



MICROCHIP

Section 6. Memory Organization

HIGHLIGHTS

This section of the manual contains the following major topics:

6.1	Introduction	6-2
6.2	Program Memory Organization	6-2
6.3	Data Memory Organization	6-8
6.4	Initialization	6-14
6.5	Design Tips	6-16
6.6	Related Application Notes	6-17
6.7	Revision History	6-18

PICmicro MID-RANGE MCU FAMILY

6.1 Introduction

There are two memory blocks in the Section 6. Memory Organization; program memory and data memory. Each block has its own bus, so that access to each block can occur during the same oscillator cycle.

The data memory can further be broken down into General Purpose RAM and the Special Function Registers (SFRs). The operation of the SFRs that control the “core” are described here. The SFRs used to control the peripheral modules are described in the section discussing each individual peripheral module.

6.2 Program Memory Organization

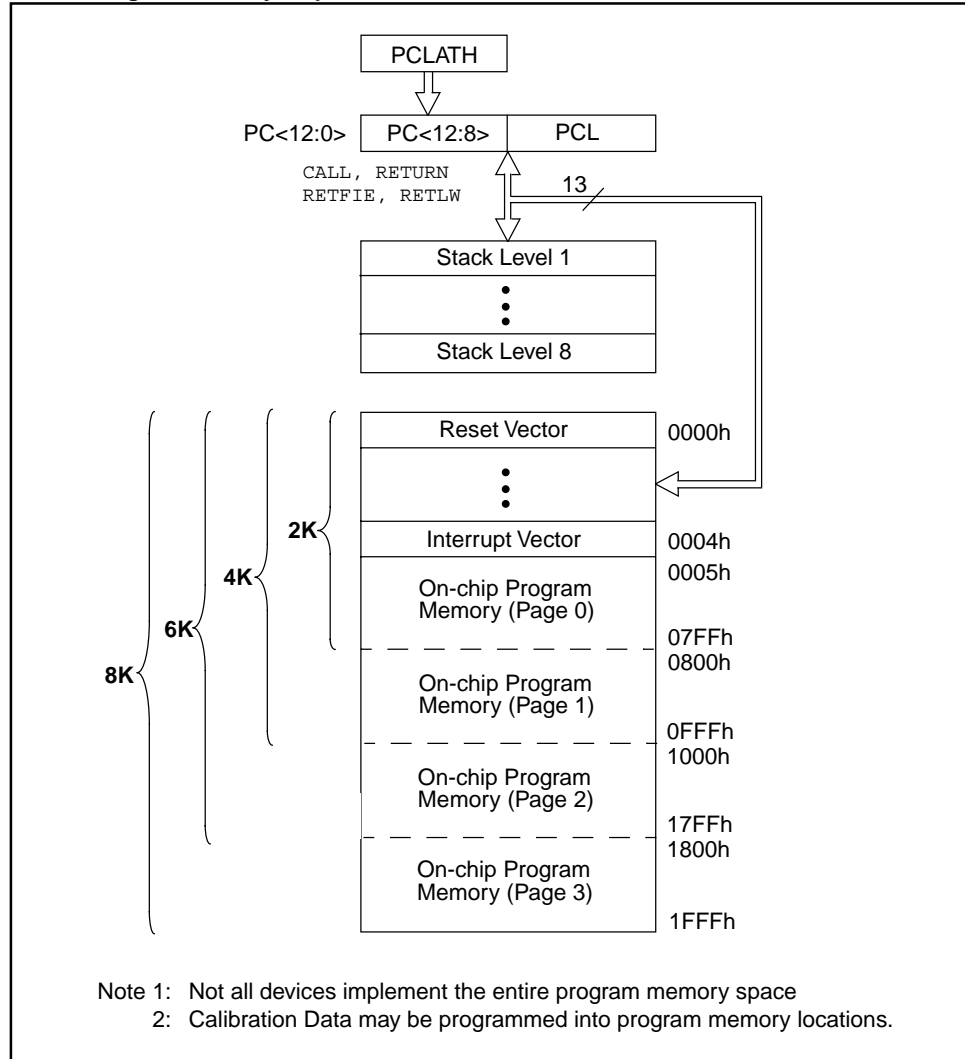
Mid-Range MCU devices have a 13-bit program counter capable of addressing an 8K x 14 program memory space. The width of the program memory bus (instruction word) is 14-bits. Since all instructions are a single word, a device with an 8K x 14 program memory has space for 8K of instructions. This makes it much easier to determine if a device has sufficient program memory for a desired application.

