ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KỸ THUẬT MÁY TÍNH

PHẠM QUỐC CƯỜNG – 20521150

NGUYỄN VĂN CHỌN – 20521138

IT012.M11.CNCL

GIẢNG VIÊN HƯỚNG DẪN

BÙI PHÙNG HỮU ĐỨC

TP. HỒ CHÍ MINH, 2021

MỤC LỤC

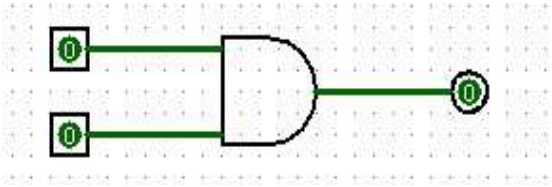
DANH MỤC HÌNH

DANH MỤC BẢNG

Chương 1. **LAB01**

1.1. Mô tả chức năng các công luận lí

1.1.1. Cổng AND

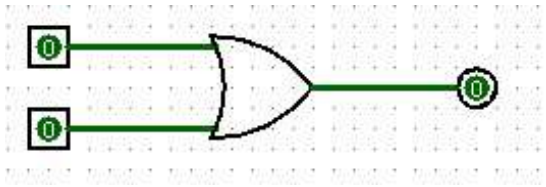


Hình 1.1-1 Cổng AND

Bảng 1.1-1 Bảng chân trị AND

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

1.1.2. Cổng OR

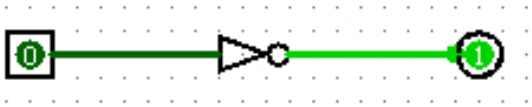


Hình 1.1-2 Cổng OR

Bảng 1.1-2 Bảng chân trị OR

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

1.1.3. Cổng NOT

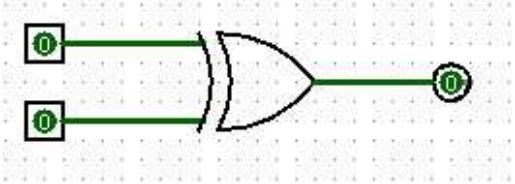


Hình 1.1-3 Cổng NOT

Bảng 1.1-3 Bảng chân trị NOT

A	F
0	1
1	0

1.1.4. Cổng XOR

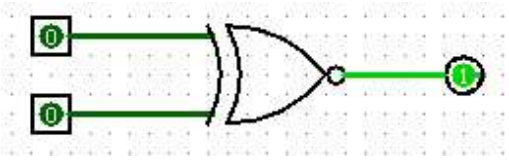


Hình 1.1-4 Cổng XOR

Bảng 1.1-4 Bảng chân trị XOR

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

1.1.5. Cổng XNOR

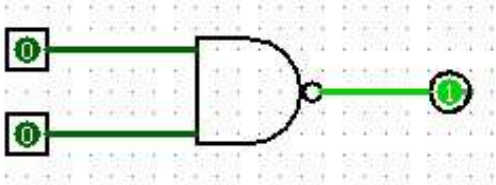


Hình 1.1-5 Cổng XNOR

Bảng 1.1-5 Bảng chân trị XNOR

A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

1.1.6. Cổng NAND

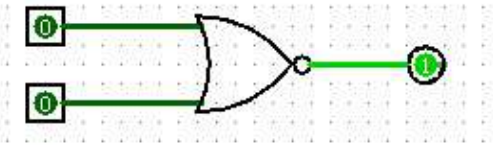


Hình 1.1-6 Cổng NAND

Bảng 1.1-6 Bảng chân trị NAND

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

1.1.7. Cổng NOR



Hình 1.1-7 Cổng NOR

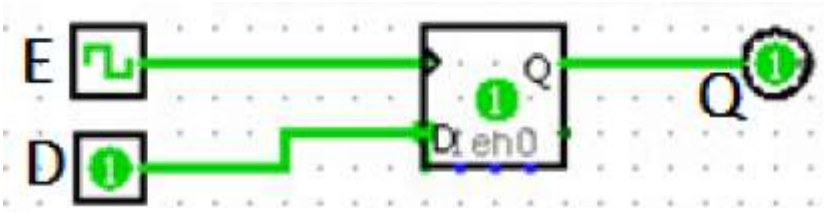
Bảng 1.1-7 Bảng chân trị NOR

A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

1.2. Mô phỏng các thiết bị lưu trữ

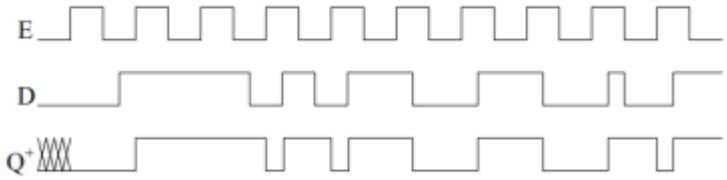
1.2.1. D latch

Selection: D Flip-Flop	
Trigger	High Level
Label	
Label Font	SansSerif Pla...



Hình 1.2-1 Mô phỏng D latch

Bảng 1.2-1 Bảng chân trị của latch



Hình 1.2-2 Minh hoạ dạng sóng latch

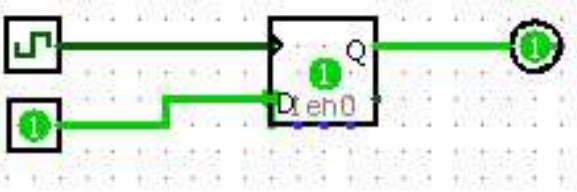
E	D	Q	Q+
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

E	Q+
0	Q
1	D

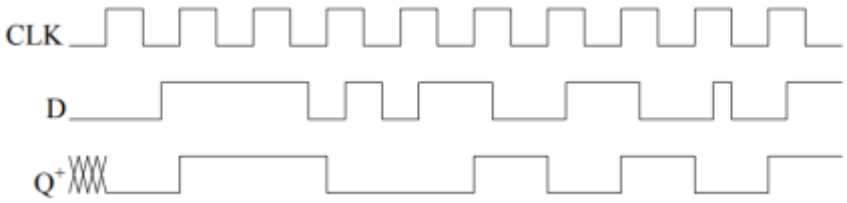
Latch là một thiết bị lưu trữ tích cực theo mức có khả năng lưu trữ 1 bit thông tin

1.2.2. D flipflop

Selection: D Flip-Flop	
Trigger	Rising Edge
Label	
Label Font	SansSerif Pla...



Hình 1.2-3 Mô phỏng D flipflop



Hình 1.2-4 Minh hoạ dạng sóng flipflop

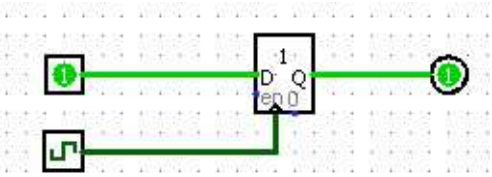
CLK	D	Q	Q+
-	0	0	0
-	0	1	1
-	1	0	0
-	1	1	1
↑	0	0	0
↑	0	1	0
↑	1	0	1
↑	1	1	1

CLK	Q+
-	Q
↑	D

Bảng 1.2-2 Bảng chân trị flipflop

Flipflop là một thiết bị lưu trữ tích cực theo cạnh có khả năng lưu trữ 1 bit thông tin

1.2.3. Thanh Ghi

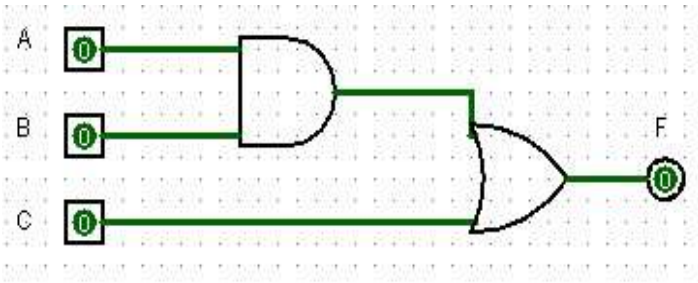


Hình 1.2-5 Mô phỏng thanh ghi

Thanh ghi là một thiết bị lưu trữ được cấu tạo bởi các flipflop nối chung ngõ vào CLK

1.3. Mô Phỏng mạch

1.3.1. Mô phỏng mạch tổ hợp

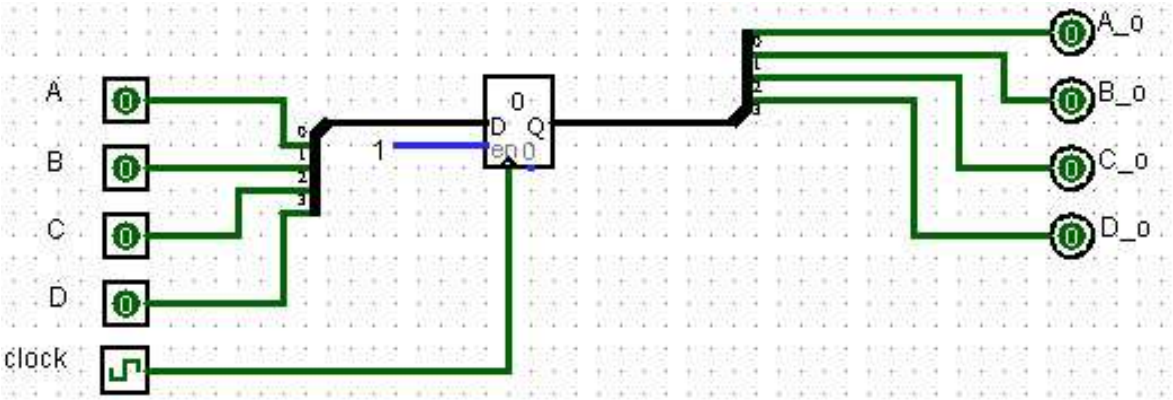


Hình 1.3-2 Mô phỏng mạch tổ hợp

Bảng 1.2-1 Bảng chân trị hình 1.3-1

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

1.3.2. Mô phỏng mạch tuần tự



Hình 1.3-3 Mô phỏng mạch tuần tự

1.4. Bài tập bổ sung Lab 01:

1.4.1. Trình bày ngắn gọn chức năng và nguyên lý hoạt động D-Flipflop, Thanh ghi

- D-Flipflop
  - Chức năng: Được sử dụng trong ứng dụng truyền dữ liệu song song.
  - Nguyên lí hoạt động:
    + Ngõ ra Q sẽ có cùng giá trị với ngõ vào D khi có tác động của cạnh xung clock.
    + Nếu ngõ vào không cho phép nạp (clock chuyển giá trị từ 0 sang 1), mạch giữ nguyên trạng thái, gõ ra Q sẽ có cùng giá trị với giá trị hiện tại Q^+.
    + Nếu ngõ vào cho phép nạp (clock chuyển giá trị từ 0 sang 1), Ngõ ra Q sẽ có cùng giá trị với ngõ vào D.
- Thanh ghi
  - Chức năng: Lưu trữ lệnh và dữ liệu cho hoạt động của CPU
  - Nguyên lí hoạt động:
    + Thanh ghi, trước hết được xoá (áp xung CLEAR) để đặt các ngõ ra về 0. Dữ liệu cần dịch chuyển được đưa vào ngõ D của tầng FF đầu tiên. Ở mỗi xung kích lên của đồng hồ clk, sẽ có 1 bit được dịch chuyển từ trái sang phải, nối tiếp từ tầng này qua tầng khác và đưa ra ở ngõ Q của tầng sau cùng.
    + Nếu tiếp tục có xung ck và không đưa thêm dữ liệu vào thì ngõ ra chỉ còn là 0. Do đó ta phải "hứng" hay ghim dữ liệu lại.

1.4.2. Phân biệt sự khác nhau giữa mạch tổ hợp và mạch tuần tự.

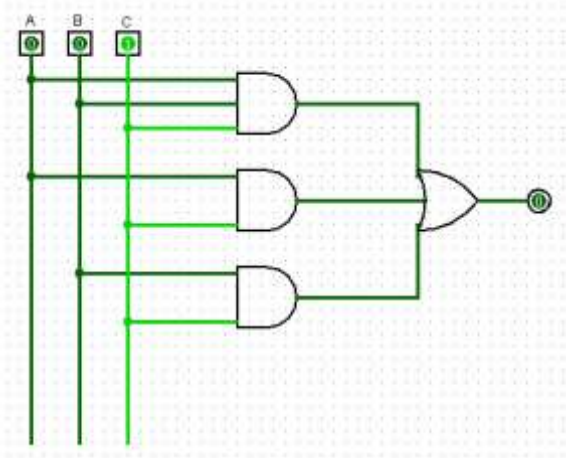
- Mạch tổ hợp:
  + Dùng những cổng luận lí kết nối lại với nhau.
  + Khi số lượng tín hiệu cần xử lí tăng lên dẫn đến tăng số lượng ngõ vào
  → Mạch trở nên phức tạp.
  + Không có thiết bị lưu trữ.
- Mạch tuần tự:
  + Có thiết bị lưu trữ.
  + Nhập và xuất tuần tự các tín hiệu
  + Sắp xếp thứ tự, chia ca cho mạch tổ hợp.
  + Giải quyết vấn đề hạn chế về mặt tài nguyên của mạch tổ hợp (số lượng ngõ vào và ngõ ra).

1.4.3. Clock (xung nhịp) CPU là gì, các trạng thái của clock.

- Trong kỹ thuật logic, người ta sử dụng tín hiệu dạng xung (có mức cao và mức thấp) để làm việc điều khiển đó. Tín hiệu này được gọi là clock (xung nhịp).
- Các trạng thái của clock:
  + Xung tích cực mức cao.
  + Xung tích cực mức thấp.
  + Xung tích cực cạnh lên.
  + Xung tích cực cạnh xuống.

