# Directives & Memory Spaces

Dr. Farid Farahmand

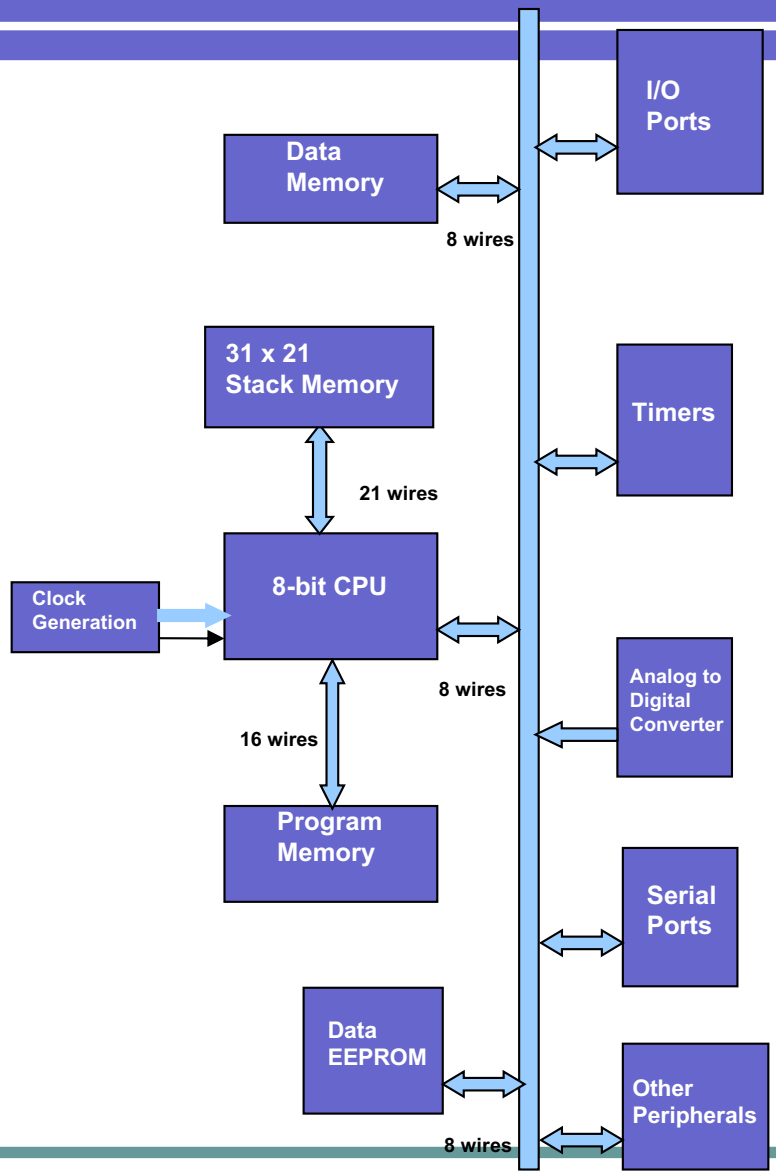
Updated: 2/18/2019

# Memory Types

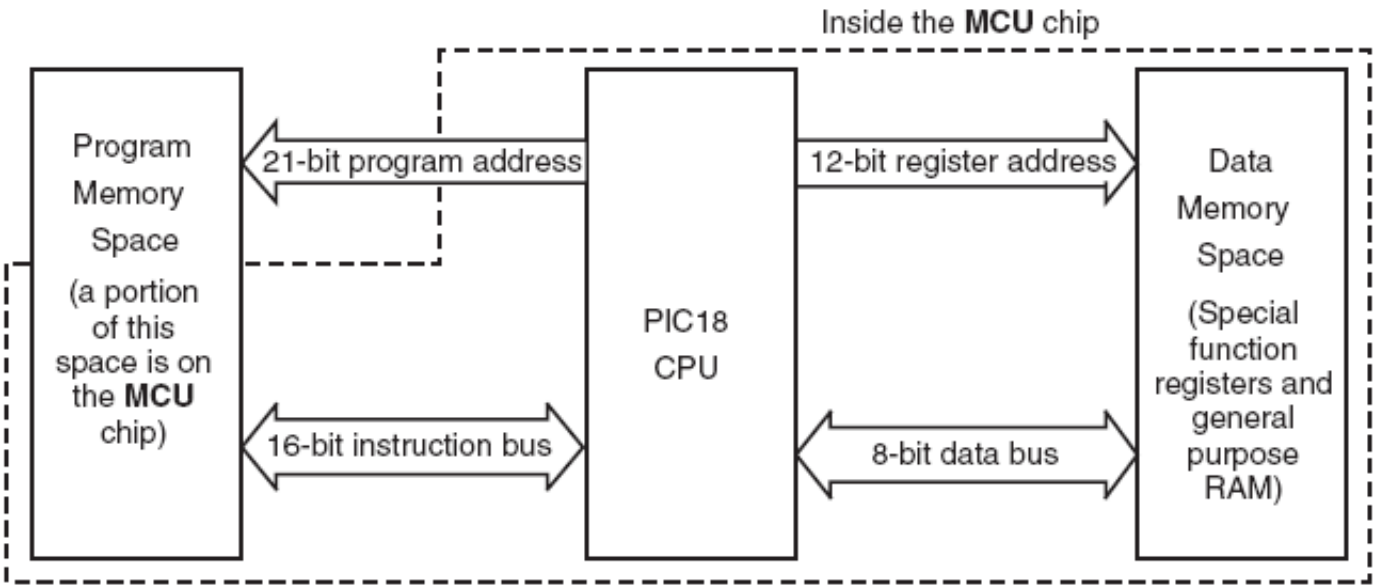
- Program Memory
- Data Memory
- Stack

| Device | Program Memory | | Data Memory | |
|---|---|---|---|---|
| | Flash (bytes) | # Single-Word Instructions | SRAM (bytes) | EEPROM (bytes) |
| PIC18F23K20 | 8K | 4096 | 512 | 256 |
| PIC18F24K20 | 16K | 8192 | 768 | 256 |
| PIC18F25K20 | 32K | 16384 | 1536 | 256 |
| PIC18F26K20 | 64k | 32768 | 3936 | 1024 |
| PIC18F43K20 | 8K | 4096 | 512 | 256 |
| PIC18F44K20 | 16K | 8192 | 768 | 256 |
| PIC18F45K20 | 32K | 16384 | 1536 | 256 |
| PIC18F46K20 | 64k | 32768 | 3936 | 1024 |

# Internal PIC18 Architecture

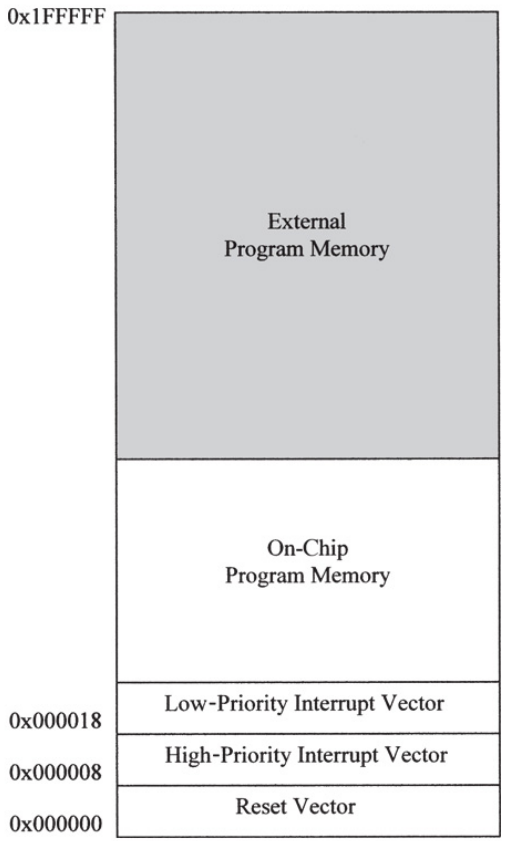


# PIC18 Memory Space



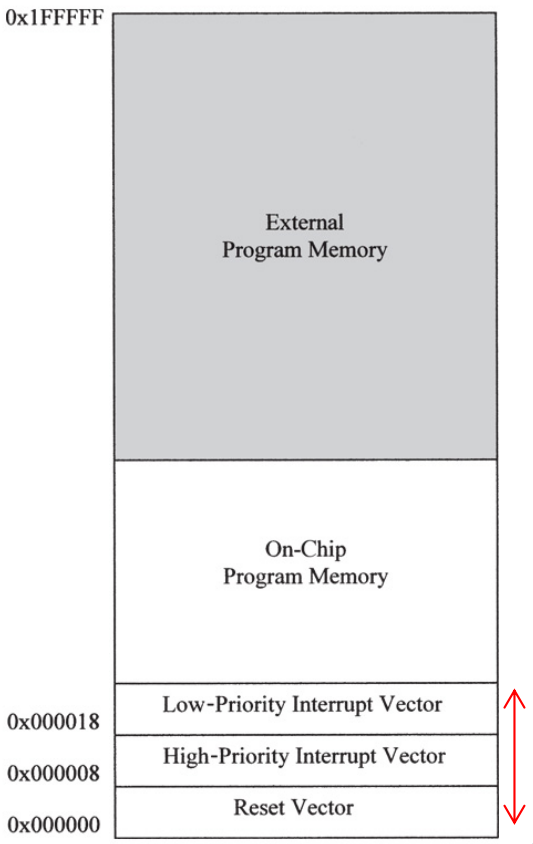
# Program Memory

- Program memory addresses are 21-bit address starting at location 0x000000.
- On-Chip Program memory or Program ROM holds the program
- The PROM is Flash



