Support for debugging/breakpointing in transputers

INMOS Technical Note 61

INMOS Bristol

January 1989
72-TCH-061-00

Contents

1 Introduction

Some transputers have additional instructions to support debugging. The instructions allow breakpointing to be implemented, particularly for C and Fortran programs.

The instructions provided fall into two groups. The first group provides support for breakpointing (used in a debugging environment) in a way which is upwards compatible with the transputers without breakpoint support. The second group helps in the identification of transputers in an arbitrary mixed array.

2 Breakpoint instructions

There is a need for a debugger to allow the user to stop execution of his program at any point, in order to discover the values of variables and the state of the program at that time. This can be done by a post compilation process if any instruction produced by the compiler can be replaced with a breakpoint instruction. The breakpoint instruction must switch control to another context, and preserve all user values, in particular the processor stack contents. The user process under investigation is replaced by the breakpoint process. Once completed the user instruction must be substituted back again, the user's values in the processor stack must be replaced, and the context switched back to the user process.

The provision of such a breakpoint facility allows single-stepping of instructions by the repeated application of the breakpoint instruction to subsequent instructions.

The breakpoint instruction must be the same size as the shortest instruction length in the instruction set. In the case of the transputer, this is one byte, the length of the primary instructions. The op-code for the breakpoint is #00, which corresponds with the instruction jump 0 (j 0). Jump 0 is in some senses a no-op instruction, although it can be compiled by a compiler in order to allow a low priority process to be timesliced. As a no-op, it can also occur as the last entry in a jump table, for example. In order to ensure that programs which do not use the breakpointing mechanism are unaffected when jump 0 is executed, the effect of jump 0 to cause a breakpoint must be explicitly switched on by a debugger.

The context of a process in the transputer model involves a workspace pointer (Wptr) and an instruction pointer (Iptr). Wptr is a word address pointer which points to a workspace in memory. Iptr points to the next instruction to be executed, for the currently executed process. The context

switch performed by the breakpoint instruction exchanges the Wptr and Iptr of the currently executing process with the Wptr and Iptr held above MemStart. The address of MemStart can be found in the relevant datasheet. There are two contexts held above MemStart, one for high priority and one for low priority, to allow processes at both levels to have breakpoints simultaneously.

The address map above MemStart is as follows:

Purpose	**Word Offset from MemStart**
IPtr (Low Priority)	3
WPtr (Low Priority)	2
IPtr (High Priority)	1
WPtr (High Priority)	0

The breakpoint mechanism has been implemented by providing a user mode and a debugger mode. There is a user mode and a debugger mode at both levels of priority. In addition to the single byte jump 0 breakpoint instruction which can be enabled or disabled, there is a two byte op-code which forces the breakpoint context swap even if jump 0 breakpoint is disabled; this allows the debug process to disable the jump 0 breakpoint instruction, and still return to the user process.

In addition, to allow the process queues to be manipulated in the debug process, four instructions have been added to disable and enable the interrupts from the two prioritised timer queues. The timer continues to count time normally while the timer interrupts are disabled. However, any pending process waiting on the timer queue will not be placed on the back of the corresponding active process queue (the queue of the same priority as the pending process) until that timer queue's interrupt is re-enabled.

An instruction ldmemstartval (load MemStart value) which returns the value of the MemStart address in the Areg of the processor stack has been provided to ease the setting up of the debug process at initialisation time.

On transputers with debugging support instructions, the effect when Reset is taken low for a reset is to clear the ErrorFlag, HaltOnErrorFlag and EnableJ0BreakFlag. On transputers without debugger support, the ErrorFlag and HaltOnErrorFlag are not initialised on reset. In all transputers analyse leaves the processor flags unchanged.

3 Other instructions

The load device identity (lddevid) instruction pushes the device type identity into Areg.

Each product is allocated a unique group of numbers for use with the lddevid instruction, allowing several revisions of the same product to be differentiated. The identity value for a specific product can be found in the relevant datasheet.

The lddevid op-code is a no-op on the IMS T414, IMS T212, and IMS T222.

The lddevid op-code has the following effect on the IMS T800:

#17C	load device identity op-code on T800
Areg' = undefined Breg' = Creg	

After executing the sequence

```
ldc 2; ldc 1; ldc 0; lddevid; stl temp
```

Areg contains 1 for a T414 or T212, 2 for a T800, and 0 for any product with a device identifier (in which case temp contains the identifier).

