MICROPOLIS USERS GROUP

MUG Newsletter # 15 - October 1981

****

WRITING GOOD PROGRAMS (PART 3)

by Burks A. Smith of DATASMITH
Box 8036, Shawnee Mission, KS 66208

Last month we discussed techniques for writing code that produces self-documenting, modular programs that are easy to maintain and less likely to have bugs. I have concentrated on planning techniques and general styles of writing programs rather than language details because good style encourages good content. In order to become fluent in any language (French, Basic, or whatever) you must use it often. This is really the only way you can learn programming, but starting out with proper planning and style will make it easier.

SERVICING OPERATOR INPUTS

Most programs for business or scientific purposes have some part of the input data entered directly from the keyboard by a human. This input data may vary from a simple "yes" or "no" answer to large volumes of figures for actual processing. The program must be able to get along with its human users, and a very important aspect of this is the ability to tolerate human error with out "crashing" or making a mistake itself. Everything coming from the keyboard should be checked for errors, and if an error is detected the operator should be politely informed and given another chance.

The simplest case is a question which requires a "yes" or "no" answer. First of all, the question put to the operator should be easy to understand and indicate what kind of a response is expected. "DO YOU WANT TO USE THE PRINTER (Y OR N)?" is a typical example of this. In response to the above question, a string of characters is INPUT from the keyboard. It is expected that the operator entered a "Y" or an "N", since this is what was instructed, but as the programmer, you must be prepared for anything and allow a little leeway too. For example, are you going to reject anything except the single characters "Y" or "N"? That may be too rigid. What if the operator enters an "N" followed by a blank space? N-space does not equal N. Will you take "YES" or "NO" too? What if the operator just hits the RETURN key?

All the possibilities must be considered when coding even a simple question. The program's treatment of the answer depends, to a great extent, on the application. However, I prefer to use an algorithm that only considers the first character entered. Use a LEFT$ function to isolate the first character in the string. If it is a "Y", assume "yes"; if it is an "N", assume "no". Otherwise, assume an invalid response, report to the operator that you can't understand, and ask again. Repeat the loop until you get a valid answer. In MICROPOLIS BASIC, just hitting the return key in response to an INPUT statement leaves the value of the input variable unchanged. Therefore, it is wise to initialize the input variable to "", or a default answer, before the INPUT statement or the results may not be predictable.

Operator response to a menu of numbered choices is a little more easy to filter out. The value input at the keyboard will be a number, so a numeric variable is used. BASIC won't allow anything but a number in this case, so you know at least a number will be returned by the INPUT statement. All you have to do is make sure the number entered is within the the limits of the menu choices. For really foolproof menus, you should avoid using an integer variable as the variable input. I seldom heed my own advice in this respect, but keep in mind that an integer variable is stored as a fixed number of digits, while a real (floating point) variable is stored in scientific notation. With the default SIZES, your program will crash with a CONVERSION ERROR if the operator enters 500000 or more through an INPUT statement with an integer variable. Try it.

When the operator needs to enter large amounts of data, it is absolutely necessary that the operator be given an opportunity to personally verify the data before it is stored away or used in a calculation. The easiest way to do this is to re-display it after it is input and ask the operator if it is OK. Facilities should be provided to change a single item, without having to re-enter all the data, if an error is detected. Programs that use a database on diskette should have facilities to manually update ANYTHING on the diskette, and to manually recover from errors made when erroneous data is used in calculations. The excuse, "It's in the computer and we can't change it," is a sure sign of a bad program. In fact, it should be easier to change data in a computer than data on paper. Computers are supposed to make things easier, not more difficult!

SERVICING DISK ACCESSES

Besides errors from data entered in INPUT statements, the most common cause of programs crashing can be attributed to disk errors. Not real I/O errors due to hardware, but things people do or forget to do. The most common ones are DRIVE NOT UP and FILE NOT FOUND, caused by the program expecting to find something that has not been provided by the operator, but there are others. See the ERR function in the MICROPOLIS BASIC manual (table 5.5). All of these disk errors cause the program to be stopped in its tracks, and can be catastrophic to a newly updated file that is open at worst or annoying at best.

BASIC comes to the rescue with the ERROR clause in the OPEN statement for a file. <ERROR linenumber> refers to a line number to GOTO if an error is encountered in the program. The line number should contain code that determines which error has occured (the ERR function) and appropriate error handling routines. In the case of many errors, all that is needed is to inform the operator of the error and wait until the operator says its OK to retry the operation. Other errors which can't be corrected by the operator need to terminate the program in an orderly way by closing files and exiting to a menu, etc. For example, a DISK FULL error handler definitely needs to close the file it was updating so no data will be lost. Beyond that, it depends on the program and the application. Some programs might instruct the operator to mount a fresh diskette and pick-up where they left off, while others would have to be terminated.

Trying to imagine all the errors that could possibly occur and devising software recourse is not easy. Murphy's Law states: "If something can go wrong, it will go wrong," and I read somewhere that: "It is impossible to make anything foolproof because fools are so ingenious." Good error handling routines take programming time that could be used for more exicting tasks, but chances are you will eventually need it. Also, no matter how hard you try, you will probably overlook something. Even so, don't be discouraged; any error handling is better than none and your programs will be more reliable as a result.

NEXT MONTH we'll cover some nitty-gritty details of how to design loop structures.

....

TABLE HANDLING TECHNIQUES - Part 2

by N. P. Dembinski
8618 Essex Ave., Chicago IL 60617

Last month we summarized some of the programming techniques and guidelines useful for efficient table handling. This month, a description of each technique is presented. Also, the flowchart on the last page is intended to aid programmers in the selection of the most suitable table handling techniques for the specific problems and applications.

A. DIRECT TECHNIQUE

The table contains as many entries as the value of the highest numeric table code being processed. The code being searched for literally becomes the index for accessing the corresponding table entry.

LIMITATIONS

The codes being processed must contain numeric digits. The codes usually are not in straight numeric sequence, (e.g., 001, 002, 003, etc.). Gaps occur between codes and the table size becomes prohibitive. Input data must be read at the beginning of each step to fill the table.

B. SERIAL TECHNIQUE

Search the table beginning with the first position, for each transaction and incrementing the index by one, or coding repetitive compares. Entries do not have to be in sequence but may be ordered by testing for the most frequently accessed items first.

LIMITATION

When the table size exceeds 30 entries, the execution time to test all the entries becomes prohibitive, and the input data must be read at the beginning of each step to fill the table.

C. BINARY TECHNIQUE

The binary search technique is a good choice for over 60 entries if memory is critical. The argument to be found is first compared against the middle of the table. On the basis of this comparison, the upper or lower half of the table will be compared against. The table will be halved until an equal condition occurs or the size of the table is reduced to zero. The table arguments must be in sequence. A count of the number of entries (easily generated when the table is filled) is also required. The table size should be one less than the power of two.

