COE 115

Lecture 2

Microcontroller (µC) vs. Microprocessor (µP)

- µC intended as a single chip solution, µP requires external support chips (memory, interface)
- µC has on-chip non-volatile memory for program storage, µP does not.
- µC has more interface functions on-chip (serial interfaces, analog-to-digital conversion, timers, etc.) than µP
- General purpose µP are typically higher performance
- (clock speed, data width, instruction set, cache) than µCs
- Division between µP and µC becoming increasingly blurred

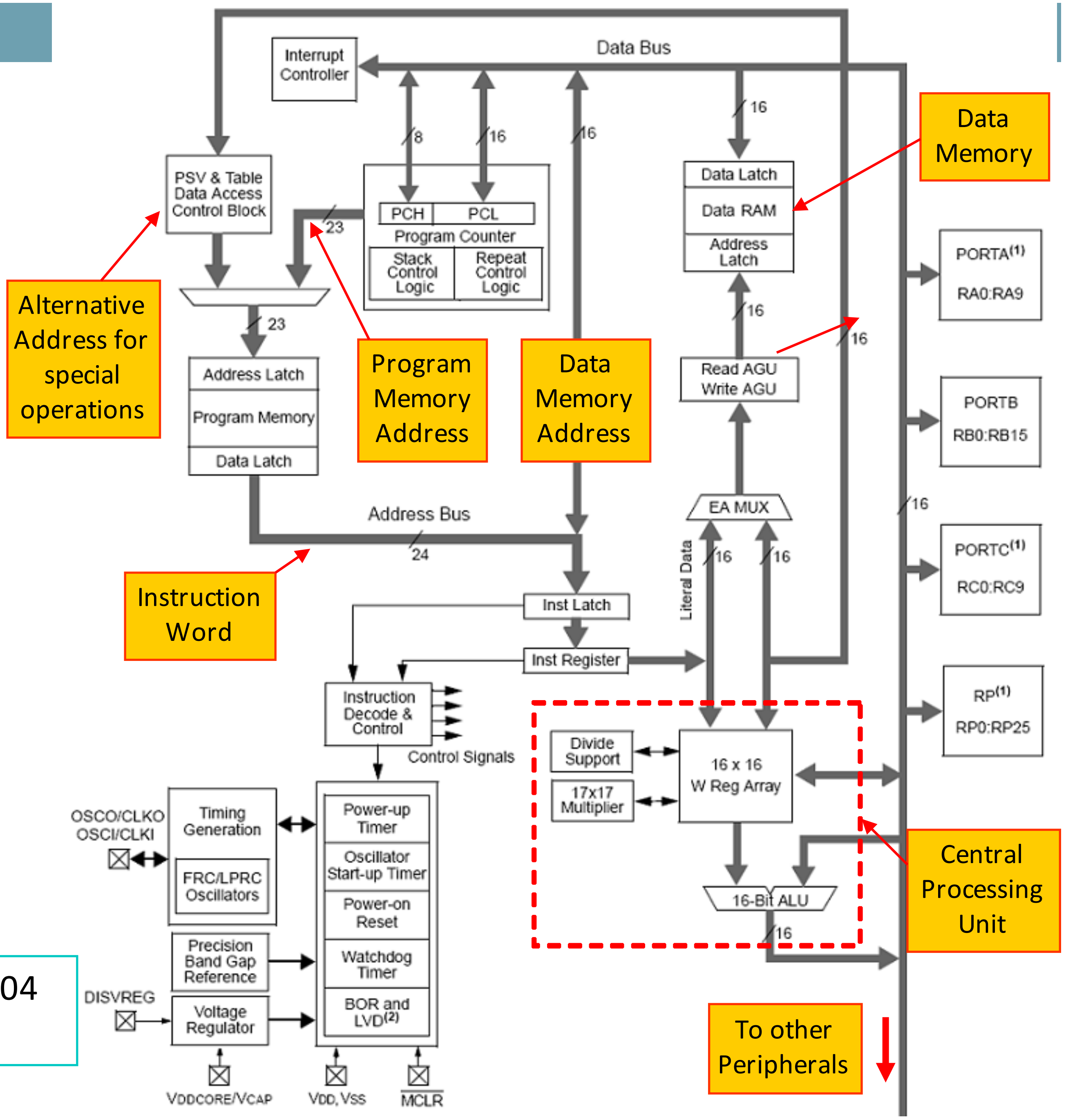
16 Bit Pic Microcontroller Overview

16-bit PIC family	Shared Features	Distinctive Features		Memory
PIC24F	same core instruction set, same peripheral set, flash program memory, same development tools, universal bit manipulation, single-cycle multiply, 32/16 and 16/16 divide support, optimised for C language, nanoWatt technology.	low cost, low power, 16MIPS at 3.3V, 2.0V to 3.6V operation, Packages from 28 to 100 pins.		to 256K program, to 16K data.
PIC24H		40MIPS at 3.3V, DMA, dual-port RAM, 3.0V to 3.6V operation, Packages from 18 to 100 pins, compatible pinouts with PIC24F.		to 256K program, to 16K data.
dsPIC30F		30MIPS at 3.3V, 2.5V to 5.5V operation, Packages from 18 to 80 pins.	*DSP Engine* added to PIC24 CPU, with DSP instructions added to instruction set.	to 144K program, to 8K data, data EEPROM.
dsPIC33F		40MIPS at 3.3V, 3.0V to 3.6V operation, Packages from 18 to 100 pins, compatible pinouts with dsPIC30F.		to 256K program, to 30K data.

Microchip PIC24 Family µC

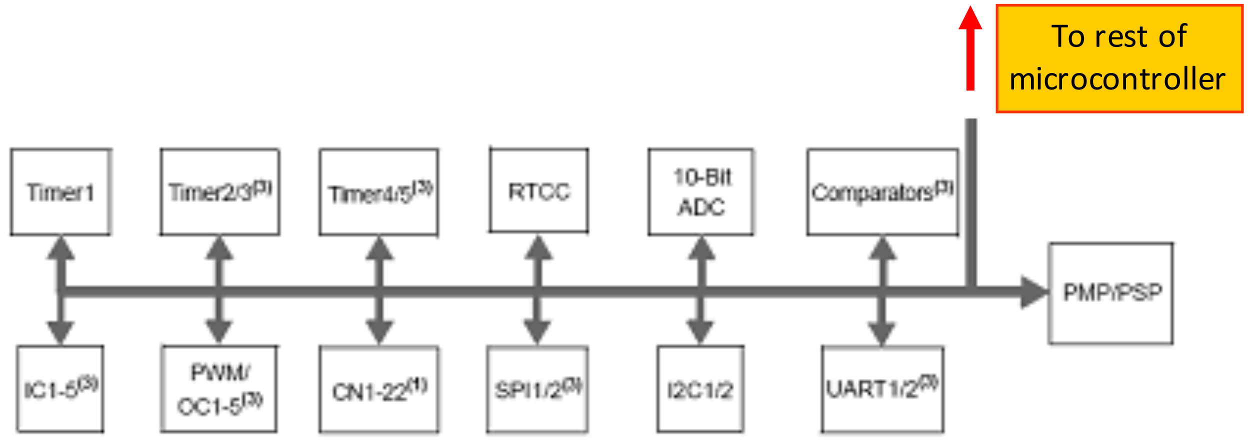
Features	Comments
Instruction width	24 bits
On-chip program memory (non-volatile, electrically erasable)	PIC24HJ32GP202 has 32Ki bytes/11264 instructions, architecture supports 24Mibytes/4Mi instructions)
On-chip Random Access Memory (RAM), volatile)	PIC24HJ32GP202 has 2048 bytes, architecture supports up 65536 bytes
Clock speed	DC to 80 MHz
16-bit Architecture	General purpose registers, 71 instructions not including addressing mode variants
On-chip modules	Async serial IO, I2C, SPI, A/D, three 16- bit timers, one 8-bit timer, comparator

The PIC24F family



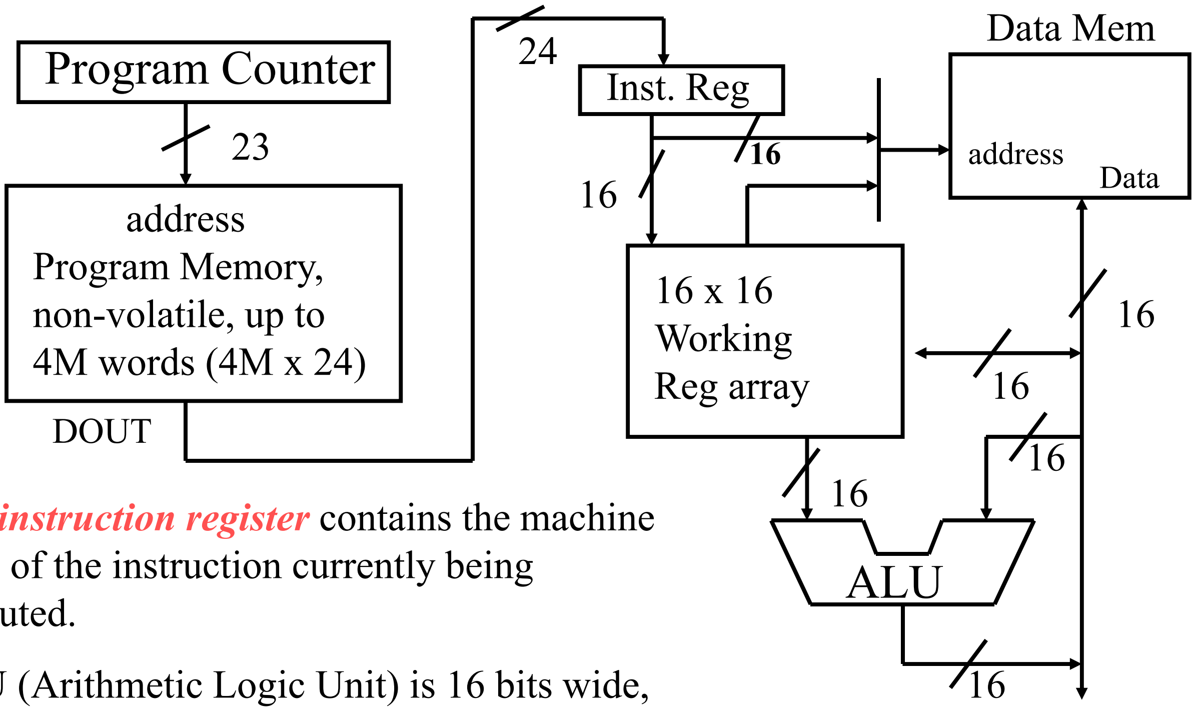
The PIC24FJ64GA004 block diagram

The PIC24F family



The PIC24FJ64GA004 peripherals block diagram

PIC24 Core (Simplified Block Diagram)


Program Counter
23
address
Program Memory, non-volatile, up to 4M words (4M x 24)
DOUT
24
Inst. Reg
16
16
Data Mem
address
Data
16 x 16 Working Reg array
16
16
16
16
ALU
16

