BF2NASM - SMARTER BRANCHING

8DCC

1. Pretested and posttested loops

When I was reading *Reversing: Secrets of Reverse Engineering* [1], I found this part specially interesting:

> The most common high-level loop construct is the pretested loop, where the loop's condition is tested before the loop's body is executed. The problem with this construct is that it requires an extra unconditional jump at the end of the loop's body in order to jump back to the beginning of the loop (for comparison, posttested loops only have a single conditional branch instruction at the end of the loop, which makes them more efficient). Because of this, it is common for optimizers to convert pretested loops to posttested loops. In some cases, this requires the insertion of an if statement before the beginning of the loop, so as to make sure the loop is not entered when its condition isn't satisfied.

The compiler will sometimes optimize the following pretested loop:

```
while (a != b) {
  ...
}
```

Into this posttested loop inside a conditional.

```
if (a != b) {
  do {
    ...
  } while (a != b);
}
```

The assembly for the first loop would be something like:

```
loop:
  cmp A, B
  je done
  ...
  jmp loop
done:
```

And the assembly for the second loop could be translated literally to something like:

```
  cmp A, B
```

```
  je done
loop:
  ...
  cmp A, B
  jne loop
done:
```

As mentioned in the quote, on the first loop there is a conditional jump and an unconditional one, while in the second one, the first condition is tested once before the loop, and the loop only has one conditional jump. If the condition fails on the second loop, the jump will not be performed and the execution will continue at the `done` label.

2. Brainfuck loops

In brainfuck, loops are defined with square brackets. The code inside the loop will be ran as long as the value at the current cell is not zero. At first, I thought this was a posttested loop, but this is wrong. The loops are pretested, meaning that the loop will be jumped over entirely if the value at the current cell is not zero.

When the program encounters a loop start, the following assembly is generated:

```
  jmp .check_N
.loop_N:
  ...
.check_N:
  cmp byte [rcx], 0
  jnz .loop_N
```

Where N is the loop counter.

That way, we even avoid the first comparison from the other example, and we jump directly to the "end" of the loop, where the condition is checked. Now we only make an "extra" unconditional jump once, for checking the condition the first time when entering the loop.

References

[1] Eldad Eilam. (2005). *Reversing: Secrets of Reverse Engineering* (pp. 56-57).