**REPORT WEEK 7**

Họ và tên: Nguyễn Văn Hưng

MSSV: 20225634

Lớp: Thực hành kiến trúc máy tính – Mã lớp: 147799

1. **Assignment 1:**
2. Đề bài: Create a new project to implement the program in Home Assignment 1. Compile and upload to simulator. Change input parameters and observe the memory when run the program step by step. Pay attention to register $pc, $ra to clarify invoking procedure process.
3. Code:

#Laboratory Exercise 7 Home Assignment 1

.text

main: li $a0, -45 #load input parameter

jal abs #jump and link to abs procedure

nop

add $s0, $zero, $v0

li $v0, 10 #terminate

syscall

endmain:

#--------------------------------------------------------------------

# function abs

# param[in] $a0 the interger need to be gained the absolute value

# return $v0 absolute value

#--------------------------------------------------------------------

abs:

sub $v0,$zero,$a0 #put -(a0) in v0; in case (a0)<0

bltz $a0,done #if (a0)<0 then done

nop

add $v0,$a0,$zero #else put (a0) in v0

done:

jr $ra

1. Text segment

A screenshot of a computer

Description automatically generated

1. Giải thích

Đây là một đoạn mã MIPS thực hiện hàm **abs** để tính giá trị tuyệt đối của một số nguyên đầu vào và trả về kết quả. Dưới đây là giải thích từng phần của đoạn mã:

* + - **.text**: Phần **.text** của đoạn mã MIPS chứa mã máy.
    - **main**: Hàm main bắt đầu.
      * **li $a0, -45**: Load tham số đầu vào là số nguyên -45 vào thanh ghi **$a0**.
      * **jal abs**: Nhảy và liên kết (jump and link) tới hàm **abs**. Các kết quả được trả về sẽ được lưu trong thanh ghi **$v0**.
      * **nop**: No operation (không có hoạt động). Thường được sử dụng để đảm bảo sự nhất quán trong việc tạo mã máy.
      * **add $s0, $zero, $v0**: Lưu giá trị trả về từ hàm **abs** vào thanh ghi **$s0**.
    - **endmain**:
      * **li $v0, 10**: Chuẩn bị syscall để kết thúc chương trình.
      * **syscall**: Gọi syscall để kết thúc chương trình.
    - Hàm **abs**:
      * **sub $v0, $zero, $a0**: Lấy số nguyên đầu vào (**$a0**), đảo dấu và lưu vào thanh ghi **$v0**. Điều này đảm bảo rằng giá trị của **$v0** sẽ là số nguyên **-($a0)**.
      * **bltz $a0, done**: Kiểm tra xem số nguyên đầu vào có nhỏ hơn 0 hay không. Nếu nhỏ hơn 0, nhảy tới nhãn **done**.
      * **nop**: No operation (không có hoạt động).
      * **add $v0, $a0, $zero**: Trong trường hợp số nguyên đầu vào không nhỏ hơn 0, lưu giá trị của **$a0** vào thanh ghi **$v0**.
      * **done**: Nhãn **done** được sử dụng để kết thúc hàm và quay trở lại địa chỉ trước khi gọi hàm **abs**.
    - **jr $ra**: Quay trở lại địa chỉ trả về.

Đây là cách thức hoạt động của chương trình tính giá trị tuyệt đối của một số nguyên trong môi trường MIPS assembly.

1. Kết quả
2. Đúng với đề bài là |-45| = |45|

Kết quả cuối cùng ta lấy được giá trị tuyệt đối của số được nạp vào trong thanh ghi $a0 và ghi kết quả đó vào thanh ghi $s0

A screenshot of a table

Description automatically generated

1. Thanh ghi $pc và $ra:

- Khi chạy lệnh jal abs (địa chỉ lệnh 0x00400004) thì thanh ghi $ra được gán bằng địa chỉ của câu lệnh tiếp theo (0x00400008) và thanh ghi pc được gán bằng địa chỉ 0x00400018 (địa chỉ tại nhãn abs)



1. **Assignment 2:**
2. Đề bài: Create a new project to implement the program in Home Assignment 2. Compile and upload to simulator. Change input parameters (register $a0, $a1, $a2) and observe the memory when run the program step by step. Pay attention to register $pc, $ra to clarify invoking procedure process.
3. Code:

#Laboratory Exercise 7, Home Assignment 2

.text

main: li $a0,22 #load test input

li $a1,12

li $a2,34

jal max #call max procedure

nop

add $s0, $v0, $zero

li $v0, 10 #terminate

syscall

endmain:

#---------------------------------------------------------------------

#Procedure max: find the largest of three integers

#param[in] $a0 integers

#param[in] $a1 integers

#param[in] $a2 integers

#return $v0 the largest value

#---------------------------------------------------------------------

max: add $v0,$a0,$zero #copy (a0) in v0; largest so far

sub $t0,$a1,$v0 #compute (a1)-(v0)

bltz $t0,okay #if (a1)-(v0)<0 then no change

nop

add $v0,$a1,$zero #else (a1) is largest thus far

okay: sub $t0,$a2,$v0 #compute (a2)-(v0)

bltz $t0,done #if (a2)-(v0)<0 then no change

nop

add $v0,$a2,$zero #else (a2) is largest overall

done: jr $ra #return to calling program

1. Text segment and data segment

A screenshot of a computer

Description automatically generated

1. Giải thích

Đoạn mã MIPS trên thực hiện hàm `max`, nhận ba số nguyên làm đầu vào và trả về số lớn nhất trong số ba số đó. Dưới đây là giải thích chi tiết về quá trình gọi hàm và cách mã hoạt động:

1. `.text`: Phần `.text` của đoạn mã MIPS chứa mã máy.

2. `main`: Hàm main bắt đầu.

- `li $a0, 22`, `li $a1, 12`, `li $a2, 34`: Load các tham số đầu vào là 22, 12, và 34 vào các thanh ghi `$a0`, `$a1`, và `$a2` tương ứng.

- `jal max`: Nhảy và liên kết (jump and link) tới hàm `max`. Địa chỉ của lệnh tiếp theo (`$ra + 4`) được lưu vào thanh ghi `$ra` (return address register).

- `nop`: No operation (không có hoạt động). Thường được sử dụng để đảm bảo sự nhất quán trong việc tạo mã máy.

- `add $s0, $v0, $zero`: Lưu giá trị trả về từ hàm `max` vào thanh ghi `$s0`.

- `li $v0, 10`: Chuẩn bị syscall để kết thúc chương trình.

- `syscall`: Gọi syscall để kết thúc chương trình.

3. `endmain`: Cuối cùng của hàm main.

4. Hàm `max`:

- `add $v0, $a0, $zero`: Copy giá trị của `$a0` vào `$v0`. Điều này đặt `$v0` là giá trị lớn nhất cho đến nay.

- `sub $t0, $a1, $v0`: Tính hiệu số thứ hai và giá trị lớn nhất đến nay, lưu vào `$t0`.

- `bltz $t0, okay`: Kiểm tra xem nếu hiệu số thứ hai là âm thì không có thay đổi.

- `nop`: No operation (không có hoạt động).

- `add $v0, $a1, $zero`: Nếu hiệu số thứ hai không âm, sao chép giá trị của `$a1` vào `$v0`, đặt `$v0` là giá trị lớn nhất đến nay.

- `okay`: Nhãn dùng để bắt đầu tính toán cho số thứ ba.

- Tương tự, hàm tiếp tục kiểm tra số thứ ba và so sánh với giá trị lớn nhất đến nay.

- `jr $ra`: Trả về địa chỉ được lưu trong `$ra` để quay lại lệnh tiếp theo sau lệnh `jal`.

1. Kết quả
2. Đúng với đề bài là trong 3 giá trị 12,22,34 thì 34 là lớn nhất

Kết quả cuối cùng ta ghi kết quả đó vào thanh ghi $s0

A table with numbers and lines

Description automatically generated

1. Thanh ghi $pc và $ra:

Khi chạy lệnh jal max thì thanh ghi $ra được gán bằng giá trị của địa chỉ câu lệnh tiếp theo sau jal trong nhãn main là 0x004000010. Thanh ghi pc được gán bằng địa chỉ của nhãn max để câu lệnh tiếp tục được thực hiện bắt đầu từ nhãn max là 0x004000020. Sau khi chạy đến jr $ra thì pc được gán bằng địa chỉ trong $ra (địa chỉ của nop) là 0x004000010

