

Bài 3. Các lệnh nhảy và lệnh rẽ nhánh

Họ và tên: Nguyễn Thành Duy

MSSV: 20235696

Assignment 1

Kết quả sau khi chạy chương trình:

The screenshot shows the MARS MIPS simulator interface. The main window displays assembly code for a program named 'riscv2.asm'. The code includes instructions like 'addi s1, zero, 1', 'blt s2, s1, else', and 'addi s2, zero, 1'. The 'Registers' panel on the right shows the state of various registers, including 's1' (value 9) and 's2' (value 18). The 'Data Segment' panel shows memory addresses and their corresponding values. The 'Messages' panel at the bottom indicates that the assembly operation completed successfully.

Sau khi chạy câu lệnh thứ nhất:

`addi s1, zero, 1` # Khởi tạo giá trị i vào thanh ghi s1

+) Thanh ghi: giá trị s1 được gán bằng 1

s1	9	0x00000001
----	---	------------

+) Thanh pc: 0x00400000 -> 0x00400004

pc		0x00400004
----	--	------------

Câu lệnh thứ hai tương tự câu lệnh thứ nhất: giá trị s2 được gán bằng 2, thanh ghi pc 0x00400004 -> 0x00400008

s2	18	0x00000002
----	----	------------

pc		0x00400008
----	--	------------

Câu lệnh thứ ba: So sánh giá trị s2 và s1, nếu s2 < s1 thì sẽ nhảy đến else còn ngược lại sẽ thực hiện câu lệnh ở ngay dưới:

```
blt s2, s1, else # if j < i then jump else
```

Trong trường hợp này: $s1 = 1$, $s2 = 2$ nên ta thấy $s2 > s1$ nên sẽ thực hiện câu lệnh ở ngay phía dưới tức là then

+) Thanh ghi: $0x00400008 \rightarrow 0x0040000c$

pc		0x0040000c
----	--	------------

```
then:
addi t1, t1, 1 # then part: x=x+1
addi t3, zero, 1 # z=1
j endif # skip else part
```

+) Câu lệnh sau đây thực hiện tăng giá trị của x lên 1

```
addi t1, t1, 1 # then part: x=x+1
```

Thanh ghi:

t1	6	0x00000001
----	---	------------

pc		0x00400010
----	--	------------

+) Câu lệnh sau đây thực hiện gán giá trị của t3 bằng 1: Do trong riscv không có câu lệnh gán trực tiếp nên ta phải thực hiện gán thông qua phép cộng với 0 hay zero trong thanh ghi

```
addi t3, zero, 1 # z=1
```

Thanh ghi:

t3	28	0x00000001
----	----	------------

pc		0x00400014
----	--	------------

+) Do đã thực hiện xong các hành động ở **then** máy tính sẽ tự động chạy các câu lệnh ở ngay sau nó theo tuần tự nên để không chạy câu lệnh ngay sau là else (ở trường hợp $s2 < s1$) thì ta cần một lệnh nhảy tự do:

```
j endif # skip else part
```

Thanh ghi:

pc		0x00400020
----	--	------------

* Thanh ghi sau khi chạy xong chương trình:

pc	0x00400024
----	------------

Assignment 2

```
# Laboratory 3, Home Assignment 2
.data
A: .word 1, 3, 2, 5, 4, 7, 8, 9, 6
.text
# TODO: Khởi tạo giá trị các thanh ghi s2, s3, s4
li s1, 0 # i = 0
li s5, 0 # sum = 0
li s3, 9 # n = 9
la s2, A # Lấy địa chỉ của A trong vùng nhớ dữ liệu
li s4, 1
loop:
bge s1, s3, endloop # if i >= n then end loop
add t1, s1, s1 # t1 = 2 * s1
add t1, t1, t1 # t1 = 4 * s1 => t1 = 4*i
add t1, t1, s2 # t1 store the address of A[i]
lw t0, 0(t1) # load value of A[i] in t0
add s5, s5, t0 # sum = sum + A[i]
add s1, s1, s4 # i = i + step
j loop # go to loop
endloop:
```

Mục đích: để tính tổng các phần tử của mảng A

Kết quả mong muốn: $\text{sum} = 1 + 3 + 2 + 5 + 4 + 7 + 8 + 9 + 6 = 45$

Kết quả sau khi chạy đoạn chương trình:

The screenshot displays the RISC-V simulator interface. The **Text Segment** window shows the assembly code with addresses and comments. The **Data Segment** window shows the memory layout of the array A. The **Registers** window shows the values of the registers, including the program counter (pc) at 0x00400024. The **Messages** window shows the successful completion of the assembly operation.

