Lecture 3

9/12/16

Memory Organization

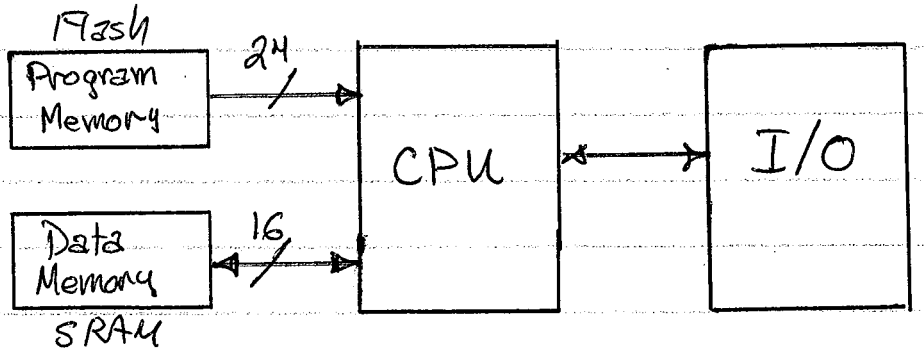
For the PIC24FJ64GA002 there are two memories

- Bytes of Program Memory 64k (Flash)
- Bytes of Data Memory 8k (SRAM)

Note that $64k = 64 \times 2^{10} = 65,536$ bytes
and $8k = 8 \times 2^{10} = 8,192$ bytes

Since there is separate memories for program and data this is a Harvard architecture

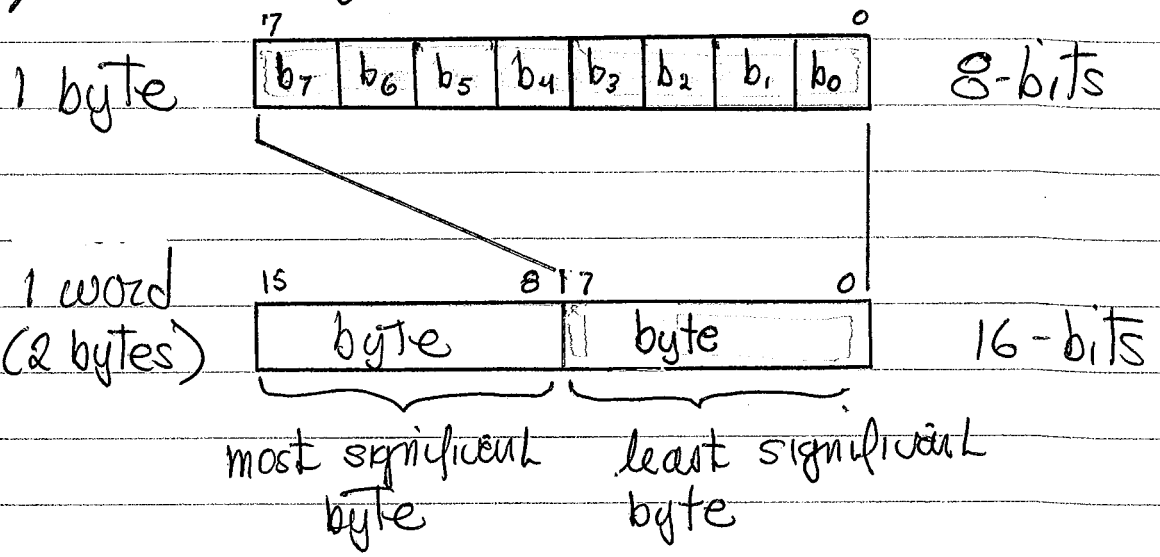
Thus



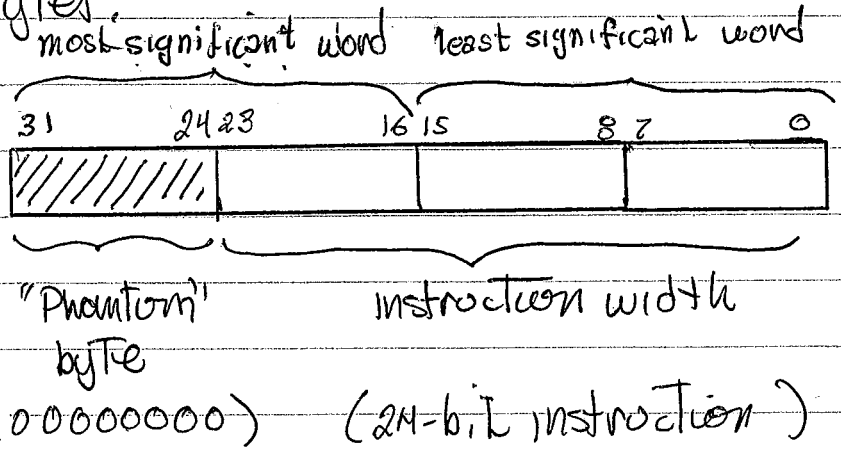
Note that each instruction passed to the CPU from program memory is 24 bits (3 bytes) while data passed from the data memory is 16 bits (2 bytes)

