WADUZITDO:

How To Write a Language in 256 Words or Less

Larry Kheriaty
Computer Center
Western Washington University
Bellingham WA 98225

Every computer owner likes to show his or her microcomputer to friends. The first question the friends usually ask is, "What does it do?" The software system presented here demonstrates what a computer can do in a manner simple enough for almost anyone to understand. Even if you have a larger, more capable system, it is often worthwhile to be able to demonstrate something that can be accomplished on a smaller scale. WADUZITDO is small enough to run on almost any microcomputer yet it allows even the novice user to make the computer "do something."

WADUZITDO is a complete high level language processor that fits in less than 256 bytes on either a 6800 or 8080 based system. The only other requirement is some kind of terminal. The system includes a text editor to allow a program to be entered and modified, and an interpreter to execute the program. The only external routines needed are single character input and single character output such as those provided by most system monitors.

The object of WADUZITDO is to run simple conversational programs. There are just five statement types, roughly derived from the PILOT language. To keep it small only the most essential capabilities are available. This also makes programming very easy. In fact, only a few minutes after my unsuspecting spouse had asked, "What does it do?", she had written the interactive dialogue program in listing 1 to help me make out a list of acceptable birthday gifts!

Programming in WADUZITDO is straightforward and uncomplicated. For example, to direct the computer to display a line of text on the terminal you use the *type* statement. The following example shows the format of the *type* statement.

```
T:WHAT COULD BE EASIER
THAN THIS?
```

The T is the operation code for *type*. A colon always follows the operation code. The text after the colon is displayed exactly as shown.

The *accept* statement allows the program to receive one input character from the terminal keyboard. Normally it is used after a *type* that asks for a response. For example:

```
T:CAN YOU TELL ME WHAT 2 + 3
EQUALS?

A:
```

The *accept* statement is just the A operation code followed by a colon. When it is encountered execution pauses until the user keys in any single character. Then the input character is saved internally for use in subsequent *match* statements.

The *match* statement is used to test the character entered by the user on the previous *accept*. *Match* is coded as an M (the operation code), followed by a colon and one character. The character in the statement is compared to the last character entered by the user. The result of the comparison is recorded internally in the match flag: Y if the match is equal, N if it is not equal.

Once set the match flag can be used to conditionally execute or skip any subsequent statement. This is done by placing either a Y (yes) or N (no) immediately before any operation code. If the Y or N is the same as the match flag the statement is executed, otherwise it is skipped. An elaboration of the previous example illustrates the use of *match*.

```
T:WHAT IS 2 + 3?

A:

M:5

YT:FIVE, RIGHT.

NT:NO, THE ANSWER IS 5.
```

Normally statements are executed se-

quentially. The *jump* statement is used to alter the normal sequence. The format of the *jump* statement is J, followed by a colon, and a number from zero to nine. The statement J:0 causes a branch back to the last *accept* statement executed. Execution resumes from that statement. The J:0 statement can be used to allow the user to reanswer a previous question. For example:

```
T:HOW MANY FEET IN A YARD?

A:

M:3

YT:RIGHT.

NT:WRONG STUPID, TRY AGAIN.

NJ:0
```

The second form of the jump makes use of program markers. A program marker is an asterisk, *, preceding any statement. The statement J:n, where n is a number from 1 to 9, causes a branch to the nth program marker forward from the *jump*. This form of the *jump* is shown in the sample program in listing 2 which plays NIM.

The last type of statement is *stop*. This statement merely terminates execution of the program and returns control to the program editor. The format of the *stop* statement is S:

To increase the versatility of the language the S: statement can, at the user's option, be made to call a user written machine language subroutine from within the WADUZITDO program. To do this requires a one statement modification to the system which is detailed below. If you choose to make this modification you can consider S: to be the operation code for *subroutine* rather than *stop*. The format of the *subroutine* statement is S:x where x is any single character which serves as a parameter to the user written program. The value x will be stored in register A in both the 6800 and 8080 version. It can be used to select different functions to be performed by the program.

During execution any statement which does not fit the syntax of one of the five statement types is printed in its entirety, then execution resumes normally with the next statement. Table 1 summarizes the WADUZITDO instruction set.

When WADUZITDO is first entered con-

```
T:IT IS BIRTHDAY LIST TIME.
T:THE PURPOSE OF THIS PROGRAM IS TO
T:DETERMINE WHAT GIFTS ARE ACCEPTABLE.
T:TYPE THE CODE LETTER ASSOCIATED WITH
T:THE POTENTIAL GIFT IDEA...
T:  A  HOME APPLIANCE
T:  B  SOMETHING BORING
T:  C  ITEM OF CLOTHING
T:  D  SOMETHING DECORATIVE FOR THE HOUSE
T:  G  GARBAGE DISPOSAL
T:  M  MY OWN COMPUTER
A:
M:A
YT:UNACCEPTABLE.
M:B
YT:NO WAY.
M:C
YT:ACCEPTABLE IF NOT UGLY.
M:D
YT:OKAY IF CHOSEN WITH GOOD TASTE
YT:SO AS NOT TO BE TACKY.
M:G
YT:YEAH !
M:M
YT:THE LAST THING IN THE WORLD
YT:I WOULD EVER WANT.
NM:A
NM:B
NM:C
NM:D
NM:G
NT:CANT YOU READ FOOL, THAT IS NOT
NT:ONE OF THE CHOICES.
NT:TRY A,B,C,D,G OR M
J:0
```