# Program Memory

- There are three important memory locations in the program memory. 0x0000, 0x0008, and 0x0018 called vectors.
  - Generally the GOTO instruction in assembly language is placed at a vector location.
  - A vector is an address that is accessed when the reset or interrupt occurs.
- Vector locations:
  - The reset vector is 0x0000,
  - The high priority interrupt vector is 0x0008,
  - The low priority interrupt vector is 0x0018.



# Accessing Program Memory

Let’s Review Directives First…

# Directives

- Special commands to the assembler
  - May or may not generate machine code
- Categories by their function
- Programming directives
  - Object file directives
  - Control Directives
  - List Directives
  - Data Directives

# Data Directives

Describe data
ASCII data can be stored in memory using declare byte (DB) or DATA

| *Directive* | *Comment* |
|---|---|
| **__badram** | Identifies unimplemented RAM |
| **__badrom** | Identifies unimplemented ROM |
| **__config** | Sets processor configuration bits |
| **config** | Sets processor configuration bits for the PIC18 family |
| **__idloca** | Sets processor ID locations |
| **__maxram** | Sets maximum RAM |
| **__maxrom** | Sets maximum ROM |
| **cblock** | Defines a block of constant data |
| **da** | Stores strings in the program memory (PIC12/16) |
| **data** | Creates numeric and text data |
| **db** | Declares bytes |
| **de** | Declares EEPROM data |
| **dt** | Defines tables (PIC16/12) |
| **dw** | Declares words |
| **endc** | Ends automatic block constants |
| **fill** | Fills memory with a contant |
| **res** | Reserves memory |

# Accessing Program Memory Using Data Directives - Example

DE "Test Data" ; Declaring data in EEPROM

EEPROM

| Address | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F | ASCII |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00 | 54 | 65 | 73 | 74 | 20 | 44 | 61 | 74 | 61 | 00 | FF | FF | FF | FF | FF | FF | Test Dat a..... |
| 10 | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | ..... ..... |

```
EEPROM_MEM    CODE    0xF00000
              DE    "This is for my EEPROM!", 0     ;C-style null string
```

# Accessing Program Memory Using Data Directives - Example

Program Memory

| Line | Address | Opcode | Disassembly |
|---|---|---|---|
| 13 | 0018 | FFFF | NOP |
| 14 | 001A | FFFF | NOP |
| 15 | 001C | FFFF | NOP |
| 16 | 001E | FFFF | NOP |
| 17 | 0020 | 2301 | ADDWFC 0x1, |
| 18 | 0022 | 8899 | BSF 0xf99, 0x |
| 19 | 0024 | 794D | BTG 0x4d, 0x |
| 20 | 0026 | 4E20 | DCFSNZ 0x20, |
| 21 | 0028 | 6D61 | NEGF 0x61, B |
| 22 | 002A | 2065 | ADDWFC 0x65, |
| 23 | 002C | 7369 | BTG 0x69, 0x |
| 24 | 002E | 4620 | RLNCF 0x20, |
| 25 | 0030 | 7261 | BTG 0x61, 0x |
| 26 | 0032 | 6469 | CPFSGT 0x69, |
| 65 | 0080 | 0E00 | MOVLW 0 |
| 66 | 0082 | 0000 | NOP |
| 67 | 0084 | 0322 | MULWF 0x22, |
| 68 | 0086 | BBAA | BTFSC 0xaa, |
| 69 | 0088 | 0003 | SLEEP |
| 70 | 008A | FFFF | NOP |
| 71 | 008C | FFFF | NOP |

ASCII of "My"

Multiple ORGs Can be used!

```
ORG     0x20            ;Begin assembly at 0000H

COUNTER_REG EQU 0x01
DW  0X2301, 0X8899

DB "My Name is Farid"
DB 'x'
DB  0x22, 0x03
DB  0xAA, 0XBB

    ORG 0x80

    MOVLW   0X0
    NOP

DB  0x22, 0x03
DB  0xAA, 0XBB

SLEEP                   ;End of program, power down
END                     ;End of assembly
```

Example of DB & DW

# List Directives

**Control listing process**
**Example:**

**LIST P=18F4520, F=INHX32 ;directive to define processor and file format**

| *Directive* | *Comment* |
|---|---|
| **error** | Issues an error message |
| **errorlevel** | Sets the error message level |
| **list** | Sets the processor and output file type and other defaults of a program listing |
| **messg** | Sets a user message |
| **nolist** | Disables listing |
| **page** | Inserts a new page in the listing |
| **space** | Inserts blank lines in a listing |
| **subtitle** | Specifies a program subtitle |
| **title** | Specifies a program title |

# Control Directives

**Control the assembly at the time of link process**

**#include <P18F4520.INC>  ;processor specific variable definitions**

| *Directive* | *Comment* |
|---|---|
| **#define** | Defines a text substitution label |
| **#include** | Includes a source file |
| **#undefine** | Removes a substitution label |
| **Constant** | Declares a symbol constant |
| **End** | Ends the program file (required) |
| **Equ** | Equates a constant |
| **Org** | Sets the origin of a block |
| **Processor** | Selects the processor type |
| **Radix** | Specifies the default radix |
| **Set** | Defines an assembler variable |
| **Variable** | Declares a symbol variable |

