 © 2000-2022 Steven S. Lumetta. All rights reserved.

ECE120: Introduction to Computer Engineering

Notes Set 3.8 Summary of Part 3 of the Course

Students often find this part of the course more challenging than the earlier parts of the course. In addition to these notes, you should read Chapters 4 and 5 of the Patt and Patel textbook, which cover the von Neumann model, instruction processing, and ISAs.

You should recognize all of these terms and be able to explain what they mean. For the specific circuits, you should be able to draw them and explain how they work. Actually, we don't care whether you can draw something from memory—a mux, for example—provided that you know what a mux does and can derive a gate diagram correctly for one in a few minutes. Higher-level skills are much more valuable.

- digital systems terms
  - module
  - fan-in
  - fan-out
  - machine models: Moore and Mealy
- simple state machines
  - synchronous counter
  - ripple counter
  - serialization (of bit-sliced design)
- finite state machines (FSMs)
  - states and state representation
  - transition rule
  - self-loop
  - next state (+) notation
  - meaning of don't care in input combination
  - meaning of don't care in output
  - unused states and initialization
  - completeness (with regard to FSM specification)
  - list of (abstract) states
  - next-state table/state transition table/state table
  - state transition diagram/transition diagram/state diagram
- memory
  - number of addresses
  - addressability
  - read/write logic
  - serial/random access memory (RAM)
  - volatile/non-volatile (N-V)
  - static/dynamic RAM (SRAM/DRAM)
  - SRAM cell
  - DRAM cell
  - design as a collection of cells
  - coincident selection
- von Neumann model
  - processing unit
    - register file
    - arithmetic logic unit (ALU)
    - word size
  - control unit
    - program counter (PC)
    - instruction register (IR)
    - implementation as FSM
  - input and output units
  - memory
    - memory address register (MAR)
    - memory data register (MDR)
  - processor datapath
  - bus
  - control signal
- tri-state buffer
  - meaning of Z/hi-Z output
  - use in distributed mux
- instruction processing
  - fetch
  - decode
  - execute
  - register transfer language (RTL)
- Instruction Set Architecture (ISA)
  - instruction encoding
  - field (of an encoded instruction)
  - operation code (opcode)
  - types of instructions
    - operations
    - data movement
    - control flow
  - addressing modes
    - immediate
    - register
    - PC-relative
    - indirect
    - base + offset

We expect you to be able to exercise the following skills:

- Transform a bit-sliced design into a serial design, and explain the tradeoffs involved in terms of area and time required to compute a result.
- Based on a transition diagram, implement a synchronous counter from flip-flops and logic gates.
- Implement a binary ripple counter (but not necessarily a more general type of ripple counter) from flip-flops and logic gates.
- Given an FSM implemented as digital logic, analyze the FSM to produce a state transition diagram.
- Design an FSM to meet an abstract specification for a task, including production of specified output signals, and possibly including selection of appropriate inputs.
- Complete the specification of an FSM by ensuring that each state includes a transition rule for every possible input combination.
- Compose memory chips into larger memory systems, using additional decoders when necessary.
- Encode LC-3 instructions into machine code.
- Read and understand programs written in LC-3 assembly/machine code.

At a higher level, we expect that you understand the concepts and ideas sufficiently well to do the following:

- Abstract design symmetries from an FSM specification in order to simplify the implementation.
- Make use of a high-level state design, possibly with many sub-states in each high-level state, to simplify the implementation.
- Use counters to insert time-based transitions between states (such as timeouts).
- Implement an FSM using logic components such as registers, counters, comparators, and adders as building blocks.
- Explain the basic organization of a computer's microarchitecture as well as the role played by elements of a von Neumann design in the processing of instructions.
- Identify the stages of processing an instruction (such as fetch, decode, getting operands, execution, and writing back results) in a processor control unit state machine diagram.

And, at the highest level, we expect that you will be able to do the following:

- Explain the difference between the Moore and Mealy machine models, as well as why you might find each of them useful when designing an FSM.
- Understand the need for initialization of an FSM, be able to analyze and identify potential problems arising from lack of initialization, and be able to extend an implementation to include initialization to an appropriate state when necessary.
- Understand how the choice of internal state bits for an FSM can affect the complexity of the implementation of next-state and output logic, and be able to select a reasonable state assignment.
- Identify and fix design flaws in simple FSMs by analyzing an existing implementation, comparing it with the specification, and removing any differences by making any necessary changes to the implementation.