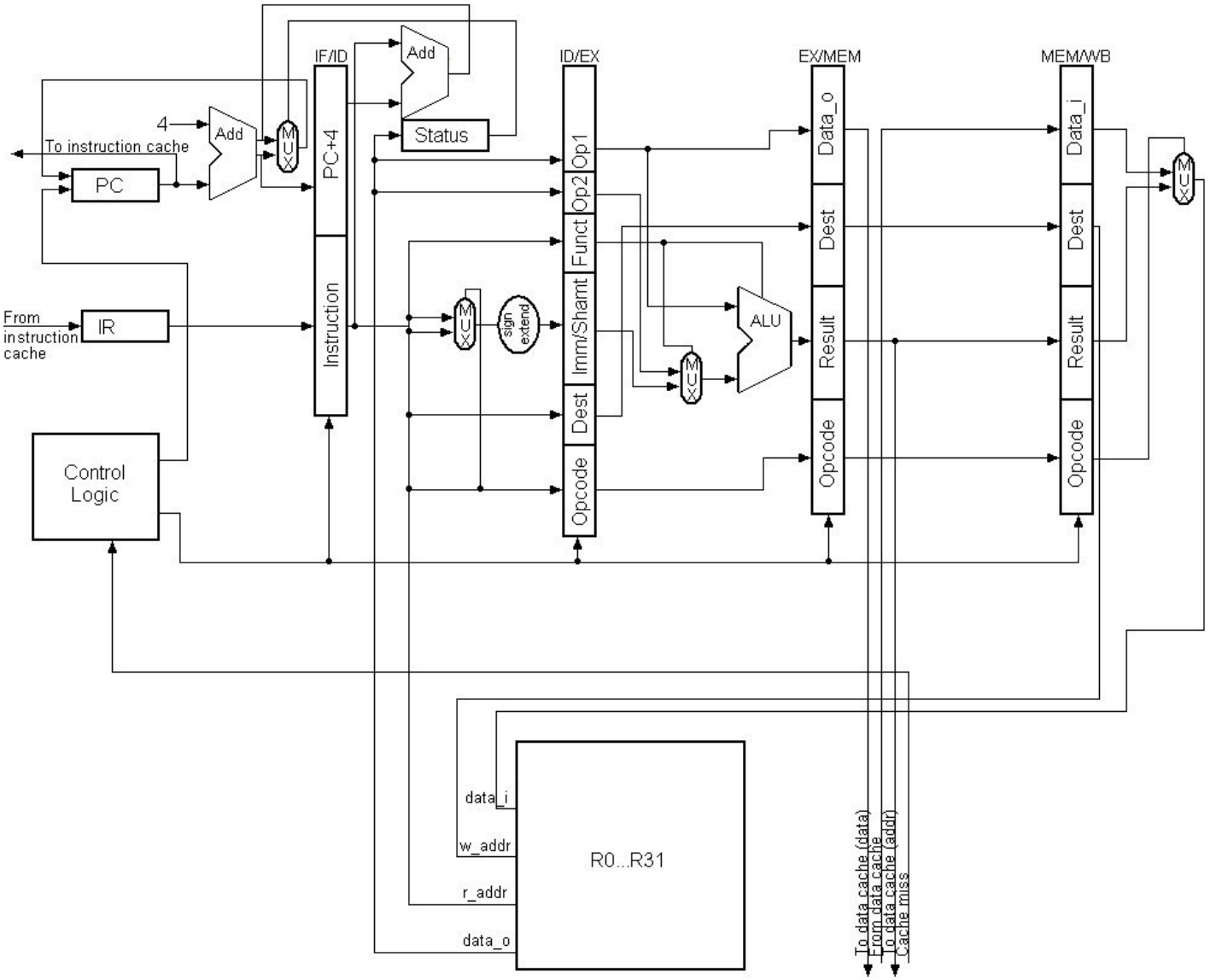
Luigi32

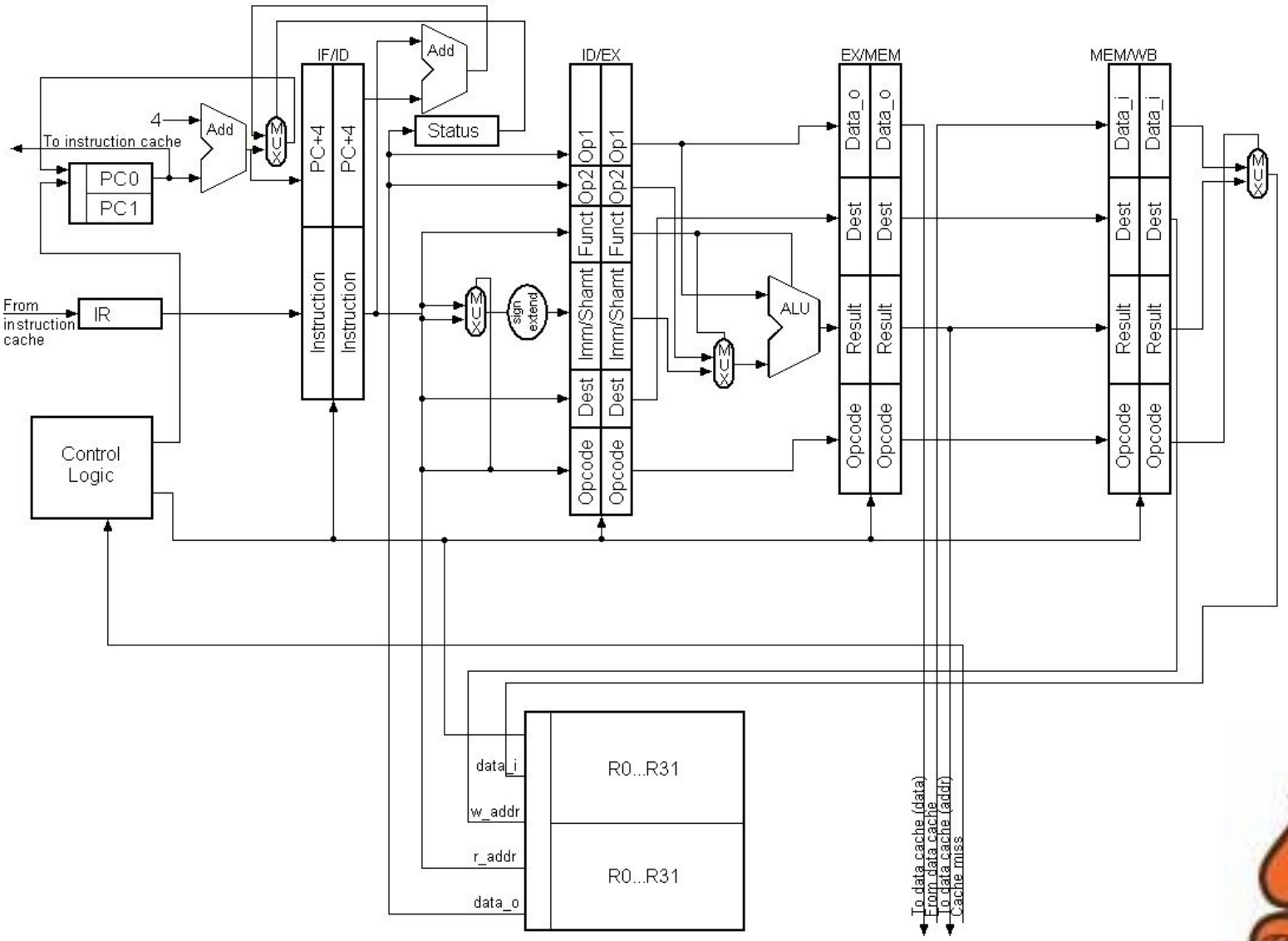
A Modified MIPS Machine

Amy Freestone
Zachary Freudenburg
Shakir James
Amanda Juarez

Luigi32 without 2-Thread Support



Luigi32 with 2-Thread Support

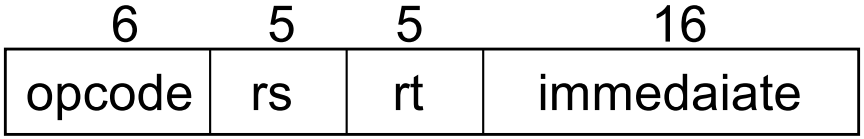


Instruction Set Architecture

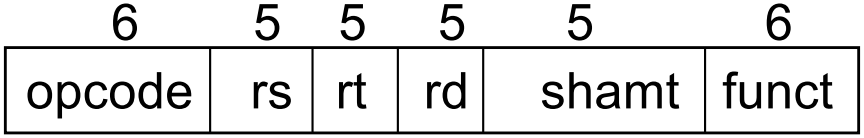
1. Instruction Types
2. Addressing Modes

