

# Laboratory Exercise 2 – Report:

## Instruction Set, Basic Instructions, Directives

Lê Văn Duẩn - 20194508

### 1. Home Assignment 1

- 32 thanh ghi:

Tên thanh ghi	Số hiệu thanh ghi	Công dụng
\$zero	0	chứa hằng số = 0
\$at	1	giá trị tạm thời cho hợp ngữ
\$v0-\$v1	2-3	các giá trị trả về của thủ tục
\$a0-\$a3	4-7	các tham số vào của thủ tục
\$t0-\$t7	8-15	chứa các giá trị tạm thời
\$s0-\$s7	16-23	lưu các biến
\$t8-\$t9	24-25	chứa các giá trị tạm thời
\$k0-\$k1	26-27	các giá trị tạm thời của OS
\$gp	28	con trỏ toàn cục
\$sp	29	con trỏ ngăn xếp
\$fp	30	con trỏ khung
\$ra	31	địa chỉ trả về của thủ tục

- Khuôn dạng của 3 loại lệnh:

Lệnh kiểu R

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Lệnh kiểu I

op	rs	rt	imm
6 bits	5 bits	5 bits	16 bits

Lệnh kiểu J

op	address
6 bits	26 bits

- Các thanh ghi đặc biệt:

Thanh ghi PC (Program Counter) để tìm nạp lệnh từ bộ nhớ, tăng giá trị trong thanh ghi PC lên 4 đơn vị để lấy địa chỉ của lệnh tiếp theo vì mỗi lệnh của MIPS chiếm 4byte (32 bits)

Thanh ghi HI (high) và LO (Low)

Thao tác nhân của MIPS có kết quả chứa trong 2 thanh ghi HI và LO. Bit 0-31 thuộc LO và 32-63 thuộc HI.

Thao tác chia thì kết quả: HI chứa phần dư, LO: chứa thương

## 2. Home Assignment 2

### 2.1. Assignment 1:

Mã nguồn:

#laboratory Exercise 2, Assignment 1

.text

addi \$s0, \$zero, 0x3007 # \$s0 = 0 + 0x3007 = 0x3007; I-type

add \$s0, \$zero, \$0 # \$s0 = 0 + 0 = 0 ; R-type

- Thanh ghi \$s0 thay đổi từ 0x00000000 sang 0x00003007 sau đó lại trở về 0x00000000 -> vì lệnh đầu tiên theo khuôn lệnh I gán cho thanh ghi giá trị 0x00003007, lệnh sau theo khuôn lệnh R gán cho thanh ghi \$s0 giá trị 0 là hằng giá trị của thanh ghi \$0

- Thanh ghi \$pc tăng 4 sau mỗi câu lệnh -> vì mỗi lệnh là 32-bit = 4 byte dẫn đến cần tăng 4 để có thể trở đến lệnh tiếp theo.

- Lệnh đầu tiên theo khuôn I: addi \$s0, \$zero, 0x3007

addi -> 001000;

\$s0 -> 16 -> 10000 (đích rt - 5bits),

\$zero -> 00000 (nguồn rs 5-bits)

hằng 0x3007 (imm)

-> mã máy : 0010|0000|0010|0000|0011|0000|0000|0111 = 0x20103007

- Lệnh thứ 2 theo khuôn R: add \$s0, \$zero, \$0

op-> 000000;

\$s0 -> 10000 (đích rt - 5bits);

\$zero -> 00000 (nguồn 1 rs - 5bits);

\$0 -> 00000 (nguồn 2 rt - 5bits);

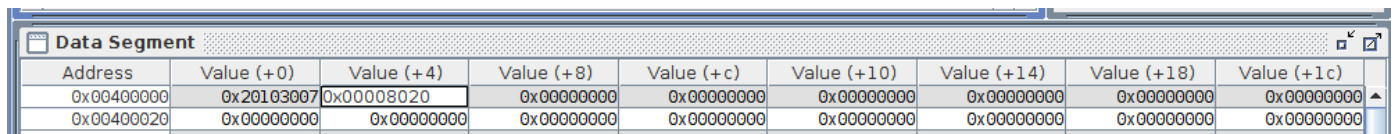
\$s0 -> 10000 (đích rd - 5 bits),

shamt -> 00000 vì không phải lệnh dịch bit

funct -> add -> 100000

-> mã máy: 0000|0000|0000|0000|1000|0000|0010|0000 = 00008020

Kết quả chạy trên Mars đúng như tính toán của 2 lệnh trên:



Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00400000	0x20103007	0x00008020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00400020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

- Khi sửa lại lệnh lui thành: addi \$s0, \$zero, 0x211003d

-> đây là trường hợp hằng số nguyên là 32bit -> sử dụng lệnh lui và ori

Lệnh lui: lui rt, constant\_hi16bit

-> Copy 16 bit cao của hằng số 32-bit vào 16 bit trái của rt và xóa 16 bits bên phải của rt về 0

Lệnh ori: ori rt,rt,constant\_low16bit

-> Đưa 16 bit thấp của hằng số 32-bit vào thanh ghi rt

=> Mars sử dụng thanh ghi \$1( là \$at trên thanh ghi) làm trung gian (rt) trong các lệnh lui và ori nên giá trị của \$at thay đổi từ 0x02110000 sau đó là 0x0211003d.

Sau đó thực hiện lệnh add \$s0, \$zero, \$1 để thực hiện gán \$s0 = 0x0211003d.

## 2.2. Assignment 2: lệnh gán số 32-bit

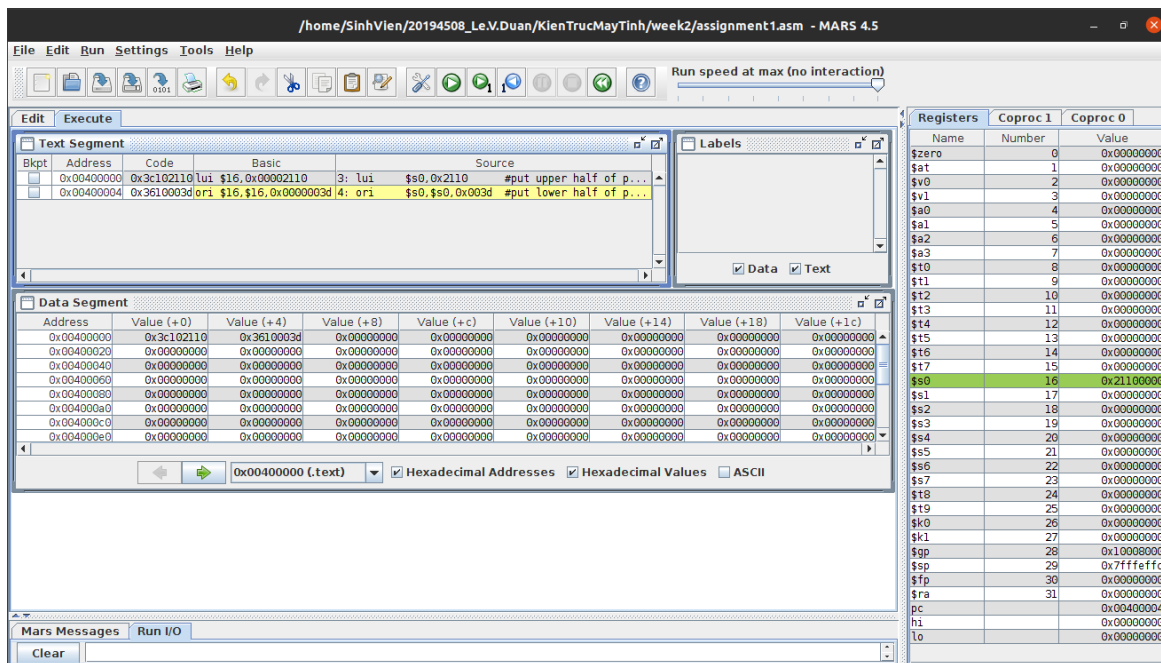
Mã nguồn:

#laboratory Exercise 2, Assignment 2

.text

lui \$s0,0x2110 #put upper half of pattern in \$s0

ori \$s0,\$s0,0x003d #put lower half of pattern in \$s0



- Giá trị thanh ghi \$s0 thay đổi từ 0x00000000 sang 0x21100000 sau đó là 0x2110003d.

- Byte đầu tiên của vùng lệnh trong data segment trùng với cột code của các lệnh trong text segment.

## 2.3. Assignment 3: lệnh gán (giả lệnh)

Mã nguồn:

#Laboratory Exercise 2, Assignment 3

.text

```
li    $s0,0x2110003d    #pseudo instruction=2 basic instructions
```

```
li    $s1,0x2           #but if the immediate value is small, one ins
```

The screenshot shows a debugger window with two main panes. The top pane is titled 'Text Segment' and contains a table of assembly instructions. The bottom pane is titled 'Data Segment' and contains a table of memory addresses and their values.

