COURSE SYLLABUS

****

YEAR COURSE OFFERED: 2015

SEMESTER COURSE OFFERED: Fall

DEPARTMENT: Computer Science

COURSE NUMBER: CS3432

NAME OF COURSE: Computer Architecture I

NAME OF INSTRUCTOR: Dr. Eric Freudenthal

****

The information contained in this class syllabus is subject to change without notice. Students are expected to be aware of any additional course policies presented by the instructor during the course.

****

Learning Objectives

Learning Outcomes for CS3432, Computer Organization (Catalog Title: Architecture 1)
(keywords used for data aggregation are parenthesized)
Skill decomposition developed by Eric Freudenthal and Francisco Zapata

1. Knowledge and comprehension
    a. 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. Also see binaryFraction.
    b. Interrupt Management: (interruptManagement): Aware that interrupt vectors may be stored in tables, may be prioritized, and (in some circumstances) can interrupt each other.
    c. Alternative architectures: instruction-level-parallelism, vector operations, caches, predicated instructions, MIMD (including heterogeneous e.g. GPU).
2. Application and analysis
    Upon successful completion of this course, students will be able to apply
    (all topics marked with an asterisk are examined in the context of programs or program fragments in the C programming language)
    a. Program Assembly Language (ProgramAssemblyLanguage)
        i. Operation Semantics (operationSemantics): can choose appropriate machine operation, avoids side-effects, appropriate use of src/dest operands, interaction with flags)*

   ii. Addressing Modes (addressingModes): can choose appropriate addressing mode to access operand (register, fixed address in memory, address specified by register, register+offset, constant) *
   iii. 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
   iv. Instruction Set Encoding (instructionEncoding): for architecture examined in course
   v. Compute Relative Offsets (computeRelatveOffset): Resolve relative address offsets using 2-pass approach; aware that other approaches are also used.
   vi. Assembly Syntax (assySyntax): can write code using correct syntax
   vii. Subroutine Linkage (subroutineLinkage): can implement a call and method that includes parameter passing by value, allocation of auto vars, and return value handling*
   viii. 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)*
   ix. Variable Alignment (alignment): can identify alignment requirements when reserving memory*
   x. Variable Storage Allocation (variableStorageAllocation): can allocate static and auto variables in memory and in registers*
   xi. 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.*
   xii. CPU Architecture (cpuArchitecture): familiar and can use PC, SP, register file. Aware that CPU also includes an ALU.

b. Asychrony (Asynchrony)
   i. Interrupt Handler (interruptHandler): can program an interrupt handler
   ii. Finite State Machines (finiteStateMachines): can construct interrupt-driven FSA implementing a simple protocol*

c. Indirection (Indirection): operands at computed effective address
   i. Pointers to Variables (pointersToVars): can translate pointer idioms from high level langauges to assy code that uses pointers*
   ii. Array Indexing (arrayIndexing): can relate array abstractions in high level langauges to and assy *
   iii. Branch tables (branchTable): can use a branch table to efficiently achieve switch statement semanticsC*
   iv. Composite Structs (compositeStructs): can implement composite data types in high languages to assy *

d. i/o and devices (Devices)

    i. Memory Mapped I/O (memoryMappedIo): can communicate with simple i/o device mapped into memory*
    ii. Pulse Width Modulation (pwm): can control duty cycle using PWM
    iii. RS232 (rs232): can implement transceiver in software
    iv. 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.
    v. Counter-Timer (counterTimer): Can determine divisor value for counter-timer that generates periodic (clock) interrupts.
    vi. Other devices likely to be added in 2012 in collaboration with ECE.

  e. General Skills (where should these go?)
    i. Clerical Accuracy (clericalAccuracy): clerical errors do not overly interfere with productivity
    ii. Written Communication (writingSkills): can communicate simple technical concepts clearly
    iii. Arithmetic and Algebra (arithmetic): misunderstandings of basic algebra do not interfere with productivity (for example, computing 2**3 * 2**5)

