ECE 212: Microprocessors, Hardware, Software, and Interfacing

Lab 5: Interrupts Handling Lab

Karysse Hay
College of Engineering
University of Miami
Coral Gables, United States
kdh85@miami.edu

***Abstract*— The ARM processor that we have been exploring has two levels of external interrupt, FIQ and IRQ. They are both level-sensitive active LOW signals into the processor. For an interrupt to be taken, the appropriate 'disable bit' in the CPSR must be clear. The objective of this lab is to program interrupts, in this case, a key press in the Altera DE1-SoC board.**

Keywords*— *FIQ, IRQ, level-sensitive, CPSR, key press, Altera DE1-SoC board

I. CHALLENGE

In this lab, a few files were given where lines of code were omitted. The task at hand was to complete the code in order to fully setup the interrupt [1]. To handle an exception involves many steps- the processor must change to the respective mode, save the previous mode CPSR into SPSR of the new node, save the PC in LR of the new node, disable other exceptions of the same priority and then branch to a specific entry in the vector table. For this lab, the steps listed were followed so that an interrupt key press is handled by Cortex A9 Processor effectively.

II. APPROACH

The arm cortex a-9 manual [1] and corresponding lectures were looked at beforehand. This allowed for a better understanding of interrupts and how they work. For Part I, the code was observed thoroughly in order to determine the missing parts which helped. Whereas, for Part II, the 'Hints and tips' were taken into consideration when creating the code for the press buttons.

III. EXPERIMENT SETUP

A. Overview

The Intel program on the computer was opened. The required files were downloaded and opened in Intel. The file "define.s" is a compilation of definitions. The main benefit to putting all the definitions in a file is to save time and increase efficiency. This way, key definitions can be defined once and referenced throughout the program. This also facilities with the organization of the program.

B. Part A

The code in the file "intrruptProgram.s" had to be completed. In the lab description, the listed instructions were explained and completed during scheduled lab hours using knowledge gained from previous lectures. Each of the instructions can be seen in parts a) - e) of this lab report with their corresponding explanations. The completed program can be seen in Figure 1.

a) Inspect the instructions in the interruptProgram.sfile. What is following instruction doing? Explain your answer!

This line is performing a bitwise OR operation. The value contained in "IRQ_MODE" was 0b10010 and the value contained in "INT_DISABLE" was 0b11000000. Therefore, 0b11010010 was moved into the register, R1 [1]. This register also corresponds to the CPSR which means that we can say that R1 was used to set the flags of the CPSR manually.

b) Complete the BL instruction. Hint: this instruction is supposed to "configure the ARM generic interrupt controller"

One of the files was named "CONFIG_GIC". This file was downloaded and added to the program. The program required for the code to branch to it at this very instant. This resulted in the instruction: "BL CONFIG_GIC" [2].

c) Complete the STR instruction. Hint: this instruction will update the interrupt mask register, and the interrupt mask bits have been saved in R1. So, what are you saving/storing, and where?

The instruction "STR r1, [r0, #8]" [2] was added so that the Interrupt Mask Register could be saved and stored into 'R0+8' [2] for later use.

d) Complete the B instruction. Hint: This program is going to be idling, waiting for the button to be pressed.

In order to make the program idle and enter a never-ending loop, the instruction "B Program" [2] had to be inserted. This helped the program stay in a loop waiting on a key to be pushed.

e) Inside the SERVICE_IRQ section, (1) explain what the code is doing and complete the BL instruction'