This program memory space is divided into four pages of 2K words each (0h - 7FFh, 800h - FFFh, 1000h - 17FFh, and 1800h - 1FFFh). [Figure 6-1](#) shows the program memory map as well as the 8 level deep hardware stack. Depending on the device, only a portion of this memory may be implemented. Please refer to the device data sheet for the available memory.

To jump between the program memory pages, the high bits of the Program Counter (PC) must be modified. This is done by writing the desired value into a SFR called PCLATH (**P**rogram **C**ounter **L**atch **H**igh). If sequential instructions are executed, the program counter will cross the page boundaries without any user intervention. For devices that have less than 8K words, accessing a location above the physically implemented address will cause a wraparound. That is, in a 4K-word device accessing 17FFh actually addresses 7FFh. 2K-word devices (or less) do not require paging.

Section 6. Memory Organization

Figure 6-1: Architectural Program Memory Map and Stack



PICmicro MID-RANGE MCU FAMILY

6.2.1 Reset Vector

On any device, a reset forces the Program Counter (PC) to address 0h. We call this address the “Reset Vector Address” since this is the address that program execution will branch to when a device reset occurs.

Any reset will also clear the contents of the PCLATH register. This means that any branch at the Reset Vector Address (0h) will jump to that location in PAGE0 of the program memory.

6.2.2 Interrupt Vector

When an interrupt is acknowledged the PC is forced to address 0004h. We call this the “Interrupt Vector Address”. When the PC is forced to the interrupt vector, the PCLATH register is not modified. Once in the service interrupt routine (ISR), this means that before any write to the PC, the PCLATH register should be written with the value that will specify the desired location in program memory. Before the PCLATH register is modified by the Interrupt Service Routine (ISR) the contents of the PCLATH may need to be saved, so it can be restored before returning from the ISR.

6.2.3 Calibration Information

Some devices have calibration information stored in their program memory. This information is programmed by Microchip when the device is under final test. The use of these values allows the application to achieve better results. The calibration information is typically at the end of program memory, and is implemented as a RETLW instruction with the literal value being the specified calibration information.

<p>Note: For windowed devices, write down all calibration values BEFORE erasing. This allows the device's calibration values to be restored when the device is re-programmed. When possible writing the values on the package is recommended.</p>