D. INCREMENT (PARTIONED) SCAN TECHNIQUE

This is most efficient method, time-wise, for over 60 entries. Search increment entries must be in sequence, but entries within a group do not have to be in order. When searching a table with this organization, locate the group of entries wanted and move that group of entries to a work area for a detailed serial search.

E. SUBSCRIPTED ITERATIVE TECHNIQUE

This is the most common method of searching a table and is the most inefficient.

F. LITERAL TECHNIQUE

A table does not appear in the data area, but the codes being searched for appear as literals in the procedure division of the program. A conditional statement must be written for each code to be processed. When the literal being tested equals the transaction code being processed, the corresponding data appears as a literal and is moved to a common storage area.

LIMITATIONS

Limited to fixed codes not subject to change since literal change requires source language changes, recompilation, and far more coding is required than in the other techniques.

With the availability of high level languages like Basic, Fortran IV, Cobol, Pascal, and a few others that may come along in the future, or some macro oriented assembler language, the literal technique becomes easy to use. The examples shown in the following text will demonstrate the use of the Basic language. Basic is the most universal language for micro-computers that is available. These techniques have already been used or can be translated to some other high level language easily.

Presenting a strong case for using the literal technique, assume that a company has offices or plants in 16 cities. To conserve space on the master record, the city and state are not stored but identified by the five digit zip code number. The direct approach is prohibitive since the table would require 999,999 entries allocated for 16 entries only. The amount of coding required for the literal approach is not prohibitive, there is little likelihood that the relationship of the zip code to city will ever change and the literal processing is four to five times as fast than either the serial or binary techniques. The Basic coding which would implement the literal technique follows below.

BASIC EXECUTIVE

```
        (ETC.)
        GOSUB 6000
        (ETC.)
   6000 !GET-ADDRESS
   6010 IF Z$>"27601"THEN GOTO 6070
   6020 IF Z$ > "14205" THEN GOTO 6120
   6030 IF Z$ > "02109" THEN GOTO 6160
   6040 IF Z$ = "00902" THEN C$ = "SAN
               JUAN PR" : RETURN
   6050 IF Z$ = "02109" THEN C$ = "BOSTON, MA"
                        :RETURN
   6060 GOTO 6300
   6070 IF Z$ > "55401" THEN GOTO 6280
   6080 IF Z$ > "43216" THEN GOTO 6220
   6090 IF Z$ = "33101" THEN C$ = "MIAMI FL"
                        :RETURN
   6100 IF Z$ = "43216" THEN C$ = "COLUMBUS OH"
                        :RETURN
   6110 GOTO 6300
   6120 IF Z$ > "19104" THEN GOTO 6250
   6130 IF Z$ = "15219" THEN C$ = "PITTS. PA"
                        :RETURN
   6140 IF Z$ = "19004" THEN C$ = "PHIL. PA"
                        :RETURN
   6150 GOTO 6300
   6160 IF Z$ = "07102" THEN C$ = "NEWARK NJ"
                        :RETURN
   6170 IF Z$ = "14205" THEN C$ = "BUFFALO NY"
                        :RETURN
   6180 GOTO 6300
   6190 IF Z$ > "85026" THEN GOTO 6280
   6200 IF Z$ = "77002" THEN C$ = "HOUSTON TX"
                        :RETURN
   6210 GOTO 6300
   6220 IF Z$ = "57302" THEN C$ = "MUNCIE IN"
                        :RETURN
   6230 IF Z$ = "55401" THEN C$ = "BLOOM IN" :RETURN
   6240 GOTO 6300
   6250 IF Z$ = "21233" THEN C$ = "BALT. MD" :RETURN
   6260 IF Z$ = "27601" THEN C$ = "RALEIGH NC"
                        :RETURN
   6270 GOTO 6300
   6280 IF Z$ = "94086" THEN C$ = "SUNNYVALE CA"
                        :RETURN
   6290 IF Z$ = "96813" THEN C$ = "HONOLULU HI"
                        :RETURN
   6300 C$ = "" : RETURN ! BAD ZIP CODE
```

GUIDE FOR SELECTING TABLE LOOK-UP TECHNIQUES

A. Are the keys (item no., etc.) all numeric (0 thru 9)?
B. Are most of the integers within the range of the control key required?
C. Use the "direct technique".
D. Is programming development time available to code the literals?
E. Will the same key always relate to the same entry?
F. Use the "Literal technique".

G. Is it impossible to sequence the table-fill data records?
H. Use the "serial technique".
I. Does the total number of table entries exceed

10?
J. Use the "binary technique".
K. Use the "serial technique".

```
         /\
        /  \
       / A  \______
       \    /      I
        \  /       I
         \/        I
         I Yes     I
         I         I
         I         I
         V         V
         /\        /\
        /  \      /  \
       / B  \____/ D  \______
       \    /    \    /      I
        \  /      \  /       I
         \/        \/        I
         IYes      IYes      I
         I         I         I
         I         I         I
     ____V___      V         V             ________
    I        I    /\        /\        /\       I        I
    I        I   /  \      /  \      /  \      I        I
    I   C    I  / E  \____/ G  \____/ I  \_____I   K    I
    I        I  \    /    \    /    \    /     I        I
    I        I   \  /      \  /      \  /      I        I
    I________I    \/        \/        \/       I________I
                  IYes      IYes      IYes
                  I         I         I
                  I         I         I
              ____V___  ____V___  ____V___
             I        II        II        I
             I        II        II        I
             I   F    II   H    II   J    I
             I        II        II        I
             I        II        II        I
             I________II________II________I
                        .......
```

PROGRAM DOCUMENTATION

by Joel Shapiro of BONJOEL ENTERPRISES
Box 2180, Des Plaines, IL 60018

Most of us will spend many hours writing programs for others and our own use. However, these programs may be used so infrequently that we forget how to work with them. We'll use a lot of prompting within the program, but the overall purpose, or operation of the program may be lost because enough detail isn't given at the time they're in use.

Including a set of instructions within the program is an excellent idea and, if instructions are required by the operator, a menu selection can provide them. This takes up valuable memory and, if the program is large, you may not have enough for the program itself. Text does take up a lot of space and becomes troublesome to enter in DATA statements when you can't see the true length of each line on the screen.

Alternative to this is the use of a data file and a program or routine that will display the instructions retained in the data file. That is the approach offered here. Two programs; INSTR.GEN and INSTR.READ respectively generate and read the instruction data file.