Next, discuss both memories

Memory is almost universally described in terms of 8-bit bytes. For the PIC24 (and dsPIC) devices a word is a 16-bit word consisting of 2, 8-bit bytes



Instructions are 24-bits in size and require 3-bytes.



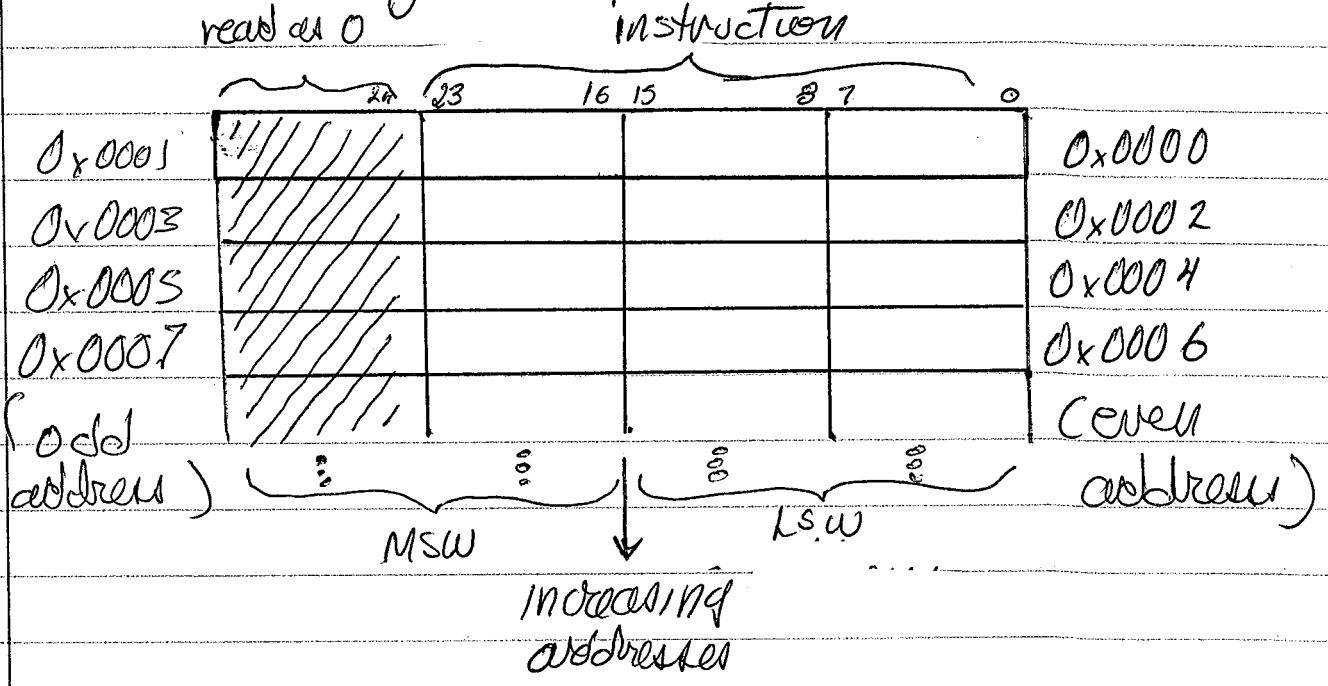
However we view the upper byte as the least significant byte of a 32-bit or two word instruction

except for

Program memory then viewed as consisting of 2 word locations.

