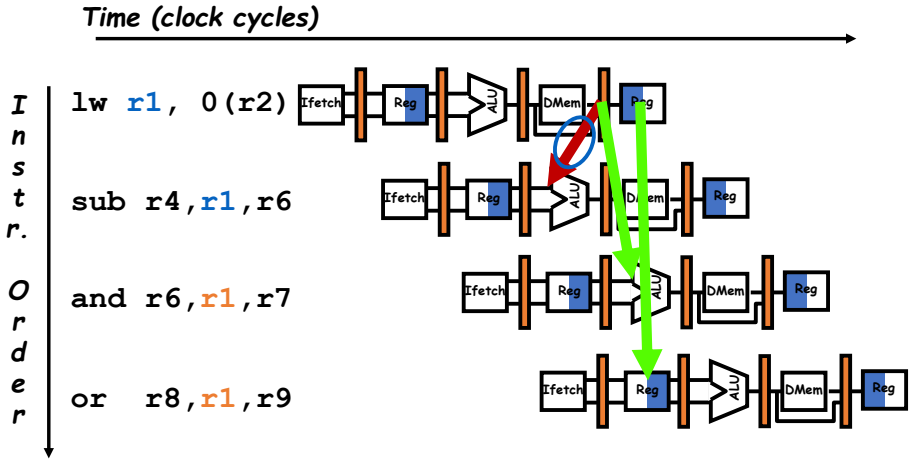
Hardware changes to support Forwarding (a.k.a. Bypassing or Short-circuiting)



17

17

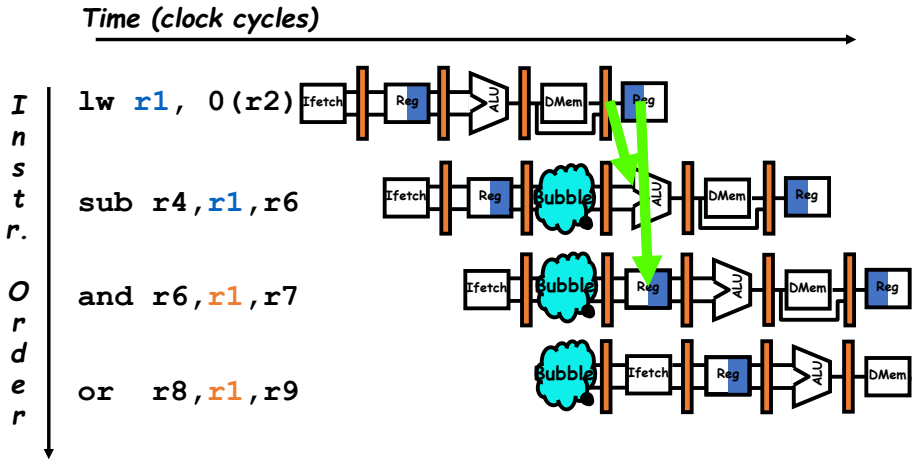
Data Hazard Even with Forwarding (1/2) Load-Use Situations



18

18

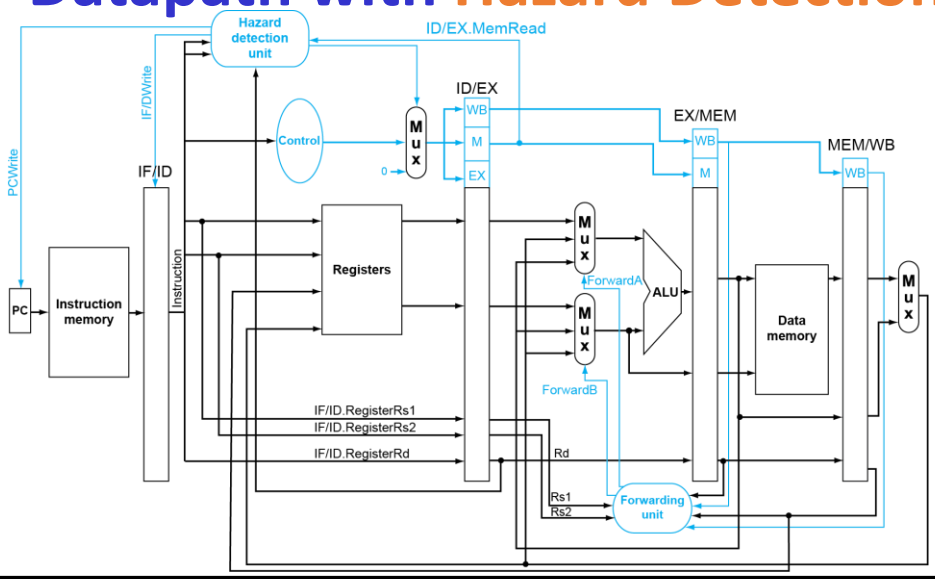
Data Hazard Even with Forwarding (2/2)



