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 microrocessor. 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.[1] The user's manual for mCP may be aquired from RCA Systems Marketing [redacted] 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

and info. on Compuserve

---

*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 CDP1804P1 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 CDP1804P1[3] 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.[2] 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 boundries of a contiguous RAM memory space for interpreter working storage (see appendix A). The 64 byte on-chip RAM of the CDP1804P1 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 CDP1804P1 (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

AND	ARRAY	BEGIN	CASE	CONST	CYCLE
DIV	DO	DOWNTO	ELSE	END	ENTRY
EXTERNAL	FOR	FUNCTION	IF	MOD	NOT
OF	OR	PROCEDURE	RECORD	REPEAT	THEN
TO	TYPE	UNIV	UNTIL	VAR	WHILE
WITH	XOR				

STANDARD FUNCTIONS

ABS	ADDR	ADR	CHR	INN
MAXINT	ORD	PEEK	PRED	SUCC

STANDARD PROCEDURES

DEC	INC	OUT	POKE

STANDARD DATA TYPES

ADDRESS	BOOLEAN	CHAR	INTEGER

TABLE 2

CDP1804P1MICRO CONCURRENT PASCAL

EXTENSION LANGUAGE (4K ROM)

WORD SYMBOLS

CLASS	DEVICE_MON	DOIO	IN	INIT
MONITOR	PROCESS	SET	STRUC_CON	

STANDARD FUNCTIONS

CONV	EMPTY	STR_STOP	TRUNC	WORD

STANDARD PROCEDURES

CONTINUE	DELAY	DIDDLE	INITQUEUE	STR_COUNT

STANDARD TYPES

QUEUE	REAL

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 $\overline{\text{WAIT}}$ and $\overline{\text{CLEAR}}$ lines and switching them (ie. WAIT to pin 3 and CLEAR to pin 2 of the CPU) and disconnecting pin 16 ($\overline{\text{ME}}/\overline{\text{EMS}}$) and letting it float. The CDM5332P1 extension interpreter may be copied onto disk using an altered CDP18S480 PROM programmer board.[4] 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