An important feature is that the system will allow you to select the starting record of the instruction file for display and this can be keyed to the calling program. It will also allow you to return to the correct program after the instructions have been displayed. The technique for doing this is detailed within the INSTR.READ program listing. This is a good feature of the system in that it allows you to reuse the general purpose instructions you may have in your file for many different programs and routines. Also, since you may select the specific routine you wish to return to after use of the INSTR.READ program, redundant instructions are possible without duplication in the file.

INSTR.GEN is charged with writing the file with the instructions as entered. It stores three (3) lines within one (1) physical file record without packing. No editing utility is provided so each line should be checked before RETURN is pressed. After three (3) lines of instructions are entered the system will pause while the record is written to the file. This process continues until you're done and a backslash is entered at the beginning of a new line to close the file. If a file already exists, additional instructions can be added to the end of the file.

Note line 1000 on the listing and you'll see how I got a modified version of Buzz's INKEY program into RAM. Of particular importance is the input subroutine in lines 20 thru 60. BASIC's INPUT statement prints a question mark and a space at the beginning of the input line, therefore, the resultant line on the screen is offset by two spaces and won't give you an accurate indication of how the printed line will appear. This routine will but must be located in the lowest line numbers of the program to be effective. Located at the back of the program the routine can't keep up with a fast typist and characters will be missed. This is a very useful routine as placing the escape sequence for direct cursor positioning in line 20 will allow you to input the line anywhere on the screen. This allows you to draw a picture of a form and fill in the blanks directly. Like I said, it's a very useful routine.

INSTR.READ can be used as a program or changed and incorporated into a program as a routine. The operation is very simple and the technique for returning to the correct program is explained in the listing.

Note the string delimiter is changed to Ascii 255 in both programs to allow you the use of commas in the text. Line 280 in INSTR.READ provides for the return of the delimiter to the default value (a comma).

Be sure to revise the variables for SCREEN WIDTH and SCREEN HEIGHT in both programs as these values are used for proper formatting.

I hope this may encourage some of you to provide instructions as part of your program development; it certainly helps to prevent mistakes when you've forgotten how the program is supposed to work. And, as editing and a few other features were left out of these, I hope you'll keep the MUG advised if you develop any enhancements.

Title: INSTR.GEN

```
 10       GOTO 410
 20  >*   PRINT:B$=""
 30   *>  C$=FAA(1):IF ASC(C$)=3 STOP
 40  <*   IF ASC(C$)=13 RETURN
 50       IF ASC(C$)=127 OR ASC(C$)=8 THEN B$=LEFT$
          (B$,LEN(B$)-1):GOTO 30
 60       B$=B$+C$:GOTO 30
 70       !   INPUT ROUTINE IS ABOVE - MUST BE AT B
          EGINNING OF PROGRAM
 80       !   CLEAR SCREEN
 90  ><   PRINT CHAR$(26):PRINT REPEAT$(CHAR$(13)+C
          HAR$(10),INT(H*.5)-2):RETURN
100       !   REVERSE VIDEO
110  ><   PRINT CHAR$(18);:RETURN
120       !   VIDEO RESET
130  ><   PRINT CHAR$(17);:RETURN
140  ><   INPUT "Press RETURN When Ready ";A$:RETUR
          N
150       !   PROGRAM START
160    >  GOSUB 90:PRINT TAB(T); "CREATE NEW INSTRU
          CTION FILE":PRINT
170       !   CREATES NEW FILE
180    >  PRINT TAB(T)"; Enter Name for New File";:
          INPUT N$:PRINT:PRINT TAB(T); "Enter Drive
          Number for File";:INPUT A:IF A<0 OR A>3 G
          OTO 180
190       !   OPEN FILE
200       N$=FMT(A,"N:9:")+N$:OPEN 1 N$ ERROR 210:G
          OTO 220
```

```
210     > PRINT:PRINT TAB(T);:GOSUB 110:PRINT "DISK
          ERROR - ";ERR$:GOSUB 130:PRINT TAB(T); "C
          orrect and ";:GOSUB 140:GOTO 180
220     > S=RECPUT(1):GOSUB 90:PRINT TAB(T); "Enter
          text, line for line, exactly as":PRINT TA
          B(T); "You wish it to appear on the scree
          n.":PRINT
230       PRINT TAB(T); "Enter <CR> at the end of e
          ach line.":PRINT:PRINT TAB(T); "Enter \ w
          hen done with instructions.":PRINT:PRINT
          TAB(T);:GOSUB 140:GOSUB 90
240       !   GET INPUTS - GETS 3 LINES AND WRITES
          TO FILE
250       !   LINE IS COMPOSED IN B$ - TRANSFERS TO
          G$(X) ARRAY
260     > FOR I=0 TO 2:GOSUB 20:IF I=0 AND B$=CHAR$
          (92) GOTO 330
270           IF B$=CHAR$(92) GOTO 310
280       IF LEN(B$)>W% THEN I=I-1:B$="":PRINT TAB(
          T);:GOSUB 110:PRINT "LINE TOO LONG - REEN
          TER":GOSUB 130:NEXT I
290       G$(I)=B$:NEXT I
300       !   WRITE TO FILE
310     > A$=G$(0)+Y$+G$(1)+Y$+G$(2)+Y$:PUT 1 A$:IF
          B$<>CHAR$(92) THEN A$="":G$(0)="":G$(1)="
          ":G$(2)="":GOTO 260
320       !   ENDING FOR THIS ROUTINE
330     > F=RECPUT(1)-1:CLOSE 1:GOSUB 90:PRINT TAB(
          T); "Starting Record =";S:PRINT:PRINT TAB
          (T); "Ending Record   =";F:PRINT:GOSUB 14
          0:GOTO 530
340       !   OPENS EXISTING FILE AND TRANSFERS TO
          INPUT ROUTINE ABOVE
350     > GOSUB 90:PRINT TAB(T); "ADD TO EXISTING I
          NSTRUCTION FILE":PRINT
360     > PRINT TAB(T); "Enter Name for Existing In
          struction File";:INPUT N$:PRINT:PRINT TAB
          (T); "Enter Drive Number for File";:INPUT
          A:IF A<0 OR A>3 GOTO 180
370       N$=FMT(A,"9:")+N$:OPEN 1 N$ ERROR 380:GOT
          O 220
380     > PRINT:PRINT TAB(T);:GOSUB 110:PRINT "DISK
          ERROR - ";ERR$:GOSUB 130:PRINT TAB(T); "C
          orrect and ";:GOSUB 140:GOTO 360
390     > GOSUB 90:A$="PROGRAM TERMINATED":PRINT TA
          B(FNA(A$));:GOSUB 110:PRINT A$:GOSUB 130:
          PRINT:PRINT:PRINT:END
400       !   PROGRAM COMES HERE FOR INITIALIZATION
410     > DIM C$(1),A$(250),G$(2,81),B$(80):GOSUB 1
          000
420       !   SCREEN WIDTH
430       W=80
440       W%=W-1
450       !   SCREEN HEIGHT
460       H=24
470       DEF FAA=16R4E:DEF FNA(X$)=INT((W-LEN(X$))
          /2):Y$=CHAR$(255):STRING Y$:T=INT(W*.25)
480       A$="INSTRUCTION FILE GENERATOR":GOSUB 90:
          PRINT TAB(FNA(A$));A$:PRINT
490       PRINT "This program generates a file cont
          aining the instructions for":PRINT "opera
          ting a program. The file is read by calli
          ng INSTR.READ":PRINT "from your program."
          :PRINT
500       PRINT "INSTR.READ and the INSTRUCTION FIL
          E must reside on drive 0 for":PRINT "prop
          er operation. INSTR.READ is called with a
          PLOADG ":PRINT "'INSTR.READ' within the c
          alling program. If you wish to return"
510       PRINT "to the proper program after the in
          structions are displayed,":PRINT "follow
          the remarks contained within the listing
          for INSTR.READ.":PRINT:PRINT TAB(T);:GOSU
          B 140
520       !   MAIN MENU
530     > GOSUB 90:PRINT TAB(T); "Functions Availab
          le":PRINT
540       PRINT TAB(T); " 1) Create New Instruction
          File"
550       PRINT TAB(T); " 2) Add To Existing Instru
          ction File"
560       PRINT TAB(T); " 3) Terminate Program"
570       PRINT:PRINT TAB(T); "Select Function You
          Desire";:A$=FAA(1):IF A$=CHAR$(3) STOP
580       IF A$<"1" OR A$>"3" GOTO 530
590       A=VAL(A$):ON A GOTO 160,350,390
990       !   LOADS MODIFIED INKEY$ ROUTINE STARTIN
          G AT LOCATION 16R4E
1000 >*   A$="3E0332A0013E0132A10132A201":B$="CD7B0
          77832A301CD7F07000000C9":A%=16R4E:B%=1:FO
          R I=0 TO 12:POKE(A%+I)=VAL("16R"+MID$(A$,
          B%,2)):B%=B%+2:NEXT I
1005 <*   B%=1:FOR I=13 TO 26:POKE(A%+I)=VAL("16R"+
          MID$(B$,B%,2)):B%=B%+2:NEXT I:RETURN
1010      ! (C) 1981 BONJOEL ENTERPRISES
1020      ! BY JOEL SHAPIRO
```