The <u>least significant word</u> (LSW) is always at an <u>even address</u>

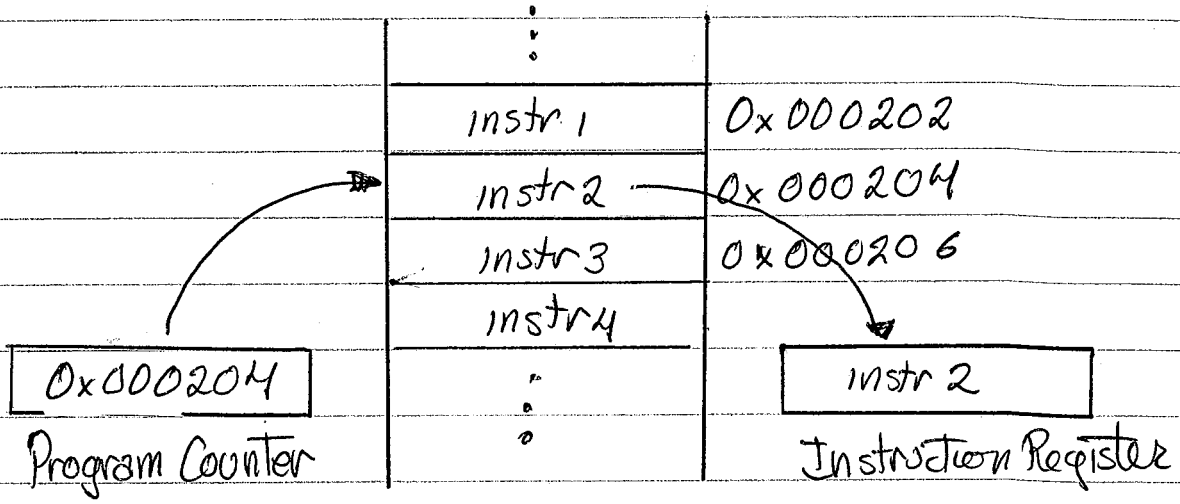
The <u>most significant word</u> (MSW) is always at an <u>odd address</u>



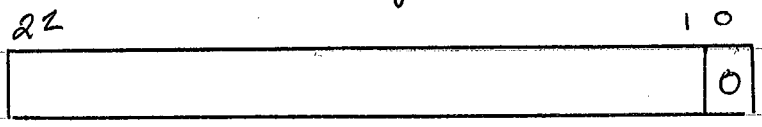
Memory address are <u>word aligned</u>, that is every address in memory points to a 16-bit word

How many instructions can PIC24FJ64GA002 have?

One of the important registers in the processor is the program counter (PC). Locations in Program Memory are pointed to by the value in the program counter. The 24-bit value at that memory location is copied into the instruction register where it is decoded.



The program counter has 23 bits, since instructions will lie at even addresses the rightmost bit in this register is 0



The value in the PC is incremented by 2 each instruction cycle except for jumps and branches which may replace the entire value.

The PIC24 family of devices can address a total of

$$2^{23}/2 = 2^{22} = 2^2 \cdot 2^{20} = 4M$$

instructions. However no devices actually implement that amount of flash memory. Moreover, some memory regions are reserved for other purposed.

For the PIC24FJ64GAxxx devices access is limited to the lower half of the memory address space. Moreover in this space locations are reserved for the reset vector, interrupt vectors, and flash configuration words.

This is shown in the figure.
(figure 4-1 in the datasheet)

NOT THAT ONLY A SMALL PART
OF THE MEMORY ADDRESS SPACE
CAN STORE YOUR PROGRAM Code

Program memory is flash E^2PROM and nonvotile

4.0 MEMORY ORGANIZATION

As Harvard architecture devices, PIC24F microcontrollers feature separate program and data memory spaces and buses. This architecture also allows the direct access of program memory from the data space during code execution.

