**CA LAB WEEK 11**

Nguyễn Trọng Minh Phương – 20225992

Assignment 1

|  |
| --- |
| .eqv IN\_ADDRESS\_HEXA\_KEYBOARD 0xFFFF0012  .eqv OUT\_ADDRESS\_HEXA\_KEYBOARD 0xFFFF0014  .text  main:  li t1, IN\_ADDRESS\_HEXA\_KEYBOARD # địa chỉ nhập row index  li t2, OUT\_ADDRESS\_HEXA\_KEYBOARD # địa chỉ đọc scan code  li s0, 0x1 # row index ban đầu (0x1)  polling:  # Gửi row index để quét  sb s0, 0(t1) # gửi row index tới IN\_ADDRESS  # Đọc scan code  lb a0, 0(t2) # đọc scan code từ OUT\_ADDRESS  # In scan code nếu có phím được nhận (khác 0)  beqz a0, check\_next\_row # nếu không có phím nhận, chuyển row tiếp  li a7, 34 # in số nguyên ở dạng hex  ecall  # In xuống dòng  li a7, 11 # in ký tự  li a0, 10 # ký tự xuống dòng '\n'  ecall  check\_next\_row:  # Tạm dừng  li a7, 32 # ngủ (sleep)  li a0, 100 # 100ms  ecall  # Chuyển sang row tiếp theo  slli s0, s0, 1 # dịch trái 1 bit để lấy row tiếp (0x1 -> 0x2 -> 0x4 -> 0x8)  li t3, 0x10 # giới hạn row (sau 0x8)  bne s0, t3, polling # nếu chưa quét hết các row thì tiếp tục polling  # Reset lại row đầu tiên  li s0, 0x1  j polling # quay lại polling |

Chương trình sử dụng phương pháp **polling** để kiểm tra các phím được nhấn trên bàn phím ma trận 4x4. Ý tưởng chính:

1. **Quét từng hàng (row):** Gửi giá trị row index đến bàn phím để kích hoạt hàng tương ứng.
2. **Kiểm tra phím nhấn:** Đọc giá trị scan code từ bàn phím. Nếu giá trị khác 0, tức là có phím được nhấn.
3. **Xử lý kết quả:** In giá trị scan code của phím được nhấn lên console.
4. **Chuyển hàng:** Sau khi kiểm tra xong một hàng, chuyển sang hàng tiếp theo. Khi đã quét hết tất cả các hàng, quay lại hàng đầu tiên và lặp lại quá trình.

Output:

A screenshot of a computer

Description automatically generated

Assignment 2

|  |
| --- |
| .eqv IN\_ADDRESS\_HEXA\_KEYBOARD 0xFFFF0012  .data  message: .asciz "Someone's pressed a button.\n"  # --------------------------------------------------  # MAIN Procedure  # --------------------------------------------------  .text  main:  # Nạp địa chỉ của interrupt handler vào thanh ghi utvec  la t0, handler  csrrs zero, utvec, t0 # utvec = địa chỉ handler  # Bật bit UEIE (User External Interrupt Enable) trong thanh ghi UIE  li t1, 0x100 # bit 8 = 1  csrrs zero, uie, t1 # set bit 8 của uie  # Bật bit UIE (User Interrupt Enable) trong thanh ghi USTATUS  csrrsi zero, ustatus, 0x1 # bật bit 0 của ustatus  # Bật interrupt cho keypad trong Digital Lab Sim  li t1, IN\_ADDRESS\_HEXA\_KEYBOARD  li t3, 0x80 # bit 7 = 1 để enable interrupt  sb t3, 0(t1)  # Vòng lặp vô hạn để demo hiệu ứng của interrupt  loop:  nop # không thực hiện gì  # Delay 10ms  li a7, 32 # syscall sleep  li a0, 10 # 10ms  ecall  nop  j loop  end\_main:  # --------------------------------------------------  # Interrupt Service Routine (ISR)  # --------------------------------------------------  handler:  # Lưu context  addi sp, sp, -8 # dành 2 word trên stack  sw a0, 0(sp) # lưu a0  sw a7, 4(sp) # lưu a7  # Xử lý interrupt  # In thông báo trong Run I/O  li a7, 4 # syscall in chuỗi  la a0, message  ecall  # Khôi phục context  lw a7, 4(sp) # lấy lại a7  lw a0, 0(sp) # lấy lại a0  addi sp, sp, 8 # giải phóng stack  # Trở về chương trình chính  uret # trở về từ interrupt |

