COMSM1302 電腦架構概論
第三週完整教材：序向邏輯與記憶

R-S 閂鎖 · D 閂鎖與 D 正反器 · 時脈 · 暫存器 · RAM

依據 John Lapinskas（University of Bristol）課程講義編寫並延伸

2026 年 7 月

本教材的使用方式：本講義整合了第三週四份投影片（3-1 Sequential logic and the R-S latch、3-2 From latches to flip-flops、3-3 From flip-flops to registers、3-4 From registers to RAM）的全部內容，並補充了亞穩態、setup ／ hold 時間、SRAM 6T 電路、記憶體階層等延伸知識。每章結尾附有「本章重點」整理；第 5 章為綜合練習題，附完整詳解。本週的主軸又是一條完整的故事線：**讓電路「記住」一個位元（閂鎖）→ 馴服它的時序（正反器與時脈）→ 包裝成可控的儲存單元（暫存器）→ 大量複製並編上地址（RAM）**。這是從邏輯閘走向真正電腦的關鍵一週。

目錄

1　序向邏輯與 R-S 閂鎖（Sequential Logic and the R-S Latch）

1.1　什麼是序向邏輯?

到目前為止你看過的所有電路，輸出**只取決於當下輸入的組合**——不管發生過什麼事，$1 \wedge 1$ 永遠是 1。這稱為**組合邏輯（combinational / combinatorial logic）**。它依然非常有用——第二週做的 ALU，正是之後打造 CPU 時要用的那一顆。

定義

組合邏輯（combinational logic）：輸出只依賴當前輸入。無記憶、無狀態——同樣的輸入永遠給出同樣的輸出。
序向邏輯（sequential logic）：輸出依賴**過去的輸入**，而不只是當前輸入。電路擁有**內部狀態（internal state）**。

但電腦不是純組合邏輯！電腦會**維護內部狀態**：變數的值、程式執行到哪一行、螢幕上的畫面……要打造電腦，我們必須更進一步——讓電路能「記住」東西。

1.2　R-S 閂鎖的行為

第一個能記住東西的電路是 **R-S 閂鎖（R-S latch）**。它沒有（正式的）真值表——因為輸出不完全由輸入決定：

R'	S'	Q	Q'
0	0	X	X
0	1	0	1
1	0	1	0
1	1	「保持（Hold）」	

- 前三列是普通的邏輯行為；但當 $R' = S' = 1$ 時，輸出**維持先前的值**——這就是「記憶」!
- $R' = S' = 0$ 的情況**我們不在乎**：這不是這個電路的預期用法，如果發生了就代表出了問題。真值表裡的標準寫法是 **X**（don't care）。
- 排除 X 的情況下，永遠有 $Q' = \neg Q$。我們把 Q 視為主要輸出。

R-S 閂鎖的使用方式

R' 是**重置（reset）**輸入、S' 是**設定（set）**輸入。兩者都是**由 1 變 0 時觸發**，不是由 0 變 1!

- 觸發 R'（拉低）：Q 設為 0，並一**直保持 0**，直到下次觸發 S';
- 觸發 S'（拉低）：Q 設為 1，並一**直保持 1**，直到下次觸發 R'。

這個行為極其重要——我們將用它來打造 RAM。

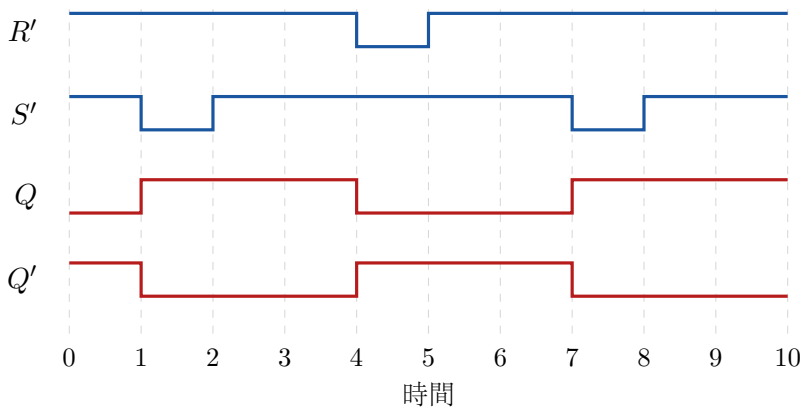
1.3　撇號是什麼意思？主動低電位與主動高電位

R'、S'、Q' 上的撇號（′）或橫槓（$\bar{R}$）正式上**沒有任何意義**——它們只是**慣例**，常用於資料手冊（datasheet），幫助使用者快速理解不熟悉的電路：

- 撇號在**輸出**上（如 Q' 或 $\bar{Q}$）：表示它是**反相**輸出——$Q' = \neg Q$，其中 Q 才是「真正的」輸出。
- 撇號在**輸入**上（如 R' 或 $\bar{R}$）：表示它**由 1 變 0 時觸發**。這種輸入稱為**主動低電位（active low）**。
- 由 0 變 1 觸發的輸入則稱為**主動高電位（active high）**。