Title: INSTR.READ

```
  10      GOTO 60
  20      !   CLEAR SCREEN S/R
  30 ><   PRINT CHAR$(26):RETURN
  40 ><   INPUT"Press RETURN When Ready ";A$:RETURN
  50 ><   PRINT:PRINT TAB(T);:GOSUB 40:GOSUB 30:RET
          URN
  60    > DIM A$(250),G$(2,81)
  70      !   SCREEN WIDTH
  80      W=80
  90      !   SCREEN HEIGHT
 100      H=24
 110      STRING CHAR$(255):T=INT(W*.25)
 120      !   GET THE CODE FOR THE CALLING PROGRAM
 130      P=PEEK(16R4D)
 140      !   SELECT THE CORRECT FILE AND STARTING
          RECORD NUMBER
 150      IF P=1 N$="INSTR-1":R%=5
 160      IF P=2 N$="INSTR-2":R%=10
 170      IF P=5 N$="INSTRUCT":R%=1
 180      !   OPEN THE FILE
 190    > OPEN 1 N$ ERROR 200 END 260:GETSEEK(1)=R%
          :GOTO 220
 200    > PRINT:PRINT "DISK ERROR - ERR$":PRINT:PRI
          NT "Correct and ";:GOSUB 40:GOTO 190
 210      !   OPEN THE TERMINAL - SET FOR SCREEN HE
          IGHT
 220    > OPEN 9 "*T" PAGESIZE H-3 ENDPAGE 50:GOSUB
          30
 230      !   READ AND DISPLAY FILE DATA
 240    > GET 1 G$(0),G$(1),G$(2):PUT 9 G$(0):PUT 9
          G$(1):PUT 9 G$(2):GOTO 240
 250      !   CLOSE AFTER FILE IS READ
 260    > ENDPAGE 9:GOSUB 50:CLOSE 1:CLOSE 9
 270      !   RETURN TO DEFAULT DELIMITER (IF REQUI
          RED)
 280      STRING","
 290      !   SELECT CORRECT PROGRAM FOR RETURN
 300      IF P=1 PLOADG "PROGRAM-1"
 310      IF P=2 PLOADG "PROGRAM-2"
 320      IF P=5 PLOADG "MAIN.PGM"
 330      ! (C) 1981 BONJOEL ENTERPRISES
 340      ! BY JOEL SHAPIRO
1000      !*****************************************
1010      !   The operating program calls this one
1020      !   whenever instructions are required by
1030      !   the operator. The proper instruction
1040      !   file and/or the proper starting
1050      !   record can be selected by a POKEd
1060      !   value from the calling program.
1065      !   Use a decimal value between 0 and
1070      !   255. This program used location 16R4D
1080      !   for the POKE and subsequent PEEK. The
1090      !   statement in the calling program
1095      !   should read as follows;
1100      !
1110      !   POKE(16R4D)=5:PLOADG"INSTR.READ"
1120      !
1130      !   This program will then select the
1140      !   correct file and starting record for
1150      !   the instructions. As you can see,
1155      !   this value is used to reload the
1160      !   calling program after the instruc-
1170      !   tions have been displayed.
1180      !
1190      !   If you wish, the value can be sensed
1200      !   by the calling program and used to
1210      !   return to a specific program segment.
1220      !
1230      !   Using a single file and selecting the
1240      !   starting record will conserve disk
1250      !   space when many separate instruction
1255      !   groups are required.  This feature
1260      !   will also allow you to have your
1270      !   general instructions in many files
1280      !   and select those you wish to use
1285      !   for a particular program. This will
1290      !   save you duplicating instructions for
1300      !   rather general applications.
1310      !*****************************************
```

INTERRUPT HARDWARE FOR REAL TIME CLOCK

by Howard Rowland WB1AJX
79 Ivan Street Apt 65
North Providence RI 02904

REAL TIME CLOCK INTERRUPT

This article describes the implementation of a simple real time clock on a Micropolis based system.

A brief description of how the CPU handles interrupts, then a description of hardware that can be used will be presented.

