

6.004 Worksheet Questions

L01 – Binary Encoding and Arithmetic

Problem 1. Encoding positive integers

1. What is the 5-bit binary representation of the decimal number 21? ★

Multiple ways to solve this: a) repeated division by 2 with remainder (build bits from right to left), b) repeated subtraction of powers of 2 (build bits from left to right)

- a. $21 / 2 = 10$ remainder 1 (least significant bit)
 $10 / 2 = 5$ remainder 0
 $5 / 2 = 2$ remainder 1
 $2 / 2 = 1$ remainder 0
 $1 / 2 = 0$ remainder 1 (most significant bit)
 $\rightarrow 10101$
- b. Find largest power of 2 that is less than or equal to remaining value and subtract it until you get to zero
 $21 - 16 = 5$
 $5 - 4 = 1$
 $1 - 1 = 0$
Fill in bit positions for the powers you used with 1's, other positions with 0's:
 $16 = 2^4 \rightarrow 0b\underline{1}0000$
 $4 = 2^2 \rightarrow 0b10\underline{1}00$
 $1 = 2^0 \rightarrow 0b1010\underline{1}$

2. What is the hexadecimal representation for decimal 219 encoded as an 8-bit binary number?

1. Convert decimal 219 to binary as in part 1 (repeated subtraction used here):
 $219 = 128 + 64 + 16 + 8 + 2 + 1 \rightarrow 0b11011011$
2. Group bits into sets of four starting on the right:
 $0b11011011 \rightarrow 0b1101_1011$
3. Lookup hex value for each set of bits in table (see lecture 1):
 $0b1101 \rightarrow 0xD$, $0b1011 \rightarrow 0xB$: $0b1101_1011 = 0xDB$

3. What is the hexadecimal representation for decimal 51 encoded as a 6-bit binary number?

Same as in part 2 but you add extra 0's on the left if the last group has fewer than 4 bits (i.e., total number of bits is not evenly divisible by 4):

$$51 = 32 + 16 + 2 + 1 \rightarrow 0b110011 \rightarrow 0b11_0011 \rightarrow 0b0011_0011 \rightarrow 0x33$$

4. The hexadecimal representation for an 8-bit unsigned binary number is 0x9E. What is its decimal representation? ★

1. Convert hex to binary by looking up bit pattern for each hex digit in table:
 $0x9E = 0b1001_1110$
2. Convert binary to decimal by multiplying each bit by the decimal value for that position:
 $0b1001_1110 \rightarrow 1 \cdot 2^7 + 0 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$
 $= 128 + 0 + 0 + 16 + 8 + 4 + 2 + 0 = 158$
(Shortcut: ignore the zeros in the binary number and just sum the values for the ones.)

5. What is the range of integers that can be represented with a single unsigned 8-bit quantity?

Unsigned always goes from zero to $2^N - 1$, where N is the number of bits. $2^8 - 1 = 255$ so the range is from 0 to 255.

6. Since the start of official pitching statistics in 1988, the highest number of pitches in a single game has been 172. Assuming that remains the upper bound on pitch count, how many bits would we need to record the pitch count for each game as an unsigned binary number?

At least three options:

- Convert 172 to binary as above and count digits [$172 = 0b10101100$, 8 bits]. This is the obvious approach but also the most work.
 - Find **smallest** power of two **larger** than 172 and take exponent [$256 = 2^8$]. This one is good if you have the powers of 2 memorized.
 - Take the ceiling of the log base 2 [$\text{ceil}(\log_2(172+1)) = 8$]. Note that we need to add one to 172 because the $\log_2 N$ is actually giving us the number of bits needed to represent N different values. If we need to be able to represent all the numbers from 0 to 172, that is actually 173 different values. We take the ceiling because we can't have a fractional number of bits and this would give us the extra value we need, except for the corner case where N is an exact power of 2.
7. Compute the sum of these two 4-bit unsigned binary numbers: $0b1101 + 0b0110$. Express the result in hexadecimal. You may use 5 bits to represent the solution if needed. ★

Add bits in each position from right to left, just as you would with decimal numbers. However, express the individual sums in binary and carry if the sum is a two-bit number (0b10 or 0b11). Convert result to hex by grouping and adding zeros as above.

