Using Micro Concurrent Pascal in RCA Development Systems

with the CDP1804P1 and CDM5332P1

By David C. Stanley
Micro Concurrent Pascal (mCP)* is designed for writing real-time computer control applications and is best suited for real-time executions that require multitasking operating systems. MCP allows the programmer to construct interrupt driven tasks that can share data, communicate information, and be synchronized for execution. RCA has developed a 2-chip set (CDP1804P1 and CDM5332P1) which contains a pseudo-code (p-code) interpreter and facilitates use of mCP in end use systems. This article describes mCP, the RCA chip set, and how to generate mCP code.

**MCP FEATURES**

A task is called a "process" in mCP. Processes are independent programs that run concurrently by sharing control of the microprocessor. Routines and the data shared between them are placed in a data structure called a "monitor". Processes access shared data only through monitors that enforce exclusive use of data (allow only one process at a time to use it). "Device monitors" allow processes to access shared devices.

**COMPILER and INTERPRETER**

The mCP compiler written by Enertec Inc., resides on Compuserve. The user's manual for mCP may be acquired from RCA Systems Marketing An mCP program may be generated with an RCA development system editor (CDS or MCDS) and up-loaded to Compuserve for compilation. The Pascal file may also be generated with

*Micro Concurrent Pascal and mCP are registered tradenames of Enertec, Inc., Lansdale, PA.*
Compuserve's FILGE (FILE GEnerator) editor and compiled. The mCP compiler outputs the p-code application program that is down-loaded to the CDP1804P microcomputer system and executed by the interpreter/kernel. This interpreter is divided into two sections.

The first section (core) resides in the 2K ROM of the CDP1804P and is independent of the second section. The core section of the interpreter permits execution of a restricted subset of mCP language (see table 1). The restrictions on the language are concurrency, set operations and SET data types, bit manipulation and string move subroutines, and REAL (floating-point) data types. This subset of mCP ("micro Pascal") is compatible with sequential Pascal defined in Jensen and Wirth's: PASCAL User Manual and Report. A memory map for the core interpreter system is shown in appendix A. A user generated branch/parameter table located on page 0 must contain address pointers for the start of the p-code program and boundaries of a contiguous RAM memory space for interpreter working storage (see appendix A). The 64 byte on-chip RAM of the CDP1804P can be used for this working storage only with the core interpreter. Immediately following the branch/parameter table should be a list of the built-in machine language subroutines for the CDP1804P (see appendix B). Users may write their own subroutines by extending the list up to a total of 128 routine addresses. Programmers must write their own interrupt routines or poll I/O ports for devices that do not generate interrupts. This is accomplished by using Pascal functions and procedures INN, OUT, PEEK, and POKE. Appendix C describes interrupt handling including CDP1804A Timer/counter interrupts.

The second section (extension) of the interpreter is designed to work with the core and extends support to the complete mCP language. Table 2 shows the additional language features of the extension ROM. A memory map
## TABLE 1

**CDP1804P1 MICRO CONCURRENT PASCAL**

**CORE LANGUAGE**

### WORD SYMBOLS

<table>
<thead>
<tr>
<th>AND</th>
<th>ARRAY</th>
<th>BEGIN</th>
<th>CASE</th>
<th>CONST</th>
<th>CYCLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIV</td>
<td>DO</td>
<td>DOWNTO</td>
<td>ELSE</td>
<td>END</td>
<td>ENTRY</td>
</tr>
<tr>
<td>EXTERNAL</td>
<td>FOR</td>
<td>FUNCTION</td>
<td>IF</td>
<td>MOD</td>
<td>NOT</td>
</tr>
<tr>
<td>OF</td>
<td>OR</td>
<td>PROCEDURE</td>
<td>RECORD</td>
<td>REPEAT</td>
<td>THEN</td>
</tr>
<tr>
<td>TO</td>
<td>TYPE</td>
<td>UNIV</td>
<td>UNTIL</td>
<td>VAR</td>
<td>WHILE</td>
</tr>
<tr>
<td>WITH</td>
<td>XOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### STANDARD FUNCTIONS

<table>
<thead>
<tr>
<th>ABS</th>
<th>ADDR</th>
<th>ADR</th>
<th>CHR</th>
<th>INN</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAXINT</td>
<td>ORD</td>
<td>PEEK</td>
<td>PRED</td>
<td>SUCCE</td>
</tr>
</tbody>
</table>

### STANDARD PROCEDURES

<table>
<thead>
<tr>
<th>DEC</th>
<th>INC</th>
<th>OUT</th>
<th>POKE</th>
</tr>
</thead>
</table>

### STANDARD DATA TYPES

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>BOOLEAN</th>
<th>CHAR</th>
<th>INTEGER</th>
</tr>
</thead>
</table>
**TABLE 2**

CDP1804P1MICRO CONCURRENT PASCAL

EXTENSION LANGUAGE (4K ROM)

<table>
<thead>
<tr>
<th>WORD SYMBOLS</th>
<th>STANDARD FUNCTIONS</th>
<th>STANDARD PROCEDURES</th>
<th>STANDARD TYPES</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASS MONITOR</td>
<td>DEVICE_MON</td>
<td>DOIO</td>
<td>CONTINUE</td>
</tr>
<tr>
<td>PROCESS</td>
<td>IN</td>
<td>DELAY</td>
<td>QUEUE</td>
</tr>
<tr>
<td>SET</td>
<td>STR_STOP</td>
<td>DIDdle</td>
<td>REAL</td>
</tr>
<tr>
<td>INIT</td>
<td>TRUNC</td>
<td>INITQUEUE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>WORD</td>
<td>STR_COUNT</td>
<td></td>
</tr>
</tbody>
</table>
of the extended interpreter system is shown in appendix A. The extension ROM replaces the branch/parameter table used by the core interpreter and sets up its own table identical in layout for linkage with the core. This table contains a page pointer to an address table for p-code subroutines. The user must generate a branch/parameter table starting at page 10 hexadecimal in same manner as with the core (see appendix A). Figure 1 shows a typical CDP1804P1 mCP system with the CDM5332P1 extension ROM.