# ASCII TABLE

| Dec | Hx | Oct | Char | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 000 | NUL (null) | 32 | 20 | 040 | &#32; | Space | 64 | 40 | 100 | &#64; | @ | 96 | 60 | 140 | &#96; | ` |
| 1 | 1 | 001 | SOH (start of heading) | 33 | 21 | 041 | &#33; | ! | 65 | 41 | 101 | &#65; | A | 97 | 61 | 141 | &#97; | a |
| 2 | 2 | 002 | STX (start of text) | 34 | 22 | 042 | &#34; | " | 66 | 42 | 102 | &#66; | B | 98 | 62 | 142 | &#98; | b |
| 3 | 3 | 003 | ETX (end of text) | 35 | 23 | 043 | &#35; | # | 67 | 43 | 103 | &#67; | C | 99 | 63 | 143 | &#99; | c |
| 4 | 4 | 004 | EOT (end of transmission) | 36 | 24 | 044 | &#36; | $ | 68 | 44 | 104 | &#68; | D | 100 | 64 | 144 | &#100; | d |
| 5 | 5 | 005 | ENQ (enquiry) | 37 | 25 | 045 | &#37; | % | 69 | 45 | 105 | &#69; | E | 101 | 65 | 145 | &#101; | e |
| 6 | 6 | 006 | ACK (acknowledge) | 38 | 26 | 046 | &#38; | & | 70 | 46 | 106 | &#70; | F | 102 | 66 | 146 | &#102; | f |
| 7 | 7 | 007 | BEL (bell) | 39 | 27 | 047 | &#39; | ' | 71 | 47 | 107 | &#71; | G | 103 | 67 | 147 | &#103; | g |
| 8 | 8 | 010 | BS (backspace) | 40 | 28 | 050 | &#40; | ( | 72 | 48 | 110 | &#72; | H | 104 | 68 | 150 | &#104; | h |
| 9 | 9 | 011 | TAB (horizontal tab) | 41 | 29 | 051 | &#41; | ) | 73 | 49 | 111 | &#73; | I | 105 | 69 | 151 | &#105; | i |
| 10 | A | 012 | LF (NL line feed, new line) | 42 | 2A | 052 | &#42; | * | 74 | 4A | 112 | &#74; | J | 106 | 6A | 152 | &#106; | j |
| 11 | B | 013 | VT (vertical tab) | 43 | 2B | 053 | &#43; | + | 75 | 4B | 113 | &#75; | K | 107 | 6B | 153 | &#107; | k |
| 12 | C | 014 | FF (NP form feed, new page) | 44 | 2C | 054 | &#44; | , | 76 | 4C | 114 | &#76; | L | 108 | 6C | 154 | &#108; | l |
| 13 | D | 015 | CR (carriage return) | 45 | 2D | 055 | &#45; | - | 77 | 4D | 115 | &#77; | M | 109 | 6D | 155 | &#109; | m |
| 14 | E | 016 | SO (shift out) | 46 | 2E | 056 | &#46; | . | 78 | 4E | 116 | &#78; | N | 110 | 6E | 156 | &#110; | n |
| 15 | F | 017 | SI (shift in) | 47 | 2F | 057 | &#47; | / | 79 | 4F | 117 | &#79; | O | 111 | 6F | 157 | &#111; | o |
| 16 | 10 | 020 | DLE (data link escape) | 48 | 30 | 060 | &#48; | 0 | 80 | 50 | 120 | &#80; | P | 112 | 70 | 160 | &#112; | p |
| 17 | 11 | 021 | DC1 (device control 1) | 49 | 31 | 061 | &#49; | 1 | 81 | 51 | 121 | &#81; | Q | 113 | 71 | 161 | &#113; | q |
| 18 | 12 | 022 | DC2 (device control 2) | 50 | 32 | 062 | &#50; | 2 | 82 | 52 | 122 | &#82; | R | 114 | 72 | 162 | &#114; | r |
| 19 | 13 | 023 | DC3 (device control 3) | 51 | 33 | 063 | &#51; | 3 | 83 | 53 | 123 | &#83; | S | 115 | 73 | 163 | &#115; | s |
| 20 | 14 | 024 | DC4 (device control 4) | 52 | 34 | 064 | &#52; | 4 | 84 | 54 | 124 | &#84; | T | 116 | 74 | 164 | &#116; | t |
| 21 | 15 | 025 | NAK (negative acknowledge) | 53 | 35 | 065 | &#53; | 5 | 85 | 55 | 125 | &#85; | U | 117 | 75 | 165 | &#117; | u |
| 22 | 16 | 026 | SYN (synchronous idle) | 54 | 36 | 066 | &#54; | 6 | 86 | 56 | 126 | &#86; | V | 118 | 76 | 166 | &#118; | v |
| 23 | 17 | 027 | ETB (end of trans. block) | 55 | 37 | 067 | &#55; | 7 | 87 | 57 | 127 | &#87; | W | 119 | 77 | 167 | &#119; | w |
| 24 | 18 | 030 | CAN (cancel) | 56 | 38 | 070 | &#56; | 8 | 88 | 58 | 130 | &#88; | X | 120 | 78 | 170 | &#120; | x |
| 25 | 19 | 031 | EM (end of medium) | 57 | 39 | 071 | &#57; | 9 | 89 | 59 | 131 | &#89; | Y | 121 | 79 | 171 | &#121; | y |
| 26 | 1A | 032 | SUB (substitute) | 58 | 3A | 072 | &#58; | : | 90 | 5A | 132 | &#90; | Z | 122 | 7A | 172 | &#122; | z |
| 27 | 1B | 033 | ESC (escape) | 59 | 3B | 073 | &#59; | ; | 91 | 5B | 133 | &#91; | [ | 123 | 7B | 173 | &#123; | { |
| 28 | 1C | 034 | FS (file separator) | 60 | 3C | 074 | &#60; | < | 92 | 5C | 134 | &#92; | \ | 124 | 7C | 174 | &#124; | \| |
| 29 | 1D | 035 | GS (group separator) | 61 | 3D | 075 | &#61; | = | 93 | 5D | 135 | &#93; | ] | 125 | 7D | 175 | &#125; | } |
| 30 | 1E | 036 | RS (record separator) | 62 | 3E | 076 | &#62; | > | 94 | 5E | 136 | &#94; | ^ | 126 | 7E | 176 | &#126; | ~ |
| 31 | 1F | 037 | US (unit separator) | 63 | 3F | 077 | &#63; | ? | 95 | 5F | 137 | &#95; | _ | 127 | 7F | 177 | &#127; | DEL |

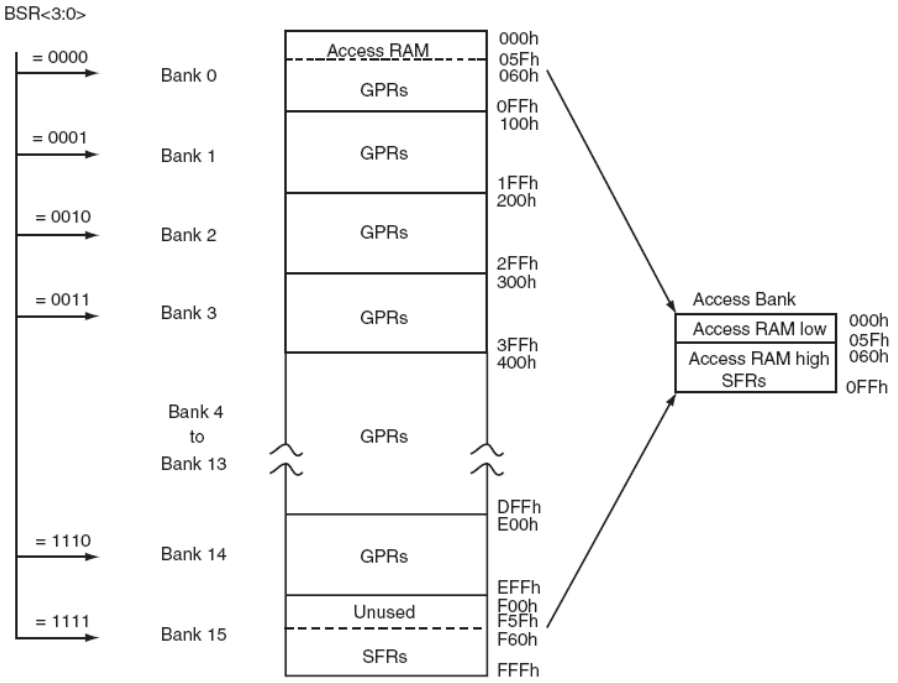
# Data Memory

- Data memory is either SRAM or EEPROM.
- SRAM data memory begins at 12-bit address 0x000 and ends at 12-bit address 0xFFF.
  - Not all PIC18 versions contain 4K or data memory space.
- Various PIC18 versions contain between 256 and 3968 bytes of data memory.

