I. Home Assignment 1:

1. Tên và ý nghĩa của 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 toàn cục

2. Các thanh ghi đặc biệt:

- PC: là thanh ghi của CPU dùng để giữ địa chỉ của lệnh sẽ được nhận vào
- HI và LO: 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.

3. Khuôn dạng của 3 loại lệnh I, J, R:

- Lệnh kiểu R:

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

  Các trường của lệnh:

  - op: Mã thao tác (với lệnh kiểu R, op = 000000)
  - rs: Số hiệu thanh ghi nguồn thứ nhất
  - rt: Số hiệu thanh ghi nguồn thứ hai
  - rd: Số hiệu thanh ghi đích
  - shamt (shift amount): số bit được dịch, chỉ dùng cho lệnh dịch bit, với các lệnh khác shamt = 00000
  - funct: mã hàm -> mã hoá cho thao tác cụ thể

- Lệnh kiểu I:

op	rs	rt	imm
6 bit	5 bit	5 bit	16 bit

Dùng cho các lệnh số học/logic với toán hạng tức thì và các lệnh load/store.
Với lệnh addi:

- rs: Số hiệu thanh ghi nguồn
- rt: Số hiệu thanh ghi đích

Với lệnh lw, sw:

- rs: Số hiệu thanh ghi cơ sở
- rt: Số hiệu thanh ghi đích (lw) hoặc số hiệu thanh ghi nguồn (sw)

- Lệnh kiểu J:

op	address
6 bit	26 bit

Toán hạng 26 địa chỉ, được sử dụng cho các lệnh nhảy.

II. Assignment 1: Lệnh gán số 16 bit

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
```

1. Quan sát cửa sổ Register:

Giá trị của các thanh ghi ở trong bảng sau

Lần chạy	$s0	$pc	Giải thích
0	0x00000000	0x00400000	Đây là các giá trị mặc định của $s0 và $pc
1	0x00003007	0x00400004	Sau khi chạy lệnh addi, thanh ghi $s0 có số hiệu thanh ghi là 16 sẽ được gán giá trị tương ứng là 0x00003007. Sau khi lệnh được nhận vào, nội dung $pc tự động tăng 4 byte để trỏ đến lệnh kế tiếp. (0x00400000 -> 0x00400004)
2	0x00000000	0x00400008	Sau khi chạy lệnh add, thanh ghi $s0 sẽ có giá trị là 0x00000000 ($s0 = $0 + $0). Sau khi lệnh được nhận vào, nội dung $pc tự động tăng 4 byte để trỏ đến lệnh kế tiếp. (0x00400004 -> 0x00400008)

2. Quan sát cửa sổ Text Segment:

Code	Basic	Source
0x20103007	addi $16,$0,0x00003007	3: addi $s0, $zero, 0x3007 # $s0 = 0 + 0x3007 = 0x3007 ;I-type
0x00008020	add $16,$0,$0	4: add $s0, $zero, $0 # $s0 = 0 + 0 = 0 ;R-type

- Khuôn dạng lệnh thứ nhất (lệnh kiểu I)

op	rs	rt	imm
001000	00000	10000	0011000000000111
Hexa: 0x20103007			

- Khuôn dạng lệnh thứ hai (Lệnh kiểu R)

op	rs	rt	rd	shamt	funct
000000	00000	00000	10000	00000	100000
Hexa: 0x00008020					

Ta thấy mã máy của các lệnh trên đúng với tập lệnh đã quy định

3. Sửa lại lệnh:

```
addi  $s0, $zero, 0x2110003d
```

Ta quan sát thấy ở cửa sổ Text Segment, xuất hiện thêm 2 lệnh lui và ori

lui $1,0x00002110	3: addi $s0, $zero, 0x2110003d
ori $1,$1,0x0000003d	
add $16,$0,$1	

Giải thích: Do hằng số được addi vào $s0 là hằng số 32 bit nên chương trình tự động chuyển thành lệnh lui và ori để xử lý hằng số 32 bit. Copy 16 bit cao của hằng số 32 bit vào nửa bên trái rt (lui), Xoá 16 bit bên phải của rt về 0 , đưa 16 bit thấp của hằng số 32-bit vào nửa bên phải rt. (ori)

III. 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
```

1. Quan sát cửa sổ Register:

Lần chạy	$0	$pc	Giải thích
0	0x00000000	0x00400000	Đây là các giá trị mặc định của $s0 và $pc
1	0x21100000	0x00400004	Lệnh lui copy 16 bit vào nửa bên trái của $s0. Sau khi lệnh được nhận vào, nội dung $pc tự động tăng 4 byte để trỏ đến lệnh kế tiếp. (0x00400000 -> 0x00400004)
2	0x2110003d	0x00400008	Lệnh ori đưa 16 bit vào nửa bên phải $s0. Sau khi lệnh được nhận vào, nội dung $pc tự động tăng 4 byte để trỏ đến lệnh kế tiếp. (0x00400004 -> 0x00400008)

2. Quan sát cửa sổ Data Segment:

Ta thấy:

Address	Value (+0)	Value (+4)
0x00400000	0x3c102110	0x3610003d

Mã máy tương ứng:

Code	Basic
0x3c102110	lui $16,0x00002110
0x3610003d	ori $16,$16,0x0000003d

Vậy địa chỉ vùng lệnh 0x00400000 tương ứng với câu lệnh thứ nhất (lui), địa chỉ vùng lệnh 0x00400004 tương ứng với câu lệnh thứ hai (ori).

