## I. GENERAL COURSE INFORMATION

**Subject and Number:** Computer Science 16
**Descriptive Title:** Assembly Language Programming for x86 (IBM PC) Processors
**Course Disciplines:** Computer Science
**Division:** Mathematical Sciences

**Catalog Description:**
This course includes detailed coverage of assembly language programming for x86 processors. Topics include hexadecimal arithmetic, two's complement arithmetic, memory organization, addressing modes, procedure calls, the stack frame, macros, calling assembly language procedures from C or C++, recursion, BIOS and DOS interrupts, the floating point unit and instructions, and the debugger.

**Conditions of Enrollment:**
**Prerequisite:** Computer Science 1 or Computer Science 3 or Computer Information Systems 80 with a minimum grade of C in prerequisite or equivalent AND Mathematics 180 with a minimum grade of C or concurrent enrollment

| | | |
|---|---|---|
| **Course Length:** | **X Full Term** | **Other (Specify number of weeks):** |
| **Hours Lecture:** | **3.00 hours per week** | **TBA** |
| **Hours Laboratory:** | **3.00 hours per week** | **TBA** |
| **Course Units:** | **4.00** | |

**Grading Method: Letter**
**Credit Status: Associate Degree Credit**

| | |
|---|---|
| **Transfer CSU:** | **X Effective Date: 5/14/2013** |
| **Transfer UC:** | **X Effective Date: Fall 2013** |

**General Education:**

**El Camino College:**

**CSU GE:**

**IGETC:**

## II. OUTCOMES AND OBJECTIVES

**A. COURSE STUDENT LEARNING OUTCOMES (The course student learning outcomes are listed below, along with a representative assessment method for each. Student learning outcomes are not subject to review, revision or approval by the College Curriculum Committee)**

1. Students will design, code, compile, test and document programming solutions to problems by developing PC assembly language code that makes direct use of processor instructions, interrupts, registers, the stack, as well as existing macro and procedure libraries.

2. Students, when given a code segment will be able to trace the execution, providing the real-time content of registers during operations, the dynamic content of the stack during procedure calls and returns, and tracing the conditional execution of code generally, and within looping structures specifically.

3. Students, when given PC assembly language code with errors, will be able to identify what those errors are and will be able to modify the PC assembly language code to eliminate those errors.

4. Students will be able to explain the concepts of PC assembly language registers, interrupts, data segment organization, addressing modes, internal data representation, decision structures, macros and procedures.

The above SLOs were the most recent available SLOs at the time of course review. For the most current SLO statements, visit the El Camino College SLO webpage at http://www.elcamino.edu/academics/slo/.

**B. Course Student Learning Objectives (The major learning objective for students enrolled in this course are listed below, along with a representative assessment method for each)**

1. Perform two's complement arithmetic.
   - Quizzes
2. Write programs that correctly use the addressing modes.
   - Other (specify)
   - Programming assignment to be done in the lab and outside of class.
3. Use the following classes of processor instructions: a. Signed and unsigned arithmetic b. Data transfer c. Comparison d. Conditional transfer e. Unconditional transfer f. Flag testing g. Loop h. Stack operations i. String j. Type conversion k. Bit operations
   - Other (specify)
   - Exams, Quizzes, and programming assignments to be done in the lab and outside of class.
4. Write program code containing procedures callable within the same file.
   - Other (specify)
   - Exams, Quizzes, and programming assignments to be done in the lab and outside of class.
5. Write procedures external to the main file and that are callable by the main procedure.
   - Other (specify)
   - Exams, Quizzes, and programming assignments to be done in the lab and outside of class.
6. Write procedures that are callable from either C or C++ programs.
   - Other (specify)
   - Exams, Quizzes, and programming assignments to be done in the lab and outside of class.
7. Use the stack, especially during recursive procedure calls.
   - Other (specify)
   - Exams, Quizzes, and programming assignments to be done in the lab and outside of class.

