Computer Architecture
Pipelines

Diagrams are from *Computer Architecture: A Quantitative Approach, 2nd, Hennessy and Patterson.*
**Instruction Execution**

![Instruction Execution Diagram](image)

**FIGURE 3.1** The implementation of the DLX datapath allows every instruction to be executed in four or five clock cycles.
1. **Instruction Fetch (IF)** - Get the instruction to execute.

   IR \( \leftarrow \) M[PC]
   NPC \( \leftarrow \) PC + 4

2. **Instruction Decode/Register Fetch (ID)** – Figure out what the instructions is supposed to do and what it needs.

   A \( \leftarrow \) Register[R_{s1}]
   B \( \leftarrow \) Register[R_{s2}]
   Imm \( \leftarrow \) (IR_{16})_{16}##IR_{16..31}
3. Execution (EX)  The instruction has been decoded, so execution can be split according to instruction type.

<table>
<thead>
<tr>
<th>Instruction Type</th>
<th>ALU Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reg-Reg</td>
<td>ALUout $\leftarrow A \text{ op } B$</td>
</tr>
<tr>
<td>Reg-Imm</td>
<td>ALUout $\leftarrow A \text{ op Imm}$</td>
</tr>
<tr>
<td>Branch</td>
<td>ALUout $\leftarrow \text{NPC + Imm}$ Target</td>
</tr>
<tr>
<td></td>
<td>cond $\leftarrow (A{=,!=}0)$</td>
</tr>
<tr>
<td>LD/ST</td>
<td>ALUout $\leftarrow A \text{ op Imm}$ Eff Address</td>
</tr>
<tr>
<td>Jump</td>
<td>??</td>
</tr>
</tbody>
</table>
4. Memory Access/Branch Completion (MEM) – Besides the IF stage this is the only stage that access the memory to load and store data.

Load: Load Memory Data = LMD = Mem[ALUout]
Store: Mem[ALUout] ← B
Branch: PC ← (cond)?ALUout:NPC
Jump/JAL: ??
JR: PC ← A
ELSE: PC ← NPC

5. Write-Back (WB) – Store all the results and loads back to registers.

ALU Instruction: Rd ← ALUoutput
Load: Rd ← LMD
JAL: R31 ← Old NPC
What is Pipelining??

Pipelining is an implementation technique whereby multiple instructions are overlapped in execution.
What are the overheads of pipelining?
What are the complexities?
What types of pipelines are there?

<table>
<thead>
<tr>
<th>Time</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instr 1</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr 2</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr 3</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr 4</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instr 5</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Pipelining

FIGURE 3.3 The pipeline can be thought of as a series of data paths shifted in time.
Pipeline Overhead

- Latches between stages
  - Must latch data and control
- Expansion of stage delay...
  - Original time was IF + ID + EX + MEM + WB
  - With phases skipped for some instructions
  - New best time is MAX (IF,ID,EX,MEM,WB) + Latch Delay
- More memory ports or separate memories
- Requires complicated control to make as fast as possible
  - But so do non-pipelined high performance designs
Pipelining

Pipeline complexities

- 3 or 5 or 10 instructions are executing at the same time
- What if the instructions are dependent?
  - Must have hardware or software ways to ensure proper execution.
- What if an interrupt occurs?
  - Would like to be able to figure out what instruction caused it!!
What causes pipeline stalls?

1. **Structural Hazard**
   Inability to perform 2 tasks.
   Example:
   - Memory needed for both instructions and data
   - SpecInt92 has 35% load and store
   - If this remains a hazard, each load or store will stall one IF.
Pipeline Stalls – Data

2.1 Data Hazards – Read after Write

Data is needed before it is written.

```
WBMEMEXIDIFADD R4,R2,R3
87654321 TIME =
WBMEMEXIDIFADD R4,R1,R5
WBMEMEXIDIFADD R4,R1,R5
Calc Done
R1 Written
ADD R4,R2,R3 IF ID EX MEM WB
R1 Read?
ADD R4,R1,R5 IF ID EX MEM WB
R1 Used
ADD R4,R1,R5 IF ID EX MEM WB
ADD R4,R2,R3 IF ID EX MEM WB
```

TIME = 1 2 3 4 5 6 7 8

R1 Read?
Pipeline Stalls – Data

- Data must be **forwarded** to other instructions in pipeline if ready for use but not yet stored.
- In 5-stage MIPS pipeline, this means forwarding to next three instructions.
- Can reduce to 2 instructions if register bank can process a simultaneous read and write.
  - Text indicates this as writing in first half of WB and reading in second half of ID.
  - More likely, both occur at once, with read receiving the data as it is written.
Pipeline Stalls – Data

Load Delays
Data available at end of MEM, not in time for the Instr + 1’s EX phase.
Pipeline Stalls – Data

So insert a bubble...

