SIEMENS

Microcontrollers

ApNote AP1611

☐ additional file
APXXXX01.EXE available

Interrupted MUL / DIV

In order to provide fast interrupt response, Multiply/Divide instructions in the microcontrollers of the C16x family are interruptable.

C. Meinold / Siemens HL MCB PD

AP1611 ApNote - Revision History		
Actual Revision : Rel.01 Previous Revison: Rel. none		
Page of actual Rel.	Page of prev. Rel.	Subjects changes since last release)

1 Interrupted MUL / DIV Instructions

In order to provide fast interrupt response, Multiply/Divide instructions in the microcontrollers of the C16x family are interruptable. Therefore, in each interrupt service routine which has interrupted a MUL/DIV instruction and which also uses MUL/DIV, the status of registers MDH, MDL, and MDC must be saved/restored. In addition, MDC must be cleared to be correctly initialized for a MUL/DIV instruction which is executed in this interrupt service routine.

In the following, two possible software implementations will be compared.

Version 1 (unconditional):

```
Save: SCXT  MDC, #0h   ; save & clear MDC
      PUSH  MDH
      PUSH  MDL
Start:      ....       ; perform multiply/divide operation
      ....
Restore:
      POP   MDL
      POP   MDH
      POP   MDC
```

Version 2 (conditional):

```
Save: JNB   MDRIU, Start   ; test MDRIU for MDH/MDL in use
      SCXT  MDC, #10h      ; save & clear MDC, leave MDRIU = 1
      PUSH  MDH
      PUSH  MDL            ; implicitly sets MDRIU = 0
      BSET  Saved_Task_xy  ;  task  specific  indication  of  stored
state
Start:      ....           ; perform multiply/divide operation,
                             implicitly sets MDRIU = 1
      ....
Restore:
      JNB   Saved_Task_xy  , Done
      POP   MDL            ; implicitly sets MDRIU = 1
      POP   MDH
      POP   MDC            ; MDRIU = 1 in MDC poped from stack
      BCLR  Saved_Task_xy
Done: ....
```

Note:

For both of the versions shown above, it is assumed that an interrupt routine which has interrupted a MUL/DIV instruction will return to the interrupted MUL/DIV with RETI at the end of the service routine, i.e. no stack manipulations of the return address have been performed. For the special case where e.g. a task scheduler in a real time operating system modifies the return address see section 'Handling of interrupted MUL/DIV by Task Schedulers ..'.

The following table compares required code space and estimated execution time overhead of the save/restore sequences.

	Code Size	State Saved	Estimated Execution Time Internal ROM/Flash	Estimated Execution Time External Bus
Version 1	14 bytes	YES	6 cycles	7 word bus cycles
Version 2, MDRIU = 0	26 bytes	NO	4 (5)[1] cycles	6 (4)[2] word bus cycles
Version 2, MDRIU = 1	26 bytes	YES	10 cycles	13 word bus cycles

Internal ROM/Flash: 1 cycle = 4 TCL (= 100 ns @ 20 MHz internal system clock)
External Bus: 1 bus cycle = f(bus type, MCTC, MTTC, ALECTL)

Notes: 1) 5 cycles if instruction at label Done is double word instruction at odd word address

2) 4 word bus cycles if multiplexed bus with Tristate Waitstate (MTTC = 0) is used

Version 1 always pushes 3 words of status information on the system stack each time an interrupt (which will also use MUL/DIV) occurs. This sequence may be interrupted after any instruction, even if the status of the interrupted MUL/DIV has not been completely saved, since any interrupting routine includes an identical save sequence. The order in which the registers are saved is uncritical.

Version 2 only pushes 3 words of status information if a MUL/DIV instruction was actually interrupted. If this sequence is interrupted before the status of the interrupted MUL/DIV is completely saved, the MDRIU flag which is still set will signal to the interrupting routine that it must save/restore the status of the interrupted MUL/DIV. In this instruction sequence, the order in which the registers are saved is important: the PUSH MDL instruction must be the last instruction in the save sequence since it clears MDRIU. Under worst case conditions (each interrupt routine which uses MUL/DIV interrupts another routine's MUL/DIV), the stack space required by the 2 versions is identical.

C-compilers for the C16x family may choose implementations which are similar to the sequences described above, depending whether optimization for code size or execution speed is performed.