Listing 1: WADUZITDO program written by a non-computer person. Notice the last line of the program, the J:0 command. This instruction will make the program execution jump back to the accept statement to try another input.

```
T:LETS PLAY NIM WITH 7 PEBBLES.
T:WE TAKE TURNS TAKING 1, OR 3.
T:THE LAST ONE TO TAKE ONE LOSES.
T:THERE ARE 7, HOW MANY ?
A:
M:1
YJ:1
M:2
YJ:2
M:3
YJ:6
T:YOU CAN TAKE ONLY 1,2, OR 3.
J:0
*T:THAT LEAVES 6, I TAKE 1 LEAVING 5.
T:HOW MANY ?
A:
M:1
YJ:5
M:2
YJ:4
M:3
YJ:3
T:YOU MUST TAKE 1,2 OR 3.
J:0
*T:THAT LEAVES 5, I TAKE 1 LEAVING 4.
T:HOW MANY ?
A:
M:1
YJ:3
M:2
YJ:2
M:3
YJ:1
T:YOU MUST TAKE 1,2 OR 3 ONLY .
J:0
*T:THAT LEAVES THE LAST ONE.
T:I TAKE IT ... YOU WIN.
J:5
*T:THAT LEAVES 2, I TAKE 1 LEAVING 1.
J:3
*T:THAT LEAVES 3, I TAKE 2 LEAVING 1.
J:2
*T:THAT LEAVES 4, I TAKE 3 LEAVING 1.
*T:HOW MANY ?
A:
M:1
NT:YOU HAVE NO CHOICE BUT TO TAKE 1.
NT:HOW MANY ?
NJ:0
T:YOU JUST TOOK THE LAST ONE ... I WIN.
*T:TO PLAY AGAIN PUSH THE DOLLAR SIGN.
S:
```

Listing 2: A NIM playing program. This program demonstrates the jumping capability of the language.

STATEMENT	FORMAT	WHAT IT DOES
type	T:text	Display text on the terminal.
accept	A:	Input one character from the terminal keyboard.
match	M:x	Compare x to last input character and set match flag to Y if equal, N if not equal.
jump	J:n	If n=0 jump to last *accept*. If n=1 thru 9 jump to nth program marker forward from the J.
stop	S:	Terminate program and return to text editor.
subroutine	S:x	Call user machine language program (requires modification).
conditionals	Y N	May precede any operation code. Execute only if match flag is Y. Execute only if match flag is N.
program marker	*	May precede any statement, serves as a *jump* destination.

Table 1: Program instructions for the WADUZITDO language.

EDIT CHARACTER	HEX	MEANING
$	24	Start execution.
\	5C	Move edit pointer to program start.
/	2F	Display next line of program.
%	25	Pad inserted line with nulls.
bs or -	08 or 5F	Backspace to correct typing error.
cr	0D	End of statement.
any other		Character stored in program and edit pointer advances.

Table 2: Editing characters used by the built-in text editor.

trol is passed to the program editor which is used to enter or alter source programs. Also an internal program pointer, called LOC, is automatically set to the beginning of the source area. As each statement is entered on the keyboard the characters are stored and the internal pointer advances. Typing errors may be corrected by entering a backspace and the correct character. To reset the pointer to the start of the program enter a backslash, \. To display the next line of the program enter the mirror image of the reset slash, /. To replace a line, display each line up to but not including the one to be replaced, then enter the new line. The new line should be no longer than the line it replaces. If it is longer, the next line of text is also overwritten. End the replacement line with a percent key rather than a carriage return. The % causes null characters to be stored as filler up to the start of the next line. To begin execution of the program enter a dollar sign, $. (The editing commands are summarized in table 2.)

If you already have a good text editor in your system it may be used instead of the one included. Each statement is variable length, terminated by a carriage return character. All other control characters between statements are ignored.

Complete 6800 and 8080 assembly listings containing source and object code are included to simplify implementation on your system. The 6800 version in listing 3 uses the MIKBUG monitor; the 8080 version in listing 4 uses the SOLOS/CUTER monitor. If you have one of these two system monitors you need not modify the program at all.

The entry point to the system is at location zero. Upon entry the stack pointer is assumed set to address some scratchpad memory area large enough to accommodate a few levels of call. In MIKBUG or SOLOS/CUTER, as with most system monitors, this is handled automatically by the GO or EXEC command. The 2 byte value stored in LOC (hexadecimal 100) must point to the place where the user program is to be stored. In the assembly listings note that this value is shown as hexadecimal 0106, the first location not occupied by the system.

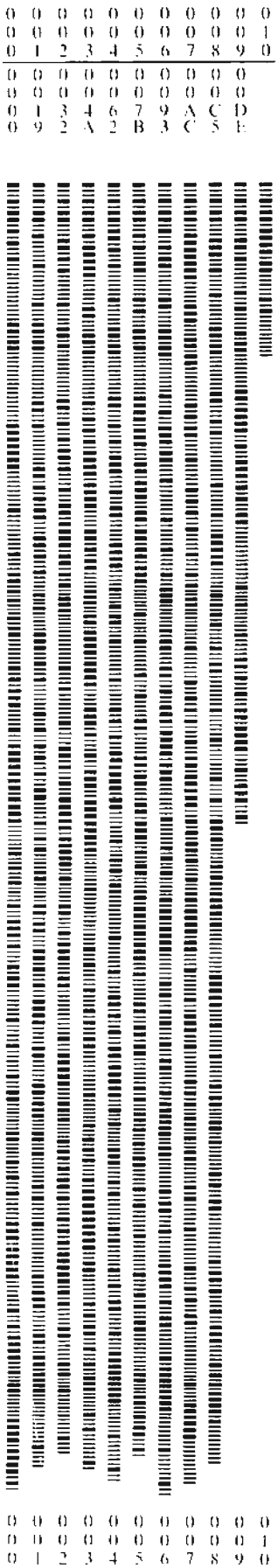
If you don't have one of the above monitors you must supply character input and character output routines and change the references to IN and OUT to address these routines. In the listings you will find one reference to IN and one to OUT which needs to be changed. If your terminal requires a delay after each carriage return you can set the number of null padding characters by a one byte modification to the statement labeled PLF.

Any of the special characters used by the text editor ($, %, \, /, bs) can be easily changed to another more convenient character on your keyboard.

As shown in the assembly listings the S: statement halts execution by branching to the text editor. If you don't modify this you can treat it as a *stop* statement. To use it as a subroutine call you must modify the JMP SUB instruction to be a JSR or CALL (depending on the system) to the appropriate address. Upon entry to the subroutine

Text continued after listings on page 173

Figure 1: Absolute loader format representation of the 6800 WADUZITDO program of listing 3.



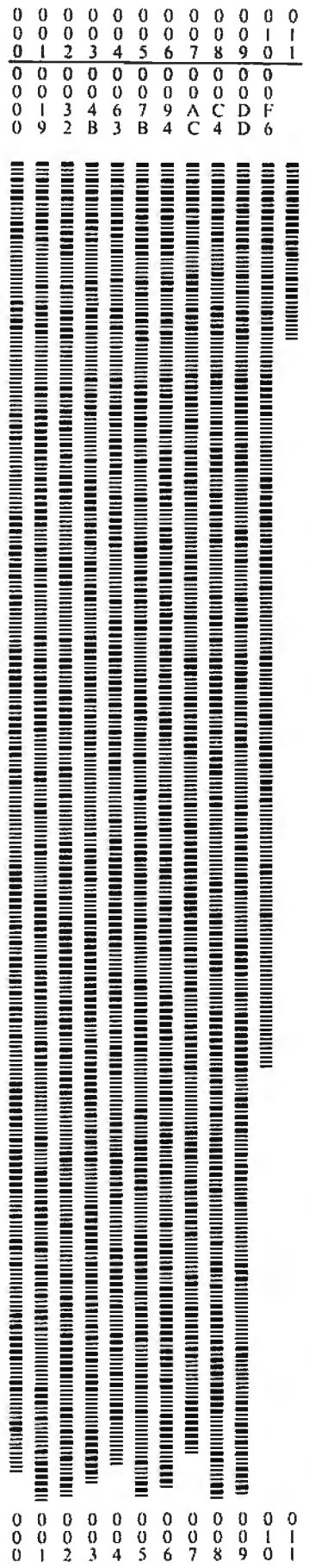

```
                        *              WADUZITDO
                        *       6800 VERSION BY LARRY KHERIATY
                        *
                        *    MIKBUG SUBROUTINES USED
                        IN      EQU   $E1AC      INPUT FROM KEYBOARD TO ACCA
                        OUT     EQU   $E1D1      OUTPUT FROM ACCA TO TERMINAL
                                ORG   $0000
                        SUB     EQU   $0000      USER SUBR START (CAN BE MODIFIED)
                        *   ENTER SYSTEM AT LOCATION 0 WITH STACK POINTER PRESET
                        *   TO SCRATCH PAD RAM ENOUGH FOR A FEW LEVELS OF CALL
0000 FE 0100    START   LDX   LOC         SOURCE PROGRAM AREA START
0003 8D 45      EGET    BSR   JIN         ACCEPT SOURCE CHAR
0005 81 5C              CMP A #$5C        \  ?
0007 27 F7              BEQ   START       YES, BACK UP TO PROGRAM START
                        *
0009 81 24              CMP A #$24        $  ?
000B 27 45              BEQ   EXEC        YES, GO EXECUTE THE PROGRAM
                        *
000D 81 08              CMP A #$08        BS  ?
000F 26 03              BNE   DIS         NO
0011 09                 DEX               YES, BACK UP ONE IN SOURCE
0012 20 EF              BRA   EGET        LOOP BACK
                        *   PROCESS DISPLAY OF NEXT LINE
0014 81 2F      DIS     CMP A #$2F        /  ?
0016 26 07              BNE   PAD         NO
0018 BD 00D5            JSR   PRT         GO PRINT TO CR
001B 8D 21      EPLF    BSR   PLF         PRINT LINE FEED AND NULLS
001D 20 E4              BRA   EGET        LOOP
                        *   DO LINE REPLACEMENT- PAD TO END OF STMT WITH NULLS
001F 81 25      PAD     CMP A #$25        %  ?
0021 26 12              BNE   CHAR        NO
0023 86 0D              LDA A #$0D        CR
0025 8D 27              BSR   JOUT        PRINT IT
0027 86 0D              LDA A #$0D        CR
0029 C6 40              LDA B #$40        COUNT OF 64
002B A1 00      PADL    CMP A 0,X         AT CR YET ?
002D 27 06              BEQ   CHAR        YES QUIT PADDING
002F 6F 00              CLR   0,X         PAD WITH NULL
0031 08                 INX               INCR LOC PTR
0032 5A                 DEC B             DECREMENT SAFETY COUNTER
0033 26 F6              BNE   PADL        LOOP TILL CR OR 64 NULLS
                        *   STORE ENTERED SOURCE CHAR IN PROGRAM
0035 A7 00      CHAR    STA A 0,X         CHAR TO SOURCE LOC
0037 08                 INX               MOVE LOC PTR UP ONE
0038 81 0D              CMP A #$0D        IS IT A CR ?
003A 27 DF              BEQ   EPLF        YES, ECHO A LINE FEED
003C 20 C5              BRA   EGET        NO, GET ANOTHER CHAR
                        *   SUBROUTINE TO PRINT LINE FEED TO TERMINAL
003E C6 00      PLF     LDA B #$00        NUMBER OF NULLS TO PRINT
0040 4F         PLFL    CLR A             NULL
0041 8D 0B              BSR   JOUT        WRITE A NULL
0043 5A                 DEC B             DECREMENT COUNTER
0044 2A FA              BPL   PLFL        LOOP TILL ENOUGH NULLS
0046 86 0A              LDA A #$0A        LINEFEED
0048 20 04              BRA   JOUT
                        *   NEXT FEW LINES MUST BE ALTERED IF YOU DONT USE MIKBUG
004A BD E1AC    JIN     JSR   IN          CALL CHAR INPUT ROUTINE
004D 39                 RTS               RETURN TO CALLER
004E BD E1D1    JOUT    JSR   OUT         CALL CHARACTER OUTPUT ROUTINE
0051 39                 RTS               RETURN TO CALLER
                        *
                        *   COME HERE TO BEGIN EXECUTION OF THE SOURCE PROGRAM
                        *
0052 FE 0100    EXEC    LDX   LOC         STARTING LOC OF PROGRAM
0055 09                 DEX               LESS ONE
0056 08         LOOPI   INX               ADR OF NEXT PGM BYTE
0057 A6 00      LOOP    LDA A 0,X         NEXT PGM BYTE
0059 81 2A              CMP A #$2A        *  CHAR ?
005B 2F F9              BLE   LOOPI       YES(OR IGNOREABLE CONT CHAR)
                        *
                        *   PROCESS Y OR N FLAG TESTS
005D 81 59              CMP A #$59        Y  ?
005F 27 04              BEQ   TFLG        YES
0061 81 4E              CMP A #$4E        N  ?
0063 26 0F              BNE   XA          BRANCH IF NOT A FLAG TEST
                        *
0065 08         TFLG    INX               STEP LOC OVER Y OR N
0066 B1 0105            CMP A FLG         COMPARE TO CURRENT MATCH FLAG
0067 27 EC              BEQ   LOOP        ITS EQUAL SO EXECUTE THE STMT
                        *
                        *   ITS A FLAG FAILURE, SKIP OVER THE STMT
006B 08         SKIP    INX               STEP LOC PTR
006C A6 00              LDA A 0,X         NEXT CHAR IN PGM
```

Listing 3: 6800 version of the WADUZITDO language. A dump of the MIKBUG format of WADUZITDO (shown in listing 3a, page 172) can be used for manual entry of the program. This version was run locally at BYTE using a SwTPC 6800.

Figure 2: Absolute loader format representation of the 8080 WADUZITDO program of listing 4.

```
006E 81 0D            CMP A #$0D      TO END OF STMT ?
0070 26 F9            BNE   SKIP      NOT YET, SO LOOP
0072 20 EZ            BRA   LOOPI     AT NEXT STMT, GO DO IT
                *
                *   PROCESS ACCEPT STATEMENT
0074 81 41    XA      CMP A #$41      A  ?
0076 26 11            BNE   XM        NO
0078 FF 0102          STX   LST       YES, SAVE LOC OF LAST ACCEPT
007B 8D CD            BSR   JIN       ACCEPT ONE CHAR FROM KYBD
007D B7 0104          STA A CHR       SAVE IT
0080 08               INX             MOVE OVER A
0081 86 0D    PCR     LDA A #$0D      CR
0083 8D C9            BSR   JOUT      PRINT IT
0085 8D B7            BSR   PLF       PRINT LINE FEED
0087 20 CD            BRA   LOOPI     STEP OVER : AND GO ON
                *
                *   PROCESS MATCH STMT
0089 81 4D    XM      CMP A #$4D      M  ?
008B 26 12            BNE   XJ        NO
008D 08               INX             STEP OVER M
008E 08               INX             STEP OVER :
008F A6 00            LDA A 0,X       GET MATCH CHAR
0091 C6 59            LDA B #$59      ASSUME Y
0093 B1 0104          CMP A CHR       COMP MATCH CHAR TO INPUT CHAR
0096 27 02            BEQ   MX        BRANCH IF IT MATCHES,FLG=Y
0098 C6 4E            LDA B #$4E      RESULT IS N
009A F7 0105  MX      STA B FLG       SET MATCH FLAG TO Y OR N
009D 20 B7            BRA   LOOPI     STEP OVER MATCH CHAR AND GO ON
                *
                *   PROCESS JUMP STATEMENT
009F 81 4A    XJ      CMP A #$4A      J  ?
00A1 26 17            BNE   XS        NO
00A3 E6 02            LDA B 2,X       DESTINATION

00A5 C4 0F            AND B #$0F      CLEAR ZONE
00A7 26 05            BNE   JF        ITS A JUMP FORWARD
00A9 FE 0102          LDX   LST       ZERO.. JUMP BACK TO LAST ACCEPT
00AC 20 A9            BRA   LOOP      CONTINUE FROM THERE
                *
                *   SKIP FORWARD UNTIL PASS N *-MARKERS ( N IS IN ACCB )
00AE 08       JF      INX             STEP PGM LOC
00AF A6 00            LDA A 0,X       NEXT CHAR
00B1 81 2A            CMP A #$2A      *-MARKER ?
00B3 26 F9            BNE   JF        NO, KEEP LOOPING
00B5 5A               DEC B           FOUND ONE, COUNT IT
00B6 26 F6            BNE   JF        LOOP IF NEED TO FIND MORE
00B8 20 9C            BRA   LOOPI     DESTINATION FOUND, GO EXECUTE
                *
                *   PROCESS STOP OR SUBROUTINE STATEMENT
00BA 81 53    XS      CMP A #$53      S  ?
00BC 26 0A            BNE   XT        NO
00BE 08               INX             STEP OVER S
00BF 08               INX             STEP OVER :
00C0 A6 00            LDA A 0,X       PARAMETER TO REG A
00C2 08               INX             STEP OVER PARAMETER
                *   NEXT STMT MAY BE MADE TO BE A  JSR TO USER SUBR
00C3 7E 0000          JMP   SUB       GO TO USER SUBR (OR TO EDITOR)
00C6 20 8F            BRA   LOOP      GO ON UPON RETURN FROM USER SUBR
                *
                *   PROCESS TYPE STATMENT AND SYNTAX ERRORS
00C8 81 54    XT      CMP A #$54      T  ?
00CA 26 02            BNE   TE        NO, ITS AN ERROR
00CC 08               INX             YES, STEP OVER T
00CD 08               INX             STEP OVER :
00CE 8D 05    TE      BSR   PRT       PRINT UP TO CR
00D0 BD 003E          JSR   PLF       PRINT LNE FEED
00D3 20 82            BRA   LOOP      DONE WITH T
                *
                *   SUBR TO PRINT UP TO NEXT CR
00D5 C6 40    PRT     LDA B #$40      COUNT OF 64
00D7 A6 00    PRTA    LDA A 0,X       NEXT CHAR
00D9 5A               DEC B           DECREMENT SAFETY COUNTER
00DA 27 0A            BEQ   PRTB      EXIT IF OVER 64 TILL CR
00DC BD 004E          JSR   JOUT      PRINT IT
00DF A6 00            LDA A 0,X       RELOAD CHAR TO ACCA
00E1 08               INX             STEP LOC PTR
00E2 81 0D            CMP A #$0D      CR ?
00E4 26 F1            BNE   PRTA      NOT CR, LOOP
00E6 39       PRTB    RTS             DONE, RETURN
                *
                *   ABOVE IS END OF READ ONLY PORTION OF THE PROGRAM
                *
                *   THE FOLLOWING IS CHANGEABLE DATA
                *
                        ORG   $100      MOVE TO START OF DATA AREA
0100    0106  LOC       FDB   $0106     ADDR OF SOURCE PROGRAM AREA
0102    0000  LST       FDB   0         PLACE TO SAVE LOC OF LAST A:
0104    00    CHR       FCB   0         PLACE TO SAVE LAST INPUT CHAR
0105    00    FLG       FCB   0         PLACE TO SAVE MATCH FLAG
                *
                        END
```



Listing 4: 8080 version of the WADUZITDO language. A hexadecimal dump (shown in listing 4a) is provided for manual entry. This version was run locally at BYTE using a SOL-20.

```
                *              WADUZITDO
                *       8080 VERSION BY LARRY KHERIATY
                *
                *    SOLOS/CUTER SUBROUTINES USED
                IN      EQU   0C01FH    INPUT FROM KEYBOARD TO A-REG
                OUT     EQU   0C019H    OUTPUT FROM B-REG TO TERMINAL
                *
                        ORG   0000H
                SUB     EQU   0000H     USER SUBR START (CAN BE MODIFIED)
                *   ENTER SYSTEM AT LOCATION 0 WITH STACK POINTER PRESET
                *   TO SCRATCH PAD RAM ENOUGH FOR A FEW LEVELS OF CALL
                *
0000 2A 0001    START   LHLD  LOC       SOURCE PROGRAM AREA START
0003 CD 4600    EGET    CALL  JIN       ACCEPT SOURCE CHAR
0006 FE 5C              CPI   5CH       \  ?
0008 CA 0000            JZ    START     YES, BACK UP TO PROGRAM START
                *
000B FE 24              CPI   24H       $  ?
000D CA 5200            JZ    EXEC      YES, GO EXECUTE THE PROGRAM
                *
0010 FE 5F              CPI   5FH       BS  ?
0012 C2 1900            JNZ   DIS       NO
0015 2B                 DCX   H         YES, BACK UP ONE IN SOURCE
0016 C3 0300            JMP   EGET      LOOP BACK
                *
                *   PROCESS DISPLAY OF NEXT LINE
0019 FE 2F      DIS     CPI   2FH       /  ?
001B C2 2400            JNZ   PAD       NO
001E CD DF00            CALL  PRT       GO PRINT TO CR
0021 C3 0300            JMP   EGET      LOOP
                *
                *   DO LINE REPLACEMENT- PAD TO END OF STMT WITH NULLS
0024 FE 25      PAD     CPI   25H       %  ?
0026 C2 3C00            JNZ   CHAR      NO
0029 06 0D              MVI   B,0DH     CR
002B 78                 MOV   A,B       CR TO A ALSO
002C CD 4D00            CALL  JOUT      PRINT IT
002F 0E 40              MVI   C,40H     COUNT OF 64
0031 BE         PADL    CMP   M         AT CR YET ?
0032 CA 3C00            JZ    CHAR      YES QUIT PADDING
0035 36 00              MVI   M,00H     PAD WITH NULL
0037 23                 INX   H         INCR LOC PTR
0038 0D                 DCR   C         DECREMENT SAFETY COUNTER
0039 C2 3100            JNZ   PADL      LOOP TILL CR OR 64 NULLS
                *
                *   STORE ENTERED SOURCE CHAR IN PROGRAM
003C 77         CHAR    MOV   M,A       CHAR TO SOURCE LOC
003D 23                 INX   H         MOVE LOC PTR UP ONE
003E FE 0D              CPI   0DH       IS IT A CR ?
0040 CC F000            CZ    PLF       YES, ECHO A LINE FEED
0043 C3 0300            JMP   EGET      NO, GET ANOTHER CHAR
                *
                *   CHANGE NEXT FEW LINES IF YOU DONT USE SOLOS/CUTER
0046 CD 1FC0    JIN     CALL  IN        CALL CHAR INPUT ROUTINE
0049 CA 4600            JZ    JIN       TRY AGAIN IF NO CHAR YET THERE
004C 47                 MOV   B,A       PREPARE TO ECHO THE CHAR
004D CD 19C0    JOUT    CALL  OUT       CALL CHARACTER OUTPUT ROUTINE
0050 78                 MOV   A,B       RESTORE JIN CHAR TO A
0051 C9                 RET             RETURN TO CALLER
                *
                *   COME HERE TO BEGIN EXECUTION OF THE SOURCE PROGRAM
                *
0052 2A 0001    EXEC    LHLD  LOC       STARTING LOC OF PROGRAM
0055 2B                 DCX   H         LESS ONE
0056 23         LOOPI   INX   H         ADR OF NEXT PGM BYTE
0057 7E         LOOP    MOV   A,M       NEXT PGM BYTE
0058 FE 2B              CPI   2BH       *  CHAR ? (NOTE 2BH IS '*'+1)
005A FA 5600            JM    LOOPI     YES(OR IGNOREABLE CONT CHAR)
                *
                *   PROCESS Y OR N FLAG TESTS
005D FE 59              CPI   59H       Y  ?
005F CA 6700            JZ    TFLG      YES
0062 FE 4E              CPI   4EH       N  ?
0064 C2 7600            JNZ   XA        BRANCH IF NOT A FLAG TEST
                *
0067 23         TFLG    INX   H         STEP LOC OVER Y OR N
0068 BA                 CMP   D         COMPARE TO CURRENT MATCH FLAG
0069 CA 5700            JZ    LOOP      ITS EQUAL SO EXECUTE THE STMT
                *
                *   ITS A FLAG FAILURE, SKIP OVER THE STMT
006C 23         SKIP    INX   H         STEP LOC PTR
006D 7E                 MOV   A,M       NEXT CHAR IN PGM
006E FE 0D              CPI   0DH       TO END OF STMT ?
0070 C2 6C00            JNZ   SKIP      NOT YET, SO LOOP
0073 C3 5600            JMP   LOOPI     AT NEXT STMT, GO DO IT
                *
                *   PROCESS ACCEPT STATEMENT
0076 FE 41      XA      CPI   41H       A  ?
0078 C2 8E00            JNZ   XM        NO
```

```
S1130000FE0100RD45915C27F7912427459R1092660
S1130010030920EFR12F2607BD00D5RD2120E4911F
S1130020252612860DRD27960DC640A10027066552
S1130030000R5A26F6A7009R910D27DF20C5C40950
S1130040F8D0B5A2AFAR60A2004BDE1AC395DE172
S113005D139FE0100090RA6009R2A2FF9R159270R
S1130060048140E260F08B1010527EC08A600R10D75
S1130070260F920E2814126110FF0102RDCD97010444A
S1130080860DRDC7RDB720CDR14D26120R08A69E
S1130090000C659B10104270ZC64EF7010520B781F5
S11300A04A2617E602C40F2605FE010220A908A667
S11300B000812A26F95A26F6209CR153260A0R0R2C
S11300C0A600087E0000208F81542602OR089D05B2
S11300D08D003E2092C640A6005A270ABD004EA697
S10A00E00008810D26F1392F
```

Listing 3a: MIKBUG format for the 6800 version of WADUZITDO.

```
00002A0001CD4600FE5CCA0000FE24CA5200
0010FE5FC219002BC30300FE2FC22400CDDF
002000C30300FE25C23C00060D7RCD4D000E
003040BECA3C00360023DC231007723FE0D
0040CCF000C30300CD1FC0CA460047CD19C0
00507BC92A0001 2B237EFE2BFA5600FE59CA
00606700FE4EC2760023BACA5700237EFE0D
0070C26C00C35600FE41C28E00220201CD46
00R0005F23060DCD4D00CDF000C35600FE4D
0090C2A100232376E1659BBCA9E00164EC356
00A000FE4AC2C30023237EE60F47C2B5002A
00B00201C35700237EFE2AC2B5005C2B500
00C0C35600FE53C2D20023237E23C30000C3
00D05700FE54C2D9002323CDDF00C357000E
00E040460DCAF000CD4D007E23FE0DC2E100
00F00E000600CD4D000DF2F200060AC34D00
0100060100000000
```

PAPERBYTE[tm] Bar Codes for WADUZITDO

In figure 1 and figure 2, we provide a PAPERBYTE[tm] bar code representation for the WADUZITDO programs of listing 3 and listing 4. These bar code representations were created in the absolute loader format documented in detail in the PAPERBYTE book, Bar Code Loader, *written by Ken Budnick of Micro-Scan Associates, and available for $2 at local computer stores or by mail (add $.75 postage and handling) from BITS Inc, 25 Route 101 W, Peterborough NH 03458.*

Text continued from page 168

the index register (6800) or HL register pair (8080) contains the location of the next program statement and should be saved and restored before returning from the subroutine. In the 8080 version the DE register pair should also be saved. Register A will contain the one character parameter, x, of the S:x. Its use is totally up to the subroutine.

The system has been organized so that the six bytes of changeable data are isolated from the read only portion. This means the rest of the 256 byte system could be placed in read only memory. It would fit in a single 1702A EROM chip.

It is easy to see how this language could be used to write a question and answer conversation using multiple choice or true, false answers. It may not be so obvious that more complex logic is possible. The example in listing 2 is a computer versus user NIM game which demonstrates a way this can be done.

Although WADUZITDO is not the ultimate answer to personal computing, it is something that almost anyone can have some fun with, and it definitely squeezes the most out of 256 bytes of memory.

A Pascal WADUZITDO

Notes by Ray Cote
Program by Larry Kheriaty

Along with the assembly language versions of WADUZITDO, Larry Kheriaty sent us the Pascal version shown in listing 5. The program is basically self-documenting and very easy to translate into assembly level programs for any particular processor. The program is indented to show logical relationships between related areas of text. This is sometimes known as prettyprinting.

The first four lines of the program are definition lines for the main program. In Pascal, all variables must be defined completely at the start of the section in which they are used. "Completely" means name and data type. This is a great help since all variables must be explicitly defined. You can easily check to see what type of variable is being used.

WADUZITDO uses two types of var-

Listing 4, continued:

```
007B 22 0201          SHLD  LST       YES, SAVE LOC OF LAST ACCEPT
007E CD 4600          CALL  JIN       ACCEPT ONE CHAR FROM KYBD
0081 5F               MOV   E,A       SAVE IT
0082 23               INX   H         MOVE OVER A
0083 06 0D            MVI   B,0DH     CR
0085 CD 4D00          CALL  JOUT      PRINT IT
0088 CD F000          CALL  PLF       PRINT LINE FEED
008B C3 5600          JMP   LOOPI     STEP OVER : AND GO ON
                      *
                      *   PROCESS MATCH STMT
008E FE 4D    XM      CPI   4DH       M  ?
0090 C2 A100          JNZ   XJ        NO
0093 23               INX   H         STEP OVER M
0094 23               INX   H         STEP OVER :
0095 7E               MOV   A,M       GET MATCH CHAR
0096 16 59            MVI   D,59H     ASSUME Y
0098 BB               CMP   E         COMP MATCH CHAR TO INPUT CHAR
0099 CA 9E00          JZ    MX        BRANCH IF IT MATCHES,FLG=Y
009C 16 4E            MVI   D,4EH     RESULT IS N
009E C3 5600 MX       JMP   LOOPI     SET MATCH FLAG TO Y OR N
                      *
                      *   PROCESS JUMP STATEMENT
00A1 FE 4A    XJ      CPI   4AH       J  ?
00A3 C2 C300          JNZ   XS        NO
00A6 23               INX   H         STEP OVER J
00A7 23               INX   H         STEP OVER :
00A8 7E               MOV   A,M       DESTINATION
00A9 E6 0F            ANI   0FH       CLEAR ZONE
00AB 47               MOV   B,A       NUMBER OF *S TO SKIP
00AC C2 B500          JNZ   JF        ITS A JUMP FORWARD
00AF 2A 0201          LHLD  LST       ZERO.. JUMP BACK TO LAST ACCEPT
00B2 C3 5700          JMP   LOOP      CONTINUE FROM THERE
                      *
                      *   SKIP FORWARD UNTIL PASS N *-MARKERS ( N IS IN BREG )
00B5 23       JF      INX   H         STEP PGM LOC
00B6 7E               MOV   A,M       NEXT CHAR
00B7 FE 2A            CPI   2AH       *-MARKER ?
00B9 C2 B500          JNZ   JF        NO, KEEP LOOPING
00BC 05               DCR   B         FOUND ONE, COUNT IT
00BD C2 B500          JNZ   JF        LOOP IF NEED TO FIND MORE
00C0 C3 5600          JMP   LOOPI     DESTINATION FOUND, GO EXECUTE
                      *
                      *   PROCESS STOP OR SUBROUTINE STATEMENT
00C3 FE 53    XS      CPI   53H       S  ?
00C5 C2 D200          JNZ   XT        NO
00C8 23               INX   H         STEP OVER S
00C9 23               INX   H         STEP OVER :
00CA 7E               MOV   A,M       PARAMETER TO REG A
00CB 23               INX   H         STEP OVER PARAMETER
                      *   NEXT STMT MAY BE MADE TO BE A CALL TO USER SUBR
00CC C3 0000          JMP   SUB       GO TO USER SUBR (OR TO EDITOR)
00CF C3 5700          JMP   LOOP      GO ON UPON RETURN FROM USER SUBR
                      *
                      *   PROCESS TYPE STATMENT AND SYNTAX ERRORS
00D2 FE 54    XT      CPI   54H       T  ?
00D4 C2 D900          JNZ   TE        NO, ITS AN ERROR
00D7 23               INX   H         YES, STEP OVER T
00D8 23               INX   H         STEP OVER :
00D9 CD DF00 TE       CALL  PRT       PRINT UP TO CR
00DC C3 5700          JMP   LOOP      DONE WITH T
                      *
                      *   SUBR TO PRINT UP TO NEXT CR
00DF 0E 40    PRT     MVI   C,40H     COUNT OF 64
00E1 46       PRTA    MOV   B,M       NEXT CHAR
00E2 0D               DCR   C         DECREMENT SAFETY COUNTER
00E3 CA F000          JZ    PLF       EXIT IF OVER 64 BEFORE CR
00E6 CD 4D00          CALL  JOUT      PRINT IT
00E9 7E               MOV   A,M       RELOAD CHAR TO ACCA
00EA 23               INX   H         STEP LOC PTR
00EB FE 0D            CPI   0DH       CR ?
00ED C2 E100          JNZ   PRTA      NOT CR, LOOP
                      *
                      *   SUBROUTINE TO PRINT LINE FEED AND PAD
00F0 0E 00    PLF     MVI   C,00H     NUMBER OF NULLS TO PRINT
00F2 06 00    PLFL    MVI   B,00H     NULL
00F4 CD 4D00          CALL  JOUT      WRITE A NULL
00F7 0D               DCR   C         DECREMENT COUNTER
00F8 F2 F200          JP    PLFL      LOOP TILL ENOUGH NULLS
00FB 06 0A            MVI   B,0AH     LINE FEED
00FD C3 4D00          JMP   JOUT      PRINT THEN RETURN
                      *
                      *   ABOVE IS END OF READ ONLY PORTION OF THE PROGRAM
                      *
                      *   THE FOLLOWING IS CHANGEABLE DATA
                      *
                            ORG   0100H     MOVE TO START OF DATA AREA
0100 0601     LOC     DW    0106H     ADDR OF SOURCE PROGRAM AREA
0102 0000     LST     DW    0000H     PLACE TO SAVE LOC OF LAST A:
                      *   THE NEXT TWO BYTES ARE ONLY FOR 6800 COMPATIBILITY
0104 00       CHR     DB    00H       UNUSED, LAST INPUT CHAR IN EREG
0105 00       FLG     DB    00H       UNUSED,MATCH-FLAG IN DREG
                      *
                            END
```

iables: integer and character. There is also a definition for constants (CONST). CONST informs the compiler that the value being assigned to this variable will not change. Integer variables will only take on whole number values.

The type character (CHAR) means that the variables will take on the values of ASCII characters, including all letters, numbers and special symbols.

The last line of the definition section defines a variable PROG as an array of characters. This definition also states that the relative base address of the array will be unity and the variable PZ will be used to specify locations within the array.

After defining our variables we are ready to start the first executable part of the program. In Pascal, the logical parts of the program are broken into procedures, equivalent to subroutines in languages such as FORTRAN. Every procedure is blocked off by BEGIN and END statements. The name of the first procedure is CHIN. After we have determined the name, we are told to begin executing procedure ACCEPT (which will return to us input values in variable CBUF). This is a subroutine which is not shown since it is specific to the processor being used. The next two procedures are also calls to subroutines used to DISPLAY the contents of the buffer and move the output to a new line. These two procedures are also machine dependent. Notice that Pascal allows you to use descriptive names. This is very important when writing a program that you want other people to read or that you want to understand at a later date.