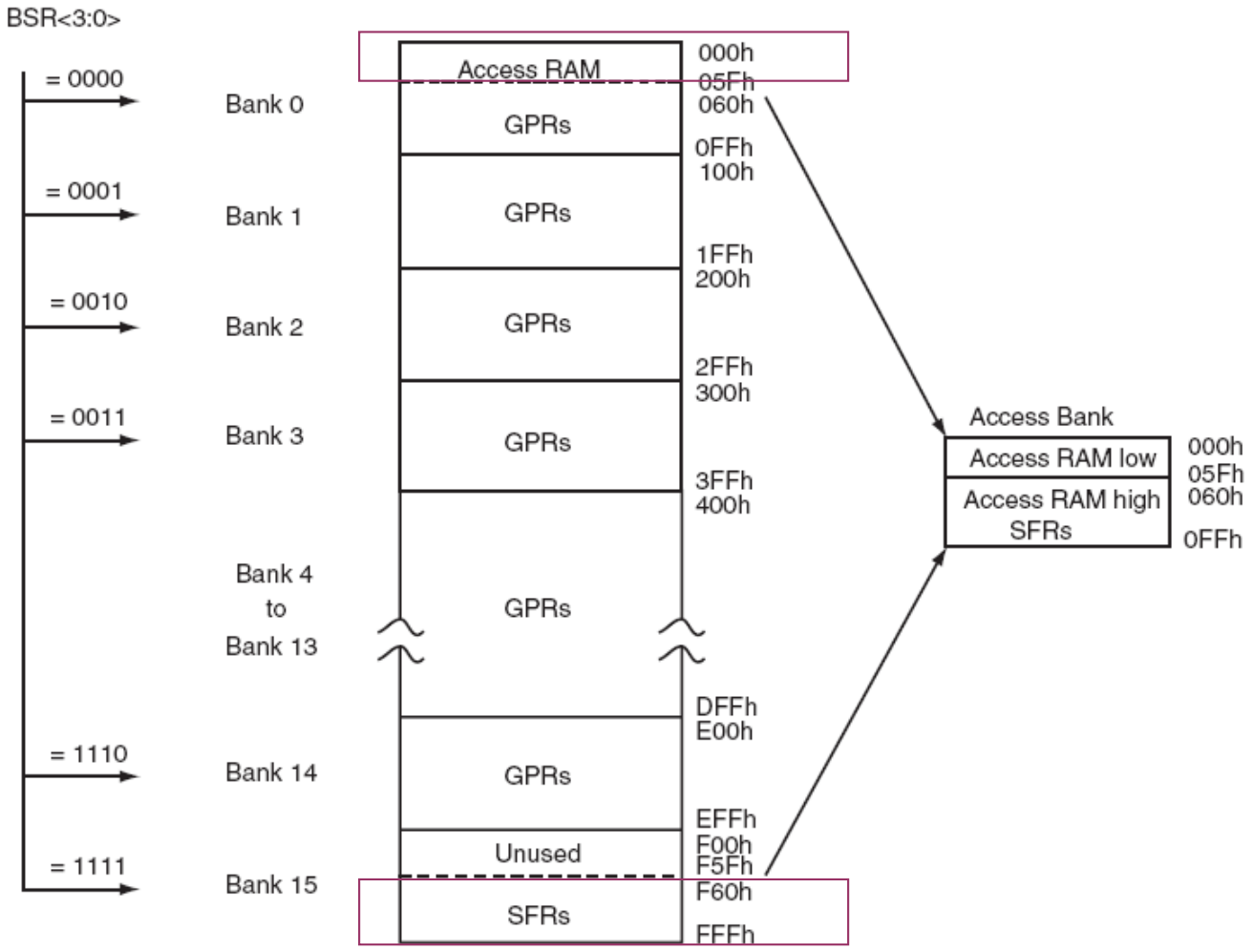
| Device | Program Memory | | Data Memory | |
|---|---|---|---|---|
| | Flash (bytes) | # Single-Wor Instructions | SRAM (bytes) | EEPROM (bytes) |
| PIC18F23K20 | 8K | 4096 | 512 | 256 |
| PIC18F24K20 | 16K | 8192 | 768 | 256 |
| PIC18F25K20 | 32K | 16384 | 1536 | 256 |
| PIC18F26K20 | 64k | 32768 | 3936 | 1024 |
| PIC18F43K20 | 8K | 4096 | 512 | 256 |
| PIC18F44K20 | 16K | 8192 | 768 | 256 |
| PIC18F45K20 | 32K | 16384 | 1536 | 256 |
| PIC18F46K20 | 64k | 32768 | 3936 | 1024 |

# Data Memory

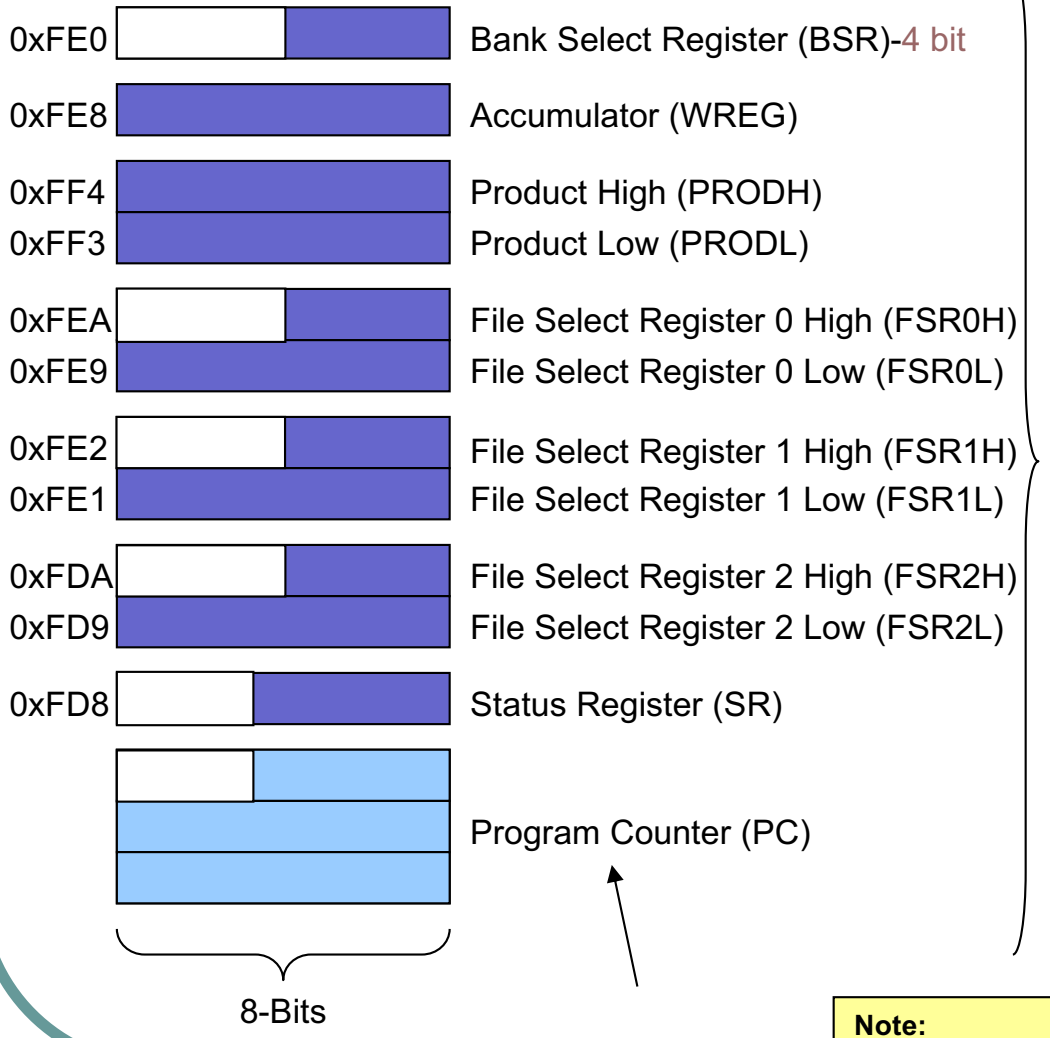
- There are two types of registers:
  - general-purpose registers (GPRs)
  - special-function registers (SFRs)
- GPRs are used to hold dynamic data when the PIC18 CPU is executing a program.
- SFRs are registers used by the CPU and peripheral modules for controlling the desired operation of the MCU.
- The upper 128 bytes of the data memory are used for special function registers (SFR) at addresses 0xF80 through 0xFFF. Some versions of the PIC18 have additional SFRs at locations below 0xF80.



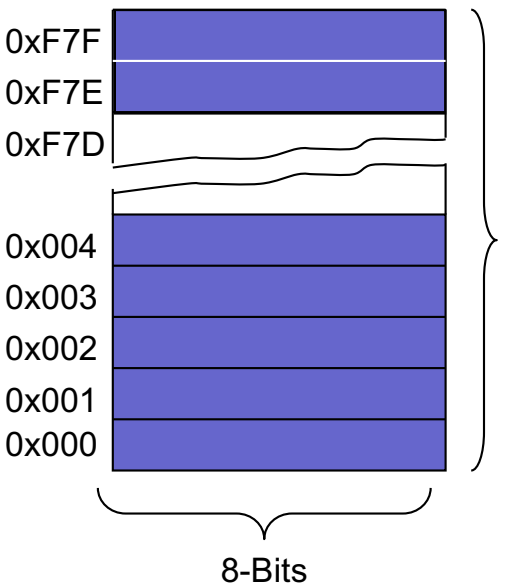
# Data Memory Structure



# Data Memory (SFR Examples)




Major Special Function Registers




**Note:**
**- The program counter is an internal 21-bit physical register**
**- The program counter is modified by the GOTO, CALL, RETURN, and branch instructions. The program counter is not directly addressable.**

# Accessing Data Memory

1. Using Direct Method (Direct Addressing)
   - Using BSRs
2. Using Indirect Method (Indirect Addressing

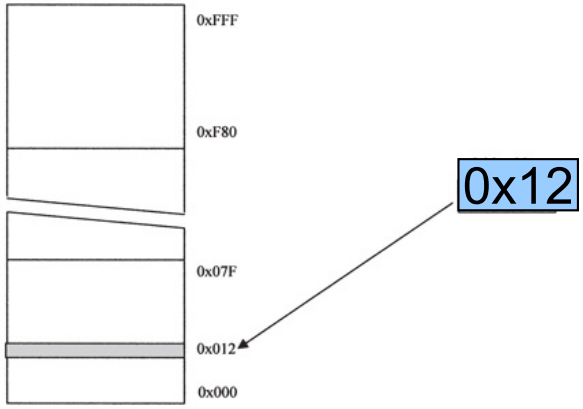
# Direct Addressing
# Using BSR – Writing into file registers

```
Main:
    BSF     BSR,0   ; BSR = 1

    movlw   0x44
    movwf   0x20,0

    movlw   0xAA
    movwf   0x21,1
```