8. Write well-organized Macros.
   - Other (specify)
   - Exams, Quizzes, and programming assignments to be done in the lab and outside of class.
9. Invoke BIOS and DOS operating-system interrupts.
   - Other (specify)
   - Exams and quizzes.
10. Use the Microsoft Debugger.
   - Other (specify)
   - Programming assignments to be done in the lab and outside of class.
11. Write at least eight assembly language programs.
   - Other (specify)
   - Programming assignments to be done in the lab and outside of class.
12. Explain the basic organization of the von Neumann machine.
   - Other (specify)
   - Exams and quizzes.
13. Explain the function of the control unit and the instruction fetch, decode, and execution process.
   - Other (specify)
   - Exams and quizzes.

**III. OUTLINE OF SUBJECT MATTER (Topics are detailed enough to enable a qualified instructor to determine the major areas that should be covered as well as ensure consistency from instructor to instructor and semester to semester.)**

| Lecture or Lab | Approximate Hours | Topic Number | Major Topic |
|---|---|---|---|
| Lecture | 3 | I | Introduction to x86 Assembly Language<br>A. Hexadecimal Arithmetic<br>B. Two's complement form to represent negative numbers<br>C. Overview of x86 Assembly Language program structure<br>D. Basic organization of the von Neumann machine<br>E. Integer arithmetic instructions overview<br>F. Data movement overview<br>G. Introduction to the use of input/output macros |
| Lecture | 6 | II | Control and Looping Instructions<br>A. IF structures and the GOTO statement<br>B. Loops - for loops, while loops, and do-while loops |
| Lecture | 7.5 | III | Arrays, Pointers, Addressing Modes<br>A. Modeling of arrays of objects<br>B. Use of pointers and their relationship to arrays and possible dynamic memory allocation<br>C. Accessing computer memory addresses |
| Lecture | 7.5 | IV | The Stack and Procedure Calls<br>A. What the Stack is and how it is structured<br>B. How procedure calls use the stack to pass data to procedures and return results from procedures |

| | | | |
|---|---|---|---|
| Lecture | 6 | V | Bit Instructions, String Instructions, Tables, System Calls<br>A. Bit shift, set, and clear instructions<br>B. String manipulation and analysis instructions<br>C. Table set-up and data use instructions<br>D. System calls syntax and use |
| Lecture | 4 | VI | Records, Arrays of Records, Sorting<br>A. Modeling the association of multiple objects within a structure known as a record<br>B. Syntax and memory considerations to master when working with arrays of records<br>C. Syntax and instructions related to the comparing of objects in order to sort the objects in a particular manner |
| Lecture | 4 | VII | Working With Files<br>A. Opening a file for input operations<br>B. Extracting objects from an input file<br>C. Opening a file for output operations<br>D. Storing data into an output file<br>E. Specific objects in a given file |
| Lecture | 6 | VIII | Calling x86 Assembly Language Procedures from C or C++<br>A. Connecting to a procedure (function) in C or C++<br>B. Extract the parameter data of the procedure (function) in C or C++<br>C. Return possibly modified objects to the invoking code of the procedure (function) in C or C++ |
| Lecture | 6 | IX | Recursion and the Stack<br>A. What a recursive function is and some examples<br>B. How recursion relates to the x86 Assembly Language stack<br>C. Stack overflow related to recursive functions<br>D. Lesson that a recursive function must be coded in such a way that the recursion must stop after a finite number of iterations |
| Lecture | 4 | X | Floating-Point Unit and Corresponding Instruction Set<br>A. Real numbers, not just integers<br>B. Understanding the much greater algorithmic complexity converting between decimal numbers and their internal binary representation as opposed to integers and their internal binary representation |
| Lab | 3 | XI | Introduction to x86 Assembly Language<br>A. Hexadecimal Arithmetic<br>B. Two's complement form to represent negative numbers<br>C. An overview of x86 Assembly Language program structure<br>D. Basic organization of the von Neumann machine<br>E. Integer arithmetic instructions overview<br>F. Data movement overview<br>G. Introduction to the use of input/output macros |
| Lab | 6 | XII | Control and Looping Instructions |

