**REPORT WEEK 1**

Họ và tên: Nguyễn Văn Hưng

MSSV: 20225634

Lớp: Thực hành kiến trúc máy tính – Mã lớp: 147799

1. **Assignment 1: lệnh gán số 16-bit**
2. Lệnh **addi $s0, $zero, 0x3007**: Thực hiện phép cộng giá trị **0x3007** với giá trị của thanh ghi **$zero** và lưu kết quả vào thanh ghi **$s0**. Do giá trị ban đầu của **$zero** là **0**, nên **$s0** sẽ được gán giá trị là **0x3007**. Giá trị của thanh ghi **pc** sẽ tăng 4 đơn vị để trỏ tới câu lệnh tiếp theo và trở thành **0x00400004.**





Lệnh **add $s0, $zero, $0**: Thực hiện phép cộng giá trị của **$zero** với chính nó (tức là **0 + 0 = 0**) và lưu kết quả vào thanh ghi **$s0**. Do đó, giá trị của **$s0** sẽ vẫn là **0**. Giá trị của thanh ghi **pc** sẽ tăng 4 đơn vị để trỏ tới câu lệnh tiếp theo và trở thành **0x00400008**.





1. Mã máy của lệnh **addi $s0, $zero, 0x3007** là **0x20103007**. Chuyển đổi này sang hệ cơ số 2, ta có:

0x20103007 = 0010 0000 0001 0000 0011 0000 0000 0101

* opcode là **001000** cho lệnh **addi** kiểu I-type
* rs: là **00000=0** số hiệu của thanh ghi nguồn là **$zero**
* rt: là **10000=16** số hiệu của thanh ghi đích (trong trường hợp này, là **$s0**)
* Giá trị ngay sau lệnh **addi là** 0011 0000 0000 0101 tức là **0x3007**

1. Mã máy của lệnh **add $s0, $zero, $0** là **0x00008020**. Chuyển đổi này sang hệ cơ số 2, ta có:

0x00008020 = 0000 0000 0000 0000 1000 0000 0010 0000

* opcode là **000000** cho lệnh **add** kiểu R-type
* rs: là **00000=0** số hiệu của thanh ghi nguồn 1 là **$zero**
* rt: là **00000=0** số hiệu của thanh ghi nguồn 2 là **$0 ($zero)**
* rd: là **10000=16** số hiệu của thanh ghi đích (trong trường hợp này, là **$s0**)
* shamt: là **00000** không sử dụng
* funct: **100000** (cho lệnh **add**)

1. Khi sửa lại lệnh **addi** như sau:

addi $s0, $zero, 0x2110003d

Lệnh này thực hiện phép cộng giá trị **0x2110003d** với giá trị của thanh ghi **$zero** .Do câu lệnh addi chỉ dùng với 16 bit nên khi dùng 32 bit câu lệnh sẽ chuyển thành 2 câu lệnh **lui** và **ori** .

* Lệnh **lui** (Load Upper Immediate) được sử dụng để chuyển giá trị 16-bit của hằng số lên 16 bit hàng cao nhất của thanh ghi đích.
* Lệnh **ori** (OR Immediate) được sử dụng để thực hiện phép OR giữa giá trị hiện tại của thanh ghi đích và giá trị 16-bit của hằng số.

Sau đó mới thực hiện lệnh cộng

1. **Assignment 2: lệnh gán số 32-bit**
2. Sự thay đổi giá trị của thanh ghi $s0:
   * Sau lệnh lui $s0, 0x2110, giá trị của $s0 sẽ là 0x21100000.
   * Sau lệnh ori $s0, $s0, 0x003d, giá trị của $s0 sẽ là 0x2110003d.
3. Sự thay đổi giá trị của thanh ghi $pc: Giá trị của thanh ghi $pc sẽ được cập nhật thêm 4 đơn vị sau mỗi lần thực thi lệnh.
4. Quan sát các byte trong vùng lệnh .text: Có thể chuyển tới cửa sổ Data Segment để xem các byte đầu tiên trong vùng lệnh và hoàn toàn trùng với cột trong cửa sổ Text Segment.

**A screenshot of a computer

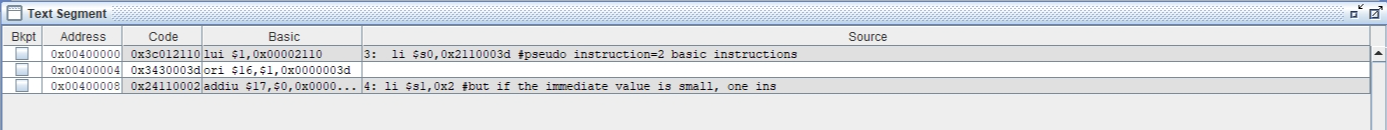
Description automatically generated**

1. **Assignment 3: lệnh gán (giả lệnh)**

Sau khi biên dịch chương trình và quan sát các lệnh mã máy trong cửa sổ Text Segment, ta sẽ nhận thấy điều bất thường là các lệnh được biên dịch không phải là hai lệnh cơ bản như mong đợi. Thay vào đó, chỉ có một số lượng lớn hơn các lệnh được tạo ra.

