EC-310 Microprocessor and Microcontroller Based Design

Chapter - 2

PIC Architecture and Assembly Language Programming

Nazar Abbas Saqib
nazar.abbas@ceme.nust.edu.pk

Outline

1. Introduction to PIC Assembly Programming
2. Assembling and Linking a PIC Program
3. The Program Counter and Program ROM Space in the PIC

What is the procedure of assembling and linking a program to generate a bit file to be burned in a PIC microcontroller?

Introduction to PIC Assembly Programming

- In early days computer programmers coded in machine language (0s and 1s).
  - Advantage: High speed
  - Disadvantage: Cumbersome for humans
- Assembly language provided mnemonics for machine code instructions along with other features that made programming faster and easier.
- Assembly language is a low level language.
- With high level language the programming doesn't need to have any concern with the architecture of the processor or sizes of the registers etc. Ex. BASIC, COBOL, C, C++, FORTRAN, LISP, Pascal, and Prolog

Introduction to PIC Assembly Programming

- A computer can only execute machine language code. Code in machine language is known as **object code**

- **Assembler:** Assembler converts the assembly language code to machine language code

- **Compiler:** Compiler is used to convert high level language code to machine language code

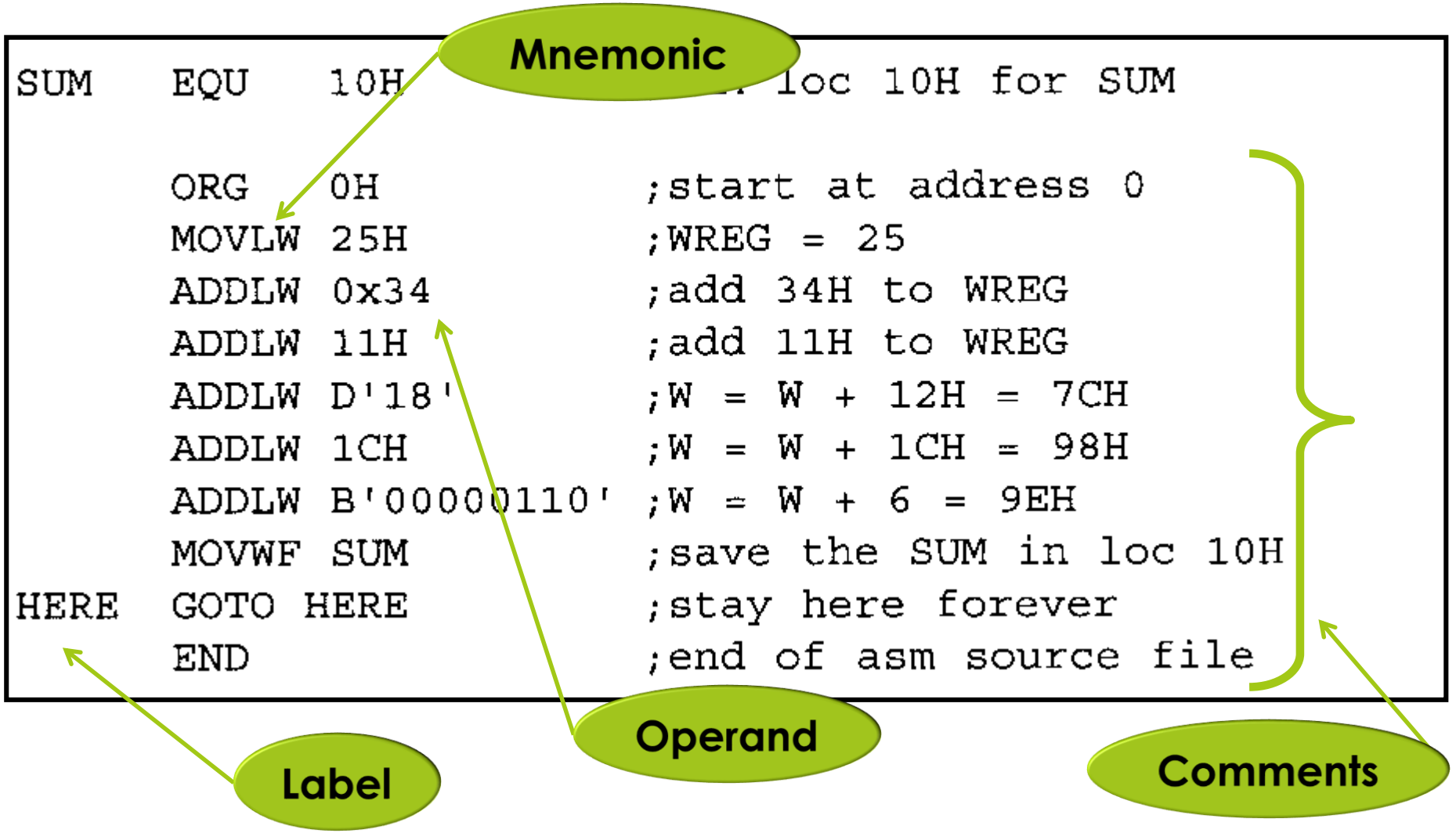
Introduction to PIC Assembly Programming

- An assembly language instruction consists of four fields:

```
[label]    mnemonic   [operands]   [;comment]
```

- Mnemonics: It is the command to the CPU
- Operands: The data which CPU processes
- Label: Allows the program to refer to a line of code by name. It is an optional field.
- Comments: Comments field begins with semi colon. Aids someone else to read your code and understand. It is also an optional field.

