**Lab Report  
Lab 11: Interrupts & IO programming**

|  |  |
| --- | --- |
| Student ID | Nguyễn Đức Anh |
| Student Name | 20235890 |

**Assignment 5:**

**Code:**

.data

    message: .asciz "Exception occurred.\n"

.text

main:

    try:

        la t0, catch

        csrrw zero, 5, t0 # Set utvec (5) to the handlers address

        csrrsi zero, 0, 1 # Set interrupt enable bit in ustatus (0)

        lw zero, 0 # Trigger trap for Load access fault

    finally:

        li a7, 10 # Exit the program

        ecall

    catch:

        # Show message

        li a7, 4

        la a0, message

        ecall

        # Load finally address to uepc

        la t0, finally

        csrrw zero, 65, t0

        uret

**Explaination:**

.data

    message: .asciz "Exception occurred.\n"

* A string message, "Exception occurred.\n", is defined in the data section. This is used later for output.

main:

    try:

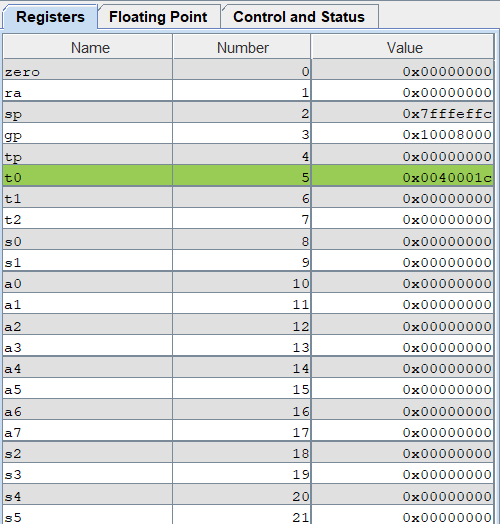
        la t0, catch

        csrrw zero, 5, t0 # Set utvec (5) to the handlers address

        csrrsi zero, 0, 1 # Set interrupt enable bit in ustatus (0)

        lw zero, 0 # Trigger trap for Load access fault

* la t0, catch: Loads the address of the catch label into register t0.
* csrrw zero, 5, t0: Updates the utvec (user trap vector register) with the value in t0 (address of catch). This sets the trap handler address.
* csrrsi zero, 0, 1: Sets the UIE (User Interrupt Enable) bit in the ustatus CSR, enabling user-mode interrupts.csrrw zero, 5, t0: Updates the utvec (user trap vector register) with the value in t0 (address of catch). This sets the trap handler address.
* lw zero, 0: Tries to load a word from memory address 0. Since this is an invalid memory access, it triggers a Load Access Fault, causing the trap handler at catch to execute.



A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

catch:

        # Show message

        li a7, 4

        la a0, message

        ecall

* li a7, 4: Loads the syscall number 4 (print string in RISC-V Linux ABI) into a7.
* la a0, message: Loads the address of the message string into a0.
* ecall: Executes the system call to print "Exception occurred.\n".

A screenshot of a computer

Description automatically generated

# Load finally address to uepc

        la t0, finally

        csrrw zero, 65, t0

        uret

* la t0, finally: Loads the address of the finally block into t0.
* csrrw zero, 65, t0: Updates the uepc (User Exception Program Counter) CSR with the address of finally. The uepc determines where execution resumes after the uret instruction.
* uret: Returns from the trap, resuming execution at the finally block.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

finally:

        li a7, 10 # Exit the program

        ecall

* li a7, 10: Loads the syscall number 10 (exit program in RISC-V Linux ABI) into a7.
* ecall: Executes the system call to terminate the program.

A screenshot of a computer

Description automatically generated

**Assignment 6:**

**Code:**

.data

    msg: .asciz "Overflow Occur!"

.text

    la t0, handler  # Loads the address of the handler function into t0.

    csrrs zero, utvec, t0   # Sets the user trap vector (utvec) to the address of handler.

    csrrsi zero ustatus, 1  # Enables user interrupts by setting the UIE (User Interrupt Enable) bit in the ustatus CSR.

    csrrsi zero, uie, 1 # Enables specific user-mode interrupts in the uie CSR.

    li s1, 0x7FFFFFFF   # s1 = 0x7FFFFFFF

    li s2, 1    # s2 = 1

    add s3, s1, s2  # s3 = s1 + s2, This operation causes an overflow since the result exceeds the maximum positive integer.

    xor t1, s1, s2  # Performs a bitwise XOR between s1 and s2. If s1 and s2 have the same sign, the XOR result will be non-negative.

    blt t1, zero, EXIT  # If t1 is negative (different signs for s1 and s2), the program exits without overflow handling.

    slt t2, s3, s1  # Checks if the result (s3) is less than the first operand (s1).

    blt s1, zero, NEGATIVE  # If s1 is negative, jumps to the NEGATIVE block for further checks.

    beq t2, zero, EXIT  # If t2 is zero (no overflow detected), exits the program.

    j OVERFLOW

    NEGATIVE:

        bne t2, zero, EXIT  # If there is no overflow in the negative case, exits the program.

    OVERFLOW:

        csrrsi zero, uip, 1 # Sets the UIP (User Interrupt Pending) bit in the uip CSR to trigger the interrupt.

    EXIT:

    # Executes the system call to exit the program.

        li a7, 10

        ecall

handler:

    addi sp, sp, -8 # Allocates space on the stack for saving registers.

    sw a0, 0(sp)    # Saves the value of a0 onto the stack.

    sw a7, 4(sp)    # Saves the value of a7 onto the stack.

# Print Message

    li a7, 4

    la a0, msg

    ecall

    lw a7, 4(sp)    # Restores the value of a7 from the stack.

    lw a0, 0(sp)    # Restores the value of a0 from the stack.

    addi sp, sp, 8  # Deallocates the stack space used.

    uret    # Returns from the trap and resumes execution at the point of interruption.

**Result:** (Input, output for each case; is the result same as the theory, etc…)

**A screenshot of a computer

Description automatically generated**

* The program adds s1 = 0x7FFFFFFF (maximum positive 32-bit signed integer) and s2 = 1.
* The result (s3) exceeds the range of a 32-bit signed integer, causing an integer overflow.
* The overflow detection logic identifies this condition and redirects control to the overflow handling mechanism.
* s3 = s1 + s2 = 0x7FFFFFFF + 0x00000001 = 0x80000000.