Tecnológico
de Monterrey

Tecnológico de Monterrey

TE2015 Microcontroladores

PIC18 Microcontroller Architecture

Tecnológico de Monterrey

OUTLINE

Tuesday, 16/August

- Evolution of the PIC Microcontroller
- Features of PIC18

Friday, 19/August

- Special Function Register
- Access Bank and Bank Select Register

Tecnológico de Monterrey

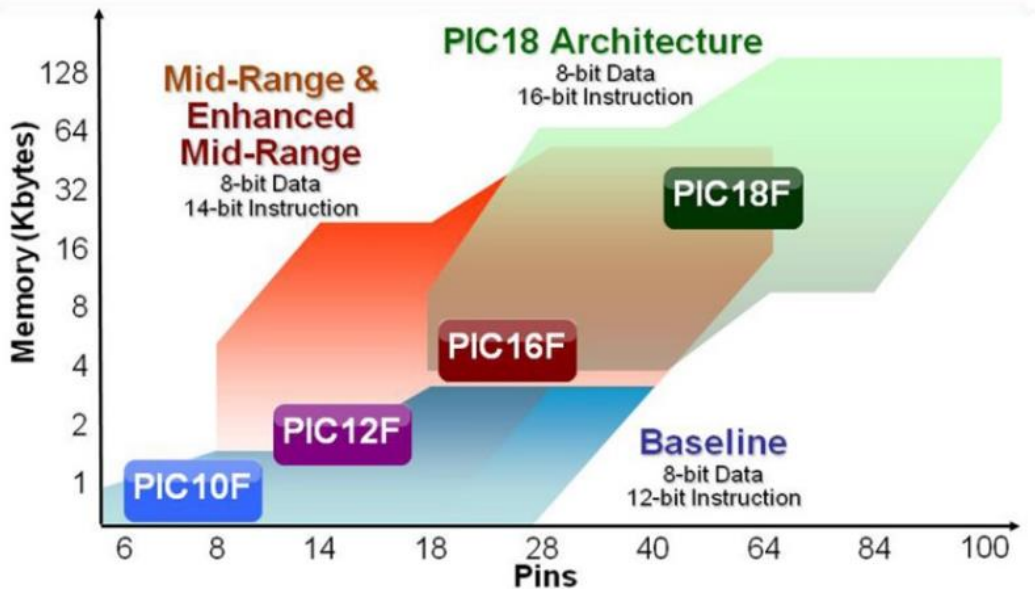
TE2015 Microcontroladores

Evolution of the PIC Microcontroller

Tecnológico de Monterrey

EVOLUTION OF PIC MICROCONTROLLER

	Baseline Architecture	Mid-Range Architecture	Enhanced Mid-Range Architecture	PIC18 Architecture
Pin Count	6-40	8-64	8-64	18-100
Interrupts	No	Single interrupt capability	Single interrupt capability with hardware context save	Multiple interrupt capability with hardware context save
Performance	5 MIPS	5 MIPS	8 MIPS	Up to 16 MIPS
Instructions	33, 12-bit	35, 14-bit	49, 14-bit	83, 16-bit
Program Memory	Up to 3 KB	Up to 14 KB	Up to 28 KB	Up to 128 KB
Data Memory	Up to 138 Bytes	Up to 368 Bytes	Up to 1,5 KB	Up to 4 KB
Hardware Stack	2 level	8 level	16 level	32 level
Features	▪ Comparator ▪ 8-bit ADC ▪ Data Memory ▪ Internal Oscillator	In addition to Baseline: ▪ SPI/I²C™ ▪ UART ▪ PWMs ▪ LCD ▪ 10-bit ADC ▪ Op Amp	In addition to Mid-Range: ▪ Multiple Communication Peripherals ▪ Linear Programming Space ▪ PWMs with Independent Time Base	In addition to Enhanced Mid-Range: ▪ 8x8 Hardware Multiplier ▪ CAN ▪ CTMU ▪ USB ▪ Ethernet ▪ 12-bit ADC
Highlights	Lowest cost in the smallest form factor	Optimal cost to performance ratio	Cost effective with more performance and memory	High performance, optimized for C programming, advanced peripherals



Tecnológico de Monterrey

TE2015 Microcontroladores

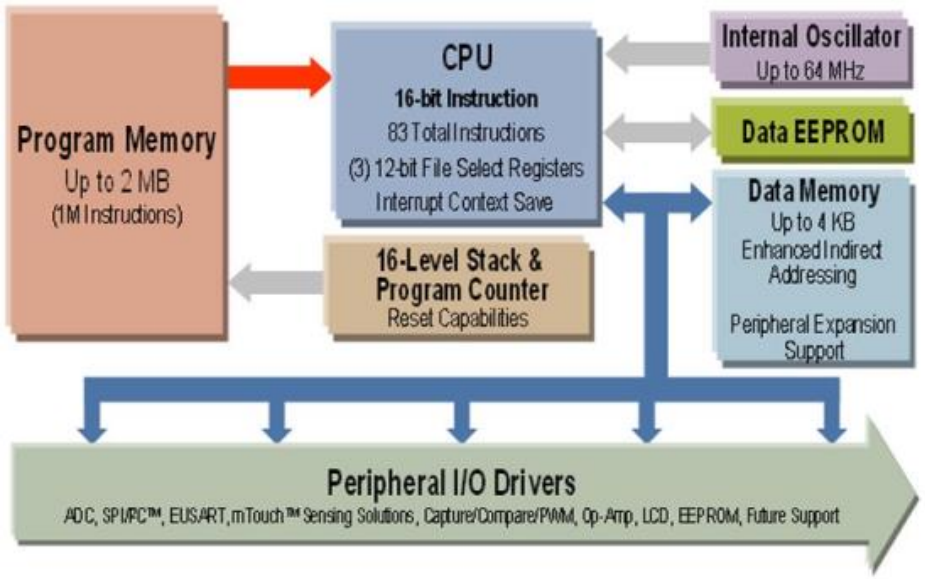
Features of PIC18

Tecnológico de Monterrey

MEMORY ORGANIZATION

- **Program memory**
  - Non-volatile memory that stores all the instructions we write
- **Data memory (RAM)**
  - Volatile memory that keeps variables and runtime-generated data
- **Data memody (EEPROM)**
  - Non-volatile with same functionality tan DATA RAM memory

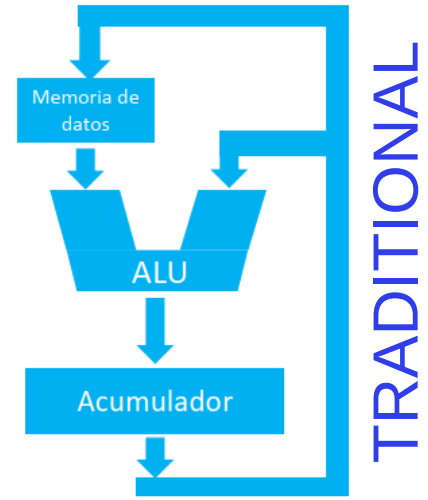
Which architecture is that shown on the right?



