Assembly Language Programming

Programs written in high-level languages such as BASIC, Ada, and Pascal are usually converted by compilers into *assembly* language (which in turn is translated into *machine* language programs — sequences of 1's and 0's — by an assembler). Even today, with very good quality compilers available, there is the need for programmers to understand assembly language: First, it provides programmers with a better understanding of the compiler and what are its constraints. Second, on occasion, programmers find themselves needing to program directly in assembly language in order to meet constraints in execution speed or space (for example, writing games for micros and arcade machines). A good programmer can typically tweek and tune an assembly language program to run significantly better than the code generated by even the best compilers.

The ACSL Assembly Language, *AAL*, runs on an ACSL computer that has an unlimited amount of memory. Each "word" of memory contains a decimal integer in the range –999,999 through +999,999. Additions, subtractions, and multiplications are performed modulo 1,000,000. For example, 999,998 plus 7 equals 5. Division is performed in the conventional sense, but the fractional part of the answer is dropped — *not rounded off*. For example, 14 divided by 5 is 2.

Execution starts at the first line of the program and continues sequentially, except for "branch" instructions, until the "end" instruction is encountered. The result of each operation is stored in a special word of memory, called the "accumulator" (ACC). Each line of an assembly language program has the following fields (lower-case italics indicates optional components):

label **OPCODE** **LOC** *comments*

The *label* is a character string beginning in the first column. Valid **OPCODE**'s are listed in the chart below. The **LOC** field is either a reference to a label or "immediate data". For example, "LOAD A" would put the contents referenced by the label "A" into the ACC; "LOAD #123" would store the value 123 in the ACC. Only those instructions that do not modify the LOC field can use the "immediate data" format. In the following chart, they are indicated by an asterisk in the first column.

literal (=)

	OPCODE	ACTION
*	LOAD	Contents of LOC are placed in the ACC. LOC is unchanged.
	STORE	Contents of ACC are placed in the LOC. ACC is unchanged.
*	ADD	Contents of LOC are added to the contents of the ACC. The sum is stored in the ACC. LOC is unchanged. Addition is modulo 1,000,000.
*	SUB	Contents of LOC are subtracted from the contents of the ACC. The difference is stored in the ACC. LOC is unchanged. Subtraction is modulo 1,000,000.
*	MULT	The contents of LOC are multiplied by the contents of the ACC. The product is stored in the ACC. LOC is unchanged. Multiplication is modulo 1,000,000.
*	DIV	Contents of LOC are divided into the contents of the ACC. The signed integer part of the quotient is stored in the ACC. LOC is unchanged.
	BE	Branch to instruction labeled with LOC if ACC=0.
	BG	Branch to instruction labeled with LOC if ACC>0.
	BL	Branch to instruction labeled with LOC if ACC<0.
	BU	Branch unconditionally to instruction labeled with LOC.
	END	Program terminates. LOC field is ignored.
	READ	Read a signed integer (modulo 1,000,000) into LOC.
*	PRINT	Print the contents of LOC.
	DC	The value of the memory word defined by the LABEL field is defined to contain the specified constant. The LABEL field is *mandatory* for this opcode. The ACC is not modified.

way to do an init without doing: Load 1 Store A3 same thing A DC 1

References

We chose to define our own assembly language rather than use a "real" one in order to eliminate the many sticky details associated with real languages. The basic concepts of *AAL* are common to all assembly languages. A reference manual for any real assembly language should prove helpful to prepare for this category. Also, the following article presents many of the concepts of assembly language programming in a very readable and fun way:

Dewdney, A. K. "Computer Recreations," in *Scientific American*, May 1984, pp. 14-22.

Sample Problems

After the following program is executed, what value is in location *TEMP*?

```
TEMP    DC      0
A       DC      8
B       DC      -2
C       DC      3
        LOAD    B
        MULT    C
        ADD     A
        DIV     B
        SUB     A
        STORE   TEMP
```

The ACC takes on values −2, −6, 2, −1 and −9 in that order. The last value, −9 is stored in *TEMP*.

After the following program is executed, what is the final value of *A*. The data for the program is 3.

```
        READ    N
A       DC      1
START   LOAD    N
        SUB     =1
        BE      RSLT
        STORE   N
        LOAD    A
        ADD     =2
        STORE   A
        BU      START
RSLT    END
```

This program finds the *N*th odd integer. The following table gives the values of *N* and *A* through execution:

N	*A*
3	1
2	3
1	5
0	5

Thus, the final value of *A* is 5.

If the following program has an input value of *N*, what is the final value of *X* which is computed? Express *X* as a mathematical expressions in terms of *N*.

```
        READ    X
        LOAD    X
TOP     SUB     =1
        BE      DONE
        STORE   A
        MULT    X
        STORE   X
        LOAD    A
        BU      TOP
DONE    END
```

This program loops between labels TOP and DONE for *A* times. *A* has an initial value of *X*, and subsequent values of *A*−1, *A*−2, ..., 1. Each time through the loop, *X* is multiplied by the current value of *A*. Thus, $X = A*A-1* \cdots *1$ or $X=A!$. Since the initial value of *A* is the number input (i.e., *N*), $X = N!$.