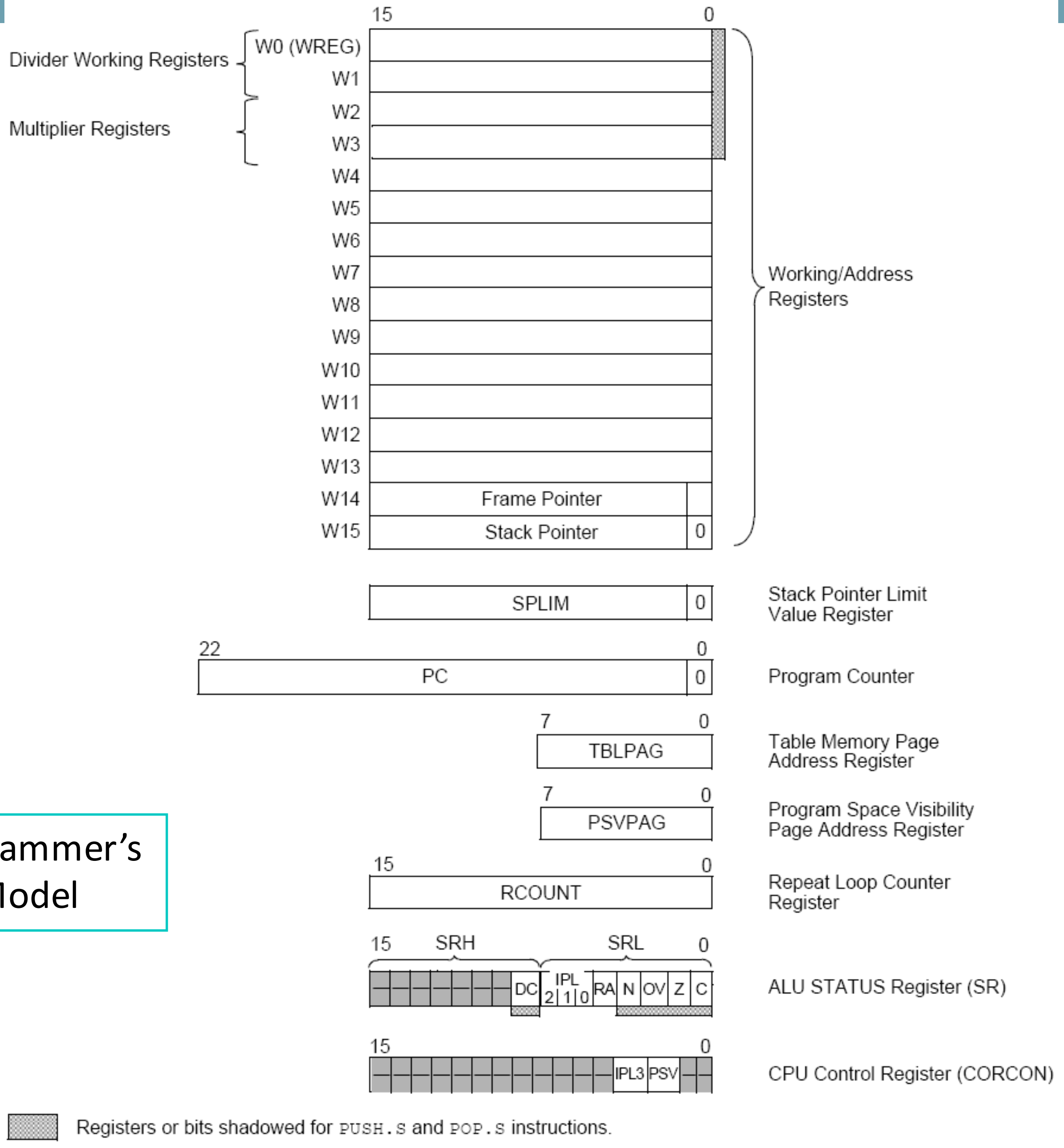
The ***instruction register*** contains the machine code of the instruction currently being executed.

ALU (Arithmetic Logic Unit) is 16 bits wide, can accept as operands working registers or data memory.

17 x 17 Multiplier not shown

PIC24 CPU

Programmer's Model


15
0
Divider Working Registers
Multiplier Registers
W0 (WREG)
W1
W2
W3
W4
W5
W6
W7
W8
W9
W10
W11
W12
W13
W14
W15
Frame Pointer
Stack Pointer
0
Working/Address Registers
SPLIM
0
Stack Pointer Limit Value Register
22
0
PC
0
Program Counter
7
0
TBLPAG
Table Memory Page Address Register
7
0
PSVPAG
Program Space Visibility Page Address Register
15
0
RCOUNT
Repeat Loop Counter Register
15
SRH
SRL
0
DC
IPL
2 1 0
RA
N
OV
Z
C
ALU STATUS Register (SR)
15
0
IPL3
PSV
CPU Control Register (CORCON)
Registers or bits shadowed for PUSH.S and POP.S instructions.

PIC24 Memory Organization

- Program Memory and Data Memory
- Program Memory
  - non-volatile (contents are retained when power is lost).
  - Stores instructions, 24 bits wide (3 bytes)
  - Architecture can support 4M instructions
    - PIC24HJ32GP202 program memory supports 1164
- Data Memory
  - File registers
  - Maximum size of 65536 x 8
  - volatile (contents are lost when power is lost).

Program Memory

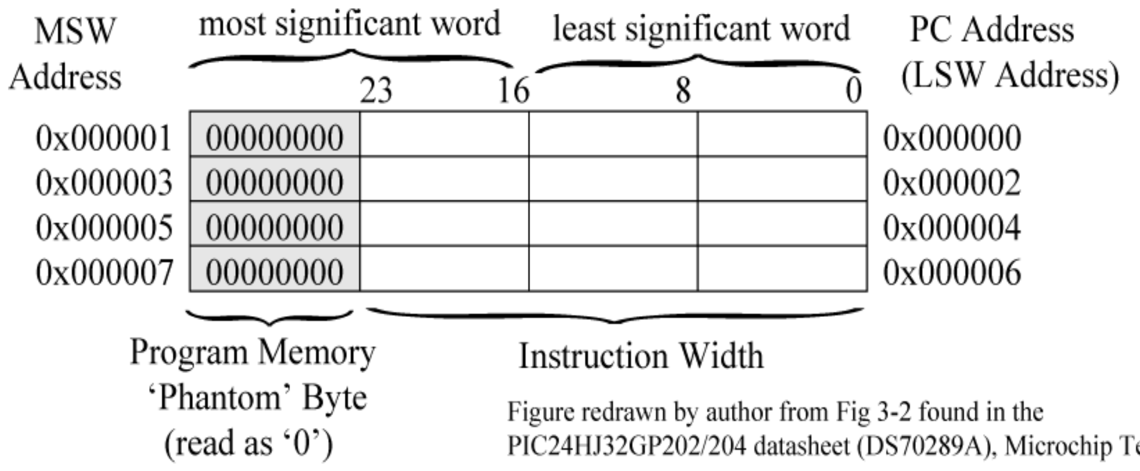
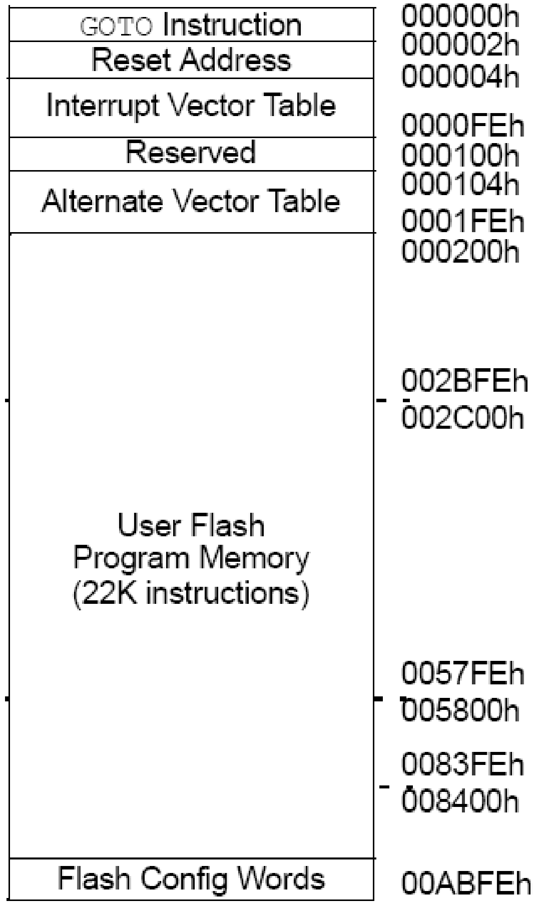


Figure redrawn by author from Fig 3-2 found in the PIC24HJ32GP202/204 datasheet (DS70289A), Microchip Technology Inc.

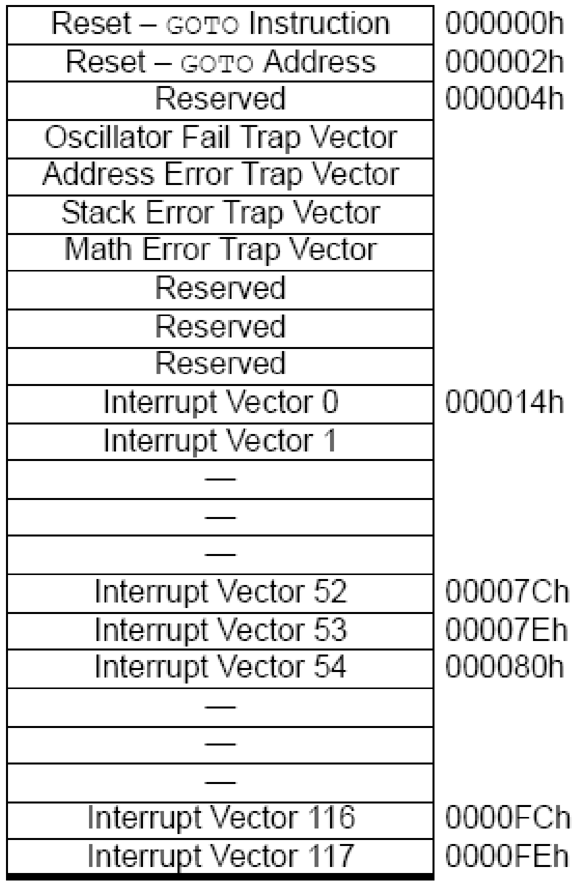
PC is 23 bits wide, but instructions start on even word boundaries (the PC least significant bit is always 0), so the PC can address 4 Mi instructions.

Locations 0x000000- 0x0001FF reserved, User program starts at location 0x000200.

Program Memory and Interrupt Vectors



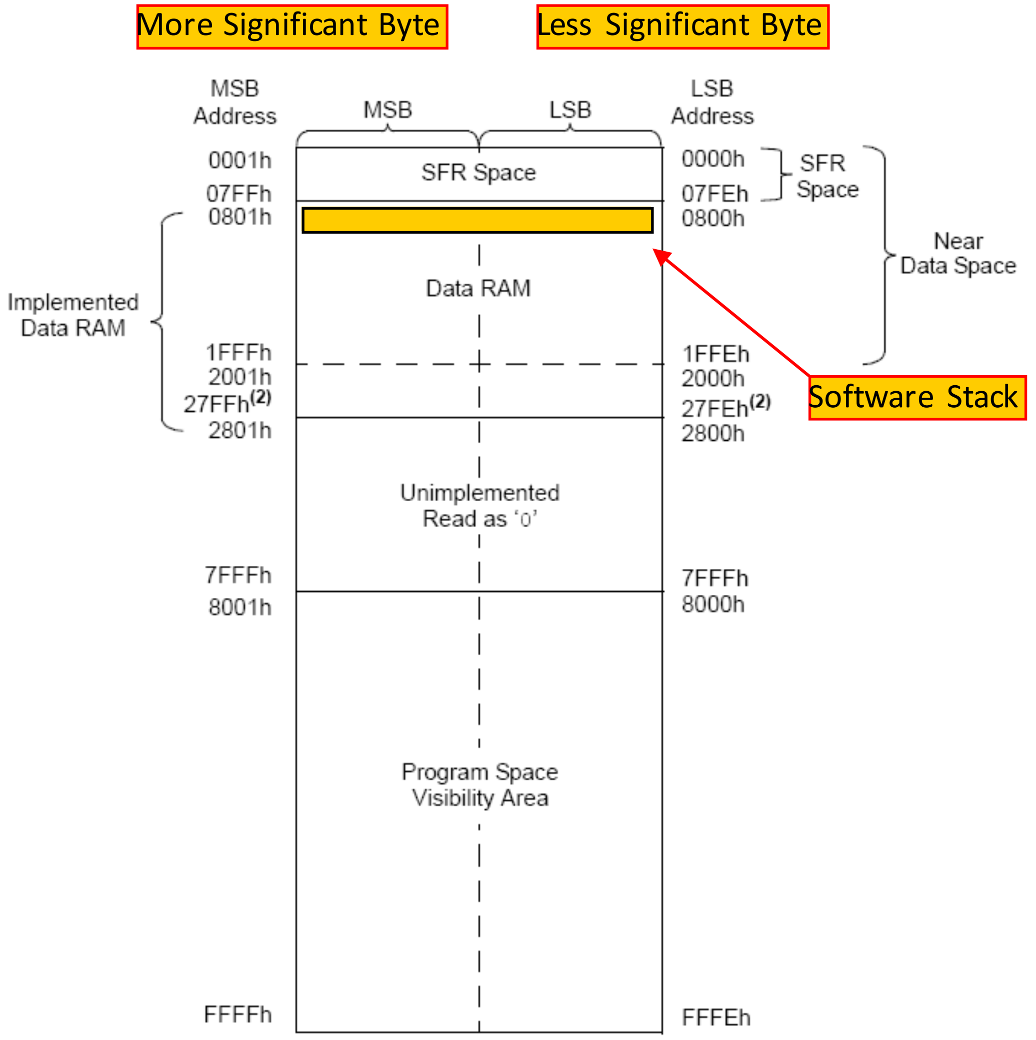
Program Memory PIC24FJ64GA004



Reset – GOTO Instruction	000000h
Reset – GOTO Address	000002h
Reserved	000004h
Oscillator Fail Trap Vector	
Address Error Trap Vector	
Stack Error Trap Vector	
Math Error Trap Vector	
Reserved	
Reserved	
Reserved	
Interrupt Vector 0	000014h
Interrupt Vector 1	
—	
—	
—	
Interrupt Vector 52	00007Ch
Interrupt Vector 53	00007Eh
Interrupt Vector 54	000080h
—	
—	
—	
Interrupt Vector 116	0000FCh
Interrupt Vector 117	0000FEh

Interrupt Vector Table

Data Memory



Note 1: Data memory areas are not shown to scale.
2: Upper memory limit for PIC24FJ16GAXXX devices is 17FFh.

