Section 44. CPU with Extended Data Space (EDS)

HIGHLIGHTS

This section of the manual contains the following topics:

44.1 INTRODUCTION

The PIC24F CPU with Extended Data Space (EDS) has a 16-bit (data) modified Harvard architecture with an enhanced instruction set. The CPU has a 24-bit instruction word with a variable length opcode field. The Program Counter (PC) is 23 bits wide and addresses up to 4M x 24 bits of user program memory space.

An instruction prefetch mechanism helps to maintain throughput and provides predictable execution. Most instructions execute in a single-cycle effective execution rate, with the exception of instructions that change the program flow, the double-word move (`MOV.D`) instruction, accessing Extended Data Space (EDS) and Program Space Visibility (PSV), and the table instructions. Overhead-free program loop constructs are supported using the `REPEAT` instruction, which is interruptible at any point.

44.1.1 Registers

The Section 44. CPU with Extended Data Space (EDS) with EDS devices have sixteen 16-bit working registers in the programmer's model. Each of the working registers can act as a data, address or address offset register. The 16th working register (W15) operates as a software Stack Pointer (SP) for interrupts and calls.

44.1.2 Data Space Addressing

The base data space can be addressed as 32K words or 64 Kbytes. The upper 32 Kbytes of the data space are referred to as a 32-Kbyte EDS window. The EDS window is used to access the internal extended memory and external data memory, which includes 16 Mbytes of data address and the entire program memory, which is referred to as PSV. The 16 Mbytes of data address includes the on-chip implemented, extended RAM and external RAM accessed through the Enhanced Parallel Master Port (EPMP). The EDS window is used in conjunction with a Read/Write Page register (DSRPAG or DSWPAG) to form an EDS address. The EDS can be addressed as 8M words or 16 Mbytes. The EDS window is used in conjunction with the DSRPAG register to form the address for PSV access. Refer to "*PIC24F Family Reference Manual*" **Section 45. "Data Memory with Extended Data Space (EDS)"** for more details on EDS and PSV accesses.

44.1.3 Addressing Modes

The CPU supports these addressing modes:

- Inherent (no operand)
- Relative
- Literal
- Memory Direct
- Register Direct
- Register Indirect

Each instruction is associated with a predefined addressing mode group, depending upon its functional requirements. As many as six addressing modes are supported for each instruction.

For most instructions, the PIC24F with EDS can execute these functions in a single instruction cycle:

- Data memory read
- Working register (data) read
- Data memory write

As a result, three operand instructions can be supported, allowing A + B = C operations to be executed in a single cycle. However, EDS and PSV read/write may take more than one instruction cycle. Refer to **Section 44.11.2.4 "Instruction Stalls During EDS and PSV Access"** for details.

44.1.4 Arithmetic and Logic Unit

A high-speed, 17-bit x 17-bit multiplier is included to significantly enhance the core arithmetic capability and throughput. The multiplier supports Signed, Unsigned and Mixed mode, 16-bit x 16-bit or 8-bit x 8-bit integer multiplication. All multiply instructions execute in a single cycle.

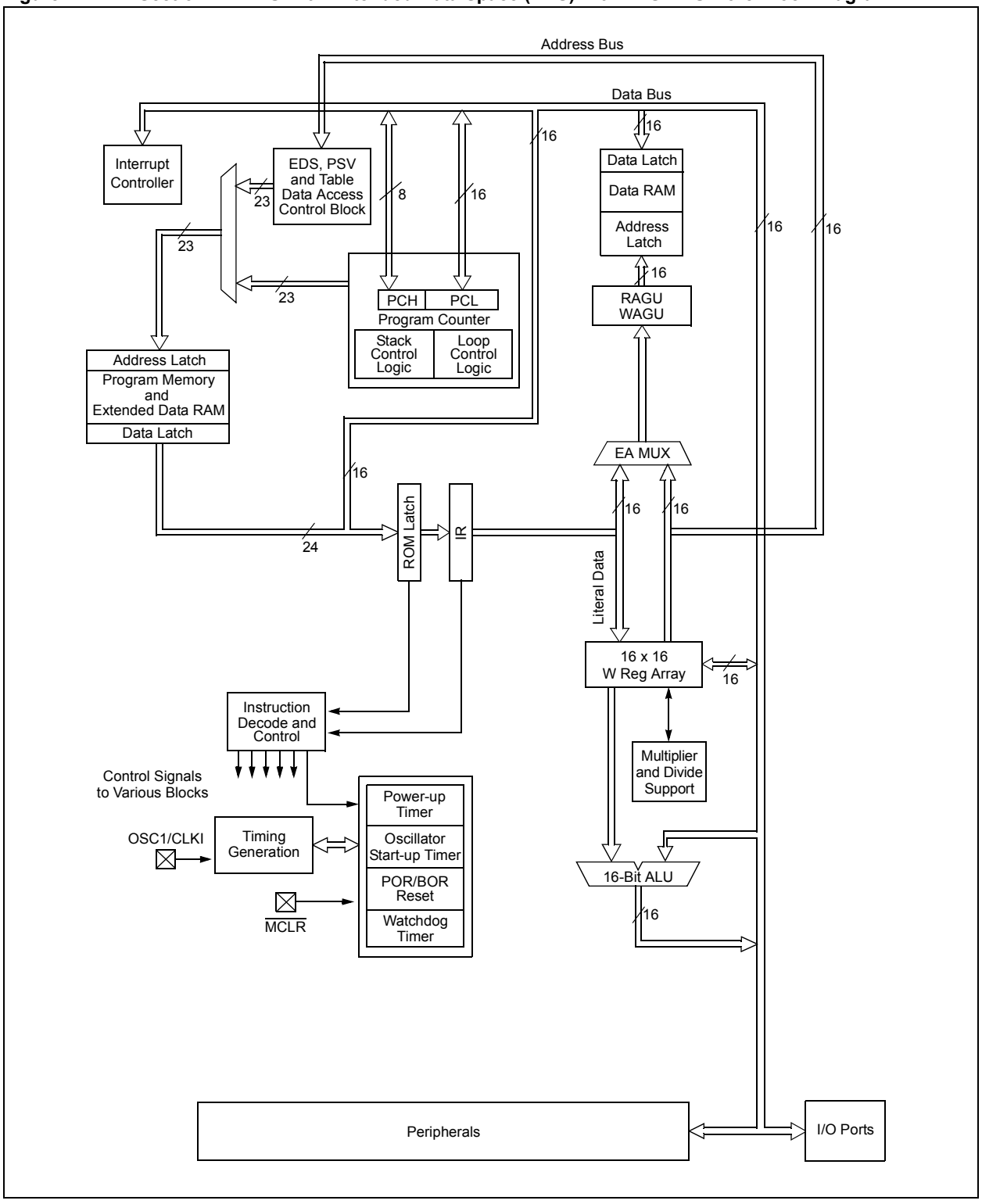
The 16-bit Arithmetic Logic Unit (ALU) is enhanced with integer divide assist hardware that supports an iterative non-restoring divide algorithm. It operates in conjunction with the REPEAT instruction looping mechanism, and a selection of iterative divide instructions, to support 32-bit (or 16-bit) divided x 16-bit integer signed and unsigned division. All divide operations require 19 cycles to complete, but are interruptible at any cycle boundary.

44.1.5 Exception Processing

The PIC24F with EDS has a vectored exception scheme with up to eight possible sources of non-maskable trap interrupt sources. Each interrupt source can be assigned to one of seven priority levels. Refer to the "*PIC24F Family Reference Manual*" **Section 8. "Interrupts"** for more details.

Figure 44-1 shows a block diagram of the CPU.

Figure 44-1: Section 44. CPU with Extended Data Space (EDS) with EDS CPU Core Block Diagram



