國立成功大學
National Cheng Kung University
1931

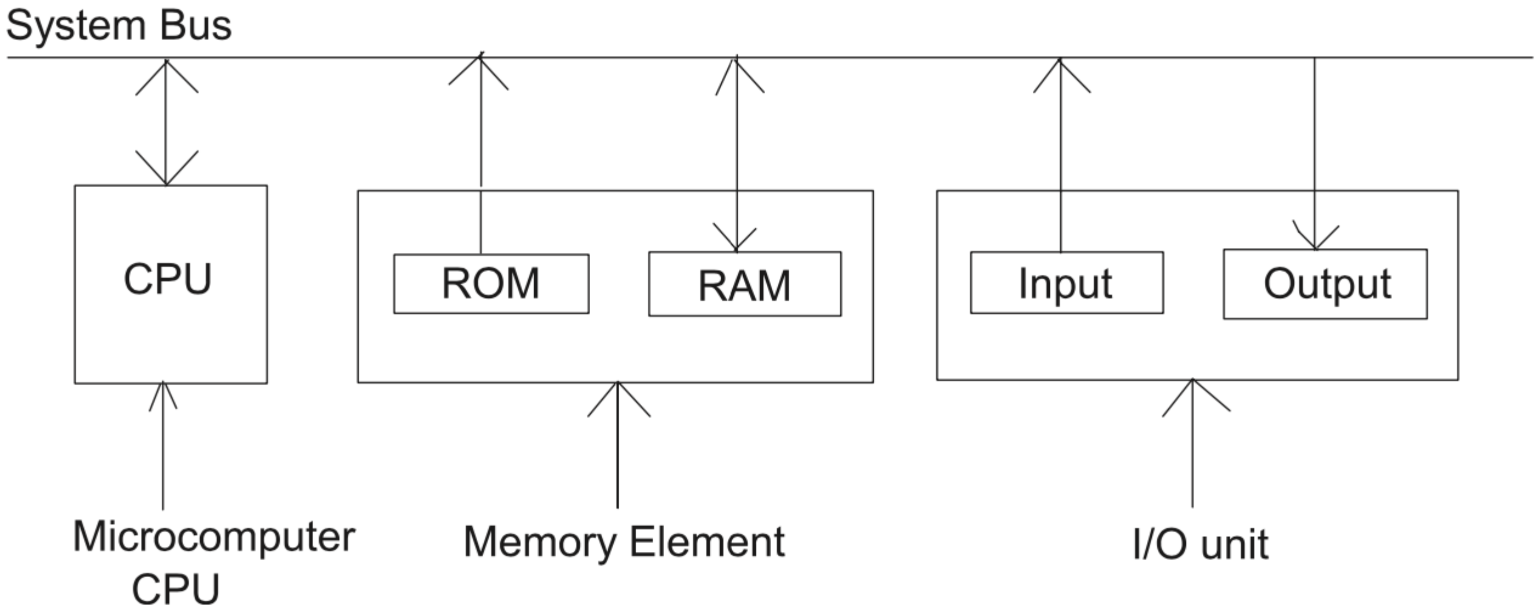
Introduction to Microcontroller

Chapter 2
Microcontroller Basics

Chien-Chung Ho (何建忠)

Basic Blocks of a Microcomputer

- A microcomputer has three basic blocks: CPU, a memory unit, and an input/output unit. A system bus (comprised of several wires) connects these blocks.



Basic Blocks of a Microcomputer

- A *memory* unit stores both instructions and data.
- The memory section typically contains ROM and RAM chips. The ROM can only be read and is nonvolatile. One can read from and write into a RAM. The RAM is volatile.

  非揮發性 → ex: SSD. HDD

  揮發性 → ex: DRAM
- An *I/O unit* transfers data between the microcomputer and the external devices via I/O ports (registers). The transfer involves data, status, and control signals.

Basic Blocks of a Microcomputer

- Microcontrollers are typically used for dedicated applications such as automotive systems, home appliances, and home entertainment systems.
- Microcontrollers include a CPU, memory, and IOP (I/O and Peripherals) on a single chip. Note that a typical IOP contains an I/O unit of a microcomputer, timers, an A/D (Analog-to-Digital) converter, analog comparators, a serial I/O, and other peripheral functions.

↳ CPU、mem、IO 封成一個 chip

Basic Blocks of a Microcomputer

- A very simplified version of a typical microcomputer

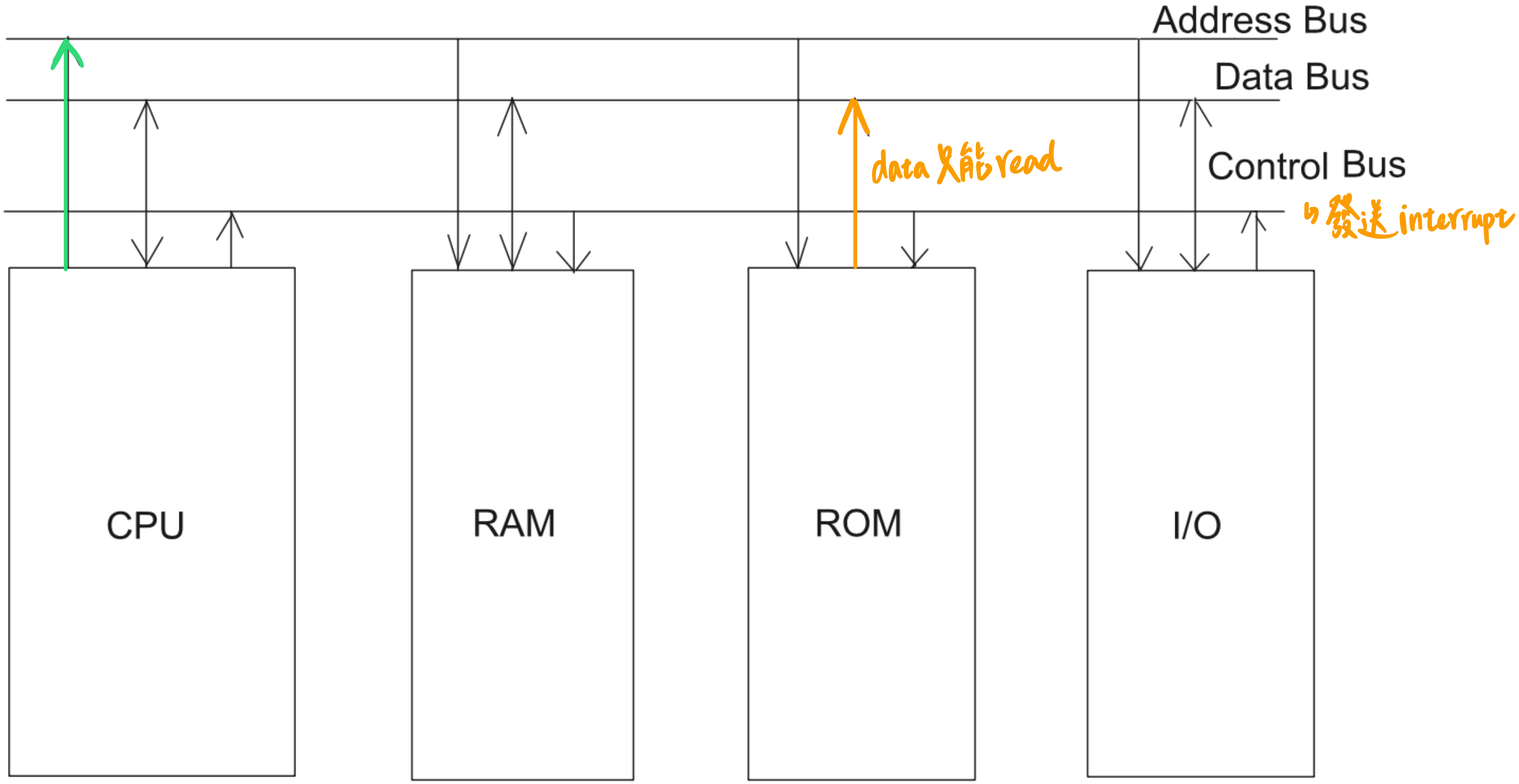


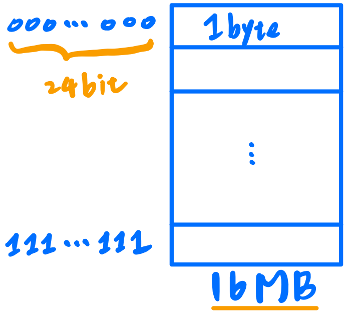
FIGURE 2.2 Simplified version of a typical microcomputer

System Bus

- The microcomputer's system bus (internal to the microcontroller) contains three buses that carry all the *address*, *data*, and *control* information involved in program execution.
- In the address bus, information transfer takes place in only one direction, from the microprocessor to the memory or I/O elements. Therefore, this is called a *unidirectional bus*.

單向 匯流排

System Bus



$16 = 2^4$
$M = 2^{20}$ ⇒ 2^{24} 個 byte

- The size of the address bus determines the total number of memory addresses available in which programs can be executed by the microprocessor.
- The address bus is specified by the total number of address bits required by the CPU. This also determines the direct addressing capability or the size of the main memory of the microcontroller.

System Bus

- For example, a CPU with 16 address bits can generate 2^{16} = 65,536 bytes [64 kilobytes (kB)] of different possible addresses on the address bus. The CPU includes addresses from 0 to $65{,}535_{10}$ (0000_{16} through $FFFF_{16}$). A memory location can be represented by each of these addresses. For example, an 8-bit data item $2B_{16}$ can be stored at 16-bit address 0200_{16}.

System Bus

- In the data bus, data can flow in both to or from the CPU. Therefore, this is a *bidirectional bus*.

  雙向

- The control bus consists of a number of signals that are used to synchronize operation of the individual microcomputer elements.

  ↳ 用來同步 operation

  The CPU sends some of these control signals to the other elements to indicate the type of operation being performed.

