Learning Objectives

CIS 351 — Fall 2021

Number Representation

- NR1 **(Core)** Binary
  1. Convert from positive decimal integers to binary
  2. Convert from binary to positive decimal integers
  3. Determine the range of non-negative integers that can be represented using n binary bits
  4. Determine the number of bits necessary to store positive integers up to x
  5. Continue a sequence of binary values (up or down) without converting to decimal (or any other base)
- NR2 **(Core)** Two's Complement
  1. Convert from any decimal integer to two's complement (using the correct number of leading 1s for the given number of bits)
  2. Convert from two's complement to any decimal integer
  3. Determine the number of bits necessary to store integers between x and y in two's complement form
  4. Continue a sequence of two's complement values (up or down) without converting to decimal (or any other base)
- NR3 Hexadecimal
  1. Convert directly from binary to hexadecimal (i.e., do not convert to decimal or any other representation as an intermediate step)
  2. Convert directly from hexadecimal to binary (i.e., do not convert to decimal or any other representation as an intermediate step)
  3. Continue a sequence of hexadecimal values (up or down) without converting to decimal (or any other base)
- NR4 Other Number Bases
  1. Convert from base 10 to any base (positive integers only)
  2. Convert from any base to base 10 (positive integers only)

Combinatorial Logic

- CL1 **(Core)** Create a truth table that describes a problem (e.g., a truth table that represents the desired outputs of a 4-bit "isPrime" circuit)
- CL2 **(Core)** Boolean Expressions
  1. Complete a truth table that describes the output of a Boolean Expression
  2. Write a Boolean expression in sum-of-products form that describes a given truth table
- CL3 **(Core)** Circuit Representation
  1. Draw a circuit that implements a given truth table
  2. Complete a truth table that describes the output of a given combinatorial circuit
  3. Draw a circuit that represents the implementation of a given Boolean expression
  4. Write a Boolean expression that describes the behavior of a given combinatorial circuit
- CL4 **(Core)** Determine the propagation delay of a combinatorial circuit
- CL5 **(Core)** Design a combinatorial circuit that uses basic gates, multiplexors, demultiplexors, adders, and subtractors.

Addition

- ADD1 **(Core)** Directly compute the sum of two numbers written in binary
- ADD2 **(Core)** Demonstrate knowledge of the operation of a ripple-carry adder
- ADD3 Demonstrate knowledge of the operation of a carry-lookahead adder
- ADD4 Demonstrate knowledge of the operation of a carry-select adder
- ADD5 Recognize when the addition of x and y will produce overflow
- ADD6$^{\text{p}}$ Demonstrate how to detect overflow when adding two numbers

Functional Completeness

- FC1 Write the outline of a proof of logical/functional completeness. This outline should clearly indicate that the student knows which "direction" the proof should go (i.e, that you use NAND to build AND, OR, NOT as opposed to using AND, OR, NOT to build NAND).
- FC2$^{\text{h}}$ Write a complete proof of logical/functional completeness.

Boolean Algebra

- BA1 **(Core)** Simplify a Boolean expression by applying DeMorgan's laws as well as the commutative, associative, and distributive properties to a Boolean expression (including factoring) then canceling terms when appropriate
- BA2 Simplify a Boolean expression using a Karnaugh Map

Sequential Logic

- SL1 Latches
  1. Explain how a feedback loop can be used to store data
  2. Complete the characteristic table of a sequential circuit that uses feedback but no clock (i.e.,the NOR latch)
  3. Incorporate a latch into a more complex sequential circuit (e.g., incorporate a NOR-latch into a D-Latch)
  4. Add a clock to a latch
- SL2 Flip-Flops
  1. Explain the operation of a master-slave (or similar) flip-flop (e.g., what gives the flip-flop its "instant" trigger point).
  2. Use a timing diagram to demonstrate the internal behavior of a master-slave flip-flop.
  3. Describe the behavior of a master-slave flip-flop (e.g., by completing a timing diagram)
- SL3[l] Demonstrate what goes wrong with traditionally-designed combinatorial circuits if clocks aren't present.
- SL4 **(Core)** Trace the behavior of a a sequential circuit that contains a flip-flop and a clock
  1. Write the characteristic table of a sequential circuit that contains a flip-flop and a clock
  2. Draw the timing diagram for a sequential circuit with given input and initial state
- SL5[i] Construct a circuit diagram given a characteristic table
- SL6 **(Core)** Design a sequential circuit that uses flip-flops, basic gates, multiplexors, demultiplexors, adders, and subtractors.