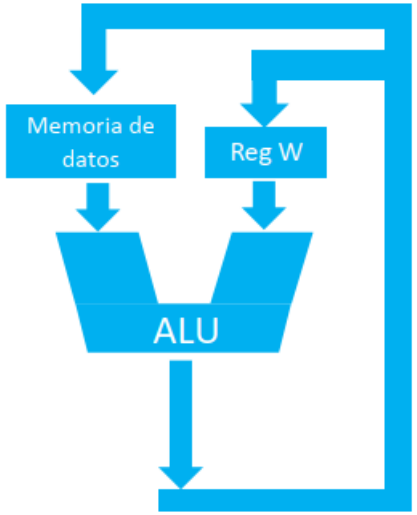
INSTRUCTION FORMAT

The instruction set of PIC microcontrollers is Orthogonal

- We can read from and write to all file registers, including SFRs
- What makes the difference is where the WORKING (ACC) register is located
  - In traditional architectures, the result of any operation is always stored in ACC
  - In orthogonal architectures, the result of any operation can be stored in W or in data memory
    - W reg is always one operand of ALU



TRADITIONAL



ORTHOGONAL

Tecnológico de Monterrey

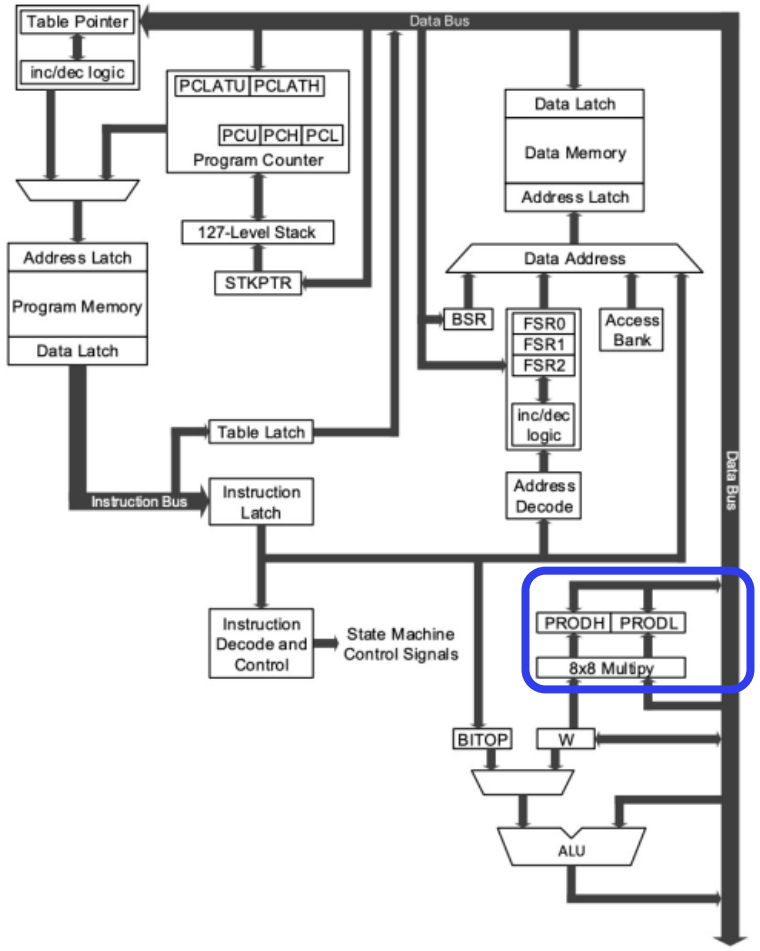
8 x 8 HARDWARE MULTIPLIER

The PIC18 family features an 8x8 hardware multiplier as part of the arithmetic-logic unit.

- Carries out an unsigned multiplication and the 16-bit product is stored in the PROD register.
- Hardware multiplier does not affect the STATUS register
- Hardward multiplication allows to complete a full operation in one instruction cycle

Table 7-2. Performance Comparison for Various Multiply Operations

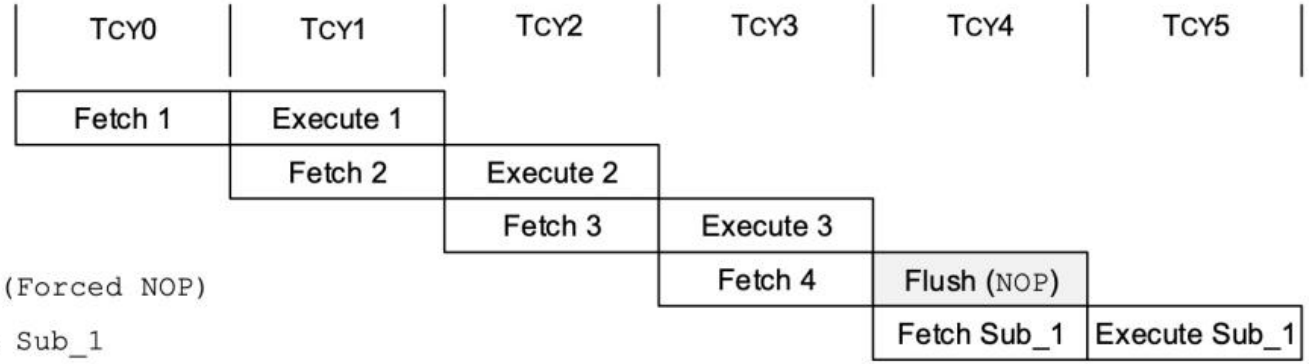
Routine	Multiply Method	Program Memory (Words)	Cycles (Max)	Time			
				@ 64 MHz	@ 40 MHz	@ 10 MHz	@ 4 MHz
8x8 unsigned	Without hardware multiply	13	69	4.3 µs	6.9 µs	27.6 µs	69 µs
	Hardware multiply	1	1	62.5 ns	100 ns	400 ns	1 µs
8x8 signed	Without hardware multiply	33	91	5.7 µs	9.1 µs	36.4 µs	91 µs
	Hardware multiply	6	6	375 ns	600 ns	2.4 µs	6 µs
16x16 unsigned	Without hardware multiply	21	242	15.1 µs	24.2 µs	96.8 µs	242 µs
	Hardware multiply	28	28	1.8 µs	2.8 µs	11.2 µs	28 µs
16x16 signed	Without hardware multiply	52	254	15.9 µs	25.4 µs	102.6 µs	254 µs
	Hardware multiply	35	40	2.5 µs	4.0 µs	16.0 µs	40 µs

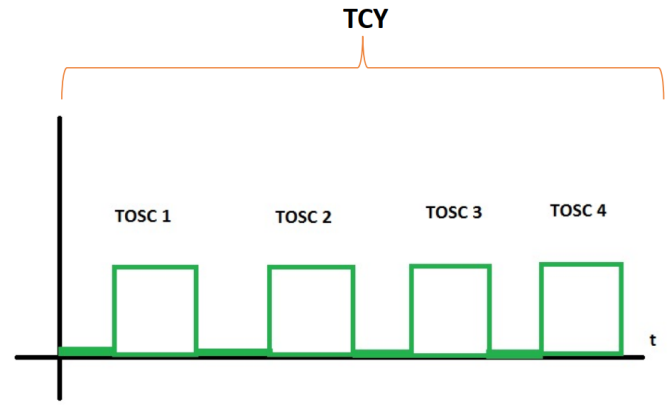


PIPELINING

- As a pipelined processor, the CPU is able to carry out one instruction per cycle:
  - EXECUTION of one of the program instructions
  - FETCH of the following program instruction
- The time it takes to execute a single-cycle instruction is called Cycle Time, TCY
  - One TCY comprises four clock cycles

```
1. MOVLW     55h
2. MOVWF     PORTB
3. BRA       Sub_1
4. BSF       PORTA, BITS (Forced NOP)
5. Instruction @ address Sub_1
```