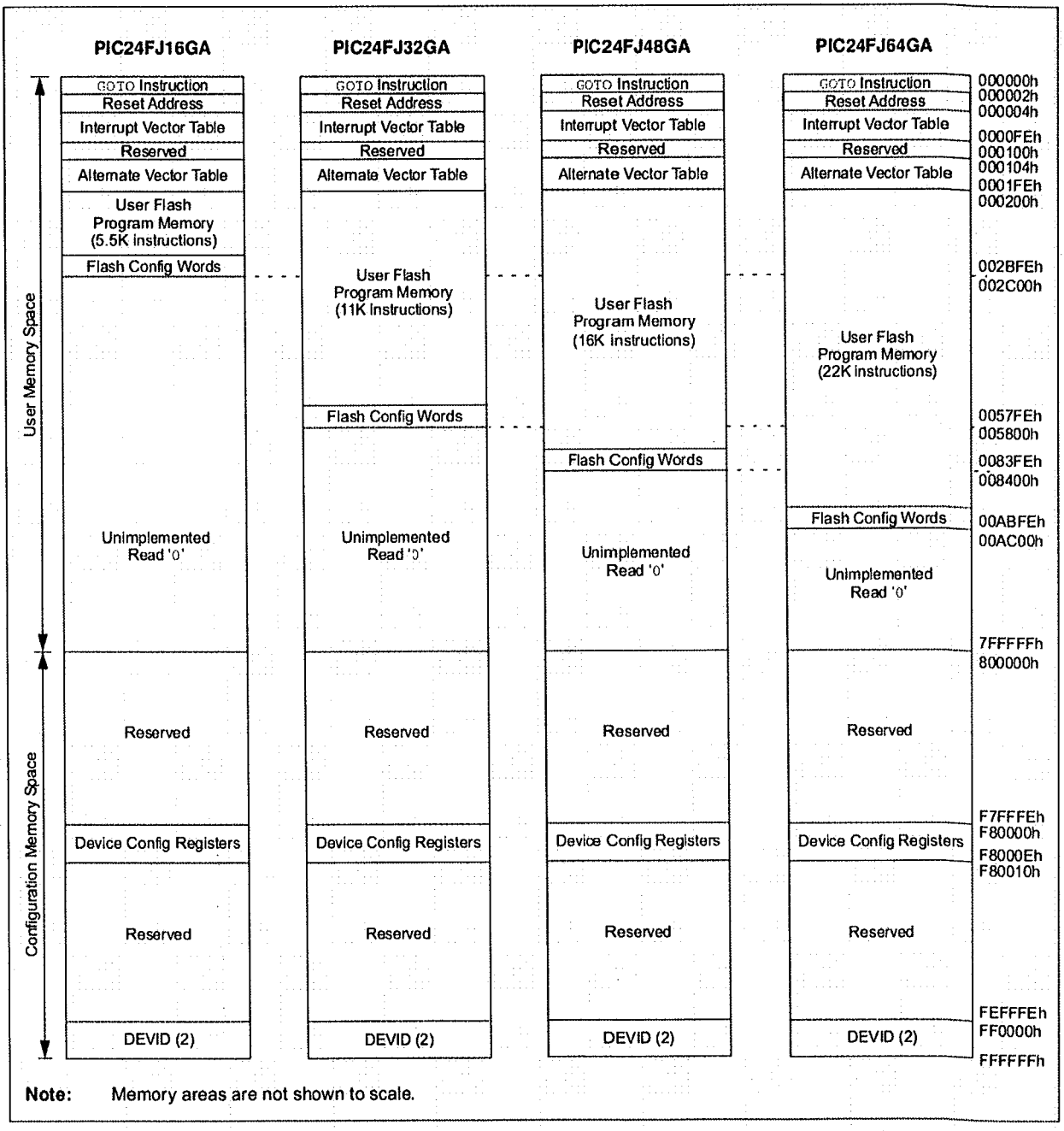
4.1 Program Address Space

The program address memory space of the PIC24FJ64GA004 family devices is 4M instructions. The space is addressable by a 24-bit value derived from either the 23-bit Program Counter (PC) during program execution, or from table operation or data space remapping, as described in **Section 4.3 "Interfacing Program and Data Memory Spaces"**.

User access to the program memory space is restricted to the lower half of the address range (000000h to 7FFFFFh). The exception is the use of TBLRD/TBLWT operations which use TBLPAG<7> to permit access to the Configuration bits and Device ID sections of the configuration memory space.

Memory maps for the PIC24FJ64GA004 family of devices are shown in Figure 4-1.

FIGURE 4-1: PROGRAM SPACE MEMORY MAP FOR PIC24FJ64GA004 FAMILY DEVICES



Note: Memory areas are not shown to scale.

15a

4.1.1 PROGRAM MEMORY ORGANIZATION

The program memory space is organized in word-addressable blocks. Although it is treated as 24 bits wide, it is more appropriate to think of each address of the program memory as a lower and upper word, with the upper byte of the upper word being unimplemented. The lower word always has an even address, while the upper word has an odd address (Figure 4-2).

Program memory addresses are always word-aligned on the lower word, and addresses are incremented or decremented by two during code execution. This arrangement also provides compatibility with data memory space addressing and makes it possible to access data in the program memory space.

4.1.2 HARD MEMORY VECTORS

All PIC24F devices reserve the addresses between 00000h and 000200h for hard-coded program execution vectors. A hardware Reset vector is provided to redirect code execution from the default value of the PC on device Reset to the actual start of code. A GOTO instruction is programmed by the user at 000000h with the actual address for the start of code at 000002h.

PIC24F devices also have two Interrupt Vector Tables (IVT), located from 000004h to 0000FFh and 000100h to 0001FFh. These vector tables allow each of the many device interrupt sources to be handled by separate ISRs. A more detailed discussion of the Interrupt Vector Tables is provided in **Section 7.1 "Interrupt Vector Table"**.

