The Access Bank is used by core PIC18 instructions that include the Access RAM bit (the 'a' parameter in the instruction). When 'a' is equal to '1', the instruction uses the BSR and the 8-bit address included in the opcode for the data memory address. When 'a' is '0', however, the instruction ignores the BSR and uses the Access Bank address map.

Using this "forced" addressing allows the instruction to operate on a data address in a single cycle, without updating the BSR first. Access RAM also allows for faster and more code efficient context saving and switching of variables.

The mapping of the Access Bank is slightly different when the extended instruction set is enabled ($\overline{\text{XINST}}$ Configuration bit = 1). This is discussed in more detail in the Mapping the Access Bank in Indexed Liberal Offset Mode section.

9.5 Data Addressing Modes

Important: The execution of some instructions in the core PIC18 instruction set are changed when the PIC18 extended instruction set is enabled. See Data Memory and the Extended Instruction Set for more information.

Information in the data memory space can be addressed in several ways. For most instructions, the Addressing mode is fixed. Other instructions may use up to three modes, depending on which operands are used and whether or not the extended instruction set is enabled.

The Addressing modes are:

- Inherent
- Literal
- Direct
- Indirect

An additional Addressing mode, Indexed Literal Offset, is available when the extended instruction set is enabled (XINST Configuration bit = 1). Its operation is discussed in greater detail in Indexed Addressing with Literal Offset.

9.5.1 Inherent and Literal Addressing

Many PIC18 control instructions do not need any argument at all; they either perform an operation that globally affects the device or they operate implicitly on one register. This Addressing mode is known as Inherent Addressing. Examples include SLEEP, RESET and DAW.

Other instructions work in a similar way but require an additional explicit argument in the opcode. This is known as Literal Addressing mode because they require some literal value as an argument. Examples include ADDLW and MOVLW, which respectively, add or move a literal value to the W register. Other examples include CALL and GOTO, which include a program memory address.

9.5.2 Direct Addressing

Direct Addressing specifies all or part of the source and/or destination address of the operation within the opcode itself. The options are specified by the arguments accompanying the instruction.

In the core PIC18 instruction set, bit-oriented and byte-oriented instructions use some version of Direct Addressing by default. All of these instructions include some 8-bit literal address as their Least Significant Byte. This address specifies either a register address in one of the banks of data RAM (see Data Memory Organization) or a location in the Access Bank (see Access Bank) as the data source for the instruction.

The Access RAM bit 'a' determines how the address is interpreted. When 'a' is '1', the contents of the BSR (see Bank Select Register) are used with the address to determine the complete 12-bit address of the register. When 'a' is '0', the address is interpreted as being a register in the Access Bank.

The destination of the operation's results is determined by the destination bit 'd'. When 'd' is '1', the results are stored back in the source register, overwriting its original contents. When 'd' is '0', the results are stored in the W register.

Instructions without the ‘d’ argument have a destination that is implicit in the instruction; their destination is either the target register being operated on or the W register.

9.5.3 Indirect Addressing

Indirect Addressing allows the user to access a location in data memory without giving a fixed address in the instruction. This is done by using File Select Registers (FSRs) as pointers to the locations which are to be read or written. Since the FSRs are themselves located in RAM as Special File Registers, they can also be directly manipulated under program control. This makes FSRs very useful in implementing data structures, such as tables and arrays in data memory.

The registers for Indirect Addressing are also implemented with Indirect File Operands (INDFs) that permit automatic manipulation of the pointer value with auto-incrementing, auto-decrementing or offsetting with another value. This allows for efficient code, using loops, such as the following example of clearing an entire RAM bank.

Example 9-3. How to Clear RAM (Bank 1) Using Indirect Addressing

```
     LFSR    FSR0,100h    ; Set FSR0 to beginning of Bank1
NEXT:
     CLRF    POSTINC0     ; Clear location in Bank1 then increment FSR0

     BTFSS   FSR0H,1      ; Has high FSR0 byte incremented to next bank?
     BRA     NEXT         ; NO, clear next byte in Bank1

CONTINUE:                 ; YES, continue
```

9.5.3.1 FSR Registers and the INDF Operand

At the core of Indirect Addressing are three sets of registers: FSR0, FSR1 and FSR2. Each represent a pair of 8-bit registers, FSRnH and FSRnL. Each FSR pair holds the full address of the RAM location. The FSR value can address the entire range of the data memory in a linear fashion. The FSR register pairs, then, serve as pointers to data memory locations.

Indirect Addressing is accomplished with a set of Indirect File Operands, INDF0 through INDF2. These can be thought of as “virtual” registers; they are mapped in the SFR space but are not physically implemented. Reading or writing to a particular INDF register actually accesses its corresponding FSR register pair. A read from INDF1, for example, reads the data at the address indicated by FSR1H:FSR1L. Instructions that use the INDF registers as operands actually use the contents of their corresponding FSR as a pointer to the instruction’s target. The INDF operand is just a convenient way of using the pointer.

Because Indirect Addressing uses a full address, the FSR value can target any location in any bank regardless of the BSR value. However, the Access RAM bit must be cleared to zero to ensure that the INDF register in Access space is the object of the operation instead of a register in one of the other banks. The assembler default value for the Access RAM bit is zero when targeting any of the indirect operands.

