Code structure:

I tried to structure a lot of steps into different procedures while keeping the main procedure (`_start`) short. For example, my program first checks if the first input is a valid integer, then checks if the second input is a valid positive integer, then checks if the given number of bits still makes the first input valid. I could've structured the third part a little better since it does a lot (converts first input from string to unsigned integer value, computes signed and unsigned bounds, and computes absolute value of signed value) (could've split up the conversions from decimal, binary, and hexadecimal into separate procedures). I also have a lot of helper procedures, especially for dealing with 128-bit integers. Specifically for the Windows version, I created two helper functions for reading from stdin and writing to stdout. Since I wrote all of this in NASM syntax and there's no actual difference between a procedure name and label, I made all procedures start with a _. Also none of the procedures preserve any registers, so whenever I needed a register value to persist I would just push and pop. I probably should've made all my procedures preserve registers but oh well.

Program flow:

- Print first prompt and read in first input (integer) and flush stdin (don't want it to contain anything for the next read)
    - Linux: If read 0 bytes or first byte = \n, exit program
    - Windows: If read < 2 bytes or second byte = \n, exit program
- Validate first input
    - Check the radix (does it start with 0b / 0x / neither)
    - Check if it's a valid integer (check if each digit in the input is a valid digit for its radix)
    - If invalid, print an error message and jump back to the beginning of the program
- Print second prompt and read in second prompt (number of bits) and flush stdin
    - Check if user exited program again
- Validate second input
    - Check if each digit is in [0,9]
    - Convert string to integer and check if > 0 and <= 128
    - If invalid, print error message and jump back to start
- Check if first input is valid for given number of bits
    - Convert first input from string to unsigned integer
    - Check and store whether or not the input was a negative integer
        - Decimal: check if first byte is -
        - Binary: check if number of bits >= given number of bits and if MSB is 1
            - Checking for >= since user can put a lot of delimiters in as well
        - Hexadecimal:
            - numBits % 4 == 0: check if most significant digit >= 8
            - numBits % 4 == 1: check if most significant digit >= 1
            - numBits % 4 == 2: check if most significant digit >= 2
            - numBits % 4 == 3: check if most significant digit >= 4
    - Compute max unsigned and signed value for given number of bits (to be used for bound check and 2's complement)
    - Convert unsigned integer to signed (if necessary) and check if within bounds
        - If positive, then unsigned val == signed val → check if <= max signed value
        - If negative, take 2's complement of unsigned val, check if the signed integer is actually negative, and then check is abs(signed int) <= abs(min signed value)
            - Only take 2's complement if unsigned value was computed from binary/hexadecimal
            - 2's complement: (unsigned int) - (max unsigned val + 1)
                - Take absolute value by doing 0 - (signed int)
    - Print integer in other 2 radices
        - Decimal: divide abs(signed int) by 10, get each digit, and put - in first byte if negative
        - Binary: divide unsigned int by 2, get each digit, and sign extend
        - Hexadecimal: divide unsigned int by 16, get each digit, and sign extend
            - If unsigned int val was computed from a decimal number, convert it to be the value it would've been had it been computed from a binary / hexadecimal number
                - From decimal, unsigned int val == abs(signed int)
                - If number is negative, do 0 - (unsigned int) + (max unsigned val) + 1

## How to compile and run:

For both Windows and Linux, you must have NASM installed.
I used this version of NASM for Windows: https://www.nasm.us/pub/nasm/releasebuilds/3.01/win32/
You will also need to update your PATH environment variable to include the path to nasm.exe

Compile for Linux / Windows by running `make linux` / `make windows` in `zuhays/part_2`
Run the program by running `bin/linux/linux.out` / `bin\windows\windows.exe`

## User manual:

This program converts a signed integer between decimal, binary, and hexadecimal. The user is prompted to enter a signed integer and the number of bits that the integer is to be stored in. If the inputs are valid, the signed integer in the other 2 radices is given. Otherwise, an error message is printed and the user is reprompted for an integer and number of bits. The user can quit out of the program by pressing enter.

