Support Group Application Note
Number: 256
Issue: 1.00
Author: JR/JB

Acorn

Writing FIQ code on RISC OS 3.5

This Application Note describes how to write/convert FIQ routines for RISC OS 3.5.

Applicable Hardware :	The RiscPC range of computers.	Related Application Notes:	None.

Copyright © Acorn Computers Limited 1994

Every effort has been made to ensure that the information in this leaflet is true and correct at the time of printing. However, the products described in this leaflet are subject to continuous development and improvements and Acorn Computers Limited reserves the right to change its specifications at any time. Acorn Computers Limited cannot accept liability for any loss or damage arising from the use of any information or particulars in this leaflet. ACORN, ECONET and ARCHIMEDES are trademarks of Acorn Computers Limited.

Support Group
Acorn Computers Ltd
Acorn House
Vision Park, Histon
Cambridge CB4 4AE
England

Introduction

For the majority of cases, Medusa will run the ARM 610/700 in 26-bit mode for compatibility with previous ARM processors and the software written for them. However, FIQ routines are entered with the ARM in 32-bit configuration. Programmers need to take the following notes into account when converting or writing FIQ code to run on a Medusa.

The FIQ routine itself

Your FIQ routine will be entered in with the ARM in 32-bit configuration. In this configuration any instructions with the PC as destination and the S bit set will do the operation as specified **and** move the saved PSR to the current PSR. The effect this has is that these instructions:

TEQP, TSTP, CMPP, CMNP

become useless as the results get discarded and the mode drops back into whatever mode was interrupted to do the FIQ. These instructions have been defined to be not necessarily supported in future ARMs when running in 32-bit mode. These instructions:

<dataop><cc>S	pc, <reg>, <other>	(eg SUBS pc,lr,#4)
LDM<cc><dba>	<reg>, {<with PC>}^	(eg LDMIA {r0-r3 ,pc}^)

can only sensibly be used as returns from interrupt.

**** do not enable interrupts in 32bit mode ****

RISC OS 3.50 does not preserve the 32 bit status when returning from IRQs or other interrupts (eg SWI). This means that a piece of code executing in 32 bit mode with IRQs enabled may, at any time, drop into 26 bit mode. Normally direct flag manipulation in 32-bit mode would be achieved using MRS and MSR instructions, but for your purposes you are unlikely to need or want to do this in a FIQ routine due to the amount of programming effort this would entail.

What does all this mean?

- subtractions in 32 bit mode will tend to not preserve flags

- returning from FIQ would be acheived using SUBS pc,lr,#4

Installing your FIQ routine

The ARM processor prohibits direct poking of the processor vectors whilst in 26 bit mode. This is

