Homework #1

Due Jan. 7

5 points each

1. LEGv8 coding

(a) 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 **X3, X4, and X5** respectively. Use a minimal number of LEGv8 assembly instructions.

f = g - (h + 5);

```
// add h + 5
ADDI X5, X5, #5 // X5 = X5 + 5
// subtract g from (h+5) to f
SUB X3, X5, X4 // X3 = X5 - X4
```

(b) For the following C statement, write a minimal sequence of LEGv8 assembly instructions that performs the identical operation. Assume d is in X2, and X3 is the base address of the array A.

d = A[5] >> 3;

```
// load offset 8*5=40
LDUR X9, [X3, #40] // X9 = A[5]
LSR X2, X9, #3 // d = A[5] >> 3
```

2. LEGv8 coding

(a) 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, and X4, respectively. Assume that the base address of the arrays A and B are in registers X6 and X7, respectively.

A[i–j] = B[4];

```
// load B[4]
// use x7 so that it will load w/e is in base address of B + offset of 32
LDUR X7, [X7, #32] // X7 = B[4]
// [i-j]]
SUB X9, X3, X4 // X9 = i - j
// offset of 8*(i-j)
LSL X9, X9, #3 // X9 = 8*(i-j)
// set address of A[i-j]
ADD X9, X9, X6 // X9 = base address of A + offset of 8*(i-j)
// store B[4]
STUR X7, [X9, #0] // A[i-j] = B[4]
```

(b) 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, and X4, respectively. Assume that the base address of the arrays A and B are in registers X6 and X7, respectively.

A[i] = A[j] + B[3];

```
// load B[3]
// use x7 so that it will load w/e is in base address of B + offset of 24
LDUR X7, [X7, #24] // X7 = B[3]
// offset of 8*j
```

```
    LSL X9, X4, #3 // X9 = 8*j
    // add base address of A to offset of 8*j
    ADD X9, X9, X6 // X9 = base address of A + offset of 8*j
    // load with offset #0
    LDUR X10, [X9, #0] // X10 = A[j]
    // add B[3] to A[j]
    ADD X10, X10, X7 // X10 = A[j] + B[3]
    // offset address of A[i]
    LSL X9, X3, #3 // X9 = 8*i
    // add base address of A to offset of 8*i
    ADD X9, X9, X6 // X9 = base address of A + offset of 8*i
    // store A[i]
    STUR X10, [X9, #0] // A[i] = A[j] + B[3]
```

3. Memory

(a) How many locations of memory (in terms of the number of bytes) can you address with 17-bit memory address?

```
2^17 = 131072 locations
```

(b) How many bits are required to address a 64-Mega-location memory, i.e, what should the number of bits of the memory address be in order to access a 64MB memory?

```
64MB = 64*1024*1024 = 67108864 bytes
```

(c) Assume `A` is an integer array with 20 elements stored in memory and its starting memory address is in X3. What is the memory address for element `A`[5]?

```
A[5] = X3 + 8*5 = X3 + 40
```

(d) Assume `C` is a character array with 20 elements stored in memory and its starting memory address is in X4. What is the memory address for element `C`[12]?

```
C[12] = X4 + 12
```

(e) Write at least five different ways to assign 0 to register X3.

```
    ADDI X3, XZR, #0 // 0 + 0
    SUBI X3, XZR, #0 // 0 - 0
    MOV X3, XZR      // X3 = 0
    EORI X3, XZR, #0 // 0 xor 0 = 0
    ANDI X3, XZR, #1 // 0 and 1 = 0
```

4. Instruction execution

(a) Follow the instructions below CONSECUTIVELY, and after each instruction write the value of destination register in decimal into the space provided.

```
    ORRI X0, X31, #5  // X0 = 5
    SUBI X1, X0, #2   // X1 = 3
    LSL X2, X1, #2    // X2 = 6
    ADD X3, X2, X2    // X3 = 12
    ANDI X4, X3, #13  // X4 = 12
```

(b) For the following LEG8 assembly, assume that the registers X1 and X6, contain the values 20 and 256, respectively. Also, assume that memory contains the following values. What is the value of X0?

Address	Contents
256	50
260	100
264	150

```
SUBI X6, X6, #20  // X6 = 256 - 20 = 236
ADD X6, X6, X1    // X6 = 236 + 20 = 256
LDUR X0, [X6, #8] // X0 = memory[256+8] = memory[264] = 150
```

5. Memory

(a) Show using the tables below how the value 0xF3AB440C would be arranged in memory of a little-endian and a big-endian machine. Assume the data are stored starting at address 4 and that **the word size is 4 bytes**.

