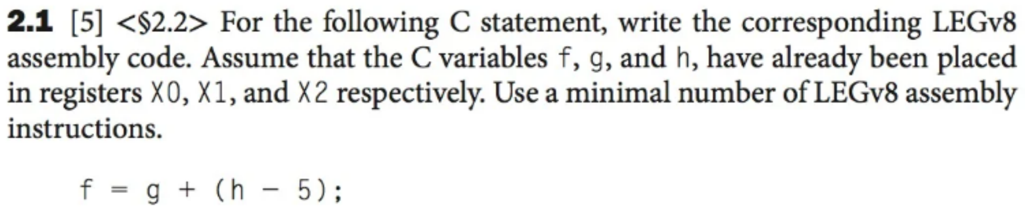
Written Assignment 2

Name: Aina Azman

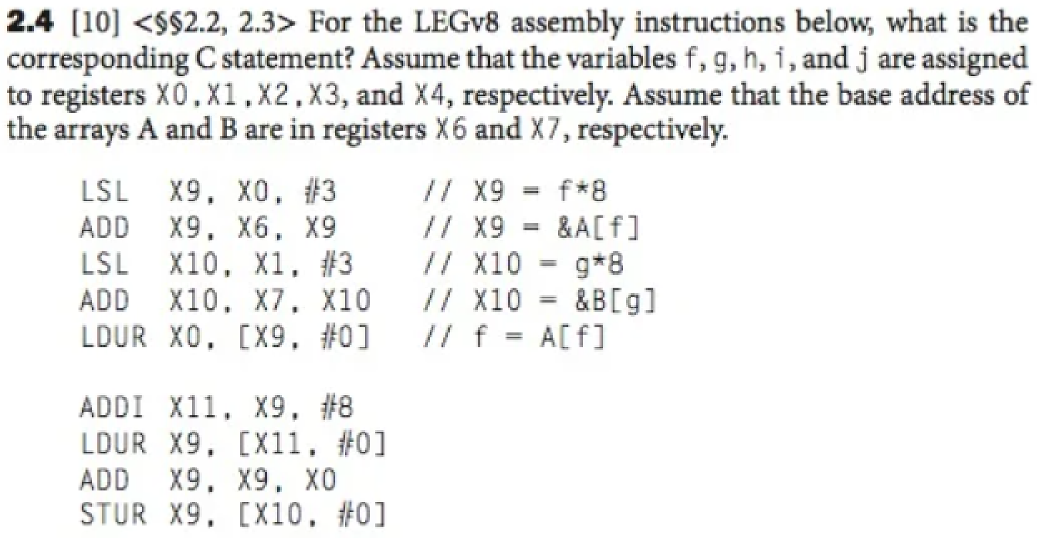
Net_ID: qistina



2.1 [5] <§2.2> For the following C statement, write the corresponding LEGv8 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 LEGv8 assembly instructions.

```
f = g + (h − 5);
```

```
SUBI X2, X2, #5  // h = h - 5
ADD X0, X1, X2   // f = g + h
```



2.4 [10] <§§2.2, 2.3> For the LEGv8 assembly instructions below, what is the corresponding C statement? Assume that the variables f, g, h, i, and j are assigned to registers X0, X1, X2, X3, and X4, respectively. Assume that the base address of the arrays A and B are in registers X6 and X7, respectively.

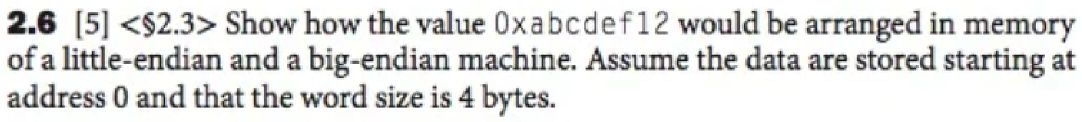
```
LSL  X9, X0, #3     // X9 = f*8
ADD  X9, X6, X9     // X9 = &A[f]
LSL  X10, X1, #3    // X10 = g*8
ADD  X10, X7, X10   // X10 = &B[g]
LDUR X0, [X9, #0]   // f = A[f]

ADDI X11, X9, #8
LDUR X9, [X11, #0]
ADD  X9, X9, X0
STUR X9, [X10, #0]
```