Instruction Set Architecture

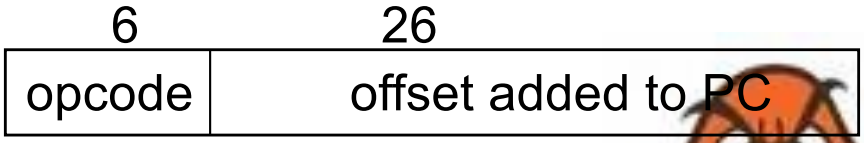
- Three instruction types:
  - I-type



    - Loads/stores from memory and all immediates
    - Jump register command
    - Conditional branch instructions
  - R-type

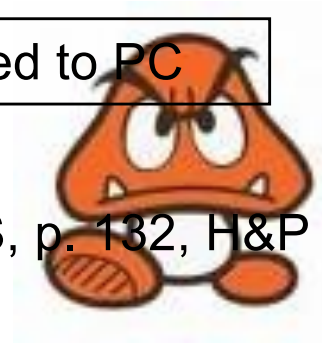


    - Register to register ALU operations.
  - J-type



    - Jump instructions.

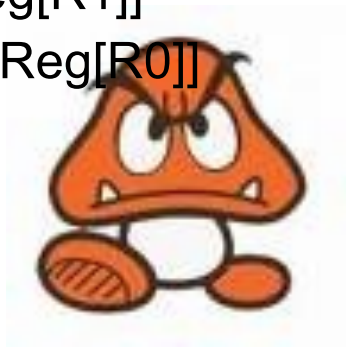
Based on Fig 2.27 Instruction layout for MIPS, p. 132, H&P



Instruction Set Architecture

- Two addressing modes for load/store operations:
  - Immediate
    - Load a value into a register
    - e.g. LI R1,2 ; Reg[R1] <-- 2
  - Displacement
    - accessing local variables
    - simulating register indirect, direct addressing modes
      (by using a register that contains zero and a 16-bit immediate)
    - e.g.

```
LW R2, 100(R1)    ; Reg[R2] <-- Mem[100+Reg[R1]]
LW R2, 0(R1)      ; Reg[R2] <-- Mem[0+Reg[R1]]
LW R2, 100(R0)    ; Reg[R2] <-- Mem[100+Reg[R0]]
```