**1. Khởi tạo hệ thống interrupt**

* **Nạp địa chỉ interrupt handler (handler) vào thanh ghi utvec:**
  + Đây là địa chỉ của routine được thực thi khi xảy ra interrupt.
* **Bật interrupt toàn cục và external interrupt:**
  + Bật bit UEIE (bit 8) trong thanh ghi uie để cho phép interrupt từ thiết bị ngoại vi.
  + Bật bit UIE (bit 0) trong thanh ghi ustatus để kích hoạt interrupt toàn cục.
* **Kích hoạt interrupt cho bàn phím:**
  + Gửi giá trị 0x80 đến địa chỉ IN\_ADDRESS\_HEXA\_KEYBOARD để bật interrupt của bàn phím trong mô phỏng Digital Lab Sim.

**2. Vòng lặp chính (Main loop)**

* Chương trình chạy vòng lặp vô hạn (loop) với nhiệm vụ "nhàn rỗi" (không làm gì cụ thể).
* **Delay 10ms:**
  + Dùng syscall để tạm dừng chương trình trong 10ms, nhằm mô phỏng thời gian chờ giữa các sự kiện.
* **Hiệu ứng của interrupt:**
  + Trong lúc chương trình đang chạy vòng lặp, nếu có phím được nhấn, interrupt sẽ xảy ra, tạm dừng vòng lặp và nhảy đến handler.

**3. Interrupt Service Routine (ISR)**

* **Lưu trạng thái (context saving):**
  + Lưu giá trị của các thanh ghi a0 và a7 vào stack để bảo toàn dữ liệu khi xử lý interrupt.
* **Xử lý interrupt:**
  + In thông báo "Someone's pressed a button.\n" ra màn hình thông qua syscall.
* **Khôi phục trạng thái (context restoring):**
  + Lấy lại giá trị a0 và a7 từ stack và giải phóng stack.
* **Trở về chương trình chính:**
  + Sử dụng lệnh uret để trở về vòng lặp chính tại vị trí bị tạm dừng.

**Tóm tắt luồng xử lý**

1. Chương trình chính khởi tạo interrupt và chạy vòng lặp chờ.
2. Khi có phím nhấn:
   * ISR được kích hoạt và tạm dừng vòng lặp chính.
   * Thông báo được in ra màn hình.
3. Sau khi xử lý xong, ISR trả quyền điều khiển về chương trình chính và tiếp tục vòng lặp.

