



KTH Microelectronics
and Information Technology

Embedded systems

Laboratory 2 - Interrupt¹

1 Introduction

1.1 Goals

After this lab you should understand polling, unsigned and signed arithmetic and basic interrupt handling.

1.2 Literature

Chapters 3.1-3.3 in: Wolf, Computers as Components.

[ARM Application Note 25: "Exception Handling on the ARM"](#)

2 Preparations

Read the required literature and this lab manual in detail and then solve the home assignments. Note that this lab requires more home preparations than the first one.

3 The ARM modes

An ARM processor can be in any of seven different modes. The user mode has the normal 16 registers and the status register CPSR. The other modes are privileged modes where there is no restriction on the kinds of instructions that may be executed. Except from the System mode, all other modes define a number of additional registers that are accessible only in these modes.

The other modes are:

- System mode: Running privileged operating system tasks.
- Fast interrupt (FIQ) mode: Processing fast interrupts.
- Supervisor (SVC) mode: Processing software interrupts (system calls).

¹ Edited for the course year 2004/2005 by Ingo Sander based on the earlier version of Mats Brorsson, Nguyen Phan Nguyen Thai and Mladen Nikitovic

- Abort mode: Processing memory faults.
- Interrupt (IRQ) mode: Processing normal interrupts.
- Undefined (Undef) mode: Processing undefined instruction traps.

All of these modes (except System mode) define their own registers r13 and r14. In addition, the FIQ mode also defines its own r8-r12 so that it has some extra registers to work with. If the user level registers need to be accessed when you are in one of the privileged modes, you first have to enter System mode and then transfer them through other registers which are visible in both modes.

More information can be found in the [ARM Application Note 25](#) about exception processing.

4 The (modified) ARMulator

For this lab, a modified version of the ARM simulator, the ARMulator, will be used. In order to use this version of the ARMulator, perform the following actions:

- Start the ARM debugger (ADW).
- Select “Options -> Configure Debugger...”
- Press the Add button in “Target Environment”
- Find the file “thaimulate.dll” and open it in the dialog window
- Make sure thaimulate has been selected as target environment
- Press configure and make sure that the processor variant ARM9TDMI

This model has a simple I/O device as shown in Figure 1. It consists of the following items:

- A row of switches which constitute an eight-bit inport connected to address 0x70001000.
- A row of LEDs which constitute an eight-bit output connected to address 0x70001004.
- Two push-buttons; K1 and K2 and a timer which forms an interrupt control port at address 0x70001008.
- The timer has a special control port at address 0x7000100c.

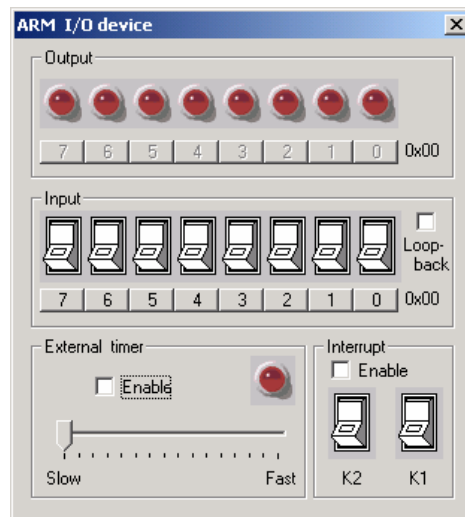


Figure 1. The user interface of the simple I/O device

The switches and the LEDs are simple to use and understand. When you write a byte to address 0x70001004, the binary value will appear on the LEDs. When you read a byte from address 0x70001000, you will get the value of the switches. The “Loop-back” check box on the user interface connects the switches to the LEDs and should only be used for testing purposes.

A read operation to address 0x70001008 returns the following:

- Bit 0: Latched K1 inactive (0) / active (1)
- Bit 1: Latched K2 inactive (0) / active (1)
- Bit 2: Latched external timer inactive (0) / active (1)
- Bit 3: External interrupt disabled (0) / enabled (1)
- Bit 4: Current value of K1
- Bit 5: Current value of K2
- Bit 6: Timer signal high/low

Buttons K1 and K2 are, as well as the timer, connected to the IRQ signal on the simulated ARM processor. K1 and K2 generates an interrupt as soon as they are pressed. The timer generates a periodic signal and generates an IRQ whenever the signal goes from 0 to 1. The interval of the period is controlled by the timer value, which can be loaded as described below. Interrupts are acknowledged by writing the value 0xf7 to the interrupt control port.

