8080 port addressing with the C/80 compiler

PHILLIP L. EMERSON
Cleveland State University, Cleveland, Ohio

This paper deals with Intel 8080 port addressing with the C/80 compiler. Specifically, two assembly language routines designed for the C/80 are described.

A compiler for the C language has become available for computers using the intel 8080 instruction set,¹ an important step toward efficient laboratory research computing on a low budget. The compiler is the C/80.² Versions for the CP/M³ and Heath H89 HDOS⁴ operating systems are available, and the user has a choice of an absolute 8080 assembler supplied with C/80 or a relocatable assembler.⁵ The C/80 compiler translates C source code to assembly language. The assembler then translates to executable object code.

The C language, first developed at Bell Laboratories in connection with the UNIX⁶ operating system, is in the class of "structured" languages. It is a higher level language in the sense that programming is largely independent of the hardware and the software operating system on which it is implemented. Also, it includes a number of modern high-level features, such as modular subroutines, data structures of arbitrary form, recursive subroutine definition, dynamic memory allocation and deallocation, and self-referential definition of data structures. At the other extreme, it provides for many operations that traditionally have required programming at the assembly language level. It goes a long way toward eliminating the need for assembly language programming on the various machines for which it is implemented. Some of the primitive operations in C closely resemble PDP-11⁷ machine code operations, but C has been implemented on many machines other than the PDP-11 and with many operating systems other than UNIX (Kernighan & Ritchie, 1978). The C language, and the C/80 version for 8080 processors, can be recommended as being very suitable for the programming of laboratory microcomputers. The C language is fairly well standardized and documented (Kernighan & Ritchie, 1978), so there is little trouble in adapting utility routines for different machines and operating systems. However, a few of the examples given by Kernighan and Ritchie do not work exactly as written⁸ with C/80. Slight modifications prove workable, and these are usually easy to discover from the C/80 diagnostics.

The author is indebted to anonymous reviewers for comments on earlier versions of this paper. The author's mailing address is: Department of Psychology, Cleveland State University, Cleveland, OH 44115.

Two small assembly language routines for use specifically with C/80 are reported here. In standard implementations of C, it is almost never necessary to invoke assembly language code (Kernighan & Ritchie, 1978). However, C/80 provides that capability, and some of the C/80 library routines are written in assembly language rather than C itself, for purposes of optimization. For laboratory research computing, nonstandard input and output are a common necessity because nonstandard peripheral equipment is the rule, rather than the exception. A rather general way to do nonstandard input and output is by direct absolute port addressing. This is the same as using the IN (sometimes PIN) and OUT functions that are available in some microcomputer versions of BASIC. In C/80, these same functions can be implemented very simply, as shown in Figures 1 and 2. It is necessary to resort to assembly language for these routines in the C/80 implementation, because 8080 ports, unlike those of the PDP-11, are addressed by special input and output instructions that are not the same as memory

```
#asm
EXECUTI DW 0
RET
#endasm
portin(port)
int port ; {
#asm
    POP D          RETURN ADDRESS.
    POP H          ARGUMENT.
    MOV H,L        PORT NUMBER.
    MVI L,3330     "IN" OPCODE.
    SHLD EXECUTI   STORE INSTR.
    CALL EXECUTI   EXECUTE IT.
    MVI H,0        MOVE RESULT TO HL.
    MOV L,A
    PUSH H
    PUSH D          RESTORE STK. PTR.
#endasm
return ; }
```

Figure 1. C/80 function to read the contents of the port whose absolute address is the value of the integer variable *port*. Conventionally, C language code is in lowercase, while 8080 assembly code is usually in uppercase. In C/80, *int* or *char* argument values are pushed onto the operating system stack in reverse order, when a subroutine is called. Also, *int* and *char* function values are returned in the 8080 HL register pair. The calling program pops, but ignores, the argument values off the stack after a subroutine call; therefore, a called subroutine must return with the stack pointer as it was on entry and with the return address at the top of the stack.

```

#asm
EXECUTO DW 0
RET
#endasm
portout(port,byte)
int port,byte; {
#asm
    POP D          RETURN ADDRESS.
    POP B          2ND ARG.
    POP H          1ST ARG.
    MOV H,L        PORT NR.
    MVI L,323Q     "OUT" OPCODE.
    SHLD EXECUTO   STORE INSTR.
    MOV A,C        DATA BYTE.
    CALL EXECUTO   EXECUTE INSTR.
    PUSH H         RESTORE STK. PTR.
    PUSH B         RESTORE STK. PTR.
    PUSH D         RESTORE RETURN ADDRESS.
#endasm
return ; }