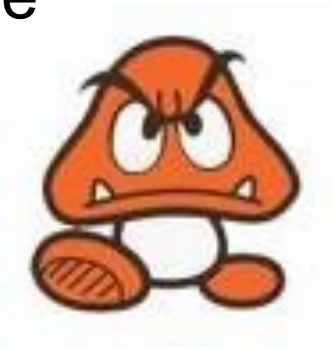
Pipeline Stages and State

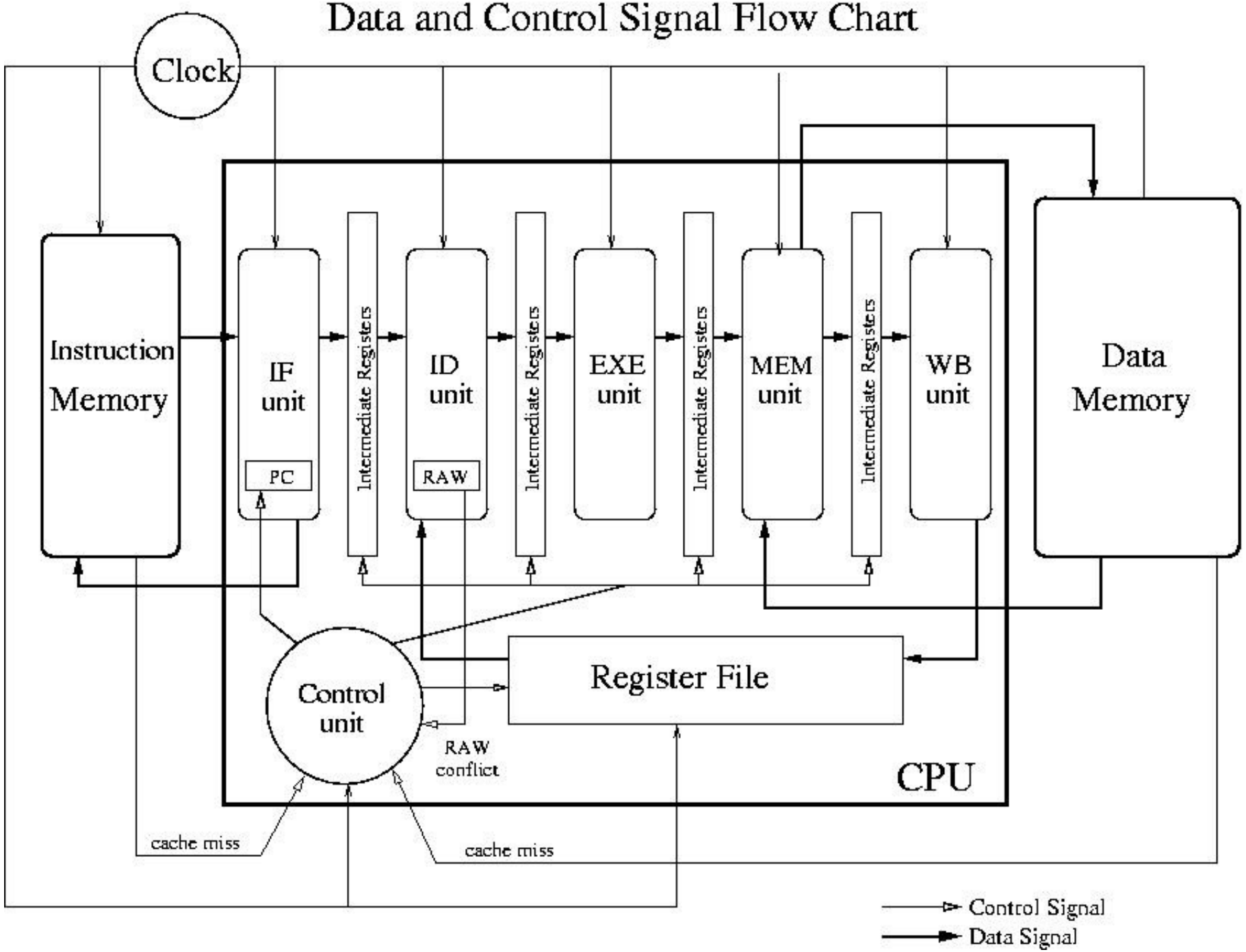
- **Instruction Fetch (IF)**
  - Fetch instruction from memory
  - Increment PC by 4 or load branch target
- **Instruction Decode (ID)**
  - Decode instruction
  - Load operands from register file
  - Resolve branches
- **Execute (EX)**
  - Perform ALU operations
  - Calculate memory address for memory operations
- **Memory Access (MEM)**
  - Access memory for memory operations
- **Register Write-back (WB)**
  - Write result to register file

- **IF/ID**
  - PC + 4
  - Instruction
- **ID/IF**
  - Branch status
  - Branch target
- **ID/EX**
  - PC + 4
  - Operands
  - ALU function
  - Immediate value / shift amount
  - Destination register
  - Opcode
- **EX/MEM**
  - Data to memory
  - Destination register
  - ALU result
  - Opcode
- **MEM/WB**
  - Data from memory
  - Destination register
  - ALU result
  - Opcode

Pipeline Hazards

- WAW, WAR, and RAR are not an issue since all instructions are issued in order.
- RAW dependency remains
  - Read in ID stage and write in WB stage
    - Two stages apart
  - Write back value ready at end of EXE stage
    - Can forward back to ID in time for next instruction
    - We use a single ‘no-op’ bubble, due to possible thread switching issues




Data and Control Signal Flow Chart
Clock
Instruction Memory
IF unit
PC
Intermediate Registers
ID unit
RAW
Intermediate Registers
EXE unit
Intermediate Registers
MEM unit
Intermediate Registers
WB unit
Data Memory
Control unit
Register File
RAW conflict
CPU
cache miss
cache miss
Control Signal
Data Signal

Questions?