1.4　時序圖（Timing Diagrams）

序向邏輯不能只靠真值表——輸出隨**時間**變化，所以我們改用**時序圖**：橫軸是時間，每條水平軌跡代表一個訊號的高低。



逐段解讀（Q 初始為 0）：

- $t = 1$：S' 由 1 掉到 0（set 觸發）⇒ Q 變 1；$t = 2$ 時 S' 回到 1，進入「保持」，Q **留在 1**。
- $t = 4$：R' 掉到 0（reset 觸發）⇒ Q 變 0；$t = 5$ 後保持 0。
- $t = 7$：S' 再度觸發 ⇒ Q 回到 1。
- 全程 $Q' = \neg Q$。

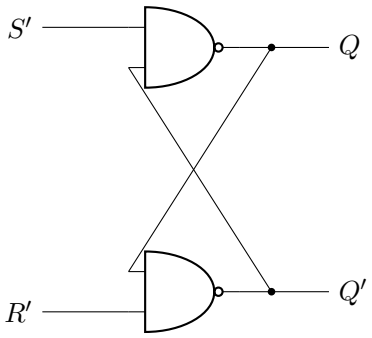
1.5　傳播延遲（Propagation Delay）

上面的時序圖包含一個**微妙的謊言**——訊號在 1 與 0 之間 **不會瞬間切換**！就算 R'、S' 真的瞬間切換，Q 與 Q' 也不會——電流通過閂鎖需要時間，這段時間稱為**傳播延遲（propagation delay）**。

傳播延遲通常以**奈秒**（ns，10^{-9} 秒）或**皮秒**（ps，10^{-12} 秒）計。本課程大多忽略它，但有時它很重要——例如它是 **CPU 時脈速度的限制條件**（第 2 章會看到原因）。

1.6　拆開神祕盒子：兩個 NAND 閘

R-S 閂鎖的內部只有——**兩個 NAND 閘**！秘訣在於**交叉耦合（cross-coupled）**：每個閘的輸出接到另一個閘的輸入。



寫成布林式：$Q = \neg(S' \wedge Q')$、$Q' = \neg(R' \wedge Q)$——**輸出出現在自己的定義式裡**，這正是回饋（feedback）。為什麼這樣就能記憶？逐步追蹤兩種情況：

追蹤 1：set 觸發（S' 低、R' 高）

1. S' 低 $\Rightarrow$ 上方 NAND **不管 Q' 是什麼**都必定輸出高（NAND 只要有一輸入為 0 就輸出 1）$\Rightarrow Q = 1$。
2. R' 高、且回饋來的 $Q = 1$ $\Rightarrow$ 下方 NAND 兩輸入皆高 $\Rightarrow Q' = 0$。**狀態穩定**。
3. 關鍵：之後 S' 回到高，上方 NAND 的輸入變成 $S' = 1$ 與 $Q' = 0$——因為 $Q' = 0$，輸出**仍然**是 1。**輸出保持不變**：記憶完成！

追蹤 2：reset 觸發（R' 低、S' 高）

完全對稱：

1. R' 低 $\Rightarrow$ 下方 NAND 不管 Q 是什麼都輸出高 $\Rightarrow Q' = 1$。
2. S' 高、回饋 $Q' = 1$ $\Rightarrow$ 上方 NAND 兩輸入皆高 $\Rightarrow Q = 0$。穩定。
3. R' 回到高後，下方 NAND 因 $Q = 0$ 仍輸出 1——狀態保持。

延伸補充：為什麼 $R' = S' = 0$ 是禁區？ 兩輸入都拉低時，兩個 NAND 都被迫輸出 1，於是 $Q = Q' = 1$——違反 $Q' = \neg Q$ 的約定（這就是表中的 X）。更糟的是**同時放開**兩個輸入的瞬間：兩個閘會互相搶著更新，結果取決於兩個閘傳播延遲的微小差異（競態，race condition），甚至可能短暫停在高低之間的**亞穩態（metastability）**——電壓懸在中間、經過不可預測的時間才隨機倒向一邊。實務設計必須保證這種輸入組合永遠不會發生——下一章的 D 閂鎖正是從結構上根除這個問題。

本章重點

- 組合邏輯無狀態；序向邏輯的輸出依賴**過去**的輸入（內部狀態）。

- R-S 閂鎖：S' 拉低 $\Rightarrow Q = 1$；R' 拉低 $\Rightarrow Q = 0$；兩者皆高 $\Rightarrow$ 保持。$R' = S' = 0$ 是 don't care（X）。
- 撇號慣例：輸出上 = 反相；輸入上 = 主動低電位（active low）。
- 序向邏輯用**時序圖**描述行為；實際訊號切換需要**傳播延遲**（ns ／ ps 級），是 CPU 時脈速度的限制。
- 內部構造：兩個交叉耦合的 NAND；回饋讓「輸出決定輸出」，因此能記憶。