Big endian

address	data
0	
4	F3AB440C
8	
12	
16	

little endian

Address	Data
0	
4	0C44ABF3
8	
12	
16	

(b) Based on the given memory map, what is the result of the LEGv8 instruction "LDUR X1, [X3, #4]" if the machine is in **little-endian** byte orders, where X3 = 0x00000010. (i.e. what is the value of X1).

0x119988CC

(c) Show on the memory map the result of the following instruction

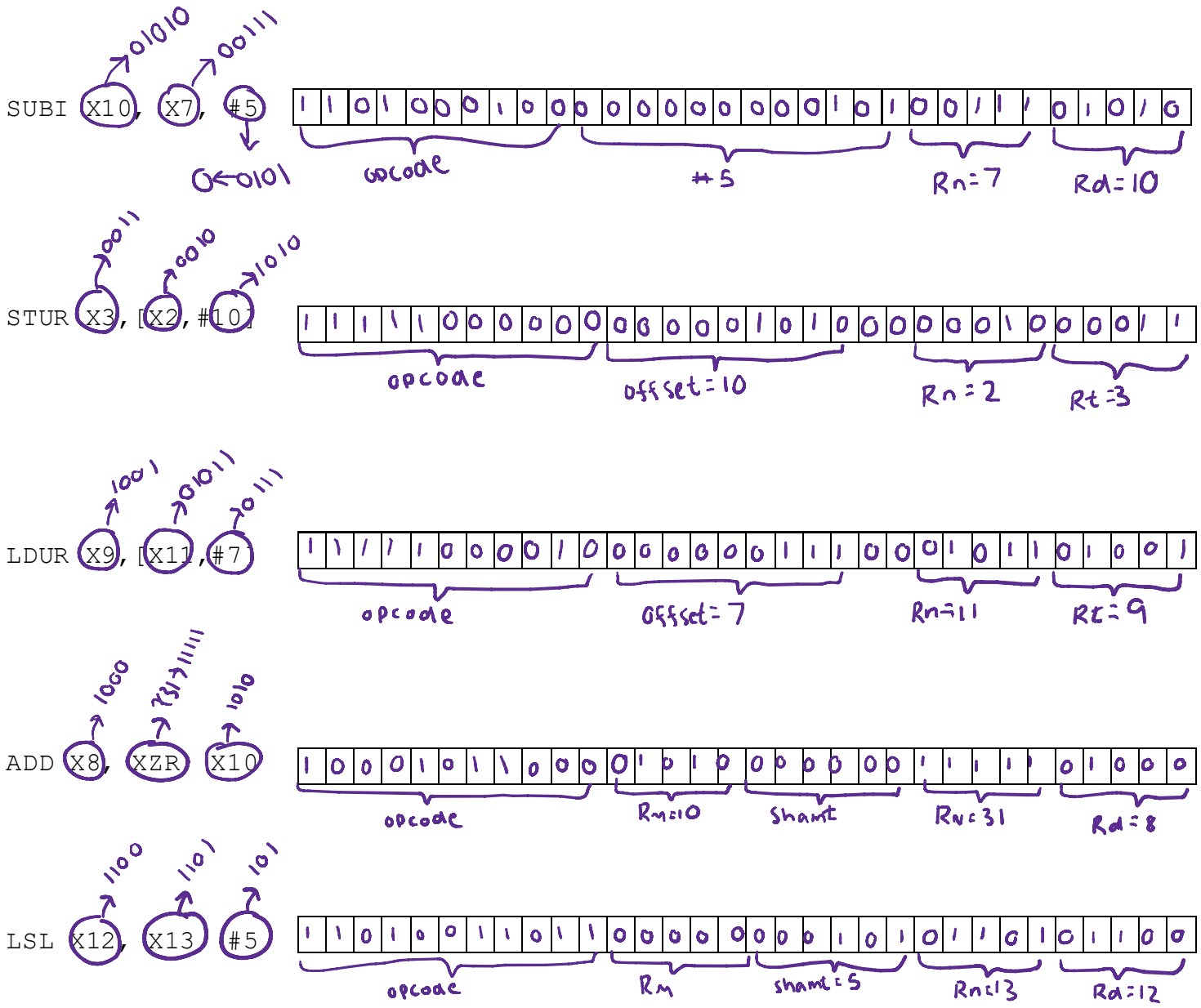
STUR X2, [X0, #-8]

