1) Number Conversion

1. 1010 01012 -> 4 digit hexadecimal

0101 = 5, 1010 = 10 (which is A in Hexadecimal), but we want 4-digit hexadecimal so we write,

Therefore, 0000 0000 1010 01012 = **0x00A516**

1. 101010 -> 16 bit signed number

* 1010/2 = 505 R0, 505/2 = 252 R .5, 252/2 = 126 R0, 126/2 = 63 R0, 63/2 = 31 R .5, 31/2 = 15 R .5, 15/2 = 7 R .5, 7/2 = 3 R .5, 3/2 = 1 R .5, 1/2 = 0 R .5, 0/2 = 0
* Therefore (from last divided remainder to first divided), we have **0b0000 0011 1111 0010**

1. 55AA16 -> 16 bit binary

516 = 01012

A16 = 10102

Therefore, the 16-bit binary representation of 55AA16 is **0101 0101 1010 10102**

1. 222210 -> 4 digit hexadecimal

First convert to binary

* 2222/2 = 1111 R 0, 1111/2 = 555 R .5, 555/2 = 277 R .5, 277/2 = 138 R .5, 138/2 = 69 R 0, 69/2 = 34 R .5, 34/2 = 17 R 0, 17/2 = 8 R .5, 8/2 = 4 R 0, 4/2 = 2 R 0, 2/2 = 1 R 0, 1/2 = 0 R .5
* Where there is a remainder, there is a “1z”, where there isn’t, there is a ”02”. I then find that (reading from the last division to the first):
  + 1000 1010 11102 = 222210
    - But to have a 4 digit hex number, we need a 16 bit binary representation, therefore 0000 1000 1010 11102

Now we have a 16 bit binary representation of 222210. All we need now is to find the hex representation of each 4 bit grouping in the 16 bit binary number.

(Left to Right)

* 00002 = 016
* 10002 = 816
* 10102 = A16
* 11102 = E16

Therefore, our 4 digit hexadecimal representation of 22222 is **0x08AE16**

1) Number Conversion (cont.)

e) 101116 -> 16 bit binary

116 = 00012; 016 = 0000

Therefore, we have **0001 0000 0001 00012**

2) Binary Addition (Key: #|xxxx xxxx -> # = carried number; x = binary numbers)

1. 0 0 1 0 1 0 1 00

+ 0 0 1 0 1 0 1 0

0|0 1 0 1 0 1 0 0

0 + 0 = 0 in binary, 1 + 1 = 0 w/carried 1 in binary, 1 + 1 + 1 = 1 w/carried 1 in binary, 1 + 0 = 1

Therefore, 0010 1010 + 0010 1010 = **0**|**0101 01002**

1. 1 0 1 0 0 0 1 00

+1 0 1 0 1 0 0 0

1|0 1 0 0 1 0 1 0

Therefore, 1010 0010 + 1010 1000 = **1**|**0100 10102**

1. 1 0 1 0 0 0 1 00

+1 0 1 0 1 0 0 0

1|0 1 0 0 1 0 1 0

Therefore, 1010 0010 + 1010 1000 = **1**|**0100 10102**

1. Yes. Because B & C’s carryout numbers (1 in these cases) differ from their initial carry in (0 in these cases), there will be an overflow and a number too large to be represented will be produced.

3) Masking

1. Force bit 4 in the variable A, HIGH

**A |= 0x0010**, because 0x0010 = 0000 0000 0001 0000

1. Force bit 9 in the variable B, LOW

**B &= ~(0x0200)**

1. Toggle bit 6 in the variable A.

**A = A ^ 0x0040**

1. Toggle bit 6 in the variable A.

**A = A ^ 0x0040**

4) Conditionals

1. If (A > 5 && A < 0)

**This will always be false. ‘A’ will never be greater than 5 & less than 0 at the same time. The conditional will never be true.**

