MicroNator

BASIC11

UNIVERSAL
DEVELOPMENT BOARD

Version 4.04a

BASIC11

Version 027

RF-232

http://www.micronator.com

ISBN 2-9803460-3-9

Dépôt Légal - Bibliothèque Nationale du Québec, novembre 1996.

PRINTED IN CANADA

MicroNator

UNIVERSAL
DEVELOPMENT BOARD
Version 4.04a

BASIC11
Version: 027

RF-232

http://www.micronator.com

MicroNator

MicroNator   UNIVERSAL DEVELOPMENT BOARD   BASIC11

RF-232
1404 rue Galt
Montréal, Qc H4E 1H9
CANADA
Tél: (514) 761-4201

RF-232
21 rue André Gide
59123 ZUYDCOOTE
FRANCE
Tél: 03 28 58 28 39

micronator@micronator.com

ISBN 2-9803460-3-9

Dépôt Légal - Bibliothèque Nationale du Québec, novembre 1996

MicroNator

UNIVERSAL
DEVELOPMENT BOARD

Basic11 Manual

Information contained in this manual applies to
Version (4.04) MicroNator UNIVERSAL DEVELOPMENT BOARD
serial numbers 4000 through 9999

PREFACE

BASIC11 is a very fast and complete control oriented BASIC interpreter for the MicroNator MC68HC11 microcomputer system. It provides all the functions of standard BASIC along with a number of enhancements that allow direct control of some of the MC68HC11's hardware features using BASIC statements.

The only limitations of BASIC11 (which usually are not limitations in a control environment) are that it only supports integer variables. Also strings are only supported in PRINT and INPUT statements.

Lines entered into a BASIC11 program must begin with a line number and must be terminated by a carrier return. Lines may be no longer than 80 characters. All lines are automatically put in numerical order by BASIC11 as they are typed in. Lines may be deleted from a program by simply typing the line number followed immediately by a carriage return.

The syntax of each line in a BASIC11 program is checked as soon as a CARRIER RETURN is entered and any errors are reported immediately. This prevents the interpreter from having to check syntax at runtime and is one of the things that contributes to BASIC11's speed.

WARRANTY

Even though many hours of work went into the writing and testing of BASIC11, it is believed to be “bug free”, BASIC11 is supplied “as-is” and without warranty. The author makes no express or implied warranties as to the fitness of use and merchantability of the product. The user assumes the entire risk as to its quality, performance and fitness of use.

In no event will the author be liable for direct, indirect, incidental, or consequential damages resulting from the use of this product. Including but not limited to loss of sales, income, service, profits, or potential profits.

In the event a situation is found where the program does not function as the manual describes, the author will attempt to correct any errors brought to his attention, however he makes no guarantee to do so.

COPYRIGHT NOTICE

†NOTE

MicroNator, CPU-11/64e2, and UCT-11/64e2 System refer to the same development system.

Contents at a Glance

Table of Contents

CHAPTER 1

THE BASICS OF BASIC11

1.1 Lines

Each line of a BASIC11 program must begin with a line number. Lines may be numbered from 1 through 32767 and each line must be terminated by a CARRIER RETURN. Lines may contain multiple statements that are separated by colons. Spaces may be used freely in BASIC11 statements to improve their readability with one exception. Assignment statements and arithmetic/logic statements may contain no imbedded blanks. Some examples follow:

```
10 PRINT X,X*X,RND(0)-5
20 X=5: Y=10: Z=15
```

1.2 Integer Constants:

All integer constants are represented internally as 16 bit two's complement numbers with a decimal range of -32768 to 32767 ($0000 to $FFFF in hex). In the source program and input statements numbers may be represented in either decimal or hexadecimal form. All hexadecimal constants must be prefixed by a dollar sign ($). Some examples of integer constants are:

```
50 X=1000
60 Y=-55
70 Z=PEEK($E010)
```

1.3 String Constants:

As mentioned earlier, BASIC11 does not support string variables. However, it does support string constants in both PRINT statements and INPUT statements to allow for prompting of input data. Some examples of string constants follow:

```
100 PRINT “Please Enter Your Name”
200 INPUT “Enter a Number”,N
```

1.4 Variables:

BASIC11 currently supports only integer variables. Integer variable names can consist of a single alphabetic letter or a letter followed by another letter or number. Examples of integer variable names are:

```
AB, XZ,R1,TO,IF
```

Notice in the above example that two of the variables are the same as the BASIC11 keywords TO and IF. In many BASIC's this is illegal but in BASIC11 it is perfectly legal.

Any legal integer variable name may also be subscripted or dimensioned using the DIM statement. A variable is dimensioned by following any legal integer variable name by an expression that is enclosed in parentheses.

†NOTE that when a variable is declared in a DIM statement storage is not allocated until runtime. This is because all array storage is allocated dynamically. All dimensioned variables start with 0. For example:

```
300 DIM AX(4)
```

Will create the following five variables:

```
AX(0), AX(1), AX(2), AX(3), AX(4)
```

Again, the same variable name may be used for both a non-dimensioned and dimensioned variable. All dimensioned variables must be declared in a DIM statement before they can be referenced in an expression or Error # 24 (Undimensioned Array) will result when the variable is referenced during a program run.

1.5 Variable Assignment

By using the LET, INPUT, INBYTE, or the READ statements variables may be assigned values. The most common way to assign a value to a variable is through the use of the LET statement. For example, the statement:

```
90 LET GD=7
```

Would assign the integer value of 7 to the variable “GD” so that each time the variable “GD” is used in an expression, the numerical value of 7 would actually be substituted.

An INPUT statement, when executed, will cause BASIC11 to stop, print a question mark on the terminal, and wait for the user to enter a numerical constant. For example, the statement:

```
40 INPUT A1
```

will assign whatever number is typed at the terminal to the variable “A1”.

The INBYTE statement is similar to the input statement except that instead of expecting an ASCII formatted number from the terminal input device, it assigns the value of the ASCII byte to the variable that follows it. For example if the statement INBYTE AX were executed and the character “Y” were typed at the terminal, the variable AX would contain the value 89 which is the numerical value of the ASCII character “Y”. The INBYTE statement is very useful for obtaining data from the control terminal.

The READ statement works almost like the INPUT statement except that the numerical constant is taken from a DATA statement instead of being typed in by a user from the terminal (more about the READ and DATA statements later).

1.6 Operators:

There are three classes of operators available in BASIC11. The one most are familiar with is the mathematical operators. Addition, subtraction, multiplication, and division. The mathematical operators are:

SYMBOL	EXAMPLE	MEANING
+	A+B	Add A to B
-	A-B	Subtract B from A
*	A*B	Multiply A and B
/	A/B	Divide A by B
\	A\B	Remainder of (A/B) or Modulo

The next class of operators is the logical operators. They are used to perform “bitwise” operations. They can be used to “ignore” certain bits within a word or in conditional tests when more than one condition needs to be tested. The logical operators are:

SYMBOL	EXAMPLE	MEANING
.AND.	A.AND.B	Bitwise logical AND of A and B.
.OR.	A.OR.B	Bitwise logical OR of A and B.
.EOR.	A.EOR.B	Bitwise logical EXCLUSIVE OR of A and B.

The last class of operators is the relational operators. These are used in the IF and WHILE statements to test whether one expression is less than, greater than, or equal to another expression. The relational operators are:

SYMBOL	EXAMPLE	MEANING
=	A=B	True if A is equal to B
<>	A<>B	True if A is not equal to B
<	A<B	True if A is less than B
>	A>B	True if A is greater than B
<=	A<=B	True if A is less than or equal to B
>=	A>=B	True if A is greater than or equal to B

1.7 Operator Precedence:

Overall operator precedence is shown below. The operator at the top of the list has the highest priority in any expression, while the operator at the bottom has the lowest priority.

()	Expressions enclosed in parenthesis
NOT	Unary minus and NOT (one's complement)
* / \	Multiplication, division, and Mod (remainder)
+ -	Addition and subtraction
=	Relational operators
<>	
<	
>	
<=	
>=	
.AND.	All logical operators have the same precedence
.OR.	
.EOR.	

1.8 Operating Modes:

BASIC11 has two operating modes, the RUN mode and the immediate Mode. In the RUN mode program lines that have previously been entered are executed starting with the smallest line number and continues until a STOP or END statement is executed, an error occurs, or an “Alternate-C” is typed on the terminal.

In the immediate Mode, any legal BASIC11 statement or command may be typed in without a line number and the statement will immediately be executed. BASIC11 may be used in this mode to debug programs by examining variables, memory locations, or I/O ports.

1.9 Remarks:

It is a good idea to place remarks throughout your programs so that someone else can understand the operation of your program if it ever becomes necessary to change it. It can even help you if you haven't worked with the program in a while. Even though the REM statement is not executable it may be referenced by other program statements (for example, by a GOTO or GOSUB statement).

CHAPTER 2

COMMANDS OF BASIC11

2.1 Commands

Commands are instructions to BASIC11 that allow it to perform "housekeeping" tasks at the user's request. None of the following commands may appear in a BASIC11 program.

CLEAR

The clear command is used to set all variables to zero and to reset the GOSUB, WHILE, and FOR - NEXT stacks. A clear is automatically performed when a RUN command is entered.

CONT

The CONT command is used to restart a BASIC11 program either after it has been stopped by either a STOP statement or an "Alternate-C" was typed at the terminal. The program can't be restarted if an error occurred in the program or if the program is modified.

LIST

```
LIST                        Lists the entire program
LIST [line #]               Lists one line
LIST [line #]-[line #]      Lists from the first line num-
                            ber
                            through the second line number
```

The LIST command can be used to display selected lines of the program on the terminal. As can be seen from the above examples, all, part, or a single line of the program may be listed.

LLIST

```
LLIST [line #]
LLIST [line #]-[line #]
```

The LLIST works in the same manner as the LIST command, except that the program lines are sent to the system printer instead of the terminal.

†NOTE: MicroNator defines the system printer as the monitor screen.

NEW

The NEW command is used to clear out both the BASIC program buffer and the variable storage space. It prepares BASIC11 to accept a “New” program.

RUN

The RUN command is used to begin execution of the program that is currently in memory.

ESAVE

The ESAVE command is used to save the program that is currently in RAM to the program storage EEPROM that resides in the system. The EEPROM storage is from $8000 [$32,768_{10}$] to $DDFF [$56,831_{10}$].

†NOTE: Each byte takes <u>10 msec</u> to be written to the EEPROM so be patient...

†NOTE: The program can be as large as 24,054 bytes if stored in EEPROM.

($8000-$8009) [$32,768_{10}$ - $32,777_{10}$] reserved for pointers,
($800A-$CFFF) [$32,778_{10}$ - $53,247_{10}$] for the user program,
($D000-$DDFF) [$53,248_{10}$ - $56,831_{10}$] for the user callable assembler subroutines.

†NOTE: The program can be as large as 27,761 bytes if used only in RAM.

($1040-$7CB0) [$4,160_{10}$ - $31,920_{10}$] start and end of user program usable RAM

†NOTE: If the user wants RAM for storage area, he can use from $7CB0 [$31,920_{10}$] and down.

ELOAD

The ELOAD command is used to transfer a program to RAM that had previously been saved using the ESAVE command.

AUTOST

The AUTOST command is used to set a flag that resides in the program storage EEPROM that will allow the BASIC11 program to execute from a powerup or reset condition.

†NOTE: When AUTOST is on, BASIC11 program is executed out of the program storage EEPROM and is not copied into RAM. This allows the entire system RAM to be used for variable storage.

NOAUTO

This command resets the auto start flag set by the AUTOST command and disables the automatic execution of a BASIC program stored in the program storage EEPROM.

FREE

The FREE command may be used to Display the amount of RAM memory that is currently available for BASIC11 program statements and variables.

CHAPTER 3

STATEMENTS OF BASIC11

All of the following statements are used in the creation of BASIC11 programs. The statements are arranged in logical groups to make similar functions easy to find. Each statement is accompanied by one or more program lines showing it's proper usage and an explanation of how the statement works if necessary.