44.2 PROGRAMMER'S MODEL

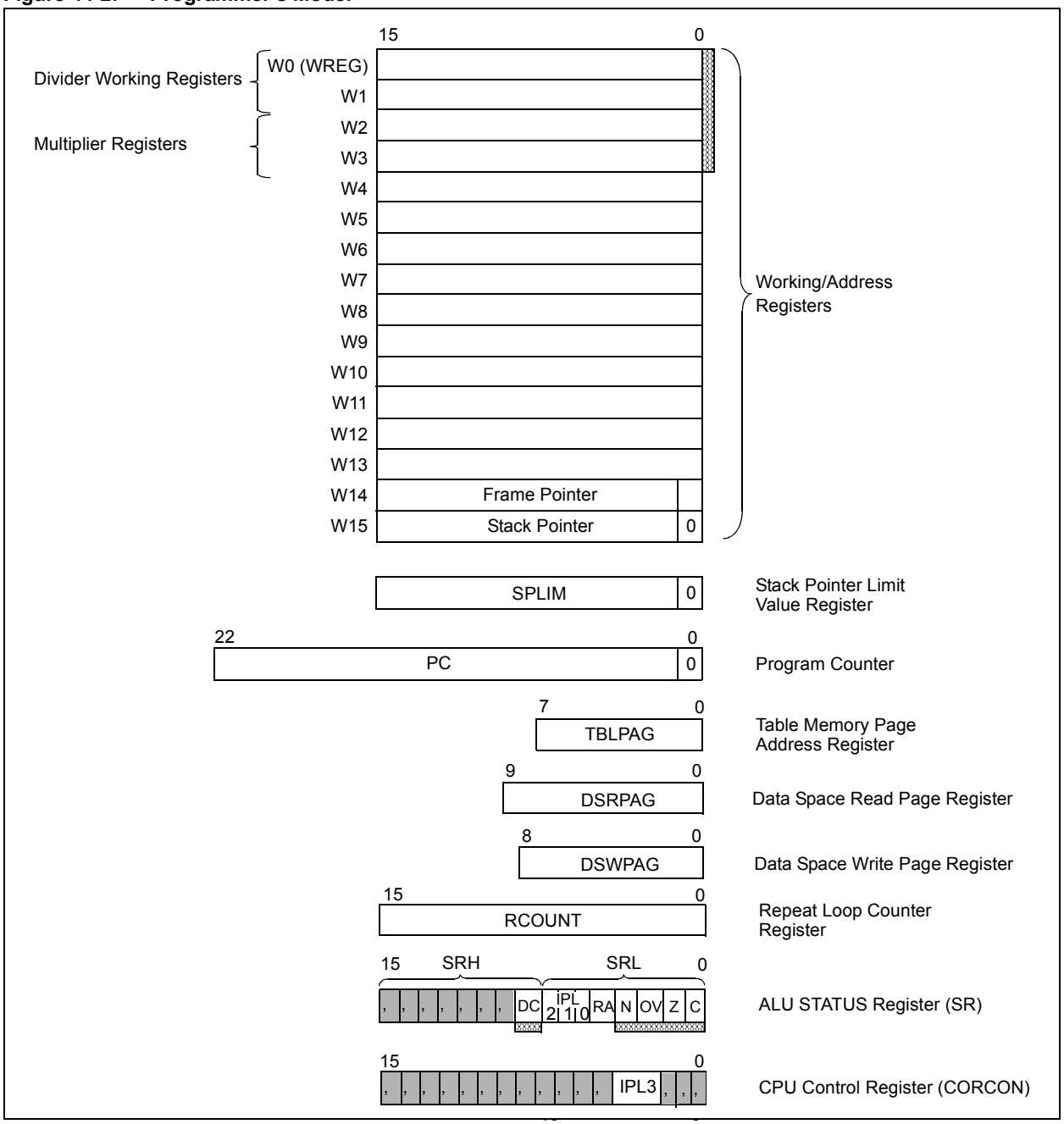
The programmer's model for the PIC24F with EDS is shown in Figure 44-2. All registers in the programmer's model are memory mapped and can be manipulated directly by instructions. Table 44-1 provides a description of each register.

Table 44-1: Programmer's Model Register Descriptions

Register(s) Name	Description
W0 through W15	Working register array
PC	23-bit Program Counter
SR	ALU STATUS Register
SPLIM	Stack Pointer Limit Value register
TBLPAG	Table Memory Page Address register
DSRPAG	Extended Data Space (EDS) Read Page register
DSWPAG	Extended Data Space (EDS) Write Page register
RCOUNT	REPEAT Loop Count register
CORCON	CPU Control register
DISICNT	Disable Interrupt Count register

All registers associated with the programmer's model are memory mapped, as shown in Table 44-5.

Figure 44-2: Programmer's Model



44.2.1 Working Register Array

The 16 working (W) registers can function as data, address or address offset registers. The function of a W register is determined by the addressing mode of the instruction that accesses it.

The PIC24F with the EDS instruction set can be divided into two instruction types: register and file register instructions.

44.2.1.1 REGISTER INSTRUCTIONS

Register instructions can use each W register as a data value or an address offset value. For example:

```
MOV     W0,W1              ; move contents of W0 to W1
MOV     W0,[W1]            ; move W0 to address contained in W1
ADD     W0,[W4],W5         ; add contents of W0 to contents pointed
                           ; to by W4. Place result in W5
```

44.2.1.2 FILE REGISTER INSTRUCTIONS

File register instructions operate on a specific memory address contained in the instruction opcode and register, W0. W0 is a special working register used in file register instructions. Working registers, W1-W15, cannot be specified as target registers in file register instructions.

The file register instructions provide backward compatibility with existing PIC® MCU devices, which have only one W register. The label, 'WREG', is used in the assembler syntax to denote W0 in a file register instruction. For example:

```
MOV     WREG,0x0100        ; move contents of W0 to address 0x0100
ADD     0x0100,WREG        ; add W0 to address 0x0100, store in W0
```

Note: For a complete description of addressing modes and instruction syntax, refer to the "*16-Bit MCU and DSC Programmer's Reference Manual*".

44.2.1.3 W REGISTER MEMORY MAPPING

Since the W registers are memory mapped, it is possible to access a W register in a file register instruction as follows:

```
MOV     0x0004, W10    ; equivalent to MOV W2, W10
```

where 0x0004 is the address in memory of W2.

Further, it is also possible to execute an instruction that uses a W register both as an address pointer and operand destination. For example:

```
MOV     W1,[W2++]
```

where:

```
    W1 = 0x1234
    W2 = 0x0004            ;[W2] addresses W2
```

In the above example, the contents of W2 are 0x0004. As W2 is used as an address pointer, it points to location, 0x0004, in memory. W2 is also mapped to this address in memory. Even though this is an unlikely event, it is impossible to detect until run time. The PIC24F with EDS ensures that the data write dominates, resulting in W2 = 0x1234 in the above example.

44.2.1.4 W REGISTERS AND BYTE MODE INSTRUCTIONS

Byte instructions that target the W register array affect only the Least Significant Byte (LSB) of the target register. As the working registers are memory mapped, the LSB and the Most Significant Byte (MSB) can be manipulated through byte-wide data memory space accesses.

44.2.2 Shadow Registers

Many of the registers in the programmer's model have an associated shadow register, as shown in Figure 44-2. None of the shadow registers are directly accessible.

The PUSH.S and POP.S instructions are useful for fast context save/restore during a function call or Interrupt Service Routine (ISR). The PUSH.S instruction transfers the following register values into their respective shadow registers:

- W0...W3
- SR (N, OV, Z, C, DC bits only)

The POP.S instruction restores the values from the shadow registers into these register locations. Following is a code example using the PUSH.S and POP.S instructions:

```
MyFunction:
    PUSH.S                    ; Save W registers, MCU status
    MOV     #0x03,W0          ; load a literal value into W0
    ADD     RAM100            ; add W0 to contents of RAM100
    BTSC    SR,#Z             ; is the result 0?
    BSET    Flags,#IsZero     ; Yes, set a flag
    POP.S                     ; Restore W regs, MCU status
    RETURN
```

The PUSH.S instruction overwrites the contents previously saved in the shadow registers. The shadow registers are only one-level deep, so care must be taken if the shadow registers are to be used for multiple software tasks.

The user application must ensure that any task using the shadow registers is not interrupted by a higher priority task that also uses the shadow registers. If the higher priority task is allowed to interrupt the lower priority task, the contents of the shadow registers saved in the lower priority task would be overwritten by the higher priority task.

44.2.3 Uninitialized W Register Reset

The W register array (with the exception of W15) is cleared during all Resets and is considered uninitialized until written to. An attempt to use an uninitialized register as an address pointer will reset the device.

A word write must be performed to initialize a W register. A byte write will not affect the initialization detection logic.

44.3 SOFTWARE STACK POINTER

The W15 register serves as a dedicated software Stack Pointer and is automatically modified by exception processing, subroutine calls and returns; however, W15 can be referenced by any instruction in the same manner as all other W registers. This simplifies reading, writing and manipulating of the SP (for example, creating stack frames).

Note:	To protect against misaligned stack accesses, W15<0> is fixed to '0' by the hardware.

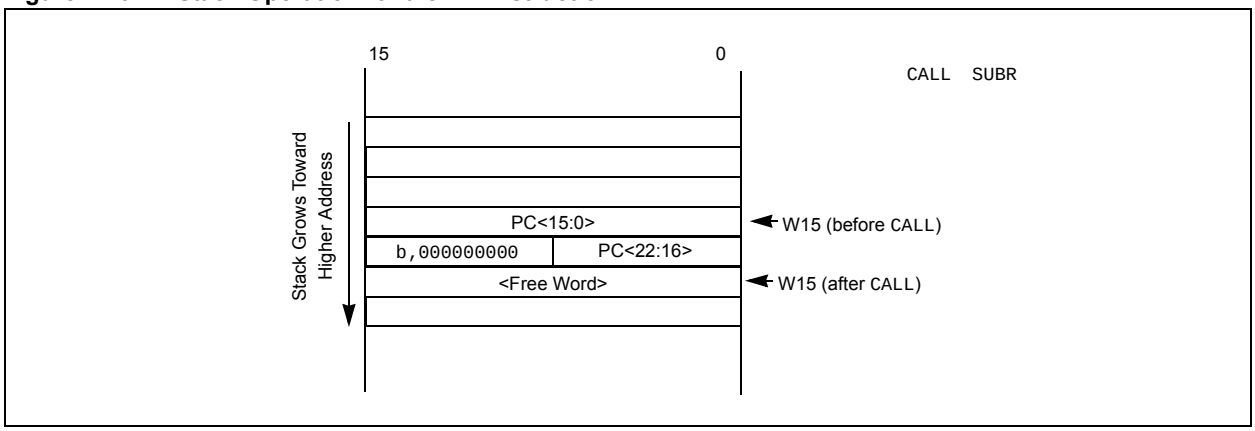
W15 is initialized to 0x0800 during all Resets. This address ensures that the SP points to valid RAM in all PIC24F with EDS devices and permits stack availability for non-maskable trap exceptions. These can occur before the SP is initialized by the user software. You can reprogram the SP during initialization to any location within the data space.