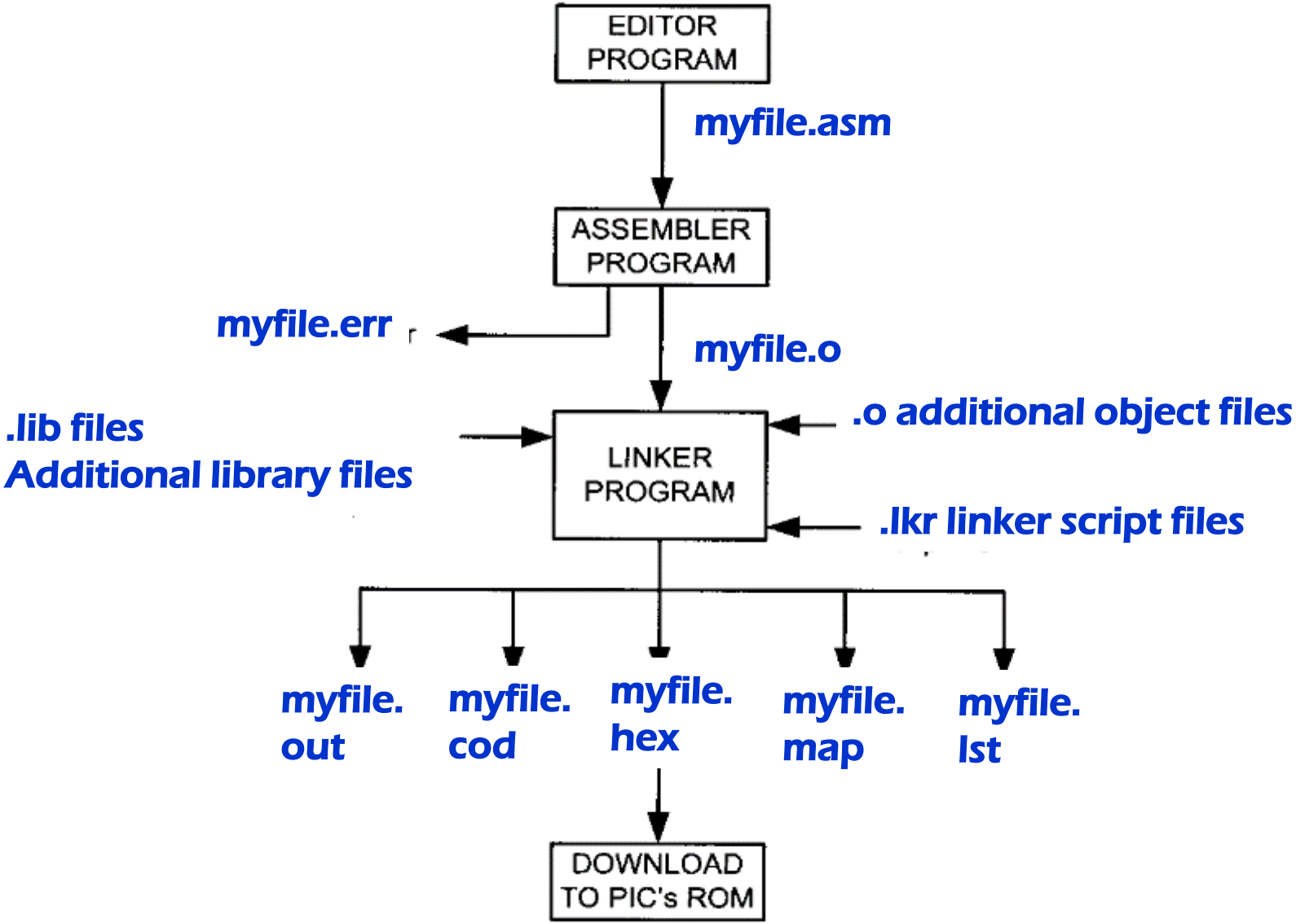
Introduction to PIC Assembly Programming



Assembling and Linking a Program

- Following are the steps in development of a program for PIC microcontroller
  - Writing a code in a text editor and saving the file with extension .asm
  - Assembling: Generating object and error file. This is done by feeding the assembler with the .asm file. The output is a .o and .err file
  - Linking: Linker takes .o files as input and produces a hex file, a list file, a map file, an intermediate object file and a debug file. After successful linking, the hex file is burned into the ROM of microcontroller

