**REPORT WEEK 4**

Họ và tên: Nguyễn Văn Hưng

MSSV: 20225634

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

1. **Assignment 1:**
2. Đề bài: Create a new project to implement the Home Assigment 1. Compile and upload to simulator. Initialize two operands (register $s1 and $s2), run this program step by step, observe memory and registers value
3. Code:

.text

li $s1, 111

li $s2, -120

start:

li $t0,0 # No Overflow is default status

addu $s3,$s1,$s2 # s3 = s1 + s2

xor $t1,$s1,$s2 # Test if $s1 and $s2 have the same sign

bltz $t1,EXIT # If not, exit

slt $t2,$s3,$s1

bltz $s1,NEGATIVE # Test if $s1 and $s2 is negative?

beq $t2,$zero,EXIT # s1 and $s2 are positive

# if $s3 > $s1 then the result is not overflow

j OVERFLOW

NEGATIVE:

bne $t2,$zero,EXIT # s1 and $s2 are negative

# if $s3 < $s1 then the result is not overflow

OVERFLOW:

li $t0,1 # the result is overflow

EXIT:

1. Text segment:

**A screenshot of a computer

Description automatically generated**

1. Giải thích:

Đoạn mã MIPS này thực hiện các phép toán số học và kiểm tra tràn số (overflow). Dưới đây là phân tích từng lệnh:

* li $s1, 111: Load immediate. Gán giá trị 111 vào thanh ghi $s1.
* li $s2, 120: Load immediate. Gán giá trị 120 vào thanh ghi $s2.
* start:: Đánh dấu điểm bắt đầu của vòng lặp.
* li $t0, 0: Gán giá trị 0 vào thanh ghi $t0. Đây là trạng thái mặc định để không có tràn số.
* addu $s3, $s1, $s2: Thực hiện phép cộng không dấu của $s1 và $s2, kết quả được lưu vào thanh ghi $s3.
* xor $t1, $s1, $s2: Thực hiện phép XOR giữa $s1 và $s2. Nếu $s1 và $s2 có cùng dấu, $t1 sẽ bằng 0, ngược lại là 1.
* bltz $t1, EXIT: Nếu $t1 là số âm (khác 0), tức là $s1 và $s2 không có cùng dấu, nhảy tới nhãn EXIT.
* slt $t2, $s3, $s1: So sánh $s3 với $s1. Nếu $s3 < $s1, $t2 sẽ bằng 1.
* bltz $s1, NEGATIVE: Nếu $s1 là số âm, nhảy tới nhãn NEGATIVE.
* beq $t2, $zero, EXIT: Nếu $s1 và $s2 dương và $s3 > $s1, tức là không có tràn số, nhảy tới nhãn EXIT.
* j OVERFLOW: Nhảy tới nhãn OVERFLOW.
* NEGATIVE:: Nhãn NEGATIVE.
* bne $t2, $zero, EXIT: Nếu $s1 và $s2 đều âm và $s3 < $s1, tức là không có tràn số, nhảy tới nhãn EXIT.
* OVERFLOW:: Nhãn OVERFLOW.
* li $t0, 1: Gán giá trị 1 vào thanh ghi $t0 để chỉ ra tràn số.
* EXIT: Nhãn EXIT.
* Nếu kết quả của phép cộng không vượt quá giới hạn của một thanh ghi MIPS (trong trường hợp này là 32 bit), thì không có tràn số và giá trị trong thanh ghi $t0 sẽ là 0. Nếu có tràn số, giá trị trong $t0 sẽ là 1.