A screenshot of a computer

Description automatically generated

Assignment 3

|  |
| --- |
| .eqv IN\_ADDRESS\_HEXA\_KEYBOARD 0xFFFF0012  .eqv OUT\_ADDRESS\_HEXA\_KEYBOARD 0xFFFF0014  .data  message: .asciz "Key scan code: "  # -----------------------------------------------------------------  # MAIN Procedure  # -----------------------------------------------------------------  .text  main:  # Tải địa chỉ của routine phục vụ ngắt vào thanh ghi UTVEC  la t0, handler  csrrw zero, utvec, t0  # Thiết lập bit UEIE (User External Interrupt Enable) trong thanh ghi UIE  li t1, 0x100  csrrs zero, uie, t1 # uie - bit ueie (bit 8)  # Thiết lập bit UIE (User Interrupt Enable) trong thanh ghi USTATUS  csrrsi zero, ustatus, 0x1 # ustatus - kích hoạt uie (bit 0)  # Kích hoạt ngắt của bàn phím số trong Digital Lab Sim  li t1, IN\_ADDRESS\_HEXA\_KEYBOARD  li t3, 128 # 0x80 trong hệ thập phân; bit 7 = 1 để kích hoạt ngắt  sb t3, 0(t1)  # ---------------------------------------------------------  # Vòng lặp in dãy số tuần tự  # ---------------------------------------------------------  xor s0, s0, s0 # count = s0 = 0  loop:  addi s0, s0, 1 # count = count + 1  prn\_seq:  addi a7, zero, 1  add a0, s0, zero # In số tuần tự tự động  ecall  addi a7, zero, 11  li a0, '\n' # In ký tự xuống dòng  ecall  sleep:  addi a7, zero, 32  li a0, 300 # Tạm dừng 300 ms  ecall  j loop  end\_main:  # -----------------------------------------------------------------  # Routine phục vụ ngắt  # -----------------------------------------------------------------  handler:  # Lưu ngữ cảnh  addi sp, sp, -24 # Điều chỉnh con trỏ stack để lưu nhiều thanh ghi hơn  sw a0, 0(sp)  sw a1, 4(sp)  sw a7, 8(sp)  sw t0, 12(sp)  sw t1, 16(sp)  sw t2, 20(sp)  # Khởi tạo t0 về 0 (chỉ số hàng)  li t0, 0  check\_rows:  # Kích hoạt lại ngắt và chọn hàng t0  li t1, IN\_ADDRESS\_HEXA\_KEYBOARD  li t2, 128 # 0x80 trong hệ thập phân; kích hoạt lại ngắt (bit 7)  or t2, t2, t0 # Kết hợp với chỉ số hàng để chọn hàng  sb t2, 0(t1) # Ghi vào thanh ghi điều khiển  # Đọc mã phím vào t2  li t1, OUT\_ADDRESS\_HEXA\_KEYBOARD  lb t2, 0(t1) # Đọc mã phím vào t2  # Kiểm tra nếu mã phím khác 0  beq t2, zero, next\_row  # In thông báo  addi a7, zero, 4  la a0, message  ecall  # Chuyển mã phím từ t2 sang a0 trước khi in  add a0, t2, zero  # In mã phím  li a7, 34 # ecall để in số nguyên dưới dạng hex  ecall  # In ký tự xuống dòng  li a7, 11  li a0, '\n'  ecall  # Thoát vòng lặp vì đã tìm thấy phím  j end\_handler  next\_row:  addi t0, t0, 1 # Tăng chỉ số hàng  li t1, 16 # Số lượng hàng (0 đến 15 cho 4 bit)  blt t0, t1, check\_rows  end\_handler:  # Phục hồi ngữ cảnh  lw t2, 20(sp)  lw t1, 16(sp)  lw t0, 12(sp)  lw a7, 8(sp)  lw a1, 4(sp)  lw a0, 0(sp)  addi sp, sp, 24  # Quay lại từ ngắt  uret |

**1. Khởi tạo chương trình**

**Thiết lập ngắt:**

* **Đặt địa chỉ handler vào utvec:**
  + Đây là routine xử lý ngắt sẽ được CPU gọi khi có ngắt.
* **Kích hoạt interrupt toàn cục (UIE) và external interrupt (UEIE):**
  + Kích hoạt toàn bộ hệ thống ngắt và cho phép ngắt từ thiết bị ngoại vi.
* **Bật interrupt cho bàn phím ma trận:**
  + Gửi giá trị 0x80 (bit 7) đến địa chỉ IN\_ADDRESS\_HEXA\_KEYBOARD để bật tính năng ngắt của bàn phím.

**2. Vòng lặp chính**

**In dãy số tuần tự:**

* **Tăng bộ đếm (s0) và in giá trị:**
  + Mỗi lần lặp, tăng giá trị của s0 (đếm số tuần tự) và in giá trị đó ra màn hình.
* **Tạm dừng 300ms:**
  + Thêm delay để giảm tốc độ in và mô phỏng khoảng thời gian chờ.