LEGv8 assembly instructions:

```
X9 = f * 8
X9 = &A[f]

X10 = g * 8
X10 = &B[g]

f = A[f]

X11 = &A[f+1]
X9 = A[f+1]
X9 = A[f+1] + A[f]

B[g] = A[f+1] + A[f]
```

C code corresponding to LEGv8 assembly instructions:

```
f = A[f]
B[g] = A[f] + A[f+1]
```



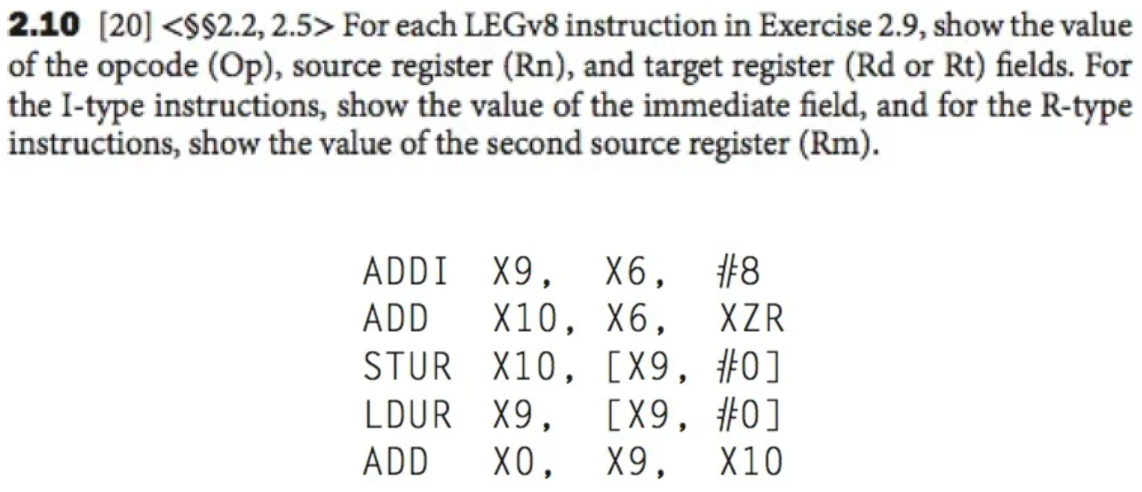
2.6 [5] <§2.3> Show how the value `0xabcdef12` would be arranged in memory of a little-endian and a big-endian machine. Assume the data are stored starting at address 0 and that the word size is 4 bytes.

Little-Endian

Address	0x00	0x01	0x02	0x03
Values	0x12	0xef	0xcd	0xab

Big-Endian

Address	0x00	0x01	0x02	0x03
Value	0xab	0xcd	0xef	0x12



2.10 [20] <§§2.2, 2.5> For each LEGv8 instruction in Exercise 2.9, show the value of the opcode (Op), source register (Rn), and target register (Rd or Rt) fields. For the I-type instructions, show the value of the immediate field, and for the R-type instructions, show the value of the second source register (Rm).

```
ADDI  X9,   X6,   #8
ADD   X10,  X6,   XZR
STUR  X10,  [X9,  #0]
LDUR  X9,   [X9,  #0]
ADD   X0,   X9,   X10
```

- ADDI X9, X6, #8

opcode (Op), 10 bits	immediate field, 12 bits	source register (Rn), 5 bits	target register (Rd/Rt), 5 bits
1001000100	0000 0000 1000	00110	01001

- ADD X10, X6, XZR

opcode (Op), 11 bits	second source register (Rm), 5 bits	shamt, 6 bits	source register (Rn), 5 bits	target register (Rd/Rt), 5 bits
00001110000	00000	000000	00110	01010

- STUR X10, [X9, #0]

opcode (Op), 11 bits	address (Rm), 9 bits	op2, 2 bits	source register (Rn), 5 bits	target register (Rt), 5 bits
11111000000	000000000	00	01001	01010