Clock Signals

- The system clock signals are contained in the control bus. These signals generate the appropriate clock periods during which instruction executions are carried out by the CPU. Typical microcontrollers have an internal clock generator circuit to generate a clock signal.
  - The CPU clock frequencies of typical microcontrollers vary from 1-MHz to 40-MHz

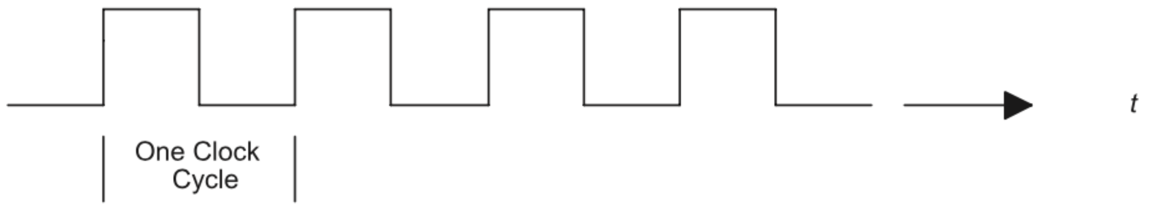


FIGURE 2.3 Typical clock signal.

Microcontroller architectures

- "Program memory" containing instructions and immediate data and "data memory" containing data whose addresses are specified with the instructions are used.

  ↳保存指令和 immediate data

  ↳存data

- Program memory and data memory may be of different sizes.
  - The PIC18F4321 contains a program memory with 13-bit address (4k × 16) and a data memory with 12-bit address (4k × 8).

Microcontroller architectures

- Harvard architecture is a type of CPU architecture which uses separate program and data memory units along with separate buses for *instructions* and *data*. This means that these processors can execute instructions and access data simultaneously.

  ↳ 分 program 和 data mem

- Processors designed with this architecture require four buses for program memory and data memory:
  - one data bus for instructions
  - one address bus for addresses of instructions
  - one data bus for data
  - one address bus for addresses of data.

Microcontroller architectures

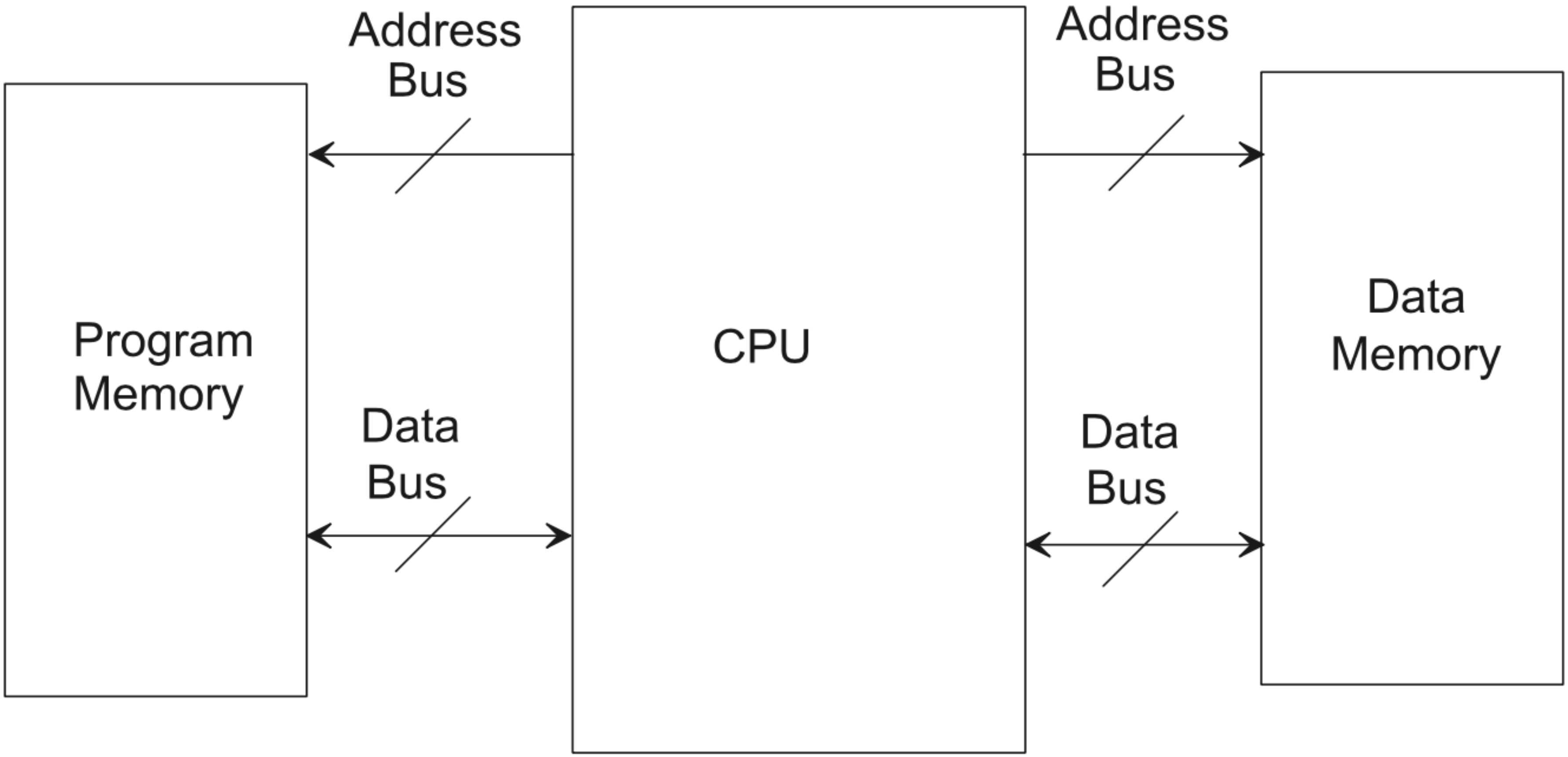


FIGURE 2.5 Harvard Architecture

Central Processing Unit (CPU)

- The power of the microcontroller is determined by the capabilities of the CPU. The CPU's clock frequency determines the speed of the microcontroller. The number of data and address bits on the CPU make up the microcontroller's *word* size and maximum memory size.
- The logic inside the CPU can be divided into three main areas: the *register section*, the *control unit*, and the *arithmetic-logic unit* (ALU).

Register Section -- Basic CPU Registers

- Basic CPU Registers: *instruction register*, *program counter*,
  ↳存 instruction
  memory address register, and *accumulator*.
  ex: WREG
- Instruction Register (IR). The instruction register stores instructions. The contents of an instruction register are always decoded by the CPU as an instruction. After fetching an instruction code from memory, the CPU stores it in the instruction register.

Register Section -- Basic CPU Registers

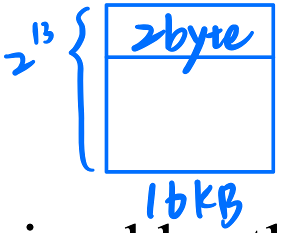
- Program Counter (PC). The program counter normally contains

  存下一個instruction位置

  the address of the next instruction to be executed.
  - Upon activating the CPU’s RESET input, the address of the first instruction to be executed is normally loaded into the program counter.
  - To execute an instruction, the CPU typically places the contents of the program counter on the address bus and reads (“fetches”) the contents of this address (i.e., instruction) from memory.

Register Section -- Basic CPU Registers

- Program Counter (PC). ↳ 大小由 address bus 定
  - The size of the program counter is determined by the size of the address bus.
  - Many instructions, such as JUMP and conditional JUMP, change the contents of the program counter from its normal sequential address value. The program counter is loaded with the address specified in these instructions.



$\frac{16KB}{2B} = 8K$

$= 2^{13}$個

↳ PC要13 bit

Register Section -- Basic CPU Registers

- Memory Address Register (MAR). The memory address register contains the address of data. The CPU uses the address as a direct pointer to memory.

  ▷存到哪存data

- Accumulator (A). The accumulator stores the results after most ALU operations. The accumulator is typically used for inputting a byte into the accumulator from an external device or for outputting a byte to an external device from the accumulator. The accumulator in PIC18F is called the *working register* (WREG).

  ↳ WREG

Register Section -- Basic CPU Registers

- Depending on the register section, the CPU can be classified either as an *accumulator* or *general-purpose register-based* machine.

  ex: add a,b,c // c=a+b

  ↳有很多 reg 可用

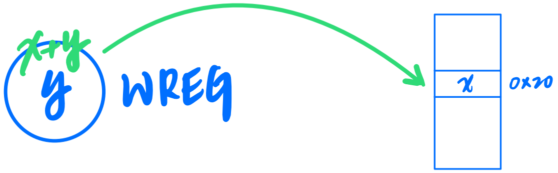
- In PIC18F, the data is assumed to be held in the accumulator. All arithmetic and logic operations are performed using this register as one of the data sources.

  accumulator based

  ↳運算都要經過 WREG

Register Section -- Use of CPU registers

- Assume that the address of program memory is 16-bit wide with 16-bit contents while the address of data memory is 8-bit wide with 8-bit contents.



- Suppose that the contents of the data memory address 0x20 are to be added to the contents of the accumulator, and then store the result in data memory address 0x20.
- Assume that [NNNN] represents the contents of the memory
  x ㄉ值
  location NNNN. Now, assume that [0x20] = 0x05.
  ㄉ位置

Register Section -- Use of CPU registers

The steps involved in adding [0x20] with the contents of the accumulator can be summarized as follows:

1. Load ‘A’ with the first data (0x02) to be added. y
2. Add the contents of the accumulator, ‘A’ to [0x20], and store the result in address, 0x20.
   y x

The following instructions for the PIC18F will be used to achieve the above addition:

Load

0x0E02 Load 0x02 into ‘A’ y

0x2620 Add [A] with [0x20] and store result address 0x20.