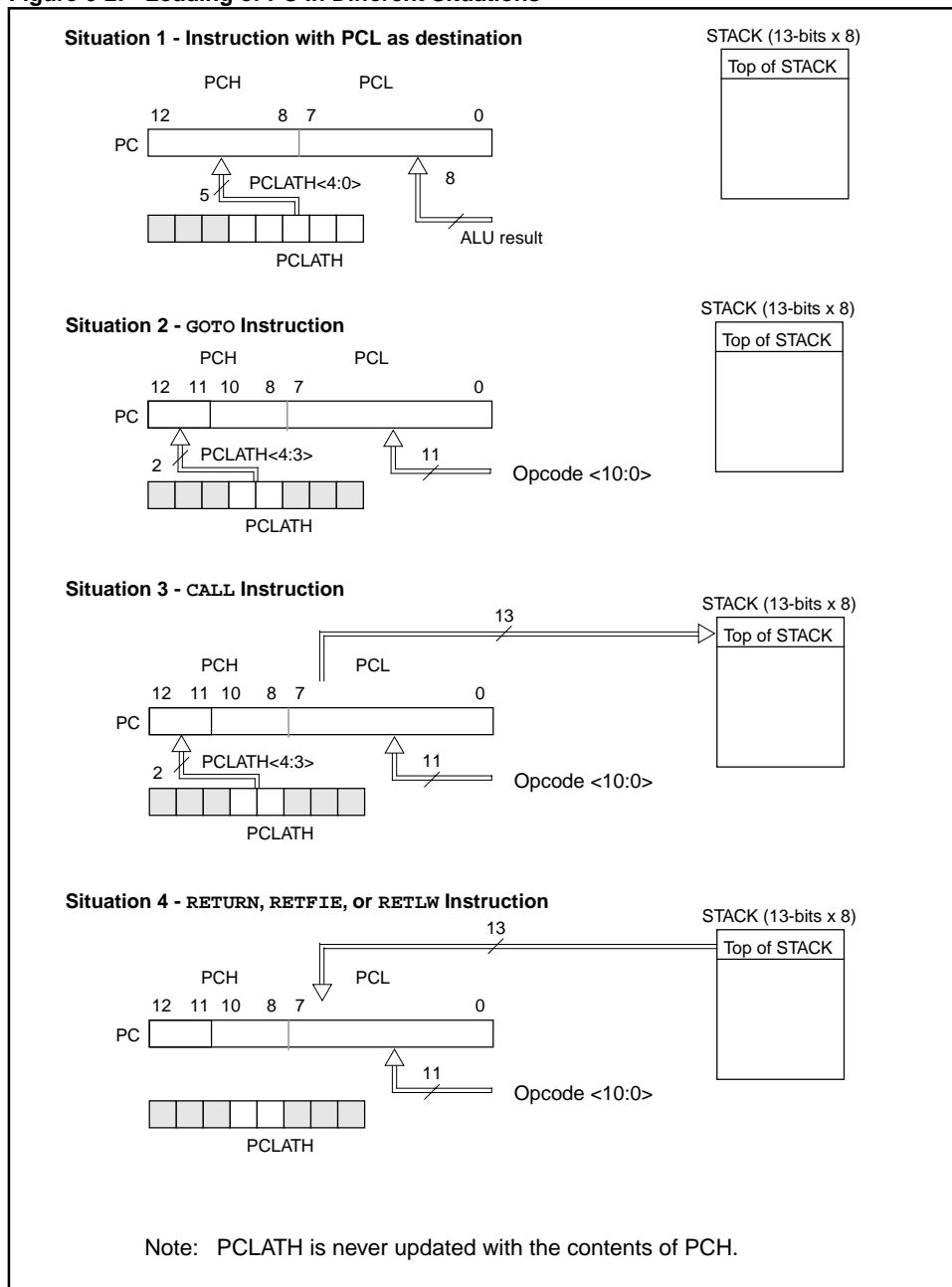
Section 6. Memory Organization

6.2.4 Program Counter (PC)

The program counter (PC) specifies the address of the instruction to fetch for execution. The PC is 13-bits wide. The low byte is called the PCL register. This register is readable and writable. The high byte is called the PCH register. This register contains the PC<12:8> bits and is not directly readable or writable. All updates to the PCH register go through the PCLATH register.

Figure 6-2 shows the four situations for the loading of the PC. Situation 1 shows how the PC is loaded on a write to PCL (PCLATH<4:0> → PCH). Situation 2 shows how the PC is loaded during a GOTO instruction (PCLATH<4:3> → PCH). Situation 3 shows how the PC is loaded during a CALL instruction (PCLATH<4:3> → PCH), with the PC loaded (PUSHed) onto the Top of Stack. Situation 4 shows how the PC is loaded during one of the return instructions where the PC loaded (POPped) from the Top of Stack.

Figure 6-2: Loading of PC In Different Situations



PICmicro MID-RANGE MCU FAMILY

6.2.4.1 Computed GOTO

A computed GOTO is accomplished by adding an offset to the program counter (`ADDWF PCL`). When doing a table read using a computed GOTO method, care should be exercised if the table location crosses a PCL memory boundary (each 256 byte block).

Note: Any write to the Program Counter (PCL), will cause the lower five bits of the PCLATH to be loaded into PCH.

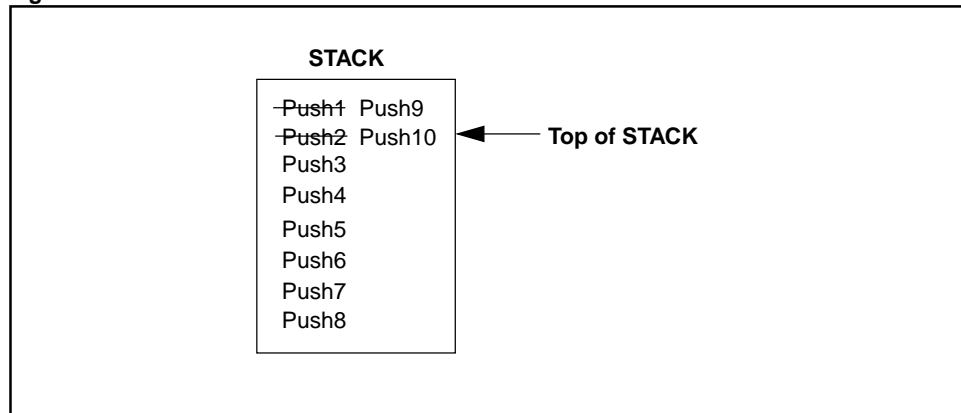
6.2.5 Stack

The stack allows a combination of up to 8 program calls and interrupts to occur. The stack contains the return address from this branch in program execution.

Mid-Range MCU devices have an 8-level deep x 13-bit wide hardware stack. The stack space is not part of either program or data space and the stack pointer is not readable or writable. The PC is PUSHed onto the stack when a `CALL` instruction is executed or an interrupt causes a branch. The stack is POPed in the event of a `RETURN`, `RETLW` or a `RETFIE` instruction execution. PCLATH is not modified when the stack is PUSHed or POPed.