reasonable as it catches almost certainly OS-damaging 26-bit code before it does any harm (in other words, it forces the authors of such code to rewrite the code in the light of the fact that this code will now be entered in 32-bit mode, not 26-bit mode as before). As FIQ routines in RISC OS tend to be very small, they also tend to need no conversion for the ARM600. Hence, to help with compatibility, RISC OS will detect a write to location &1C (the FIQ vector) and allow this to happen. Other writes to the processor vectors are left as exceptions. If you need rapid FIQ vector updates then your code must be altered to switch to 32 bit mode whilst doing the FIQ vector update. ADFS does this and the relevant section of code is shown below :-

```
        ; Switch to _32 mode with IRQs and FIQs off
        ; Note must switch interrupts off before switching mode as
        ; there can be an interrupt after the msr instruction
        ; but before the following instruction.
        ; For non-32-bit processors this section reads:
        ; NOP
        ; Push "r1"
        ; ORR   r1, r1, #number
        ; NOP
        ; ORR   r1, r1, #number
        ; NOP

        mrs     AL, r1, CPSR_all
        Push    "r1"
        ORR     r1, r1, #I32_bit :OR: F32_bit
        msr     AL, CPSR_all, r1
        ORR     r1, r1, #2_10000
        msr     AL, CPSR_all, r1

        NOP
        MOV     LR, #FiqVector          ; FIQ vector address

        ; Copy handler
40      LDR     R1, [R0], #4            ; Get opcode
        TEQS    R1, #0                  ; All done?
        STRNE   R1, [LR], #4            ; No then copy to FIQ area
        BNE     %BT40                   ; And repeat

        ; And switch back - this bit reads as follows for non-32-bit processors:
        ; Pull  "r1"
        ; NOP
        Pull    "r1"
        msr     AL, CPSR_all, r1

******* and from the headers: ******************

; ARM6 PSR transfer macros

; Condition code symbols

Cond_EQ *       0  :SHL: 28
Cond_NE *       1  :SHL: 28
```

```
Cond_CS *       2  :SHL: 28
Cond_HS * Cond_CS
Cond_CC *       3  :SHL: 28
Cond_LO * Cond_CC
Cond_MI *       4  :SHL: 28
Cond_PL *       5  :SHL: 28
Cond_VS *       6  :SHL: 28
Cond_VC *       7  :SHL: 28
Cond_HI *       8  :SHL: 28
Cond_LS *       9  :SHL: 28
Cond_GE *       10 :SHL: 28
Cond_LT *       11 :SHL: 28
Cond_GT *       12 :SHL: 28
Cond_LE *       13 :SHL: 28
Cond_AL *       14 :SHL: 28
Cond_   * Cond_AL
Cond_NV *       15 :SHL: 28

; New positions of I and F bits in 32-bit PSR

I32_bit *       1 :SHL: 7
F32_bit *       1 :SHL: 6
IF32_26Shift *  26-6

; Processor mode numbers

USR26_mode      *       2_00000
FIQ26_mode      *       2_00001
IRQ26_mode      *       2_00010
SVC26_mode      *       2_00011
USR32_mode      *       2_10000
FIQ32_mode      *       2_10001
IRQ32_mode      *       2_10010
SVC32_mode      *       2_10011
ABT32_mode      *       2_10111
UND32_mode      *       2_11011

; New register names

r13_abort       RN      13
r14_abort       RN      14
lr_abort        RN      14

r13_undef       RN      13
r14_undef       RN      14
lr_undef        RN      14

        MACRO
        mrs     $cond, $rd, $psrs
        LCLA    psrtype
psrtype SETA    -1
```

```
[ "$psrs" = "CPSR" :LOR: "$psrs" = "CPSR_all"
psrtype SETA    0 :SHL: 22
 ]
 [ "$psrs" = "SPSR" :LOR: "$psrs" = "SPSR_all"
psrtype SETA    1 :SHL: 22
 ]
        ASSERT  psrtype <> -1
        ASSERT  $rd <> 15
         &       Cond_$cond :OR: 2_00000001000011110000000000000000 :OR: psrtype :OR: ($rd
:SHL: 12)
        MEND

        MACRO
        msr     $cond, $psrl, $op2a, $op2b
        LCLA    psrtype
        LCLS    op2as
        LCLA    op
        LCLA    shift
psrtype SETA    -1
 [ "$psrl" = "CPSR" :LOR: "$psrl" = "CPSR_all"
psrtype SETA    (0:SHL:22) :OR: (1:SHL:19) :OR: (1:SHL:16)
 ]
 [ "$psrl" = "CPSR_flg"
psrtype SETA    (0:SHL:22) :OR: (1:SHL:19) :OR: (0:SHL:16)
 ]
 [ "$psrl" = "CPSR_ctl"
psrtype SETA    (0:SHL:22) :OR: (0:SHL:19) :OR: (1:SHL:16)
 ]
 [ "$psrl" = "SPSR" :LOR: "$psrl" = "SPSR_all"
psrtype SETA    (1:SHL:22) :OR: (1:SHL:19) :OR: (1:SHL:16)
 ]
 [ "$psrl" = "SPSR_flg"
psrtype SETA    (1:SHL:22) :OR: (1:SHL:19) :OR: (0:SHL:16)
 ]
 [ "$psrl" = "SPSR_ctl"
psrtype SETA    (1:SHL:22) :OR: (0:SHL:19) :OR: (1:SHL:16)
 ]
        ASSERT  psrtype <> -1
 [ ("$op2a" :LEFT: 1) = "#"
 ; Immediate operand

op2as   SETS    "$op2a" :RIGHT: ((:LEN: "$op2a")-1)
op      SETA    $op2as

  [ "$op2b" = ""
  ; Rotate not specified in immediate operand
shift   SETA    0
        WHILE   (op :AND: &FFFFFF00)<>0 :LAND: shift<16
op      SETA    ((op:SHR:30):AND:3):OR:(op:SHL:2)
shift   SETA    shift + 1
        WEND
```

```
ASSERT  (op :AND: &FFFFFF00)=0
  |
  ; Rotate of immediate operand specified explicitly
        ASSERT  (($op2b):AND:&FFFFFFE1)=0
shift   SETA    ($opt2b):SHR:1
  ]
op      SETA    (shift :SHL: 8) :OR: op :OR: (1:SHL:25)
 |

 ; Not an immediate operand
  [ "$op2b" = ""
  ; Unshifted register
op      SETA    ($op2a) :OR: (0:SHL:25)
  |
        ! 1, "Shifted register not yet implemented in this macro!"
  ]
 ]
        &       Cond_$cond :OR: 2_00000001001000001111000000000000 :OR: op :OR: psrtype
        MEND

; SetMode newmode, reg1, regoldpsr
;
; Sets processor mode to constant value newmode
; using register reg1 as a temporary.
; If regoldpsr is specified, then this register
; on exit holds the old PSR before the mode change
; reg1 on exit always holds the new PSR after the mode change

        MACRO
        SetMode $newmode, $reg1, $regoldpsr
 [ "$regoldpsr"=""
        mrs     AL, $reg1, CPSR_all
        BIC     $reg1, $reg1, #&1F
        ORR     $reg1, $reg1, #$newmode
        msr     AL, CPSR_all, $reg1
 |
        mrs     AL, $regoldpsr, CPSR_all
        BIC     $reg1, $regoldpsr, #&1F
        ORR     $reg1, $reg1, #$newmode
        msr     AL, CPSR_all, $reg1
 ]
        MEND
```

This code is also provided on the disc which comes with this Application Note.