[W]=[W]+x

y = x + y

Op-codes

What instructions do

Register Section -- Use of CPU registers

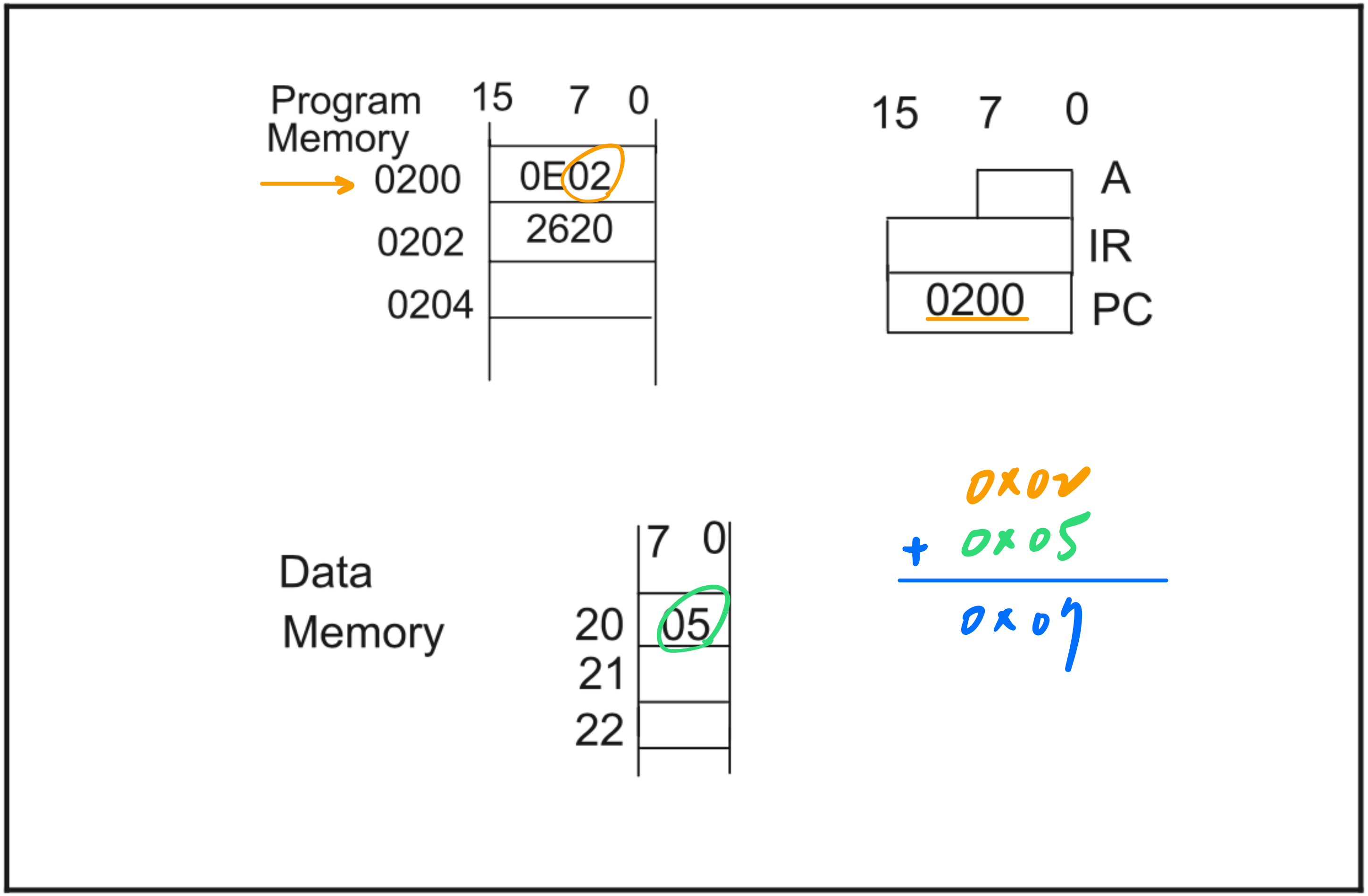


FIGURE 2.7 Addition program with initial register and memory contents

Register Section -- Use of CPU registers

- Program memory address stores 16 bits. Hence, memory addresses are shown in increments of 2. Data memory stores 8-bit data. Hence, memory addresses are shown in increments of 1.
- Assume that the CPU can be instructed so that the starting address of the program is 0x0200 (the program counter can be initialized to contain 0x0200).
- The CPU loads the contents of memory location addressed by the program counter into IR. Thus, the first instruction $0E02_{16}$ stored in address 0x200 is transferred into IR.

Register Section -- Use of CPU registers

- The program counter contents are then incremented by 2 by the ALU to hold 0x0202. The register contents along with the program are shown in Figure 2.8.

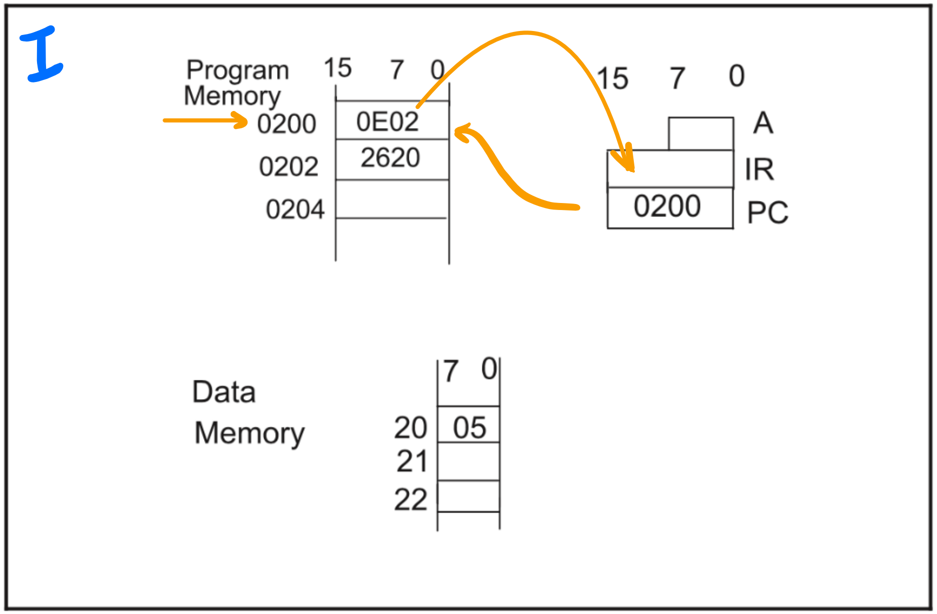


FIGURE 2.7 Addition program with initial register and memory contents

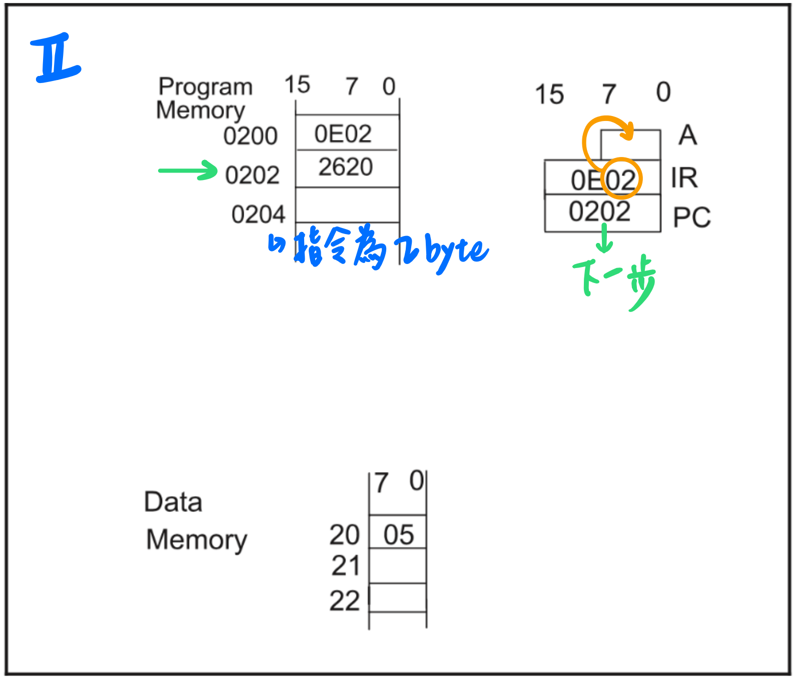


FIGURE 2.8 Addition program (modified during execution)

Register Section -- Use of CPU registers

- The binary code 0x0E02 in the IR is executed by the CPU. The CPU then takes appropriate actions. Note that the instruction 0x0E02 loads 0x02 into ‘A’ register. This is shown in Figure 2.9.

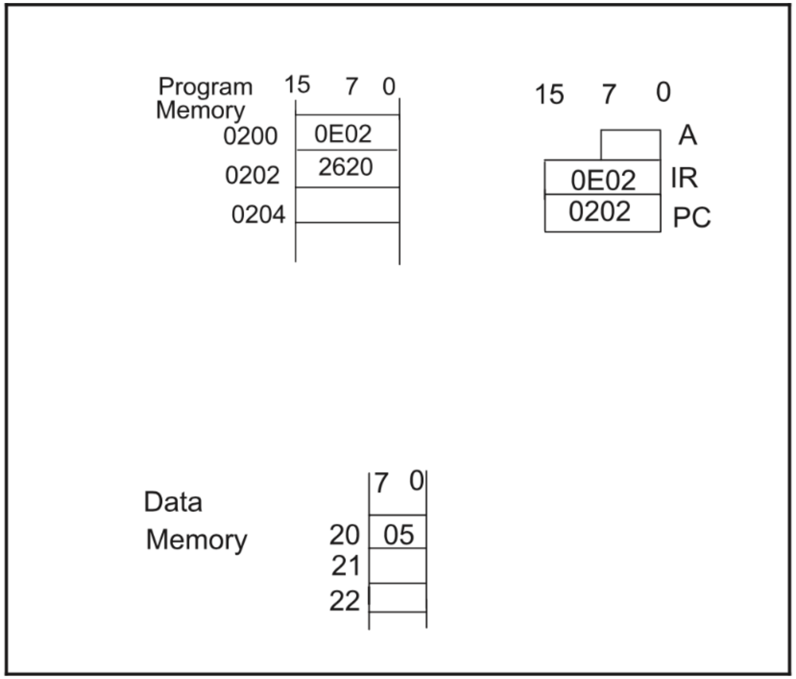


FIGURE 2.8 Addition program (modified during execution)

