

---

## Section 4. Program Memory

---

### HIGHLIGHTS

This section of the manual contains the following topics:

4.1	Program Memory Address Map .....	4-2
4.2	Program Counter .....	4-4
4.3	Data Access from Program Memory .....	4-4
4.4	Program Space Visibility from Data Space .....	4-8
4.5	Program Memory Writes .....	4-10
4.6	PSV Code Examples .....	4-11
4.7	Related Application Notes .....	4-12
4.8	Revision History .....	4-13

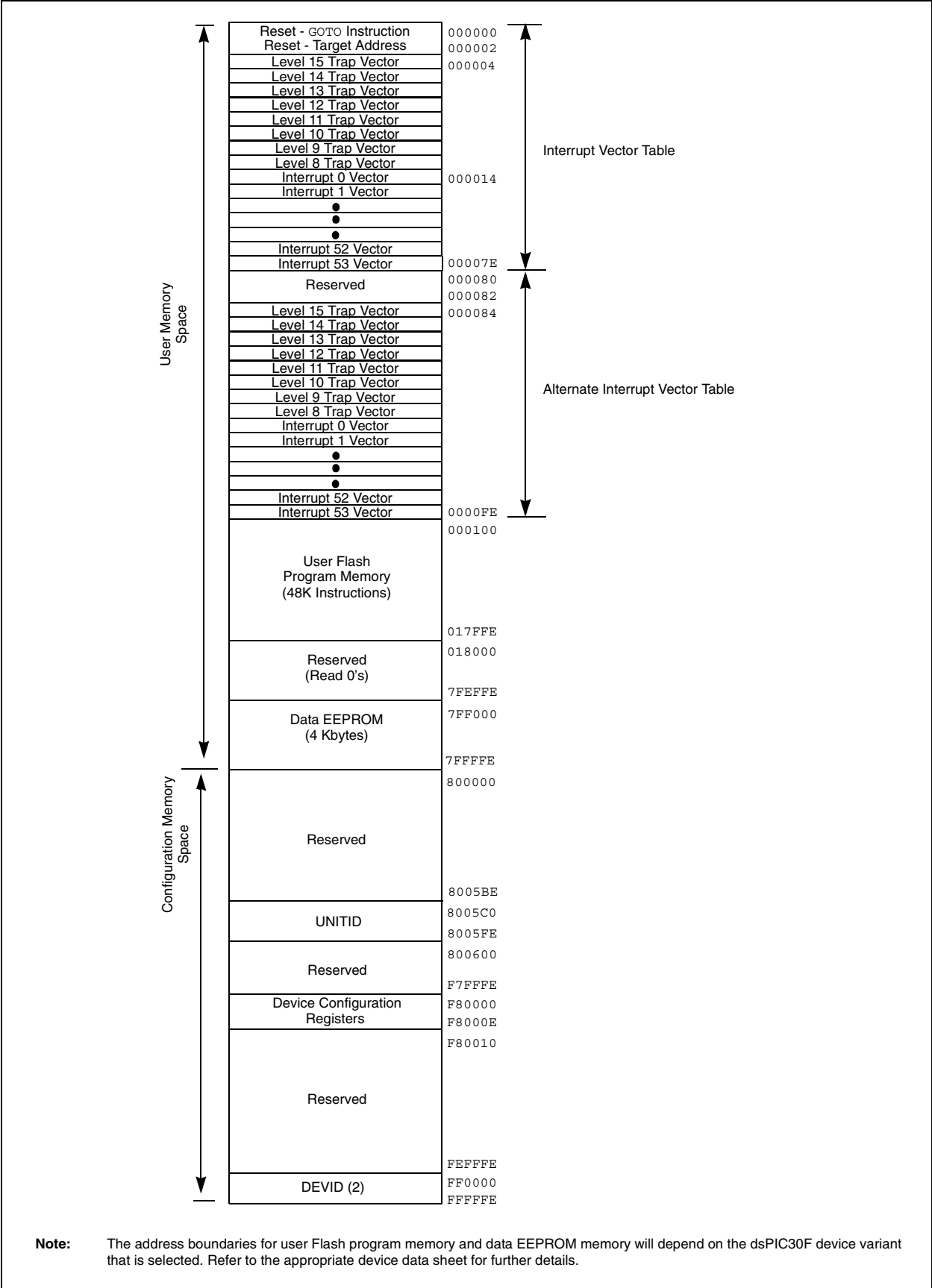
## 4.1 Program Memory Address Map

The dsPIC30F devices have a 4M x 24-bit program memory address space, shown in Figure 4-1. There are three available methods for accessing program space.