4.1.3 FLASH CONFIGURATION WORDS

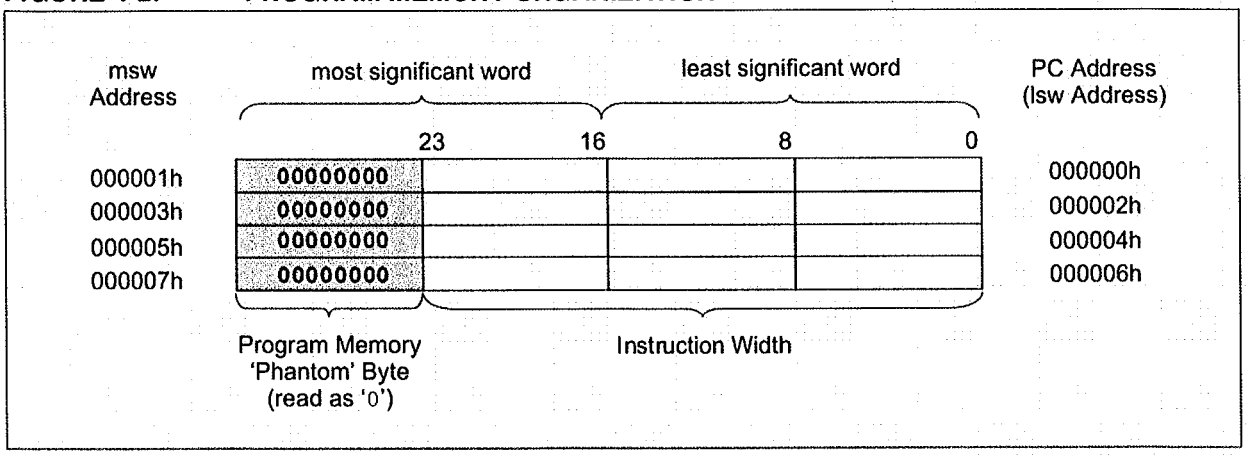
In PIC24FJ64GA004 family devices, the top two words of on-chip program memory are reserved for configuration information. On device Reset, the configuration information is copied into the appropriate Configuration registers. The addresses of the Flash Configuration Word for devices in the PIC24FJ64GA004 family are shown in Table 4-1. Their location in the memory map is shown with the other memory vectors in Figure 4-1.

The Configuration Words in program memory are a compact format. The actual Configuration bits are mapped in several different registers in the configuration memory space. Their order in the Flash Configuration Words does not reflect a corresponding arrangement in the configuration space. Additional details on the device Configuration Words are provided in **Section 24.1 "Configuration Bits"**.

TABLE 4-1: FLASH CONFIGURATION WORDS FOR PIC24FJ64GA004 FAMILY DEVICES

Device	Program Memory (K words)	Configuration Word Addresses
PIC24FJ16GA	5.5	002BFCh: 002BFEh
PIC24FJ32GA	11	0057FCh: 0057FEh
PIC24FJ48GA	16	0083FCh: 0083FEh
PIC24FJ64GA	22	00ABFCh: 00ABFEh

FIGURE 4-2: PROGRAM MEMORY ORGANIZATION



156

- Data RAM

The 8k bytes of data RAM occupy 0x0800 to 0x27FF

- Program Space Visibility Area

The PSV is a "window" which allows a region in the program memory to be accessed in the data memory space. This is entire upper half of the data address space 0x8000 to 0xFFFF

Note that the Near Data Space is the region which only requires 13-bits to address, the upper limit is $2^{13}-1 = 0x1FFF$.

Note also that the region from 0x2800 to 0x7FFF, 22,528 bytes, is unimplemented

Figure 4-3 in data sheet illustrates data memory.

Data Memory is implemented with SRAM and is <u>volitale</u> (no power ⇒ no data)

4.2 Data Address Space

The PIC24F core has a separate, 16-bit wide data memory space, addressable as a single linear range. The data space is accessed using two Address Generation Units (AGUs), one each for read and write operations. The data space memory map is shown in Figure 4-3.

