bcc	title	prefix/class-number.revision
	CPU TEST PROGRAMS	TSTX/W-45

checked	authors	approval date	revision date
Buffy W Kemp	Charles Simonyi	5/29/70	
checked	Charles Simonyi	classification: Working Paper	
approved- Mel		distribution: Company Private	pages: 8

ABSTRACT and CONTENTS

A comprehensive set of CPU Test Programs is described. Instructions to run the programs on the Model 1 are given.

1. Purpose.

The purpose of the CPU Test Programs is to verify that the CPU executes all instructions according to specifications (MICPU/M-4.4). This implies that the correct response of all functional hardware is verified. However, if an error occurs, it will be identified as an incorrectly working CPU instruction rather than a faulty hardware component. To find the latter, conventional debugging techniques will have to be used.

2. Method.

A microprocessor (other than the CPU being tested) is used to run a CTP program called SYSDDT which contains a CPU simulator. The test programs run on both the simulator and the CPU in compare mode; that is, the results of the simulation and CPU are compared for every instruction. This is realized as follows: both the simulator and the CPU reference the central memory, but they have two different sets of central registers. The simulator may change its own set of central registers, but it never stores into the central memory. Rather, it assembles a list of changes which should be made. When the simulation of an instruction is complete, the CPU is requested to execute the same instruction using the CPU's single step feature. The CPU, of course, may change central memory as well as the central registers. After the instruction has been executed, the CPU stores the central registers in the memory and returns the control to SYSDDT. The two sets of central registers are then compared, and the list of changes is verified against the actual contents of memory. Any disagreement will cause an error message to be given. A shortcoming of this method

is that if a malfunctioning CPU changed a core location in addition to all changes it was supposed to make, the problem would go undetected. SYSDDT also implements a powerful control language which enables the operator to print out and change core locations and central registers in different formats and to set breakpoints. These features are described elsewhere.

3. The Set of Test Programs.

Because of the complexity of the CPU, the test programs are arranged in 4 packages:

- TST1 Test of Addressing Modes
- TST2 Test of Instructions and Ring Dependent Traps
- TST3 Test of BLL
- TST4 Test of Fixed Traps

All 4 test packages are loaded the same way as described on page 6. TST4, as an exception, does not use compare mode because it partly depends on the timer.

3.1 TST1.

The program is organized as a sequence of macro calls. A macro call $Z(P,...P_n)$ expands into a piece of code which first sets up the necessary environment and then executes an EAX or LDA instruction which will use adressing mode Z with the relevant parameters $P,...P_n$. Each macro is called several times so that each parameter can assume some special (all ones, all zeros) and random values.

For example MI(1) is used to test the I type addressing mode in instruction, with the W field equal to 1. The macro expands into a few instructions which first store into G'[1] a reasonable IAW (to

avoid traps), then execute an EAX I:1. The S1(1, 2, 73958) is a particular call of the macro to test string-type IAW-s. The macro expands into code to store a pattern into location 73958 and then execute LDA $R'[2] addressing an IAW, TYPE = 2, CSIZ = 1, CPOS = 1, WA = 73958. Note that both tests contain some implicit parameters ('reasonable IAW', 'a pattern'). These parameters are held constant, since varying them would only duplicate tests made elsewhere.

3.2 TST2

This test program is divided into 7 phases:

3.2.1 Test of ETR, IOR, EOR, ADD, SUB, ADC, SUC, MUL, ICP, CPZ, CMZ, LDA, LDB, LDX, LNX, ADX, EAX, LAX, TSB, STA, STB, STX, ADM, XMA.

A nested loop executes each of the above instructions with about 70 different pairs of values for the A register and the operand. These operands are taken from a table which includes worst case (0, 4B7, -1 etc) and random numbers.

Some of the instructions do not use both the A register and the operand, but are included into this phase for brevity.

3.2.2 Test of ASHA, ASHD, LSHA, LSHD, CYA, CYD, CAB, XAB, CBA, CBX, XXB, CXB, CAX, XXA, CXA, CNX, ZOA, ZAB, ZOB, LLT, COB.

Each of the listed instructions is executed with about 70 different pairs of values in the A and B registers, taken from the operand table. Actually the list of instructions includes 10 different left and right shifts of each kind.

3.2.3 Test of STD, LDD, ISD, DSD, CLS, ASP.

These instructions will operate on 12 pairs of string descriptors. The pairs are selected to cause the different responses of ISD and DSD,

and have varying CSIZE fields to check the special multiply and divide actions in CLS and ASP.