Data Memory Organization for PIC24HJ32GP202

- 32 KB program memory
- 2 KB RAM

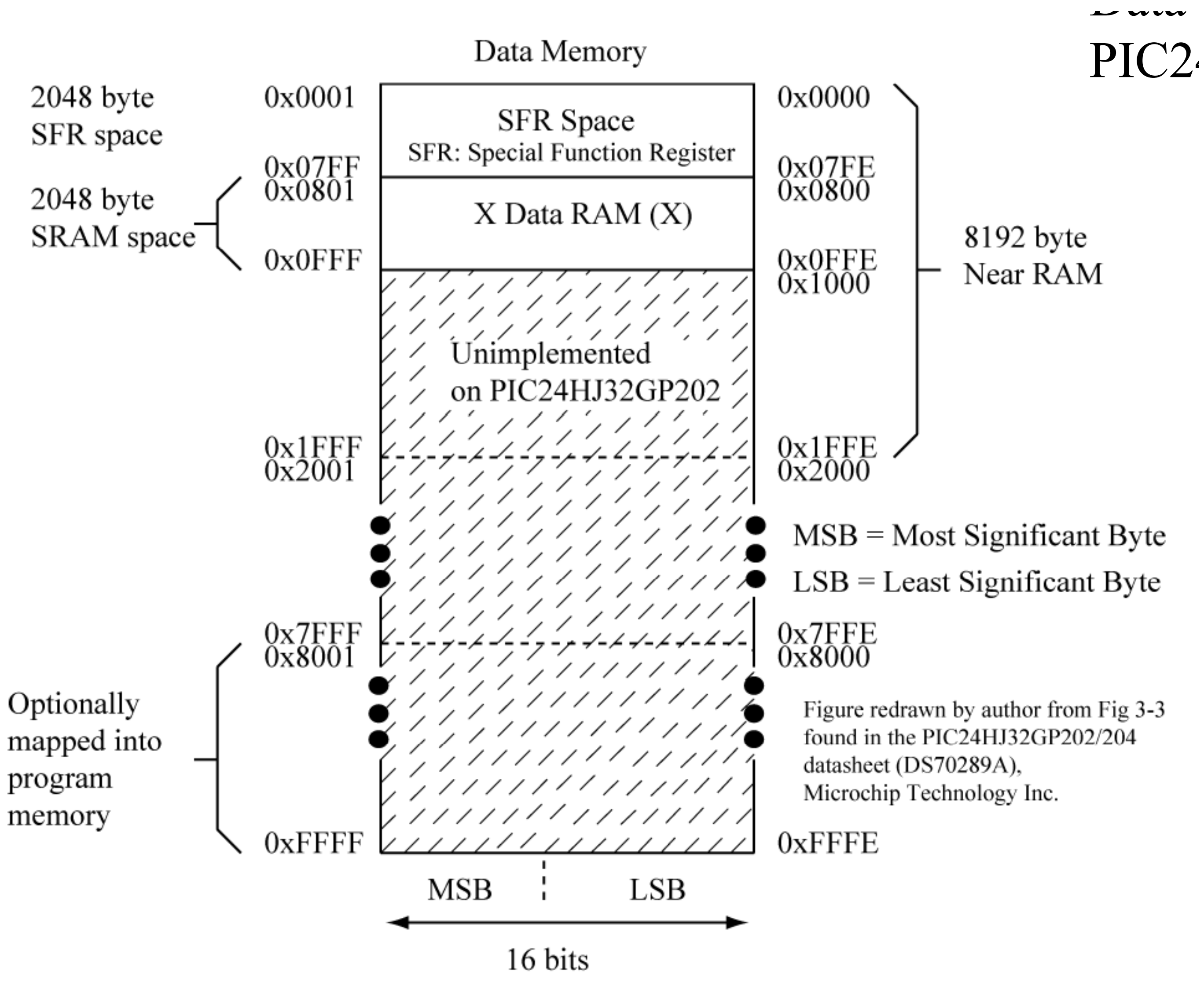
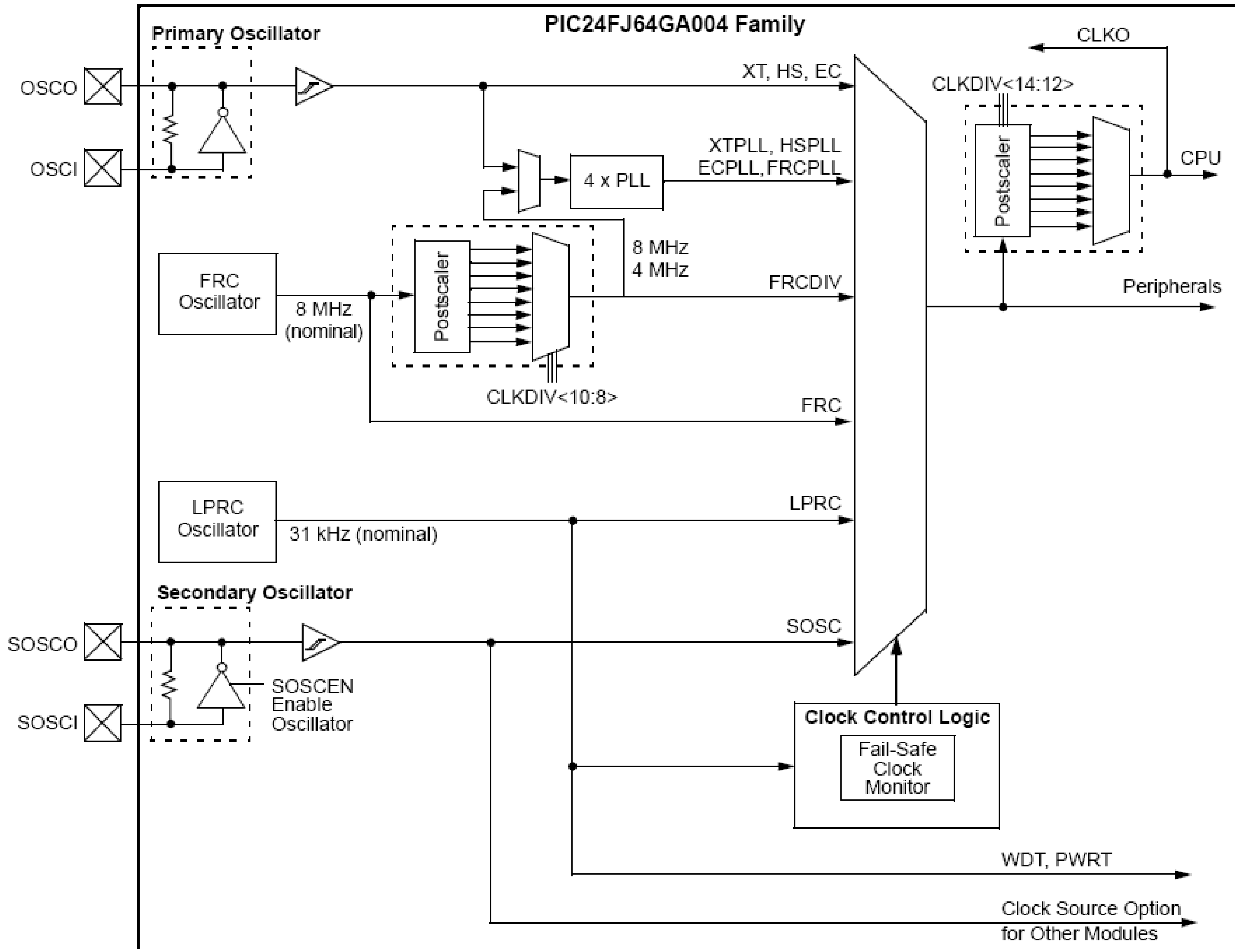


Figure redrawn by author from Fig 3-3 found in the PIC24HJ32GP202/204 datasheet (DS70289A), Microchip Technology Inc.