2　從閂鎖到正反器（From Latches to Flip-Flops）

2.1　R-S 閂鎖的三個問題

R-S 閂鎖很好，但有幾個麻煩：

1. 必須小心**不能同時觸發** set 與 reset（$R' = S' = 0$ 禁區）；
2. 如果能把「**下一個狀態是什麼**」與「**何時改變**」分開控制，時序問題會好想得多；
3. 如果把輸出**迴接回輸入**，事情會變得一團糟！

前兩個問題用 **D 閂鎖（D latch）**解決；第三個問題稍後用 **D 正反器（D flip-flop）**解決。

2.2　D 閂鎖：想要的行為

en	D	Q	Q'
0	0	「保持」	「保持」
0	1	「保持」	「保持」
1	0	0	1
1	1	1	0

en（enable）是**主動高電位**輸入。en 高時，Q **跟隨** D 的值；en 低時，Q 保持不變。維持 $Q' = \neg Q$。
——「存什麼」（D）與「何時存」（en）分開了！

2.3　用 R-S 閂鎖打造 D 閂鎖

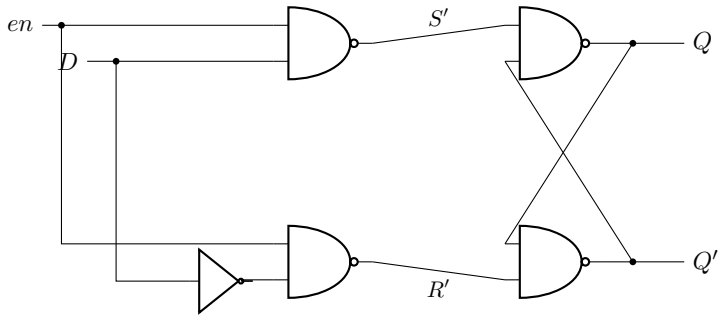
回到基本功：用**組合邏輯**，讓 en 與 D 產生 R-S 閂鎖所需的輸入！先把想要的行為翻譯成 R-S 閂鎖的輸入：

en	D	R'	S'	Q	Q'
0	0	1	1	「保持」	「保持」
0	1	1	1	「保持」	「保持」
1	0	0	1	0	1
1	1	1	0	1	0

讀表可得（各欄何時為 0）：

$$\boxed{S' = \neg(en \wedge D), \qquad R' = \neg(en \wedge \neg D)}$$

兩條都是 NAND！所以 D 閂鎖 = **4 個 NAND + 1 個 NOT**：



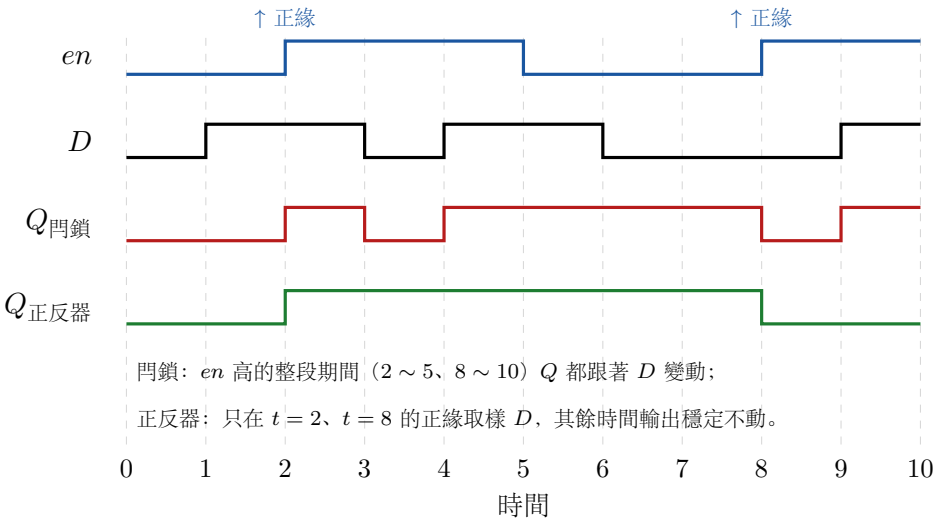
注意 $en = 1$ 時 R'、S' 恰為 D 與 $\neg D$ 的 NAND——**兩者永遠不會同時為 0**（結構上互補），禁區問題根除 ✓。