1. If (A)

**‘A’ could be any value (except 0) and this conditional would execute.**

**Therefore, A = all values, excluding 0**

1. If(A < 10 && A > -10)

**This will only be true when ‘A’ is inside the range {-9, 9} or**

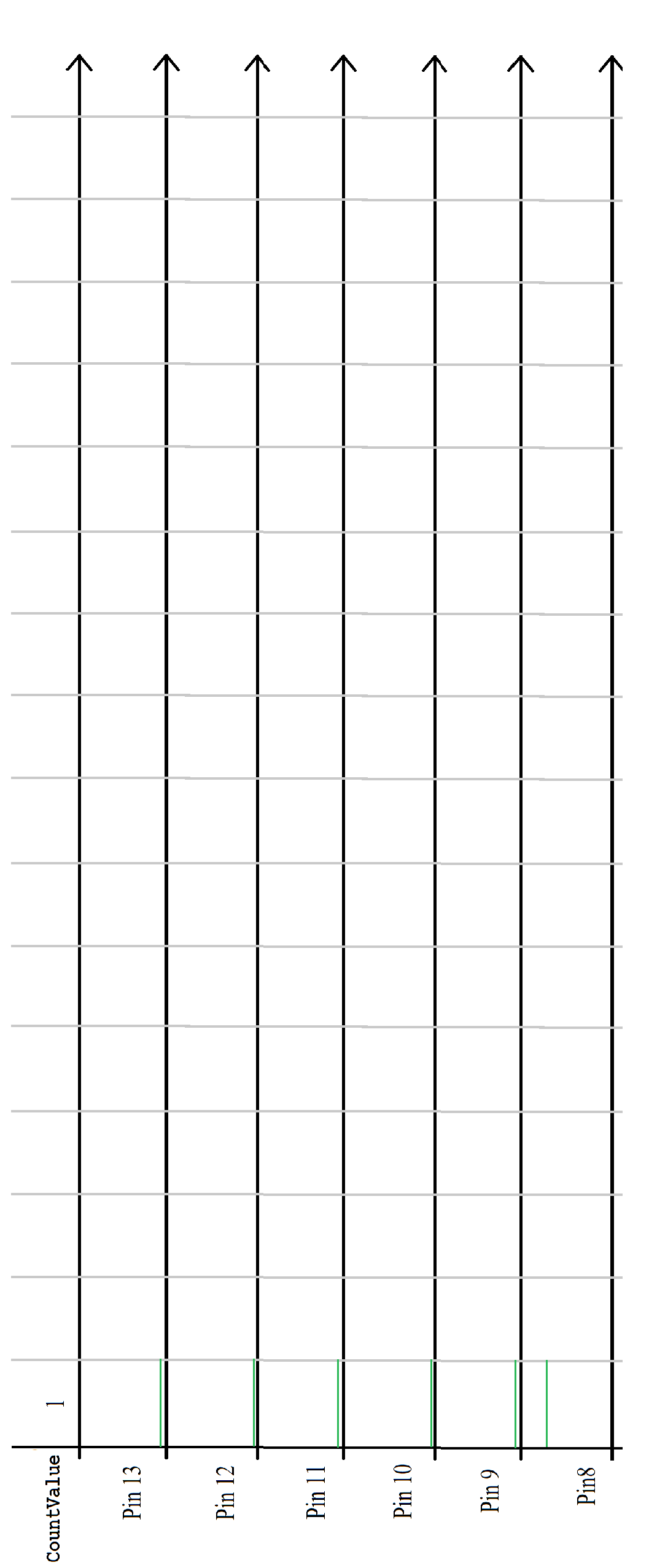
**-9. -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9**

1. If(A & 0x0042)

**0010 = 2; 0100 = 4; so 0000 0000 0100 0010**

**If bit 1 and 6 are, HIGH in A, the conditional statement will be true and execute**

5) Code Reading/Timing



#include <MsTimer2.h> int CountValue = 1; void CountValueISR() { // Compute next output value.

// if bit 5 set is equal to bit 4

if (bitRead(CountValue, 5) == bitRead(CountValue, 4))

CountValue = CountValue << 1; // shift to the left else

CountValue = (CountValue << 1) // shift to the left + 1; // and set bit 0.

CountValue &= 0x3f; // Send output to portb, or pins 13-8.

PORTB = CountValue;

} // End of ISR

// Runs at start up to initialize system. void setup() { DDRB |= 0x3f; // Set pins 13-8 as outputs.

// set up Timer2 for 50 millisecond interval.

MsTimer2::set(50, CountValueISR);

MsTimer2::start();

} // End of setup()

// loop called continuously. void loop() { // Nothing in loop.

} // End of loop.