The SP always points to the first available free word and fills the software stack working from lower toward higher addresses. Figure 44-3 shows how it pre-decrements for a stack pop (read) and post-increments for a stack push (writes).

When the Program Counter (PC) is pushed onto the stack, PC<15:0> are pushed onto the first available stack word, then PC<22:16> are pushed onto the second available stack location. For a PC push during any CALL instruction, the MSB of the PC is zero-extended before the push, as shown in Figure 44-3. During exception processing, the MSB of the PC is concatenated with the lower 8 bits of the CPU STATUS Register (SR). This allows the contents of SRL to be preserved automatically during interrupt processing.

Note:	The SP, W15, is never subject to paging; therefore, stack addresses are restricted to the base data space (0x0000-0xFFFF).

Figure 44-3: Stack Operation for a CALL Instruction



44.3.1 Software Stack Examples

The software stack is manipulated using the PUSH and POP instructions. The PUSH and POP instructions are the equivalent of a MOV instruction, with W15 used as the destination pointer. For example, the contents of W0 can be pushed onto the stack by:

```
PUSH W0
```

This syntax is equivalent to:

```
MOV W0,[W15++]
```

The contents of the Top-Of-Stack (TOS) can be returned to W0 by:

```
POP W0
```

This syntax is equivalent to:

```
MOV [--W15],W0
```

Figure 44-4 through Figure 44-7 show examples of how the software stack is used. Figure 44-4 shows the software stack at device initialization. W15 has been initialized to 0x0800. This example assumes the values, 0x5A5A and 0x3636, have been written to W0 and W1, respectively. The stack is pushed for the first time in Figure 44-5 and the value contained in W0 is copied to the stack. W15 is automatically updated to point to the next available stack location (0x0802). In Figure 44-6, the contents of W1 are pushed onto the stack. Figure 44-7 shows how the stack is popped and the Top-Of-Stack value (previously pushed from W1) is written to W3.

Figure 44-4: Stack Pointer at Device Reset

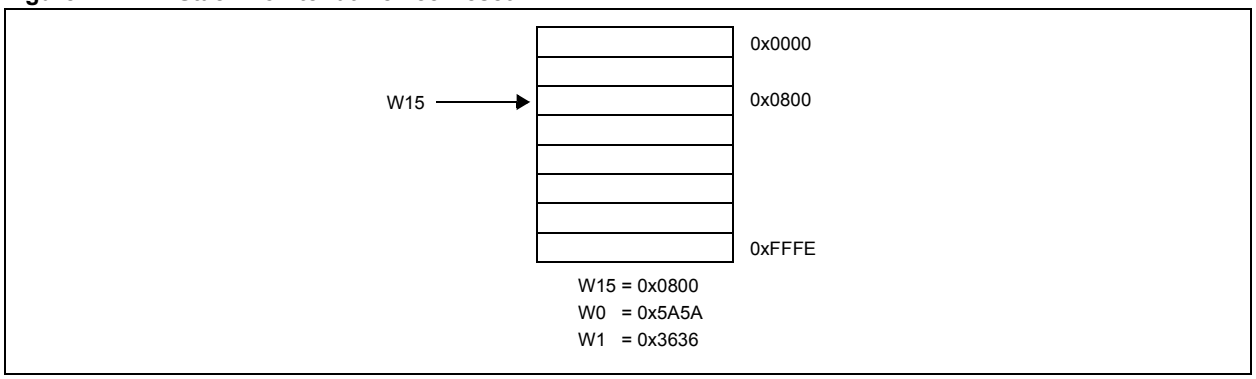


Figure 44-5: Stack Pointer After the First PUSH Instruction

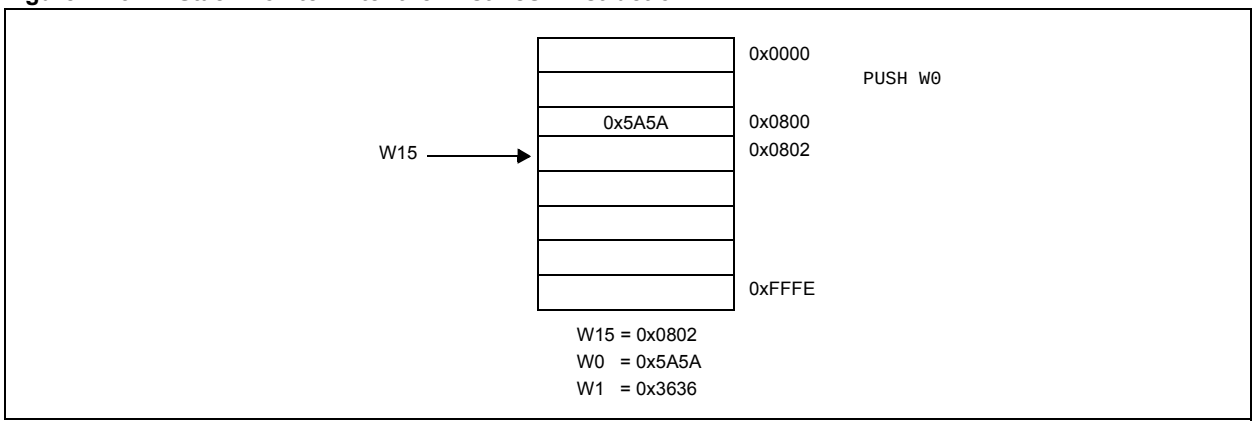


Figure 44-6: Stack Pointer After the Second PUSH Instruction

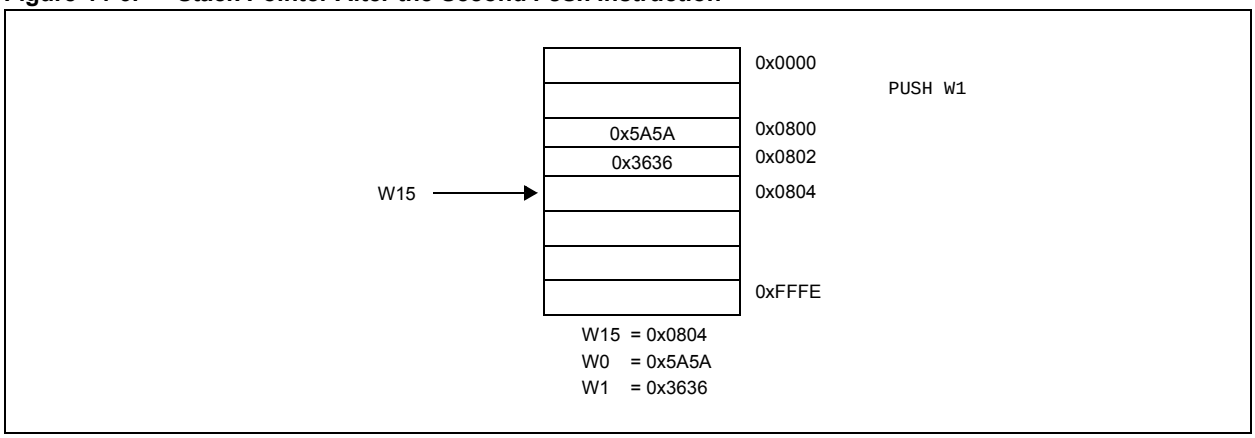
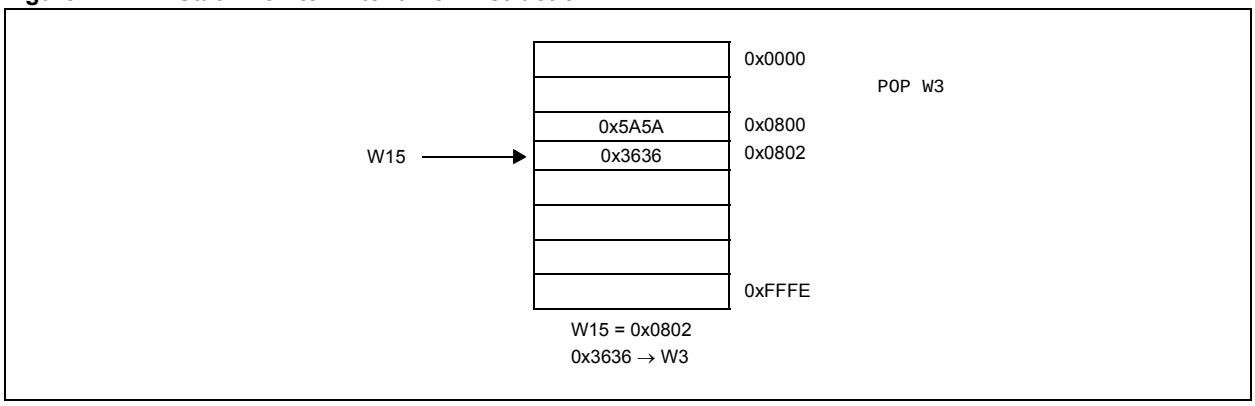


Figure 44-7: Stack Pointer After a POP Instruction



44.3.2 W14 Software Stack Frame Pointer

A frame is a user-defined section of memory in the stack that is used by a single subroutine. Working register, W14, can be used as a Stack Frame Pointer with the LNK (link) and ULNK (unlink) instructions. W14 can be used in a normal working register by instructions when it is not used as a Frame Pointer.