Valid inputs:

- Signed integer
  - Can be in decimal, binary, or hexadecimal
  - Decimal:
    - Each digit must be from 0-9
    - First character in the input can be - to indicate a negative integer
  - Binary
    - Each digit must be 0 or 1
    - Prefixed by 0b
    - Can put delimiters ( ' ) anywhere in the input after 0b and the first digit
  - Hexadecimal
    - Each digit must be from 0-9 or a-b (case-insensitive)
    - Prefixed by 0x
- Number of bits
  - Must be a whole number n, $0 < n <= 128$
  - Signed integer must fit in the range $[-2^{n-1}, 2^{n-1} - 1]$
  - Binary:
    - If the number of digits given in the binary string == n, then the first digit is the sign bit (0 = positive, 1 = negative)
  - Hexadecimal:
    - If ceil((number of digits) / 4) == n, then the first digit determines the sign
      - If numBits % 4 == 0, then the first digit being in [8,f] indicates a negative number
      - If numBits % 4 == 1, then the first digit being 1 indicates a negative number
      - If numBits % 4 == 2, then the first digit being in [2,3] indicates a negative number
      - If numBits % 4 == 3, then the first digit being in [4,7] indicates a negative number

Examples:

```
meep633@Siyan:~/Classes/Windows/CSCI/SA&D/HW/HW2/zuhays/part_2$ bin/linux.out
Input a signed value (at most 128 bits) to convert or hit Enter to exit.
Use 0x for hexadecimal, 0b for binary, and no prefix for decimal: 1
Enter the number of bits this value uses or hit Enter to exit: 4
Binary: 0b0001
Hexadecimal: 0x1

Input a signed value (at most 128 bits) to convert or hit Enter to exit.
Use 0x for hexadecimal, 0b for binary, and no prefix for decimal: -1
Enter the number of bits this value uses or hit Enter to exit: 1
Binary: 0b1
Hexadecimal: 0x1

Input a signed value (at most 128 bits) to convert or hit Enter to exit.
Use 0x for hexadecimal, 0b for binary, and no prefix for decimal: -1
Enter the number of bits this value uses or hit Enter to exit: 2
Binary: 0b11
Hexadecimal: 0x3

Input a signed value (at most 128 bits) to convert or hit Enter to exit.
Use 0x for hexadecimal, 0b for binary, and no prefix for decimal: -1
Enter the number of bits this value uses or hit Enter to exit: 3
Binary: 0b111
Hexadecimal: 0x7

Input a signed value (at most 128 bits) to convert or hit Enter to exit.
Use 0x for hexadecimal, 0b for binary, and no prefix for decimal: -1
Enter the number of bits this value uses or hit Enter to exit: 4
Binary: 0b1111
Hexadecimal: 0xf

Input a signed value (at most 128 bits) to convert or hit Enter to exit.
Use 0x for hexadecimal, 0b for binary, and no prefix for decimal: 0b1001
Enter the number of bits this value uses or hit Enter to exit: 4
Decimal: -7
Hexadecimal: 0x9

Input a signed value (at most 128 bits) to convert or hit Enter to exit.
Use 0x for hexadecimal, 0b for binary, and no prefix for decimal: 0b1001
Enter the number of bits this value uses or hit Enter to exit: 5
Decimal: 9
Hexadecimal: 0x09

Input a signed value (at most 128 bits) to convert or hit Enter to exit.
Use 0x for hexadecimal, 0b for binary, and no prefix for decimal: 0x1a
Enter the number of bits this value uses or hit Enter to exit: 5
Decimal: -6
Binary: 0b11010

Input a signed value (at most 128 bits) to convert or hit Enter to exit.
Use 0x for hexadecimal, 0b for binary, and no prefix for decimal: 0x1a
Enter the number of bits this value uses or hit Enter to exit: 6
Decimal: 26
Binary: 0b011010

Input a signed value (at most 128 bits) to convert or hit Enter to exit.
Use 0x for hexadecimal, 0b for binary, and no prefix for decimal:
Thank you for using our number conversion tool!
```

```
Input a signed value (at most 128 bits) to convert or hit Enter to exit.
Use 0x for hexadecimal, 0b for binary, and no prefix for decimal: 1
Enter the number of bits this value uses or hit Enter to exit: 0
Invalid input

Input a signed value (at most 128 bits) to convert or hit Enter to exit.
Use 0x for hexadecimal, 0b for binary, and no prefix for decimal: invalid int
Invalid input

Input a signed value (at most 128 bits) to convert or hit Enter to exit.
Use 0x for hexadecimal, 0b for binary, and no prefix for decimal: 1
Enter the number of bits this value uses or hit Enter to exit: -1
Invalid input
```