Department of Electrical and Computer Engineering
University of Wisconsin-Madison

ECE/CS 252
INTRO TO COMPUTER ENGINEERING

Educational Objectives

The numbers in brackets next to each objective refers to its Bloom's Taxonomy level.

Level 1: Knowledge	*example: list terms and recall specific facts*
Level 2: Comprehension	*example: summarize/explain a concept*
Level 3: Application	*example: apply knowledge to solve problems*
Level 4: Analysis	*example: determine whether a program operates correctly or not*
Level 5: Synthesis	*example: write a program to perform a required task*
Level 6: Evaluation	*example: evaluate merits of different approaches to solving a problem*

Last updated 1/18/2018 10:02 AM

Week 01: Computing Systems, Universal Computing Devices, Complexity, Abstraction, Electrical Information

[2] Describe the role of the processor (CPU) in a computing system.
[2] Describe the concept of a universal computational device, and specifically what it means in terms of computation capability.
[2] Describe the following requirements for a good algorithm: definiteness, effective computability, finiteness. It is not necessary to memorize these terms provided the same meaning can be concisely stated (e.g., "steps need to be specific and without other interpretations", "each step must be possible", "the algorithm must be able to finish and not continue forever").
[2] Describe, at a high level, the role of compilers and operating systems in computers and abstraction.
[2] For each of these abstraction layers: [transistor devices/circuits, logic circuits, microarchitecture, instruction set architecture, assembly language, high-level language, program, algorithm, problem]:
- Describe it and identify its location to the other layers.
- Describe how solutions are transformed from it to the layer below.
- Indicate if it relates more strongly to abstracting software or abstracting hardware.

[1] Recall that all values (including numbers, colors, text, etc.) in a computer are eventually converted to sequences of 1s and 0s, which are in turn abstractions of different voltage levels.
[1] Recall that the encoding of a value is not inherent in the value itself, so to interpret the encoded value correctly the encoding must be known and the encoded value interpreted accordingly.

Week 02: Numeric Bases and Representation, Base Conversion, Signed Representation, Binary Arithmetic

Note: references to “any base” or “a given base” refer to bases between 2 and 16, inclusive; “signed format” refers to 2’s-complement and signed magnitude; students are not expected to know 1’s-complement.

[2] Define the following terms: datatype, digit, bit, bitwidth, byte, base, radix, signed, unsigned, integer, sign, magnitude, 2’s-complement, signed-magnitude, sign extension, overflow.
[2] Define “least-significant” and “most-significant” and identify these digit positions within a number.

Unsigned Representation:

[2] Describe the difference between “number” and “value”, and explain how bases are used to interpret numbers to determine their value.
[1] Define binary, octal, decimal, and hexadecimal (hex) in terms of numeric representation.
[2] Describe why direct conversions between binary and octal or between binary and hexadecimal are much simpler than conversions between binary and decimal or between hexadecimal and decimal.
[3] Determine the minimum and maximum value for a digit in any base.
[3] Determine the range of unsigned values representable in a stated base for a stated number of digits.
[3] Determine the number of digits required in a stated base in order to represent a particular unsigned value or a particular quantity of unique values.
[3] Use zero-extension to express an unsigned value using more bits.
[3] Convert an unsigned number in one base to a number in another base. The greatest emphasis will be on bases 2, 8, 10 and 16, but it could be any base.

Signed Binary Representation

[1] Recall that the signed formats can represent both positive values and negative values.
[2] Describe why 2’s-complement is generally used instead of signed magnitude to represent signed numbers in computing.
[2] Describe the process for converting positive and negative numbers into a signed binary format.
[3] Convert a number in one (signed or unsigned) format to a number in a different (signed or unsigned) format, and identify situations in which it this is not possible (e.g., converting a negative value to unsigned format).
[3] Perform negation of a signed binary number.
[3] Determine the range of values representable in a stated signed binary format for a given number of bits.

[3] Determine the number of bits required in a stated signed format to represent a particular signed (positive or negative) value or a particular quantity of unique values.
[3] Use sign-extension to express a signed number's value using more bits.
[2] Use hexadecimal to represent any signed or unsigned binary number.

Binary Arithmetic:

[3] Perform addition of unsigned single- or multi-digit values in any specified base.
[3] Determine, for a particular arithmetic operation, if overflow occurs.
[3] Use 2's-complement representation, negation, and binary arithmetic to perform binary subtraction.