8080 INTERRUPTS

When the 8080 receives an interrupt request while interrupts are enabled, it finishes the current instruction and starts an interrupt acknowledge cycle. This cycle is identified by the output of the INTA status signal on S-100 pin 96. During this cycle, the interrupt hardware in the system will place an instruction on the data in bus. Typically, this is a restart instruction, which is a one byte call to a set address in low memory. After receiving this instruction, the processor will push the contents of the program counter onto the stack. A return executed at the end of the interrupt service routine will then continue execution at the interrupted point. The 8 restart instructions and the address called by them are listed below:

Restart	Opcode	Adress Called
0	C7H	0000H
1	CFH	0008H
2	D7H	0010H
3	DFH	0018H
4	E7H	0020H
5	EFH	0028H
6	F7H	0030H
7	FFH	0038H

Restart 7 is interesting for a simple interrupt implementation, as will be seen later.

Since an interrupt can occur at any time (if enabled) the interrupt service routine must save any flags and registers it will use and restore them before returning. This is done with the PUSH and POP instructions. Interrupts should be reenabled before returning, because the processor disables further interrupts when it acknowledges one.

Since the Micropolis disk system is software controlled, it uses software timing loops in its operation. Any interrupts during one of these loops or during a disk read or write can cause errors. For this reason, the software disk driver disables interrupts when it is called, preventing any interruptions in its execution. Interupts must be reenabled at the completion of the sector data transfer when using interrupts in the system. Space is provided for placing an EI (FBH) instruction in the RES module. This byte, normally a NOP (00H) is located at 0A04H in version 4 of the Micropolis PDS. Any interrupt software initialization routines must change this location to the enable interrupt instruction so the disk driver will reenable interrupts at the completion of the sector transfer.

REAL TIME INTERRUPT HARDWARE

This circuit divides the 2 MHZ bus clock down to 1/60 hz (1 min period) and uses this divided rate to generate an interrupt. 7 divide-by-10 counters and one divide-by-12 counter are used to get the required divide by 1.2 * 108.

Counters A1 through A7 consist of two sections, a divide by two and a divide by five. The output of the divide by two (pin 12) is fed into the divide by five (pin 1). A8 is similar except the second section is a divide by six instead of a divide by five. The reset to zero inputs on A1 through A8 are grounded and the reset to nine inputs on A1 through A7 are grounded to allow the counters to count normally.

The output of the counter chain (A8 pin 8) feeds a D-latch which latches the interrupt request.

A rising edge on the counter output (A8 pin 8) will clock the "1" on the D input (A9 pin 3) to the Q output (A9 pin 5). The Q output of the latch drives the input of an inverter (A10 pin 1) whose output drives the PINT (S100 pin 73) line. This line is active level low so the inverter provides the proper signal polarity as well as isolating the latch output from the bus.

An open collector inverter is used to drive this line as it is pulled up on the processor card in most systems. This allows any other possible interrupts to be OR tied together. The software must determine the cause of the interrupt if multiple interrupts are used this way.

When the CPU acknowledges the interrupt, SINTA is asserted on S-100 pin 96 which will clear the interrupt. This is inverted to provide the proper polarity for the clear input on the interrupt latch.

The next rising edge on the input of the D-latch will then generate another interrupt request 1 minute later.

Systems implementing a vectored interrupt scheme may use one of the lines VI0 to VI7 (S-100 pins 4-11) instead of PINT. The vectored interrupt hardware will then generate the appropriate restart instruction on the interrupt acknowledge cycle. The service routine for the real time clock must then have a jump instruction at the appropriate restart vector location.

If multiple interrupts are employed in the system, a more sophisticated means of reseting the real time clock interrupt must be employed. This is to prevent the interrupt acknowledge cycle from another interrupt from clearing a pending clock interrupt. This can be accomplished by some additional hardware and using an OUT instruction to clear the interrupt in the clock service routine.

The input to the counter chain is taken from the clock line, S-100 pin 49. The required frequency is 2 Mhz for this counter. This line was used rather than phase 2 because phase 2 (pin 24) is at the processor speed, which is 4 MHZ in some systems. Clock, pin 49 is specified (by the IEEE) to always be 2 MHZ, independent of processor speed. This should be verified, as some older CPU cards put out a clock at the processor speed rather than 2 MHZ.

In my system, this is the only interrupt used so nothing drives the data in bus during the interrupt acknowledge cycle. The processor therfore reads an 0FFH which is a Restart-7, which is the interrupt vector I used in the Real Time program. The data in bus must be pulled up to 5 volts through resisters to assure it will read as all ones when it is undriven.

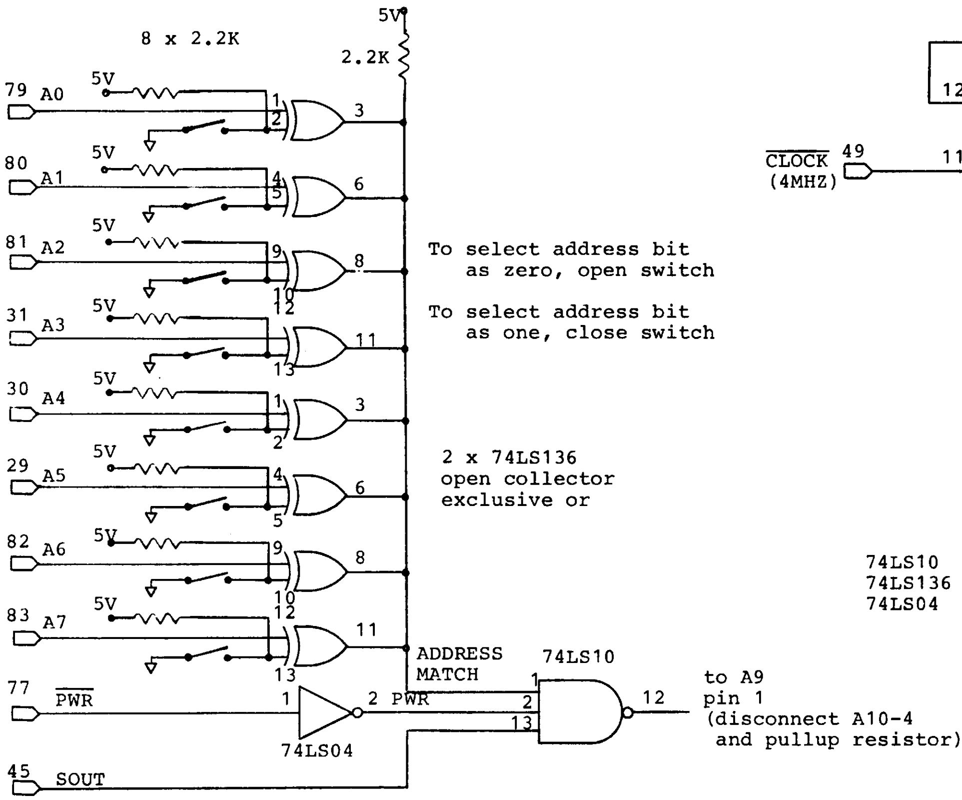
This allows a simple implementation when a single interrupt is used in the system.

