Homework 3

1. Consider the binary numbers 1010 1101 0001 0000 0000 0000 0000 0010 and 1111 1111 1111 1111 1011 0011 0101 0011.

- **A.** Convert them to decimal (assuming two's complement representation). *(-1,391,460,350; -19,629)*
- **B.** Convert them to decimal (assuming unsigned representation). *(2,903,506,946; 4,294,947,667)*
- **C.** Convert them to hexadecimal. *(0xAD100002; 0xFFFFB353)*

2. Consider the decimal numbers 2,147,483,647 and 1,000.

- **A.** Convert them to binary using two's complement representation. *(0111 1111 1111 1111 1111 1111 1111 1111; 0000 0000 0000 0000 0000 0011 1110 1000)*
- **B.** Convert them to hexadecimal using two's complement representation. *(0x7FFFFFFF; 0x000003E8)*
- **C.** Convert the negatives of the above numbers to hexadecimal using two's complement representation. *(0x80000001; 0xFFFFFC18)*

3. Let register `$s1` hold the values 2,147,483,647 and 0xD0000000 in turn. (Note that the first is decimal and the second is hexadecimal.)

- **A.** Will there be overflow if `$s0` holds the value 0x70000000, and the instruction `add $s0, $s0, $s1` is executed? *(overflow; no overflow)*
- **B.** Will there be overflow if `$s0` holds the value 0x80000000, and the instruction `sub $s0, $s0, $s1` is executed? *(overflow; no overflow)*
- **C.** Will there be overflow if `$s0` holds the value 0x7FFFFFFF, and the instruction `sub $s0, $s0, $s1` is executed? *(no overflow; overflow)*

4. Let register `$s0` hold the hexadecimal value 0x70000000 , and `$s1` hold the binary values 1010 1101 0001 0000 0000 0000 0000 0010 and 1111 1111 1111 1111 1011 0011 0101 0011 in turn.

- **A.** The instruction `add $s0, $s0, $s1` is executed. Will there be overflow? *(no overflow; no overflow)*
- **B.** What is the result in hex? *(0x1D100002; 0x6FFFB353)*
- **C.** What is the result in decimal? *(487,587,842; 1,879,028,563)*

5. Consider the hexadecimal numbers 0xAE0BFFFCand 0x8D08FFC0.

- **A.** Convert them to binary. *(1010 1110 0000 1011 1111 1111 1111 1100; 1000 1101 0000 1000 1111 1111 1100 0000)*
- **B.** Convert them to decimal (assuming unsigned representation). *(2,920,022,012; 2,366,177,216)*
- **C.** What MIPS instructions do they represent? *(`sw $t3, -4($s0); lw $t0, -64($t0)`)*

6. Consider one instruction with the fields `op` = 0, `rs` = 1, `rt` = 2, `rd` = 3, `shamt` = 0, and `funct` = 32, and a second instruction with fields `op` = 0x2B, `rs` = 0x10 , `rt` = 0x5, `const` = 0x4.

A. Are these instructions R-type, I-type, or J-type? How can you tell? *(R-type, I-type)*

B. What instructions are they? *(add $v1, $at, $v0; sw $a1, 4($s0))*

C. What are their raw, binary machine-language equivalents?
(0000 0000 0010 0010 0001 1000 0010 0000; 1010 1110 0000 0101 0000 0000 0000 0100)

Programming

Download the file `array.asm`. Currently, it does three things:

1. It allocates 20 bytes to use as an array of five 4-byte ints.
2. It stores the values {`1,2,3,4,5`} into that array.
3. It exits cleanly via a syscall.

Your job is to add code after the array's initialization which does the following (in order):

1. Subtract `array[0]` from `array[2]`, and store the result in `array[0]`.
2. Add `array[0]`, `array[2]`, and `array[4]`, and store the result in `array[4]`.
3. Bitwise-or `array[1]` with `array[3]`, storing the result in `array[1]`.
4. Shift-left `array[1]` by 2, storing the result in `array[1]`.
5. Bitwise-and `array[1]` with 21, storing the result in `array[3]`.
6. Bitwise-invert `array[4]`, storing the result in `array[2]`.
7. Print the array nicely .

Some hints:

- Write the code to print the array first. That way, you can test it more easily. It might also help to do the math on paper first, so you know what to expect.
- The final array should be {`2,24,-11,16,10`}.

Turn in your source code with the same name, `array.asm`.