2.4　位準觸發與邊緣觸發

定義

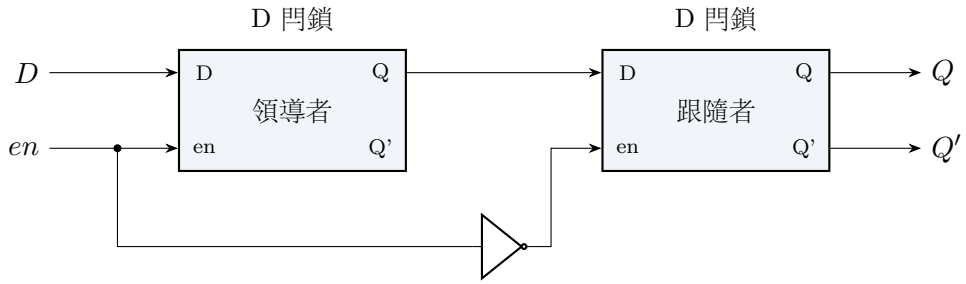
D 閂鎖是**位準觸發（level-triggered）**：只要 en **持續為高**，D 的任何變化都會反映到輸出。**邊緣觸發（edge-triggered）**：Q 只在 en **變高的那一瞬間**取樣 D 的值。由低變高觸發稱為**正緣觸發（positive edge triggering）**；由高變低觸發稱為**負緣觸發（negative edge triggering）**。

位準觸發也能用，但必須非常小心控制 en 維持高電位的時間長度；邊緣觸發用起來容易得多。下圖對比同一組輸入下，位準觸發的 D 閂鎖與正緣觸發的 D 正反器的輸出差異：



2.5　D 正反器的內部構造：領導者與跟隨者

負緣觸發比較容易做，先從它開始。把**兩個 D 閂鎖串接**，並讓第二個的 en 經過一個 NOT 閘：



左邊的閂鎖稱為**領導者 (leader)** 或 primary，右邊的稱為**跟隨者 (follower)** 或 secondary。（歷史上分別叫 master 與 slave，但這些詞帶有不好的聯想，正逐漸被淘汰。）

逐步追蹤（假設 en 一開始為高）：

1. en **高**：領導者輸出跟隨 D；跟隨者的 en 經過 NOT 為低，**被停用**（輸出不受影響）。
2. en **落下（負緣）**：跟隨者的 en 變高，它立刻取用領導者當下的輸出——即 D 在**落緣瞬間**的值。之後 D 再怎麼變都沒用，因為領導者的 en 已是低。
3. en **再升起**：**關鍵一步**——即使 D 已改變，新的 en 訊號穿過 NOT 閘的速度**比穿過整個領導者快**。所以跟隨者的 en 先變低（鎖住），領導者的輸出才更新完成——跟隨者的輸出不變。
4. 之後 D 的變化同樣無效，直到下一個負緣。

整體效果：**輸出只在 en 的負緣更新一次**——負緣觸發完成！

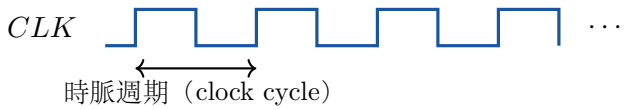
那要怎麼改成正緣觸發？——**在 en 前面再加一個 NOT 閘**！這就是**（正緣觸發的）D 正反器 (D flip-flop)**。

閂鎖 vs. 正反器

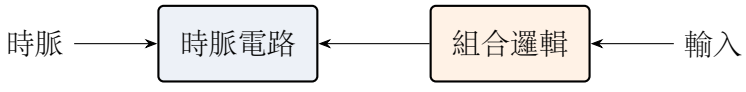
閂鎖（latch）是位準觸發，正反器（flip-flop）是邊緣觸發。這是兩個名詞的定義性差異。

2.6　時脈（The Clock）

現在可以一舉解決所有時序問題了！我們用單一**方波**驅動所有電路：**時脈（clock）**，常記作 CLK。



然後用 D 正反器（及其變體）作為**時序問題的緩衝**：



時脈設計的核心論證

每個**上升緣**，正反器更新一次。接下來變化在組合邏輯中傳播的期間，正反器的**輸出保持恆定**——即使其輸入一直在變。所以只要

$$\text{相鄰上升緣的間隔} \;\geq\; \text{組合邏輯的最大傳播延遲}$$

我們就可以**完全忽略傳播延遲，純粹用邏輯思考**！這正是「同步數位設計」的基本契約，也解釋了為什麼傳播延遲限制了 CPU 的時脈上限。

2.7　時脈術語

- **時脈週期（clock cycle）**：相鄰兩個上升緣之間的時間區間。
- **週期時間（time period）**：一個時脈週期所花的時間長度。
- **頻率（frequency）**：每秒的時脈週期數，單位**赫茲（Hz）**。1 Hz = 每秒 1 個週期，且

$$\text{頻率（Hz）} = \frac{1}{\text{週期時間（秒）}}$$

- 例：頻率 20 kHz 的時脈，週期時間為 1/20,000 秒，即 50 μs。

時脈訊號通常由專用電路中的**壓電石英晶體（piezoelectric crystal）**產生——用的是我們的宿敵：物理。

2.8　SI 詞頭速查

10^n	符號	名稱
10^{15}	P	Peta-
10^{12}	T	Tera-
10^{9}	G	Giga-
10^{6}	M	Mega-
10^{3}	k	Kilo-
10^{0}	—	—
10^{-3}	m	Milli-
10^{-6}	μ	Micro-
10^{-9}	n	Nano-
10^{-12}	p	Pico-

