Lecture 5: ISA Implementation

Kunle Olukotun
Gates 302
kunle@ogun.stanford.edu

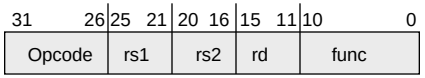
http://www-leland.stanford.edu/class/ee282h/

Basic Pipelining

- Pipelining is the organizational implementation technique that has been responsible for the most dramatic increases in computer performance
- Overview of basic pipelining
  - » DLX without pipelining
  - » What is pipelining?
  - » Computing pipeline speedup
  - » Clocking pipelines
  - » DLX with pipelining
  - » Pipeline hazards
  - » Handling interrupts

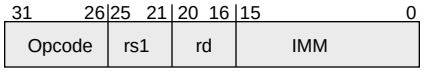
DLX review: Instruction format summary

R-type instructions



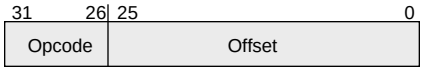
- Register ALU ops

I-type instructions



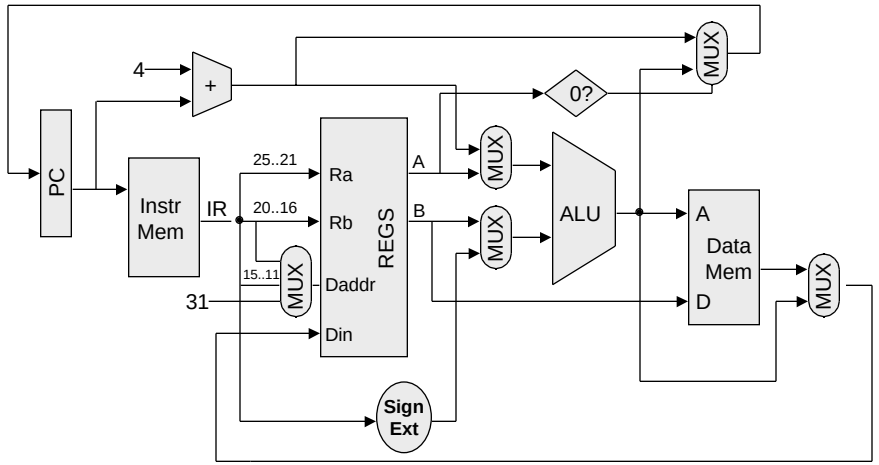
- Load/store
- ALU immediate
- Cond. branch
- Jump register
- JALR

J-type instructions



- Jump
- JAL

The Basic RISC datapath without pipelining



Data Flow: The details

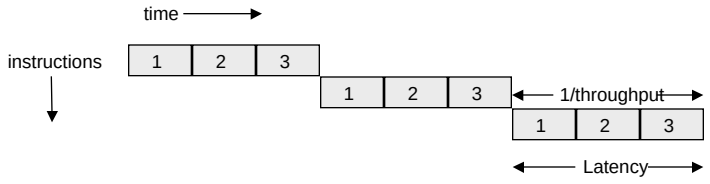
Reg-Reg ALU	*Reg-immed ALU*	*Load*	*Store*	*Branch*	*Jump*
IR = IMem[PC] NPC = PC+4					
A = Regs[IR25..21] B = Regs[IR20..16]					
ALU = A op B	ALU = A op IR15..0	ALU = A + IR15..0		ALU = NPC + IR15..0	ALU = NPC + IR25..0
		Data = DMem[ALU]	DMem[ALU] = B		
Regs [IR15..11] = ALU	Regs [IR20..16] = ALU	Regs [IR20..16] = Data;			
PC = NPC				PC = cond? ALU : NPC	PC = ALU

All operations are combinatorial except the shaded boxes, which store results.

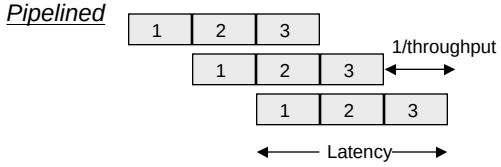
Pipelining

- Exploits instruction-level parallelism by overlapping the execution of consecutive instructions