- LDUR X9, [X9, #0]

opcode (Op), 11 bits	address (Rm), 9 bits	op2, 2 bits	source register (Rn), 5 bits	target register (Rt), 5 bits
11111000010	000000000	00	01001	01001

- ADD X0, X9, X10

opcode (Op), 11 bits	second source register (Rm), 5	shamt, 6 bits	source register (Rn), 5 bits	target register (Rd/Rt), 5 bits

	bits			
10001011000	01010	000000	01001	00000

2.14 [5] <§§2.2, 2.5> Provide the instruction type and hexadecimal representation of the following instruction:

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

- Instruction Type:
  - STUR (Store Register) Instruction
  - Stores a value from a register into memory
- Binary Representation (D-format):

opcode (Op), 11 bits	address, 9 bits	op2, 2 bits	source register (Rn), 5 bits	target register (Rt), 5 bits
11111000000	000100000	00	01010	01001

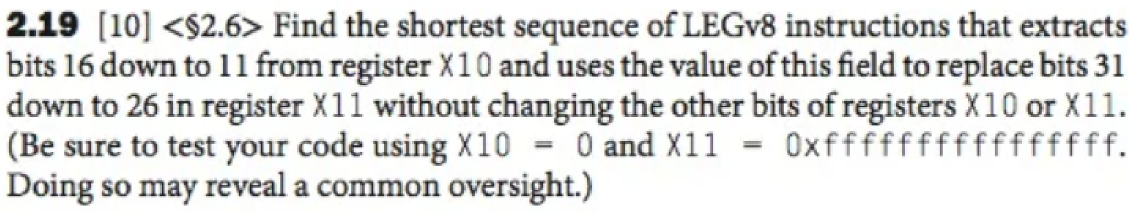
- Hexadecimal Representation: 0xf8020149

2.16 [5] <§2.5> Provide the instruction type, assembly language instruction, and binary representation of instruction described by the following LEGv8 fields:

```
op=0x7c2, Rn=12, Rt=3, const=0x4
```

- Instruction type: Load register with value
- Assembly language instruction: LDUR X3, [X12, #4]
- Binary representation (D-format):1111 1000 0100 0000 0100 0001 1000 0011

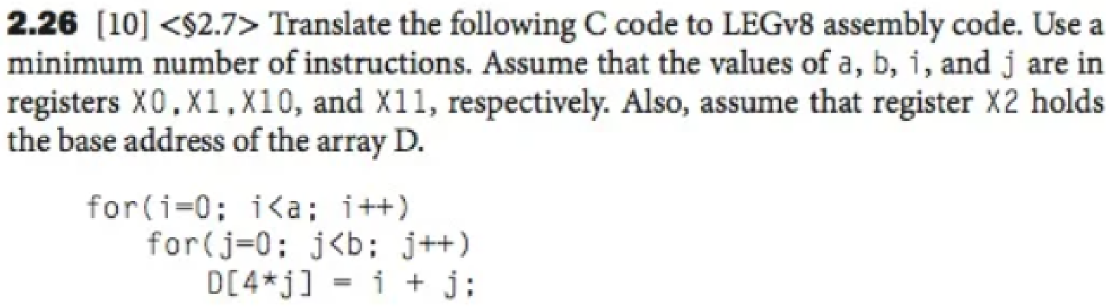
opcode (Op), 11 bits	address, 9 bits	op2, 2 bits	source register (Rn), 5 bits	target register (Rt), 5 bits
111 1100 0010	000000100	00	01100	00011


2.19 [10] <§2.6> Find the shortest sequence of LEGv8 instructions that extracts bits 16 down to 11 from register X10 and uses the value of this field to replace bits 31 down to 26 in register X11 without changing the other bits of registers X10 or X11. (Be sure to test your code using X10 = 0 and X11 = 0xffffffffffffffff. Doing so may reveal a common oversight.)

Instructions:

- Extracts bits 16 down to 11 from register X10
- Replace bits 31 down to 26 in register X11 with the extracted value
- The other bits in both register should remain unchanged.

