HERE, NOW: A MICRO APL FOR 8080's

Dear Mr. Warren: *May 18, 1977*

I have just finished a micro APL for the 8080. It is completely debugged and operational. I am enclosing the documentation for publication which covers all of the operators and gives examples that thoroughly explain how to use it.

Since I hand assembled the language, I don't have a source listing available for publication (Manual typing of a 5.25K program is impossible!)

I am selling Tarbell Cassette tapes of EMPL 1.0 for $10 each. The tape comes with a user's manual, including complete information on implementing EMPL on any 8080 system, details of EMPL's operation (such as workspace organization), tape loading software, etc. Any improvements I make in the meantime will be included and completely documented. I hope this will be of interest to your readers.

Sincerely, *Britton House*
Erik T. Mueller *Roosevelt, NJ 08555*

E M P L , an 8080 APL

Britton House
Roosevelt, NJ 08555

8K EMPL is a micro APL for the Intel 8080. The current version fits in the first 5,376 bytes of memory. All the special symbols and operators of APL have been adapted to the ASCII character set.

The following description explains EMPL. Examples are used to clarify all operations.

EMPL has two modes: The normal Execution Mode in which all instructions are executed immediately, and the Definition Mode which permits the user to enter functions.

When in Execution Mode, the computer indents five spaces and waits for input.

Type an arithmetic expression and EMPL will respond with the result.

```
    2-6-3
-1
```

(Computer responses are underlined.) All expressions are evaluated by the right-to-left rule. That is, each operator takes as its right argument everything to its right. For example, in the above problem, EMPL first evaluates 6-3 which equals 3, then it evaluates 2-3 which equals -1.

The range is ±32767-- double-byte integer arithmetic is used.

Variable names may be any length of alphabetic characters. They are assigned values with the ":=" (typed as Control I.)

```
  SPEED:=2059
  SPEED - 59
2000
```

One dimensional arrays are called vectors in EMPL. A variable may be specified to be a vector of any length provided sufficient memory is available.

```
    A:= 5 -4 2 33
    A
5  -4  2  33
     A+1
6  -3  3  34
     A*4 3 2 0
20 -12 4 0
```

A Dyadic Operator (such as "+","*","?", etc.) takes two arguments, one on either side of the operator; as compared with a Monadic Operator such as "!" (absolute value), or "?" (random), which takes only one argument to its right.

Constants may be numeric or string vectors. Since the internal representation of both is the same, operations may be performed between the two types. To print a vector as a string, type "$" before it.

```
    X:='STRING'
    X
83  84  82  73  78  71
    $X
STRING
    $X=1
TUSJOH
```

A description of some of EMPL's unique operators follows: "\" is used to create a vector of consecutive integers between 1 and the right argument.

```
    \5
1  2  3  4  5
```

"^" gives the length of the right argument.

```
    ^'HELLO'
5
    ^\23
23
```

"'" returns the smaller of two numbers; """ returns the greater.

```
    -1'2
-1
    4  1  9"3  2  56
4  2  56
```

"." gives the remainder when the right argument is divided by the left.

```
    2.5
1
```

"‹","‹=","›","›=","=", and "#" return a 1 if the condition

is true, otherwise they return a 0.

```
    5<4
0
    7 3 4# 3
1 0 1
```

"," chains together two vectors.

```
     HI:= 1 2 3 4
     WN:=419 512
     XX:=HI,WN
     XX
1 2 3 4 419 512
     A:='CATE'
     B:='NATION'
     $A,B
CATENATION
     18,2,10*4
18 2 40
     18 2 10*4
72 8 40
```

"[" selects elements of the left argument at the locations given by the right argument.

```
     A:=2 17 -3 4
     A[1 1 2
2 2 17
     $'ABCRST'[3 1 6
CAT
```

"f%" performs the Dyadic Scalar Operator f from right to left on all the elements of the right argument. Thus, it treats +%\5 as if it were 1+2+3+4+5.
To find the largest element in a vector, type "%VECTOR.