Assembly Language

- AL1 **(Core)** Write "basic" assembly by hand
  1. Translate a single, complex arithmetic expression (e.g. $x = (y + z) - (w + t)$) into a sequence of assembly statements.
  2. Translate an `if-else` statement into a sequence of assembly statements
  3. Translate a `for` loop into a sequence of assembly statements
- AL2 **(Core)** Write "intermediate" assembly by hand
  1. Write an assembly function that iterates over an array or string and examines/processes each item (e.g., sum the values in an array, or count the vowels in a string)
  2. By hand, write an assembly function that iterates over an array or string and manipulates some of the items (e.g., iterate over an array of integers and set each negative value to 0)
- AL3 Read assembly code
  1. Describe the behavior of an assembly language function containing branches at a high level
  2. Describe the behavior of an assembly language function containing loops at a a high level
  3. Describe the behavior of an assembly language function that uses the offset parameter for memory operations
- AL4[h] Write, test, and debug assembly code.
  1. Write, test, and debug an assembly function that uses if-else, loops, and memory accesses
  2. Write, test, and debug assembly code that uses the offset parameter for memory accesses
  3. Write assembly code containing multiple functions that correctly uses MIPS conventions for calling functions (a0-a4 for parameters and v0 for return values)
- AL5[h] Write assembly code containing multiple functions (including non-leaf functions) that use MIPS conventions for saving and preserving registers when necessary
- AL6[l] Explain how the use of the stack enables recursion

Single Cycle CPU Design/Implementation

- SS1[h] Machine Language
  1. Assemble machine instructions "by hand"
  2. Disassemble machine instructions "by hand"

- SS2[p] Build a single cycle CPU using a digital logic simulator such as JLS
- SS3 **(Core)** Describe the operation of the single-cycle CPU
    1. Identify the value on any given wire given a specific instruction
    2. Describe the purpose of the value on a specific wire (i.e., how it is used).
    3. Describe the consequences of a particular wire being "broken".
    4. Identify the width of a given wire
- SS4 Make a small change to the design of a single-cycle CPU (e.g., add or modify an instruction)
- SS5 Single-Cycle Design and Patterson and Hennessey's Four Design Principles
    1. Explain the reasoning behind the design decisions in the example Single Cycle CPU (motivation for each instruction type, why each field has the size it does, fixed vs. variable width, absolute vs. relative addresses for branches and jumps, use of function code etc.) In most cases, explanations should reference Patterson and Hennessey's four design principles.
    2. Given a particular CPU design choice (e.g., I-type instructions) list and explain alternate choices as well as the benefits and costs of each.
    3. Explain the benefits of the MIPS offset addressing mode. Describe alternatives to this addressing mode and the pros and cons of each alternative

Cache and Memory

- M1 *(not used F21)* Compare and contrast the construction / operation of DRAM vs. SRAM. Explain why SRAM is faster but more expensive.
- M2 **(Core)** Cache mechanics
    1. Given a cache configuration, determine the number of index, offset, and tag bits
    2. Compute the total size of the cache
    3. Sketch the cache's configuraiton
- M3 **(Core)** Trace the hits and misses in a cache
- M4 **(Core)** Cache parameters
    1. Explain the effects of modifying basic cache parameters (e.g., increasing and/or decreasing a cache's block size)
    2. Identify whether a program would benefit from a large or small block size
    3. Identify whether a program would benefit from a larger associativity

- M5 *(not used F21)* Cache replacement policies
    1. Describe several cache replacement policies and their pros and cons
    2. Explain why LRU is the most common replacement policy
    3. Explain why pseudo-LRU is more common than LRU

Pipeline

- P1 **(Core)** Basic Pipeline concepts
- P2 Data Hazards
- P3 Control Hazards