1. **Assignment 3:**
2. Đề bài: Create a new project to implement the program in Home Assignment 3. Compile and upload to simulator. Pass test value to registers $s0 and $s1, observe run process, pay attention to stack pointer. Goto memory space that pointed by $sp register to view push and pop operations in detail.
3. Code:

#Laboratory Exercise 7, Home Assignment 3

.text

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

lw $s1,4($sp) #pop from stack to $s1

addi $sp,$sp,8 #adjust the stack pointer

1. Text segment

A screenshot of a computer

Description automatically generated

1. Giải thích

Đoạn mã MIPS trên thực hiện hai hàm **push** và **pop** để thêm và loại bỏ giá trị từ ngăn xếp (stack). Dưới đây là giải thích chi tiết về cách mã hoạt động:

1. **.text**: Phần **.text** của đoạn mã MIPS chứa mã máy.
2. **push**: Hàm **push** được sử dụng để đẩy các giá trị từ các thanh ghi **$s0** và **$s1** vào ngăn xếp.
   * **addi $sp, $sp, -8**: Điều chỉnh con trỏ ngăn xếp (**$sp**) để mở rộng ngăn xếp thêm 8 byte.
   * **sw $s0, 4($sp)**: Lưu giá trị của thanh ghi **$s0** vào địa chỉ mới trên ngăn xếp.
   * **sw $s1, 0($sp)**: Lưu giá trị của thanh ghi **$s1** vào địa chỉ mới trên ngăn xếp.
3. **work**: Một số lệnh không hoạt động (no operation) được thực thi. Có thể là để thực hiện công việc nào đó trong quá trình lập trình, nhưng trong đoạn mã này, không có hoạt động cụ thể nào được thực hiện.
4. **pop**: Hàm **pop** được sử dụng để loại bỏ các giá trị từ ngăn xếp và lưu chúng vào các thanh ghi **$s0** và **$s1**.
   * **lw $s0, 0($sp)**: Tải giá trị từ địa chỉ trên ngăn xếp vào thanh ghi **$s0**.
   * **lw $s1, 4($sp)**: Tải giá trị từ địa chỉ trên ngăn xếp (offset 4 byte) vào thanh ghi **$s1**.
   * **addi $sp, $sp, 8**: Điều chỉnh con trỏ ngăn xếp (**$sp**) để giảm kích thước ngăn xếp đi 8 byte, loại bỏ các giá trị đã được lấy ra khỏi ngăn xếp.

Hai hàm này có thể được sử dụng để lưu trữ và khôi phục giá trị của các thanh ghi **$s0** và **$s1** một cách an toàn trong quá trình thực thi của chương trình MIPS.

Top of Form

1. Kết quả
2. Trường hợp 1: $s0=100 , $s1 = 66 Kết quả sau khi đổi chỗ



1. Trường hợp 2: $s0=55 , $s1 = 66 Kết quả sau khi đổi chỗ



1. Trường hợp 1: $s0=55 , $s1 = 45 Kết quả sau khi đổi chỗ



1. Đối với thanh ghi $sp

* Trước khi chạy lệnh addi ở nhãn push: $sp = 0x7fffeffc
* Sau khi chạy lệnh addi ở nhãn push: $sp = 0x7fffeff4
* Thanh ghi $sp được giảm đi 8 byte (tức là có sự cấp phát cho bộ nhớ stack 8 byte)
* Sau đó lần lượt ghi giá trị trong $s0 vào $sp + 4, giá trị trong $s1 vào $sp + 0
* Sau khi chạy lệnh addi ở nhãn pop:
* Thực hiện đổi chỗ hai số bằng cách load giá trị tại địa chỉ $sp + 0 vào $s1, load giá trị tại địa chỉ $sp + 4 vào $s1
* Lệnh add $sp, $sp, 8 (giúp giải phóng stack, trả lại đỉnh stack)

1. **Assignment 4:**
2. Đề bài: Create a new project to implement the program in Home Assignment 4. Compile and upload to simulator. Pass test input through register $a0, run this program and test result in register $v0. Run this program in step-by-step mode, observe the changing of register $pc, $ra, $sp and $fp. Draw the stack through this recursive program in case of n=3 (compute 3!).
3. Code:

#Laboratory Exercise 7, Home Assignment 4

.data

Message: .asciiz "Ket qua tinh giai thua la: "

.text

main: jal WARP

print: add $a1, $v0, $zero # $a0 = result from N!

li $v0, 56

la $a0, Message

syscall

quit: li $v0, 10 #terminate

syscall

endmain:

#---------------------------------------------------------------------

#Procedure WARP: assign value and call FACT

#---------------------------------------------------------------------

WARP: sw $fp,-4($sp) #save frame pointer (1)

addi $fp,$sp,0 #new frame pointer point to the top (2)

addi $sp,$sp,-8 #adjust stack pointer (3)

sw $ra,0($sp) #save return address (4)

li $a0,10 #load test input N

jal FACT #call fact procedure

nop

lw $ra,0($sp) #restore return address (5)

addi $sp,$fp,0 #return stack pointer (6)

lw $fp,-4($sp) #return frame pointer (7)

jr $ra

wrap\_end:

#---------------------------------------------------------------------

#Procedure FACT: compute N!

#param[in] $a0 integer N

#return $v0 the largest value

#---------------------------------------------------------------------

FACT: sw $fp,-4($sp) #save frame pointer

addi $fp,$sp,0 #new frame pointer point to stack’s top

addi $sp,$sp,-12 #allocate space for $fp,$ra,$a0 in stack

sw $ra,4($sp) #save return address

sw $a0,0($sp) #save $a0 register

slti $t0,$a0,2 #if input argument N < 2

beq $t0,$zero,recursive#if it is false ((a0 = N) >=2)

nop

li $v0,1 #return the result N!=1

j done

nop

recursive:

addi $a0,$a0,-1 #adjust input argument

jal FACT #recursive call

nop

lw $v1,0($sp) #load a0

mult $v1,$v0 #compute the result

mflo $v0

done: lw $ra,4($sp) #restore return address

lw $a0,0($sp) #restore a0

addi $sp,$fp,0 #restore stack pointer

lw $fp,-4($sp) #restore frame pointer

jr $ra #jump to calling

fact\_end:

1. Text segment

A screenshot of a computer

Description automatically generated

1. Giải thích

Đoạn mã MIPS trên thực hiện hai thủ tục: **WARP** và **FACT**, cũng như hàm **main** để gọi thủ tục và in ra kết quả. Dưới đây là giải thích chi tiết về cách mã hoạt động:

1. **.data**: Phần **.data** chứa dữ liệu tĩnh, trong trường hợp này là chuỗi "Ket qua tinh giai thua la: " được đặt tên là **Message**.
2. **.text**: Phần **.text** của đoạn mã MIPS chứa mã máy.
3. **main**: Hàm main bắt đầu.
   * **jal WARP**: Nhảy và liên kết (jump and link) tới thủ tục **WARP**. Địa chỉ của lệnh tiếp theo (**$ra + 4**) được lưu vào thanh ghi **$ra** (return address register).
   * **print**: In ra kết quả tính giai thừa.
     + **add $a1, $v0, $zero**: Sao chép giá trị kết quả từ **$v0** (giá trị trả về của thủ tục **WARP**) vào **$a1**.
     + **li $v0, 56**: Chuẩn bị syscall để in ra một chuỗi và một số nguyên.
     + **la $a0, Message**: Đưa địa chỉ của chuỗi **Message** vào **$a0**.
     + **syscall**: Gọi syscall để in ra chuỗi.
   * **quit**: Kết thúc chương trình.
4. **endmain**: Kết thúc hàm main.
5. **WARP**: Thủ tục **WARP** được sử dụng để gán giá trị và gọi thủ tục **FACT**.
   * Lưu lại **frame pointer** và **return address** vào ngăn xếp.
   * Gọi thủ tục **FACT** với tham số là 10.
   * Khôi phục giá trị của **frame pointer** và **return address** từ ngăn xếp.
   * Trả về địa chỉ trả về.
6. **FACT**: Thủ tục **FACT** được sử dụng để tính giai thừa của một số nguyên.
   * Lưu lại **frame pointer**, **return address** và tham số **N** vào ngăn xếp.
   * Kiểm tra nếu **N < 2**, trả về 1.
   * Trong trường hợp còn lại, thực hiện đệ quy bằng cách gọi lại chính nó với **N-1**.
   * Sau đó, nhân kết quả của đệ quy với **N**.
   * Khôi phục **frame pointer**, **return address** và tham số **N** từ ngăn xếp.
   * Trả về kết quả.