Assembling and Linking a Program



.asm file

SUM = 25H + 34H + 12H + 1CH + 06H

```
SUM     EQU     10H             ;RAM loc 10H for SUM

        ORG     0H              ;start at address 0
        MOVLW   25H             ;WREG = 25
        ADDLW   0x34            ;add 34H to WREG
        ADDLW   11H             ;add 11H to WREG
        ADDLW   D'18'           ;W = W + 12H = 7CH
        ADDLW   1CH             ;W = W + 1CH = 98H
        ADDLW   B'00000110'     ;W = W + 6 = 9EH
        MOVWF   SUM             ;save the SUM in loc 10H
HERE    GOTO    HERE            ;stay here forever
        END                     ;end of asm source file
```

The Program Counter and ROM Space

- Code in program ROM:

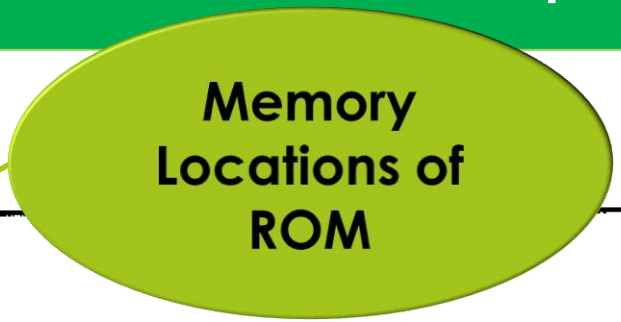
Program starts here. PC is initially pointing here.

```
LOC    OBJECT CODE LINE SOURCE TEXT

                   00001
                   00002 ;PIC Asm Language Program To Add Some Data
                   00003 ;store sum in fileReg location 10H
  00000010         00004 SUM EQU 10H          ;RAM loc 10H for sum
                   00005
000000             00006 ORG 0H               ;start at address 0
000000 0E25        00007 MOVLW 25H            ;WREG = 25
000002 0F34        00008 ADDLW 0x34           ;add 34H to WREG
000004 0F11        00009 ADDLW 11H            ;add 11H to WREG
000006 0F12        00010 ADDLW D'18'          ;W = W + 12H = 7CH
000008 0F1C        00011 ADDLW 1CH            ;W = W + 1CH = 98H
00000A 0F06        00012 ADDLW B'00000110'    ;W = W + 6 = 9EH
00000C 6E10        00013 MOVWF SUM            ;save the sum in loc 10H
00000E EF07 F000   00014 HERE GOTO HERE       ;stay here forever
                   00015 END                  ;end of asm source file
```

Program 2-1: List File

The Program Counter and ROM Space




```
LOC   OBJECT CODE LINE SOURCE TEXT

                  00001
                  00002 ;PIC Asm Language Program To Add Some Data
                  00003 ;store sum in fileReg location 10H
  00000010        00004 SUM EQU 10H          ;RAM loc 10H for sum
                  00005
000000            00006 ORG 0H               ;start at address 0
000000 0E25       00007 MOVLW 25H            ;WREG = 25
000002 0F34       00008 ADDLW 0x34           ;add 34H to WREG
000004 0F11       00009 ADDLW 11H            ;add 11H to WREG
000006 0F12       00010 ADDLW D'18'          ;W = W + 12H = 7CH
000008 0F1C       00011 ADDLW 1CH            ;W = W + 1CH = 98H
00000A 0F06       00012 ADDLW B'00000110'    ;W = W + 6 = 9EH
00000C 6E10       00013 MOVWF SUM            ;save the sum in loc 10H
00000E EF07 F000  00014 HERE GOTO HERE       ;stay here forever
                  00015 END                  ;end of asm source file
```

Program 2-1: List File

List File

Object Code:
1st byte shows code for instruction
2nd byte shows the operand

```
LOC    OBJECT CODE LINE SOURCE TEXT

                   00001
                   00002 ;PIC Asm Language Program To Add Some Data
                   00003 ;store sum in fileReg location 10H
  00000010         00004 SUM EQU 10H          ;RAM loc 10H for sum
                   00005
000000             00006 ORG 0H               ;start at address 0
000000 0E25        00007 MOVLW 25H            ;WREG = 25
000002 0F34        00008 ADDLW 0x34           ;add 34H to WREG
000004 0F11        00009 ADDLW 11H            ;add 11H to WREG
000006 0F12        00010 ADDLW D'18'          ;W = W + 12H = 7CH
000008 0F1C        00011 ADDLW 1CH            ;W = W + 1CH = 98H
00000A 0F06        00012 ADDLW B'00000110'    ;W = W + 6 = 9EH
00000C 6E10        00013 MOVWF SUM            ;save the sum in loc 10H
00000E EF07 F000   00014 HERE GOTO HERE       ;stay here forever
                   00015 END                  ;end of asm source file
```

Program 2-1: List File

err file

```
;PIC ASSEMBLY LANGUAGE PROGRAM TO ADD SOME DATA
;STORE SUM IN fileReg LOCATION 10H


        SUM EQU 10H             ;RAM OC 10H FOR SUM

        ORG 0H                  ;START AT ADDRESS 0
        MOVLW 25H               ;WREG = 25
        ADDLW 0X34              ;ADD 34H TO WREG
        ADDLW 11H               ;ADD 11H TO WREG
        ADDLW D'18'             ;W = W + 12H = 7CH
        ADDLW 1CH               ;W = W + 1CH = 98H
        ADDLW B'00000110'       ;W = W + 6 = 9EH
        MOVWF SUMm              ;SAVE THE SUM IN LOC 10H
HERE_Test   GOTO HERE               ;STAY HERE FOREVER
        END                     ;END OF ASM SOURCE FILE
```

