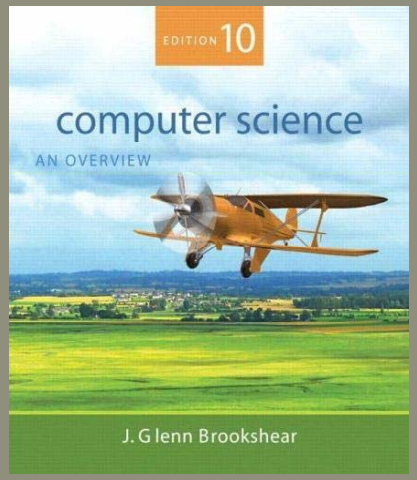
Chapter 2

Data Manipulation
資料處理

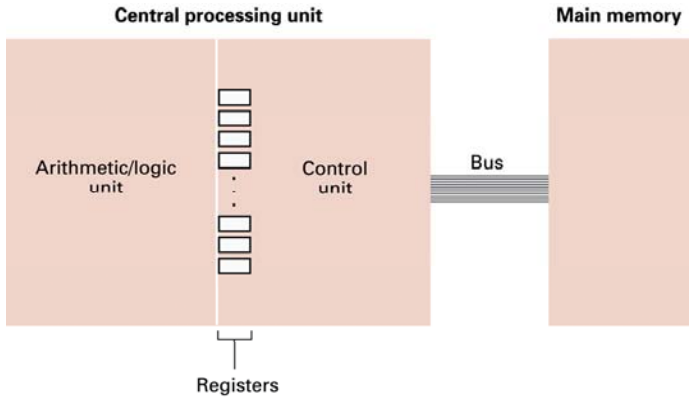


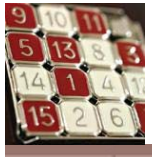
Computer Architecture

- Central Processing Unit (CPU) or processor
  - Arithmetic/Logic unit versus Control unit
  - Registers
    - General purpose
    - Special purpose
- Bus (匯流排)
- Motherboard (主機板)

Chapter 2: Data Manipulation

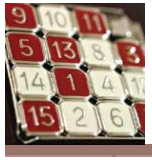
Figure 2.1 CPU and main memory connected via a bus





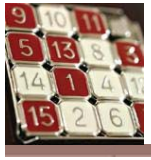
Stored Program Concept

A program can be encoded as bit patterns and stored in main memory. From there, the CPU can then extract the instructions and execute them. In turn, the program to be executed can be altered easily.



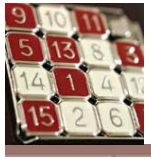
Terminology

- **Machine instruction:** An instruction (or command) encoded as a bit pattern recognizable by the CPU
- **Machine language:** The set of all instructions recognized by a machine



Machine Language Philosophies

- Reduced Instruction Set Computing (RISC)
  - Few, simple, efficient, and fast instructions
  - Example: PowerPC from Apple/IBM/Motorola and SPARK from Sun Microsystems
- Complex Instruction Set Computing (CISC)
  - Many, convenient, and powerful instructions
  - Example: Pentium from Intel



Machine Instruction Types

- Data Transfer: copy data from one location to another
- Arithmetic/Logic: use existing bit patterns to compute a new bit patterns
- Control: direct the execution of the program

Figure 2.2 Adding values stored in memory

Step 1. Get one of the values to be added from memory and place it in a register.

Step 2. Get the other value to be added from memory and place it in another register.

Step 3. Activate the addition circuitry with the registers used in Steps 1 and 2 as inputs and another register designated to hold the result.

Step 4. Store the result in memory.

Step 5. Stop.

Figure 2.3 Dividing values stored in memory

Step 1. LOAD a register with a value from memory.

Step 2. LOAD another register with another value from memory.

Step 3. If this second value is zero, JUMP to Step 6.