| | | | |
|---|---|---|---|
| | | | A. IF structures and the GOTO statement<br>B. Loops - for loops, while loops, and do-while loops |
| Lab | 7.5 | XIII | Arrays, Pointers, Addressing Modes<br>A. Modeling of arrays of objects<br>B. Use of pointers and their relationship to arrays and possible dynamic memory allocation<br>C. Different ways of accessing computer memory addresses |
| Lab | 7.5 | XIV | The Stack and Procedure Calls<br>A. What the Stack is and how it is structured<br>B. How procedure calls use the stack to pass data to procedures and return results from procedures |
| Lab | 6 | XV | Bit Instructions, String Instructions, Tables, System Calls<br>A. Bit shift, set, and clear instructions<br>B. String manipulation and analysis instructions<br>C. Table set-up and data use instructions<br>D. System calls syntax and use |
| Lab | 4 | XVI | Records, Arrays of Records, Sorting<br>A. Modeling the association of multiple objects within a structure known as a record<br>B. Memory considerations to master when working with arrays of records<br>C. Comparing of objects in order to sort the objects in a particular manner |
| Lab | 4 | XVII | Working With Files<br>A. Opening a file for input operations<br>B. Extracting objects from an input file<br>C. Opening a file for output operations<br>D. Storing data into an output file<br>E. Specific objects in a given file |
| Lab | 6 | XVIII | Calling x86 Assembly Language Procedures from C or C++<br>A. Connecting to a procedure (function) in C or C++<br>B. Extracting the parameter data of the procedure (function) in C or C++<br>C. Return possibly modified objects to the invoking code of the procedure (function) in C or C++ |
| Lab | 6 | XIX | Recursion and the Stack<br>A. What a recursive function is and some examples<br>B. How recursion relates to the x86 Assembly Language stack<br>C. Stack overflow related to recursive functions<br>D. Lesson that a recursive function must be coded in such a way that the recursion must stop after a finite number of iterations |
| Lab | 4 | XX | Floating-Point Unit and Corresponding Instruction Set<br>A. Real numbers, not just integers |

| | | | |
|---|---|---|---|
| | | | B. Understanding the much greater algorithmic complexity converting between decimal numbers and their internal binary representation as opposed to integers and their internal binary representation |
| Total Lecture Hours | | 54 | |
| Total Laboratory Hours | | 54 | |
| Total Hours | | 108 | |

**IV. PRIMARY METHOD OF EVALUATION AND SAMPLE ASSIGNMENTS**

A. **PRIMARY METHOD OF EVALUATION:**
Problem solving demonstrations (computational or non-computational)

B. **TYPICAL ASSIGNMENT USING PRIMARY METHOD OF EVALUATION:**
Write a set of procedures to act on the bits of a word stored in ax-register, with cl holding the position of the bit to be processed. cl holds 0..15. The procedures are 1. SetBit: set the desired bit to 1. 2. ClearBit: set the desired bit to 0. 3. TestBit: set CF if desired bit is 1 else clears CF. 4. CompBit: Complement the bit: ie if bit is 1 set it to 0 if bit is 0 set it to 1. In addition, design an adequate testing "main" procedure to enable a user program to test the procedures to determine that they are acting properly.

**C. COLLEGE-LEVEL CRITICAL THINKING ASSIGNMENTS:**

1. Develop an assembly program which will receive, from the keyboard, integers represented in character form. Conversion from character form to binary form is to take place with the integers stored in an array.

2. Write assembly code to allow the user to define an "active" on the screen within which all write actions will be confined. Line-wrapping is to be enabled.