3. Synthesis and Evaluation
   Upon successful completion, students will be able to apply the following in new situations
  a. Integer Representations (IntegerRepresentations)
    i. Unsigned Representations (unsigned): is adept at interpreting and encoding unsigned integers
    ii. Twos Complement (signed): is adept at interpreting and encoding 2's complement integers
    iii. Radix (radix): is adept at interpreting and encoding integer values in binary, octal, hexadecimal
    iv. Powers of Two: (powersOfTwo): is adept at representing powers of two in terms of common pseudo-metric (e.g. Ki, Gi, Ti) units.
    v. Fractional Values in Binary (binaryFraction). Can represent fractional values in binary. E.g. ½ = 0.1 in binary.
  b. Integer Arithmetic (IntegerArithmetic)
    i. Flags (flags): can predict values resulting from addition, subtraction, and shift
    ii. Comparson (comparison): can use subtraction order and flags to implement a 2's complement or unsigned comparison
    iii. Bit Slice (bitSlice): can use carry/borrow to implement multi-word addition, subtraction, shift
    iv. Bitwise Operations (bitwiseOps): can isolate, set, clear, and shift bits
    v. Bitwise Math (bitwiseMath): can do power-of-two modulus, division, multiplication
    vi. Linearize Arithmetic Expressions (linearizeExpressions): can convert infix algebraic expressions to a linear sequence of operations, including use of temporary variables
  c. Control Flow (ControlFlow)

i. Linearize Control Flow (linearizeCcontrol): can convert block-structured idiom to branching code
ii. Logic (booleanLogic): can convert logical and & or to branching code

Course Grading

- Grading
  - Course grade is an aggregation of:
    - Approximately 4 partial-term grades
    - Final exam grade
  - Partial term grades
    - Are computed approximately once every 7 lectures
    - The minimum of grades collected during partial terms
      - Graded instruments
      - Graded assignments
      - Lab assignments
- Graded instruments
  - Tests/quizzes
    - Final exam: date and time are specified by the university.
    - Frequent Quizzes (in lieu of tests)
      - Generally unannounced, at least one every two weeks, generally focus on recently studied topics, but may contain topics studied earlier
      - Short: Generally 10-20 minutes
      - May not be "made up"
    - Graded homeworks (assigned intermittently)
  - Multiple skills may be measured by the same problem within an instrument
  - To the extent that it is practical, **useful competency (generally a binary value)** for distinct skills, called a proficiency, will be assessed independently
    - Thus, if an instrument with four questions measures ten skills, ten measurements will be computed.
    - A single problem may provide opportunity to demonstrate all or most of the proficiencies being measured.
    - A proficiency not demonstrated by any answer provided by a student due to not providing complete and correct answers to all problems will be assessed as not meeting the threshold of **useful competency.**
    - It is possible that all skills measured by a particular instrument will be demonstrated in multiple problems.
    - **In-class Instruments are designed to require substantially less time than allotted**. Therefore not completing a quest within the allotted time may be an indication of weak understanding worthy of discussion with the instructor.
  - Notes on test-taking strategy

