Tên: Võ Văn Phúc
MSSV: 22521147

BÀI TẬP (ASSEMBLY)

Bài 1:

Phần 1:

Câu 1.1:

a, f = g + h + i + j

Mã hợp ngữ MIPS:

```
add f, g, h
add f, f, i
add f, f, j
```

b, f = g + (h + 5)

Mã hợp ngữ MIPS:

```
add f, h, 5
addi f, f, g
```

Câu 1.2:

a, Có 3 lệnh MIPS để thực hiện các lệnh C trên

b, Có 2 lệnh MIPS để thực hiện các lệnh C trên

Câu 1.3:

a, f = 14

b, f = 10

Phần 2:

Câu 1.4:

a, add f, g, h —> f = g + h

b, addi f, f, 1 —> f = f + 1
add f, g, h —> f = g + h

Câu 1.5:

a, f = 5

b, f = 5

Bài 2:

Phần 1:

Câu 2.1:

a, f = g + h + B[4]

Chuyển sang MIPS:

```
add $t0, $s1, $s2
lw $t1, 16($s7)
add $s0, $t0, $t1
```

b, f = g - A[B[4]]

Chuyển sang MIPS:

```
lw $t0, 16($s7)
sll $t1, $t0, 2
```

add $t2, $t1, $s6
lw $t3, 0($t2)
sub $s0, $s1, $t3

Câu 2.2:
a, Cần 3 lệnh hợp ngữ MIPS để có chức năng tương đương với từng câu lệnh C : f = g + h + B[4]
b, Cần 5 lệnh hợp ngữ MIPS để có chức năng tương đương với từng câu lệnh C : f = g - A[B[4]]

Câu 2.3:
a, Có 5 thanh ghi khác nhau được dùng cho từng câu lệnh C bên trên : $t0, $s1, $s2, $t1, $s7, $s0
b, Có 8 thanh ghi khác nhau được dùng cho từng câu lệnh C bên trên : $t0, $s7, $t1, $t2, $s6, $t3, $s0, $s1

Phần 2:
Câu 2.4:
a, Chuyển sang câu lệnh C:
add $s0, $s0, $s1 —> f = f + g
add $s0, $s0, $s2 —> f = f + h
add $s0, $s0, $s3 —> f = f + i
add $s0, $s0, $s4 —> f = f + j
—> f = f + g + h + i + j
b, Chuyển sang câu lệnh C:
lw $s0, 4($s6) —> f = A[1]

Câu 2.5:
a, Không thể rút gọn hơn được nữa
b, Không thể rút gọn hơn được nữa

Câu 2.6:
a, Có 5 thanh ghi được sử dụng trong đoạn hợp ngữ trên: $s0, $s1, $s2, $s3, $s4
Không thể rút gọn được hơn nữa
b, Có 2 thanh ghi được sử dụng trong đoạn hợp ngữ trên: $s0, $s6
Không thể rút gọn được hơn nữa

Bài 3:
Câu 3.1:
a, f = -g + h + B[1]
Chuyển sang MIPS:
lw $t0, 4($s7)
add $t1, $s2, $t0
sub $s0, $t1, $s1

b, f = A[B[g] + 1]
Chuyển sang MIPS:
sll $s1, $s1, 2

```
add $t0, $s1, $s7
lw $t1, 0($t0)
addi $t2, $t1, 1
sll $t2, $t2, 2
add $t3, $t2, $s6
lw $s0, 0($t3)
```

Câu 3.3:

a, Có 6 thanh ghi khác nhau được sử dụng cho từng câu lệnh C bên trên: $t0, $s7, $t1, $s2, $s0, $s1

b, Có 8 thanh ghi khác nhau được sử dụng cho từng câu lệnh C bên trên: $s1, $s6, $s7, $s0, $t0, $t1, $t2, $t3

Bài 4:

Câu 4.1:

a, $s0 = 0x70000000 , # s0 = 1879048192 (Hệ 10)
$s1 = 0x0FFFFFFF , # s1 = 268435455 (Hệ 10)
add $t0, $s0, $s1

Kết quả là: $t0 = 0x7FFFFFFF , # t0 = 2147483647 (Hệ 10)

Kết quả như mong muốn và không bị tràn

b, $s0 = 0x40000000 , # s0 = 1073741824 (Hệ 10) = 01000000000000000000000000000000 (Hệ 2 - 32 bits)
$s1 = 0x40000000 , # s1 = 1073741824 (Hệ 10) = 01000000000000000000000000000000 (Hệ 2 - 32 bits)
add $t0, $s0, $s1