**Top of Form**

1. Kết quả:

* Trường hợp 1:

Với sum = 111 + (-120) = -9 thì không gây tràn số





* Trường hợp 2:

Với sum = 2147483647 + 17 thì gây tràn số

****

****

* Trường hợp 3:

Với sum = 111 + 120 = 231 thì không gây tràn số



****

* Trường hợp 4:

Với sum = (-2147483647) + (-17) thì gây tràn số

****



* Trường hợp 5:

Với sum = (-111) + (-120) = -231 thì không gây tràn số





1. **Assignment 2:**
2. Đề bài: Write a program to do the following tasks:

* Extract MSB of $s0
* Clear LSB of $s0
* Set LSB of $s0 (bits 7 to 0 are set to 1)
* Clear $s0 ($s0=0, must use logical instructions)

MSB: Most Significant Byte

LSB: Least Significant

1. Code:

.text

li $s0, 0x12345678

andi $t0,$s0,0xff000000

srl $t0,$t0,24 #Trích xuất 8 bit đầu (Kết quả ở thanh $t0)

andi $t1,$s0,0xffffff00 #Xoá 8 bit cuối (Kết quả ở thanh $t1)

ori $t2,$s0,0x000000ff #Biến 8 bit cuối thành 1 (Kết quả ở thanh $t2)

andi $s0,$s0, 0 #Xoá sạch $s0

1. Text segment:

A screenshot of a computer

Description automatically generated

1. Giải thích:

* Đoạn mã MIPS trên thực hiện các thao tác trên giá trị được lưu trong thanh ghi $s0. Dưới đây là phân tích từng lệnh:
* **li $s0, 0x12345678**: Gán giá trị hex 0x12345678 vào thanh ghi $s0.
* **andi $t0, $s0, 0xff000000**: Thực hiện phép AND giữa giá trị của $s0 và hằng số 0xff000000. Phép AND này sẽ giữ lại 8 bit đầu tiên của $s0 và đặt kết quả vào thanh ghi $t0.
* **srl $t0, $t0, 24**: Dịch phải (Shift Right Logical) thanh ghi $t0 đi 24 bit. Điều này trích xuất 8 bit đầu tiên đã được lấy từ $s0 và di chuyển chúng về phải để chúng nằm ở vị trí thấp nhất của thanh ghi $t0.
* **andi $t1, $s0, 0xffffff00**: Thực hiện phép AND giữa giá trị của $s0 và hằng số 0xffffff00. Phép AND này xoá đi 8 bit cuối cùng của $s0 và đặt kết quả vào thanh ghi $t1.
* **ori $t2, $s0, 0x000000ff**: Thực hiện phép OR giữa giá trị của $s0 và hằng số 0x000000ff. Phép OR này đặt 8 bit cuối cùng của $s0 thành 1 và đặt kết quả vào thanh ghi $t2.
* **andi $s0, $s0, 0**: Thực hiện phép AND giữa giá trị của $s0 và hằng số 0. Phép AND này xoá sạch tất cả các bit trong $s0 và đặt kết quả vào chính thanh ghi $s0, làm cho $s0 trở thành 0.

1. Kết quả:



1. **Assignment 3:**
2. Đề bài: Pseudo instructions in MIPS are not-directly-run-on-MIPS-processor instructions which need to be converted to real-instructions of MIPS. Re-write the following pseudo instructions using real-instructions understood by MIPS processors:

a. abs $s0,$s1

$s0 <= | $s1 |

b. move $s0,$s1

$s0 <= $s1

c. not $s0, $s1

$s0 <= bit invert ($s1)

d. ble $s1,$s2,label

if ($s1 <= $s2)

j label

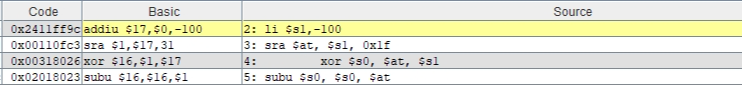
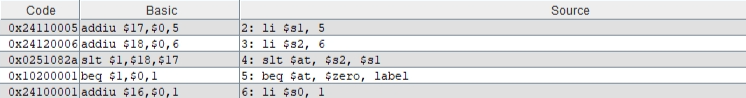
1. Code:
2. sra $at, $s1, 0x1f

xor $s0, $at, $s1

subu $s0, $s0, $at

1. addu $s0, $zero, $s1
2. nor $s0, $s1, $zero
3. slt $at, $s2, $s1

beq $at, $zero, label

1. Text segment:
2. 
3. 
4. 
5. 
6. Giải thích:
7. Đoạn mã MIPS trên thực hiện một loạt các phép toán số học và logic trên giá trị lưu trong thanh ghi $s1. Dưới đây là phân tích từng lệnh:

* **li $s1, -100**: Gán giá trị -100 vào thanh ghi $s1.
* **sra $at, $s1, 0x1f**: Dịch phải dấu (Arithmetic Shift Right) của $s1 đi 31 bit (tương đương với việc mở rộng dấu). Kết quả được lưu vào thanh ghi $at.
* **xor $s0, $s1, $s1**: Thực hiện phép XOR giữa $s1 và chính nó. Điều này dẫn đến việc xóa tất cả các bit của $s1 (do XOR với chính nó cho kết quả là 0), và kết quả được lưu vào thanh ghi $s0.
* **subu $s0, $s0, $at**: Thực hiện phép trừ không dấu (Unsigned Subtract) giữa $s0 và $at, và kết quả được lưu vào thanh ghi $s0.
* Mã này có thể được sử dụng để tính toán giá trị tuyệt đối của một số nguyên trong MIPS, tức là tính toán |x| của giá trị số nguyên x. Để tính toán giá trị tuyệt đối, ta cần thực hiện hai bước chính:
* Mở rộng dấu giá trị số nguyên (để đảm bảo nó đang ở dạng dấu 2) bằng cách dịch phải dấu đi 31 bit.
* Sau đó, thực hiện phép XOR giữa giá trị ban đầu và chính nó để xóa dấu của nó.
* Cuối cùng, trừ kết quả từ bước 2 cho giá trị được mở rộng dấu từ bước 1 để loại bỏ các bit dấu.
* Kết quả cuối cùng trong $s0 sẽ là giá trị tuyệt đối của $s1.

1. Đoạn mã MIPS này thực hiện việc gán giá trị của thanh ghi $s1 vào thanh ghi $s0 bằng cách sử dụng lệnh **addu**. Dưới đây là phân tích của lệnh này:

* **li $s1, -100**: Gán giá trị -100 vào thanh ghi $s1.
* **addu $s0, $zero, $s1**: Thực hiện phép cộng không dấu giữa giá trị trong thanh ghi $zero và giá trị trong thanh ghi $s1, kết quả được lưu vào thanh ghi $s0. Lưu ý rằng phép cộng không dấu này thực ra không làm thay đổi giá trị, nó chỉ diễn đạt ý nghĩa "gán" giá trị từ một thanh ghi vào một thanh ghi khác một cách rõ ràng. Trong trường hợp này, nó gán giá trị của $s1 vào $s0 mà không làm thay đổi giá trị của $s1.
* Lệnh này tương đương với lệnh **move $s0, $s1**, nó chỉ dịch chuyển giá trị của $s1 vào $s0 mà không thay đổi giá trị.

1. Đoạn mã MIPS này thực hiện phép NOT (hoặc là phép NOR với giá trị 0) trên giá trị lưu trong thanh ghi $s1 và lưu kết quả vào thanh ghi $s0. Dưới đây là phân tích của lệnh:

* **li $s1, 100**: Gán giá trị 100 vào thanh ghi $s1.
* **nor $s0, $s1, $zero**: Thực hiện phép NOR giữa giá trị trong thanh ghi $s1 và giá trị trong thanh ghi $zero. Phép NOR trả về kết quả là 0 khi cả hai toán hạng đều là 1, ngược lại trả về 1. Trong trường hợp này, vì $zero luôn là 0, kết quả phép NOR sẽ là phản của giá trị trong $s1. Kết quả của phép NOR được lưu vào thanh ghi $s0.
* Vì $zero luôn chứa giá trị 0, phép NOR với $zero đơn giản là phép NOT trên giá trị trong thanh ghi $s1. Do đó, lệnh này thực hiện việc đảo ngược tất cả các bit của giá trị trong $s1 và lưu vào $s0.