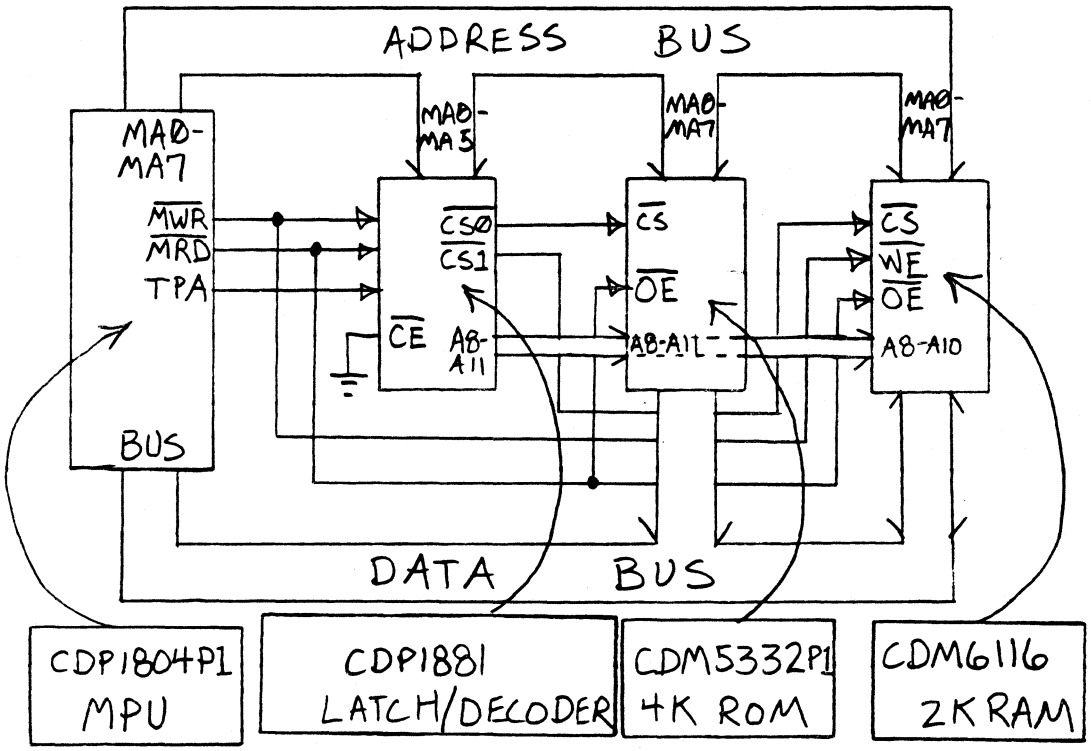


Figure 1-1804P1 mCP system

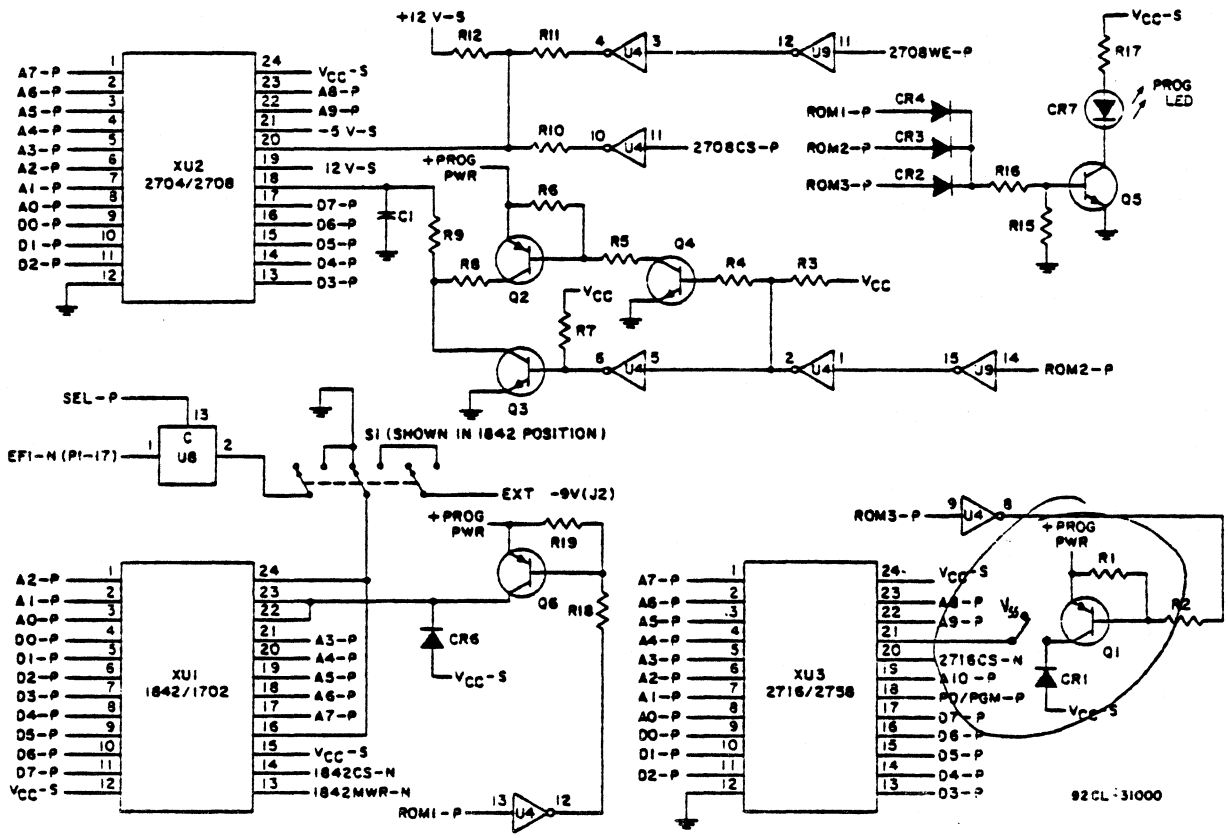


Fig2 - Programming logic of PROM Programmer module

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 - 0FFF 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
   A11-P and connect to + 5V
5. Connect A11-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 E000) of the CDP18S652 Tape I/O and Memory board. The ROM in socket U3 must be removed because it is mapped at C000 (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

1. RCA Microprocessors User Group Compuserve User's Guide, July 1982
2. K. Jensen and N. Wirth, PASCAL User Manual and Report (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

APPENDIX A - MEMORY

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.