Kết quả là: $t0 = 10000000000000000000000000000000

Bù 2 của $t0 = 10000000000000000000000000000000 (Hệ 2 - 32 bits) = -2147483648 (Hệ 10)

Kết quả trong thanh ghi $t0 không đúng như mong muốn và có xảy ra tràn vì phép cộng hai số dương nhưng kết quả lại là số âm

Câu 4.2:

a, $s0 = 0x70000000 , # s0 = 1879048192 (Hệ 10)
$s1 = 0x0FFFFFFF , # s1 = 268435455 (Hệ 10)
sub $t0, $s0, $s1

Kết quả là: $t0 = 0x60000001 , # t0 = 1610612737 (Hệ 10)

Kết quả như mong muốn và không xảy ra tràn

b, $s0 = 0x40000000 , # s0 = 1073741824 (Hệ 10) = 01000000000000000000000000000000 (Hệ 2 - 32 bits)
$s1 = 0x40000000 , # s1 = 1073741824 (Hệ 10) = 01000000000000000000000000000000 (Hệ 2 - 32 bits)
sub $t0, $s0, $s1

Kết quả là: $t0 = 0

Kết quả như mong muốn và không xảy ra tràn

Câu 4.3:

a, $s0 = 0x40000000 , # s0 = 1073741824 (Hệ 10) =
01000000000000000000000000000000 (Hệ 2 - 32 bits)
$s1 = 0x40000000 , # s1 = 1073741824 (Hệ 10) =
01000000000000000000000000000000 (Hệ 2 - 32 bits)
add $t0, $s0, $s1 , # t0 = s0 + s1
add $t0, $t0, $s0 , # t0 = t0 + s0
Kết quả là: $t0 = 0x70000000 + (0x70000000 + 0x0FFFFFFF) =
0xEFFFFFFF(11101111111111111111111111111111)
$t0 nhận được có giá trị là : -268435457 (Hệ 10)
Kết quả trong thanh ghi $t0 không đúng như mong muốn và có xảy ra tràn vì phép cộng các số dương nhưng kết quả lại là số âm

b, $s0 = 0x40000000 , # s0 = 1073741824 (Hệ 10) =
01000000000000000000000000000000 (Hệ 2 - 32 bits)
$s1 = 0x40000000 , # s1 = 1073741824 (Hệ 10) =
01000000000000000000000000000000 (Hệ 2 - 32 bits)
add $t0, $s0, $s1 , # t0 = s0 + s1
add $t0, $t0, $s0 , # t0 = t0 + s0
Kết quả là: $t0 = 0x40000000 + (0x40000000 + 0x40000000) =
0xC0000000(11000000000000000000000000000000)
$t0 nhận được có giá trị là: -1073741824
Kết quả trong thanh ghi $t0 không đúng như mong muốn và có xảy ra tràn vì phép cộng các số dương nhưng kết quả lại là số âm

Bài 5:
Câu 5.1 & 5.2:
a, opcode = 101011 (Hệ 2) = 2B (Hệ 16)
—> Lệnh sw —> Thuộc kiểu lệnh I-type
rs = 10000 (Hệ 2) = 16 (Hệ 10)
—> Thanh ghi thứ 16 —> Thanh ghi $s0
rt = 01011 (Hệ 2) = 11 (Hệ 10)
—> Thanh ghi thứ 11 —> Thanh ghi $t3
immediate (16 bits) = 0000 0000 0000 0100 (Hệ 2) = 4 (Hệ 10)
Kết luận: sw $t3, 4($s0)

b, opcode = 100011 (Hệ 2) = 23 (Hệ 16)
—> Lệnh lw —> Thuộc kiểu lệnh I-type
rs = 01000 (Hệ 2) = 8 (Hệ 10)
—> Thanh ghi thứ 8 —> Thanh ghi $t0
rt = 01000 (Hệ 2) = 8 (Hệ 10)
—> Thanh ghi thứ 8 —> Thanh ghi $t0
immediate (16 bits) = 0000 0000 0100 0000 = 64 (Hệ 10)
Kết luận: lw $t0, 64($t0)

Câu 5.3:
a, 10101110000010110000000000000100 (Hệ 2 - 32 bits) = 0xAE0B0004 (Hệ 16)
b, 10001101000010000000000001000000 (Hệ 2 - 32 bits) = 0x8D080040 (Hệ 16)