$$Tosc = \frac{1}{Fosc}$$

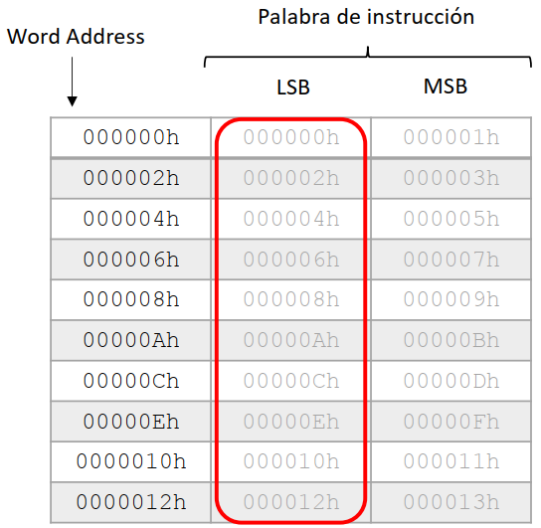
$$Tosc = \frac{1}{32Mhz} = 31.25nS$$

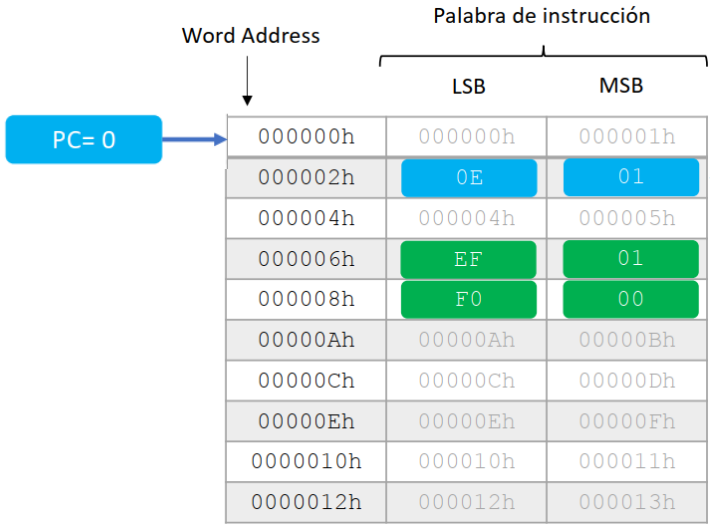
$$TCY = Tosc \times 4 = 125nS$$

$$TCY = \frac{4}{32Mhz} = 125nS$$

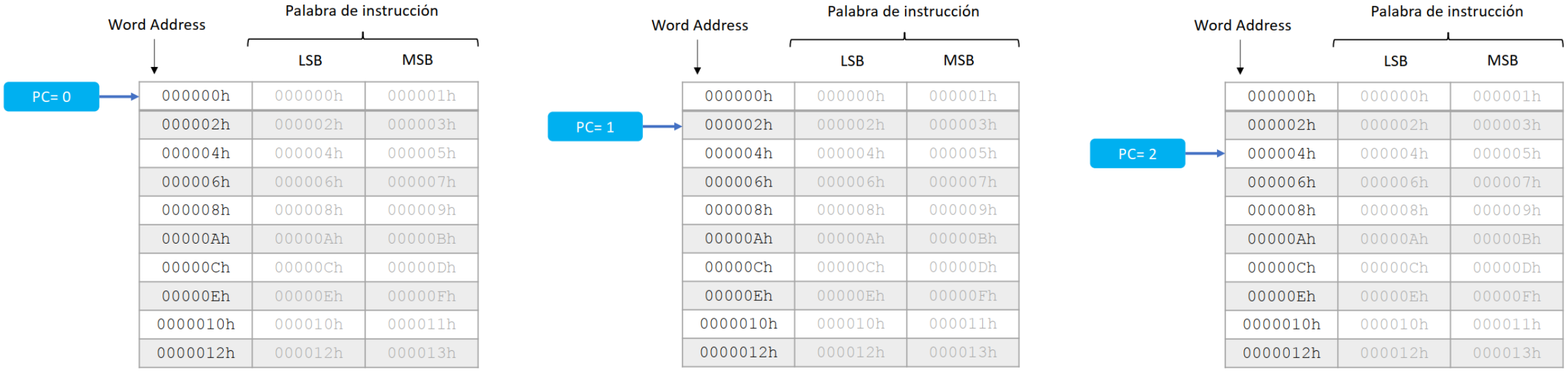
INSTRUCTIONS IN PROGRAM MEMORY

- Program memory is addressed per byte
- PC is incremented by 2
- Instructions are stored in two bytes or four bytes
  - LSB is always stored on an even address
  - 1 x TCY instructions are stored in two bytes
  - 2 x TCY instructions are stored in four bytes





Tecnológico de Monterrey



Tecnológico de Monterrey

EXAMPLE

Word Address	Palabra de instrucción	
	LSB	MSB
000000h	000000h	000001h
000002h	000002h	000003h
000004h	000004h	000005h
000006h	000006h	000007h
000008h	0E	55
00000Ah	00000Ah	00000Bh
00000Ch	00000Ch	00000Dh
00000Eh	00000Eh	00000Fh
0000010h	000010h	000011h
0000012h	000012h	000013h

ASM ejemplo.s

```
movlw 055h
```

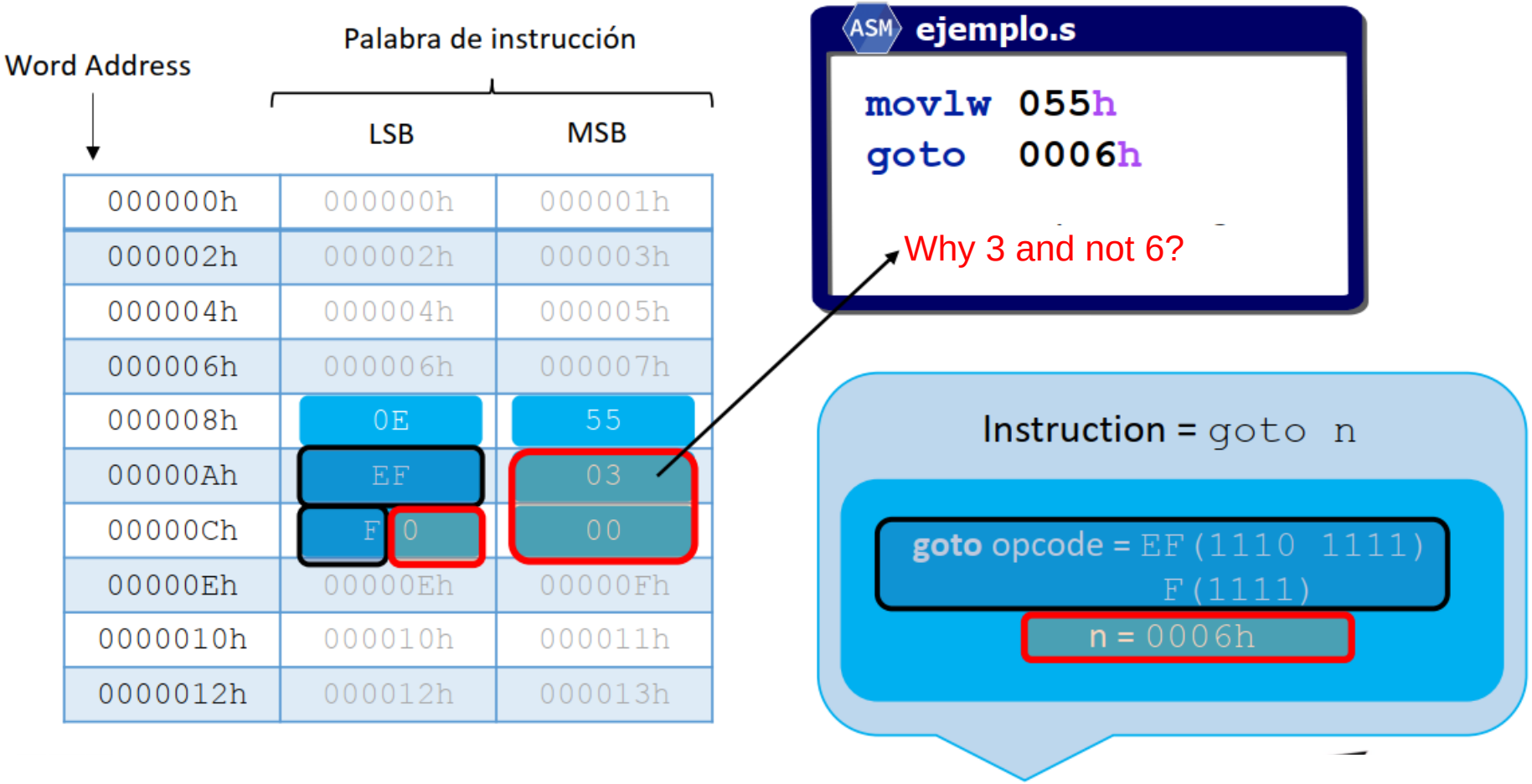
Instruction = `movlw k`

movlw opcode = 0E (0000 1110)