To write a program (or function), the user must enter EMPL's Definition Mode. Type "&" followed by the new name of the function.

```
     &PAS
)
```

The computer now prompts with a ")" indicating that it is in Definition Mode. Editing is now like a BASIC editor: Type a line to insert it in the function; type the line number alone to delete the line; type an old line over to replace the old one. Type "@" to list the function; type "$" to renumber by a specified amount.
For example, the user types in a program to print the first N rows of Pascal's triangle.

```
)10 P:=1
)20 =:PRINT*N)=⍵P:=(0,P)+P,0
)15 PRINT: P
)$ 10
)@
     &PAS
[10] P:=1
[20] PRINT: P
[30] =: PRINT*N) =⍵P:=(0,P)+P,0
)
```

Note that ":=" can be used inside any expression. "PRINT:" is used as a label in line 20. Upon exiting Definition Mode, PRINT becomes a variable whose value is 20.
"=:" (Control H) is a branch to the line given by the expression which follows it. Branch to a null vector is no branch (computer proceeds to next line), and branch to an undefined line is the end of program. The user now types a "&" to leave Definition Mode.
To execute a program, type its name.

```
)
     N:=4
     PAS
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
```

Conditional branches are accomplished by the following:

```
     =:LINE*\TEST
```

If the condition TEST is true, the computer will branch to LINE. Otherwise, the computer will proceed to the next line.
Strings inside double quotes may be used to print comments.

```
     "N=";N
N=4
```

Note that semicolons are used to separate items, and a semicolon at the end will suppress CRLF.

"%:" is used to execute a string as if it had been typed in execution mode.

```
     STR:='A:=5*\5'
     %:STR
     A
5 10 15 20 25
```

Specific elements of a vector may be assigned values.

```
     A:=\6
     A[3 4]:= 17 18
     A
1 2 17 18 5 6
```

When "@" is assigned a value, the value is printed on the terminal.

```
     A:=5*@:=5
5
```

When "$" is assigned a value, the value is printed as a string on the terminal.

```
     A:=$:='TESTING'
TESTING
```

If "@" is used as an operand in an expression, the computer first requests input from the terminal. "$" enables string input to be used. (The input is taken as a literal.)

```
     &TEST
[10] "WHAT IS YOUR NAME?";
[20] NAME:=$
[30] "HI,";$NAME
[40] "INPUT AN EXPRESSION"
[50] EXPR:=0
[60] "THE ANSWER IS";EXPR
     &
     TEST
WHAT IS YOUR NAME?RACHEL
HI, RACHEL
INPUT AN EXPRESSION
@:5*5
THE ANSWER IS 25
```

User defined monadic or dyadic functions are possible. Here is an example of a user defined monadic function to take the factorial of the right argument.

```
     &F:=FACT VL
 10  F:=*%\VL
     &
```

"FACT" may now be used as any EMPL monadic operator can be used.

```
     FACT 3
6
```

When FACT 3 is typed, a function called FACT is searched for. When it is found, VL is set equal to 3 and the function is executed. Upon completion of the execution, the computer uses the variable specified to the left of the assignment operator ":=" as the result.

Here is a user defined dyadic function.

```
     &RS:=SZ RHO VTR
[10] RS:=\0
[20] =:20*\(⍴RS:=RS,VTR)⌿SZ
[30] RS:=RS[\SZ
     &
```

This function is used to restructure the vector on the right such that it has the length indicated on the left.

```
     4 RHO 5 2 3
5 2 3 5
     $3 RHO 'ABCDEFG'
ABC
```

To reenter Definition mode of an old function, just type its name.

```
     &RHO
)
```

Parentheses are used to override the normal order of evaluation.

```
     (3-5)+7
5
```

The following commands may be used inside a program, or in Execution Mode.

```
)CLEAR - Clear the workspace.
)FNS   - List the names of all functions
         in the workspace.
)VARS  - List the names of all variables
         in the workspace.
)STOP  - Return to execution mode.
)ERASE object - Erase the specified variable or function.
)SI    - Display the State Indicator.
)PUR   - Clear the State Indicator.
```

The State Indicator is the stack of the return addresses of all programs whose execution is pending.

Suppose a program is written, PROG, which calls for the execution of another one, ASK:

```
     &PROG
[10] ASK
[20] "OK"
      .
      .
      .
     &ASK
[10] HI THERE
     &
     PROG

ASK[10]HI THERE
ERROR  2810
```

The computer prints the program (or function) and line in which the error occurred.

```
     )SI
ASK[-1] *
PROG [20]⍴
2210 FREE
```

An error has occurred in line 10 of ASK. Since the execution of ASK hasn't been completed, the computer still has the return address saved. (Line 20 of PROG) Now whenever a =:0 instruction be typed, execution will continue at line 20 of PROG.