This part of the code is comparing the contents of register, R5 [1] and FPGA_IRQ1. If they are not equal, the code will branch to "UNEXPECTED" which will result in the code being stopped.

```
.include "address_map_arm.s"
.include "defines.s"
.include "interrupt_ID.s"
/*
*********************************************
*********************************
 * This program demonstrates use of interrupts with
assembly language code.
 * The program responds to interrupts from the
pushbutton KEY port in the FPGA.
 *
 * The interrupt service routine for the pushbutton KEYs
controls HEX3-0
 *********************************************
*********************************/

.section .vectors, "ax"

B _start // reset vector
B SERVICE_UND // undefined instruction vector
B SERVICE_SVC // software interrupt vector
B SERVICE_ABT_INST // aborted prefetch vector
B SERVICE_ABT_DATA // aborted data vector
.word 0 // unused vector
B SERVICE_IRQ // IRQ interrupt vector
B SERVICE_FIQ // FIQ interrupt vector

.text
.global _start
_start:

/* Set up stack pointers for IRQ and SVC processor
modes */
MOV R1, #INT_DISABLE | IRQ_MODE
MSR CPSR_c, R1 // change to IRQ mode
LDR SP, =A9_ONCHIP_END - 3 // set IRQ stack to top of
A9 on chip memory
/* Change to SVC (supervisor) mode with interrupts
disabled */
MOV R1, #INT_DISABLE | SVC_MODE
MSR CPSR, R1 // change to supervisor mode
LDR SP, =DDR_END - 3 // set SVC stack to top of DDR3
memory

BL CONFIG_GIC
//complete this instruction // configure the ARM generic
interrupt controller

// write to the pushbutton KEY interrupt mask register
LDR R0, =KEY_BASE // pushbutton key base address
```

```
MOV R1, #0xF // enable interrupts from all four KEYs
STR r1, [r0, #8] //Complete this instruction // interrupt
mask register is (base + 8)

// enable IRQ interrupts in the processor
MOV R0, #INT_ENABLE | SVC_MODE // IRQ unmasked,
MODE = SVC
MSR CPSR_c, R0
Program: //this is the main program label.
B Program
//Complete this instruction //main program idles
. /* Define the exception service routines */
/* We are only concerned  IRQs here */
/*--- Undefined instructions ---------------------------------
----------*/
SERVICE_UND:
   B SERVICE_UND

/*--- Software interrupts ------------------------------------
----------*/
SERVICE_SVC:
   B SERVICE_SVC

/*--- Aborted data reads -------------------------------------
----------*/
SERVICE_ABT_DATA:
   B SERVICE_ABT_DATA

/*--- Aborted instruction fetch ------------------------------
----------*/
SERVICE_ABT_INST:
   B SERVICE_ABT_INST

/*--- IRQ ----------------------------------------------------
------*/
/*This section is the IRQ interrupt handler. Look at the
provided instructions,
study what they do and complete the missing
instruction. */
SERVICE_IRQ:
   PUSH {R0-R7, LR}
/* Read the ICCIAR in the CPU interface */
   LDR R4, =MPCORE_GIC_CPUIF
   LDR R5, [R4, #ICCIAR] // read the interrupt ID

FPGA_IRQ1_HANDLER:
   CMP R5, #FPGA_IRQ1
UNEXPECTED: BNE UNEXPECTED     // if not recognized,
stop here
    BL KEY_ISR
//Complete this instruction ..
//if recognized then go somewhere
//and that somewhere is the key-pressed interrupt!
```

```
    EXIT_IRQ:
    /* Write to the End of Interrupt Register (ICCEOIR) */
    STR R5, [R4, #ICCEOIR]

    POP {R0-R7, LR}
    SUBS PC, LR, #4

/*--- FIQ ---------------------------------------------------------------
-----*/
SERVICE_FIQ:
    B SERVICE_FIQ

.end
```

Figure 1: Assembly Code for Part A

C. Part B

For Part B, the file titled, "key.isr" had to be completed. This file was responsible for displaying 0 and 1 on the HEX0 and HEX1 displays [1] of the DE1-SoC board. It can be seen in Part A that the main code idles therefore, in order to display 0 or 1, an interrupt had to be instigated so that when a corresponding pushbutton KEY is pressed, the program will stop, and the desired number will be displayed.

The based address of the keys and the 7-SegDiplay were determined and used when completing Part B. KEY 0 was assigned 0b0001 and KEY1 was assigned 0b0010. Whereas the address for the seven-segment display used was 0xFF200020.

