

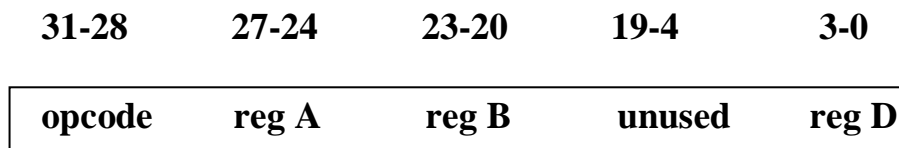
Datapath and Control (Ch 5)

Path from source to object

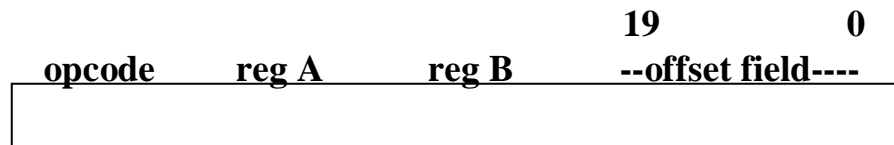
- source file (.c file)
- compiler (.s file)
- assembler (.o file)
- linker (stitch together individual modules)
- object file (a.out file)
 - header, text, data, relocation info, symbol table, debugging info
- loader (from disk to memory)
- memory layout

LC-2200

- R-type (add, nand)



- I-type (lw, sw, beq, addi)



- J-type (jalr)



- O-type (halt, noop)



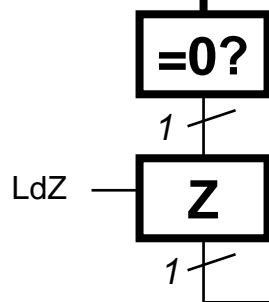
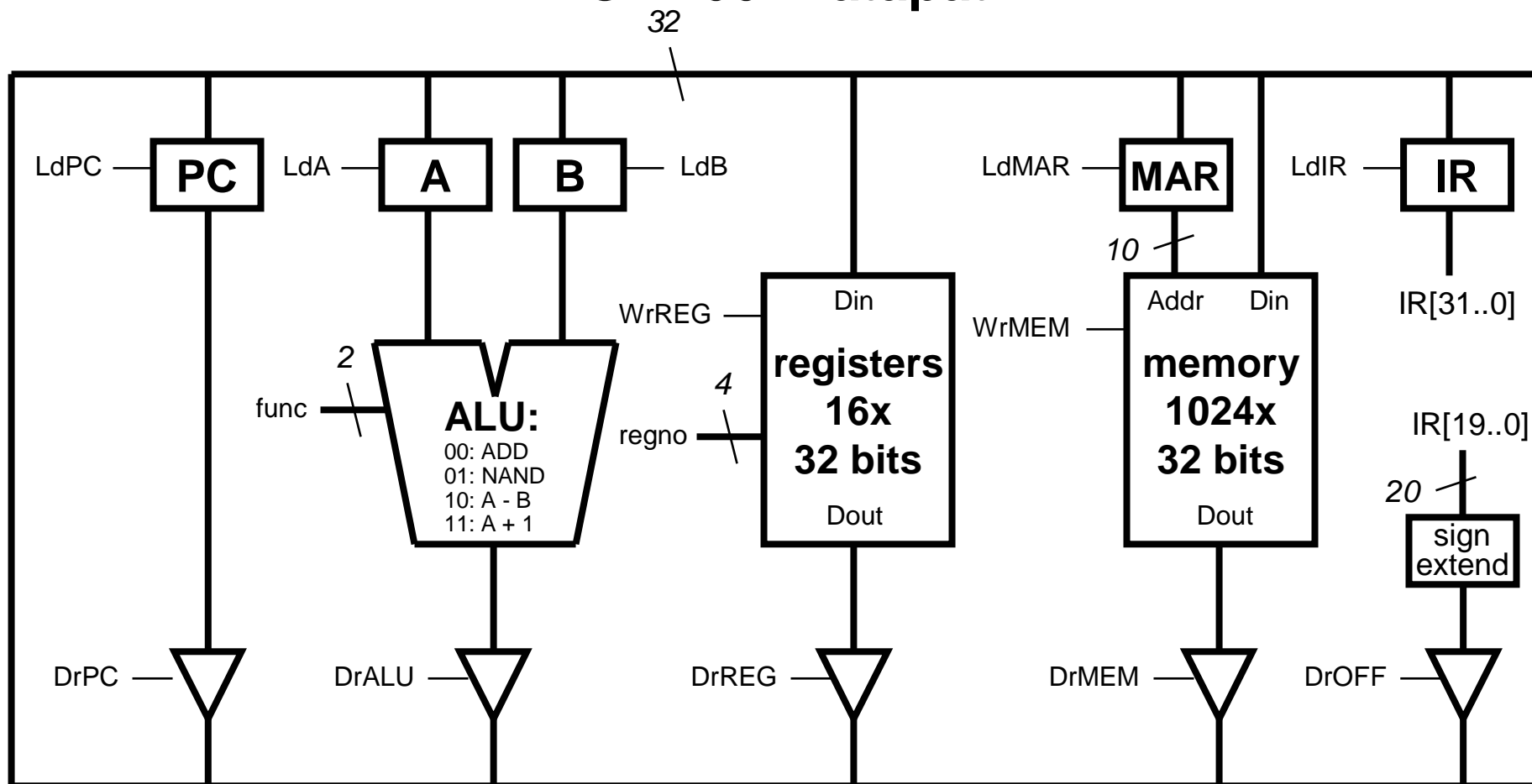
Key Hardware Concepts