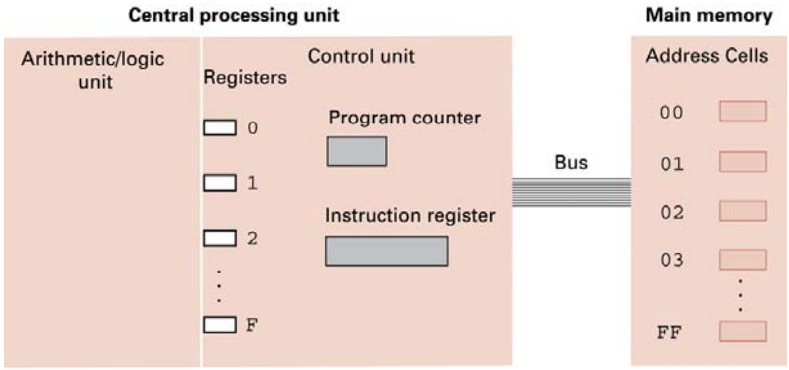
判斷除數不得為0

Step 4. Divide the contents of the first register by the second register and leave the result in a third register.

Step 5. STORE the contents of the third register in memory.

Step 6. STOP.

Figure 2.4 The architecture of the machine described in Appendix C



Parts of a Machine Instruction

- **Op-code(運算碼):** Specifies which operation to execute
- **Operand(運算元):** Gives more detailed information about the operation
  - Interpretation of operand varies depending on op-code

Figure 2.5 The composition of an instruction for the machine in Appendix C

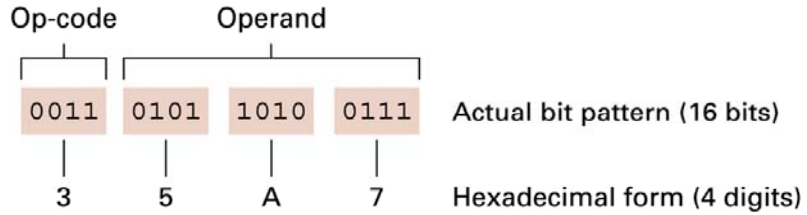


Figure 2.6 Decoding the instruction 35A7

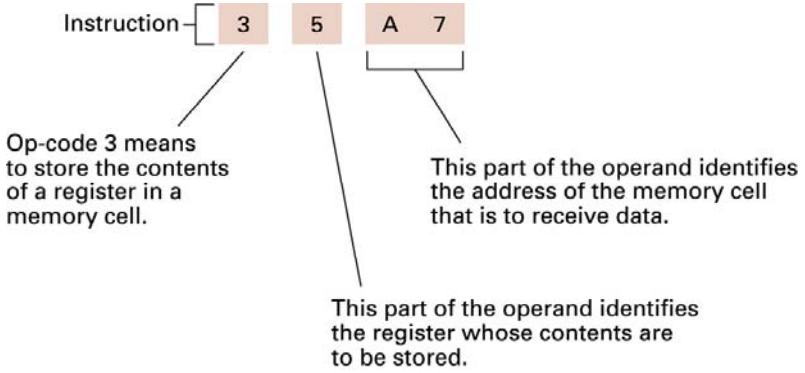


Figure 2.7 An encoded version of the instructions in Figure 2.2

Encoded instructions	Translation
156C	Load register 5 with the bit pattern found in the memory cell at address 6C.
166D	Load register 6 with the bit pattern found in the memory cell at address 6D.
5056	Add the contents of register 5 and 6 as though they were two's complement representation and leave the result in register 0.
306E	Store the contents of register 0 in the memory cell at address 6E.
C000	Halt.

Program Execution

- Controlled by two special-purpose registers
  - Program counter: address of next instruction
  - Instruction register: current instruction
- Machine Cycle
  - Fetch
  - Decode
  - Execute