1.4.4. Mô phỏng các mạch

(ABC+AC+AB+BC)C = ABC+AC+BC

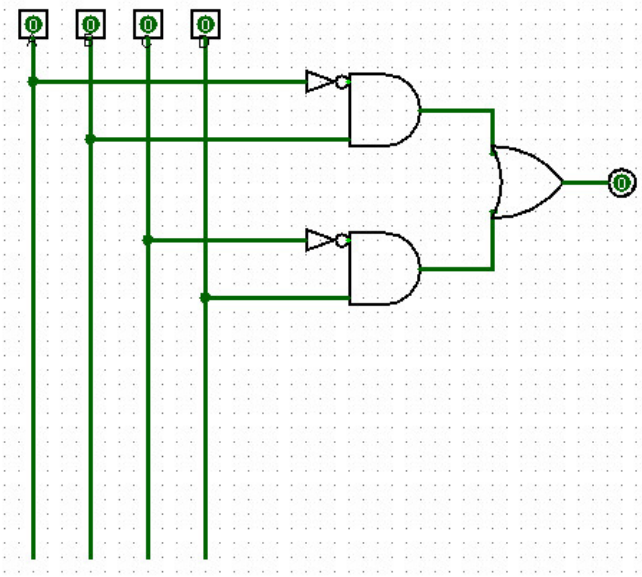


Hình 1.4-1 Mô phỏng mạch (ABC+AC+AB+BC)C

Bảng 1.4-1 Bảng chân trị mạch (ABC+AC+AB+BC)C

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

(AD+ABC+ABD+ACD) (¬A) + (¬A)B + (¬C)D = (¬A)B + (¬C)D

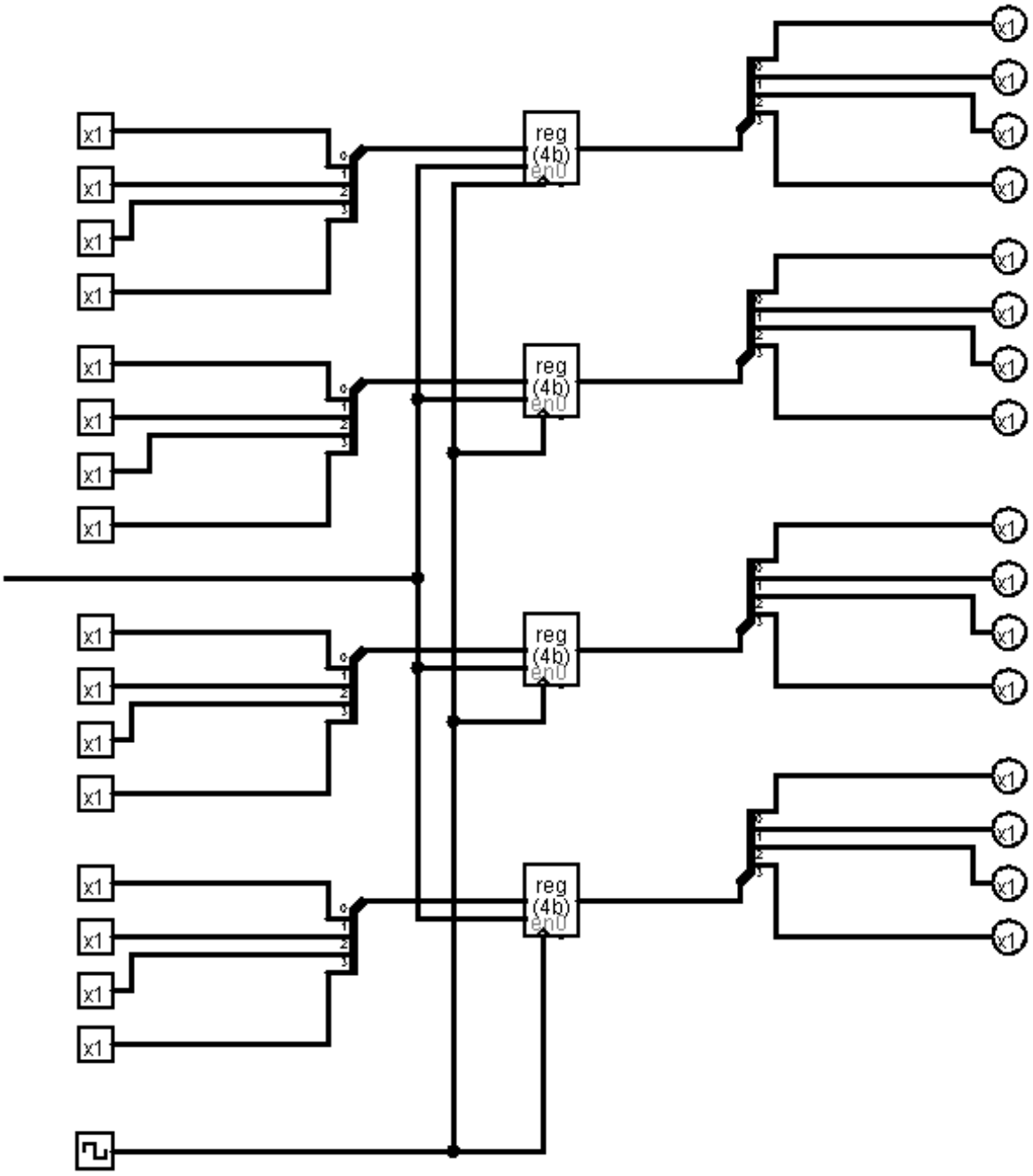


Hình 1.4-2 Mô phỏng mạch
(AD+ABC+ABD+ACD) (¬A) + (¬A)B + (¬C)D

Bảng 1.4-2 Bảng chân trị mạch
(AD+ABC+ABD+ACD) (¬A) + (¬A)B + (¬C)D

A	B	C	D	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0

1.4.5. Thiết kế lại thanh ghi ở bài tập 3.2 với 16 bit dữ liệu (dùng 4 thanh ghi 4 bit).

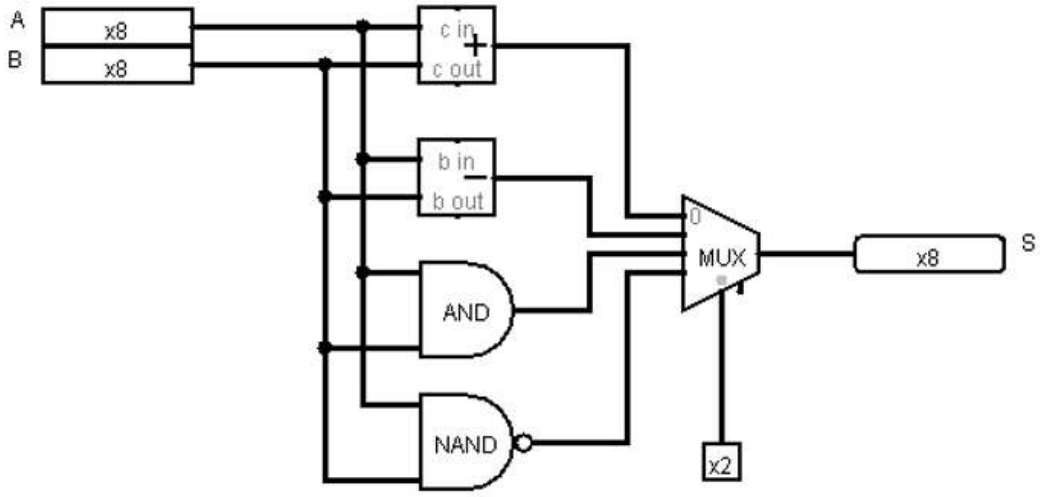


Hình 1.4-3 Mô phỏng thiết kế lại thanh ghi

Chương 2. LAB02

2.1. Mô phỏng ALU và Register Files

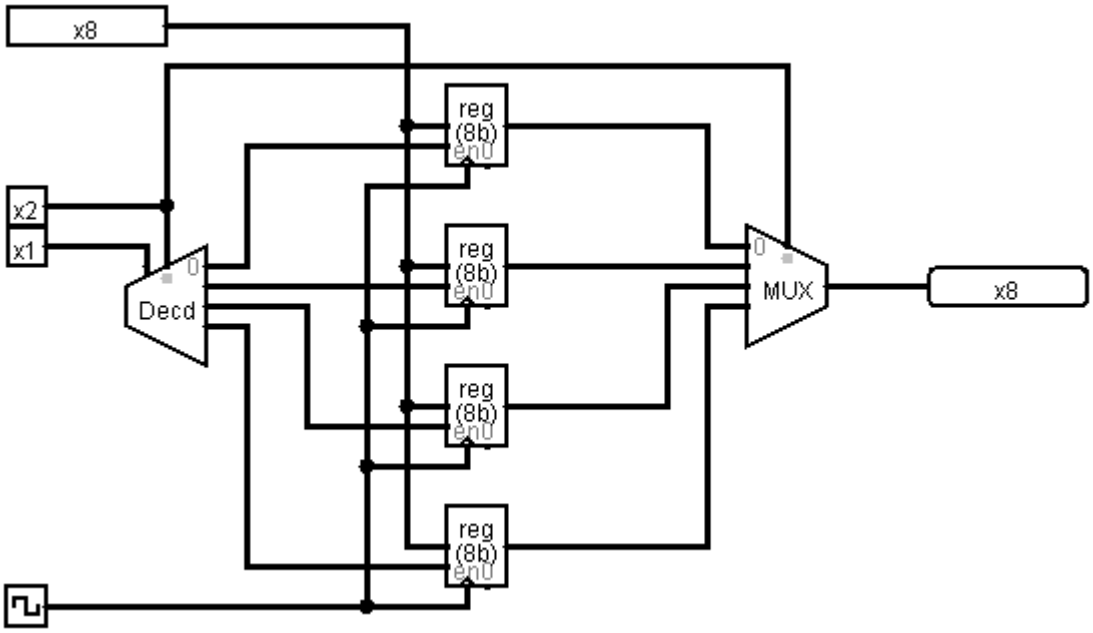
2.1.1. Mô phỏng ALU sau:



Hình 2.1-1 Mô phỏng ALU

Trong đó, tất cả các thiết bị đều có thuộc tính **Data Bits: 8**

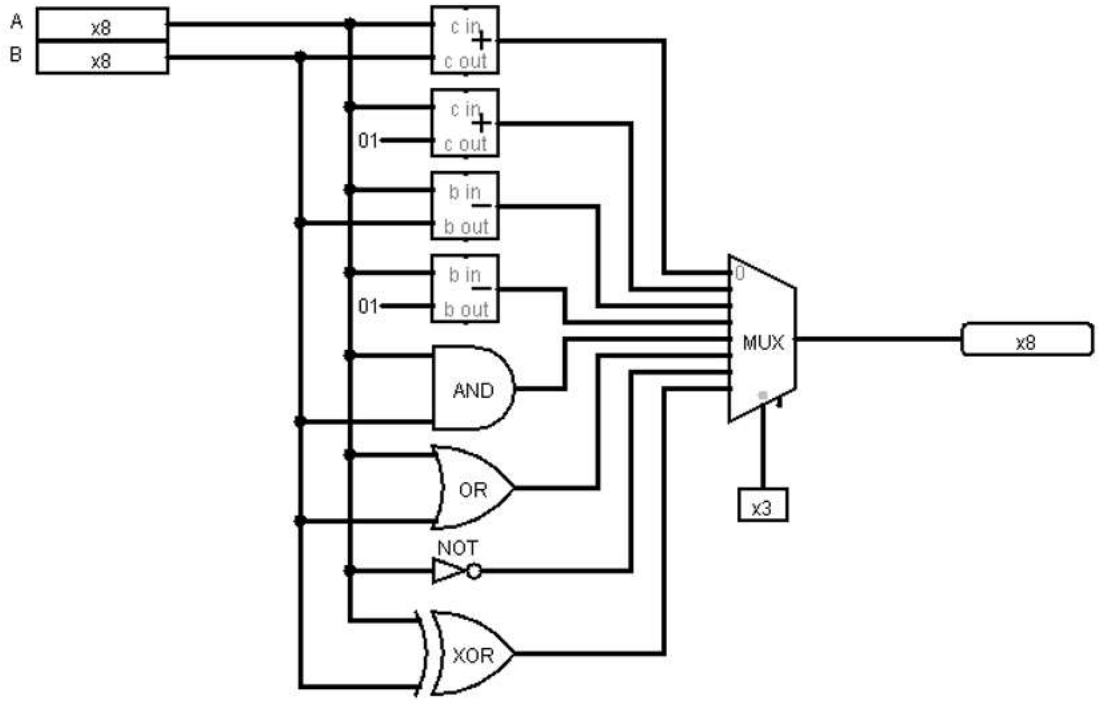
2.1.2. Mô phỏng Register Files gồm 4 thanh ghi 8 bit sau:



Hình 2.1-2 Mô phỏng Register Files

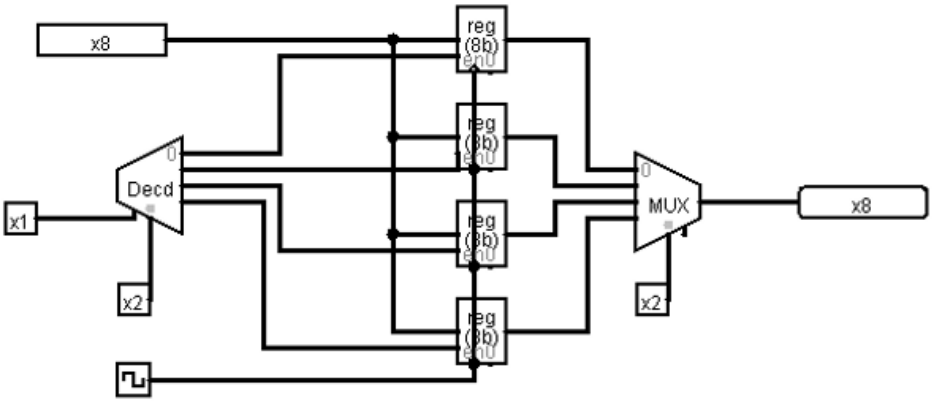
2.2. Cải tiến ALU, thiết kế và mô phỏng lại Register Files

2.2.1. Cải tiến ALU với các phép toán: A + B, A + 1, A – B, A – 1, A AND B, A OR B, NOT A, A XOR B



Hình 2.2-1 Cải tiến ALU với các phép toán: A + B, A + 1, A – B, A – 1, A AND B, A OR B, NOT A, A XOR B

2.2.2. Thiết kế và mô phỏng lại Register Files với địa chỉ xuất riêng với địa chỉ ghi?



Hình 2.2-2 Mô phỏng lại Register Files với địa chỉ xuất riêng với địa chỉ ghi

2.3. Bài tập bổ sung lab 02:

2.3.1. Phân biệt Mux và Demux? Thiết kế mux 4to1 và demux 2to4 bằng các cổng luận lý.

- Mux (Mạch dồn kênh, mạch ghép kênh):

+ Là một dạng mạch tổ hợp cho phép chọn 1 trong nhiều dường ngõ vào song song(các kênh vào) để đưa tới một ngõ ra(gọi là kênh truyền nối tiếp). Việc chọn đường nào trong các đường ngõ vào do các ngõ chọn quyết định.