CDS
To up-load and down-load Pascal programs from Compuserve, the CDS Micro NET Exective program is needed. This program is available in the RCA User Group and may be booted to a CDS system disk using the boot load program described in the Compuserve Manual. A modem is required. The CDS IV (18S008) connects to the modem through the connector on the back of the system. The CDS III (18S007) needs an 18S641 UART board and a modified 18S516 cable to connect to the modem. This modification consists of reversing pins 2 and 3 of the 10 pin connector on the 641 UART board. The modem is set to full duplex and originate mode. The 641 UART board is placed in an I/O slot with the N lines wired to it. Appendix D shows an example of the up-load and down-load sequence.

To use the CDP1804P1 in the CDS IV and CDS III, the CDP18S605 (CDS IV) and the CDP18S102V1 (CDS III) CPU boards must be modified by inverting the WAIT and CLEAR lines and switching them (ie. WAIT to pin 3 and CLEAR to pin 2 of the CPU) and disconnecting pin 16 (ME/EMS) and letting it float. The CDM5332P1 extension interpreter may be copied onto disk using an altered CDP18S480 PROM programmer board. This alteration consists of adding switch to pin 21 of socket XU3 (see figure 2). The 5332P1 can then be placed in socket XU3 and read as two 2716's. The first 2K of the interpreter can be
Figure 1-1804P1 mCP system
Fig 2 - Programming logic of PROM programmer module
copied into locations 0800 - 1000 with the switch at Vss. The second 2K is copied into locations 1000 - 17FF with the switch at Vdd (collector of transistor Q1). The PROM board may also be used as normal with the switch in this position. Once the interpreter is loaded to disk (using save command), it may be loaded into RAM at location 0000 for program development. Once development is complete, the CDM5332P1 can be used for prototype hardware.

**MCDS**

The 18S601 CPU board must be modified for the CDP1804P1 (see appendix E). The following links must be altered:

1. Remove links 4:13 and 2:15 of LK36
2. Add links 3:14 and 1:16 of LK36
3. Remove link A:B of LK37

The extension ROM must be located at 0000 - OFFF and RAM must immediately follow for the branch/parameter table (see appendix F). The following must be done:

1. Add link 6:11 and 1:16 of LK11
2. Add link 5:12 and 1:16 of LK10
3. Remove link 1:18 of LK4
4. Disconnect pin 2 of U28 from All-P and connect to +5V
5. Connect All-P to pin 21 of U24

With this set-up, the editor and assembler cannot be used because RAM is needed at location 0000 for their operation. The FILGE editor on Compuserve may be used or the MCDS editor may be used by replacing the hardwire links
with DIP switches at LK10 and LK11. Then when using the editor, switch RAM to location 0000 (LK11 - 5:12) and the extension ROM to location 1000 (LK10 - 6:11). When ready to interpret a p-code program, switch them back (LK11 - 6:11; LK10 - 5:12). Another option is to purchase a second CDP18S601 board and modify it for use with mCP. The Pascal program can be edited with the MCDS 601 board and replaced with the mCP 601 board for p-code interpretation. The extension ROM is the 4K CDP5332P1 and is placed in socket U24 of the 18S601 board.

The MCDS Micro NET Executive is needed to up-load and down-load Pascal programs and is also available on Compuserve. The program may be booted into the MCDS system and burned into a 2716 EPROM which is placed in socket U13 (location EO00) of the CDP18S652 Tape I/O and Memory board. The ROM in socket U3 must be removed because it is mapped at CO00 (since this ROM is part of BASIC 3, BASIC cannot be used). The MCDS communicates with the modem with the same hardware as the CDS III (18S007). The CDP18S641 UART board plugs right into any open socket. An example of the up-load and down-load sequence is shown in appendix G.

Summary

Real-time mCP application programs can be developed on MCDS and CDS development systems. A version of sequential Pascal (micro Pascal) is also available when using only the CDP1804P1. Both mCP and micro Pascal can be compiled using the mCP compiler residing on Compuserve. These programs are up-loaded into Compuserve for compilation and down-loaded into the development system for execution. With minimal changes to existing development systems, the CDP1804P1 and CDM5332P1 can be developed and then used in the final application system.
FOOTNOTES


   (second edition), Springer-Verlag, New York, 1974

3. Hardware Reference for the CDP1804A data sheet,
   File 1371

4. D. Block, Programming 2732 PROM's with the CDP18S480
   PROM Programmer, ICAN-6847
P-CODE STORAGE REQUIREMENTS

The length of the P-code is the sum of the code length and the constant length found in the program listing file output by the mCP compiler. The constant storage area follows the program code contiguously.

RAM WORKING STORAGE REQUIREMENTS

The amount of RAM actually used by a program is computed during interpreter initialization. The address of the next free byte of RAM is located in the first word of RAM storage.

The amount of RAM used by a program is calculated:

24 + stack length + variable size

The stack length is the first parameter on the INIT_PROCESS P-code. The variable size is the second parameter. The parameter length is the third parameter (it is zero for micro Pascal). The INIT_PROCESS P-code is found at the end of a "long" listing file output by the compiler.

Micro Pascal uses three bytes of the R2 system stack. Micro Concurrent Pascal uses three bytes of the R2 system stack plus 13 bytes when floating point subroutines are used.

