TMS320 DSP
DESIGNER'S NOTEBOOK

A Novel Way of Using TMS320C40 Cache

APPLICATION BRIEF: SPRA222

Keith Larson
Digital Signal Processing Products
Semiconductor Group

Texas Instruments
March 1993

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain application using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

TRADEMARKS

TI is a trademark of Texas Instruments Incorporated.

Other brands and names are the property of their respective owners.

CONTACT INFORMATION

US TMS320 HOTLINE	(281) 274-2320
US TMS320 FAX	(281) 274-2324
US TMS320 BBS	(281) 274-2323
US TMS320 email	dsph@ti.com

Contents

Examples

A Novel Way of Using TMS320C40 Cache

Abstract

This document discusses how to place any value into the TMS320C40 cache. A code example is included.

Design Problem

How can I place any value into the TMS320C40 cache?

Solution

A usual approach to loading the cache is to unfreeze the cache and let it always be filled, hoping for a looped block of code. By freezing the cache at the end of time-sensitive routines, a little more performance can be expected since the cache does not always have to be filled from external memory on the first pass through. However, the cache may not always fill completely due to code dependencies or conditional branching. In this case, it would be desirable to load the contents of any address into the cache.

The routine shown in Figure 1 will poke opcodes from an arbitrary address into the cache using a feature of the interrupt processor. In this case, when the RETI opcode is executed, writing PGIE to GIE, one opcode following the RETI is protected from interrupts and is always fetched (and executed). By properly controlling the value of TOS, it is possible to load any external address pointed to by TOS into the cache! In this case, an interrupt vector is used to loop the cache loader back to itself each time an opcode is loaded into the cache.

Caution: Since any opcode can be executed in any order, it is important to control the potential action of all opcodes fetched in this manner. For example, if an opcode is supposed to write data to a location pointed to by an auxiliary register, it would make sense to make sure that all the auxiliary registers point to a safe “dummy” location. Likewise, adequate controls should be placed on the loader to ensure that the correct status is always loaded back into the CPU after each cache load.

Also note that DATA values can be poked into the cache. Since all opcodes going into the cache are executed, unpredictable results may occur when loading such a value. If a DATA value is loaded into the cache, that value is NOT accessible as data from the cache since the DDATA bus cannot be connected to the cache for a transfer. IE-only program fetching is allowed from the cache.

NOTE:

The routine shown in Example 1 does not include a full save and restore, nor does it control the values of the data pointers (DP and ARs). It is the programmer’s responsibility to add the code necessary to provide the context save routines and other error checking.

Since the ’C40 cache is filled on 32 words boundary, sometimes the program address alignment is needed in order to put the maximum length of the program into the cache.

Example 1. Code Listing

```
;————————————————————————————————————————;
; void lcache(*ptr, len);                  ;
;                                          ;
; loads the program pointed to by ptr into ;
; the cache from external memory           ;
;————————————————————————————————————————;
        .global  start,test,FLAG_0,_lcache
        .global  dec,inc,more,RST,NMI,TINT_0
        .text
RST     .word    $     ;set up temporary IVTP in
   ; need to align at 512 word boundary
NMI     .word    $+1   ;external RAM
TINT_0  .word    _lcache
   ;————————————-;
start:  ldp      RST   ;set up a new vector table
        ldi      @RST,R0
        ldpe     R0,IVTP
        ldi      @stack,SP   ;set up a runtime stack
        ldi      @a_test,R0  ;subroutine to load is
                             ; “test”
        sti      R0,@APC
        ldi      16,R0       ;load 16 cache locations
        sti      R0,@CNT
        sti      IIF,@FLAG   ;keep original IIF
        and      0E3FFh,ST   ;clear, thaw and enable
                             ; cache
        or       5800h,ST
        call     $+1      ;a way to push PC on stack
        pop      R0    ;takes care of first dummy pop
        addi     4,R0
        push     R0
        call     _lcache     ;call the cache loader
        or       00C00h,ST   ;freeze and enable cache
        ldi      @FLAG,IIF   ;restore IIF
   ;————————————-;
test    ldi      15,R0    ;Test code to cram into the
                          ;cache
dec     subi     1,R0     ;with conditional branches
        bnn      dec
        ldi      -15,R0
inc     addi     1,R0
        bn       inc
        bud      test
        nop
        nop
        nop ;
   ;————————————-;
```

```
_lcache  ldp      APC
         ldi      @CNT,R1
         subi     1,R1
         bnz      more
         pop      R0       ;pop junk address
         rets              ;return to original caller
   ;_____________
more     sti      R1,@CNT
         pop      R1      ;pop junk address
         ldi      @APC,R1 ;create “new” return address
         addi     1,R1
         sti      R1,@APC
         push     R1
         ldi      @TINT0,IIF  ;turn on TINT0
         ldi      1,IIE       ;enable TINT0
         reti           ;after return, fetch 1 opcode
   ;_____________-;
         .global  FLAG,APC,CNT
         .global  a_test,stack
FLAG     .word    0        ;Original IIF register
APC      .word    0        ;Auxiliary Program Counter
CNT      .word    0        ;length to load
TINT0    .word    01000000h
a_test   .word    test-1
stack    .word    $        ;reserve stack locations
         .end
```