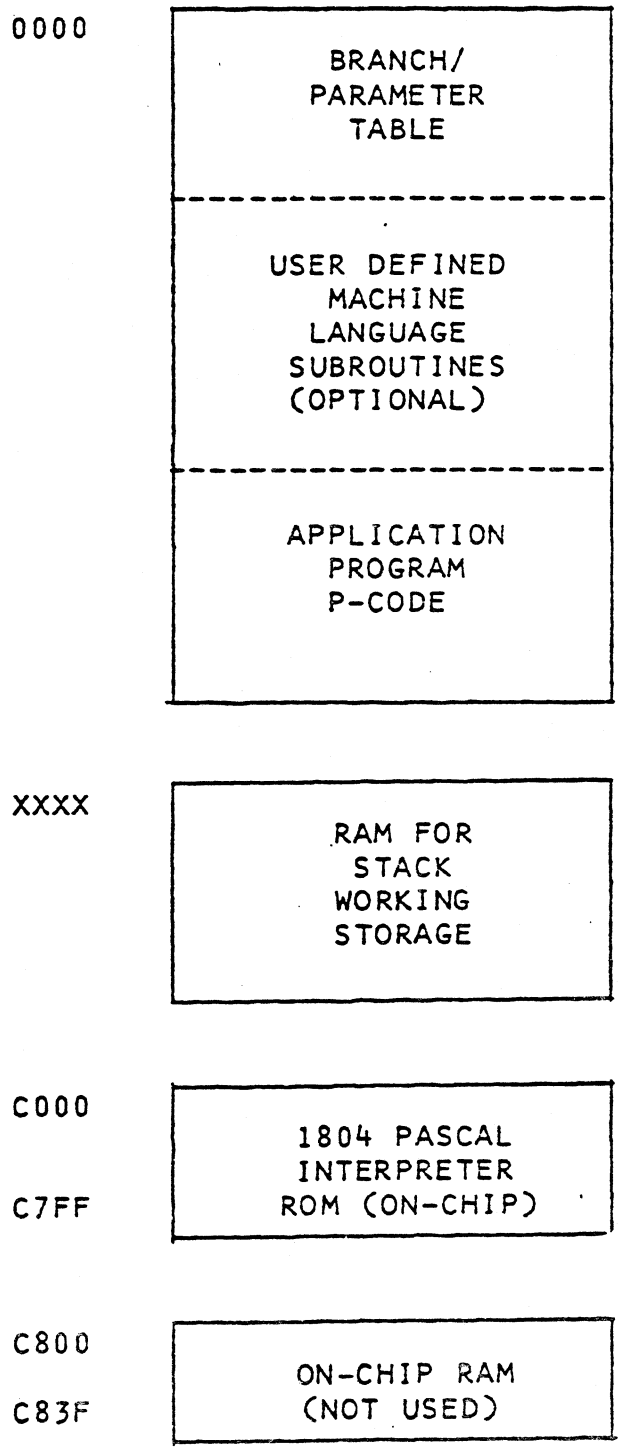
Micro Pascal uses three bytes of the R2 system stack. Micro Concurrent Pascal uses three bytes of the R2 system stack plus 13 bytes when floating point subroutines are used.

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.

APPENDIX A CONT.