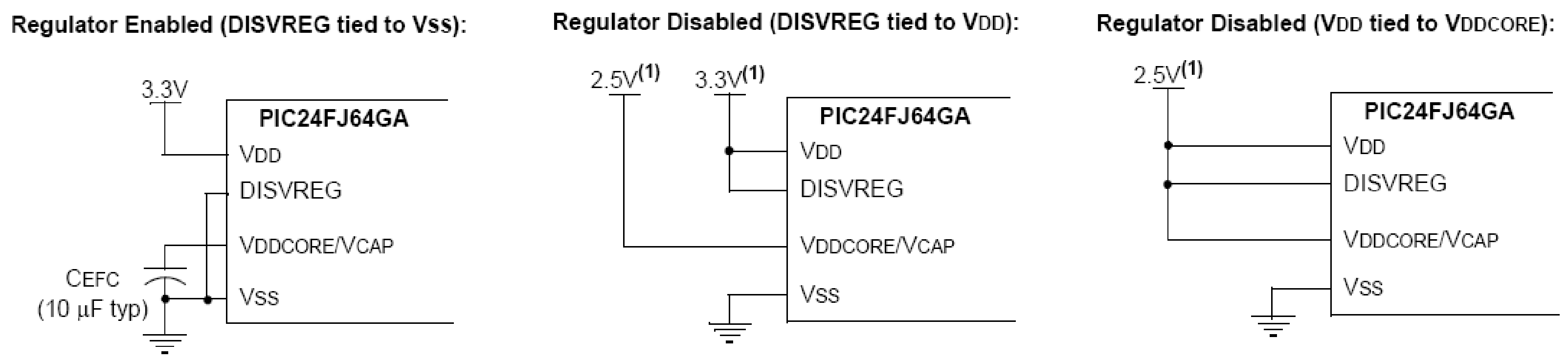
Clock Sources



Power Supply

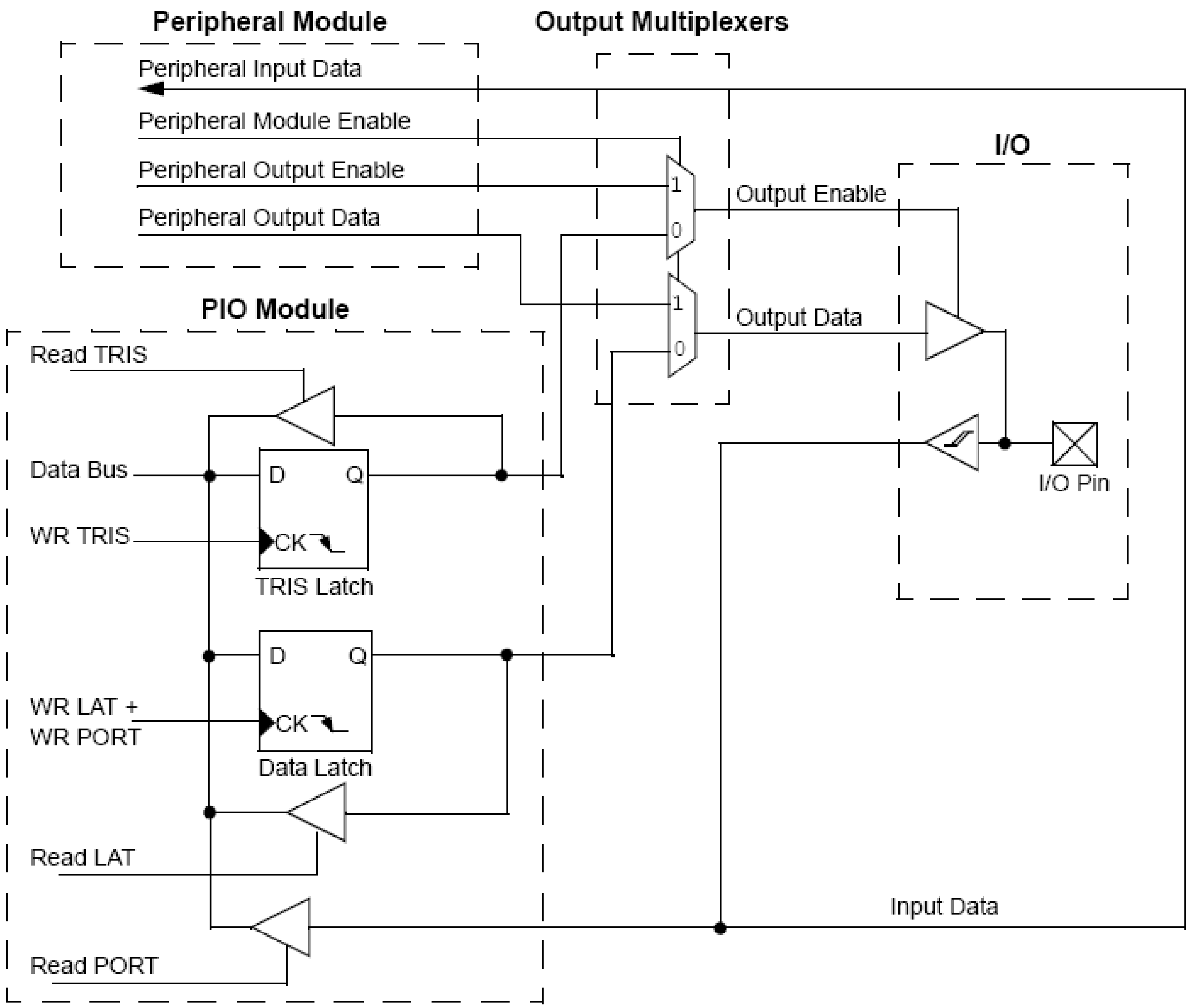
Supply Voltage	**Minimum** (operating frequency restricted)	**Maximum**	**Absolute Maximum**
V_{DDCORE} with respect to V_{SS}	2.0V	2.75V	3.0V
V_{DD} with respect to V_{SS}	the higher of: 2.0V, or V_{DDCORE}	3.6V	4.0V

Power Supply Voltages



Power Supply Modes

PIC24 Pins and Ports



PIC 24 Peripherals

Peripheral	**Brief description**
Timer 1	16-bit timer with connections for external crystal (SOSC0/1). Built-in digital comparator and period register allow easy set up of periodic interrupts.
Timer 2/3	Can be configured as a single 32-bit timer, or as two 16-bit timers, both options with built-in comparator and period register.
Timer 4/5	As Timer 2/3.
Input Capture	Captures a Timer value (Timer 2 or 3) at instant of selected edge applied to input pin. Input can be optionally divided by 4 or 16. Captured values can be held in a 4-level FIFO buffer.
Output Compare	Generates an output pulse when the value of a selected Timer is equal to a compare register, with added option of interrupt on compare match.

PIC 24 Peripherals

Peripheral	**Brief description**
Serial Peripheral Interface (SPI)	Operates in 8-bit or 16-bit mode (both receive and transmit), with 8-level receive and transmit buffers.
I^{2}C	I^{2}C module with independent Master and Slave logic, supporting 100kHz and 400kHz bit rates.
UART	8 or 9-bit UART, with 4-level receive and transmit buffers, and support for LIN and IrDA.
Parallel Master Port/ Parallel Slave Port	Highly configurable 8-bit i/o parallel port with up to 16 bits of address, suitable for interfacing with conventional addressed parallel buses, configurable also as slave port.
Real Time Clock and Calendar	Calendar and clock with 1 second resolution, suitable for timing over long durations, optimised for low power applications.
ADC	10-bit ADC, with up to 16 analog inputs, and up to 500Ksamples per second, 16 word results buffer.
Comparator	Highly configurable analogue comparators, with a scalable voltage reference.

PIC 24 Summary

- **Harvard – RISC**
- **16 working registers**
- **16 bit registers**
- **Remappable pins**
- **Power supply options and Clock options**

Data Memory

Special Function Registers

- **Special Function Registers (SFR)**
  - addressed like normal data memory locations but have specified functionality tied to hardware subsystems in the processor.
  - SFRs by name (W0, T3CON, STATUS, etc) instead of by address.
- **SFRs in the PIC24 µC**
  - used as control registers and data registers for processor subsystems (like the serial interface, or the analog- to-digital converter).
- **SFRs address range**
  - 0x0000 to 0x07FE in data memory.
  - datasheet for a complete list of SFRs.
- **Other locations in data memory that are not SFRs**
  - used for storage of temporary data; they are not used by the processor subsystems.
  - These are sometimes referred to as GPRs (general purpose registers).
  - MPLAB refers to these locations as file registers.

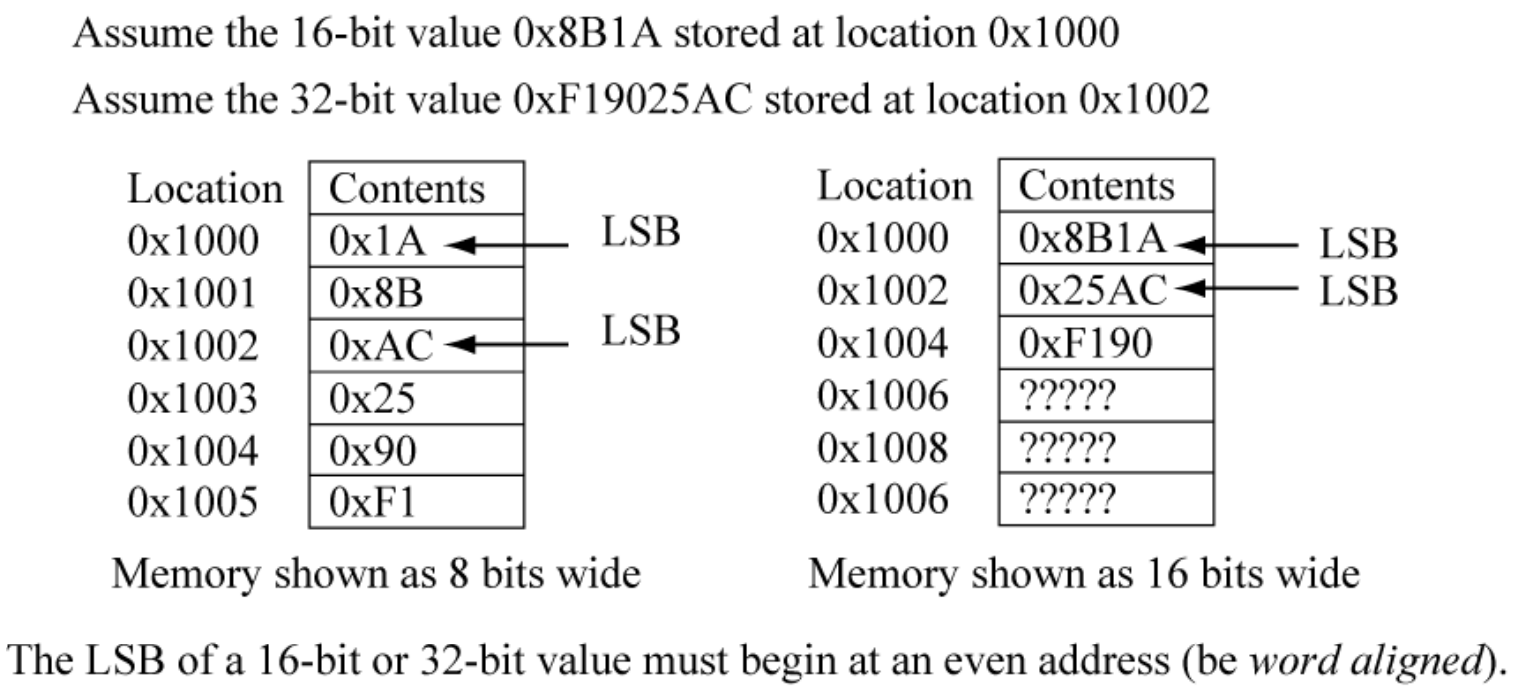
8-Bit , 16-Bit, 32-Bit Data

- **Data width**
  - 8 bits (byte), 16 bits (2 bytes), and 32 bits (4 bytes) in size

Size	**Unsigned Range**
8-bits	0 to 28-1 (0 to 255, 0 to 0xFF)
16-bit	0 to 216-1 (0 to 65536, 0 to 0xFFFF)
32-bit	to 232-1 (0 to 4,294,967,295), 0 to 0xFFFFFFFF)

Storing Multi Byte Values in Memory

- **16-bit and 32-bit values**
  - stored in memory from least significant byte to most significant byte,
  - in increasing memory locations (little endian order).



Assume the 16-bit value 0x8B1A stored at location 0x1000

Assume the 32-bit value 0xF19025AC stored at location 0x1002

Location	Contents	
0x1000	0x1A	LSB
0x1001	0x8B	
0x1002	0xAC	LSB
0x1003	0x25	
0x1004	0x90	
0x1005	0xF1	

Memory shown as 8 bits wide

Location	Contents	
0x1000	0x8B1A	LSB
0x1002	0x25AC	LSB
0x1004	0xF190	
0x1006	?????	
0x1008	?????	
0x1006	?????	

Memory shown as 16 bits wide

The LSB of a 16-bit or 32-bit value must begin at an even address (be *word aligned*).

Data Transfer Instruction

- Copies data from Source (src) location to Destination (dst) Location

(src) → dst '()' read as 'contents of'

- This operation uses *two operands*.
- The method by which an operand ADDRESS is specified is called the *addressing mode*.
- There are many different addressing modes for the PIC24.
- We will use a very limited number of addressing modes in our initial examples.