3.2.4 Test of DIV.

The divide instruction is executed with about 100 triplets of numbers as A, B and the operand. The triplets are taken from the operand table.

3.2.5 Test of BRU, BEQ, BNE, BLT, BLE, BGE, BGT, BSX, BRX.

All six conditional branches are executed with the three different CC settings. Token BRU, BSX, and BRX branches are also executed.

3.2.6 Test of MVB, MVC

A block of storage is moved towards one lower and higher addresses. A token MVC is executed.

3.2.7 Test of ABE, IATRP, RO, and STKOV Traps.

After the necessary transfer vectors are set up, each of the above traps is caused by the appropriate 'erroneous' code.

After executing phase 7, TST2 returns to the first phase.

3.3 TST3

The BLL test program executes about 20 BLLN, BLL, MCAL and POP instructions, with different parameters, activating all features of BLL. In particular:

- BLLN is executed with STK and CPR 0 and 1
- Labels are passed and jumped to in fixed or stacked environments.
- Actual arguments are copied from central registers
- All structures are copied by address
- ROD address is copied

- G or L relative formal parameters are used
- All types are copied by value
- Label, Array and String are copied by value
- POPs 0 and 1 are executed
- Fortran skip - noskip feature tested
- MCALLN 1 executed in user and monitor rings
- MCALL2 passes a label from user ring
- Monitor returns to label provided by user.
- MCALL3 passes 4 different arrays and a string.

3.4 TST4

The test program of fixed traps does not depend on an external simulator, but it assumes that addressing and simple instructions work. If the program detects an error, it stops with an error code in the A register:

1. Incorrect trap parameter
2. Wrong trap number
3. Unexpected trap, or a trap did not occur when expected.

The following trap situations are tested:

- Memory access errors in the user and utility rings.
- Page read only errors, with read only bit set in the map or PMT.
- Page not in map trap.
- Page not in core trap, caused by the SF bit in PMT, U bits in CHT or missing CHT entry.
- Timer overflow

- About 15 cases of Trapped Instructions
- XMON and XUTIL traps caused by a BLL
- XMON trap caused by a LOADS
- 5 types of BLL traps

4. Loading and Running the Test Programs.

A test program can be loaded from magnetic tape, using IPL, together with SYSDDT, the control program.

Note that the 4 test programs use the same area in central memory (0-27777B), so only the program loaded last will be able to run. SYSDDT does not need to be reloaded since it resides in a micro-processor's private memory.

After loading, the following sequence of commands can start execution: (Characters typed by the operator are underlined. Comments are in square brackets).

#GOTO 15000.	[Transfer control IPL - SYSDDT]
DDT	[RESET the CPU]
%R	[Load SYSDDT's map]
0%X	[Set compare mode. 1%X for TST4]
4100010;U	[Send a STROBE to the CPU]
;P	[Proceed]

The state of the running program may be observed using the switch register and indicator lights on the control panel. SYSDDT continuously reads the contents of the switch register, interprets it as a virtual address, and displays the contents of the memory cell addressed on the indicator lights. When the CPU is in step mode, the central registers are stored after every instruction at the following addresses:

602764B	P
602765B	A
602766B	B
602767B	C
602770B	D
602771B	E
602772B	X
602773B	L
602774B	G
602775B	SR
602776B	CTC
602777B	IT

A standard practice is to select 602764B and to observe the advancing P counter.

The following messages indicate the failure of a system component:

A: SIM32 CPU0

SIM#CPU AT 611073: ADX R:10 A=32 B=1046 is given for example when the simulator and the CPU disagree about the content of the A register after the execution of the instruction at 611072. Type either:

;L-1/ [to find out what the last instruction was]

;P [to proceed]

CPU HUNG UP AT 611031: SLOK A=10

A serious failure prevented the CPU to execute the instruction at 611031. To start at location M, reset the CPU and type:

```
Z 6/ 0  100 ¢     [set abs. location 6]
M;L¢              [set ;L to M]
4100010;U         [send a STROBE.  Observe that the
$                  P counter changes, otherwise the CPU
                   hung up again]
;P                [Proceed]
```

If TST4 stops (that is the P counter does not change), type the following:

```
                  [Hit RUBOUT]
;L+1;L            [Bypass the stop instruction]
;A=  1            [error number]
602752/ 610742    [location where the erroneous trap
                   occured]
;P                [proceed]
```

To load the next test program, type:

```
                  [Hit RUBOUT]
0100310;U         [transfer control SYSDDT - IPL]
#INIT.            [etc.]
```