k = 055h

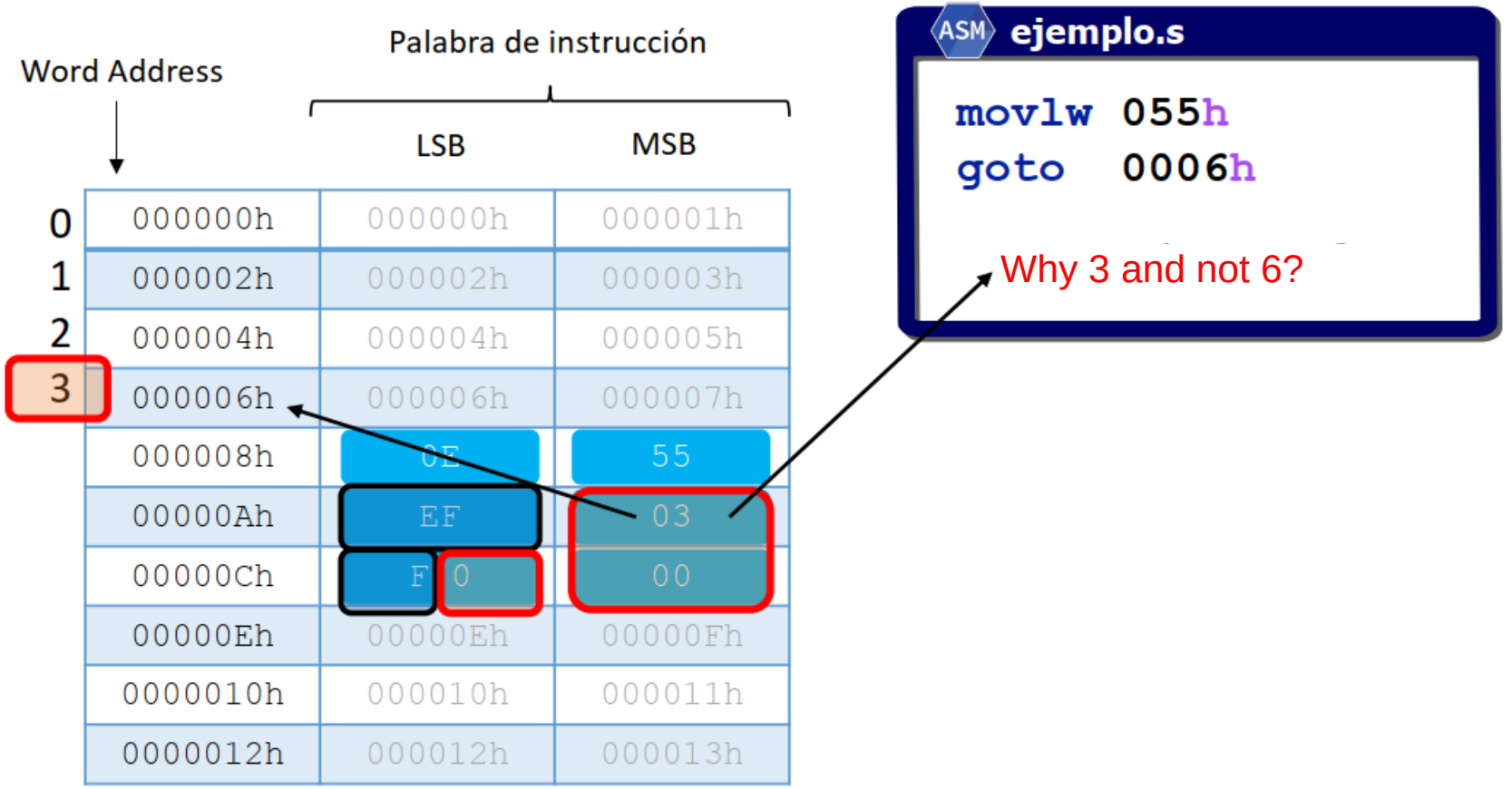
Tecnológico de Monterrey

EXAMPLE



Tecnológico de Monterrey

EXAMPLE



Tecnológico de Monterrey

PROGRAM COUNTER, PC[20:0]

Points to the address of next instruction

- In PIC18 has a length of 21 bits, thus

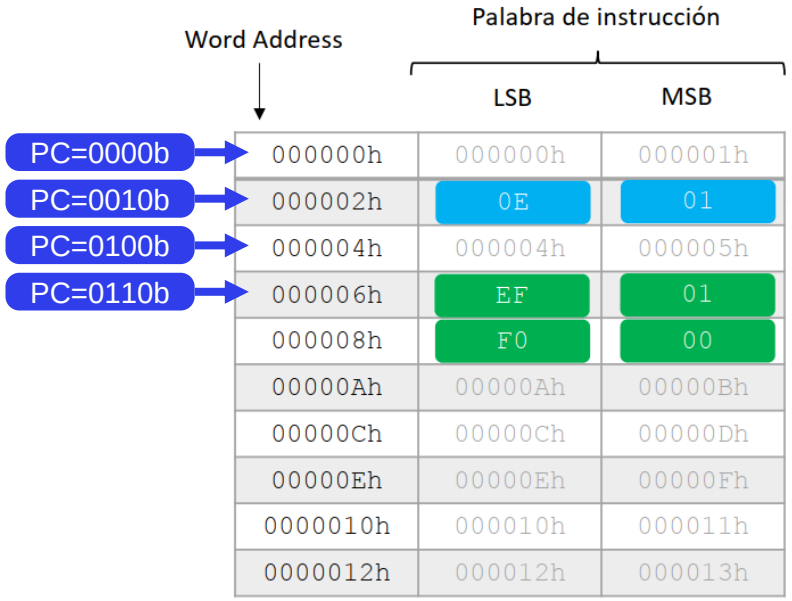
  2^{21} = 20,097,152 addresses

  = 2 MBytes

- Those 21 bits are split in three registers
- Contents of PCLATH and PCLATU are transfered to PCH and PCU by any operation that writes on PCL
- Conversely, contents of PCLATH and PCLATU are transfered to PCH and PCU by any operation that writes on PCL

REG	MEANING	RANGE	ACCESS
PCL	Low byte	PC[7:0]	R/W
PCH	High byte	PC[15:8]	PCLATH
PCU	Upper byte	PC[20:16]	PCLATU

LSB of PC is fixed to 0 and increments are by 2 to keep PC aligned to full instruction addresses



Tecnológico de Monterrey

TE2015 Microcontroladores

Special Function Registers

Tecnológico de Monterrey

RAM DATA MEMORY

- 12-bit addresses

  2^{12} = 4096 addresses = 4 KBytes

- It holds
  - **Special Function Registers (SFRs)**
    - Control and status of µC
    - Control and configuration of peripherals
  - **General Function Registers (GPRs)**
    - To store user data and variables
    - For fast memory access, we bank the memory

