Electrical and Computer Engineering Department
University of Maryland
College Park, MD 20742-3285

Glenn L. Martin Institute of Technology ◇ A. James Clark School of Engineering

Dr. Charles B. Silio, Jr.
Telephone 301-405-3668
FAX 301-314-9281
silio@eng.umd.edu

Summer 2007

ENEE350 Homework Set No. 2

(Due: Class 5, Mon., June 11)

Before making problem assignments we consider some definitions of "bit-by-bit" (or "bitwise") logical operations on binary *n-tuples* that represent the content of n-bit computer registers or memory locations. These definitions extend the standard logical operations of AND ($\wedge$), OR ($\vee$), exclusive-OR ($\oplus$), and NOT (complement). When the context is clear, logical complement or NOT is usually indicated by an overbar over the appropriate logical variable. Furthermore, exclusive-OR is also usually abbreviated as XOR.

Notation: A binary n-tuple is denoted: $X = (x_1, x_2, \ldots, x_n)$, where $x_j \in \{0,\ 1\}$, $j = 1,\ 2,\ \ldots,\ n$, and similarly for any other letter denoting a binary n-tuple (or the contents of a register or a memory location). (Recall that the notation $x_j \in \{0,\ 1\}$ means that x_j takes values in the set $\{0,\ 1\}$.

Definitions: Given binary n-tuples $X = (x_1, x_2, \ldots, x_n)$, $Y = (y_1, y_2, \ldots, y_n)$, and $Z = (z_1, z_2, \ldots, z_n)$, then

1. AND: $Z = X.\text{AND}.Y = X \wedge Y$ if and only if $z_i = x_i \wedge y_i$, $i = 1,\ 2,\ \ldots,\ n$;
2. OR: $Z = X.\text{OR}.Y = X \vee Y$ if and only if $z_i = x_i \vee y_i$, $i = 1,\ 2,\ \ldots,\ n$;
3. XOR: $Z = X.\text{XOR}.Y = X \oplus Y$ if and only if $z_i = x_i \oplus y_i$, $i = 1,\ 2,\ \ldots,\ n$;
4. NOT: $Z = .\text{NOT}.X = \overline{X}$ if and only if $z_i = \overline{x}_i$, $i = 1,\ 2,\ \ldots,\ n$.

Problems:

1. The UNIVAC 1100 series computers have 36-bit words and perform 1's complement arithmetic. Suppose UNIVAC 1100 registers $A1$ and $A2$ contain the following bit patterns in octal shorthand (express appropriate answers in octal shorthand too):

$$(A1) = 572053777777 \qquad\qquad (A2) = 206556400000$$

   a. Perform the following 1's complement fixed-point integer arithmetic operations and note whether magnitude overflow has occurred in each case. (Hint: Use 7's complement arithmetic on the octal shorthand).

      i) $A3 = A1 + A2$

      ii) $A3 = A1 - A2$

   b. Compute the following bit-by-bit logical operations and express your answers as 12-digit octal numbers:

      i) $A3 = A1.\text{AND}.A2 = A1 \wedge A2$
      ii) $A3 = A1.\text{OR}.A2 = A1 \vee A2$
      iii) $A3 = A1.\text{XOR}.A2 = A1 \oplus A2$
      iv) $A3 = A1.\text{XOR}.A1 = A1 \oplus A1$
      v) $A3 = .\text{NOT}.A1 = \overline{A1}$

   c. Read sections 5.2, 5.5, and 5.6 of Chapter 5 in the text by A. Tanenbaum, *Structured Computer Organization,* 5th ed., Prentice-Hall, 2006. Concentrate on general principles and scan the details of the machine examples.

2. The 12-bit register A in the DEC PDP-8 computer contains the following bit pattern in octal shorthand

$$(A) = 4071$$

   a. Assuming decimal digits in BCD code have been packed into A what decimal number is represented there?

   b. Pack the decimal number 957 into 12-bit register A on this machine using XS-3 (excess three) code, and specify in octal shorthand the resulting bit pattern.

3. $(31302.2023)_4 = (?)_{16}$

4. Read Chapter 2 of text by A. Tanenbaum, *Structured Computer Organization,* 5^{th} ed., Prentice-Hall, 2006, and work the following problems from Chapter 2:
   a. Problem 2-1.
   b. Problem 2-5.
   c. Problem 2-7.
   d. Problem 2-8.
   e. Problem 2-12.
   f. Decode the following binary ASCII text: 1001001 0100000 1001100 1001111 1010110 1000101 0100000 1011001 1001111 1010101 0101110.