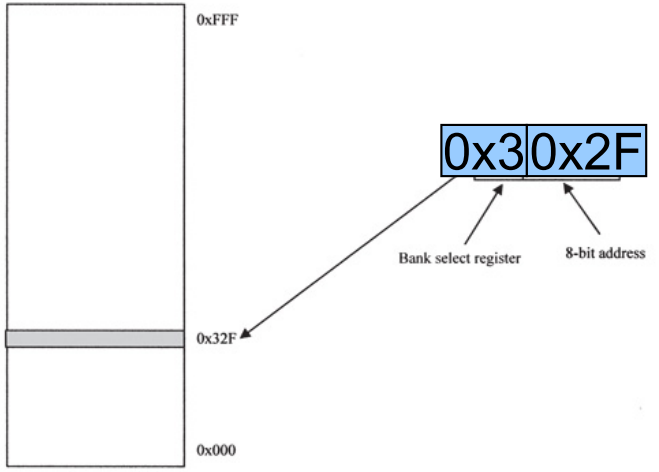
File Registers

| Address | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 000 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 010 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 020 | 44 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 030 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 040 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 050 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 060 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 070 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 080 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 090 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0A0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0B0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0C0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0D0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0E0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 0F0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 100 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 110 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 120 | 00 | AA | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 130 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |

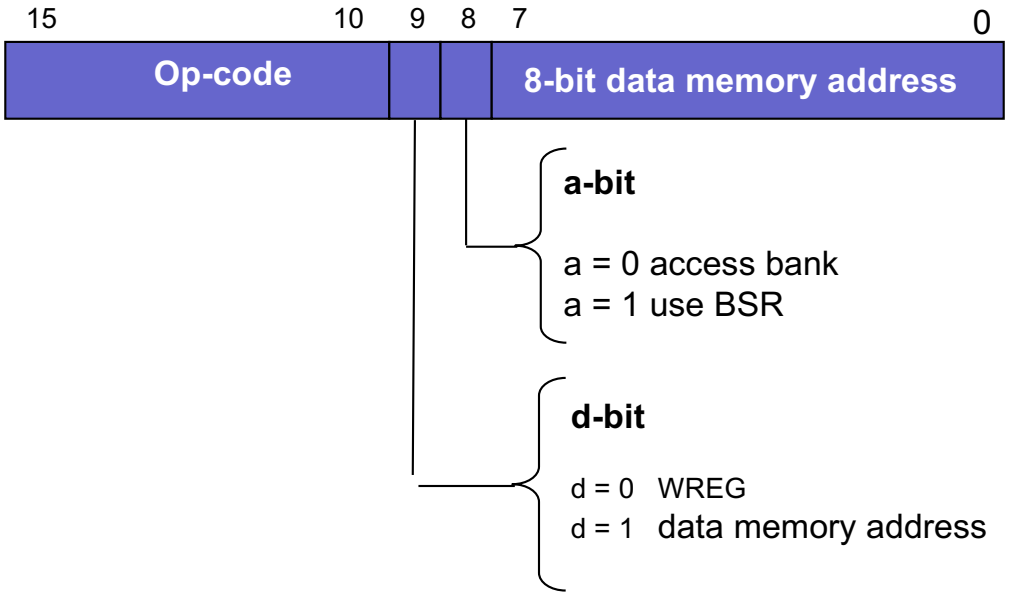
a=0; access bank




a=1; bank selection



# Direct Addressing
# A Typical Instruction showing the a-bit



# Direct Addressing Instruction Examples

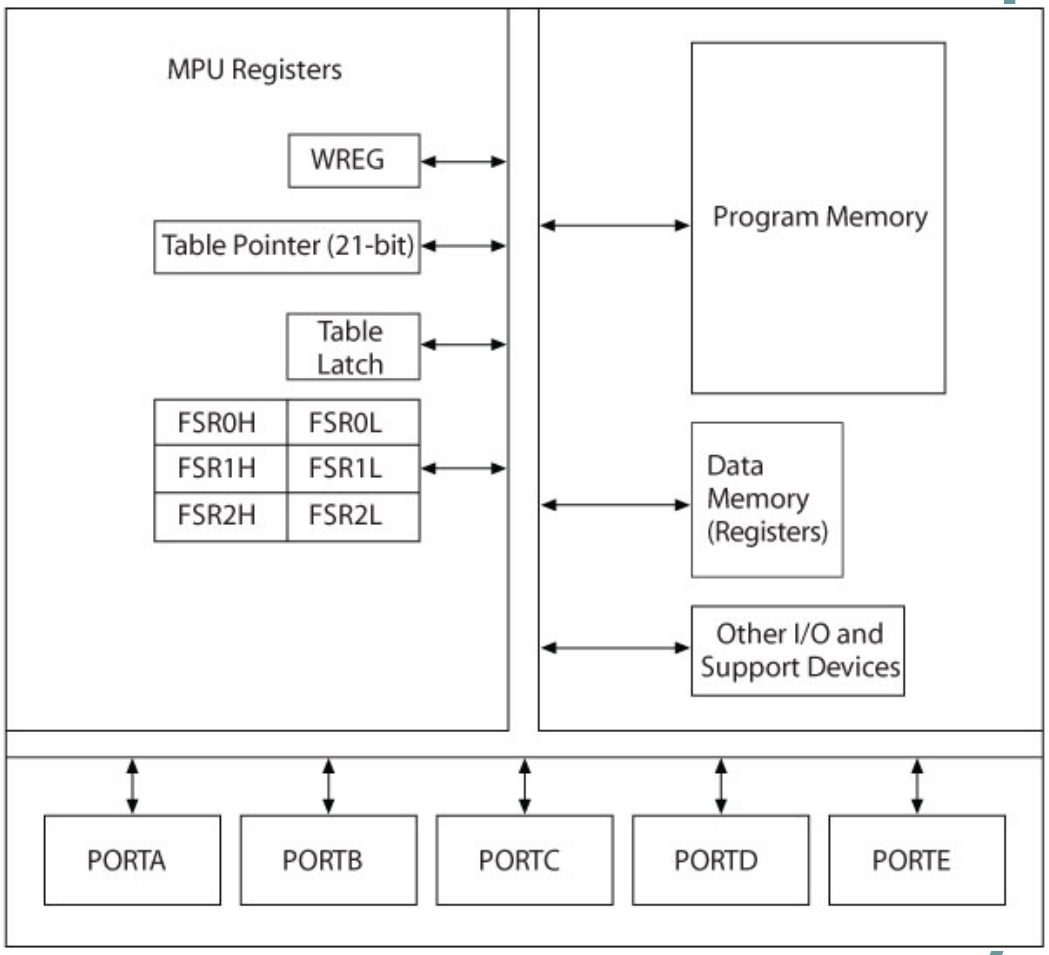
```
        MOVLW 0x06              ;place a 0x06 into W
        ADDLW 0x02              ;add a 0x02 to W
        MOVWF 0x00, 0           ;copy W to access bank register 0x00

;   OR  another version using the ACCESS keyword

        MOVLW 0x06              ;place a 0x06 into W
        ADDLW 0x02              ;add a 0x02 to W
        MOVWF 0x00, ACCESS      ;copy W to access bank register 0x00
```

# Direct Addressing Instruction Examples

```
MOVLW    0x06          ;place a 0x06 into W
ADDLW    0x02          ;add a 0x02 to W
MOVLB    2             ;load BSR with bank 2
MOVWF    0x00, 1       ;copy W to data register 0x00
                       ;of bank 2 or address 0x200

; OR using the BANKED keyword

MOVLW    0x06          ;place a 0x06 into W
ADDLW    0x02          ;add a 0x02 to W
MOVLB    2             ;load BSR with 2
MOVWF    0x00, BANKED  ;copy W to data register 0x00
                       ;of bank 2 or address 0x200

; OR without any bank indication

MOVLW    0x06          ;place a 0x06 into W
ADDLW    0x02          ;add a 0x02 to W
MOVLB    2             ;load BSR with bank 2
MOVWF    0x00          ;copy W to data register 0x00
                       ;of bank 2 or address 0x200
```

# Indirect Addressing
# Using File Select Registers (FSRs) as Pointers

- Memory pointer is a register that holds the address of a data register
  - This is called indirect addressing
  - Easy to move/copy an entire block
- Three pointer registers: FSR0, FSR1, and FSR2