The sections CHECK_KEY (0 to 1) were completed and tested. The code can be seen in Figure 2.

```
.include "address_map_arm.s"
.include "defines.s"

/****************************************
*******0xFF200050*****************************
*****
 * Pushbutton - Interrupt Service
Routine
 *

 * This routine checks which KEY(s) have been pressed. It
writes to HEX3-0
 ****************************************
****************************************/

.global KEY_ISR
KEY_ISR:

LDR R0, =KEY_BASE // base address of pushbutton KEY
parallel port

LDR R1, [R0, #0xC] // read edge capture register
STR R1, [R0, #0xC] // clear the interrupt
LDR R0, =KEY_HOLD // base address of pushbutton KEY
parallel port
LDR R2, [R0] // read global variable
EOR R1, R2, R1 // toggle the bits of the global
STR R1, [R0] // write to the global variable
LDR R0, =HEX3_HEX0_BASE //Complete this instruction
// based address of HEX display
MOV R2, #0 // blank the display by default

CHECK_KEY0:

CMP R1, #0B0001 //Complete this instruction
BNE CHECK_KEY1
MOV R2, #0b00111111

CHECK_KEY1:

CMP R1, #0b00010 //Complete this instruction
BNE END_KEY_ISR
MOV R2, #0b0000110

CHECK_KEY2: B END_KEY_ISR

CHECK_KEY3: B END_KEY_ISR


END_KEY_ISR:
STR R2, [R0] // display " "
BX LR

.global KEY_HOLD
KEY_HOLD: .word 0b0000 // remembers which KEY is
pressed/not
```

Figure 2: Assembly Code for Part B

IV. Results and Analysis

A. Results and Analysis

The ‘config_gic.s” file acted as a setup for the “key_isr.s” file. The contents in the first file named, assisted in saving the original information in a special CPSR [2], that is only created once dealing with interrupts, so that when the interrupt is handled/ processed, the information would be preserved.

For Part B, when KEY0 was pressed, the digit 0 on HEX0 [2] would be displayed and when pressed again, the display would be blank. Hence, the pushbuttons allowed for the display to toggle between the digits 0 and 1 and blank. This can also be seen in Figures 3 and 4.

B. Results for Part A

a) Photo of Results

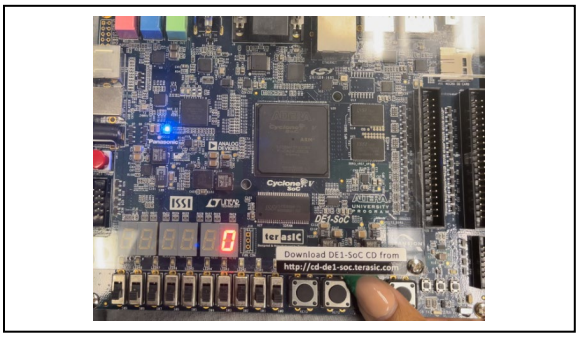

Figure 3: Digit zero displayed on HEX 0 display

b) Comments

In Figure 3, the digit zero is displayed when KEY0 is pressed and blank when it is pressed again.

C. Results for Part B

a) Photo of Results

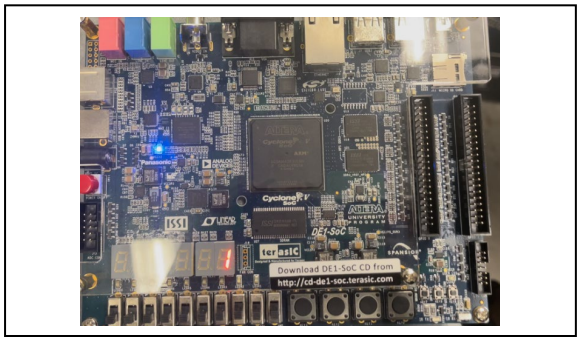

Figure 4: Digit one displayed on HEX 0 display

b) Comments

In Figure 4, the digit one is displayed when KEY1 is pressed and blank when it is pressed again.

V. Conclusion

This lab allowed for the better understanding of interrupt handling. The files given, assisted in referencing to different addresses and definitions. This resulted in an efficient code that was able to display digits at any moment once their respective buttons were pushed. Upon completion of this lab, assembly code, with regards to interrupt handling, was better understood.

References

[1] A. Holdings, “Cortex-a9 technical reference manual, revision r2p0,” ARM Holdings. Report number: DDI0388E, 2009.

[2] ARM Ltd and ARM Germany GmbH, “ Writing ARM Assembly Language,” Assembler User Guide: Writing ARM Assembly Language. [Online]. Available: https://www.keil.com/support/man/docs/armasm/armasm_dom1359731144635.htm. [Accessed: 20-Mar-2021].

[3] “ASCII,” Wikipedia, 19-Aug-2020. [Online]. Available: https://simple.wikipedia.org/wiki/ASCII#/media/File:ASCII-Tablewide.svg. [Accessed: 20-Mar-2021].