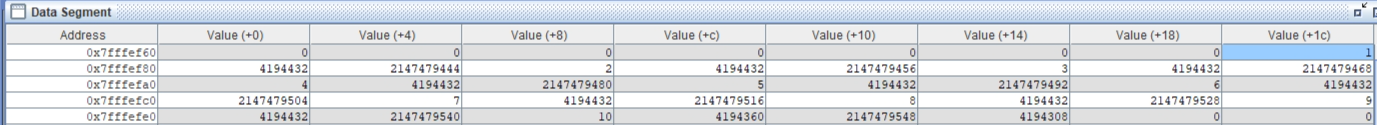
Những hàm này kết hợp với nhau để tính toán và in ra kết quả của một phép tính giai thừa.

1. Kết quả
2. Với n = 10 Kết quả đúng với tính toán là 10! = 3628800

A screenshot of a computer

Description automatically generated

1. Sự thay đổi của thanh ghi $sp:



1. Bảng thể hiện giá trị ngăn xếp:

Với trường hợp n=3 và tính 3! = 6

|  |  |
| --- | --- |
| 0x7fffeff8 | $fp = 0x00000000 |
| 0x7fffeff4 | $ra = 0x00400004 |
| 0x7fffeff0 | $fp = 0x7fffeffc |
| 0x7fffefec | $ra = 0x00400038 |
| 0x7fffefe8 | $a0 = 0x00000003 |
| 0x7fffefe4 | $fp = 0x7fffeff4 |
| 0x7fffefe0 | $ra = 0x00400080 |
| 0x7fffefdc | $a0 = 0x00000002 |
| 0x7fffefd8 | $fp = 0x7fffefe4 |
| 0x7fffefd4 | $ra = 0x00400080 |
| 0x7fffefd0 | $a0 = 0x00000001 |

1. **Assignment 5:**
2. Đề bài: Write a procedure to find the largest, the smallest and these positions in a list of 8 elements that are stored in regsiters $s0 through $s7. For example: Largest: 9,3 => The largest element is stored in $s3, largest value is 9 Smallest: -3,6 => The smallest element is stored in $s6, smallest value is -3

Tips: using stack to pass arguments and return results.

1. Code:

.data

max: .asciiz"Largest: "

line: .asciiz"\n"

coma: .asciiz ","

min: .asciiz"Smallest: "

.text

# Khoi tao cac gia tri

li $s0, 12

li $s1, 60

li $s2, 13

li $s3, 100

li $s4, -200

li $s5, -100

li $s6, 30

li $s7,-10

# Luu dong gia tri kia vao trong stack

# Mo rong stack

#luu cac gia tri cua $s vao trong stack

main: addi $sp, $sp ,-32

sw $s0, 0($sp)

sw $s1, 4($sp)

sw $s2, 8($sp)

sw $s3, 12($sp)

sw $s4, 16($sp)

sw $s5, 20($sp)

sw $s6, 24($sp)

sw $s7, 28($sp)

# nhay den ham tim Max

jal findMax

# nhay den ham tim Min

jal findMin

li $v0 , 10

syscall

# tim kiem nhu binh thuong

findMax:

la $t1, 0($sp)

la $s3, 0($sp) # t3 = &A[0] =s3

la $s4, 32($sp)# t4 = &A[n] = s4

lw $s0, 0($sp) # tmp = A[0] = s0

addi $s1, $0 ,0 # index = 0 = s1

addi $t2, $0, 0 # i = 0 = t2

loop: slt $t3, $s3, $s4 # &A[n] > &a[0] -> t3 = 1

bne $t3, $0 , check

j done

check: # s5 = A[0]

lw $s5, 0($s3)

# tmp < A[i] -> t3 = 1 -> doi cho

slt $t3, $s0, $s5

beq $t3, $0, exit # tiep tuc vong lap

# tmp = A[i]

addi $s0, $s5, 0

# index = i

addi $s1, $t2, 0

j exit

done:

#-------------------------------------

#in ra max

li $v0, 4

la $a0, max

syscall

li $v0, 1

addi $a0, $s0, 0

syscall

li $v0, 4

la $a0, coma

syscall

li $v0, 1

addi $a0, $s1,0

syscall

li $v0, 4

la $a0, line

syscall

#Khoi phuc lai $s

lw $s0, 0($sp)

lw $s1, 4($sp)

lw $s2, 8($sp)

lw $s3, 12($sp)

lw $s4, 16($sp)

lw $s5, 20($sp)

lw $s6, 24($sp)

lw $s7, 28($sp)

#chu y chua duoc khoi phuc stack

jr $ra

exit:

addi $t2, $t2, 1

mul $t2, $t2, 4

add $s3, $t1, $t2

div $t2, $t2, 4

j loop

findMin:

la $t1, 0($sp)

la $s3, 0($sp) # t3 = &A[0] =s3

la $s4, 32($sp)# t4 = &A[n] = s4

lw $s0, 0($sp) # tmp = A[0] = s0

addi $s1, $0 ,0 # index = 0 = s1

addi $t2, $0, 0 # i = 0 = t2

loop2: slt $t3, $s3, $s4 # &A[n] > &a[0] -> t3 = 1

bne $t3, $0 , check2

j done2

check2: # s5 = A[0]

lw $s5, 0($s3)

# tmp > A[i] -> t3 = 1 -> doi cho

slt $t3, $s5, $s0

beq $t3, $0, exit2 # tiep tuc vong lap

# tmp = A[i]

addi $s0, $s5, 0

# index = i

addi $s1, $t2, 0

j exit2

done2:

#-------------------------------------

#in ra min

li $v0, 4

la $a0, min

syscall

li $v0, 1

addi $a0, $s0, 0

syscall

li $v0, 4

la $a0, coma

syscall

li $v0, 1

addi $a0, $s1,0

syscall

#Khoi phuc lai $s

lw $s0, 0($sp)

lw $s1, 4($sp)

lw $s2, 8($sp)

lw $s3, 12($sp)

lw $s4, 16($sp)

lw $s5, 20($sp)

lw $s6, 24($sp)

lw $s7, 28($sp)

# Khoi phuc lai stack

addi $sp, $sp, 32

jr $ra

exit2:

addi $t2, $t2, 1

mul $t2, $t2, 4

add $s3, $t1, $t2

div $t2, $t2, 4

j loop2

1. Giải thích

Đoạn mã MIPS trên thực hiện hai nhiệm vụ: tìm giá trị lớn nhất và nhỏ nhất trong một mảng các số nguyên và in ra kết quả.

Dưới đây là giải thích chi tiết về cách mã hoạt động:

1. **.data**: Phần **.data** chứa các chuỗi cần in ra khi in kết quả.
2. **.text**: Phần **.text** của đoạn mã MIPS chứa mã máy.
3. **main**: Hàm **main** bắt đầu.
   * Khởi tạo các giá trị của mảng.
   * Mở rộng stack và lưu các giá trị của mảng vào trong stack.
   * Gọi hàm **findMax** để tìm giá trị lớn nhất.
   * Gọi hàm **findMin** để tìm giá trị nhỏ nhất.
   * Kết thúc chương trình.
4. **findMax**: Hàm này được sử dụng để tìm giá trị lớn nhất trong mảng.
   * Khởi tạo con trỏ **$s3** trỏ đến đầu mảng và con trỏ **$s4** trỏ đến cuối mảng.
   * Sử dụng vòng lặp để duyệt qua từng phần tử trong mảng.
   * So sánh giá trị hiện tại với giá trị lớn nhất hiện tại, nếu tìm thấy giá trị lớn hơn, cập nhật giá trị lớn nhất và vị trí của nó.
   * In ra kết quả giá trị lớn nhất.
5. **findMin**: Hàm này được sử dụng để tìm giá trị nhỏ nhất trong mảng.
   * Tương tự như **findMax**, nhưng thay vì tìm giá trị lớn nhất, ta tìm giá trị nhỏ nhất.
   * In ra kết quả giá trị nhỏ nhất.

Mã cũng chú ý đến việc khôi phục các giá trị thanh ghi và stack sau khi hoàn thành các nhiệm vụ.

1. Kết quả

Với mảng lần lượt là 12, 60, 13, 100, -200, -100, 30, -10

Đúng với kết quả tính toán được