3.1 Assignment:

DATA **<line number> DATA <number> [,<number>,<number>...]**

```
10 DATA 500,-10,200,99,$CD03
20 DATA $FE, 1000, -300
```

The data statement is used to specify data that will be assigned to variables with a READ statement. The data is read from left to right and always begins with the first data statement in the program. When the program has read all the data in a single DATA statement, BASIC11 will search the program for the next DATA statement starting at the line following the just exhausted DATA line. This is done because all data statements in a program are considered logically to be one long DATA statement.

LET **<line number> LET <variable>=<expression>**

```
10 LET X=5
20 LET Y=25*(Y/3)
30 LET AX(3)=AX(5)*10
40 CD=DE+23
50 XZ=-55
```

The LET statement is the most often used way to assign a value to a variable. Notice in line numbers 40 and 50 above do not contain the keyword LET. This is what is known as an implied LET and is a feature of BASIC11 to help cut down typing time when entering a program since this is one of the most often used statements.

†NOTE: As stated earlier, assignment statements and arithmetic/logic statements may contain no imbedded spaces. This means that there may be no spaces between the variable and equals, the equal and the start of the expression, and no spaces within the expression.

READ **<line number> READ <variable> [,<variable>,<variable>,...]**

```
READ A,B,C
```

The READ statement is used in conjunction with the DATA statement to assign values to variables. The first time the READ statement is executed, it will assign the first item in the first DATA statement to the first variable in its variable list. If additional variables are present in its variable list, each one will sequentially be assigned the next item in the DATA statement. Care must be taken when a program is written so that BASIC11 does not try to read past the last item in the last DATA statement. If this happens, Error # 38 (Out of Data in “READ” or “RESTORE” Statement) will be issued.

RESTORE **<line number> RESTORE**

```
330 RESTORE
```

The RESTORE statement is used to reset BASIC11's internal “pointer to the next item” in a DATA statement to the first item in the first DATA statement that appears in the program.

EEP() **<line number> EEP(<expression>)=<expression>**

```
25 EEP(30)=$55
30 EEP(X+1)=A/B
```

The EEP() statement is actually a special form of the implied LET. EEP() is actually a subscripted variable that allows the BASIC program to directly write a word (2 bytes) to the external EEPROM ($8000 to $DDFE). Writing to the BASIC11 program ($DE00-$FFFF) area is not allowed. The high byte is written to the low address then the low byte is written to the high address. All the timing and control information necessary to write to the EEPROM is taken care of by BASIC11. This feature makes it very convenient to save any kind of calibration data that must be retained in the event of a power failure. Currently the subscript of the EEP() statement is limited to $8000-$DDFE.

†NOTE: It takes 10 msec to write a byte in the external EEPROM.

CAUTION: Since the number of write/erase cycles of the EEPROM is guaranteed to about 100,000 times, be very careful that the EEP() statement doesn't get executed repeatedly for the same location by having it reside within a loop.

†NOTE: The routine that writes to the EEPROM disables, for 10 msec, the IRQ while it is writing. This means that he TIME function in BASIC11 is not updated while the EEP()

functions is executing.

```
#10 EEP($9000)=$ABCD
#20 I=PEEK($9000)
#30 J=PEEK($9001)
#40 PRINT HEX2(I), HEX2(J)
#RUN
AB      CD
```

†NOTE: Beware that the ESAVE program storage area begins at $8000 and ends at $DDFF. Use the FREE command to calculate the beginning of your safe storage area.

PORTA
PORTB
PORTC
PORTD **<line number> PORTx=<expression>**

```
75 PORTA=$A5
85 PORTA=X+(E-K)
```

The PORTx statement is also a special form of the implied LET statement. It allows BASIC11 to directly assign an 8-bit value to one of the MC68HC11's I/O ports.

†NOTE: For a logic value to actually appear on one of the port pins, that particular pin must have been programmed as an output by using the POKE() statement to write a “1” to that particular port's Data Direction Register (DDR). If a value of greater than 255 ($FF) is written to a port, Error #46 (Tried to Assign a Value of < 0 or > 255 to a PORT) will be issued.

†NOTE: Please take notice that PORTB and PORTC are taken by data and address in MicroNator multiplex mode. Port PD2..PD5 are used for the SPI communication but can be used for other purposes.

TIME **<line number> TIME=<expression>**

```
65 TIME=0
75 TIME=SC/60
```

The TIME statement, like the EEP() and PORTx statement, is a special form of the implied LET statement that allows the BASIC program to assign a value to the system variable TIME which is used as BASIC11's “Real Time Clock”. BASIC11 uses the output compare one (OC1) register to generate a periodic interrupt which is then divided down by software so that the variable TIME is incremented once per second. Since the variable is a 16 bit number, elapsed time can be kept track of for 65536 seconds (approximately 18 hours) without any software overhead.

See MicroNator user's manual for the MC146818 RTC (Real Time Clock) and functions.

PACC **<line number> PACC=<expression>**

```
85 PACC=25
95 PACC=-5.AND.$00FF
```

Like the TIME, EEP(), and PORT statements, PACC statement is a special form of the implied LET statement that allows the programmer to directly alter the value of the MC68HC11s Pulse Accumulator. Since the Pulse Accumulator is only an eight bit register, the value must be in the range 0 <= expression <= 255 or Error #53 (Tried to assign a value of <0 or >255 to PACC) will be issued.

3.2 Control Transfer:

GOSUB **<line number> GOSUB <line number>**

```
100 GOSUB 1000
```

The GOSUB statement is used to transfer control of the program to the subroutine whose line number follows the GOSUB statement. The last statement of any subroutine should be a RETURN statement which will return control back to the statement following the GOSUB.

RETURN **<line number> RETURN**

```
1100 RETURN
```

As mentioned above the RETURN statement should be the last executed statement in a subroutine and will return program execution to the statement following the GOSUB.

GOTO **<line number> GOTO <line number>**

```
50 GOTO 10
```

The GOTO statement is used just to transfer control of program execution to the line number following the GOTO statement.

ON GOSUB <line number> ON <expression> GOSUB <line number> [,<line number>,......]

```
200 ON X+1 GOSUB 10,90,300,550
```

The ON - GOSUB statement provides a facility to allow BASIC11 to decide which of a number of subroutines to execute based on the value of an expression. When the expression is evaluated, the resulting number is used to pick one of the line numbers following the GOSUB it should execute. In the above example if X were equal to 0, the expression would evaluate to 1 and the subroutine starting at line 10 would be executed. If X were equal to 1, then the subroutine at line 90 would be executed and so on. If the expression evaluates to 0, a negative number or a number that is greater than the number of lines listed after the GOSUB, Error #32 (“ON” argument is Negative, Zero, or Too Large) will be issued.

ON GOTO <line number> ON <expression> GOTO <line number> [,<line number>,....<line number>]

```
500 ON X GOTO 100,200,300,400,500
```

The ON - GOTO statement works in basically the same manner as the ON - GOSUB except that control is transferred directly to the line number that is selected from the list following the GOTO. No return address is saved and hence control cannot be returned to the statement following the ON - GOTO statement.

3.3 Conditional Tests:

IF THEN <line number> IF <expression> THEN <line number>

```
55 IF A=1 THEN 200
70 IF A=1.AND.B=1 THEN 500
```

The IF - THEN statement is used to transfer control of the program to another statement based on the results of the evaluation of the expression. If the expression is true (evaluates to any non-zero value) then control is transferred to the statement at the line number following THEN. If the expression evaluates as false (equal to zero) then the next sequential statement in the program will be executed. Notice in the second example that multiple conditions may easily be tested in a single IF statement by use of the logical operators.

IF THEN ELSE

IF <expression> THEN <line number> ELSE <line number>

```
75 IF PORTA=$FE THEN 200 ELSE 300
```

This form of the IF - THEN statement is a slight variation in that if the expression is evaluated as false control of the program is transferred to the line number following the ELSE clause.

†NOTE: In the above examples a space follows the expression in the IF statement. This IS REQUIRED so that BASIC11 will know where the expression ends. Failure to follow the expression with a space will result in an Error being reported, most likely Error #6 (Illegal Operator).

3.4 Input/Output:

INPUT **<line number> INPUT [“string constant”,] <variable> [,<variable>,....<variable>]**

```
45 INPUT “ENTER THREE NUMBERS”,A,B,C
55 INPUT XE,ZE,PI
```

The input statement is one of the ways that a value may be assigned to a variable. When the INPUT statement is executed, the prompt string, if present, will be printed on the terminal followed by a question mark and will wait for the user to enter the requested data. If the user enters less data than is requested, BASIC11 will respond by printing a question mark on the next line and will wait for the next piece of data to be entered. This will continue until all requested data has been entered by the user. If more data is entered by the user than was requested by the INPUT statement, the excess will be ignored.

†NOTE that if the user responds to an INPUT statement with a “Alternate-C”, program execution will be halted and BASIC11 will return to the command mode. The program cannot be restarted by the use of the CONT command.

PRINT **<line number> PRINT [variable, expression, “string constant”]**

```
10 PRINT “THE VALUE OF X IS “; X
20 PRINT X,X*X,X/Z+3
30 PRINT X, Y, Z
35 PRINT A, B, C;
65 PRINT
```

The PRINT statement may be optionally followed by any combination of variables, expressions, or string constants each separated by either a comma or semicolon. The significance of separating the items in a PRINT statement by either a comma or a semicolon is explained below.

BASIC11 divides each output line into "fields" of eight (8) characters. When the arguments following a PRINT statement are separated by commas, BASIC11 will print each item beginning at the next field in the line. In line 30 in the above example, BASIC11 would print the value of variable X beginning in column 0, the value of variable Y would be printed starting in column 8 and the value of variable Z would be printed starting in column 16.

Separating variables with semicolons effectively disables this "fielding" feature by printing variables and constants next to one another. There will still be a space or two between successive numerical expressions that are printed because each number is printed with one trailing space. Also if a number is not negative a space will be printed in front of the number in place of the minus sign.

Notice in line number 35 above that a semicolon (it could have been a comma) follows the last variable. This has the effect of suppressing the normal carriage return/line feed sequence that would normally be issued after printing the last expression.

As mentioned in the first paragraph, the argument list that follows the PRINT statement is optional as is illustrated in the example of line 65 above. This form of the print statement has the effect of printing only a blank line.

? **<line number> ? [variable, expression, "string constant"]**

The question mark can be entered instead of the keyword "PRINT" to save typing time when entering a program or executing a line in the immediate mode. When entered in a program line the question mark is replaced by the same token as the keyword PRINT. Because of this, when the program line is listed the keyword PRINT will appear instead of the question mark.

INBYTE **<line number> INBYTE <variable>**

```
10 INBYTE DC
20 INBYTE AX(Z)
30 INBYTE CV
```

The INBYTE statement is another way that a value may be assigned to a variable. The INBYTE statement is similar to the input statement except that instead of expecting an ASCII formatted number from the terminal device, it assigns the value of an ASCII byte to the variable that follows it. If the statement in line 10 were executed and the character “Y” were typed at the terminal, the variable DC would contain the decimal value 89 which is the numerical value of the ASCII character “Y”.

3.5 Looping Constructs:

FOR **<variable>=<expression> TO <expression> [STEP<expression>]**

```
85 FOR X=1 TO 1000
90 FOR X=A TO B+C STEP 10
95 FOR X=100 TO 0 STEP -1
```

The FOR - NEXT statements are what is known as a deterministic looping construct because the number of times the loop will be executed is determined at the start of the loop when the FOR statement is executed. When a FOR statement is executed all instructions between it and the matching NEXT will repeatedly execute until one of two conditions is met. Each pass through the loop the STEP value is added to the value of the control variable. If the STEP value is positive, the loop will be executed again if the control variable is less than or equal to the value of the expression following TO. If the step value is negative the loop will be executed again if the control variable is greater than or equal to the value of the expression following the TO.

†NOTE: If no STEP value is supplied (it's optional) that a value of one (+1) is assumed.

†NOTE: All of the expressions in the FOR statement are evaluated only once at the start of the loop. This means that the terminating value and the step value may not be changed in the body of the loop, however; since the control variable is the same as any other variable, its value may be changed within the body of the loop. This would allow for exiting the loop before it normally would.

†NOTE: The test of the control variable against the terminating value is actually performed when the NEXT statement is executed, so <u>the code between FOR and NEXT will be executed at least once</u>.