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000001	0x00000003	0x00000002	0x00000005	0x00000004	0x00000007	0x00000008	0x00000009
0x10010004	0x00000006	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010008	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1001000c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010010	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010014	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010018	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1001001c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010024	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010028	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1001002c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010030	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010034	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010038	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1001003c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010044	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010048	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1001004c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010050	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010054	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010058	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1001005c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010064	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010068	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1001006c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010070	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010074	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010078	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1001007c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010084	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010088	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1001008c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010090	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010094	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010098	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1001009c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a4	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a8	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100ac	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100b0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100b4	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100b8	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100bc	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c4	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c8	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100cc	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100d0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100d4	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100d8	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100dc	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e4	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e8	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100ec	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100f0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100f4	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100f8	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100fc	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010104	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010108	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1001010c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010110	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010114	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010118	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1001011c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010124	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010128	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1001012c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010130	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010134	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010138	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x1001013c	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010144	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010148	0x00000000	0x00000000	0x00000000	0x00				

+) Câu lệnh đầu tiên: Khởi tạo mảng A có 9 phần tử và lần lượt có các giá trị như trong code

```
# Laboratory 3, Home Assignment 2
.data
A: .word 1, 3, 2, 5, 4, 7, 8, 9, 6
```

+) Ở đoạn lệnh sau:

```
# TODO: Khởi tạo giá trị các thanh ghi s2, s3, s4
li s1, 0 # i = 0
li s5, 0 # sum = 0
li s3, 9 # n = 9
la s2, A # Lấy địa chỉ của A trong vùng nhớ dữ liệu
li s4, 1
```

- Câu lệnh li (load immediate) là một giả lệnh dùng để nạp một số nguyên tức thời (immediate) vào một thanh ghi

+ Câu lệnh đầu tiên: Câu lệnh sẽ gán giá trị của s1 với giá trị là 0

```
li s1, 0 # i = 0
```

Thanh ghi:

s1	9	0x00000000
pc		0x00400004

+ Câu lệnh thứ hai và thứ ba và thứ tư tương tự như câu lệnh đầu tiên lần lượt gán giá trị của s3, s4, s5 là 9 (số phần tử của mảng A ban đầu khởi tạo), 1 (step trong vòng lặp ở dưới), 0 (khởi tạo tổng của mảng bằng 0)

Thanh ghi:

s5	21	0x00000000
pc		0x00400004
s3	19	0x00000009

pc		0x0040000c
----	--	------------

s4	20	0x00000001
----	----	------------

pc		0x00400018
----	--	------------

+ Ở câu lệnh thứ tư: câu lệnh **la** là lấy địa chỉ của A trong vùng nhớ dữ liệu

- Lệnh la được biên dịch thành 2 câu lệnh: auipc và addi

+ Địa chỉ tuyệt đối được chia thành hai phần: phần cao (20-bit) và phần thấp (12-bit).

Kết hợp hai câu lệnh này để tải một địa chỉ đầy đủ 32-bit.

+ auipc: giúp lấy phần trên (20-bit) của địa chỉ nhãn dựa vào pc, auipc tính toán phần cao của địa chỉ label bằng cách lấy giá trị pc hiện tại và cộng thêm một giá trị offset.

+ addi: bổ sung phần thấp (12-bit) để hoàn chỉnh địa chỉ

-> Giúp tải địa chỉ của nhãn (label) vào thanh ghi

-> Cách này giúp chương trình có thể chạy độc lập với vị trí của bộ nhớ

la s2, A # Lấy địa chỉ của A trong vùng nhớ dữ liệu

Labels	
Label	Address ▲
riscv2.asm	
loop	0x00400018
endloop	0x00400038
A	0x10010000
<input checked="" type="checkbox"/> Data <input checked="" type="checkbox"/> Text	

Ta thấy địa chỉ của A: 0x10010000

Thanh ghi: đã lấy được địa chỉ s2 như mong muốn

s2	18	0x10010000
----	----	------------

pc		0x00400014
----	--	------------

+ Ở đoạn code:

```
loop:
bge s1, s3, endloop # if i >= n then end loop
add t1, s1, s1 # t1 = 2 * s1
add t1, t1, t1 # t1 = 4 * s1 => t1 = 4*i
add t1, t1, s2 # t1 store the address of A[i]
lw t0, 0(t1) # load value of A[i] in t0
add s5, s5, t0 # sum = sum + A[i]
add s1, s1, s4 # i = i + step
j loop # go to loop
endloop:
```

Câu lệnh đầu tiên: điều kiện nếu $s1 \geq s3$ thì ta sẽ kết thúc vòng lặp tức end loop. Ta có $s1 = 0$, $s3 = 9$ thấy $s1 < s3$ nên máy tính sẽ thực hiện câu lệnh ngay ở phía dưới

```
bge s1, s3, endloop # if i >= n then end loop
```

Câu lệnh: Mục đích là lấy được địa chỉ của $A[i]$ vì mỗi phần tử cách nhau 4 bit nên ta thực hiện phép cộng **add** là vì vậy

```
add t1, s1, s1 # t1 = 2 * s1
add t1, t1, t1 # t1 = 4 * s1 => t1 = 4*i
add t1, t1, s2 # t1 store the address of A[i]
```