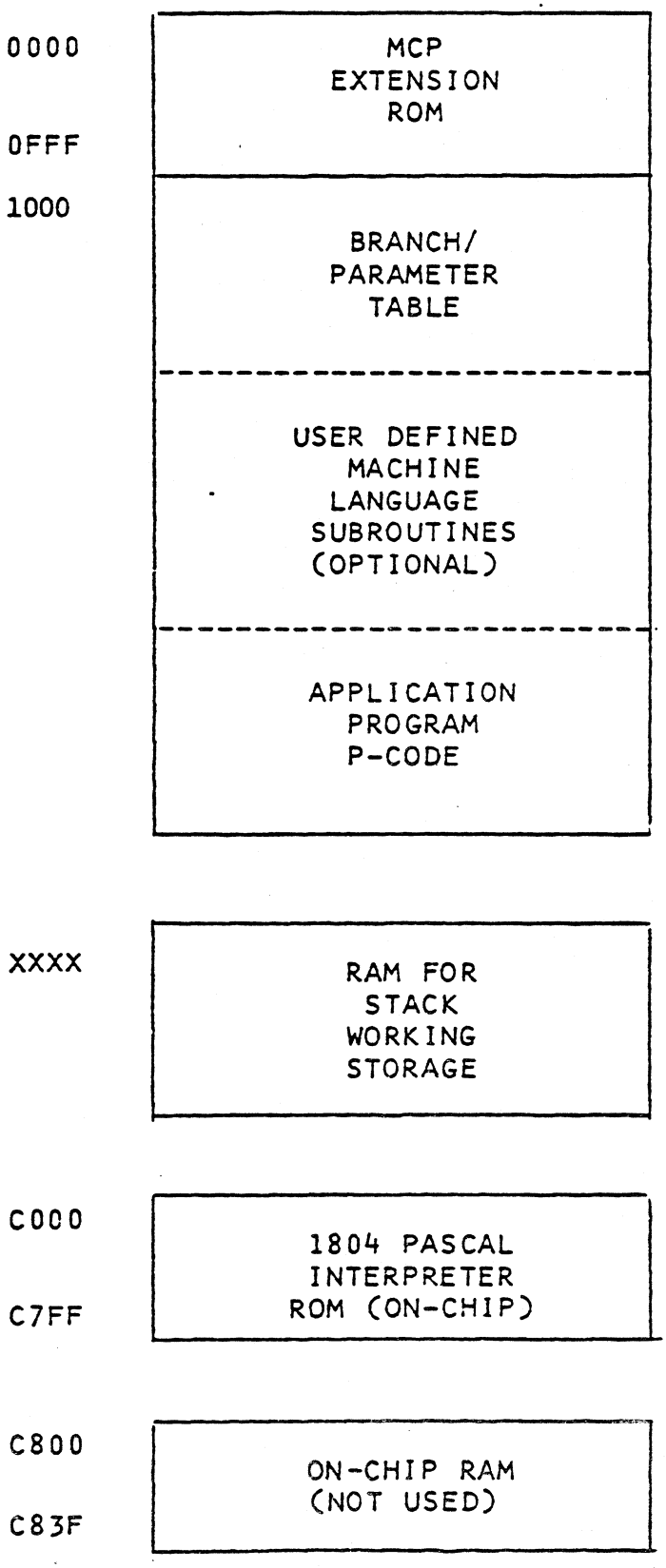
CDP1804P1MICRO CONCURRENT PASCAL
MEMORY MAP

(CORE INTERPRETER ONLY)



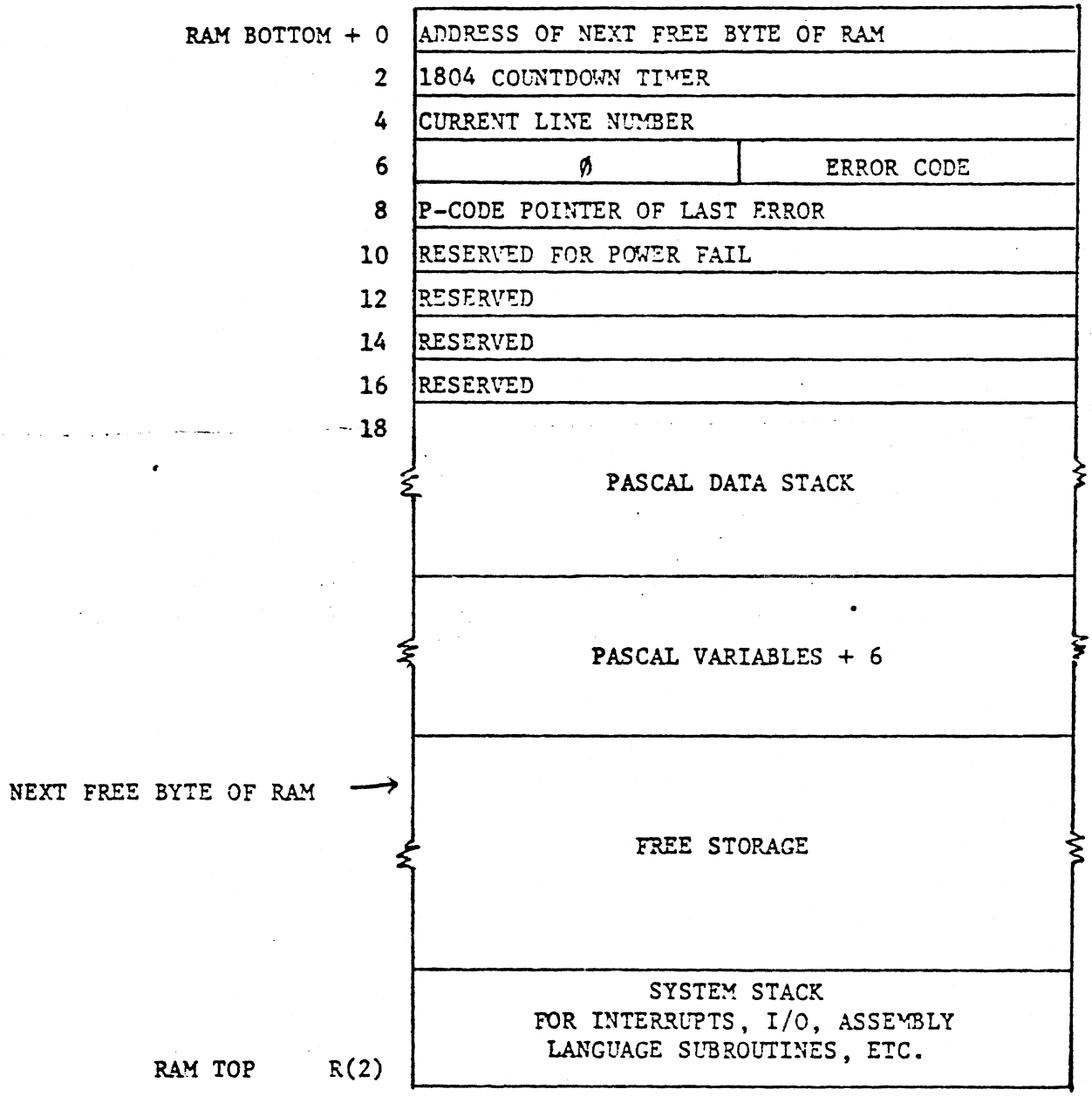
APPENDIX A CONT.