Data Transfer Instruction Summary

Dest / Source	Memory	Register direct	Register indirect
Literal	X	MOV{.B} #lit8/16, Wnd *lit → Wnd*	X
Memory	X	MOV f_{ALL}, Wnd MOV{.B} f, {WREG} *$(f_{\{ALL\}}) \rightarrow$ Wnd/WREG*	X
Register direct	MOV Wns, f_{ALL} MOV{.B} WREG, f *(Wns/WREG) $\rightarrow f_{\{ALL\}}$*	MOV{.B} Wso, Wdo *(Wso) → Wdo*	MOV{.B} Wso, [Wdo] *(Wso) → (Wdo)*
Register indirect	X	MOV{.B} [Wso], Wdo *((Wso)) → Wdo*	MOV{.B} [Wso], [Wdo] *((Wso)) → (Wdo)*

Key:

MOV{.B} #lit8/16, Wnd	PIC24 assembly
lit → Wnd	Data transfer

Yellow shows varying forms of the same instruction

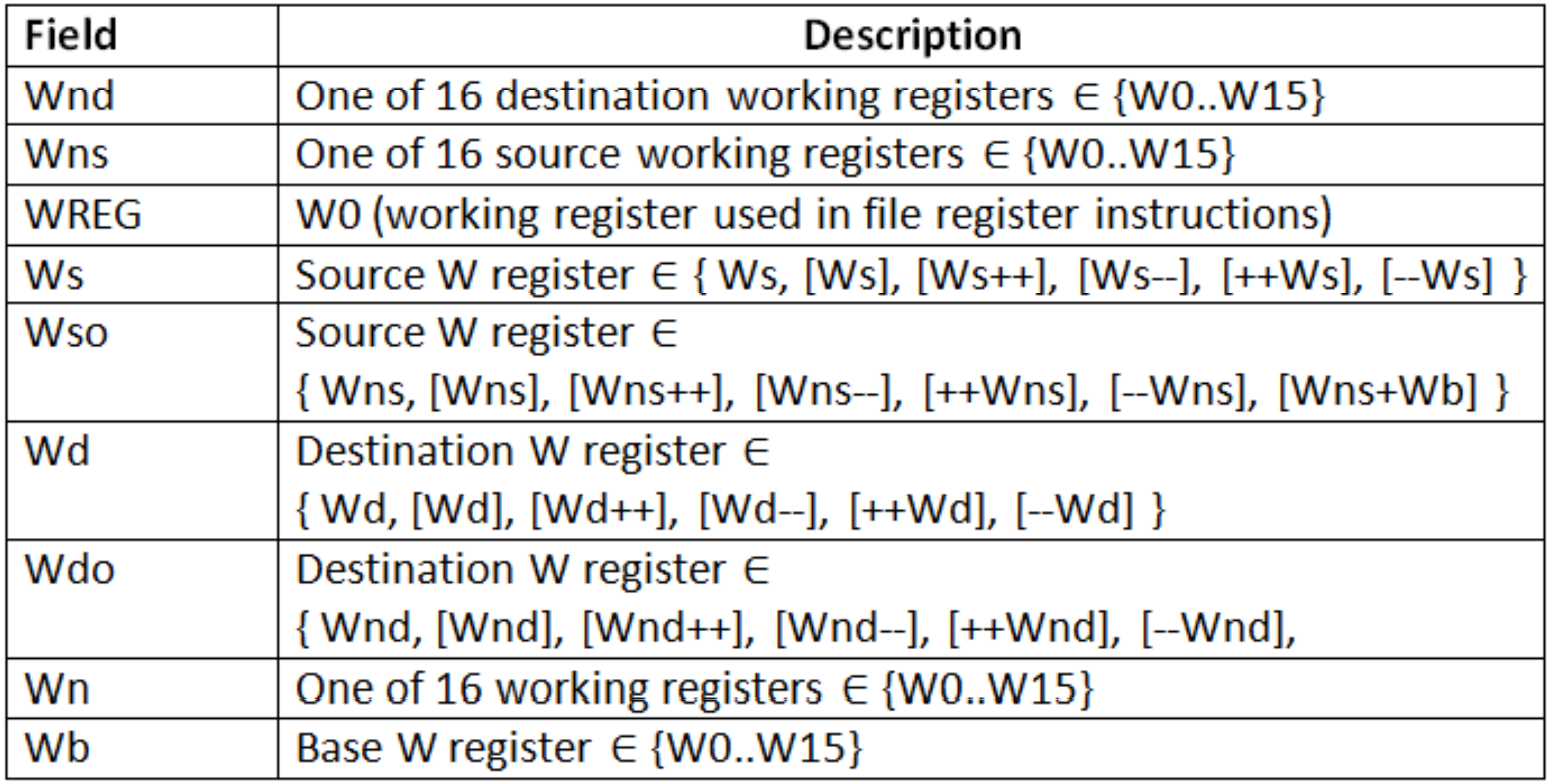
f: near memory (0…8095)

f_{ALL}: all of memory (0…65534)

Wso, Wsd, Wn

- **MOV Wso, Wdo**

Symbols used in opcode descriptions



Field	Description
Wnd	One of 16 destination working registers ∈ {W0..W15}
Wns	One of 16 source working registers ∈ {W0..W15}
WREG	W0 (working register used in file register instructions)
Ws	Source W register ∈ { Ws, [Ws], [Ws++], [Ws--], [++Ws], [--Ws] }
Wso	Source W register ∈ { Wns, [Wns], [Wns++], [Wns--], [++Wns], [--Wns], [Wns+Wb] }
Wd	Destination W register ∈ { Wd, [Wd], [Wd++], [Wd--], [++Wd], [--Wd] }
Wdo	Destination W register ∈ { Wnd, [Wnd], [Wnd++], [Wnd--], [++Wnd], [--Wnd],
Wn	One of 16 working registers ∈ {W0..W15}
Wb	Base W register ∈ {W0..W15}

MOV{.B} Wso, Wdo Instruction

- **“Copy contents of Wso register to Wdo register” General form:**

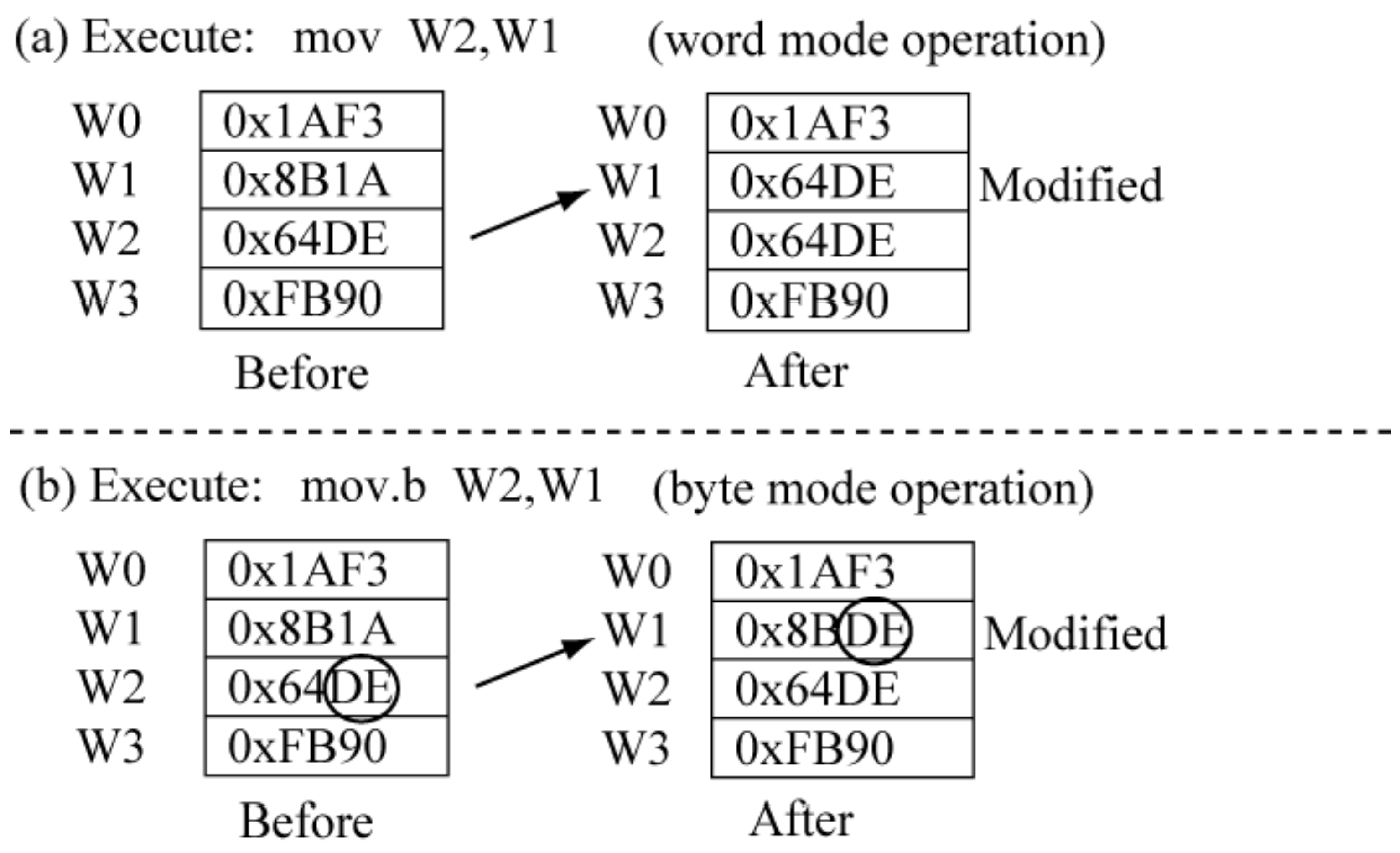
```
mov{.b} Wso, Wdo      (Wso) → Wdo
```

  - Wso is one of the 16 working registers W0 through W15 (‘s’ indicates Wso is an operand source register for the operation).
  - Wdo is one of the 16 working registers W0 through W15 (‘d’ indicates Wdo is an operand destination register for the operation).

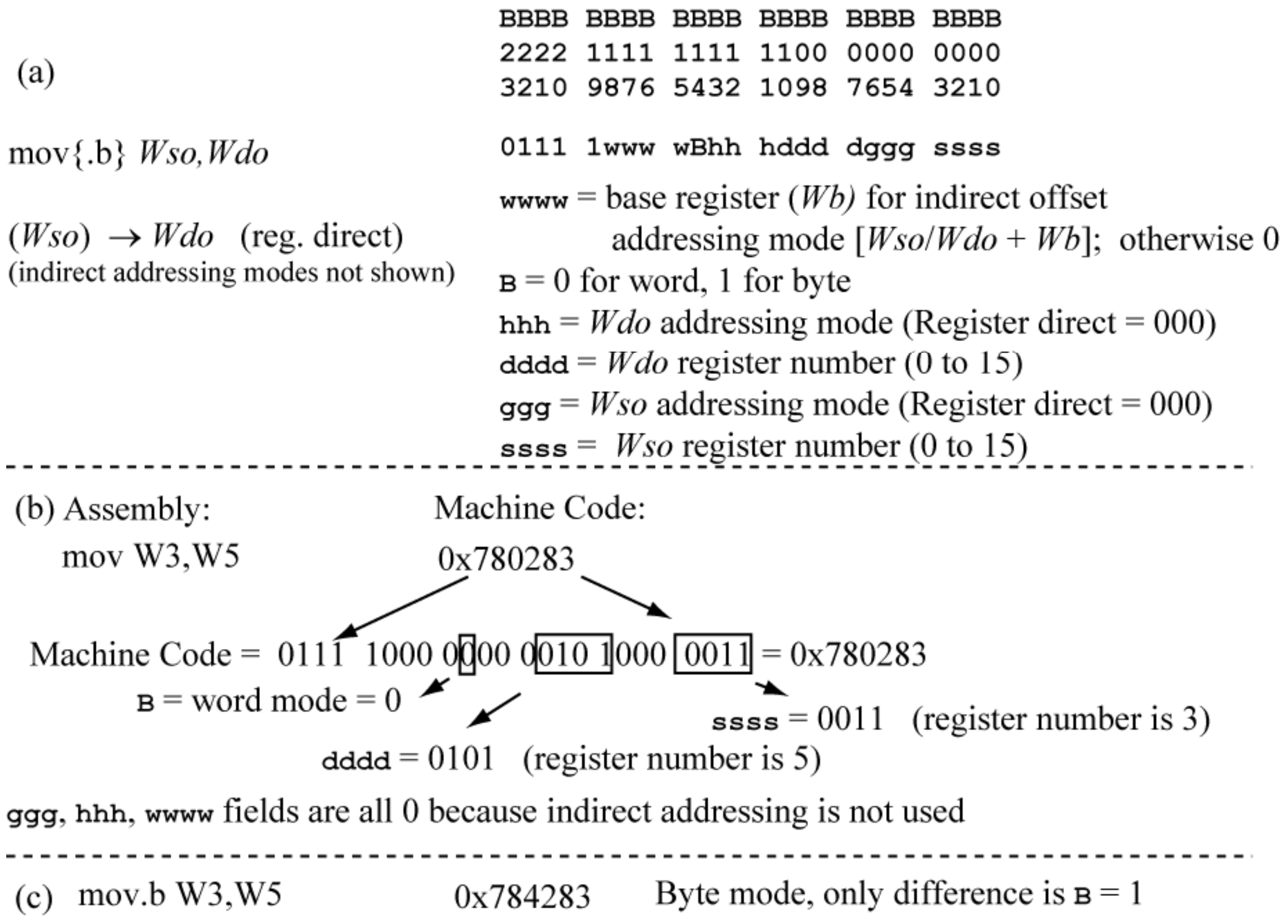
```
mov W3, W5         (W3) → W5 (word operation)
mov.b W3, W5       (W3.LSB) → W5.LSB (byte operation)
```

  - Contents of working register W3 copied to working register W5.
  - This can either be a word or byte operation. The term ‘copy’ is used here instead of ‘move’ to emphasize that Wso is left unaffected by the operation.
  - The addressing mode used for both the source and destination operands is called *register direct*. The *mov* instruction supports other addressing modes which are not shown.