For software examples that use W14 as a Stack Frame Pointer, refer to the "*16-Bit MCU and DSC Programmer's Reference Manual*".

44.3.3 Stack Pointer Overflow

The Stack Pointer Limit register (SPLIM) specifies the size of the stack buffer. SPLIM is a 16-bit register, but SPLIM<0> is fixed to '0' as all stack operations must be word-aligned.

The stack overflow check is not enabled until a word write to SPLIM occurs. After this, it can only be disabled by a device Reset. All Effective Addresses (EA), generated using W15 as a source or destination, are compared against the value in SPLIM. If the contents of the Stack Pointer (W15) exceed the contents of the SPLIM register by two, and a push operation is performed, a stack error trap occurs on a subsequent push operation. Thus, for example, if it is desirable to cause a stack error trap when the stack grows beyond address, 0x2000 in RAM, initialize the SPLIM with the value, 0x1FFE.

Note:	A stack error trap can be caused by any instruction that uses the contents of the W15 register to generate an Effective Address (EA). Thus, if the contents of W15 exceed the contents of the SPLIM register by two, and a CALL instruction is executed or an interrupt occurs, a stack error trap is generated.

If stack overflow checking is enabled, a stack error trap also occurs if the W15 Effective Address calculation wraps over the end of data space (0xFFFF).

Note:	A write to the SPLIM register should not be followed by an indirect read operation using W15.

For more information on the stack error trap, refer to "*PIC24F Family Reference Manual*" **Section 8. "Interrupts"**.

44.3.4 Stack Pointer Underflow

The stack is initialized to 0x0800 during a Reset. A stack error trap is initiated if the Stack Pointer address is less than 0x0800.

Note:	Locations in data space, between 0x0000 and 0x07FF, are in general, reserved for core and peripheral Special Function Registers (SFRs).

44.4 CPU REGISTER DESCRIPTIONS

44.4.1 SR: CPU STATUS Register

The PIC24F with EDS has a 16-bit STATUS Register (SR). A detailed description of the CPU STATUS Register is shown in Register 44-1. The LSB of this register is referred to as the STATUS Register, Lower Byte (SRL). The MSB is referred to as the STATUS Register, Higher Byte (SRH).

The SRL contains:

- All MCU ALU Operation Status flags
- The CPU Interrupt Priority Status bits, IPL<2:0>
- The REPEAT Loop Active Status bit, RA (SR<4>)

During exception processing, the SRL is concatenated with the MSB of the PC to form a complete word value, which is then stacked.

The SRH contains the Digit Carry bit, DC (SR<8>).

The SR bits are readable/writable with the following exception: the RA bit (SR<4>) is read-only.

Note:	A description of the STATUS Register bits affected by each instruction is provided in the *"16-Bit MCU and DSC Programmer's Reference Manual"*.

44.4.2 CORCON: Core Control Register

The CORCON register contains the IPL3 status bit, which is concatenated with IPL<2:0> (SR<7:5>) to form the CPU Interrupt Priority Level (IPL).

Register 44-1: SR: CPU STATUS Register

U-0	U-0	U-0	U-0	U-0	U-0	U-0	R/W-0,HSC
—	—	—	—	—	—	—	DC
bit 15							bit 8

R/W-0,HSC(1,2)	R/W-0,HSC(1,2)	R/W-0,HSC(1,2)	R-0,HSC	R/W-0,HSC	R/W-0,HSC	R/W-0,HSC	R/W-0,HSC
IPL2	IPL1	IPL0	RA	N	OV	Z	C
bit 7							bit 0

Legend:	HSC = Hardware Settable/Clearable bit		
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 15-9 **Unimplemented:** Read as '0'

bit 8 **DC:** MCU ALU Half Carry/$\overline{\text{Borrow}}$ bit
- 1 = A carry-out from the 4th low-order bit (for byte-sized data) or 8th low-order bit (for word-sized data) of the result occurred
- 0 = No carry-out from the 4th low-order bit (for byte-sized data) or 8th low-order bit (for word-sized data) of the result occurred

bit 7-5 **IPL<2:0>:** CPU Interrupt Priority Level Status bits(1,2)
- 111 = CPU Interrupt Priority Level is 7 (15); user interrupts disabled
- 110 = CPU Interrupt Priority Level is 6 (14)
- 101 = CPU Interrupt Priority Level is 5 (13)
- 100 = CPU Interrupt Priority Level is 4 (12)
- 011 = CPU Interrupt Priority Level is 3 (11)
- 010 = CPU Interrupt Priority Level is 2 (10)
- 001 = CPU Interrupt Priority Level is 1 (9)
- 000 = CPU Interrupt Priority Level is 0 (8)

bit 4 **RA:** REPEAT Loop Active Status bit
- 1 = REPEAT loop in progress
- 0 = REPEAT loop not in progress

bit 3 **N:** MCU ALU Negative bit
- 1 = Result was negative
- 0 = Result was non-negative (zero or positive)

bit 2 **OV:** MCU ALU Overflow bit

This bit is used for signed arithmetic (2's complement). It indicates an overflow of the magnitude that causes the sign bit to change state.
- 1 = Overflow occurred for signed arithmetic (in this arithmetic operation)
- 0 = No overflow occurred

bit 1 **Z:** MCU ALU Zero bit
- 1 = An operation that affects the Z bit has set it at some time in the past
- 0 = The most recent operation that affects the Z bit has cleared it (i.e., a non-zero result)

bit 0 **C:** MCU ALU Carry/$\overline{\text{Borrow}}$ bit
- 1 = A carry-out from the Most Significant bit of the result occurred
- 0 = No carry-out from the Most Significant bit of the result occurred

Note 1: The IPL<2:0> bits are concatenated with the IPL<3> (CORCON<3>) bit to form the CPU Interrupt Priority Level. The value in parentheses indicates the IPL if IPL<3> = 1. User interrupts are disabled when IPL<3> = 1.

2: The IPL<2:0> Status bits are read-only when NSTDIS = 1 (INTCON1<15>).

Register 44-2: CORCON: Core Control Register

U-0	U-0	U-0	U-0	U-0	U-0	U-0	U-0
—	—	—	—	—	—	—	—
bit 15							bit 8

U-0	U-0	U-0	U-0	R/C-0, HSC	r-1	U-0	U-0
—	—	—	—	IPL3(1)	Reserved	—	—
bit 7							bit 0

Legend:	C = Clearable bit	r = Reserved bit	HSC = Hardware Settable/Clearable bit
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 15-4 **Unimplemented:** Read as '0'

bit 3 **IPL3:** CPU Interrupt Priority Level Status bit 3(1)
1 = CPU Interrupt Priority Level is greater than 7
0 = CPU Interrupt Priority Level is 7 or less

bit 2 **Reserved:** Read as '1'

bit 1-0 **Unimplemented:** Read as '0'

Note 1: The IPL3 bit is concatenated with the IPL<2:0> (SR<7:5>) bits to form the CPU interrupt priority level.

44.4.3 Other PIC24F Devices with EDS Control Registers

The following registers are associated with the PIC24F with EDS devices, but are described in further detail in other sections of this manual.

- **TBLPAG: Table Page Register**

  The TBLPAG register holds the upper 8 bits of a program memory address during table read and write operations. Table instructions are used to transfer data between program memory space and data memory space. For further details, refer to "*PIC24F Family Reference Manual*" **Section 4. "Program Memory"**.

- **DSRPAG: Extended Data Space Read Page Register**

  The 10-bit DSRPAG register extends DS read access address space to a total of 32 Mbytes. DSRPAG page values, between 0x001 and 0x1FF, provide read access for a 16-Mbyte address space referred to as EDS. DSRPAG page values, between 0x200 and 0x2FF, provide read access to lower words, while DSRPAG page values, between 0x300 to 0x3FF, provide the read access to the lower byte of upper words of the Program Memory (PM).

- **DSWPAG: Extended Data Space Write Page Register**

  The 9-bit DSWPAG register extends DS write access space to 16 Mbytes (writes to PM space are not permitted using EDS). DSWPAG page values, between 0x01 and 0x1FF, provide write access to EDS.

Note 1: Use DSRPAG register while performing read-modify-write operations, such as bit-oriented instructions.

2: PSV is usually the program memory in the device; it cannot be written using the EDS mechanism. However, table writes can be used to write data into PSV space.

- **DISICNT: Disable Interrupts Count Register**

  The DISICNT register is used by the DISI instruction to disable interrupts of priority 1-6 for a specified number of cycles. For further information, refer to the "*PIC24F Family Reference Manual*" **Section 8. "Interrupts"**.