IV. 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
```

1. Quan sát cửa sổ Text Segment

Sau khi biên dịch thì ta thấy 2 câu lệnh gán (li) được chuyển thành 3 câu lệnh (lui, ori, addiu)

Code	Basic
0x3c012110	lui $1,0x00002110
0x3430003d	ori $16,$1,0x0000003d
0x24110002	addiu $17,$0,0x00000002

Giải thích:

- Ở câu lệnh thứ nhất là lệnh gán 32 bit -> chương trình sẽ tự động tách thành 2 lệnh lui và ori. Lệnh lui copy 16 bit cao của hằng số 32 bit vào nửa bên trái của $1 (2110), lệnh ori copy 16 bit thấp của hằng số 32 bit vào nửa bên trái của $1 (003d).

- Ở câu lệnh thứ hai là lệnh gán 16 bit -> chương trình thực hiện phép gán li bằng câu lệnh addiu như bình thường. (addiu khác với addi ở chỗ sẽ không báo lỗi khi tràn số).

V. 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
```

1. Quan sát cửa sổ Register:

Lần chạy	$0	$t1	$t2	Giải thích
0	0x00000000	0x00000000	0x00000000	Đây là các giá trị khởi đầu
1	0x00000000	0x00000005	0x00000000	Gán giá trị 5 cho $t1
2	0x00000000	0x00000005	0xffffffff	Gán giá trị -1 cho $t2
3	0x0000000a	0x00000005	0xffffffff	$s0 = $t1 + $t1 = 5 + 5 = 10
4	0x00000009	0x00000005	0xffffffff	$s0 = $s0 + $s2 = 10 – 1 = 9

Sau khi kết thúc chương trình, kết quả chính xác (2x – y = 2*5 + (-1) = 9)

2. Quan sát cửa sổ Text Segment

- Ta thấy hợp ngữ và mã máy có điểm tương đồng thông qua khuôn mẫu của kiểu lệnh I

Code	Basic
0x20090005	addi $9,$0,0x00000005
0x200affff	addi $10,$0,0xffffffff

- Chuyển mã máy của lệnh add sang hệ 2:
  - Lệnh thứ nhất `add $s0, $t1, $t1`

op	rs	rt	rd	shamt	funct
000000	01001	01001	10000	00000	100000
Hexa: 0x01298020					

  - Lệnh thứ hai `add $s0, $s0, $t2`

op	rs	rt	rd	shamt	funct
000000	10000	01010	10000	00000	100000
Hexa: 0x020a8020					

Ta thấy mã máy của lệnh add sau khi chuyển đúng với khuôn mẫu của kiểu lệnh R

VI. 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
```

1. Quan sát cửa sổ Text Segment:

Sau khi biên dịch mã máy, chương trình đã tự động tách 5 lệnh thành 6 lệnh.

Code	Basic
0x20090004	addi $9,$0,0x00000004
0x200a0005	addi $10,$0,0x00000005
0x712a8002	mul $16,$9,$10
0x20010003	addi $1,$0,0x00000003
0x72018002	mul $16,$16,$1
0x00008812	mflo $17

2. Quan sát cửa sổ Register

Lần chạy	$s0	$s1	$t1	$t2	Hi	Lo	$at
0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
1	0x00000000	0x00000000	0x00000004	0x00000000	0x00000000	0x00000000	0x00000000
2	0x00000000	0x00000000	0x00000004	0x00000005	0x00000000	0x00000000	0x00000000
3	0x00000014	0x00000000	0x00000004	0x00000005	0x00000000	0x00000014	0x00000000
4	0x00000014	0x00000000	0x00000004	0x00000005	0x00000000	0x00000014	0x00000003
5	0x0000003c	0x00000000	0x00000004	0x00000005	0x00000000	0x0000003c	0x00000003
6	0x0000003c	0x0000003c	0x00000004	0x00000005	0x00000000	0x0000003c	0x00000003

Giải thích:

Lần chạy	Giải thích
0	Đây là các giá trị khởi đầu
1	Gán giá trị 4 cho $t1
2	Gán giá trị 5 cho $t2
3	Gán giá trị $s0 = $t1 * $t2. Do kết quả là 32 bit nên thanh ghi Lo sẽ được cập nhật kết quả này. (Lo = 20)
4	Phép nhân $0 = $s0 * 3 đã được tách thành 2 lệnh ở lần chạy 4 và 5. Ở lần chạy 4, thực hiện phép gán $at = 3
5	Ở lần chạy 5, thực hiện nhân $s0 = $s0 * $at = $s0 * 3 = 0x0000003c = 60. Thanh ghi Lo cũng cập nhật kết quả này.
6	Lấy giá trị của thanh ghi Lo ghi vào $s1. ($s1 = 60)

Sau khi kết thúc chương trình, ta thấy kết quả thu được là chính xác. ($s0 = 3 * X * Y = 3 * 4 * 5 = 60 = 0x0000003c)

VII. 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 =
Y :   .word   -1            # Variable Y, word type, init value =
Z :   .word                 # Variable Z, word type, no init value

.text                       # DECLARE INSTRUCTIONS
     #  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

     #  Calcuate the expression Z = 2X + Y with registers only
     add   $s0, $t1, $t1    # $s0 = $t1 + $t1 = X + X = 2X
     add   $s0, $s0, $t2    # $s0 = $s0 + $t2 = 2X + Y

     # Store result from register to variable Z
     la    $t7, Z           # Get the address of Z in Data Segment
     sw    $s0, 0($t7)      # Z = $s0 = 2X + Y
```

