## *Review Questions:*

# *Machine Code and Assembly*

1. If all the op codes of a particular computer are constrained to have the same length, i.e., the same number of bits, then what are the critical factors that determine the minimum value of the op code length?

2. Compare the portability of programs written in the language *C* with that of programs in Assembler. Explain the factors that are responsible for the presence or the absence of portability.

3. How does "Australian Rules Football" differ from American Football?

4. What are the two kinds of labels and what is the difference between them?

5. What are the four possible fields of a *Beboputer* assembly language statement? Which of them are obligatory? What more rigid field definition exists in some assemblers for other computers?

6. When are certain fields in a *Beboputer* Assembly Language statement forbidden?

7. What is a *declaration statement?*

8. What is the difference between the two unary operators '-' and '!' in *Beboputer* assembly language?

9. Explain with the aid of a diagram the meaning of each of the principal addressing modes: *Implied, Immediate, Absolute, Indexed* (which should really be called either *Indexed-Absolute* or *Absolute-Indexed*), *Indirect, Pre-Indexed Indirect,* and *Indirect Post-Indexed.*

Revised 27 Jan 99, 1420 hrs

# Answers to Selected Questions:

1. If all the op codes of a particular computer are constrained to have the same length, i.e., the same number of bits, then what are the critical factors that determine the minimum value of the op code length?

   *Answer:* The single determining factor is the total number of different op codes that must be represented. This is, in turn, a function of the total number of instructions and the number of addressing modes applicable to each instruction. Consider the set of instructions $\{I_i\}$, where $i$ ranges from 1 to $n$, and where each instruction $I_i$ is associated with some number $A_i$ of addressing modes. Then the total number of different opcodes that must be represented is given by:

   $$N = \mathrm{Sum}_{i=1}^{n} (I_i * A_i)$$

   The minimum length of the op codes is then defined by:

   $$L = \lceil (\log_2 N) \rceil$$

   Note that the symbols $\lceil\ \rceil$ denote the *ceiling function,* which is defined as the smallest integer equal to or greater than the quantity between the two symbols. Thus, for example, if $N$ has a value of $240_{10}$, then the base-2 logarithm of $N$ will be somewhere between 7 and 8, actually very close to 8, and the ceiling function of that would be 8. Likewise, if $N$ has a value of 130, then again the base-2 logarithm will be between 7 and 8, in this case very close to 7, and the ceiling function would again have a value of 8.

2. Compare the portability of programs written in the language *C* with that of programs in Assembler. Explain the factors that are responsible for the presence or the absence of portability.

   *Answer:* See text on pages 12-5 through 12-7 of *Bebop Bytes Back,* as well as Figure 12.4.

3. How does "Australian Rules Football" differ from American Football?

   *Answer:* See the sidebar on page 12-7 of *Bebop Bytes Back.* (NOTE for excessively serious Graduate Students: No, you do NOT need to know the answer to this question to get a good grade for this course.)

4. What are the two kinds of labels and what is the difference between them?

   *Answer:* An address label defines only an address; its value is the memory address associated with the machine instruction where its op code is placed in the program, and therefore it is almost always utilized within the program inside square brackets as a memory reference. Its principal utility is in specifying destination locations for circumstances where the execution order of program operations is required to deviate from the default sequential order. A constant label, on the other hand, defines a memory location where data are stored; it can therefore be used in the program either inside square brackets to specify the address of the data or without square brackets, in which case the reference is to the contents of its memory location (i.e., its value). See pages 12-8 through 12-12 in *Bebop Bytes Back.*

5. What are the four possible fields of a *Beboputer* assembly language statement? Which of them are obligatory? What more rigid field definition exists in some assemblers for other computers?

   *Answer:* The four possible fields are the *Label* field, the *Operation* field, the *Operand* field, and the *Comment* field. In *Beboputer* assembly language, none of these fields is universally obligatory, although the *Operand* field is obligatory **when** the instruction mnemonic specifies an operation that requires an operand, and the *Label* field is obligatory for an .EQU directive statement. The fields are all defined syntactically. A character sequence that constitutes a legal *Beboputer* label, followed by a colon and situated either at the left extremity of the statement line or separated from it exclusively by spaces, defines the label field. The *Operation* field is either the leftmost field in the statement or the second field after a label field, and it consists of either one of the 43 operation mnemonics listed in Figure C.2 or one of the assembler directives (.ORG, .END, .EQU, .BYTE, .2BYTE, or .4BYTE), also known as *pseudo-ops.* The *Operand* field in the *Beboputer* assembly language consists of a single legal operand specification in the syntax appropriate to the addressing mode of the instruction, i.e., either:

   (i) an immediate operand consisting of a hex or binary value of up to 8 bits in length or the decimal equivalent, or a label or an expression which translates to a value of 8-bit length; or

   (ii) a pair of square brackets enclosing a memory address consisting of either a numeric value that can be expressed in up to 16 bits or a label or expression which translates to a value of up to 16 bits, and possibly also including a reference to the index register; or

   (iii) two pairs of square brackets enclosing a memory address, and possibly also including a reference to the index register.

   The *Comment* field consists of the '#' character followed by any number of text characters up to the end of the statement line.

   In some assemblers, the various fields are rigidly defined on the basis of number of character positions from the leftmost extremity of the statement line. Thus, a colon might not be necessary to delimit the end of the label field, or a '#' to delimit the beginning of the comment field. This is an artifact of the "old days" of computer history, when programs were entered on punched cards having 80 columns of rigidly fixed equal width, each of which held one text character.

6. When are certain fields in a *Beboputer* Assembly Language statement forbidden?

   *Answer:* Label fields are not allowed in statements bearing the *.ORG* or *.END* pseudo-ops (directives).

7. What is a *declaration statement?*

   *Answer:* A *declaration statement* is used to declare the value of a constant that will be used in the program.

8. What is the difference between the two unary operators '-' and '!' in *Beboputer* assembly language?

   *Answer:* The unary operator '-' takes the Two's Complement of the number specified immediately to its right, whereas the '!' operator takes the Ones' Complement.

9. Explain with the aid of a diagram the meaning of each of the principal addressing modes: *Implied, Immediate, Absolute, Indexed* (which should really be called either *Indexed-Absolute* or *Absolute-Indexed*), *Indirect, Pre-Indexed Indirect,* and *Indirect Post-Indexed.*

   *Answer:* See pages 12-29 to 12-32 and Figures 12.27 to 12.34 in *Bebop Bytes Back,* **or** pages C-3 through C-12 and Figures C.1 through C.9.