FIGURE 3.13 The load interlock causes a stall to be inserted at clock cycle 4, delaying the SUB instruction and those that follow by one cycle.
Pipeline Stalls – Data

A Store requires two registers at two different times:
- Rs1 is required for address calculation during EX
- Data to be stored (Rd) is required during MEM

Data can be loaded and immediately stored
Data from the ALU must be forwarded to MEM for stores

FIGURE 3.11 Stores require an operand during MEM, and forwarding of that operand is shown here.
2.2 Data Hazards – Write After Write
Out-of-order completion of instructions

Pipeline Stalls – Data

Length of FDIV pipe could cause WAW error

FMUL1
FMUL2

FDIV R1, R2, R3
FMUL R1, R2, R3

IF
ID/RF
ISSUE

FDIV1
FDIV2
FDIV3
FDIV4

WB
2.3 Data Hazards – Write After Read
A slow reader followed by a fast writer

```
FMADD R1, R2, R3, R4  IF  ID  IS  FM1  FM2  FM3  FM4  WB
FLOAD R4, #300(R5)     IF  ID  IS  F1   WB
```

Read registers for multiply
Read registers for add
2.4 Data Hazards – Read After Read

A read after a read is not a hazard

• Assuming that a read does not change any state
3. Control Hazards
Branch, Jump, Interrupt
Or why computers like straight line code…

3.1 Control Hazards – Jump

• When is knowledge of jump available?
• When is target available?

With absolute addressing, may be able to fit jump into IF phase with zero overhead.
Pipeline Stalls – Control

- This relies on performing simple decode in IF.
- If IF is currently the slowest phase, this will slow the clock.
- Relative jumps would require an adder and its delay.
3.2 Control Hazards – Branch

Branch is resolved in MEM and information forwarded to IF, causing three stalls.
16% of the SpecInt instructions are branches, and about 67% are taken,

\[
\text{CPI} = \text{CPI}_{\text{ideal}} + \text{STALLS} = 1 + 3(0.67)(0.16) = 1.32
\]

Must
- Resolve branch sooner
- Update PC sooner
- Fetch correct next instruction more frequently
Pipeline Stalls – Control

Delayed branch and branch delay slot

• Ideally, fill slot with instruction from before branch
• Otherwise from target (since branches are more often taken than not)
• Or from fall-through
• Slots filled 50-70% of time
• Interrupt recovery complicated
Summary of Pipelines

- Why use a pipeline? Seems so complicated.
- What are some of the overheads?
Motorola 68000 Family (circa late ’70’s)

• Instructions can specify 8, 16, or 32 bit operands.
• Has 32 bit registers.
• Has 2 register files A and D both have 8 32-bit registers. (A is for addresses and D is for Data). Used to save bits in the opcode since implicit use of A or D is in the instruction.
• Uses a two-address instruction set.
• Instructions are usually 16-bits but can have up to 4 more 16-bit words.
• Supports lots of addressing modes.
Intel x86 Family (circa late ’70’s)

• Probably the least elegant design available.
• Uses 16-bit words, so only 64k unique address can be provided.
• Used **segment addressing** to overcome this limit. Basically shift the address by 4 (multiply by 16) and then add a 16-bit offset giving an effective 20-bit address, thus 1MB of memory.
• Has 4 registers (segment registers) used to hold segment addresses: CS, DS, SS, ES.
• Has 4 general registers: AX, BX, CX, and DX.
• Expanded to 32-bit addressing with the 386.
Sun SPARC Architecture (late 1980’s)

- Scalable Processor ARChitecture
- Developed around the same time as the MIPS architecture.
- Very similar to MIPS except for register usage.
- Has a set of global registers (8) like MIPS.
- Has a large set of registers from which only a subset can be accessed at any time, called a register file.
- Is a load/store architecture with 3-addressing instructions.
- Has 32-bit registers and all instructions are 32-bits.
- Designed to support procedure call and returns efficiently.
- Number of register in the register file vary with implementation but at most 24 can be accessed.
- Those 24 plus the 8 global give a register window.