Figure 2.8 The machine cycle

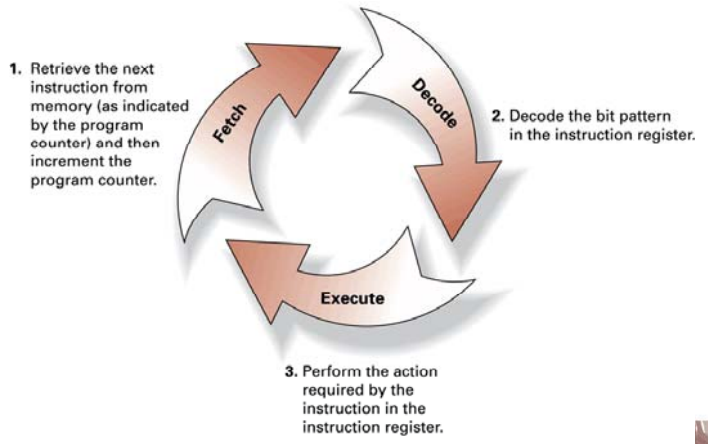


Figure 2.9 Decoding the instruction B258

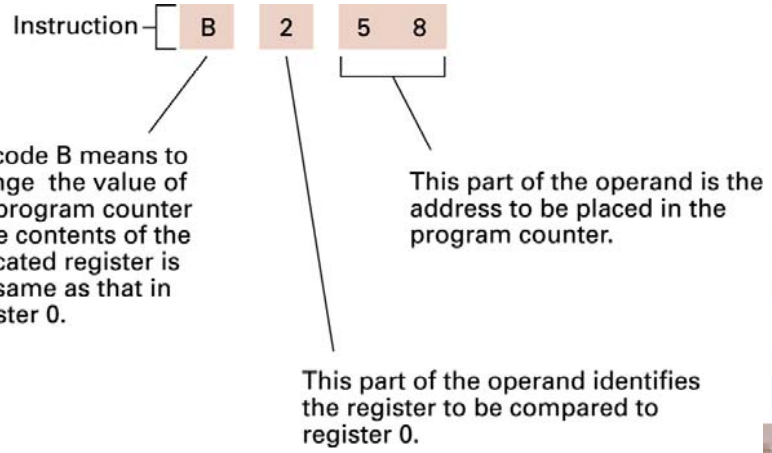


Figure 2.10 The program from Figure 2.7 stored in main memory ready for execution

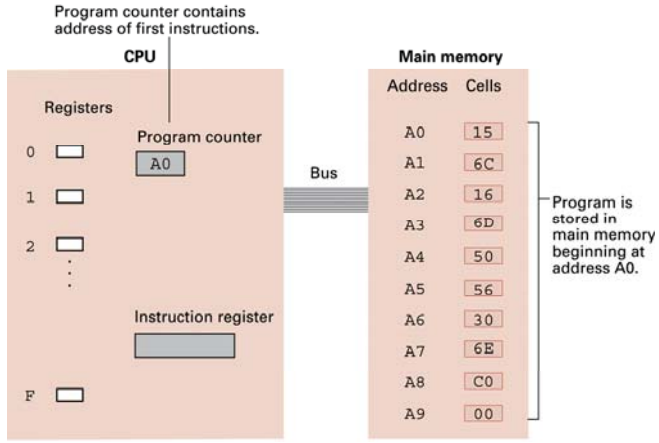
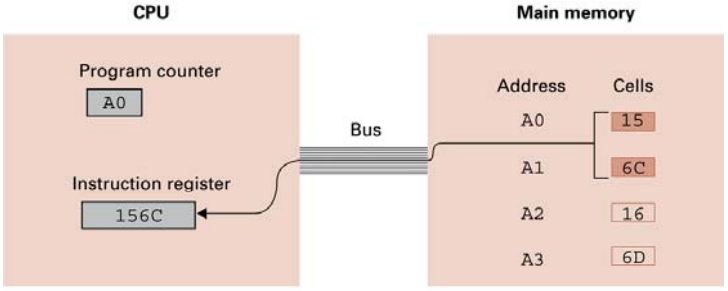
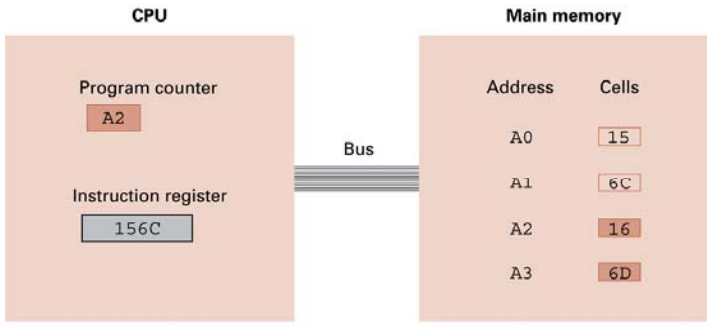


