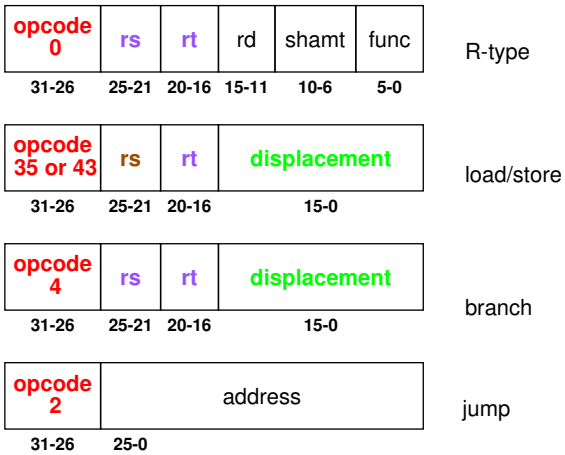
Control Unit

CPU hardware that controls instruction execution

- sends signals to the datapath to operate it
- **hardwired**
  - an implementation of control that uses **PLAs** (programmable logic arrays) & random logic
- **microprogrammed**
  - an implementation of control that uses a small **ROM** (read-only memory) of simple instructions (**microcode** instructions) to drive the signals needed to execute instructions in the ISA

Instruction Formats

Many control signals driven by the instruction



Regularity of the MIPS formats

- **opcode** always in bits 31-26 (Op[5-0])
- **source registers** are always rs & rt
- **base register** always rs
- **branch offset** always bits 15-0

Control Signals

Register file

- register write signal (**RegWrite**: asserted for R-type & load)
- register destination field (**RegDst**: rt, rd)
- results value (**MemToReg**: loaded value, R-type result)
  - all generated by the opcode

ALU

- type of the second operand (**ALUSrc**: register, immediate)
  - generated by the opcode
- ALU operation **ALUOp**: add, subtract, and, or, set-on-less-than)
  - generated by a small control unit
    - inputs: opcode & func field
    - output: ALU operation
    - lw/sw ⇒ add, beq ⇒ subtract, or R-type ⇒ func value

Memory

- read signal (**MemRead**)
- write signal (**MemWrite**)
  - both generated by the opcode

Control Signals

Branch control

- new PC value (**PCSrc**: incremented PC, target address)
  - generated by the opcode AND'd with Zero

Jump control

- new PC value (**Jump**: incremented PC or target address, jump address)
  - generated by the opcode

Changing the Implementation

How should you approach a problem in which you had to redesign the implementation to include another instruction?

- What does the instruction do?
- What parts of the datapath does it need?
  - Can it use what is there already?
  - What new logic or registers does it need?
- How is the datapath activated?
  - What control lines does it need
  - Where should the control lines come from?