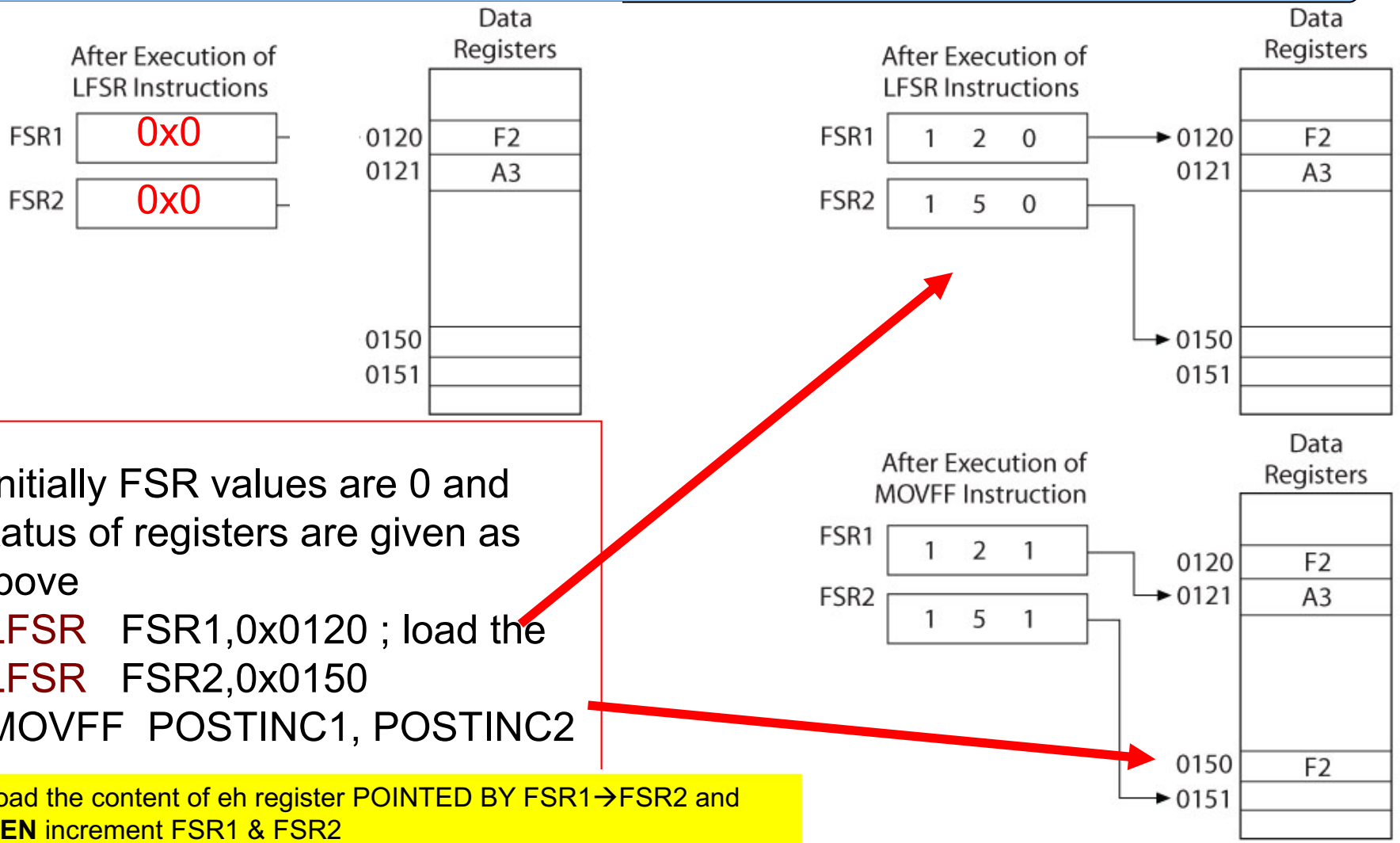


# Indirect Addressing
# Using File Select Registers (FSRs) as Pointers

- Memory pointer is a register that holds the address of a data register
  - This is called indirect addressing
  - Easy to move/copy an entire block
- Three registers: FSR0, FSR1, and FSR2
  - Each FSR has a High and Low byte associated with an index
  - Used as memory pointers to data registers
- Each can be used in five different formats (operands) :
  - INDF0: Use FSR0 as pointer (index)
  - POSTINC0: Use FSR0 as pointer and increment FSR0
  - POSTDEC0: Use FSR0 as pointer and decrement SR0
  - PREINC0: Increment FSR0 first and use as pointer
  - PLUSW0: Add W to FSR0 and use as pointer

# Indirect Addressing - Example

```
LFSR        FSR1,120        ; LOAD 12-BIT ADDRESS 120h INTO FSR1
```




- Initially FSR values are 0 and status of registers are given as above
- LFSR FSR1,0x0120 ; load the
- LFSR FSR2,0x0150
- MOVFF POSTINC1, POSTINC2

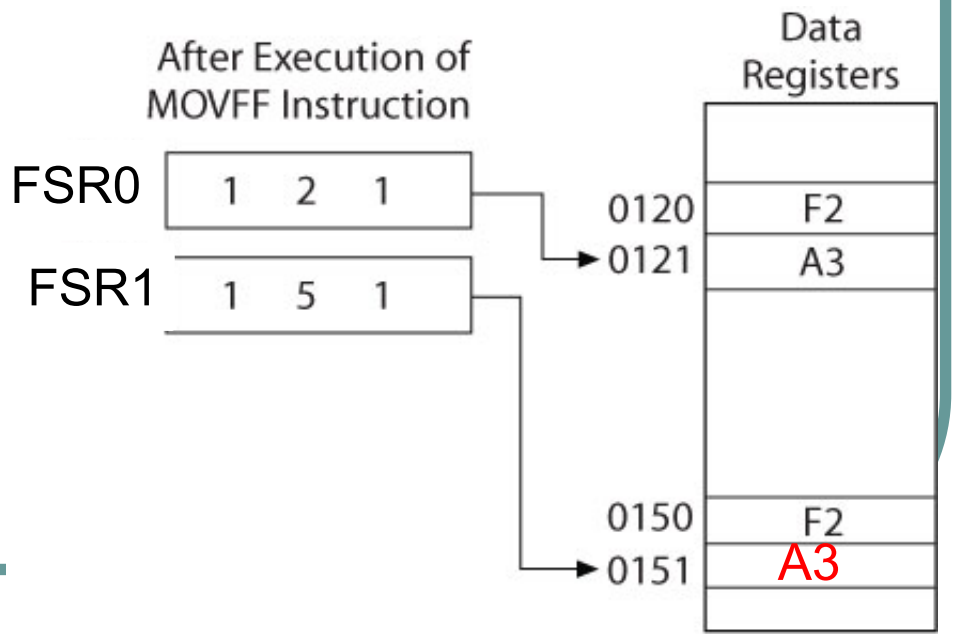
; Load the content of eh register POINTED BY FSR1→FSR2 and **THEN** increment FSR1 & FSR2

# Indirect Addressing – Example (1)

Examples:

```
MOVFF     INDF0,INDF1     ; COPY BYTE FROM REGISTERS SHOWN BY
                          ;FSR0 TO FSR1- NO CHANGE IN FSR
ADDWF     POSTINC0,1      ; ADD BYTE FROM REGISTERS SHOWN BY
                          ;FSR0 AND W and then leave the result in REG ;
                          ;THEN increment FSR0
```

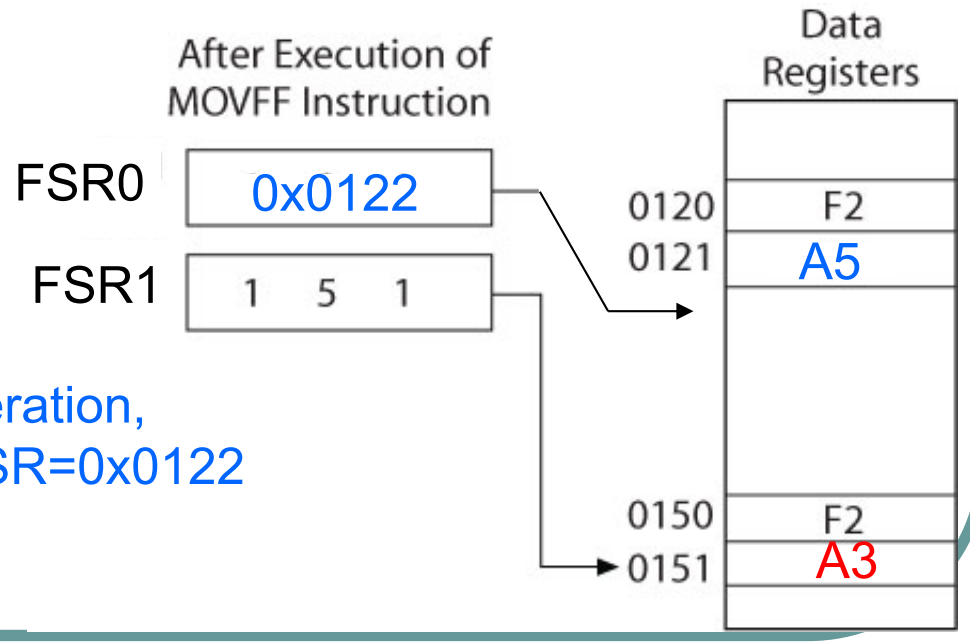
- Hence, we will have A3 in register 0x151 after the MOVFF instruction
- Note that the pointer indexes are not changing!