Note: The solution here uses 5 bits even though the addends are 4-bit values. This is because we are dealing with *unsigned* values. For the rest of the course, we can assume two's complement (signed) values. Unless otherwise instructed, perform two's complement arithmetic, which requires discarding the final carry bit and checking for overflow (as discussed in the solution to problem 2-6). Hardware is fixed in size, so two's complement arithmetic is actually computing **sum % 2^N**

$$\begin{array}{r} 11 \\ 1101 \\ +0110 \\ \hline 10011 = 1_0011 = 0001_0011 = 0x13 \end{array}$$

Problem 2. Two's complement representation

1. What is the 6-bit two's complement representation of the decimal number -21?

Easiest method is to find the representation of +21 and then use the fact that we can negate a two's complement binary number by inverting all the bits individually (bitwise NOT) and then adding 1 to the result:

$21 = 16+4+1 = 0b010101$, (we were asked for 6 bits so fill in the other 3 positions with 0's)
Inverting each bit individually gives: $0b101010$
 $-21 = 0b101010 + 1 = \mathbf{0b101011}$

Note that if you are *only* given a sequence of bits, you don't actually know what number they represent. You also need be told how to *interpret* those bits: as an unsigned number or as a signed number stored in two's complement representation. For example, $0b101011$ represents -21 if those bits are interpreted as two's complement but 43 ($32+8+2+1$) if they are interpreted as unsigned. The same sequence of bits can *mean* two different things. In fact, there are many other possible ways to interpret these bits. For example, we might have meant for each bit to represent a simple yes/no flag for six different things. We will see another possible interpretation in Problem 4. For the rest of this course, you may assume two's complement unless you are told otherwise.

2. What is the hexadecimal representation for decimal -51 encoded as an 8-bit two's complement number?

Hexadecimal is just a shorthand way of writing down a sequence of bits. Therefore, it doesn't matter what those bits represent. To solve this problem we find the two's complement binary representation of -51 and then translate groups of four bits just as we did in Problem 1 for unsigned numbers:

$51 = 32+16+2+1 = 0b0011_0011$ (using 8-bit binary)
 $-51 = 0b1100_1100 + 1 = 0b1100_1101 = \mathbf{0xCD}$

3. The hexadecimal representation for an 8-bit two's complement number is 0xD6. What is its decimal representation?

Again, hexadecimal is just a compressed form of binary so the first step is to decompress it:
 $0xD6 = 0b1101_0110$

Now, we are told that we are supposed to interpret these bits as two's complement. The process is the same as for signed numbers except that we flip the sign for the value represented by the most significant bit:

$0b1101_0110 = 1*\underline{-128} + 1*64 + 0*32 + 1*16 + 0*8 + 1*4 + 1*2 + 0*1 = -42$

Alternative (*may* be easier, especially if more than half the bits are 1):

$0xD6 = 0b1101_0110$

By looking at the most significant bit (MSB), we can see that this number must be negative (for two's complement, MSB=1 implies negative and MSB=0 implies non-negative). So we can negate this binary number (by flipping all bits and adding 1), find the decimal equivalent (exactly as for unsigned numbers since we now know the MSB=0), and then negate that (which is just adding a minus sign in front for decimal numbers):

$-0xD6 = 0b0010_1001 + 1 = 0b0010_1010 = 32+8+2 = 42$, which gives $+0xD6 = -42$

4. Using a 5-bit two's complement representation, what is the range of integers that can be represented with a single 5-bit quantity?

$$-2^{N-1} \text{ to } 2^{N-1}-1 \rightarrow -2^4 \text{ to } (2^4)-1 \rightarrow -16 \text{ to } 15$$

Note that the negative range is one larger than the positive range.

5. Can the value of the sum of two 2's complement numbers $0xB3 + 0x47$ be represented using an 8-bit 2's complement representation? If so, what is the sum in hex? If not, write NO. ★

Yes: from the MSBs ("sign" bits) $0xB3$ is negative and $0x47$ is positive. The sum of a negative number and a positive number must always have an absolute value less than or equal to either addend (*i.e.*, the sum will be in between the addends on a number line). Since the addends were in range, the sum must be too.

$$0xB3 + 0x47 =$$

$$0b1011_0011 +$$

$$0b0100_0111 =$$

$$0b1111_1010 = \mathbf{0xFA} \text{ (in decimal: -6, can you see why it's a very small negative?)}$$

6. Can the value of the sum of two 2's complement numbers $0xB3 + 0xB1$ be represented using an 8-bit 2's complement representation? If so, what is the sum in hex? If not, write NO. ★

This one is trickier. $0xB3$ and $0xB1$ are both negative so their sum will be more negative than either addend and *might* be outside the range we can represent with 8 bits (-128 to 127). For example, if they were both -1, we would be fine but if they were both -128, we would be in trouble. To check, we have to actually perform the addition and check the result:

$$0xB3 + 0xB1 =$$

$$0b1011_0011 +$$

$$0b1011_0001 =$$

$$0b0110_0100$$

There was a carry from the 8th bit to the 9th bit, but when performing two's complement arithmetic, we drop any carries beyond the MSB so that the result stays the same size. Remember that we are actually computing the result modulo 2^N . (The reason for this is that hardware is built with a fixed number of bits for each piece of data.) The fact that we dropped the carry is not necessarily bad. To see if our result is valid, we can compare the signs of the addends and the sum. In this case, we summed two negative numbers but the result came out positive (MSB of $0b0110_0100$ is 0). Since this is impossible, we know that the result is outside the range and the answer is **NO**.