- clock
- circuits
 - combinational, sequential
- state
- hardware resources
- connections
 - buses: single, two, three
 - asynchronous vs. synchronous
- FSM
 - predetermined inputs
 - context dependent inputs
- control signals

CPU organization

- Datapath and control
 - combination of resources and their connections
 - how to decide the datapath?
 - What resources do we need in the datapath?
- A simple datapath for LC-2200
 - PC, IR, register file, memory, decoder
 - ALU
- Single bus datapath

LC-2200 Datapath

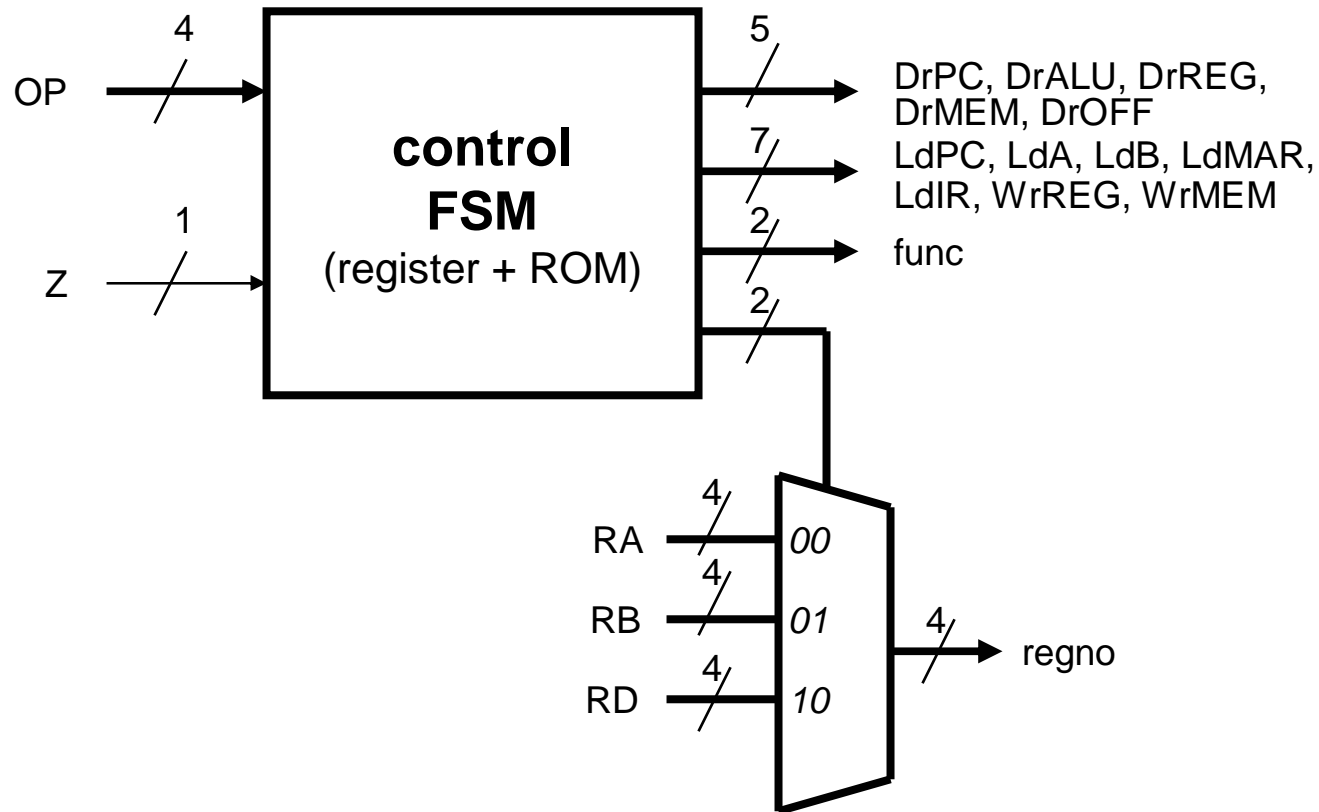


- $IR[27..24]$ \longrightarrow RA: 4-bit register number to control logic
- $IR[23..20]$ \longrightarrow RB: 4-bit register number to control logic
- $IR[3..0]$ \longrightarrow RD: 4-bit register number to control logic
- $IR[31..28]$ \longrightarrow OP: 4-bit opcode to control logic
- Z \longrightarrow Z: 1-bit boolean to control logic

CPU registers

- IR, PC
 - where does each get its data input?
 - how do we use each of its contents?
 - control signals needed to actuate each?
- GPR or register file
 - external interface?
 - internal logic?
 - data input?
 - load vs. arithmetic instructions
 - control signals needed?
 - multiplexers and their role

LC-2200 Control Logic



- Why do we need buffers at ALU input?
- Why MAR?

Bus-based Datapath

- data bus
- address bus
- units electrically connected to the bus
 - what does that mean?
 - how to ensure only one driving the bus at a time?

Control Design

- What is involved in instruction execution?
 - fetch, decode, execute, fetch operands, store operands
- designing the control unit
 - given instruction set, and DP
 - write flowcharts: this is no different from writing C code!
 - adopt a control regime
 - hardwired
 - » FSM
 - microprogrammed control
 - » stored program

Hardwired Control