FOR - NEXT statements may be nested but they must each use their own separate control variable. Currently the maximum number of nested FOR - NEXT loops is eight (8). Loops may be exited early by use of GOTO's however this is not good programming practice and is not recommended.

†NOTE: In the above examples a space follows each of the expressions in the FOR statement. This <u>IS REQUIRED</u> so that BASIC11 will know where the expression ends. Fail-

ure to follow each expression with a space will result in an Error being reported, most likely Error #6 (Illegal Operator).

NEXT **<line number> NEXT <variable>**

```
100 NEXT X
```

The NEXT statement is used in programs to complete a FOR loop. The variable specified in the NEXT statement must be the same as the control in the matching FOR. If it is not, Error #36 (Mismatched “FOR - NEXT” loop) will be issued and program execution will stop. As mentioned above, the test to see whether the loop should be terminated or not is actually performed when the NEXT statement is executed.

WHILE **<line number> WHILE <expression>**

```
500 WHILE X<=10000
```

The WHILE - ENDWH statements are considered to be a non-deterministic type of looping construct because the number of times the loop will execute is not determined at the start of the loop. In fact since the expression following the WHILE statement is evaluated at the start of the loop, the loop may never be executed if the expression is false (evaluates to zero) upon entry of the loop. There is one important point that needs to be made about the WHILE looping construct. The statements within the loop must contain a statement that changes the value of the test expression following WHILE so that the expression eventually becomes false otherwise the loop will never terminate and the statements bounded by WHILE and ENDWH will execute forever!

The WHILE statement may be used as part of a multiple statement line, however; in order to provide improved program readability and to show the structure of the program this practice is discouraged.

WHILE - ENDWH loops may be nested up to eight (8) levels deep. WHILE loops may be exited early by use of GOTO's however this is not good programming practice and is not recommended.

ENDWH **<line number> ENDWH**

```
600 ENDWH
```

The ENDWH statement is used only in conjunction with a matching WHILE statement to enclose a group of lines within a loop. The effect of the ENDWH statement is to evaluate the expression following WHILE to determine whether the loop should be executed again.

†NOTE The ENDWH statement may be part of a multi-statement line however, it must be the first statement on the line.

3.6 Program Termination:

STOP **<line number> STOP**

```
1000 STOP
```

The STOP statement is essentially a software break “Alternate-C” instruction. When the STOP statement is executed, program execution is temporarily suspended and the message:

```
STOPPED AT LINE # <line number>
```

is printed on the terminal. In the above example <line number> would be 1000. If no alterations are made to the program after it has been suspended, execution may be restarted with the CONT command. The first statement executed will be the one immediately following the STOP statement.

END **<line number> END**

```
300 END
```

The END statement is used to terminate program execution. It does not have to be the last statement and may appear anywhere in the program. In fact an end statement need not appear anywhere in the program. If BASIC11 tries to execute past the end of the program, an END statement will automatically be executed. Unlike the STOP statement, after an END statement has been executed the program may not be restarted via the CONT command.

3.7 Real Time Event Statements:

In any control environment, events usually occur asynchronously with respect to main program execution. To cope with this kind of environment the MC68HC11 was designed with an extensive interrupt structure to support all of its on chip subsystems. The following statements all provide control of interrupt driven events directly from BASIC11.

ONTIME <line number> ONTIME <expression>,<line number>

```
25 ONTIME 120,500
35 ONTIME HR+1,200
95 ONTIME 0,500
```

In many control situations it is necessary to take periodic measurements or record certain events at fixed time intervals. The ONTIME statement frees the main program from having to continuously check the value of the system variable TIME in order to determine when to take a measurement or record an event. The ONTIME statement allows program control to be transferred directly to an interrupt handling routine beginning at <line number> when the value of <expression> matches the value of the system variable TIME. The value of <expression> may evaluate to any legal integer, however; if <expression> evaluates to zero (0) it has the effect of disabling the ONTIME function.

One of two methods may be used to generate periodic interrupts using the ONTIME statement. The first method involves zeroing the system variable TIME in the interrupt handling routine with the statement TIME=0. This method may be used if continuous timekeeping is not required by the system. The second method involves executing the ONTIME statement in the interrupt routine, adding the desired time interval (in seconds) to the current value of the system variable TIME. This second method should be used if continuous timekeeping is required by the system. The following examples should clarify things.

First Method:

```
10 TIME=0
20 ONTIME 10,100
...
...
...
100 TIME=0
...
...
...
150 RETI
```

The above example will produce a timer interrupt every 10 seconds.

Second Method:

```
10 TIME =0
20 ONTIME 20,500
...
...
...
```

```
500 ONTIME TIME+20,500
...
...
...
550 RETI
```

The above example will produce a timer interrupt every 20 seconds.

ONIRQ **<line number> ONIRQ <expression>,<line number>**

```
10 ONIRQ 1,355
25 ONIRQ MD,225
```

The ONIRQ statement allows BASIC11 to directly handle interrupts that are generated by an active transition on the MC68HC11's IRQ pin. The <expression> following the ONIRQ keyword is used to select the mode of the statement. If the expression evaluates to any non-zero integer, the servicing of the IRQ interrupt by BASIC11 is enabled. If the expression evaluates to zero (0), IRQ interrupts are effectively disabled. The <line number> following the expression may be any legal BASIC11 line number.

ONPACC **<line number> ONPACC <expression>,<expression>,<line number>**

```
105 ONPACC 1,0,1000
255 ONPACC A,B,3000
```

The ONPACC statement allows the programmer to handle events associated with the MC68HC11's Pulse Accumulator on an interrupt basis. The first expression following the ONPACC keyword is used to set the operating mode of the pulse accumulator. The expression must evaluate to a number from 0 through 4. The operating modes of the pulse accumulator are described in the table below.

Mode	Action On Clock
0	Disables the Pulse Accumulator
1	Falling Edge on PA7 Increments the Counter
2	Rising Edge on PA7 Increments the counter
3	A “0” on PA7 Inhibits E/64 from Incrementing Counter
4	A “1” on PA7 Inhibits E/64 from Incrementing Counter

The second expression is used to determine which of two events will cause an interrupt to be generated by the pulse accumulator. If the expression evaluates to zero (0) then an interrupt will be generated each time an active edge is detected on PA7 as described in the table above. If it evaluates to 1, the pulse accumulator will generate an interrupt only when it overflows from $FF to $00. The <line number> tells BASIC11 where the interrupt routine begins when a Pulse Accumulator interrupt occurs.

For more information on the Pulse Accumulator subsystem, please refer to the MC68HC11's data sheet.

RETI **<line number> RETI**

```
485 RETI
```

All BASIC11 interrupt service routines must end with this statement. Failure to end an interrupt routine with RETI will result in all successive interrupts being masked! This will effectively stop the system TIME function.

SLEEP **<line number> SLEEP**

```
700 SLEEP
```

The SLEEP statement allows the MC68HC11 to be put into the 'Stop Mode' which is its lowest power consumption mode. In the “Stop Mode”, all clocks, including the internal oscillator, are stopped and all internal processing is halted. Recovery from the SLEEP statement may be accomplished by either a processor RESET or a XIRQ interrupt. When an XIRQ interrupt is used, BASIC11 will continue execution with the next BASIC program statement. When a hardware RESET is used to exit the sleep mode, the action taken by BASIC11 will depend on a couple of factors. If the “Auto Start” flag has been set with the AUTOST command, the BASIC program stored in external EEPROM/EPROM will automatically be executed. If the “Auto Start” flag has not been set, BASIC11 will return to the command mode.

3.8 Miscellaneous Statements:

DIM **<line number> DIM <subscripted variable> [,subscripted variable...]**

```
10 DIM AX(100),DX(9),LK(1000)
20 DIM Z(A+5),D(X)
30 DIM X(0)
```

The DIM statement, which was discussed briefly in section 1.4 on page 9, is used to allocate storage for subscripted variables when a program is run. As can be seen from the example in line 20 above, the expression in parenthesis does not have to be a constant. This is because array storage is dynamically allocated at runtime. This feature is especially nice in control applications where memory is usually at a premium because large arrays don't have to be dimensioned in advance to fit the worse case. All subscripted variables must appear in a DIM statement before they may be used in an expression. Failure to do this will result in Error # 24 (Undimensioned Array) being issued when the variable is referenced.

The storage required by subscripted integer variables is:

2*(<expression>+1)+2   bytes

Remember that all subscripts start at zero. In the example in line 10 above, the variable AX(100) would actually create 101 integer variables, AX(0) through AX(100). Although it may seem strange the example in line 30 is legal. This will create a single integer subscripted variable X(0).

POKE **<line number> POKE(<expression>,<expression>)**

```
45 POKE($6000,$5A)
55 POKE(AD,X*5)
```

The POKE statement allows the BASIC11 program to directly modify RAM memory or I/O locations ***not the external EEPROM***. The first expression within the parenthesis is the address at which the second expression will be stored. The first expression may evaluate to any legal integer number ($0000-$7FFF). However the second expression must be in the range 0 <= expression <= 255 since a byte location is being written to. If the second expression is outside the above range, Error #48 (Illegal Device Number). Care should be taken when using this statement so that part of the BASIC11 program or its data are not overwritten ***especially $0000-$00FF and $7CB1-$7FFF*** of the RAM as it is used by BASIC11 to store variables, stack area, and special routines.

†NOTE If POKE is used to write in the range $7CB1-$FFFF MicroNator will become unstable. It might be necessary to re-download BASIC11 again.

REM **<line number> REM [any text]**

```
10 REM THIS IS A REMARK STATEMENT
```

The REM statement is used to insert comments about the operation or structure of a program. Any text following the REM statement is ignored, so if it appears in a multiple statement line, it should be the last statement on the line. If control is passed to a REM statement by a GOTO GOSUB, etc., control is just passed to the line following the REM statement.

TRON <line number> TRON

```
20 TRON
```

The TRON statement is used to turn the trace mode on. The trace mode, when turned on, will print line numbers in brackets as each line is executed. This can be used as an aid in debugging programs.

TROFF <line number> TROFF

```
100 TROFF
```

The TROFF statement is used to turn the trace mode off.

CHAPTER 4

BUILT IN FUNCTIONS OF BASIC11

BASIC11 has a number of built in functions that are used to perform common operations on numerical quantities, perform special calculations, call user written assembly language subroutines, and access some of the special hardware features of the MC68HC11.

4.1 Mathematical Functions:

ABS(X)

The ABS function will return the ABSolute value of the expression in parenthesis. The function will always return a positive number as its result.

FDIV(X,Y)

The FDIV function is used to perform an unsigned fractional divide using the MC68HC11's FDIV instruction. This function allows BASIC11 to resolve fractional parts of the remainder of an integer divide without using floating point math. The result is a binary weighted decimal number. Some examples may clarify what the function does.

3 / 4 = .75 decimal	3 / 4 = $C000 binary weighted decimal
2 / 4 = .50 decimal	2 / 4 = $8000 binary weighted decimal
1 / 4 = .25 decimal	1 / 4 = $4000 binary weighted decimal
.99999... = $FFFF	

For the function to execute properly X must be less than Y and Y may not be equal to zero. If either condition exists Error #44 (Overflow or Divide by Zero in “FDIV()” Function) will be issued and program execution will terminate.

RND(X)

The RND function will return a pseudo random number between 0 and 32767 inclusive. The value of the argument X has the following effect on the function:

For X < 0 a new series of random numbers will be started by reading the current value of the timer/counter and using it as the new seed value.

For X = 0 a new random number will be returned each time the function is called.

For X > 0 the last random number that was generated is returned.

†NOTE that the function only generates pseudo random numbers and that a particular series will repeat every 65536 calls of the function.

SGN(X)

The SGN function will return a plus one (1) if the argument is positive, zero (0) if the argument is zero, and a minus one (-1) if the argument is negative.

4.2 Print Functions:

CHR$(X)

The CHR$ function will return a single character string whose ASCII value is the argument X. This function is very useful for sending non-printable ASCII characters to an output device. The value of the argument X must be in the range 0 <= X <= 255 or Error #43 (Argument < 0 or > 255 in “CHR$()” Function) will be issued. This function may only be used in the PRINT statement.

HEX(X)

The HEX function is used to convert a binary number to a four digit hexadecimal string. This function is very useful when printing the contents of memory locations or I/O ports. This function may only be used in the PRINT statement.

