**BÁO CÁO THỨC HÀNH KIẾN TRÚC MÁY TÍNH TUẦN 7**

**Họ và tên: Vũ Đức Hoàng Anh**

**MSSV: 20235658**

1. **Assignment 1**

* **Nhập chương trình:**

|  |  |
| --- | --- |
| 1  2  **3**  4  **5**  6  **7**  8  9  10  11  12  **13**  14  15 | # Laboratory Exercise 7 Home Assignment 1  .text  main:  li a0, -2105 # load input parameter  jal abs # jump and link to abs procedure  li a7, 10 # terminate  ecall  end\_main:  abs:  sub s0, zero, a0 # put -a0 in s0; in case a0 < 0  blt a0, zero, done # if a0<0 then done  add s0, a0, zero # else put a0 in s0  done:  jr ra |

* **Các bước thực hiện của chương trình:**

+ Các câu lệnh trong thẻ main: để khai báo giá trị của số cần tính.

+ Câu lệnh jal abs để gọi chương trình con để tính abs.

+ Câu lệnh trong thẻ abs để thực hiện tính trị tuyệt đối của số: lấy số đối của số đó, sau đó so sánh với 0 nếu số ban đầu lớn hơn 0 thì nhảy đến done còn nếu nhỏ hơn 0 thì tiếp tực thực hiện gán giá trị của số đối cho số ban đầu.

* **Quan sát kết quả của các thanh ghi và chương trình:**

**+** Với giá trị của a0 được khởi tạo là a0 = -2105 ta quan sát các thanh ghi:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **No.** | **Vị trí** | **a0** | **s0** | **pc** | **ra** | **Chú thích** |
| 1 | (3) | 0x00000 | 0x00000 | 0x400000 | 0x00000 | Bắt đầu chương trình |
| 2 | (5) | 0xfffff7c7 | 0x00000 | 0x400004 | 0x00008 | Nhập số cần xử lí |
| 3 | (13) | 0xfffff7c7 | 0x000839 | 0x0040001c | 0x00008 | Lấy trị tuyệt đối |
| 4 | (7) | 0xfffff7c7 | 0x000839 | 0x00400010 | 0x00008 | Kết thúc chương trình |

* Kết quả của thanh ghi a0 = -45 và s0 = 45 nên chương trình hoạt động bình thường.

1. **Assignment 2**

* **Nhập chương trình:**

|  |  |
| --- | --- |
| 1  2  **3**  4  5  6  **7**  8  9  **10**  11  12  13  14  15  **16**  17  18  19  20  **21** | # Laboratory Exercise 7, Home Assignment 2  .text  main:  li a0, 4 # load test input  li a1, 7  li a2, 5  jal max # call max procedure  li a7, 10 # terminate  ecall  end\_main:  max:  add s0, a0, zero # copy a0 in s0; largest so far  sub t0, a1, s0 # compute a1 - s0  blt t0, zero, okay # if a1 - v0 < 0 then no change  add s0, a1, zero # else a1 is largest thus far  okay:  sub t0, a2, s0 # compute a2 - v0  blt t0, zero, done # if a2 - v0 <0 then no change  add s0, a2, zero # else a2 is largest overall  done:  jr ra # return to calling program |

* **Các bước thực hiện chương trình:**

+ Các câu lệnh sau thẻ main để khai báo các giá trị của chương trình: a0=4, a1 = 7, a2 = 5, sau đó gọi hàm con tìm max.

+ Các câu lệnh trong thẻ max để kiểm tra xem đâu là giá trị lớn nhất được lưu trong 3 thanh ghi a0, a1, a2: Bước đầu lưu giá trị của a0 vào s0 sau đó tính hiệu của a1- a0 = t0, kiểm tra t0 nếu nhỏ hơn 0 thì tiếp tục so sánh s0( đang có giá trị là s0) với a2 bằng cách tương tự, nếu lớn hơn 0 thì thay đổi s0 = a1 và tiếp tục thực hiện chương trình.

+ Sau các câu lệnh trong thẻ max quay lại và kết thúc chương trình.