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c012110	lui \$1,0x00002110	3: li \$s0,0x2110003d #pseudo inst...
	0x00400004	0x3430003d	ori \$16,\$1,0x0000003d	
	0x00400008	0x24110002	addiu \$17,\$0,0x0000...	4: li \$s1,0x2 #but if the ...

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00400000	0x3c012110	0x3430003d	0x24110002	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00400020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

- Điều bất thường là dùng cùng giả lệnh li nhưng lệnh đầu được thực thi thành 2 lệnh lui và ori. vì hằng số nguyên đưa vào là 0x211003d là 32-bits lớn hơn 16-bits. Đối với câu lệnh 2 là 0x02 nhỏ hơn 16-bits nên giả lệnh gán li có thể thực thi ngay thông qua lệnh addiu

#### 2.4. Assignment 4: tính biểu thức $2x + y = ?$

Mã nguồn:

#Laboratory Exercise 2, Assignment 4

.text

# Assign X, Y

```
addi $t1, $zero, 5    # X = $t1 = ?
```

```
addi $t2, $zero, -1   # Y = $t2 = ?
```

# Expression  $Z = 2X + Y$

```
add  $s0, $t1, $t1    # $s0 = $t1 + $t1 = X + X = 2X
```

```
add  $s0, $s0, $t2    # $s0 = $s0 + $t2 = 2X + Y
```

- mã máy của 2 lệnh add

0x01298020 = 0000 00|01 001|0 1001| 1000 0|000 00|10 0000

-> op = 00000 -> lệnh R

rs và rt (\$t1) = 010 001 ;

rd (\$s0) = 10000 ;

funct = 10 0000 -> lệnh add

0x020a8020 = 0000 00 | 10 000 | 0 1010 | 1000 0 | 000 00 | 10 0000

-> op = 00000 -> lệnh R

rs (\$s0) = 10 000;

rt (\$t2) = 0 1010 ;

rd (\$s0) = 10000;

funct = 10 0000 -> lệnh add

## 2.5. Assignment 5: phép nhân

Mã nguồn:

#laboratory Exercise 2, Assignment 5

.text

# Assign X, Y

addi \$t1, \$zero, 4 # X = \$t1

addi \$t2, \$zero, 5 # Y = \$t2

# Expression Z = 3\*XY

mul \$s0, \$t1, \$t2 # HI-LO = \$t1 \* \$t2 = X \* Y; \$s0 = LO

mul \$s0, \$s0, 3 # \$s0 = \$s0 \* 3 = 3 \* X \* Y

# Z' = Z

mflo \$s1

## - Text Segment:

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x20090004	addi \$9,\$0,4	4: addi \$t1, \$zero, 4 # X = \$t1
<input type="checkbox"/>	0x00400004	0x200a0005	addi \$10,\$0,5	5: addi \$t2, \$zero, 5 # Y = \$t2
<input type="checkbox"/>	0x00400008	0x712a8002	mul \$16,\$9,\$10	8: mul \$s0, \$t1, \$t2 # HI-LO = \$t1 * \$t2 = X * Y;...
<input type="checkbox"/>	0x0040000c	0x20010003	addi \$1,\$0,3	9: mul \$s0, \$s0, 3 # \$s0 = \$s0 * 3 = 3 * X * Y
<input type="checkbox"/>	0x00400010	0x72018002	mul \$16,\$16,\$1	
<input type="checkbox"/>	0x00400014	0x00008812	mflo \$17	12: mflo \$s1

+ Trước khi thực thi lệnh mul thứ 2 xuất hiện lệnh addi trước. Vì mul chỉ thao tác với các thanh ghi nên phải gán 3 vào thanh ghi tạm thời \$1 ( hay \$at)

+ 2 lệnh đầu tiên gán giá trị 4 và 5 vào các thanh ghi \$t1 và \$t2 đại diện cho X và Y

+ Lệnh: mul \$s0, \$t1, \$t2 -> tính toán giá trị  $X * Y$  và gán vào thanh ghi \$s0 đồng thời cũng gán kết quả vào thanh ghi đặc biệt HI-LO trong trường hợp này là 20 nên kết quả ở LO và HI vẫn là 0.

EditExecute

Text Segment

Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x20090004	addi \$9,\$0,4	4: addi \$t1, \$zero, 4 # X = \$t1
<input type="checkbox"/>	0x00400004	0x200a0005	addi \$10,\$0,5	5: addi \$t2, \$zero, 5 # Y = \$t2
<input type="checkbox"/>	0x00400008	0x712a8002	mul \$16,\$9,\$10	8: mul \$s0, \$t1, \$t2 # HI-LO = \$t1 * \$t2 = X * Y; \$s0 = LO
<input type="checkbox"/>	0x0040000c	0x20010003	addi \$1,\$0,3	9: mul \$s0, \$s0, 3 # \$s0 = \$s0 * 3 = 3 * X * Y
<input type="checkbox"/>	0x00400010	0x72018002	mul \$16,\$16,\$1	
<input type="checkbox"/>	0x00400014	0x00008812	mflo \$17	12: mflo \$s1

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0	0	0	0	0	0	0	0
0x10010020	0	0	0	0	0	0	0	0
0x10010040	0	0	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0	0	0
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0
0x100100e0	0	0	0	0	0	0	0	0
0x10010100	0	0	0	0	0	0	0	0
0x10010120	0	0	0	0	0	0	0	0
0x10010140	0	0	0	0	0	0	0	0

0x10010000 (.data)

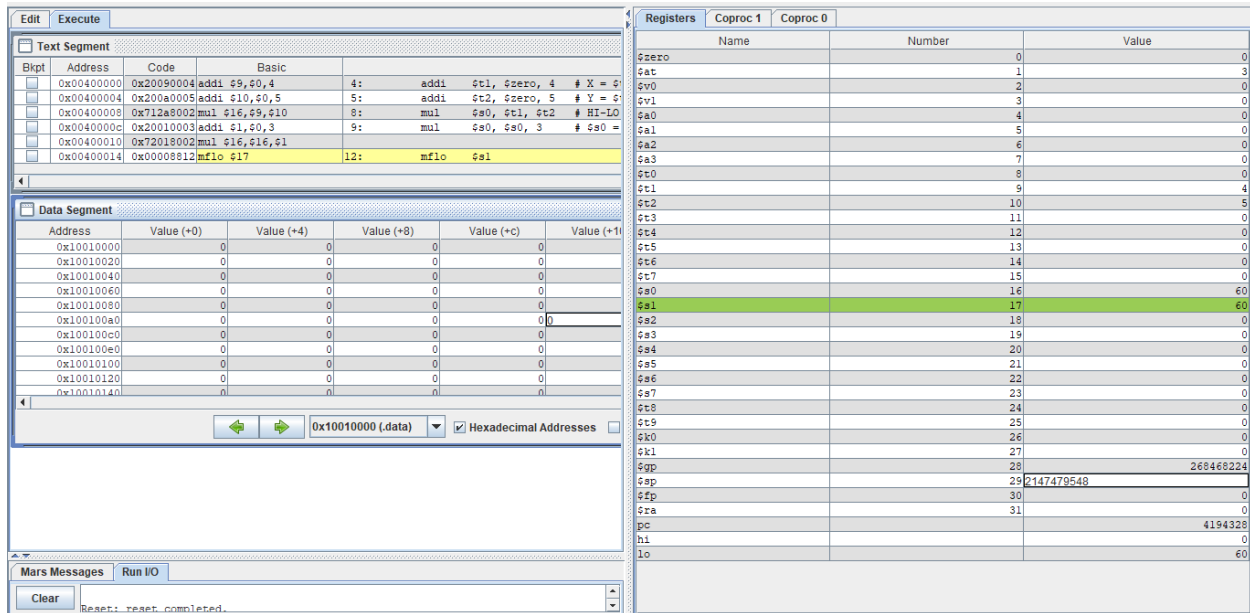
☒Hexadecimal Addresses☐Hexadecimal Values☐ASCII

RegistersCoproccoproc 0

Name	Number	Value
\$zero	0	0
\$at	1	0
\$v0	2	0
\$v1	3	0
\$a0	4	0
\$a1	5	0
\$a2	6	0
\$a3	7	0
\$t0	8	0
\$t1	9	4
\$t2	10	5
\$t3	11	0
\$t4	12	0
\$t5	13	0
\$t6	14	0
\$t7	15	0
\$s0	16	20
\$s1	17	0
\$s2	18	0
\$s3	19	0
\$s4	20	0
\$s5	21	0
\$s6	22	0
\$s7	23	0
\$s8	24	0
\$s9	25	0
\$k0	26	0
\$k1	27	0
\$gp	28	268468224
\$fp	29	2147479548
\$sp	30	0
\$ra	31	0
pc		4194316
hi		0
lo		20

+ Khi đến lệnh mul \$s0, \$s0, 3 -> thực thi lệnh addi như trên để lưu giá trị vào thanh ghi tạm \$1 sau đó tính toán kết quả phép nhân và gán lại vào \$s0 và LO như trên vì kết quả chỉ đang là 60.

+ Lệnh cuối: mflo \$s1 để gán kết quả từ HI-LO (TH này là LO) vào thanh ghi \$s1



## 2.6. Assignment 6: tạo biến và truy cập biến

Mã nguồn:

# Laboratory Exercise 2, Assignment 6

.data                   # DECLARE VARIABLES

X:     .word 5         # Variable X, word type, init value = 5

Y:     .word -1        # Variable Y, word type, init value = -1

Z:     .word           # Variable Z, word type, no init value

.text

# Load X, Y to registers

la     \$t8, X          # Get the address of X in Data segment

la     \$t9, Y          # Get the address of Y in Data segment

lw     \$t1, 0(\$t8)     # \$t1 = X

lw     \$t2, 0(\$t9)     # \$t2 = Y



# Calculate the expression  $Z = 2X + Y$  with registers only

add \$s0, \$t1, \$t1 # \$s0 = 2X

add \$s0, \$s0, \$t2 # \$s0 = 2X + Y

# Store result from registers to variable Z

la \$t7, Z # Get the address of Z in Data segment

sw \$s0, 0(\$t7) # Z = \$s0 = 2X + Y

Text Segment:

Text Segment					Labels	
Bkpt	Address	Code	Basic	Source	Label	Address
	0x00400000	0x3c011001	lui \$1,4097	8: la \$t8, X # Get the address of X in Data segment	assignment6.asm	
	0x00400004	0x34380000	ori \$24,\$1,0		X	0x10010000
	0x00400008	0x3c011001	lui \$1,4097	9: la \$t9, Y # Get the address of Y in Data segment	Y	0x10010004
	0x0040000c	0x34390004	ori \$25,\$1,4		Z	0x10010008
	0x00400010	0x8f090000	lw \$9,0(\$24)	10: lw \$t1, 0(\$t8) # \$t1 = X		
	0x00400014	0x8f2a0000	lw \$10,0(\$25)	11: lw \$t2, 0(\$t9) # \$t2 = Y		
	0x00400018	0x01298020	add \$16,\$9,\$9	14: add \$s0, \$t1, \$t1 # \$s0 = 2X		
	0x0040001c	0x020a8020	add \$16,\$16,\$10	15: add \$s0, \$s0, \$t2 # \$s0 = 2X + Y		
	0x00400020	0x3c011001	lui \$1,4097	18: la \$t7, Z # Get the address of Z in Data segment		
	0x00400024	0x342f0008	ori \$15,\$1,8			
	0x00400028	0xadf00000	sw \$16,0(\$15)	19: sw \$s0, 0(\$t7) # Z = \$s0 = 2X + Y		

- Lệnh la được biên dịch thay bằng hai lệnh lui và ori

- Tại cửa sổ Label:

+ Địa chỉ của 3 biến X, Y, Z không thay đổi trong quá trình biên dịch và cách nhau lần lượt 4

+ Các địa chỉ của các biến đều đúng với các giá trị gán cho chúng

Data Segment								
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	5	-1	9	0	0	0	0	0
0x10010020	0	0	0	0	0	0	0	0

- Khi thực hiện lệnh la thì lệnh lui và ori được thực thi, dùng thanh ghi \$1 làm trung gian để gán các thanh ghi \$t8 và \$t9 các giá trị

- Lệnh lw \$t1, 0(\$t8) -> đọc giá trị từ địa chỉ 0(\$t8) và lưu vào trong thanh ghi \$t1, tương tự với lệnh lw còn lại. Địa chỉ 0(\$t8) nghĩa là: \$t8 là địa chỉ cơ sở, hằng số (offset) 0 dịch chuyển -> Địa chỉ cần đọc = địa chỉ cơ sở + Offset.

- Lệnh Sw \$s0, 0(\$t7) -> dùng để lấy giá trị từ thanh ghi có địa chỉ \$s0 để đưa vào bộ nhớ 0(\$t7)