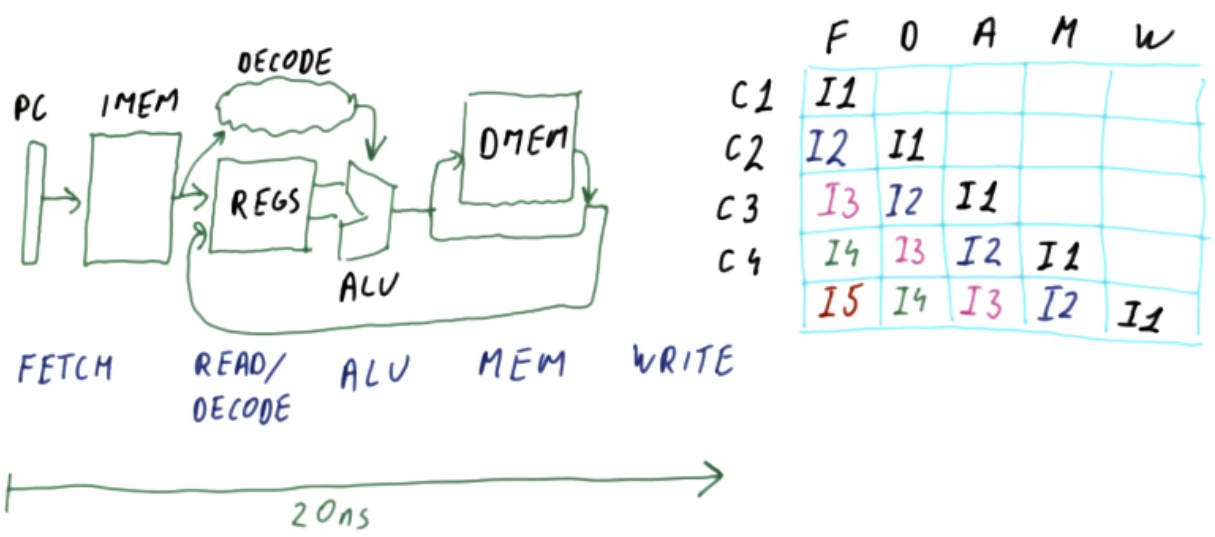
Pipelining

Introduction

1. Core concept to computer architecture; used in almost every computer
2. Pipelining doesn't affect latency, only throughput

Pipelining in a Processor

1. Five phases
   - Fetch
   - Read/decode
   - ALU
   - Access memory
   - Write back
2. If we divide the pipeline into five stages...
   - Latency for each instruction is still 20 ns
   - Throughput is 0.25 instructions per second



Pipelining in a Processor

Laundry Pipelining Quiz

1. Washer takes 1 hour
2. Dryer takes 1 hour
3. Folder takes 1 hour
4. 10 loads of laundry, no pipelining -> 10 * 3 = 30 hours
5. 10 loads of laundry, with pipelining -> 3 + 9 * 1 = 12 hours

Instruction Pipelining Quiz

1. 5-stage pipeline (1 clock cycle/stage)
2. 10 instructions in our program
3. No pipelining -> 10 * 5 = 50 cycles
4. With pipelining -> 1 * 5 + 9 * 1 = 14 cycles

Pipeline CPI

1. Pipeline CPI is ideally 1, but this isn't actually true:
   - Initial fill (CPI -> 1 when # instructions -> infinity)
   - Pipeline stalls (one stage takes longer than others)
     - Previous states are all stalled
     - If this happens regularly, can affect CPU
     - A stall every 5 cycles means a CPI of $6/5 = 1.2$

Processor Pipeline Stalls

1. I1: LW R1, ...
2. I2: ADD R2, R1, 1 (this must stall until the correct value is loaded in R1)
3. I3: ADD R3, R2, 1 (this must stall until the incremented value is in R2)
4. X's are bubbles in the pipeline due to stalls

Cycle	Fetch	Decode	ALU	Memory	Write
C1	I3	I2	I1		
C2	I3	I2	X	I1	
C3	I3	I2	X	X	I1
C4		I3	I2	X	X

Processor Pipeline Stalls and Flushes

1. I4: JUMP
2. I5: SUB ...
3. I6: ADD ...
4. I7: SHIFT ...
5. At C6, we discover the branch should be taken, so the next two instructions are incorrect. I6 should have actually been loaded
   - C6b voids the two instructions

Cycle	Fetch	Decode	ALU	Memory	Write
C1	I3	I2	I1		
C2	I3	I2	X	I1	
C3	I3	I2	X	X	I1
C4	I4	I3	I2	X	X
C5	I5	I4	I3	I2	X
C6a	I6	I5	I4	I3	I2
C6b	X	X	I4	I3	I2
C7	I7	X	X	I4	I3

Control Dependencies

1. Consider the following program:

```
ADD R1, R1, R2
BEQ R1, R3, Label
ADD R2, R3, R4
SUB R5, R6, R8
Label: MUL R5, R6, R8
```

2. Instructions 3, 4, and 5 are said to have a control dependency on the branch

3. How do control dependencies impact CPI in a 5-stage pipeline?
    - 20% of instructions are branch/jump
    - Slightly more than 50% of branch/jump are taken
    - Correct target for branch/jump computed in the 3rd stage
    - 1 + 0.5 * 0.2 * 2 = 1 + 0.1 * 2 = 1.2
    - Better branch prediction can lower the 0.1 term
    - Deeper pipelines increase the 2 cycle penalty