例：1 GHz = 10^9 = 1,000,000,000 Hz。（位元組 byte 有一個惱人的例外，第 4 章詳談。）

延伸補充：setup ／ hold 時間與亞穩態。真實的正反器對輸入有兩個時序要求：D 必須在時脈緣**之前**穩定至少 t_{su}（setup time）、**之後**保持穩定至少 t_h（hold time）。若在這個窗口內改變 D（例如取樣一個與時脈無關的非同步訊號），正反器可能進入**亞穩態**：輸出電壓懸在高低之間，

經過統計分布、無上界的時間後才隨機倒向 0 或 1。工程解法是**同步器（synchroniser）**：串接兩級正反器，給亞穩態一整個時脈週期的時間去消解，把出錯機率壓到天文數字分之一。

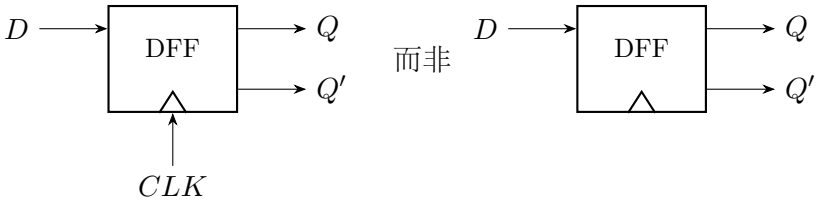
本章重點

- D 閂鎖 = R-S 閂鎖 + 兩個 NAND 前端：$S' = \neg(en \wedge D)$、$R' = \neg(en \wedge \neg D)$；結構上根除禁區，並分離「存什麼／何時存」。
- 閂鎖是位準觸發；正反器是邊緣觸發（正緣／負緣）。
- D 正反器 = 領導者閂鎖 + 跟隨者閂鎖 + NOT（負緣觸發）；再加一個 NOT 變正緣觸發。核心巧思：NOT 比整個閂鎖快。
- 時脈（方波）+ 正反器緩衝 ⇒ 只要週期 ≥ 最大傳播延遲，就能忽略時序、純用邏輯設計。
- 頻率 = 1/ 週期時間；kHz ／ MHz ／ GHz 等 SI 詞頭。

3　從正反器到暫存器（From Flip-Flops to Registers）

3.1　時脈電路的記號

所有電路共用同一個時脈，所以畫圖時常**省略時脈線**：邊緣觸發的元件以一個**三角形**記號標示時脈輸入。例如 D 正反器（DFF）通常這樣畫：

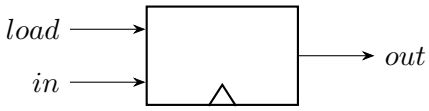


（右邊省略了時脈線，只留三角形——因為大家都接同一個 CLK。）

3.2　1 位元暫存器

定義

（1 位元）暫存器（register）是 D 正反器的簡單變體：在**時脈上升緣**，若 *load* 為高，*out* 取用 *in* 的值；若 *load* 為低，*out* **保持不變**。



跟 DFF 的差別：DFF **每個**上升緣都會更新；暫存器多了一個 *load* 開關，可以選擇「這一拍要不要存」。

3.3　一個會失敗的設計

直覺想法：把 *load* 和 CLK 做 AND，接到 DFF 的時脈輸入——「*load* 高的時候才讓時脈通過」。

這個設計會失敗。原因：這樣一來，*out* 是在

$$load \wedge CLK$$

的上升緣取值，而**不是** CLK 的上升緣。例如 CLK 為高的期間 *load* 若震盪，$load \wedge CLK$ 就會產生一堆假的上升緣，*out* 隨之不穩定。

設計守則

使用正反器時，**不要對時脈訊號動手腳（don't gate the clock）**——太容易引入這類 bug。要控制「存不存」，應該去改**資料路徑**，而不是時脈路徑。

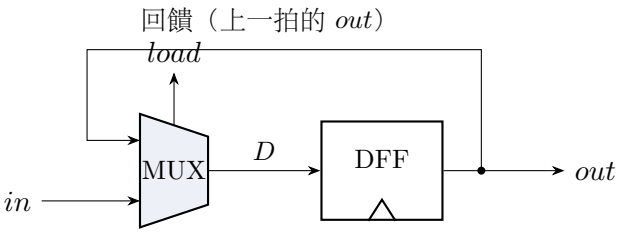
3.4　正確的做法：回饋 + 多工器

正確思路：**用這一個時脈週期的輸出，經組合邏輯決定下一個週期的輸入**。我們想要的行為（X 表示 don't care）：

in	$load$	out_{old}	out_{new}
X	0	0	0
X	0	1	1
0	1	X	0
1	1	X	1

可以展開成完整真值表再用 K-map……或者直接看出：**這就是一個多工器（multiplexer）**！$load$ 是選擇位：$load = 0$ 選舊值（保持）、$load = 1$ 選 in（載入）：