1. Via the 23-bit PC.
2. Via table read (TBLRD) and table write (TBLWT) instructions.
3. By mapping a 32-Kbyte segment of program memory into the data memory address space.

The program memory map is divided into the user program space and the user configuration space. The user program space contains the Reset vector, interrupt vector tables, program memory and data EEPROM memory. The user configuration space contains non-volatile configuration bits for setting device options and the device ID locations.

Figure 4-1: Example Program Space Memory Map



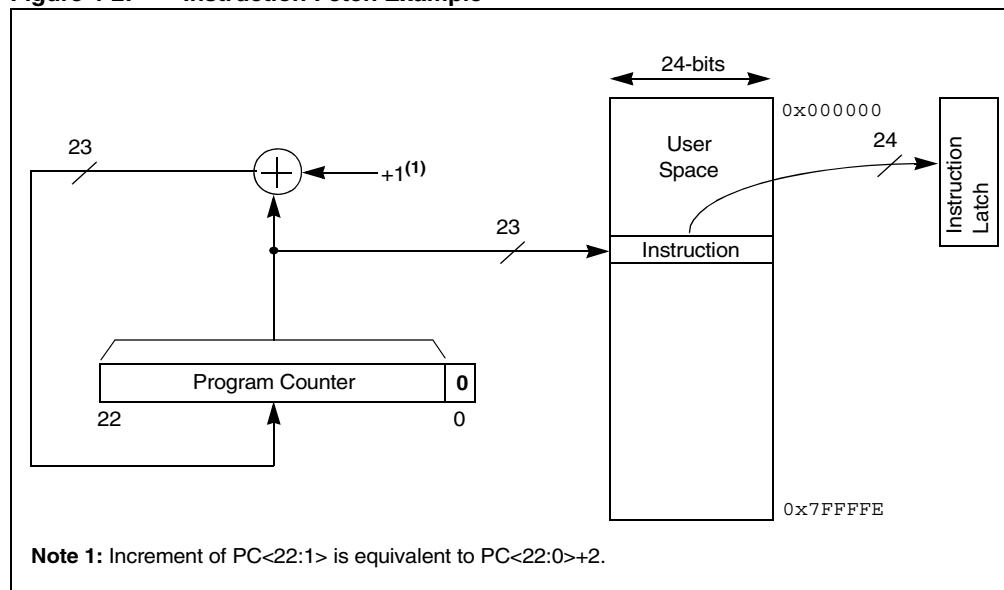
## 4.2 Program Counter

The PC increments by 2 with the LSb set to '0' to provide compatibility with data space addressing. Sequential instruction words are addressed in the 4M program memory space by PC<22:1>. Each instruction word is 24-bits wide.

The LSb of the program memory address (PC<0>) is reserved as a byte select bit for program memory accesses from data space that use Program Space Visibility or table instructions. For instruction fetches via the PC, the byte select bit is not required. Therefore, PC<0> is always set to '0'.

An instruction fetch example is shown in Figure 4-2. Note that incrementing PC<22:1> by one is equivalent to adding 2 to PC<22:0>.

**Figure 4-2: Instruction Fetch Example**



## 4.3 Data Access from Program Memory

There are two methods by which data can be transferred between the program memory and data memory spaces: via special table instructions, or through the remapping of a 32-Kbyte program space page into the upper half of data space. The TBLRD<sub>L</sub> and TBLWT<sub>L</sub> instructions offer a direct method of reading or writing the LSWord of any address within program space without going through data space, which is preferable for some applications. The TBLRD<sub>H</sub> and TBLWT<sub>H</sub> instructions are the only method whereby the upper 8-bits of a program word can be accessed as data.