+ MUX hoạt động như 1 công tắc nhiều vị trí được điều khiển bởi mã số. Mã số này là dạng số nhị phân, tuỳ tổ hợp số nhị phân này mà ở bất kì thời điểm nào chỉ có 1 ngõ vào được chọn và cho phép đưa tới ngõ ra.

+ Nếu có 2n ngõ vào song song thì phải cần n ngõ điều khiển chọn.

- Demux (Bộ chuyển mạch phân kênh, mạch tách kênh):

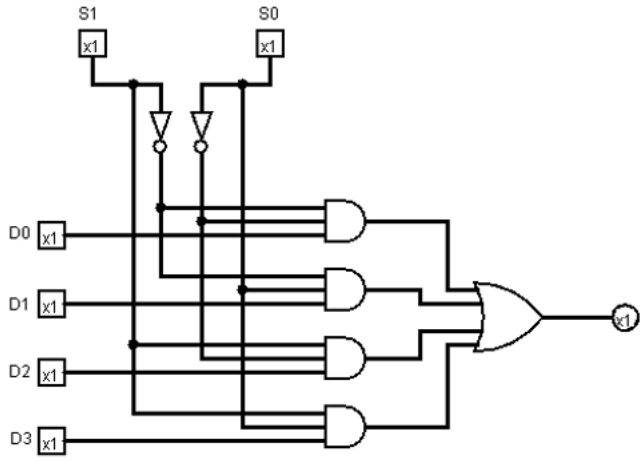
+ Có chức năng ngược lại với mạch dồn kênh tức là : tách kênh truyền thành 1 trong các kênh dữ liệu song song tuỳ vào mã chọn ngõ vào. Có thể xem mạch tách kênh giống như 1 công tắc cơ khí được điều khiển chuyển mạch bởi mã số. Tuỳ theo mã số được áp vào ngõ chọn mà dữ liệu từ 1 đường sẽ được đưa ra đường nào trong số các đường song song.

+ Mạch tách kênh hoạt động như mạch giải mã

+ Nhiều mạch tách kênh còn có chức năng như 1 mạch giải mã.

	MUX	DEMUX
Chức năng	Một mạch tổ hợp có chức năng lựa chọn một trong những ngõ vào dữ liệu để gửi tới một ngõ ra duy nhất dựa trên các ngõ vào điều khiển.	Tách kênh truyền thành 1 trong các kênh dữ liệu song song tuỳ vào mã chọn ngõ vào.
Số cổng	Có nhiều đầu vào và 1 đầu ra.	Có nhiều đầu ra.
Kết nối	Chuyển đổi song song với nối tiếp.	Kết nối nối tiếp sang song song.

Mux 4-1

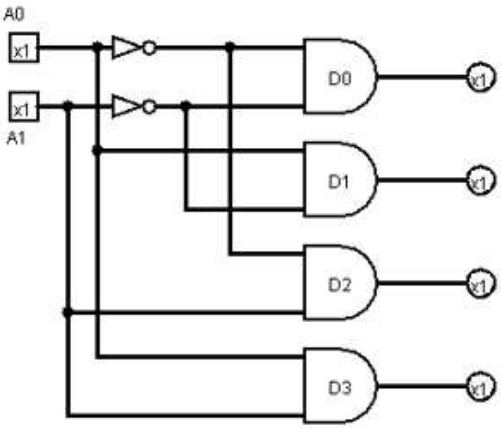


Hình 2.3-1 Mô phỏng MUX 4-1

Bảng 2.3-1 Bảng chân trị MUX 4-1

S1	S0	Y
0	0	D0
0	1	D1
1	0	D2
1	1	D3

Demux 2-4

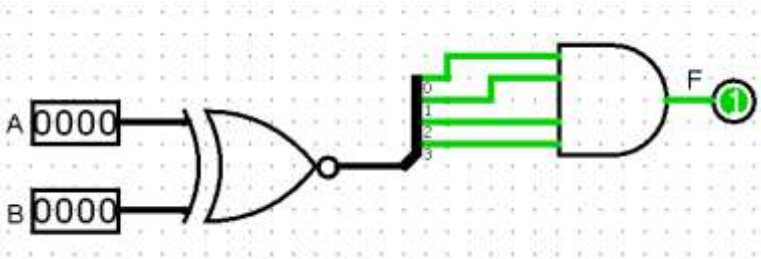


Hình 2.3-2 Mô phỏng Demux 2-4

Bảng 2.3-2 Bảng chân trị Demux 2-4

A1	A0	D3	D2	D1	D0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

2.3.2. Thiết kế mạch có chức năng so sánh hai input 4 bit có bằng nhau hay không? Trường hợp bằng nhau, output bằng 1 ngược lại output bằng 0

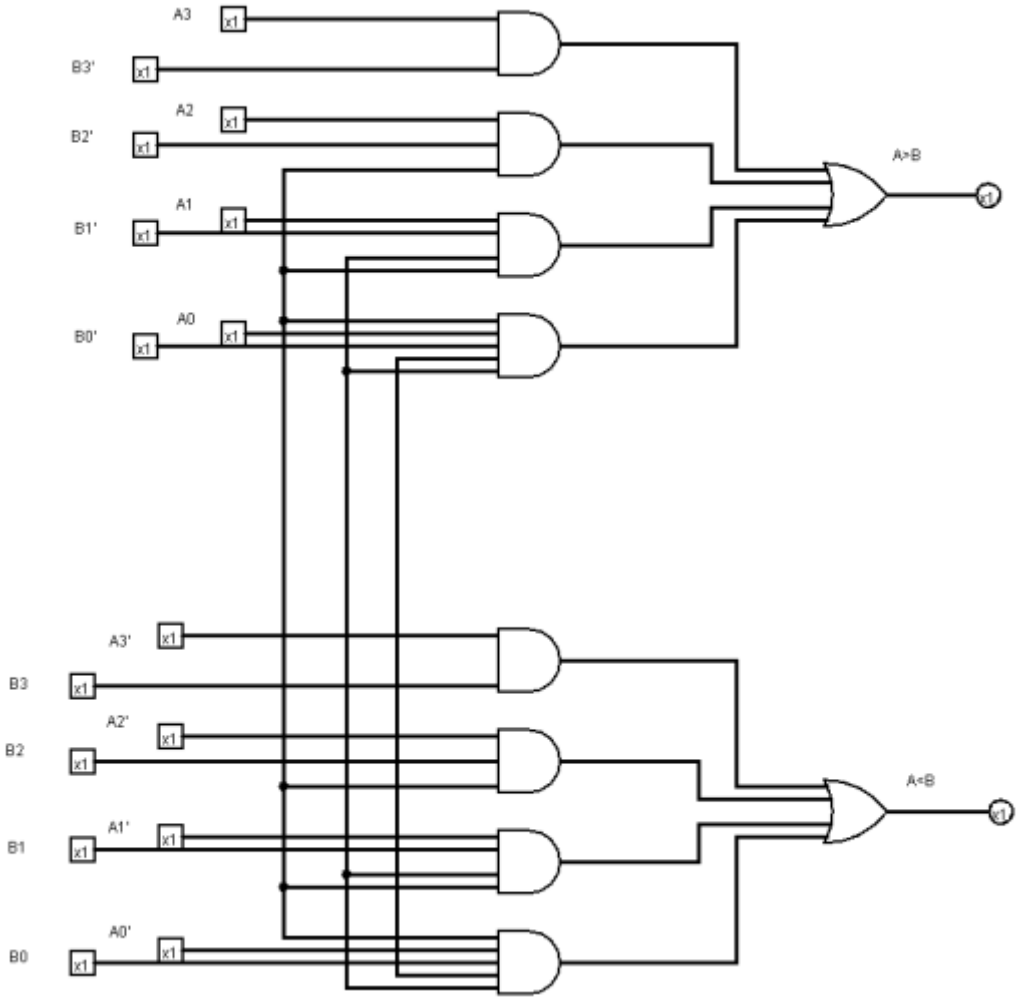


Hình 2.3-3 Mô phỏng mạch so sánh bằng

Bảng 2.3-3 Bảng chân trị mạch so sánh bằng

Inputs				Outputs
A_1	A_0	B_1	B_0	A=B
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

2.3.3. Mở rộng vấn đề ở câu c, sinh viên thiết kế mạch so sánh 2 số 4 bit trong các trường hợp A>B và A<B.



Hình 2.3-4 Mạch so sánh A>B, A<B

Bảng 2.3-4 Bảng chân trị mạch so sánh A>B, A<B

A1	A0	B1	B0	A > B	A < B
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	0	1

0	1	0	0	1	0
0	1	0	1	0	0
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	0	0
1	0	1	1	0	1
1	1	0	0	1	0
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	0	0

Chương 3. LAB 03

3.1. Mô phỏng thực thi các lệnh và chức năng của các lệnh cơ bản :

Bảng 3.1-1 Mô phỏng thực thi và chức năng các lệnh cơ bản

STT	Lệnh	Chức năng	VD
1	add	Cộng giá trị trong 2 thanh ghi và lưu vào 1 thanh ghi	Add $s0, $s1, $s2
2	addi	Cộng giá trị trong 1 thanh ghi với 1 hằng số rồi lưu vào 1 thanh ghi	Addi $s0, $s1, 5
3	addu	Cộng giá trị trong 1 thanh ghi với 1 số nguyên không dấu rồi lưu vào 1 thanh ghi	Addu $s0, $s1, $s2
4	addiu	Cộng giá trị trong 1 thanh ghi với 1 hằng số là số nguyên không dấu rồi lưu vào 1 thanh ghi	Addiu $s0, $s1, -5
5	sub	Trừ giá trị trong 2 thanh ghi và lưu vào 1 thanh ghi	Sub $s0, $s1, $s2
6	subu	Trừ giá trị trong 1 thanh ghi với 1 số nguyên không dấu rồi lưu vào 1 thanh ghi	Subu $s0, $s1, $s2
7	and	Thực hiện and từng bit giá trị của 2 thanh ghi với nhau, kết quả lưu vào 1 thanh ghi	And $s0, $s1, $s2
8	andi	Thực hiện and từng bit giá trị của 1 thanh ghi với 1 hằng số, kết quả lưu vào 1 thanh ghi	Andi $s0, $s1, 17

9	or	Thực hiện or từng bit giá trị của 2 thanh ghi với nhau và lưu vào 1 thanh ghi	Or $s0, $s1, $s2
10	nor	Thực hiện nor từng bit giá trị của 1 thanh ghi với 1 hằng số, kết quả lưu vào 1 thanh ghi	Nor $s0, $s1, $s2
11	lw	Lấy giá trị trong thanh ghi cộng với hằng số ta được địa chỉ của từ nhớ cần lấy dữ liệu và lưu dữ liệu này vào 1 thanh ghi	Lw $s0 4($s1)
12	sw	Lưu giá trị 1 thanh ghi vào từ nhớ có địa chỉ được tính bằng giá trị thanh ghi khác cộng với hằng số	Sw $s0 4($s1)
13	slt	Kiểm tra xem giá trị trong 1 thanh ghi có nhỏ hơn thanh ghi còn lại hay không, nếu nhỏ hơn thì thanh ghi đích nhận giá trị là 1, ngược lại thanh ghi đích sẽ nhận giá trị 0	Slt $s0, $s1, $s2
14	slti	So sánh giá trị một thanh ghi với hằng số, nếu giá trị trong thanh ghi nhỏ hơn hằng số thì thanh ghi đích nhận giá trị là 1, ngược lại thanh ghi đích sẽ nhận giá trị 0 (2 giá trị so sánh theo kiểu có dấu dạng bù 2)	Slti $s0, $s1, 5
15	sltu	Chức năng giống như slt. Nhưng việc kiểm tra giá trị thanh ghi này có nhỏ hơn thanh ghi kia hay không trong	Sltu $s0, $s1, $s2

		lệnh slt thực hiện trên số có dấu, còn trong sltu thực hiện trên số không dấu	
16	sltiu	So sánh giá trị một thanh ghi với hằng số, nếu giá trị trong thanh ghi nhỏ hơn hằng số thì thanh ghi đích nhận giá trị là 1, ngược lại thanh ghi đích sẽ nhận giá trị 0 (2 giá trị so sánh theo kiểu số không dấu)	Sltiu $s0, $s1, -5
17	syscall	Làm treo sự thực thi của chương trình và chuyển quyền điều khiển cho HĐH. Sau đó, HĐH sẽ xem giá trị thanh ghi $v0 để xác định xem chương trình muốn nó làm việc gì.	

3.2. Mô phỏng các chương trình bên dưới và có biết ý nghĩa của chương trình:

STT	VD	Giải thích

1	.data
	var1: .word 23
	.text
	__start:
	lw $t0, var1
	li $t1, 5
	sw $t1, var1
		Var1 là biến nguyên 4 byte, giá trị khởi tạo là 23
		Gán vào thanh ghi $t0 giá trị được lưu tại biến word var1
		Khởi tạo $t1 = 5
		Gán giá trị trong thanh ghi $t1 vào biến var1

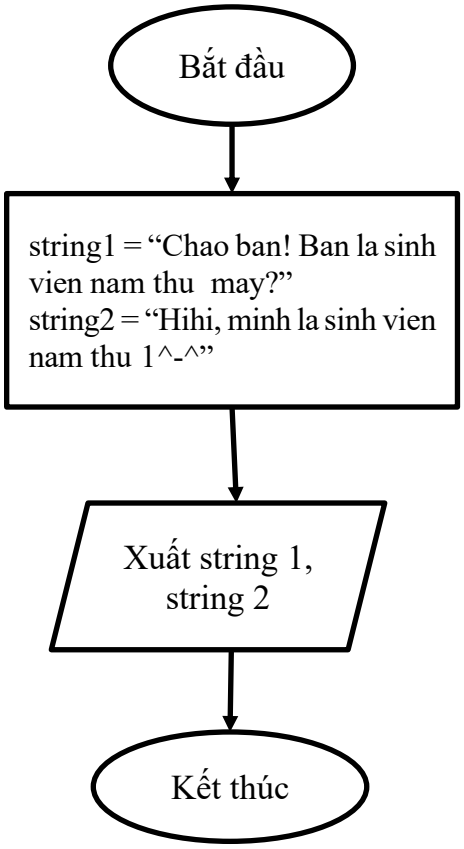
2	.data array1: .space 12	Khai báo mảng array1 có 12 byte (có thể lưu trữ 3 phần tử)

		.text	
	__start:	la $t0, array1	Load địa chỉ nền của array1 vào thanh ghi $t0
		li $t1, 5	Khởi tạo $t1 = 5
		sw $t1, ($t0)	array[0] = $t1 = 5
		li $t1, 13	Khởi tạo $t1 = 13
		sw $t1, 4($t0)	array[1] = $t1 = 13
		li $t1, -7	Khởi tạo $t1 = -7
		sw $t1, 8($t0)	array[2] = $t1 = -7
3		li $v0, 5 syscall	Syscall nhập một số nguyên
4	.data string1: .ascii “Print this.\n” .text __Main: li $v0, 4 la $a0, string1 syscall		Khai báo chuỗi với giá trị là “Print this.\n” Syscall xuất chuỗi Load địa chỉ của chuỗi string1 vào thanh ghi $a0 để in

3.3. Nhập vào một chuỗi, xuất ra cửa sổ I/O của MARS theo từng yêu cầu:

3.3.1. Khai báo và xuất ra cửa sổ I/O 2 chuỗi có giá trị như sau:

- Chuỗi 1: Chao ban! Ban la sinh vien nam thu may?
- Chuỗi 2: Hihi, minh la sinh vien nam thu 1^-^



Hình 3.3-1 Lưu đồ thuật toán xuất chuỗi

Code	Giải thích
.data	#Khai báo vùng nhớ data
string1: .asciiz "Chao ban! Ban la sinh vien nam thu may ?\n"	#Khai báo chuỗi string1
string2: .asciiz "Hihi, minh la sinh vien nam thu 1 ^-^"	#Khai báo chuỗi string2
.text	# Khai báo vùng nhớ chứa mã lệnh
main: li $v0,4	# Syscall để in chuỗi
la $a0, string1	# Load địa chỉ string1 vào $a0 để in
syscall	
la $a0, string2	# Load địa chỉ string2 vào $a0 để in
syscall	

3.3.2. Biểu diễn nhị phân của 2 chuỗi trên dưới bộ nhớ:

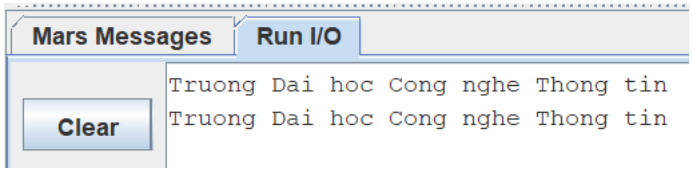
1111110111100101100001	\0 ? ya
11011010010000001110101101000	m uh
11101000010000001101101100001	t ma
11011100010000001101111001100101	n ne
11010010111011000100000001101000	iv h
11011100110100101110011001000000	nis
11000010110110000100000001101110	al n
11000010100001000100000000100001	aB !
11011100110000101100010001000000	nab
11011110110000101101000010000011	oahC
1011110	^
10110101011110001000000001100011	^-
10000001110101011010000011101000	uht
10000001101101011000010110111 0	man
10000001101110011001010110100 1	nei
11101100010000001101000011011 10	v hn
11010010111001100100000011000 01	is a
11011000010000001101000011011 10	l hn
11010010110110100100000000101100	im ,
11010010110100001101001010010 00	Hihi

3.3.3. Xuất ra lại đúng chuỗi đã nhập

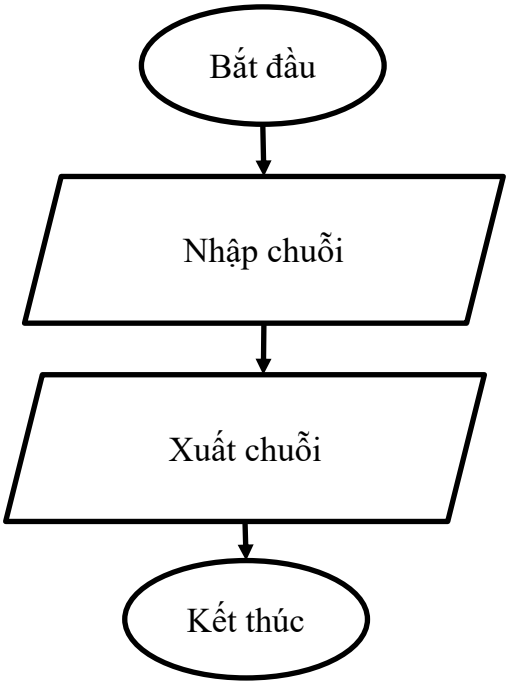
Ví dụ:

Nhap: Truong Dai hoc Cong nghe Thong tin

Xuất: Truong Dai hoc Cong nghe Thong tin



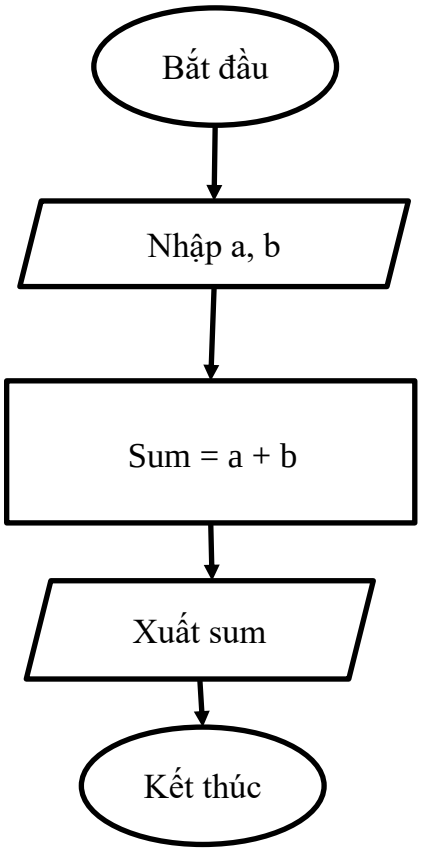
Hình 3.3-2 Kết quả thực thi nhập/ xuất chuỗi



Hình 3.3-3 Lưu đồ nhập/ xuất chuỗi

Code	Giải thích
.data	#Khai báo vùng nhớ data
string1: .asciiz	#Khai báo chuỗi string1
.text	# Khai báo vùng nhớ chứa mã lệnh
main: li $v0, 8	# Syscall để nhập chuỗi
la $a0, string1	# Load địa chỉ string1 vào $a0 để nhập
li $a1, 100	# Giới hạn 100 kí tự nhập
syscall	
li $v0, 4	# syscall để xuất chuỗi
la $a0, string1	# Load địa chỉ string1 vào $a0 để xuất
syscall	

3.3.4. Nhập vào 2 số nguyên sau đó xuất tổng của 2 số



Hình 3.3-4 Lưu đồ tính tổng 2 số nguyên

Code		Giải thích
	.data	#Khai báo vùng nhớ data
	.text	# Khai báo vùng nhớ chứa mã lệnh
main:	li $v0, 5	#Syscall để nhập số nguyên
	syscall	
	move $a1, $v0	# $a1 = $v0
	li $v0, 5	#Syscall để nhập số nguyên
	syscall	
	move $a2, $v0	# $s2 = $v0
	li $v0, 1	# Syscall để xuất số nguyên
	add $a0, $a1, $a2	# $a0 = $a1 + $a2
	syscall	

3.4. Bài tập bổ sung lab 03:

3.4.1. Assembly là gì? Trình bày các quá trình một chương trình viết bằng ngôn ngữ C/C++ được thực hiện trên máy tính?

Assembly là một loại ngôn ngữ lập trình cấp thấp cho bộ vi xử lý và các thiết bị có thể lập trình khác.
Quá trình được chia ra làm 4 giai đoạn chính:

- Giai đoàn tiền xử lý (Pre-processor)
- Giai đoạn dịch NNBC sang Asembly (Compiler)
- Giai đoạn dịch asembly sang ngôn ngữ máy (Asember)
- Giai đoạn liên kết (Linker)

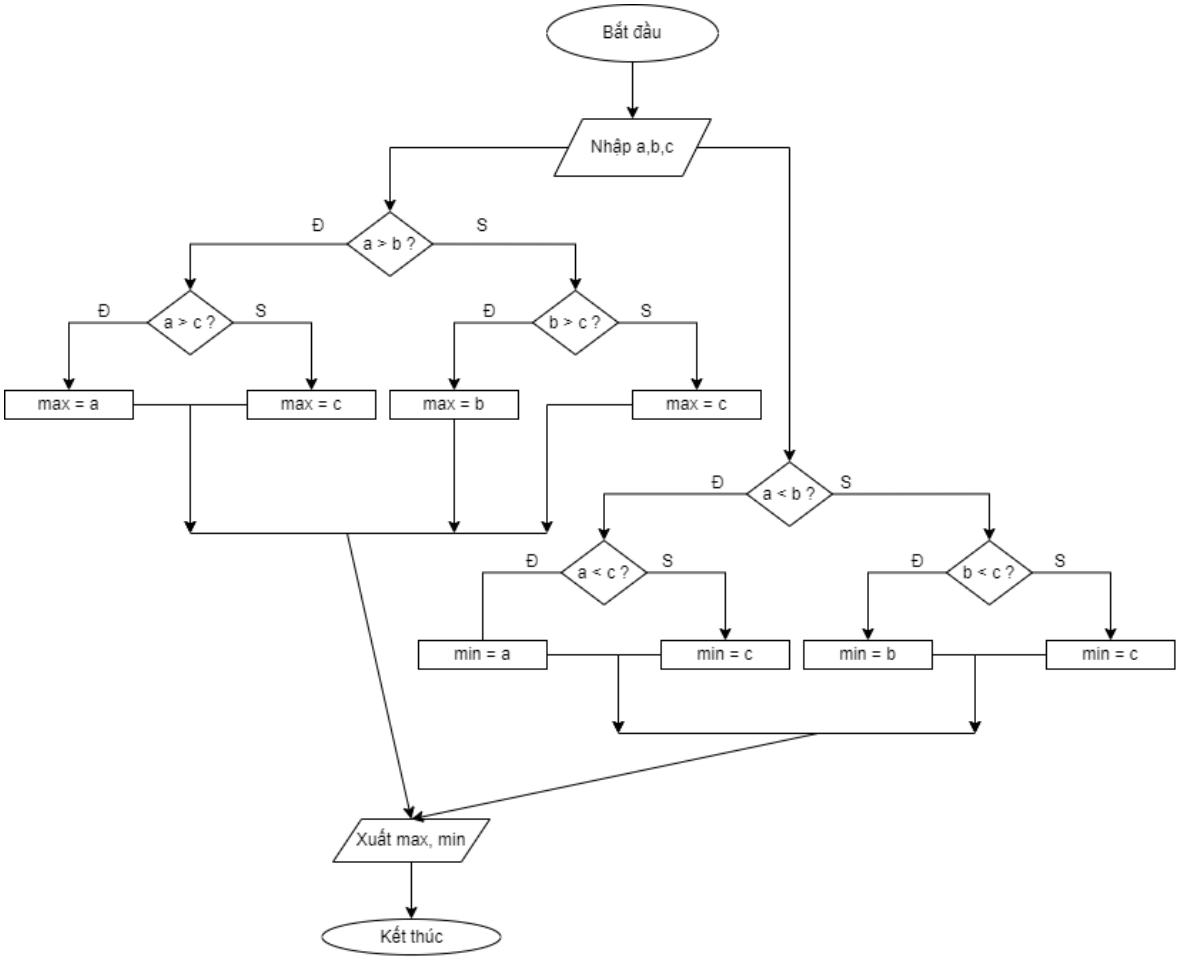
3.4.2. Trình bày các kiểu dữ liệu trong MIPS32 và kích thước của từng kiểu dữ liệu.

Các kiểu dữ liệu	Kích thước
.word	32-bit
.half	16-bit
.byte	8-bit
.ascii	ASCII
.asciiz	ASCII
.space	n-byte
.align	2^n byte

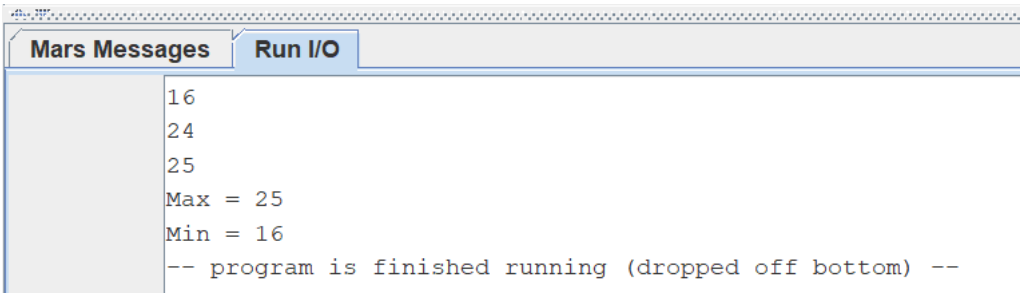
3.4.3. Trình bày cấu trúc bộ nhớ của một chương trình C++(layout memory).

1. Text segment: vùng chứa mã lệnh
2. Initialized data segment: chứa các biến toàn cục, biến static, hằng số,…
3. Uninitialized data segment: chứa các biến toàn cục, biến static, hằng số…
4. Stack: là một phần bộ nhớ đặc biệt được quản lý một cách tự động
5. Heap: vùng nhớ mặc định

3.4.4. Viết chương trình hợp ngữ nhập vào ba số a,b,c. Kiểm tra và in ra số lớn nhất, số bé nhất(không dùng vòng lặp)



Hình 3.4-1 Lưu đồ tìm max, min giữa 3 số



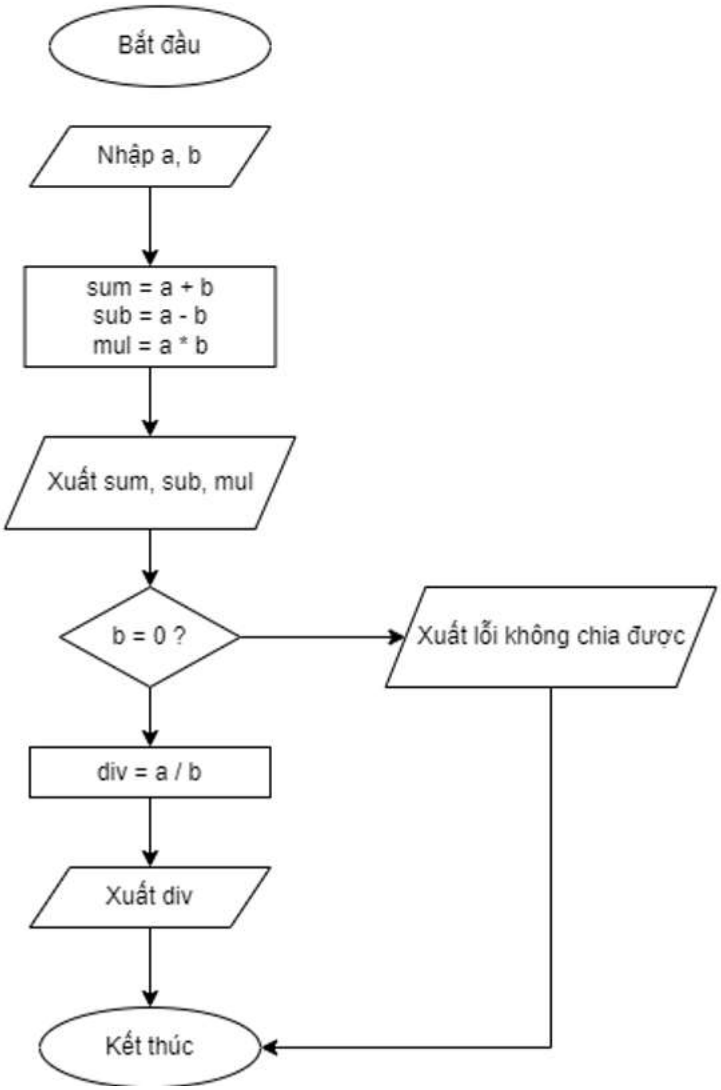
Hình 3.4-2 Minh hoạ kết quả thực thi tìm max, min giữa 3 số

Code	Giải thích
.data	#Khai báo vùng nhớ data
StringMax: .asciiz "Max = "	#Khai báo chuỗi Stringmax
StringMin: .asciiz "\nMin = "	#Khai báo chuỗi Stringmax
.text	# Khai báo vùng nhớ chứa mã lệnh
# a: $t0 b: $t1 c: $t2 max: $t3 min: $t4	
li $v0, 5	#Syscall để nhập số nguyên
syscall	
move $t0, $v0	# $t0 = $v0 (Nhập vào a)
li $v0, 5	# Syscall để nhập số nguyên
syscall	
move $t1, $v0	# $t1 = $v0 (Nhập vào b)
li $v0, 5	# Syscall để nhập số nguyên
syscall	
move $t2, $v0	# $t2 = $v0 (Nhập vào c)
j TimMax	# Nhảy đến label TimMax
TimMax:	
blt $t1, $t0, isAMoreThanC	# If (b < a) => nhảy label isAMoreThanC
j isCMoreThanB	# Else => nhảy label isCMoreThanB
isAMoreThanC:	
move $t3, $t0	# Max = a
blt $t2, $t0, inMax	# If (c < a) => nhảy label inMax
move $t3, $t2	# Else => Max = c
j inMax	# Nhảy label inMax
isCMoreThanB:	
move $t3, $t2	# Max = c
blt $t1, $t2, inMax	# If (b < c) => nhảy inMax
move $t3, $t1	# Max = b

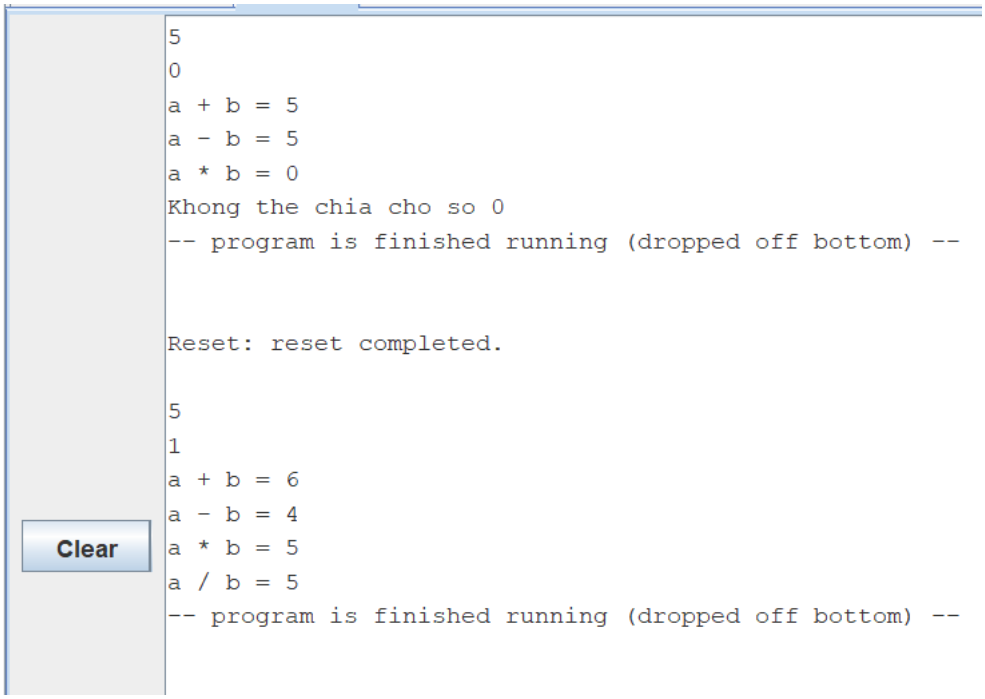
```
            j inMax                     # Nhảy label inMax
inMax:
        li $v0, 4                       # Syscall in string
        la $a0, StringMax               # Load địa chỉ StringMax vào $a0 để in
        syscall
        li $v0, 1                       # Syscall in giá trị max (int)
        move $a0, $t3                   # $a0 = $t3 ($t3 = max)
        syscall
        j TimMin
TimMin:                                 # Nhảy label TimMin
        blt $t0, $t1, isALessThanC
        j isBLessThanC                  # If(a < b) => jump isALessThanC
        isALessThanC:                   # Else => jump isBLessThanC
            move $t4, $t0
            blt $t0, $t2, inMin         # Min = a
            move $t4, $t2               # If(a < c) => jump inMin
            j inMin                     # Else => Min = c
        isBLessThanC:                   # jump inMin
            move $t4, $t1
            blt $t1, $t2, inMin         # Min = b
            move $t5, $t2               # If (b < c) => jump inMin
            j inMin                     # Min = c
inMin:
        li $v0, 4
        la $a0, StringMin               # Syscall in chuỗi
        syscall                         # Load địa chỉ StringMin vào $a0 để in
        li $v0, 1
        move $a0, $t4                   # Syscall in min (int)
        syscall                         # In ra giá trị $t4 (min)
```

j endProgram endProgram:	# Nhảy đến nhãn kết thúc

3.4.5. Viết chương trình hợp ngữ nhập vào số nguyên a,b. In ra kết quả của phép cộng, trừ nhân, chia



Hình 3.4-3 Lưu đồ thuật toán tính tổng, hiệu, tích thương 2 số nguyên



Hình 3.4-4 Minh hoạ kết quả thực thi tính tổng, hiệu, tích, thương của 2 số

Code	Giải thích
.data	# Khai báo vùng nhớ data
inSum: .asciiz "a + b = "	# Khai báo chuỗi inSum
inSub: .asciiz "\na - b = "	# Khai báo chuỗi inSub
inMul: .asciiz "\na * b = "	# Khai báo chuỗi inMul
inDiv: .asciiz "\na / b = "	# Khai báo chuỗi inDiv
inLoi: .asciiz "\nKhong the chia cho so 0"	# Khai báo chuỗi chia lỗi
.text	# Khai báo vùng nhớ chứa mã lệnh
li $v0, 5	# Syscall nhập vào a
syscall	
move $t0, $v0	# $t0 = $v0 (a = $t0)
li $v0, 5	# Syscall nhập vào b
syscall	
move $t1, $v0	# $t1 = $v0 (b = $t1)

```
add $t2, $t0, $t1          # sum = a + b
sub $t3, $t0, $t1          # sub = a – b
mul $t4, $t0, $t1          # mul = a * b

li $v0, 4                  # Syscall in chuỗi inSum
la $a0, inSum
syscall
li $v0, 1                  # Syscall in giá trị sum
move $a0, $t2
syscall

li $v0, 4                  # Syscall in chuỗi inSub
la $a0, inSub
syscall
li $v0, 1                  # Syscall in giá trị sub
move $a0, $t3
syscall

li $v0, 4                  # Syscall in chuỗi inMul
la $a0, inMul
syscall
li $v0, 1                  # Syscall in giá trị mul
move $a0, $t4
syscall

beqz $t1, Loi              # If  (b == 0) => jump label Loi

li $v0, 4                  # Syscall in chuỗi inDiv
la $a0, inDiv
```

syscall li $v0, 1 div $a0, $t0, $t1 syscall j EndProgram Loi: li $v0, 4 la $a0, inLoi syscall EndProgram:	 # $a0 = a / b # jump label EndProgram # Syscall xuất string inLoi

3.4.6. Viết chương trình hợp ngữ in ra địa chỉ của chuỗi "Hello UIT" và biến var_a kiểu word có giá trị là 10 trong bộ nhớ ở dạng thập lục phân

Hình 3.4-5 Kết quả thực thi chương trình in địa chỉ chuỗi và biến

Code	Giải thích
.data String: .asciiz "Hello UIT" var_a: .word 10 .text li $v0, 34 la $t0, String move $a0, $t0 syscall	#Khai báo vùng nhớ data #Khai báo chuỗi String = "Hello UIT" #Khai báo biến var_a = 10 #Khai báo vùng nhớ chứa mã lệnh #Syscall để xuất địa chỉ chuỗi String

li $v0, 34 la $t0, var_a move $a0, $t0 syscall	#Syscall để xuất địa chỉ biến var_a

Chương 4. LAB 04

4.1. Chuyển đoạn code trong bảng theo sau sang MIPS và sử dụng MARS để kiểm tra lại kết quả:

```
if (i == j)
   f = g + h;
else
   f = g – h;
```

(Với giá trị của i, j, f, g, h lần lượt chứa trong các thanh ghi $s0, $s1, $s2, $t0, $t1)

```
int Sum = 0
for (int i = 1; i <=N; ++i){
   Sum = Sum + i;
}
```

(Với giá trị của i, N, Sum lần lượt chứa trong các thanh ghi $s0, $s1, $s2)

Code C++	Code MIPS

if (i == j)
f = g + h;
else
f = g – h;
	bne $s0, $s1, Else
	add $s2,$t0,$t1
	j Exit
	Else:
	sub $s2,$t0,$t1
	End:

int Sum = 0
for (int i = 1; i <=N; ++i){
Sum = Sum + i;
}
	li $s0, 1 # index = 1
	li $s2, 0 # Sum = 0
	Loop:
	bge $s0, $s1, End
	add $s2, $s2, $t0
	addi $s0, $s0, 1
	j Loop
	End:

4.2. Nhập vào một ký tự, xuất ra cửa sổ I/O của MARS theo từng yêu cầu sau:

✓ Ký tự liền trước và liền sau của ký tự nhập vào

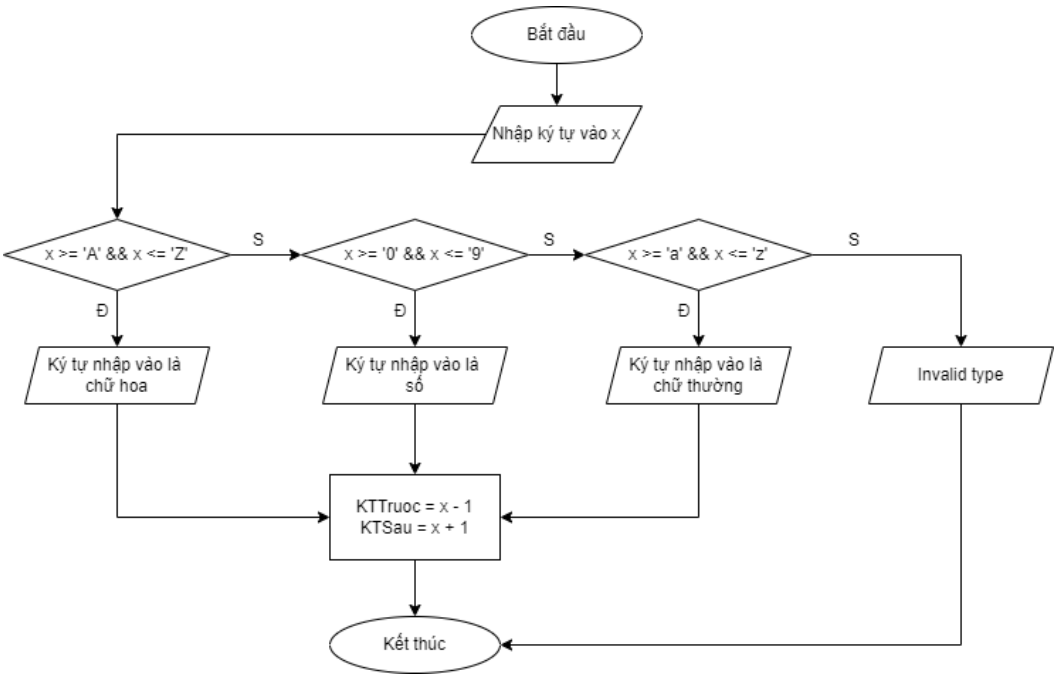
Ví dụ:

Nhap ky tu (chỉ một ký tự): b

Ky tu truoc: a

Ky tu sau: c

✓ Ký tự nhập vào chỉ được phép là ba loại: số, chữ thường và chữ hoa. Nếu ký tự nhập vào rơi vào một trong ba loại, xuất ra cửa sổ đó là loại nào; nếu ký tự nhập không rơi vào một trong ba loại trên, xuất ra thông báo “invalid type”



Hình 4.2-1 Lưu đồ kiểm tra loại ký tự

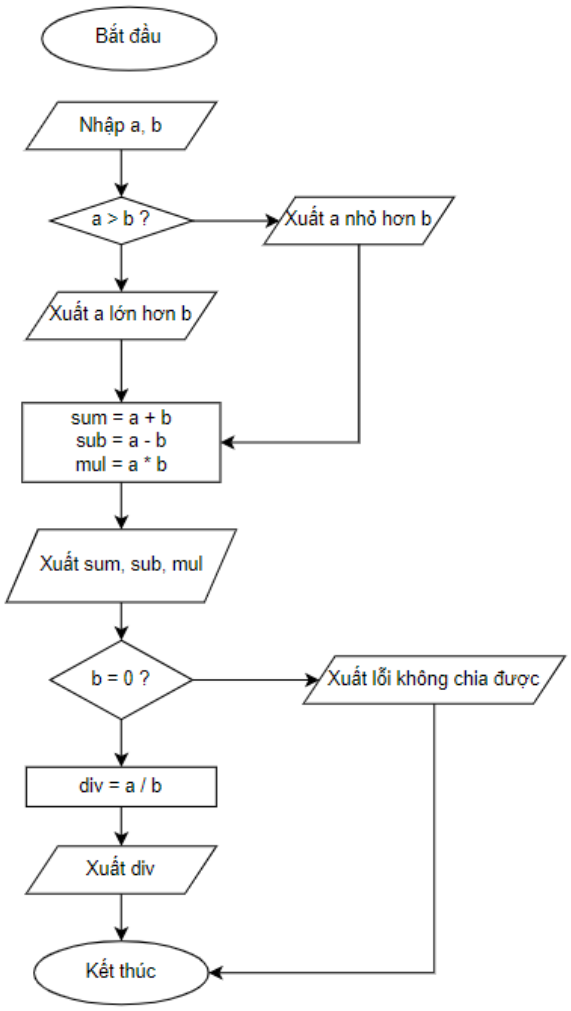
Hình 4.2-2 Kết quả thực thi chương trình kiểm tra loại ký tự

Code	Giải thích
.data string0: .asciiz "Nhap ky tu: " string1: .asciiz"\nKy tu nhap vao la chu hoa" string2: .asciiz"\nKy tu nhap vao la chu thuong" string3: .asciiz"\nKy tu nhap vao la chu so" string4: .ascitz"\nKy tu truoc: " string5: .asciiz"\nKy tu sau: " string6: .asciiz "\ninvalid type" .text li $v0, 4 la $a0, string0 syscall li $v0, 12 syscall move $a1, $v0 KTChuHoa: li $s1, 'A' li $s2, 'Z' bge $a1, $s1, KTChuHoa1 j KTSo KTChuHoa1: ble $a1,$s2,LaChuHoa j KTChuThuong LaChuHoa: li $v0, 4 la $a0, string1 syscall j Xuat KTChuThuong: li $s1, 'a' li $s2, 'z' bge $a1,$s1,KTChuThuong1 j KhongThuoc KTChuThuong1: ble $a1,$s2,LaChuThuong	#Khai báo vùng nhớ data # Khai báo vùng nhớ chứa mã lệnh # Gọi syscall xuất string # Gọi syscall nhập vào ký tự # $a1 = $v0 (Gán ký tự vừa nhập vào) # If (x >= ‘A’) => jump KTChuHoa1 # Else => jump KTSo # If (x <= ‘Z’) => jump LaChuHoa # Else => jump KTChuThuong # Gọi syscall in chuỗi là chữ hoa # jump Xuat # If (x >= ‘a’) => jump KTChuThuong1 # Else => jump KhongThuoc # If (x <= ‘z’) => jump LaChuThuong

```
                j KhongThuoc            # Else jump KhongThuoc
            LaChuThuong:
                li $v0, 4
                la $a0, string2         # Gọi syscall in ra chuỗi là chữ thường
                syscall
                j Xuat                  # jump Xuat
        KTSo:
            li $s1, '0'
            li $s2, '9'
            bge $a1, $s1, KTSo1         # If (x >= '0') => jump KTSo1
            j KhongThuoc                # Else => jump KhongThuoc
            KTSo1:
                ble $a1,$s2,LaSo        # If (x <= '9') => jump LaSo
                j KhongThuoc            # Else => jump KhongThuoc
            LaSo:  li $v0, 4
                la $a0, string3         # Gọi syscall in ra chuỗi là số
                syscall
                j Xuat                  # Jump Xuat
        KhongThuoc:
            li $v0, 4                   # Gọi syscall in chuỗi invaild type
            la $a0, string6
            syscall
            j EndProgram                # jump End

        Xuat:
        li $v0, 4                       # Gọi syscall in ra chuỗi KT đứng trước
        la $a0, string4
        syscall
        li $v0, 11                      # Gọi syscall in ra 1 ký tự
        subi $a0, $a1, 1                # In ra $a0 = x – 1
        syscall

        li $v0, 4                       # Gọi syscall in ra chuỗi KT đứng sau
        la $a0, string5
        syscall
        li $v0, 11                      # Gọi syscall in ra 1 ký tự
        addi $a0, $a1, 1                # In ra $a0 = x + 1
        syscall

EndProgram:
```

4.3. Nhập từ bàn phím 2 số nguyên, in ra cửa sổ I/O của MARS theo từng yêu cầu sau:

- Nhập 2 số nguyên, sau đó xuất ra số lớn hơn, tổng, hiệu, tích và thương



Hình 4.3-2 Lưu đồ tìm số lớn hơn và tính tổng, hiệu, tích, thương

```
5
1
a lon hon b
a + b = 6
a - b = 4
a * b = 5
a / b = 5
-- program is finished


Reset: reset completed.

0
5
a nho hon b
a + b = 5
a - b = -5
a * b = 0
a / b = 0
-- program is finished
```

Hình 4.3-1 Kết quả thực thi tìm số lớn hơn và tính tổng, hiệu, tích, thương

Code	Giải thích
.data	# Khai báo vùng nhớ data
inSum: .asciiz "\na + b = "	# Khai báo chuỗi inSum
inSub: .asciiz "\na - b = "	# Khai báo chuỗi inSub
inMul: .asciiz "\na * b = "	# Khai báo chuỗi inMul
inDiv: .asciiz "\na / b = "	# Khai báo chuỗi inDiv
inLoi: .asciiz "\nKhong the chia cho so 0"	# Khai báo chuỗi chia lỗi
inMax: .asciiz "a lon hon b"	# Khai báo chuỗi inMax
inMin: .asciiz "a nho hon b"	# Khai báo chuỗi inMin
.text	# Khai báo vùng nhớ chứa mã lệnh
li $v0, 5	# Syscall nhập vào a
syscall	
move $t0, $v0	# $t0 = $v0 (a = $t0)
li $v0, 5	# Syscall nhập vào b
syscall	
move $t1, $v0	# $t1 = $v0 (b = $t1)
bgt $t0, $t1, AMoreThanB	# If (a > b) => Jump AMoreThanB
li $v0, 4	# Else => In ra chuỗi a nhỏ hơn b
la $a0, inMin	
syscall	
j Calculation	# jump Calculation
AMoreThanB:	
li $v0, 4	# In ra chuỗi a lớn hơn b
la $a0, inMax	
syscall	
Calculation:	

```
add $t2, $t0, $t1
sub $t3, $t0, $t1
mul $t4, $t0, $t1

li $v0, 4
la $a0, inSum                # sum = a + b
syscall                      # sub = a – b
li $v0, 1                    # mul = a * b
move $a0, $t2
syscall                      # Syscall in chuỗi inSum

li $v0, 4
la $a0, inSub                # Syscall in giá trị sum
syscall
li $v0, 1
move $a0, $t3
syscall                      # Syscall in chuỗi inSub

li $v0, 4
la $a0, inMul                # Syscall in giá trị sub
syscall
li $v0, 1
move $a0, $t4
syscall                      # Syscall in chuỗi inMul

beqz $t1, Loi
                             # Syscall in giá trị mul
li $v0, 4
la $a0, inDiv
syscall
```

li $v0, 1 div $a0, $t0, $t1 syscall j EndProgram Loi: li $v0, 4 la $a0, inLoi syscall EndProgram:	# If (b == 0) => jump label Loi # Syscall in chuỗi inDiv # $a0 = a / b # jump label EndProgram # Syscall xuất string inLoi

4.4. Bài tập bổ sung lab 04:

4.4.1. Con trỏ là gì? Chức năng của con trỏ? Mảng là gì? Chức năng của mảng?

-Con trỏ là một đối tượng ngôn ngữ lập trình, mà giá trị nó chỉ tới giá trị khác được chứa nơi nào đó trong bộ nhớ máy tính sử dụng địa chỉ bộ nhớ

Chức năng:

- Tham chiếu tới địa chỉ bộ nhớ
- Thay đổi giá trị của biến

- Mảng (Array) là một tập hợp tuần tự các phần tử có cùng kiểu dữ liệu và các phần tử được lưu trữ trong một dãy các ô nhớ liên tục trên bộ nhớ.

Chức năng:

- Thêm
- Xóa
- Tìm kiếm
- …

4.4.2. Thủ tục là gì? Trình bày luồng hoạt động của một thủ tục trong MIPS

Phương pháp thủ tục chia một chương trình (chức năng) lớn thành các khối chức năng hay hàm (thủ tục) đủ nhỏ để dễ lập trình và kiểm tra.

Luồng hoạt động thủ tục (hàm con):

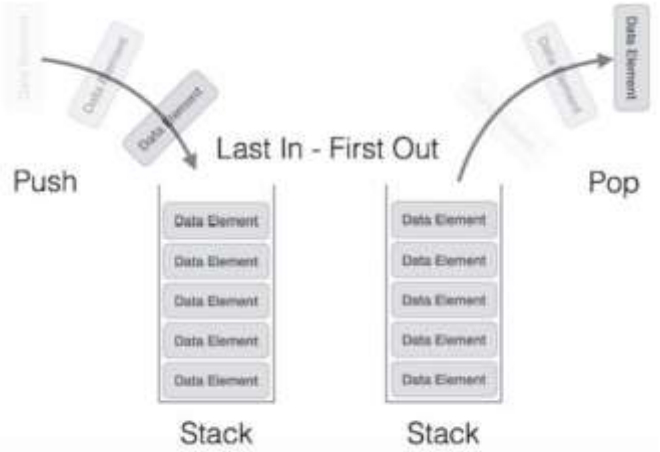
- Thanh ghi $at ($1), $k0 ($26), $k1 ($27) được dành cho hệ điều hành và assembler; không nên sử dụng trong lúc lập trình thông thường.
- Thanh ghi $a0-$a3 ($4-$7) được sử dụng để truyền bốn tham số đến một hàm con (nếu có hơn 4 tham số truyền vào, các tham số còn lại được lưu vào stack). Thanh ghi $v0-$v1 ($2-$3) được sử dụng để lưu giá trị trả về của hàm.
- Các thanh ghi $t0-$t9 ($9-$15, 24, 25) được sử dụng như các thanh ghi tạm. Các thanh ghi $s0-$s7 ($16-$23) được sử dụng như các thanh ghi lưu giá trị bền vững. (Trong MIPS, việc tính toán có thể cần một số thanh ghi trung gian, tạm thời, các thanh ghi $t nên được dùng cho mục đích này; còn kết quả cuối của phép toán nên lưu vào các thanh ghi $s)

Nếu quy ước này được tuân thủ, một hàm con nếu sử dụng bất kỳ thanh ghi loại $s nào sẽ phải lưu lại giá trị của thanh ghi $s đó trước khi thực thi hàm. Và trước khi thoát ra khỏi hàm, các giá trị cũ của các thanh ghi $s này cũng phải được trả về lại. Các thanh ghi $s này được xem như biến cục bộ của hàm

4.4.3. Ngăn xếp(stack là gì)? Trình bày cấu trúc của ngăn xếp và kể tên ứng dụng của ngăn xếp ?

Một ngăn xếp (Stack) là một cấu trúc dữ liệu trừu tượng hoạt động theo nguyên lý "vào sau ra trước" (Last In First Out (LIFO). Tức là, phần tử cuối cùng được chèn vào ngăn xếp sẽ là phần tử đầu tiên được lấy ra khỏi ngăn xếp.

Cấu trúc của stack:



Một số ứng dụng của stack

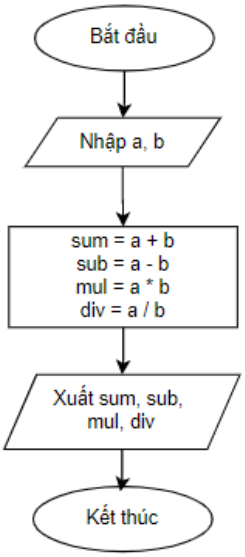
- Đảo ngược xâu
- Đổi một số n từ hệ thập phân sang hệ nhị phân

- Thao tác duyệt stack, tìm kiếm trên stack…..

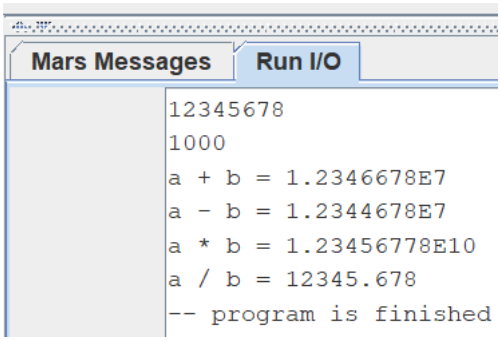
4.4.4. Viết chương trình hợp ngữ nhập vào số nguyên a,b. In ra kết quả của phép cộng, trừ nhân, chia.

```
Input : a = {nhập a}
b = {nhập b}

Output a + b = {Kết quả phép cộng}
a – b = {Kết quả phép trừ}
.
.
Program is finished …
```

Lưu ý: kết quả phép đúng với những phép tính có giá trị lớn



Hình 4.4-2 Lưu đồ tính tổng, hiệu, tích thương

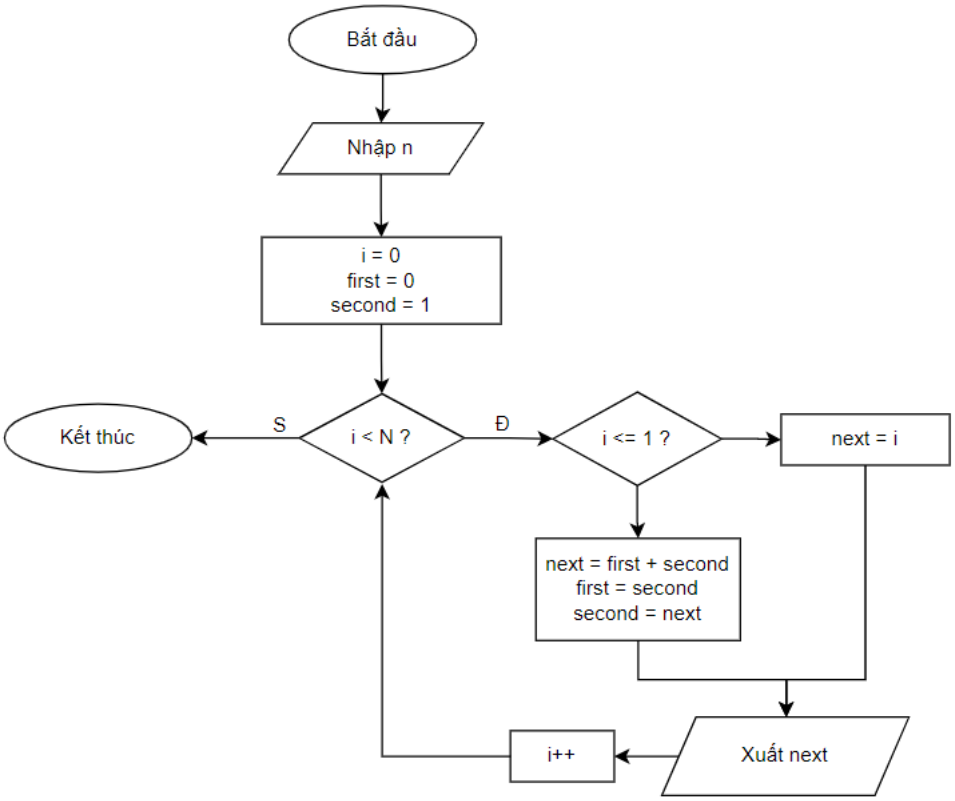


Hình 4.4-1 Kết quả thực thi tính tổng, hiệu, tích thương

Code	Giải thích
.data	# Khai báo vùng nhớ data
inSum: .asciiz "a + b = "	# Khai báo chuỗi inSum
inSub: .asciiz "\na - b = "	# Khai báo chuỗi inSub
inMul: .asciiz "\na * b = "	# Khai báo chuỗi inMul
inDiv: .asciiz "\na / b = "	# Khai báo chuỗi inDiv
.text	# Khai báo vùng nhớ chứa mã lệnh
li $v0, 6	# Syscall nhập vào a
syscall	
mov.s $f1, $f0	# $f1 = $f0 (a = $f1)
li $v0, 6	# Syscall nhập vào b
syscall	
mov.s $f2, $f0	# $f2 = $f0 (b = $f2)
add.s $f3, $f1, $f2	# sum = a + b
sub.s $f4, $f1, $f2	# sub = a – b
mul.s $f5, $f1, $f2	# mul = a * b
div.s $f6, $f1, $f2	# div = a / b
li $v0, 4	# Syscall in chuỗi inSum
la $a0, inSum	
syscall	
li $v0, 2	# Syscall in giá trị sum
mov.s $f12, $f3	
syscall	
li $v0, 4	# Syscall in chuỗi inSub
la $a0, inSub	
syscall	
li $v0, 2	# Syscall in giá trị sub
mov.s $f12, $f4	
syscall	
li $v0, 4	# Syscall in chuỗi inMul
la $a0, inMul	
syscall	
li $v0, 2	# Syscall in giá trị mul
mov.s $f12, $f5	
syscall	

li $v0, 4 la $a0, inDiv syscall li $v0, 2 mov.s $f12, $f6 syscall j EndProgram EndProgram:	# Syscall in chuỗi inDiv # In ra giá trị div # jump label EndProgram

4.4.5. Viết chương trình in ra N(N>2) số fibonaci đầu tiên.



Hình 4.4-3 Lưu đồ in dãy số Fibonacci

```
8
0 1 1 2 3 5 8 13
-- program is finished


Reset: reset completed

5
0 1 1 2 3
-- program is finished
```

Hình 4.4-4 Kết quả thực thi chương trình

Code	Giải thích
.data	#Khai báo vùng nhớ data
space: .asciiz " "	#Khai báo chuỗi space
.text	#Khai báo vùng nhớ chứa mã lệnh
li $v0, 5	
syscall	#Syscall để nhập số nguyên
move $s0, $v0	#$s0 = $v0
move $a1, $s0	#$a1 = $s0 (=N)
li $t1, 0	#index = 0
li $t2, 0	#first = 0
li $t3, 1	#second = 1
jal Fibonacci	#Fibonacci(N)
j End	
Fibonacci:	
blt $t1, $a1, Loop	#If (index < N) => jump Loop
jr $ra	#return
Loop:	
ble $t1, 1, LessOrEqual	#If (index == 1) => jump LessOrEqual
add $t4, $t2, $t3	#Else next = first + second
move $t2, $t3	#First = second
move $t3, $t4	#Second = next
j Print	#jump Print
LessOrEqual:	
move $t4, $t1	#Next = index
j Print	#jump Print
Print:	
li $v0, 1	
move $a0, $t4	
syscall	#Syscall in giá trị Next
li $v0, 4	
la $a0, space	
syscall	#Syscall in ra space
addi $t1, $t1, 1	#index++
j Fibonacci	
End:	

Chương 5. Lab 05

5.1. Thao tác với mảng

+ In ra cửa sổ I/O của MARS tất cả các phần tử của mảng array1 và array2.

Code	Giải thích
.data	#Khai báo vùng nhớ data
array1: .word 5,6,7,8,1,2,3,9,10,4	
size1: .word 10	
array2: .byte 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16	
size2: .word 16	
array3: .space 8	
size3: .word 8	
space:.asciiz " "	
newLine:.asciiz "\n"	
.text	#Khai báo vùng nhớ chứa mã lệnh
la $a1, array1	
lw $a2, size1	
li $t0, 0	#index = 0
jal PrintArray1	#PrintArray1(array1, size1)
li $v0, 4	
la $a0, newLine	
syscall	#syscall in dòng mới
la $a1, array2	
lw $a2, size2	
li $t0, 0	
jal PrintArray2	#PrintArray2(array2, size2)
j End	
PrintArray1:	
blt $t0, $a2, LoopPrintArray1	#If(index < size1) => LoopPrintArr
jr $ra	#Return
LoopPrintArray1:	
li $v0, 1	
lw $a0, 0($a1)	
syscall	
li $v0, 4	

la $a0, space
syscall
addi $a1, $a1, 4
addi $t0, $t0, 1
j PrintArray1
 	
PrintArray2:
blt $t0, $a2, LoopPrintArray2
jr $ra
LoopPrintArray2:
li $v0, 1
lb $a0, 0($a1)
syscall
li $v0, 4
la $a0, space
syscall
addi $a1, $a1, 1
addi $t0, $t0, 1
j PrintArray2
End:
	#Chuyển phần tử tiếp theo
	
	#If(index < size2) => LoopPrintArr
	#Return
	
	#Chuyển phần tử tiếp theo

+ Gán các giá trị cho mảng array3 sao cho

array3[i] = array2[i] + array2[size2 - 1 - i]

Code	Giải thích

.data
array1: .word 5, 6, 7, 8, 1, 2, 3, 9, 10, 4
size1: .word 10
array2: .byte 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16
size2: .word 16
array3: .space 8
size3: .word 8
.text
li $t0, 0
lw $s2, size2
lw $s4, size3
 	
j IfLoop
IfLoop:
	#Khai báo vùng nhớ data
	
	#Khai báo vùng nhớ chứa mã lệnh

blt $t0, $s4, Loop
j End
Loop:
la $s1, array2
la $s3, array3
add $s0, $s1, $t0
lb $t9, 0($s0)
 	
subi $t8, $s2, 1
sub $t8, $t8, $t0
add $s0, $s1, $t8
lb $t8, 0($s0)
 	
add $t9, $t8, $t9
sll $t1, $t0, 2
add $s3, $s3, $t1
sw $t9, 0($s3)
 	
addi $t0, $t0, 1
j IfLoop
End:
	# If(index < size3) => Loop
	
	#array2[i]
	
	#array2[size2-1-i]
	
	#array2[i] + array2[size2 - 1 - i]
	
	#Chuyển phần tử tiếp theo
	#array3[i] = ...
	
	#index++

+ Người sử dụng nhập vào mảng thứ mấy và chỉ số phần tử cần lấy trong mảng đó, chương trình xuất ra phần tử tương ứng.

Code	Giải thích
.data array1: .word 5, 6, 7, 8, 1, 2, 3, 9, 10, 4 size1: .word 10 array2: .byte 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 size2: .word 16 array3: .space 8 size3: .word 8 inputArray: .asciiz "Nhap mang can tim (chi nhap vao so): " inputPos: .asciiz "Nhap vao chi so: " error: .asciiz "Khong ton tai chi so nay"	#Khai báo vùng nhớ data

```
.text                                   #Khai báo vùng nhớ chứa mã lệnh
        li $v0, 4
        la $a0, inputArray
        syscall
        li $v0, 5
        syscall
        move $t0, $v0

        li $v0, 4
        la $a0, inputPos
        syscall
        li $v0, 5
        syscall
        move $t1, $v0

        beq $t0, 1, Array1
        beq $t0, 2, Array2
Array1:                                 #In array1
        la $s0, array1
        lw $s1, size1
        bge $t1, $s1, Error             # if(I > size) => Error
        sll $t1, $t1, 2
        add $s0, $s0, $t1               # arr1[i] = …
        lw $a0, 0($s0)
        li $v0, 1
        syscall
        j End
Array2:                                 #In array2
        la $s0, array2
        lw $s1, size2
        bge $t1, $s1, Error             # If(I > size) => Error
        add $s0, $s0, $t1               #arr2[i] = …
        lb $a0, 0($s0)
        li $v0, 1
        syscall
        j End
Error:                                  #In lỗi
        li $v0, 4
        la $a0, error
        syscall
        j End
End:
```

5.2. Thao tác với con trỏ

+ In ra cửa sổ I/O của MARS tất cả các phần tử của mảng array1 và array2.

Code	Giải thích
.data	#Khai báo vùng nhớ data
array1: .word 5,6,7,8,1,2,3,9,10,4	
size1: .word 10	
array2: .byte 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16	
size2: .word 16	
array3: .space 8	
size3: .word 8	
space:.asciiz " "	
newLine:.asciiz "\n"	
.text	#Khai báo vùng nhớ chứa mã lệnh
la $a1, array1	
lw $a2, size1	
li $t0, 0	#index = 0
jal PrintArray1	#PrintArray1(array1, size1)
li $v0, 4	
la $a0, newLine	
syscall	#syscall in dòng mới
la $a1, array2	
lw $a2, size2	
li $t0, 0	
jal PrintArray2	#PrintArray2(array2, size2)
j End	
PrintArray1:	
blt $t0, $a2, LoopPrintArray1	#If(index < size1) => LoopPrintArr
jr $ra	#Return
LoopPrintArray1:	
li $v0, 1	
lw $a0, 0($a1)	
syscall	
li $v0, 4	
la $a0, space	
syscall	
addi $a1, $a1, 4	#Chuyển phần tử tiếp theo
addi $t0, $t0, 1	

j PrintArray1
 	
PrintArray2:
blt $t0, $a2, LoopPrintArray2
jr $ra
LoopPrintArray2:
li $v0, 1
lb $a0, 0($a1)
syscall
li $v0, 4
la $a0, space
syscall
addi $a1, $a1, 1
addi $t0, $t0, 1
j PrintArray2
End:
	
	#If(index < size2) => LoopPrintArr
	#Return
	
	#Chuyển phần tử tiếp theo

+ Gán các giá trị cho mảng array3 sao cho

array3[i] = array2[i] + array2[size2 - 1 - i]

Code	Giải thích

.data
array1: .word 5, 6, 7, 8, 1, 2, 3, 9, 10, 4
size1: .word 10
array2: .byte 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16
size2: .word 16
array3: .space 8
size3: .word 8
.text
li $t0, 0
lw $s2, size2
lw $s4, size3
 	
j IfLoop
IfLoop:
blt $t0, $s4, Loop
j End
Loop:
la $s1, array2
	#Khai báo vùng nhớ data
	
	#Khai báo vùng nhớ chứa mã lệnh
	
	# If(index < size3) => Loop

la $s3, array3
add $s0, $s1, $t0
lb $t9, 0($s0)
 	
subi $t8, $s2, 1
sub $t8, $t8, $t0
add $s0, $s1, $t8
lb $t8, 0($s0)
 	
add $t9, $t8, $t9
sll $t1, $t0, 2
add $s3, $s3, $t1
sw $t9, 0($s3)
 	
addi $t0, $t0, 1
j IfLoop
End:
	
	#array2[i]
	
	#array2[size2-1-i]
	
	#array2[i] + array2[size2 - 1 - i]
	
	#Chuyển phần tử tiếp theo
	#array3[i] = ...
	
	#index++

+ Người sử dụng nhập vào mảng thứ mấy và chỉ số phần tử cần lấy trong mảng đó, chương trình xuất ra phần tử tương ứng.