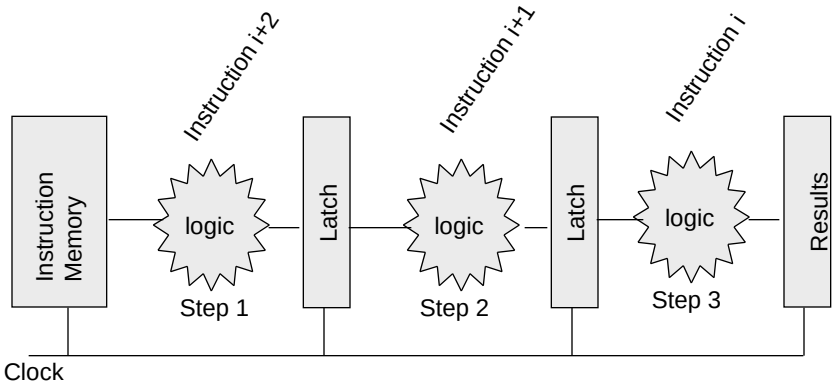
Unpipelined



Pipelined



Implementing pipelining



Pipelining: Computing the speedup

- Time per instruction
  - » TPI = CPI * cycle_time
  - » We can think about pipelining as reducing either CPI or cycle_time
- Ideal Speedup

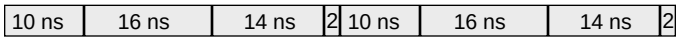
  $$\text{Speedup} = \frac{\text{TPI}_{\text{without pipeline}}}{\text{TPI}_{\text{with pipeline}}} = \text{number of pipeline stages}$$

  - » Requires all stages to be perfectly balanced
  - » No latch overhead
  - » No stall cycles
- The speedup from pipelining is limited
  - » $\text{CPI}_{\text{real}} = \text{CPI}_{\text{ideal}} + \text{CPI stall}$
  - » $\text{CCT}_{\text{real}} = \text{Time}_{\text{longest pipestage}} + \text{Time}_{\text{latch overhead}}$

Pipeline example: 3 stages

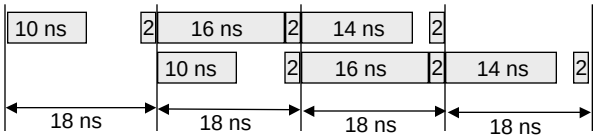
Assume a 2 nsec latch delay.

Unpipelined



Latency = 42 nsec
Throughput = 1/42 nsec

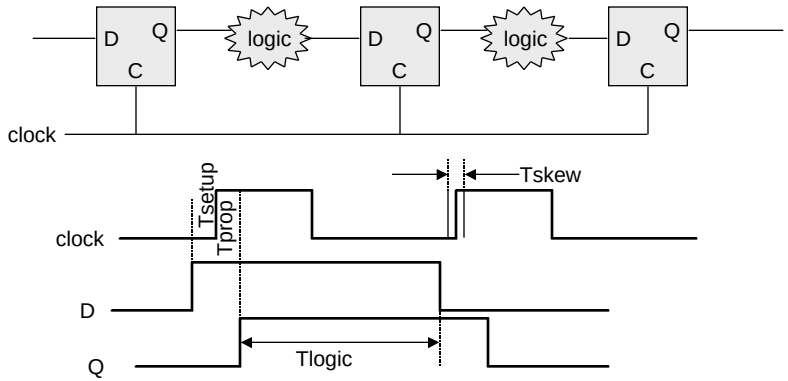
Pipelined



Latency = 3 x 18 = 54 nsec (longer!)
Throughput = 1/18 nsec (2.3x, not 3x)

Clock overhead and reliable clocking

Single-phase edge-triggered flipflops (data sampled on rising edge)



Period >= Tlogic + Tsetup + Tprop + Tskew

Tskew from: wire delay, buffers, non-vertical edges with varying trigger points

Pipelining DLX: What are the stages?

- Divide into stages so that
  - » Time of each stage is about the same
  - » Each stage is used at most once in each instruction
  - » Registers can be easily defined to hold the state information between stages
  - » Not so many stages that latch overhead dominates
  - » Not so few stages that we give up performance that could be gained

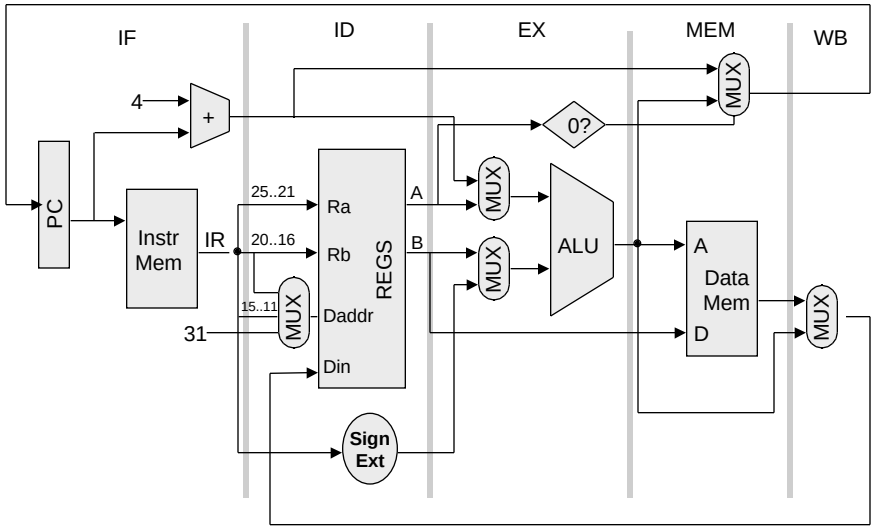
- For DLX, choose 5 stages:
  - » IF: Instruction fetch
  - » ID (&RF): Instruction decode and register fetch
  - » EX: execution of ALU operation
  - » MEM: Memory access
  - » WB: Write back to register file