MOV{.B} Wso, Wdo Instruction Execution



MOV Wso, Wdo Instruction Format



(a)

```
BBBB BBBB BBBB BBBB BBBB BBBB
2222 1111 1111 1100 0000 0000
3210 9876 5432 1098 7654 3210
```

mov{.b} *Wso,Wdo*

```
0111 1www wBhh hddd dggg ssss
```

(*Wso*) → *Wdo* (reg. direct)
(indirect addressing modes not shown)

`wwww` = base register (*Wb)* for indirect offset addressing mode [*Wso*/*Wdo* + *Wb*]; otherwise 0

`B` = 0 for word, 1 for byte

`hhh` = *Wdo* addressing mode (Register direct = 000)

`dddd` = *Wdo* register number (0 to 15)

`ggg` = *Wso* addressing mode (Register direct = 000)

`ssss` = *Wso* register number (0 to 15)

(b) Assembly: mov W3,W5

Machine Code: 0x780283

Machine Code = 0111 1000 0$\boxed{0}$00 0$\boxed{010\ 1}$000 $\boxed{0011}$ = 0x780283

`B` = word mode = 0

`dddd` = 0101 (register number is 5)

`ssss` = 0011 (register number is 3)

`ggg`, `hhh`, `wwww` fields are all 0 because indirect addressing is not used

(c) mov.b W3,W5 0x784283 Byte mode, only difference is `B` = 1

MOV Wns, f Instruction

"Copy content of Wns register to data memory location *f* ".

General form:

MOV Wns,f $\quad\quad$ (Wns) → f

f is a location in data memory, Wns is one of the 16 working registers W0 through W15 ('s' indicates Wns is an operand source register for the operation)

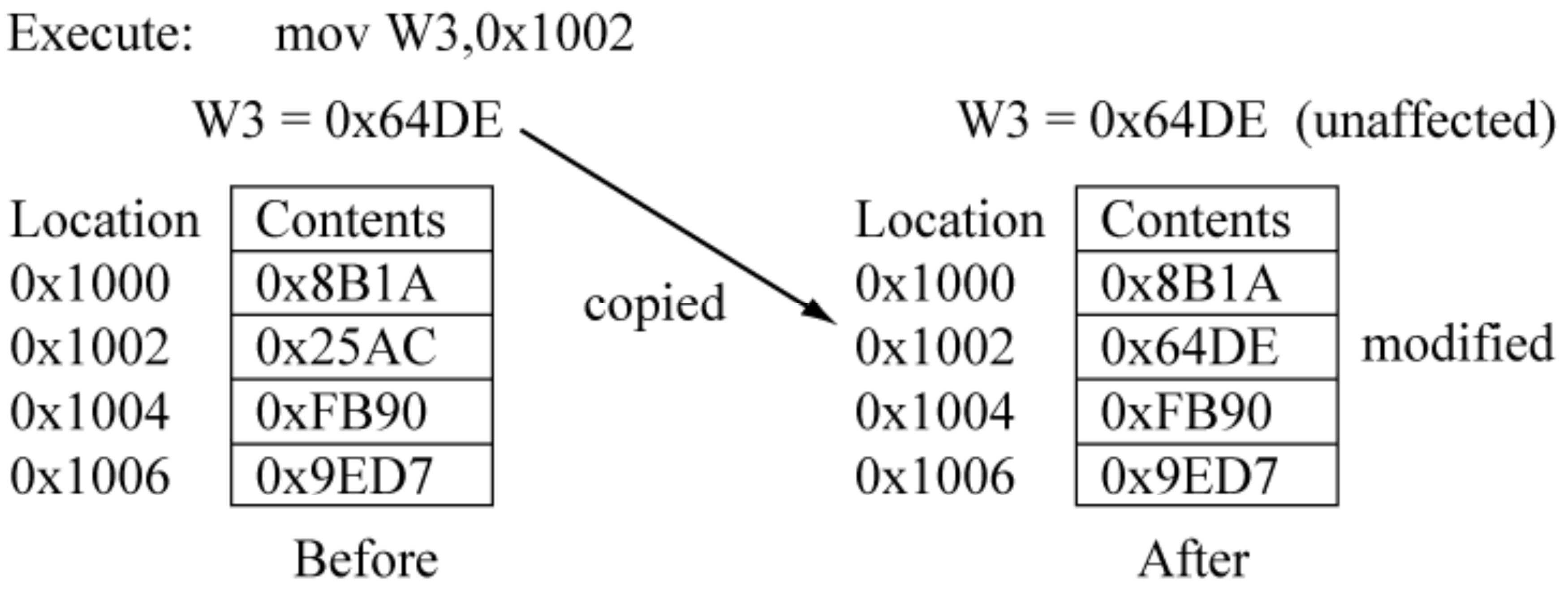
MOV W3, 0x1000 $\quad\quad$ (W3) → 0x1000

Contents of register W3 copied to data memory location 0x1000.
This instruction supports only WORD operations:

The source operand uses *register direct addressing,* while the destination operand uses *file register* addressing.

File registers is how microchip refers to data memory.

MOV Wns, f Instruction Execution



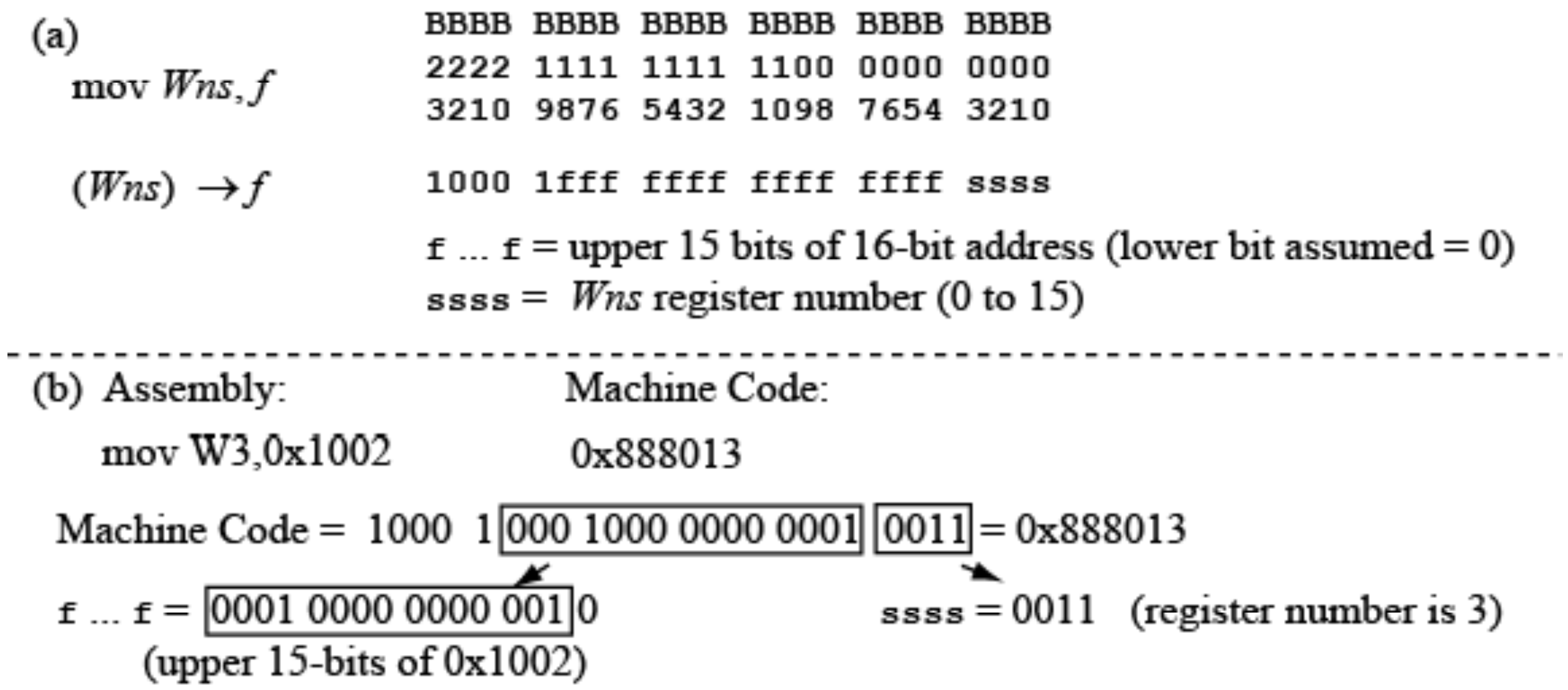
Location	Contents
0x1000	0x8B1A
0x1002	0x25AC
0x1004	0xFB90
0x1006	0x9ED7

Before

Location	Contents
0x1000	0x8B1A
0x1002	0x64DE
0x1004	0xFB90
0x1006	0x9ED7

After

MOV Wns, f Instruction Format



(a)

mov *Wns*, *f*

```
BBBB BBBB BBBB BBBB BBBB BBBB
2222 1111 1111 1100 0000 0000
3210 9876 5432 1098 7654 3210
```

$(Wns) \rightarrow f$

```
1000 1fff ffff ffff ffff ssss
```

f ... f = upper 15 bits of 16-bit address (lower bit assumed = 0)

ssss = *Wns* register number (0 to 15)

(b) Assembly: mov W3,0x1002

Machine Code: 0x888013

Machine Code = $1000\ 1\boxed{000\ 1000\ 0000\ 0001}\ \boxed{0011}$ = 0x888013

f ... f = $\boxed{0001\ 0000\ 0000\ 001}0$
(upper 15-bits of 0x1002)

ssss = 0011 (register number is 3)

MOV f, Wnd Instruction

- **"Copy contents of data memory location *f* to register Wnd". General form:**

MOV *f*, Wnd (f) → Wnd

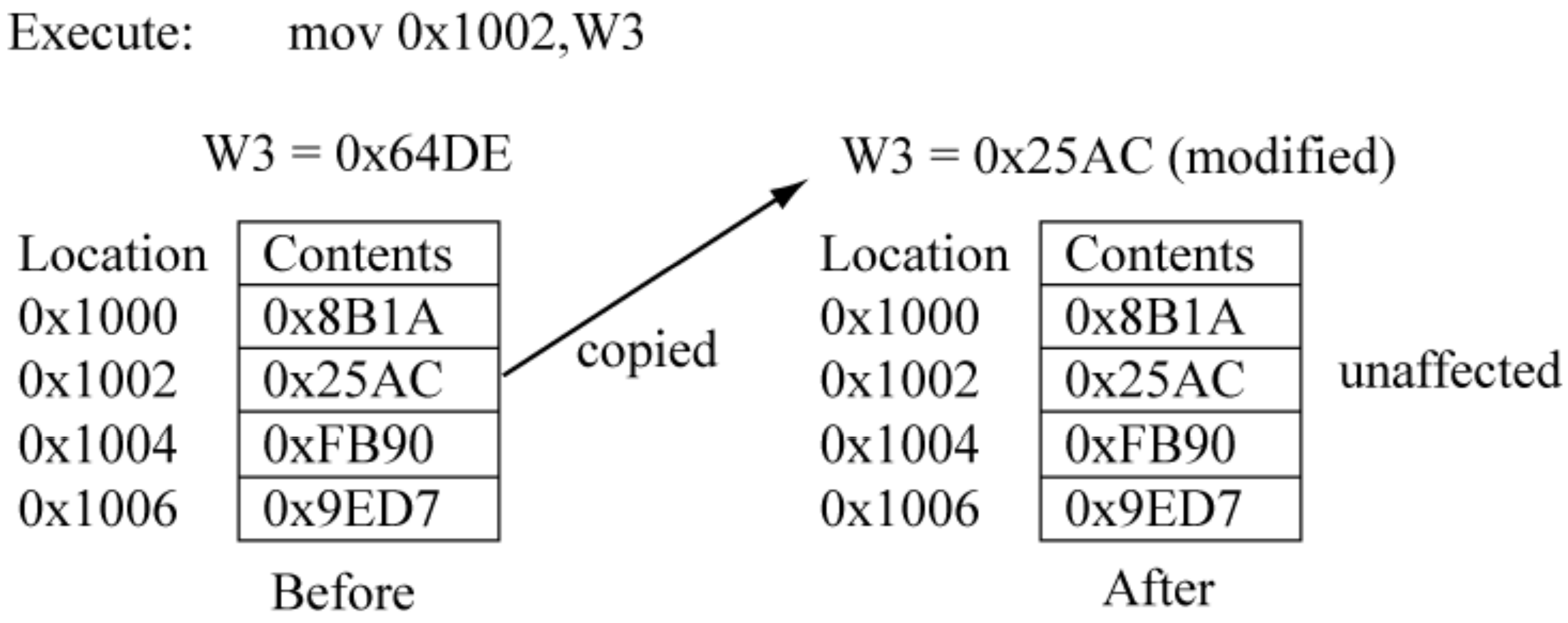
f is a memory location in data memory, Wnd is one of the 16 working registers W0 through W15 ('d' indicates Wnd is an operand destination register for the operation).