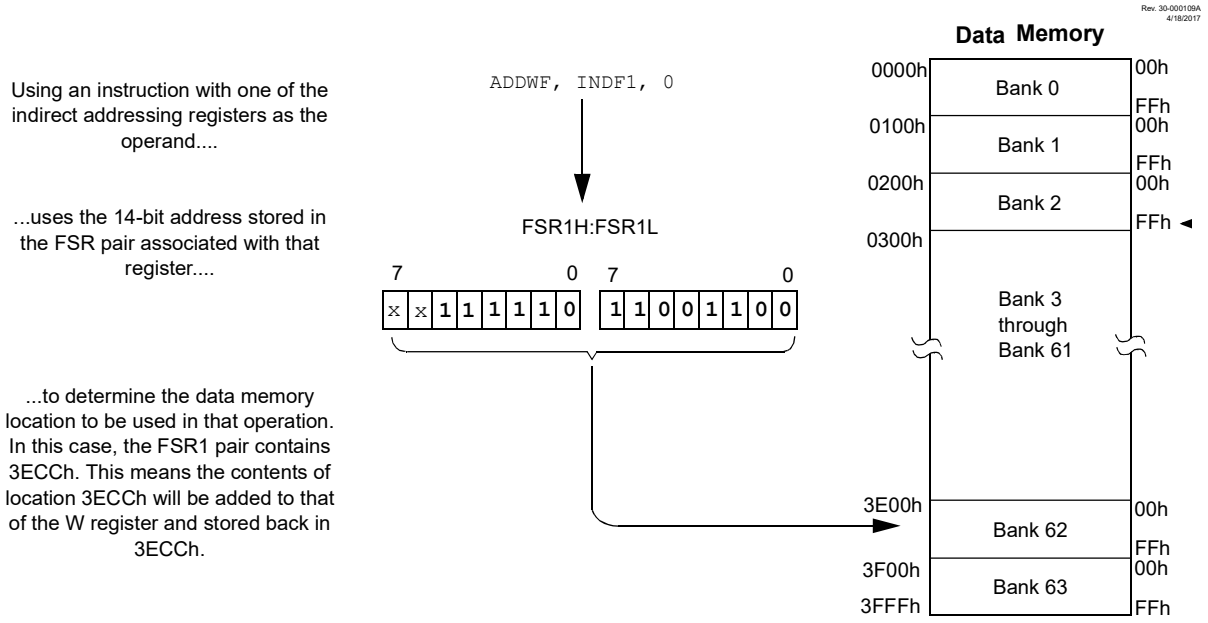
9.5.3.2 FSR Registers and POSTINC, POSTDEC, PREINC and PLUSW

In addition to the INDF operand, each FSR register pair also has four additional indirect operands. Like INDF, these are “virtual” registers that cannot be directly read or written. Accessing these registers actually accesses the location to which the associated FSR register pair points, and also performs a specific action on the FSR value. They are:

- POSTDEC: Accesses the location to which the FSR points, then automatically decrements the FSR by 1 afterwards
- POSTINC: Accesses the location to which the FSR points, then automatically increments the FSR by 1 afterwards
- PREINC: Automatically increments the FSR by one, then uses the location to which the FSR points in the operation
- PLUSW: Adds the signed value of the W register (range of -127 to 128) to that of the FSR and uses the location to which the result points in the operation.

In this context, accessing an INDF register uses the value in the associated FSR register without changing it. Similarly, accessing a PLUSW register gives the FSR value an offset in the W register; however, neither W nor the FSR is actually changed in the operation. Accessing the other virtual registers changes the value of the FSR register.

Figure 9-5. Indirect Addressing



Operations on the FSRs with POSTDEC, POSTINC and PREINC affect the entire register pair; that is, rollovers of the FSRnL register from FFh to 00h carry over to the FSRnH register. On the other hand, results of these operations do not change the value of any flags in the STATUS register (e.g., Z, N, OV, etc.).

The PLUSW register can be used to implement a form of Indexed Addressing in the data memory space. By manipulating the value in the W register, users can reach addresses that are fixed offsets from pointer addresses. In some applications, this can be used to implement some powerful program control structure, such as software stacks, inside of data memory.

9.5.3.3 Operations by FSRs on FSRs

Indirect Addressing operations that target other FSRs or virtual registers represent special cases. For example, using an FSR to point to one of the virtual registers will not result in successful operations. As a specific case, assume that FSR0H:FSR0L contains the address of INDF1. Attempts to read the value of the INDF1 using INDF0 as an operand will return 00h. Attempts to write to INDF1 using INDF0 as the operand will result in a `NOP`.

On the other hand, using the virtual registers to write to an FSR pair may not occur as planned. In these cases, the value will be written to the FSR pair but without any incrementing or decrementing. Thus, writing to either the INDF2 or POSTDEC2 register will write the same value to the FSR2H:FSR2L.

Since the FSRs are physical registers mapped in the SFR space, they can be manipulated through all direct operations. Users need to proceed cautiously when working on these registers, particularly if their code uses Indirect Addressing.

Similarly, operations by Indirect Addressing are generally permitted on all other SFRs. Users need to exercise the appropriate caution that they do not inadvertently change settings that might affect the operation of the device.

9.6 Data Memory and the Extended Instruction Set

Enabling the PIC18 extended instruction set ($\overline{\text{XINST}}$ Configuration bit = `1`) significantly changes certain aspects of data memory and its addressing. Specifically, the use of the Access Bank for many of the core PIC18 instructions is different; this is due to the introduction of a new Addressing mode for the data memory space.

What does not change is just as important. The size of the data memory space is unchanged, as well as its linear addressing. The SFR map remains the same. Core PIC18 instructions can still operate in both Direct and Indirect