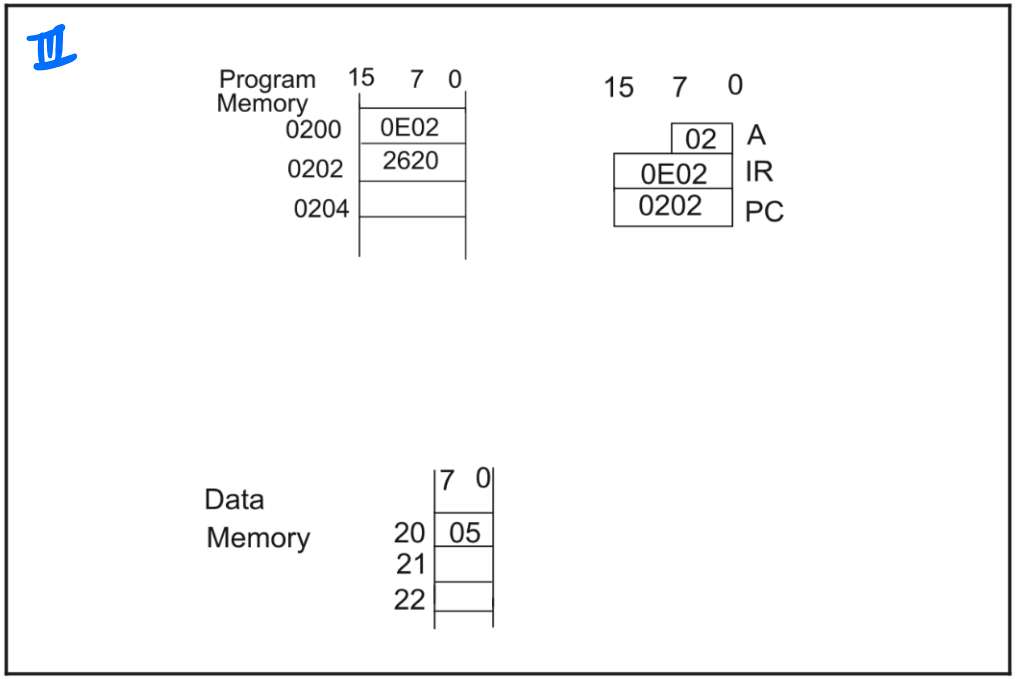


FIGURE 2.9 Addition program (modified during execution)

Register Section -- Use of CPU registers

- Next, the CPU loads the contents of the memory location addressed by the PC into the IR; thus, 0x2620 is loaded into the IR. The PC contents are then incremented by 2 to hold 0x0204.

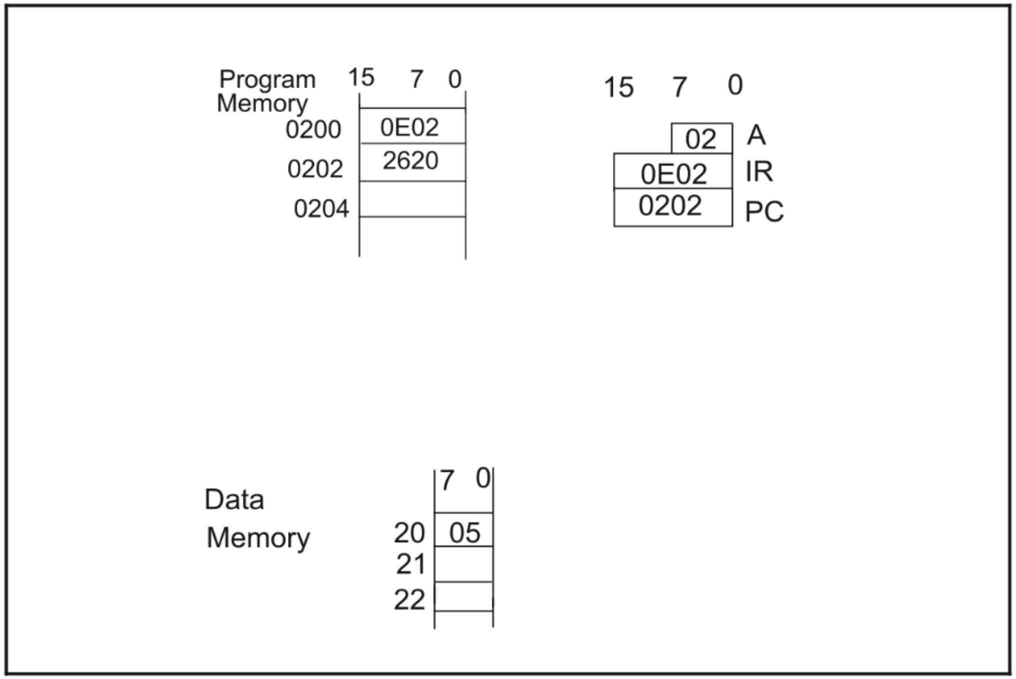


FIGURE 2.9 Addition program (modified during execution)

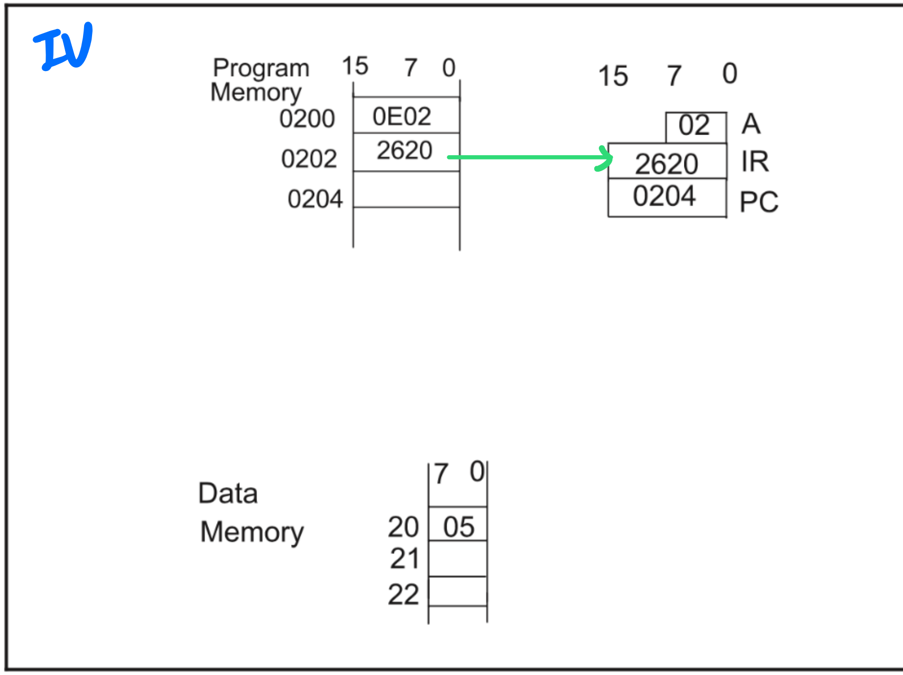


FIGURE 2.10 Addition program (modified during execution)

Register Section -- Use of CPU registers

- In response to the instruction 0x2620, the contents of the data memory location addressed by the data memory address 0x20 are added to the contents in the accumulator A; thus, 0x05 is added to 0x02.
- The result 0x07 is stored in data memory address 0x20. Note that the previous contents (0x05) of data memory address 0x20 are lost. The contents of the PC are not incremented this time. This is because 0x05 is obtained from data memory.

Register Section -- Use of CPU registers

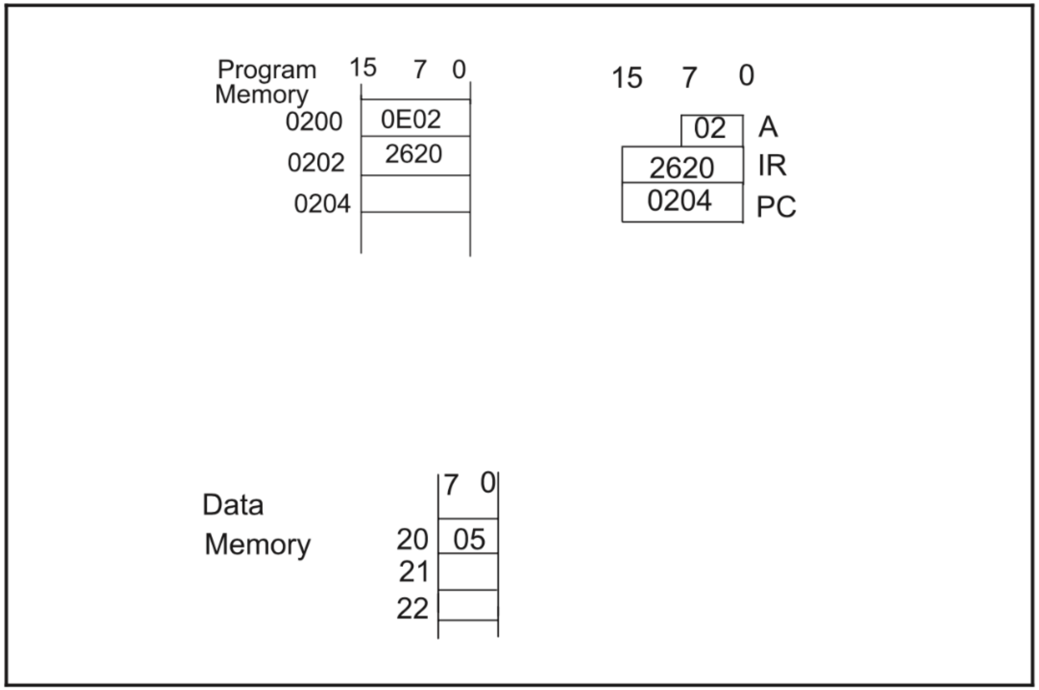


FIGURE 2.10 Addition program (modified during execution)

