The VC1620 Assembly Language

The following is a summary of the assembly language for the VC1620. In class we will discuss the rational for the choices made here.

Statement Format:

An assembly language statement consists of from one to four fields. These are:

- Label - used to reference the statement. It is optional.
- Operation Code - a symbolic name for the numeric machine language op code.
- Operands - Used to supply additional information. For a machine language instruction these will be labels for the addresses specified.

Labels start in column 1, all other fields separated by blanks and/or tabs and/or a comma. The purpose of using commas is to enhance the readability when separating operands.

Symbolic Operation Codes: Note: operation code 6 is deliberately missing

01 ADD	02 SUB	03 MULT	04 DIV	05 COPY	07 READ	08 WRITE
09 B	10 BM	11 BZ	12 BP	13 HALT		

Symbols: (I.e. labels and operands)

Symbols are from 1 to 10 characters in length, the first of which is a letter and the remaining may be letters and digits. The length limit is mostly to add another rule to the syntax of symbols. For most assemblers, this would not be a requirement.

Addresses:

An address my be specified by using the a label.

Assembler Language Instructions

These provide information to the assembler about translating a program.

DC - define constant. The constant is a decimal integer placed in the operand field.

DS - define storage. The operand specifies the number of words of storage to be set aside.

ORG -define origin. The operand specifies the address at which the translation of the next instruction will be generated, END – indicates that there are no additional statements to translate.

Comments:

Data after a ";" is a comment. Comments may appear anywhere within an instruction or by themselves. Blank lines are ignored.

Case Sensitivity

All symbols will be case sensitive. Operation codes (E.g. like "ADD") may be written in upper or lower case or some combination of the two. So operation codes are case insensitive.

Commas

Commas may be used as optional separators. They are not required, but they can make the code a more readable. Especially between operands.

Example:

The following is an assembler language program which will read in a number "n" and then compute and print the value of n!.   This is a little more efficient than the one we did in machine language.

```
        org 100
        read n       ; No need to supply seconds operand
more    mult fac, n;This is a comment with no space

;Here is a comment that sit on its own line.
        SUB n, one
        bp more, n
        write fac
        halt         ; no need for any operands.
n       ds 100; just to show that you code can handle big areas.
fac     dc 1
one     dc 1
test    dc 123456 ; show your program can handle big constants.
        end
```