After the stack has been PUSHed eight times, the ninth push overwrites the value that was stored from the first push. The tenth push overwrites the second push (and so on). An example of the overwriting of the stack is shown in [Figure 6-3](#).

Figure 6-3: Stack Modification



Note 1: There are no status bits to indicate stack overflow or stack underflow conditions.

Note 2: There are no instructions/mnemonics called PUSH or POP. These are actions that occur from the execution of the `CALL`, `RETURN`, `RETLW`, and `RETFIE` instructions, or the vectoring to an interrupt address.

Section 6. Memory Organization

6.2.6 Program Memory Paging

Some devices have program memory sizes greater than 2K words, but the `CALL` and `GOTO` instructions only have a 11-bit address range. This 11-bit address range allows a branch within a 2K program memory page size. To allow `CALL` and `GOTO` instructions to address the entire 1K program memory address range, there must be another two bits to specify the program memory page. These paging bits come from the `PCLATH<4:3>` bits (Figure 6-2). When doing a `CALL` or `GOTO` instruction, the user must ensure that page bits (`PCLATH<4:3>`) are programmed so that the desired program memory page is addressed (Figure 6-2). When one of the return instructions is executed, the entire 13-bit PC is POPed from the stack. Therefore, manipulation of the `PCLATH<4:3>` is not required for the return instructions.

Note: Devices with program memory sizes 2K words and less, ignore both paging bits (`PCLATH<4:3>`), which are used to access program memory when more than one page is available. The use of `PCLATH<4:3>` as general purpose read/write bits (for these devices) is not recommended since this may affect upward compatibility with future products.

Devices with program memory sizes between 2K words and 4K words, ignore the paging bit (`PCLATH<4>`), which is used to access program memory pages 2 and 3 (1000h - 1FFFh). The use of `PCLATH<4>` as a general purpose read/write bit (for these devices) is not recommended since this may affect upward compatibility with future products.

Example 6-1 shows the calling of a subroutine in page 1 of the program memory. This example assumes that `PCLATH` is saved and restored by the interrupt service routine (if interrupts are used).

Example 6-1: Call of a Subroutine in Page1 from Page0

```
ORG 0x500
BSF   PCLATH,3 ; Select Page1 (800h-FFFh)
CALL  SUB1_P1 ; Call subroutine in Page1 (800h-FFFh)
:
:
ORG 0x900
SUB1_P1: ; called subroutine Page1 (800h-FFFh)
:
RETURN ; return to Call subroutine in Page0 (000h-7FFh)
;
```

PICmicro MID-RANGE MCU FAMILY

6.3 Data Memory Organization

Data memory is made up of the Special Function Registers (SFR) area, and the General Purpose Registers (GPR) area. The SFRs control the operation of the device, while GPRs are the general area for data storage and scratch pad operations.

The data memory is banked for both the GPR and SFR areas. The GPR area is banked to allow greater than 96 bytes of general purpose RAM to be addressed. SFRs are for the registers that control the peripheral and core functions. Banking requires the use of control bits for bank selection. These control bits are located in the STATUS Register (STATUS<7:5>). [Figure 6-5](#) shows one of the data memory map organizations, this organization is device dependent.

To move values from one register to another register, the value must pass through the W register. This means that for all register-to-register moves, two instruction cycles are required.

The entire data memory can be accessed either directly or indirectly. Direct addressing may require the use of the RP1:RP0 bits. Indirect addressing requires the use of the File Select Register (FSR). Indirect addressing uses the Indirect Register Pointer (IRP) bit of the STATUS register for accesses into the Bank0 / Bank1 or the Bank2 / Bank3 areas of data memory.

6.3.1 General Purpose Registers (GPR)

Some Mid-Range MCU devices have banked memory in the GPR area. GPRs are not initialized by a Power-on Reset and are unchanged on all other resets.

The register file can be accessed either directly, or using the File Select Register FSR, indirectly. Some devices have areas that are shared across the data memory banks, so a read / write to that area will appear as the same location (value) regardless of the current bank. We refer to this area as the Common RAM.

6.3.2 Special Function Registers (SFR)

The SFRs are used by the CPU and Peripheral Modules for controlling the desired operation of the device. These registers are implemented as static RAM.

The SFRs can be classified into two sets, those associated with the “core” function and those related to the peripheral functions. Those registers related to the “core” are described in this section, while those related to the operation of the peripheral features are described in the section of that peripheral feature.

