Test 2 will be Thursday, March 23 in class. It will be closed book and notes, except for one 8.5" x 11" sheet of paper (front and back) with notes, and your hand-out of MIPS assembly language instructions (available at: http://www.cs.uni.edu/~fienup/cs041s06/lectures/lec12.pdf).

Chapter 3.
Signed number representation: sign bit and magnitude, one's complement, two's complement
Addition and subtraction of signed and unsigned numbers
Overflow in integer arithmetic
Floating-point Representation: 32-bit and 64-bit IEEE 754 standard and special values
Addition and multiplication of floating-point numbers

Chapter 2. MIPS Assembly Language
MIPS Processor Architecture: registers, register conventions, addressing modes, memory layout
MIPS Instruction Set: loads/stores, arithmetic instructions, logical instructions, shift/rotate instructions, branch/jump instructions
SPIM System Calls: Section A.9 on CD with text
SPIM Assembler Directives: .data, .text, .word, .asciiz, .ascii, .GLOBL, .align, .space
Arrays: element addressing 1-d, 2-d, 3-d, and higher
Walking pointer (section 2.15 called "Arrays versus Pointers")

In addition to knowledge about the above concepts, the following assembly-language programming skills are to be tested too:
1) translate high-level language control statements (while, for, if, etc.) into MIPS assembly language (be able to handle complex Boolean expressions involving ANDs, ORs, etc.)
2) translate high-level language code containing array accesses into MIPS assembly language

MATERIAL THAT WILL NOT BE ON TEST 2:

Chapter 2. MIPS Assembly Language
MIPS Instruction Set: three ML instruction formats
Subprograms: MIPS Register conventions
Section 2.10 Translating and Starting a Program
Section 2.11 How Compilers Optimize

ASSEMBLY-LANGUAGE PROGRAMMING SKILLS THAT WILL NOT BE ON TEST 2:

3) use MIPS register conventions to decide which arguments/parameters and local variables should be stored in caller-saved ($a and $t-registers) or callee-saved ($s-registers)
4) translate high-level language subprograms into MIPS assembly language (passing parameters into the subprogram using the $a registers, building the call frame on the run-time stack if necessary, save $s and $ra registers if necessary, passing the value returned by a function in the $v0 register, restoring $s and $ra registers if necessary, jr back to the caller)