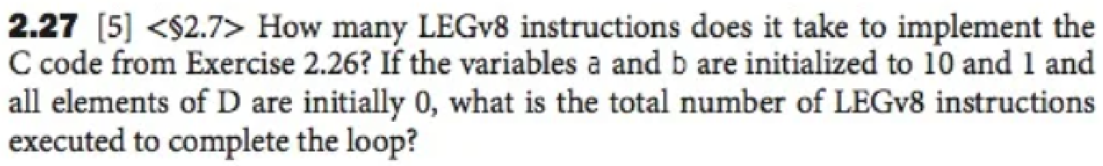
```
ANDI X11, X11, 0xFFFFFFFF03FFFFFF // Clear bits 31 down to 26
LSL X10, X10, #15
ORR X11, X11, X10
```


2.26 [10] <§2.7> Translate the following C code to LEGv8 assembly code. Use a minimum number of instructions. Assume that the values of a, b, i, and j are in registers X0, X1, X10, and X11, respectively. Also, assume that register X2 holds the base address of the array D.

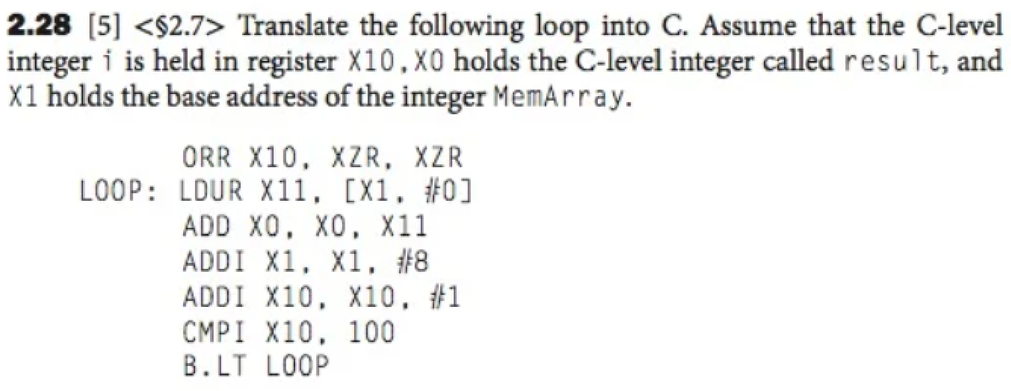
```
for(i=0; i<a; i++)
   for(j=0; j<b; j++)
      D[4*j] = i + j;
```

```
           ADD X10, XZR, XZR
loop1: SUBS X9, X10, X0
             B.GE exit
             ADD X11, XZR, XZR
loop2: SUBS X9, X11, X1
```

```
             B.GE Next
             ADD X9, X10, X11
             LSL X12, X11, #5
             ADD X12, X12, X2
             STUR X9, [X12, #0]
             ADDI X11, X11, #1
             B loop2
Next: ADDI X10, X10, #1
          B loop1

Exit:
```


2.27 [5] <§2.7> How many LEGv8 instructions does it take to implement the C code from Exercise 2.26? If the variables a and b are initialized to 10 and 1 and all elements of D are initially 0, what is the total number of LEGv8 instructions executed to complete the loop?

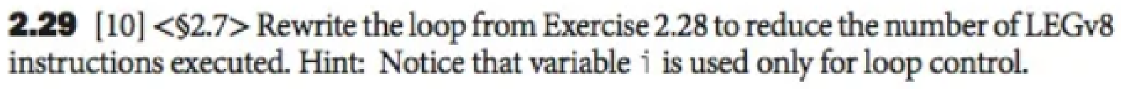
- Since a = 10, the outer loop executes 10 times
- Since b = 1, the inner loop executes only one time each time.
    - Line 1 executed = 1 time
    - Line 2 and 3 executes 11 times
    - Line 13 and 14 executes 10 times
    - In each iteration,
        - line 4 executes once = 1
        - line 5 and 6 executes twice = 2
        - line 7-12 executes only once = 2
    - Inner loop = 11
- Therefore, total number of LEGV8 instructions executed: 1 + (10 ∗ 11) + 2 + (2∗10) = 133