Here is the complete process for detecting overflow (results out of range) for two's complement addition: If the sign bits of the addends are the different, the result will be in range. If they are the same, perform the addition (mod N) and compare the sign bit of the result to the sign bit of one of the addends. If they match, the result is in range, otherwise no.

7. Please compute the value of the expression $0xBB - 8$ using 8-bit two's complement arithmetic and give the result in decimal (base 10).

To perform subtraction, we simply negate the second operand and perform addition. (That way we only need to put an adder in our machine instead of an adder and a subtractor. We will see more about this in later lectures.)

$$\begin{aligned} 0xBB - 8 &= 0xBB + (-8) = 0b1011_1011 + 0b1111_1000 \\ 0b1011_1011 &+ \\ 0b1111_1000 &= \\ 0b1011_0011 &= -128 + 32 + 16 + 2 + 1 = -77 \end{aligned}$$

Note that the sign bits of the addends are the same but the sign bit of the sum matches. So the result is valid, even though we had a carry to the 9th bit that we dropped.

8. Consider the following subtraction problem where the operands are 5-bit two's complement numbers. Compute the result and give the answer as a decimal (base 10) number. ★

$$\begin{array}{r} 10101 \\ - \underline{00011} \end{array}$$

$$\begin{aligned} &0b10101 \quad \quad 0b10101 \quad \quad 0b10101 \\ - &0b00011 \Rightarrow + 0b11100 + 1 \Rightarrow + 0b11101 \\ &= 0b10010 = -(0b01101 + 1) = -14 \end{aligned}$$

This solution combines the technique for performing subtraction from part 7 with the alternative technique from converting from binary to decimal from part 3.

Problem 3. Multiples of 4

1. Given an unsigned n -bit binary integer $= b_{n-1} \dots b_1 b_0$, prove that v is a multiple of 4 if and only if $b_0 = 0$ and $b_1 = 0$.

To solve this problem, remember that $v = \sum_{k=0}^{n-1} b_k 2^k$

- Powers of 2 greater than or equal to 4 are multiples of 4 (for all $k \geq 2$, $2^k = 4 \cdot 2^{k-2}$ and 2^{k-2} is an integer)
- The sum of numbers evenly divisible by 4 is evenly divisible by 4: $4x + 4y = 4(x+y)$. Therefore, any number of the form $b_{n-1} \dots 00$ is a multiple of 4.

More intuitively, if a number ends in one of 01, 10, or 11 we are adding 1, 2, or 3 respectively to a multiple of 4 (the sum of factors from all the higher bits). Therefore, a number is a multiple of 4 only if it ends in 00.

2. Does the same relation hold for two's complement encoding?

Yes, the above proof still works. In two's complement notation the highest-order bit represents -2^{n-1} instead of $+2^{n-1}$, and negating a number does not change its divisibility (-2^{n-1} is still evenly divisible by 4 for $n \geq 3$).

Problem 4. Encoding text

There are multiple standards to encode characters and strings using binary values. ASCII is a classic standard to encode English alphabet characters (modern formats like UTF support other alphabets, but are typically based on ASCII). ASCII encodes each character using an 8-bit (1-byte) value. The table below shows ASCII's mapping of characters to values.