1. Đoạn mã MIPS này thực hiện một phép so sánh giữa hai giá trị lưu trong các thanh ghi $s1 và $s2, sau đó sử dụng một lệnh nhảy điều kiện để thực hiện một hành động nếu kết quả của phép so sánh là 0 (tức là không đúng). Dưới đây là phân tích chi tiết:

* **li $s1, 5**: Gán giá trị 5 vào thanh ghi $s1.
* **li $s2, 6**: Gán giá trị 6 vào thanh ghi $s2.
* **slt $at, $s2, $s1**: Thực hiện phép so sánh dưới dạng "set less than" (SLT) giữa giá trị trong $s2 và giá trị trong $s1. Kết quả của phép so sánh sẽ được lưu vào thanh ghi $at. Nếu $s2 < $s1, kết quả sẽ là 1, ngược lại sẽ là 0.
* **beq $at, $zero, label**: Lệnh nhảy điều kiện này kiểm tra giá trị của thanh ghi $at. Nếu giá trị của $at bằng 0 (tức là kết quả của phép so sánh SLT là 0, tức là $s2 không nhỏ hơn $s1), lệnh nhảy sẽ được thực hiện, và chương trình sẽ nhảy đến nhãn **label**.
* **li $s0, 1**: Nếu lệnh nhảy không được thực hiện (tức là $s2 >= $s1), giá trị 1 sẽ được gán vào thanh ghi $s0.
* **label:**: Nhãn **label**.
* Vậy nếu $s2 < $s1 thì giá trị 1 sẽ được gán vào thanh ghi $s0, ngược lại nếu $s2 không nhỏ hơn $s1 thì không có gì xảy ra.

Top of Form

1. Kết quả:
2. 
3. 
4. 
5. 
6. **Assignment 4:**
7. Đề bài: To dectect overflow in additional operation, we also use other rule than the one in Assignment 1. This rule is: when add two operands that have the same sign, overflow will occur if the sum doesn’t have the same sign with either operands. You need to use this rule to write another overflow detection program.
8. Code:

.text

li $s0, 0

li $s1, 0

li $t0, 0 # Mac dinh t0 = 0 la trang thai khong xay ra Overflow

addu $t2, $s0, $s1 # Tinh tong s0 va s1

xor $t1, $s0, $s1 # Neu hai so s0 va s1 cung dau thi t1 > 0

blez $t1, Exit # Neu t1 < 0 (khac dau) thi nhay den Exit

xor $t1, $s1, $t2 # Neu hai so s1 va t2 cung dau thi t1 > 0

bgez $t1, Exit # Neu t1 > 0 (cung dau) thi nhay den Exit

Overflow:

li $t0, 1 # Overflow xay ra

Exit:

1. Text segment:

**A screenshot of a computer

Description automatically generated**

1. Giải thích:

Đoạn mã MIPS này thực hiện các phép toán số học và logic trên giá trị lưu trong các thanh ghi $s0 và $s1, sau đó kiểm tra xem có xảy ra tràn số (overflow) không. Dưới đây là phân tích chi tiết:

* **li $s0, 100**: Gán giá trị 100 vào thanh ghi $s0.
* **li $s1, -111**: Gán giá trị -111 vào thanh ghi $s1.
* **li $t0, 0**: Gán giá trị 0 vào thanh ghi $t0. Giá trị này sẽ được sử dụng để biểu thị trạng thái không xảy ra tràn số (overflow) mặc định.
* **addu $t2, $s0, $s1**: Thực hiện phép cộng không dấu giữa giá trị trong các thanh ghi $s0 và $s1, kết quả được lưu vào thanh ghi $t2.
* **xor $t1, $s0, $s1**: Thực hiện phép XOR giữa giá trị trong các thanh ghi $s0 và $s1. Nếu hai số $s0 và $s1 cùng dấu, $t1 sẽ là số dương (> 0).
* **blez $t1, Exit**: Lệnh nhảy điều kiện này kiểm tra xem giá trị của thanh ghi $t1 có nhỏ hơn hoặc bằng 0 không. Nếu như $t1 < 0, tức là hai số $s0 và $s1 không cùng dấu, lệnh nhảy sẽ được thực hiện, và chương trình sẽ nhảy tới nhãn **Exit**, bỏ qua phần code xử lý tràn số.
* **xor $t1, $s1, $t2**: Thực hiện phép XOR giữa giá trị trong thanh ghi $s1 và giá trị trong thanh ghi $t2. Nếu hai số $s1 và $t2 cùng dấu, $t1 sẽ là số dương (> 0).
* **bgez $t1, Exit**: Lệnh nhảy điều kiện này kiểm tra xem giá trị của thanh ghi $t1 có lớn hơn hoặc bằng 0 không. Nếu như $t1 > 0, tức là hai số $s1 và $t2 cùng dấu, lệnh nhảy sẽ được thực hiện, và chương trình sẽ nhảy tới nhãn **Exit**, bỏ qua phần code xử lý tràn số.
* **Overflow:**: Nhãn **Overflow**, đánh dấu nơi xử lý khi tràn số xảy ra.
* **li $t0, 1**: Gán giá trị 1 vào thanh ghi $t0 để biểu thị rằng tràn số đã xảy ra.
* **Exit:**: Nhãn **Exit**, chương trình tiếp tục thực hiện tại đây sau khi kiểm tra xong về trạng thái của các số liệu.Top of Form