HEX2(X)

The HEX2 function performs a similar operation to the HEX function except that it is used to convert a number in the range 0 <= X <= 255 to a two digit hexadecimal string. If a number outside the specified range is passed as an argument to the HEX2 function, Error #50 (Argument < 0 or > 255 in “HEX2()” Function) will be reported.

TAB(X)

The TAB function will move the cursor to column X on the output device. If the output device is already past column X then no action is performed. The argument to the TAB function must be in the range 0 <= X <= 255 or Error # 42 (Argument < 0 or > 255 in “TAB()” Function) will be issued. This function may only be used in the PRINT statement.

4.3 Hardware Related Functions:

ADC(X)

The ADC function allows a program to directly access the MC68HC11's on board 8-bit A-to-D converter. Any one of the eight channels may be read by calling the function with the proper argument. If the argument is not in the proper range (between 0 and 7) Error #45 (Invalid Channel Number in “ADC()” Function) will be issued. The A-to-D converter is operated in the single channel mode and is converted four times. These four conversions are then averaged by BASIC11 and the result is then returned. Since the A-to-D conversion time is fast (26µs at 1.2290 MHz or 16µs at 2.0 MHz) this tends to help average out any noise in the reading.

CALL(X)

Even though BASIC11 is extremely fast for an interpreted BASIC, there are still some things that may need to be controlled that it can't keep up with. The CALL function allows machine language subroutines to be called directly from BASIC11. The CALL function must appear in an expression since it will return a 16-bit number as a result of the function call. Some examples follow:

```
10 F=CALL($EAF0)
20 Z=CALL(AX*2)
30 PRINT CALL($100)
```

The users machine language program must only preserve the Y-index register, the stack pointer, and the current state of the stack. All other registers may be destroyed. The user’s subroutine is entered via a JSR (Jump to SubRoutine) instruction, therefore <u>it must end with the execution of an RTS</u> (ReTurn from Subroutine) instruction. Generally the user’s subroutine should have about 100 bytes of stack space available. If more than this is needed, the subroutine will have to allocate its own stack storage space.

This is where MicroNator comes in action. MicroNator with the help of “Alternate-L” is able to download from the PC any assembler program in the S0-S9 format anywhere into the external EEPROM or external RAM. Refer to MicroNator user manual for the “Alternate-L” function.

†NOTE: The “Alternate-L” function issues a RESET after the download so use it before entering your BASIC11 program in RAM because RESET erases all the RAM area.

†NOTE: Make sure you don’t erase part of the BASIC11 program.

†NOTE: $D000-$DDFF is reserved for the user to place his routines.

EEP(X)

As mentioned in section earlier the EEP statement allows a BASIC11 program to directly write a “WORD” of information to the MC68HC11’s external EEPROM when the EEP statement appears to the left of the equal as a basic “statement”. When EEP appears on the right side of the equals it will act like a function and will return the “BYTE” value currently stored in the location specified. It is identical to the PEEK(X).

Although X can be any location from $0000 to $DDFF, it is recommended to use it in the range $00-$FF.

PEEK(X)

The PEEK function performs the opposite action of the POKE function. It allows BASIC11 to directly retrieve the contents of any memory or I/O location in the MC68HC11's memory map. The argument X, since it is an address, is taken to be an unsigned number so X may take on any integer value. A single byte is returned by the function so its value will be >= 0 and <= 255.

PORTA
PORTB
PORTC
PORTD
PORTE

The PORTx functions are different from the other functions in that they do not require an argument. Essentially these functions act as special variables that allow direct reading of the MC68HC11's I/O ports from BASIC.

PORTC and PORTD are general purpose I/O ports and as such may have each pin of the port programmed as either an input or an output. Each ports Data Direction Register (DDR) is used to specify the primary direction of data on the I/O pin. If the corresponding port pins DDR bit is set to a one (1) the port pin will be configured as an output. If the DDR bit is cleared to a zero (0) the port pin will be configured as an input and will become high impedance. When a bit which is configured for output is read, the value returned is the value at the input to the pin driver. If a write is executed to a pin that is configured as an input, the value will be stored in an internal latch so that if the pin is later configured as an output, the latched value will then appear on the output

PORTA, PORTB, and PORTE are all fixed direction Ports with the exception of bit-7 of Port A. When PORTB is being used for general purpose outputs, it is configured for output only and reads return the actual level sensed at the input of the pin drivers. When PORTA is being used for general purpose I/O, bits 0,1, and 2 are configured as inputs and writes to these bits have no effect or meaning. Bits 3, 4, 5, and 6 are configured for output only and reads return the actual level sensed at the input of the pin drivers. Bit 7 of PORTA can be configured as either an input or an output via the DDRA7 bit in the PORTA control register (PACTL). PORTE contains the eight inputs to the A-to-D converter, however they may also be used as digital inputs. Writes to the PORTE address have no meaning or effect.

For a more complete discussion of the function of the I/O subsystems contained in the MC68HC11, it is suggested that the parts data sheet be consulted.

†NOTE: Please take notice that PORTB and PORTC are taken by data and address in MicroNator multiplex mode. Port PD2..PD5 are used for the SPI communication but can be used for other purposes.

TIME

Like the PORTx functions, the TIME function requires no arguments and is used to retrieve the current value of the system time.

PACC

When the keyword PACC appears to the right of the equals sign it allows the program to retrieve the current value of the Pulse Accumulator. Effectively PACC is a function that requires no arguments.

CHAPTER 5

ERROR REPORTING OF BASIC11

BASIC11 has an extensive error reporting structure that reports two basic types of errors. The first category is command line errors. If a mistake is made by either typing an illegal command or a syntax error is detected either in a program line or a statement that is to be executed in the direct mode, BASIC11 will print the contents of the input buffer. On the next line asterisks and arrows will be printed showing the approximate location of the error within the line. Finally, a number is printed telling the operator what is wrong with the line. In the example shown below programmer input is underlined.

```
#10 FOR X=1 100 STEP 2
10 FOR X=1 100 STEP 2
*********^^^
ERROR #17


READY

#
```

Looking up error #17 in the error table we find that we have inadvertently left out the “TO” in the FOR statement. By retyping the line with “TO” between the 1 and 100 BASIC11 will accept the line.

When the programmer mistypes a command, Error number 3 (Invalid Expression) will generally be issued. An example follows.

```
#LOST (what the programmer meant to type was LIST)
LOST
*^^^
ERROR #3

READY

#
```

The reason error number 3 is issued is that BASIC11 first searches its command table to see if the programmer has typed a command. If no match is found, BASIC11 then searches its statement table to try to match the input buffer with one of the keywords. If no match is found, BASIC11 assumes that the statement is an implied LET. In the above example the first two letters, “LO”, would be assumed to be a variable name, and the rules say that in an implied (or declared) LET the assignment variable must be immediately followed by an equals (“=”).

The second category of errors is runtime errors. These errors, which are context dependent, occur while the program is running. All runtime errors are considered to be fatal in BASIC11 and will immediately terminate program execution. A message will be printed on the terminal indicating what error occurred and in which line it occurred. Even though BASIC11 does not list the source line for runtime errors, the error number is specific enough that the problem can easily be identified.

A list of error numbers and their meanings follows.

Error #	**Meaning**
1	Line number < 0 or > 32767
2	Syntax Error
3	Invalid Expression
4	Unbalanced Parenthesis
5	Data Type Mismatch
6	Illegal Operator
7	Illegal Variable
8	Illegal Token
9	Out of Memory
10	Integer Overflow
11	Invalid Hex Digit
12	Hex Number Overflow
13	Missing Quote
14	Missing Open or Closing Parenthesis
15	Syntax Error in “ON” Statement
16	Missing “THEN” in an “IF” Statement
17	Missing “TO” in a “FOR” Statement
18	Line Number Zero (0) Not Allowed
19	Illegal Data Type
20	Expression Too Complex
21	Missing Comma
22	Missing Comma or Semicolon
23	Math Stack Overflow
24	Undimensioned Array
25	Subscript Out of Range
26	Divide By Zero
27	Line Number Not Found
28	Too Many Nested “GOSUB's” (maximum is eight)
29	“RETURN” without “GOSUB”

30	Too Many Active “WHILE's” (maximum is eight)
31	“ENDWH” without “WHILE”
32	“ON” argument is Negative, Zero, or Too Large
33	Non-subscriptable Variable Found in “DIM” statement
34	Variable has Already Been DIMensioned
35	Too Many Active “FOR - NEXT” loops (maximum is eight)
36	Mismatched “FOR - NEXT” loop
37	Can't Continue
38	Out of Data in “READ” or “RESTORE” Statement
39	Negative Subscripts Not Allowed
40	“EEP()” Subscript Negative or > 255
41	Function Only Allowed in “PRINT” Statement
42	Argument < 0 or > 255 in “TAB()” Function
43	Argument < 0 or > 255 in “CHR$()” Function
44	Overflow or Divide by Zero in “FDIV()” Function
45	Invalid Channel Number in “ADC()” Function
46	Tried to Assign a Value of < 0 or > 255 to a PORT
47	Illegal PORT
48	Illegal Device Number
49	Uninitalized I/O Vector
50	Argument < 0 or > 255 in “HEX2()” Function
51	Statement not allowed in immediate mode
52	RETI executed when not in an interrupt routine
53	Tried to assign a value of <0 or >255 to PACC
54	Interrupt or Count mode error in ONPACC
55	Program storage EEPROM is too small
56	EEPROM range not legal to be written by user

APPENDIX A

Interrupt Vector Table:

All twenty of the interrupt vectors for the different subsystems on the MC68HC11 are located in the memory map at locations $FFD6 through $FFFF. To provide for more flexibility in using the subsystems in an interrupt driven mode, the EEPROM hardware vectors “point” to a second “JUMP” vector table located in RAM on page zero. The table, as shown below, may be altered by the programmer to point to special interrupt handlers for a particular application. The PACCIE, PACCOVF, TOC1, and IRQI vectors are initialized by BASIC11 to point to its own interrupt routines for the various real time control functions provided by BASIC11. The ILLOP, COP, and CMF vectors are initialized to jump to the start of BASIC11. All the rest of the vectors point to an RTI instruction.

TABLE: 1 “JUMP” VECTOR TABLE LOCATED IN RAM ON PAGE ZERO

```
009e                ORG       $009E
           *
009e       CONSTAT  RMB       3           GET CONSOLE STATUS FOR BREAK ROUTINE.
00a1       INCONNE  RMB       3           GET BYTE DIRECTLY FROM CONSOLE FOR BREAK RTN.
           *
00a4                ORG       $00A4
           *
00a4       INTABLE  RMB       16          RESERVE SPACE FOR 8 DIFFERENT INPUT ROUTINES.
00b4       OUTABLE  RMB       16          RESERVE SPACE FOR 8 DIFFERENT OUTPUT ROUTINES.
           *
           *
           *
           *
00c4                ORG       $00C4       START OF RAM INTERRUPT VECTORS.
           *
00c4       RAMVECTS EQU       *
00c4       SCISS    RMB       3           SCI SERIAL SYSTEM.
00c7       SPITC    RMB       3           SPI TRANSFER COMPLETE.
00ca       PACCIE   RMB       3           PULSE ACCUMULATOR INPUT EDGE.
00cd       PACCOVF  RMB       3           PULSE ACCUMULATOR OVERFLOW.
00d0       TIMEROVF RMB       3           TIMER OVERFLOW.
00d3       TOC5     RMB       3           TIMER OUTPUT COMPARE 5.
00d6       TOC4     RMB       3           TIMER OUTPUT COMPARE 4.
00d9       TOC3     RMB       3           TIMER OUTPUT COMPARE 3.
00dc       TOC2     RMB       3           TIMER OUTPUT COMPARE 2.
00df       TOC1     RMB       3           TIMER OUTPUT COMPARE 1.
00e2       TIC3     RMB       3           TIMER INPUT CAPTURE 3.
00e5       TIC2     RMB       3           TIMER INPUT CAPTURE 2.
00e8       TIC1     RMB       3           TIMER INPUT CAPTURE 1.
00eb       REALTIMI RMB       3           REAL TIME INTERRUPT.
00ee       IRQI     RMB       3           IRQ INTERRUPT.
00f1       XIRQ     RMB       3           XIRQ INTERRUPT.
00f4       SWII     RMB       3           SOFTWARE INTERRUPT.
00f7       ILLOP    RMB       3           ILLEGAL OPCODE TRAP.
00fa       COP      RMB       3           WATCH DOG TIMER FAIL.
00fd       CMF      RMB       3           CLOCK MONITOR FAIL.
```