5. Read Section 5.3 of Chapter 5 in text by A. Tanenbaum (actually read the part of Section 5.3 preceding section 5.3.1., read sections 5.3.1. and 5.3.2. and then scan sections 5.3.3 through 5.3.5) and work the following problem, which we will first introduce with some text about data representation of machine instructions using what are called "expanding opcodes".

   Because of short word length and other reasons cited by Tanenbaum in Section 5.3, the instruction repertoires of many computers take differing formats depending on the type of instruction. For example, those that refer to memory (called memory reference instructions) must somehow supply a memory address. Whereas, those that refer to only CPU registers or input/output (I/O) operations and not to memory locations (called non-memory reference instructions) can take on a different format. This differs from UNIVAC 1100 instructions which all have the same format. For instance, the IBM 360/370 series machines use a form of expanding instruction length comprising 2, 4, or 6 bytes, depending on the class of instruction; however, the operation code part of an instruction is always contained in the first byte (8-bits or octet) and never varies in length. The Digital Equipment Corp. PDP-11 has a basic 16-bit word length, where the length of the operation code (op code) field in an instruction varies with the type of instruction (*e.g.*, one operand instructions have a different op code length than two operand instructions, and so on). Within each op code class at least one opcode is usually reserved to point to the next class of instruction. For example, the DEC PDP-8 computer with a 12-bit word length has a 3-bit opcode field. Seven opcodes are memory reference instructions that allocate two more bits for address modification (an indirect bit and a page-zero bit) and use 7-bits for a memory address. The eighth opcode indicates non-memory reference instructions (called "operate class" instructions) in which the remaining 9 bits are used for encoding instructions that do not refer to memory.

   As the designer of a new computer it is your job to specify a set of "expanding opcodes" for a computer with a 20-bit instruction word that satisfies simultaneously the following instruction set requirements with 3 classes of instructions. Clearly indicate the bit patterns you choose to specify each format. Indicate bits that vary through (almost) all possible combinations with x's or y's; you can annotate them with an asterisk and clearly indicate which combinations among the x's are reserved for expansion to indicate the next format (clearly specify which combinations are reserved in format i and show explicitly how these combinations are used in format $i + 1$).

Requirements:

1st format: 30 instructions, each with one 3-bit and one 12-bit address field;
2nd format: 12 instructions, each with one 12-bit address field;
3rd format: 64 instructions, each with two 4-bit address fields.

(No provision need be made for any additional address modifier bits, such as those used to indicate immediate, indirect, indexed, etc., mode addressing.) Note that this is basically a counting problem. With n-bits one can represent uniquely no more than 2^n things, including machine instructions. Address and register designator fields represent bits in a fixed-length word that are reserved for use by assembly (and machine) language programmers and are thus not available in a particular format for encoding of the operation codes by the machine designer.

See p. 3 for yet another problem.

6. The IBM 370 series computers have 32-bit words and perform 2's complement arithmetic; so too do Intel 80386, 80486, and Pentium computers. The IBM 370 series has sixteen 32-bit general purpose registers ($R0, \ldots, R15$) and four double length (64-bit) floating point registers in its CPUs. Suppose IBM 370 registers $R1$ and $R2$ contain the following bit patterns in hexadecimal shorthand (express appropriate answers in hexadecimal shorthand too):

$$(R1) = BCDFAF00 \qquad\qquad (R2) = 206E5640$$

   a. Perform the following 2's complement fixed-point integer arithmetic operations and note whether magnitude overflow has occurred in each case. (Hint: Use 16's complement arithmetic on the hexadecimal shorthand).

      i) $R3 = R1 + R2$

      ii) $R3 = R1 - R2$

   b. Compute the following bit-by-bit logical operations and express your answers as 8-digit hexadecimal numbers:

      i) $R3 = R1.\text{AND}.R2 = R1 \wedge R2$
      ii) $R3 = R1.\text{OR}.R2 = R1 \vee R2$
      iii) $R3 = R1.\text{XOR}.R2 = R1 \oplus R2$
      iv) $R3 = R1.\text{XOR}.R1 = R1 \oplus R1$
      v) $R3 = .\text{NOT}.R1 = \overline{R1}$