WATCOM
Waterloo Computing Systems Limited

158 University Ave. W.,
Waterloo, Ontario
Canada N2L 3E9

(519) 884-2700

infoWAT

Volume 1, Issue 4

December 1982

6502 Assembly-Language System

A software package has been written for developing assembly-language programs to execute on the 6502 microprocessor. In 6809 mode on the SuperPET, 6502 machine-language programs are prepared using the microEDITOR, a 6502 Assembler and a Linker. The software has functional characteristics similar to the 6809 Development System included with the SuperPET. For example, the 6502 assembler supports the same macro features described in the "Macros" article of this issue.

Features of the package facilitate Modular Design and Structured Programming techniques employed by professional software developers. The following are highlights: software can be developed for use with other Commodore models such as the 8032, 4032, VIC 20 or Commodore 64; a utility converts Waterloo Linker-format files to Commodore LOAD-format files; a loader can read Waterloo Linker-format files into the SuperPET direct and bank-switched memory for execution; and a subroutine facilitates execution of programs in bank-switched memory of the SuperPET.

VAX Software

The Waterloo portable full-screen EDITOR and interpreters for APL, BASIC, COBOL, FORTRAN and Pascal have now been implemented for the Digital Equipment VAX with the VMS operating system. More details will follow in the next issue of infoWAT.

SuperPET microPIP

A PIP (peripheral-interchange program) utility has been implemented for use with Waterloo software on the Commodore SuperPET. Waterloo microPIP provides convenient utilities for common operations involving SuperPET devices such as "host", "disk" and "serial". Diskette handling is facilitated by commands such as "backup" and "format". File-transfer commands support copying between local peripherals as well as host peripherals. Executable modules can be copied to a host for subsequent downloading with HOSTCM. Wildcard file specifications support powerful filename matching. Indirect command-file support allows batching of PIP commands.

Availability

The new products described on this page are available from:

WATSOFT Products Inc.
158 University Avenue West
Waterloo, Ontario
Canada N2L 3E9
telephone (519) 886-3700
telex 06-955458

Macros

Many users of the SuperPET 6809 Assembler Development System have expressed interest in obtaining more information on the use of macros. The following is intended to be a summary of the features of the assembler's macro processing capabilities.

A macro must be defined before its use. The general form of a macro definition is:

```
name    MACR
        <macro body>
        ENDM
```

The name of the macro is "name". This name may not be an assembler keyword (e.g., IFEQ).

Macro arguments (parameters) are referenced symbolically by using a "backslash" character ("\") appended with a number (0,1,2,...). The first argument is "\0", the second is "\1", and so on.

Macro arguments may be tested at assembly time using the "conditional assembly" directives.

```
  Example       Test
ifc \0       ; null argument
ifc \0,xyz   ; argument = "xyz"
ifnc \0      ; non-null argument
ifnc \0,xyz  ; argument <> "xyz"
```

The conditional assembly directives that test numeric expressions may be used when a macro argument is an integer numeric string.

```
  Example       Test
ifeq \2-1   ; third argument = 1
ifge \0     ; first argument >= 0
ifgt \1+1   ; second argument >= 0
ifle \0-12  ; first argument <= 12
iflt \3+1   ; third argument < -1
ifne \4     ; fourth argument <> 0
```

Macro arguments may be absent (null). For example,

```
double ,ans,ansd
double x,,ansd
```

demonstrate missing arguments which may be tested using the "ifc" and "ifnc" conditional assembly directives.

Macros may invoke other macros including themselves (macro recursion). For example,

```
double macr
     ifc        \0,x
        regdouble    \1,\2,\4
     endc
     .
     .
     .
     endm
```

illustrates a macro invoking another macro when the first argument is identical to the string "x". The invoked macro is passed the second, third and fifth arguments.

Within a macro body, arguments can be concatenated with other strings to produce new strings.

```
label\0
lab\1el
       st\1    \1\0
```

If "\0" is "a" and "\1" is "b" then the result is

```
labela
labbel
       stb     ba
```

Unique labels can be generated. The form for such a label is <string>\.<number>. Here <string> is any valid assembler label and <number> is any digit from 0 to 9. For example,

```
actst  macr
       cmpd    #0
       beq     zero\.0
       std     \0
       jmp     next\.1chk
zero\.0 addd   #1
next\.1chk equ *
       endm
```

shows the use of assembler-generated labels. Initially "\.0" has the string value "1" and "\.1"

has the string value "2". If the macro "actst" is invoked with argument "arg", the result is

```
        actst   arg
+       cmpd    #0
+       beq     zero1
+       std     arg
+       jmp     next2chk
+zero1  addd    #1
+next2chk equ   *
```

The next time the macro body is expanded, "\.0" will have the string value "3" and "\.1" will have the string value "4". In this way unique labels are guaranteed.

