Nguyễn Đình Dương – 20225966

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**

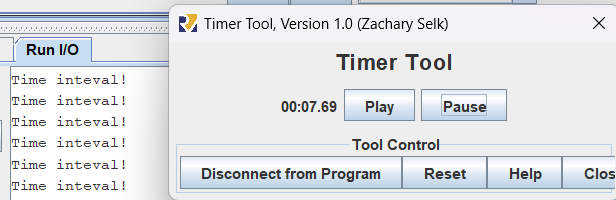
1. **Khởi tạo hệ thống ngắt:**
   * Thiết lập địa chỉ handler và bật interrupt cho bàn phím.
2. **Chương trình chính chạy vòng lặp vô hạn:**
   * In số tuần tự và chờ ngắt.
3. **Khi có ngắt:**
   * Handler xử lý sự kiện ngắt bằng cách quét hàng, đọc mã phím và in ra mã phím nếu có phím nhấn.
4. **Quay 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 "Ai đó đã nhấn phím!\n"  msg\_timer: .asciz "Khoảng thời gian!\n"  # -----------------------------------------------------------------  # Thủ tục chính (MAIN Procedure)  # -----------------------------------------------------------------  .text  main:  la t0, handler # Tải địa chỉ của handler (routine xử lý ngắt) vào t0  csrrs zero, utvec, t0 # Thiết lập địa chỉ của trình xử lý ngắt vào thanh ghi `utvec`    li t1, 0x100 # Tải giá trị để bật bit ngắt bên ngoài (ueie)  csrrs zero, uie, t1 # uie - bật bit ngắt ngoài (bit 8)  csrrsi zero, uie, 0x10 # uie - bật bit ngắt timer (bit 4)    csrrsi zero, ustatus, 1 # ustatus - bật uie (ngắt toàn cục)  # ---------------------------------------------------------  # Bật các ngắt mà bạn mong đợi  # ---------------------------------------------------------  # Bật ngắt bàn phím số của Digital Lab Sim  li t1, IN\_ADDRESS\_HEXA\_KEYBOARD  li t2, 0x80 # Bật bit 7 = 1 để cho phép ngắt  sb t2, 0(t1)    # Bật ngắt timer  li t1, TIMER\_CMP  li t2, 1000 # Thiết lập thời gian so sánh cho ngắt  sw t2, 0(t1)    # ---------------------------------------------------------  # Vòng lặp vô tận, chương trình chính, để demo hiệu quả của ngắt  # ---------------------------------------------------------  loop:  nop # Không làm gì (no operation)  li a7, 32 # Lệnh hệ thống  li a0, 10 # Hàm thoát chương trình  ecall # Gọi lệnh hệ thống  nop # Không làm gì  j loop # Quay lại vòng lặp  end\_main:  # -----------------------------------------------------------------  # Routine phục vụ ngắt (Interrupt Service Routine)  # -----------------------------------------------------------------  handler:  # Lưu trạng thái hiện tại  addi sp, sp, -16  sw a0, 0(sp)  sw a1, 4(sp)  sw a2, 8(sp)  sw a7, 12(sp)    # Xử lý ngắt  csrr a1, ucause # Đọc mã nguyên nhân ngắt  li a2, 0x7FFFFFFF  and a1, a1, a2 # Xóa bit ngắt để lấy giá trị thực của nguyên nhân    li a2, MASK\_CAUSE\_TIMER  beq a1, a2, timer\_isr # Nếu là ngắt timer, nhảy đến xử lý timer  li a2, MASK\_CAUSE\_KEYPAD  beq a1, a2, keypad\_isr # Nếu là ngắt bàn phím, nhảy đến xử lý bàn phím  j end\_process # Nếu không phải, kết thúc xử lý    timer\_isr:  li a7, 4 # Lệnh in ra màn hình  la a0, msg\_timer # Nạp địa chỉ chuỗi thông báo timer  ecall # In chuỗi ra màn hình    # Thiết lập TIMER\_CMP = TIMER\_NOW + 1000  li a0, TIMER\_NOW  lw a1, 0(a0)  addi a1, a1, 1000  li a0, TIMER\_CMP  sw a1, 0(a0)    j end\_process # Kết thúc xử lý ngắt timer  keypad\_isr:  li a7, 4 # Lệnh in ra màn hình  la a0, msg\_keypad # Nạp địa chỉ chuỗi thông báo bàn phím  ecall # In chuỗi ra màn hình  j end\_process # Kết thúc xử lý ngắt bàn phím    end\_process:  # Khôi phục trạng thái trước khi ngắt  lw a7, 12(sp)  lw a2, 8(sp)  lw a1, 4(sp)  lw a0, 0(sp)  addi sp, sp, 16  uret # Quay lại chương trình chính sau khi xử lý ngắt |

****

**1 Khởi tạo chương trình chính (main):**

* Thiết lập utvec để xử lý ngắt.
* Bật các ngắt toàn cục (ustatus) và từng loại ngắt cụ thể (timer và bàn phím số).
* Cấu hình ngắt timer với thời gian so sánh cụ thể.
* Chạy một vòng lặp vô tận, thực hiện một lệnh hệ thống (ecall) để duy trì chương trình chạy.