## 4.3.1 Table Instruction Summary

A set of table instructions is provided to move byte or word-sized data between program space and data space. The table read instructions are used to read from the program memory space into data memory space. The table write instructions allow data memory to be written to the program memory space.

**Note:** Detailed code examples using table instructions can be found in **Section 5. “Flash and EEPROM Programming”**.

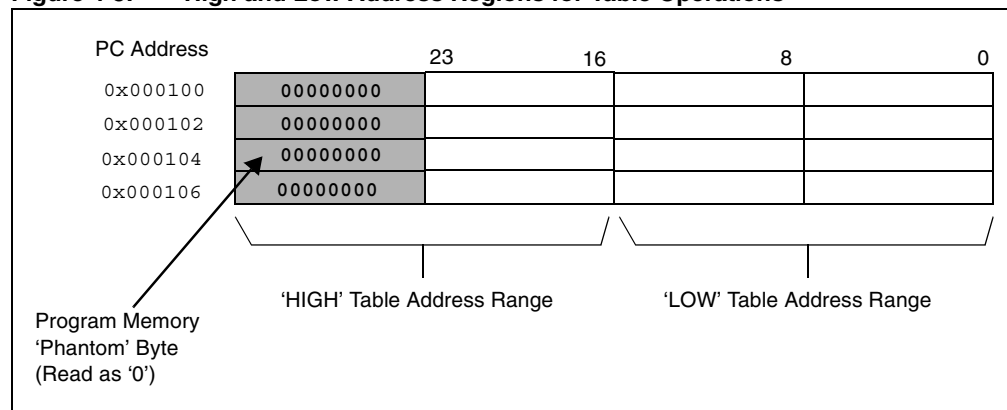
The four available table instructions are listed below:

- TBLRDL: Table Read Low
- TBLWTL: Table Write Low
- TBLRDH: Table Read High
- TBLWTH: Table Write High

For table instructions, program memory can be regarded as two 16-bit word wide address spaces residing side by side, each with the same address range as shown in Figure 4-3. This allows program space to be accessed as byte or aligned word addressable, 16-bit wide, 64-Kbyte pages (i.e., same as data space).

TBLRDL and TBLWTL access the LS Data Word of the program memory, and TBLRDH and TBLWTH access the upper word. As program memory is only 24-bits wide, the upper byte from this latter space does not exist, though it is addressable. It is, therefore, termed the ‘phantom’ byte.

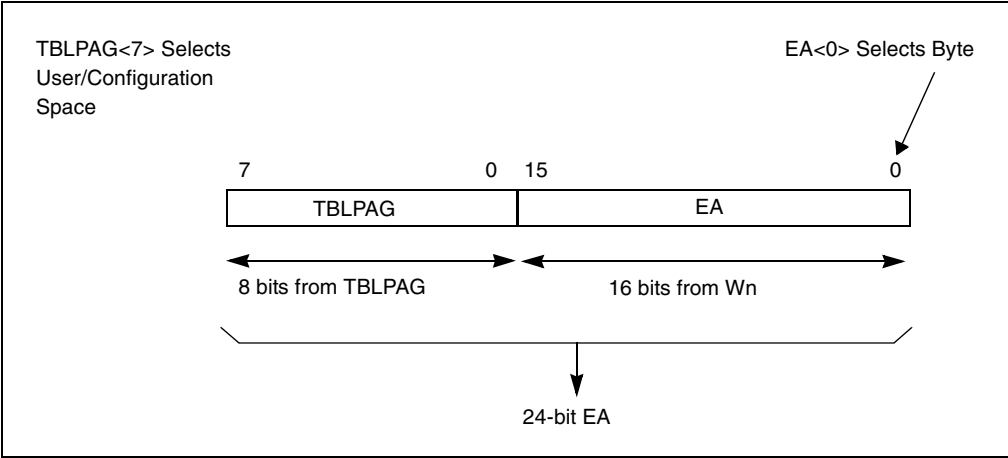
**Figure 4-3: High and Low Address Regions for Table Operations**



4.3.2 Table Address Generation

For all table instructions, a W register address value is concatenated with the 8-bit Data Table Page register, TBLPAG, to form a 23-bit effective program space address plus a byte select bit, as shown in Figure 4-4. As there are 15 bits of program space address provided from the W register, the data table page size in program memory is, therefore, 32K words.

