LECTURE - 05

DLX

- DLX pronounced “D eluxe”
- Has the features of many recent experimental and commercial machines
- [AMD 29K, DECstation 3100, HP 850, IBM 801, Intel i860, MIPS M/120A, MIPS M/1000, Motorola 88K, RISC I, SGI 4D/60, SPARCstation-1, Sun-4/110, Sun-4/260]/13 = 560 = DLX (Roman)
- Good architectural features (e.g. simplicity), easy to understand

DLX Architecture: Registers and Data Types

- Has 32 32-bit GPRs: R0...R31
- Also, FP registers
  - 32 single precision: F0...F31
  - Or, 16 double precision: F0, F2, ... F30
- Value of R0 is always ZERO!
- Data types:
  - Integer: bytes, half-words, words
  - FP: single/double precision

DLX Memory Addressing

- Uses 32-bit, big-endian mode
- Addressing modes:
  - Only immediate and displacement, with 16-bit fields
    - Register deferred?
      - Place zero in displacement field
    - Absolute
      - Use R0 for the register

DLX Instruction Format

Opcode (6)	RS1 (5)	RD (5)	Immediate (16)

I-type instruction: loads, stores, all immediates, conditional branch, jump register, jump and link register

Opcode (6)	RS1 (5)	RS2 (5)	RD (5)	Func (11)

R-type instruction: register-register ALU operations

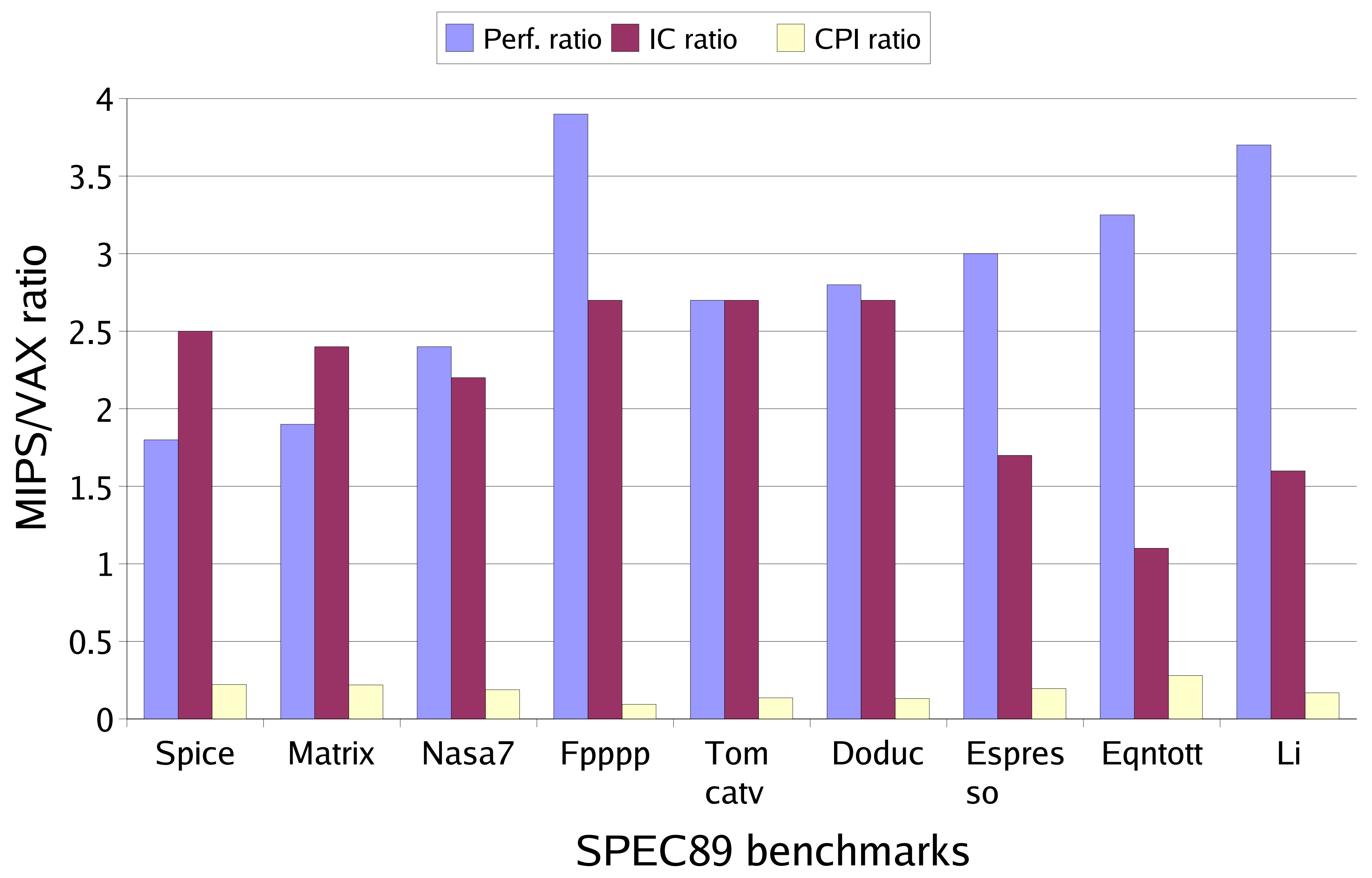
Opcode (6)	Offset relative to PC (26)

J-type instruction: jump, jump and link, trap and return

DLX Operations

- Four classes: Load/store, ALU, branch, FP
- ALU instructions are register-register
- R0 used to synthesize some operations:
  - Examples: loading a constant, reg-reg move
- Compares “set” a register
- Jump and link pushes next PC onto R31
- FP operations in single/double precision
- FP compares set a bit in a special status reg
- FP unit also used for integer multiply/divide!