[Assessment A1 covers through week 02 objectives]

Week 03: Fixed- and Floating-Point Representation, ASCII

[2] Define the following terms: fixed-point, radix point, binary point, floating-point, floating-point bias, mantissa/fraction, range, precision, ASCII (not necessary to know what it stands for), character, string.

Fractional Values:

[2] Explain the principles and purpose of representing numbers in integer, fixed-point, and floating-point binary formats (i.e., how numbers are represented in these forms, and what each form can and cannot represent). *Note: students are not expected to memorize any particular floating-point format, but do need to understand the components of the standard IEEE formats (see below).*
[2] List the fields used to represent numbers in standard IEEE floating-point binary formats (sign, exponent, mantissa/fraction), and explain what each field refers to. *Note: students are not expected to memorize the number of bits used for each field or the order in which they appear in a floating point format.*
[2] Explain how the exponent bias is used in floating-point representation.
[2] Explain the tradeoffs between range and precision in terms of the number of bits used to represent the exponent vs. the mantissa/fraction.
[2] Explain why there is an implicit leading 1 in the IEEE floating-point format when the exponent is not 0, and why there is a special case without the leading 1 when the exponent is 0.
[3] Given a floating-point format, including the number of bits used for the exponent and mantissa as well as the bias, convert values between binary and decimal floating-point representation (both directions).
[3] Convert values between binary fixed-point and decimal fixed-point representation (both directions)

ASCII Representation:

[3] Given an ASCII table, convert short strings to null-terminated 8-bit ASCII format using the requested representation of ASCII characters (hexadecimal or decimal).
[3] Given an ASCII table, convert a sequence of decimal or hexadecimal values that represent ASCII characters to a string.

Week 04: Logical Operations, Truth Tables, Logic Gates, Combinational Logic Circuits, Waveforms

[2] Define the following terms: Boolean, bitwise, truth table, logical operation, combinational logic.

Logical Operations:

[2] Contrast a Boolean logic operation with a binary arithmetic operation.
[1] List the three basic logic functions: AND, OR, and NOT.
[2] Identify and write the operator symbol for AND, OR, NOT, and XOR. Also identify and write logic equations representing NAND, NOR, and XNOR.
[3] For each of the NOT, AND, OR, XOR, NAND, NOR, and XNOR logic functions, write out and identify its truth table and perform the operation on two provided Boolean values.
[3] Perform the above functions as bitwise operations on two multi-bit binary values.

[3] For a given Boolean logic equation, complete the truth table (and vice versa). When writing a function for a truth table, it is sufficient to create an AND term for each truth table row where the output is 1 and OR the terms together. *Note: students will not be expected to further simplify these equations.*

Transistors and Logic Gates:

[2] Describe (at a high level) the role of transistors in digital logic circuits, and their relationship to logic gates.

Logic Gates:

[2] Identify, write the Boolean logic expression, and draw the symbol for each of the following gate types: NOT, AND, OR, XOR, NAND, NOR, and XNOR.
[2] Describe the role of the "bubble" when drawing a gate symbol.
[3] Given a gate symbol and input signals and/or values, determine the gate's output.

Combinational Circuits:

[3] For a given Boolean logic function, draw the corresponding logic gate diagram that implements it.
[3] Given a combinational logic circuit and the values of its inputs, determine the value of its output(s).
[3] Identify, draw, and describe the behavior (including determining the output(s) based on provided input value(s)) of each of the following circuits: decoder, multiplexer, full adder. *Note: students are not expected to memorize how these are built (their internal design), but students do need to know the block-level symbols and how they work.*
[3] Correctly connect multiple full adder blocks to form a ripple-carry adder of a stated bit-width.
[3] Connect the inputs of a ripple-carry adder structure to the necessary signals in order to perform a requested addition operation.

Waveforms:

[3] For a given Boolean logic function, plot the output waveform when given the input(s) as waveforms.

Week 05: Sequential Logic, Flip-Flops, and State Machines, Registers, Registers with Enable, Memory

Note: The book refers first to "latches" and later to "flip-flops". In the context of this class, we will only concern ourselves with flip-flops. Students are not required to know the difference between latches and flip-flops for this class (which will be, however, important in follow-on courses). In this class, students should think of registers and memories as being built out of flip-flops. In actuality, memory design is much more complicated and may use latches or even other structures. But to keep it simple, and because the behavior is "close enough", in this class we will consider flip-flops to be the only form of single-bit storage that students are expected to know.