2 Detailled Explanation

Two status flags are associated with the multiply/divide logic of the C16x microcontrollers:
MDRIU (Multiply/Divide Registers In Use) in register MDC
MULIP (Multiply In Progress) in register PSW
The purpose of flag **MDRIU** is to indicate to interrupt service routines that the MDL/MDH registers are in use by another routine, and that these registers must be saved before a new MUL/DIV is started in the interrupt routine, and restored before return to the interrupted program section. MDRIU is set to '1' either when MDL or MDH is written to by software, or when a MUL/DIV instruction has been started. MDRIU is reset to '0' when MDL is read by software. This means that when MDRIU = 1, either MDL/MDH have been initialized with the dividend before a Divide instruction, or they contain intermediate results when a MUL/DIV has been interrupted, or they contain the final result of a MUL/DIV operation. When accessing MDL/MDH, MDH should be read before MDL to guarantee the correct semaphore function of the MDRIU flag with respect to interrupts.

After reset, and after any completed MUL/DIV operation, register MDC is set to 0000h. In case a MUL/DIV instruction has been interrupted, however, register MDC contains additional status information about how to restart the interrupted instruction (e.g. in which cycle). In this case, the contents of MDC must be saved/restored along with MDH/MDL in an interrupt routine which also uses MUL/DIV. In addition, register MDC must be cleared before a new MUL/DIV is started in an interrupt routine, otherwise a wrong result will be generated. Although the case that a MUL/DIV has been interrupted could simply be identified by examining the MULIP flag in the PSW, it is not worth the overhead of testing this flag and only saving/restoring MDC when MULIP = 1. Therefore, this option is not considered in the program examples, and MDC is always saved/restored when MDL/MDH are saved/restored.

The purpose of the **MULIP** flag is to properly restart an interrupted Multiply instruction. When a MUL/DIV instruction is interrupted, the CPU automatically saves the PSW, IP (and CSP if segmentation is enabled) on the internal system stack, and then sets MULIP = 1 in the PSW of the task which has interrupted the MUL/DIV instruction. The return address (IP/CSP) saved on the stack in this case is the address of the interrupted MUL/DIV instruction. When the RETI instruction is executed, the current state of MULIP (before it is restored from the stack) is tested. If MULIP = 1, the interrupted MUL/DIV will automatically be restarted and continued from where it was interrupted.

When an interrupt occurs after an instruction other than MUL/DIV, the current PSW contents (including the current state of MULIP) is saved on the stack, and the MULIP flag in the PSW of the interrupting task will be set to '0'.

Exception:
When a (higher priority) interrupt occurs immediately after a RETI instruction, and the state of MULIP was '1' in the cycle before the RETI instruction, MULIP will again be set to '1' in the PSW of the interrupting task. This is the special case where a MUL/DIV had been interrupted, the interrupting task has been completed, and the interrupted MUL/DIV is about to be restarted, but a higher priority interrupt was generated while the RETI instruction was processed.

In the first cycle of a restarted Multiply instruction, (only) the multiplier is read again from the specified register. This situation is different from the case where a Multiply instruction is originally fetched from memory and started from the beginning: in this case, both the multiplicand and the multiplier are read in the first cycle. As long as an interrupted Multiply instruction is returned to by the corresponding RETI instruction, the CPU automatically handles interrupt nesting correctly via the system stack, and no special actions are required by the user software.

3 Handling of interrupted MUL/DIV Instructions by Task Schedulers of Real-Time Operating Systems

Special care of the correct handling of the MULIP flag must be taken by task schedulers when the return address is modified on the system stack before a RETI is performed to realize a task switch:

Since the MULIP flag in the PSW of the **interrupting** task (in this case: interrupt from task scheduler) is set according to the status of the (last) MUL/DIV instruction (interrupted: MULIP = 1/not interrupted: MULIP = 0) of the **interrupted** task, it must be considered as part of the task context of the interrupted task. Therefore, it must be saved/restored accordingly by the task scheduler together with the rest of the task context.

The following problems may occur if MULIP in the current PSW and/or the return address on the system stack have been manipulated by software, and then a RETI istruction is executed:

- If **MULIP = 1** in current PSW and the instruction which is returned to by RETI is **not** a Multiply instruction, and this instruction uses a GPR (R0 .. R15/RL0 ..RH7), the wrong register may be selected and wrong results may be generated.
- If **MULIP = 0** in current PSW while RETI returns to an interrupted **Multiply** instruction, wrong multiply results will be generated.