Pipelined Instruction Execution

Instruction	1	2	3	4	5	6	7	8	9
i	IF	ID	EX	MEM	WB				
i+1		IF	ID	EX	MEM	WB			
i+2			IF	ID	EX	MEM	WB		
i+3				IF	ID	EX	MEM	WB	
i+4					IF	ID	EX	MEM	WB

Looks Simple!

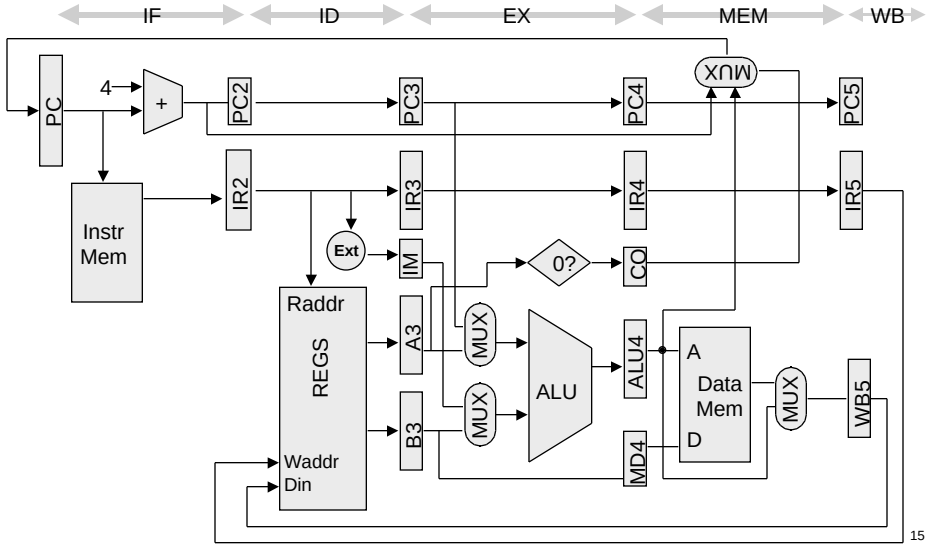
DLX pipeline division (with bugs!)



Rules for pipeline registers

- Each stage must be independent, so the inter-stage registers must hold:
  - » data values
  - » control signals, including
    - – Decoded instruction fields
    - – MUX controls
    - – ALU controls
- Think of the register file as two independent units:
  - » Read file, accessed in ID
  - » Write file, accessed (simultaneously!) in WB
- There is no “final” set of registers after WB, (WB/IF) because the instruction is finished and all results are recorded in permanent machine state (register file, memory, and PC).

A more accurate Pipeline diagram



Pipelined dataflow: The details

	Reg-Reg ALU	*Reg-immed ALU*	*Load*	*Store*	*Branch*	*Jump*
IF	IR2=IMem[PC]; PC2=PC=PC+4;					
ID	A3=Regs[IR25..21]; B3=Regs[20..16]; IR3=IR2; PC3=PC2; IM3=IR2[15]^{16}##IR2[14..0];					
EX	ALU4= A3 op B3; IR4=IR3; PC4=PC3;	ALU4= A3 op IM3; IR4=IR3; PC4=PC3;	ALU4 =A3 + IM3; IR4=IR3; PC4=PC3; MD4=B3;		ALU4 =PC3+IM3; CO4= A3 op 0; IR4=IR3; PC4=PC3;	ALU4 = PC3+IM3; IR4=IR3; PC4=PC3;
MEM	IR5=IR4; PC5=PC4;	IR5=IR4; PC5=PC4;	WB5 = DMem[ALU4];	DMem[ALU4] = MD4;	IR5=IR4; PC5=PC4; if (CO4) PC=ALU4;	IR5=IR4; PC5=PC4; PC=ALU4;
WB	Din=WB;	Din=WB;	Din=WB;			