$$out_{\text{new}} = (\neg load \wedge out_{\text{old}}) \vee (load \wedge in)$$



運作：每個時脈上升緣 DFF 都照常更新（時脈路徑乾乾淨淨），但 $load = 0$ 時它存回的是**自己原本的值**——效果上就是「保持」。✓

3.5　多位元暫存器與匯流排

Hack CPU 用的不是 1 位元而是 **16 位元暫存器**——視為各存放一個 16 位元二進位值。與其畫 16 條線，我們畫**一條帶斜線記號的線**，同樣視為承載一個二進位值。這種「二進位線」稱為**匯流排（bus）**。（Logisim 的記號略有不同。）

16 位元暫存器的實作很簡單：16 個 1 位元暫存器並排，共用同一個 $load$ 與 CLK，第 i 個管 $in_i \to out_i$。

本章重點

- 三角形記號 = 邊緣觸發的時脈輸入；時脈線常省略不畫。
- 暫存器 = DFF + $load$ 控制：上升緣且 $load$ 高才載入，否則保持。
- **不要 gate 時脈**：$load \wedge CLK$ 會製造假上升緣。
- 正確做法：MUX 選擇「舊值回饋」或「新值 in」餵回 DFF——用組合邏輯改資料路徑，讓時脈路徑保持純淨。

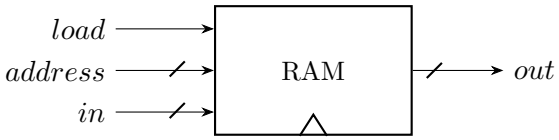
- 匯流排（bus）：一條斜線記號的多位元線；16 位元暫存器 = 16 個並排的 1 位元暫存器共用 *load*。

4　從暫存器到 RAM（From Registers to RAM）

4.1　記憶體是什麼?

定義

記憶體（memory）的行為就像**一組暫存器**。每個暫存器有一個**地址（address）**，儲存一個固定長度的二進位**字（word）**（Hack 為 16 位元）。



介面行為:

- **讀取**：*out* 永遠等於地址為 *address* 的暫存器所存的值——**立即生效，不等時脈**（讀取是組合邏輯）。
- **寫入**：若 *load* 為高，則在時脈上升緣，地址為 *address* 的暫存器被設為 *in*；*load* 低則不變。

關鍵術語:

- 每個暫存器的大小稱為**字長（word size）**;
- 有效地址的集合稱為**地址空間（address space）**;
- （大暫存器內部的）每個 1 位元暫存器稱為一個**記憶胞（cell）**。

怎麼打造記憶體? 你已經具備全部零件了——用 DEMUX 把 *load* 導向被選中的暫存器、用 MUX 從所有暫存器的輸出中選出 *out*（見課程作業!）。

4.2　RAM 與 ROM

用正反器蓋出來的記憶體是**揮發性（volatile）**的：只有持續供電才能保住資料。揮發性記憶體也稱為 **RAM（Random Access Memory，隨機存取記憶體）**——這名字出於**已不再適用**的歷史原因。

非揮發性（non-volatile）儲存則相反：斷電後資料仍在。

- **ROM（Read-Only Memory，唯讀記憶體）**：只寫入一次（物理性地改變晶片）的非揮發性記憶體。優點是成本低、存取快——適合放不可升級的韌體（firmware）。
- Hack CPU 使用 16KB RAM 與 32KB ROM（為了實作簡單）。
- 現代多數非揮發性記憶體是 **EEPROM**（Electrically Erasable Programmable ROM，電子可抹除可程式化 ROM），包括**快閃記憶體（flash memory）**——可以（以 CPU 的時間尺度來說**很慢地**）抹除並重寫。

所以：隨身碟與 SSD 用的是 flash，屬於 EEPROM，但它們**絕對不是**唯讀的；而且像所有現代記憶體一樣，它們都是**隨機存取**的。這套術語真是「非常合理」呢。（——講義原文的吐槽。教訓：這些名字是歷史包袱，別按字面理解。）

4.3　SRAM 與 DRAM

我們蓋出來的 RAM 類似 **SRAM（Static RAM，靜態隨機存取記憶體）**：

- SRAM 用高度精簡的設計，每位元只需 **4–6 個電晶體**（我們的做法約需 40 個）；原理相似——只要供電就保住資料。
- **DRAM（Dynamic RAM，動態隨機存取記憶體）**運作方式完全不同：資料的儲存方式需要**每隔幾毫秒**以及**每次讀取後**手動更新（refresh）。與 SRAM 相比：更便宜、容量更大，但**慢得多**。
- 現代電腦**兩者都用**——本課程稍後（記憶體階層）詳談。
- **SDRAM（Synchronised DRAM）**是由時脈驅動的 DRAM。現代 DRAM 全是 SDRAM；它與 SRAM **毫無關係**（雖然名字很像）。
- **DDR（Double Data Rate）1–5** 是一族與 DRAM 之間的 **資料傳輸協定**，不是根本不同的技術。