The next procedure, LIST, first defines its own local variables, which it will use only within the LIST routine. As before, the procedure is delimited by BEGIN and END statements. This procedure introduces us to the concept of loops. Here we have a related pair of commands: REPEAT and UNTIL. These two commands cause the one line of three instructions and the call to procedure CHOUT to execute until either the value I is greater than 64 or the variable CBUF is equal to CEOL. Once either of these two conditions occurs, the program logic proceeds to call procedure NEWLINE. At this point the LIST procedure ends and returns to whatever procedure called it.

Procedure EXECUTE looks structurally the same as procedure LIST. There is a definition of variables, the BEGIN and END delimiters, and a REPEAT-UNTIL structure. This time the REPEAT-UNTIL statement is not waiting for a relation to be true, but is rather checking against one variable. Looking at how DONE was defined at the beginning of the procedure, we see that its designation is BOOLEAN. This means that the variable is being used as a logical variable and can take on the value true or false. The REPEAT-UNTIL instruction waits to see if the variable DONE is true. If so, we have finished this procedure and can stop it.

Listing 5: Pascal listing of WADUZITDO. See notes by Ray Cote.

```
PASCAL VERSION OF WADUZITDO, LARRY KHERIATY
PROGRAM WADUZITDO;
  CONST PZ=5000; BS=127; EOL=10;
  VAR LOC,LST,I : INTEGER; LCHR,FLG,CBUF,CBS,CEOL : CHAR;
    PROG : ARRAY[1..PZ] OF CHAR;

  PROCEDURE CHIN; BEGIN ACCEPT (CBUF); END;
  PROCEDURE CHOUT; BEGIN DISPLAY (CBUF); END;
  PROCEDURE NEWLINE; BEGIN DISPLAY (NL) ; END;

  PROCEDURE LIST; VAR I:INTEGER;
    BEGIN I:= 0;
      REPEAT
        CBUF := PROG [LOC]; LOC := LOC+1; I:=I+1;
        CHOUT
      UNTIL (I=64) OR (CBUF=CEOL); NEWLINE
    END;

  PROCEDURE EXECUTE; VAR DONE : BOOLEAN;
    BEGIN LOC := 1; DONE := FALSE;
      REPEAT
        CBUF := PROG[LOC] ; IF CBUF < '*' THEN CBUF := '*';
        IF NOT(CBUF IN ['*','Y','N','A','M','J','T','S'])THEN LIST ELSE
          CASE CBUF OF
            '*': LOC := LOC+1;
            'Y','N' : IF CBUF=FLG THEN LOC := LOC+1
                      ELSE REPEAT CBUF := PROG[LOC]; LOC:=LOC+1
                           UNTIL CBUF=CEOL;
            'A' : BEGIN LST := LOC; CHIN; LCHR :=CBUF;
                  NEWLINE; LOC :=LOC+2 END;
            'M' : BEGIN IF LCHR=PROG[LOC+2] THEN FLG :='Y'
                        ELSE FLG := 'N';
                  LOC := LOC+3 END;
            'J' : IF PROG[LOC+2] = '0' THEN LOC :=LST
                  ELSE BEGIN I:= ORD(PROG[LOC+2])-48;
                         REPEAT LOC:=LOC+1;
                           IF PROG[LOC] = '*' THEN I := I-1;
                         UNTIL I=0 END;
            'T' : BEGIN LOC := LOC+2; LIST END;
            'S' : BEGIN DONE := TRUE; LOC := 1 END
          END
      UNTIL DONE
  END;

  BEGIN CBS := CHR(BS); CEOL := CHR(EOL); CBUF :='\';
    WHILE TRUE DO BEGIN
      IF CBUF ='\' THEN LOC :=1
      ELSE IF CBUF=CBS THEN LOC := LOC-1
      ELSE IF CBUF='/' THEN LIST
      ELSE IF CBUF='$' THEN EXECUTE
      ELSE IF CBUF='%' THEN
        BEGIN I:=0;
          WHILE (I<64) AND (PROG[LOC] <> CEOL) DO
            BEGIN PROG[LOC] := CHR(0); LOC := LOC+1 END;
          PROG[LOC] := CEOL; LOC := LOC +1; NEWLINE
        END
      ELSE BEGIN PROG[LOC] := CBUF; LOC := LOC+1;
        IF CBUF=CEOL THEN NEWLINE END;
      CHIN
   END
  END.
```

Procedure EXECUTE also contains an IF-THEN-ELSE statement. If the value of CBUF is not contained within the brackets, perform procedure LIST. If the value of CBUF is somewhere within the square brackets, we want to perform an operation related to that value. We now come to another Pascal instruction, the CASE statement.

We are given a set of cases to choose from. The CASE statement tells us that we will be using the value in variable CBUF to determine what is to be done. We scan down each of the cases and find the one labeled with the value in CBUF. Since CBUF is type character we are looking at ASCII characters. Once we find the value of CBUF we execute the statements associated with it that are blocked off by another set of BEGIN and END statements. After we have finished, we move to the end of the CASE statement and then the last line of REPEAT-UNTIL statement.

The next section of the program does not look like the preceding sections. It does not start with a PROCEDURE statement, but has a BEGIN statement. So far we have discussed procedures. Any of the procedures that needed to use variables have defined their own. So why did we define those variables at the very beginning of the program? The reason is not to use them in a procedure, but to use them in the main program. This BEGIN statement is nothing more than the start of the mainline logic for program WADUZITDO. The mainline logic inputs characters and either stores them in an array as program or executes them as commands. This routine will not jump out of the loop and will have to be interrupted to stop. Of course it is possible to create another command that will allow you to exit from this cycle.

Now that we have looked at the Pascal version of WADUZITDO, the reader should refer back to either of the assembly versions. The Pascal version performs the same function as the assembly versions.

*The assembly language versions need to be heavily commented for the reader to understand what is happening. Even liberal comments will not help when converting from one assembly language to another. The Pascal version can be easily converted into any machine language. It is also self-documenting. Notice that even without a single comment, the Pascal listing is extremely easy to decipher. . . .*RGAC■