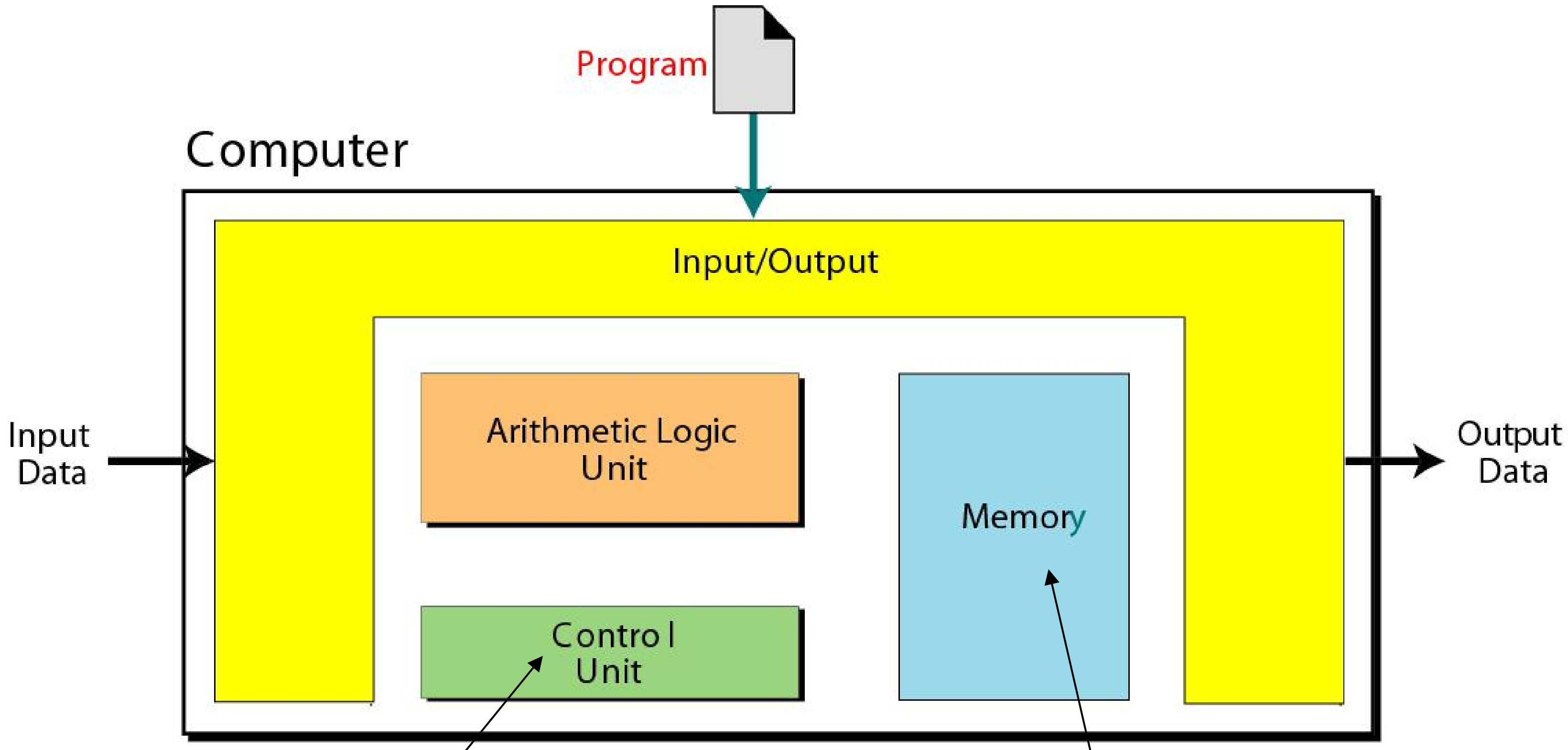
電腦的演進歷史

- 自從真空管電腦出現後，電腦的組織元件不斷推陳出新，歷經真空管、電晶體、積體電路、超大型機體電路等四個階段，分別被稱之為第一代、第二代、第三代、第四代電腦。其特色如下表格：

	第一代	第二代	第三代	第四代
元件	真 空 管	電 晶 體	積體電路	超大型積體電路
時期	1942~1958	1959~1963	1964~1970	1971年~現今
內部作業速度	10^{-3} 秒	10^{-6}秒	10^{-9} 秒	10^{-9}~$^{-13}$ 秒
輸入裝置	打孔卡紙 紙條裝置	打孔卡紙	按鍵 磁碟	鍵盤輸入 指標裝置 光學掃描
輸出裝置	打孔卡紙 列印輸出	打孔卡紙 列印輸出	列印輸出 螢幕影像	螢幕影像 語音輸出 列印輸出
主記憶體材質	磁鼓	磁蕊	磁蕊	半導體晶片
輔助記憶體材質	磁鼓 磁帶	磁帶 磁碟	磁碟 磁帶	磁碟 光碟 磁帶