**Chức năng ngắt trong vòng lặp:**

* Khi phím được nhấn, chương trình chính tạm dừng để thực hiện xử lý ngắt. Sau khi xử lý xong, vòng lặp chính tiếp tục.

**3. Routine phục vụ ngắt (handler)**

**Lưu ngữ cảnh (context saving):**

* Lưu các thanh ghi quan trọng (a0, a1, a7, t0, t1, t2) vào stack để đảm bảo giá trị không bị thay đổi trong quá trình xử lý ngắt.

**Xử lý ngắt bàn phím:**

1. **Quét từng hàng của bàn phím:**
   * **Gửi giá trị hàng hiện tại (t0) đến IN\_ADDRESS\_HEXA\_KEYBOARD:**
     + Kích hoạt hàng tương ứng và tiếp tục ngắt (bit 7).
   * **Đọc mã phím (t2):**
     + Lấy giá trị mã phím từ OUT\_ADDRESS\_HEXA\_KEYBOARD.
2. **Kiểm tra mã phím:**
   * Nếu mã phím khác 0, nghĩa là có phím được nhấn.
   * In thông báo "Key scan code: " và mã phím (ở dạng hex) ra màn hình.
3. **Chuyển hàng tiếp theo:**
   * Nếu mã phím là 0, tăng chỉ số hàng (t0) để kiểm tra hàng tiếp theo.
   * Khi đã quét hết tất cả các hàng (16 hàng), thoát khỏi routine.

**Khôi phục ngữ cảnh (context restoring):**

* Lấy lại giá trị các thanh ghi đã lưu từ stack và khôi phục trạng thái ban đầu.

**Quay lại chương trình chính:**

* Sử dụng lệnh uret để quay lại vòng lặp chính tại điểm bị tạm dừng.

**4. Tổng kết flow**

1. **Khởi tạo hệ thống ngắt:**
   * Thiết lập địa chỉ handler và bật interrupt cho bàn phím.
2. **Chương trình chính chạy vòng lặp vô hạn:**
   * In số tuần tự và chờ ngắt.
3. **Khi có ngắt:**
   * Handler xử lý sự kiện ngắt bằng cách quét hàng, đọc mã phím và in ra mã phím nếu có phím nhấn.
4. **Quay lại chương trình chính:**
   * Sau khi xử lý xong, handler trả quyền điều khiển lại cho vòng lặp chính.

Output:

|  |
| --- |
| 1  2  3  4  5  6  7  8  9  10  Key scan code: 0x00000011  11  12  13  14  15  16  Key scan code: 0x00000041  17  18  19  20  21  Key scan code: 0x00000042  22  23  24  25  26  Key scan code: 0x00000028  27  28  29  30  31  32  33  34  35  36  37  Key scan code: 0x00000042  38  39  40 |