TABLE: 2 BASIC11 EEPROM HARDWARE INTERRUPT VECTOR

```
ffd6                   ORG      $FFD6      START OF VECTOR TABLE.
ffd6 00 c4             FDB      SCISS      SCI SERIAL SYSTEM
ffd8 00 c7             FDB      SPITC      SPI TRANSFER COMPLETE
ffda 00 ca             FDB      PACCIE     PULSE ACCUMULATOR INPUT EDGE
ffdc 00 cd             FDB      PACCOVF    PULSE ACCUMULATOR OVERFLOW
ffde 00 d0             FDB      TIMEROVF   TIMER OVERFLOW
ffe0 00 d3             FDB      TOC5       TIMER OUTPUT COMPARE 5
ffe2 00 d6             FDB      TOC4       TIMER OUTPUT COMPARE 4
ffe4 00 d9             FDB      TOC3       TIMER OUTPUT COMPARE 3
ffe6 00 dc             FDB      TOC2       TIMER OUTPUT COMPARE 2
ffe8 00 df             FDB      TOC1       TIMER OUTPUT COMPARE 1
ffea 00 e2             FDB      TIC3       TIMER INPUT CAPTURE 3
ffec 00 e5             FDB      TIC2       TIMER INPUT CAPTURE 2
ffee 00 e8             FDB      TIC1       TIMER INPUT CAPTURE 1
fff0 00 eb             FDB      REALTIMI   REAL TIME INTERRUPT
fff2 00 ee             FDB      IRQI       IRQ INTERRUPT
fff4 00 f1             FDB      XIRQ       XIRQ INTERRUPT
fff6 00 f4             FDB      SWII       SOFTWARE INTERRUPT
fff8 00 f7             FDB      ILLOP      ILLEGAL OPCODE TRAP
fffa 00 fa             FDB      COP        WATCH DOG FAIL
fffc 00 fd             FDB      CMF        CLOCK MONITOR FAIL
fffe ec 50             FDB      POWERUP    RESET
```

APPENDIX B

TABLE: 3 “BASIC11” Memory Map

```
        *       /*********** define variables ***********/

0000            ORG    $0000
        *
        *              char
        *
0000    IBUFPTR  RMB    2        /* input buffer pointer */
0002    TBUFPTR  RMB    2        /* token buffer pointer */
        *
        *  the next 5 variables must remain grouped together
        *
0004    BASBEG   RMB    2        /* start of basic program area */
0006    BASEND   RMB    2        /* end of basic program */
0008    VARBEGIN RMB    2        /* start of variable storage area */
000a    VAREND   RMB    2        /* end of variable storage area */
000c    HILINE   RMB    2        /* highest line number in program buffer */
        *
        *
        *
000e    BASMEND  RMB    2        /* physical end of basic program memory */
0010    VARMEND  RMB    2        /* physical end of variable memory */
        *
        *              int
        *
0012    FIRSTLIN RMB    2        /* first line to list */
0014    LASTLIN  RMB    2        /* last line to list */
0016    INTPTR   RMB    2        /* integer pointer */
        *
        *              short
        *
0018    ERRCODE  RMB    1        /* error code status byte */
0019    IMMID    RMB    1        /* immediate mode flag */
001a    BREAKCNT EQU    *        /* also use for break check count */
001a    COUNT    EQU    *        /* count used in ESAVE & ELOAD routines */
001a    IFWHFLAG RMB    1        /* translating IF flag */
001b    TRFLAG   RMB    1        /* trace mode flag */
001c    CONTFLAG RMB    1        /* continue flag */
001d    RUNFLAG  RMB    1        /* indicates we are in the run mode */
001e    PRINTPOS RMB    1        /* current print position */
001f    NUMSTACK RMB    2        /* numeric operand stack pointer */
0021    OPSTACK  RMB    2        /* operator stack pointer */
0023    FORSTACK RMB    2        /* FOR stack pointer */
0025    WHSTACK  RMB    2        /* WHILE stack pointer */
0027    GOSTACK  RMB    2        /* GOSUB stack pointer */
0029    CURLINE  RMB    2        /* line # that we are currently interpreting */
002b    ADRNXLIN RMB    2        /* address of the next line */
002d    STRASTG  RMB    2        /* dynamic string/array pool pointer */
002f    FENCE    RMB    2        /* varend in case of an error in xlation */
0031    IPSAVE   RMB    2        /* interpretive pointer save for “BREAK” */
0033    DATAPTR  RMB    2        /* pointer to data for read statement */
0035    RANDOM   RMB    2        /* random number/seed */
0037    DEVNUM   RMB    1        /* I/O device number */
0038    TIMEREG  RMB    2        /* TIME register */
003a    TIMECMP  RMB    2        /* TIME compare register */
```

```
003c     TIMEPRE  RMB    1        /* software prescaler for TIME */
003d     ONTIMLIN RMB    2        /* ONTIME line number to goto */
003f     ONIRQLIN RMB    2        /* ONIRQ line number to goto */
0041     ONPACLIN RMB    2        /* ONPACC line number to goto */
0043     XONCH    RMB    1        /* XON character for printer */
0044     XOFFCH   RMB    1        /* XOFF character for printer */
0045     SCURLINE RMB    2        /* to save CURLINE during int. process */
0047     SADRNXLN RMB    2        /* to save ADRNXLIN during int. process */
0049     INBUFFS  RMB    2        /* ptr to the start of the input buffer */
004b     TKNBUFS  RMB    2        /* ptr to the start of the token buffer */
004d     EOPSTK   RMB    2        /* end of operator stack */
004f     STOPS    RMB    2        /* start of operator stack */
0051     ENUMSTK  RMB    2        /* end of operand stack */
0053     STNUMS   RMB    2        /* start of operand stack */
0055     EFORSTK  RMB    2        /* end of FOR - NEXT stack */
0057     STFORSTK RMB    2        /* start of FOR - NEXT stack */
0059     EWHSTK   RMB    2        /* end of WHILE stack */
005b     STWHSTK  RMB    2        /* start of WHILE stack */
005d     EGOSTK   RMB    2        /* end of GOSUB stack */
005f     STGOSTK  RMB    2        /* start of GOSUB stack */
0061     IOBaseV  RMB    2        /* Address vector for I/O Registers */
0063     DNAME    RMB    3        /* to put the var name when doing a dump */
0066     SUBMAX   RMB    2        /*  */
0068     SUBCNT   RMB    2        /*  */
006a     TOKPTR   RMB    2        /* token pointer (used for list command) */
006c     VarSize  RMB    2        /* size of the variable table */
         *
         *+++++   if *>$9E
         *+++++   error  “Ran out of Page 0 RAM”
         *+++++   endif
         *
009e              ORG    $009E
         *
009e     CONSTAT  RMB    3        GET CONSOLE STATUS FOR BREAK ROUTINE.
00a1     INCONNE  RMB    3        GET BYTE DIRECTLY FROM CON FOR BREAK RTN
         *
00a4              ORG    $00A4
         *
00a4     INTABLE  RMB    16       RESERVE SPACE FOR 8 INPUT ROUTINES.
00b4     OUTABLE  RMB    16       RESERVE SPACE FOR 8 OUTPUT ROUTINES.
         *
00c4              ORG    $00C4    START OF RAM INTERRUPT VECTORS.
         *
00c4     RAMVECTS EQU    *
00c4     SCISS    RMB    3        SCI SERIAL SYSTEM.
00c7     SPITC    RMB    3        SPI TRANSFER COMPLETE.
00ca     PACCIE   RMB    3        PULSE ACCUMULATOR INPUT EDGE.
00cd     PACCOVF  RMB    3        PULSE ACCUMULATOR OVERFLOW.
00d0     TIMEROVF RMB    3        TIMER OVERFLOW.
00d3     TOC5     RMB    3        TIMER OUTPUT COMPARE 5.
00d6     TOC4     RMB    3        TIMER OUTPUT COMPARE 4.
00d9     TOC3     RMB    3        TIMER OUTPUT COMPARE 3.
00dc     TOC2     RMB    3        TIMER OUTPUT COMPARE 2.
00df     TOC1     RMB    3        TIMER OUTPUT COMPARE 1.
00e2     TIC3     RMB    3        TIMER INPUT CAPTURE 3.
00e5     TIC2     RMB    3        TIMER INPUT CAPTURE 2.
00e8     TIC1     RMB    3        TIMER INPUT CAPTURE 1.
00eb     REALTIMI RMB    3        REAL TIME INTERRUPT.
00ee     IRQI     RMB    3        IRQ INTERRUPT.
00f1     XIRQ     RMB    3        XIRQ INTERRUPT.
00f4     SWII     RMB    3        SOFTWARE INTERRUPT.
```

```
00f7     ILLOP    RMB    3        ILLEGAL OPCODE TRAP.
00fa     COP      RMB    3        WATCH DOG TIMER FAIL.
00fd     CMF      RMB    3        CLOCK MONITOR FAIL.
         *
         *+++++      RAMStart = $1040
         *+++++      RAMSize  = $6EBF    (EESTART - RAMStart - $0100 -1)
         *+++++      PROGRAM  = $7CB0    (RAMStart + RAMSize - SWSTKSize + 1)
         *
1040     RAMStart    EQU     $1040       Start of the RAM
7cb0     LASTUSER    EQU     (RAMStart+RAMSize-SWSTKSize-1)
         *                               Highest possible byte for user program
         *
         *      /*********** The rest of the RAM is reserved for BASIC11 ***********/
         *
7cb1     LOSTACK     EQU     (RAMStart+RAMSize-SWSTKSize)
         *                               Lowest possible byte for any stack
7eff     HISTACK     EQU     (RAMStart+RAMSize)
         *                               Highest possible byte for any stack
         *
         *      /*********** The last $0100 bytes reserved for BASIC11 special routine *****/

0100     SPECIAL     EQU     $0100       Special routines
7f00     RAMSAVE     EQU     (RAMStart+RAMSize+1)
7f03     NEWESAVE    EQU     RAMSAVE+3
7f5d     NEWDLY      EQU     RAMSAVE+$5D
7f68     WXOFFRAM    EQU     RAMSAVE+$68
7f73     RAUTOSTF    EQU     RAMSAVE+$73
7fff     LASTBAS     EQU     EESTART-1     Last RAM byte used by BASIC11
         *
         *      /*********** Beginning of EEPROM ***********/
         *
8000                 ORG     EESTART     Beginning of EEPROM
8000     SBASBEG     RMB     2           pointer for start of basic program area
8002     SBASEND     RMB     2           pointer for end of basic program
8004     SVARBEG     RMB     2           pointer for start of variable storage area
8006     SVAREND     RMB     2           pointer for end of variable storage area
8008     SHILINE     RMB     2           pointer for highest line number in program buffer
800a     AUTOSTF     RMB     1           autostart flag
800b     SSTART      RMB     1           storage start
         *
d000     CALLBEG     EQU     EESTART+$5000 User assembler call subroutine storage
ddfe     MAXEESUB    EQU     ROMBEG-2    maximum EEP subscript
ddff     CALLEND     EQU     ROMBEG-1    End of user assembler call subroutine storage
de00     ROMBEG      EQU     $DE00       Begin of BASIC11
2200     ROMSIZE     EQU     $2200

ffd6 00 c4           FDB     SCISS       SCI SERIAL SYSTEM
ffd8 00 c7           FDB     SPITC       SPI TRANSFER COMPLETE
ffda 00 ca           FDB     PACCIE      PULSE ACCUMULATOR INPUT EDGE
ffdc 00 cd           FDB     PACCOVF     PULSE ACCUMULATOR OVERFLOW
ffde 00 d0           FDB     TIMEROVF    TIMER OVERFLOW
ffe0 00 d3           FDB     TOC5        TIMER OUTPUT COMPARE 5
ffe2 00 d6           FDB     TOC4        TIMER OUTPUT COMPARE 4
ffe4 00 d9           FDB     TOC3        TIMER OUTPUT COMPARE 3
ffe6 00 dc           FDB     TOC2        TIMER OUTPUT COMPARE 2
ffe8 00 df           FDB     TOC1        TIMER OUTPUT COMPARE 1
ffea 00 e2           FDB     TIC3        TIMER INPUT CAPTURE 3
ffec 00 e5           FDB     TIC2        TIMER INPUT CAPTURE 2
ffee 00 e8           FDB     TIC1        TIMER INPUT CAPTURE 1
fff0 00 eb           FDB     REALTIMI    REAL TIME INTERRUPT
fff2 00 ee           FDB     IRQI        IRQ INTERRUPT
```

```
fff4 00 f1          FDB     XIRQ        XIRQ INTERRUPT
fff6 00 f4          FDB     SWII        SOFTWARE INTERRUPT
fff8 00 f7          FDB     ILLOP       ILLEGAL OPCODE TRAP
fffa 00 fa          FDB     COP         WATCH DOG FAIL
fffc 00 fd          FDB     CMF         CLOCK MONITOR FAIL
fffe ec 50          FDB     POWERUP     RESET
```