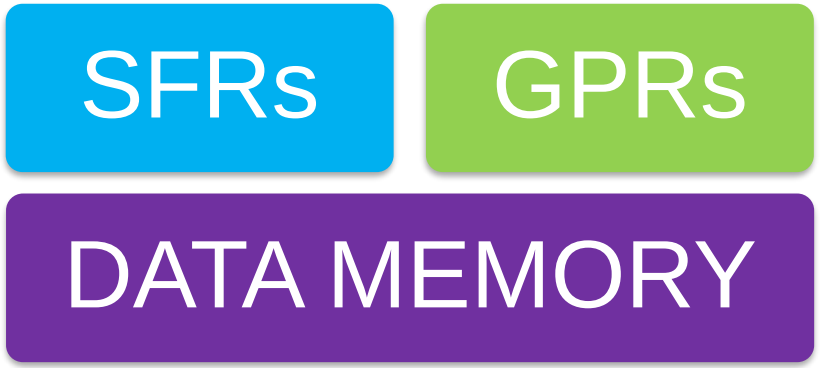


Figure 9-3. Data Memory Map

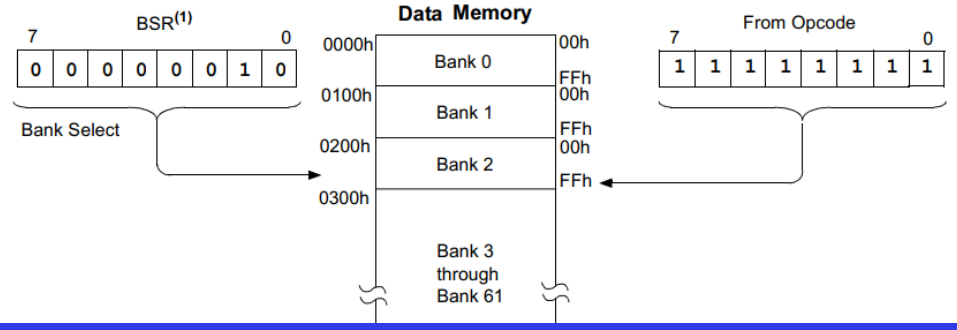
BanK	BSR addr[13:8]	addr[7:0]	PIC18F x5Q43	PIC18F x6Q43	PIC18F x7Q43
0	'b00 0000	0x00-0x5F			
1	'b00 0001	0x00-0xFF			
2	'b00 0010	0x00-0xFF			
3	'b00 0011	0x00-0xFF			
4	'b00 0100	0x00-0x5F			
	'b00 0100	0x60-0xFF			
5	'b00 0101	0x00-0x5F			
	'b00 0101	0x60-0xFF			
6	'b00 0110	0x00-0xFF			
7	'b00 0111	0x00-0xFF			
8	'b00 1000	0x00-0xFF			
9	'b00 1001	0x00-0xFF			
10	'b00 1010	0x00-0xFF			
11	'b00 1011	0x00-0xFF			
12	'b00 1100	0x00-0xFF			
13	'b00 1101	0x00-0xFF			
14	'b00 1110	0x00-0xFF			
15	'b00 1111	0x00-0xFF			
16	'b01 0000	0x00-0xFF			
17	'b01 0001	0x00-0xFF			
18	'b01 0010	0x00-0xFF			
19	'b01 0011	0x00-0xFF			
20	'b01 0100	0x00-0xFF			
21	'b01 0101	0x00-0xFF			
22	'b01 0110	0x00-0xFF			
23	'b01 0111	0x00-0xFF			
24	'b01 1000	0x00-0xFF			
25	'b01 1001	0x00-0xFF			
26	'b01 1010	0x00-0xFF			
27	'b01 1011	0x00-0xFF			
28	'b01 1100	0x00-0xFF			
29	'b01 1101	0x00-0xFF			
30	'b01 1110	0x00-0xFF			
31	'b01 1111	0x00-0xFF			
32	'b10 0000	0x00-0xFF			
33	'b10 0001	0x00-0xFF			
34	'b10 0010	0x00-0xFF			
35	'b10 0011	0x00-0xFF			
36	'b10 0100	0x00-0xFF			
37	'b10 0101	0x00-0xFF			
38 to 63	'b10 0110 - 'b11 1111	0x00-0xFF - 0x00-0xFF			

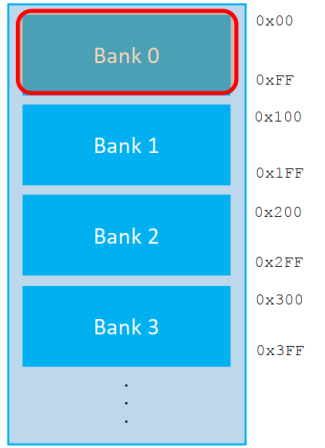
Virtual Access Bank
Access RAM 0x00-0x5F
Fast SFR 0x60-0xFF

GPR
SFR
Buffer RAM
Unimplemented

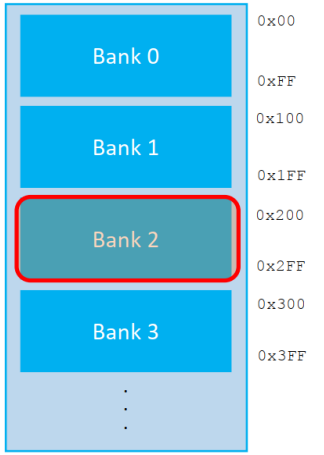
BANK SELECT REGISTER, BSR

- We can access every memory location by using its full 12-bit address
  - This is called Access Bank
- Or we can use a register called Bank Select Register, BSR
  - We just need the 4 most significant bits of an address
  - The instruction includes the remaining 8 bits
  - To load those 4 bits to BSR, we use MOVLB





BSR<3:0> = 0



BSR<3:0> = 2

EXAMPLE WITH PIC18F4550

Open PIC18F4550 datasheet and go to section 5.3 Data Memory Organization.

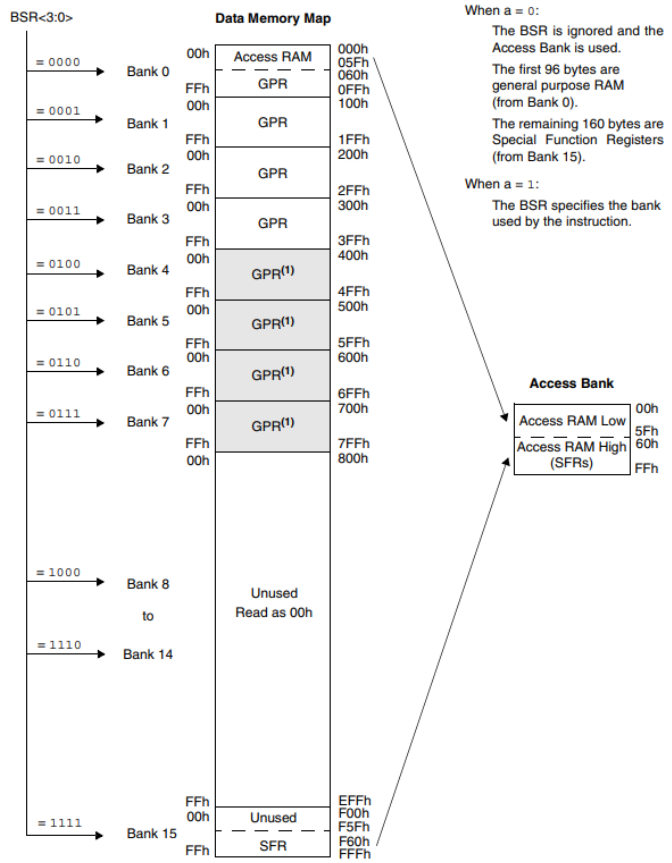
- How many data memory banks are available for the 4550 device? 15 banks
- How many of these are implemented and how many are not implemented? 9 are implemented and 7 are not implemented
- What is the size of each bank (in bytes)? 256 bytes
- What is the address range for Bank 3? 0x300 to 0x3FF

Use MPLAB X to simulate data load in PIC18F4550 through the Bank Select Register:

- Load 0xFA on address 0x100 in Bank 1
- Load 9_{10} on address 0x200 in Bank 2
- Load 0xA0 on address 0x345
- Load 156_{10} ton address 0x710
- What happens if we specify an address but do not switch to its corresponding bank?

File Registers

Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
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	FA	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	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00



Note 1: These banks also serve as RAM buffer for USB operation. See **Section 5.3.1 "USB RAM"** for more information.

EXAMPLE WITH PIC18F4550

Open PIC18F4550 datasheet and go to section 5.3 Data Memory Organization.