Assignment 4

|  |
| --- |
| .eqv IN\_ADDRESS\_HEXA\_KEYBOARD 0xFFFF0012  .eqv TIMER\_NOW 0xFFFF0018  .eqv TIMER\_CMP 0xFFFF0020  .eqv MASK\_CAUSE\_TIMER 4  .eqv MASK\_CAUSE\_KEYPAD 8    .data  msg\_keypad: .asciz "Someone has pressed a key!\n"  msg\_timer: .asciz "Time inteval!\n"    # -----------------------------------------------------------------  # MAIN Procedure  # -----------------------------------------------------------------  .text  main:  la t0, handler  csrrs zero, utvec, t0    li t1, 0x100  csrrs zero, uie, t1 # uie - ueie bit (bit 8) - external interrupt  csrrsi zero, uie, 0x10 # uie - utie bit (bit 4) - timer interrupt    csrrsi zero, ustatus, 1 # ustatus - enable uie - global interrupt    # ---------------------------------------------------------  # Enable interrupts you expect  # ---------------------------------------------------------  # Enable the interrupt of keypad of Digital Lab Sim  li t1, IN\_ADDRESS\_HEXA\_KEYBOARD  li t2, 0x80 # bit 7 of = 1 to enable interrupt  sb t2, 0(t1)    # Enable the timer interrupt  li t1, TIMER\_CMP  li t2, 1000  sw t2, 0(t1)    # ---------------------------------------------------------  # No-end loop, main program, to demo the effective of interrupt  # ---------------------------------------------------------  loop:  nop  li a7, 32  li a0, 10  ecall  nop  j loop  end\_main:    # -----------------------------------------------------------------  # Interrupt service routine  # -----------------------------------------------------------------  handler:  # Saves the context  addi sp, sp, -16  sw a0, 0(sp)  sw a1, 4(sp)  sw a2, 8(sp)  sw a7, 12(sp)    # Handles the interrupt  csrr a1, ucause  li a2, 0x7FFFFFFF  and a1, a1, a2 # Clear interrupt bit to get the value    li a2, MASK\_CAUSE\_TIMER  beq a1, a2, timer\_isr  li a2, MASK\_CAUSE\_KEYPAD  beq a1, a2, keypad\_isr  j end\_process    timer\_isr:  li a7, 4  la a0, msg\_timer  ecall    # Set cmp to time + 1000  li a0, TIMER\_NOW  lw a1, 0(a0)  addi a1, a1, 1000  li a0, TIMER\_CMP  sw a1, 0(a0)    j end\_process    keypad\_isr:  li a7, 4  la a0, msg\_keypad  ecall  j end\_process    end\_process:    # Restores the context  lw a7, 12(sp)  lw a2, 8(sp)  lw a1, 4(sp)  lw a0, 0(sp)  addi sp, sp, 16  uret |

1. **Định nghĩa hằng số**:
   * Các giá trị được định nghĩa là địa chỉ và mặt nạ dùng để giao tiếp với các thành phần ngoại vi như bàn phím và bộ đếm thời gian. Điều này giúp chương trình dễ đọc hơn thay vì sử dụng trực tiếp các giá trị thập lục phân.
2. **Khai báo dữ liệu**:
   * Chuỗi thông báo "Someone has pressed a key!" được lưu trữ để in ra khi có sự kiện ngắt từ bàn phím.
   * Chuỗi "Time interval!" được lưu trữ để in ra khi ngắt từ bộ đếm thời gian xảy ra.
3. **Thiết lập trình xử lý ngắt**:
   * Tải địa chỉ của trình xử lý ngắt vào thanh ghi điều khiển liên quan, để khi xảy ra ngắt, CPU sẽ tự động nhảy đến trình xử lý này.
4. **Bật ngắt toàn cục và cụ thể**:
   * Ngắt toàn cục được kích hoạt để CPU có thể xử lý mọi loại ngắt được cho phép.
   * Ngắt từ bàn phím được bật bằng cách ghi giá trị vào thanh ghi tương ứng, cho phép nhận tín hiệu từ bàn phím.
   * Ngắt từ bộ đếm thời gian được bật bằng cách đặt giá trị thời gian so sánh vào thanh ghi của bộ đếm thời gian. Điều này tạo ra một ngắt định kỳ sau một khoảng thời gian xác định.
5. **Vòng lặp chính của chương trình**:
   * CPU sẽ chờ đợi trong vòng lặp vô tận, thực hiện một số lệnh "rỗng" (NOP) hoặc gọi hệ thống.
   * Vòng lặp này không thực hiện công việc chính mà chỉ tồn tại để giữ chương trình hoạt động. Công việc thực sự được xử lý bởi các trình xử lý ngắt.
6. **Xử lý ngắt**:
   * Khi có tín hiệu từ bàn phím hoặc từ bộ đếm thời gian, CPU sẽ tạm dừng vòng lặp và chuyển đến trình xử lý ngắt. Tại đây, thông báo tương ứng sẽ được in ra hoặc xử lý thêm.
7. **Tính năng trình diễn**:
   * Chương trình minh họa cách xử lý ngắt hoạt động trong hệ thống, ví dụ như phản hồi với một sự kiện bên ngoài (như nhấn phím) hoặc ngắt định kỳ từ bộ đếm thời gian.