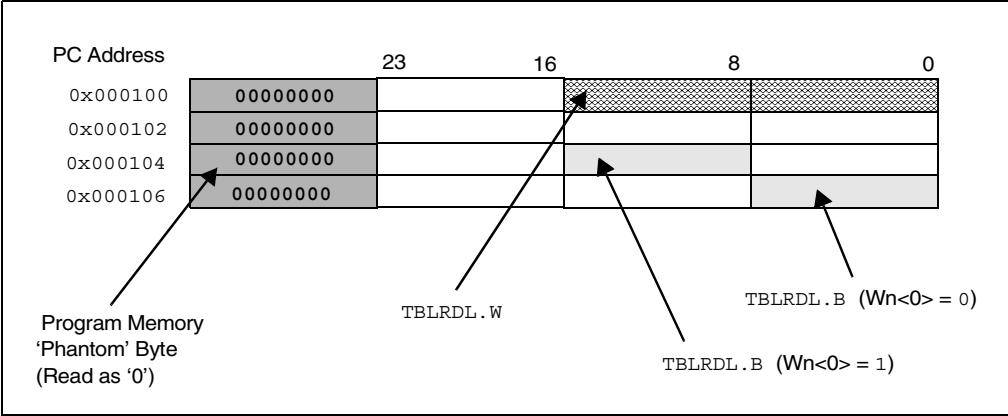
Figure 4-4: Address Generation for Table Operations



4.3.3 Program Memory Low Word Access

The TBLRDL and TBLWTL instructions are used to access the lower 16 bits of program memory data. The LSb of the W register address is ignored for word-wide table accesses. For byte-wide accesses, the LSb of the W register address determines which byte is read. Figure 4-5 demonstrates the program memory data regions accessed by the TBLRDL and TBLWTL instructions.

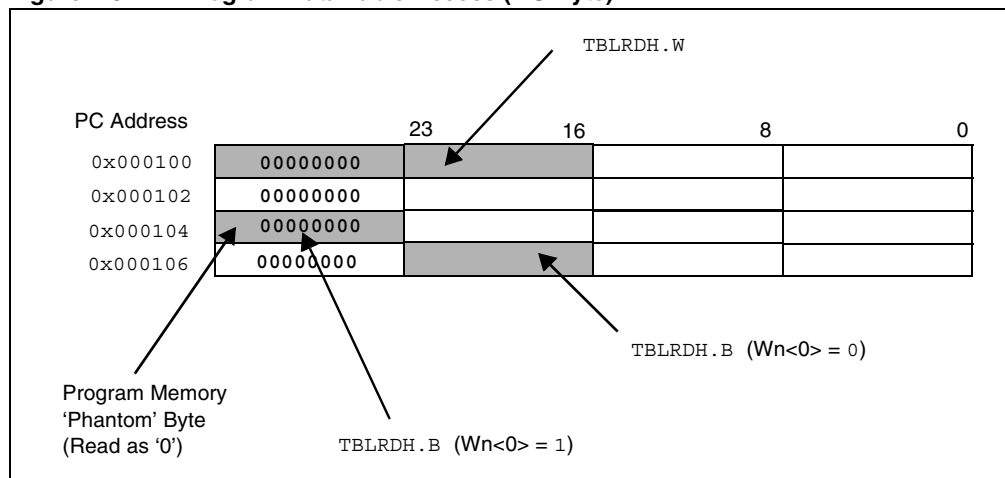
Figure 4-5: Program Data Table Access (LSWord)



## 4.3.4 Program Memory High Word Access

The `TBLRDH` and `TBLWTH` instructions are used to access the upper 8 bits of the program memory data. These instructions also support Word or Byte Access modes for orthogonality, but the high byte of the program memory data will always return '0', as shown in Figure 4-6.

