Review Problem 3

- In assembly, compute the average of positive values X0, X1, X2, X3, and put into X10

```
ADD X4, X0, X1
ADD X5, X2, X3
ADD X5, X4, X5
LSR ~~X5~~ X10, X5, #2
```

Addressing Example

The address of the start of a character array is stored in X0. Write assembly to load the following characters

X2 = Array[0]

LDURB X2, [X0, #0]

X3 = Array[1]

LDURB ~~X2~~ X3, [X0, ~~#8~~ #1]

X4 = Array[2]

LDURB X4, [X0, #2]

X5 = Array[k] // Assume the value of k is in X1 MEM[X0 + X1 k]

ADD X5, X0, X1 // X5 = &(Array[k])

LDURB X5, [X5, #0]

v[0] = MEM[X0] = mem[928]   v[1] = MEM[X0+8] = MEM[93

v[k] = MEM[X0+8*k]   v[k+1] = MEM[X0+8*k+8]

Array Example

```
/* Swap the kth and (k+1)th element of an array */
swap(int v[], int k) {
    int temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}

// Assume v in X0, k in X1
```

GPRs

X0:	928
X1:	10
X2:	
X3:	
X4:	

Load

Store

Memory

1000	0A12170D34BC2DE1
1008	1111111111111111
1016	0000000000000000
1024	0F0F0F0F0F0F0F0F
1032	FFFFFFFFFFFFFFFF
1040	FFFFFFFFFFFFFFFF

```
SWAP:
    LSL   X2, X1, #3       // X2 = 8*K
    ADD   X2, X0, X2       // X2 = &V[K]
    LDUR  X3, [X2, #0]     // get V[K]
    LDUR  X4, [X2, #8]     // get V[K+1]
    STUR  X4, [X2, #0]
    STUR  X3, [X2, #8]
```

Execution Cycle Example

PC: Program Counter

IR: Instruction Register

Note:
Word addresses
Instructions are 32b

General Purpose Registers

X0:	928
X1:	10
X2:	~~80~~ 1008
X3:	1111111111111111
X4:	

PC: ~~0~~ ~~4~~ 8

IR: ~~D3600C22~~

~~8B020002~~

F8400043

Load

Store

Memory

Address	Contents
0000	D3600C22
0004	8B020002
0008	F8400043
0012	F8408044
0016	F8000044
0020	F8008043

Address	Contents
1000	0A12170D34BC2DE1
1008	1111111111111111
1016	0000000000000000
1024	0F0F0F0F0F0F0F0F
1032	FFFFFFFFFFFFFFFF
1040	FFFFFFFFFFFFFFFF

Instruction Fetch → Instruction Decode → Operand Fetch → Execute → Result Store → Next Instruction math Results, 4 → flags en

Single?

Flags/Condition Codes

Flag register holds information about result of recent math operation

- Negative: was result a negative number?
- Zero: was result 0?
- Overflow: was result magnitude too big to fit into 64-bit register?
- Carry: was the carry-out true?

Operations that set the flag register contents:

ADDS, ADDIS, ANDS, ANDIS, SUBS, SUBIS, some floating point.

Most commonly used are subtracts, so we have a synonym: CMP

CMP X0, X1   same as SUBS X31, X0, X1

CMPI X0, #15 same as SUBIS X31, X0, #15

Control Flow

Unconditional Branch – GOTO different next instruction

```
B START          // go to instruction labeled with "START" label
BR X30           // go to address in X30: PC = value of X30
```

Conditional Branches – GOTO different next instruction if condition is true

1 register: CBZ (==0), CBNZ (!= 0)

```
CBZ X0, FOO      // if X0 == 0 GOTO FOO: PC = Address of instr w/FOO label
```

2 register: B.LT (<), B.LE(<=), B.GE (>=), B.GT(>), B.EQ(==), B.NE(!=)

first compare (CMP X0, X1, CMPI X0, #12), then b.cond instruction

```
CMP X0, X1       // compare X0 with X1 - same as SUBS X31, X0, X1
B.EQ FOO         // if X0 == X1 GOTO FOO: PC = Address of instr w/FOO label
```

a!=b

```
if (a == b)
   a = a + 3;
else
   b = b + 7;
c = a + b;
```

a==b

a!=b

```
// X0 = a, X1 = b, X2 = c
     CMP X0, X1           // set flags
     B.NE ELSEIF          // branch if a!=b
     ADDI X0, X0, #3      // a = a + 3
     B DONE               // avoid else
ELSEIF:
     ADDI X1, X1, #7      // b = b + 7
DONE:
     ADD X2, X0, X1       // c = a + b
```