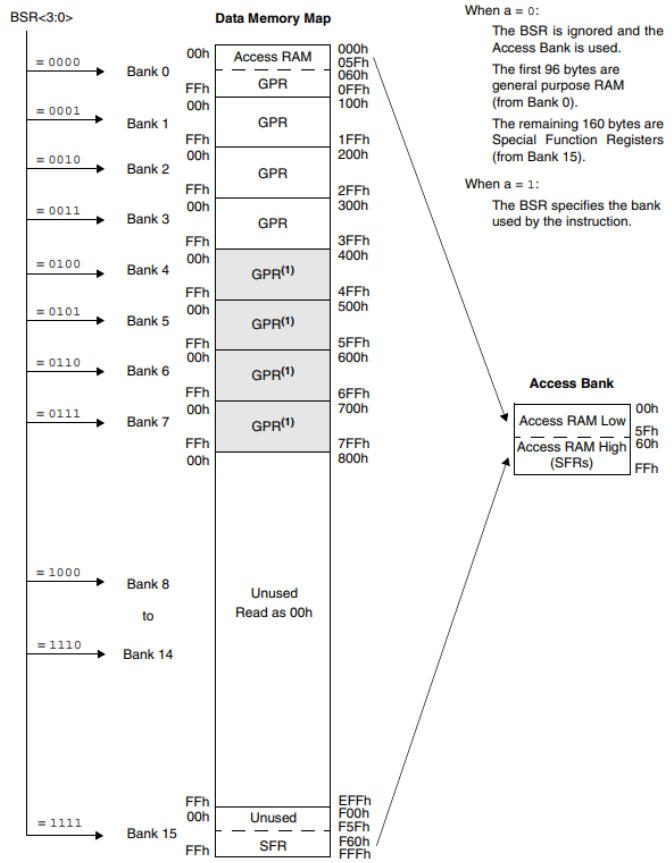
- How many data memory banks are available for the 4550 device? 15 banks
- How many of these are implemented and how many are not implemented? 9 are implemented and 7 are not implemented
- What is the size of each bank (in bytes)? 256 bytes
- What is the address range for Bank 3? 0x300 to 0x3FF

Use MPLAB X to simulate data load in PIC18F4550 through the Bank Select Register:

- Load 0xFA on address 0x100 in Bank 1
- Load 9_{10} on address 0x200 in Bank 2
- Load 0xA0 on address 0x345
- Load 156_{10} ton address 0x710
- What happens if we specify an address but do not switch to its corresponding bank?

File Registers

Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
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	FA	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	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00



Note 1: These banks also serve as RAM buffer for USB operation. See **Section 5.3.1 "USB RAM"** for more information.

EXERCISE WITH PIC18F57Q43

Open PIC18F57Q43 datasheet and go to section 9.4 Data Memory Organization

- How many data memory banks are available for the **x7Q43** device? 64 banks
- How many of these are implemented and how many are not implemented? 38 are implemented and 25 are not implemented
- What is the size of each bank (in bytes)? 256 bytes
- What is the address range for Bank 5? 0x500 to 0x5FF

Use MPLAB X to simulate data load in PIC18F57Q43 through the Bank Select Register

a) Load 0xAE to address 0x100 in Bank 1

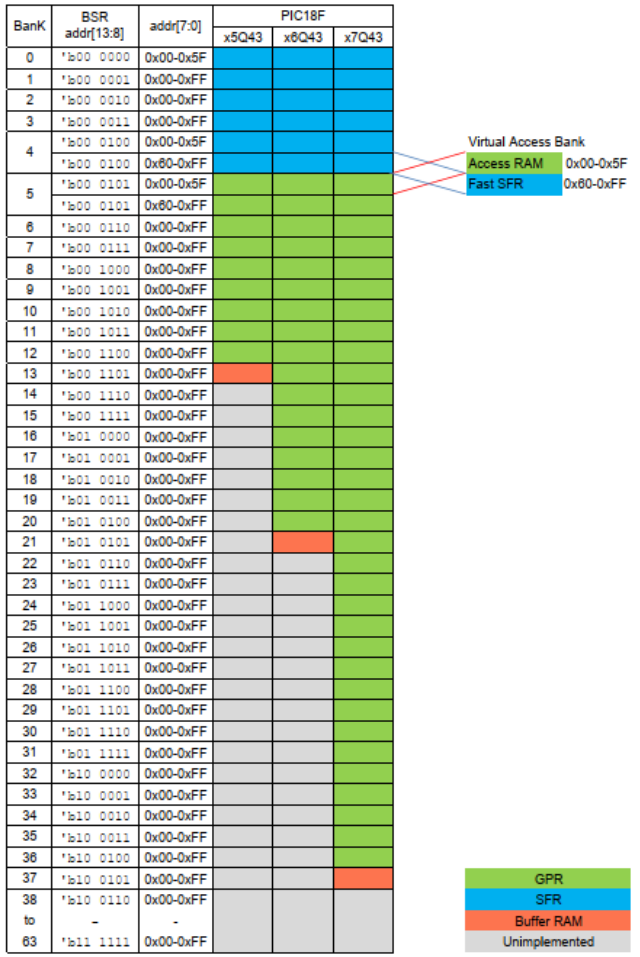
b) Load 254_{10} to address 0x200 in Bank 2

c) Load 0x03 to address 0x345

d) Load 58_{10} to address 0x2FF

e) What happens if we specify an address but do not switch to the corresponding bank?

   – Test this by loading a 0x50 into address 0x010 but staying in the same bank than the previous instruction (Bank 2)



Tecnológico de Monterrey

ACCESS BANK

The BSR is useful to address data memory; **however**, care must be taken when switching banks to make sure you write/read the correct memory location

This is why we have the **Access Bank**

- allows to access a memory block **bypassing the BSR** However, this is limited to: (for 18F57Q43)
  - Access RAM: first 96 bytes (0x00 – 0x5F) of Bank 5
  - Special SFRs: last 160 bytes (0x60 – 0xFF) of Bank 4

Instructions with Access RAM capabilities include parameter “a” as one of its bit arguments:

- `a = 1` uses BSR and an 8-bit included with the opcode
- `a = 0` ignores BSR and uses map address of Access Bank
- Access Bank addressing allows to make operations with data memory in one instruction since no BSR updating is required

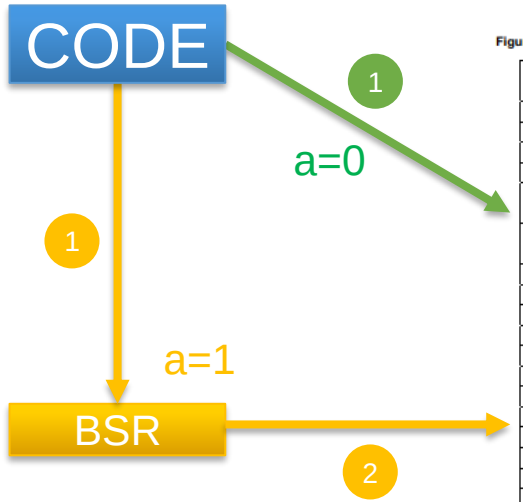


Figure 9-3. Data Memory Map

BANK	BSR addr[13:8]	addr[7:0]	PIC18F x5Q43	PIC18F x6Q43	PIC18F x7Q43
0	'b00 0000	0x00-0x5F			
1	'b00 0001	0x00-0xFF			
2	'b00 0010	0x00-0xFF			
3	'b00 0011	0x00-0xFF			
4	'b00 0100	0x00-0x5F			
	'b00 0100	0x60-0xFF			
5	'b00 0101	0x00-0x5F			
	'b00 0101	0x60-0xFF			
6	'b00 0110	0x00-0xFF			
7	'b00 0111	0x00-0xFF			
8	'b00 1000	0x00-0xFF			
9	'b00 1001	0x00-0xFF			
10	'b00 1010	0x00-0xFF			
11	'b00 1011	0x00-0xFF			
12	'b00 1100	0x00-0xFF			
13	'b00 1101	0x00-0xFF			
14	'b00 1110	0x00-0xFF			
15	'b00 1111	0x00-0xFF			
16	'b01 0000	0x00-0xFF			
17	'b01 0001	0x00-0xFF			
18	'b01 0010	0x00-0xFF			
19	'b01 0011	0x00-0xFF			
20	'b01 0100	0x00-0xFF			
21	'b01 0101	0x00-0xFF			
22	'b01 0110	0x00-0xFF			
23	'b01 0111	0x00-0xFF			
24	'b01 1000	0x00-0xFF			
25	'b01 1001	0x00-0xFF			
26	'b01 1010	0x00-0xFF			
27	'b01 1011	0x00-0xFF			
28	'b01 1100	0x00-0xFF			
29	'b01 1101	0x00-0xFF			
30	'b01 1110	0x00-0xFF			
31	'b01 1111	0x00-0xFF			
32	'b10 0000	0x00-0xFF			
33	'b10 0001	0x00-0xFF			
34	'b10 0010	0x00-0xFF			
35	'b10 0011	0x00-0xFF			
36	'b10 0100	0x00-0xFF			
37	'b10 0101	0x00-0xFF			
38 to 63	'b10 0110 – 'b11 1111	- 0x00-0xFF			

