

Example, in C:

```
ne = ((x != y) ? 1 : 0);
```

Figure 3-15 shows how to calculate this expression using an ordinary control flow translation.

Figure 3-15. Predicate Calculation: Branching Code Sequence

```
cmpw    cr0,Rx,Ry    # place compare result in
                    cr0
li      R3,1         # R3 = 1
bne     cr0,lab     # x != y
li      R3,0         # R3 = 0

lab:
```

You can avoid the branch by using Condition Register logical instructions, as shown in Figure 3-16. In this case, the result of the comparison is directly transferred from the Condition Register to a General-Purpose Register, from which the bit is extracted and then flipped.

Figure 3-16. Predicate Calculation: Condition Register Logical Sequence

```
cmpw    cr0,Rx,Ry    # place compare result in
                    cr0
mfcrr   R4           # R4 = condition register
rlwinm  R5,R4,3,31,31 # extract the cr0[eq] bit
xori    R3,R5,1      # flip the bit to obtain
                    0/1
```

Some implementations have delays associated with accessing the Condition Register using the *mfcrr* instruction. An alternative that uses only fixed-point operations is shown in Figure 3-17.

Figure 3-17. Predicate Calculation: Fixed-Point Operation Code Sequence

```
subf    R0,Rx,Ry     # R0 = y - x
subf    R3,Ry,Rx     # R3 = x - y
or      R3,R3,R0     # R3 = R3 | R0
                    # sign bit holds desired
                    # result
rlwinm  R3,R3,1,31,31 # extract the sign bit
```

You can generate all boolean predicates with straight-line code that does not use the Condition Register. Figure 3-18 shows arithmetic expressions that yield a sign-bit reflecting the appropriate result.

Figure 3-18. Arithmetic Expressions for Boolean Predicates

<i>Boolean Predicate</i>	<i>Arithmetic Expression</i>
--------------------------	------------------------------