The "fail" directive may be used to produce error messages at assembly time. Using the conditional assembly directives you may detect invalid macro arguments. The "fail" directive will cause a message to be printed in the listing file and on the terminal.

The ";include" directive is very useful for macro libraries. You may place a number of macro definitions in a separate file and then include them in the assembly of a module by using the ";include" directive.

Data Transfer Using the Serial Port

Many users of the Waterloo software on microcomputers are interested in using the serial port to transmit data to and from other microcomputers or mainframe computers. However they are not quite sure how to go about it. In this article we will describe how to send and receive files of textual data between two computers that support the Waterloo micro languages. We will use Waterloo microBASIC as the programming language.

The various implementations of our software treat the serial port as a file with the special name "serial". Some knowledge of the interaction of BASIC's input/output routines with the serial port may aid in understanding how a program should deal with this device. Two example programs show a simple means of text file transfer. The first program is used to "send" files, using the serial port, to another computer. The second program is used to "receive" a file that is being "sent" with the first program by another computer via the serial port.

```
 1 ! File Transmission
 2 on ioerr ignore
 3 on eof ignore
 4 open #3,"serial",inout
 5 linput "File to send? ",file$
 6 open #4,file$,input
 7 if io_status <> 0
 8   print "No such file"
 9 else
10   linput #3,rply$
11   loop
12     linput #4,rec$
13     if io_status <> 0 then quit
14     print #3,rec$
15     linput #3,rply$
16     if rply$(1:1)=chr$(10) then
rply$(1:1)=""
17     if rply$ <> "ok" then quit
18   endloop
19   close #4
20 endif
21 print #3,"<eof>"
22 close #3
23 end
```

```
 1 ! File Reception
 2 on ioerr ignore
 3 open #3,"serial",inout
 4 linput "File to get? ",file$
 5 open #4,file$,output
 6 if io_status <> 0
 7   print "Cannot open file"
 8 else
 9   print #3,"go"
10   loop
11     linput #3,rec$
12     if rec$(1:1)=chr$(10) then
rec$(1:1)=""
13     if rec$ = "<eof>" then quit
14     print #4,rec$
15     if io_status <> 0 then quit
16     print #3,"ok"
17   endloop
18   close #4
19 endif
20 close #3
21 end
```

The serial port is capable of both transmitting and receiving data. If we wish to both send and receive records via this port then, in BASIC, we open the "serial" device for input/output (e.g., OPEN #3,"serial",INOUT).

When you "print" a string to the device (e.g., PRINT #3,"go"), the characters 'g', 'o', a carriage return (chr$(13)), and a line feed (chr$(10)) are transmitted over the serial port. If the port is connected to the serial port of another computer then each of these characters will appear as input data on the port of this computer. A program must be executing on the receiving computer which "grabs" each character as it appears. If there is no such program then the transmitted data will be lost. The rates of transmission and reception must be identical, otherwise the transmitted data will not be "assembled" correctly by the receiver. Therefore you must match the BAUD rates of the two serial ports.

If characters arrive at a rate faster than the receiving program can deal with them, then some of the transmitted data will be lost. This is a problem that will likely show up at high BAUD rates. It is a problem that cannot be easily dealt with from a BASIC program since it typically executes much slower than is necessary to ensure no loss of data.

In addition, even if data arrives at a slow enough rate, there may be points at which we wish to do other processing. For example, we may wish to write a number of characters, that have been received, to a file. It would be convenient if we could somehow inform the program executing on the "sending" computer that it should wait a while until we are ready for more data. This problem is fairly easy to solve in the program. We can invent a simple protocol which will enable the two programs executing on different computers to "talk" to each other. One program will "drive" the conversation. In other words, one program will instruct the other program when to "speak" (i.e., send a record) and when to "be silent" (i.e., wait until it is alright to send the next record).

This is very simple to do using BASIC input/output statements. If you examine the "receive" program above, it may be apparent to you that it is the program that drives the conversation. The "send" program will not "speak" until it is "spoken" to. Since the two programs must synchronize their dialogue it is very important to know which of the above programs must be "run" first. In this case it is the "send" program which must be run first. If it is not executing at the time the "receive" program says "go" then it will not "hear" the command to begin transmission.

If the programs are started in the wrong order, both will be "listening" and neither will "say" anything. Clearly, it is desirable to avoid such a "standoff" or "deadlock" situation. If this event occurs, both computers may no longer respond. If you type in the above programs, save them first before you attempt to run them.

The two programs, shown above, were written to use the serial port of a microcomputer. On large computers this port may also be the one to which your terminal/microcomputer is connected. If this is the case, the programs may require some slight modification. The device name and dialogue synchronization may need to be changed.