**Output:**

TIMER\_CMP=1000ms:

A screenshot of a computer program

Description automatically generated

TIMER\_CMP=1500ms:

A screenshot of a program

Description automatically generated

Presskey:

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

Presskey and TIMER\_CMP=1000ms:

A screenshot of a computer program

Description automatically generated

Assignment 5

|  |
| --- |
| .data  message: .asciz "Exception occurred.\n"  .text  main:  try:  la t0, catch  csrrw zero, utvec, t0 # Set utvec (5) to the handlers address  csrrsi zero, ustatus, 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  # Since uepc contains address of the error instruction  # Need to load finally address to uepc  la t0, finally  csrrw zero, uepc, t0  uret |

**1. Khai báo và thiết lập dữ liệu:**

* .data và message:
  + Khai báo một chuỗi ký tự "Exception occurred.\n" trong phần .data của chương trình. Chuỗi này sẽ được sử dụng để hiển thị thông báo khi xảy ra ngoại lệ.

**2. Phần .text - Mã lệnh chính của chương trình:**

* main::
  + Đây là nhãn (label) bắt đầu của chương trình chính.

**3. Cấu hình trình xử lý ngoại lệ (trap handler):**

* la t0, catch:
  + Tải địa chỉ của trình xử lý ngoại lệ (trap handler), được gắn nhãn là catch, vào thanh ghi tạm thời t0.
* csrrw zero, utvec, t0:
  + Cấu hình thanh ghi utvec để chỉ định địa chỉ của trình xử lý ngoại lệ (trap handler). Điều này đảm bảo rằng khi xảy ra ngoại lệ, chương trình sẽ nhảy đến catch.

**4. Kích hoạt ngắt:**

* csrrsi zero, ustatus, 1:
  + Bật cờ "interrupt enable" trong thanh ghi ustatus. Đây là bước quan trọng để đảm bảo các ngoại lệ hoặc ngắt được xử lý.

**5. Thực hiện lỗi có chủ ý:**

* lw zero, 0:
  + Đây là lệnh tạo ra lỗi truy cập bộ nhớ (Load Access Fault). Cụ thể, chương trình cố gắng đọc một địa chỉ bộ nhớ không hợp lệ, dẫn đến kích hoạt ngoại lệ.

**6. Xử lý ngoại lệ tại trình xử lý catch:**

* Khi lỗi xảy ra, chương trình sẽ nhảy đến nhãn catch do đã thiết lập utvec trước đó.
* Trong catch:
  + Hiển thị thông báo:
    - Gọi syscall để in chuỗi "Exception occurred.\n" ra màn hình, thông qua việc nạp giá trị và địa chỉ chuỗi vào các thanh ghi.
  + Sửa lỗi và tiếp tục:
    - Lấy địa chỉ nhãn finally (phần mã kết thúc chương trình) và nạp vào thanh ghi uepc. Điều này cho phép chương trình tiếp tục thực thi từ phần mã finally.

**7. Phần mã cuối cùng (finally):**

* li a7, 10 và ecall:
  + Thực hiện syscall để thoát khỏi chương trình (mã lệnh exit).

**8. Tổng kết:**

* Đoạn mã này minh họa cơ chế xử lý ngoại lệ (trap handling) trong hệ thống RISC-V. Nó:
  1. Thiết lập trình xử lý ngoại lệ.
  2. Cố ý tạo ngoại lệ (Load Access Fault).
  3. Xử lý ngoại lệ, hiển thị thông báo lỗi, và đưa chương trình về trạng thái bình thường.
  4. Kết thúc chương trình an toàn thông qua syscall.