Virtual Access Bank
Access RAM 0x00-0x5F
Fast SFR 0x60-0xFF

GPR
SFR
Buffer RAM
Unimplemented

Tecnológico de Monterrey

MPLAB EXERCISE WITH PIC18F57Q43

Use MPLAB X debugger to access data memory of PIC18F57Q43 through Access Bank

a) Load 0x68 to address 0x100 in Bank 1
b) Load 194_{10} to address 0x200 in Bank 2
c) Load 0x89 to address 0x345
d) Load 58_{10} to address 0x2FF

```
PROCESSOR 18F57Q43
#include <xc.inc>

PSECT resetVec, class=CODE, reloc=2

resetVec:
    goto    main

PSECT code
main:
        ; YOUR CODE GOES HERE
end resetVec
```

Tecnológico de Monterrey

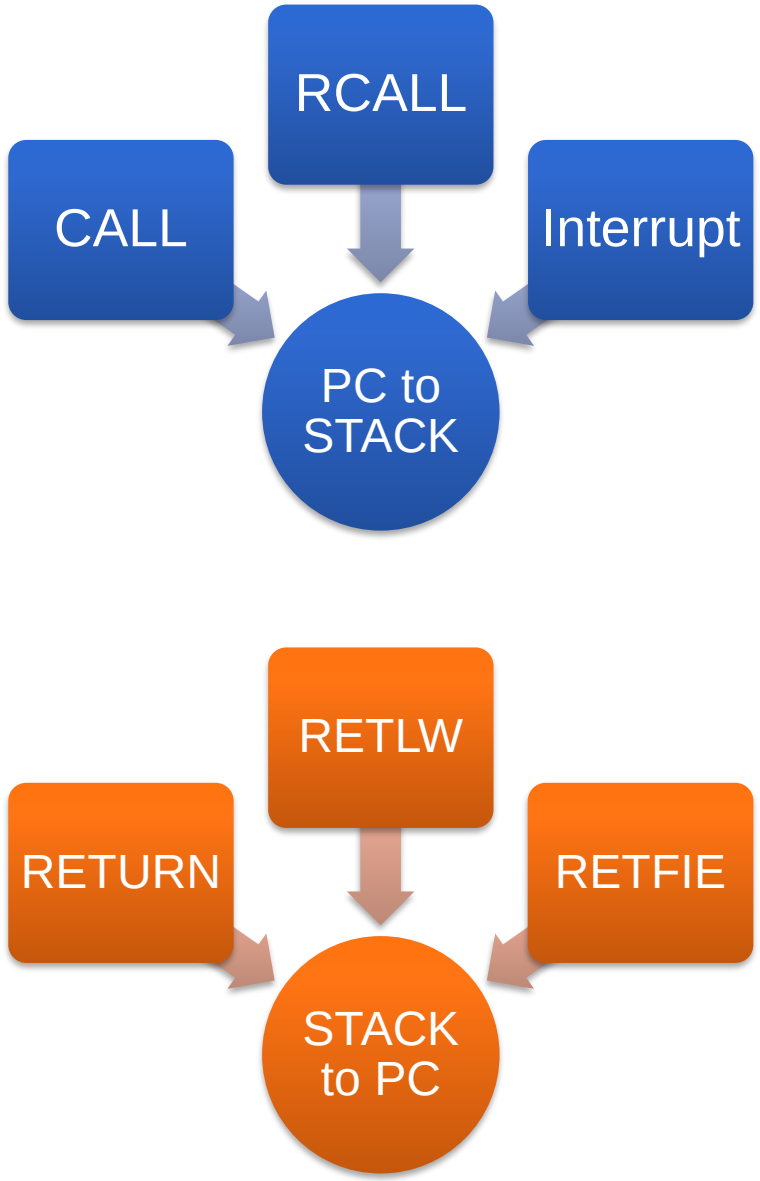
TE2015 Microcontroladores

Hardware Stack

Tecnológico de Monterrey

HARDWARE STACK

- Hardware stack is a memory we use to store return addresses after interrupts and subroutine calls
- PIC18 family accounts for 31 stack levels; however, **PIC18F57Q43 implements 127 stack levels**
- Each time a CALL or RCALL instruction is executed, or an interrupt is generated, the value on **PC is transferred to the Stack**
- Then, the value from the **Stack to the PC** is transferred back with instructions RETURN, RETLW and RETFIE



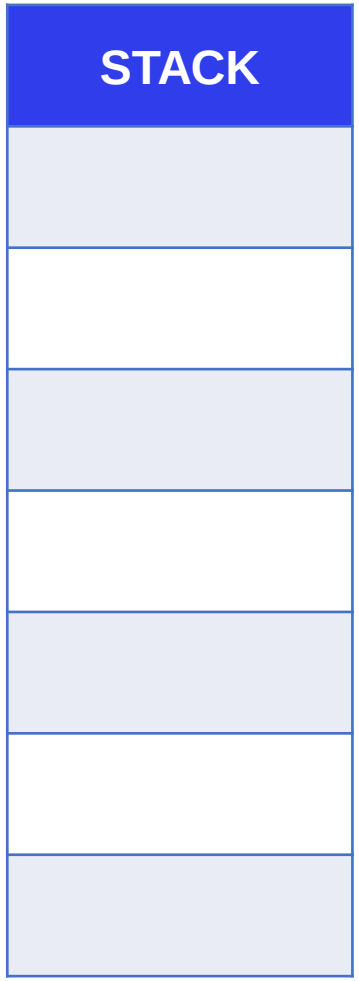
Tecnológico de Monterrey

HARDWARE STACK

PC

```
      loop:
0x00  bcf PORTD, 0
0x01  call delay
0x02  bsf PORTD, 0
0x03  call delay
0x04  goto loop

      delay:
0x0A  …
0x0B  …
0x0C  …
0x0D  return
```



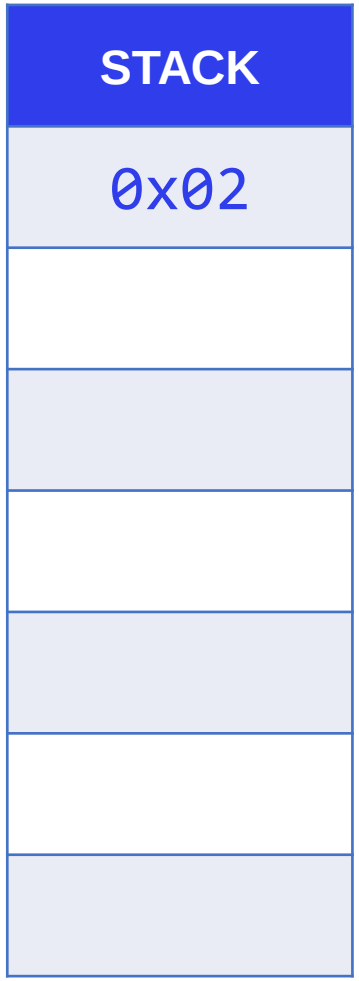
Tecnológico de Monterrey

HARDWARE STACK

```
      loop:
0x00  bcf PORTD, 0
0x01  call delay
0x02  bsf PORTD, 0
0x03  call delay
0x04  goto loop

      delay:
0x0A  …
0x0B  …
0x0C  …
0x0D  return
```