Câu lệnh: Cộng phần tử $A[i]$ vào sum rồi tăng step rồi lặp lại vòng lặp cho tới khi $i > n$ sẽ thoát khỏi vòng lặp

```
lw t0, 0(t1) # load value of A[i] in t0
add s5, s5, t0 # sum = sum + A[i]
add s1, s1, s4 # i = i + step
j loop # go to loop
```

Giá trị thanh ghi s5 khi kết thúc chương trình: $s5 = 45$ đúng như mong muốn

s5	21	0x0000002d
----	----	------------

Ta thử thay đổi bộ giá trị: $A = \{1, 3, 4, 5\}$ kết quả mong muốn khi chạy $\text{sum} = 1 + 3 + 4 + 5 = 13$

Code:

```
# Laboratory 3, Home Assignment 2
.data
A: .word 1, 3, 4, 5
.text
# TODO: Khởi tạo giá trị các thanh ghi s2, s3, s4
li s1, 0 # i = 0
li s5, 0 # sum = 0
li s3, 4 # n = 4
la s2, A # Lấy địa chỉ của A trong vùng nhớ dữ liệu
li s4, 1
loop:
bge s1, s3, endloop # if i >= n then end loop
add t1, s1, s1 # t1 = 2 * s1
add t1, t1, s1 # t1 = 4 * s1 => t1 = 4*i
add t1, t1, s2 # t1 store the address of A[i]
lw t0, 0(t1) # load value of A[i] in t0
add s5, s5, t0 # sum = sum + A[i]
add s1, s1, s4 # i = i + step
j loop # go to loop
endloop:
```

The screenshot displays the MARS MIPS simulator interface. The **Text Segment** window shows the assembly code being executed, with addresses and comments. The **Data Segment** window shows memory addresses and values. The **Registers** window shows the state of registers, with **s5** containing the value 13. The **Messages** window shows the program completion message.

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7ffffefc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000005
t1	6	0x10010000
t2	7	0x00000000
a0	8	0x00000000
a1	9	0x00000004
a2	10	0x00000000
a3	11	0x00000000
a4	12	0x00000000
a5	13	0x00000000
a6	14	0x00000000
a7	15	0x00000000
s0	16	0x00000000
s1	17	0x00000000
s2	18	0x10010000
s3	19	0x00000004
s4	20	0x00000001
s5	21	0x0000000d
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
s12	28	0x00000000
s13	29	0x00000000
s14	30	0x00000000
s15	31	0x00000000
pc		0x0040003c

Giá trị thanh ghi s5 = 13 (chương trình đã chạy đúng)

Assignment 3

Giải lệnh:

```
switch(test) {  
  
    case 0:  
  
        a=a+1; break;  
  
    case 1:  
  
        a=a-1; break;  
  
    case 2:  
  
        b=2*b; break;  
  
}
```

Ta kiểm tra giá trị test nếu test = i thì chương trình nhảy tới case i: (tương ứng với test = i) và thực hiện hành động rồi thoát khỏi chương trình. Ta sẽ minh họa trên riscv:

Kết quả sau khi chạy chương trình:

The screenshot displays the RISC-V simulator interface. The **Text Segment** window shows the assembly code for the switch statement, including labels for case 0, case 1, case 2, and default. The **Registers** window shows the state of registers, with \$a2 highlighted at address 0x10010000. The **Messages** window shows the program finishing running.

Text Segment

Addr	Code	Basic	Source
0x00400000	0x00401041	auipc x8,0x00000000	5: la x8, test # Map địa chỉ của biến te...
0x00400004	0x00404113	addi x8,x8,0	
0x00400008	0x00002403	lw x5,0(x8)	6: lw x1, 0(x0) # Map giá trị của biến t...
0x0040000c	0x00000293	addi x5,x5,0	7: li t0, 0 # Map giá trị cần kiểm tra
0x00400010	0x00100313	addi x6,x5,1	8: li t1, 1 # Map giá trị cần kiểm tra
0x00400014	0x00200393	addi x7,x5,2	9: li t2, 2 # Map giá trị cần kiểm tra
0x00400018	0x00548863	beq x5,x5,0x00000010	10: beq x1, t0, case_0
0x0040001c	0x00648a63	beq x5,x5,0x00000014	11: beq x1, t1, case_1
0x00400020	0x00748c63	beq x5,x5,0x00000018	12: beq x1, t2, case_2
0x00400024	0x01c000ff	jal x0,0x0000001c	13: j default
0x00400028	0x00190913	addi x18,x18,1	15: addi x2, x2, 1 # a = a + 1
0x0040002c	0x0140006f	jal x0,0x00000014	16: j continue