In a mCP program there may be several INIT_PROCESS p-codes. These must be summed to calculate the RAM storage requirements. Also the DOIO table size and stack margin parameter must be included.
CDP1804P1MICRO CONCURRENT PASCAL
MEMORY MAP
(CORE INTERPRETER ONLY)

0000

<table>
<thead>
<tr>
<th>BRANCH/ PARAMETER TABLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>-------------------------</td>
</tr>
<tr>
<td>USER DEFINED MACHINE LANGUAGE SUBROUTINES (OPTIONAL)</td>
</tr>
<tr>
<td>-------------------------</td>
</tr>
<tr>
<td>APPLICATION PROGRAM P-CODE</td>
</tr>
</tbody>
</table>

XXXX

<table>
<thead>
<tr>
<th>RAM FOR STACK WORKING STORAGE</th>
</tr>
</thead>
</table>

C000

| 1804 PASCAL INTERPRETER ROM (ON-CHIP) |

C7FF


C800

<table>
<thead>
<tr>
<th>ON-CHIP RAM (NOT USED)</th>
</tr>
</thead>
</table>
CDP1804P1MICRO CONCURRENT PASCAL MEMORY MAP WITH EXTENSION ROM

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>MCP EXTENSION ROM</td>
</tr>
<tr>
<td>OFFF</td>
<td>BRANCH/PARAMETER TABLE</td>
</tr>
<tr>
<td>1000</td>
<td>USER DEFINED MACHINE LANGUAGE SUBROUTINES (OPTIONAL)</td>
</tr>
<tr>
<td></td>
<td>APPLICATION PROGRAM P-CODE</td>
</tr>
<tr>
<td>XXXX</td>
<td>RAM FOR STACK WORKING STORAGE</td>
</tr>
<tr>
<td>C000</td>
<td>1804 PASCAL INTERPRETER ROM (ON-CHIP)</td>
</tr>
<tr>
<td>C7FF</td>
<td>ON-CHIP RAM (NOT USED)</td>
</tr>
</tbody>
</table>
APPENDIX A CONT.

RAM STORAGE LAYOUT

<table>
<thead>
<tr>
<th>RAM BOTTOM + 0</th>
<th>ADDRESS OF NEXT FREE BYTE OF RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1804 COUNTDOWN TIMER</td>
</tr>
<tr>
<td>4</td>
<td>CURRENT LINE NUMBER</td>
</tr>
<tr>
<td>6</td>
<td>ERROR CODE</td>
</tr>
<tr>
<td>8</td>
<td>P-CODE POINTER OF LAST ERROR</td>
</tr>
<tr>
<td>10</td>
<td>RESERVED FOR POWER FAIL</td>
</tr>
<tr>
<td>12</td>
<td>RESERVED</td>
</tr>
<tr>
<td>14</td>
<td>RESERVED</td>
</tr>
<tr>
<td>16</td>
<td>RESERVED</td>
</tr>
<tr>
<td>18</td>
<td>PASCAL DATA STACK</td>
</tr>
<tr>
<td></td>
<td>PASCAL VARIABLES + 6</td>
</tr>
<tr>
<td></td>
<td>FREE STORAGE</td>
</tr>
<tr>
<td>RAM TOP</td>
<td>R(2)</td>
</tr>
</tbody>
</table>

NEXT FREE BYTE OF RAM

SYSTEM STACK
FOR INTERRUPTS, I/O, ASSEMBLY
LANGUAGE SUBROUTINES, ETC.
The interpreter clears memory as part of its initialization. To prevent memory clear the initialization sequence may be altered.

The following patch will prevent memory clear in micro Pascal:

<table>
<thead>
<tr>
<th>Location</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0003</td>
<td>0060 address of patch (can be any suitable location)</td>
</tr>
<tr>
<td>0060</td>
<td>68CD000F ED 68676865 6866686A</td>
</tr>
<tr>
<td></td>
<td>E2 68B5E5 68 B725 COC01B</td>
</tr>
</tbody>
</table>

The following patch will prevent memory clear in micro Concurrent Pascal:

<table>
<thead>
<tr>
<th>Location</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1003</td>
<td>1060 address of patch (can be any suitable location)</td>
</tr>
<tr>
<td>1060</td>
<td>ED 6867 6865 6866686A</td>
</tr>
<tr>
<td></td>
<td>E2 68B5 E5 68B7 25 COC01B</td>
</tr>
</tbody>
</table>
USING 1804 MICRO PASCAL

In order to use 1804 micro Pascal, the programmer must set up a branch table and a parameter area starting at location 0000 (hex). Programmer defined external machine language subroutines may follow this area or the number of built-in subroutines may be extended. Actual Pascal P-code follows the built-in subroutine address table.

Page 00 branch table and parameter area:

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>DATA</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>7100</td>
<td>Disable interrupts</td>
</tr>
<tr>
<td>0002</td>
<td>C0C000</td>
<td>Long branch to pre-initialization</td>
</tr>
<tr>
<td>0005</td>
<td>EEDF00</td>
<td>Post-initialization (could be long branch)</td>
</tr>
<tr>
<td>0008</td>
<td>C0C079</td>
<td>Long branch to software error continuation</td>
</tr>
<tr>
<td>000B</td>
<td>C0C68B</td>
<td>Long branch to external interrupt</td>
</tr>
<tr>
<td>000E</td>
<td>C1</td>
<td>P-code subrouting adress table page</td>
</tr>
<tr>
<td>000F</td>
<td>----</td>
<td>Bottom of RAM address</td>
</tr>
<tr>
<td>0011</td>
<td>----</td>
<td>Top of RAM address</td>
</tr>
<tr>
<td>0013</td>
<td>----</td>
<td>Size of RAM (RAM Top-RAM bottom +1)</td>
</tr>
<tr>
<td>0015</td>
<td>----</td>
<td>P-code starting address</td>
</tr>
<tr>
<td>0017</td>
<td>0000</td>
<td>Reserved for micro Concurrent Pascal extension program</td>
</tr>
</tbody>
</table>

The programmer defines the location of RAM working storage and the starting P-code address. All other fields should remain as specified above unless special requirements must be met. The branch table is used by the optional mCP ROM to link with the 1804 micro Pascal interpreter.

The addresses of bottom and top of RAM may be adjusted so that memory space is available for working storage by external or built-in machine language and interrupt subroutines. Only the area defined between RAM bottom and ROM top will be cleared to zero.