Figure 2.11 Performing the fetch step of the machine cycle



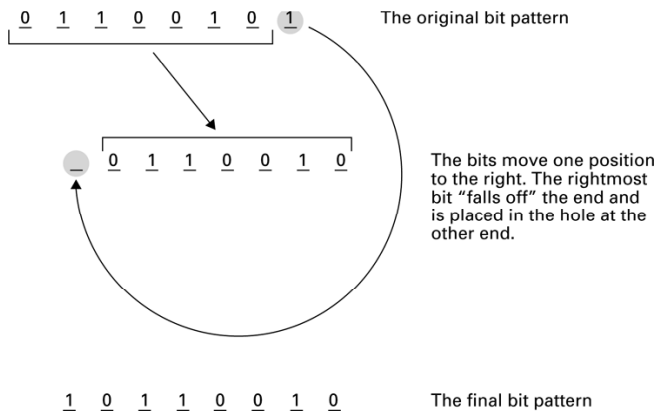
a. At the beginning of the fetch step the instruction starting at address A0 is retrieved from memory and placed in the instruction register.

Figure 2.11 Performing the fetch step of the machine cycle (cont'd)



b. Then the program counter is incremented so that it points to the next instruction.

Figure 2.12 Rotating the bit pattern 65 (hex) one bit to the right



Arithmetic/Logic Operations

- Logic: AND, OR, XOR
  - Masking
- Rotate and Shift: circular shift, logical shift, arithmetic shift
- Arithmetic: add, subtract, multiply, divide
  - Precise action depends on how the values are encoded (two's complement versus floating-point).

Communicating with Other Devices

- **Controller:** An intermediary apparatus that handles communication between the computer and a device
  - Specialized controllers for each type of device
  - General purpose controllers (USB and FireWire)
- **Port:** The point at which a device connects to a computer
- **Memory-mapped I/O:** CPU communicates with peripheral devices as though they were memory cells

Figure 2.13 Controllers attached to a machine's bus

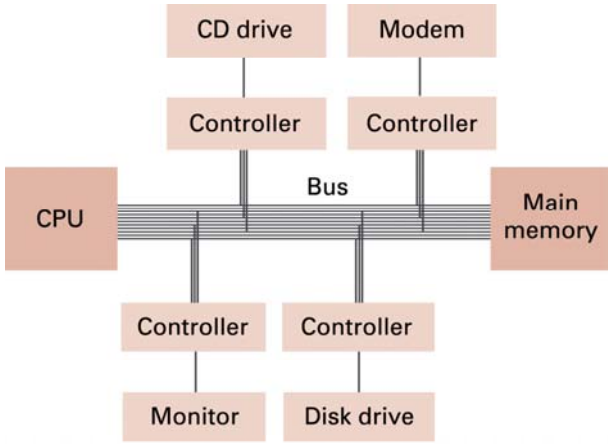
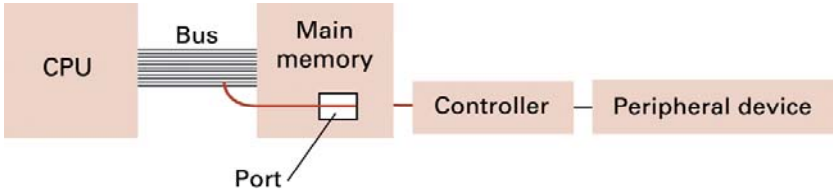


Figure 2.14 A conceptual representation of memory-mapped I/O

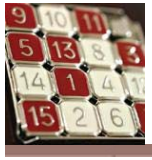


Communicating with Other Devices (continued)

- **Direct memory access (DMA):** Main memory access by a controller over the bus
- **Von Neumann Bottleneck:** Insufficient bus speed impedes performance
- **Handshaking:** The process of coordinating the transfer of data between components

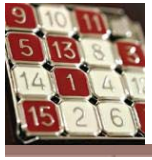
Communicating with Other Devices (continued)

- **Parallel Communication:** Several communication paths transfer bits simultaneously.
- **Serial Communication:** Bits are transferred one after the other over a single communication path.



Data Communication Rates

- Measurement units
  - Bps: Bits per second
  - Kbps: Kilo-bps (1,000 bps)
  - Mbps: Mega-bps (1,000,000 bps)
  - Gbps: Giga-bps (1,000,000,000 bps)
- Bandwidth: Maximum available rate



Other Architectures

- Technologies to increase throughput:
  - Pipelining(管線): Overlap steps of the machine cycle
  - Parallel Processing: Use multiple processors simultaneously
    - SISD: No parallel processing
    - MIMD: Different programs, different data
    - SIMD: Same program, different data