MOV 0x1000, W3 (0x1000) → W3

Contents of data memory location 0x1000 copied to W3.

() is read as "Contents of".
This instruction form only supports a word operation.

MOV f, Wnd Instruction Execution



Before:

Location	Contents
0x1000	0x8B1A
0x1002	0x25AC
0x1004	0xFB90
0x1006	0x9ED7

After:

Location	Contents
0x1000	0x8B1A
0x1002	0x25AC
0x1004	0xFB90
0x1006	0x9ED7

Instruction Format

- **Instructions format (machine code) are presented informally**
- **More details in Programmers reference manual**

MOV.{B} WREG f, Wnd Instruction

"Copy content of WREG (default working register) to data memory location *f*". General form:

MOV{.B} WREG,	*f* (WREG) → f

This instruction provides upward compatibility with earlier PIC µC. WREG is register W0, and f is a location within the first 8192 bytes of data memory (*near* data memory)

MOV WREG, 0x1000	(W0) → 0x1000

Contents of register W0 copied to data memory location 0x1000.

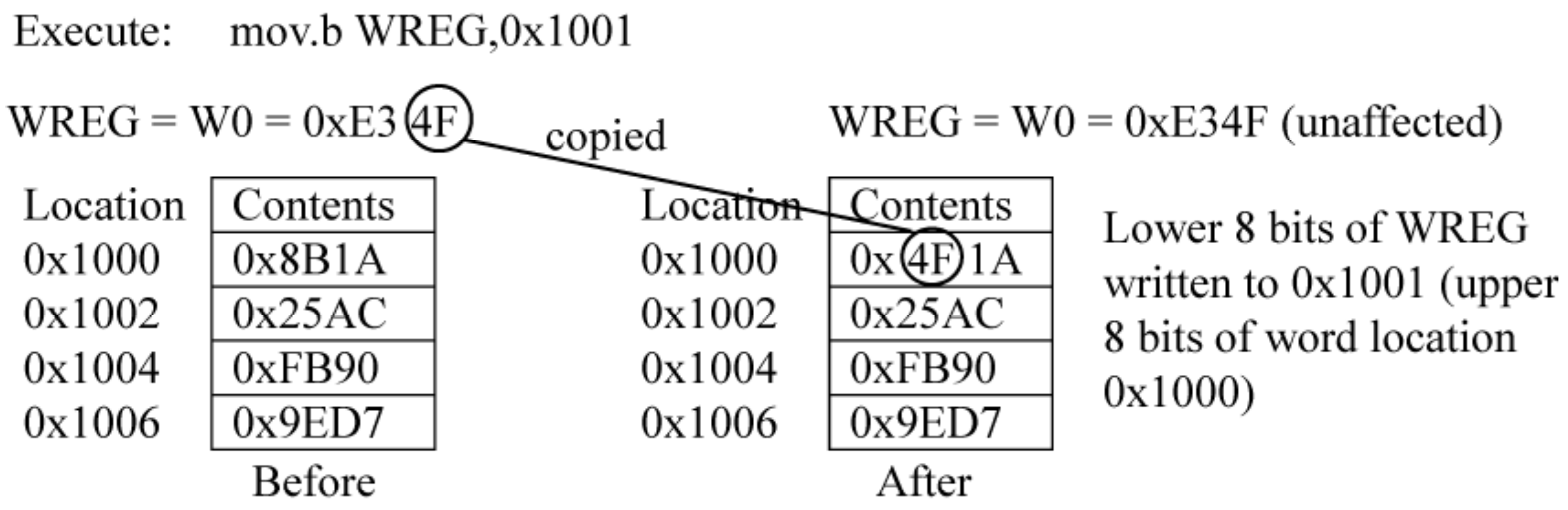
Can be used for either WORD or BYTE operations:

MOV WREG, 0x1000	word operation
MOV.B WREG, 0x1001	lower 8-bits of W0 copied to 0x1001

Word copy must be to even (word-aligned) location.

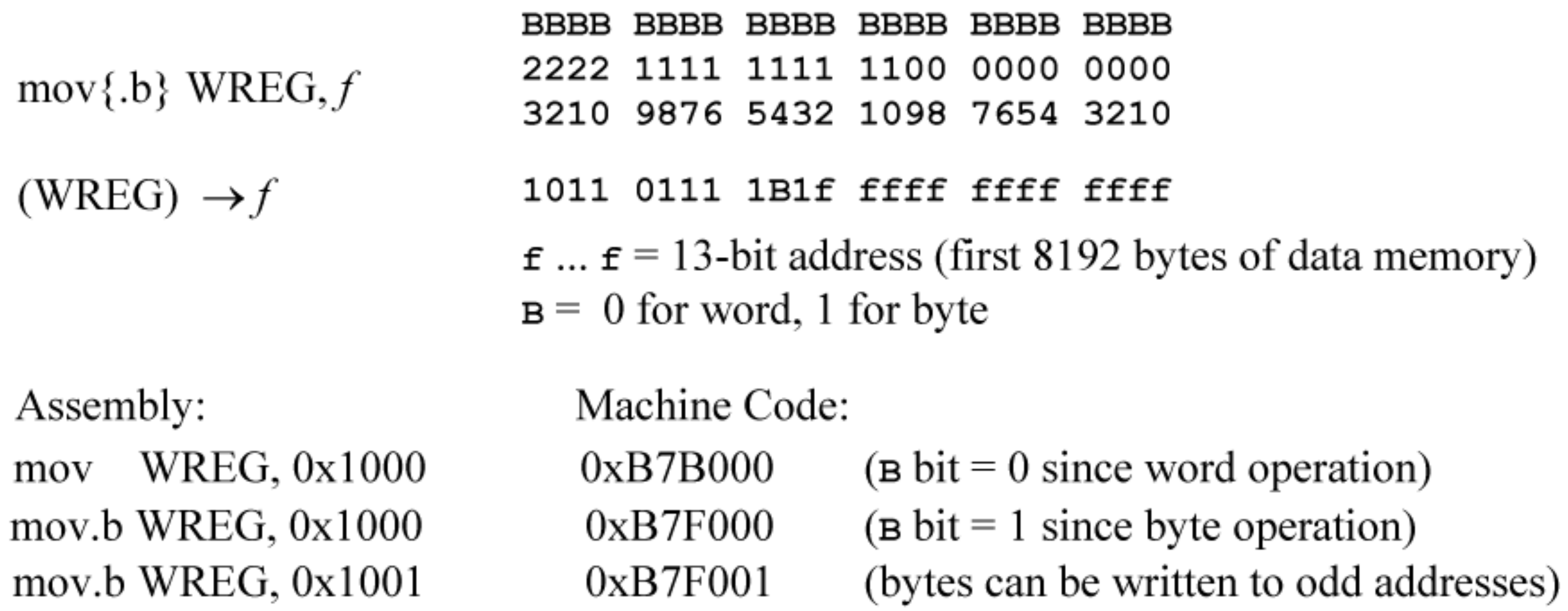
Note: The previously covered *MOV Wns, f* instruction cannot be used for byte operations!

MOV.{B} WREG f, Wnd Instruction Execution



A byte copy operation is shown.

MOV.{B} WREG f, Wnd Instruction Format



mov{.b} WREG, *f*

(WREG) $\rightarrow f$

```
BBBB BBBB BBBB BBBB BBBB BBBB
2222 1111 1111 1100 0000 0000
3210 9876 5432 1098 7654 3210

1011 0111 1B1f ffff ffff ffff
```

`f` ... `f` = 13-bit address (first 8192 bytes of data memory)
`B` = 0 for word, 1 for byte

Assembly:	Machine Code:	
mov WREG, 0x1000	0xB7B000	(`B` bit = 0 since word operation)
mov.b WREG, 0x1000	0xB7F000	(`B` bit = 1 since byte operation)
mov.b WREG, 0x1001	0xB7F001	(bytes can be written to odd addresses)

MOV {.B} f {,WREG} Instruction

- **“Copy contents of data memory location *f* to WREG (default working register) . General form:**

MOV{.B} *f*, WREG	(*f*)→ WREG
MOV{.B} *f*	(*f*)→*f*

This instruction provides upward compatibility with earlier PIC µC. WREG is register W0, and f is a location within the first 8192 bytes of data memory (*near* data memory)

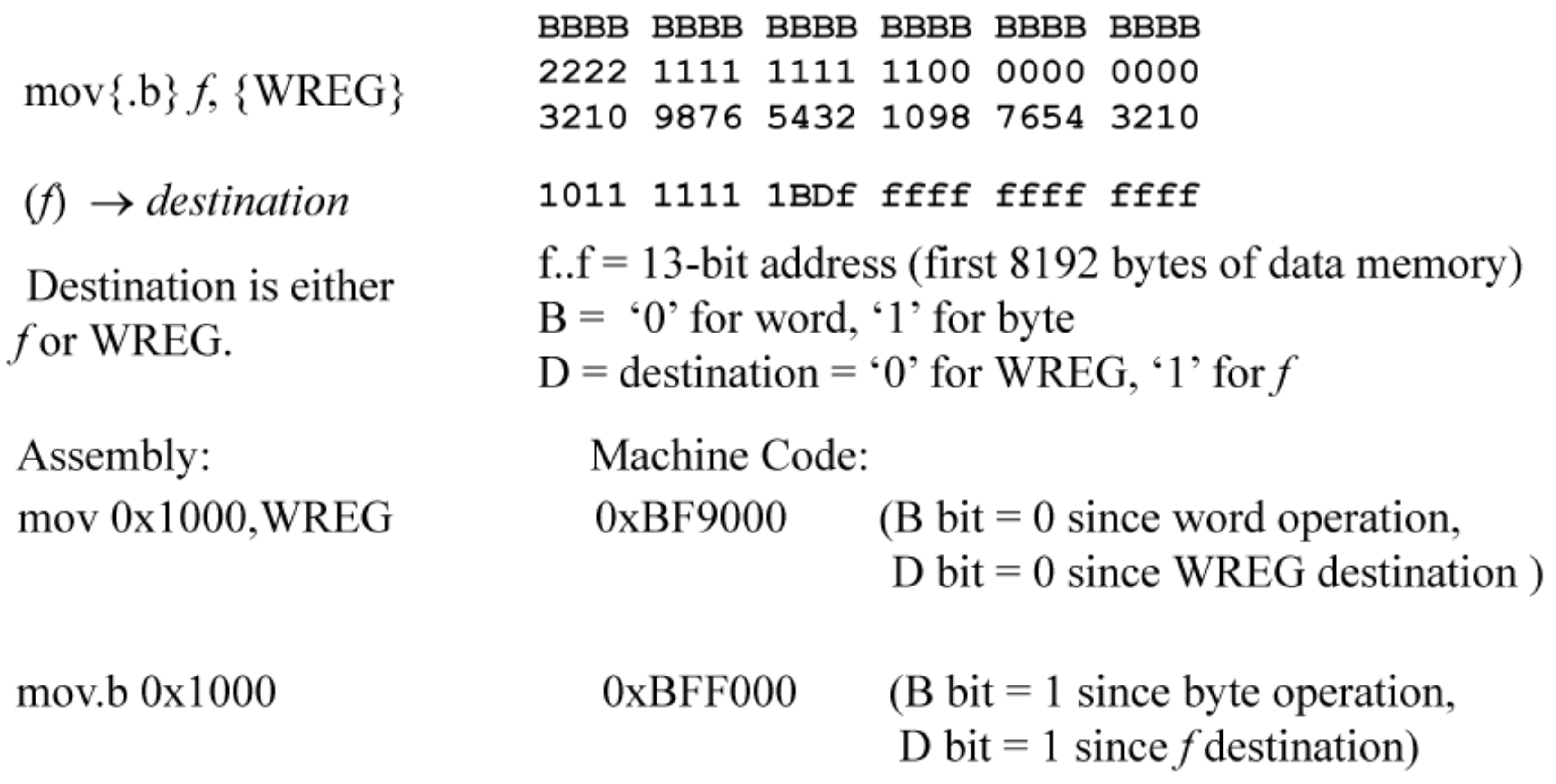
Can be used for either WORD or BYTE operations:

MOV 0x1000, WREG	word operation
MOV.B 0x1001, WREG	only lower 8-bits of W0 are affected.
MOV 0x1000	Copies contents of 0x1000 back to itself, will see usefulness of this later

Word copy must be from even (word-aligned) data memory location.