The application program P-codes output by the cross-compiler are re-locatable anywhere in memory. The starting P-code base address is specified in the parameter area, location 0015 hexadecimal.
APPENDIX A CONT

Page 00 must also contain the built-in machine language subroutine address table:

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>SUBROUTINE LOCATION</th>
<th>SUBROUTINE NAME</th>
<th>SELECTOR NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>0019</td>
<td>C558</td>
<td>Test external flag 1 (EF1)</td>
<td>1</td>
</tr>
<tr>
<td>001B</td>
<td>C55C</td>
<td>Test external flag 2 (EF2)</td>
<td>2</td>
</tr>
<tr>
<td>001D</td>
<td>C560</td>
<td>Test external flag 3 (EF3)</td>
<td>3</td>
</tr>
<tr>
<td>001F</td>
<td>C564</td>
<td>Test external flag 4 (EF4)</td>
<td>4</td>
</tr>
<tr>
<td>0021</td>
<td>C600</td>
<td>Set DMA register (RO) to address</td>
<td>5</td>
</tr>
<tr>
<td>0023</td>
<td>C605</td>
<td>Get DMA register (RO) address</td>
<td>6</td>
</tr>
<tr>
<td>0025</td>
<td>C277</td>
<td>Set INTERRUPT register (R1) address</td>
<td>7</td>
</tr>
<tr>
<td>0027</td>
<td>C60A</td>
<td>Set Q flag on or off</td>
<td>8</td>
</tr>
<tr>
<td>0029</td>
<td>C610</td>
<td>Test Q flag</td>
<td>9</td>
</tr>
<tr>
<td>002B</td>
<td>C20E</td>
<td>Load counter</td>
<td>10</td>
</tr>
<tr>
<td>002D</td>
<td>C212</td>
<td>Get counter</td>
<td>11</td>
</tr>
<tr>
<td>002F</td>
<td>C22E</td>
<td>Stop counter</td>
<td>12</td>
</tr>
<tr>
<td>0031</td>
<td>C231</td>
<td>Decrement Counter</td>
<td>13</td>
</tr>
<tr>
<td>0033</td>
<td>C234</td>
<td>Start timer</td>
<td>14</td>
</tr>
<tr>
<td>0035</td>
<td>C237</td>
<td>Start counter mode 1</td>
<td>15</td>
</tr>
<tr>
<td>0037</td>
<td>C23A</td>
<td>Start counter mode 2</td>
<td>16</td>
</tr>
<tr>
<td>0039</td>
<td>C257</td>
<td>Start pulse mode 1</td>
<td>17</td>
</tr>
<tr>
<td>003B</td>
<td>C25A</td>
<td>Start pulse mode 2</td>
<td>18</td>
</tr>
<tr>
<td>003D</td>
<td>C2B5</td>
<td>Enable toggle Q flag</td>
<td>19</td>
</tr>
<tr>
<td>003F</td>
<td>C2B8</td>
<td>External interrupt enable</td>
<td>20</td>
</tr>
<tr>
<td>0041</td>
<td>C2BB</td>
<td>External interrupt disable</td>
<td>21</td>
</tr>
<tr>
<td>0043</td>
<td>C0DB</td>
<td>Counter interrupt enable</td>
<td>22</td>
</tr>
<tr>
<td>0045</td>
<td>C0DE</td>
<td>Counter interrupt disable</td>
<td>23</td>
</tr>
<tr>
<td>0047</td>
<td>C2CE</td>
<td>Set RAM timer word</td>
<td>24</td>
</tr>
<tr>
<td>0049</td>
<td>C0E7</td>
<td>Get RAM timer word</td>
<td>25</td>
</tr>
<tr>
<td>004B</td>
<td>C0E1</td>
<td>Enable interrupts</td>
<td>26</td>
</tr>
<tr>
<td>004D</td>
<td>C0E4</td>
<td>Disable interrupts</td>
<td>27</td>
</tr>
<tr>
<td>004F</td>
<td>C275</td>
<td>Wait for interrupt</td>
<td>28</td>
</tr>
</tbody>
</table>

The built-in machine language subroutines reside in the 1804 ROM.
Additional built-in subroutines may be coded by extending the list of sub-
routine addresses. Up to a total of 128 built-in subroutine addresses may exist. If a particular built-in subroutine entry is not used, it may be replaced by another subroutine address.
USING MCP EXTENSION ROM

With the micro Concurrent Pascal extension ROM installed, the programmer must set up a branch table and parameter area starting at location 1000 (hex). Programmer defined external machine language subroutines may follow this area or the number of built-in subroutines may be extended. Actual Pascal P-code follows the built-in subroutine address table. The parameters in page 10 (hex) have the same relative position as parameters specified in page 00 for micro Pascal.

Page 10 (hexadecimal) branch table and parameter area:

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>DATA</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>7100</td>
<td>Disable interrupts</td>
</tr>
<tr>
<td>1002</td>
<td>COCO04</td>
<td>Long branch to pre-initialization</td>
</tr>
<tr>
<td>1005</td>
<td>EEDF00</td>
<td>Post-initialization (could be long branch)</td>
</tr>
<tr>
<td>1008</td>
<td>CO0043</td>
<td>Long branch to software error continuation</td>
</tr>
<tr>
<td>100B</td>
<td>CO0304</td>
<td>Long branch to interrupt subroutine</td>
</tr>
<tr>
<td>100E</td>
<td>01</td>
<td>P-code subroutine address table page</td>
</tr>
<tr>
<td>100F</td>
<td>------</td>
<td>Bottom of RAM address</td>
</tr>
<tr>
<td>1011</td>
<td>------</td>
<td>Top of RAM address</td>
</tr>
<tr>
<td>1013</td>
<td>------</td>
<td>Size of RAM (RAM top - RAM bottom +1)</td>
</tr>
<tr>
<td>1015</td>
<td>------</td>
<td>P-code starting address</td>
</tr>
<tr>
<td>1017</td>
<td>0040</td>
<td>DOIO size and stack margin</td>
</tr>
</tbody>
</table>

The programmer defines the location of RAM working storage and the starting P-code address. All other fields should remain as specified above unless special requirements must be met. Adding initialization code, changing the P-code branch table, etc., can be made by altering the branch and parameter table entries.

The addresses of bottom and top of RAM may be adjusted so that memory space is available for working storage by external or built-in machine language and interrupt subroutines. Only the area defined between RAM bottom and top will be cleared to zero.
APPENDIX A CONT.