The fptesterr op-code is a valid instruction on some processors, such as the IMS T425, which do not incorporate a FPU. Bootstrap loaders which incorporate this instruction will operate on all processors for which fptesterr is a valid op-code. The fptesterr instruction (set Areg of the processor stack to true if the FPU error flag is clear, set the Areg to false if the FPU error flag is set) returns true in Areg on processors without a FPU. The other op-codes used for FPU operations do not have any defined function on such processors and should not be executed.

The pop (pop processor stack) instruction is available on some transputers. This instruction discards the contents of Areg, placing the contents of Breg in Areg and the contents of Creg in Breg. The final value of the Creg is not defined.

4 Summary of new instructions

Code	Abbreviation	Cycles	Name
#0	j 0	3	jump 0 (break not enabled)
		11	jump 0 (break enabled, high priority)
		13	jump 0 (break enabled, low priority)
#B1	break	9	break (high priority)
		11	break (low priority)
#B2	clrj0break	1	clear jump 0 break enable flag
#B3	setj0break	1	set jump 0 break enable flag
#B4	testj0break	2	test if jump 0 break enable flag is set
#7A	timerdisableh	1	disable high priority timer interrupt
#7B	timerdisablel	1	disable low priority timer interrupt
#7C	timerenableh	6	enable high priority timer interrupt
#7D	timerenablel	6	enable low priority timer interrupt
#7E	ldmemstartval	1	load value of MemStart address
#79	pop	1	pop processor stack
#17C	lddevid	1	load device identity
#9C	fptesterr	1	load value true (as FPU not present)

5 Specification of new instructions

In the following specifications the notation used is that defined in Appendix F of "Transputer Instruction Set - a Compiler Writer's Guide".

j #0_ jump
Oreg' = 0 "break not enabled" ⇒ Iptr' = ByteIndex NextInst Oreg° "break enabled" ∧ Oreg° ≠ 0 ⇒ Iptr' = ByteIndex NextInst Oreg° "break enabled" ∧ Oreg° = 0 ⇒ Mem' = Mem ⊕ {Index MemStart Offset ↦ Wptr, Index MemStart (Offset+1) ↦ Iptr} Wptr' = Mem (Index MemStart Offset) Iptr' = Mem (Index MemStart (Offset+1)) where "at high priority" ⇒ Offset = 0 "at low priority" ⇒ Offset = 2 This instruction has the potential of causing a process to be timesliced

break #B1	break (swap process context)
Mem' = Mem $\oplus$ {Index MemStart Offset $\mapsto$ Wptr, Index MemStart (Offset+1) $\mapsto$ Iptr} Wptr' = Mem (Index MemStart Offset) Iptr' = Mem (Index MemStart (Offset+1)) where "at high priority" $\Rightarrow$ Offset = 0 "at low priority" $\Rightarrow$ Offset = 2	

clrj0break #82	clear jump 0 break enable flag
EnableJ0BreakFlag' = clear Iptr' = NextInst	

setj0break #B3	set jump 0 break enable flag
EnableJ0BreakFlag' = set Iptr' = NextInst	

testj0break #B4	test if jump 0 break enable flag
Creg' = Breg Breg' = Areg EnableJ0BreakFlag' = set $\Rightarrow$ Areg' = true EnableJ0BreakFlag' = clear $\Rightarrow$ Areg' = false Iptr' = NextInst	

timerdisableh #7A	disable high priority timer interrupt
"disable high priority timer interrupt" Iptr' = NextInst	

timerdisablel #7B	disable low priority timer interrupt
"disable low priority timer interrupt" Iptr' = NextInst	

timerenableh #7C	enable high priority timer interrupt
"enable high priority timer interrupt" Iptr' = NextInst	

timerenablel #7D	enable low priority timer interrupt
"enable low priority timer interrupt" Iptr' = NextInst	

ldmemstartval #7E	load value of MemStart address
Areg' = MemStart Breg' = Areg Creg' = Breg Iptr' = NextInst	

lddevid #17C	load device identity
Areg' = "Product identity value" Breg' = Areg Creg' = Breg Iptr' = NextInst	

pop #79	pop processor stack
Areg' = Breg Breg' = Creg Creg' = Areg Iptr' = NextInst	

fptesterr #9C	load TRUE (as FPU not present)
Areg' = true Breg' = Areg Creg' = Breg Iptr' = NextInst	