# Indirect Addressing Example (2)

Examples:

```
MOVFF     INDF0,INDF1     ; COPY BYTE FROM REGISTERS SHOWN BY
                          ;FSR0 TO FSR1- NO CHANGE IN FSR
ADDWF     POSTINC0,1      ; ADD BYTE FROM REGISTERS SHOWN BY
                          ;FSR0 AND W→REG ;
                                        ;FSR0 IS INCREMENTED
```




Assume W=2; after the ADD operation,
A3+2=A5→Reg 0x0121, then FSR=0x0122

# Initializing the RAM – Application of Indirect Addressing

```
Main:
     CLRF     FSR1H
     MOVLW    0x40
     MOVWF    FSR1L
NEXT
     SETF     POSTINC1
     BTFSS    FSR1L, 4
     GOTO     NEXT
     sleep
```

FSR1 **0x 00 40**

→ Set the content of the register pointed by FSR1 (0xFF)

→ Increment until bit 4 is set in FSRL1

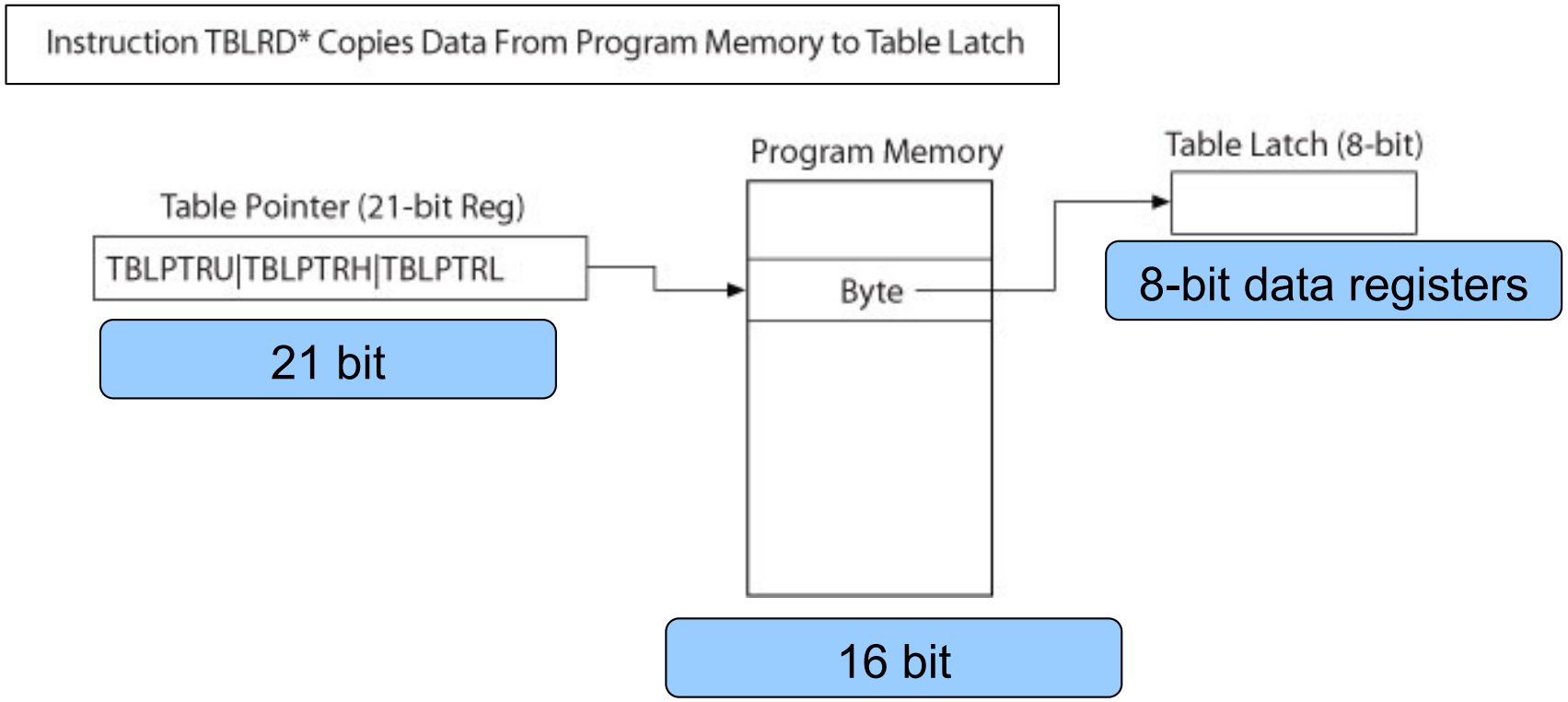
```
Main:
     CLRF     FSR1H
     CLRF     FSR1L
```

FSR1 **0x 00 00**

File Registers

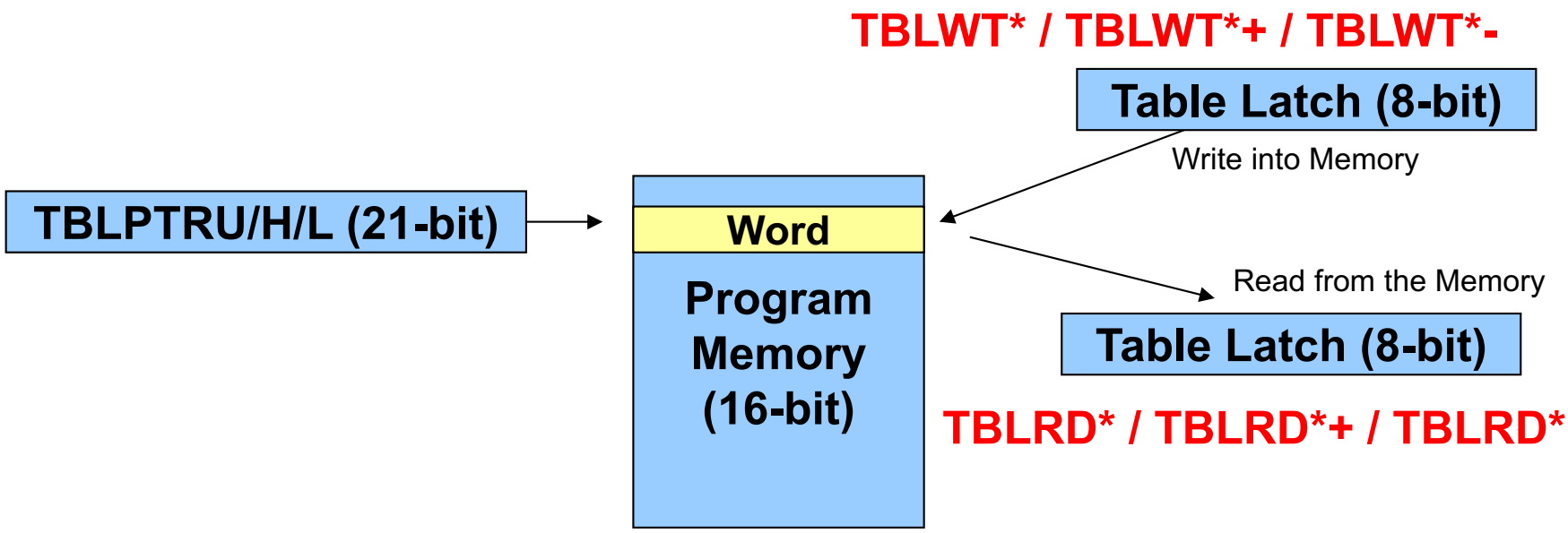
| Address | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 000 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 010 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 020 | 44 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 030 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |
| 040 | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF | FF |
| 050 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 |

