Lab2 – IO

ECSE-324 Computer Organization

Sen Wang (260923645)

Friday 16, October 2020

McGill University
Montreal Quebec, Canada

Part1_1: HEX displays

Brief Description:

This assembly program contains three main subroutines. The first one is HEX_clear_ASM. This subroutine will turn off all the segments of the HEX displays passed in the argument through R0. The second one is HEX_flood_ASM. This subroutine will turn on all the segments of the HEX displays passed in the argument through R0. Finally, the third one is HEX_write_ASM. This subroutine receives the HEX displays indices and an integer value between 0-15 through R0 and R1 registers as an argument, respectively. Based on the second argument value (R1), the subroutine will display the corresponding hexadecimal digit on the displays.

Approach Taken:

The approach taken for HEX_clear_ASM and HEX_flood_ASM are similar and straightforward. The subroutines first receive the arguments through R0 register in binary. The program then iterates through each of the bits and use AND instruction to determine which bit contains a one (the bit we want to clear or to flood). The program then branches to the corresponding subroutine depending on which one needs to be displayed or cleared. In those branches the program stores the corresponding value (0s or 7Fs) in the corresponding address. Finally, the HEX_write_ASM uses the similar logic to determine which display needs to be written. However, it has an additional layer of complexity that takes the actual values that needs to be displayed into consideration. So, this subroutine uses an additional subroutine that compares the actual value that is stored in R1 and generate the corresponding hexadecimal value that needs to be stored in the hex_memory address.

Challenges Faced and Solutions

The logic used in this part of the lab is relatively simple. However, the main difficulty for this part of the lab is understanding of these I/O interface memories. It was also very time consuming to determine the value in hexadecimal that represent the numbers 0 to 15 for the hexadecimal displays. To solve these problems, multiple internet searches and readings at the documentation were needed.

Possible Improvement to the Program

This program works but several improvements to the program could be made. For example, this program compared the values (0-15) one by one to determine the hexadecimal value that needs to be stored in the interface memory to displays the corresponding value. Instead of comparing the values one by one, a for loop could be used to reduce the length and clarity of the program. Additionally, this program used a lot of registers. Some registers were only being used to store intermediate calculation values. Reducing the number of registers used could make the program more flexible in the future if we want to make modification on it.

Part1_2: Pushbuttons

Brief Description:

This program contains several subroutines. PB_data_is_pressed_ASM receives pushbuttons indices an argument. Then, it returns 0x00000001 when the corresponding pushbutton is pressed. Read_PB_edgecp_ASM subroutines returns the indices of the pushbuttons that have been pressed and then released (the edge bits from the pushbutton Edgecapture register). The PB_edgecp_is_pressed_ASM receives pushbuttons indices as an argument then it returns 0x00000001 when the corresponding pushbutton has been asserted. The PB_clear_edgecp_ASM clears the pushbuttons Edgecapture register. The enable_PB_INT_ASM receives pushbuttons indices as an argument. Finally, the disable_PB_INT_ASM receives pushbuttons indices as an argument. Then, it disables the interrupt function for the corresponding pushbuttons by setting the interrupt mask bits to 0. This program uses the last four slider switches to set the value of a number from 0-15. This number will be displayed on the HEX display when the corresponding pushbutton is pressed.

Approach Taken:

The approach taken for this program consists of a main loop. This main loop first calls read_slider_switches_ASM and write_LEDs_ASM that are being used to fetch the value that needs to be displayed in the hexadecimal display. The program then branches to the read_PB_data_ASM subroutine that fetch the index of the hexadecimal display that will display the corresponding value. The program then branches to read_PB_edgecp_ASM to determine which of these pushbuttons has been released. With that information, we can use a similar logic to display the corresponding value to the desired hexadecimal display.

Challenges Faced and Solutions:

Multiple challenges were faced for this part of the lab. Just like part1_1, the main challenge for this part was the understanding and the use of the different registers like the edgecapture register. To solve these problems, multiple internet searches and readings at the documentation were needed.

Possible Improvement to the Program:

For part1_2, the program works but several improvements to it could be made. The display_value part of the program used the same implementation as part1_1 of the lab. This means that this program also compared the values (0-15) one by one to determine the hexadecimal value that needs to be stored in the interface memory to displays the corresponding value. Instead of comparing the values one by one, a for loop could be used to reduce the length and clarity of the program. Additionally, this program used a lot of registers. Some registers were only being used to store intermediate calculation values. Reducing the number of registers used could make the program more flexible in the future if we want to make modification on it.

Part 2_1: ARM A9 Private Timer drivers

Brief Description:

This part of the lab contains 3 main subroutines. ARM_TIM_config_ASM is used to configure the timer. ARM_TIM_read_INT_ASM returns the "F" value from the ARM A9 private timer interrupt status register. ARM_TIME_clear_INT_ASM subroutine clears the "F" value in the ARM A9 private timer Interrupt status register. This program should count from 0 to 15 and show the count value on the HEX display (HEX0).

Approach Taken:

The approach taken for this part of the lab is relatively simple. We first have a counter that stars at 0. The program then branches to the display subroutine that display the corresponding value on the HEX0 display. The program then branches to the ARM_TIM_config_ASM that loads the value of 200000000 (1 second for the counter) in its load_address. The subroutine then stores the value 1 in the control_address to start the ticking of the clock. Finally, the counter increments by one and loop back to the start of the program and starts it all over again.