```

Figure 2. C/80 function to write the bit pattern of the integer variable *byte* to the port whose absolute address is the value of *port*. With some implementations of C, *portout()* would be declarable as *type-void*. C/80 does not recognize such a type.

```

char a[80];
main() {putstr("Testing port IO \n\n");
  while (1) repeat(); }
repeat() { int port;
  putstr("in or out? \n"); getstr(a);
  putstr("type port number \n");
  port = getint();
  if ((!cmpstr(a,"in")) || (!cmpstr(a,"IN")))
    { putint(portin(port)); putchar('\n');
    return ; }
  putstr("type byte, decimal form \n");
  portout(port,getint()); return ; }
/* some standard functions */
cmpstr(p,q) char *p,*q; { while(*p == *q)
  {if (*p++ == 0) return(0);
  ++q;} return(*p < *q ? -1:1); }
putstr(s) char *s; {while(*s) putchar(*s++); }
getstr(a) char *a; {
  while ((*a=getchar()) != '\n') ++a; *a=0; }
getint() {char c; int n;
  for (n=0; ((c=getchar()) >= '0') &&
    (c <= '9'); )
    n = 10*n + c - '0'; return(n); }
putint(n) int n; { int i;
  if (i = n/10) putint(i);
  putchar('0' + n % 10); }

```

Figure 3. Example C/80 main program calling the *portin()* and *portout()* functions to test the operations of interfaced peripheral devices.

```

rndint(n) int n;
{static int ptr;
  static int m[] = {17323,-30133,
    22293,01845,12893,-01773,01179,
    -14233,22103,01283,-17232,00678,
    28991,-02323,-30017,19234,31772};
  n += m[ptr];
  ptr = (++ptr) % 17;
  return (m[ptr] += n);
}

```

Figure 4. A C/80 pseudo random number generator, for a uniform distribution of integers between -32768 and 32767. The argument, *n*, is for seeding the random number sequence on the first call, with some nonzero integer. On subsequent calls, *n*, can be set equal to zero.

reference instructions. Figure 3, which shows a main program in C/80, illustrates the use of the port addressing routines.⁹ It is merely a test program to take numbers from the console and either put out or read in contents of specified ports.

The functions *putstr()*, *getstr()*, *cmpstr()*, *getint()*, and *putint()* below the comment line in Figure 3 are variations on fairly standard routines (Kernighan & Ritchie, 1978) that ordinarily would reside in a library file along with the functions *portin()* and *portout()* of Figures 1 and 2. However, these can be typed into the same file and compiled together as a single unit. The functions *getchar()* and *putchar()* are standard C library functions for single-character console input and output.

The researcher equipped with a microcomputer system of the 8080 family, the CP/M or HDOS operating system, the C/80 compiler, the C/80 MATHPAK, and a few small utility routines is well prepared to begin interfacing peripheral devices for a variety of experiments. The monetary cost is modest.¹⁰ Rarely is it necessary to invoke machine or assembly language programming beyond the two routines presented in Figures 1 and 2. Therefore, the additional cost in terms of programming effort should also be modest. Gone are the pioneering days of POKE-ing long 8080 machine language routines from BASIC DATA arrays, programming which required a great deal of effort.

One further very simple C/80 function is presented in Figure 4 which presents a pseudo random number generator.¹¹ It is appended here because the C/80 utilities that are provided with the compiler and MATHPAK do not include a random number generator, which is very important to many behavioral research projects. The subroutine in Figure 4 is based on a well-tested additive method of Green, Smith, and Klem (1959). This method generates a generalized Fibonacci sequence, and it is speedy because the main computations are just a few integer additions. Compared with the more common multiplicative congruential method (Hammersley & Handscomb, 1964), this additive method is better suited to simple processors, such as the 8080, that lack special hardware to do arithmetic beyond addition and subtraction. The range of the produced integers is from -32768 to 32767. Other ranges of integers, as well as pseudo random floating point numbers, can be obtained by simple operations. On the other hand, if only integers within this range are needed, the floating and long integer routines (occupying about 3K bytes) of the C/80 MATHPAK need not be included in an applications program unless they are needed for other purposes.

