

## Báo cáo Thực hành KTMT buổi 2

Họ và tên: Nguyễn Đức Phú

MSSV: 20215116

### Assignment 1: Lệnh gán số 16-bit

- Với code ban đầu:
  - Với mỗi lệnh, tại cửa sổ Register:
    - Thanh ghi **pc** tăng thêm 4 vì mỗi mệnh có độ dài 32-bit tương đương với 4 byte.
    - Giá trị thanh ghi \$s0 thay đổi dần như sau:  
0x00000000 – 0x00003007- 0x00000000  
Do ban đầu giá trị thanh ghi là 0, thực hiện lệnh addi đầu tiên là lệnh cộng với hằng số (0+0x3007) nên giá trị \$s0 thay đổi thành 0x00003007, lệnh add tiếp theo là lệnh cộng 0+0 nên giá trị trở lại thành 0x00000000.
  - So sánh mã máy với khuôn dạng lệnh:  
**addi \$s0, \$zero, 0x3007** là lệnh kiểu I:  
op: 8 => 001000  
rs: \$0 => 00000  
rt: \$16 => 10000  
imm: 0x00003007 => 0011 0000 0000 0111  
⇒ Code: 0010 0000 0001 0000 0011 0000 0000 0111  
0x20103007  
⇒ Trùng khớp => Đúng như tập lệnh đã quy định  
**add \$s0, \$zero, \$0** là lệnh kiểu R:  
op: 0 => 000000  
rs: \$0 => 00000  
rt: \$0 => 00000  
rd: \$16 => 10000  
sh: 0 => 00000  
fn: 32 => 100000  
⇒ Code: 0000 0000 0000 0000 1000 0000 0010 0000  
0x00008020  
⇒ Trùng khớp => Đúng như tập lệnh đã quy định

- Thay đổi thành lệnh **addi \$s0, \$zero, 0x21110003d**
  - Lệnh được tách thành 3 lệnh **lui**, **ori** và **add** do 0x21110003d có độ dài 32 bit trong khi lệnh **addi** có giới hạn với 16 bit
  - Lệnh **lui** thực hiện ghi 0x2110 vào nửa trên của thanh ghi tạm \$at
  - Lệnh **ori** thực hiện OR 0x003d với các giá trị cuối của \$at (hiện đang toàn bộ là 0) tương đương với việc gán 0x003d vào các giá trị cuối
  - Lệnh **add** thực hiện cộng giá trị từ \$zero và \$at, lưu vào \$s0

## Assignment 2: Lệnh gán số 32-bit

- Thanh ghi **pc** tiếp tục tăng thêm 4 với mỗi lệnh
- Thanh ghi \$s0 thay đổi từ 0x00000000 - 0x21100000 - 0x2110003d dưới sự thực hiện của các lệnh lui và ori đã giải thích ở Assignment 1
- Tại cửa sổ Data Segment ta thấy lệnh **lui** có code 0x3c102110 trùng với cột value (+0) và lệnh **ori** có code 0x3610003d trùng với cột value (+4) tại hàng có address 0x00400000 là địa chỉ của lệnh đầu tiên, số value (+4) là tăng thêm 4 với mỗi lệnh - tương ứng 4 byte

## Assignment 3: Lệnh gán

- **li** là lệnh khởi tạo thanh ghi với hằng số
- Lệnh **li** đầu khởi tạo với hằng 0x2110003d có độ dài 32 bit trong khi nó có giới hạn thao tác với 16 bit nên lệnh tiếp tục được chia thành 2 lệnh lui và ori với cơ chế đã được giải thích ở trên
- Lệnh **li** tiếp theo được thực hiện trực tiếp do hằng số ở đây là 0x2 là 2 đã đủ nhỏ

## Assignment 4: Tính 2X+Y

- Thanh ghi **pc** tăng thêm 4 với mỗi lệnh
- Thanh ghi \$t1 và \$t2 thay đổi giá trị theo lệnh **addi** thành 0x00000005 (5) và 0xffffffff (-1)
- Lệnh **add** đầu: \$s0 tăng thành 0x0000000a (10) = 2X
- Lệnh **add** tiếp theo: \$s0 tăng thành 0x00000009 (9) = 2X+Y
- Kết quả thu được đúng:  $2*5+(-1) = 9$
- Lệnh **addi \$t1, \$zero, 5** có code:  
0x20090005 => 0010 0000 0000 1001 0000 0000 0000 0101  
Op: 001000 => 8: op của lệnh addi

