bcc	title MICROPROCESSOR DEBUGGER FOR THE AMC		prefix/class-number.revision DBAMC/W-43
checked [illegible]	authors R. R. Van Tuyl [illegible]	approval date 1/6/70	revision date
checked [illegible]		classification Working Paper	
approved [illegible]		distribution R+O Company Private	pages 6

ABSTRACT and CONTENTS

A debugger has been prepared to debug the APU code of the AMC. This debugger can be converted in a straight-forward manner to debug the APU running on the hardware. The user interface is described briefly.

Microprocessor Debugger for the AMC

A debugger has been written which converts the microprocessor simulator into a debugger for the APU code. The primary changes in the microprocessor simulator involve using the NRH for the simulated memory and patching in two instructions in the microcode to effect communication between the simulator and debugger.

The philosophy of the design is that the debugger should correspond to debugging with the automatic breakpoint hardware. That is when a zero appears in the APU code the microprocessor is forced to the instruction in Ø by simulating an automatic breakpoint at the half instruction. The state is saved, then control is given to the debugger. Thus, only memory contains pertinent information and the debugger need only allow access to it.

The debugger allows the user many operations to aid in looking at memory and retaining control over the program. The interface is through one dispatcher which uses Larry Barnes' "forgiving command recognizer". The syntax is SPL like, in that the names and formats are similar. The command line is:

```
        command = function [' ']  argument's  CRLF
```

The functions have both a long and short symbol. The short symbol is a nonalphanumeric character and does not require a space following it. A long symbol is recognized when at least enough characters are input to distinguish between similar names. Arguments are generally input as integers:

```
  integer = '+'/'-'  digit    {digit} [(B|D)[digit]]
```

Integers without the letter suffix are input as octal integers.

Debugger Functions

Examine /

Examine takes Ø, 1, 2 arguments. If no argument, then the cell at the address given by the Last Value Typed (in the local address space if the Last Register Opened is in the local address space) is typed. The Last Register Opened is unaffected.

If one argument, then the cell addressed is typed. This cell becomes the Last Register Opened.

If two arguments, then the cells between the first and second argument are typed. The Last Register Opened is the last cell typed.

Next >

Next takes one optional argument. If no argument, the cell following the Last Register Opened is typed. It becomes the Last Register Opened.

If one argument, then that number of cells following the Last Register Opened is typed. The last cell typed becomes the Last Register Opened.

Previous <

Previous takes one optional argument. It is similar to Next.

Follow ↑

Follow takes one optional argument. If no argument, then the cell at the address given by the Last Value Typed (in the local address space if the Last Register Opened is in the local address space) is typed. This register becomes the Last Register Opened.

If one argument, then the cell at the address given by the Last Value Typed is typed. Then the next or previous N cells are typed as given by the argument.

GOTO $

GOTO takes one argument, the address in the microprocessor which control is given to after the state is loaded. The main loop is at 1ØØØB.

Continue ,

Continue takes no arguments. It takes one of two actions depending on how the debugger was entered. If the debugger was entered from a breakpoint, then the state is loaded and a control sent to ICONT in the APU. If the debugger was entered to get a flag set, then a return to the routine calling the debugger is done.

Break

Takes Ø, 1, or 2 arguments. If no argument given, a breakpoint is set on the Last Register Opened.

If one argument, a breakpoint is set on the cell addressed.

If two arguments, breakpoints are set on all cells between the two addresses. If the number of breakpoints exceed the number remaining, the function will fail. However, all the breakpoints set before failure will remain set.

KILL [

Takes Ø, 1, or 2 arguments. Similar to Break, except breakpoints are cleared from cells specified. Prints the number of breakpoints cleared.

DISPLAY]

Takes one argument, some unique part of the word FLAGS, or BREAKS.

> BREAKS
>
> If the argument is breaks, then all the breakpoints are typed.
>
> FLAGS
>
> Takes Ø or 1 argument, If no argument, then all the flags are typed.

If one argument, then the flag requested is typed.

Currently flags are:

REQ1LT

REQ2LT

ATTNFG

SMASK

PATCH-SPACE

LAST-REG-OPENED

SET

Takes 2 arguments, the name of the flag, followed by the value.

LOAD

Takes 1 argument, the name of a random structure file. The simulated memory is loaded from the file as well as break-points and flags.

DUMP

Takes 1 argument, the name of a file. The simulated memory, breakpoints, and flags are placed in the file.

Find)

Takes 1, 2, 3 or 4 arguments.

If one argument present, it attempts to print all cells which satisfy:

cell AND SMASK = arg AND SMASK

between the current limits.

If first argument is NOT (or subsequence) the search is modified to satisfy:

cell AND SMASK $\neq$ arg AND SMASK

If there are one or two additional arguments after the value, these are used to determine the limits of the search.

DDT, ODDT

Takes no argument, merely fires up a fork with either DDT or ODDT and transfers control to the fork. %F in DDT or ODDT will return control to the dispatcher. This aids in locating symbolic information.

CDDT

Takes no argument, does a continue like the 940 executive. Control goes to DDT or ODDT whichever was entered last.