**Figure 4-6: Program Data Table Access (MS Byte)**



## 4.3.5 Data Storage in Program Memory

It is assumed that for most applications, the high byte ( $P<23:16>$ ) will not be used for data, making the program memory appear 16-bits wide for data storage. It is recommended that the upper byte of program data be programmed either as a `NOP`, or as an illegal opcode value, to protect the device from accidental execution of stored data. The `TBLRDH` and `TBLWTH` instructions are primarily provided for array program/verification purposes and for those applications that require compressed data storage.

## 4.4 Program Space Visibility from Data Space

The upper 32 Kbytes of the dsPIC30F data memory address space may optionally be mapped into any 16K word program space page. This mode of operation is called Program Space Visibility (PSV) and provides transparent access of stored constant data from X data space without the need to use special instructions (i.e., `TBLRD`, `TBLWT` instructions).

### 4.4.1 PSV Configuration

Program Space Visibility is enabled by setting the PSV bit (`CORCON<2>`). A description of the `CORCON` register can be found in **Section 2. “CPU”**.

When PSV is enabled, each data space address in the upper half of the data memory map will map directly into a program address (see Figure 4-7). The PSV window allows access to the lower 16 bits of the 24-bit program word. The upper 8 bits of the program memory data should be programmed to force an illegal instruction, or a `NOB`, to maintain machine robustness. Note that table instructions provide the only method of reading the upper 8 bits of each program memory word.

Figure 4-8 shows how the PSV address is generated. The 15 LSbs of the PSV address are provided by the `W` register that contains the effective address. The MSb of the `W` register is not used to form the address. Instead, the MSb specifies whether to perform a PSV access from program space or a normal access from data memory space. If a `W` register effective address of `0x8000` or greater is used, the data access will occur from program memory space when PSV is enabled. All accesses will occur from data memory when the `W` register effective address is less than `0x8000`.

The remaining address bits are provided by the `PSVPAG` register (`PSVPAG<7:0>`), as shown in Figure 4-8. The `PSVPAG` bits are concatenated with the 15 LSbs of the `W` register, holding the effective address to form a 23-bit program memory address. PSV can only be used to access values in program memory space. Table instructions must be used to access values in the user configuration space.

The LSb of the `W` register value is used as a byte select bit, which allows instructions using PSV to operate in Byte or Word mode.

### 4.4.2 PSV Mapping with X and Y Data Spaces

The `Y` data space is located outside of the upper half of data space for most dsPIC30F variants, such that the PSV area will map into `X` data space. The `X` and `Y` mapping will have an effect on how PSV is used in algorithms.

As an example, the PSV mapping can be used to store coefficient data for Finite Impulse Response (FIR) filter algorithms. The FIR filter multiplies each value of a data buffer containing historical filter input data with elements of a data buffer that contains constant filter coefficients. The FIR algorithm is executed using the `MAC` instruction within a `REPEAT` loop. Each iteration of the `MAC` instruction pre-fetches one historical input value and one coefficient value to be multiplied in the next iteration. One of the pre-fetched values must be located in `X` data memory space and the other must be located in `Y` data memory space.

To satisfy the PSV mapping requirements for the FIR filter algorithm, the user must locate the historical input data in the `Y` memory space and the filter coefficients in `X` memory space.



4  
Program  
Memory

The diagram illustrates the Program Space Visibility mechanism. It shows two memory spaces: Data Space and Program Space. The Data Space ranges from 0x0000 to 0xFFFF. The Program Space ranges from 0x000100 to 0x017FFF. A PSVPAG register is shown with a value of 0x01. A read operation is indicated by 'Data Read'.

Upper 8 bits of Program Memory Data cannot be read using Program Space Visibility.

Diagram illustrating the PSVPAG Reg structure:

- The register is 23 bits wide, divided into an 8-bit PSVPAG Reg field and a 15-bit Wn field.
- The Wn field contains a '1' in its most significant bit, which is labeled 'Select'.
- The Wn field is also labeled 'Wn<0> is Byte Select'.
- The entire register is labeled '23-bit EA'.

## 4.4.3 PSV Timing

Instructions that use PSV will require two extra instruction cycles to complete execution, except the following instructions that require only one extra cycle to complete execution:

- The MAC class of instructions with data pre-fetch operands
- All MOV instructions including the MOV.D instruction

The additional instruction cycles are used to fetch the PSV data on the program memory bus.