Build | Version Control | Find in Files

Debug build of project `D:\PICPROG\myProj.mcp' start
Language tool versions: MPASMWIN.exe v5.37, mplink
Preprocessor symbol `__DEBUG' is defined.
Sat Oct 18 23:01:47 2014

Make: The target "D:\PICPROG\myadd.o" is out of date
Executing: "C:\Program Files\Microchip\MPASM Suite\
Warning[207] D:\PICPROG\MYADD.ASM 5 : Found lab
Error[113] D:\PICPROG\MYADD.ASM 14 : Symbol not
Error[113] D:\PICPROG\MYADD.ASM 15 : Symbol not
Halting build on first failure as requested.

Debug build of project `D:\PICPROG\myProj.mcp' faile
Language tool versions: MPASMWIN.exe v5.37, mplink
Preprocessor symbol `__DEBUG' is defined.
Sat Oct 18 23:01:48 2014

BUILD FAILED

```
Warning[207] D:\PICPROG\MYADD.ASM 5 : Found label after column 1.
(SUM)
Error[113]   D:\PICPROG\MYADD.ASM 14 : Symbol not previously
defined (SUMm)
Error[113]   D:\PICPROG\MYADD.ASM 15 : Symbol not previously
defined (HERE)
```

The Program Counter and ROM Space

- Code in program ROM:

ROM Address	Machine Language	Assembly Language
00000	0E25	MOVLW 25H
00002	0F34	ADDLW 34
00004	0F11	ADDLW 11H
00006	0F12	ADDLW D'18'
00008	0F1C	ADDLW 1CH
0000A	0F06	ADDLW B'00000110'
0000C	6E10	MOVWF SUM
0000E	EF07 F000	HERE GOTO HERE

Program 2-1: ROM Contents

Address	Code
000000	0E
000001	25
000002	0F
000003	34
000004	0F
000005	11
000006	0F
000007	12
000008	0F
000009	1C
00000A	0F
00000B	06
00000C	6E
00000D	10
00000E	07
00000F	EF
000010	00
000011	F0
000012	

What about memory size and program counter in PIC microcontroller?

The Program Counter and ROM Space

- **Program ROM:**
  - The start address of ROM of all PIC microcontrollers is 00000H
  - The address of last location of ROM depends upon the total size of the ROM which various with model numbers available.
  - Each ROM location is 1-byte wide.

How to find the address range?

Table 2-6: PIC18 On-chip ROM Size and Address Space

	On-Chip Code ROM (Bytes)	Code Address Range (Hex)
PIC18F2220	4K	00000–00FFF
PIC18F2410	16K	00000–03FFF
PIC18F458	32K	00000–07FFF
PIC18F6680	64K	00000–0FFFF
PIC18F8722	128K	00000–1FFFF

The Program Counter and ROM Space

Example 2-11

Find the ROM memory address of each of the following PIC chips:
(a) PIC18F2220 with 4 KB
(b) PIC18F2410 with 16 KB
(c) PIC18F458 with 32 KB

Solution:

(a) With 4K of on-chip ROM memory space, we have 4096 bytes ($4 \times 1024 = 4096$). This maps to address locations of 0000 to 0FFFH. Notice that 0 is always the first location.
(b) With 16K of on-chip ROM memory space, we have 16,384 bytes ($16 \times 1024 = 16,384$),which gives 0000–3FFFH.
(c) With 32K we have 32,768 bytes ($32 \times 1024 = 32,768$). Converting 32,768 to hex, we get 8000H; therefore, the memory space is 0000 to 7FFFH.