All Mid-Range MCU devices have banked memory in the SFR area. Switching between these banks requires the RP0 and RP1 bits in the STATUS register to be configured for the desired bank. Some SFRs are initialized by a Power-on Reset and other resets, while other SFRs are unaffected.

Note: The Special Function Register (SFR) Area may have General Purpose Registers (GPRs) mapped in these locations.
--

The register file can be accessed either directly, or using the File Select Register FSR, indirectly.

Section 6. Memory Organization

6.3.3 Banking

The data memory is partitioned into four banks. Each bank contains General Purpose Registers and Special Function Registers. Switching between these banks requires the RP0 and RP1 bits in the STATUS register to be configured for the desired bank when using direct addressing. The IRP bit in the STATUS register is used for indirect addressing.

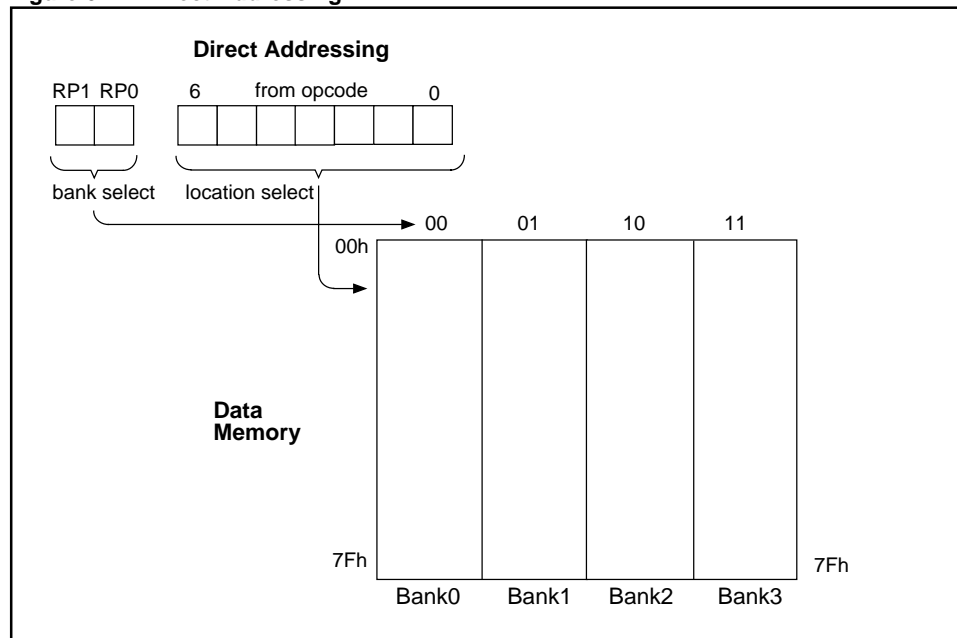
Table 6-1: Direct and Indirect Addressing of Banks

Accessed Bank	Direct (RP1:RP0)	Indirect (IRP)
0	0 0	0
1	0 1	
2	1 0	1
3	1 1	

Each Bank extends up to 7Fh (128 bytes). The lower locations of each bank are reserved for the Special Function Registers. Above the Special Function Registers are General Purpose Registers. All data memory is implemented as static RAM. All Banks may contain special function registers. Some "high use" special function registers from Bank0 are mirrored in the other banks for code reduction and quicker access.

Through the evolution of the products, there are a few variations in the layout of the Data Memory. The data memory organization that will be the standard for all new devices is shown in [Figure 6-5](#). This Memory map has the last 16-bytes mapped across all memory banks. This is to reduce the software overhead for context switching. The registers in **bold** will be in every device. The other registers are peripheral dependent. Not every peripheral's registers are shown, because some file addresses have a different registers from those shown. As with all the figures, tables, and specifications presented in this reference guide, verify the details with the device specific data sheet.