REFERENCES

- GREEN, B. F., JR., SMITH, J. E. K., & KLEM, L. (1959). Empirical tests of an additive random number generator. *Journal of the Association for Computing Machinery*, 6, 527-537.
- HAMMERSLEY, J. M., & HANDSCOMB, D. C. (1964). *Monte-Carlo methods*. London: Methuen.

KERNIGHAN, B. W., & RITCHIE, D. M. (1978). *The C programming language*. Englewood Cliffs, NJ: Prentice Hall.
 ZAKS, R. (1980). *The CP/M handbook with MP/M*. Berkeley, CA: Sybex.

NOTES

1. The Intel Corporation introduced the 8080 microprocessor, which quickly became one of the most used 8-bit processors. Intel has since introduced other upgraded 8-bit processors, but has maintained upward compatibility with the 8080 instruction set. Other manufacturers have also introduced upgraded processors that maintain the compatibility, and thus the 8080 instruction set has become a widely adopted standard.³
2. The C/80 compiler is available from Software Toolworks, 15233 Ventura Boulevard, Suite 1118, Sherman Oaks, CA 91403. A MATHPAK arithmetic package is available separately for floating point operations and long integers (longer than 16 bits). The MATHPAK includes all the common transcendental functions, such as the trigonometric, exponential, and logarithmic. There is no random number generator, but it is easy to write one in C/80 (e.g., see Figure 4).
3. The CP/M operating system is available from Digital Research, Inc., Box 579, Pacific Grove, CA 93950, and also is distributed by many retail computer outlets. This has been the most universal operating system for microcomputing systems with floppy disk storage. It runs on microcomputers using the Intel 8080 instruction set, including the 8085 and 8086, and the Zilog Z-80 and Z-8000. CP/M is also available on 6502-based computers such as the Apple and Pet, but requires the addition of an inexpensive 8080 or Z-80 processor board (Zaks, 1980).
4. Heath Company, Benton Harbor, MI 49022.
5. Microsoft MACRO-80.
6. UNIX is a trademark of Bell Laboratories.
7. PDP-11 is a trademark of the Digital Equipment Company.
8. The environment in which C/80 is implemented is quite restricted compared with those of most other implementations. It seems rather surprising—almost a breakthrough—that such a good facsimile of the language could be implemented for the 8-bit 8080 family of processors with small memory capacity. Some features of C are not implemented, and some others are implemented in restricted ways. The C/80 manual seems to be explicit on most of the exceptions.
9. The entities *port*, *byte*, and *portin* () could have been declared as

type-char (8-bit quantities), since the 8080 port addresses and port contents are all 8-bit quantities. Instead, they have been declared as *type-int* (16-bit quantities) here, for two reasons. First, the C/80 manual and Kernighan and Ritchie (1978) recommend *type-int* declarations for 8-bit numerical functions and arguments. Second, it is common on other machines (PDP-11) that port addresses are 16 bits in length, and that port contents range from 1 to 16 bits in length. To some extent, source programs that call *portin* () and *portout* () can be written in a portable way, if there is any reason to do so. Obviously, *portin* () and *portout* () would have to be rewritten for different systems. Also, some implementations of C define *type-int*, as well as *type-char*, differently (Kernighan & Ritchie, 1978).

10. The prices of microprocessor systems, components, accessories, and software have been on a decline for the past few years due to volume production. At present (1984), \$2,000 seems adequate for a usable system, including 64K bytes of memory, two disk drives, dot-matrix printer, video terminal, telephone modem, and extra serial and parallel interfaces. If such a system is not dedicated to the control of experiments, other software is readily available for use with the CP/M operating system, such as BASIC, word processors, relocatable programming systems with assembler and linker, FORTRAN, COBOL, spread sheets, Pascal, and statistical packages. These comments concerning the cost factor are not meant to suggest that such 8080-C/80 systems are recommended only for impoverished researchers. Cost effectiveness is a concern to principle investigators on even the most lavish of projects.

11. The question of portability to non-C/80 implementations of C can be raised, since *type-int* is assumed to refer to signed 16-bit quantities. But concern is not justified practically because: (1) many non-C/80 systems include a random number generator as a software utility, (2) the present *rndint* () routine would work properly, if there was any reason to use it, on PDP-11 implementations in which *type-int* referred to 16-bit quantities, and (3) many applications programs for 8080-based laboratory computer systems are necessarily nonportable anyway, because of variations in hardware configurations and research objectives. Portability of source programs, when it is achieved, is only one of the many advantages of programming in C. In some applications, other priorities can supersede.

(Manuscript received September 14, 1984;
 accepted for publication November 21, 1984.)