Challenges Faced and Solutions:

For this part of the lab, I did not face any challenge. In fact, most of the difficulties for this part were already solved in the previous parts (configuring the load and control address). The biggest difficulty for this part would be the understanding of the interrupt status. In order to solve this problems, multiple internet searches and readings at the documentation were needed.

Possible Improvement to the Program:

For part2_1, the program works but several improvements to it could be made. The display_value part of the program used the same implementation as part1_1 of the lab. This means that this program also compared the values (0-15) one by one to determine the hexadecimal value that needs to be stored in the interface memory to displays the corresponding value. Instead of comparing the values one by one, a for loop could be used to reduce the length and clarity of the program. Additionally, this program used a lot of registers. Some registers were only being used to store intermediate calculation values. Reducing the number of registers used could make the program more flexible in the future if we want to make modification on it.

Part 2_2: Polling Based Stopwatch

Brief Description:

This part of the lab contains several subroutines. The program has a MAIN_LOOP that constantly checks the value of register R11 that stores the index of the corresponding pushbutton that is being pressed. It also contains several subroutines that clears the pushbutton edge register. ARM_TIM_config_ASM is used to configure the timer. ARM_TIM_read_INT_ASM returns the "F" value from the ARM A9 private timer interrupt status register. ARM_TIME_clear_INT_ASM subroutine clears the "F" value in the ARM A9 private timer Interrupt status register. Finally, UPDATE_VALUES will increment the counter and updates the registers R0 to R5.

Approach Taken:

The approach taken for this program is in certain parts, similar to the implementation of part 2_1. However, instead of incrementing the values every 1 second, we need to increment the values every 0.01 seconds. This means we need to set the value of the timer's load_address 2000000. This program uses 6 registers (from R0-R5) to store all the values that needs to be displayed in the HEX display. The program then branches to a main loop that checks the value R11 (updated using the pushbutton edge register) that stores which pushbutton has been pushed. Depending on the value, the program then branches to the corresponding subroutine that clears the pushbutton edge register. After that, the program branches to the display_values subroutine that has the same implementation as part 1 of the lab. It then branches to ARM_TIM_config_ASM that configures and starts the timer. The program then branches to Check_INT subroutines that check if there is an interrupt which means that the timer ticking stopped. It reads the value at the interrupt_status address and if it is equals to 1, then it branches to the ARM_TIM_clear_INT_ASM subroutine which clears the interrupt_status. Finally, the program branches to update_value which increments the counter and updates the registers R0 to R5.

Challenges Faced and Solutions:

The main challenge I faced for this part of the lab is the understanding and implementation of the pushbutton edge register. During the lab, I often forget to clear this register right after I read it, and this caused s a lot of unwanted behaviors throughout the program. To solve these problems, multiple internet searches and readings at the documentation were needed.

Possible Improvement to the Program:

Several improvements could be made to the program. In fact, the program has several subroutines that serve the same purpose: clearing the pushbutton edge register. This leads to a useless repetition of code. This could be solved by using one single subroutine. The display_value part of the program also used the same implementation as part1_1 of the lab. This means that this program also compared the values (0-15) one by one to determine the hexadecimal value that needs to be stored in the interface memory to displays the corresponding value. Instead of comparing the values one by one, a for loop could be used to reduce the length and clarity of the program.

Part 3: ARM Generic Interrupt Controller

Brief Description:

The program contains several parts. The program includes an exception vector table that allocates one word to each of the various exception types. This word contains branch instructions to the address of the relevant exception handlers. The second part of the program configure the interrupt routine for the pushbuttons and for the timer. We want a stopwatch like part2_2 so we need to load the value 2000000 into the load_address of the timer. The third part of the program define the exception service routines. This part contains an important subroutine called SERVICE_IRQ that checks which interrupt request has occurred using interrupt IDs then the program calls the corresponding ISR. The fourth part of the program is the configuration of the Generic Interrupt Controller.

Approach Taken

The program starts by setting up the exception vector table and configures all the interrupt routines needed by storing the value 15 into their corresponding addresses. It then branches to IDLE that first configure the registers R0-R5 that stores the values for the stopwatch. The program then branches to MAIN_LOOP and waits for an interrupt. When an interrupt happens, the program then branches to SERVICE_IRQ that checks whether if it is the timer or the pushbuttons that requested an interrupt using interrupt IDs. Depending the result, program branches to KEY_ISR or ASM_TIM_ISR. The KEY_ISR subroutine stores the value of the index of the pushed pushbutton in PB_int_flag and the ARM_TIM_ISR subroutines stores the value of the timer interrupt flag in tim_int_flag. Depending on the value in PB_int_flag, the program branches to the corresponding subroutine (BEFORE_MAIN, BEFORE_STOP or BEFORE_RESET). Depending on the value in tim_int_flag, the program decides whether the timer ticking in over and if it should update the values or not.

Challenges Faced and Solutions

The main challenge faced for this part of the lab is the understanding and implementation of the interrupt routines. At the beginning, it was hard to debug because it didn't know I had to indicates a breakpoint in order to branch to the SERVICE_IRQ during the debugging process. To solve these problems, multiple internet searches and readings at the documentation were needed.

Possible Improvement to the Program

Several improvements could be made to this program. Similar to part 2_2, the program has several subroutines that serve the same purpose. This leads to a useless repetition of code. This could be solved by using one single subroutine. The display_value part of the program also used the same implementation as part1_1 of the lab. This means that this program also compared the values (0-15) one by one to determine the hexadecimal value that needs to be stored in the interface memory to displays the corresponding value. Instead of comparing the values one by one, a for loop could be used to reduce the length and clarity of the program.