PC



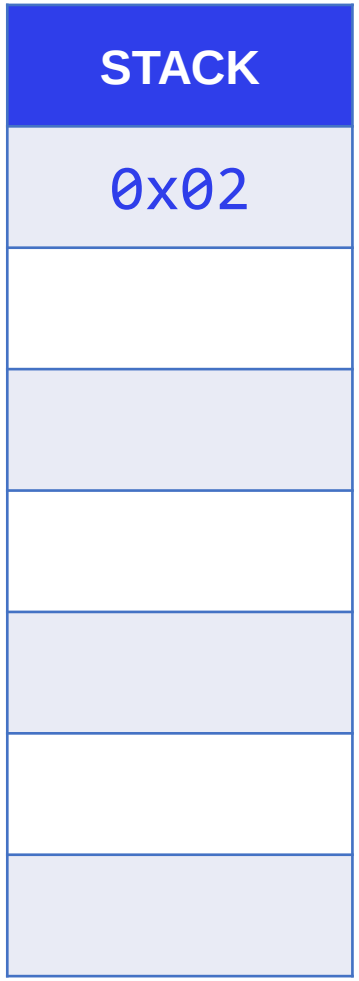
Tecnológico de Monterrey

HARDWARE STACK

```
        loop:
0x00    bcf PORTD, 0
0x01    call delay
0x02    bsf PORTD, 0
0x03    call delay
0x04    goto loop

        delay:
0x0A    …
0x0B    …
0x0C    …
0x0D    return
```

PC



Tecnológico de Monterrey

HARDWARE STACK

```
      loop:
0x00  bcf PORTD, 0
0x01  call delay
0x02  bsf PORTD, 0
0x03  call delay
0x04  goto loop

      delay:
0x0A  …
0x0B  …
0x0C  …
0x0D  return
```

PC

STACK
0x02

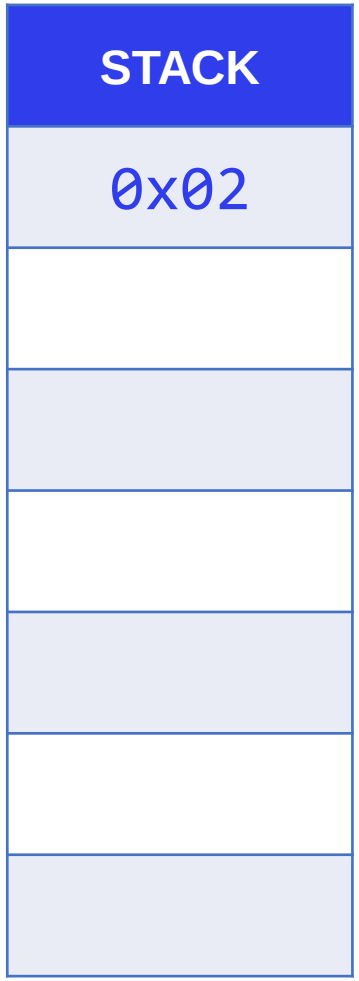
Tecnológico de Monterrey

HARDWARE STACK

```
      loop:
0x00  bcf PORTD, 0
0x01  call delay
0x02  bsf PORTD, 0
0x03  call delay
0x04  goto loop

      delay:
0x0A  …
0x0B  …
0x0C  …
0x0D  return
```

PC



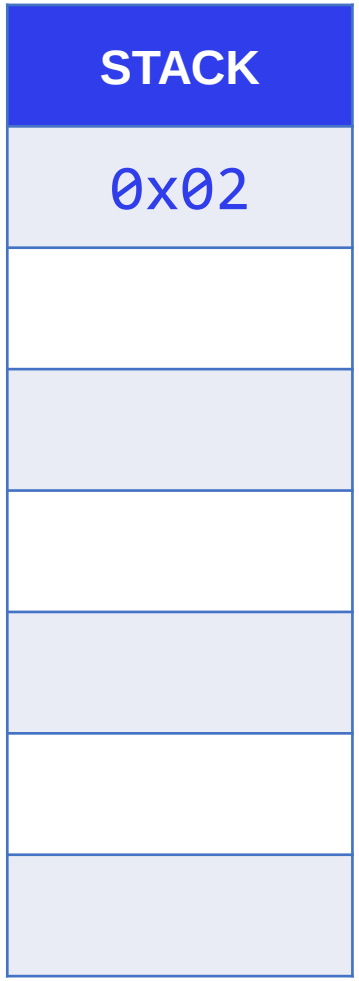
Tecnológico de Monterrey

HARDWARE STACK

```
      loop:
0x00  bcf PORTD, 0
0x01  call delay
0x02  bsf PORTD, 0
0x03  call delay
0x04  goto loop

      delay:
0x0A  …
0x0B  …
0x0C  …
0x0D  return
```

PC

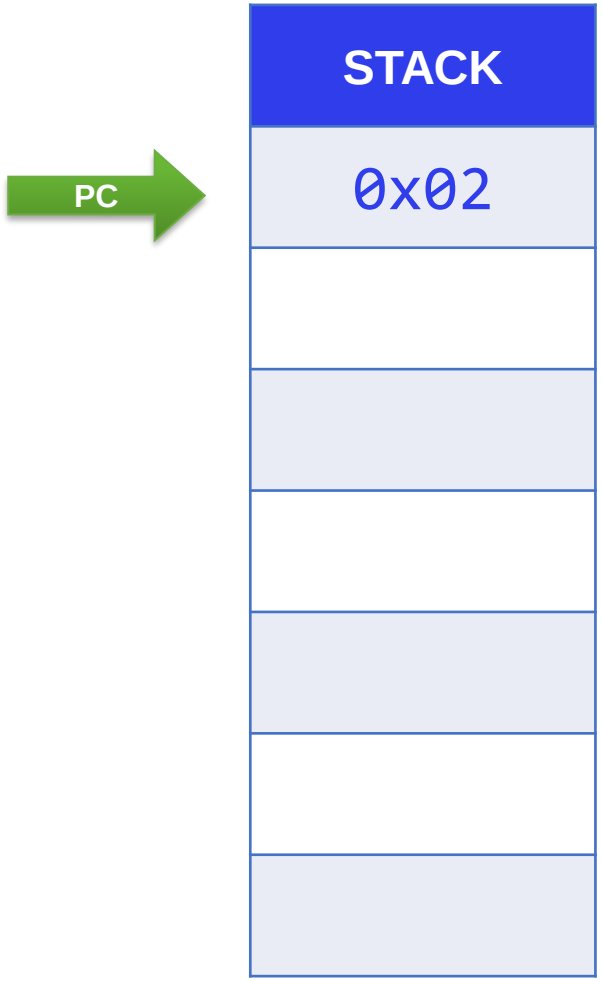


Tecnológico de Monterrey

HARDWARE STACK

```
      loop:
0x00  bcf PORTD, 0
0x01  call delay
0x02  bsf PORTD, 0
0x03  call delay
0x04  goto loop

      delay:
0x0A  …
0x0B  …
0x0C  …
0x0D  return
```



Tecnológico de Monterrey

HARDWARE STACK

```
      loop:
0x00  bcf PORTD, 0
0x01  call delay
0x02  bsf PORTD, 0
0x03  call delay
0x04  goto loop

      delay:
0x0A  …
0x0B  …
0x0C  …
0x0D  return
```

PC

STACK
0x02

Tecnológico de Monterrey

HARDWARE STACK

```
      loop:
0x00  bcf PORTD, 0
0x01  call delay
0x02  bsf PORTD, 0
0x03  call delay
0x04  goto loop

      delay:
0x0A  …
0x0B  …
0x0C  …
0x0D  return
```

PC