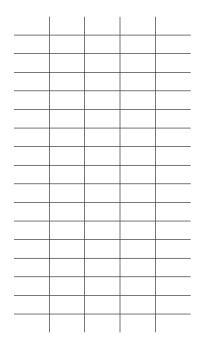
Combinational Design

Your friend¹ is rather picky about what pizzas they are willing to eat. From your observations, they will only eat pizzas that conform to the following rules:

- 1. The pizza should have exactly one vegetable (either mushrooms or peppers).
- 2. The pizza should not have bacon unless it also has mushrooms.
- 3. The pizza shouldn't have both sausage and peppers.

Write a boolean expression, L, that evaluates to true for any combination of ingredients that your friend is willing to eat on pizza. (Use B = bacon, M = mushrooms, P = peppers, S = sausage.) A blank truth table is provided for your convenience.

L =



¹FWIW, my own rules for pizza are much simpler – it must only have cheese, and lots of it

Compiler Magic

GCC's optimizer converts the absolute value function, where an int is 32 bits wide:

```
int abs(int num) {
    if (num >= 0) {
        return num;
    } else {
        return -num;
    }
}
into the following:
int abs(int num) {
    int temp = num >> 31;
    return (num ^ temp) - temp;
}
```

You'll learn why this is an optimization when we talk about pipelining and branch prediction. For now, explain why this works. Recall that ^ is the bitwise XOR operator, and how 2's complement negation and right shifts on signed quantities work.

Bit-wise Fun

Given an 8-bit signed integer x, write expressions using only bit-wise logical and shifting operators to do the following:

- Check if x is odd (*i.e.*, the expression should evaluate to 0 when x is even and 1 when x is odd).
- Invert only the least significant bit of x.
- Set bit 2 of x (setting a bit means making it 1, and recall that the least significant bit is numbered bit 0).
- Clear bit 1 of x (clearing a bit means making it 0).
- Multiply x by 8.
- Extract bits 4 through 7 of x (i.e., if x is $b_7b_6b_5b_4b_3b_2b_1b_0$, your expression should evaluate to $0000b_7b_6b_5b_4$).
- Round x down to the nearest smaller multiple of 4 (e.g., 9 should become 8, 4 should remain 4, and -9 should become -12).

Putting that 173 pre-req to use

Recall that in 2's complement, $-x \equiv \sim x+1$. Using this, prove that $-x \equiv \sim (x-1)$. Then use this new identity to explain what the following expression accomplishes: (x + 3) & -4. Hint: Try it out on a few numbers to see what's happening.