CDP1804P1MICRO CONCURRENT PASCAL
MEMORY MAP WITH EXTENSION ROM



APPENDIX A CONT.

RAM STORAGE LAYOUT



APPENDIX A CONT.

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:

Location	Data
0003	0060 address of patch (can be any suitable locatio
0060	68CD000F
	ED 68676865 6886 686A
	E2 68B5E5 68 B725
	COC01B

66

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

Location	Data
1003	1060 address of patch (can be any suitable location)
1060	ED 6867 6865 6866686A
	E2 68B5 E5 68B7 25
	COC01B

APPENDIX A CONT

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:

ADDRESS	DATA	COMMENT
0000	7100	Disable interrupts
0002	C0C000	Long branch to pre-initialization
0005	EEDF00	Post-initialization (could be long branch)
0008	C0C079	Long branch to software error continuation
000B	C0C6BB	Long branch to external interrupt
000E	C1	P-code subrouting adress table page
000F	----	Bottom of RAM address
0011	----	Top of RAM address
0013	----	Size of RAM (RAM Top-RAM bottom +1)
0015	----	P-code starting address
0017	0000	Reserved for micro Concurrent Pascal extension program

The programmer defines the location of RAM working storage and the starting P-code address. All other fields should reamin 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 maching 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 relocatable 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:

ADDRESS	SUBROUTINE LOCATION	SUBROUTINE NAME	SELECTOR NUMBER	
0019	C558	Test external flag 1 (EF1)	1	1802/ 1804 built-in subr.
001B	C55C	Test external flag 2 (EF2)	2	
001D	C560	Test external flag 3 (EF3)	3	
001F	C564	Test external flag 4 (EF4)	4	
0021	C600	Set DMA register (R0) to address	5	
0023	C605	Get DMA register (R0) address	6	
0025	C277	Set INTERRUPT register (R1) address	7	
0027	C60A	Set Q flag on or off	8	
0029	C610	Test Q flag	9	
002B	C20E	Load counter	10	1804 built-in subr.
002D	C212	Get counter	11	
002F	C22E	Stop counter	12	
0031	C231	Decrement Counter	13	
0033	C234	Start timer	14	
0035	C237	Start counter mode 1	15	
0037	C23A	Start counter mode 2	16	
0039	C257	Start pulse mode 1	17	
003B	C25A	Start pulse mode 2	18	
003D	C2B5	Enable toggle Q flag	19	
003F	C2B8	External interrupt enable	20	
0041	C2BB	External interrupt disable	21	
0043	C0DB	Counter interrupt enable	22	
0045	C0DE	Counter interrupt disable	23	
0047	C2CE	Set RAM timer word	24	
0049	C0E7	Get RAM timer word	25	
004B	C0E1	Enable interrupts	26	
004D	C0E4	Disable interrupts	27	
004F	C275	Wait for interrupt	28	

The built-in machine language subroutines reside in the 1804 ROM. Additional built-in subroutines may be coded by extending the list of sub-

APPENDIX A CONT.

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.

APPENDIX A CONT.

