COURSE DESCRIPTION

Dept., Number	**CS 3432 Required**	Course Title	Computer Architecture I: Basic Computer Organization and Design
Semester hours	45 hours+ 21 lab hours	Course Coordinator	Eric Freudenthal

Current Catalog Description

Compile and assembly processes; machine organization; fetch/ decode/execute process; symbolic coding of instructions and data, including instruction types, formats, and addressing modes; implementation of data and control structures, subroutines, and linkage; and input/output handling at the assembly level, including memory-mapped I/O and interrupt and exception handling.

Textbook:

Kerningham, Brian W & Ritchie, Dennis M. "*The C Programming Language*, Second edition," Prentice Hall, ISBN: 0-13-115817-1.

Course Outcomes:

Level 1: Knowledge and Comprehension

Level 1 outcomes are those in which the student has been exposed to the terms and concepts at a basic level and can supply basic definitions. The material has been presented only at a superficial level. Upon successful completion of this course, students will be able to:

1a. Floating Point (floats): Aware that magnitude and mantissa, and sign are stored separately in normalized hidden-one's form. Aware that more bits are used to represent magnitude and mantissa in double-precision. Aware that arithmetic operations may require de-normalization.

1b. Interrupt Management: (interruptManagement): Aware that interrupt vectors may be stored in tables, may be prioritized, and (in some circumstances) can interrupt each other.

Level 2: Application and Analysis

Level 2 outcomes are those in which the student can apply the material in familiar situations, e.g., can work a problem of familiar structure with minor changes in the details. Upon successful completion of this course, students will be able to:

2a. Operation Semantics (operationSemantics): can choose appropriate machine operation, avoids side-effects, appropriate use of src/dest operands, interaction with flags)

2b. Addressing Modes (addressingModes): can choose appropriate addressing mode to access operand (register, fixed address in memory, address specified by register, register+offset, constant)

2c. Instruction Timing (instructionTiming): can determine number of cycles required for instruction execution (assuming sequential execution and constant number of cycles for memory reference); can predict execution time for a loop; can construct loop in order to implement a specified delay

2d. Instruction Set Encoding (instructionEncoding): for architecture examined in course

2e. Compute Relative Offsets (computeRelatveOffset): Resolve relative address offsets using 2-pass approach; aware that other approaches are also used.
2f. Assembly Syntax (assySyntax): can write code using correct syntax
2g. Subroutine Linkage (subroutineLinkage): can implement a call and method that includes parameter passing by value, allocation of auto vars, and return value handling
2h. Variable Size (variableSize): can determine number of bytes required to store a variable, allocate space for it, and can choose appropriate operation width (e.g., word, byte)
2i. Variable Alignment (alignment): can identify alignment requirements when reserving memory
2j. Variable Storage Allocation (variableStorageAllocation): can allocate static and auto variables in memory and in registers
2k. Separate Compilation (separateCompliation): can compose programs in multiple modules that are compiled separately; can choose whether to make symbols global or local. Can choose between and allocate space within data & text segments.
2l. CPU Architecture (cpuArchitecture): familiar and can use PC, SP, register file. Aware that CPU also includes an ALU.
2m. Interrupt Handler (interruptHandler): can program an interrupt handler
2n. Finite State Machines (finiteStateMachines): can construct interrupt-driven FSA implementing a simple protocol
2o. Pointers to Variables (pointersToVars): can translate pointer idioms from high level langauges to assembly code that uses pointers
2p. Array Indexing (arrayIndexing): can relate array abstractions in high level languages to assembly
2q. Branch tables (branchTable): can use a branch table to efficiently achieve switch statement semantics
2r. Composite Structs (compositeStructs): can implement composite data types in high languages and assembly
2s. Memory Mapped I/O (memoryMappedIo): can communicate with simple i/o device mapped into memory
2t. Pulse Width Modulation (pwm): can control duty cycle using PWM
2u. RS232 (rs232): can implement transceiver in software
2v. Memory Devices (memoryType): Understand gross characteristics of RAM, ROM, EPROM, EEPROM, FLASH, and can identify which of these classes of devices are suitable for a particular application.
2w. Counter-Timer (counterTimer): Can determine divisor value for counter-timer that generates periodic (clock) interrupts.

Level 3: Synthesis and Evaluation

Level 3 outcomes are those in which the student can apply the material in new situations. This is the highest level of mastery. Upon successful completion, students will be able to apply the following in new situations:

3a. Unsigned Representations (unsigned): is adept at interpreting and encoding unsigned integers
3b. Twos Complement (signed): is adept at interpreting and encoding 2's complement integers
3c. Radix (radix): is adept at interpreting and encoding integer values in binary, octal, hexadecimal

3d. Powers of Two: (powersOfTwo): is adept at representing powers of two in terms of common pseudo-metric (e.g. Ki, Gi, Ti) units.
3e. Fractional Values in Binary (binaryFraction). Can represent fractional values in binary. E.g. ½ = 0.1 in binary.
3f. Flags (flags): can predict values resulting from addition, subtraction, and shift
3g. Comparison (comparison): can use subtraction order and flags to implement a 2's complement or unsigned comparison
3h. Bit Slice (bitSlice): can use carry/borrow to implement multi-word addition, subtraction, shift
3i. Bitwise Operations (bitwiseOps): can isolate, set, clear, and shift bits
3j. Bitwise Math (bitwiseMath): can do power-of-two modulus, division, multiplication
3k. Linearize Arithmetic Expressions (linearizeExpressions): can convert infix algebraic expressions to a linear sequence of operations, including use of temporary variables
3l. Linearize Control Flow (linearizeCcontrol): can convert block-structured idiom to branching code
3m. Logic (booleanLogic): can convert logical and & or to branching code

Student Outcomes

Student Outcomes: 1, 2, 3, 6, 9, 10

Prerequisites by Topic

CS 2302 and EE 2369 each with a grade of C or better