Output:

A screenshot of a computer program

Description automatically generated

Assignment 6

|  |
| --- |
| .data  overflow\_msg: .asciz "Overflow occurred during addition!\n"  .text  main:  # Set up software interrupt handling  # Load the interrupt service routine address to the UTVEC register  la t0, overflow\_handler  csrrs zero, utvec, t0  # Enable software interrupt  li t1, 0x2 # USIE bit (bit 1 in uie register)  csrrs zero, uie, t1  # Enable global interrupts  csrrsi zero, ustatus, 1  # Two numbers to add (choose values that will cause overflow)  li a0, 0x7FFFFFFF # Maximum positive 32-bit signed integer  li a1, 0 # Adding 1 will cause overflow  # Perform addition with overflow check  add t0, a0, a1 # This will cause overflow  bltz t0, trigger\_interrupt # Branch if result is negative (overflow occurred)  j end\_program    end\_program:  # If no interrupt triggered, exit program  li a7, 10  ecall    trigger\_interrupt:  # Trigger software interrupt by setting USIP bit in uip register  li t1, 0x2 # USIP bit (bit 1 in uip register)  csrrs zero, uip, t1  # Interrupt Service Routine for overflow  overflow\_handler:  # Save context  addi sp, sp, -8  sw a0, 0(sp)  sw a7, 4(sp)  # Print overflow message  li a7, 4  la a0, overflow\_msg  ecall  # Restore context  lw a7, 4(sp)  lw a0, 0(sp)  addi sp, sp, 8  # Clear the software interrupt bit  li t1, 0x2  csrrc zero, uip, t1  # Terminate the program  li a7, 10  ecall |

**1. Đăng Ký Địa Chỉ Trình Xử Lý Ngắt**

- Sử dụng lệnh `csrrs` để thiết lập địa chỉ của trình xử lý ngắt (overflow\_handler) vào thanh ghi UTVEC

- UTVEC là thanh ghi chứa địa chỉ của vector ngắt cho các ngắt ở chế độ người dùng

**2. Kích Hoạt Ngắt Phần Mềm**

- Bật bit USIE (User Software Interrupt Enable) trong thanh ghi UIE

- Điều này cho phép hệ thống chấp nhận các ngắt phần mềm

**3. Kích Hoạt Ngắt Toàn Cục**

- Sử dụng `csrrsi` để bật các ngắt toàn cục ở chế độ người dùng

Logic Phát Hiện Tràn Số

Phép Cộng Gây Tràn

- Nạp giá trị 0x7FFFFFFF (số nguyên dương lớn nhất 32-bit) vào thanh ghi a0

- Cộng với 0, điều này sẽ gây ra tình trạng tràn số

Kiểm Tra Tràn Số

- Sử dụng lệnh `bltz` để kiểm tra kết quả phép cộng

- Nếu kết quả âm, nghĩa là đã xảy ra tràn số

**Trình Xử Lý Ngắt (Interrupt Service Routine)**

Bước Chuẩn Bị

- Giảm con trỏ stack (sp) để lưu trữ các thanh ghi

- Lưu các thanh ghi a0 và a7 để bảo toàn ngữ cảnh

Xử Lý Ngắt

- In ra thông báo "Overflow occurred during addition!"

- Sử dụng lệnh hệ thống ecall để in thông báo

Dọn Dẹp

- Khôi phục lại các thanh ghi

- Xóa bit ngắt phần mềm trong thanh ghi UIP

- Kết thúc chương trình bằng lệnh ecall

**Ý Nghĩa Kỹ Thuật**

- Chương trình minh họa kỹ thuật xử lý ngoại lệ tràn số một cách an toàn

- Sử dụng cơ chế ngắt để thông báo và xử lý lỗi mà không làm sập chương trình

- Thể hiện cách quản lý ngữ cảnh (context) và điều khiển luồng thực thi khi xảy ra sự kiện bất thường

Output:

a0= 0x7FFFFFFF, a1 =0 (không tràn số):

A screenshot of a computer program

Description automatically generated

a0= 0x7FFFFFFF, a1 = 1 (có tràn số):

A screenshot of a computer program

Description automatically generated