1. Quan sát cửa sổ Text Segment

Basic	
lui $1,0x00001001	9: la $t8, X # Get the address of X in Data Segment
ori $24,$1,0x00000000	
lui $1,0x00001001	10: la $t9, Y # Get the address of Y in Data Segment
ori $25,$1,0x00000004	
lw $9,0x00000000($24)	11: lw $t1, 0($t8) # $t1 = X
lw $10,0x00000000($25)	12: lw $t2, 0($t9) # $t2 = Y
add $16,$9,$9	14: add $s0, $t1, $t1 # $s0 = $t1 + $t1 = X + X = 2X
add $16,$16,$10	15: add $s0, $s0, $t2 # $s0 = $s0 + $t2 = 2X + Y
lui $1,0x00001001	17: la $t7, Z # Get the address of Z in Data Segment
ori $15,$1,0x00000008	
sw $16,0x00000000($15)	18: sw $s0, 0($t7) # Z = $s0 = 2X + Y

Lệnh la được tách ra thành 2 lệnh lui và ori. Lệnh lui copy địa chỉ 16 bit bên trái của biến vào thanh ghi tạm thời $at. Lệnh ori copy địa chỉ 16 bit bên phải của biến, kết hợp với 16 bit bên trái của thanh ghi $at, ghi vào thanh ghi đích.

2. Quan sát cửa sổ Label

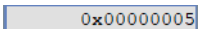
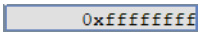
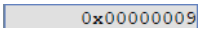
- So sánh với hằng số khi biên dịch
  - Hằng số khi quan sát trên cửa sổ Label:

X	0x10010000
Y	0x10010004
Z	0x10010008

  - Hằng số khi biên dịch lệnh la thành mã máy:

lui $1,0x00001001	9: la $t8, X # Get the address of X in Data Segment
ori $24,$1,0x00000000	
lui $1,0x00001001	10: la $t9, Y # Get the address of Y in Data Segment
ori $25,$1,0x00000004	
lui $1,0x00001001	12: la $t7, Z # Get the address of Z in Data Segment
ori $15,$1,0x00000008	

Ta thấy giá trị hằng số trên cửa sổ Label giống với hằng số khi biên dịch lệnh la thành mã máy.

- Click đúp vào các giá trị X, Y, Z
  - Giá trị của X: 
  - Giá trị của Y: 
  - Giá trị của Z: 

Ta thấy giá trị của các biến biến X, Y, Z trong bộ nhớ ở cửa số Data Segment giống với các giá trị đã khai báo ban đầu.

3. Quan sát cửa sổ Register:

- Quan sát sự thay đổi của các thanh ghi qua từng lần chạy
- Xác định vai trò của lệnh lw và sw
  - Lệnh lw: **nạp** word dữ liệu 32-bit từ bộ nhớ đưa vào thanh ghi, thông qua địa chỉ sơ sở và hằng số dịch chuyển địa chỉ.
  - Lệnh sw: **lưu** word dữ liệu 32-bit từ thanh ghi ra bộ nhớ, thông qua địa chỉ sơ sở và hằng số dịch chuyển địa chỉ.

4. Tìm hiểu thêm về lệnh lb, sb

- Lệnh lb:
  - Nạp 1 byte hoặc 2 byte (halfword) từ bộ nhớ vào bên phải thanh ghi đích rt
  - Phần còn lại của thanh ghi rt được mở rộng có dấu thành 32-bit (Sign-extended)
- Lệnh sb:
  Chỉ lưu byte/halfword bên phải thanh ghi rt ra bộ nhớ .