* **Quan sát giá trị các thanh ghi của chương trình:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **No** | **Vị trí** | **a0** | **a1** | **a2** | **s0** | **ra** | **t0** | **pc** | **Chú thích** |
| 1 | (3) | 0x00000 | 0x00000 | 0x00000 | 0x00000 | 0x00000 | 0x00000 | 0x400000 | Bắt đầu chương trình |
| 2 | (7) | 0x00004 | 0x00007 | 0x00005 | 0x00000 | 0x400010 | 0x00000 | 0x400018 | Khởi tạo các giá trị |
| 3 | (16) | 0x00004 | 0x00007 | 0x00005 | 0x00007 | 0x400010 | 0x00003 | 0x400028 | So sánh a0 và a1 |
| 4 | (21) | 0x00004 | 0x00007 | 0x00005 | 0x00007 | 0x400010 | 0xfffffffe | 0x400010 | So sánh max(a0,a1) , a2 |
| 5 | (10) | 0x00004 | 0x00007 | 0x00005 | 0x00007 | 0x400010 | 0xfffffffe | 0x40002c | Kết thúc chương trình |

1. **Assigntment 3**

* **Nhập chương trình:**

|  |  |
| --- | --- |
| 1  2  **3**  4  5  **6**  7  8  9  **10**  11  12  13  14  15  16  **17**  18  19  20  **21** | # Laboratory Exercise 7, Home Assignment 3  .text  main:  li s0, 21  li s1, 5  push:  addi sp, sp, -8 # adjust the stack pointer  sw s0, 4(sp) # push s0 to stack  sw s1, 0(sp) # push s1 to stack  work:  nop  nop  nop  pop:  lw s0, 0(sp) # pop from stack to s0  sw zero, 0(sp) # remove from stack  addi sp, sp, 4 # adjust the stack pointer  lw s1, 0(sp) # pop from stack to s1  sw zero, 0(sp) # remove from stack  addi sp, sp, 4 # adjust the stack pointer  end: |

* **Các bước hoạt động của chương trình:**

+ Thẻ main để khởi tạo giá trị của s0, s1

+ Câu lệnh trong thẻ push để thực hiện đưa giá trị vào stack: Đầu tiên khởi tạo vùng nhớ stack cho s0, s1 thông qua sp (stack pointer). Sau đó lần lượt đẩy giá trị của s0, s1 vào vùng nhớ stack

+ Câu lệnh trong thẻ work để mô phỏng vùng hoạt động của chương trình

+ Câu lệnh trong thẻ pop để lấy giá trị ra khỏi stack: Đầu tiên lấy giá trị của s1 ra khỏi vùng nhớ rồi lưu vào s0, sau đó xóa giá trị đã lấy và điều chỉnh lại giá trị của sp. Tương tự, lấy giá trị của s0 cũ đã lưu trong stack và lưu vào s1, sau đó xóa giá trị đã lưu cho stack và trả lại giá trị của sp

* **Quan sát giá trị các thanh ghi trong chương trình:**

+ Với giá trị của s1 và s2 lần lượt là s1 = 21 và s2 = 5

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **No** | **Vị trí** | **s1** | **s2** | **sp** | **pc** | **Chú thích** |
| 1 | (3) | 0x00000 | 0x00000 | 0x7fffeffc | 0x400000 | Bắt đầu chương trình |
| 2 | (6) | 0x00015 | 0x00005 | 0x7fffeffc | 0x400008 | Khởi tạo giá trị ban đầu |
| 3 | (10) | 0x00015 | 0x00005 | 0x7fffeff4 | 0x400014 | Đưa lần lượt s1, s2 vào stack |
| 4 | (17) | 0x00005 | 0x00005 | 0x7fffeff8 | 0x40002c | Lấy giá trị của s2 ra khỏi stack |
| 5 | (21) | 0x00005 | 0x00015 | 0x7fffeffc | 0x0040003c | Kết thúc chương trình |

* **Quan sát kết quả được lưu ở của sổ DataSegment:**

+ Giá trị ban đầu khi mới khởi tạo giá trị:

|  |
| --- |
|  |
|  |

+ Giá trị sau từng bước pop ra ngoài:

|  |
| --- |
|  |
|  |

* Kết quả của chương trình hoạt động đúng với yêu cầu

1. **Assignment 4**

* **Nhập chương trình:**

|  |  |
| --- | --- |
| 1  2  3  4  **5**  6  7  8  9  10  11  12  13  14  15  16  17  **18**  19  20  21  22  **23**  24  25  **26**  **27**  28  29  30  31  32  **33**  34  **35**  36  37  **38**  39  40  41  42  43  44  45  **46**  47 | # Laboratory Exercise 7, Home Assignment 4  .data  message: .asciz "Ket qua tinh giai thua la: "  .text  main:  jal WARP  print:  add a1, s0, zero # a0 = result from N!  li a7, 4  la a0, message  ecall  li a7, 1  mv a0, s0  ecall  quit:  li a7, 10 # terminate  ecall  end\_main:  WARP:  addi sp, sp, -4 # adjust stack pointer  sw ra, 0(sp) # save return address    li a0, 3 # load test input N  jal FACT # call fact procedure    lw ra, 0(sp) # restore return address  sw zero, 0(sp)  addi sp, sp, 4 # return stack pointer  jr ra  wrap\_end:  FACT:  addi sp, sp, -8 # allocate space for ra, a0 in stack  sw ra, 4(sp) # save ra register  sw a0, 0(sp) # save a0 register    li t0, 2  bge a0, t0, recursive  li s0, 1 # return the result N!=1  j done  recursive:  addi a0, a0, -1 # adjust input argument  jal FACT # recursive call  lw s1, 0(sp) # load a0  mul s0, s0, s1  done:  lw ra, 4(sp) # restore ra register  lw a0, 0(sp) # restore a0 register  sw zero, 0(sp)  addi sp,sp,4 # restore stack pointer  sw zero, 0(sp)  addi sp,sp,4  jr ra # jump to caller  fact\_end: |

* **Các bước hoạt động của chương trình:**

+ Câu lệnh trong cụm data để khởi tạo giá trị cho chuỗi để in ra màn hỉnh

+ Câu lệnh trong thẻ main để gọi đến hàm tính lũa thừa

+ Câu lệnh trong thẻ print để sau khi chương trình đã được thực hiện thì in kết quả ra màn hình và kết thúc chương trình

+ Câu lệnh trong thẻ quit để kết thúc chương trình

+ Câu lệnh trong thẻ WARP chuẩn bị cho bước gọ hàm giai thừa: Trước tiên lưu giá trị của ra vào stack, sau đó khởi tạo giá trị của a0 = 3 là số giai thừa, tiếp tục gọi hàm giai thừa để thực hiện tính giai thừa, cuối cùng là lấy giá trị đã được lưu của ra trong stack để quay lại main tiếp tục chương trình.

+ Câu lệnh trong thẻ FACT để tính đệ quy: Cấp phát thêm 8 byte bộ nhớ trong stack, lưu địa chỉ của ra vào stack, sau đó lưu địa chỉ a0 vào stack, tiếp tục khởi tạo giá trị t0 = 2. Kiểm tra xem nếu n >= 2 thì tiếp tục thực hiện đệ quy, nếu n < 2 thì trả vè giá trị 1 và nhảy đến thẻ done

+ Câu lệnh trong thẻ reacursive để gọi đệ quy: Giảm giá trị của a0 sau mỗi lần đệ quy để gọi đệ quy của n – 1 sau đó lấy lại giá trị ban đầu, rồi tính tích của hàm đệ quy .

+ Câu lệnh trong thẻ done để hoàn thành chương trình và quay trở lại main: Khôi phục giá trị của con trỏ sau đó quay lại caller

* **Quan sát giá trị các thanh ghi trong chương trình:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **No** | **Vị trí** | **a0** | **s0** | **ra** | **sp** | **pc** | **Chú thích** |
| 1 | (5) | 0x00000 | 0x00000 | 0x00000 | 0x7fffeffc | 0x4000000 | Bắt đầu chương trình |
| 2 | (23) | 0x00003 | 0x00000 | 0x0040003c | 0x7fffeff8 | 0x00400048 | Chuẩn bị cho hàm giai thừa |
| 3 | (33) | 0x00003 | 0x00000 | 0x0040003c | 0x7fffeff0 | 0x00400064 | Kiểm tra điều kiện dừng |
| 4 | (38) | 0x00002 | 0x00000 | 0x0040006c | 0x7fffeff0 | 0x00400048 | Gọi đến 2 giai thừa |
| 5 | (33) | 0x00002 | 0x00000 | 0x0040006c | 0x7fffefe8 | 0x00400064 | Kiểm tra điều kiện dừng |
| 6 | (38) | 0x00001 | 0x00000 | 0x0040006c | 0x7fffefe8 | 0x00400048 | Gọi đến 1 giai thừa |
| 7 | (33) | 0x00001 | 0x00000 | 0x0040006c | 0x7fffefe0 | 0x0040005c | Kiểm tra điều kiện dừng |
| 8 | (35) | 0x00001 | 0x00001 | 0x0040006c | 0x7fffefe0 | 0x00400074 | Hoàn thành bước gọi hàm |
| 9 | (45) | 0x00001 | 0x00001 | 0x0040006c | 0x7fffefe8 | 0x0040006c | Quay lại hàm 1! |
| 10 | (45) | 0x00002 | 0x00002 | 0x0040006c | 0x7fffeff0 | 0x0040006c | Quay lại hàm 2! |
| 11 | (45) | 0x00003 | 0x00006 | 0x0040003c | 0x7fffeff8 | 0x0040003c | Quay lại hàm 3! |
| 12 | (26) | 0x00003 | 0x00006 | 0x00400004 | 0x7fffeffc | 0x00400004 | Quay lại hàm main |
| 13 | (18) | 0x00006 | 0x00006 | 0x00400004 | 0x7fffeffc | 0x0040002c | Kết thúc chương trình |