**2 Khi xảy ra ngắt (handler):**

* Lưu trạng thái thanh ghi (context) của chương trình hiện tại.
* Xác định nguyên nhân ngắt từ ucause.
* Chuyển đến routine xử lý ngắt tương ứng (timer\_isr hoặc keypad\_isr).

**3 Xử lý ngắt timer (timer\_isr):**

* Hiển thị thông báo "Khoảng thời gian!" trên màn hình.
* Thiết lập giá trị mới cho TIMER\_CMP để tiếp tục chu kỳ timer.

**4 Xử lý ngắt bàn phím (keypad\_isr):**

* Hiển thị thông báo "Ai đó đã nhấn phím!" trên màn hình.

**5 Kết thúc xử lý ngắt (end\_process):**

* Khôi phục trạng thái của thanh ghi trước khi xảy ra ngắt.
* Quay lại chương trình chính với lệnh uret.

Assignment 5

|  |
| --- |
| .data  message: .asciz "Xảy ra ngoại lệ.\n" # Thông báo ngoại lệ  .text  main:  try:  la t0, catch # Tải địa chỉ của trình xử lý lỗi (catch) vào t0  csrrw zero, utvec, t0 # Đặt utvec (5) trỏ tới địa chỉ của trình xử lý lỗi  csrrsi zero, ustatus, 1 # Bật bit cho phép ngắt trong ustatus (bit 0)  lw zero, 0 # Kích hoạt bẫy (trap) cho lỗi truy cập bộ nhớ (Load access fault)  finally:  li a7, 10 # Mã thoát chương trình  ecall # Gọi lệnh hệ thống để thoát  catch:  # Hiển thị thông báo  li a7, 4 # Mã lệnh in ra màn hình  la a0, message # Tải địa chỉ của chuỗi thông báo vào a0  ecall # Gọi lệnh in ra màn hình    # Vì uepc chứa địa chỉ của lệnh gây lỗi,  # cần thay thế địa chỉ này bằng địa chỉ của `finally`  la t0, finally # Tải địa chỉ của khối finally vào t0  csrrw zero, uepc, t0 # Đặt uepc trỏ tới địa chỉ finally  uret # Quay lại tiếp tục thực thi từ finally |

1. **Khởi tạo chương trình chính (main):**
   * Đặt trình xử lý lỗi (catch) bằng cách thiết lập thanh ghi utvec.
   * Bật bit cho phép ngắt trong thanh ghi ustatus để hệ thống có thể xử lý ngoại lệ.
   * Thực hiện lệnh gây lỗi (lw zero, 0) để cố ý kích hoạt một ngoại lệ truy cập bộ nhớ (Load access fault).
2. **Khi xảy ra ngoại lệ:**
   * Lệnh gây lỗi nhảy tới trình xử lý ngoại lệ (catch), được chỉ định trong utvec.
   * Trình xử lý ngoại lệ hiển thị thông báo "Xảy ra ngoại lệ." trên màn hình.
   * Sau đó, trình xử lý thay thế địa chỉ trong thanh ghi uepc (chứa địa chỉ của lệnh gây lỗi) bằng địa chỉ của khối finally, để tiếp tục thực thi chương trình từ đó.
3. **Hoàn thành chương trình (finally):**
   * Lệnh trong khối finally thoát khỏi chương trình (li a7, 10; ecall).

**Mô tả Luồng Điều Khiển**

1. **Bình thường:**
   * Chương trình bắt đầu tại main.
   * Khi gặp lệnh gây lỗi (lw zero, 0), chương trình chuyển sang trình xử lý lỗi (catch).
2. **Xử lý lỗi:**
   * Trình xử lý ngoại lệ in thông báo lỗi.
   * Sau đó, thay đổi thanh ghi uepc để bỏ qua lệnh gây lỗi, thay vào đó thực thi từ khối finally.
3. **Kết thúc:**
   * Chương trình tiếp tục từ khối finally, thực hiện lệnh thoát, và kết thúc.

A screen shot of a computer screen

Description automatically generated

Assignment 6

|  |
| --- |
| .data  overflow\_msg: .asciz "Tràn số xảy ra trong phép cộng!\n" # Thông báo khi tràn số  .text  main:  # Cài đặt trình xử lý ngắt phần mềm  # Tải địa chỉ của trình xử lý ngắt vào thanh ghi UTVEC  la t0, overflow\_handler  csrrs zero, utvec, t0  # Bật ngắt phần mềm  li t1, 0x2 # Bật bit USIE (bit 1 trong thanh ghi uie)  csrrs zero, uie, t1  # Bật ngắt toàn cục  csrrsi zero, ustatus, 1  # Hai số cần cộng (chọn giá trị để gây ra tràn số)  li a0, 0x7FFFFFFF # Giá trị dương lớn nhất của số nguyên 32-bit có dấu  li a1, 1 # Cộng 1 sẽ gây ra tràn số  # Thực hiện phép cộng và kiểm tra tràn số  add t0, a0, a1 # Phép cộng sẽ gây tràn số  bltz t0, trigger\_interrupt # Nhảy nếu kết quả là số âm (dấu hiệu tràn số)  j end\_program  end\_program:  # Nếu không xảy ra ngắt, thoát chương trình  li a7, 10  ecall  trigger\_interrupt:  # Kích hoạt ngắt phần mềm bằng cách bật bit USIP trong thanh ghi uip  li t1, 0x2 # Bật bit USIP (bit 1 trong thanh ghi uip)  csrrs zero, uip, t1  # Trình xử lý ngắt cho lỗi tràn số  overflow\_handler:  # Lưu ngữ cảnh  addi sp, sp, -8  sw a0, 0(sp)  sw a7, 4(sp)  # In thông báo tràn số  li a7, 4  la a0, overflow\_msg  ecall  # Khôi phục ngữ cảnh  lw a7, 4(sp)  lw a0, 0(sp)  addi sp, sp, 8  # Xóa bit ngắt phần mềm  li t1, 0x2  csrrc zero, uip, t1  # Kết thúc chương trình  li a7, 10  ecall |

Output:

|  |  |
| --- | --- |
|  |  |
|  |  |

1. **Thiết lập chương trình chính (main):**
   * **Đặt trình xử lý ngắt (Interrupt Service Routine - ISR):**
     + Tải địa chỉ của overflow\_handler vào thanh ghi utvec để xử lý ngắt.
   * **Bật ngắt phần mềm:**
     + Kích hoạt bit USIE trong thanh ghi uie để cho phép ngắt phần mềm.
   * **Bật ngắt toàn cục:**
     + Kích hoạt bit toàn cục trong thanh ghi ustatus để hệ thống cho phép xử lý ngắt.
2. **Thực hiện phép cộng với kiểm tra tràn số:**
   * Thực hiện phép cộng hai giá trị:
     + a0 = 0x7FFFFFFF (giá trị dương lớn nhất của số nguyên có dấu 32-bit).
     + a1 = 1.
   * Nếu kết quả là số âm (bltz t0, trigger\_interrupt), điều này báo hiệu tràn số, và chương trình sẽ nhảy đến trigger\_interrupt.
3. **Kích hoạt ngắt phần mềm:**
   * Thiết lập bit USIP trong thanh ghi uip để báo hiệu ngắt phần mềm.
4. **Xử lý ngắt trong overflow\_handler:**
   * **Lưu ngữ cảnh:**
     + Lưu giá trị của các thanh ghi a0 và a7 vào ngăn xếp.
   * **Hiển thị thông báo lỗi:**
     + In thông báo "Tràn số xảy ra trong phép cộng!" lên màn hình.
   * **Khôi phục ngữ cảnh:**
     + Khôi phục giá trị của các thanh ghi từ ngăn xếp.
   * **Xóa bit ngắt phần mềm:**
     + Xóa bit USIP trong thanh ghi uip để tắt trạng thái ngắt.
   * **Kết thúc chương trình:**
     + Thoát chương trình thông qua lệnh ecall.
5. **Nếu không có tràn số:**
   * Chương trình nhảy đến end\_program và thoát bình thường mà không kích hoạt ngắt.

**Luồng điều khiển tóm tắt:**

1. **Bắt đầu:** Thiết lập trình xử lý ngắt và bật ngắt phần mềm.
2. **Phép toán:** Cộng hai số, kiểm tra tràn số.
3. **Tràn số:** Nếu tràn, kích hoạt ngắt phần mềm.
4. **Xử lý ngắt:** Hiển thị thông báo lỗi, xóa trạng thái ngắt, và thoát chương trình.
5. **Không tràn:** Thoát chương trình một cách bình thường.