Figure 6-4: Direct Addressing



PICmicro MID-RANGE MCU FAMILY

Figure 6-5: Register File Map

File Address	File Address	File Address	File Address
INDF 00h	INDF 80h	INDF 100h	INDF 180h
TMR0 01h	OPTION_REG 81h	TMR0 101h	OPTION_REG 181h
PCL 02h	PCL 82h	PCL 102h	PCL 182h
STATUS 03h	STATUS 83h	STATUS 103h	STATUS 183h
FSR 04h	FSR 84h	FSR 104h	FSR 184h
PORTA 05h	TRISA 85h		
PORTB 06h	TRISB 86h	PORTB 106h	TRISB 186h
PORTC 07h	TRISC 87h	PORTF 107h	TRISF 187h
PORTD 08h	TRISD 88h	PORTG 108h	TRISG 188h
PORTE 09h	TRISE 89h		
PCLATH 0Ah	PCLATH 8Ah	PCLATH 10Ah	PCLATH 18Ah
INTCON 0Bh	INTCON 8Bh	INTCON 10Bh	INTCON 18Bh
PIR1 0Ch	PIE1 8Ch		
PIR2 0Dh	PIE2 8Dh		
TMR1L 0Eh	PCON 8Eh		
TMR1H 0Fh	OSCCAL 8Fh		
T1CON 10h			
TMR2 11h			
T2CON 12h	PR2 92h		
SSPBUF 13h	SSPADD 93h		
SSPCON 14h	SSPATAT 94h		
CCPR1L 15h			
CCPR1H 16h			
CCP1CON 17h			
RCSTA 18h	TXSTA 98h		
TXREG 19h	SPBRG 99h		
RCREG 1Ah			
CCPR2L 1Bh			
CCPR2H 1Ch			
CCP2CON 1Dh			
ADRES 1Eh			
ADCON0 1Fh	ADCON1 9Fh		
General Purpose Registers ⁽²⁾	ADCON1 9Fh		
	20h	A0h	120h
	7Fh	EFh	16Fh
	General Purpose Registers ⁽³⁾	General Purpose Registers ⁽³⁾	General Purpose Registers ⁽³⁾
	Mapped in Bank0 70h - 7Fh ⁽⁴⁾	Mapped in Bank0 70h - 7Fh ⁽⁴⁾	Mapped in Bank0 70h - 7Fh ⁽⁴⁾
	FFh	17Fh	1FFh
Bank0	Bank1	Bank2 ⁽⁵⁾	Bank3 ⁽⁵⁾

Note 1: Registers in **BOLD** will be present in every device.
 Note 2: Not all locations may be implemented. Unimplemented locations will read as '0'.
 Note 3: These locations may not be implemented. Depending on the device, accesses to the unimplemented locations operate differently. Please refer to the specific device data sheet for details.
 Note 4: Some device do not map these registers into Bank0. In devices where these registers are mapped into Bank0, these registers are referred to as common RAM
 Note 5: Some devices may not implement these banks. Locations in unimplemented banks will read as '0'.
 Note 6: General Purpose Registers (GPRs) may be located in the Special Function Register (SFR) area.

Section 6. Memory Organization

The map in [Figure 6-6](#) shows the register file memory map of some 18-pin devices. Unimplemented registers will read as '0'.

Figure 6-6: Register File Map

File Address		File Address	
INDF	00h	INDF	80h
TMR0	01h	OPTION_REG	81h
PCL	02h	PCL	82h
STATUS	03h	STATUS	83h
FSR	04h	FSR	84h
PORTA	05h	TRISA	85h
PORTB	06h	TRISB	86h
	07h	PCON	87h
ADCON0 / EEDATA (2)	08h	ADCON1 / EECON1 (2)	88h
ADRES / EEADR (2)	09h	ADRES / EECON2 (2)	89h
PCLATH	0Ah	PCLATH	8Ah
INTCON	0Bh	INTCON	8Bh
General Purpose Registers (3)	0Ch		8Ch
	7Fh	General Purpose Registers (4)	FFh
Bank0		Bank1	

Note 1: Registers in **BOLD** will be present in every device.
 Note 2: These registers may not be implemented, or are implemented as other registers in some devices.
 Note 3: Not all locations may be implemented. Unimplemented locations will read as '0'.
 Note 4: These locations are unimplemented in Bank1. Access to these unimplemented locations will access the corresponding Bank0 register.