[2] Define the following terms: sequential logic, clock, flip-flop, state, finite state machine, current state, next state, state table, register, memory, address, data, address space, data word size, bitwidth, read, write.

Sequential Logic Concepts:

[2] Describe the key difference between a combinational logic circuit and a sequential logic circuit.
[2] Describe the role of the clock signal in a sequential circuit.
[2] Describe the characteristics of a clock signal.

Flip-Flops:

[2] Describe the behavior of a flip-flop.
[2] Describe (at a high level) the purpose of storage in a sequential circuit.

Finite State Machines:

[2] Identify, on a provided (well-drawn and correctly-annotated) finite state machine diagram: the states, transitions between states, the starting state, and input and output signals.
[3] Determine the minimum number of flip-flops required to implement a state machine with a given number of states.

[3] Given a finite state machine diagram, the current state, and the current values of the inputs, determine the output and the next state.
[3] Given a finite state machine diagram, complete the corresponding state table.
[3] Given a state table or a description of the behavior of a simple finite state machine, draw the corresponding finite state machine diagram.
[3] Identify, on a circuit that implements a finite state machine, the state-holding elements, the next-state logic, and the output logic. Also identify, in particular, which wires correspond to the current state and which to the next state.

Register and Memory Structures:

[3] Complete a waveform describing the behavior or a register in terms of its data input value, its enable signal, and the clock signal.
[2] Describe the behavior of a memory structure in terms of reading and writing information, and the roles of the address and the data in these operations.
[2] Describe the roles of multiplexers, the decoder, and the write enable signal in memory structures.
[3] Calculate any one of the following given the values of the other two: total memory capacity (in bits), the number of memory locations, and the data word size.
[3] Given an address and the contents of memory, determine the value output by the memory during a read operation.
[3] Given the contents of memory before a write operation, determine the contents of memory after a write operation that uses a given address and data input value.

[Assessment A2 covers through week 05 objectives]

Week 06: von Neumann Compute Model, Register File, Instruction Processing

[2] Define the following terms: load, store, I/O, input, output, instruction, opcode, operand, destination register, source register, stored program computer, instruction, program counter (PC), instruction register (IR), control unit, processing unit, field (in an instruction format).

Register File:

[2] Describe the behavior of a register file in terms of reading and writing information, and the roles of the address and the data in these operations.
[2] Describe the roles of multiplexers, the decoder, and the write enable signal in register files.
[3] Calculate any one of the following given the values of the other two: total register file capacity (in bits), the number of registers, and the data word size.
[3] Given a register address and the contents of the register file, determine the value output by the register file during a read operation.
[3] Given the contents of the register file before a write operation, determine the contents of the register file after a write operation that writes a particular data input value to a particular register address.

von Neumann Compute Model:

[2] Name and describe the role of the major components of a von Neumann processor model.
[1] For I/O devices commonly used with computers, identify whether it is input, output, or both.
[2] Describe the role of the control unit in a processor, including that it is a finite state machine, and including the purpose of the program counter (PC) and Instruction Register (IR).
[2] Describe (at a high level) the purpose of each of the phases of the instruction cycle.
[2] Given the names of the phases of the instruction cycle, put them in order.
[1] Recall in which phase of the instruction cycle the PC of the LC-3 is incremented.

[3] Based on the number of different possible operations that a processor supports, calculate the bitwidth of the opcode in its instructions.
[3] Based on the number of registers in the register file, determine the number of bits required to identify the destination register and source register(s) in an instruction.
[3] Given an operation specified in register transfer notation (R3 ← R1 + R2), identify the destination register and any source registers, and describe the operation it represents.

Note: Students will be given a copy of the relevant part(s) of the LC-3 Programmer's Reference in any assessment/exam that tests the remaining objectives. In other words, students are **not** expected to memorize the LC-3 ISA, but are expected to be familiar with the information on the reference and know how to use that information!

Students are also not expected to memorize the LC-3 architectural diagrams presented in the book or video.

Week 07: LC-3 Instruction Set Architecture and Processor, LC-3 Operate Instructions

[2] Define the following terms: data type, immediate operand, constant, register operand, addressing mode, condition code.