Note: The *MOV f,Wnd* instruction cannot be used for byte operations!

MOV {.B} f {,WREG} Instruction Format



mov{.b} *f*, {WREG}

```
BBBB BBBB BBBB BBBB BBBB BBBB
2222 1111 1111 1100 0000 0000
3210 9876 5432 1098 7654 3210
```

(*f*) → *destination*

```
1011 1111 1BDf ffff ffff ffff
```

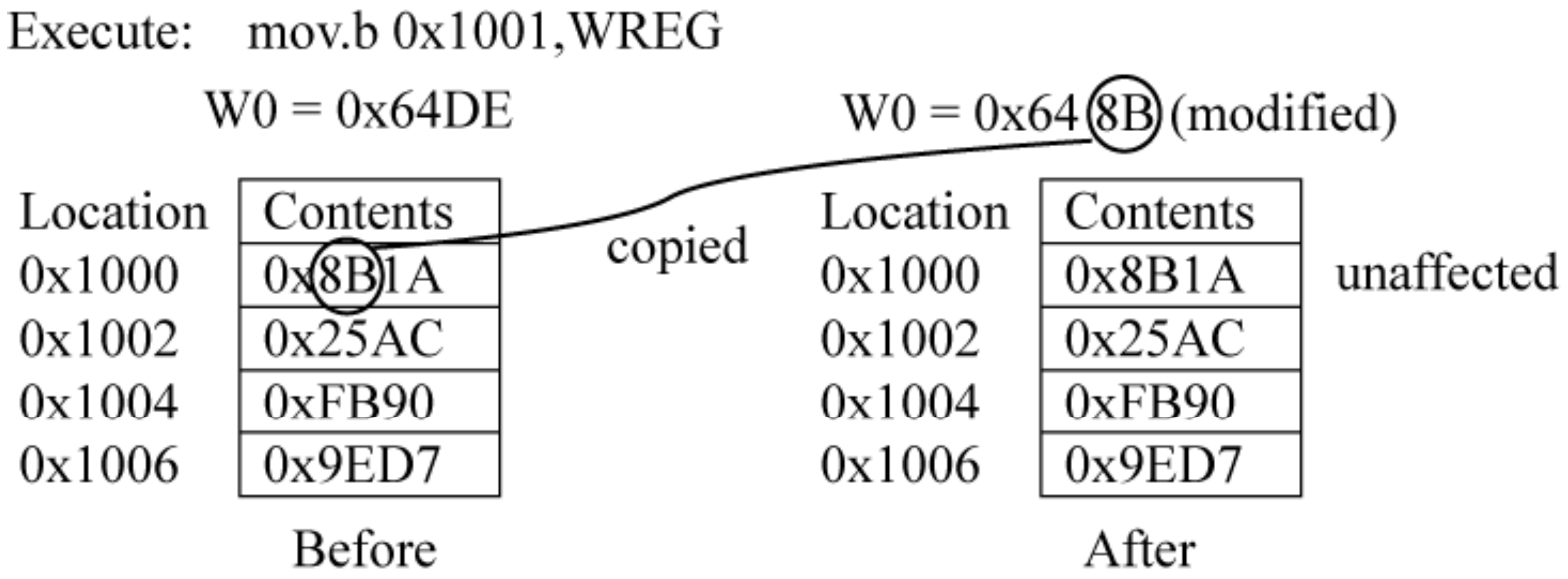
Destination is either *f* or WREG.

f..f = 13-bit address (first 8192 bytes of data memory)
B = '0' for word, '1' for byte
D = destination = '0' for WREG, '1' for *f*

Assembly:	Machine Code:	
mov 0x1000,WREG	0xBF9000	(B bit = 0 since word operation, D bit = 0 since WREG destination)
mov.b 0x1000	0xBFF000	(B bit = 1 since byte operation, D bit = 1 since *f* destination)

From: Reese/Bruce/Jones, "Microcontrollers: From Assembly to C with the PIC24 Family".

MOV {.B} f {,WREG} Instruction Execution



Execute: mov.b 0x1001,WREG

W0 = 0x64DE

Location	Contents
0x1000	0x8B1A
0x1002	0x25AC
0x1004	0xFB90
0x1006	0x9ED7

Before

W0 = 0x648B (modified)

Location	Contents
0x1000	0x8B1A
0x1002	0x25AC
0x1004	0xFB90
0x1006	0x9ED7

After

Move a literal into a working Register

- **Moves a *literal* into a working register. The '#' indicates the numeric value is a literal, and NOT a memory address.**

General form:

MOV #lit16, Wnd	lit16 → Wnd (word operation)
MOV.B #lit8, Wnd	lit8 → Wnd.lsb (byte operation)

The source operand in these examples use the *immediate* addressing mode.

Examples:

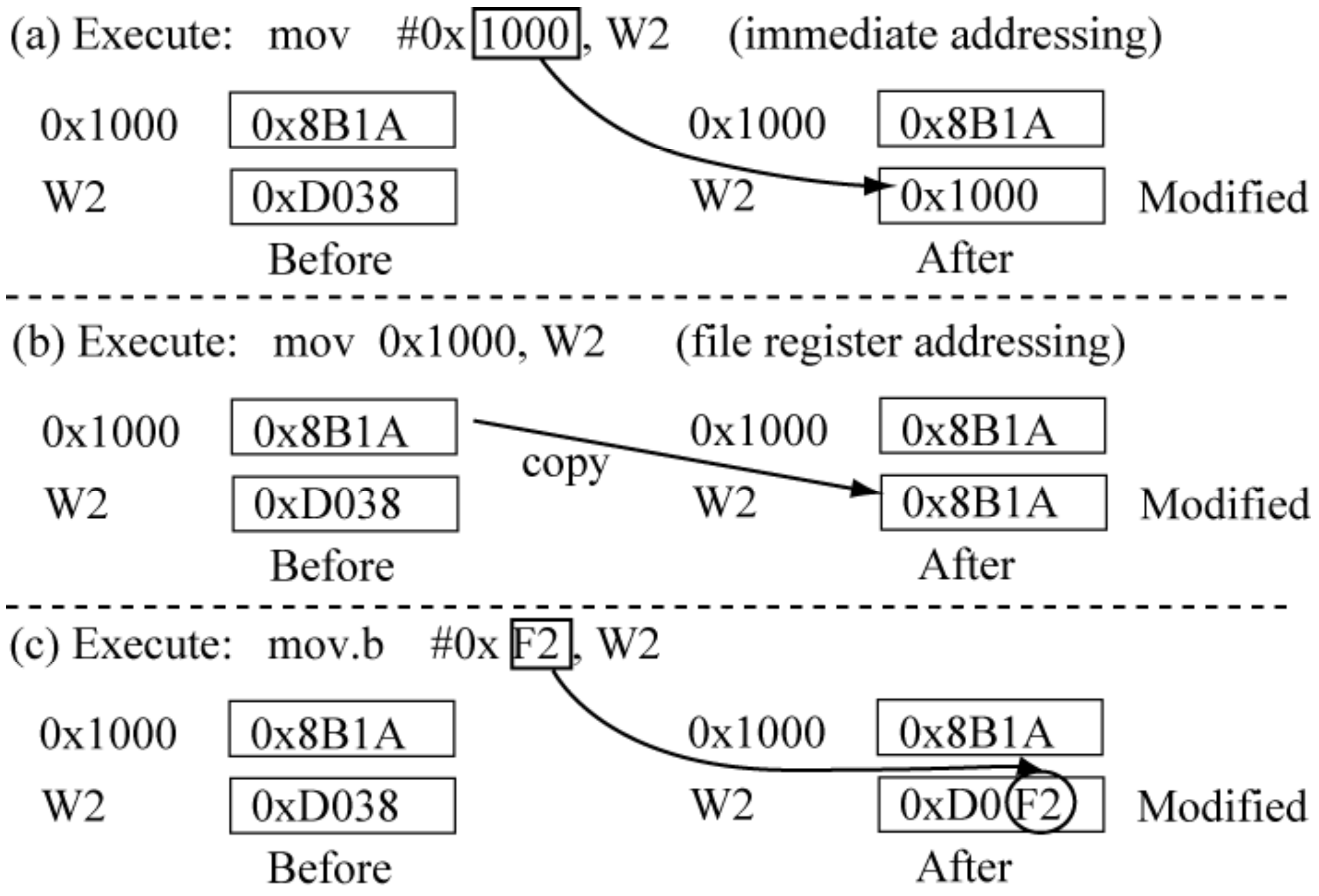
MOV #0x1000, W2	0x1000 → W2
MOV.B #0xAB, W3	0xAB → W3.lsb

More on Literals

Observe that the following two instructions are very different!

MOV #0x1000, W2	after execution, W2=0x1000
MOV 0x1000,W2	after execution, W2 = (0x1000), the contents of memory location 0x1000

MOV Literal Execution



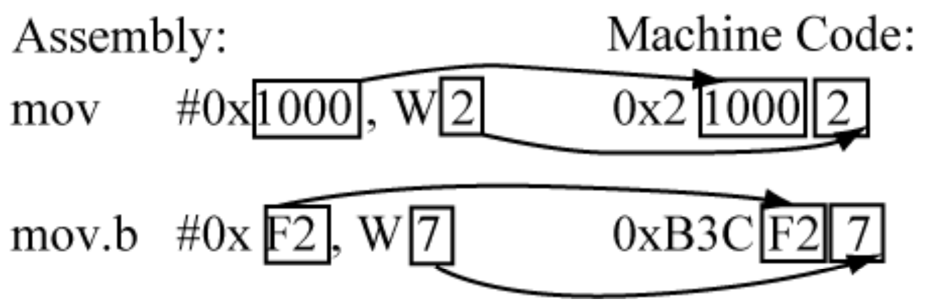
From: Reese/Bruce/Jones, “Microcontrollers: From Assembly to C with the PIC24 Family”.

MOV Literal Instruction Formats

		BBBB 2222 3210	BBBB 1111 9876	BBBB 1111 5432	BBBB 1100 1098	BBBB 0000 7654	BBBB 0000 3210
mov *#lit16*, *Wn*	*#lit16* $\rightarrow$ *Wn*	0010	kkkk	kkkk	kkkk	kkkk	dddd
mov.b *#lit8*, *Wn*	*#lit8* $\rightarrow$ *Wn*	1011	0011	1100	kkkk	kkkk	dddd

#lit16: 16-bit literal
#lit8: 8-bit literal

k … k = literal
dddd = *Wn* register number (0 to 15)



Observe that the literal is encoded directly in the instruction machine code.