Page 10 must also contain the built-in machine language subroutine address table:

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>SUBROUTINE LOCATION</th>
<th>SUBROUTINE NAME</th>
<th>SELECTOR NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1019</td>
<td>C55B</td>
<td>Test external flag 1 (EF1)</td>
<td>1</td>
</tr>
<tr>
<td>101B</td>
<td>C55C</td>
<td>Test external flag 2 (EF2)</td>
<td>2</td>
</tr>
<tr>
<td>101D</td>
<td>C560</td>
<td>Test external flag 3 (EF3)</td>
<td>3</td>
</tr>
<tr>
<td>101F</td>
<td>C564</td>
<td>Test external flag 4 (EF4)</td>
<td>4</td>
</tr>
<tr>
<td>1021</td>
<td>C600</td>
<td>Set DMA register R(0) to address</td>
<td>5</td>
</tr>
<tr>
<td>1023</td>
<td>C605</td>
<td>Get DMA register R(0) address</td>
<td>6</td>
</tr>
<tr>
<td>1025</td>
<td>022E</td>
<td>Change address of interrupt table</td>
<td>7</td>
</tr>
<tr>
<td>1027</td>
<td>C60A</td>
<td>Set Q flag on or off</td>
<td>8</td>
</tr>
<tr>
<td>1029</td>
<td>C610</td>
<td>Test Q flag</td>
<td>9</td>
</tr>
<tr>
<td>102B</td>
<td>C20E</td>
<td>Load counter</td>
<td>10</td>
</tr>
<tr>
<td>102D</td>
<td>C212</td>
<td>Get counter</td>
<td>11</td>
</tr>
<tr>
<td>102F</td>
<td>C22E</td>
<td>Stop counter</td>
<td>12</td>
</tr>
<tr>
<td>1031</td>
<td>C231</td>
<td>Decrement counter</td>
<td>13</td>
</tr>
<tr>
<td>1033</td>
<td>C234</td>
<td>Start timer</td>
<td>14</td>
</tr>
<tr>
<td>1035</td>
<td>C237</td>
<td>Start counter mode 1</td>
<td>15</td>
</tr>
<tr>
<td>1037</td>
<td>C23A</td>
<td>Start counter mode 2</td>
<td>16</td>
</tr>
<tr>
<td>1039</td>
<td>C257</td>
<td>Start pulse mode 1</td>
<td>17</td>
</tr>
<tr>
<td>103B</td>
<td>C25A</td>
<td>Start pulse mode 2</td>
<td>18</td>
</tr>
<tr>
<td>103D</td>
<td>C2B5</td>
<td>Enable toggle Q flag</td>
<td>19</td>
</tr>
<tr>
<td>103F</td>
<td>C2B8</td>
<td>External interrupt enable</td>
<td>20</td>
</tr>
<tr>
<td>1041</td>
<td>C2BB</td>
<td>External interrupt disable</td>
<td>21</td>
</tr>
<tr>
<td>1043</td>
<td>CODEB</td>
<td>Counter interrupt enable</td>
<td>22</td>
</tr>
<tr>
<td>1045</td>
<td>CODE</td>
<td>Counter interrupt disable</td>
<td>23</td>
</tr>
<tr>
<td>1047</td>
<td>C2CE</td>
<td>Set RAM timer word</td>
<td>24</td>
</tr>
<tr>
<td>1049</td>
<td>C0E7</td>
<td>Get RAM timer word</td>
<td>25</td>
</tr>
</tbody>
</table>

Note: 1. Selector number 7 is different from the micro Pascal built-in subroutine.

2. The last three built-in subroutines in micro Pascal are omitted here because they do not apply to micro Concurrent Pascal.
APPENDIX A CONT

DEBUGGING USING LINE DIRECTIVE

For debugging using the LINE directive, the line number of the statement being executed can be found at RAM bottom + 4 (word). It may also be found at the address found in register 11, the LOCAL variable pointer. The line number is saved on the stack during a procedure or function call. However, it is not restored to the LINE variable in RAM (RAM bottom + 4) by micro Pascal. Micro Concurrent Pascal does restore the LINE number.

DEBUGGING INFORMATION

When a software error occurs the error code will be stored at RAM bottom + 7 (byte) and the P-code program counter (QPTR) will be stored at RAM bottom + 8 (word).

APPENDIX B

BUILT-IN ASSEMBLY LANGUAGE ROUTINES

Built-in assembly language routines are provided for common machine oriented functions. These include subroutines for the counter/timer and special instructions.

All built-in external routine declaration examples given below assume the following type declarations:

\[
\begin{align*}
\text{TYPE} & \quad \text{BUILT\_IN\_SUBR} = 1\ldots28; \\
\text{INT\_BYTE} & \quad = 0\ldots255;
\end{align*}
\]
1. TEST_EF1 (1)

Assembly language function tests external flag 1 (EF1) and returns either 0 or 1 (boolean).
Routine declaration:

FUNCTION TEST_EF1 (SELECTOR: BUILT_IN_SUBR): BOOLEAN;
EXTERNAL 'TESTEF1';

2. TEST_EF2 (2)

Assembly language function tests external flag 2 (EF2) and returns either 0 or 1 (boolean).
Routine declaration:

FUNCTION TEST_EF2 (SELECTOR: BUILT_IN_SUBR): BOOLEAN;
EXTERNAL 'TESTEF2';

3. TEST_EF3 (3)

Assembly language function tests external flag 3 (EF3) and returns either 0 or 1 (boolean).
Routine Declaration:

FUNCTION TEST_EF3 (SELECTOR: BUILT_IN_SUBR): BOOLEAN;
EXTERNAL 'TESTEF3';

4. TEST_EF4 (4)

Assembly language function tests external flag 4 (EF4) and returns with 0 or 1 (boolean)
Routine delcaration:

FUNCTION TEST_EF4 (SELECTOR: BUILT_IN_SUBR): BOOLEAN;
EXTERNAL 'TESTEF4';
5. DMA_SET (address to set RO, 5)

