ACTxx Cross Assemblers

From SORCIM, 2310 Lundy Avenue, San Jose, CA 95131
Requires CP/M 1.4 or better with 32K of RAM
$175 each
Reviewed by Steve Newberry

For some time now I have urged that microprocessor manufacturers should make 8080-hosted cross-assemblers for their processors available if they are really interested in selling *chips* (instead of development systems). For every development system capable of producing code for a given processor, XYZ, there are at least ten 8080/Z80-based systems running on CP/M, and I leave it to you to guess at the number of 8080/Z80 products that have been developed on these systems.

Alas, to no avail. Cross-assemblers that have been offered in the past have generally been written in ANSI Fortran and only been accessible on the maxicomputers – at nontrivial hourly rates. The motives for such a policy may have seemed plausible at the time, but the practical consequences have proven to be less than ideal. In order to do development work for the XYZ processor one had to have either access to an XYZ development system running the resident assembler or to a mainframe (usually a 370) running the cross-assembler. Either approach imposed an initial overhead (for rental fees alone!) of such magnitude that hundreds (thousands?) of projects which might have been developed for the XYZ were never undertaken. One may certainly speculate about the consequent loss of sales (and jobs!) which might have resulted.

Fortunately, the small software houses are beginning to address themselves to this need, and it is now possible to purchase 8080-hosted cross-assemblers for all the major micros except the Z8000 and the MC68000.

Sorcim, the producer of Pascal/M and Supercalc, has for nearly two years been distributing a family of cross-assemblers which run on the 8080 under CP/M and generate code for 8080, Z80, 6502, 6800, 8086/8088, and 6809. The family is called ACT ("Assembly Coded Translator"), and each assembler package is called ACTxx (where "xx" is one of: 80, 65, 68, 86, or 69). The current release, Version 3.5, consists of a CP/M-compatible, 8-inch diskette and a 70-plus page *User's Reference Manual.* The ACTxx diskette contains seven files (eight in the case of ACT80):

- ACTxx.COM – assembler
- DOCOPxx.asm – source file containing all ACTxx mnemonics
- IPARAMS.acd – file of definitions common to all assemblers
- ACD/EG.asm – example demonstrating use of LINK pseudo-op
- UTIL.acd – utility subroutines (ACT-80 only)
- DATTIM.com – sets Date and Time for ACTxx listing header
- INSTALLA.sub – copies ACT system to system disk under CP/M
- INSTALLC.cmd – copies ACT system to system disk under CDOS

The manual is punched for a three-hole binder. There is no index, but the Table of Contents is intelligently done, and all the information is conveniently accessible. The command line to CP/M invoking ACTxx follows the same protocol as that of Pascal/M, for example:

A>ACT80 <filename> <cr>

assembles <filename>, puts the (absolute) hex file on the selected disk under the name <filename>.hex, and doesn't give you a listing unless you ask for it.

Assembly is extremely rapid. I used RELOC.ASM (from the CPMUG library) as a benchmark and timed ACT80 against the CP/M assembler ASM. (ASM is *really* fast.) RELOC is a 10K file which ASM assembled in 16 seconds as compared to 18 seconds for ACT80. (The tests were run on a 4 MHz Z80 with a double-density 8-inch floppy disk.) That means that the very substantially increased power of ACT80 over ASM (macros, many pseudos) is purchased at about a 10% increase in assembly time. (MAC also ran the same job in 18 seconds.)

Although the other cross-assemblers were not compared for speed, they should be comparable because of the way in which they are implemented. The whole macro expansion portion is constant from one assembler to another; the only thing that changes is the table containing the pseudo instructions, the target machine mnemonics, and the corresponding machine codes.

8080-Resident and Z80 Cross-Assembler

ACT80 will assemble both for the 8080/8085 and for the Z80. Originally, the assembler had been produced for Sorcim in-house use in writing Pascal/M. Since the programmers were already familiar with the mnemonics for CDC mainframes, the Sorcim mnemonics initially had a distinctively CDC-like flavor. When it was decided to market the ACT family of cross-assemblers, the mnemonics were extended to include the entire Intel set (including the '85 RIM and SIM instruction). The Zilog set is still only partially represented, although there are "Zilog-like" mnemonics which complete the Z80 instruction set. However, the Sorcim mnemonics regard movement between registers as moves (MOV), but memory-register moves as loads (LD), register-memory moves as stores (STO), and constant-register moves as load-constants (LK).

In other words, Z80 programs written on ACT80 can be assembled with only a little massaging (a few macros in ED or WM), by translators which recognize the standard Zilog set of mnemonics, but transforming source programs in the Zilog notation to the Sorcim set is rather tedious.

6502 Cross-Assembler

Turning to the other cross-assemblers, we find that ACT65 recognizes the MOS Technology mnemonics with only three exceptions, and these are trivial to translate in either direction:

(1) MOS Technology encloses address expressions in round parentheses where Sorcim uses square brackets.

(2) Some 6502 assemblers use a single prefixed quote-mark (apostrophe) to signal a one-character string. Sorcim requires that all strings be enclosed between balanced quote-pairs.

(3) In the Sorcim assemblers, the symbols "<" and ">" are only used as *binary* operators, and the corresponding unary operators are "low" and "high."

6800 Cross-Assembler

The ACT68 mnemonics are a compatible superset of the Motorola 6800 set, and only minor differences exist in expression evaluation and pseudos:

(1) ACT68 observes normal operator precedence.

(2) ACT68 requires that quote marks demarcating character strings be

paired (as in the ACT65 case noted above).

(3) ACT68 does not support

NAM (use TITLE instead)
MON
OPT (use LIST instead)

(4) ACT68 requires that comments be preceded by semicolon (or asterisk if in column one).