44.5 ARITHMETIC LOGIC UNIT (ALU)

The PIC24F devices with EDS ALU is 16 bits wide and is capable of addition, subtraction, single bit shifts and logic operations. Unless otherwise mentioned, arithmetic operations are 2's complement in nature. Depending on the operation, the ALU can affect the values of these bits in the SR register:

- Carry (C)
- Zero (Z)
- Negative (N)
- Overflow (OV)
- Digit Carry (DC)

The C and DC Status bits operate as $\overline{\text{Borrow}}$ and $\overline{\text{Digit Borrow}}$ bits, respectively, for subtraction operations.

The ALU can perform in 8-bit or 16-bit operation, depending on the mode of the instruction that is used. Data for the ALU operation can come from the W register array or data memory depending on the addressing mode of the instruction. Likewise, output data from the ALU can be written to the W register array or a data memory location.

For information on the SR bits affected by each instruction, addressing modes and 8-/16-bit instruction modes, refer to the *"16-Bit MCU and DSC Programmer's Reference Manual"*.

Note 1:	Byte operations use the 16-bit ALU and can produce results in excess of 8 bits. However, to maintain backward compatibility with PIC MCU devices, the ALU result from all byte operations is written back as a byte (i.e., MSB is not modified) and the SR register is updated based only upon the state of the LSB of the result.
2:	All register instructions, performed in Byte mode, affect only the LSB of the W registers. The MSB of any W register can be modified by using file register instructions that access the memory mapped contents of the W registers.

44.6 MULTIPLICATION AND DIVIDE SUPPORT

44.6.1 Overview

The PIC24F devices with EDS contain a 17-bit x 17-bit multiplier and is capable of unsigned, signed or mixed sign operation with the following multiplication modes:

- 16-bit x 16-bit signed
- 16-bit x 16-bit unsigned
- 16-bit signed x 5-bit (literal) unsigned
- 16-bit unsigned x 16-bit unsigned
- 16-bit unsigned x 5-bit (literal) unsigned
- 16-bit unsigned x 16-bit signed
- 8-bit unsigned x 8-bit unsigned

The divide block is capable of supporting 32/16-bit and 16/16-bit signed and unsigned integer divide operation with the following data sizes:

- 32-bit signed/16-bit signed divide
- 32-bit unsigned/16-bit unsigned divide
- 16-bit signed/16-bit signed divide
- 16-bit unsigned/16-bit unsigned divide

44.6.2 Multiplier

Figure 44-8 is a block diagram of the multiplier. It supports the multiply instructions, which include integer 16-bit signed, unsigned and mixed sign multiplies. All multiply instructions only support Register Direct Addressing mode for the result. A 32-bit result (from all multiplies other than `MULWF`) is written to any two aligned, consecutive W register pairs, except W15:W14, which are not allowed.

The `MULWF` instruction may be directed to use byte or word-sized operands. The destination is always the W3:W2 register pair in the W array. Byte multiplicands direct a 16-bit result to W2 (W3 is not changed) and word multiplicands direct a 32-bit result to W3:W2.

Note:	The destination register pair for multiply instructions must be ‘aligned’ (i.e., odd:even), where ‘odd’ contains the most significant result word and ‘even’ contains the least significant result word. For example, W3:W2 is acceptable, whereas W4:W3 is not.

The multiplicands for all multiply instructions are derived from the W array (1st word) and data space (2nd word). The `MULWF` instruction derives these multiplicands from W2 (1st word or byte) and data space (2nd word or byte) using a zero-extended, 13-bit absolute address.

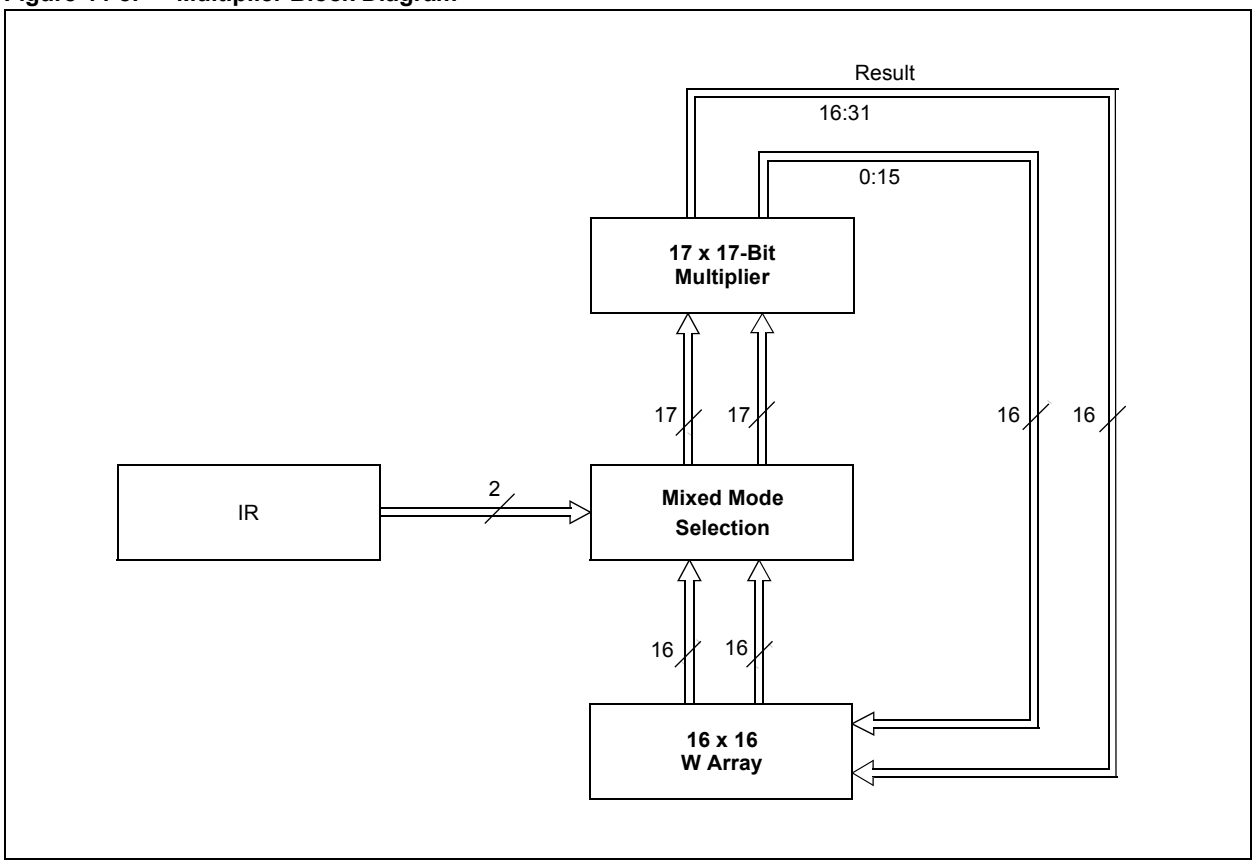
Figure 44-8 shows additional data paths that are provided to allow these instructions to write the result back into the W array and data bus (via the W array).

44.6.2.1 SINGLE AND MIXED MODE INTEGERS

Simple data preprocessing logic, either zero-extend or sign-extend operand to 17 bits, such that unsigned, signed or mixed sign multiplications can be executed as signed values. All unsigned operands are always zero-extended into the 17th bit of the multiplier input value. All signed operands are always sign-extended into the 17th bit of the multiplier input value.

- For unsigned 16-bit multiplies, the multiplier produces a 32-bit, unsigned result
- For signed 16-bit multiplies, the multiplier produces 30 bits of data and 2 bits of sign
- For 16-Bit Mixed mode (signed/unsigned) multiplies, the multiplier produces 31 bits of data and 1 bit of sign

Figure 44-8: Multiplier Block Diagram



44.6.3 Divider

The PIC24F with EDS features both 32/16-bit and 16/16-bit signed and unsigned, integer divide operations implemented as single instruction iterative divides.

The quotient for all divide instructions ends up in W0 and the remainder in W1. The 16-bit signed and unsigned DIV instructions can specify any W register for both the 16-bit divisor (Wn) and any W register (aligned) pair (W(m + 1):Wm) for the 32-bit dividend. The divide algorithm takes 1 cpb (cycle per bit) of the divisor, so both 32/16-bit and 16/16-bit instructions take the same number of cycles to execute.

The divide instructions must be executed within a REPEAT loop. Any other form of execution (e.g., a series of discrete divide instructions) will not function correctly because the instruction flow function is conditional on RCOUNT. The divide flow does not automatically set up the REPEAT, which must therefore, be explicitly executed with the correct operand value as shown in Table 44-2 (REPEAT will execute the target instruction {operand value + 1} time).

