Build1144 ..1147

Adds an extensive handler for #ELSE within conditional statements, and, the additional of #IFNOT to support negated condition of #IF.

Examples:

```
#IF CHIPMHZ = 64
   _VAR1 = 64
#ELSE
   #IF CHIPMHZ = 32
      _VAR1 = 32
   #ELSE
      #IF CHIPMHZ = 24
         _VAR1 = 24
      #ELSE
         #IF CHIPMHZ = 16
            _VAR1 = 16
         #ELSE
            _VAR1 = 0
         #ENDIF
      #ENDIF
   #ENDIF
#ENDIF
```

```
DIM _BAR1 as Byte
#DEFINE VAR
#DEFINE BAR

#IFDEF PIC
   #IFDEF VAR
         _VAR1 = 1
         #IFDEF BAR
            _BAR1 = 11
         #ELSE
            _BAR1= 111
         #ENDIF
   #ELSE
         _VAR1 = 2
         #IFDEF BAR
            _BAR1 = 22
         #ELSE
            _BAR1= 222
         #ENDIF
   #ENDIF
#ELSE
   #IFDEF VAR
         _VAR1 = 2
   #ELSE
         _VAR1 = 3
   #ENDIF
#ENDIF
```

Note: This change Build1144 does not change the behaviour of #CHIP or #DEFINE this change is focused on the handling of conditional with the inclusion of #ELSE and #IFNOT

The change is implemented as follows, noting many of these changes are not visible to the user but this explains the change in some detail.

1. The compiler reads a source line from the source program and if the source line contains any of the conditional statement the compiler now caches this source line into an array. The array is required to handle nested conditional statements. The compiler uses the array as a cache (with the source type of condition
    a. #IF=2

   b. #IFNOT=3
   c. #IFDEF=0
   d. #IFNDEF=1
2. If the current source line is #ELSE then the compiler replaces the #ELSE with appropriate code to enable the compiler to handle the #ELSE, as follows:
   a. #ELSE when the condition is #IF – is transformed to #ENDIF & #IFNOT plus the condition
   b. #ELSE when the condition is #IFNOT – is transformed to #ENDIF & #IF plus the condition
   c. #ELSE when the condition is #IFDEF – is transformed to #ENDIF & #IFNDEF plus the condition
   d. #ELSE when the condition is #IFNDEF – is transformed to #ENDIF & #IFDEF plus the condition

3. The compiler increments the array location upon each conditional statement (#IF…) that is nested.
4. The compiler decrements the array location of each #ENDIF

An example

```
#IF CHIPMHZ = 64
   _VAR1 = 64
#ELSE
   _VAR = 0
#ENDIF
```

This is transformed within the compiler to

```
#IF CHIPMHZ = 64
   _VAR1 = 64
#ENDIF
#IFNOT CHIPMHZ = 64
   _VAR = 0
#ENDIF
```

Where the #ELSE is transformed into #ENDIF : #IFNOT condition

5. A key principle of operation is changing #ELSE to #ENDIF: IF... condition. This leaves the compiler to split the new line of code using the existing split line handler to the new negated condition.
6. The compiler has been updated to handle #IFNOT. This means the compiler has been updates in appropriate places.
7. A key principle of operation is the method to support for #IFNOT. #IFNOT is treated as #IF within the compilers IFDEFs handler and using the source type attribute to negate the operation

   #IFNOT is source type 3.

The #IFNOT command is transformed from #IFNOT to #IF, and, then the result of the calculation is negated - this creates #IFNOT. There are other trivial changes in IFDEFs method to recognised #IFNOT.

This is a large change to the compiler but low risk. The risk is minimal as users have to activate most of the new code by using #ELSE, and, the other main risk is the nested condition cache – as most code does not use nested conditions this is assessed as a low risk.

The only way to tell …. compare asm for an old compiler to new compiler.

Evan Venn August 2022