The timer can be controlled through port 0x7000100c according to the following:

Reading from timer control port:

- Bits: 0-2: N/A
- Bit 3: 1: Timer is enabled: caused interrupt at the time timer LED turn on
0: Timer is disabled: do not use the external timer
- Bits 4-7: Value of timer.

In write command:

- 0b*0011: Disable timer
- 0b*1011: Enable timer
- 0bxxxx0111: Set external timer value (0bxxxx: value)
- 0bxxxx1111: Set external timer value (0bxxxx: value) and enable timer

In addition, the timer can be enabled/disabled using the “Enable” check box on the user interface.

4.1 Home Assignment 1

Study the following assembly program, which is designed for investigation of arithmetic operations:

```
; Laboratory Exercise 2, Home Assignment 1
```

```
SWITCHES    EQU    0x70001000
LEDS        EQU    0x70001004
BUTTONS     EQU    0x70001008
```

```

        AREA lab2_1, CODE
        ENTRY

Start
        BL    wait            ; Wait for button click
        LDR   r4,=SWITCHES   ; Load switch port address
        LDRB  r5,[r4]        ; Read first number from switches
        BL    wait            ; Wait for button click
        LDRB  r6,[r4]        ; Read second number from switches
        ADDS  r3,r5,r6       ; Perform an arithmetic operation
        LDR   r4,=LEDS       ; Load LED port address
        STRB  r3,[r4]        ; Write the result to LEDs
        B     Start          ; Repeat all over again

        ;; Add code for wait subroutine here! ;;

        END

```

Write a subroutine *wait* that will wait until the button K2 is pressed, then wait until the button is released, and then just return. It should read from the address 0x70001008 until it receives a 1 in bit position 5 from this read operation, and then read until the bit becomes 0 again, and then return. This method of receiving information from the outer world is called *polling*. The subroutine should be called with the instruction **BL wait**, as shown in the main program above. *Hint*: Use the instruction TST to test a particular bit. It does the same thing as an ANDS but throws away the result.

Integrate your *wait* routine into the source code Lab2_HW1.s that can be found on the course web page.

4.2 Home Assignment 2

Study the following assembly program, which waits for interrupts and prints out information about which interrupts it receives. Go over the code in detail and make sure that you understand everything, especially how to write and install an interrupt routine, how to enable an interrupt, and what happens when an interrupt is activated.

```

        ; Laboratory Exercise 2, Home Assignment 2

        ; This is a simple program to illustrate the idea of
        ; interrupts. The interrupt (IRQ) routine start address is
        ; 0x18. Only a branch instruction that immediately
        ; jumps to the real interrupt routine is stored at this
        ; address. The branch instruction is put at this address
        ; during the program initialization.

        AREA my_data, DATA

K1_string      DCB "K1 generated interrupt\n",0

```

```

K2_string      DCB "K2 generated interrupt\n",0
Timer_string   DCB "Timer generated interrupt\n",0
Next           DCB "Waiting for next...\n",0

```

```

        AREA my_code, CODE
        ; Interrupt routine.

```

```

BUTTONS EQU    0x70001008

```

```

introutine

```

```

        STMFD sp!,{r0-r4,r14} ; Push registers r0-r4 and r14
                                ; on the IRQ stack

```

```

        LDR  r4,=BUTTONS      ; Put address to Control port in r4
        LDR  r2,[r4]          ; Read control port
;
        AND  r2,r2,#0x3       ; Mask out button bits
        TST  r2,#0x1         ; Test if K1
        LDRNE r0,=K1_string   ; If so, use K1_string
        BLNE wrline          ; Print the string
        TST  r2,#0x2         ; Test if K2
        LDRNE r0,=K2_string   ; If so, use K2_string
        BLNE wrline          ; Print the string
        TST  r2,#0x4         ; Test if timer
        LDRNE r0,=Timer_string; If so, use Timer_string
        BLNE wrline          ; Print the string

        LDR  r0,=Next        ; Waiting for next
        BL   wrline          ; Print the string

        MOV  r0,#0xf7        ;
        STR  r0,[r4]         ; Acknowledge interrupts

        LDMFD sp!,{r0-r4,r14} ; Pop registers r0-r4,r14 from the
                                ; IRQ stack

        SUBS r15,r14,#4      ; Return from interrupt