Registers

Name	Number	Value
\$zero	0	0x00000000
\$ra	1	0x00000000
\$sp	2	0x7ffffefc
\$gp	3	0x10008000
\$tp	4	0x00000000
\$t0	5	0x00000000
\$t1	6	0x00000001
\$t2	7	0x00000002
\$a0	8	0x10010000
\$a1	9	0x00000000
\$a2	10	0x00000000
\$a3	11	0x00000000
\$a4	12	0x00000000
\$a5	13	0x00000000
\$a6	14	0x00000000
\$a7	15	0x00000000
\$a8	16	0x00000000
\$a9	17	0x00000000
\$a10	18	0x00000001
\$a11	19	0x00000000
\$a12	20	0x00000000
\$a13	21	0x00000000
\$a14	22	0x00000000
\$a15	23	0x00000000
\$a16	24	0x00000000
\$a17	25	0x00000000
\$a18	26	0x00000000
\$a19	27	0x00000000
\$a20	28	0x00000000
\$a21	29	0x00000000
\$a22	30	0x00000000
\$a23	31	0x00000000
\$pc		0x00400044

Messages

```
-- program is finished running (dropped off bottom) --  
Clear  
-- program is finished running (dropped off bottom) --
```


Giá trị thanh ghi sau khi chạy câu lệnh đầu tiên:

```
la s0, test # Nạp địa chỉ của biến test vào s0
```

s0	8	0x10010000
pc		0x00400008

Giá trị thanh ghi sau khi chạy câu lệnh thứ hai:

```
lw s1, 0(s0) # Nạp giá trị của biến test vào s1
```

s1	9	0x00000000
s1	9	0x00000000

Giá trị thanh ghi sau khi chạy câu lệnh thứ ba:

```
li t0, 0 # Nạp giá trị cần kiểm tra
```

t0	5	0x00000000
pc		0x00400010

Giá trị thanh ghi sau khi chạy câu lệnh thứ tư:

```
beq s1, t1, case_1
```

t1	6	0x00000001
pc		0x00400014

Giá trị thanh ghi sau khi chạy câu lệnh thứ năm:

```
li t2, 2 # Nạp giá trị cần kiểm tra
```

pc		0x00400014
----	--	------------

```
li t2, 2 # Nạp giá trị cần kiểm tra
```

Câu lệnh tiếp theo sẽ xem giá trị test là bao nhiêu để nhảy tới câu lệnh tương ứng để thực hiện hành động

```
beq s1, t0, case_0
```

Ta thấy s1 = 0 (giá trị của test), t0 = 0 Vậy nên sẽ nhảy tới case 0 tương ứng

```

case_0:
addi s2, s2, 1 # a = a + 1
j continue

```

Do tăng giá trị của s2 lên 1 nên ở thanh ghi giá trị của s2 là 1 (đúng như mong muốn)

s2	18	0x00000001
----	----	------------

Ta sẽ thử một số trường hợp khác để xem chương trình chạy đúng không:

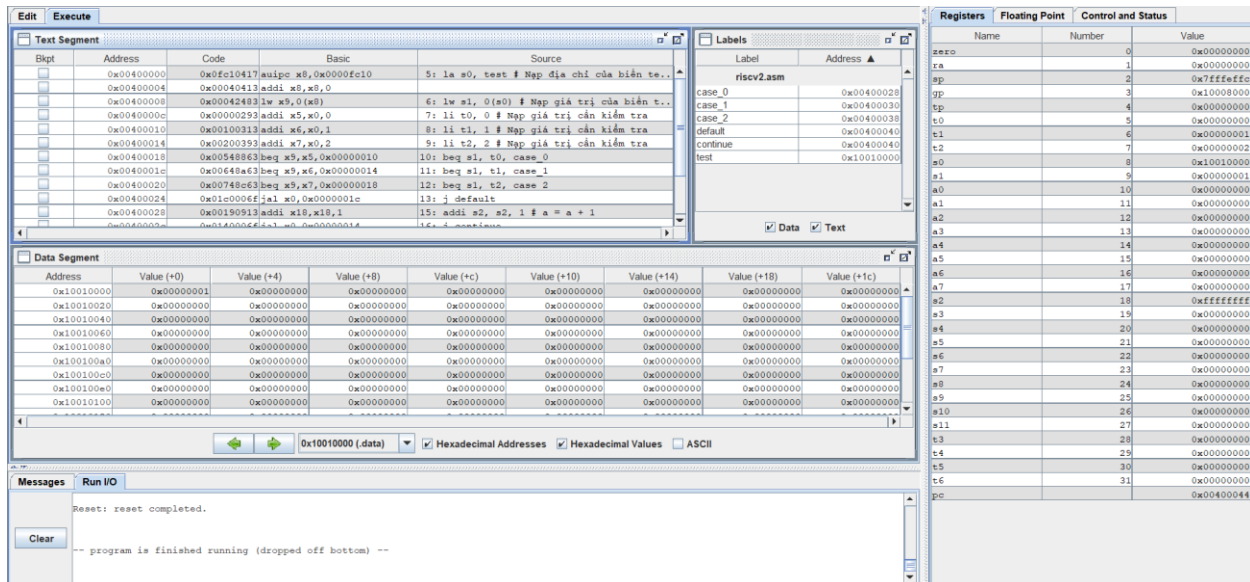
TH1: khởi tạo test = 1 thì theo mong muốn thì giá trị s2 = -1

```

# Laboratory Exercise 3, Home Assignment 3
.data
test: .word 1
.text
la s0, test # Nạp địa chỉ của biến test vào s0
lw s1, 0(s0) # Nạp giá trị của biến test vào s1
li t0, 0 # Nạp giá trị cần kiểm tra
li t1, 1 # Nạp giá trị cần kiểm tra
li t2, 2 # Nạp giá trị cần kiểm tra
beq s1, t0, case_0
beq s1, t1, case_1
beq s1, t2, case_2
j default
case_0:
addi s2, s2, 1 # a = a + 1
j continue
case_1:
sub s2, s2, t1 # a = a - 1
j continue
case_2:
add s3, s3, s3 # b = 2 * b
j continue
default:
continue:

```

Kết quả:



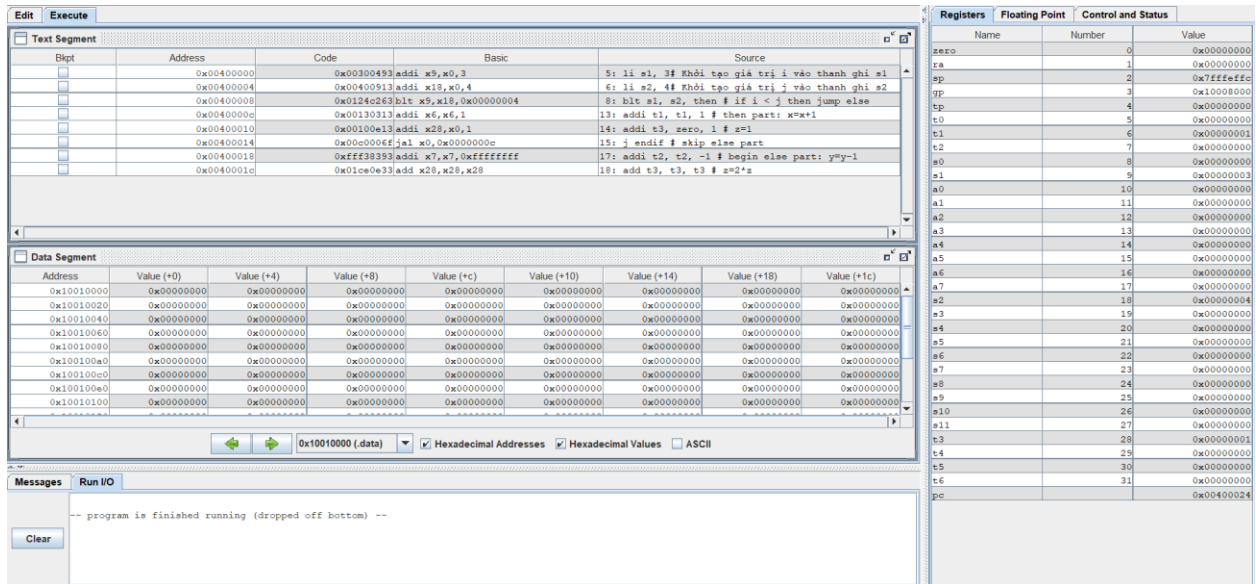
Giá trị thanh ghi s2 = -1 (như mong muốn)

Assignment 4

a) $i = 3, j = 4$. Nếu $i < j$ thì $t1 = 1, t3 = 1$ ngược lại sẽ thực hiện câu lệnh else

Code:

```
# Laboratory Exercise 3, Home Assignment 1
.text
start:
# TODO:
li s1, 3# Khởi tạo giá trị i vào thanh ghi s1
li s2, 4# Khởi tạo giá trị j vào thanh ghi s2
# Cách 1:
blt s1, s2, then # if i < j then jump else
# Cách 2:
# slt t0, s2, s1 # set t0 = 1 if j < i else clear t0 = 0
# bne t0, zero, else # t0 != 0 means t0 = 1, jump else
then:
addi t1, t1, 1 # then part: x=x+1
addi t3, zero, 1 # z=1
j endif # skip else part
else:
addi t2, t2, -1 # begin else part: y=y-1
add t3, t3, t3 # z=2*z
endif:
```



Kết quả: $t1 = 1$, $t3 = 1$ (như mong muốn)

b) khởi tạo $i = 4$, $j = 4$. Do $i = j$ nên thực hiện câu lệnh then nên $t1 = 1$, $t3 = 1$

Code:

```
# Laboratory Exercise 3, Home Assignment 1
.text
start:
# TODO:
li s1, 4# Khởi tạo giá trị i vào thanh ghi s1
li s2, 4# Khởi tạo giá trị j vào thanh ghi s2
# Cách 1:
blt s1, s2, else # if i >= j then jump else
# Cách 2:
# slt t0, s2, s1 # set t0 = 1 if j < i else clear t0 = 0
# bne t0, zero, else # t0 != 0 means t0 = 1, jump else
then:
addi t1, t1, 1 # then part: x=x+1
addi t3, zero, 1 # z=1
j endif # skip else part
else:
addi t2, t2, -1 # begin else part: y=y-1
add t3, t3, t3 # z=2*z
endif:
```