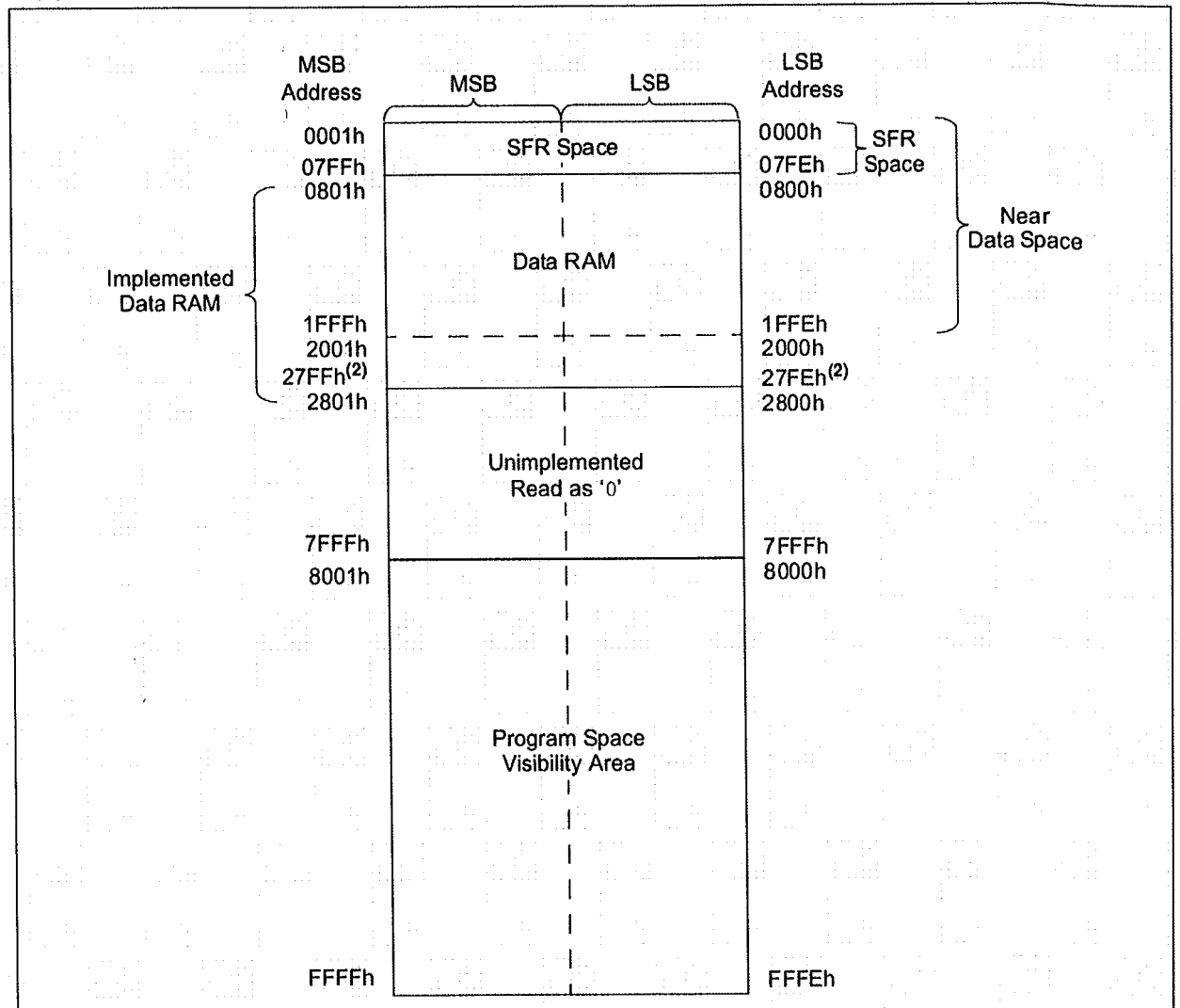
All Effective Addresses (EAs) in the data memory space are 16 bits wide and point to bytes within the data space. This gives a data space address range of 64 Kbytes or 32K words. The lower half of the data memory space (that is, when EA<15> = 0) is used for implemented memory addresses, while the upper half (EA<15> = 1) is reserved for the Program Space Visibility (PSV) area (see **Section 4.3.3 "Reading Data From Program Memory Using Program Space Visibility"**).

PIC24FJ64GA004 family devices implement a total of 8 Kbytes of data memory. Should an EA point to a location outside of this area, an all zero word or byte will be returned.

4.2.1 DATA SPACE WIDTH

The data memory space is organized in byte-addressable, 16-bit wide blocks. Data is aligned in data memory and registers as 16-bit words, but all data space EAs resolve to bytes. The Least Significant Bytes (LSBs) of each word have even addresses, while the Most Significant Bytes (MSBs) have odd addresses.