Table 44-2: Divide Execution Time

Instruction	Description	Iterations	REPEAT Operand Value	Total Execution Time (including REPEAT)
DIV.SD	Signed divide: W(m + 1):Wm/Wn → W0; Rem → W1	18	17	19
DIV.SW	Signed divide: Wm/Wn → W0; Rem → W1	18	17	19
DIV.UD	Unsigned divide: W(m + 1):Wm/Wn → W0; Rem → W1	18	17	19
DIV.UW	Unsigned divide: Wm/Wn → W0; Rem → W1	18	17	19

All intermediate data is saved in W1:W0 after each iteration. The N, C and Z Status flags convey control information between iterations. Consequently, although the divide instructions are listed as 19-cycle operations, the divide iterative sequence is interruptible, just like any other REPEAT loop.

Dividing by zero initiates an arithmetic error trap. The divisor is evaluated during the first cycle of the divide instruction, so the first cycle executes prior to the start of exception processing for the trap. For more details, refer to "*PIC24F Family Reference Manual*" **Section 8. "Interrupts"**.

44.7 COMPILER FRIENDLY ARCHITECTURE

The CPU architecture is designed to produce an efficient (code size and speed) C compiler.

- For most instructions, the core is capable of executing a data (or program data) memory read, a working register (data) read, a data memory write and a program (instruction) memory read per instruction cycle. As a result, three parameter instructions are supported, allowing A + B = C operations to be executed in a single cycle.
- Instruction addressing modes are very flexible and are matched closely to compiler needs.
- There are sixteen, 16 x 16-bit working register arrays, each of which can act as data, address or offset registers. One working register (W15) operates as a Software Stack Pointer for interrupts and calls.
- Linear indirect access of all data space is supported, plus the memory direct address range is extended to 8 Kbytes, with additional 16-bit direct address load and store instructions.
- Linear indirect access of 32K word (64 Kbytes) pages within program space (user and test space) is supported using any working register via new table read and write instructions.
- Part of the program space can be mapped into data space, allowing constant data to be accessed as if it were in data space using PSV mode.

44.8 MULTI-BIT SHIFT SUPPORT

The PIC24F devices with EDS support single-cycle, multi-bit arithmetic and logic shifts, using a shifter block. It also supports single bit shifts through the ALU. The multi-bit shifter is capable of performing up to a 15-bit arithmetic right shift, or up to a 15-bit left shift, in a single cycle.

Table 44-3 provides a full summary of instructions that use the shift operation.

Table 44-3: Instructions Using Single and Multi-Bit Shift Operations

Instruction	Description
ASR	Arithmetic Shift Right Source register by one or more bits.
SL	Shift Left Source register by one or more bits.
LSR	Logical Shift Right Source register by one or more bits.

All multi-bit shift instructions only support Register Direct Addressing mode for both the operand source and result destination.

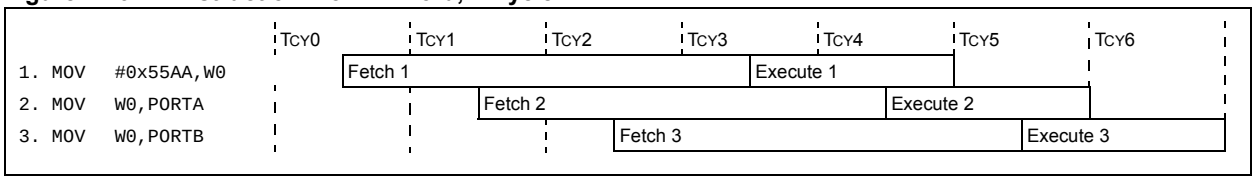
44.9 INSTRUCTION FLOW TYPES

Most instructions in the PIC24F with EDS architecture occupy a single word of program memory and execute in a single cycle. An instruction prefetch mechanism facilitates single-cycle (1 TCY) execution. However, some instructions take two or three instruction cycles to execute. Consequently, there are six different types of instruction flow in the PIC24F with EDS microcontroller architecture. These are described in the succeeding sections:

44.9.1 1 Instruction Word, 1 Instruction Cycle

These instructions take one instruction cycle to execute, as shown in Figure 44-9. Most instructions are 1-word, 1-cycle instructions.

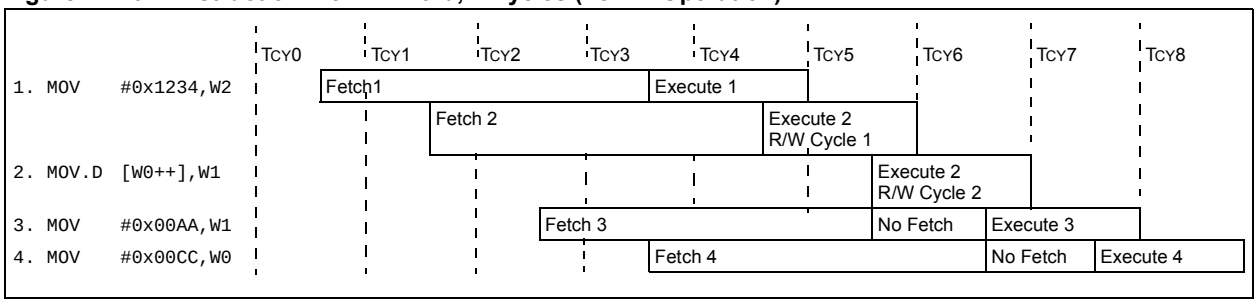
Figure 44-9: Instruction Flow: 1 Word, 1 Cycle



44.9.2 1 Instruction Word, 2 Instruction Cycles

In these instructions, there is no prefetch flush. The only instructions of this type are the `MOV.D` instructions (load and store double-word), SFR reads and SFR bit operations. Two cycles are required to complete these instructions, as shown in Figure 44-10.

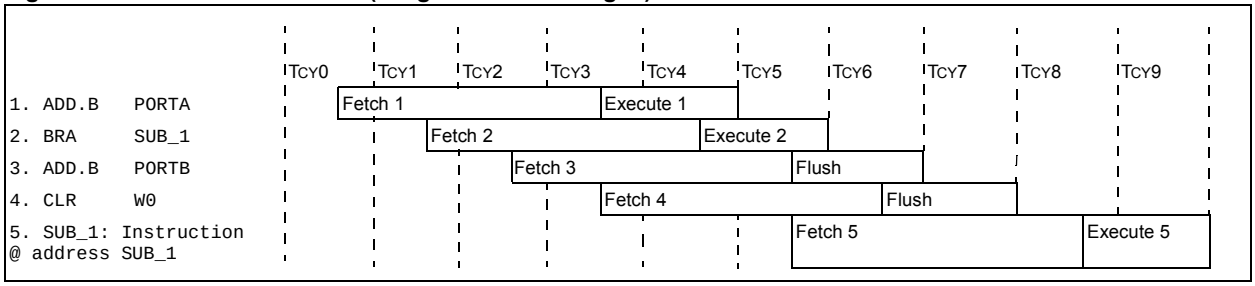
Figure 44-10: Instruction Flow: 1 Word, 2 Cycles (`MOV.D` Operation)



44.9.3 1 Instruction Word, 2 or 3 Instruction Cycles (Program Flow Changes)

These instructions include relative call and branch instructions, and skip instructions. When an instruction changes the PC (other than to increment it), the program memory prefetch data must be discarded. This makes the instruction take two effective cycles to execute, as shown in Figure 44-11.

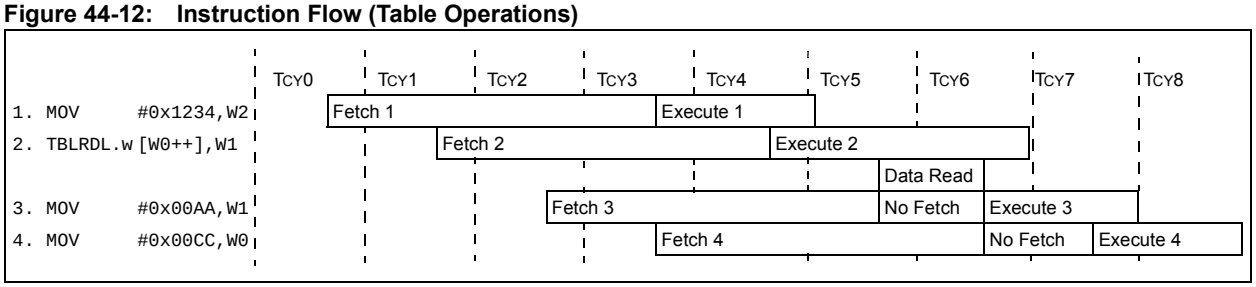
Figure 44-11: Instruction Flow (Program Flow Changes)



44.9.4 Table Read/Write Instructions

These instructions suspend fetching to insert a read or write cycle to the program memory. Figure 44-12 shows the instruction fetched, while executing the table operation, is saved for one cycle and executed in the cycle immediately after the table operation.