### 4.4.3.1 Using PSV in a Repeat Loop

Instructions that use PSV within a REPEAT loop eliminate the extra instruction cycle(s) required for the data access from program memory, hence incurring no overhead in execution time. However, the following iterations of the REPEAT loop will incur an overhead of two instruction cycles to complete execution:

- The first iteration
- The last iteration
- Instruction execution prior to exiting the loop due to an interrupt
- Instruction execution upon re-entering the loop after an interrupt is serviced

### 4.4.3.2 PSV and Instruction Stalls

Refer to **Section 2. “CPU”** for more information about instruction stalls using PSV.

## 4.5 Program Memory Writes

The dsPIC30F family of devices contains internal program Flash memory for executing user code. There are two methods by which the user can program this memory:

1. Run-Time Self Programming (RTSP)
2. In-Circuit Serial Programming™ (ICSP™)

RTSP is accomplished using TBLWT instructions. ICSP is accomplished using the SPI interface and integral bootloader software. Refer to **Section 5. “Flash and EEPROM Programming”** for further details about RTSP. ICSP specifications can be downloaded from the Microchip Technology web site ([www.microchip.com](http://www.microchip.com)).

## 4.6 PSV Code Examples

### 4.6.1 PSV Code Example in C:

```
// PSV code example in C
// When defined as below the const string uses the PSV feature of dsPIC

const unsigned char hello[] = {"Hello World:\r\n"};
unsigned char *TXPtr;          // Transmit pointer

int main(void)
{
    // Initialize the UART1
    U1MODE = 0x8000;
    U1STA = 0x0000;
    U1BRG = ((FCY/16)/BAUD) - 1;    // set baud rate = BAUD
    TXPtr = &hello[0];              // point to first char in string
    U1STAbits.UTXEN = 1;            // Initiate transmission
    while (1)
    {
        while (*TXPtr)              // while valid char in string ...
            if (!U1STAbits.UTXBF)    // and buffer not full ...
                U1TXREG = *TXPtr++;  // transmit string via UART

        DelayNmSec(500);             // delay for 500 mS
        TXPtr = &hello[0];          // re-initialize pointer to first char
    }
}                                     // end main
```

### 4.6.2 PSV code Example in Assembly:

```
.equ CORCONL, CORCON
.section .const, "r"
hello:
    .ascii "Hello World:\n\r\0"

.global __reset ;Declare the label for the start of code

.text                ;Start of Code section

__reset:

    clr    U1STA
    mov    #0x8000,W0    ; enable UART module
    mov    W0,U1MODE
    mov    #BR,W0        ; set baudrate using formula value
    mov    W0, U1BRG      ; /
    bset   U1STA,#UTXEN   ; initiate transmission

Again:
    rcall  Delay500mSec    ; delay for 500 mS
    mov    #psvpage(hello),w0
    mov    w0, PSVPAG
    bset.b CORCONL,#PSV
    mov    #psvoffset(hello),w0

TxSend:
    mov.b  [w0++], w1      ; get char in string
    cp     w1,#0           ; if Null
    bra    Z,Again        ; then re-initialize

BufferTest:
    btsc   U1STA,#UTXBF    ; see if buffer full
    bra    BufferTest      ; wait till empty
    mov    w1,U1TXREG      ; load value in TX buffer
    bra    TxSend          ; repeat for next char.
```

## 4.7      Related Application Notes

This section lists application notes that are related to this section of the manual. These application notes may not be written specifically for the dsPIC30F Product Family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the Program Memory module are:

Title	Application Note #
No related application notes at this time.	

**Note:** Please visit the Microchip web site ([www.microchip.com](http://www.microchip.com)) for additional Application Notes and code examples for the dsPIC30F Family of devices.

### 4.8 Revision History

#### Revision A

This is the initial released revision of this document.

#### Revision B

There were no technical content or editorial revisions to this section of the manual, however, this section was updated to reflect Revision B throughout the manual.

#### Revision C

This revision incorporates all known errata at the time of this document update.

#### Revision D

Section 4.6 “PSV Code Examples”, has been added.

NOTES: