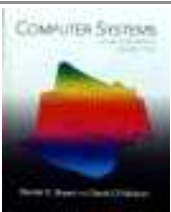


CS 301 - Assembly Language

Meets MWF 3:30-4:30 PM (New!) Room 106 Chapman Building University of Alaska Fairbanks	CS F301-F01 (#73868) 3.0 Credits, Fall 2005 Prerequisite: CS 201 (Programming)	Instructor: Dr. O. Lawlor ffosl@uaf.edu, 474-7678 Office: 210C Chapman Hours: 1-2 MWF or by appointment	
	Required Textbook: Computer Systems, Bryant and O'Hallaron, Prentice Hall 2003	ADA Compliance: Will work with Office of Disabilities Services (203 WHIT, 474-7043) to provide reasonable accommodation to students with disabilities.	Course Website (& links to Blackboard): http://www.cs.uaf.edu/2005/fall/cs301/ UNIX Machines: on nanook.uaf.edu, in Chapman lab, or Linux CDs available

Course Goals and Requirements

By the end of the course, you will understand how your code actually executes on a real machine: from electrons on a semiconductor, to registers and binary arithmetic, to machine code and assembly, to C code. This course will focus on the middle levels of this chain of abstractions--you'll eventually learn much more about the lower levels (electrons, semiconductors, logic circuits) in EE 341 & 443, and about the higher levels (compilers and languages) in CS 331. To understand this course, you will have to be familiar with all the basics of C or C++: variables, loops, arrays, pointers, structures, and subroutines.

Calendar

First day of class: 3:30pm Friday, September 2. Last day to drop: Friday, September 16. Midterm: 3:30pm Friday, October 21.	Last day to withdraw: Friday, October 28. Thanksgiving break (no class): Friday, November 25. Last day of class: Monday, December 12. Final Exam: 3:15-5:15 PM Wednesday, December 14.
---	---

Student Resources

[Google](#), [Rasmuson Library](#), [Academic Advising Center](#) (509 Gruening, 474-6396), Math Lab (Chapman Room 305), [English Writing Center](#) (801 Gruening Bldg, 478-5246).

Grading

Your work will be evaluated on correctness, rationale, and insight, not on successful regurgitation of random trivia. Grades for each assignment and test may be curved upward by scaling. Each homework and the midterm will then be clamped to the range [0%,100%]. Your grade is then computed based on four categories of work:

1. **HW:** Homeworks and machine problems, to be distributed through the semester.
2. **MT:** Midterm Exam, to be held 3:30pm Friday, October 21.
3. **FINAL:** Final Exam (comprehensive), to be held 3:15-5:15 PM Wednesday, December 14.

The final score is then calculated as:

$$\text{TOTAL} = 40\% \text{ HW} + 30\% \text{ MT} + 30\% \text{ FINAL}$$

Homeworks are due by midnight at the end of the day they are due.

THE TEN COMMANDMENTS (OF CS 301)

1. THOU SHALT ASK QUESTIONS IN CLASS WHEN THY **PROFESSOR** STOPS MAKING SENSE.
2. THOU SHALT LEARN THE GENERAL PRINCIPLES, BY LEARNING THE CURRENT SPECIFIC IMPLEMENTATIONS.
3. THOU SHALT COME TO CLASS. EVEN WHEN SLEEPY. BUT THOU SHALT NOT SLEEP IN CLASS.
4. REMEMBER THY BOOK, AND KEEP IT HANDY.
5. THOU SHALT TURN IN THY ASSIGNMENTS BEFORE MIDNIGHT ON THE REQUIRED DAY. THOU SHALT RECEIVE A ZERO FOR LATE ASSIGNMENTS.
6. THOU SHALT NOT START WORK ON THY ASSIGNMENTS 20 MINUTES BEFORE THEY ARE DUE.
7. THOU SHALT CITE ALL THY SOURCES. EVEN THOSE FROM THE INTERNET.
8. THOU SHALT NOT COPY THY NEIGHBOR'S ASSIGNMENTS, NOR HIS TESTS.
9. ALL THY ASSIGNMENTS AND TESTS SHALL BE THY OWN WORK. ANY [CHEATING OR PLAGIARISM](#) SHALL INCUR THE WRATH OF THY **PROFESSOR**.
10. THOU SHALT REGULARLY CHECK BLACKBOARD. I MAY POST ASSIGNMENTS THERE AT ANY TIME, FOR I AM THY **PROFESSOR**.

At my discretion, I may allow late assignments without penalty when due to circumstances beyond your control. Major assignments that are slightly late may be accepted at a 50% grade penalty (e.g., on-time grade: 80%; late grade: 40%). Even substantial reuse of other people's work is fine (and not plagiarism) if it is clearly cited; you'll be graded on what you've added to others' work. Group work on substantial assignments (not homeworks, not tests) is acceptable if you clearly label who did what work; but I do expect a two-person group project to represent twice as much work as a one-person project. Department policy does not allow tests to be taken early; but in extraordinary circumstances may be taken late. All classes and exams will be in Chapman 104. Please do take the time to do quality work--your checkbook and/or happy spouse will eventually thank you!

Course Outline (Tentative)

<p>Data representation (Chapter 2.1)</p> <ul style="list-style-type: none"> • Memory, files as big arrays of bytes • Integer representation as bits, bytes <ul style="list-style-type: none"> ◦ Big endian ◦ Little endian • Binary, decimal, hex, octal, and base conversion <p>Operations</p> <ul style="list-style-type: none"> • Bitwise operations (Chapter 2.1) <ul style="list-style-type: none"> ◦ AND, OR, XOR ◦ "SIMD Within A Register" (SWAR): Cohen-Sutherland clipping ◦ Left & right shifting; finite integer range. ◦ Extract integer into bits, reassemble from bits. • Arithmetic operations (Chapter 2.2 & 2.3) <ul style="list-style-type: none"> ◦ Addition: unsigned. Overflow. Wraparound. Range. ◦ Subtraction: two's complement addition; signed numbers ◦ Multiplication & acceleration via bit shifts ◦ Division & acceleration via bit shifts ◦ Modulus, implementation, acceleration via bit masks ◦ Relative speed of each operation on various machines ◦ Multiple-precision implementations of numerical operations <p>Instruction encoding (Chapter 3.1-4)</p> <ul style="list-style-type: none"> • Tiny example encoding: use of all above features in a tiny emulator <ul style="list-style-type: none"> ◦ Concept of registers: stash stuff here <ul style="list-style-type: none"> ▪ Register hardware implementation ▪ Register uses: program counter, address, data, etc. ◦ Concept of memory: big bunch o' bytes ◦ Opcodes: do this now • Hardware implementation of above encoding (preview of EE 341) • Real examples <ul style="list-style-type: none"> ◦ PPC (clean 4-byte register-based RISC) ◦ Java (clean 1-byte stack-based unboxed) ◦ CIL (1 or 2-byte stack-based boxed) ◦ x86 (hideous variable-length CISC) <p>Assembly & disassembly (Chapter 3.15)</p> <ul style="list-style-type: none"> • opcode mnemonics, naming a register, immediate values • Inline (<code>__asm</code>) assembly syntaxes; standalone (.S) assembly syntaxes <ul style="list-style-type: none"> ◦ Operand order dyslexia ◦ Labels, macros, etc. • Win32, gcc x86 inline assembly • AT&T .S files 	<p>Memory</p> <ul style="list-style-type: none"> • Structures (Chapter 3.9) <ul style="list-style-type: none"> ◦ In-memory layout ◦ Alignment & padding ◦ sizeof, offsetof • Array indexing (Chapter 3.8) <ul style="list-style-type: none"> ◦ 1D, 2D, 3D, nD ◦ For structs • Global variables <p>Subroutines (Chapter 3.7)</p> <ul style="list-style-type: none"> • Stack allocation: push & pop • Program Counter push & pop: call & return • Parameter passing, pass by reference • Calling conventions • Subroutine linkage and naming <p>Heap memory</p> <ul style="list-style-type: none"> • Allocation & free (Chapter 10.9) • Garbage collection (Chapter 10.10) <p>Performance and Optimization (Chapters 4, 5, and 9)</p> <ul style="list-style-type: none"> • General optimization checklist • Timing and profiling • Algorithmic Optimization • Invariant hoisting, constant propagation • Memory Performance <ul style="list-style-type: none"> ◦ Caching ◦ Levels & performance of cache ◦ Program transformations to improve memory performance • Concurrency <ul style="list-style-type: none"> ◦ Hardware and Software Pipelining ◦ Cost of branches <p>Advanced control flow</p> <ul style="list-style-type: none"> • Function pointers, implementation • C++ virtual method <code>_vtable</code> implementation • Dynamic linking (Chapter 7) <p>Floating point (Chapter 2.4, 3.14, and beyond)</p> <ul style="list-style-type: none"> • Instructions • IEEE floating-point representation <ul style="list-style-type: none"> ◦ Sign, exponent, mantissa ◦ normalization ◦ Fun bitwise hacks (fast absolute value, log-base-2, float-to-int, etc.) ◦ denormalized numbers, NaNs, and performance penalty • Operations <ul style="list-style-type: none"> ◦ Addition • Interfaces <ul style="list-style-type: none"> ◦ PPC sensibility ◦ x86 stack horror • 4-vector of floating point numbers <ul style="list-style-type: none"> ◦ x86 SSE & <code><mmintrin.h></code> intrinsics ◦ PPC <code>Altivec</code> ◦ Graphics card <code>ARB_fragment_program</code>
--	---