The program with the "*" to the right of it is the current suspended program. All

"=:" (branch) statements refer to this program. Line -1 indicates that it is currently in execution mode. All programs with "^" to the right of them are pending execution. The program to return to next is always on the top of the list. It is a good idea to clear the State indicator often to prevent excess garbage from taking up memory. The State Indicator is automatically cleared when entering Definition Mode, and after ERROR 5097 (workspace full.)

Here are some sample functions to illustrate EMPL's use.

(1) Generate n prime numbers:

```
     &P:=PR END
[10] P:=1+T:=1
[20] TEST: =:0*\END⟨=^P
[30] ADD: =:ADD*\+%0=P.T:=T+2
[40] P:=P,T
[50] =:TEST
     &
     PR 10
2  3  5  7  11  13  17  19  23  29
     1 + PR 3
3  4  6
```

(2) Compress a vector:

```
     &ANS:=SLCT COMP VCTR
[10] IDX:=1, ANS:=\0
[20] BK: ANS:=ANS,(VCTR[IDX)*\SLCT[IDX
[30] =:BK*(IDX:=IDX+1)⟨=^VCTR
     &
     1 0 1 1 0 COMP\5
1  3  4
```

(3) Sort a group of numbers:
(This function uses COMP)

```
     &ORD:=SORT UNS
[10] ORD:=\0
[20] LB: =:0*\&^UNS
[30] WHICH:=UNS='%UNS
[40] ORD:=ORD,WHICH COMP UNS
[50] UNS:=(&WHICH) COMP UNS
[60] =:LB
     &
     SORT -2 1 -1 0 2
-2  -1  0  1  2
```

(4) Reverse a vector:

```
     &RV:=REV VC
[10] VC[(1+^VC)-\^VC
     &
     $REV'HI THERE'
 EREHT IH
```

(5) Raise the left argument to the right argument. (They must both be single numbers.)

```
     &E:=BSE RAIS NUM
[10] EX:=*%BSE*&&\NUM
     &
     2 RAIS 5
32
```

E M P L Language Summary

Monadic Scalar Operators

-Y	Minus Y
⌈Y	The absolute value of Y
&Y	Not Y
?Y	Random from 1 to Y

Monadic Mixed Operators

\Y	Vector of consecutive integers from 1 to Y
^Y	The length of Y

Dyadic Scalar Operators

X+Y	X plus Y; X or Y
X-Y	X minus Y
X*Y	X times Y;X and Y
X/Y	X divided by Y
X'Y	Minimum of X and Y
X"Y	Maximum of X and Y
X.Y	Remainder of Y/X
X⟨Y	X less than Y
X⟨=Y	X less than or equal to Y
X⟩Y	X greater than Y
X⟩=Y	X greater than or equal to Y
X=Y	X equal to Y
X#Y	X not equal to Y

Dyadic Mixed Operators

X,Y	Y catenated to X
X[Y	Elements of X at locations Y

Composite Operators

f%Y	The Operation f performed from right to left on all the elements of Y

Special Operators and characters

X:=Y	Assign Y to X
$X	Print X as a string
"TEXT"	Print literal text
%:X	Execute a string
=:X	Branch to X
VAR X	Indexed variable assignment
@	Numeric Input/Output
$	String Input/Output

```
'TEXT' String vector
(X)    Expression in parentheses

Function Definitions
&(function)    Niladic function; no result
&(result):=(function)(rightvar)   Monadic
           function with result
&(result):=(leftvar)(function)(rightvar)
           Dyadic function with result


Definition Mode Commands
@    List function
$    Renumber
&    Return to execution mode

EMPL Range = ±32,767

Commands
)CLEAR      Clear Workspace
)FNS        Display function names
)VARS       Display variable names
)PUR        Clear State Indicator
)ERASE o b Erase function of variable
)SCN        Clear screen
)SI         Display State indicator
)STOP       Return to execution Mode

Special Control characters
Control B  Backspace
Control C  Stop
Control F  Forward space
Control H  =:
Control I  :=
Control N  )=
Control Y  (=

Error messages
412  String is too large to execute
885  Object already erased
1082 Unknown operand type in expression
1469 Length of vectors doesn't match
2268 Illegal Operation in reduction
2495 Variable not present in indexing
2529 Length of index doesn't match length
     of value to be assigned
2559 Index greater than length of variable
2628 Unknown assignment operand type
2690 Function not found
2810 Variable or function not found
3503 Backspace less than beginning of line
     buffer
3521 Forward spacing greater than end of
     line buffer
3542 Input over 72 characters
3737 Bad syntax in function definition
3827 Illegal command in definition mode
4058 Illegal redefinition of monadic func-
     tion header
4083 Illegal redefinition of dyadic func-
     tion header
5097 Workspace full
5302 Division by zero
```

MANUAL AVAILABLE FROM THE DISE REPOSITORY

A COMPUTER SCIENCE HARDWARE LABORATORY MANUAL

Professors B. Carey and M. Barber

Electrical Engineering and Computer Science
University of Connecticut
Storrs, Connecticut 06268
(203) 429-4816

This manual presents a sequence of nine experiments involving the design, construction, and testing of digital hardware to be completed in a fourteen week semester. Each experiment requires four hours of lab work per week. The laboratory is physically organized around a number of general purpose, digital, bread-boarding modules to permit a convenient structuring of experiments and the implementation of problem solutions. The experiments consist of a logical sequence of problems which start with the design of combinational modules, progress to standard sequential functions such as shirt registers and binary counters, move on to multifunction modules, explore the transmission of information between and synchronization of two digital networks, look at the design of an arithmetic function in a digital computer, and ends with the design and programming of a complete general purpose computer system. The experiments involve the reasonable range of SSI, MSI and LSI devices seen in the 7400 logic family and may directly concern themselves with other logic families if appropriate.

HOW TO OBTAIN: A single copy will be provided without charge to any individual involved in teaching or in the development of digital systems education. Requests should be made on the letterhead stationery of the individual's organization and include the individual's title. Please send requests to: DISE REPOSITORY, School of Electrical Engineering, Purdue University, West Lafayette, IN 47907.

MUG OFFERS MEDICINE FOR MUMPS

1976 MUMPS Users' Group Meeting is a Huge Success

The fifth MUMPS Users' Group Meeting was held 9/29-10/1 in Madison, WI. [MUMPS stands for Massachusetts (General Hospital) Utility Multi-Programming Systems. It is a FOCAL-like language with extensive file-handling facilities for tree-structured files as a major feature. Notable aspects from the hobbyist viewpoint are (1) it's in the public domain, and (2) it runs on a PDP-11. It was originally developed for use in hospital and biomedical applications.—Editor] The meeting was packed with exciting presentations and live demonstrations of interactive applications developed using the MUMPS computer language. Among the topics of special interest were voice-computer interfaces, medical information systems, radiology systems, patient counseling by computer, patient appointment systems, computer-aided instruction, on-line cataloging for the National Library of Medicine, and the formation of the MUMPS Users' Group Application Library to facilitate the transfer of MUMPS applications.

An introductory tutorial on programming in MUMPS was given, and was attended by administrators and physicians as well as programmers. A tutorial and a workship on advanced MUMPS techniques were held for experienced MUMPS programmers. There were other workshops on such topics as computer-aided instruction and MUMPS application transfer.

Seven vendors of MUMPS applications and implementations participated in a vendors' forum. Two of the vendors that exhibited at the Meeting (namely, Artronix, Inc. of St. Louis, and Digital Equipment Corporation of Maynard) announced their completion of implementations of the NBS-defined Standard MUMPS which is now seeking approval as an ANSI Standard.

There are now about 400 institutions around the world that use MUMPS, and this total is growing by about 80% per year. A variety of computers are used, including DEC's PDP-11, PDP-15 and PDP-10, Data General's Eclipse and Nova, Artronix' PC-12 and PC-16, Burroughs' B6700, and IBM's System/360 or System/370. For information on MUMPS, future MUG meetings, and MUMPS applications, contact the MUG Executive Secretary, Dr. Joan Zimmerman, Biomedical Computer Laboratory, 700 S. Euclid Ave., St. Louis, MO 63110, (314) 454-3364.