Control Dependence Quiz

1. 25% of all instructions are taken branch/jump
2. 10-stage pipeline
3. Correct target for branch/jump computed in the 6th stage
4. Everything else flows smoothly
5. Actual CPI = 1 + 0.25 * 5 = 2.25

Data Dependencies

1. Consider the following program:

```
ADD R1, R2, R3
SUB R7, R1, R8
MUL R1, R5, R6
```

2. ADD must finish before SUB can continue since it depends on the value in R1
    - Called "Read after Write" dependence (RAW), flow dependence, or true dependence
3. MUL must complete after ADD so the correct value is in R1
    - Called "Write after Write" dependence (WAW), output dependence, or false dependence
4. SUB must read value in R1 before MUL writes to it
    - Called "Write after Read" dependence (WAR), anti-dependence, or false dependence
5. Read after Read is not a dependence since the instructions can be reordered

Data Dependencies Quiz

1. Consider the following program:
    - I1: MUL R1, R2, R3
    - I2: ADD R4, R4, R1
    - I3: MUL R1, R5, R6
    - I4: SUB R4, R4, R1
2. I1 -> I4 has no RAW dependence since I3 overwrites R1

Instr	RAW	WAR	WAW
I1 -> I2	X		
I1 -> I3			X
I1 -> I4			
I2 -> I3		X	

Dependencies and Hazards

1. Dependence: Property of program alone
2. Consider the following program:

```
I1: ADD R1, R2, R3
I2: MUL R1, R4, R5
I3: SUB R4, R6, R7
```

```
I4: DIV R10, R4, R8
I5: XOR R11, R1, R7

* The WAW dependence between I1 and I2 isn't an issue since I2 is guaranteed
to finished after I1
* The WAR dependence between I2 and I3 isn't an issue since the writeback
occurs many cycles after I2 reads the value
* The RAW dependence between I3 and I4 is an issue because I3 hasn't updated
the value in R4 before I4 can read it
```

3. Hazard: Dependence that results in incorrect execution
    - Property of both the program (must have a dependence) and the pipeline
    - In the example, output and anti dependencies cannot become hazardous, but true dependencies can
4. Not all true dependencies are hazardous
    - RAW dependence between I1 and I5 is not a hazard because I1 has already written its value back
    - RAW is not a hazard if there are 3 or more instructions between

Dependencies and Hazards Quiz

1. Consider a three stage pipeline with the following stages:
    - Fetch/decode
    - Read register/ALU
    - Memory/Writeback to register
2. Consider the following program:

```
I1: ADD R1, R2, R4
I2: SUB R5, R1, R4
I3: DIV R6, R1, R7
I4: MUL R7, R1, R8
```

Instr	Dependence?	Hazard?
I2	X	X
I3	X	
I4	X	

Handling of Hazards

1. Detect hazard situations
    - Flush dependent instructions
        - Needed for control dependencies (the wrong instructions are present)
    - Stall dependent instructions
        - Can be used for data dependencies until writeback occurs
    - Fix values read by dependent instructions
        - Can also "forward" the value produced by the writer to the reader
2. Forwarding: An instruction may have computed the value to be written to a register in the ALU stage, but not written it back yet. In this case, this value can be sent to the instructions dependending on the value so they don't have to stall
    - For some data hazards, the value hasn't been computed yet, so it can't be forwarded

Flushes - Stalls - Forwarding Quiz

1. Consider a five stage pipeline with the following stages:
    - Fetch
    - Read

- ALU/Branch
- Memory
- Writeback

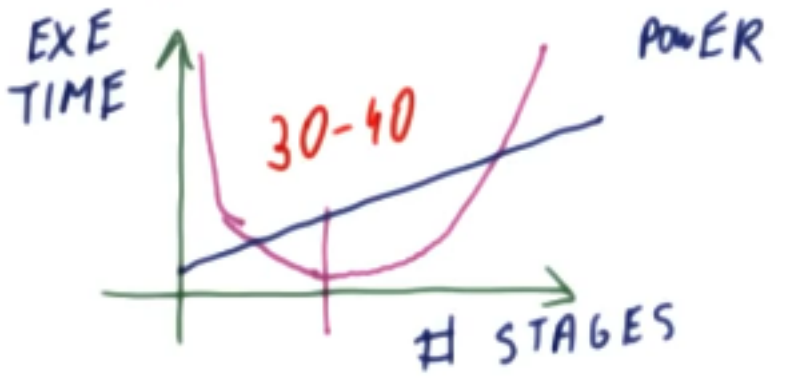
2. Consider the following program:

```
I1: BNE R1, R0, Label
I2: ADD R4, R5, R6
I3: SUB R5, R4, R3
I4: MUL R1, R2, R3
I5: LW R1, 0(R1)
I6: ADD R1, R1, R1
```

Instruction	Flush	Stall	Forward
I2 and I3	X		
I4 and I5			X
I6		X	X

How Many Stages

1. Ideal CPI is 1 (some advanced pipelines aim to complete more than 1 IPC)
2. As the number of stages increases...
   - More hazards: Number of stall/flush cycles depends on depth of pipeline
     - Increases CPI
   - Less work/stage
     - Decreases cycle time
   - ExeTime = #Insts * CPI * CycleTime
     - Number of stages should balance CPI and cycle time
     - Ideal performance (minimum) is 30-40 stages in a modern pipeline
     - However, 30-40 stages increases power usage
   - When considering power, modern processors have 10-15 stages
     - Not ideal performance, but good performance with manageable power consumption



Execution Time vs Number of Stages

Conclusion

1. Pipelines must balance increasing clock frequency versus the stalls and flushes caused by hazards