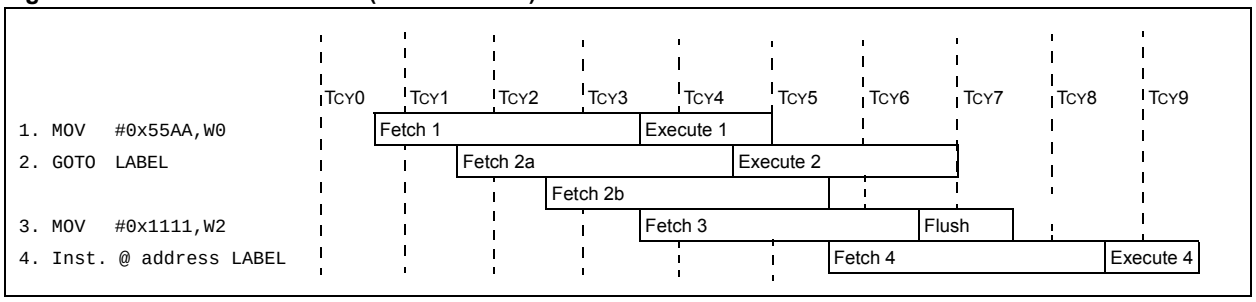
Figure 44-12: Instruction Flow (Table Operations)



44.9.5 2 Instruction Words, 2 Instruction Cycles – GOTO or CALL

In these instructions, the fetch after the instruction contains the data. This results in a two-cycle instruction, as shown in Figure 44-13. The second word of a two-word instruction is encoded so that it executes as a NOP if it is fetched by the CPU when the CPU did not first fetch the first word of the instruction. This is important when a two-word instruction is skipped by a skip instruction (refer to Figure 44-13).

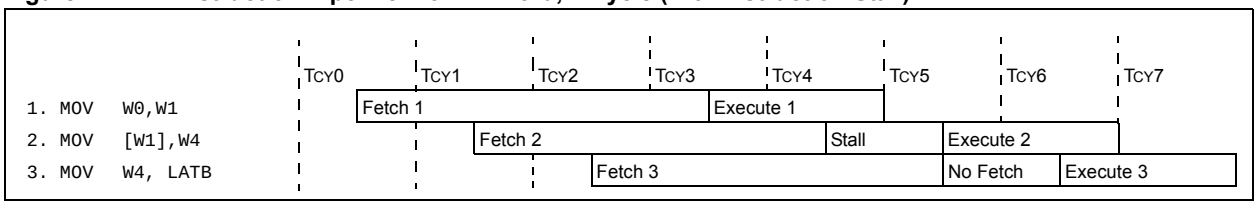
Figure 44-13: Instruction Flow (GOTO or CALL)



44.9.6 Address Register Dependencies

These are instructions subjected to a stall due to a data address dependency between the data space read and write operations. An additional cycle is inserted to resolve the resource conflict, as discussed in **Section 44.11 "Address Register Dependencies"**.

Figure 44-14: Instruction Pipeline Flow: 1 Word, 1 Cycle (with Instruction Stall)



44.10 LOOP CONSTRUCTS

The PIC24F with EDS supports the REPEAT instruction construct to provide unconditional automatic program loop control. The REPEAT instruction implements a single instruction program loop. This instruction uses control bits within the CPU STATUS Register (SR) to temporarily modify CPU operation.

44.10.1 REPEAT Loop Construct

The REPEAT instruction causes the instruction that follows it to be repeated a specified number of times. A literal value contained in the instruction, or a value in one of the W registers, can be used to specify the REPEAT count value. The W register option enables the loop count to be a software variable.

An instruction in a REPEAT loop is executed at least once. The number of iterations for a REPEAT loop is the 14-bit literal value: + 1 or Wn + 1.

The syntax for the two forms of the REPEAT instruction is:

```
REPEAT #lit14             ; RCOUNT <-- lit14
(Valid target Instruction)
```

or

```
REPEAT Wn                 ; RCOUNT <-- Wn
(Valid target Instruction)
```

44.10.1.1 REPEAT OPERATION

The loop count for REPEAT operations is held in the Repeat Loop Counter (RCOUNT) register, which is memory mapped. RCOUNT is initialized by the REPEAT instruction. The REPEAT instruction sets the Repeat Loop Active (RA) Status bit (SR<4>) to '1' if the RCOUNT is a non-zero value.

RA is a read-only bit and cannot be modified through software. For REPEAT loop count values greater than '0', the PC is not incremented. Further PC increments are inhibited until RCOUNT = 0. For an instruction flow example of a REPEAT loop, refer to Example 44-1.

For a loop count value equal to '0', REPEAT has the effect of a NOP and the RA (SR<4>) bit is not set. The REPEAT loop is essentially disabled before it begins, allowing the target instruction to execute only once while prefetching the subsequent instruction (i.e., normal execution flow). A REPEAT loop exit will incur a 2 TCY stall due to Flash latency.

Note:	The instruction immediately following the REPEAT instruction (i.e., the target instruction) is always executed, at least one time, and it is always executed one time more than the value specified in the 14-bit literal or the W register operand.

Example 44-1: REPEAT Instruction Pipeline Flow

	TCY0	TCY1	TCY2	TCY3	TCY4	TCY5
1.REPEAT #0x2	Fetch 1	Execute 1				
2.MOV [W0++], [W1++]		Fetch 2	Execute 2			
			No Fetch	Execute 2		
				No Fetch	Execute 2	
3.BSET PORTA, #3					Fetch 3	Execute 3

PC (at end of instruction)	PC	PC + 2	PC + 2	PC + 2	PC + 4	PC + 6
RCOUNT (at end of instruction)	x	2	1	0	0	0
RA (at end of instruction)	0	1	1	0	0	0

Note: A consequence of repeating the instruction is that, even when the repeated instruction is performing a PSV read, only the first and last iteration incurs a 2 TCY stall due to Flash latency. All other iterations execute with an effective throughput of 1 instruction per cycle. However, this data pipelining is limited to certain addressing modes: post-increment or post-decrement by 1 or 2.

44.10.1.2 INTERRUPTING A REPEAT LOOP

A REPEAT instruction loop can be interrupted at any time.

The state of the RA bit is preserved on the stack during exception processing to enable the user application to execute further REPEAT loops from within any number of nested interrupts. After SRL is stacked, the RA Status bit is cleared to restore normal execution flow within the Interrupt Service Routine (ISR).

Note 1: If a REPEAT loop has been interrupted, and an ISR is being processed, the user application must stack the Repeat Count (RCOUNT) register before it executes another REPEAT instruction within an ISR.

2: If a REPEAT instruction is used within an ISR, the user application must unstack the RCOUNT register before it executes the RETFIE instruction.

Returning into a REPEAT loop from an ISR, using the RETFIE instruction, requires no special handling. Interrupts prefetch the repeated instruction during the 5th cycle of the RETFIE instruction. The stacked RA bit is restored when the SRL register is popped, and if set, the interrupted REPEAT loop is resumed.

Note: Should the repeated instruction (target instruction in the REPEAT loop) access data from program space using PSV, it will require two additional instruction cycles, the first time it is executed, after a return from an exception.

44.10.1.2.1 Early Termination of a REPEAT Loop

An interrupted REPEAT loop can be terminated earlier than normal in the ISR by clearing the RCOUNT register in software.

44.10.1.3 RESTRICTIONS ON THE REPEAT INSTRUCTION

Any instruction can immediately follow a REPEAT except for the following:

- Program Flow Control instructions (any branch, compare and skip, subroutine calls, returns, etc.)
- Another REPEAT instruction
- DISI, ULNK, LNK, PWRSAV or RESET instruction
- MOV.D instruction

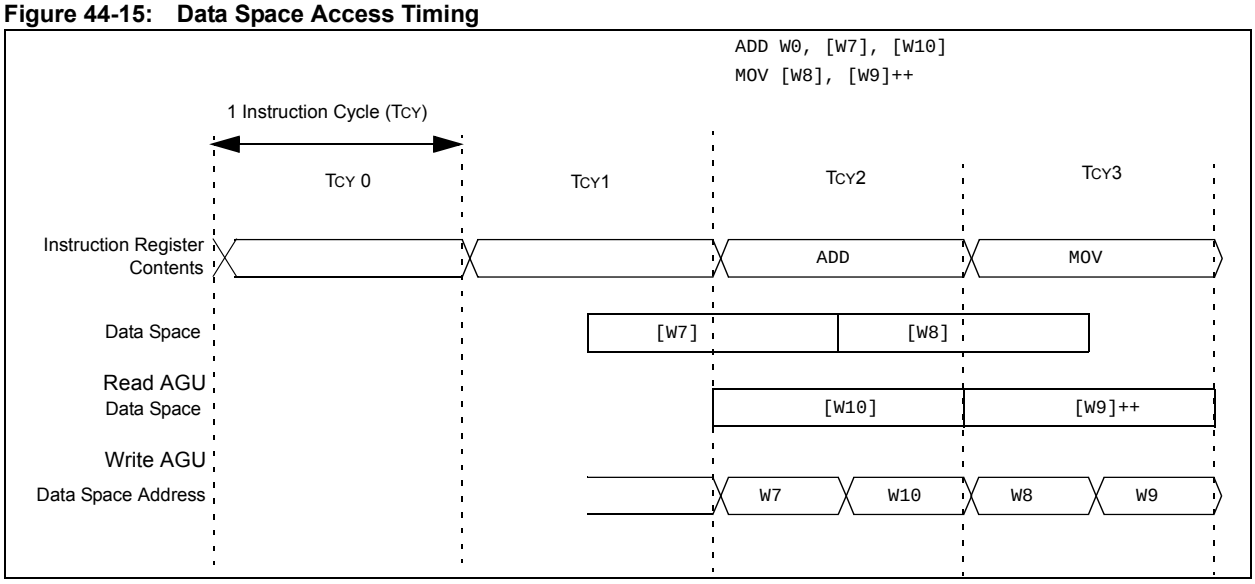
Note 1: Some instructions and/or instruction addressing modes can be executed within a REPEAT loop, but it might not make sense to repeat all instructions.