2.28 [5] <§2.7> Translate the following loop into C. Assume that the C-level integer `i` is held in register `X10`, `X0` holds the C-level integer called `result`, and `X1` holds the base address of the integer `MemArray`.

```
      ORR X10, XZR, XZR
LOOP: LDUR X11, [X1, #0]
      ADD X0, X0, X11
      ADDI X1, X1, #8
      ADDI X10, X10, #1
      CMPI X10, 100
      B.LT LOOP
```

```
int result, MemArray;
int i = 0;

do{

result += MemArray[i];
i++;
}
while(i < 100)
```

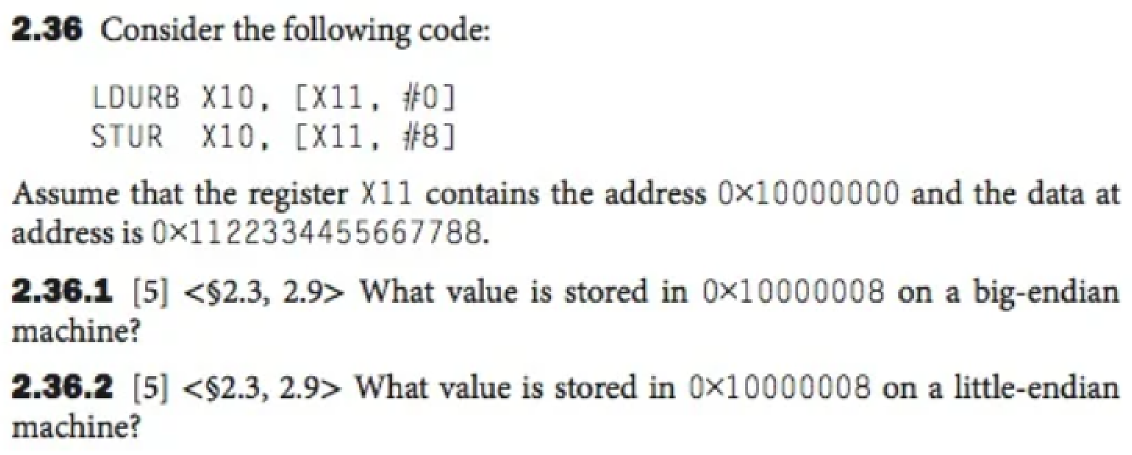


2.29 [10] <§2.7> Rewrite the loop from Exercise 2.28 to reduce the number of LEGv8 instructions executed. Hint: Notice that variable `i` is used only for loop control.

```
ADDI X10, X1, 400

loop: LDUR X11, [X1, #0]
            ADD X0, X0, X11
            ADDI X1, X1, #8
            CMPI X1, X10
            B.LE X1, X10, loop
```

```
int result, MemArray;
int i = 0;
```

```
for(i = 0; i < 100; i++){
result += MemArray[i];
}
```



2.36 Consider the following code:

```
    LDURB X10, [X11, #0]
    STUR  X10, [X11, #8]
```

Assume that the register X11 contains the address 0×10000000 and the data at address is 0×1122334455667788.

2.36.1 [5] <§2.3, 2.9> What value is stored in 0×10000008 on a big-endian machine?

2.36.2 [5] <§2.3, 2.9> What value is stored in 0×10000008 on a little-endian machine?

2.36.1 What value is stored in 0×10000008 on a big-endian machine?

- MSB of multi-byte value is stored at the lowest memory address.
- LDURB X10, [X11, #0] loads a byte from [X11, #0] which is 0x11.
- STUR X10, [X11, #8] stores the 64-bit value from X10 into memory starting at address 0×10000008.
- Hence, the value at 0×10000008 on a big-endian machine is 0x0000000000000011.

2.36.2 What value is store in 0×10000008 on a little-endian machine?

- LSB of multi-byte value is stored at the lowest memory address.
- LDURB X10, [X11, #0] loads a byte from [X11, #0] which is 0x88.
- STUR X10, [X11, #8] stores the 64-bit value from X10 into memory starting at address 0×10000008.

- Hence, the value at 0×10000008 on a little-endian machine is 0x8811223344556677.