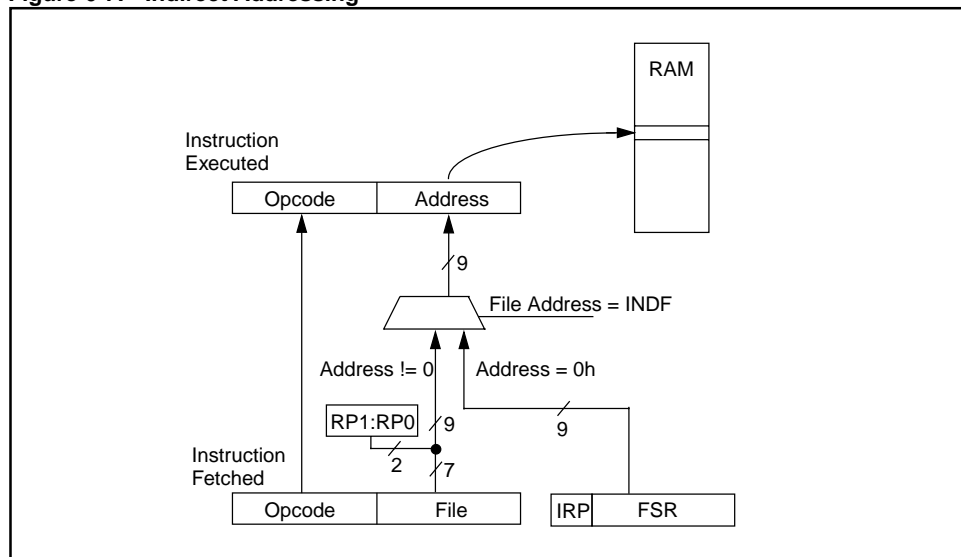
PICmicro MID-RANGE MCU FAMILY

6.3.4 Indirect Addressing, INDF, and FSR Registers

Indirect addressing is a mode of addressing data memory where the data memory address in the instruction is not fixed. An SFR register is used as a pointer to the data memory location that is to be read or written. Since this pointer is in RAM, the contents can be modified by the program. This can be useful for data tables in the data memory. Figure 6-7 shows the operation of indirect addressing. This shows the moving of the value to the data memory address specified by the value of the FSR register.

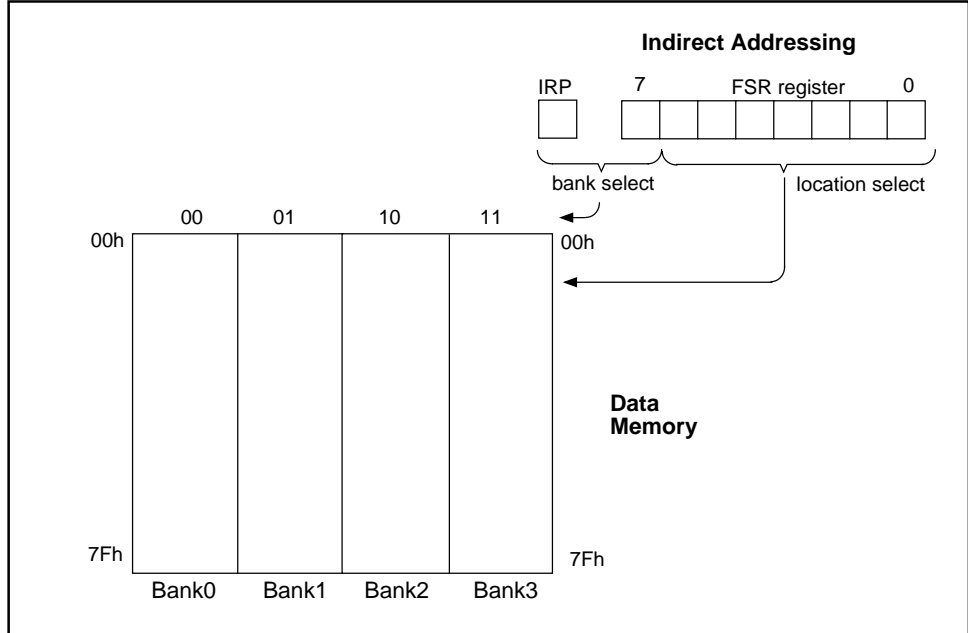
Indirect addressing is possible by using the INDF register. Any instruction using the INDF register actually accesses the register pointed to by the File Select Register, FSR. Reading the INDF register itself indirectly (FSR = '0') will read 00h. Writing to the INDF register indirectly results in a no-operation (although status bits may be affected). An effective 9-bit address is generated by the concatenation of the IRP bit (STATUS<7>) with the 8-bit FSR register, as shown in Figure 6-8.

Figure 6-7: Indirect Addressing



Section 6. Memory Organization

Figure 6-8: Indirect Addressing



Example 6-2 shows a simple use of indirect addressing to clear RAM (locations 20h-2Fh) in a minimum number of instructions. A similar concept could be used to move a defined number of bytes (block) of data to the USART transmit register (TXREG). The starting address of the block of data to be transmitted could easily be modified by the program.

Example 6-2: Indirect Addressing

```

        BCF     STATUS, IRP    ; Indirect addressing Bank0/1
        MOVLW  0x20           ; Initialize pointer to RAM
        MOVWF  FSR            ;
NEXT    CLRF   INDF           ; Clear INDF register
        INCF   FSR,F          ; Inc pointer
        BTFSS FSR,4           ; All done?
        GOTO   NEXT           ; NO, clear next
CONTINUE :                    ;
        :                     ; YES, continue
    
```

PICmicro MID-RANGE MCU FAMILY

6.4 Initialization

[Example 6-3](#) shows how the bank switching occurs for Direct addressing, while [Example 6-4](#) shows some code to do initialization (clearing) of General Purpose RAM.