1. Kết quả:

* Trường hợp 1:

Với sum = 111 + (-120) = -9 thì không gây tràn số

A table with numbers and letters

Description automatically generated with medium confidence

* Trường hợp 2:

Với sum = 2147483647 + 17 thì gây tràn số

**A screenshot of a table

Description automatically generated**

* Trường hợp 3:

Với sum = 111 + 120 = 231 thì không gây tràn số

A screenshot of a computer

Description automatically generated

* Trường hợp 4:

Với sum = (-2147483647) + (-17) thì gây tràn số

A screenshot of a table

Description automatically generated

* Trường hợp 5:

Với sum = (-111) + (-120) = -231 thì không gây tràn số

A table with numbers and a few words

Description automatically generated with medium confidence

1. **Assignment 5:**
2. Đề bài: Write a program that implement multiply by a small power of 2. (2, 4, 8, 16, etc for example).
3. Code:

.text

addi $s0, $zero, 11 #Dua so bi nhan vao thanh ghi $s0

addi $s1, $zero, 2048 #Dua so nhan vao thanh ghi $s1

addi $t0, $zero, 1 #Cai dat thanh $t0 co gia tri 1

loop:

beq $s1, $t0, exit #Neu $s1 chi con gia tri la 1 thi ket thuc vong lap

sll $s0, $s0, 1 #Tang gia tri thanh ghi $s0 len 2 lan

srl $s1, $s1, 1 #Giam gia tri thanh ghi $s1 di 2 lan

j loop #Lap lai

exit:

1. Text segment:

**A screenshot of a computer

Description automatically generated**

1. Giải thích:

Đoạn mã MIPS này thực hiện việc nhân một số với một lũy thừa của 2 (power of 2). Dưới đây là phân tích chi tiết của mã:

* **addi $s0, $zero, 11**: Gán giá trị 11 vào thanh ghi $s0. Đây là số cần nhân.
* **addi $s1, $zero, 2048**: Gán giá trị 2048 vào thanh ghi $s1. Đây là lũy thừa của 2.
* **addi $t0, $zero, 1**: Gán giá trị 1 vào thanh ghi $t0. Đây là giá trị để kiểm tra xem lũy thừa của 2 có giảm về 1 hay không.
* **loop:**: Nhãn bắt đầu vòng lặp.
* **beq $s1, $t0, exit**: Kiểm tra nếu giá trị trong $s1 (số nhân) bằng 1 thì nhảy đến nhãn **exit**, kết thúc vòng lặp.
* **sll $s0, $s0, 1**: Dịch trái (Shift Left Logical) giá trị trong $s0 đi 1 bit. Điều này tương đương với việc nhân $s0 với 2.
* **srl $s1, $s1, 1**: Dịch phải (Shift Right Logical) giá trị trong $s1 đi 1 bit. Điều này tương đương với việc chia $s1 cho 2.
* **j loop**: Nhảy về nhãn **loop** để tiếp tục vòng lặp.
* **exit:**: Nhãn **exit**, kết thúc vòng lặp.

1. Kết quả:

****