TABLE: 4 "BASIC11" MEMORY MAP

Address	Description
\$0000 - \$00FF	BASIC11 RAM for variables
\$0100 - \$01FF	Other HC11 internal RAM
\$0200	Reserved 16 bytes
\$0210	Reserved 16 bytes for LCD & KBY expansion board
\$0220	Reserved 16 bytes for UIO expansion board
\$0230	Reserved 16 bytes for GAL programmer expansion board
\$0240 - \$027F	Spare Chip Select for WW area
\$0280 - \$02BF	If READ enables (HIGH) the RTC chip select for SPI If WRITEn disabled (LOW) the RTC chip select for SPI
\$02C0 - \$0FFF	Reserved by 16 bytes increment for future expansion and I/O
\$1000 - \$103F	HC11 registers
\$1040 - \$7CB0	Used by BASIC11 to store USER program in RAM
\$7CB1 - \$7EFF	Used by BASIC11 for Stack area
\$7F00 - \$7FFF	Special routine for BASIC11
\$8000 - \$800A	Start of EEPROM The first 10 bytes are used for storage pointers
\$800B	Used by BASIC11 to store USER program in EEPROM with ESAVE
\$D000	Free to USER to store assembler routines
\$DDFE	MAX EEP() subscript
\$DDFF	End of user assembler routines
\$DE00 - \$FFD5	BASIC11 interpreter
\$FFD6 - \$FFFF	Vector table

TABLE: 5 "MicroNator" Reserved Memory

ADDRESS	**DESCRIPTION**
\$0000-\$00ff	Not used by MicroNator, free for the user or used by BASIC11
\$0100-\$01FF	Other HC11 internal RAM, i.e. HC11E0, HC11E1, HC11E8...
\$0200-\$020F	Reserved
\$0210-\$021F	LCD & KBY expansion board
\$0220-\$022F	UIO (Relays & Opto couplers) expansion board
\$0230-\$023F	GAL Programmer expansion board

$0240-$027F	SPARE chip select for WW
$0280-$02BF	*** If Read, enables (HIGH) the **RTC** chip select for SPI
	*** If Written, disables (LOW) the **RTC** chip select for SPI
$02C0-$0FFF	Reserved, by 16 bytes increment, for future expansion and I/O