```

```

ENTRY

```

```

Start  LDR  r0,=BUTTONS
        MOV  r1,#0xf7
        STR  r1,[r0]        ; Reset interrupt port

```

```

; The following code initializes the Interrupt vector
; for IRQ.

```

```

ADR  r1,introutine ; Get Introutine address
SUB  r1,r1,#0x8    ; Subtract by 8 because of pipeline
SUB  r1,r1,#0x18   ; Subtract by 0x18 is the address of
                    ; PC at the time IRQ start execute
MOV  r1,r1,LSR #2  ; Shift right 2 bit to get offset by

```

```

; word instead of offset by byte
ADD r1,r1,#0xea000000 ; Add Branch instruction
MOV r0,#0x18 ;
STR r1,[r0] ; Copy to IRQ vector

BL enable_irq

Loop b Loop ; Wait for interrupt

; Support routines

IRQ EQU 0x80
; This function enables the IRQ interrupt
enable_irq
MRS r0, CPSR ; Read status word
BIC r0,r0,#IRQ ; Clear the IRQ enable bit
MSR CPSR_c, r0 ; Write back control part of CPSR
MOV r15,r14 ; Return

; This function prints a null-terminated string
; r0 points to the string
wrline MOV r1,r0 ; Copy the pointer to r1
MOV r0,#0x4 ; Set the SYS_WRITE0 code
SWI 0x123456 ; Make the system call
MOV r15,r14 ; return from this function

END

```

The source code Lab2_HW2.s can be found on the course web page.

4.3 Home Assignment 3

Study the following assembly program. Whenever a button is pressed (K1 or K2), it will copy the current position of the switches on the lab board to the LEDs. At the same time, the program pretends to perform a demanding computation, in this case a long loop. Make sure you understand how the program works and why. This method of repeatedly checking for input is called polling.

```

; Laboratory Exercise 2, Home Assignment 3

SWITCHES EQU 0x70001000
LEDS EQU 0x70001004
BUTTONS EQU 0x70001008

AREA my_code, CODE
ENTRY

Start

Loop BL Comp ; Perform heavy computations

```

```

LDR    r0,=BUTTONS      ; Place adress of buttons in r0
LDR    r4, [r0]         ; Load button port value
TST    r4, #0x70        ; Mask out button indication bits
BEQ    Loop             ; Loop if no button pressed

LDR    r0,=SWITCHES    ; Place adress of switches in r0
LDR    r4, [r0]         ; Load switch position
LDR    r0,=LEDS        ; Place adress of leds in r0
STR    r4, [r0]         ; Output switch position to LEDs
B      Loop             ; Repeat polling loop

Comp   LDR    r0,=0x7ffff ; Initialize counter value
Delay  SUBS   r0,r0,#1    ; Decrease counter by 1
      BNE   Delay        ; Test if ready
      MOV   r15,r14      ; Return to polling loop

      END

```

The source code Lab2_HW3.s can be found on the course web page.

4.4 Home Assignment 4

Study the following assembly program. It should perform the same tasks as the program of Home Assignment 3, but is implemented using interrupts instead of polling. Add the missing code for the interrupt routine.

```

      ; Laboratory Exercise 2, Home Assignment 4

SWITCHES    EQU    0x70001000
LEDS        EQU    0x70001004
BUTTONS     EQU    0x70001008

IRQ EQU 0x80

      AREA my_code, CODE

Introutine
      ; Add your code here!

      ENTRY

Start  LDR    r0,=BUTTONS
      MOV    r1,#0x7f
      STR    r1,[r0]          ; Reset interrupt port

      ADR    r1,Introutine    ; Get Introutine address
      SUB    r1,r1,#0x8       ; Substract by 8 because of pipeline
      SUB    r1, r1, #0x18    ; Substract by 0x18 is the address of PC
      ; at the time IRQ start execute
      MOV    r1, r1, LSR #2   ; Shift right 2 bit to get offset by word
      ; instead of offset by byte

```

```

        ADD    r1, r1, #0xea000000 ; Add Branch instruction
        MOV    r0,#0x18           ;
        STR    r1,[r0]           ; Copy to IRQ vector

        BL     enable_irq        ; Enable interrupts

Loop    BL     Comp              ; Perform heavy computations
        B     Loop              ; Repeat loop

; This function enables the IRQ interrupt
enable_irq
        MRS   r0, CPSR           ; Read status word
        BIC   r0,r0, #IRQ       ; Clear the IRQ enable bit
        MSR   CPSR_c, r0        ; Write back control part of CPSR
        MOV   r15,r14           ; Return

Comp    LDR   r0,=0xffffffff     ; Initialize counter value
Delay   SUBS  r0, r0, #1         ; Decrease counter by 1
        BNE  Delay              ; Test if ready
        MOV  r15,r14           ; Return loop

        END

```

The source code Lab2_HW4.s can be found on the course web page.

5 Lab exercises

5.1 Assignment 1

Start the ARM Project Manager and create a project where you use the source code of home assignment 1 together with your *wait* routine. Build the program and upload it to the computer. Use the disassembler to verify that the code is correct.

5.2 Assignment 2

Run the program and use the switches and LEDs on the simulator to investigate integer addition, as defined by the instruction **ADDS**. This operation performs integer addition. Try the additions $1 + 1$, $0xFF + 1$, and others. Also investigate 32-bit additions. You may feed these longer values directly into the registers or into the data of a program. Try to load an 8-bit value from the switches by using the instruction **LDRSB** instead of **LDRB** and loading the value $0xFF$. What happens with the data in the register? Why? When do special or unusual results appear in addition? Verify the expected behaviour by looking in the registers in the debugger.

Repeat the experiments with **SUBS** instead of **ADDS**.

5.3 Assignment 3

Interrupts are used to handle external events and as an interface to the operating system. In this and the following assignments we will study how interrupts can be used, and what interrupt programming looks like. We will also compare polling to using interrupts.

Create a new project, type in, and build the program of Home Assignment 2. Execute the program on the simulator and investigate the effects of the different interrupts in detail.

- What does the CPSR register contain after an interrupt?
- How is the interrupted program CPSR saved?
- How does the processor know when the interrupt routine should be executed?
- How does the code that initializes the interrupt (IRQ) vector work?
- What does it mean to enable an interrupt?
- How does the processor know which interrupts that are enabled?

5.4 Assignment 4

A computer can react to external events either by polling or by using interrupts. One method is simpler, while the other one is more systematic and also more efficient. We will study the similarities and differences of these methods using a simple "toy" example program.

Let us assume that we want a program to respond to the pressing of one of the buttons on the simulated computer system by reading the positions of the switches and outputting a similar pattern on the LEDs. In other words, the user should be allowed to set the switches, and *at the moment* the K1 or K2 button is pressed, the pattern should be transferred from the switches to the LEDs.

At the same time, the program should also perform some time-consuming computations. In this case, these will be simulated by a long loop in which a counter is decreased to zero. The point of this "toy" program is to exemplify how a program can handle two different tasks (responding to a pressed button and performing a CPU-intensive computation) seemingly almost simultaneously.

Create a new project, type in, and build the program of Home Assignment 3. Execute the program on the lab computer hardware.

- How long does it take the program to respond to a pressed button?
- Why does it take the program this long to respond?
- What can be done to get a quicker response?

5.5 Assignment 5

Create a new project, type in, and build the program of Home Assignment 4. Execute the program on the simulator. Use the single step facility (F10) to verify that the interrupt routine works as expected.

6 Conclusions

Before you pass the laboratory exercise, think about the questions below and explain to your lab assistant:

- How does the computer know whether a number is signed or unsigned?
- What is an overflow?
- What happens when an overflow occurs?
- What is polling?
- What are interrupts?
- What are interrupt routines?
- What are the advantages of polling?
- What are the advantages of using interrupts?
- What are the differences between interrupts, exceptions, and traps?