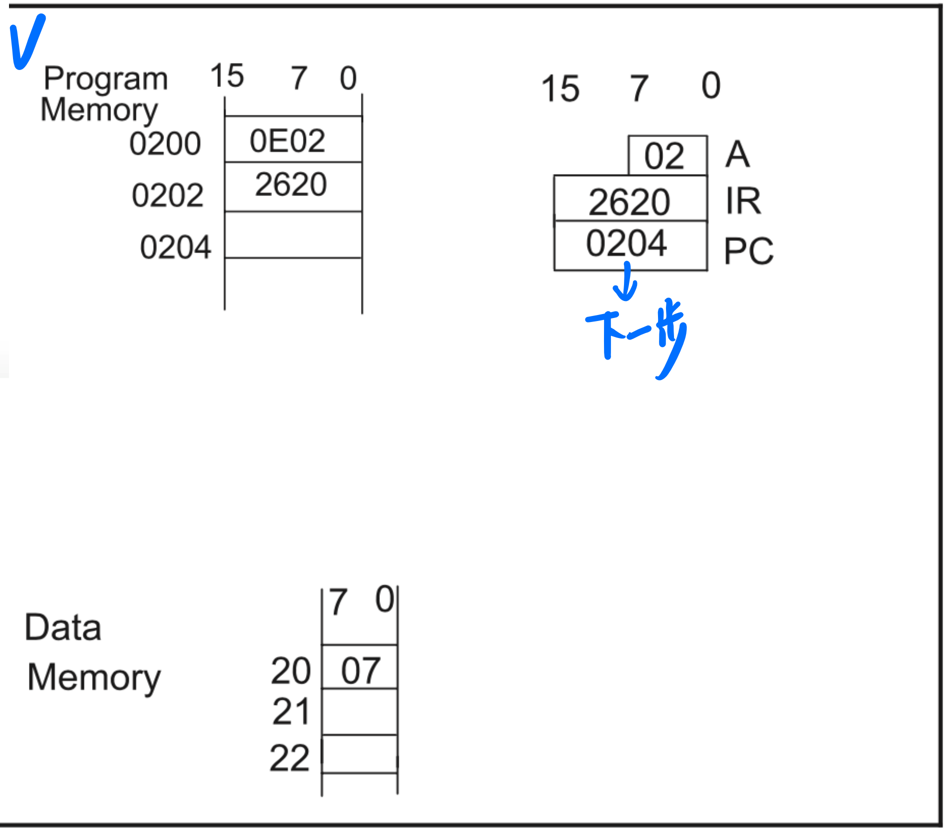


FIGURE 2.11 Addition program (modified during execution)

Register Section -- Index Register

- PIC18F provides indexed addressing mode using an *index register* to access an element in an array. ex: $\overset{addr}{A[0]} + n = \overset{addr}{A[n]}$
- The effective address for an instruction using the *indexed addressing* mode is determined by adding the address portion of the instruction to the contents of the index register. Note that the accumulator is used as the index register in the PIC18F.

Register Section -- Status Register

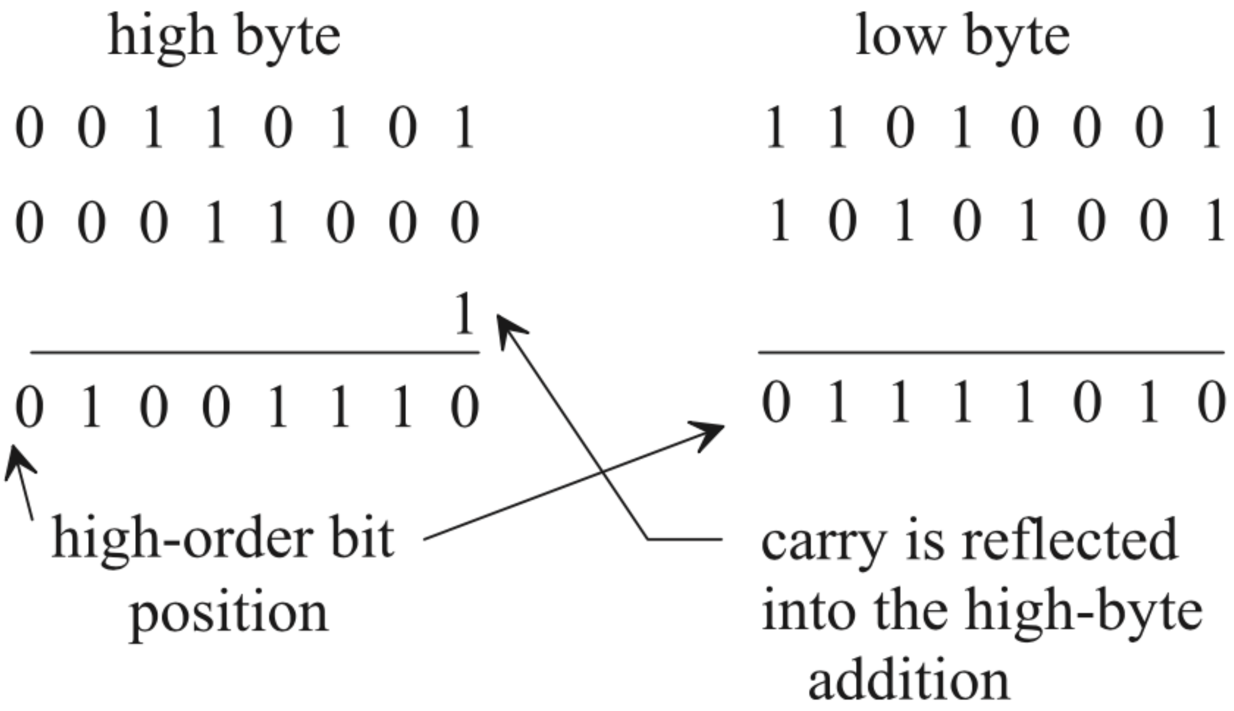
- A status register contains individual bits with each bit having special significance. The bits in the status register are called *flags*.
- A *carry flag* is used to reflect whether or not the result generated by an arithmetic operation is greater than the microcontroller’s word size.

↳判別進位

Register Section -- Status Register

- For example, the addition of two 8-bit numbers might produce a carry. The carry is generated out of the 8th bit position (bit 7), which results in setting the carry flag. However, the carry flag will be zero if no carry is generated from the addition.
- In multibyte arithmetic, any carry out of the low-byte addition must be added to the high-byte addition to obtain the correct result.

Register Section -- Status Register



high byte

0 0 1 1 0 1 0 1

0 0 0 1 1 0 0 0

1

0 1 0 0 1 1 1 0

low byte

1 1 0 1 0 0 0 1

1 0 1 0 1 0 0 1

0 1 1 1 1 0 1 0

high-order bit position

carry is reflected into the high-byte addition

Register Section -- Status Register

- While performing BCD arithmetic with microcontrollers, the carry out of the low nibble (4 bits) has a special significance. Because a BCD digit is represented by 4 bits, any carry out of the low 4 bits must be propagated into the high 4 bits for BCD arithmetic. This carry flag is known as "*Digit Carry* (DC)" flag

只看後4 bit相加
有無進位

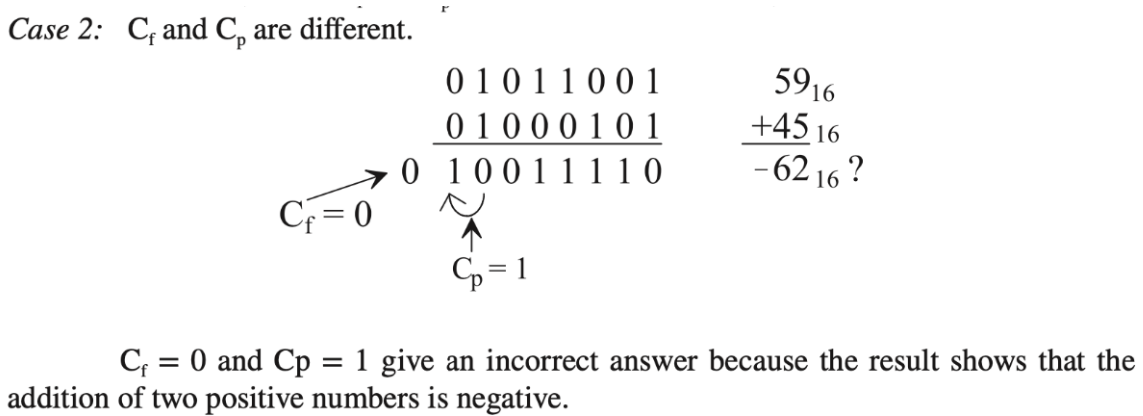
Register Section -- Status Register

- A *zero flag* is used to show whether the result of an operation is zero. It is set to 1 if the result is zero, and it is reset to 0 if the result is nonzero.
- A *sign flag* (sometimes called a negative flag) is used to indicate whether the result of the last operation is positive or negative. If the most significant bit of the last operation is 1, this flag is set to 1 to indicate that the result is negative.

Register Section -- Status Register

- An *overflow flag* arises from representation of the sign flag by the most significant bit of a word in signed binary operation. The overflow flag is set to 1 if the result of an arithmetic operation is too big for the microcontroller's maximum word size.
- Let C_f be the final carry out of the most significant bit (sign bit) and C_p be the previous carry. The overflow flag is the exclusive-OR of the carries C_p and C_f.

$$\text{overflow} = C_p \oplus C_f$$



Case 2: C_f and C_p are different.

$$\begin{array}{rr} & 0\,1\,0\,1\,1\,0\,0\,1 \\ & \underline{0\,1\,0\,0\,0\,1\,0\,1} \\ 0 & 1\,0\,0\,1\,1\,1\,1\,0 \end{array} \qquad \begin{array}{r} 59_{16} \\ \underline{+45_{16}} \\ -62_{16}\,? \end{array}$$

$C_f = 0$

$C_p = 1$

$C_f = 0$ and Cp = 1 give an incorrect answer because the result shows that the addition of two positive numbers is negative.