[X0-8] ← X2

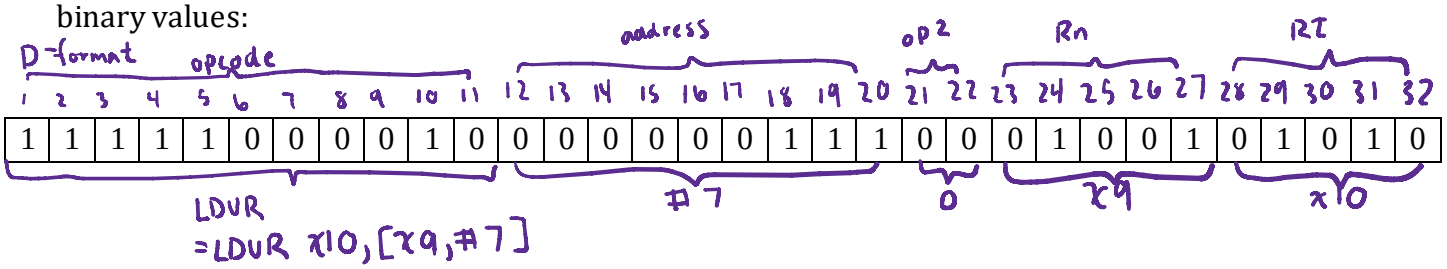
where X0 = 0x0000001C, X2 has the value of 0xFFEE00A80C003352 (i.e. show the content changes of the corresponding memory cells). Assume that this is **a big endian** machine.

Address	Contents
0x0000000F	0x00
0x00000010	0x00
0x00000011	0x00
0x00000012	0xAA
0x00000013	0xCC
0x00000014	0x88 FF
0x00000015	0x99 EE
0x00000016	0x11 00
0x00000017	0x23 A8
0x00000018	0x45 0C
0x00000019	0x00 00
0x0000001A	0xFF 33
0x0000001B	0x00 52
0x0000001C	0xAA
0x0000001D	0xBB
0x0000001E	0x1C
0x0000001F	0x2F
0x00000020	0x55
0x00000021	0x44
0x00000022	0xFB

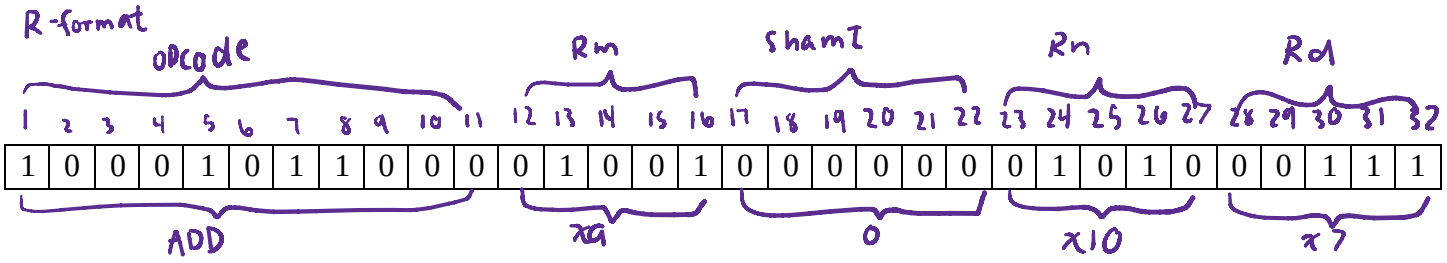
6. Instruction encoding: represent each of the following instructions in binary.



7. Instruction decoding: provide the instruction type and assembly language instruction for the following binary values:



=LDUR x10, [x9, #7]



=ADD x7, x10, x9

D-format

opcode											address									op2		Rn					Rt				
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	1	1	0	1	0	1	1	0
STUR											#30									0		x6					x22				

= STUR x22, [x6, #30]

I-format

opcode										immediate												Rn					Rd				
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	1	0	1
ADDI										#15												x1					x5				

= ADDI x5, x1, #15

8. Assume that we would like to expand the LEGv8 register file to 128 registers and expand the instruction set to contain four times as many instructions.

(a) How would this affect the size of each of the bit fields in the R-type instructions?

$2^7 = 128 \rightarrow$ 7 bits to represent reg Rm, Rn, Rd

8 + 3·7

8 + 21

29 bits

opcode	Rm	shamt	Rn	Rd
8 bits	7 bits	3 bits	7 bits	7 bits

(b) How would this affect the size of each of the bit fields in the I-type instructions?

opcode	immediate	Rn	Rd
8 bits	8 bits	7 bits	7 bits

32
− 14
18

9. For the following program, represent the CBZ and B instructions in binary:

```
      ADDI X10, X1, #792
LOOP:  CBZ X10, END
PC →   LDUR X11, [X10, #0]   8
       ADD X0, X0, X11       8
       SUBI X10, X10, #8     8
       SUB X10, X10, X1      8
END:   B LOOP
```