FIGURE 4-3: DATA SPACE MEMORY MAP FOR PIC24FJ64GA004 FAMILY DEVICES(1)



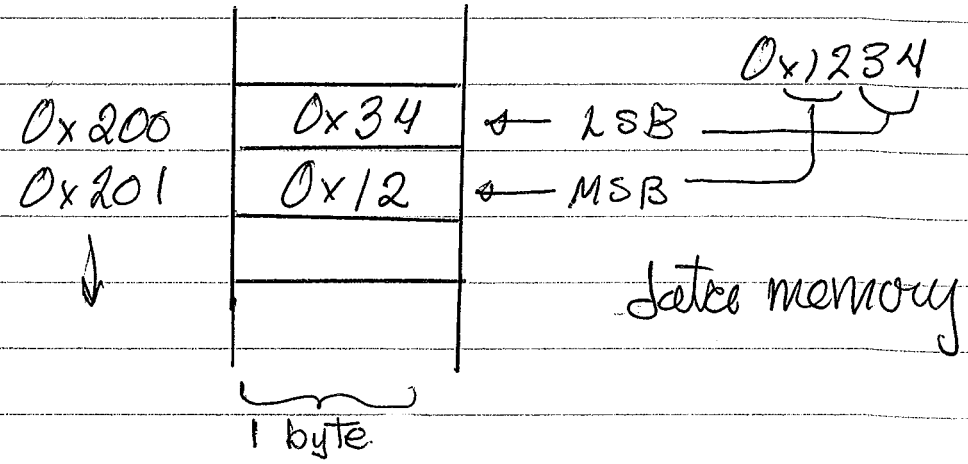
Note 1: Data memory areas are not shown to scale.
2: Upper memory limit for PIC24FJ16GAXXX devices is 17FFh.

17a

Endianess

The PIC24 stores 16-bit or 32-bit values in the order least significant byte to most significant byte in increasing address locations.

So if we have the value 0x1234 corresponding to the unsigned integer 4660 it would be stored as



This is reffered to as little-endian byte ordering. An alternative is big-endian which starts with the MSB and reverses the byte ording.

There is no inherent advantage to either ording and is the choice of the architech

Instructions

After the 24-bit instruction has been fetched from program memory and placed in the instruction register it is decoded

ALL THE INFORMATION NEEDED TO EXECUTE THE INSTRUCTION IS CONTAINED IN THE INSTRUCTION

The instructions in machine language are in binary code, a collection of 1s and 0s, although these are commonly written using hex notation for readability

IT IS DIFFICULT, TEDIOUS, AND ERROR-PRONE, TO PROGRAM DIRECTLY USING MACHINE CODE.

Assembly language replaces the tedious binary coding of directly using machine code. It does this by replacing the bit patterns of machine code with more understandable mnemonics.

Assembly instructions typically have a one-to-one relationship with machine instructions. Assembly language, like machine language, is machine specific. To understand what a machine instruction, based on the binary or hex representation does, requires referencing documentation for the corresponding processor. However, the mnemonic representation used with assembly instructions usually "hints" at the operation

Data Transfer

One of the most common forms of operation is a microcontroller is moving data from on location to another.

Consider a simple example of moving the contents of register W3 to W5.

We assume both registers contain data.

Microchip, 16-bit MCU and DSC Programmer's Reference Manual (ds70157f, 2011)

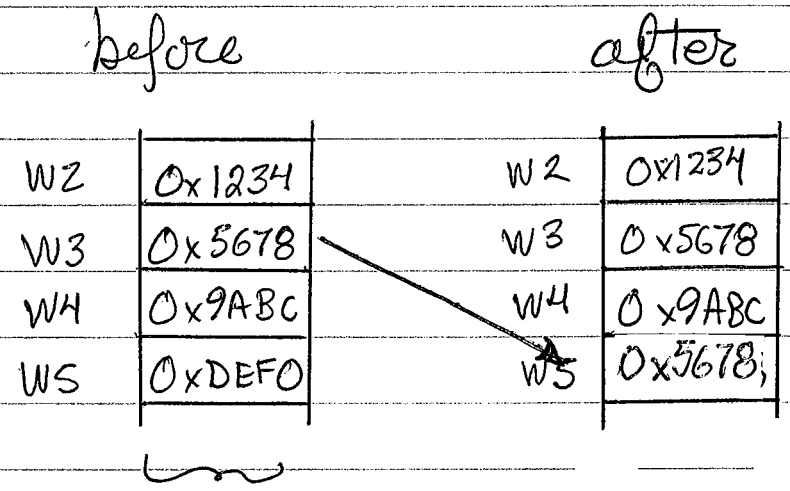
Example: Copy the contents of register W3 to W5. The assembly code instruction is

source → ← destination

```
mov   W3, W5
```

mnemonic ↑ operands

and this will overwrite the contents of W5 with those of W3.