        - If short of time - it **is not generally advantageous** to partially answer multiple questions in a manner that repeatedly demonstrates the same skills..   .
    - The grade for an instrument will correspond to the **fraction f of skills in which useful competency is demonstrated.** Generally
        - **100% corresponds to an A+ (4.3 on a 4-pt scale)**
        - **50% corresponds to an F (zero on a 4-pt scale)**
        - Conversion back to 4-pt scale: *Grade = (f-0.5) * 8.6* where *f* is the fraction described above
    - As in life, all instruments are cumulative.
        - If test (midterm/quiz/final) T1 occurs before test T2, a skill measured in T1 may also be assessed in T2.
- Graded assignments (including labs)
  All labs involve low-level programming in the C programming language and must be developed using the "official" linux command-line tools (including gcc, make, svn, and emacs) using the "arch1 virtual machine."
    - Intention:
        - Assignments and labs provide an opportunity for students to practice and explore concepts presented in class.
        - Students are expected to act professionally
            - By helping each other select and design problem-solving approaches
            - By reading whatever resources they find relevant
            - By attributing credit to any person or reference materials that substantively contributed to their solutions
            - By only submitting solutions they fully understand.
            - Professionalism includes honesty, clarity, and accuracy.
    - Submission and due dates
        - Submission is via SVN commit. Solutions will not be accepted via email.
        - Due dates are posted on course web site
        - Graded twice (averaged)
            - As submitted at due date
            - As updated one week after grade is distributed, **only upon email request from the student.**
    - Rules
        - Students must only submit solutions that fairly reflect their own understandings.
        - Solutions must clearly and fairly attribute credit people and resources that contributed to their design or preparation.
        - Descriptive text included with solutions must be composed by the student submitting it.
    - Implication
        - It is academic dishonesty for a student to submit a solution they cannot replicate individually or to not fairly credit their sources.

Grade Computation

- Lab and exam grades are computed (mostly) separately.
  - Lab grades are averaged. They also provide evidence of skill proficiency.
  - Skill proficiency is computed as a fraction of measured skills.
- Overall course grades are the minimum of lab and skill grades.

Required Reading

- Absolutely required: Kerningham, Brian W & Ritchie, Dennis M. "The C Programming Language, Second edition," Prentice Hall, ISBN: 0-13-115817-1.
- The course web site contains an online text on assembly language programming for the MSP-430.

Recommended Reading

- MSPGCC cross-tools manual (55 pages). Can be downloaded from the course web site..
- Recommended by students (and by no way required): Android app "Programmer Mental Math" by Joel Jurix.

List of discussion/lecture topics

Week	Topic
0	Introduction; Concepts from prior courses
1	Introduction to Data Movement and Arithmetic Instructions
1	Intro to Variables in 'C' and Assembly Language,Shifts, Flags, And Masks In 'C'
2,3	Computer math:Two's Complement, Quotients and Remainders, Translating complex expressions into assembly language using expression trees
3	compiling programs and make, Pseudo-ops and Assembler Directives
3,4,5	Arithmetic Flags,Arithmetic Comparison And Control Flow, Relational operators and subtraction orderJump instructions
6-10	Registers And Register Addressing Mode, Constants and immediate addressing, Constant Generator Registers, pointers and Indexed Addressing
11	Stacks, Subroutines And Subroutine Linkage
10	Finally: ▪ A full survey of the MSP430's Addressing Modes ▪ The entire MSP430 Instruction Set

	▪ Includes printable 1-page summary of all instructions & addressing modes.
11	Multiplying and dividing by two, Type conversion
10-11	Examination of TI documents (links on MSP-430 resource page). Attention paid to ▪ i/o ports ▪ setting cpu clock frequency ▪ instruction timing ▪ instruction "emulation" (e.g. `ret` is implemented using `pop`)
11	RS-232 Serial Communication
12	Interrupts
12	State Machines
13	Switch statements implemented using Jump Tables
14	Storage Devices, Floating Point Numbers, Alternative architectures

STANDARDS OF CONDUCT: Students are expected to conduct themselves in a professional and courteous manner, as prescribed by the Standards of Conduct. Students may discuss programming exercises in a general way with other students, but the solutions must be done independently. Similarly, groups may discuss project assignments with other groups, but the solutions must be done by the group itself. Graded work should be unmistakably your own. You may not transcribe or copy a solution taken from another person, book, or other source, e.g., a web page). Professors are required to - and will - report academic dishonesty and any other violation of the Standards of Conduct to the Dean of Students.

DISABILITIES: If you feel you may have a disability that requires accommodation, contact the Disabled Student Services Office at 747-5148, go to Room 106 E. Union, or e-mail to dss@utep.edu.