....

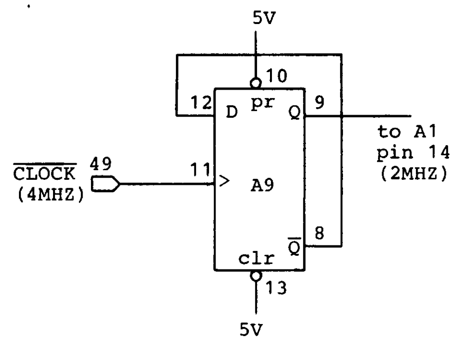
INDEX TO MUG'S FIRST YEAR

Ken Findlay (937 Briar Hill Ave., Toronto, Ontario M6B 1M1) has researched all of the MUG's first 12 issues, and coded the information by program name/topic author, program/article type, and category. He then wrote a set of programs to input the information to a data base, to manipulate it, and to print it. Although the printout can be in many forms, what follows is an index in alphabetical order by programname/topic.

HARDWARE TO CLEAR CLOCK INTERRUPT BY AN OUT XX INSTRUCTION

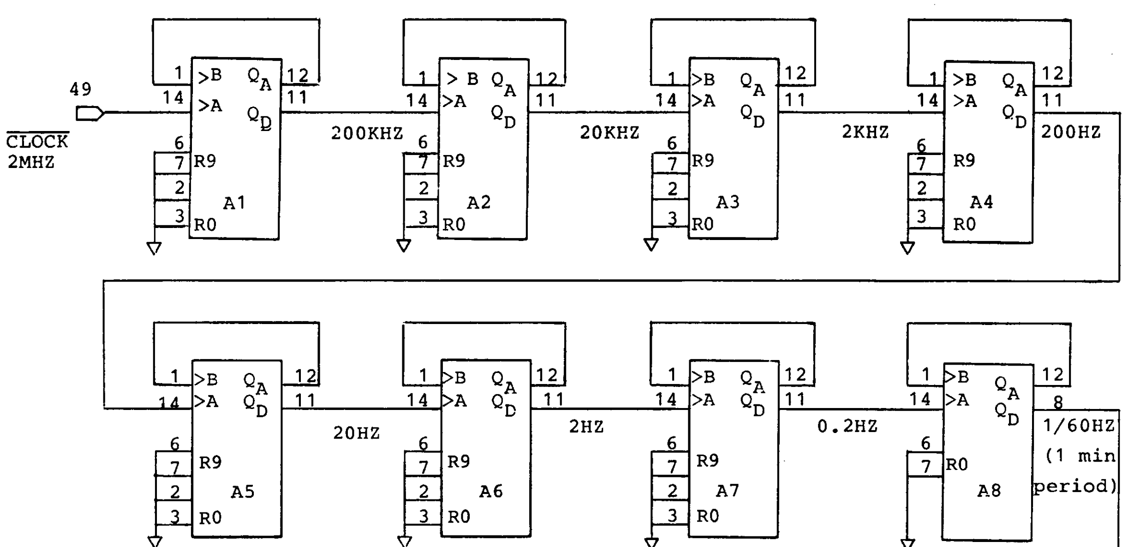


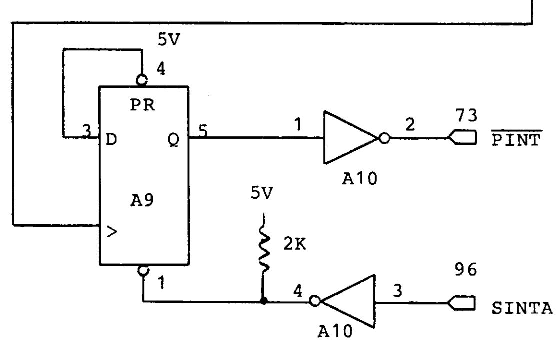
DIVIDE BY 2 FOR 4MHZ CLOCK



	5V	GND
74LS10	14	7
74LS136	14	7
74LS04	14	7

1 MINUTE INTERRUPT





Total / 1.2×10^8
2MHZ input = 1 min period out

A1-A7 74LS90 /10 counter
A8 74LS92 /12 counter
A9 74LS74 D Latch
A10 7405 Open Collector Invertor

IC	5V	GND
A1-A8	5	10
A9,A10	14	7

MICROPOLIS USER'S GROUP NEWSLETTER INDEX ISSUES 1 THROUGH 12

PGMNAME/TOPIC	CO./AUTHOR	PGM/ARTICLE TYPE	CATEGORY	VOL-PG
A-FORTH	ACROPOLIS	HIGH LEVEL LANGUAGE	COMPILER	012-03
AMORT	MUG LIBR	BASIC APPL PGM	BUSINESS	008-05
ASMTEXT*	B.RUDOW	8080 APPL PGM	TERM I/O	005-05
ASSEMBLER PGMING BOOKS		8080 PGMING	REFERENCE	008-04
ASSEMBLER PROGRAMMING*		8080 PGMING		005-01
BANKING PGM		8080 APPL PGM	BUSINESS	012-05
BASIC LOAD+GO	MICROPOLIS	8080 UTILITY PGM		006-10
BASIC PGMING		BASIC PGMING		012-04
BASIC PGMING BOOKS		BASIC PGMING	REFERENCE	012-06
BASIC TOKENS		BASIC DOC	REFERENCE	009-03
BASIC/S COMPILER	SYSTEMATION	BASIC SYSTEM PGM	COMPILER	001-01
BASIC/S COMPILER	SYSTEMATION	BASIC SYSTEM PGM	COMPILER	006-06
BASIC/S COMPILER	SYSTEMATION	BASIC SYSTEM PGM	COMPILER	010-06
BASIC/S COMPILER	SYSTEMATION	BASIC SYSTEM PGM	COMPILER	012-01
BOOKKEEPING	DATASMITH	BASIC APPL PGM	BUSINESS	006-03
BSS	INV ANAL SYS	BASIC APPL PGM	BUSINESS	008-03
CCA DMS	CUSTOM ELEC	BASIC APPL PGM	DATA BASE	003-07
CCA DMS	CUSTOM ELEC	BASIC APPL PGM	DATA BASE	004-13
CLASSIFIED ADS		GENERAL INFO		009-06
CLASSIFIED ADS		GENERAL INFO		011-06
CLASSIFIED ADS		GENERAL INFO		012-06
CLEAR SCREEN		BASIC PGM TECHNIQUE	TERM I/O	006-06
CLEAR SCREEN*	B.RUDOW	BASIC PGM TECHNIQUE	TERM I/O	001-02
CLEAR SCREEN*	B.RUDOW	BASIC PGM TECHNIQUE	TERM I/O	002-01
CLEAR SCREEN*	J.CALLAWAY	BASIC PGM TECHNIQUE	TERM I/O	006-05
CP/M		8080 SYSTEM PGM	OP SYSTEM	004-06
CP/M		8080 SYSTEM PGM	OP SYSTEM	006-01
CP/M		8080 SYSTEM PGM	OP SYSTEM	006-02
CP/M		8080 SYSTEM PGM	OP SYSTEM	008-05
CP/M		8080 SYSTEM PGM	OP SYSTEM	011-02
CP/M		8080 SYSTEM PGM	OP SYSTEM	012-03
CP/M*	S.TATTERSALL	8080 SYSTEM PGM	OP SYSTEM	010-05
CRUNCH	SYSTEMATION	BASIC PGMING AID		004-09
CRUNCH	SYSTEMATION	BASIC PGMING AID		008-07
CURSOR CONTROLS*	DAVE LAND	BASIC PGM TECHNIQUE	TERM I/O	008-01
DATABASE	J.SHAPIRO	BASIC APPL PGM	DATA BASE	003-07
DATABASE	BONJOEL	BASIC APPL PGM	DATA BASE	005-01
DATABASE TWO	BONJOEL	BASIC APPL PGM	DATA BASE	006-07
DATABASE TWO	BONJOEL	BASIC APPL PGM	DATA BASE	006-08
DATE ACCESSES*	D.O'BRIEN	BASIC PGM TECHNIQUE		009-03
DATE ACCESSES*	ED BURKHARDT	BASIC PGM TECHNIQUE		012-05
DISASSEMBLER	CUSTOM ELEC	8080 SYSTEM PGM	DISASSM	003-07
DISK DRIVE SALE	PRIORITY ONE	HARDWARE		004-12
DISK FILE ACCESS		BASIC PGM TECHNIQUE	DISK FILES	004-05
DISK I/O ERRORS		DISK MEDIA		005-10
DISK RUDIMENTS		DISK MEDIA		010-04
DISK RUDIMENTS		DISK MEDIA		011-03
EXECUTION TIME*	B.RUDOW	BASIC PGM TECHNIQUE		002-02
EXECUTION TIME*	B.RUDOW	BASIC PGM TECHNIQUE		012-04
FILE OPEN ROUTINE*	B.SMITH	BASIC PGM TECHNIQUE	DISK FILES	009-04
FINANCIAL PLANNING PGMS	SYNTAX CORP.	BASIC APPL PGMS	BUSINESS	006-04
FLASHWRITER II	VECTOR GR	HARDWARE		011-01
FLIPPY DISKS	MICRO-SERVE	HARDWARE		012-06
FMT		BASIC STATMENT	REFERENCE	009-03
FMT*	B.SMITH	BASIC STATEMENT	REFERENCE	012-01
GENSORT*	ED BURKHARDT	BASIC UTILITY PGM	SORT	007-01
GOTO*	B.RUDOW	BASIC STATEMENT	REFERENCE	002-02
GRAPHICS		BASIC PGM TECHNIQUE		010-01
GRAPHICS*	B.SMITH	BASIC PGM TECHNIQUE		011-01

MICROPOLIS USER'S GROUP NEWSLETTER INDEX ISSUES 1 THROUGH 12

PGMNAME/TOPIC	CO./AUTHOR	PGM/ARTICLE TYPE	CATEGORY	VOL-PG
HAM PGMS AVAILABLE		8080 APPL PGMS	HAM RADIO	005-12
HIGH LEVEL LANGUAGES		GENERAL INFO		005-11
HIGH LEVEL LANGUAGES		GENERAL INFO		012-02
I/O PORTS		BASIC PGM TECHNIQUE	TERM I/O	006-05
IMS	INV ANAL SYS	BASIC APPL PGM	DATA BASE	003-06
INKEY ROUTINE*	B.SMITH	8080 PGM TECHNIQUE	TERM I/O	009-04
INVENTORY ONE	BONJOEL	BASIC APPL PGM	BUSINESS	006-09
KEYBOARD INPUT*	B.RUDOW	BASIC PGM TECHNIQUE	TERM I/O	001-02
KEYBOARD INPUT*	B.RUDOW	BASIC PGM TECHNIQUE	TERM I/O	002-01
LEFT-FILL WITH ZEROS		BASIC PGM TECHNIQUE		008-07
MDOS ALTERATIONS	MICROPOLIS	8080 SYSTEM PGM	OP SYSTEM	008-09
MDOS ALTERATIONS	MICROPOLIS	8080 SYSTEM PGM	OP SYSTEM	012-06
MDOS DISK RECORD ADDR		8080 SYSTEM PGM	OP SYSTEM	009-03
MEMORY ALLOCATION		BASIC PGM TECHNIQUE		009-03
MICOPOLIS SOFTWARE DIR		REFERENCE LISTING	MENTION	004-07
MICROPOLIS HARD DISKS	MICROPOLIS	HARDWARE	REFERENCE	003-05
MICROPOLIS HARD DISKS	MICROPOLIS	HARDWARE		006-02
MICROPOLIS HARDWARE	MICROPOLIS	HARDWARE	REFERENCE	001-02
MICROPOLIS NEWS	MICROPOLIS	GENERAL INFO		003-06
MICROPOLIS NEWS	MICROPOLIS	GENERAL INFO		006-02
MICROPOLIS REPS	MICROPOLIS	REFERENCE LISTING	LIST	003-09
MICROPOLIS SYSTEM TIPS	MICROPOLIS	GENERAL INFO		009-02
MODEM	D.C.HAYES	HARDWARE		005-10
MODEM PGM	DAVE LOGAN	8080 SYSTEM PGM		006-01
MODEM PGM FOR SOL*	BOB BARNUM	8080 SYSTEM PGM		007-03
MODI-MODII CONVERSION		HARDWARE		002-01
MUG DIRECTORY		REFERENCE LISTING	MENTION	006-06
MUG LIBRARY		GENERAL INFO		006-01
MUG LIBRARY		DISK01 DIR LISTING	REFERENCE	009-01
MUG LIBRARY		GENERAL INFO		009-01
MUG LIBRARY		GENERAL INFO		011-02
MUG LIBRARY		DISK03 DIR LISTING	REFERENCE	011-04
MUG LIBRARY		DISK02 DIR LISTING	REFERENCE	011-04
MUG LIBRARY		GENERAL INFO	PRICE LIST	011-05
MUG MEMBERSHIP DIR		GENERAL INFO		010-04
MUG NEWSLETTER		GENERAL INFO		009-01
MUG OBJECTIVES		GENERAL INFO		001-01
MUG OBJECTIVES		GENERAL INFO		006-01
MUG OBJECTIVES		GENERAL INFO		007-01
NEVADA COBOL	ELLIS COMP	HIGH LEVEL LANGUAGE	COMPILER	012-03
P/DIM	SYSTEMATION	BASIC UTILITY PGM		010-02
PAS	INV ANAL SYS	BASIC APPL PGM	PROP MGMT	003-06
PAS	INV ANAL SYS	BASIC APPL PGM	PROP MGMT	008-03
PAYROLL	DATASMITH	BASIC APPL PGM	BUSINESS	006-03
PGMING AIDS AVAILABLE	DATASMITH	BASIC PGMING AIDS	REFERENCE	006-03
PMS	CUSTOM ELEC	BASIC APPL PGM	PROP MGMT	003-07
PMS II	INV ANAL SYS	BASIC APPL PGM	PROP MGMT	008-03
QUIKSORT*	B.RUDOW	BASIC UTILITY PGM	SORT	003-03
R'S AND E'S		BASIC PGM TECHNIQUE		011-02
REACT	BONJOEL	BASIC APPL PGM	BUSINESS	008-03
READING DISK DIR	ED BURKHARDT	BASIC UTILITY PGM	DISK DIR	012-04
RES-CONDENSED		8080 SYSTEM PGM	OP SYSTEM	006-06
ROUNDING*	B.RUDOW	BASIC PGM TECHNIQUE		002-03
ROUNDING*	J.CALLAWAY	BASIC PGM TECHNIQUE		006-05
ROUNDING*	DAVE LAND	BASIC PGM TECHNIQUE		008-01
SAVING VARIABLES		BASIC PGM TECHNIQUE		004-08
SAVING VARIABLES	B.MITCHELL	BASIC PGM TECHNIQUE		008-08
SAVING VARIABLES*	B.RUDOW	BASIC PGM TECHNIQUE		008-08
SAVING VARIABLES*		BASIC UTILITY PGM		010-02

MICROPOLIS USER'S GROUP NEWSLETTER INDEX ISSUES 1 THROUGH 12

PGMNAME/TOPIC	CO./AUTHOR	PGM/ARTICLE TYPE	CATEGORY	VOL-PG
SCREEN DISPLAY TEST*	J.CALLAWAY	BASIC UTILITY PGM	TERM I/O	006-06
SCREEN DISPLAY TEST*	B.RUDOW	BASIC UTILITY PGM	TERM I/O	006-07
SIZES		BASIC STATEMENT	REFERENCE	002-03
SOFTWARE VENDOR DIR	MICRO-SERVE	REFERENCE LISTING	LIST	005-05
SOFTWARE VENDOR DIR	MICRO-SERVE	REFERENCE LISTING	UPDATES	006-02
SOFTWARE VENDOR DIR	MICRO-SERVE	REFERENCE LISTING	MENTION	007-10
SOFTWARE VENDOR DIR	MICRO-SERVE	REFERENCE LISTING	USAGE	011-03
SORT RETAINING SEQ		BASIC PGM TECHNIQUE	SORT	003-03
SORT/A	SYSTEMATION	8080 UTILITY PGM	SORT	001-01
SORT/A	SYSTEMATION	8080 UTILITY PGM	SORT	003-01
SORT/B	SYSTEMATION	BASIC-80 UTILITY PGM	SORT	010-05
SUBROUTINE LIBRARY		BASIC PGM TECHNIQUE		010-01
SYSTEMATION DISCOUNTS	SYSTEMATION	GENERAL INFO		009-01
TAPEREC		BASIC APPL PGM	MUSIC	011-06
TAX PGMS	SYNTAX CORP.	BASIC APPL PGMS	BUSINESS	006-04
TICTACTOE	MUG LIBR	BASIC APPL PGM	GAME	008-06
TX*	B.RUDOW	BASIC UTILITY PGM	DISK FILES	004-01
UNPROTECT	SYSTEMATION	CP/M UTILITY PGM		009-05
UTILITY PGMS AVAILABLE	BONJOEL	GENERAL INFO		006-07
UTILITY PGMS AVAILABLE	SYSTEMATION	UTILITY PGMS	REFERENCE	007-09
UTL-1 PGM PACKAGE	SYSTEMATION	8080 UTILITY PGMS	DISK FILES	005-12
VARIABLE ALLOCATION*	B.SMITH	BASIC PGM TECHNIQUE		008-06
VECTOR GRAPHIC SYSTEM	VEC GRAPH	HARDWARE		003-05
WAMSORT	BONJOEL	8080 UTILITY PGM	SORT	006-09
XREF	SYSTEMATION	BASIC PGMING AID		004-10
YES/NO INPUT RESPONSES*	B.SMITH	BASIC PGM TECHNIQUE		009-04
YES/NO INPUT RESPONSES*	ED BURKHARDT	BASIC PGM TECHNIQUE		012-05

The programs and data file will be put on Library Disk 7 - which isn't done yet. You'll be able to use the programs for indexing magazine articles, records, books - whatever. They are a very useful set of routines.

....

LIBRARY DISKS 4 & 5??

I have succeeded in thoughly confusing a lot of people last month by referring to library disks 4 and 5. What I did, without telling you, was to make the Membership Directory be Disk 4, and the S/W Vendor's Directory be Disk 5.

....

CLASSIFIED

WANTED: People to input names and addresses at five cents each. I supply the programs, the disk, and the data. You mail the completed input back to me. Estimated need is for the first quarter of 1982. Tim Dawalt, P.O. Box 253, Lightstreet PA 17839, 717/784-4496

WANTED: A person to perform a disk conversion from 5 1/4 Micropolis CP/M to 8" Standard (and/or double-density) CP/M. Martin Rothstein, 21 E. 40 St., NY NY 10016, 212/683-5310

NOTICE: Micropolis will present a floppy disk maintenance course on Oct. 7 & 8 at the Los Angeles Airport Marriott. Cost is $200 for first student, $175 for additional students from the same company. Fees include lunches, manuals and diagnostic software. Contact Bob Louch at 213/709-3300.

Published Monthly by the MUG
Subscription rates:
U.S., Canada, Mexico; $18/year: Other, $25/year

FIRST CLASS MAIL

FIRST CLASS MAIL

MICROPOLIS USERS GROUP
Buzz Rudow, Editor
604 Springwood Circle
Huntsville AL 35803
(205) 883-2621

FIRST CLASS MAIL