TABLE: 6 MOTOROLA ASSEMBLER (AS11.EXE) HC11 REGISTERS

```
*ADDR LABEL        DEFINITIONS

1000   REGS    EQU    $1000

0000   PORTA   EQU    0           PORT A DATA REGISTER
0001   RESVD   EQU    1           UNUSED
0002   PIOC    EQU    2           PARALLEL I/O CONTROL REGISTER
       *                          STROBE A FLAG
       *                          0= INACTIVE
       *                          1= SET AT ACTIVE EDGE OF STRA PIN
0080   STAF    EQU    %10000000
       *                          STROBE A INTERRUPT ENABLE
       *                          0= NO HARDWARE INTERRUPT GENERATED
       *                          1= HARDWARE INTERRUPT REQ WHEN STAF=1
0040   STAI    EQU    %01000000
       *                          PORT C WIRE-OR MODE
       *                          0= PORT C OUTPUTS NORMAL
       *                          1= OPEN DRAIN
0020   CWOM    EQU    %00100000
       *                          HANDSHAKE/SIMPLE STROBE MODE SELECT
       *                          0= SIMPLE STROBE MODE
       *                          1= FULL HANDSHAKE MODES
0010   HNDS    EQU    %00010000
       *                          OUTPUT/INPUT HANDSHAKE SELECT
       *                          0= INPUT
       *                          1= OUTPUT
0008   OIN     EQU    %00001000
       *                          PULSE MODE SELECT FOR STRB OUTPUT
       *                          0= STRB LEVEL ACTIVE
       *                          1= STRB PULSES
0004   PLS     EQU    %00000100
       *                          ACTIVE EDGE SELECT FOR STRA
       *                          0= HI TO LO (FALLING)
       *                          1= LO TO HI (RISING)
0002   EGA     EQU    %00000010
       *                          INVERT STRB OUTPUT
       *                          0= STRB ACTIVE LOW
       *                          1= STRB ACTIVE HIGH
0001   INVB    EQU    %00000001
       *
0003   PORTC   EQU    3           PORT C DATA REGISTER
0004   PORTB   EQU    4           PORTB DATA REGISTER
0005   PORTCL  EQU    5           PORT C LATCHED DATA REGISTER
0006   RESVD1  EQU    6           UNUSED
0007   DDRC    EQU    7           DATA DIRECTION REGISTER FOR PORT C
0008   PORTD   EQU    8           PORT D DATA REGISTER
0009   DDRD    EQU    9           DATA DIRECTION REGISTER FOR PORT D
000a   PORTE   EQU    $A          PORT E DATA REGISTER
000b   CFORC   EQU    $B          TIMER COMPARE FORCE REGISTER
0080   FOC1    EQU    %10000000
0040   FOC2    EQU    %01000000
0020   FOC3    EQU    %00100000
```

```
0010    FOC4    EQU    %00010000
0008    FOC5    EQU    %00001000
000c    OC1M    EQU    $C           OUTPUT COMPARE 1 MASK REGISTER
0080    OC1M7   EQU    %10000000
0040    OC1M6   EQU    %01000000
0020    OC1M5   EQU    %00100000
0010    OC1M4   EQU    %00010000
0008    OC1M3   EQU    %00001000
000d    OC1D    EQU    $D           OUTPUT COMPARE 1 DATA REGISTER
0080    OC1D7   EQU    %10000000
0040    OC1D6   EQU    %01000000
0020    OC1D5   EQU    %00100000
0010    OC1D4   EQU    %00010000
0008    OC1D3   EQU    %00001000
000e    TCNT    EQU    $E           TIMER COUNTER REGISTER (2 BYTES)
0010    TIC1    EQU    $10          TIMER INPUT CAPTURE REGISTERS (3 REGS, 6 BYTES
0012    TIC2    EQU    $12
0014    TIC3    EQU    $14
        *
0016    TOC1    EQU    $16          TIMER OUTPUT COMPARE REGISTERS (5 REGS, 10 BYTES)
0018    TOC2    EQU    $18
001a    TOC3    EQU    $1A
001c    TOC4    EQU    $1C
001e    TOC5    EQU    $1E
0020    TCLT1   EQU    $20          TIMER CONTROL REGISTER 1
        *                           OMx  OLx  ACTION UPON SUCCESSFUL COMPARE
        *                           0    0    TIMER DISC FROM OUTPUT PIN
        *                           0    1    TOGGLE OCx OUTPUT LINE
        *                           1    0    CLEAR OCx OUTPUT LINE TO ZERO
        *                           1    1    SET OCx OUTPUT LINE TO ONE
0080    OM2     EQU    %10000000
0040    OL2     EQU    %01000000
0020    OM3     EQU    %00100000
0010    OL3     EQU    %00010000
0008    OM4     EQU    %00001000
0004    OL4     EQU    %00000100
0002    OM5     EQU    %00000010
0001    OL5     EQU    %00000001
0021    TCLT2   EQU    $21          TIMER CONTROL REGISTER 2
        *                           EDGxB EDGxA  CONFIGURATION
        *                           0    0       CAPTURE DISABLED
        *                           0    1       CAPTURE ON RISING EDGES ONLY
        *                           1    0       CAPTURE ON FALING EDGES ONLY
        *                           1    1       CAPTURE ON ANY EDGE (RISING OR FALLING)
0020    EDG1B   EQU    %00100000
0010    EDG1A   EQU    %00010000
0008    EDG2B   EQU    %00001000
0004    EDG2A   EQU    %00000100
0002    EDG3B   EQU    %00000010
0001    EDG3A   EQU    %00000001
0022    TMSK1   EQU    $22          MAIN TIMER INTERRUPT MASK REG 1
0080    OC1I    EQU    %10000000
0040    OC2I    EQU    %01000000
0020    OC3I    EQU    %00100000
0010    OC4I    EQU    %00010000
0008    OC5I    EQU    %00001000
0004    IC1I    EQU    %00000100
0002    IC2I    EQU    %00000010
0001    IC3I    EQU    %00000001
0023    TFLG1   EQU    $23          MAIN TIMER INTERRUPT FLAG REG 1
0080    OC1F    EQU    %10000000
```

```
0040    OC2F    EQU    %01000000
0020    OC3F    EQU    %00100000
0010    OC4F    EQU    %00010000
0008    OC5F    EQU    %00001000
0004    IC1F    EQU    %00000100
0002    IC2F    EQU    %00000010
0001    IC3F    EQU    %00000001
0024    TMSK2   EQU    $24         MISC TIMER INTERRUPT MASK REG 2
0080    TOI     EQU    %10000000   TIMER OVERFLOW INTERRUPT ENABLE
0040    RTII    EQU    %01000000   RTI INTERRUPT ENABLE
0020    PAOVI   EQU    %00100000   PULSE ACCUMULATOR OVERFLOW INTERRUPT ENABLE
        *                          PULSE ACCUMULATOR INPUT INTERRUPT ENABLE
        *                          0= INTERRUPT INHIBITED
        *                          1= INTERRUPT REQUESTED IF FLAG SET
0010    PAII    EQU    %00010000
        *                          PR1  PR2  PRESCALE FACTOR
        *                           0    0          1
        *                           0    1          4
        *                           1    0          8
        *                           1    1          16
0002    PR1     EQU    %00000010
0001    PR0     EQU    %00000001
0025    TFLG2   EQU    $25         MISC TIMER INTERRUPT FLAG REG 2
0080    TOF     EQU    %10000000   TIMER OVERFLOW FLAG
0040    RTIF    EQU    %01000000   REAL TIME (PERIODIC) INTERRUPT FLAG
0020    PAOVF   EQU    %00100000   PULSE ACCUMULATOR OVERFLOW FLAG
0010    PAIF    EQU    %00010000   PULSE ACCUMULATOR INPUT EDGE FLAG
0026    PACTL   EQU    $26         PULSE ACCUMULATOR CONTROL REGISTER
        *                          DATA DIRECTION FOR PA7
        *                          0= INPUT
        *                          1= OUTPUT
0080    DDRA7   EQU    %10000000
        *                          PULSE ACCUMULATOR SYSTEM ENABLE
        *                          0= DISABLED
        *                          1= ENABLED
0040    PAEN    EQU    %01000000
        *                          PULSE ACCUMULATOR MODE
        *                          0= EVENT COUNTER
        *                          1= GATED TIME ACCUMULATION
0020    PAMOD   EQU    %00100000
        *                          PULSE ACCUMULATOR EDGE CONTROL
        *                          0= FALLING EDGES, HIGH LEVEL ENABLES ACCUM
        *                          1= RISING EDGES, LOW LEVEL ENABLES ACCUM
0010    PEDGE   EQU    %00010000
        *                          RTI INTERRUPT RATE
        *                          RTR1 RTR0  DIV E BY
        *                           0    0      2^13
        *                           0    1      2^14
        *                           1    0      2^15
        *                           1    1      2^16
0002    RTR1    EQU    %00000010
0001    RTR0    EQU    %00000001
0027    PACNT   EQU    $27         PULSE ACCUMULATOR COUNT REGISTER
0028    SPCR    EQU    $28         SPI CONTROL REGISTER
0080    SPIE    EQU    %10000000   SPI INTERRUPT ENABLE
0040    SPE     EQU    %01000000   SPI SYSTEM ENABLE
        *                          PORT D WIRE-OR MODE
        *                          0=PORT D OUTPUTS NORMAL
        *                          1=OPEN DRAIN
0020    DWOM    EQU    %00100000
        *                          MASTER/SLAVE MODE SELECT
```

```
        *                           0=SLAVE MODE
        *                           1=MASTER MODE
0010    MSTR    EQU    %00010000
*
0008    CPOL    EQU    %00001000    CLOCK POLARITY
0004    CPHA    EQU    %00000100    CLOCK PHASE
        *                           SPI CLOCK (SCK) RATE BIT
        *                           SPR1 SPR0  E DIV BY
        *                            0    0       2
        *                            0    1       4
        *                            1    0       16
        *                            1    1       32
0002    SPR1    EQU    %00000010
0001    SPR0    EQU    %00000001
0029    SPSR    EQU    $29          SPI STATUS REGISTER
0080    SPIF    EQU    %10000000    SPI INTERRUPT REQUEST
0040    WCOL    EQU    %01000000    WRITE COLLISION STATUS FLAG
0010    MODF    EQU    %00010000    SPI MODE ERROR INTERRUPT STATUS FLAG
002a    SPDR    EQU    $2A          SPI DATA REGISTER
002b    BAUD    EQU    $2B          SCI BAUD RATE CONTROL REGISTER
0080    TCLR    EQU    %10000000    CLEAR BAUD COUNTER CHAIN (TEST ONLY)
        *                           SERIAL PRESCALER SELECTS
        *                           SCP1 SCP0  DIV E BY
        *                            0    0       1
        *                            0    1       3
        *                            1    0       4
        *                            1    1      13
0020    SCP1    EQU    %00100000
0010    SCP0    EQU    %00010000
0008    RCKB    EQU    %00001000    SCI BAUD RATE CLOCK TEST (TEST ONLY)
        *                           SCI RATE SELECT BIT 2 THRU BIT 0
        *                           SCR2 SCR1 SCR0  PRESC OUT DIV BY
        *                            0    0    0       1
        *                            0    0    1       2
        *                            0    1    0       4
        *                            0    1    1       8
        *                            1    0    0       16
        *                            1    0    1       32
        *                            1    1    0       64
        *                            1    1    1       128
0004    SCR2    EQU    %00000100
0002    SCR1    EQU    %00000010
0001    SCR0    EQU    %00000001
002c    SCCR1   EQU    $2C          SCI CONTROL REGISTER 1
0080    R8      EQU    %10000000    RECEIVE BIT 8
0040    T8      EQU    %01000000    TRANSMIT BIT 8
        *                           MODE SELECT
        *                           0 = 1 START, 8 DATA, 1 STOP
        *                           1 = 1 START, 8 DATA, 9TH DATA, 1 STOP BIT
0010    M       EQU    %00010000
        *                           WAKE = WAKE UP (BY ADDRESS MARK/IDLE)
        *                           0 = WAKE UP BY IDEL LINE
        *                           1 = WAKE UP BY ADDRESS MARK
0008    WAKE    EQU    %00001000
002d    SCCR2   EQU    $2D          SCI CONTROL REGISTER 2
0080    TIE     EQU    %10000000    TRANSMIT INTERRUPT ENABLE
0040    TCIE    EQU    %01000000    TRANSMIT COMPLETE INTERRUPT ENABLE
0020    RIE     EQU    %00100000    RECEIVER INTERRUPT ENABLE
        *                           IDLE LINE INTERRUPT ENABLE
        *                           0=INHIBIT INTERRUPTS
        *                           1=ENABLE INTERRUPTS
```

```
0010   ILIE    EQU   %00010000
0008   TE      EQU   %00001000   TRANSMITER ENABLE (TOGGLE TO QUEUE IDLE CHAR)
       *                         RECEIVER ENABLE
       *                         0=OFF
       *                         1=ON
0004   RE      EQU   %00000100
       *                         RECEIVER WAKE-UP CONTROL
       *                         0=NORMAL
       *                         1=RECEIVER ASLEEP
0002   RWU     EQU   %00000010
0001   SBK     EQU   %00000001   SEND BREAK
002e   SCSR    EQU   $2E         SCI STATUS REGISTER
0080   TDRE    EQU   %10000000   TRANSMIT DATA REG EMPTY FLAG
0040   TC      EQU   %01000000   TRANSMIT COMPLETE FLAG
0020   RDRF    EQU   %00100000   RECEIVE DATA REG FULL FLAG
0010   IDLE    EQU   %00010000   IDLE LINE DETECTED FLAG
0008   OR      EQU   %00001000   OVER-RUN ERROR FLAG
0004   NF      EQU   %00000100   NOISE ERROR FLAG
0002   FE      EQU   %00000010   FRAMING ERROR FLAG
       *                         SCI DATA REGISTER
002f   SCDR    EQU   $2F         RECEIVE AND TRANSMIT DOUBLE BUFFERED
0030   ADCTL   EQU   $30         A/D CONTROL/STATUS REGISTER
0080   CCF     EQU   %10000000   CONVERSIONS COMPLETE FLAG (SETS AFTER 4TH CONVERSION)
       *                         CONTINUOUS SCAN CONTROL
       *                         0=4 CONVERSIONS AND STOP
       *                         1=CONVERT CONTINUOUSLY
0020   SCAN    EQU   %00100000
       *                         MULTIPLE CHANNEL/SINGLE CHANNEL CONTROL
       *                         0=CONVER SINGLE CHANNEL
       *                         1=CONVERT FOUR CHANNEL GROUP
0010   MULT    EQU   %00010000
       *                         CD  CC  CB  CA    CHANNEL SIGNAL  RESULT IN ADRX
*---------------------------------------------------
       *                         0   0   0   0     ADO PORT E0     ADR1
       *                         0   0   0   1     ADO PORT E1     ADR2
       *                         0   0   1   0     ADO PORT E2     ADR3
       *                         0   0   1   1     ADO PORT E3     ADR4
       *                         0   1   0   0     ADO PORT E4     ADR1
       *                         0   1   0   1     ADO PORT E5     ADR2
       *                         0   1   1   0     ADO PORT E6     ADR3
       *                         0   1   1   1     ADO PORT E7     ADR4
       *                         1   0   0   0     RESERVED        ADR1
       *                         1   0   0   1     RESERVED       ADR2
       *                         1   0   1   0     RESERVED       ADR3
       *                         1   0   1   1     RESERVED       ADR4
       *                         1   1   0   0     VREF HI        ADR1
       *                         1   1   0   1     VREF LOW       ADR2
       *                         1   1   1   0     VREF HI/2      ADR3
       *                         1   1   1   1     TEST/RESERVED  ADR4
0008   CD      EQU   %00001000
0004   CC      EQU   %00000100
0002   CB      EQU   %00000010
0001   CA      EQU   %00000001
       *
0031   ADR1    EQU   $31         A/D RESULT REGISTERS
0032   ADR2    EQU   $32
0033   ADR3    EQU   $33
0034   ADR4    EQU   $34

0035   RESVD2  EQU   $35         UNUSED
0036   RESVD3  EQU   $36         UNUSED
```

```
0037   RESVD4  EQU   $37          UNUSED
0038   RESVD5  EQU   $38          UNUSED
       *
       *                          SYSTEM CONFIGURATION OPTIONS
0039   OPTION  EQU   $39
       *                          A TO D POWER UP
       *                          0= A/D SYSTEM POWERED DOWN
       *                          1= A/D SYSTEM POWERED UP
0080   ADPU    EQU   %10000000
       *                          CLOCK SELECT
       *                          SHOULD BE USED IF E LESS THAN 1MHZ
       *                          0= A/D & EE USE SYSTEM E CLOCK
       *                          1= A/D & EE USE AN INTERNAL R-C CLOCK
0040   CSEL    EQU   %01000000
       *                          IRQ SELECT EDGE SENSITIVE ONLY (TIME PROTECTED)
       *                          0= IRQ CONFIGURED FOR LOW LEVEL
       *                          1= IRQ CONFIGURED FOR FALLING EDGES
0020   IRQE    EQU   %00100000
       *                          ENABLE OSCILATOR START UP DELAY (EXITING FROM STOP)
       *                          0= NO DELAY
       *                          1= A DELAY IS IMPOSED
0010   DLY     EQU   %00010000
       *                          CLOCK MONITOR ENABLE
       *                          0= DISABLED
       *                          1= SLOW OR STOPPED CLOCKS CAUSE RESET
0008   CME     EQU   %00001000
       *                          COP TIMER RATE SELECT BITS
       *                          CR1 CR0 E/2^15 DIV BY
       *                          0   0      1
       *                          0   1      4
       *                          1   0     16
       *                          1   1     64
0002   CR1     EQU   %00000010
0001   CR0     EQU   %00000001
       *                          CR1 CR0  E/2^15 DIV BY
       *                          ----------------------
       *                          0    0       1
       *                          0    1       4
       *                          1    0       16
       *                          1    1       64

003a   COPRST  EQU   $3A          ARM/RESET COP TIMER CIRCUITRY
003b   PPROG   EQU   $3B          EEPROM PROGRAMMING REGISTER
003c   HPRIO   EQU   $3C
       *                          READ BOOTSTRAP ROM (ONLY WRITABLE IF SMOD=1)
       *                          0= BOOT ROM NOT IN MAP (NORMAL)
       *                          1= BOOT ROM ENABLED
0080   RBOOT   EQU   %10000000
       *                          INTERNAL READ VISIBILITY
       *                          0= NO VISIBILITY OF INTERNAL READS ON EXTERNAL BUS
       *                          1= DATA FROM INTERNAL READS IS DRIVEN OUT DATA BUS
0010   IRV     EQU   %00010000
       *                          SPECIAL MODE SELECT
       *                          MODB MODA   MODE DESCR    SMOD MDA
       *                          ----------------------------------
       *                          1    0      SINGLE CHIP   0    0
       *                          1    1      EXPANDED MUX  0    1
       *                          0    0      BOOTSTRAP     1    0
       *                          0    1      SPECIAL TEST  1    1
0040   SMOD    EQU   %01000000
       *                          MODE SELECT
```

```
0020    MDA     EQU    %00100000
        *                           PRIORITY SELECT
        *                           MAY ONLY BE WRITTEN IF I BIT IN CC REG IS 1
        *                           PSEL3 PSEL2 PSEL1 PSEL0  INTERRUPT
        *                           ---------------------------------
        *                           0     0     0     0      TIMER OVERFLOW
        *                           0     0     0     1      PULSE ACCUM OVERFL
        *                           0     0     1     0      PULSE ACC EDGE
        *                           0     0     1     1      SPI XFER COMPLETE
        *                           0     1     0     0      SCI SERIAL SYSTEM
        *                           0     1     0     1      RESERVED (DEFAULT IRQ)
        *                           0     1     1     0      IRQ (PIN OR PAR I/O)
        *                           0     1     1     1      REAL TIME INTERRUPT
        *                           1     0     0     0      TIMER INPUT CAPTURE 1
        *                           1     0     0     1      TIMER INPUT CAPTURE 2
        *                           1     0     1     0      TIMER INPUT CAPTURE 3
        *                           1     0     1     1      TIMER OUTPUT COMPARE 1
        *                           1     1     0     0      TIMER OUTPUT COMPARE 2
        *                           1     1     0     1      TIMER OUTPUT COMPARE 3
        *                           1     1     1     0      TIMER OUTPUT COMPARE 4
        *                           1     1     1     1      TIMER OUTPUT COMPARE 5
0008    PSEL3   EQU    %00001000
0004    PSEL2   EQU    %00000100
0002    PSEL1   EQU    %00000010
0001    PSEL0   EQU    %00000001

        *                           RAM AND I/O MAPPING REGISTER

003d    INIT    EQU    $3D
0080    RAM3    EQU    %10000000
0040    RAM2    EQU    %01000000
0020    RAM1    EQU    %00100000
0010    RAM0    EQU    %00010000
0008    REG3    EQU    %00001000
0004    REG2    EQU    %00000100
0002    REG1    EQU    %00000010
0001    REG0    EQU    %00000001

        *                           FACTORY TEST REGISTER
        *                           RESTRICTED TEST MODES ONLY
003e    TEST1   EQU    $3E
0080    TILOP   EQU    %10000000   TEST ILLEGAL OPCODE
0020    OCCR    EQU    %00100000   OUTPUT CONDITION CODE REG STAT TO TIMER PORT
0010    CBYP    EQU    %00010000   TIMER DIVIDER CHAIN BYPASS
0008    DISR    EQU    %00001000   DISABLE RESETS FROM COP AND CLOCK MONITOR
0004    FCM     EQU    %00000100   FORCE CLOCK MONITOR FAILURE
0002    FCOP    EQU    %00000010   FORCE COP WATCHDOG FAILURE
0001    TCON    EQU    %00000001   TEST CONFIGURATION
        *                           CONFIGURATION CONTROL REGISTER
003f    CONFIG  EQU    $3F
        *                           SECURITY MODE DISABLE (MASK)
        *                           0=SECURITY MODE
        *                           1=NO SECURITY
0008    NOSEC   EQU    %00001000
        *                           COP SYSTEM DISABLE
        *                           0=COP SYSTEM ENABLED (FORCES RESET ON TIMEOUT)
        *                           1=COP SYSTEM DISABLED
0004    NOCOP   EQU    %00000100
        *                           ROM ENABLE
        *                           0= ROM IS NOT IN THE MEMORY MAP
        *                           1= ROM ON AT $E000 TO $FFFF
```

```
0002    ROMON   EQU    %00000010
        *                          EEPROM ENABLE
        *                          0= EEPROM IS NOT IN THE MEMORY MAP
        *                          1= EEPROM ON AT $B600 TO $B7FF
0001    EEON    EQU    %00000001

        *                          INTERRUPT VECTOR ASSIGNMENT

ffc0    RESVEC0 EQU    $FFC0       RESERVED
ffc2    RESVEC1 EQU    $FFC2       RESERVED
ffc4    RESVEC2 EQU    $FFC4       RESERVED
ffc6    RESVEC3 EQU    $FFC6       RESERVED
ffc8    RESVEC4 EQU    $FFC8       RESERVED
ffca    RESVEC5 EQU    $FFCA       RESERVED
ffcc    RESVEC6 EQU    $FFCC       RESERVED
ffce    RESVEC7 EQU    $FFCE       RESERVED
ffd0    RESVEC8 EQU    $FFD0       RESERVED
ffd2    RESVEC9 EQU    $FFD2       RESERVED
ffd4    RESVECA EQU    $FFD4       RESERVED
ffd6    VECSCI  EQU    $FFD6       SCI SERIAL SYSTEM
ffd8    VECSPI  EQU    $FFD8       SPI SERIAL TRANSFER COMPLETE
ffda    VECPAI  EQU    $FFDA       PULSE ACC INPUT EDGE
ffdc    VECPAO  EQU    $FFDC       PULSE ACC OVERFLOW
ffde    VECTOV  EQU    $FFDE       TIMER OVERFLOW
ffe0    VECTO5  EQU    $FFE0       TIMER OUTPUT COMPARE 5
ffe2    VECTO4  EQU    $FFE2       TIMER OUTPUT COMPARE 4
ffe4    VECTO3  EQU    $FFE4       TIMER OUTPUT COMPARE 3
ffe6    VECTO2  EQU    $FFE6       TIMER OUTPUT COMPARE 2
ffe8    VECTO1  EQU    $FFE8       TIMER OUTPUT COMPARE 1
ffea    VECTI3  EQU    $FFEA       TIMER INPUT CAPTURE 3
ffec    VECTI2  EQU    $FFEC       TIMER INPUT CAPTURE 2
ffee    VECTI1  EQU    $FFEE       TIMER INPUT CAPTURE 1
fff0    VECRTI  EQU    $FFF0       REAL TIME INTERRUPT
fff2    VECIRQ  EQU    $FFF2       IRQ
fff4    VECXIRQ EQU    $FFF4       XIRQ
fff6    VECSWI  EQU    $FFF6       SWI
fff8    VECILL  EQU    $FFF8       ILLEGAL OPCODE TRAP
fffa    VECCOP  EQU    $FFFA       COP FAILURE (RESET)
fffc    VECCMF  EQU    $FFFC       COP CLOCK MONITOR FAIL (RESET)
fffe    VECRES  EQU    $FFFE       RESET
```