- rs: 00000 => 0: \$0 (\$zero)  
 rt: 01001 => 9: \$9 (\$t1)  
 imm: 0000 0000 0000 0101 => 5  
 ⇒ Đúng với khuôn mẫu của lệnh kiểu I
- Lệnh **add \$s0, \$t1, \$t1** có code:  
 0x01298020 => 0000 0001 0010 1001 1000 0000 0010 0000  
 Op: 000000 => 0: op của lệnh add  
 rs: 01001 => 9: là \$9 (\$t1)  
 rt: 01001 => 9: là \$9 (\$t1)  
 rd: 10000 => 16: là \$16 (\$s0)  
 sh: 00000 => 0  
 fn: 100000 => 32: là funct của lệnh add  
 ⇒ Đúng với khuôn mẫu của kiểu lệnh R

## Assignment 5: Phép nhân

- Lệnh **mul \$s0, \$t1, \$t2** được thực hiện trực tiếp
- Tuy nhiên lệnh **mul \$s0, \$s0, 3** được chia thành 2 lệnh **addi \$1, \$0, \$00000003** và **mul \$16, \$16, \$1**  
 Lệnh **addi** thực hiện ghi 3 vào thanh ghi tạm \$1 sau đó mới nhân \$1 với \$16 bằng lệnh **mul** và lưu vào \$16
- Trên cửa sổ Registers:
  - Thanh **LO** thay đổi như sau:  
 0x00000000 – 0x00000014 là kết quả của việc thực hiện phép nhân \$t1 và \$t2 ( $4*5=20 \Rightarrow 0x14$ )  
 0x00000014 – 0x0000003c là kết quả của việc nhân \$s0 (đang có giá trị 0x14) với 3 ( $20*3=60 \Rightarrow 0x3c$ )
  - Thanh **HI** không thay đổi do giá trị của các phép nhân vẫn chưa vượt quá độ dài 32-bit
- Kết quả trả về là 0x3c => 60 đúng ( $3*4*5=60$ )

## Assignment 6: Tạo biến và truy cập biến

- Lệnh **la** được tách thành 2 lệnh lui và ori và sử dụng biến tạm \$1, trong đó lệnh lui để thực hiện gán nửa trên phần địa chỉ của biến vào \$1 và ori sẽ kết hợp với \$1 để gán phần nửa dưới của địa chỉ biến
- Địa chỉ của các biến có sự trùng khớp với phần hằng số của lệnh **la** (chia làm 2 lệnh **lui** và **ori**): Phần nửa trên trùng với phần hằng số trong lệnh **lui** và phần nửa dưới trùng với hằng số của lệnh **ori**
- Trên cửa sổ Register:

\$t8 trở thành 0x10010000 là địa chỉ của biến X

\$t9 trở thành 0x10010004 là địa chỉ của biến Y

Lệnh **lw \$t1, 0(\$t8)** lấy giá trị của biến có địa chỉ được lưu tại \$t8 và gán cho \$t1 => \$t1 trở thành 0x00000005 là giá trị của biến X

Tương tự với lệnh **lw \$t2, 0(\$t9)**: \$t2 trở thành 0xffffffff - giá trị của Y

Lệnh **sw \$s0, 0(\$t7)** lấy giá trị của thanh ghi \$s0 ghi vào biến có địa chỉ được lưu tại \$t7 => giá trị của Z được cập nhật tại cửa sổ Data Segment thành 0x00000009

- Các lệnh **lb** (load byte), **sb** (store byte): có chức năng tương tự **lw** và **sw** tuy nhiên chỉ thực hiện với 1 byte thay vì 4 byte cụ thể:
  - o **lb**: Lấy ra dữ liệu kiểu byte ( 8 bit) từ bộ nhớ thông qua địa chỉ trở đến ô nhớ đó và lưu vào thanh ghi
  - o **sb**: Ghi dữ liệu kiểu byte vào bộ nhớ thông qua con trỏ trỏ tới ô nhớ đó (lưu vào 8 bit thấp của ô nhớ)