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:

ADDRESS	DATA	COMMENT
1000	7100	Disable interrupts
1002	C0C004	Long branch to pre-initialization
1005	EEDF00	Post-initialization (could be long branch)
1008	C00043	Long branch to software error continuation
100B	C00304	Long branch to interrupt subroutine
100E	01	P-code subroutine address table page
100F	------	Bottom of RAM address
1011	------	Top of RAM address
1013	------	Size of RAM (RAM top - RAM bottom +1)
1015	------	P-code starting address
1017	0040	DOIO size and stack margin

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:

ADDRESS	SUBROUTINE LOCATION	SUBROUTINE NAME	SELECTOR NUMBER
1019	C558	Test external flag 1 (EF1)	1
101B	C55C	Test external flag 2 (EF2)	2
101D	C560	Test external flag 3 (EF3)	3
101F	C564	Test external flag 4 (EF4)	4
1021	C600	Set DMA register R(0) to address	5
1023	C605	Get DMA register R(0) address	6
1025	022E	Change address of interrupt table	7
1027	C60A	Set Q flag on or off	8
1029	C610	Test Q flag	9
102B	C20E	Load counter	10
102D	C212	Get counter	11
102F	C22E	Stop counter	12
1031	C231	Decrement counter	13
1033	C234	Start timer	14
1035	C237	Start counter mode 1	15
1037	C23A	Start counter mode 2	16
1039	C257	Start pulse mode 1	17
103B	C25A	Start pulse mode 2	18
103D	C2B5	Enable toggle Q flag	19
103F	C2B8	External interrupt enable	20
1041	C2BB	External interrupt disable	21
1043	CODB	Counter interrupt enable	22
1045	CODE	Counter interrupt disable	23
1047	C2CE	Set RAM timer word	24
1049	C0E7	Get RAM timer word	25

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:

```
TYPE    BUILT_IN_SUBR = 1...28;

        INT_BYTE = 0...255;
```

APPENDIX B CONT.

1. TEST_EF1 (1)

Assembly language function tests external flag 1 (EF1) and returns either 0 ot 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';
```

APPENDIX B CONT.

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 'DMASET';
```

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_INTTBLS (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_INTTBLS (NEW_INTTBL_ADR:  ADDRESS; SELECTOR:
  BUILT_IN_SUBR); EXTERNAL 'INTCH';
```

APPENDIX B CONT.

8. 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';
```

APPENDIX B CONT.

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:

```
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:

```
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:

```
PROCEDURE START_TIMER (DUMMY:  INTEGER; SELECTOR:  BUILT_IN_SUBR);
  EXTERNAL 'STM';
```

15. START_COUNTER_MODEL (0, 15)

Assembly language procedure sets counter mode 1 and starts the counter. The first parameter is a dummy parameter. This procedure executed the SCM1 instruction.

Routine declaration:

```
PROCEDURE START_COUNTER_MODE1 (DUMMY:  INGETER; SELECTOR:
  BUILT_IN_SUBR); EXTERNAL SCM1';
```

APPENDIX B CONT.

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_MODE 2 (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';
```

APPENDIX B CONT.

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 prodecure 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 prodecure executes the CID instruction.

Routine declaration:

```
PROCEDURE CNT_INTERRUPT_DISABLE (DUMMY:  INTEGER; SELECTOR:
  BUILT_IN_SUBR); EXTERNAL 'CID';
```

APPENDIX B CONT.

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 declatation:

```
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';
```

APPENDIX B CONT

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';
```

APPENDIX C

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.

APPENDIX C CONT

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 occuring 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.

APPENDIX C CONT

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.

APPENDIX D

```
*$PE000

CDOS Micro NET Executive
Copyright (C) 1981 Compuserve Incorporated
^C

User ID: 70161,112
Password: XXXXXXXX

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
```