INDEX

Symbols

Numerics

A

B

C

INDEX

INDEX

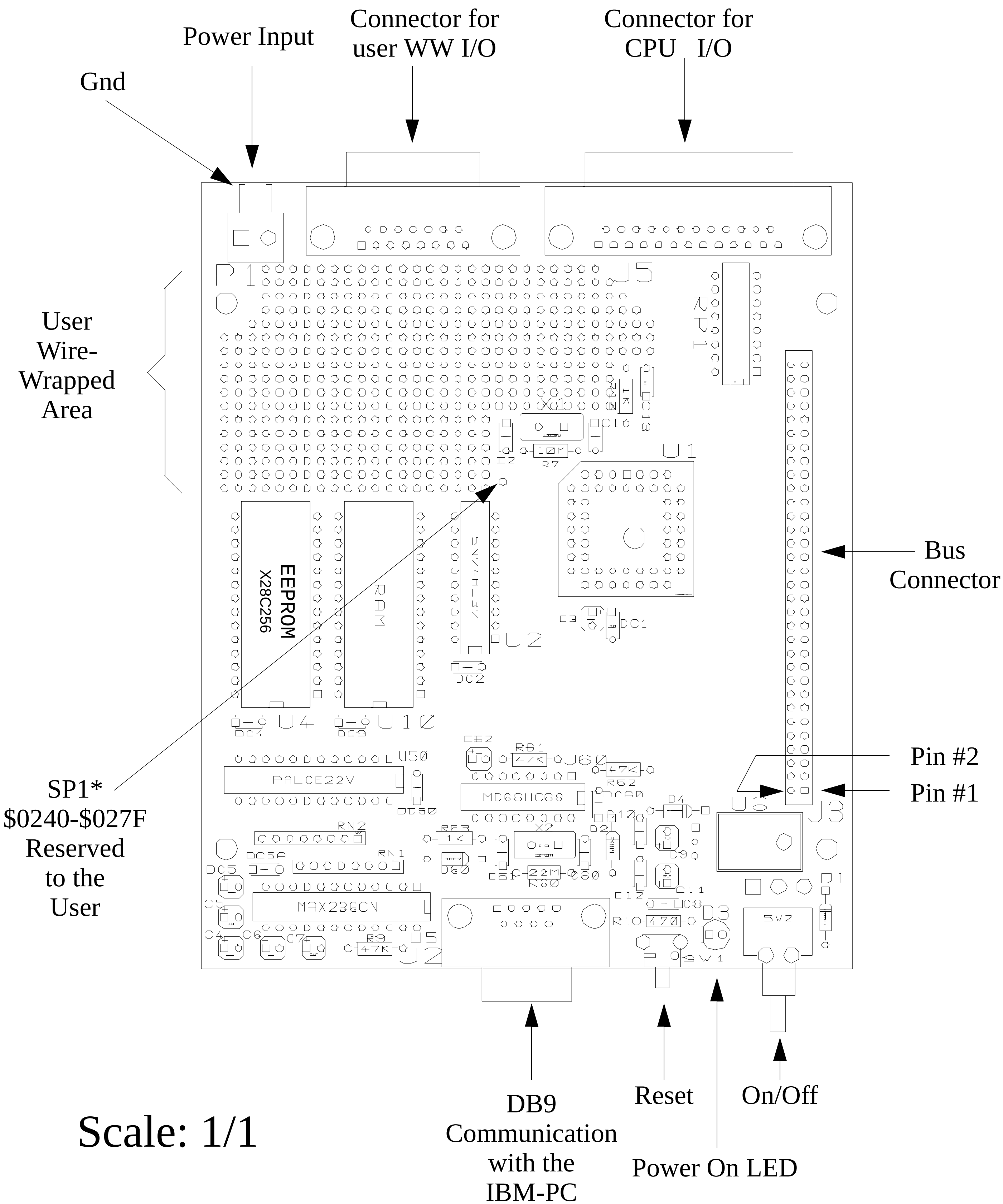
INDEX

T

V

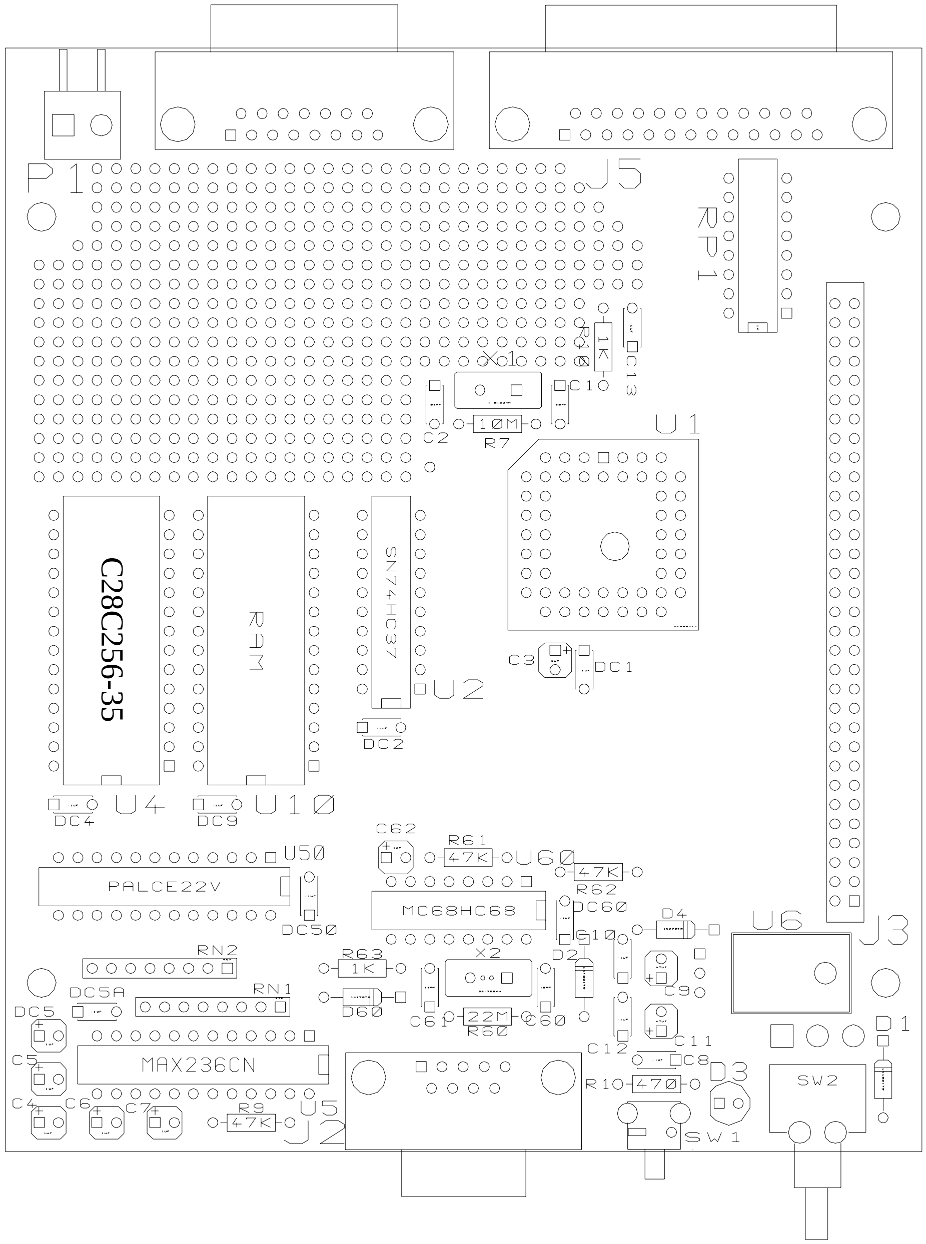
W

X



Scale: 1/1

Fig: 1 **MicroNator** System



RF-232

1404, rue Galt
Montréal Qc H4E 1H9
(514) 761-4201