8086/8088 Cross-Assembler

To appreciate the difference between the Intel and the Sorcim 8086/8088 assembler mnemonic sets, it is helpful to know something about the structure of the Intel assembler MCS-86.

In designing a mnemonic instruction set, one usually attempts to strike a balance between two desirable but mutually incompatible ends: (1) a small simple fast assembler program, and (2) a small simple easily learned set of mnemonic operator names.

When the instruction set is small to begin with, the problem is fairly tractable, but as the instruction set grows, it becomes necessary to resort to the device of "operator overloading" to keep the distinct mnemonics down to a reasonable number. This means that the assembler must be capable of recognizing that an operator can take several different categories of operand and be able to distinguish these categories correctly.

For example, the Zilog mnemonics for the Z80 compress all the LOAD, LOADCONSTANT, STORE, and MOVE operations into the single mnemonic "LD," and give to the assembler the responsibility of distinguishing between registers and memory and constants, as well as operand lengths (byte versus word). This makes for a larger, more complex (expensive), slower assembler, but also makes it much easier for the programmer to learn the machine.

The designers of the Intel MCS-86 assembler saw fit to "make a virtue of necessity" by making the MCS-86 assembly language a *strongly typed* language. Thus, in MCS-86 (as in Pascal and Ada) the assembler looks up the number of bytes assigned to a variable every time that variable is accessed in the source code, and checks that only arguments of the same length may be acted upon by a single operation. (There is an override, but it is not relevant here.) Aside from enforcing an additional level of assembly-time error checking, this also permits the large instruction set of the 8086 to be compressed into a much smaller, highly overloaded set of mnemonics.

Sorcim adheres to the Intel mnemonics *almost* as nearly as can be accomplished without actually going over to the strongly typed restrictions of MCS-86. This is largely accomplished by distinguishing word from byte operations through a "B suffix" convention. In effect, the programmer learns (almost) the same set of mnemonics, *but* is required to inform the assembler when byte operands are forthcoming by suffixing the letter "B" to the operator mnemonic. Thus, an ADD instruction is applied to byte (rather than word) operands by writing ADDB. The other differences are (1) ACT86, like its siblings, also differentiates between register-register MOVes, register-memory STOres, and memory-register LoaDs; and (2) the "destination, source" ordering of operands is reversed in the case of the STOre instructions.

Here again, the practical consequences are that it is easy to transport ACT86 source code to MCS-86 (by defining macros to delete all the "B suffixes," replace the LD and STO, etc.) but rather tedious to port in the other direction.

6809 Cross-Assembler

The remarks concerning ACT68 apply here as well. There are two additional pseudos: SETDP (sets the assembler's pseudo Direct Page Register), and FAIL (synonym for ERR). ACT69 also does not support the Motorola REG pseudo. Otherwise the Sorcim mnemonics are a compatible superset of the Motorola standard.

Features Common to All Members of the Family

Command line options allow you to: specify the disk drive to which the hexfile will be sent (or you may send no hexfile at all), specify the destination of the listfile (lineprinter, console, diskdrive, or no listfile), set page size (defaults to PS=58), set line length (defaults to SL=140), specify either of two flavors of cross-reference map (default lists globals only), specify library files to be linked in with the program, and specify the origin (ORG) at which the program is to be loaded. The manual is very clear about how to specify these options.

The pseudos include the standard set you'd expect to find: DB, DW, DS, ORG, EQU, SET, PAGE (or EJECT), TITLE, SPACE, the conditional assembly controls IF, ELSE, ENDIF, a toggle (LIST) to control listings (invaluable in debugging macros), END, LINK (approximately equivalent to $INCLUDE or MACLIB), MACRO/ENDMACRO, and several built-in macros. Macros may take up to nine parameters and may be tested.

In addition to these are several pseudos which call for special comment. LOC permits you to assemble a chunk of code to LOAD at a different address than that at which it is to execute, load it at the ORG address, then move it to the LOC address and execute. This sort of thing arises when programs or pieces of programs in ROM need to be moved to RAM for execution, or in writing transient device drivers which are to be relocated to lie just beneath the operating system. You may not use LOC very often, but when you do need it you'll bless the designer of ACT for having provided it.

Another nice touch is VFD. This permits you to write your own assembler (for some other processor) in an *intelligent* manner. Of course, the macro facility would allow you to grind out an assembler by writing a separate macro for every possible opcode – but this would entail writing out hundreds (or even thousands!) of separate macros. VFD permits you to specify arbitrary bit-fields within a byte. In this way you can construct your assembler the way it *should* be done: as a fixed pattern of bits for each generic opcode, with variations in particular bit-fields specifying the source/destination operands. (To find a feature like this in a $175.00 assembler is extraordinary!)

The file IPARAMS.acd is a library file of symbol definitions: EQU's for the standard ASCII control characters and the CP/M system-call functions and magic entry-points. I found it convenient to edit in an "END" statement and adopt the convention that *all* programs are terminated with a "LINK IPARAMS.ACD" statement. The file UTIL.acd (found only in the ACT80 package) contains several flavors of string move/compare, block-fill, and other commonly used functions.

Another thing I like about these assemblers is their ability to treat all letters as though they were upper case without forcing lower-case to upper-case conversion on the listing. It's much easier to read a listing when it looks like the original source! The ACT assemblers also allow you to use "_", ".", and "#" as separation ("break") characters. ACT treats the underline "_" as a "ghost" character (it is not significant). Thus, you can write variable names and labels as things like "iteration_#" or "error.exit". All this makes it convenient to write very *readable* source and to generate very *readable* listings. And that means time and money saved in development and maintenance.

DDJ

Sorcim also provides the same packages for a variety of other chips besides the 8080. Contact them for details. – Ed.