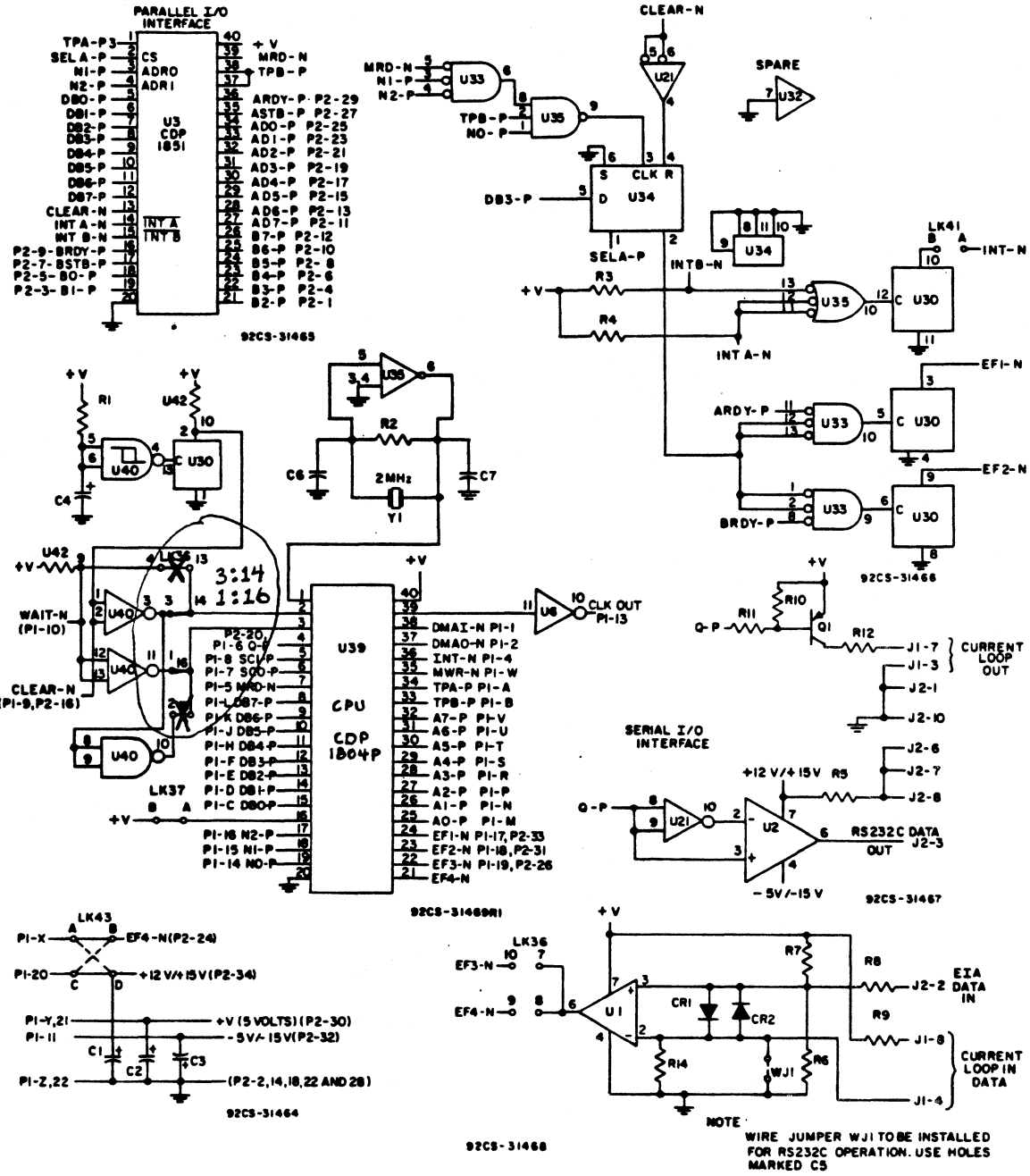
APPENDIX D CONT.

```
Name of your source file: PASCAL.SRC

OK


-compiler passes-

OK                                          R RHEX
```



```
R FILTRN
CompuServe File Transfer Program


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

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
```