How to STALL the Pipeline

1. Force control values in ID/EX register to 0
   - EX, MEM and WB will therefore do NOP (no-operation)
2. Prevent update of PC and IF/ID register
   - Using instruction is decoded again
   - Following instruction is fetched again
   - 1-cycle stall allows MEM to read data for ld

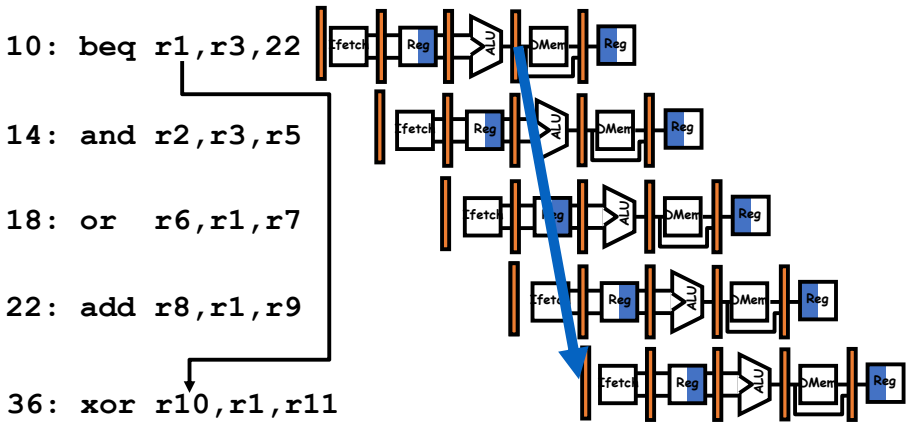
Datapath with Hazard Detection



21

21

3. Control Hazard on Branches - Three Stage Stall



If we had to wait until the end of EX state to find out if branch is taken or not taken (untaken); Let's assume it's taken; What do you do with the 3 instructions in between? We need to **STALL** (i.e., do nothing or delay, NOP) the pipeline 3 stages now or **FLUSH** (i.e., discard) later.

22

22

Branch STALL Impact

- If CPI = 1, 30% branch,
  Stall 3 cycles → new CPI = 1.9!
- Two-part solution:
  - **Determine branch taken or not sooner**
  - **Compute taken branch address earlier**
- RISC-V branch tests if register = 0 or ≠ 0
- RISC-V Solution:
  - Move Zero test to ID/RF stage
  - Adder to calculate new PC in ID/RF stage
  - **1 clock cycle penalty for branch versus 3**

How to FLUSH

- To flush the pipe (when branch is taken)
  - Set IF.Flush (new control line)
  - Zero all control lines
    - Similar to stall, except don't disable the PC and IF/ID write controls – this effectively writes over what the previous instruction was doing.
  - No memory or register writes will have yet happened, so everything else is OK