LC-3 ISA and Operate instructions:

[2] Describe each of the three categories of instructions: Operate, Data movement, Control flow.
[2] Given an LC-3 instruction mnemonic (the short capital-letter name for the instruction), determine which category of instructions it belongs to.
[3] For a given instruction, determine if it will update the LC-3 processor's condition codes and what the condition code values will be after its execution.
[3] Decode a binary Operate instruction and state what it does in register transfer representation.
[3] Given its register transfer representation, encode an instruction into binary executable by the LC-3.
[3] Based on the size and format (unsigned or 2's-complement) of a constant field in an instruction, determine the range of values representable within that field.
[3] Given an Operate instruction and the current contents of the register file, determine the type of operand(s) it use(s) (register vs. immediate) and the operand value(s).
[3] Given an Operate instruction and the contents of the register file prior to its execution, determine the contents of the register file and the values of the condition codes after its execution.
[5] Write a short program using operate instructions to meet the requirements of a provided specification.

LC-3 Processor:

[2] Label the regions of an LC-3 processor diagram from the **videos** with the corresponding parts of the von Neumann processor model (control unit, processing unit, memory).
[2] Describe at a high level how bits of an instruction can control logic structures in the processor, such as multiplexer selects, immediate or offset values, register or memory addresses, write enables, etc.
[3] Given an LC-3 instruction and a diagram of the LC-3 Processing Unit, determine the inputs/output(s) of the ALU (mode signals, data inputs, output), register file (addresses, data, write enable), multiplexer select signals, etc. *Note: students are **not** expected to memorize the LC-3 hardware architecture, but are expected to understand a provided diagram and use it as described above.*

Week 08: LC-3 Data Movement Instructions

[2] Define the following terms: memory addressing mode, load, store, label, assembler directive
[2] List and describe the following addressing modes: PC-Relative, Base+Offset. Indirect.
[2] List the order and type (i.e., read vs. write) of memory accesses performed by a given instruction.
[3] Determine the source and destination of a given Data Movement instruction.
[3] Given the LC-3 state and the contents of memory, determine the result of executing a given data movement instruction.

[3] Based on a description of the results of a data movement operation (i.e., what happens when it is executed), determine the instruction and its operands (including computing any needed offset values).
[3] Given an assembly language program, determine any required offset values.
[2] Describe the role of labels in Data Movement instructions.
[2] Describe the purpose and use of the .FILL assembler directive.
[3] Use labels and .FILL directives in conjunction with Data Movement instructions in assembly language programs to load pre-determined values into registers and to store results to memory.

[Assessment A3 covers through week 08 objectives]

Week 09: LC-3 Control Flow Instructions and Beginning Programming

Basic Terminology:

[2] Define the following terms: systematic decomposition, stepwise refinement, task decomposition.
[2] Define the following terms in the context of program debugging: tracing, breakpoint, step, syntax error, logic error, data error.
[2] Define the following terms: unconditional branch/jump, conditional branch, branch target.

Control Flow Instruction Basics:

[2] Describe what it means (i.e., what happens) if a branch is "taken" vs. "not taken".
[2] Describe the differences between unconditional and conditional branches, and the role of the condition codes in conditional branches.
[3] Based on the current PC and condition code values, determine if a given conditional branch will be taken, and (in either case), the address of the next instruction that will be executed after the branch.
[3] Based on the contents of the relevant registers, for a given jump instruction, determine the address of the next instruction that will be executed after that jump instruction.

Basic Programming Constructs:

[2] Draw flow charts for and describe the three basic structured programming constructs: sequential, conditional, and iterative.
[3] For a given phrase from a problem statement (e.g., "repeat N times", "if…, else…", etc.), determine which of the three basic programming constructs is most applicable.
[3] Use LC-3 control flow instructions to implement the three basic programming constructs.

Programming:

[3] Given a program represented as a flow chart, determine the result of executing the program.
[5] Draw a flow chart to solve a particular problem based on a provided problem statement.
[3] Given a flow chart, rearrange operations (i.e., move a register decrement so that it is immediately before a test of its value) and/or restate conditions (branch if something is positive instead of branching if it is zero or negative) to make the flow chart more directly implementable as an LC-3 program.
[3] Write a sequence of LC-3 instructions in register transfer notation (e.g., R3 ← mem[R0]) to implement a given flow chart.
[3] Use the provided tools to assemble a program and verify that it assembles correctly.
[3] Use the provided tools to load and run a program.
[3] Determine the effects (i.e., memory contents, register values, condition codes, etc.) when executing an assembly language program.
[5] Write or modify an assembly-language program to meet a given specification (textual or flowchart).
[2] Describe the purpose of debugging and why it is necessary.
[4] Given a description of the intended behavior of a short sequence of instructions, and the actual register contents before and after it was executed, correct the instructions to have the desired behavior.

