Review Problem 2

- In assembly, set $X0$ to $-X1$.

```assembly
SUB $x0, $x31, $x1  // x0 = $0 - x1 = -x1
```
Basic Operations

(Note: just subset of all instructions)

Mathematic: ADD, SUB, MUL, SDIV

\[
\begin{align*}
\text{ADD } x_0, x_1, x_2 & \quad \text{// } x_0 = x_1 + x_2 \\
\text{ADDI } x_0, x_1, \#100 & \quad \text{// } x_0 = x_1 + 100
\end{align*}
\]

Immediate (one input a constant)

Logical: AND, ORR, EOR

\[
\begin{align*}
\text{AND } x_0, x_1, x_2 & \quad \text{// } x_0 = x_1 \& x_2 \\
\text{ANDI } x_0, x_1, \#7 & \quad \text{// } x_0 = x_1 \& \text{b0111}
\end{align*}
\]

Immediate

Shift: left & right logical (LSL, LSR)

\[
\begin{align*}
\text{LSL } x_0, x_1, \#4 & \quad \text{// } x_0 = x_1 << 4
\end{align*}
\]

Example: Take bits 6-4 of X0 and make them bits 2-0 of X1, zeros otherwise:

\[
\begin{align*}
\text{AND} & \quad x_1, x_0, \#\{6:7\} \\
\text{LSR} & \quad x_1, x_1, \#4 \\
\text{ANDI} & \quad x_1, x_1, \#7
\end{align*}
\]
Memory Organization

Viewed as a large, single-dimension array, with an address. A memory address is an index into the array. "Byte addressing" means that the index points to a byte of memory.
Memory Organization (cont.)

Bytes are nice, but most data items use larger units.

Double-word = 64 bits = 8 bytes
Word = 32 bits = 4 bytes

<table>
<thead>
<tr>
<th></th>
<th>64 bits of data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>64 bits of data</td>
</tr>
<tr>
<td>16</td>
<td>64 bits of data</td>
</tr>
<tr>
<td>24</td>
<td>64 bits of data</td>
</tr>
</tbody>
</table>

Registers hold 64 bits of data

$2^{64}$ bytes with byte addresses from 0 to $2^{64} - 1$

$2^{64}$ double-words with byte addresses 0, 8, 16, ... $2^{64} - 8$

Double-words and words are aligned

i.e., what are the least 3 significant bits of a double-word address?

<table>
<thead>
<tr>
<th>word</th>
<th>0</th>
<th>0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0100</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td></td>
</tr>
</tbody>
</table>

Div 0, set 3 bits are zero

Zero
Addressing Objects: Endian and Alignment

Doubleword: \[2^{63}..2^{56} \quad 2^{55}..2^{48} \quad 2^{47}..2^{40} \quad 2^{39}..2^{32} \quad 2^{31}..2^{24} \quad 2^{23}..2^{16} \quad 2^{15}..2^{8} \quad 2^{7}..2^{0}\]

Big Endian

| 000 | 2^{63}..2^{56}          |
| 001 | 2^{55}..2^{48}          |
| 010 | 2^{47}..2^{40}          |
| 011 | 2^{39}..2^{32}          |
| 100 | 2^{31}..2^{24}          |
| 101 | 2^{23}..2^{16}          |
| 110 | 2^{15}..2^{8}           |
| 111 | 2^{7}..2^{0}            |

Little Endian

0 DWord 0
8 DWord 1
16 DWord 2
24 DWord 3
32 DWord 4

Big Endian: address of most significant byte = doubleword address
Motorola 68k, MIPS, IBM 360/370, Xilinx Microblaze, Sparc

Little Endian: address of least significant byte = doubleword address
Intel x86, DEC Vax, Altera Nios II, Z80

ARM: can do either – this class assumes Little-Endian.
Data Storage

Characters: 8 bits (byte)
Integers: 64 bits (D-word)
Array: Sequence of locations
Pointer: Address (64 bits)

```c
// G = ASCII 71
char a = 'G';
int x = 258; 1x256+2
char *b;
int *y;
b = new char[4];
y = new int[10];
```

1001 = 3x256+233
1032 = 4x256+8

(Note: real compilers place local variables (the “stack”) from beginning of memory, new’ed structures (the “heap”) from end. We ignore that here for simplicity)
Loads & Stores

Loads & Stores move data between memory and registers
All operations on registers, but too small to hold all data

LDUR X0, [X1, #14]  // X0 = Memory[X1+14]

130 + 14 = 144

STUR X2, [X3, #20]  // Memory[X3+20] = X2

4 + 20 = 24

64-bit

General Purpose Registers

X0: 66
X1: 130
X2: 723
X3: 4

Memory

24: 723
144: 66

Note: LDURB & STURB load & store bytes
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]
  
  \texttt{LDURB X2, [X0, #0]}

X3 = Array[1]
  
  \texttt{LDURB X3, [X0, #1]}

X4 = Array[2]
  
  \texttt{LDURB X4, [X0, #2]}

X5 = Array[k] // Assume the value of k is in X1
  
  \texttt{ADD X18, X0, X1}

  \texttt{LDURB X5, [X18, #0]}