1.1.1.1 Reading Program Memory Using the IREAD Instruction

There is another method to build tables in program memory that are created with the **dw** directive, using the **iread** (immediate read) instruction. Different from the **retw** instruction that returns an 8-bit value, **iread** makes it possible to read all 12 bits stored in a program memory location.

Using **iread** is a bit "tricky" - let's demonstrate it with the following program:

```
; ==================================================================
; Programming the SX Microcontroller
; TUT039.SRC
; ==================================================================
LIST Q = 37
DEVICE  SX28L, TURBO, STACKX, OSCHS2
IRC_CAL IRC_FAST
FREQ   50_000_000

RESET   Main

org     $08
Ix      ds 1
Data    ds 2

Main
  mov   Ix, #Table

Loop
  mov   m, #Table >> 8

  mov   w, Ix
  iread
  mov   Data, w
  mov   Data+1, m
  inc   Ix
  test  Data
  sz
    jmp Loop
  test  Data+1
  sz
    jmp Loop
  jmp   Main

org     $400
Table
        dw    'PARALLAX'
        dw    12, 123, 1234, 0
```

When your debugger allows watching variables, configure a watch window that displays the contents of **Data** in character format as well as in 12-bit unsigned decimal format.

At memory page $400, we have defined the table. As you can see, the **dw** directive is used for initializing locations in the program memory to constant values. The **dw** directive accepts character strings, like 'PARALLAX'. In this case, for each character in the string, the lower eight bits of a memory location will be set to the ASCII code of that character (the upper four bits are cleared). The **dw** directive also accepts numerical constants like 12, 123, 1234, or 0. For each numerical constant, the assembler initializes one 12-bit memory location with the specified value with the upper bits cleared when necessary. The greatest number that can be stored in a memory location is $fff or 4,095 in decimal.

We use the **Ix** variable as table index. The instruction **mov Ix, #Table** copies the lower eight bits of the table address to **Ix**, i.e. Ix now "points" to the first table item.

As **Ix** is only eight bits wide, this is not enough to fully address all table items.

The expression `Table >> 8` is calculated at assembly-time, and its result are the upper four bits of the table address. This value is stored in the `m` register's lower four bits 3...0.

The contents of `Ix` are copied to `w` before executing the `iread` instruction.

The `iread` instruction expects the address to be read in `m:w`. This means that the upper four address bits are expected in the lower four bits (3...0) of `m` and the lower eight bits of the address are expected in `w`. In our example, this is the case because `m` and `w` were set accordingly before. The 12-bit contents of the addressed memory location is returned by iread in `m:w`. Similar to the format that was used to pass an address to `iread`, the result's upper four bits (11...8) are returned in the lower four bits of `m` (3...0) and the lower eight bits of the result (7...0) are returned in `w`.

In our program, the return value is stored to `Data` (lower eight bits) and `Data+1` (upper four bits). It then increments the table index `Ix`.

The program then tests if the value stored in `Data+1:Data` is $000. In this case, the program loops back to `Main` in order to re-initialize the table index `Ix`. Otherwise, the program stays in `Loop` by reading the next value from the table.

If you test the program in single-step mode or in "slow-motion", you can see, the values read from the table displayed in the watch window.

The size of a table read with iread is not limited to 256 items because the instruction uses direct addressing via `m:w`.