延伸補充：SRAM 6T 與 DRAM 1T1C。工業級 SRAM 記憶胞是「6 電晶體（6T）」設計：4 個電晶體組成兩個交叉耦合的反相器（跟我們的 NAND 閂鎖同一個原理——雙穩態回饋圈），另外 2 個是存取電晶體。DRAM 記憶胞只有「1 電晶體 + 1 電容（1T1C）」：位元存成電容上的電荷，電荷會漏，所以要定期 refresh；讀取還是**破壞性**的（電荷被讀掉，得寫回去）。這就是兩者特性差異的物理根源：

	SRAM	DRAM
每位元成本	6 電晶體	1 電晶體 + 1 電容
存取延遲	< 1 ns	∼10–60 ns
需要 refresh	否	是（每幾 ms + 每次讀取後）
密度／單價	低密度、昂貴	高密度、便宜
典型用途	CPU 快取（L1/L2/L3）	主記憶體（DDR5 等）

4.4　儲存容量的 SI 單位

記憶體大小「天生」就是 2 的冪次（地址空間 $= 2^{\text{地址位元數}}$），所以我們通常用這套**以 2 為底的替代單位**——例如 1KB 指 1024 位元組而非 1000：

10^n	符號	名稱	2^n	符號	名稱
10^{15}	P	Peta-	2^{50}	P 或 Pi	Peta- 或 pebi-
10^{12}	T	Tera-	2^{40}	T 或 Ti	Tera- 或 tebi-
10^{9}	G	Giga-	2^{30}	G 或 Gi	Giga- 或 gibi-
10^{6}	M	Mega-	2^{20}	M 或 Mi	Mega- 或 mebi-
10^{3}	k	Kilo-	2^{10}	K 或 Ki	Kilo- 或 kibi-

這樣一來，例如兩條 16GB 的 RAM 可以合併成 32GB——**只要在地址空間多加一個位元**。

陷阱：所有人都同意用 2 為底——除了**非揮發性儲存的製造商**。他們很樂意宣傳「1TB 硬碟」卻只提供 10^{12} 位元組（≈ 0.909 TiB，少了約 9%）。而且他們請得起律師，所以 P ／ T ／ G ／ M 詞頭在市場上常常是模糊的……**本課程中，位元組一律使用 2 為底的單位**。

4.5　附錄：對數複習

定義

$\log_b a$（「以 b 為底 a 的對數」）是使 $b^{\log_b a} = a$ 成立的數。例：$\log_2 16 = 4$、$\log_2 256 = 8$、$\log_2 1 = 0$。

對數對我們的用處：想知道**需要幾個位元**來表示無號數 x、或替一個 x 字的記憶體儲存地址，答案就是 $\log_2 x$ **向上取整**。

計算機通常只有 $\log_{10}$（log）或 $\log_e$（ln，$e \approx 2.718$）。換底公式：對任何底 b，

$$\log_2 x = \frac{\log_b x}{\log_b 2}, \qquad \text{因為} \quad 2^{(\log_b x)/(\log_b 2)} = \left(b^{\log_b 2}\right)^{(\log_b x)/(\log_b 2)} = b^{\log_b x} = x.$$

另外，$\log_b x$ 的成長**極慢**：若 $\log_2 n \geq 50$，則 $n \geq 1\text{PB}$！所以在演算法分析裡（Programming in C 會看到 $\log_2 n$ 時間的演算法），可以把 $\log_2 n$ 當成一個「偏大的常數」，與其他常數因子互相權衡。——這就是你（在這個學位裡）需要知道的所有對數知識！

本章重點

- 記憶體 = 一組帶地址的暫存器；讀取即時（組合邏輯）、寫入在上升緣（*load* 高時）。術語：字長、地址空間、記憶胞。
- 揮發性 = RAM（斷電即失）；非揮發性：ROM（寫一次）、EEPROM ／ flash（可慢速重寫）。名字都是歷史包袱。
- SRAM（快、貴、免 refresh，用於快取）vs. DRAM（慢、便宜、要 refresh，用於主記憶體）；SDRAM = 時脈驅動的 DRAM；DDR = 傳輸協定。
- 容量用 2 為底：1 KiB = 2^{10}、1 MiB = 2^{20}、1 GiB = 2^{30} 位元組；硬碟商愛用 10 為底灌水。
- x 個字的記憶體需要 $\lceil \log_2 x \rceil$ 個地址位元；換底公式。

5　綜合練習題（附詳解）

練習 1：R-S 閂鎖追蹤

一個 R-S 閂鎖初始 $Q = 0$，輸入依序發生：(i) S' 短暫拉低後回高；(ii) S' 再度短暫拉低；(iii) R' 短暫拉低後回高。寫出每步之後的 Q。