2: Use of the REPEAT instruction with read-modify-write operations on EDS is not supported.

44.11 ADDRESS REGISTER DEPENDENCIES

The PIC24F with EDS architecture supports a data space read (source) and a data space write (destination) for most MCU class instructions. The Effective Address (EA) calculation by the AGU, and subsequent data space read or write, each take one instruction cycle to complete. This timing causes the data space read and write operations for each instruction to partially overlap, as shown in Figure 44-15. Because of this overlap, a 'Read-After-Write' (RAW) data dependency can occur across instruction boundaries. RAW data dependencies are detected and handled at run time by the Section 44. CPU with Extended Data Space (EDS) with EDS.

Figure 44-15: Data Space Access Timing



44.11.1 Read-After-Write (RAW) Dependency Rules

If the W register is used as a write operation destination in the current instruction and the W register being read in the prefetched instruction are the same, the following rules apply:

- If the destination write (current instruction) does not modify the contents of Wn, no stalls will occur.
- If the source read (prefetched instruction) does not calculate an EA using Wn, no stalls will occur.

During each instruction cycle, the PIC24F with EDS hardware automatically checks to see if a RAW data dependency is about to occur. If the conditions specified above are not satisfied, the CPU automatically adds a one instruction cycle delay before executing the prefetched instruction. The instruction stall provides enough time for the destination W register write to take place before the next (prefetched) instruction needs to use the written data. Table 44-4 is a RAW dependency summary.

Table 44-4: Read-After-Write (RAW) Dependency Summary

Destination Addressing Mode Using Wn	Source Addressing Mode Using Wn	Status	Examples (Wn = W2)
Direct	Direct	Allowed	`ADD.w W0, W1, W2` `MOV.w W2, W3`
Direct	Indirect	Stall	`ADD.w W0, W1, W2` `MOV.w [W2], W3`
Direct	Indirect with modification	Stall	`ADD.w W0, W1, W2` `MOV.w [W2++], W3`
Indirect	Direct	Allowed	`ADD.w W0, W1, [W2]` `MOV.w W2, W3`
Indirect	Indirect	Allowed	`ADD.w W0, W1, [W2]` `MOV.w [W2], W3`
Indirect	Indirect with modification	Allowed	`ADD.w W0, W1, [W2]` `MOV.w [W2++], W3`
Indirect with modification	Direct	Allowed	`ADD.w W0, W1, [W2++]` `MOV.w W2, W3`

Indirect	Indirect	Stall	`ADD.w W0, W1, [W2]`
			`MOV.w [W2], W3`
			`; W2=0x0004 (mapped W2)`

Indirect	Indirect with modification	Stall	`ADD.w W0, W1, [W2]`
			`MOV.w [W2++], W3`
			`; W2=0x0004 (mapped W2)`

Indirect with modification	Indirect	Stall	`ADD.w W0, W1, [W2++]` `MOV.w [W2], W3`
Indirect with modification	Indirect with modification	Stall	`ADD.w W0, W1, [W2++]` `MOV.w [W2++], W3`

44.11.2 Instruction Stall Cycles

An instruction stall is essentially a wait period instruction cycle added in front of the read phase of an instruction to allow the prior write to complete before the next read operation. For the purposes of interrupt latency, the stall cycle is associated with the instruction following the instruction where it was detected (i.e., stall cycles always precede instruction execution cycles).

If a RAW data dependency is detected, the PIC24F with EDS begins an instruction stall. During an instruction stall, the following events occur:

- The write operation underway (for the previous instruction) is allowed to complete as normal.
- Data space is not addressed until after the instruction stall.
- PC increment is inhibited until after the instruction stall.
- Further instruction fetches are inhibited until after the instruction stall.

44.11.2.1 INSTRUCTION STALL CYCLES AND INTERRUPTS

When an interrupt event coincides with two adjacent instructions that will cause an instruction stall, one of two following possible outcomes can occur:

- If the interrupt coincides with the first instruction, the first instruction is allowed to complete while the second instruction is executed after the ISR completes. In this case, the stall cycle is eliminated from the second instruction because the exception process provides time for the first instruction to complete the write phase.
- If the interrupt coincides with the second instruction, the second instruction and the appended stall cycle are allowed to execute before the ISR completes. In this case, the stall cycle associated with the second instruction executes normally. However, the stall cycle is effectively absorbed into the exception process timing. The exception process proceeds as if an ordinary two-cycle instruction was interrupted.

44.11.2.2 INSTRUCTION STALL CYCLES AND FLOW CHANGE INSTRUCTIONS

The CALL and RCALL instructions write to the stack using working register, W15, and can therefore, force an instruction stall prior to the next instruction if the source read of the next instruction uses W15.

The RETFIE and RETURN instructions can never force an instruction stall prior to the next instruction because they only perform read operations. However, the RETLW instruction can force a stall because it writes to a W register during the last cycle.

The GOTO and branch instructions can never force an instruction stall because they do not perform write operations.

44.11.2.3 INSTRUCTION STALLS AND REPEAT LOOPS

Other than the addition of instruction stall cycles, RAW data dependencies do not affect the operation of REPEAT loops.

The prefetched instruction within a REPEAT loop does not change until the loop is complete or an exception occurs. Although register dependency checks occur across instruction boundaries, the PIC24F with EDS effectively compares the source and destination of the same instruction during a REPEAT loop.

44.11.2.4 INSTRUCTION STALLS DURING EDS AND PSV ACCESS

Accessing data from physically implemented internal memory (program memory or data memory), via EDS or PSV addressing, takes one extra instruction cycle from those on non-EDS; therefore, incur a 1-cycle stall to ensure that the data is available.

Instructions operating in EDS or PSV address space are subject to RAW data dependencies and consequent instruction stalls.

Consider the following code segment:

```
ADD     W0,[W1],[W2++]     ; W1=0x8000, DSRPAG=0x200
MOV     [W2],[W3]
```

This sequence of instructions would take five instruction cycles to execute. Two instruction cycles are added to perform the EDS or PSV access via W1. An instruction stall cycle is inserted to resolve the RAW data dependency caused by W2. Refer to the "*PIC24F Family Reference Manual*" **Section 42.** "**Enhanced Parallel Master Port (EPMP)"** for more details on how to handle RAW data dependency while accessing EDS via another bus master, like the EPMP module.

Note:	With EDS/PSV reads under a REPEAT instruction, the first two accesses take three cycles each and the subsequent accesses take only one cycle.

44.12 REGISTER MAPS

A summary of the registers associated with the PIC24F with EDS is provided in Table 44-5.

Table 44-5: PIC24F CPU with EDS Register Map

Name	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Reset State
W0	W0 (WREG)																0000 0000 0000 0000
W1	W1																0000 0000 0000 0000
W2	W2																0000 0000 0000 0000
W3	W3																0000 0000 0000 0000
W4	W4																0000 0000 0000 0000
W5	W5																0000 0000 0000 0000
W6	W6																0000 0000 0000 0000
W7	W7																0000 0000 0000 0000
W8	W8																0000 0000 0000 0000
W9	W9																0000 0000 0000 0000
W10	W10																0000 0000 0000 0000
W11	W11																0000 0000 0000 0000
W12	W12																0000 0000 0000 0000
W13	W13																0000 0000 0000 0000
W14	W14																0000 0000 0000 0000
W15	W15																0000 0000 0000 0000
SPLIM	SPLIM																0000 0000 0000 0000
PCL	PCL																0000 0000 0000 0000
PCH	—	—	—	—	—	—	—	—	—	PCH							0000 0000 0000 0000
DSRPAG	—	—	—	—	—	—	DSRPAG<9:0>										0000 0000 0000 0001
DSWPAG	—	—	—	—	—	—	—	DSWPAG<8:0>									0000 0000 0000 0001
RCOUNT	RCOUNT																xxxx xxxx xxxx xxxx
SR	—	—	—	—	—	—	—	DC	IPL2	IPL1	IPL0	RA	N	OV	Z	C	0000 0000 0000 0000
CORCON	—	—	—	—	—	—	—	—	—	—	—	—	IPL3	r	—	—	0000 0000 0000 0100
DISICNT	—	—	DISICNT<13:0>														0000 0000 0000 0000
TBLPAG	—	—	—	—	—	—	—	—	TBLPAG<7:0>								0000 0000 0000 0000

Legend: x = uninitiated, r = reserved

Note: Refer to the specific device data sheet for Core Register Map details.

44.13 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 PIC24F device family, but the concepts are pertinent and could be used with modification and possible limitations. The current application notes related to the CPU with Extended Data Space (EDS) are:

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

Note: Please visit the Microchip web site (www.microchip.com) for additional application notes and code examples for the PIC24F family of devices.

44.14 REVISION HISTORY

Revision A (December 2009)

This is the initial released version of this document.

