

A switch/case Implementation in Assembly

Often when programming in assembly, we run across situations where we'd like to implement a programming construct from a high level language like C. C's `switch` statement finds many uses in embedded applications, but no such construct exists natively in assembly language. However, it is fairly easy to recreate the functionality using the microcontroller's fundamental operations along with an infrequently used macro feature of the assembler and some good old boolean algebra trickery that many of us have long since forgotten.

Boolean Algebra Refresher: The XOR Operator

In order to create the switch/case construct in assembly, we need to review some of the fundamental properties of the XOR operator.

Truth Table

Out	In ₁	In ₂
0	0	0
1	0	1
1	1	0
0	1	1

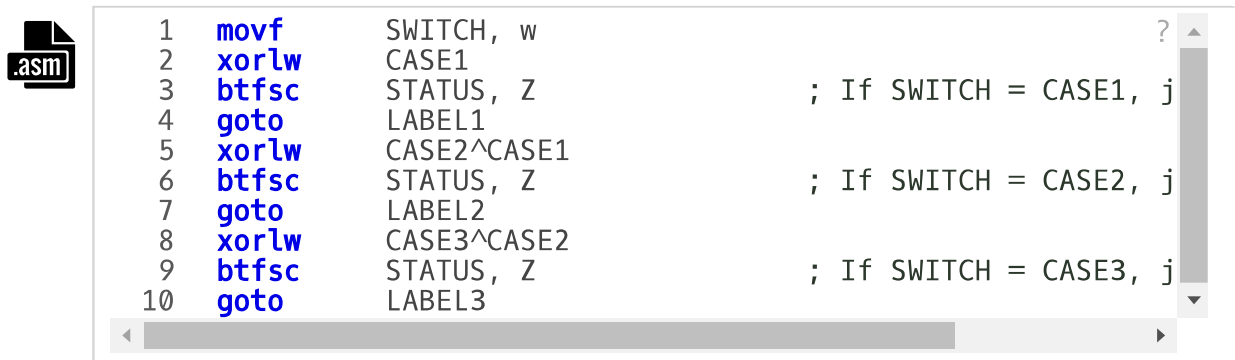
Algebraic Properties

Property	Notation	Description
Commutative	$A \oplus B = B \oplus A$	Operand order doesn't change the result
Associative	$(A \oplus B) \oplus C = A \oplus (B \oplus C)$	Operand grouping doesn't change the result
Non-Idempotency	$A \oplus A = 0$	XORing an operand with itself is zero
Identity	$A \oplus 0 = A$	XORing an operand with zero doesn't change the operand

The above properties are all used in combination with each other to perform a little trick of logic that will be outlined step-by-step below. Without a working knowledge of these properties, the code won't make much sense.

The Code

In this code snippet, `SWITCH` is a register (variable) that contains the value we are trying to match to each of the cases. The labels `CASE1`, `CASE2`, and `CASE3` are constants and are the values we are checking against the one stored in `SWITCH`. The labels `LABEL1`, `LABEL2`, and `LABEL3` are the names of subroutines we want to jump to for each match condition.



```

1  movf    SWITCH, w
2  xorlw   CASE1
3  btfsc   STATUS, Z           ; If SWITCH = CASE1, jump
4  goto    LABEL1
5  xorlw   CASE2^CASE1
6  btfsc   STATUS, Z           ; If SWITCH = CASE2, jump
7  goto    LABEL2
8  xorlw   CASE3^CASE2
9  btfsc   STATUS, Z           ; If SWITCH = CASE3, jump
10 goto    LABEL3

```

The code snippet above takes advantage of the properties of the XOR operator and the assembler's ability to perform calculations on constants at build time (more in the step-by-step analysis below). There are several variations floating around the net, so this is certainly not the only way to implement a `switch`-like construct in assembly. The code above should work on any 8-bit PIC[®] microcontroller. With minor modifications to the syntax, it should also work on a 16-bit PIC microcontrollers and dsPIC[®] Digital Signal Controllers.

Step-by-step Analysis

1 `movwf SWITCH, w`

This simply moves the value from the register labelled `SWITCH` into the W register. Mathematically, we can write this as:

$$W = \text{SWITCH}$$

2 `xorlw CASE1`

This performs a bitwise exclusive OR operation (at runtime) between the W register and the constant `CASE1`. The result is stored in W. Mathematically, this can be written as:

$$W = W \oplus \text{CASE1}$$

Substituting for the original value of W established on line 1:

$$W = \text{SWITCH} \oplus \text{CASE1}$$

3 `btfsc STATUS,Z`

This tests the outcome of the previous operation. If the result of $\text{SWITCH} \oplus \text{CASE1}$ is zero, then the Z bit in the STATUS register will be set. So what this line is saying is that if the Z bit is clear (the previous operation did not result in a zero), then skip the *next* instruction. The reason we are performing this test is based on the property of non-idempotency: $A \oplus A = 0$. So, if the value in `SWITCH` is the same value as `CASE1`, then $\text{SWITCH} \oplus \text{CASE1} = 0$. If that is the case, we found our match and want to execute the next instruction...

4 `goto LABEL1`

This line takes us to a subroutine with the name `LABEL1` to handle the situation when $\text{SWITCH} \oplus \text{CASE1} = 0$. If the Z bit in the STATUS register was not set above, then this instruction would be skipped and we would test the next condition.

5 `xorlw CASE1^CASE2`

This line is where most of the magic occurs. We are playing several tricks at once. The first thing to point out is that the '^' symbol is the XOR operator in the assembler's macro language. Macro operators perform calculations on the computer at build time and are never executed on the PIC[®] microcontroller. Therefore, this operator may be used only with constants or other values that are known at build time. In this example, `CASE1`, `CASE2` and `CASE3` are all constants defined in our code and therefore known at build time. So, this line of code will XOR the value in the W register with the calculated value `CASE1^CASE2`. Mathematically, this can be written as:

$$W = W \oplus (\text{CASE1} \oplus \text{CASE2})$$

However, W contains the result of the previous operation $\text{SWITCH} \oplus \text{CASE1}$. Substituting for the previous value of W:

$$W = (\text{SWITCH} \oplus \text{CASE1}) \oplus (\text{CASE1} \oplus \text{CASE2})$$

At this point, we can take advantage of some of XOR's properties. First, we use the associative and commutative properties to rewrite the equation:

$$W = (\text{SWITCH} \oplus \text{CASE2}) \oplus (\text{CASE1} \oplus \text{CASE1})$$

Next, we use the property of non-idempotency ($A \oplus A = 0$):

$$W = (\text{SWITCH} \oplus \text{CASE2}) \oplus 0$$

And finally, we use the identity property ($A \oplus 0 = A$):

$$W = \text{SWITCH} \oplus \text{CASE2}$$

Which is exactly what we want to test to see if `SWITCH` = `CASE2`! Now this is just like what we did on line 2. From this point forward, the code just repeats itself with different values.

The code may be repeated for as many `CASE` n values you wish to use.