Week 10: Assembly Language Programming

Assembly Language and Assembly Process:

[2] Define what a "label" is in the context of assembly language programming, and use and understand them in assembly language statements.
[2] Identify comments in an assembly-language program and describe their purpose.
[3] Use comments to increase understandability of code (i.e., not to restate the line of assembly language, but to provide added useful information not obviously conveyed by the instruction).
[2] Compare and contrast assembly language and machine language.
[2] Describe the role of the assembler and linker.
[2] Identify that "loading" means copying the program into memory *(the book's discussion about the capabilities of more sophisticated loaders, such as relocating code, will not be covered in this course).*
[2] List and define the two steps of a two-step assembler.
[2] Describe the role/purpose of a symbol table.
[3] Create a symbol table based on provided assembly code.
[3] Use a provided symbol table (or one you create) when converting assembly code into machine language.
[2] Define the role of assembler directives in an assembly-language program.
[3] Describe, interpret, and use the assembler directives listed on a provided LC-3 Instruction Set sheet.
[2] Compare/contrast the meanings of .FILL, .BLKW, and .STRINGZ.
[3] Translate an assembly language program that includes assembler directives into machine language.

Assembly and Testing:

[3] Use the provided tools to assemble a program and verify that it assembles correctly.
[3] Use the provided tools to load and run a program.
[4] Determine a set of inputs or data values that will reasonably determine whether or not a program operates correctly.
[3] Use a set of inputs or data values to verify that a program operates correctly.

[Assessment A4 covers through week 10 objectives]

Week 11: Subroutines

[2] Define and describe the following terms: subroutine, argument, return value.
[2] Explain how subroutines are different from programs, describe the purpose of using subroutines.
[2] Describe and compare "caller-save" and "callee-save", and explain why "callee-save" is generally preferred when possible.
[2] Explain why we cannot use "callee-save" for register R7 in a subroutine.
[3] Given the code for a subroutine that does not include save/restore of registers, determine which registers must be saved and restored, which must not be saved and restored (any return values), and which do not need to be saved and restored.
[2] Trace through a program that uses subroutines to determine what the processor does.
[3] Use JSR to invoke a subroutine, being sure to save/restore R7 if needed for correctness.
[3] Use RET to return from a subroutine.
[5] Write a subroutine based on a description of its required functionality.
[5] Write a program or subroutine that uses a provided subroutine to accomplish a required task.
[2] Describe the type of information that should be provided in comments at the top of every subroutine.

Week 12: Input and Output

[2] Define and describe the following terms in the context of I/O: input, output, asynchronous, synchronous, handshaking, polling, interrupt, memory-mapping.
[1] Given a common computer I/O device, identify whether it is input or output.

[2] Describe the purpose of control or status registers for I/O devices.
[2] Describe the purpose of data registers for I/O devices.
[2] Compare/contrast memory-mapping with the use of special instructions for accessing I/O devices.
[2] Describe polling and interrupts at a high level, and how each is used to control I/O transfers (including pros/cons of each approach).
[4] Given the addresses of the status and data registers, as well as the characteristics of the status register (e.g., bit 15 is 1 when the device is ready), write code to implement polling for I/O operations for a memory-mapped device using LDI/STI instructions or a combination of LD and LDR/STR instructions.
[2] For polling, describe the pitfalls of not checking the status of an I/O device before accessing its data.

Week 13: Operating Systems, TRAPs

[2] Describe the role of an operating system in the context of programming.
[2] Describe, at a high level, what a TRAP instruction is and why we have TRAP instructions.
[3] Describe, interpret, and use TRAP instructions and TRAP aliases in a program, including saving/restoring R7 where needed for correct program operation.

[Assessment A5 covers through week 13 objectives]

Week 14: Ethics

[2] Define “conflict of interest” as it is used in the IEEE code of ethics.
[2] Compare and contrast the meaning of “legal” and “ethical”.
[2] List several possible consequences of failing to follow ethical standards at work.
[3] Analyze a given scenario, and identify facts, legal issues, and ethical dilemmas. Also identify possible consequences of different decisions on the person making a decision, their loved ones, their employer, the general public, the environment, etc.

Week 15: Wrap-Up and Review

No additional objectives for this week

[Final Exam is Cumulative]