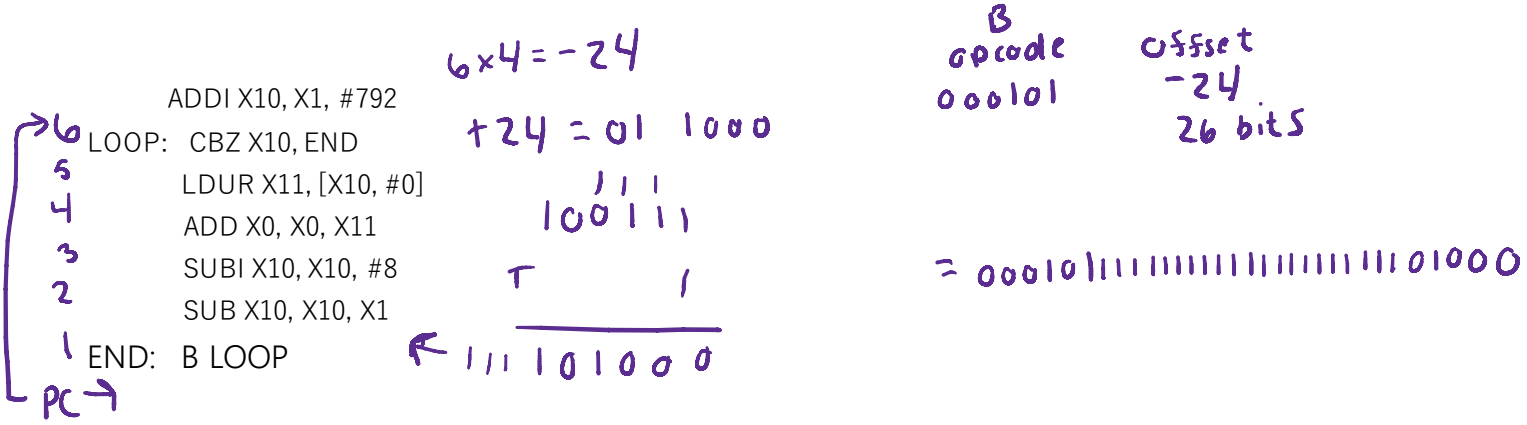
8·4 = 32

x32

opcode CBZ	offset	0x10
10110100	32	01010

10110100 0.... 100000 01010

32 | 10



9. Write a sequence of LEGv8 instructions to do the following:
set X8 to 100 if the contents of X1 is an odd number and set X9 to 200 otherwise.
Do not use division instruction in you code. Instead, you use an AND instruction to test if a number is even or odd. Comment your assembly code.

```
// mask all but the least significant bit of X1
AND X10, X1, #1
// test if X10 is 0, if so, jump to EVEN, else jump to next instruction
CBZ X10, EVEN
// if not 0, must be odd
LDI X9, #200 // X9 = 200
B END // jump to end
// if LSB is 0, must be zero
EVEN: LDI X8, #100 // X8 = 100
END: // end of program
```

10. Write the corresponding LEGv8 code for the following fragment of C code

```
for ( int i = 0; i < 50; i++ )
    C[i] = C[i-1] – C[i+1]*9;
```

Assume that the index *i* is in register X0, C is an integer array and the base address of C is in X2. Comment your assembly code. How many instructions are executed? How many data memory references have been made?

```
// start register at 0
ADD X0, XZR, XZR // 0 + 0 = 0
// begin loop
LOOP: SUBI X15, X0, #50 // X15 = 50 - 0 = 50
CBZ X15, END // if X15 = 0, jump to END
// offset
// i is going to have an offset of 0
LSL X9, X0, #3 // X9 = i*8
// add base address of C to offset
ADD X2, X2, X9 // X2 = base address of C + offset of i*8
// i-1 is going to have an offset of -8
LDUR X10, [X2, #-8] // X10 = C[i-1]
```

```
// i+1 is going to have an offset of +8
LDUR X11, [X2, #8] // X11 = C[i+1]
// multiply C[i+1] by 9
MUL X12, X11, #9 // X12 = C[i+1] * 9
// subtract C[i+1]*9 from C[i-1]
SUB X13, X10, X12 // X13 = C[i-1] - C[i+1]*9
// store C[i]
STUR X13, [X2, #0] // C[i] = C[i-1] - C[i+1]*9
// increment i
ADD X0, X0, #1 // i = i + 1
// jump to LOOP
B LOOP
END: // end of program

@ HOW MANY INSTRUCTIONS ARE EXECUTED ? - 11 * 50 = 550 + 1 = 551 + 2 = 553
@ HOW MANY DATA MEMORY REFERENCES HAVE BEEN MADE ? - 3 * 50 = 150
```