* Kết quả của chương trình bằng 3! = 6 đúng với yêu cầu bài toán
* **Kết quả in ở cửa sổ Run I/O**

|  |
| --- |
|  |

* **Dữ liệu được lưu ở vùng nhớ ngăn xếp**

|  |
| --- |
|  |

+ Dữ liệu được đưa dần vào stack theo thứ tự từ phải sang trái sau đó lấy dữ liệu ra theo thứ tự từ trái sang phải

1. **Assignment 5**

* **Nhập chương trình:**

|  |  |
| --- | --- |
| **0**  **1**  **2**  **3**  **5**  **4** | .data  msg1: .asciz "gia tri lon nhat la "  msg2: .asciz "gia tri nho nhat la "  msg3: .asciz " duoc luu tru trong a"  msg4: .asciz "\n"  .text  main:  # Nhap cac thanh ghi  li a0 2  li a1 1  li a2 0  li a3 5  li a4 4  li a5 7  li a6 9  li a7 8  # Dua cac thanh ghi vao stack  addi sp, sp, -4  sw a7, 0(sp)  addi sp, sp, -4  sw a6, 0(sp)  addi sp, sp, -4  sw a5, 0(sp)  addi sp, sp, -4  sw a4, 0(sp)  addi sp, sp, -4  sw a3, 0(sp)  addi sp, sp, -4  sw a2, 0(sp)  addi sp, sp, -4  sw a1, 0(sp)  addi sp, sp, -4  sw a0, 0(sp)  # Khởi tạo các biến lưu kết quả chương trình  add s0, a0, zero # Khởi tạo giá trị min  add s1, a0, zero # Khởi tạo giá trị max  li t0 0 # Khởi tạo biến đếm byte  li t2 0 # Khởi tạo biến dếm vị trí  li s4 32 # Khởi tạo số byte của sp  li s2 0 # Khởi tạo vị trí của số nhỏ nhất  li s3 0 # Khởi tạo vị trí của số lớn nhất  jal Max\_Min    print:  li a7, 4  la a0, msg1  ecall  li a7, 1  mv a0, s1  ecall  li a7, 4  la a0, msg3  ecall  li a7, 1  mv a0, s3  ecall  li a7, 4  la a0, msg4  ecall  li a7, 4  la a0, msg2  ecall  li a7, 1  mv a0, s0  ecall  li a7, 4  la a0, msg3  ecall  li a7, 1  mv a0, s2  ecall  end\_main:  li a7, 10  ecall  Max\_Min:  beq t0, s4, end\_max\_min  lw t1, 0(sp)  blt t1, s0, swap\_min  countinue1:  blt s1, t1, swap\_max  countinue2:  sw zero, 0(sp) # Làm trống stack sau khi sử dụng  addi t0, t0, 4 # Tăng biến đếm  addi sp, sp, 4 # Trả lại giá trị của sp  addi t2, t2, 1 # Tăng biến đếm vị trí  j Max\_Min # Vòng lặp tiếp tục thực hiện  end\_max\_min:  jr ra    swap\_min:  #Thay đổi giá trị và vị trí của min  add s2, t2, zero  add s0, t1, zero  j countinue1  swap\_max:  # Thay đổi giá trị và vị trí của min  add s3, t2, zero  add s1, t1, zero  j countinue2 |

* **Các bước thực hiện chương trình:**

+ Cụm câu lệnh trong thẻ data để khởi tạo giá trị các chuỗi để in ra màn hình.

+ Cụm câu lệnh trong thẻ main để gán giá trị cho các thanh ghi từ a0 đên a7

+ Cụm câu lệnh addi sp, sp, -4, sw a7, 0(sp) ... để đưa lần lượt giá trị các thanh ghi từ a7 đến a0 vào stack

+ Tiếp sau đó khởi tạo các giá trị trước khi gọi hàm tìm max và min: Khởi tạo giá trị max, min, biến đếm, vị trí của giá trị max, min, độ dài của stack

+ Câu lệnh trong thẻ Max\_Min để thực hiện việc tìm giá trị nhỏ nhất và lớn nhất của chương trình: duyệt qua lần lượt các giá trị trong stack, nếu giá trị đó nhỏ hơn min thì cập nhật đó là min và lấy vị trí của thanh ghi đó, tương tự nếu giá trị đó lớn hơn max thì cập nhật đó giá trị max và lấy vị trí của biến đó. Sau khi kết thúc vòng lặp quay lại chương trình chính

+ Câu lệnh trong thẻ print để in kết quả ra màn hình với lần lượt các lệnh gọi hệ thông li a7, 4 để in ra các msg1, msg2, msg3 và li a7, 1 để in các giá trị max, min và vị trí của chúng ra ngoài màn hình, sau đó kết thúc chương trình.

* **Quan sát giá trị của các thanh ghi:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| No | Vị trí | s0 | s1 | s2 | s3 | sp | ra | pc | Chú thích |
| 1 | (0) | 0x00000 | 0x00000 | 0x00000 | 0x00000 | 0x7fffeffc | 0x00000 | 0x400000 | Bắt đầu chương trình |
| 2 | (1) | 0x00000 | 0x00000 | 0x00000 | 0x00000 | 0x7fffeffc | 0x00000 | 0x400020 | Nhập giá trị các thanh ghi |
| 3 | (2) | 0x00000 | 0x00000 | 0x00000 | 0x00000 | 0x7fffefdc | 0x00000 | 0x400060 | Đưa giá trị vào stack |
| 4 | (3) | 0x00002 | 0x00002 | 0x00000 | 0x00000 | 0x7fffefdc | 0x400080 | 0x400108 | Tạo các biến lưu max, min |
| 5 | (4) | 0x00002 | 0x00002 | 0x00000 | 0x00000 | 0x7fffefe0 | 0x400080 | 0x400108 | Vòng lặp 1 |
| 6 | (4) | 0x00001 | 0x00002 | 0x00001 | 0x00000 | 0x7fffefe4 | 0x400080 | 0x400108 | Vòng lặp 2 |
| 7 | (4) | 0x00000 | 0x00002 | 0x00002 | 0x00000 | 0x7fffefe8 | 0x400080 | 0x400108 | Vòng lặp 3 |
| 8 | (4) | 0x00000 | 0x00005 | 0x00002 | 0x00003 | 0x7fffefec | 0x400080 | 0x400108 | Vòng lặp 4 |
| 9 | (4) | 0x00000 | 0x00005 | 0x00002 | 0x00003 | 0x7fffeff0 | 0x400080 | 0x400108 | Vòng lặp 5 |
| 10 | (4) | 0xfffffff9 | 0x00005 | 0x00005 | 0x00003 | 0x7fffeff4 | 0x400080 | 0x400108 | Vòng lặp 6 |
| 11 | (4) | 0xfffffff9 | 0x00009 | 0x00005 | 0x00006 | 0x7fffeff8 | 0x400080 | 0x400108 | Vòng lặp 7 |
| 12 | (4) | 0xfffffff9 | 0x00009 | 0x00005 | 0x00006 | 0x7fffeffc | 0x400080 | 0x400108 | Vòng lặp 8 |
| 13 | (5) | 0xfffffff9 | 0x00009 | 0x00005 | 0x00006 | 0x7fffeffc | 0x400080 | 0x400108 | Kết thúc chương trình |

* **Kết quả hiển thị ở của sổ Run i/o**

|  |
| --- |
|  |

* Đúng với yêu cầu của đề bài
* **Giá trị các thanh ghi được lưu trong stack:**

|  |
| --- |
|  |

* Kết quả chương trình đúng với yêu cầu đề bài.