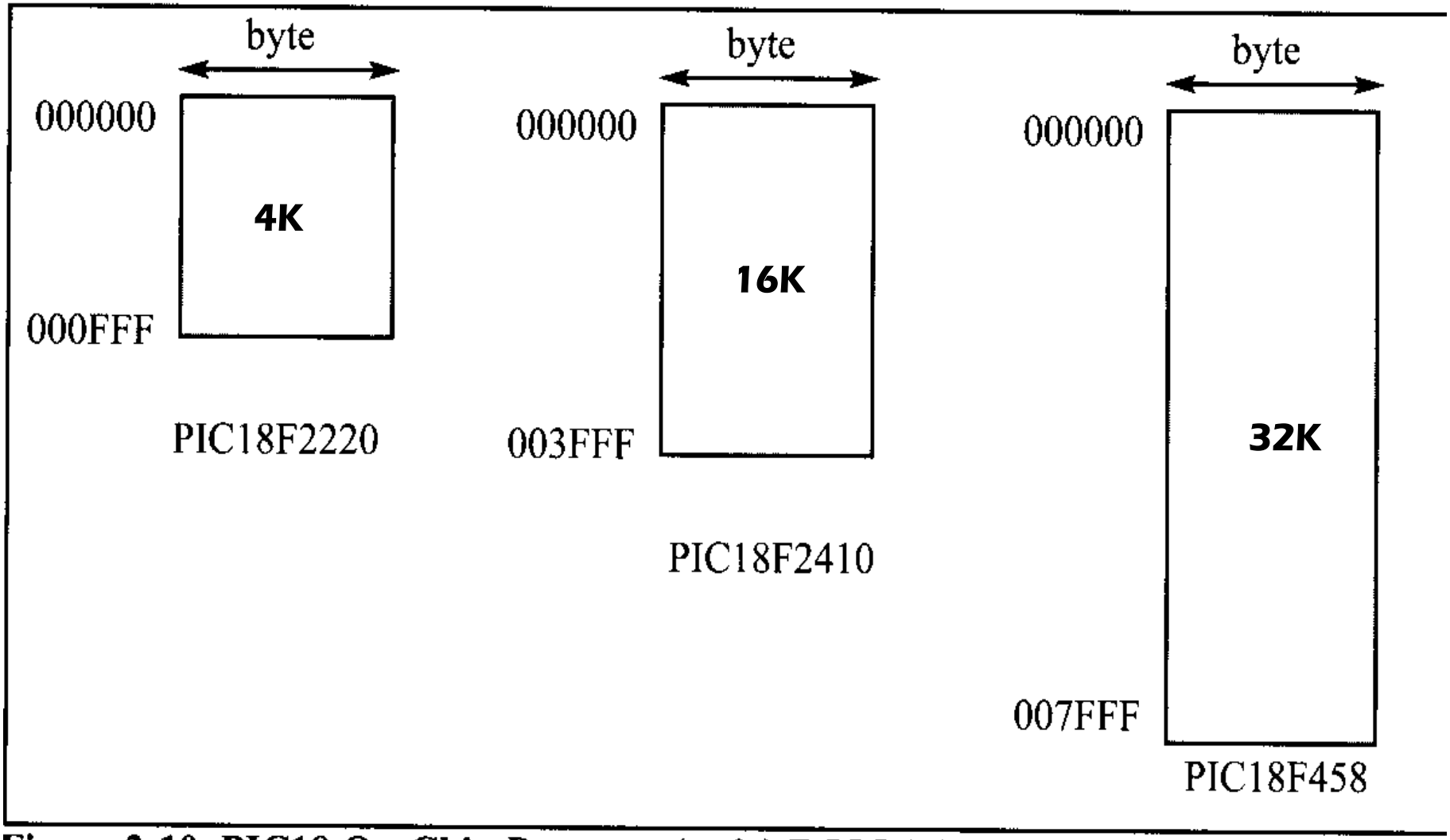


Figure 2-10. PIC18 On-Chip Program (code) ROM Address Range

The Program Counter and ROM Space

- **Program Counter:**

  - Register used by the CPU to point to the next instruction to be executed.
  - As the CPU fetches the opcode from program ROM, the program counter is incremented automatically to point to next instruction.
  - The wider the program counter, the more memory locations a CPU can access.
  - PIC18 has 21-bit PC. It can access memory address from 000000H to 1FFFFFH i.e. 2M of cod.

The Program Counter and ROM Space

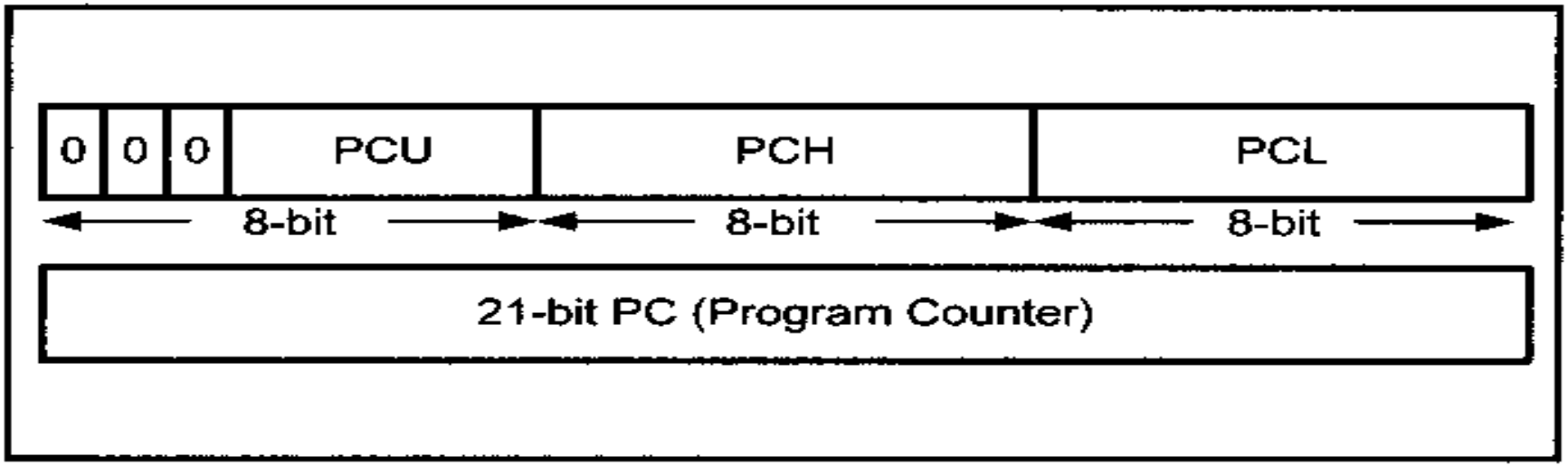


Figure 2-9. Program Counter in PIC18

The Program Counter and ROM Space

- Executing a program byte by byte:
  - PIC microcontroller starts reading from 000000 (as mentioned thru 'org')
  - Upon executing the opcode 0E, the value 25H is loaded into WREG, & program counter is incremented to 000002
  - Upon executing the opcode 0F, the value 34H is added to WREG. Then the program counter is incremented to 000004
  - Note: each instruction is 2-byte wide. Hence the PC is incremented by 2-bytes every time.
  - Note: Last instruction GOTO is a 4-byte instruction.

Program 2-1: ROM Contents

Address	Code
000000	0E
000001	25
000002	0F
000003	34
000004	0F
000005	11
000006	0F
000007	12
000008	0F
000009	1C
00000A	0F
00000B	06
00000C	6E
00000D	10
00000E	07
00000F	EF
000010	00
000011	F0
000012	

ROM Width in PIC18

- PICs internal data bus between CPU and code ROM is 16-bit wide.
- The data bus is like a traffic lanes on a road. Where each lane is 8-bits wide. The more lanes you have, the more data can be brought into the CPU for processing.
- The 2M code space of PIC18 is organized as 1M x 16-bits instead of 2M x 8-bits.
- The reason behind having 16 bits data bus are:
  - Higher computation power
  - Every instruction in PIC18 is either **2-byte or 4-byte**. There is no 1-byte or 3-byte instruction.
  - This makes the instruction fetch operation a single cycle process.

ROM Width in PIC18

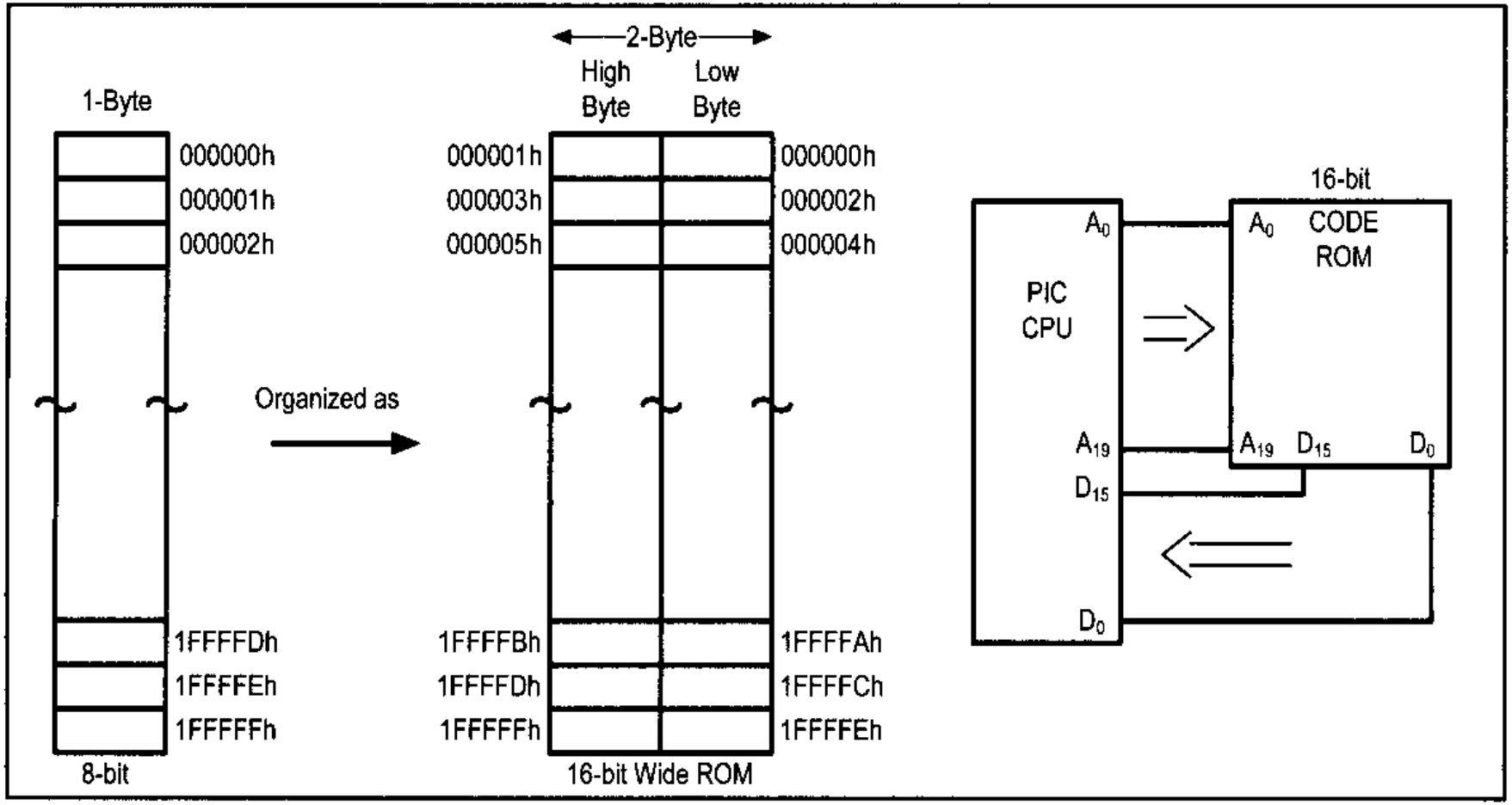
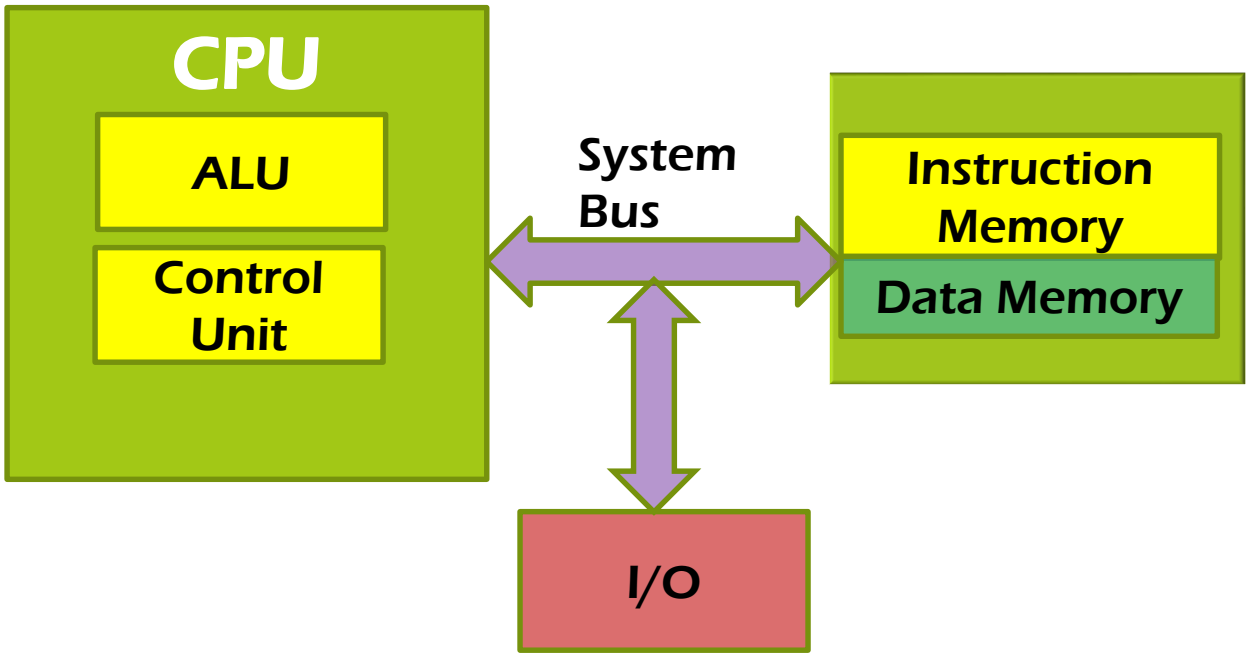


Figure 2-12. Program ROM Width for the PIC18

Von Neumann Architecture

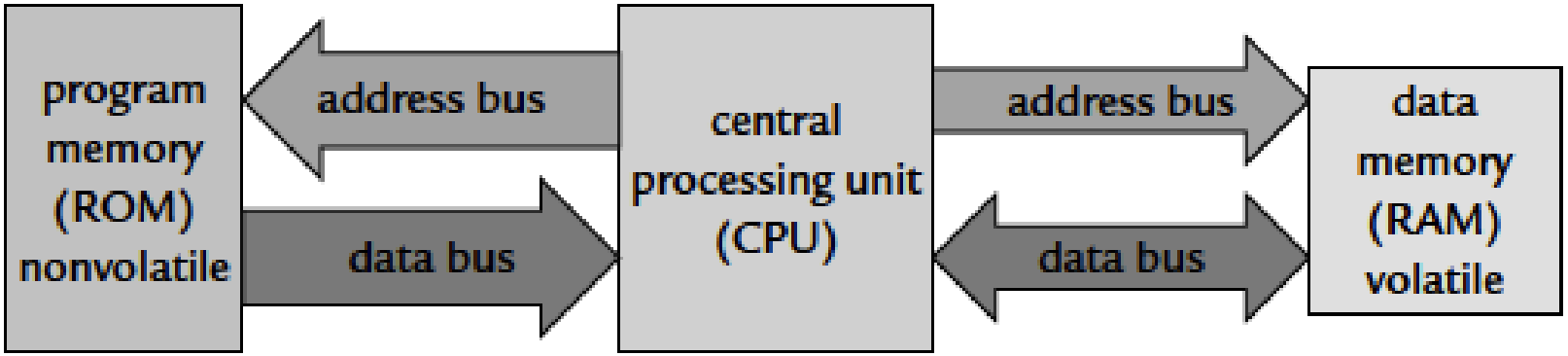
The Von Neumann Architecture is a computer architecture in which:

1. Both instruction and data are stored in the same memory
2. The contents are accessed by the location
3. Instructions are accessed and executed sequentially



Harvard Architecture

1. The **Harvard architecture** is a computer architecture with physically separate storage and signal pathways for instructions and data.
2. Today most computer use separate pathways and modify Harvard architecture.



Harvard Architecture in PIC

- **Harvard Architecture in PIC**

  - PIC has code ROM space and data RAM space.
  - Code provides instructions to the CPU while data provides information to be processed.
  - The CPU uses buses to access code in ROM and data in RAM.
  - Internally PIC is based on Harvard Architecture.

Harvard Architecture in PIC

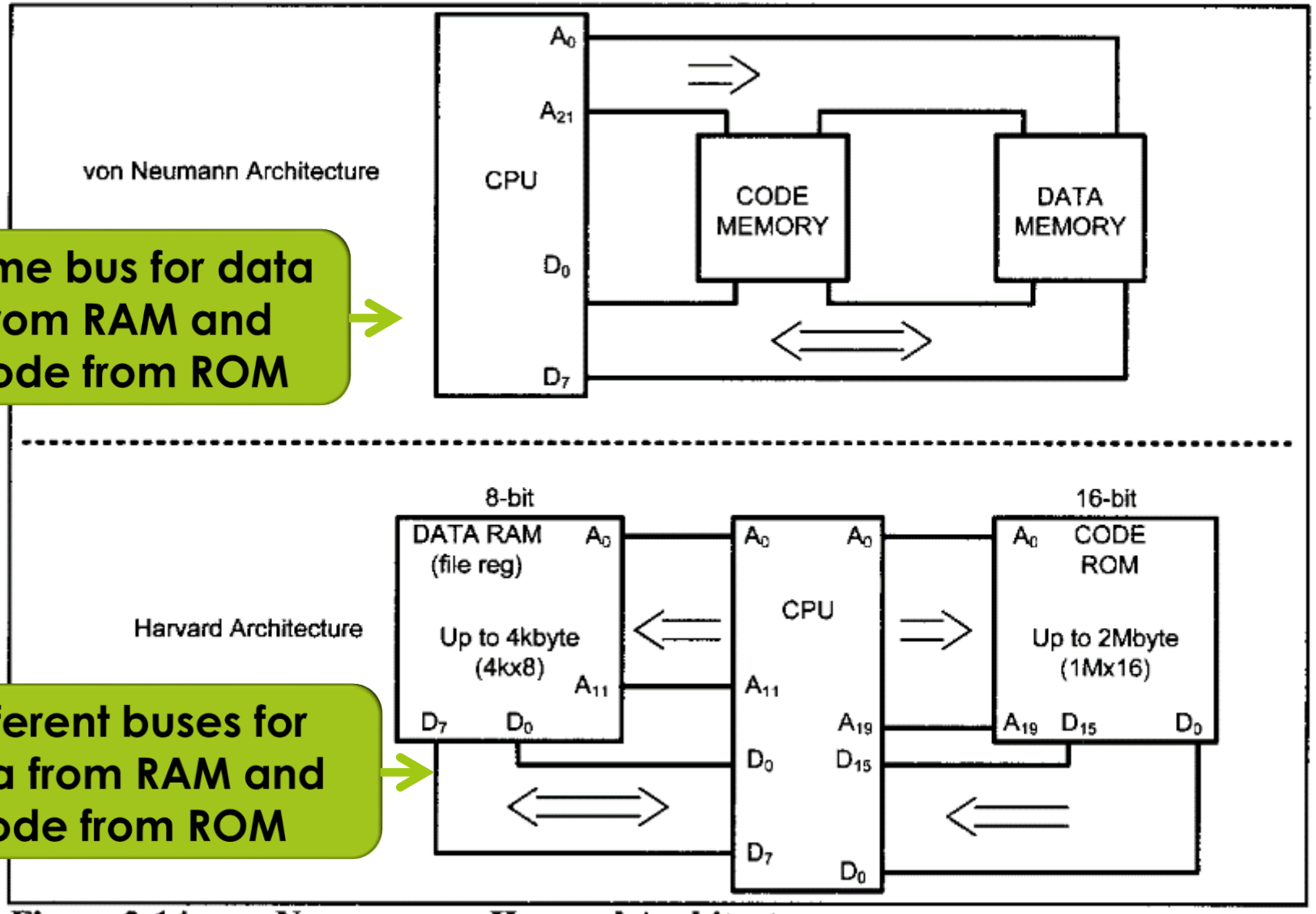


Figure 2-14. von Neumann vs. Harvard Architecture