Arch: Single Cycle Processor

Alex Chi

Update: April 7, 2020

Contents

1 How to design a processor

- Analyze instruction set (data path requirements) using RTL
  - refer to slides for examples
  - ADDU $R[rd] \leftarrow R[rs] + R[rt], PC \leftarrow PC + 4$

1.1 Requirements of the Instruction

- mem: inst & data
- registers: read rs, read rt, write rt or rd
- pc
- sign / zero extender
- ALU
- add 4

1.2 Components of the Data Path

- combinational elements
  - adder (A, B, CarryIn → Sum, Carry)
  - MUX (A, B, Select → Y)
  - ALU (A, B, Op → result)
- storage elements
  - register file (RA, RB, RD, busW, WE, Clk → busA, busB)
  - ideal memory (Address, WE, data in, Clk → data out)

1.3 Clocking Methodologies

- state elements: register
- edge-triggered: all state changes occur on a clock edge
- cycle time = qlk-to-Q + longest delay path + setup + clock skew

1.4 Generic Steps of Datapath

- Instruction Fetch
- Decode
- Execute
- Memory
- Register Write

2 Design

2.1 Fetch Instruction

- PC = PC + 4
- Instruction = M[PC]

2.2 Decoding Instruction

- rs: read addr 1
- rt: read addr 2
- rd: write addr
- use MUX to wire rt to write addr (lw, sw)

2.3 Data path for R-type

- add: R[rd] = R[rs] op R[rt]
- ori: R[rt] = R[rs] op imm (zero extended)
- lw: Addr = R[rs] + signext(imm), R[rt] = M[Addr]
- sw: Addr = R[rs] + signext(imm), M[Addr] = R[rt]
- beq: PC = PC + 4 + offset if rs = rt else PC + 4
- try to draw the graph by yourself!

2.4 Decoding

- ALU
  - op code 2:0 for ALUop
  - func 3:0 for sub-op
  - just recite the table by induction: what operation should ALU do?
- Main Logic
  - use a lot of and gate to identify op
  - assemble main control (PLA)