Register Section -- Stack Pointer Register

- A stack consists of a number of RAM locations set aside for reading data from or writing data into these.

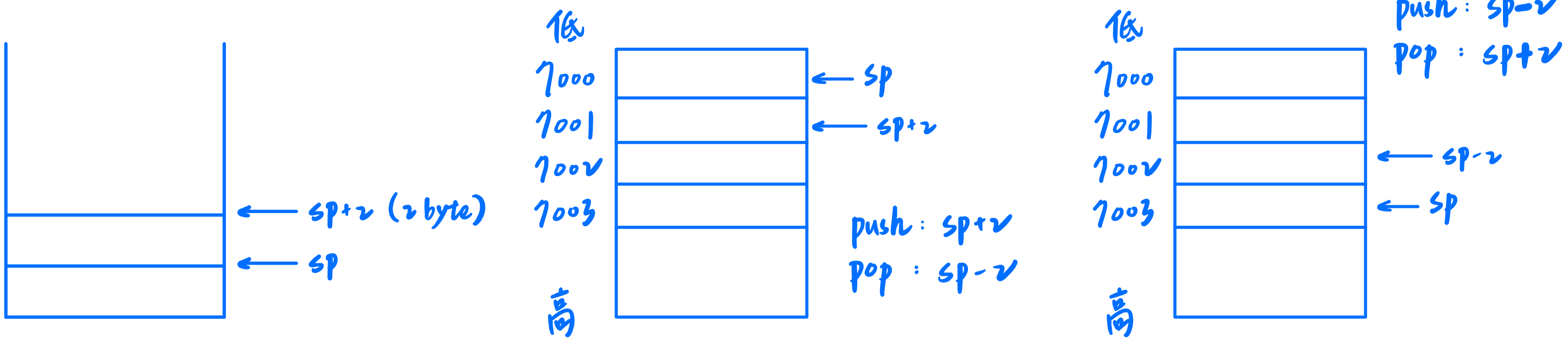
↳ 存 return address

- The address of the stack is contained in a register called a *stack pointer*.
- Two instructions, PUSH (writing to the stack) and POP (reading from the stack), are usually available with a stack. Some microcontrollers access the stack from the top; others access via the bottom.

Register Section -- Stack Pointer Register

- The PUSH operation is defined as writing to the top or bottom of the stack, whereas the POP operation means reading from the top or bottom of the stack.
- When the stack is accessed from the bottom, the stack pointer is incremented after a PUSH and decremented after a POP operation.

* sp 的加減視方向而定



Register Section -- Stack Pointer Register

- In Figure 2.12, the stack pointer is incremented by 2 (16-bit register) after the PUSH to contain the value 20CA.
- Now, consider the POP operation of Figure 2.13. The stack pointer is decremented by 2 after the POP. The contents of address 20CA are assumed to be empty conceptually after the POP operation.

Register Section -- Stack Pointer Register

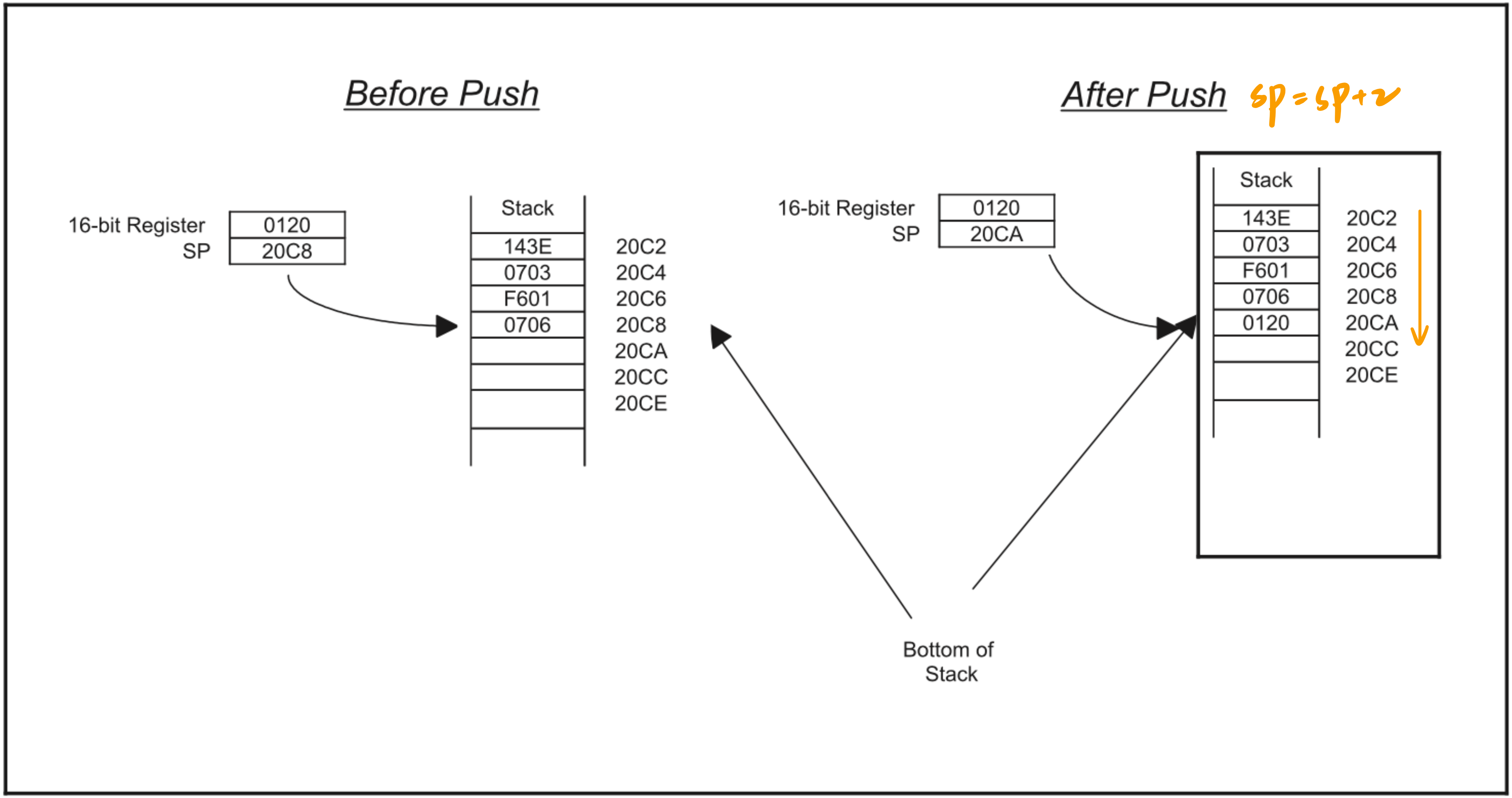


FIGURE 2.12 PUSH operation when accessing a stack from the bottom.

Register Section -- Stack Pointer Register

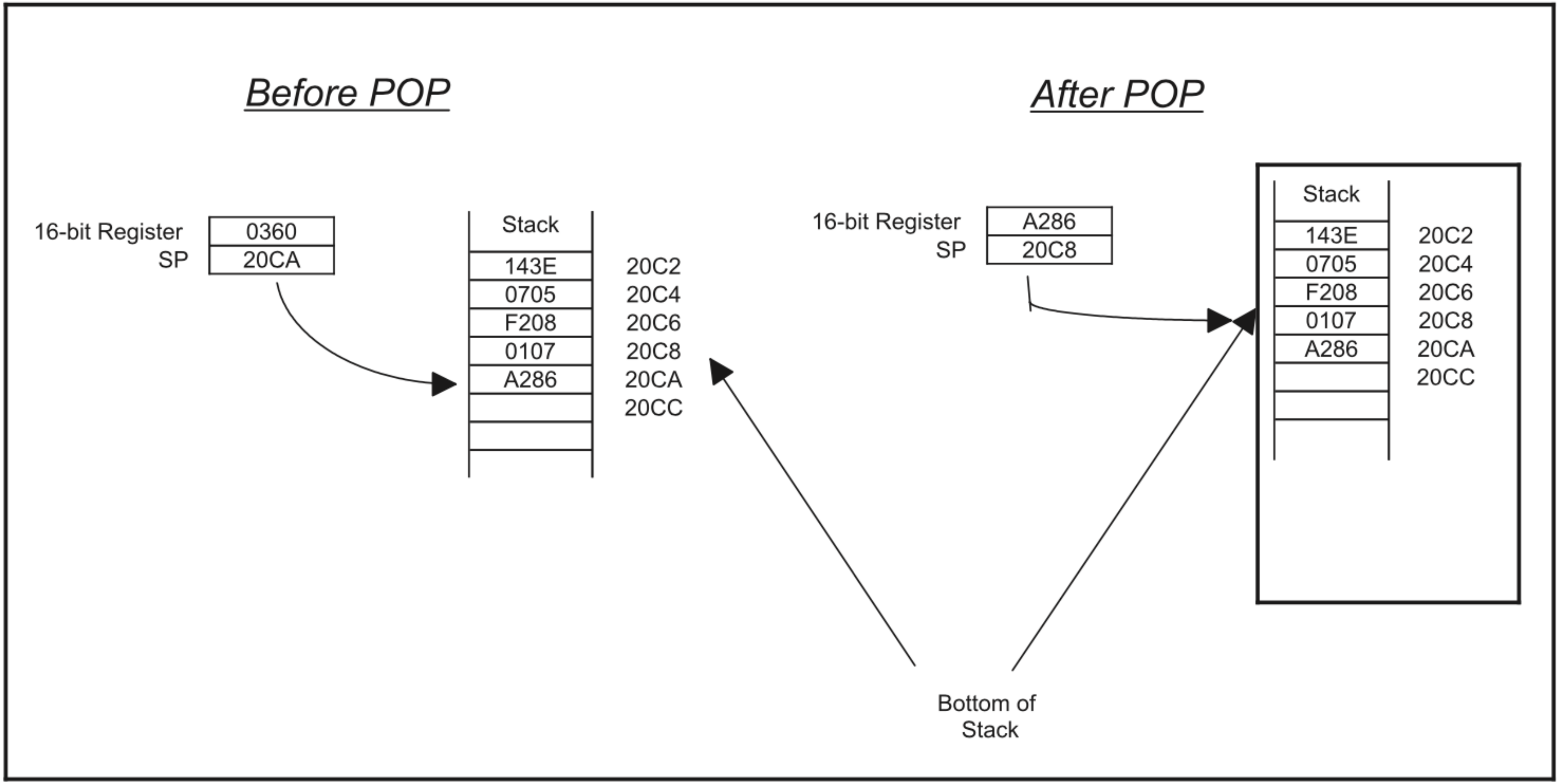


FIGURE 2.13 POP operation when accessing a stack from the bottom.

Register Section -- Stack Pointer Register

- Consider the PUSH operation of Figure 2.14. The stack is accessed from the top. The stack pointer is decremented by 2 after a PUSH.
- Finally, consider the POP operation of Figure 2.15. The Stack pointer is incremented by 2 after the POP. The contents of address 20C6 are assumed to be empty conceptually after a POP operation.

Register Section -- Stack Pointer Register

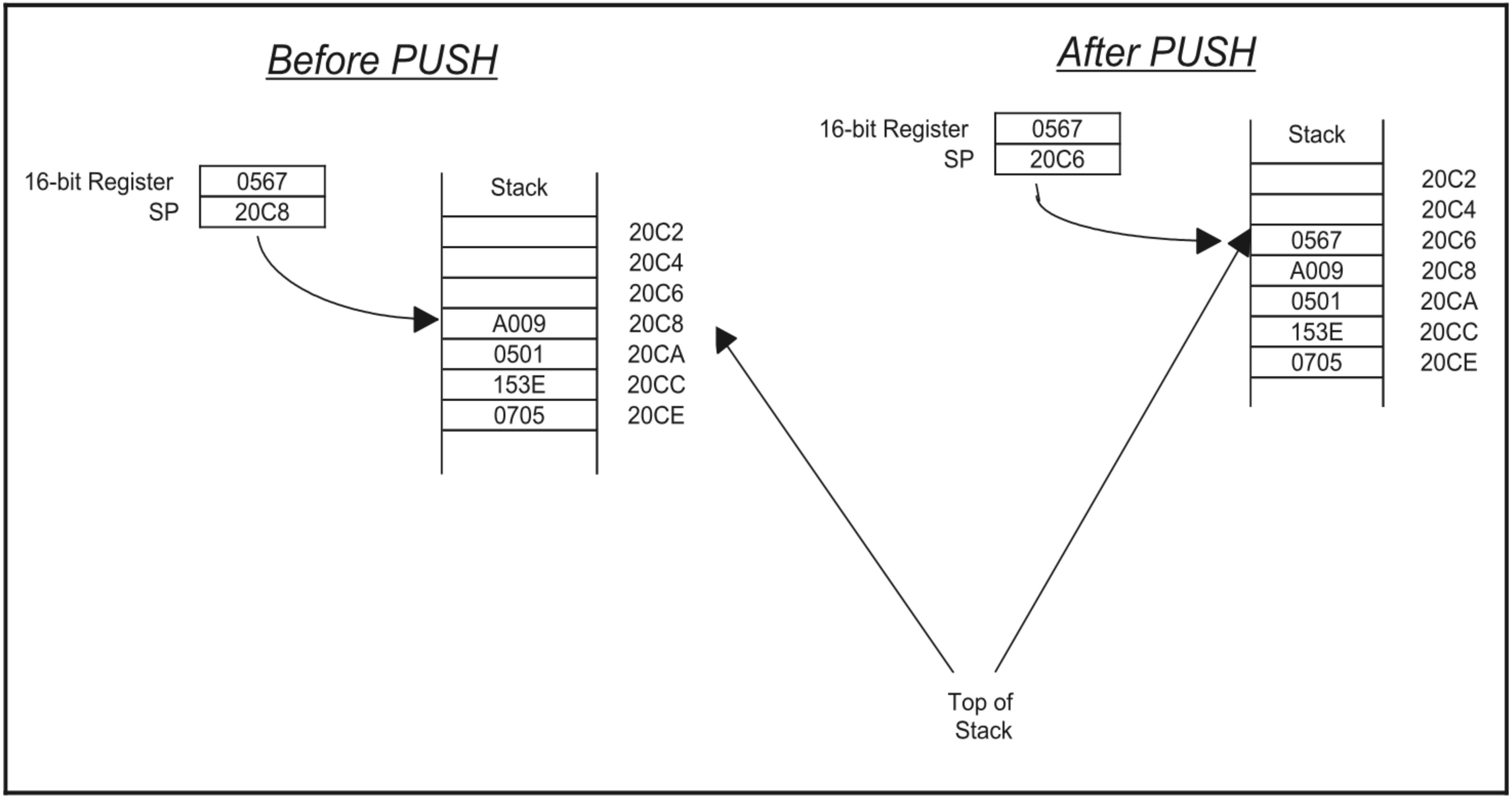


FIGURE 2.14 PUSH operation when accessing a stack from the top.

Register Section -- Stack Pointer Register

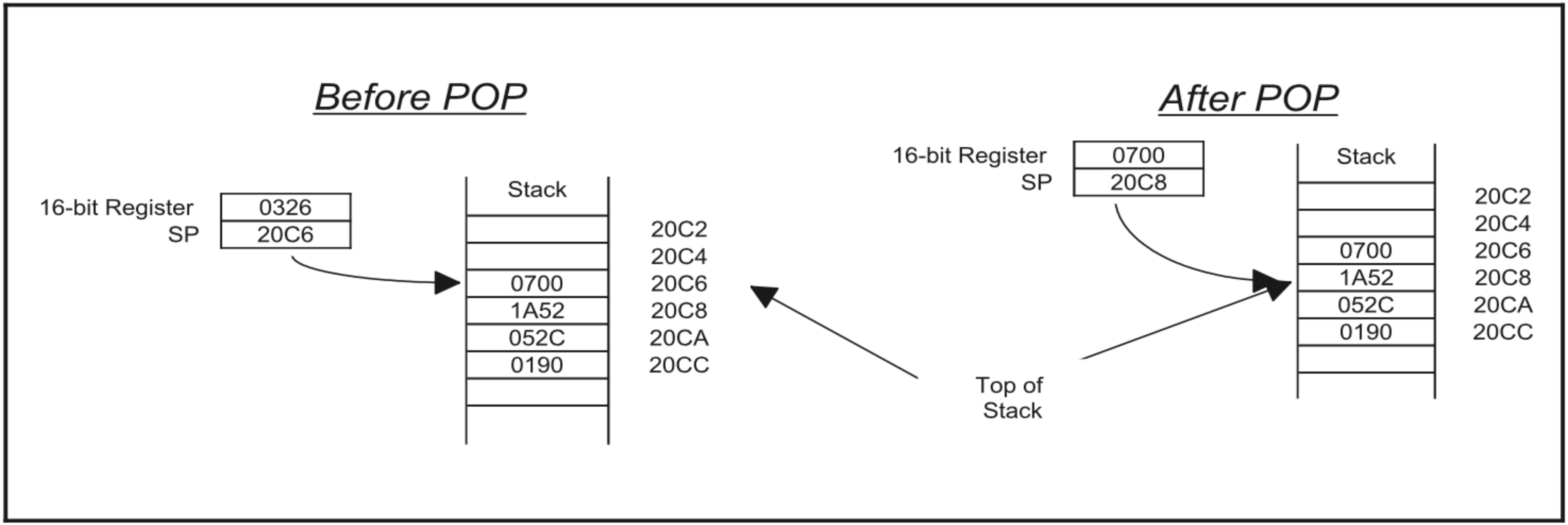


FIGURE 2.15 POP operation when accessing a stack from the top.

Register Section -- Stack Pointer Register

- The stack is a LIFO (last in first out) memory. A stack is typically used during subroutine CALLs. The CPU automatically PUSHes the return address onto a stack after executing a subroutine CALL instruction. After executing a RETURN from a subroutine instruction, the CPU automatically POPs the return address from the stack (previously PUSHed) and then returns control to the main program.

↳用stack存 return addr

Register Section -- Stack Pointer Register

- Note that the PIC18F *accesses stack from the bottom*. This means that the stack pointer in the PIC18F holds the address of the bottom of the stack. Hence, in the PIC18F, the stack pointer is *incremented after a PUSH, and decremented after a POP*.

Control Unit

- The main purpose of the control unit is to *read and decode instructions* from the program memory.
- The control unit interprets the contents of the instruction register and then responds to the instruction by generating a sequence of enable signals. These signals activate the appropriate ALU logic blocks to perform the required operation.

Control Unit

- Control Bus Signals. The control unit generates the control signals, which are output to the other microcontroller elements via the control bus.
- The control unit also takes appropriate actions in response to the control signals on the control bus provided by the other microcontroller elements.

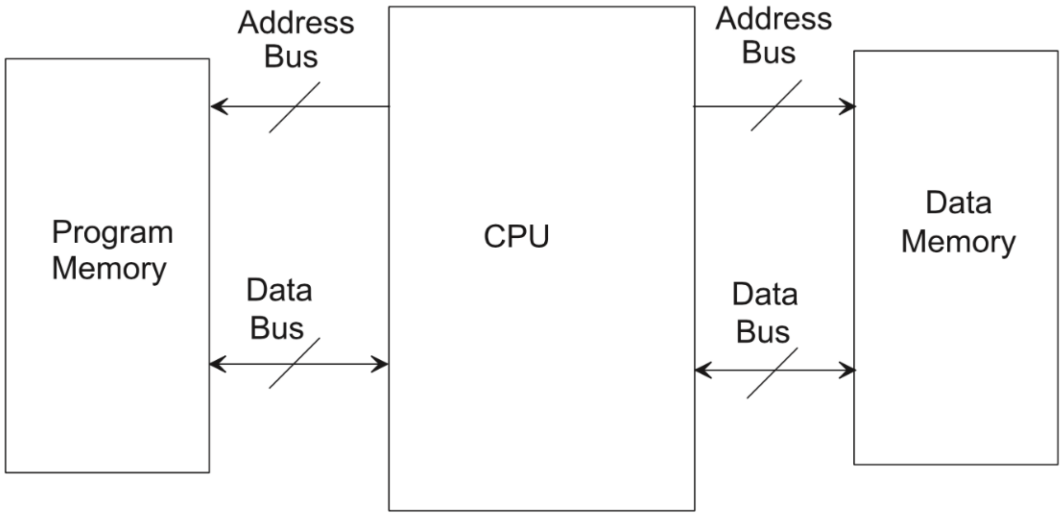


FIGURE 2.5 Harvard Architecture

Control Unit

$\overline{\textit{RESET}}$. This input is common to all CPUs. When this input pin is driven HIGH or LOW (depending on the CPU), the program counter is loaded with a predefined address specified by the manufacturer. As mentioned before, the PIC18F upon hardware reset loads the 21-bit program counter with 0's. This means that the instruction stored at memory location 0 is executed first.

↪ (1/0)

READ/ $\overline{\textit{WRITE}}$ (R/$\overline{\textit{W}}$). This output line is common to all CPUs. The status of this line tells the other microcontroller elements whether the CPU is performing a READ or a WRITE operation. A HIGH signal on this line indicates a READ operation, and a LOW signal indicates a WRITE operation. Some CPUs have separate READ and WRITE inputs.

INTERRUPT REQUEST. The external I/O devices can interrupt the microcontroller via this input signal on the CPU. When this signal is activated by the external devices, the CPU jumps to a special program called the *interrupt service routine*. This program is normally written by the user for performing tasks that the interrupting device wants the CPU to carry out. After completing this program, the CPU returns to the main program it was executing when the interrupt occurred. This topic will be covered in more detail in chapters 3, 8, 9, and 10.

Arithmetic and Logic Unit (ALU)

- The size of the ALU conforms to the word length of the microcontroller. This means that an 8-bit microcontroller will have an 8-bit ALU. Some of the typical functions:
  - Binary addition and logic operations
  - Finding the one’s complement of data
    ↳ 01互換
  - Shifting or rotating the contents of a general-purpose register 1 bit to the left or right through a carry

Control Unit Design

- The main purpose of the control unit is to translate or decode instructions and generate appropriate enable signals to accomplish the desired operation. Based on the contents of the instruction register, the control unit sends the data items selected to the appropriate processing hardware at the right time. The control unit drives the associated processing hardware by generating a set of signals that are synchronized with a master clock.

Control Unit Design

- The control unit performs two basic operations: *instruction interpretation* and *instruction sequencing*. In the interpretation phase, the control unit reads (fetches) an instruction from the memory into the instruction register. It recognizes the instruction type, obtains the necessary operands, and routes them to the appropriate functional units of the execution unit (registers and ALU).

Control Unit Design

- The control unit then issues the necessary signals to the execution unit to perform the desired operation and routes the results to the destination specified.
- In the sequencing phase, the control unit generates the address of the next instruction to be executed and loads it into the program counter.

Basic Concept of Pipelining

- Typical microcontrollers such as the PIC18F utilize the control unit and ALU efficiently by prefetching the next instruction(s) and the required data before the control unit and ALU require them.

Basic Concept of Pipelining

- Assume that a task T is carried out by performing four activities: Al, A2, A3, and A4, in that order. Hardware Hi is designed to perform activity Ai.

  ↳拆成不同階段

- Consider the arrangement shown in Figure 2.16. In this configuration, a latch is placed between two segments so the result computed by one segment can serve as an input to the following segment during the next clock period.

Basic Concept of Pipelining

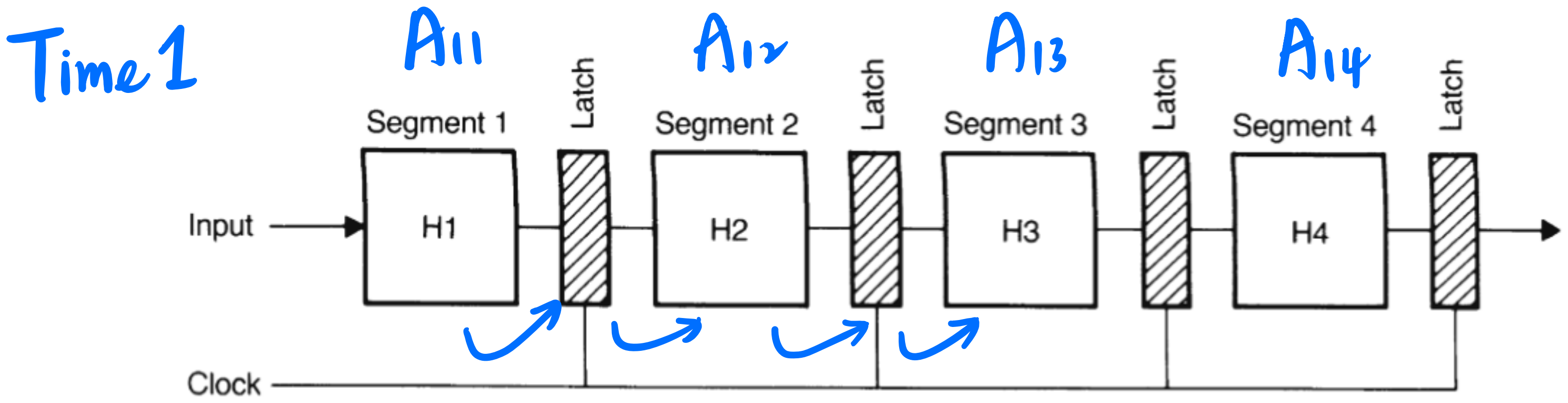


FIGURE 2.16 Four-segment pipeline.

Basic Concept of Pipelining

- Initially, task Tl is handled by segment 1. After the first clock, segment 2 is busy with Tl while segment 1 is busy with T2. Continuing in this manner, task Tl is completed at the end of the fourth clock.
- Following this point, one task is shipped out per clock. This is the essence of the pipelining concept.

Basic Concept of Pipelining

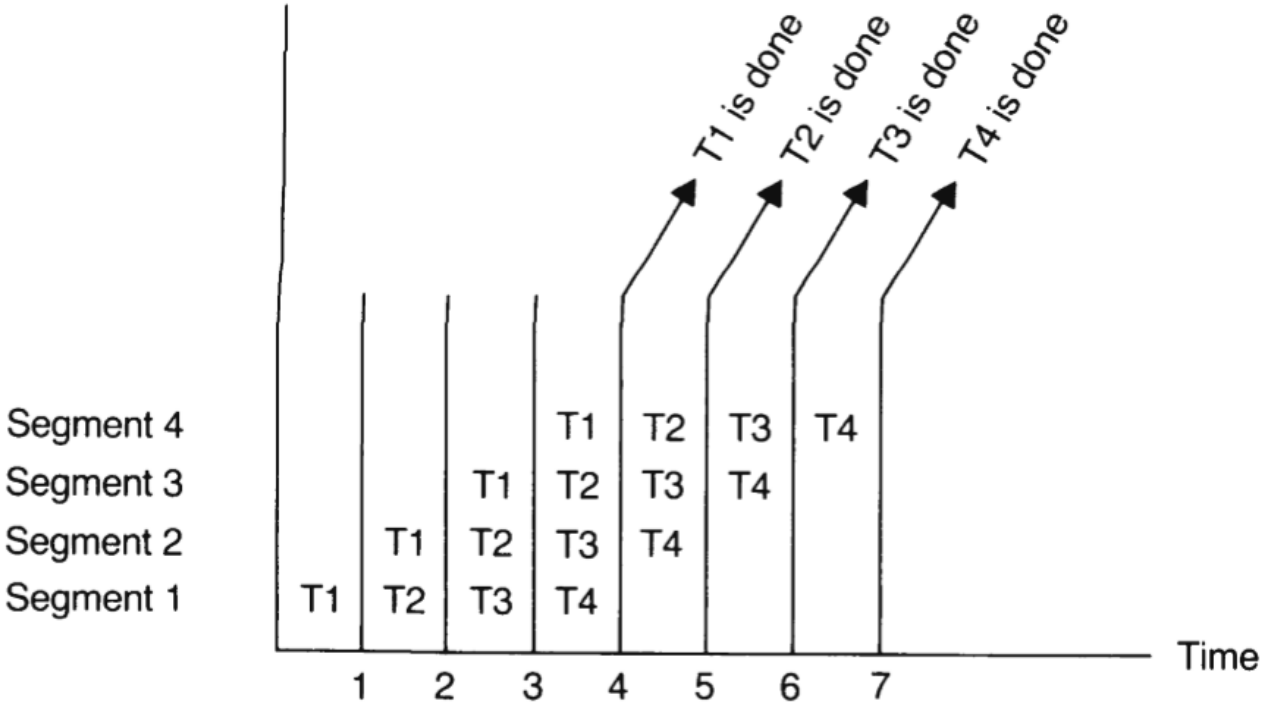


FIGURE 2.17 Overlapped execution of four tasks using a pipeline.

RISC vs. CISC

- The goal of RISC architecture is to maximize the effective speed
  ↳減少不必要的指令
  of a design by performing infrequent operations in software and frequent functions in hardware.
- RISC CPUs are suitable for embedded applications.
- RISC CPUs are well suited for applications such as image processing, robotics, and instrumentation.
- RISC supports a small set of commonly used instructions that are executed at a fast clock rate compared to CISC.

RISC vs. CISC

- CISC is more difficult to pipeline; RISC provides more efficient pipelining.
- An advantage of CISC over RISC is that complex programs require fewer instructions in CISC with fewer fetch cycles, while RISC requires a large number of instructions to accomplish the same task with several fetch cycles.