Câu 5.4 & 5.5:

a, add $t0, $t0, $zero

—> Dạng R-type

opcode = 0 (Hệ 16) = 000000 (Hệ 2 - 6 bits)

rs = 01000 (Hệ 2 - 5 bits) , vì $t0 là thanh ghi thứ 8

rt = 00000 (Hệ 2 - 5 bits), vì $zero là thanh ghi thứ 0

rd = 01000 (Hệ 2 - 5 bits), vì $t0 là thanh ghi thứ 8

shamt = 00000 (Hệ 2 - 5 bits), vì không được sử dụng

funct = 20 (Hệ 16) = 100000 (Hệ 2 - 6 bits)

Kết luận: Mã máy: 0000 0001 0000 0000 0100 0000 0010 0000 (Hệ 2) = 0x1004020 (Hệ 16)

b, lw $t1, 4($s3)

—> Dạng I-type

opcode = 23 (Hệ 16) = 100011 (Hệ 2 - 6 bits)

rs = 10011 (Hệ 2 - 5 bits), vì $s3 là thanh ghi thứ 19

rt = 01001 (Hệ 2 - 5 bits), vì $t1 là thanh ghi thứ 9

immediate = 4 (Hệ 10) = 0000 0000 0000 0100 (Hệ 2 - 16 bits)

Kết luận: Mã máy: 1000 1110 0110 1001 0000 0000 0000 0100 (Hệ 2 - 32 bits) = 0x8E690004 (Hệ 16)

Câu 5.6:

a, opcode = 0x0

rs = 0x8

rt = 0x0

rd = 0x8

shamt = 0x0

funct = 0x20

b, opcode = 0x23

rs = 0x19

rt = 0x9

immediate = 0x4

Bài 6:

Câu 6.1:

a, $t0 = 0x55555555 (Hệ 16) —> 01010101010101010101010101010101 (Hệ 2)

$t1 = 0x12345678 (Hệ 16) —> 00010010001101000101011001111000 (Hệ 2)

sll $t2, $t0, 4

Kết quả: $t2 = 11101010110111111110111011010000 (Hệ 2)

or $t2, $t2, $t1

Kết quả: $t2 = 11111110111111111111111011011110 (Hệ 2) —> 0xFEFFFEDE (Hệ 16)

b, $t0 = 0xBEADFEED (Hệ 16) —> 10111110101011011111111011101101 (Hệ 2)

$t1 = 0xDEADFADE (Hệ 16) —> 11011110101011011111101011011110 (Hệ 2)

sll $t2, $t0, 4

Kết quả: $t2 = 11101010110111111110111011010000 (Hệ 2)

or $t2, $t2, $t1

Kết quả: $t2 = 11111110111111111111111011011110 (Hệ 2) —> 0xFEFFFEDE (Hệ 16)

Câu 6.2:

a, $t0 = 0x55555555 (Hệ 16) —> 01010101010101010101010101010101 (Hệ 2)
$t1 = 0x12345678 (Hệ 16) —> 00010010001101000101011001111000 (Hệ 2)
srl $t2, $t0, 3
Kết quả: $t2 = 00001010101010101010101010101010 (Hệ 2)
andi $t2, $t2, 0xFFEF
Kết quả: $t2 = 00000000000000001010101010101010 (Hệ 2) —> 0x0000AAAA (Hệ 16)
b, $t0 = 0xBEADFEED (Hệ 16) —> 10111110101011011111111011101101 (Hệ 2)
$t1 = 0xDEADFADE (Hệ 16) —> 11011110101011011111101011011110 (Hệ 2)
srl $t2, $t0, 3
Kết quả: $t2 = 00010111110101011011111111011101 (Hệ 2)
andi $t2, $t2, 0xFFEF
Kết quả: $t2 = 00000000000000001010101010101010 (Hệ 2) —> 0x0000AAAA (Hệ 16)

Bài 7:
Câu 7.1:
a, slt $t2, $t0, $t1
Ta có: Câu lệnh “slt” làm việc với số có dấu. Nếu thanh ghi $t0 nhỏ hơn thanh ghi $t1 thì thanh ghi $t2 có giá trị là 1, ngược lại thì thanh ghi $t2 có giá trị là 0.
$t0 = 1010 1101 0001 0000 0000 0000 0000 0010 (Hệ 2 - 32 bits)
$t1 = 0011 1111 1111 1000 0000 0000 0000 0000 (Hệ 2 - 32 bits)
Ta thấy thanh ghi $t0 có giá trị âm và thanh ghi $t1 có giá trị dương nên $t0 < $t1 nên $t2 = 1
beq $t2, $zero, ELSE
Ta thấy câu lệnh bằng không xảy ra nên câu lệnh “J DONE” được thực hi
Kết luận: $t2 = 1

b, slt $t2, $t0, $t1
Ta có: Câu lệnh “slt” làm việc với số có dấu. Nếu thanh ghi $t0 nhỏ hơn thanh ghi $t1 thì thanh ghi $t2 có giá trị là 1, ngược lại thì thanh ghi $t2 có giá trị là 0.
$t0 = 1111 1111 1111 1111 1111 1111 1111 1111 (Hệ 2 - 32 bits)
$t1 = 0011 1111 1111 1000 0000 0000 0000 0000 (Hệ 2 - 32 bits)
Ta thấy thanh ghi $t0 có giá trị âm và thanh ghi $t1 có giá trị dương nên $t0 < $t1 nên $t2 = 1
beq $t2, $zero, ELSE
Ta thấy câu lệnh bằng không xảy ra nên câu lệnh “J DONE” được thực hiện
Kết luận: $t2 = 1

Câu 7.2:
a, $t0 = 1010 1101 0001 0000 0000 0000 0000 0010 (Hệ 2 - 32 bits)
$t0 = -1391460350 (Hệ 10)
slti $t2, $t0, X
Nếu $t0 < X thì $t2 có giá trị là 1, ngược lại $t2 có giá trị là 0. Mà theo đề bài muốn $t2 = 1 nên $t0 < X. Vì X chỉ được biểu diễn tối đa trong 16 bits, dạng bù 2 nên giá trị của X không thể quá 2^15 - 1 = 32767.
Kết luận: -1391460350 < X <= 32767

b, $t0 = 1111 1111 1111 1111 1111 1111 1111 1111 (Hệ 2 - 32 bits)
$t0 = -1 (Hệ 10)

slti $t2, $t0, X

Nếu $t0 < X thì $t2 có giá trị là 1, ngược lại $t2 có giá trị là 0. Mà theo đề bài muốn $t2 = 1 nên $t0 < X. Vì X chỉ được biểu diễn tối đa trong 16 bits, dạng bù 2 nên giá trị của X không thể quá 2^15 - 1 = 32767.

Kết luận: -1 < X <= 32767

Câu 7.3:

* Với lệnh " j ”

a, Không

b, Không

Giải thích:

Lệnh Jump thuộc kiểu lệnh J-type nên trường địa chỉ (address) có 26 bits.

Địa chỉ mà lệnh j sẽ nhảy tới tức là địa chỉ sẽ gán cho con trỏ PC được tính bằng cách:

JumpAddr = { PC + 4[31:28], address, 2’b0}

Tức vùng address trong mã máy của lệnh j được lấy ra, dịch trái 2 bits, sau đó gán thêm 4 bits cao nhất được lấy từ 4 bits cao nhất (từ 28 đến 31) của PC hiện tại + 4.

PC hiện tại bằng 0x0000 0020 —> PC + 4 = 0x0000 0024

4 bits từ 28 đến 31 của PC + 4 = 0000

Vậy 4 bits cao nhất của PC mới với bất kì lệnh j nào cũng phải có 0000. Những giá trị cho trong bảng trên có giá trị lớn hơn 0000. Vì thế không thể sử dụng lệnh j để nhảy tới các địa chỉ như trong bảng.

* Với lệnh “ beq “

a, Không

b, Không

Giải thích:

Lệnh beq thuộc kiểu I-type

Ví dụ lệnh: beq $t0, $t1, 4

Thì 4 được lưu vào trường immediate

Khi beq thực hiện lệnh nhảy, giá trị mới gán cho PC = PC (hiện tại) + 4 + (immediate << 2)

Immediate là số 16 bits, sau khi dịch trái 2 bits thành số 18 bits, giá trị lớn nhất của số 18 bits này là 2^18 - 1.

Vì vậy, PC lớn nhất chỉ có thể nhận giá trị = PC (hiện tại) + 4 + 2^18 - 1 = 0x0000 0020 + 4 + 2^18 - 1.

Giá trị lớn nhất này nhỏ hơn các giá trị được cho trong bảng trên

Vậy ta không thể gán giá trị PC tới các giá trị trong bảng bằng cách sử dụng lệnh beq.