**D. OTHER TYPICAL ASSESSMENT AND EVALUATION METHODS:**

Other exams
Quizzes
Written homework
Laboratory reports
Homework Problems
Other (specify):
Write Computer Programs

**V. INSTRUCTIONAL METHODS**

Laboratory
Lecture

**Note: In compliance with Board Policies 1600 and 3410, Title 5 California Code of Regulations, the Rehabilitation Act of 1973, and Sections 504 and 508 of the Americans with Disabilities Act, instruction delivery shall provide access, full inclusion, and effective communication for students with disabilities.**

## VI. WORK OUTSIDE OF CLASS

Study
Required reading
Problem solving activities
Other (specify)
Development and testing of assembly programs

**Estimated Independent Study Hours per Week:** 6

## VII. TEXTS AND MATERIALS

### A. UP-TO-DATE REPRESENTATIVE TEXTBOOKS

Kip R. Irvine. Assembly Language for x86 Processors. 8th ed. Prentice Hall, 2019.

### B. ALTERNATIVE TEXTBOOKS

### C. REQUIRED SUPPLEMENTARY READINGS

### D. OTHER REQUIRED MATERIALS

## VIII. CONDITIONS OF ENROLLMENT

### A. Requisites (Course and Non-Course Prerequisites and Corequisites)

| Requisites | Category and Justification |
|---|---|
| Course Prerequisite Computer Science-1 or | Other Knowledge and Skills |
| Course Prerequisite Computer Science-3 or | Other Knowledge and Skills |
| Course Prerequisite Computer Information Systems-80 or | Other Knowledge and Skills |
| Non-Course Prerequisite AND | Students must have completed an introductory course in a high level programming language. Computer Science 1 is an introductory course in the C++ language. Computer Science 3 in an introductory course in the JAVA programming language. The student will be exempted from the prerequisite if she/he can demonstrate sufficient programming knowledge in a high level programming language (such as C/C++, C#, Visual Basic, Java, PHP, ... ) through work portfolio or oral and/or written examination by the department of computer science faculty. |
| Course Prerequisite Mathematics-180 | Computational/Communication Skills |

**B. Requisite Skills**

| Requisite Skills |
|---|
| Manipulate algebraic expressions at the Pre-Calculus level.<br>MATH 180 - Analyze functions (including polynomial, algebraic, rational, exponential, logarithmic, trigonometric) for critical features, including: intercepts, asymptotes, domain, range, and average rate of change. |
| Solve application problems at the Pre-Calculus level.<br>MATH 180 - Solve application problems using the topics of the course. |
| Evaluate and perform operations on functions at the Pre-Calculus level.<br>MATH 180 - Analyze functions (including polynomial, algebraic, rational, exponential, logarithmic, trigonometric) for critical features, including: intercepts, asymptotes, domain, range, and average rate of change.<br>MATH 180 - Determine the inverse of a function (polynomial, algebraic, rational, exponential, logarithmic, trigonometric) and analyze it in terms of critical features.<br>MATH 180 - Solve equations involving polynomial, rational, exponential, logarithmic, trigonometric functions. |
| Write computer code expressing mathematical expressions.<br>CSCI 1 - Design solutions requiring translation of mathematical and algebraic steps into a C++ program, using appropriate mathematical operators and math library functions of the C++ implementation in use. |
| Write computer code implementing functions or procedures.<br>CSCI 1 - Utilize problem analysis and design techniques in developing solutions to programming problems. In particular, break programming problems down into chunks, leading to efficient use of top-down design in order to create and implement modular solutions. |
| Write computer code implementing arrays.<br>CSCI 1 - Design programming solutions requiring the storage and manipulation of large amounts of data (with random access ability during execution cycle), using single and multi-dimensional arrays, such as numerical, string, char, and structure types. |

**C. Recommended Preparations (Course and Non-Course)**

| Recommended Preparation | Category and Justification |
|---|---|

**D. Recommended Skills**

| Recommended Skills |
|---|

**E. Enrollment Limitations**

| Enrollment Limitations and Category | Enrollment Limitations Impact |
|---|---|

**Course created by Joseph E. Hyman on 03/01/1988.**

**BOARD APPROVAL DATE: 05/23/1988**

**LAST BOARD APPROVAL DATE: 12/16/2019**

20238