Lecture 20, Oct 25, 2022

Simple Processor Continued

- Instructions:
  - `load Rx, Data`
    - Load `Data` into `Rx`, where `Data` comes from the external signal
    - Need to enable the external signal input tri-state buffer, and enable a write to the correct register
  - `move Rx, Ry`
    - Copies data from `Ry` into `Rx`
    - Need to enable the tri-state buffer on the output of `Ry`, and enable a write to `Rx`
  - `add Rx, Ry`
    - Store `Rx + Ry` into `Rx`
    - First enable the tri-state on output `Rx`, and enable the temporary `A` register for the ALU
    - On the second clock cycle, enable the tri-state on output `Ry`, the ALU does the computation and stores into `G`
    - On the third clock cycle, store `G` into `Rx`
  - `sub Rx, Ry`
    - Store `Rx - Ry` into `Ry`
    - Same thing as the add instruction but subtraction
- Tradeoff between instruction usability and complexity
  - e.g. old processors used to have very complex instructions that compilers could not always take advantage of
- Each instruction and register has an encoding
  - 00 for load, 01 for move, 10 for add, 11 for sub
  - These are referred to as “opcodes”, in this case 2-bit opcodes
  - Registers are encoded as the register number (register index)
  - e.g. `add R1, R2` is encoded as `10, 01, 10`
- When designing the processor we need to know how many steps (clock cycles) can instructions take
  - The longest instruction is add/sub at 3 steps
  - Therefore we need a 2-bit counter to count which step we’re currently on
    - Counter with a clock, clear, and produces Q_1, Q_0
- Now turn those 2 bits into a 1-hot code
  - 1-hot codes make the control FSM logic much easier to derive
- For the control FSM:
  - Need a function register, taking in f_1, f_0, the opcode, 2-bit values for R_x and R_y, an input FR_{in} which is used to indicate when we’re loading a new instruction
  - First decoder decodes the opcode into a 1-hot code for each instruction
  - Second and third decoders decode the register inputs into 1-hot codes
- We then derive control signals for each step
  - $A_{in} = (I_2 + I_3)T_1$
  - $G_{in} = (I_2 + I_3)T_2$
  - $G_{out} = (I_2 + I_3)T_3$
  - extern $= I_0T_1$
  - Done $= (I_0 + I_1)T_1 + (I_2 + I_3)T_3$
  - FR$_{in} = wT_0$
  - Clear $=$ Done $+ \bar{w}T_0$
    - $\bar{w}T_0$ means if we’re in state 0 and we aren’t starting an operation, we stay in state 0