# Using Table Pointers to Copy Data

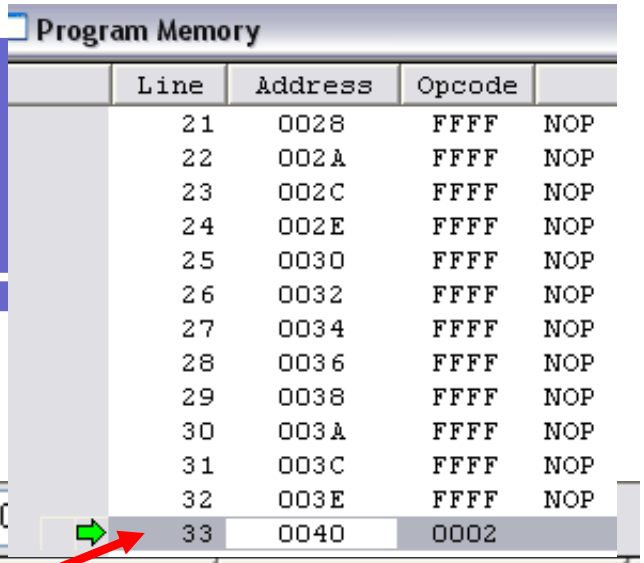


# Using Table Pointers

- Reading/writing values from/into the program memory one byte at a time

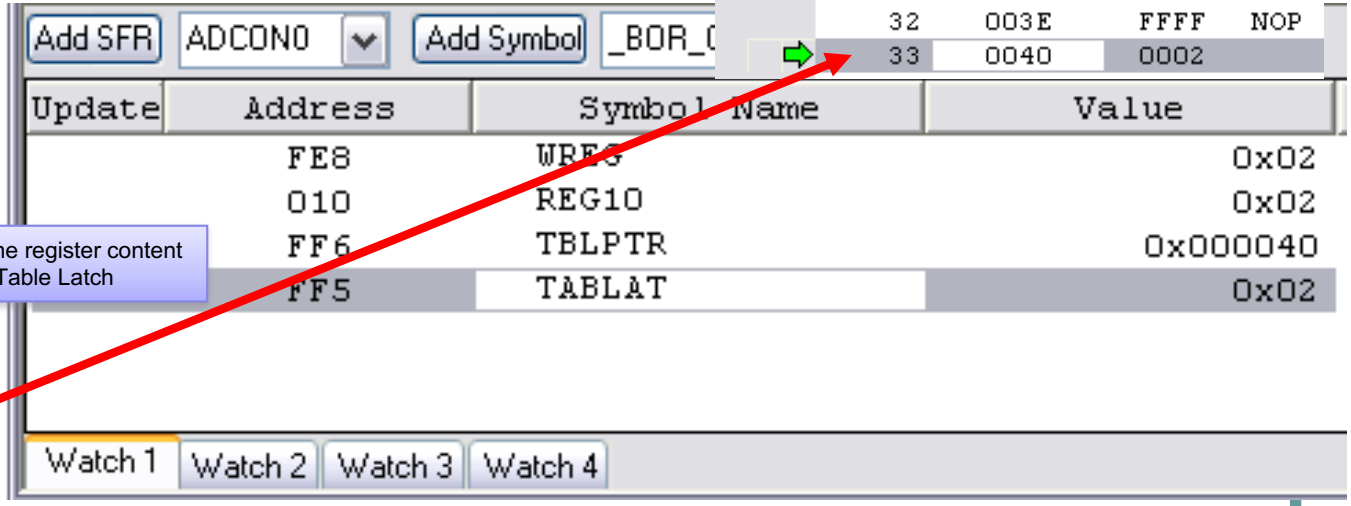


# Table Example




```
        MOVLW   UPPER BUFFER
        MOVWF   TBLPTRU
        MOVLW   HIGH BUFFER
        MOVWF   TBLPTRH
        MOVLW   LOW BUFFER
        MOVWF   TBLPTRL
        TBLRD*
        MOVF    TABLAT,WREG
        MOVWF   REG10

        ORG     0X40
BUFFER  DB      0X2
```

Rd the register content into Table Latch

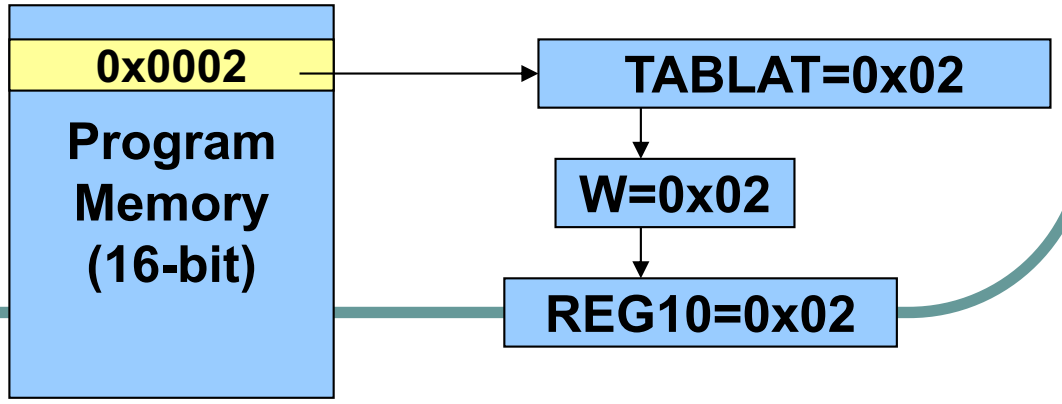



Save in Prog Memory; Label the value as BUFFER

**TBLPTR(U/H/L)= 00 00 40**

→ **0x000040**

Pointing to the address



# Table Example

**LAB: Modify this program such that values 0xaa, 0xbb, 00cc stored in the program memory are copied into REG60,61,62, respectively**

```
        MOVLW   UPPER BUFFER
        MOVWF   TBLPTRU
        MOVLW   HIGH BUFFER
        MOVWF   TBLPTRH
        MOVLW   LOW BUFFER
        MOVWF   TBLPTRL
        TBLRD*
        MOVF    TABLAT,WREG
        MOVWF   REG10

        ORG     0X40
BUFFER  DB      0X2
```

Add SFR | ADCON0 | Add Symbol | _BOR_OFF_2L

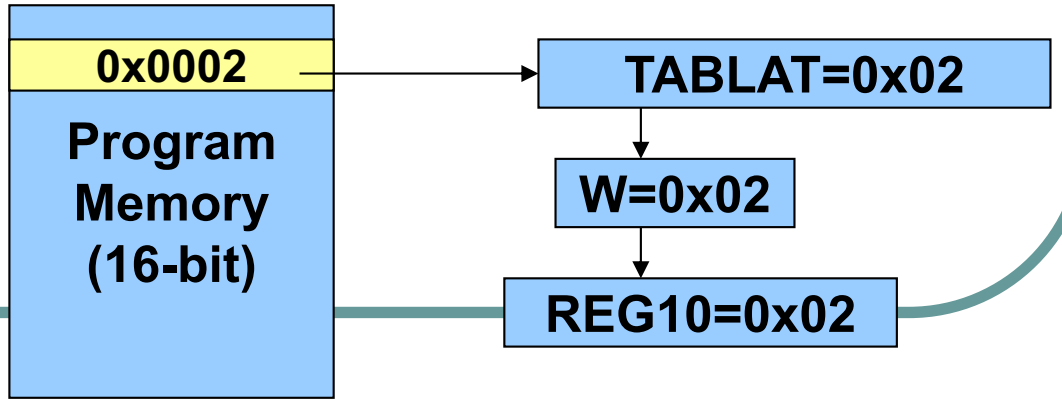
| Update | Address | Symbol Name | Value |
|---|---|---|---|
| | FE8 | WREG | 0x02 |
| | 010 | REG10 | 0x02 |
| | FF6 | TBLPTR | 0x000040 |
| | FF5 | TABLAT | 0x02 |

Watch 1 | Watch 2 | Watch 3 | Watch 4

**TBLPTR = 00 00 40**

→ **0x000040**

Pointing to the address



# Practice Program

- Save your first name into EEPROM starting Register 0x20
- Save your last name into EEPROM starting Register 0x80
- Save your first name into RAM starting Register 0x20
- Save your last name into RAM starting Register 0x80
- Write your last name in bank 2 starting with register 0x20
- Multiply 0x8 and 0x5 and leave the result in registers 0x50 of the RAM in Bank 4
- What is the address of the PC SFR ?
- Load FF into RAM registers 0x30-0x40. Use Indirect addressing only!

# Backup