ASCII (1977/1986), adapted from <https://en.wikipedia.org/wiki/ASCII>

| | <u>0</u> | <u>1</u> | <u>2</u> | <u>3</u> | <u>4</u> | <u>5</u> | <u>6</u> | <u>7</u> | <u>8</u> | <u>9</u> | <u>A</u> | <u>B</u> | <u>C</u> | <u>D</u> | <u>E</u> | <u>F</u> |
|----|---------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|------------|-------------|-------------|------------|------------|------------|-------------|
| 0_ | NUL 0x00 | SOH 0x01 | STX 0x02 | ETX 0x03 | EOT 0x04 | ENQ 0x05 | ACK 0x06 | BEL 0x07 | BS 0x08 | HT 0x09 | LF 0x0A | VT 0x0B | FF 0x0C | CR 0x0D | SO 0x0E | SI 0x0F |
| 1_ | DLE 0x10 | DC1 0x11 | DC2 0x12 | DC3 0x13 | DC4 0x14 | NAK 0x15 | SYN 0x16 | ETB 0x17 | CAN 0x18 | EM 0x19 | SUB 0x1A | ESC 0x1B | FS 0x1C | GS 0x1D | RS 0x1E | US 0x1F |
| 2_ | space 0x20 | ! 0x21 | " 0x22 | # 0x23 | \$ 0x24 | % 0x25 | & 0x26 | ' 0x27 | (0x28 |) 0x29 | * 0x2A | + 0x2B | , 0x2C | - 0x2D | . 0x2E | / 0x2F |
| 3_ | 0 0x30 | 1 0x31 | 2 0x32 | 3 0x33 | 4 0x34 | 5 0x35 | 6 0x36 | 7 0x37 | 8 0x38 | 9 0x39 | : 0x3A | ; 0x3B | < 0x3C | = 0x3D | > 0x3E | ? 0x3F |
| 4_ | @ 0x40 | A 0x41 | B 0x42 | C 0x43 | D 0x44 | E 0x45 | F 0x46 | G 0x47 | H 0x48 | I 0x49 | J 0x4A | K 0x4B | L 0x4C | M 0x4D | N 0x4E | O 0x4F |
| 5_ | P 0x50 | Q 0x51 | R 0x52 | S 0x53 | T 0x54 | U 0x55 | V 0x56 | W 0x57 | X 0x58 | Y 0x59 | Z 0x5A | [0x5B | \ 0x5C |] 0x5D | ^ 0x5E | _ 0x5F |
| 6_ | ` 0x60 | a 0x61 | b 0x62 | c 0x63 | d 0x64 | e 0x65 | f 0x66 | g 0x67 | h 0x68 | i 0x69 | j 0x6A | k 0x6B | l 0x6C | m 0x6D | n 0x6E | o 0x6F |
| 7_ | p 0x70 | q 0x71 | r 0x72 | s 0x73 | t 0x74 | u 0x75 | v 0x76 | w 0x77 | x 0x78 | y 0x79 | z 0x7A | { 0x7B | 0x7C | } 0x7D | ~ 0x7E | DEL 0x7F |

 Letter
 Number
 Punctuation
 Symbol
 Other/non-printable

Computers often store variable-length text as a null-terminated string: a sequence of bytes, where each byte denotes a different character, terminated by the value 0x00 (null) to denote the end of the string. For example, the string “6.004” is encoded as the 6-byte sequence 0x36 0x2E 0x30 0x30 0x34 0x00. For brevity, we can also just stick these hex values together to form one large hex number: 0x362E30303400.

1. Encode your name as a null-terminated ASCII string (use the best approximation if your name contains non-English characters)

0x 59 6F 75 72 20 4E 61 6D 65 20 3A 29 00

2. Decode the following null-terminated ASCII string:

0x 52 49 53 43 2D 56 20 69 73 20 63 6F 6D 69 6E 67 21 00

RISC-V is coming!

Problem 5. From Past Quizzes

(A) What is the maximum decimal value that can be represented in **7-bit unsigned binary**? What is the minimum decimal value that can be represented in **6-bit 2's complement**?

Largest 7-bit unsigned = $0b1111111 = 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 64 + 32 + 16 + 8 + 4 + 2 + 1 = 127$
Smallest 6-bit 2's complement = $0b100000 = -2^5 = -32$

Largest 7-bit unsigned binary (in decimal): 127

Smallest 6-bit 2's complement number (in decimal): -32

(B) What is -25 in 7-bit 2's complement encoding? What is -40 in 7-bit 2's complement encoding? Show how to compute -25-40 using 2's complement addition. Is it possible to represent the result in 7-bit 2's complement encoding? If so, **show your binary addition work and write the result in binary**. If not, write “**Not Possible**” and **explain why it's not possible**.

$25 = 0b0011001 \rightarrow -25 = \sim(0b0011001) + 1 = 0b1100111$

$40 = 0b0101000 \rightarrow -40 = \sim(0b0101000) + 1 = 0b1011000$

+ -----

0b1011111

(Taking 7 bits, it becomes positive)

\rightarrow Not possible

(Alternative) 7-bit 2's complement range: $-2^7 \sim (2^7 - 1) = -64 \sim 63 \rightarrow$ Not possible

-25 in 7-bit 2's complement notation (0b): 1100111

-40 in 7-bit 2's complement notation (0b): 1011000

-25 -40 in 7-bit 2's complement notation or “Not Possible” (0b): Not Possible