

Assignment 2 - John Akujobi v2

Owner	John Akujobi
Type	Homework
Created time	@February 3, 2024 5:44 PM
Status	In progress

Score 49/50

CSC 317 Spring 2024 | Assignment #2 | Due: 2-12-24

From the textbook (ARM edition) do the following problems that start on page 175.

Assignments are to be submitted to D2L by 2:00pm on the due date.

Exercise 2.1 (5 points)

For the following C statement, write the corresponding LEGBv8 assembly code.

Assume that the C variables f, g, and h, have already been placed in registers X0, X1, and X2 respectively. Use a minimal number of LEGBv8 assembly instructions.

```
f = g + (h - 5);
```

```
; f = X0
```

```
; g = X1
```

```
; h = X2
```

```
SUBI X3, X2, #5 ; Subtract 5 from h(X2) and put result in temp (X3)
```

```
ADD X0, X1, X3; Add temp (X3) to g(X1) and put result in f(X0)
```

OK

2. Exercise 2.2 (5 points)

Write a single C statement that corresponds to the two LEGBv8 assembly instructions below.

```
ADD f, g, h
```

```
ADD f, i, f
```

```
// f = g + h
// f = f + i
// f = g + h + i

f = g + h + i;
```

OK

3. Exercise 2.3 (5 points)

For the following C statement, write the corresponding LEGv8 assembly code.

Assume that the variables f, g, h, i, and j are assigned to registers X0, X1, X2, X3, & X4, respectively.

Assume that the base address of the arrays A and B are in registers X6 and X7, respectively.

```
B [8] = A [i - j];
```

```
// f = X0 - in use
// g = X1 - in use
// h = X2 - in use
// i = X3
// j = X4
// A [0] = X6
// B [0] = X7
// B [8] = 8 * 8 = 64 Offset
```

```
SUB X5, X3, X4 // Calculate the index i - j and store it in X5
```

```
LSL X5, X5, #2 // Multiply index by 4 to convert to byte offset
```

```
LSL X5,X5,#3 // we use double words (-1)
```

```
// Add base address of A to the offset to get address of A[i - j]
```

```
ADD X5, X6, X5
```

```
LDUR X8, [X5, #0] // Load the value from A[i - j] into register
```

```
// Calculate the address for B[8]
```

```
// by adding the 32-byte offset to the base address of B
```

```
ADD X5, X7, #32
```

```
STUR X8, [X5, #0] // Store the value from A[i - j] into B[8]
```

4. Exercise 2.11 (all parts) (12 points)

Assume that registers X0 and X1 hold the values 0x8000000000000000 and 0xD000000000000000, respectively.

2.11.1

What is the value of X9 for the following assembly code?

```
ADD X9, X0, X1
```

X9 = 0x5000000000000000 OK

Decimal

8+13=21

21 = 15 hex

```
0x8000000000000000
+ 0xD000000000000000
-----
0x1500000000000000
```

ARM registers can't store more than 64 bits, so, the 1 in from v
leaving behind 0x5000000000000000

2.11.2

Is the result in X9 the desired result, or has there been overflow?

There was overflow OK

0x150000000000000000 was the desired result
But because it exceeded the space for ARM registers, we had to
leaving 0x150000000000000000

2.11.3

For the contents of registers X0 and X1 as specified above, what is the value of X9 for the following assembly code?

```
SUB X9, X0, X1
```

X9 = 0x0B OK

8 - 13 (we have to borrow 1h or 16)
24 - 13 = 11 = B
0x18 = 0xD = 0xB

```
0x8000000000000000 = X0
- 0xD000000000000000 = X1
-----
0x0B0000000000000000
-----
```

2.11.4

Is the result in X9 the desired result, or has there been overflow?

There wasn't exactly an overflow.
But there was a wrap around

OK

2.11.5

For the contents of registers X0 and X1 as specified above, what is the value of X9 for the following assembly code?

```
ADD X9, X0, X1
```

```
ADD X9, X9, X0
```

| X9 = 0xD

```
ADD X9, X0, X1
```

```
0x8000000000000000 = X0
```

```
+ 0xD000000000000000 = X1
```

```
-----
```

```
0x1500000000000000
```

```
X9 = 0x5000000000000000
```

```
ADD X9, X9, X0
```

```
0x5000000000000000
```

```
+ 0x8000000000000000
```

```
-----
```

```
0xD000000000000000
```

OK

```
8h + 8h + 13h =
```

```
8 + 8 + 13 = 29 = 1D hex
```

2.11.6

Is the result in X9 the desired result, or has there been overflow?

The desired result is 0x1D

but X9 does not contain it because of overflow

OK

5. Exercise 2.14 (5 points)

Provide the instruction type and hexadecimal representation of the following instruction:

```
STUR X9, [X10, #32]
```

| 0xF8020149

STUR opcode is 1984ten

[X10, #32] means there is a 32 bit offset from X10

So the instruction wants to store the data from there into X9

x10, [x10, #16], [x10, #32]

That is, storing the third data from x10

Using D-format (opcode, rm address, Rn, Rt)

opcode: 1984ten -> 0x7C0h -> 11111000000 in binary (11 bits)

op2: 00 in binary (2 bits)

Rn (X10): 01010 in binary (5 bits)

Rt (X9): 01001 in binary (5 bits)

Rm address (#32): This needs to fit into 9 bits.

32 in decimal is 100000 in binary,

but we need to pad it to 9 bits: 0010000

= 11111000000 000100000 00 01010 01001

= 1111 1000 0000 0010 0000 0001 0100 1001

Splitting it into groups of 4

= 1111 1000 0000 0010 0000 0001 0100 1001

= 0xF8020149

```
binary_string = '1111100000000001000000000101001001'
hex_string = hex(int(binary_string, 2))
print(hex_string)
```

6. Exercise 2.30 (18 points)

Implement the following C code in LEGv8 assembly. Hint:

Remember that the stack pointer must remain aligned on a multiple of 16.

```
int fib (int n) {
    if (n==0)
        return 0;
    else if (n == 1)
        return 1;
    else
        return fib(n-1) + fib(n-2);
}
```

```
fib:
    //Lets assume that n is in x0
    //since that is where arguments go to

    sub sp, sp, #16 // Allocate stack space
    // Save frame pointer and return address
    stp x29, x30, [sp, #16]
    add x29, sp, #16 // Update frame pointer

    //if (n == 0) {return 0;}
    cmp x0, #0 //check if n is 0
    beq end_if_zero //jump to return 0 if n is 0

    //else if (n == 1) { return 1;}
    cmp x0, #1
    beq end_if_one //jump to return 1 if n is 1

    //else {return fib(n-1) + fib(n-2);}
    sub x1, x0, #1
    sub x2, x0, #2
    blt x1, x0, fib
    blt x2, x0, fib
    add x3, x0, #-1
    add x4, x0, #-2
    mov x5, x3
    mov x6, x4
    add x7, x5, x6
    ret
```

```

    cmp x0, #1      //check if n is 1
    beq end_if_one  //jump to return 1 if n is 1

    //fib(n - 1)
    mov x1, x0      //copy n to x1 for keeping
    sub x0, x0, #1  //n = n - 1
    bl fib          // call fib(n - 1)
    mov x19, x0     //store fib(n - 1) in x19

    //fib(n - 2)
    mov x0, x1      //copy n to x0
    sub x0, x0, #2  //subtract 2 from n
    bl fib          //call fib

    add x0, x19, x1 //add fib(n - 1) + fib(n - 2)

    b end           //jump to 'end'

end_if_zero:
    mov x0, #0      // If n is 0, return 0
    b end           // Branch to label 'end'

end_if_one:
    mov x0, #1      // If n is 1, return 1

end:
    // Restore frame pointer and return address
    ldp x29, x30, [sp, #16]
    add sp, sp, #16 // Deallocate stack space
    ret             // Return to caller function


```

I used the compiler below to translate and understand part of it

OK

Compiler Explorer

Compiler Explorer is an interactive online compiler which shows the assembly output of compiled C++, Rust, Go (and many more) code.

 <https://godbolt.org/>

