Here's some Cobol stuff I've been playing with on the Altair Clone.

I ported the primes program from Basic to Cobol and was able to compile,link, and run it from the CP/M command line.

The primes benchmark runs quite a bit slower with Cobol. Shortly after Microsoft released this compiler, they abandoned it and began licensing Cobol from MicroFocus. Had they kept working on it, they would have needed many improvements to be compliant with the standards current at that time. For example, there is no support for inline/nested PERFORM statements and that is why the primes program uses GO TOs instead of PERFORM structure. Performance is also reduced because of the lack of floating point arithmetic and the use of Packed Decimal for non-integers. As a programmer forced to use comparison tolerance code with floating point in financial apps, in languages such as Fortran, I always appreciated the exact precision with Packed Decimal even though it ran more slowly. Thankfully, Java brought Packed Decimal back with the BigDecimal data type although I usually had to explain to junior programmers why they would want to spend the extra effort to use it. And Python has a decimal module you can import.

Regards,

-Mark Rosenthal

The Cobol-80 runtime is a P-code interpreter. The Cobol-80 compiler generates P-code which makes calls to the Cobol-80 runtime library using the RUNCOB.COM executor at runtime (this is done for you automatically).

Floppy disk image requirements:

The Cobol-80 compiler requires more memory than was available on early PCs. As a result, it's been structured to run in phases that follow the 4 "Divisions" (IDENTIFICATION, ENVIRONMENT, DATA, PROCEDURE) of a standard Cobol program. Each phase is executed in sequence with the code in the four overlay ("OVR") files that must accompany the COBOL.COM compiler front end. These overlays overwrite the memory used by the previous compiler phase. Because the compiler expects the "OVR" files to reside on the same drive as the compiler front end, it is necessary to run the compiler from the device containing the compiler. When all of the necessary system files for Cobol are gathered on a single CP/M diskette image (there are more we've not mentioned), there is not much free diskette space left for program source and intermediate files for compile and link. A few of the non-essential files supplied with the Cobol-80 distribution have been omitted from the Cobol-80 diskette image due to space constraints of the diskette:

1. All of the compiled terminal drivers except for ANSI which works with Teraterm.
2. CREF80.COM, the Microsoft cross-reference utility that should be included with the other compiler/assembler diskettes.
3. LD80.COM, a duplicate Microsft link/loader. The provided L80.COM works fine.
4. LIB.COM, the Microsoft library maintenance utility also included with the other compiler/assembler diskettes such as Fortran.

5. M80.COM, the Microsoft assembler included with the CP/M software development diskettes such as Fortran.
6. SEQCVT.COM, a program for converting the format of old cobol data files.
All of these files/programs can be downloaded if desired from the altairclone website.

This leads to the following "ideal" floppy configuration for compiling and linking Cobol-80 programs on the Altair Clone:
Floppy drive 0: CP/M version 2.2 diskette image. (CP/M device A:)
Floppy drive 1: Data diskette for Cobol source files, intermediate files created during compile and link. (CP/M device B:) Use the CP/M FORMAT command to create an empty diskette. Use the Altair Clone Configuration Monitor to save the empty diskette as a disk image (DSK) file.
Floppy drive 2: Cobol-80 compiler, libraries, runtime support and utilities. (CP/M device C:)

Cobol source file format requirements:

The usual Cobol source file as shown in the Cobol-80 documentation are required. However, Cobol-80 requires the 4 four (at least minimal) program divisions mentioned above. A program without all four divisions may compile with out error, but unlike most Cobol compilers, such a Cobol-80 program will compile and link successfully, but fail at runtime with an error something like this:

** RUN-TIME ERR:
OBJ. CODE ERROR
HELLO
00000

Where "HELLO" above is the PROGRAM-ID from the IDENTIFICATION DIVISION of your Cobol program.

Running the compiler, linker, and the executable program:

Before compiling your first Cobol program, you must configure the display terminal device driver. Assuming that you will be using TeraTerm with your Altair Clone, copy CDANSI.REL to CRTDRV.REL. CDANSI.REL provides ANSI/VT terminal emulation. This only needs to be done once unless you switch to a different type of display terminal.

With CP/M 2.2 loaded on floppy device 0, the cobol-80 diskette into floppy device 2 and an empty diskette into floppy device 1 log onto device C:.

1. Copy your Cobol source file(s) to CP/M device B:. The primes.cob file provided is a good example and lets you compare Cobol performance with the Basic version of the same program on the Altair Clone.

From the C prompt (C>)
A:PIP B:PRIMES.COB=C:PRIMES.COB
2. Compile with the following command from device C:
COBOL ,TTY:=B:PRIMES/R

3. Link with the following command (this will take a while) from device C:

L80 B:PRIMES/N,B:PRIMES/E

4. Run the program with the following command from device C:

B:PRIMES

Because the Cobol-80 launcher for your linked COM file expects the Executor (RUNCOB.COM) to exist on your logged device, if you try to run your program while logged to device B:, you will get an error message like this:

** COBOL Executor not found