Von Neumann model

The model defines a computer as four subsystems: memory, arithmetic logic unit, control unit, and I/O



Sequential execution of instructions

The data and program are stored as binary patterns in memory

Binary to decimal conversion

0	1	0	1	1	0	1	binary number
64	32	16	8	4	2	1	position values

0 + 32 + 0 + 8 + 4 + 0 + 1 results

45 decimal number

Example 1

Add two numbers in two's complement representation: (+17) + (+22) ➔ (+39)

Solution

Carry			1							
	0	0	0	1	0	0	0	1	+	
	0	0	0	1	0	1	1	0		
	--									
Result	0	0	1	0	0	1	1	1	➔	39

Example 2

Add two numbers in two's complement representation: (+24) + (-17) → (+7)

Solution

Carry	1	1	1	1	1					
		0	0	0	1	1	0	0	0	+
		1	1	1	0	1	1	1	1	
Result		0	0	0	0	0	1	1	1	→ +7

A simple Fortran program.

```
PROGRAM my_first_program

INTEGER :: i, j, k              ! All variables are integers

! Get the variables to multiply together.
WRITE (*,*)  'Enter the numbers to multiply:
READ (*,*) i, j
! Multiply the numbers together
k = i* j

! Write out the result.
WRITE (*,*) ‘Result = ' , k

STOP
END PROGRAM
```

FIGURE 2–2
Creating an executable Fortran program involves two steps, compiling and linking.

The term list-directed output means that the types of the values in the output list of the write statement determine the format of the output data. For example, consider the following statements:

```
PROGRAM output_example
INTEGER :: ix = 1
LOGICAL :: test = .TRUE.
REAL :: theta = 3.141593
ix = 1
test = .TRUE.
theta = 3.141593
WRITE (*,*) ' IX =              ', ix
WRITE (*,*) ' THETA =           ', theta
WRITE (*,*) ' COS(THETA) =      ', COS(theta)
WRITE (*,*) ' TEST =            ', test
WRITE (*,*) REAL(ix), NINT(theta)
END PROGRAM
```

3.3.1 The Block `IF` Construct

The commonest form of the `IF` statement is the block `IF` construct. This construct specifies that a block of code will be executed if and only if a certain logical expression is true. The block `IF` construct has the form

```
IF (logical_expr) THEN
   Statement 1      }
   Statement 2      } Block 1
   ...              }
END IF
```

If the logical expression is true, the program executes the statements in the block between the `IF` and `END IF` statements. If the logical expression is false, then the program skips all of the statements in the block between the `IF` and `END IF` statements and executes the next statement after the `END IF`. The flowchart for a block `IF` construct is shown in Figure 3–5.

3.3.1 The Block IF Construct

The commonest form of the IF statement is the block IF construct. This construct specifies that a block of code will be executed if and only if a certain logical expression is true. The block IF construct has the form

```
IF (logical_expr) THEN
   Statement 1     }
   Statement 2     } Block 1
   ...             }
END IF
```

If the logical expression is true, the program executes the statements in the block between the IF and END IF statements. If the logical expression is false, then the program skips all of the statements in the block between the IF and END IF statements and executes the next statement after the END IF. The flowchart for a block IF construct is shown in Figure 3–5.

3.3.2 The ELSE and ELSE IF Clauses

```
IF (logical_expr_1) THEN
   Statement 1
   Statement 2               Block 1
   ...

ELSE IF (logical_expr_2) THEN
   Statement 1
   Statement 2               Block 2
   ...
ELSE
   Statement 1
   Statement 2               Block 3
   ...
END IF
```

DO LOOP

```
PROGRAM FACTORIAL
IMPLICIT NONE
INTEGER::FACT,I,N
N=5
FACT=1
DO I=1,N
      FACT=FACT*I
END DO
WRITE(*,*) N,'!=',FACT
END PROGRAM
```

6.8.2 Summary of Fortran Statements and Structures

CALL Statement

```
CALL subname( arg1, arg2, ... )
```

Example:

```
CALL sort ( number, data1 )
```

Description:

This statement transfers execution from the current program unit to the subroutine, passing pointers to the calling arguments. The subroutine executes until either a `RETURN` or an `END SUBROUTINE` statement is encountered, and then execution will continue in the calling program unit at the next executable statement following the `CALL` statement.

CONTAINS Statement

```
CONTAINS
```

Examples:

```
MODULE test
...
CONTAINS
   SUBROUTINE sub1(x, y)
   ...
   END SUBROUTINE sub1
END MODULE
```

Description:

The `CONTAINS` statement specifies that the following statements are separate procedure(s) within a module. The `CONTAINS` statement and the module procedures following it must appear after any type and data definitions within the module, and before the `END MODULE` statement.