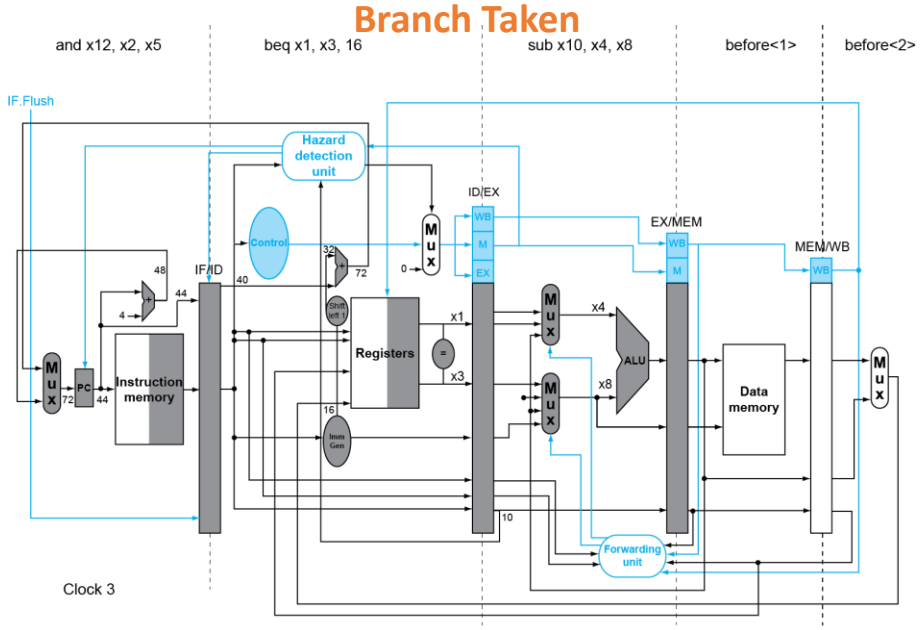
Reducing Branch Delay

- Move hardware to determine outcome to ID stage
  - Target address adder
  - Register comparator
- This means the branch decision can be made during the ID stage instead of the EX stage.
  - This is why we need an IF.Flush, but not an ID.Flush

25

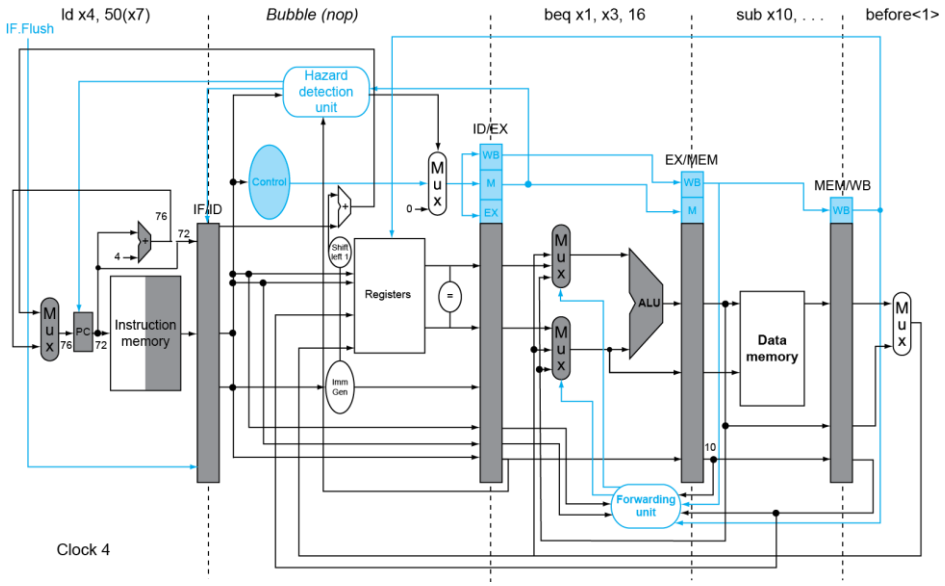
Datapath – with 1 cycle penalty for control hazards

Branch Taken



26

Datapath – with 1 cycle penalty for control hazards
Branch Taken



Four Compile Time Schemes to Reduce Branch Hazard Penalties due to One-Delay Stalls

#1: Stall until branch direction is clear

#2: Predict (or Treat) Branch as Not Taken

- Execute successor instructions in sequence as if the branch were a normal instruction
- "Squash" instructions in pipeline if branch actually taken
- Advantage of late pipeline state update
- 47% branches not taken on average
- PC+4 already calculated, so use it to get next instruction

#3: Predict Branch as Taken

- 53% branches taken on average
- But have not calculated branch target address
  - So, still incurs 1 cycle branch penalty

Four Compile Time Schemes to Reduce Branch Hazard Penalties due to One-Delay Stalls

#4: Delayed Branch

- Introduce the sequential successor instruction, which is executed irrespective of whether or not the branch is taken
- The job of the Compiler is to make the successor instructions valid and useful

29

29

Concluding Remarks

- ISA influences design of datapath and control
- Datapath and control influence design of ISA
- Pipelining improves instruction throughput using parallelism – ILP (instruction level parallelism)
  - More instructions completed per second
  - Latency for each instruction is NOT reduced!
- Hazards: structural, data, control
- Multiple issue and dynamic scheduling
  - Dependencies limit achievable parallelism
  - Complexity leads to the power wall

30

30