Example 6-3: Bank Switching

```
CLRF  STATUS          ; Clear STATUS register (Bank0)
:
BSF   STATUS, RP0     ; Bank1
:
BCF   STATUS, RP0     ; Bank0
:
MOVLW 0x60           ; Set RP0 and RP1 in STATUS register, other
XORWF STATUS, F      ; bits unchanged (Bank3)
:
BCF   STATUS, RP0     ; Bank2
:
BCF   STATUS, RP1     ; Bank0
```

Section 6. Memory Organization

Example 6-4: RAM Initialization

```
        CLRf    STATUS      ; Clear STATUS register (Bank0)
        MOVLW  0x20        ; 1st address (in bank) of GPR area
        MOVWF  FSR         ; Move it to Indirect address register
Bank0_LP
        CLRf    INDF0      ; Clear GPR at address pointed to by FSR
        INCF   FSR         ; Next GPR (RAM) address
        BTFSS  FSR, 7     ; End of current bank? (FSR = 80h, C = 0)
        GOTO   Bank0_LP   ; NO, clear next location
;
; Next Bank (Bank1)
; (** ONLY REQUIRED IF DEVICE HAS A BANK1 **)
;
        MOVLW  0xA0        ; 1st address (in bank) of GPR area
        MOVWF  FSR         ; Move it to Indirect address register
Bank1_LP
        CLRf    INDF0      ; Clear GPR at address pointed to by FSR
        INCF   FSR         ; Next GPR (RAM) address
        BTFSS  STATUS, C  ; End of current bank? (FSR = 00h, C = 1)
        GOTO   Bank1_LP   ; NO, clear next location
;
; Next Bank (Bank2)
; (** ONLY REQUIRED IF DEVICE HAS A BANK2 **)
;
        BSF    STATUS, IRP ; Select Bank2 and Bank3
                          ; for Indirect addressing
        MOVLW  0x20        ; 1st address (in bank) of GPR area
        MOVWF  FSR         ; Move it to Indirect address register
Bank2_LP
        CLRf    INDF0      ; Clear GPR at address pointed to by FSR
        INCF   FSR         ; Next GPR (RAM) address
        BTFSS  FSR, 7     ; End of current bank? (FSR = 80h, C = 0)
        GOTO   Bank2_LP   ; NO, clear next location
;
; Next Bank (Bank3)
; (** ONLY REQUIRED IF DEVICE HAS A BANK3 **)
;
        MOVLW  0xA0        ; 1st address (in bank) of GPR area
        MOVWF  FSR         ; Move it to Indirect address register
Bank3_LP
        CLRf    INDF0      ; Clear GPR at address pointed to by FSR
        INCF   FSR         ; Next GPR (RAM) address
        BTFSS  STATUS, C  ; End of current bank? (FSR = 00h, C = 1)
        GOTO   Bank3_LP   ; NO, clear next location
;
; YES, All GPRs (RAM) is cleared
```

PICmicro MID-RANGE MCU FAMILY

6.5 Design Tips

Question 1: *Program execution seems to get lost.*

Answer 1:

When a device with more than 2K words of program memory is used, the calling of subroutines may require that the PCLATH register be loaded prior to the `CALL` (or `GOTO`) instruction to specify the correct program memory page that the routine is located on. The following instructions will correctly load PCLATH register, regardless of the program memory location of the label `SUB_1`.

```
        MOVLW    HIGH (SUB_1)    ; Select Program Memory Page of
        MOVWF   PCLATH          ; Routine.
        CALL    SUB_1           ; Call the desired routine
        :
        :
SUB_1   :                       ; Start of routine
        :
        RETURN                  ; Return from routine
```

Question 2: *I need to initialize RAM to '0's. What is an easy way to do that?*

Answer 2:

[Example 6-4](#) shows this. If the device you are using does not use all 4 data memory banks, some of the code may be removed.

Section 6. Memory Organization

6.6 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 Mid-range MCU family (that is they may be written for the Base-Line, or High-End families), but the concepts are pertinent, and could be used (with modification and possible limitations). The current application notes related to memory are:

Title	Application Note #
Implementing a Table Read	AN556

PICmicro MID-RANGE MCU FAMILY

6.7 Revision History

Revision A

This is the initial released revision of the Memory Organization description.