because its extensive instruction set has to be mapped to the 8086 instructions, as well as the 8080 registers.

The 8086 does, however, have a rich instruction set that covers all the bit instructions and most of the block instructions. The problems encountered in mapping instructions and registers can be formalized and solved by using a variety of software tools.

To convert a source file running under CP/M-80 so as to assemble it with an 8086 assembler requires either the use of a code translator or a considerable effort with a program editor. At least three commercial products come under the translator category:

- XLT86 from Digital Research Inc.
- TRANS86/ACT86 from Sorcim
- The Z80-to-8086 translator from Seattle Computer Products

Intel also has a translator, CONVERT 86, but it is sold only as part of a large software package intended for its OEM customers and doesn't run under CP/M. In addition, some reports indicate that Microsoft may soon bring out a translator.

Two issues must be addressed when converting from a CP/M-80 source:

1. Where is the source coming from?

   Z80 instruction mnemonics?
   (a) Zilog
   (b) TDL (Technical Design Labs)
   (c) Intel (Digital Research with macros)
   (d) Sorcim/ACT-80

   or

   Straight 8080/8085 code with no macros?

2. Where is the resultant code going?

   CP/M-86 or MS-DOS?

The three translators under review do not always approach these issues in the same way. Furthermore, they all handle register mapping somewhat differently. We'll look at the translators now one at a time. Then we'll see how they translated the same CP/M-80 listing. With the listing in hand, we'll see how each of the translators handled register mapping. Finally, we'll turn to the subject of how the two different 16-bit operating systems affect program translation.

XLT86 from Digital Research

Digital Research's XLT86 takes standard 8080 source code in a format compatible with ASM, MAC, or RMAC assemblers and converts the 8080 source code to 8086 source code in a format compatible with *ASM86* operating under either CP/M-80 or CP/M-86. Since XLT86 is written in PL/I-80, the translator can run either stand-alone under CP/M-80 or for cross development under VAX/VMS. It produces optimized 8086 code in a five-phase, multipass process, doing global data-flow analysis to determine optimal register usage.

Although macro definitions are not supported, conditional-assembly directives are. The *XLT86 User's Guide* suggests that if you want macro expansion, you can use a pass through MAC or RMAC to produce a PRN file that can be edited (removing the first few columns of generated hexadecimal code) to produce an expanded source file for input acceptable to XLT86. XLT86 does not recognize Z80 instructions. XLT86 passes repeat loops through to the 8086 source code.

XLT86 analyzes the source program in its entirety, determining the block structure and the register/flag usage. Working from this information, it translates the code to 8086 assembler code in an optimized way. The decision algorithm for each instruction type is given in a section of the manual to allow the user to see what happens in each situation.

Register mapping generally follows the correspondence shown in figure 4, with a loose relationship between the 8086 AX and the 8080 PSW; the exact relationship is determined from register usage at translate time.

Many run-time options are available to control the translation pro-