Note that this operation moved the entire 16-bit word in W3 to W5. This is the default, more specifically this instruction can be written

```
mov.w  W3, W5
```

where the .w appended to the mov indicates this instruction operates on a word.

The move instruction also has a byte mode. Consider the same initial values in W3 and W5

```
mov.b  W3, W5
```

will move the lsb from W3 to W5

W2	0x1234
W3	0x5678
W4	0x9ABC
W5	0xDEF0

W2	0x1234
W3	0x5678
W4	0x9ABC
W5	0xDE78

The machine code corresponding to the first example, mov W3, W5 is the following

0111 1000 0000 0010 1000 0011

or in hex

0x780283

It's not at all obvious what this does with reference to the Programmers Reference Manual (ds70157f)

From the entry for mov the encoding is

01111 → opcode

wwww → offset register Wb

B → select "0" for word, "1" for byte

hhh → destination Address mode'

dddd → destination register

ggg → source Address mode'

ssss → soure register

For our instruction: ① ② ③ ④ ⑤ ⑥ ⑦

0111 1000 0000 0010 1000 0011

opcode wwww B direct Wb direct Ws

① 01111 → MOV instruction
② 0000 → (register direct), no offset
③ 0 → move word
④ 000 → register direct
⑤ 0101 → W5 is destination register
⑥ 000 → register direct
⑦ 0011 → W3 is source register

'addressing mode encodings are on p.95, table 5-2 of ds70157A

Example 2:

```
MOV  W11, [W1-0x400]   ; store W11 to [W1-0x400]
                       ; (Word mode)
```

	Before Instruction	After Instruction
W1	1000	1000
W11	8813	8813
Data 0C00	FFEA	8813
SR	0000	0000

MOV

Move Ws to Wd

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	X	X	X	X	X	X

Syntax: {label:} MOV{.B} Ws, Wd
[Ws], [Wd]
[Ws++], [Wd++]
[Ws--], [Wd--]
[--Ws], [--Wd]
[++Ws], [++Wd]
[Ws + Wb], [Wd + Wb]

Operands: Ws ∈ [W0 ... W15]
Wb ∈ [W0 ... W15]
Wd ∈ [W0 ... W15]

Operation: (Ws) →Wd

Status Affected: None

Encoding:

0111	1www	wBhh	hddd	dggg	ssss

Description: Move the contents of the source register into the destination register. Either register direct or indirect addressing may be used for Ws and Wd.

The 'w' bits define the offset register Wb.
The 'B' bit selects byte or word operation ('0' for word, '1' for byte).
The 'h' bits select the destination Address mode.
The 'd' bits select the destination register.
The 'g' bits select the source Address mode.
The 's' bits select the source register.

Note 1: The extension `.B` in the instruction denotes a byte move rather than a word move. You may use a `.W` extension to denote a word move, but it is not required.

2: When Register Offset Addressing mode is used for both the source and destination, the offset must be the same because the 'w' encoding bits are shared by Ws and Wd.

3: The instruction "`PUSH Ws`" translates to `MOV Ws, [W15++]`.

4: The instruction "`POP Wd`" translates to `MOV [--W15], Wd`.

23a

MOV.B

Move 8-bit Literal to Wnd

Implemented in:	PIC24F	PIC24H	PIC24E	dsPIC30F	dsPIC33F	dsPIC33E
	X	X	X	X	X	X

Syntax: {label:} MOV.B #lit8, Wnd

Operands: lit8 ∈ [0 ... 255]
Wnd ∈ [W0 ... W15]

Operation: lit8 →Wnd

Status Affected: None

Encoding:

1011	0011	1100	kkkk	kkkk	dddd

Description: The unsigned 8-bit literal 'k' is loaded into the lower byte of Wnd. The upper byte of Wnd is not changed. Register direct addressing must be used for Wnd.

The 'k' bits specify the value of the literal.
The 'd' bits select the address of the working register.

Note: This instruction operates in Byte mode and the `.B` extension must be provided.

Words: 1

Cycles: 1

Example 1:

```
MOV.B   #0x17, W5     ; load W5 with #0x17 (Byte mode)
```

	Before Instruction	After Instruction
W5	7899	7817
SR	0000	0000

Example 2:

```
MOV.B   #0xFE, W9     ; load W9 with #0xFE (Byte mode)
```

	Before Instruction	After Instruction
W9	AB23	ABFE
SR	0000	0000

23b