Trong trường hợp này, giá trị immediate là khá lớn (**0x2110003d**), điều này dẫn đến việc trình biên dịch phải tạo ra hai lệnh cơ bản (**lui** và **ori**) để đặt giá trị này vào thanh ghi **$s0** .

Tuy nhiên, với giá trị immediate nhỏ như **0x2**, trình biên dịch có thể tối ưu hóa bằng cách sử dụng chỉ một lệnh cơ bản. Điều này giải thích tại sao lệnh **li $s1, 0x2** chỉ biên dịch thành một lệnh cơ bản.



Điều bất thường ở đây là sự khác biệt về số lượng lệnh được tạo ra cùng một câu lệnh **li** dựa trên giá trị immediate của nó.

1. **Assignment 4: tính biểu thức 2x + y = ?**
2. Sự thay đổi giá trị của thanh ghi $s1:

* Sau lệnh addi $t1, $zero, 5 , giá trị của $s1 sẽ là 0x00000005.

Sự thay đổi giá trị của thanh ghi $s2:

* Sau lệnh addi $t2, $zero, -1, giá trị của $s1 sẽ là 0xffffffff.

Sự thay đổi giá trị của thanh ghi $s0:

* Sau lệnh add $s0, $t1, $t1, giá trị của $s0 sẽ là 0x0000000a.
* Sau lệnh add $s0, $s0, $t2, giá trị của $s0 sẽ là 0x00000009.

Sau khi kết thúc chương trình, kiểm tra giá trị của thanh ghi **$s0** đúng với kết quả mong đợi **2X + Y = 9.**

A screenshot of a computer

Description automatically generated

1. Lệnh **addi $t1, $zero, 5**:
   * Hợp ngữ: **addi $t1, $zero, 5**
   * Mã máy: **0x20090005**
   * Kiểu I-Type
   * Trong đó:
     + **0x20**: Mã opcode cho lệnh **addi**.
     + **0x09**: Số hiệu của thanh ghi đích **$t1**.
     + **0x00**: Số hiệu của thanh ghi nguồn **$zero**.
     + **0x05**: Giá trị ngay sau lệnh **addi**, tức là **5**.

Lệnh **addi $t2, $zero, -1**:

* + Hợp ngữ: **addi $t2, $zero, -1**
  + Mã máy: **0x200affff**
  + Kiểu I-Type
  + Trong đó:
    - **0x20**: Mã opcode cho lệnh **addi**.
    - **0x0a**: Số hiệu của thanh ghi đích **$t2**.
    - **0x00**: Số hiệu của thanh ghi nguồn **$zero**.
    - **0xffff**: Giá trị ngay sau lệnh **addi**, tức là **-1**.

Như vậy, các lệnh **addi** được mã hóa dưới dạng hợp ngữ và mã máy như trên.

1. Mã máy của lệnh **add $s0, $t1, $t1** là **0x01298020**. Chuyển đổi này sang hệ cơ số 2, ta có:

0x01298020 = 0000 0001 0010 1001 1000 0000 0010 0000

* opcode là **000000** cho lệnh **add** kiểu R-type
* rs: là **01001=9** số hiệu của thanh ghi nguồn 1 là **$s1**
* rt: là **01001=9** số hiệu của thanh ghi nguồn 2 là **$s1**
* rd: là **10000=16** số hiệu của thanh ghi đích (trong trường hợp này, là **$s0**)
* shamt: là **00000** không sử dụng
* funct: **100000** (cho lệnh **add**)

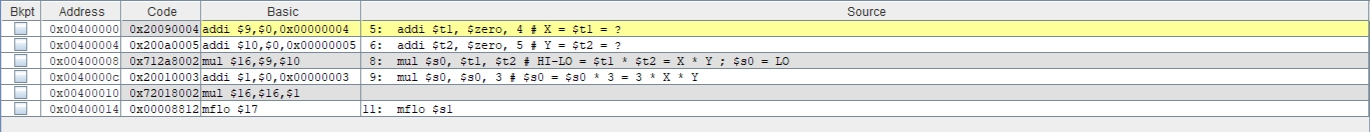
1. Mã máy của lệnh **add $s0, $s0, $t2** là **0x020a8020**. Chuyển đổi này sang hệ cơ số 2, ta có:

0x020a8020 = 0000 0010 0000 1010 1000 0000 0010 0000

* opcode là **000000** cho lệnh **add** kiểu R-type
* rs: là **10000=16** số hiệu của thanh ghi nguồn 1 là **$s0**
* rt: là **01010=9** số hiệu của thanh ghi nguồn 2 là **$t2**
* rd: là **10000=16** số hiệu của thanh ghi đích (trong trường hợp này, là **$s0**)
* shamt: là **00000** không sử dụng
* funct: **100000** (cho lệnh **add**)

1. **Assignment 5: phép nhân**

* Sau khi biên dịch chương trình và quan sát các lệnh mã máy trong cửa sổ Text Segment, ta có thể thấy điều bất thường là lệnh **mul** được sử dụng hai lần trong chương trình này. Điều này là do trong kiến trúc MIPS, chỉ có lệnh **mul** duy nhất để nhân hai số và lưu kết quả vào thanh ghi **HI** và **LO**. Điều này gây ra một vấn đề nếu chúng ta muốn nhân một số với một hằng số khác không phải là 2.
* Sau đó,sử dụng công cụ gỡ lỗi để chạy từng lệnh và quan sát sự thay đổi của các thanh ghi, đặc biệt là **HI** và **LO**.
* Sau lệnh addi $l, $0,0x00000003 , giá trị của $sl sẽ là 0x00000014 = 20 .
* Sau lệnh mul $s0, $s0, 3, giá trị của $l sẽ là 0x0000003c = 60.
* Cuối cùng, sau khi kết thúc chương trình, giá trị của thanh ghi **$s1 = 0x0000003c = 60** đúng với kết quả của biểu thức **Z = 3\*XY = 3\*4\*5 = 60**.



1. **Assignment 6: tạo biến và truy cập biến**

A screenshot of a computer

Description automatically generated

Sau khi biên dịch chương trình và quan sát các lệnh mã máy trong cửa sổ Text Segment, ta sẽ thấy lệnh **la** được biên dịch như sau:

* Lệnh **la** sẽ biên dịch thành hai lệnh MIPS:
  + Lệnh **lui** để tải giá trị hàng cao nhất của địa chỉ của biến vào thanh ghi.
  + Lệnh **ori** để tải giá trị hàng thấp nhất của địa chỉ của biến vào thanh ghi.
* Trong cửa sổ Labels, bạn có thể so sánh địa chỉ của các biến X, Y, Z với giá trị được tải bằng các lệnh **la**. Ta cũng có thể xác minh giá trị của các biến bằng cách nhấp đúp vào các biến X, Y, Z trong cửa sổ Data Segment.

A screenshot of a computer

Description automatically generated

* **Lệnh lw (load word)**:
  + Vai trò: Lệnh **lw** được sử dụng để tải dữ liệu từ bộ nhớ vào thanh ghi.
  + Cú pháp: **lw $t, offset($s)**
    - **$t**: Thanh ghi đích, nơi dữ liệu được tải vào.
    - **offset**: Độ lệch (offset) so với địa chỉ được lưu trong thanh ghi **$s**.
    - **$s**: Thanh ghi chứa địa chỉ cơ sở (base address) của vùng nhớ mà chúng ta muốn truy cập.
  + Hoạt động: Lệnh **lw** tải một từ (word) từ bộ nhớ tại địa chỉ được tính bằng cách cộng địa chỉ cơ sở và offset, sau đó lưu giá trị này vào thanh ghi đích **$t**.
* **Lệnh sw (store word)**:
  + Vai trò: Lệnh **sw** được sử dụng để lưu dữ liệu từ thanh ghi vào bộ nhớ.
  + Cú pháp: **sw $t, offset($s)**
    - **$t**: Thanh ghi chứa dữ liệu cần lưu vào bộ nhớ.
    - **offset**: Độ lệch (offset) so với địa chỉ được lưu trong thanh ghi **$s**.
    - **$s**: Thanh ghi chứa địa chỉ cơ sở (base address) của vùng nhớ mà chúng ta muốn truy cập.
  + Hoạt động: Lệnh **sw** lưu một từ (word) từ thanh ghi **$t** vào bộ nhớ tại địa chỉ được tính bằng cách cộng địa chỉ cơ sở và offset.
* Lệnh **lb** và **sb** được sử dụng để tải và lưu byte từ/đến bộ nhớ. **lb** sẽ tải một byte từ bộ nhớ và ký hiệu mở rộng nó thành một word, trong khi **sb** sẽ lưu một byte từ thanh ghi vào bộ nhớ.
  + Top of Form