Code	Giải thích
.data array1: .word 5, 6, 7, 8, 1, 2, 3, 9, 10, 4 size1: .word 10 array2: .byte 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 size2: .word 16 array3: .space 8 size3: .word 8 inputArray: .asciiz "Nhap mang can tim (chi nhap vao so): " inputPos: .asciiz "Nhap vao chi so: " error: .asciiz "Khong ton tai chi so nay" .text li $v0, 4 la $a0, inputArray syscall	#Khai báo vùng nhớ data #Khai báo vùng nhớ chứa mã lệnh

```
        li $v0, 5
        syscall
        move $t0, $v0

        li $v0, 4
        la $a0, inputPos
        syscall
        li $v0, 5
        syscall
        move $t1, $v0

        beq $t0, 1, Array1
        beq $t0, 2, Array2
Array1:                          #In array1
        la $s0, array1
        lw $s1, size1
        bge $t1, $s1, Error      # if(I > size) => Error
        sll $t1, $t1, 2
        add $s0, $s0, $t1        # arr1[i] = …
        lw $a0, 0($s0)
        li $v0, 1
        syscall
        j End
Array2:                          #In array2
        la $s0, array2
        lw $s1, size2
        bge $t1, $s1, Error      # If(I > size) => Error
        add $s0, $s0, $t1        #arr2[i] = …
        lb $a0, 0($s0)
        li $v0, 1
        syscall
        j End
Error:                           #In lỗi
        li $v0, 4
        la $a0, error
        syscall
        j End
End:
```

5.3. Bài tập

5.3.1. Nhập một mảng các số nguyên n phần tử (nhập vào số phần tử và giá trị của từng phần tử), xuất ra cửa sổ I/O của MARS theo từng yêu cầu sau:

✓ Xuất ra giá trị lớn nhất và nhỏ nhất của mảng

✓ Tổng tất cả các phần tử của mảng

✓ Người sử dụng nhập vào chỉ số của một phần tử nào đó và giá trị của phần tử đó được in ra cửa sổ

Code	Giải thích
.data Array: .space 100 InputN: .asciiz "Nhap N: " MaxArr: .asciiz "\nMax = " MinArr: .asciiz "\nMin = " SumArr: .asciiz "\nSum = " inputPos: .asciiz "\nNhap vao chi so: "	#Khai báo vùng nhớ data

.text
la $a0, InputN
li $v0, 4
syscall
li $v0, 5
syscall
move $a2, $v0
 	
la $a1, Array
li $t0, 0
jal InputArray
 	
la $a1, Array
li $t0, 0
lw $t1, 0($a1)
jal MaxArray
li $v0, 4
la $a0, MaxArr
syscall
	#Khai báo vùng nhớ chứa mã lệnh
	
	#syscall nhập
	
	#$a2 = N
	
	#InputArray(Array, N)
	
	#$t1 = Max
	#MaxArray(Array, N, Max)
	
	#Print Max

```
        li $v0, 1
        move $a0, $t1
        syscall

        la $a1, Array
        li $t0, 0
        lw $t1, 0($a1)          #$t1 = Min
        jal MinArray            #MinArray(Array, N, Min)
        li $v0, 4
        la $a0, MinArr          #Print Min Value
        syscall
        li $v0, 1
        move $a0, $t1
        syscall

        la $v0, 4
        la $a0, inputPos
        syscall
        li $v0, 5
        syscall                 #Input vị trí mảng cần xuất
        la $a1, Array
        sll $v0, $v0, 2
        add $a1, $a1, $v0       #Nhảy đên vị trí cần xuất
        lw $a0, 0($a1)
        li $v0, 1
        syscall                 #Xuất giá trị

        j End
InputArray:
        blt $t0, $a2, LoopInput
        jr $ra
        LoopInput:
                li $v0, 5
                syscall

                sw $v0, ($a1)
                add $a1, $a1, 4     #Nhảy đến vị trí tiếp theo
                addi $t0, $t0, 1
                j InputArray
MaxArray:
        blt $t0, $a2, FindMax
        jr $ra
        FindMax:
```

add $a1, $a1, 4
lw $t2, ($a1)
bgt $t2, $t1, GanMax
addi $t0, $t0, 1
j MaxArray
GanMax:
move $t1, $t2
addi $t0, $t0, 1
j MaxArray
MinArray:
blt $t0, $a2, FindMin
jr $ra
FindMin:
lw $t2, ($a1)
add $a1, $a1, 4
blt $t2, $t1, GanMin
addi $t0, $t0, 1
j MinArray
GanMin:
move $t1, $t2
addi $t0, $t0, 1
j MinArray
 	
End:
	#a[i+1] > a[i] => GanMax
	
	#Swap(A[i], A[i+1])
	
	#a[i+1] < a[i] => GanMin
	
	#Swap(A[i], A[i+1])

5.3.2. Nhập một mảng các số nguyên n phần tử (nhập vào số phần tử và giá trị của từng phần tử). Mảng này gọi là A.

Chuyển dòng lệnh C dưới đây sang mã assembly của MIPS. Với các biến nguyên i, j được gán lần lượt vào thanh ghi $s0, $s1; và địa chỉ nền của mảng số nguyên A được lưu trong thanh ghi $s3

C	MIPS

if(i < j)
A[i] = i;
else
A[i] = j;
	blt $s0,$s1,L #so sanh $s0<$s1 đung thi nhay sang L
	sw $s1,($s3) #A[i]=$s1
	j exit
	L:
	sw $s0,($s3) #A[i]=$s0
	exit:

5.4. Bài tập Bổ sung:

5.4.1. Viết chương trình hợp ngữ nhập mảng gồm N phần tử. Sắp xếp mảng theo thứ tự giảm dần.

Input : N = {nhập N}

arr = {nhập phần tử mảng}

Output: {Decreasing array}

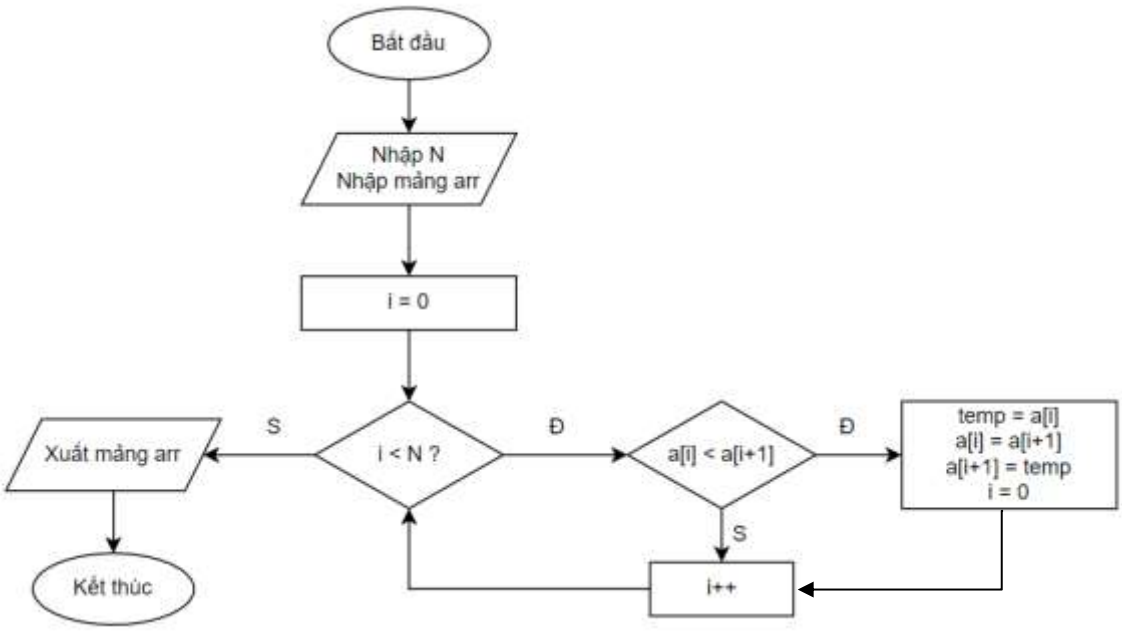
#vd: N= 5

#arr = [2 ,3, 5, 1, 4]

 # output : [5, 4, 3, 2, 1]

.

Program is finished …



Hình 5.4-1 Lưu đồ thuật toán sắp xếp mảng giảm dần

Code	Giải thích
.data Array: .space 100 InputN: .asciiz "Nhap N = " Space: .asciiz " " .text la $a0, InputN li $v0, 4 syscall li $v0, 5 syscall move $s0, $v0 la $s1, Array li $t0, 0 jal InputArray la $s1, Array li $t0, 0 jal SortArray la $s1, Array li $t0, 0 jal OutputArray j End InputArray: blt $t0, $s0, LoopInput jr $ra LoopInput: li $v0, 5	# N: s0 Array: s1

```
                syscall

                addi $s1, $s1, 4        #Nhảy vị trí tiếp theo
                sw $v0, ($s1)
                addi $t0, $t0, 1
                j InputArray

OutputArray:
        blt $t0, $s0, LoopOutput
        jr $ra
        LoopOutput:
                addi $s1, $s1, 4
                addi $t0, $t0, 1
                lw $a0, 0($s1)
                li $v0, 1
                syscall

                la $a0, Space
                li $v0, 4
                syscall
                j OutputArray

SortArray:
        blt $t0, $s0, LoopSort          # for (int i=0; i<n; i++)
        jr $ra                          #       if(A[i] < A[i+1])
        LoopSort:                       #           swap(A[i], A[i+1])
                addi $s1, $s1, 4        #           i = 0
                lw $t2, ($s1)

                addi $t0, $t0, 1
```

```
                addi $s1, $s1, 4
                lw $t3, ($s1)

                subi $s1, $s1, 4

                blt $t2, $t3, HoanVi
                j SortArray
                HoanVi:                 # temp = A[i]
                        move $t4, $t2   # A[i] = A[i+1]
                        move $t2, $t3   # A[i+1] = temp
                        move $t3, $t4

                        sw $t2, ($s1)
                        addi $s1, $s1, 4
                        sw $t3, ($s1)
                        subi $s1, $s1, 4

                        li $t0, 0
                        la $s1, Array
                        j SortArray

End:
```

5.4.2. Viết chương trình hợp ngữ nhập vào N và mang gồm N phần tử. In ra mảng đảo ngược của mảng vừa nhập

Input : N = {nhập N}

Arr = {Nhập mảng}

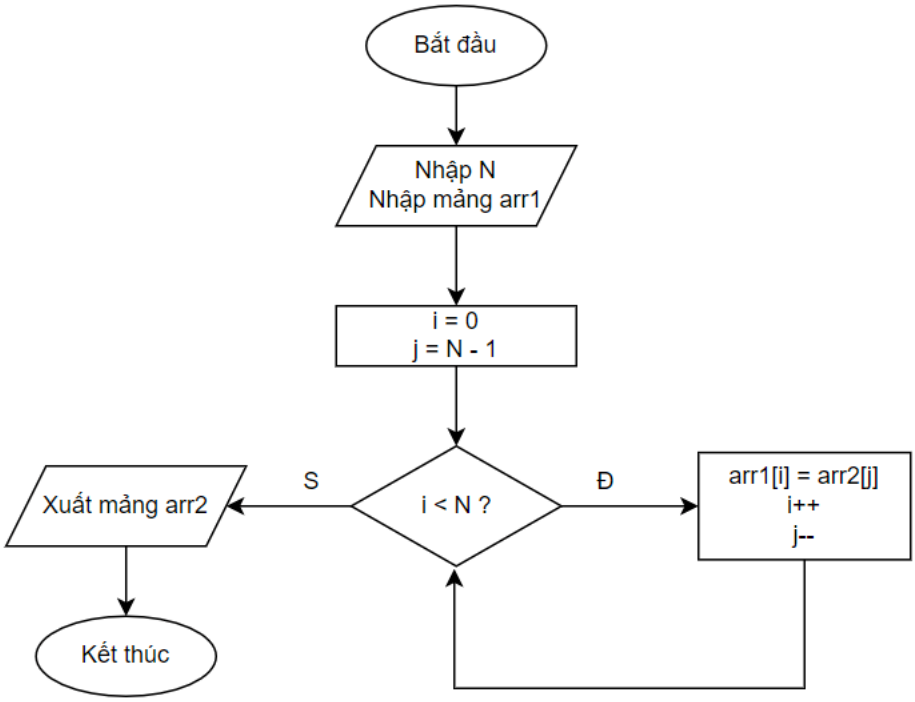
Output: {Reverse array}

#Vd: N = 5

#arr = [1 1 2 3 5]

#output: [5 3 2 1 1]

Program is finished …



Hình 5.4-2 Lưu đồ in mảng đảo ngược

Code	Giải Thích
.data	
Array: .space 100	
Array2: .space 100	
InputN: .asciiz "Nhap N = "	
Space: .asciiz " "	
.text	
la $a0, InputN	# N: s0 Array: s1
li $v0, 4	
syscall	
li $v0, 5	
syscall	
move $s0, $v0	
move $a0, $v0	
la $a1, Array	
li $t0, 0	
jal InputArray	#InputArray(Array, N)
subi $t9, $a0, 1	
la $a2, Array2	
jal DaoArray	#DaoArray(Array2, N)
li $t0, 0	
la $a1, Array2	
jal OutputArray	#OutputArray(Array2)
j End	
InputArray:	
blt $t0, $a0, LoopInput	#If(I < N) => LoopInput
jr $ra	
LoopInput:	

```
                li $v0, 5
                syscall

                sw $v0, ($a1)           #a[i] = input
                addi $a1, $a1, 4
                addi $t0, $t0, 1
                j InputArray

OutputArray:
        blt $t0, $s0, LoopOutput
        jr $ra
        LoopOutput:

                lw $a0, 0($a1)
                li $v0, 1
                syscall

                addi $a1, $a1, 4        #Chuyển tới phần tử tiếp theo
                addi $t0, $t0, 1        #i++

                la $a0, Space
                li $v0, 4
                syscall
                j OutputArray

DaoArray:
        la $a1, Array
        bge $t9, $zero, LoopDao         #If(n-1 >= 0) => LoopDao
        jr $ra
        LoopDao:
                sll $t8, $t9, 2         #t8 = t9 * 4
```

add $a1, $a1, $t8	#array2[n-1-i]
lw $a0, ($a1)	#giá trị của array1[n-1-i]
sw $a0, ($a2)	#array2[i] = array2[n-1-i]
addi $a2, $a2, 4	#nhảy array2 đến vị trí tiếp
subi $t9, $t9, 1	#i--
j DaoArray	
End:	

TÀI LIỆU THAM KHẢO

https://khotrithucso.com/doc/p/cuon-sach-kinh-dien-ve-kien-truc-may-tinh-250190

https://freetuts.net/ngan-xep-stack-la-gi-3146.html

https://electronicscoach.com/difference-between-multiplexer-and-demultiplexer.html

https://khuenguyencreator.com/mang-la-gi-cach-su-dung-mang-lap-trinh-c/