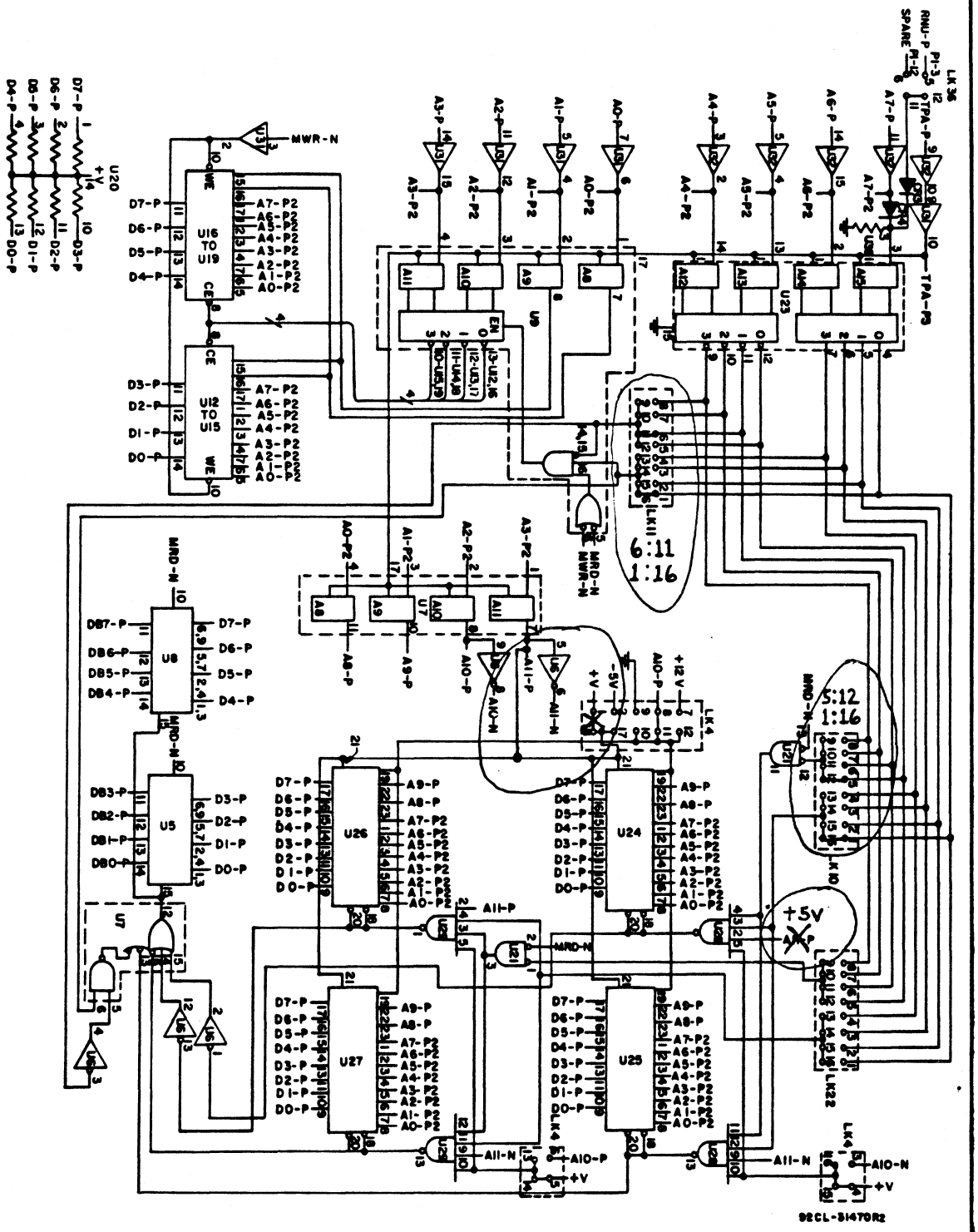
APPENDIX E



Logic diagram of Microboard Computer CDP18S601 - CPU and interface portions.

APPENDIX F

CDP18S601



Logic diagram of Microboard Computer CDP18S601 - memory portions.

APPENDIX G

```
*PE000                                    (execute program)
MCDS Micro NET Executive
Copyright (C) 1981 Compuserve, Inc.
(Dial up the Micro Net telephone number, wait for the modem ready signal)

↑C                                        (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)

R FILGE return                            (run file generator)
*I0601.MCP return                         (enter file name)
New file I0601.MCP created-ready          (new file response)
File I0601.MCP  ready                     (old file response)
                                          (Edit MCP source file)

(*$ PERMIT, HIGHBYTE, LIST-SHORT $*)
(*Turn on output port using switches from input*)
CONST
APORT = ADR (#0804);                      (*output*)
BPORT = ADR (#080E);                      (*input*)
CNTRL = ADR (#0802);                      (*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                                        (execute compiler)
RCA/Enertec Pascal Computer
Name of your source file:  I06Ø1.MCP return

OK

-compiler passes-

OK <------------------------------ R RHEX

DIR return                                   (display directory)
I06Ø1.LST
I06Ø1.DBG
I06Ø1.HEX
I0601.MCP

R FILTRN                                     (run file transfer to perform down load)
Compuserve File Transfer Program
Select direction:
    1.  if to your RCA COSMAC Microboard Development System (down load)
    2.  if to the PDP - 10                   (up load)

? 1 return
Enter the PDP-10 file specification:  I0601.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
.........                                    (down load)
*** File transfer completed ***

OK

↑X                                           (control X enters MCDS utility monitor
                                              program)
* R                                          (read tape for down loaded program)
TAPE # Ø return
LOADING
*R                                           (read tape for Pascal parameter table)
```

APPENDIX H

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 C0C6EB. 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.

APPENDIX I

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   Ø
        LSNQ        ..test Q flag
        LDI   1
        STXD        ..store least significant byte
        LDI   Ø
        STR   14    ..store most significant byte
        SEP   15    ..return to interpreter
```

APPENDIX I CONT

USER-DEFINED EXTERNAL ASSEMBLY LANGUAGE SUBROUTINES

A selector equal to 0 identifies a user-defined assembly langugage 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.

APPENDIX I CONT

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