Assembly language procedure sets the DMA register R(0) to the address given by the first parameter.
Routine declaration:

```
PROCEDURE DMA_SET (RO_ADDRESS: ADDRESS; SELECTOR: BUILT_IN_SUBR);
EXTERNAL 'DMA SET';
```

6. DMA_ASK (6)

Assembly language function returns the address value of the DMA register R(0).
Routine declaration:

```
FUNCTION DMA_ASK (SELECTOR: BUILT_IN_SUBR): ADDRESS;
EXTERNAL DMAASK';
```

7A. INTERRUPT (interrupt subroutine address to set R1, 7)

Micro Pascal:
Assembly language procedure sets the interrupt register R(1) to the address of the interrupt subroutine given by the first parameter.
Routine declaration:

```
PROCEDURE INTERRUPT (interrupt_ADR: ADDRESS; SELECTOR: BUILT_IN_SUBR);
EXTERNAL 'INTCH';
```

7B. SWITCH_INTTBL (address of interrupt table, 7)

Micro Concurrent Pascal:
Assembly language procedure changes the address of the interrupt table. Procedure alters RAM bottom + 16 (word) with new interrupt table addresses.

```
PROCEDURE SWITCH_INTTBL (NEW_INTTBL_ADR: ADDRESS; SELECTOR:
BUILT_IN_SUBR); EXTERNAL 'INTCH';
```
3. SET_Q_FLAG (Q flag value, 8)

Assembly language procedure sets the Q flag to either 0 or 1 given by the first parameter.
Routine declaration:

PROCEDURE SET_Q_FLAG (Q_FLAG: BOOLEAN; SELECTOR: BUILT_IN_SUBR); EXTERNAL 'SETQ';

9. GET_Q_FLAG (9)

Assembly language function tests the Q flag and returns either 0 or 1 (boolean).
Routine declaration:

FUNCTION GET_Q_FLAG (SELECTOR: BUILT_IN_SUBR): BOOLEAN;
EXTERNAL 'GETQ';

10. LOAD_COUNTER (count value, 10)

Assembly language procedure sets counter to the counter value given by the first parameter. The value ranges from 0 to 255. The procedure executes the LDC instruction.
Routine declaration:

PROCEDURE LOAD_COUNTER (COUNT: INT_BYTE; SELECTOR: BUILT_IN_SUBR); EXTERNAL 'LDC';

11. GET_COUNTER (11)

Assembly language function returns the value of the counter. This function executes the GEC instruction.
Routine declaration:

FUNCTION GET_COUNTER (SELECTOR: BUILT_IN_SUBR): INTEGER;
EXTERNAL 'GEC';
12. **STOP COUNTER (0, 12)**

Assembly language procedure stop the counter. The first parameter is a dummy parameter. This procedure executes the STPC instruction.

Routine declaration:

```plaintext
PROCEDURE STOP COUNTER (DUMMY: INTEGER; SELECTOR: BUILT_IN_SUBR); EXTERNAL 'STPC';
```

13. **DECREMENT COUNTER (0, 13)**

Assembly language procedure decrements the counter by 1. The first parameter is a dummy parameter. This procedure executes the DTC instruction.

Routine declaration:

```plaintext
PROCEDURE DECREMENT COUNTER (DUMMY: INTEGER; SELECTOR: BUILT_IN_SUBR); EXTERNAL 'DTC';
```

14. **START TIMER (0, 14)**

Assembly language procedure sets the timer mode and starts the timer. The first parameter is a dummy parameter. This procedure executes the STM instruction.

Routine declaration:

```plaintext
PROCEDURE START TIMER (DUMMY: INTEGER; SELECTOR: BUILT_IN_SUBR); EXTERNAL 'STM';
```

15. **START COUNTER MODE (0, 15)**

Assembly language procedure sets counter mode 1 and starts the counter. The first parameter is a dummy parameter. This procedure executed the SC11 instruction.

Routine declaration:

```plaintext
PROCEDURE START COUNTER MODE (DUMMY: INTEGER; SELECTOR: BUILT_IN_SUBR); EXTERNAL 'SC11';
```
16. START_COUNTER_MODE2 (0, 16)

Assembly language procedure sets counter mode 2 and starts
the counter. The first parameter is a dummy parameter. This
procedure executes the SCM2 instruction.
Routine declaration:

    PROCEDURE START_COUNTER_MODE2 (DUMMY: INTEGER; SELECTOR:
        BUILT_IN_SUBR); EXTERNAL 'SCM2';

17. START_PULSE_MODE1 (0, 17)

Assembly language procedure sets pulse mode 1 and starts
counter. The first parameter is a dummy parameter. This
procedure executed SPM instruction.
Routine declaration:

    PROCEDURE START_PULSE_MODE1 (DUMMY: INTEGER; SELECTOR:
        BUILT_IN_SUBR); EXTERNAL 'SPM1';

18. START_PULSE_MODE2 (0, 18)

Assembly language procedure sets pulse mode 2 and starts
counter. The first parameter is a dummy parameter. This
procedure executes the SPM2 instruction.
Routine declaration:

    PROCEDURE START_PULSE_MODE2 (DUMMY: INTEGER; SELECTOR:
        BUILT_IN_SUBR); EXTERNAL 'SPM2';

19. ENABLE_TOGGLE_Q (0, 19)

Assembly language procedure sets the counter to toggle Q
whenever the counter decrements from 01 to its next value. The
first parameter is a dummy parameter. This procedure executes
the ETQ instruction.
Routine declaration:

    PROCEDURE ENABLE_TOGGLE_Q (DUMMY: INTEGER; SELECTOR:
        BUILT_IN_SUBR); EXTERNAL 'ETQ';
20. EXT_INTERRUPT_ENABLE (0, 20)

Assembly language procedure enables external interrupts. The first parameter is a dummy parameter. This procedure executes the XIE instruction.
Routine declaration:

PROCEDURE EXT_INTERRUPT_ENABLE (DUMMY: INTEGER; SELECTOR: BUILT_IN_SUBR); EXTERNAL 'XIE';

21. EXT_INTERRUPT_DISABLE (0, 21)

Assembly language procedure disables external interrupts. The first parameter is a dummy parameter. This procedure executes the XID instruction.
Routine declaration:

PROCEDURE EXT_INTERRUPT_DISABLE (DUMMY: INTEGER; SELECTOR: BUILT_IN_SUBR); EXTERNAL 'XID';

22. CNT_INTERRUPT_ENABLE (0, 22)

Assembly language procedure enables counter interrupts. The first parameter is a dummy parameter. This procedure executes the CIE instruction.
Routine declaration:

PROCEDURE CNT_INTERRUPT_ENABLE (DUMMY: INTEGER; SELECTOR: BUILT_IN_SUBR); EXTERNAL 'CIE';

23. CNT_INTERRUPT_DISABLE (0, 23)

Assembly language procedure disables counter interrupts. The first parameter is a dummy parameter. This procedure executes the CID instruction.
Routine declaration:

PROCEDURE CNT_INTERRUPT_DISABLE (DUMMY: INTEGER; SELECTOR: BUILT_IN_SUBR); EXTERNAL 'CID';
24. SET_TIME (time value, 24)

Assembly language procedure sets the timer word to the time value given by the first parameter. The timer word is decremented once whenever the timer/counter interrupt subroutine is executed until the timer word reaches zero.

Routine declaration:
```
PROCEDURE SET_TIME (TIME: INTEGER; SELECTOR: BUILT_IN_SUBR);
EXTERNAL 'SETIME';
```

25. GET_TIME (25)

Assembly language function returns the current value of the timer word.

Routine declaration:
```
FUNCTION GET_TIME (SELECTOR: BUILT_IN_SUBR): INTEGER;
EXTERNAL 'GETIME';
```

26. ENABLE_INTERRUPTS (0, 26)

Assembly language procedure enables interrupts. The first parameter is a dummy parameter. This procedure executes the RET instruction.

Routine declaration:
```
PROCEDURE ENABLE_INTERRUPTS (DUMMY: INTEGER; SELECTOR:
  BUILT_IN_SUBR); EXTERNAL 'ENB';
```

27. DISABLE_INTERRUPTS (0, 27)

Assembly language procedure disables interrupts. The first parameter is a dummy parameter. This procedure executes the DIS instruction.

Routine declaration:
```
PROCEDURE DISABLE_INTERRUPTS (DUMMY: INTEGER; SELECTOR:
  BUILT_IN_SUBR); EXTERNAL 'DIS';
```
28. IDLE (0, 28)

Assembly language procedure waits for interrupts. The first parameter is a dummy parameter. This procedure executes the IDL instruction.

Routine declaration:

PROCEDURE IDLE (DUMMY: INTEGER; SELECTOR: BUILT_IN_SUBR);
    EXTERNAL 'IDL';
HANDLING INTERRUPTS

Interrupts can be enabled or disabled at any time at the option of the programmer. Interrupts do not affect P-code implementation. Interrupts are not enabled/disabled between P-code execution. The interrupt routine must save and restore the D register and DF flag on the interrupt stack (R2), along with any registers used.

Because concurrency is omitted from the 1804 ROM, external interrupts must be handled by the programmer. The ROM contains a built-in interrupt subroutine for processing counter interrupts.

The built-in counter interrupt subroutine decrements a timer word in RAM if it is non-zero. Two built-in assembly language subroutines let the programmer read and write this timer word.

The interrupt subroutine branches to a fixed location in the user P-code ROM. This linkage is provided for handling external hardware interrupts. Hardware interrupts must be latched. External interrupts have higher priority than counter interrupts.
USER DEFINED INTERRUPT SUBROUTINE

A User interrupt handling subroutine may replace the built-in counter/timer subroutine by setting register 1 to the interrupt subroutine address. This may be accomplished by calling the built-in subroutine selector number 7 (interrupt change). The interrupt subroutine must save DF, D, and any registers used on the R2 stack (R2 must first be decremented before storing data). The counter/timer interrupt subroutine may serve as a useful model.

MCP COUNTER/TIMER INTERRUPT TABLE ENTRY

With the mCP extension ROM installed, timer/counter interrupt is specified in the interrupt table with a group number of zero and an EF flag number of zero.

The Micro Concurrent Pascal User's Guide describes how to set up the interrupt table.

MCP COUNTER/TIMER PRESCALING

An 1804 timer/counter prescaling feature is available in the interrupt subroutine. All 1804 timer/counter interrupts are scaled by the value in the 1804/TIMER word in RAM. The TIMER word must be set to the initial prescale value minus 1 in each byte. For example, timer interrupts occurring at 1/60th of a second can be scaled to every second by placing 3B3B in the TIMER word (3BH=59). The high order byte is the initial prescale value and the low order byte is a prescale variable. The SET RAM tuner word (selector number 24) built-in subroutine initializes the timer word.

Normally the timer word is zero so that no prescaling takes place.
EXTENDING THE NUMBER OF TIMERS

Extra counter/timers may be added to the built-in interrupt routine by using the link for external interrupts. Use the external interrupt link to branch to a BCI instruction that checks for counter/timer interrupt. If this is a counter/timer interrupt request decrement each storage word containing the timer. Finally, branch to location C6CO (PTIMER) to reenter the interrupt subroutine. Machine language subroutines would also have to be added to read and write the extra timer words. These subroutines must use RLXA and RSXD instructions to read and write timer words to prevent the interrupt program from updating timer words while the main program accesses the timer words.
**PE000**

CDOS Micro NET Executive
Copyright (C) 1981 Compuserve Incorporated

User ID: 70161.112
Password: XXXXXXXXXX

CompuServe Information Service
13:15 EDT Friday 24-Sep-82

OK

R FILTRN
CompuServe File Transfer Program

Select direction:
1 if to your RCA COSMAC DEVELOPMENT SYSTEM IV
2 if to the PDP-10

? 2

Enter the PDP-10 file specification: PASCAL.SRC

Please give me a filespec for CDOS: PASCAL.SRC:1

INITIALIZING PROTOCOL

UPLOAD STARTING

*** File transfer completed! ***

OK

R MCP

RCA/Enertec Pascal Compiler
Name of your source file: PASCAL.SRC

OK

.compiler passes-

OK

R HEX

R FILTRN
CompuServe File Transfer Program

Select direction:
   1 if to your RCA COSMAC DEVELOPMENT SYSTEM IV
   2 if to the PDP-10
?