DLX Performance: MIPS vs VAX



Pipelining

- Its natural!
- Laundry example... (Randy Katz's slides)
- DLX has a simple architecture
  - Easy to pipeline
- Pipelining speedup:
  - Can be viewed as reduction in CPI
  - Or, reduction in clock cycle
    - Defining clock cycle as the amount of time between two successive instruction completions

A Simple DLX Implementation

- **Instruction Fetch (IF) cycle:**
  - IR <-- M[PC]
  - NPC <-- PC + 4
- **Instruction Decode (ID) cycle:**
  - Done in parallel with register read (fixed field decode)
  - Register/Immediate read:
    - A <-- R[IR6..10]
    - B <-- R[IR11..15]
    - Imm <-- sign-extend(IR16..31)

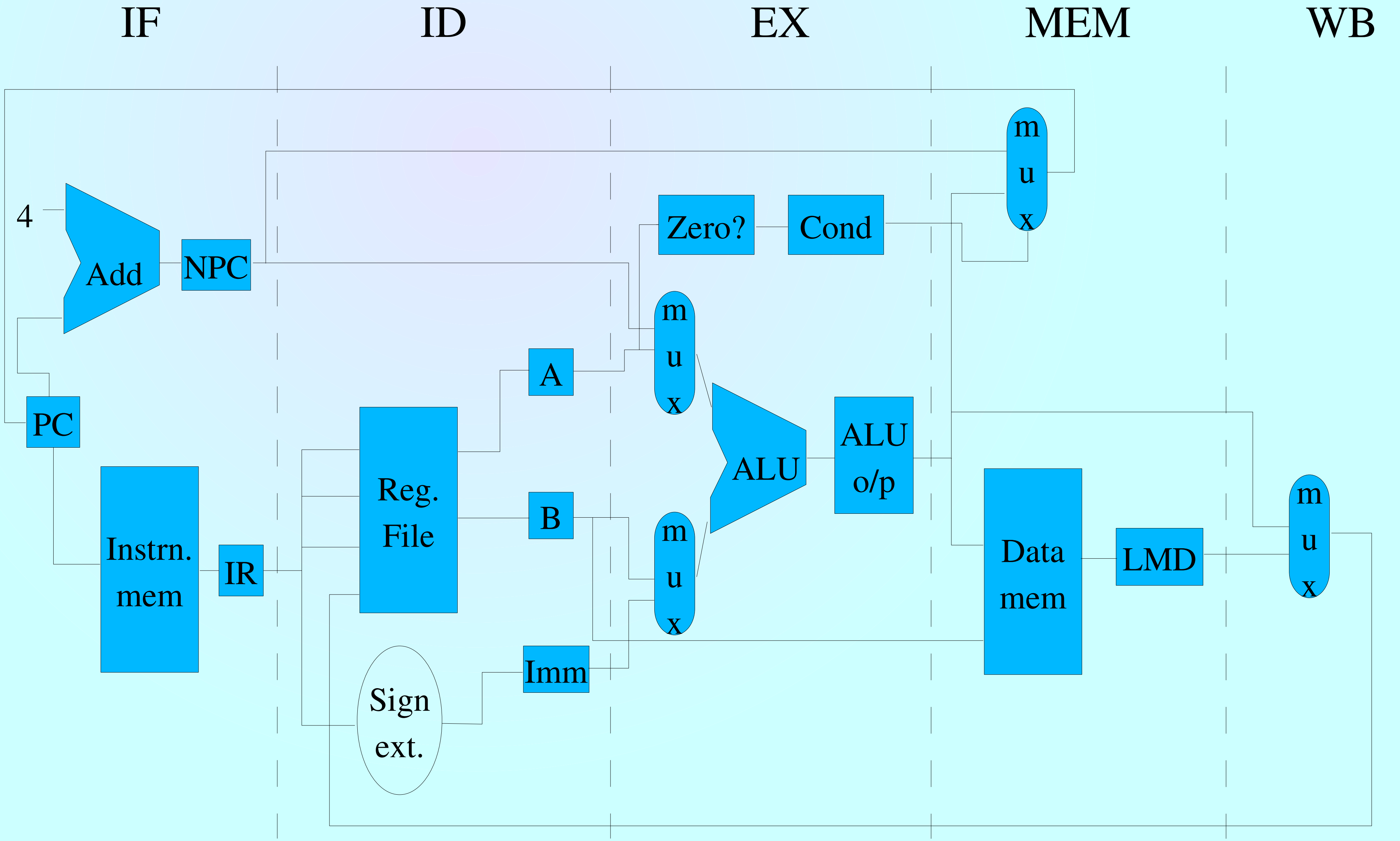
A Simple DLX Implementation (continued)

- **Execution/effective address (EX) cycle:**
  - Memory reference:
    - ALUOutput <-- A + Imm
  - Register-register ALU instruction:
    - ALUOutput <-- A func B
  - Register-immediate ALU instruction:
    - ALUOutput <-- A op Imm
  - Branch:
    - ALUOutput <-- NPC + Imm
    - Cond <-- A op 0 [op is one of == or !=]

A Simple DLX Implementation (continued)

- **Memory access/branch completion (MEM) cycle:**
  - Memory access:
    - LMD <-- M[ALUOutput]
    - Or, M[ALUOutput] <-- B
  - Branch: PC = (cond) ? ALUOutput : NPC
- **Write-back (WB) cycle:**
  - Reg-reg ALU opn: R[IR16..20] <-- ALUOutput
  - Reg-imm ALU opn: R[IR11..15] <-- ALUOutput
  - Load instruction: R[IR11..15] <-- LMD

The DLX Data-path



Further lectures...

- Pipelining this data-path
- Pipelining issues