- What states do we need in the FSM?
- Fetch macro operation
 - PC to Memory; Read memory data into IR; Increment PC
 - Can we do all this in one cycle?
 - Actions during instruction fetch - ifetch1
 - PC to BUS
 - BUS into MAR
 - PC to bus: how to effect this?
 - Bus to MAR: how to effect this?

Hardwired Control

- What states do we need in the FSM?
 - Actions during instruction fetch - ifetch2
 - Want to put instruction from memory onto bus
 - Memory to bus: how to effect this?
 - Want to store bus into IR: how to effect this?

Hardwired Control

- What states do we need in the FSM?
 - Actions during instruction fetch - ifetch3
 - Want to increment PC
 - PC to bus: how to effect this?
 - Bus to A: how to effect this?

Hardwired Control

- What states do we need in the FSM?
 - Actions during instruction fetch - ifetch4
 - Want to increment PC
 - Increment A: how to effect this?
 - ALU to Bus: how to effect this?
 - Bus to PC: how to effect this?

- What next?
 - 4 different FSM sequences depending on instruction type
 - R-type, I-type, J-type, O-type

- R-type FSM
 - select regA and regB from register file: how?
 - ALU op (ADD or NAND); store result in regD: how?
 - can all of the above be done in one cycle?
- Where do we go from here?

- I-type FSM: load/store
 - address arithmetic: regA plus offset
 - result into MAR
 - load
 - read memory onto bus
 - latch into regB
 - store
 - drive regB onto the bus
 - write memory
 - back to F1

- I-type FSM: BEQ
 - subtract regA and regB
 - if equal
 - latch 1 into Z (if *all* ALU bits are 0)
 - latch 0 into Z (if *any* ALU bit is a 1)
 - if Z is 1 then
 - then PC+offset to PC
 - where did the +1 go?
 - else nothing
 - back to F1

- J-type FSM: JALR
 - PC to register regB specified in IR
 - contents of register regA in IR to PC
 - back to F1

- O-type FSM: HALT
 - Quiesce state