Enter the PDP-10 file specification: PASCAL.HEX

Please give me a filespec for CDOS: PASCAL.HEX:1

INITIALIZING PROTOCOL

DOWNLOAD STARTING

...

*** File transfer completed! ***

OK
Logic diagram of Microboard Computer CDP18S601 - CPU and interface portions.
Logic diagram of Microboard Computer CDP18S601 - memory portions.
APPENDIX G

execute program

MCDS Micro NET Executive

Copyright (C) 1981 Compuserve, Inc.

(Dial up the Micro Net telephone number, wait for the modem ready signal)

(control C)

User ID: 70007,530 return
Password: MCDS; APR, return

Compuserve Information Service
10:20 EDT Monday, 03-May-82

OK

(Enter and/or Edit a MCP source program)

run file generator

enter file name

new file response

old file response

Edit MCP source file

(output*)

(*input*)

(*control*)

VAR
X : Integer;

BEGIN
OUT (#4B, CNTRL); (*set A output*)
OUT (#13, CNTRL); (*set B output*)

CYCLE
X : = INN (BPORT);
OUT (X, APORT)

END.

/EXIT

OK
APPENDIX G CONT.

R MCP
RCA/Enertec Pascal Computer
Name of your source file: 10601.MCP return
OK
-compiler passes-
OK

DIR return
10601.LST
10601 DBG
10601.HEX
10601.MCP

R FLTRN
Compuserve File Transfer Program
Select direction:

to your
1. if RCA COSMAC Microboard Development System (down load)
2. if to the PDP-10 (up load)

? 1 return
Enter the PDP-10 file specification: 10601.HEX
Please give me a filespace for MCDS; Ø (tape drive Ø)

INITIALIZING PROTOCOL
Rewind, then hit any key return
Depress Play/Record and any key return

DOWNLOAD STARTING

............

*** File transfer completed ***

OK

* R

TAPE Ø return

LOADING

* R

(control X enters MCDS utility monitor program)
(read tape for down loaded program)
(read tape for Pascal parameter table)
HANDLING SOFTWARE ERRORS

The micro Pascal interpreter halts for the following software errors:

- BAD P-CODE
- MEMORY OVERFLOW
- RANGE ERROR
- DIVIDE BY ZERO

The micro Pascal interpreter continues processing P-codes for the following software error:

- ARITHMETIC OVERFLOW
  (use PEEK to access software error code and POKE to reset error code to zero)

To halt the interpreter on arithmetic overflow instead of continuing, change the branch table location 0008H to COE6EH. Arithmetic overflow occurs in ADD, SUBTRACT, NEGATE, ABS, DEC, INC, COPY BYTE and MULTIPLY integer P-code subroutines. The stack contains the overflow result and may be used in subsequent P-code operations.

Micro Concurrent Pascal handles errors differently from micro Pascal. If the software error process exists, micro Concurrent Pascal will execute the error process when an error occurs, otherwise the program halts.

MCP COMPILER DIRECTIVES

The following directives must be defined in the micro Pascal source program:

- HIGHBYTE  (determines HI-LO byte order for P-code generation)
- PERMIT    (permits calls to INN, OUT, PEEK, POKE)

For example:

(*$ HIGHBYTE, PERMIT $*)

should be the first line of the program.
USER DEFINED BUILT-IN SUBROUTINE LINKAGE

Build-in subroutines use register 8 as the program counter with X set to register 14. The first parameter is the integer value for indexing into the subroutine address table. This value is popped off the stack by the interpreter before entry into the subroutine. Register 5 contains the second parameter on entry to the subroutine. Built-in assembly language procedures must have at least two formal parameters while functions must have at least one formal parameter. Register 14 points to the top of stack most significant byte on entry; X is set to register 14.

On exit from the subroutine X must be set to register 14. To return use the instruction set P to register 15.

Registers that may be used without saving and restoring their constants are R4, R5, and R7. All others must be saved and restored using the R2 stack. Register 2 must be decremented before storing registers.

The subroutine linkage technique does not use standard call/return because it references a table of addresses and is faster than SCAL/SRET instructions.

An example of a built-in subroutine is GET_Q_FLAG:

.X is 14; P is 8
ASKQ DEC 14 ..reserve stack space
LDI 0
LSNQ ..test Q flag
LDI 1
STXD ..store least significant byte
LDI 0
STR 14 ..store most significant byte
SEP 15 ..return to interpreter
USER-DEFINED EXTERNAL ASSEMBLY LANGUAGE SUBROUTINES

A selector equal to 0 identifies a user-defined assembly language routine. The user provides the address of the assembly language routine in the call. The interpreter pops it from the stack and uses the standard call and return technique to execute the routine. Register 6 contains the address for returning to the interpreter. Register 3 is the program counter used to enter the assembly language routine. Register 14 is the data parameter pointer to the first byte of any parameters before execution of the routine. After execution the stack pointer register 14 points to the top of the stack entry. Register 2 is the system stack used to save registers. It is used to return control to the interpreter using the standard return instruction.

The user is responsible for setting register 14 to point beyond all parameters. Registers available to run assembly language routines are 4, 5, 7, and 8.

Registers 1 and 13 must not be altered at any time if interrupts are enabled. Other registers may be used if restored to their original value. Register 2 must be decremented before storing registers.
The following model and example describes the construction of an external assembly language subroutine:

```
SUBR SEX R2               ..set X to system stack
  optional
    DEC R2
    RSXD register ..save register on R2 stack
    SEX R14 ..set X to parameter stack
    RLXA register ..get data from stack
    DEC R14
    RSXD register ..store data on stack
    INC R14 ..adjust stack pointer to
               point to most significant
               byte

  SEX R2
  optional
  restore
  registers
    INC R2 ..adjust stack pointer
    RXLA register ..restore register saved
    SRET R6 ..standard return
```