Kết quả:

Text Segment

Addr	Code	Basic	Source
0x00400000	0x00300493	addi x9,x0,3	5: li s1, 3 # Map giá trị i = 3
0x00400004	0x00400913	addi x19,x0,4	6: li s2, 4 # Map giá trị j = 4
0x00400008	0x012489b3	add x19,x9,x18	7: add s3,s1,s2 # s3 = s1 + s2
0x0040000c	0x01304863	blt x0,x19,0x00000010	10: blt zero, s3, else # if i + j >= 0 th..
0x00400010	0x00130313	addi x6,x6,1	15: addi t1, t1, 1 # then part: x=x+1
0x00400014	0x00100e13	addi x28,x0,1	16: addi t3, zero, 1 # z=1
0x00400018	0x00c0006f	jal x0,0x0000000c	17: j endif # skip else part
0x0040001c	0xffff3839	addi x7,x7,0xffffffffff	19: addi t2, t2, -1 # begin else part: y=-1
0x00400020	0x01ce0e33	add x28,x28,x28	20: add t3, t3, t3 # z=2*z

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Registers

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7ffffefc
gp	3	0x10008000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0xffffffff
a0	8	0x00000000
a1	9	0x00000003
a0	10	0x00000000
a1	11	0x00000000
a2	12	0x00000000
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000004
s3	19	0x00000007
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
s12	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x00400028

Messages

```
-- program is finished running (dropped off bottom) --
```

Kết quả: đúng như mong muốn

- d) Nếu $m + n \geq i + j$ sẽ thực hiện else còn $i + j > m + n$ thực hiện then. Ta có $m + n = 3$, $i + j = 7$ nên sẽ thực hiện then

Laboratory Exercise 3, Home Assignment 1

.text

start:

TODO:

li s1, 3 # i = 3

li s2, 4 # j = 4

li a0, 1 # m = 1

li a1, 2 # n = 2

add s3, s1, s2 # s3 = s1 + s2

add s4, a0, a1 # s4 = m + n

Khởi tạo giá trị j vào thanh ghi s2

Cách 1:

bge s4, s3, else # if m + n < i + j then jump else

Cách 2:

slt t0, s2, s1 # set t0 = 1 if j < i else clear t0 = 0

bne t0, zero, else # t0 != 0 means t0 = 1, jump else

then:

addi t1, t1, 1 # then part: x=x+1

addi t3, zero, 1 # z=1

j endif # skip else part

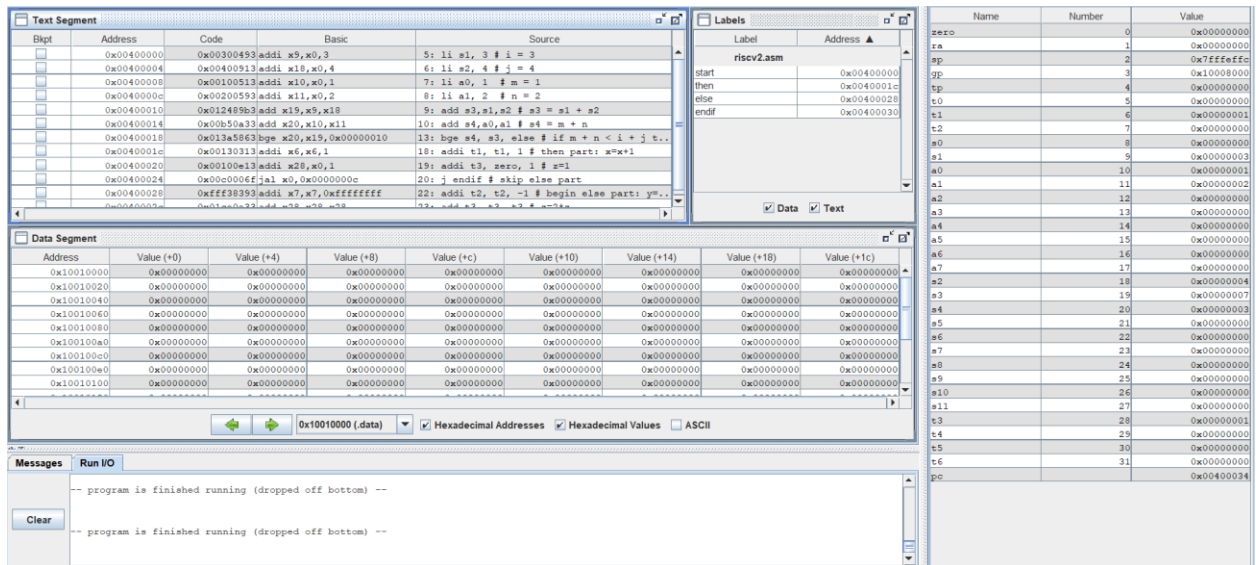
else:

addi t2, t2, -1 # begin else part: y=y-1

add t3, t3, t3 # z=2*z

endif:

Kết quả:



Assignment 5

a) $i > n$ hay $n < i$ thì chương trình kết thúc

Laboratory 3, Home Assignment 2

.data

A: .word 1, 3, 2, 5

.text

TODO: Khởi tạo giá trị các thanh ghi s2, s3, s4

li s1, 0 # i = 0

li s5, 0 # sum = 0

li s3, 9 # n = 4

la s2, A # Lấy địa chỉ của A trong vùng nhớ dữ liệu

li s4, 1 # s4 = 1

loop:

blt s3, s1, endloop # if i > n then end loop

add t1, s1, s1 # t1 = 2 * s1

add t1, t1, t1 # t1 = 4 * s1 => t1 = 4*i

add t1, t1, s2 # t1 store the address of A[i]

lw t0, 0(t1) # load value of A[i] in t0

add s5, s5, t0 # sum = sum + A[i]

add s1, s1, s4 # i = i + step

j loop # go to loop

endloop:

Text Segment						Name			Number			Value		
Byte	Address	Code	Basic	Source										
	0x00400000	0x00000493	addi s5,x0,0	6: li s1, 0 # i = 0		zero		0				0x00000000		
	0x00400004	0x00000a93	addi s21,x0,0	7: li s5, 0 # sum = 0		ra		1				0x00000000		
	0x00400008	0x00900993	addi x19,x0,9	8: li s3, 9 # n = 4		sp		2				0x7ffffefc		
	0x0040000c	0x0fc10917	auipc x18,0x0000fc10	9: la s2, A # Lấy địa chỉ của A trong vùng nhớ		gp		3				0x10008000		
	0x00400010	0xff490913	addi x18,x18,0xffffffff4			t0		4				0x00000000		
	0x00400014	0x00100a13	addi x20,x0,1	10: li s4, 1 # s4 = 1		t1		6				0x10010024		
	0x00400018	0x0299c063	blt x19,s5,0x00000020	12: blt s3, s1, endloop # if i > n then end loop		t2		7				0x00000000		
	0x0040001c	0x00948133	add x6,s5,s5	13: add t1, s1, s1 # t1 = 2 * s1		s0		8				0x00000000		
	0x00400020	0x00630333	add x6,s6,s6	14: add t1, t1, t1 # t1 = 4 * s1 => t1 = 4*i		s1		9				0x0000000a		
	0x00400024	0x01230333	lw x6,s6,s18	15: add t1, t1, s2 # t1 store the address of A[i]		s0		10				0x00000000		
	0x00400028	0x00032283	lw x5,0(x6)	16: lw t0, 0(t1) # load value of A[i] in t0		a1		11				0x00000000		
	0x00400030	0x008a8b33	add s31,s31,s6	17: add s5, s5, s0 # sum = sum + A[i]		a2		12				0x00000000		
						a3		13				0x00000000		
						a4		14				0x00000000		
						a5		15				0x00000000		
						a6		16				0x00000000		
						a7		17				0x00000000		
						a8		18				0x10010000		
						a9		19				0x00000000		
						a10		20				0x00000000		
						a11		21				0x0000000b		
						a12		22				0x00000000		
						a13		23				0x00000000		
						a14		24				0x00000000		
						a15		25				0x00000000		
						a16		26				0x00000000		
						a17		27				0x00000000		
						a18		28				0x00000000		
						a19		29				0x00000000		
						a20		30				0x00000000		
						a21		31				0x00000000		
						gp						0x0040003c		

Kết quả: tính tổng các phần tử mảng A (sum = 11)

b) sum < 0 thì chương trình kết thúc

Chương trình:

Laboratory 3, Home Assignment 2

.data

A: .word 1, 3, -5, 6

.text

TODO: Khởi tạo giá trị các thanh ghi s2, s3, s4

li s1, 0 # i = 0

li s5, 0 # sum = 0

li s3, 9 # n = 4

la s2, A # Lấy địa chỉ của A trong vùng nhớ dữ liệu

li s4, 1 # s4 = 1

loop:

blt s5, zero, endloop # if sum < 0 then end loop

add t1, s1, s1 # t1 = 2 * s1

add t1, t1, t1 # t1 = 4 * s1 => t1 = 4*i

add t1, t1, s2 # t1 store the address of A[i]

lw t0, 0(t1) # load value of A[i] in t0

add s5, s5, t0 # sum = sum + A[i]

add s1, s1, s4 # i = i + step

j loop # go to loop

endloop:

Text Segment					Name		
Offset	Address	Code	Basic	Source		Number	Value
	0x00400000	0x00000493	addi x5,x0,0	6: li s1, 0 # i = 0	zero	0	0x00000000
	0x00400004	0x00000a93	addi s2,s0,0	7: li s5, 0 # sum = 0	ra	1	0x00000000
	0x00400008	0x00900953	addi x19,x0,9	8: li s3, 9 # n = 4	sp	2	0x7ffffffc
	0x0040000c	0x0fc10917	auipc w18,0x0000fc10	9: la s2, A # lấy địa chỉ của A trong vùng nhớ	gp	3	0x10008000
	0x00400010	0xff490913	addi x18,x18,0xffffffff		t0	5	0xffffffff
	0x00400014	0x00100a13	addi x20,x0,1	10: li s4, 1 # s4 = 1	t1	6	0x10010008
	0x00400018	0x020a0e63	blt x21,x0,0x00000020	12: blt s5, zero, endloop # if sum < 0 then end	t2	7	0x00000000
	0x0040001c	0x00948333	add x6,s5,s5	13: add t1, s1, s1 # t1 = 2 * s1	s0	8	0x00000000
	0x00400020	0x00630333	add x6,x6,s6	14: add t1, t1, t1 # t1 = 4 * s1 => t1 = 4 * i	s1	9	0x00000003
	0x00400024	0x01230333	add x6,x6,x18	15: add t1, t1, s2 # t1 store the address of A[i]	a0	10	0x00000000
	0x00400028	0x00032283	lw x5,0(x6)	16: lw t0, 0(t1) # load value of A[i] in t0	a1	11	0x00000000
	0x00400030	0x00848b3c	add x3,x3,x5	17: add s5, s5, x0 # sum = sum + A[i]	a2	12	0x00000000
Data Segment					a3	13	0x00000000
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	a4	14	0x00000000
0x10010000	0x00000001	0x00000003	0xffffffff	0x00000006	a5	15	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	a6	16	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	a7	17	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	a8	18	0x10010000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	a9	19	0x00000009
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	a4	20	0x00000001
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	a5	21	0xffffffff
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	a6	22	0x00000000
0x10010100	0x00000000	0x00000000	0x00000000	0x00000000	a7	23	0x00000000
0x10010120	0x00000000	0x00000000	0x00000000	0x00000000	a8	24	0x00000000
0x10010140	0x00000000	0x00000000	0x00000000	0x00000000	a9	25	0x00000000
0x10010160	0x00000000	0x00000000	0x00000000	0x00000000	a10	26	0x00000000
0x10010180	0x00000000	0x00000000	0x00000000	0x00000000	a11	27	0x00000000
0x100101a0	0x00000000	0x00000000	0x00000000	0x00000000	t3	28	0x00000000
0x100101c0	0x00000000	0x00000000	0x00000000	0x00000000	t4	29	0x00000000
0x100101e0	0x00000000	0x00000000	0x00000000	0x00000000	t5	30	0x00000000
0x10010200	0x00000000	0x00000000	0x00000000	0x00000000	t6	31	0x00000000
0x10010220	0x00000000	0x00000000	0x00000000	0x00000000	pc		0x0040003c

Kết quả: $sum = 1 + 3 - 5 = -1 < 0$ nên chương trình kết thúc.

c) $A[i] == 0$ chương trình kết thúc

Chương trình:

.data

A: .word 1, 3, 0, 5, 4, 7, 8, 9, 6

.text

li s1, 0 # i = 0

la s2, A # địa chỉ số đầu tiên

li s3, 9 # n = 9

li s4, 1 # step = 1

li s5, 0 # sum = 0

addi s3, s3, 1

loop:

add t1, s1, s1

add t1, t1, t1

add t1, t1, s2

lw t0, 0(t1)

```
beq t0, zero, endloop
```

```
add s5, s5, t0
```

```
add s1, s1, s4
```

```
j loop
```

```
endloop:
```

Assignment 6

Chương trình:

```
.data
```

```
A: .word -10 20 -30 4 0x80000000 -90 #Mang
```

```
n: .word 6 #so phan tu
```

```
.text
```

```
la t0, A
```

```
lw t1, n
```

```
li t2, 0 # i = 0
```

```
li t3, 1 # step = 1
```

```
li t4, 0 # max ||
```

```
loop:
```

```
bge t2, t1, endloop
```

```
add s0, t2, t2
```

```
add s0, s0, s0
```

```
add s0, s0, t0
```

```
li s3, 0x80000000
```

```
lw s1, 0(s0)
```

```
bge s1, zero, continue # neu >= 0 thi tiep tục
```

```
beq s1, s3, extra # neu gap 0x80000000 thi thoat luôn
```

sub s1, zero, s1 # doi dau

continue:

blt t4, s1, update #neu max < s1 thi cap nhat max

add t2, t2, t3 #tang i

j loop

update:

addi t4, s1, 0 #cap nhat max bang s1

add t2, t2, t3

j loop

endloop:

li a7, 1

mv a0, t4 # Đưa max vào a0

ecall # Gõ syscall để in

j end

extra:

li a7, 1

mv a0, s1

ecall

end:

Kết luận:

- Các lệnh Branch thực hiện nhảy khi thỏa mãn một điều kiện nào đó. Được sử dụng trong các cấu trúc if/else, các cấu trúc lặp.
- Các lệnh Jump thực hiện nhảy mà không cần điều kiện nào cả. Được sử dụng để gọi hàm.

