What is the AS8 assembler?

As8 is a simple assembler for an Intel 8008 microprocessor. It uses the original (old) mnemonics, unlike the 8080-like mnemonics of later compilers. It is written as a single ANSI-c source file, so should compile on any computer that has a C compiler.

How to run it:

To run it, the program is called from a command-line interpreter, such as a DOS window on an Windows computer. If you have an assembly file named "test.asm" you could simply call the program as...

```
as8 test
```

and the program will open the file called "test.asm", assemble it, write the output list file into "test.lst" and write an intel hex output file called "test.hex".

More specifically, the program usage is...

```
Usage: as8 [options] infile
    where  is assembly code file, extension defaults to .asm
    and options include...
    -v        verbose output
    -nl       no list file (default is to make .lst file.)
    -d        debug assembler (extra output)
    -bin      makes output binary ROM file, otherwise intel hex
    -octal    makes unidentified 3-digit numbers octal (default decimal)
    -single   makes .lst file single byte per line, otherwise 3/line.
    -markascii makes highest bit in ascii bytes a one (mark).
```

where "infile" can have ".asm" specified, or just the base name of the file, with ".asm" inferred. The options are..

- **-v** A verbose output, with lots of information, and tells what it is doing as it processes each line. This is mostly debugging information, and won't be very helpful to most users, unless the program isn't working quite right for you.
- **-nl** No list, do not make a "*.lst" list file. By default, a list file is made which shows the details of the assembly and machine codes.
- **-d** Turns on extra debug information. This is mostly if the program isn't working right, and won't be useful to most users.
- **-bin** This makes the output file a "*.bin" which will be a binary 16K file of each assembled byte. If you don't want the whole 16K, change the program yourself, or have a post-processor program pull out the portion you want. By default, and Intel-Hex format ASCII file is generated, which is a fairly obsolete ASCII file that specifies addresses, and then data that follows at that address. Historically, most EPROM writers expected Intel-Hex format files, which are documented well on internet file format sites.
- **-octal** makes any 3-digit number interpreted as Octal. By default, any number starting with zero is interpreted as octal (like "045") but starting with a non-zero will be decimal (like "255" or "14"). But if this this option is specified, any 3-digit number, even if it starts with a non-zero, will be interpreted as an octal representation (this is how many old 8008 assemblers worked, and is

needed for SCELBAL.)

- **-single** makes the list file just have a single vertical line of the code bytes. For example, if a "jump" instruction is assembled, that line will generate 3 separate bytes. By default, all 3 bytes will be listed on a single listfile line. But if this option is specified, each output line in the listfile will have just one byte. Assembly code lines like "DATA" statements may still generate multiple lines in the list file if they generate over 3 bytes. In general, this option is helpful if long programs will be manually entered on a front panel, as all bytes and data will be in a long vertical list, at the expense of somewhat longer files.
- **-markascii** makes any ASCII letter in a "data" statement be assembled so that the highest order bit (128) is set (marked.) This is required for SCELBAL. I believe this type of ASCII convention was popular with PDP-8 computers by Digital Equipment Corporation, but of course is now obsolete.

Input File Format

The program takes an assembly code ascii file full of lines of code and comments. Where ever the ';' or '\' character is seen, the rest of the line is considered a comment. The general format of the line is as follows:

```
label:   opcode argument(s)
```

where "label:" is optional, if not included in a line, at least one white space character (a space or a tab is reasonable) before the opcode. Several pseudo-ops are available:

- ORG set the address of the current code to the argument.
- EQU requires a label before it, and sets the label to the following argument.
- DATA specifies some number of bytes are to be inserted at the current location. the argument can be a list of numbers ("2,4,6,8,232") or a character string ("ENTER THE DATA") or to just define space an asterisk followed by a decimal number will reserve some number of bytes ("*32" will just reserve 32 bytes.)
- CPU requires an argument of "8008" or "i8008" or an error will occur. This pseudo-op is not required or recommended, but here for cross-compatibility with some other assemblers.
- END is not required, but typically signifies the end of the code in some assembly programs.

The opcodes or mnemonics are mostly 3-letter, and are as specified by early 8008 documents. It should be noted that after Intel came out with the 8080 microprocessor, they created slightly different opcodes for the 8008 to make the assembly code a bit more similar (and somewhat interchangeable.)

Number Specifications

Numbers in the assembler may be specified in many ways.

A multiple digit number which does not start with zero is interpreted as a decimal number (unless the octal option is set, in which case any 3-digit number will be considered octal, even if it doesn't start with a zero.

A multiple digit number which starts with a zero is always interpreted as an octal number.

A number which starts with "0x" (like "0xFF" or "0x1F3") will be read as a hexadecimal number.

A string of ones and zeros followed by a "B" will be interpreted as a binary number (like "01001011B").

Arithmetic/Operations

An argument can include simple arithmetic which will be interpreted from left to right. "LOC*2+1" would be valid if "LOC" is a valid defined label. Up to 4 arguments can be separated by 3 operands ("+","-","*", or "/").

The operand "\HB\" can precede an argument/label, and returns the high-byte of the value. Similarly, the operand "\LB\" can precede an argument/label, and returns the lower-byte of the value. These are useful for loading the "H" register with the high-byte of an address, and the "L" register with the low-byte.

Just as "\HB\" and "\LB\" can separate the high-order-byte and low-order-byte of a number, the "#" sign will attach them. This is commonly need in "ORG" statements where the high and low bytes are known, but must be placed into a single number (example "START: ORG 001#0120" is helpful when we work with octal numbers (common with the 8008) and it's not obvious how to combine them directly ("ORG 001*0400+120" would do the same thing, but is not as straightforward.)

More about DATA

As mentioned, the "DATA" pseudoop will either define numeric values to be inserted into the program, or reserve some number of bytes in the program. A list of comma delimited arguments may have up to 12 such arguments. They can be simple numbers (interpreted as above) or labels, or mathematic expressions, as mentioned above. More than 12 on a line, however, generates an error.

If the "DATA" statement has a string a few escape/control symbols are allowed, such as "HELLO\n" which will place a newline at the end of the string. The allowed sequences are "\n", "\t" or "\0" at this point. Other numbers will require a following DATA statment with numeric values specifically entered.

If the "DATA" statement is followed by a '*' and then a number, this number will be interpreted as decimal, and reserve that number of bytes.

But How Do I use it?

Okay, the best way to learn this assembler is to look at example input files. Look at the "SCELBAL" source code or the "memtst.asm" code file. That's the best way to figure out what's allowed and not.

About Me:

I'm a vintage computer hobbyist, have very little free time. If you have questions or suggestions, email me, but realize I may be hard to get a hold of. If you're persistent, I tend to reply. You can email me at tejones777 at (use the at sign) aol.com. If I don't respond, I may just be swamped, but don't give up if you really need something.