解答

(i) set 觸發 $\Rightarrow Q = 1$；S' 回高後進入保持，Q 仍為 1。
(ii) Q 已經是 1，再 set 一次**沒有變化**，$Q = 1$。（set 是「把 Q 變 1」，不是「翻轉」。）
(iii) reset 觸發 $\Rightarrow Q = 0$，之後保持 0。
重點：閂鎖記住的是**最後一次被觸發的是誰**。

練習 2：撇號慣例

資料手冊上有一顆晶片，腳位標著 EN'、OUT、OUT'。(a) EN' 如何觸發？(b) OUT' 與 OUT 的關係？(c) 這些撇號在邏輯上有什麼正式意義？

解答

(a) EN' 是**主動低電位**輸入：由 1 變 0 時觸發（啟用）。
(b) $OUT' = \neg OUT$：反相輸出，OUT 是「真正的」輸出。
(c) **沒有正式意義**——純粹是慣例記號，幫助使用者快速理解不熟悉的電路。

練習 3：NAND 閂鎖的穩定性驗證

R-S 閂鎖由 $Q = \neg(S' \wedge Q')$、$Q' = \neg(R' \wedge Q)$ 組成。驗證 $R' = S' = 1$ 時，$(Q, Q') = (1, 0)$ 與 $(0, 1)$ 都是穩定狀態，並說明為什麼 $R' = S' = 0$ 後同時放開會出問題。

解答

代入檢查（$R' = S' = 1$）：

- $(Q, Q') = (1, 0)$：$\neg(1 \wedge 0) = 1 = Q$ ✓、$\neg(1 \wedge 1) = 0 = Q'$ ✓——自洽，穩定。
- $(Q, Q') = (0, 1)$：$\neg(1 \wedge 1) = 0 = Q$ ✓、$\neg(1 \wedge 0) = 1 = Q'$ ✓——同樣穩定。

同一組輸入有**兩個**穩定狀態——這正是「1 位元記憶」的本質：落在哪個狀態取決於歷史。
$R' = S' = 0$ 時兩個閘都被迫輸出 1（$Q = Q' = 1$，違反互補約定）。**同時**放開（兩輸入同時回到 1）的瞬間，兩個閘都想更新成 $\neg(1 \wedge 1) = 0$，然後又逼對方變回 1……結果由兩個閘傳播延遲的微小差異決定（競態），甚至可能短暫懸在亞穩態後隨機落定——輸出**不可預測**，所以設計上必須避免。

練習 4：D 閂鎖公式推導

從 D 閂鎖的四列行為表出發，推導 S'、R' 的公式，並驗證 $en=1, D=1$ 的情況。

解答

觀察表格：$S'=0$ 只發生在 $(en, D)=(1,1) \Rightarrow S'=\neg(en \wedge D)$；$R'=0$ 只發生在 $(en, D)=(1,0) \Rightarrow R'=\neg(en \wedge \neg D)$。

驗證 $en=1, D=1$：$S'=\neg(1\wedge 1)=0$（set 觸發），$R'=\neg(1\wedge 0)=1$（不觸發）$\Rightarrow Q=1$ ✓——正是「en 高時 Q 跟隨 D」。且因為 S'、R' 分別由 D 與 $\neg D$ 控制，兩者**不可能同時為 0**：禁區從結構上被根除。

練習 5：位準觸發 vs. 邊緣觸發

en ／ CLK 在 $t\in[2,5]$ 為高，其餘為低。D 在 $t=3$ 由 1 變 0、在 $t=4$ 由 0 變 1。初始輸出為 0。分別寫出 (a) D 閂鎖、(b) 正緣觸發 D 正反器的輸出變化。

解答

(a) **D 閂鎖（位準觸發）**：$t=2$ 時 en 變高，Q 開始跟隨 D：$Q=1$（$D=1$）；$t=3$：$Q=0$；$t=4$：$Q=1$；$t=5$ 後 en 低，Q 凍結在 1。輸出跳了三次：$0\to1\to0\to1$。

(b) **D 正反器（正緣觸發）**：只在 $t=2$ 的上升緣取樣一次，$D=1\Rightarrow Q=1$，之後**全程不動**（$t=3,4$ 的變化無效，$t=5$ 是下降緣也無效）。

這就是邊緣觸發的價值：en 高的期間內輸入怎麼抖，輸出都穩定。

練習 6：D 正反器的內部機制

在領導者—跟隨者構造中：(a) 為什麼負緣觸發「比較容易」先做出來？(b) 「NOT 閘比領導者快」這件事為什麼是必要的？

解答

(a) 兩個閂鎖串接後，跟隨者的 en 接的是 $\neg en$：en 高時領導者跟隨 D、跟隨者鎖住；en **落下**時跟隨者才開門，接收領導者剛凍結的值——所以輸出自然在**負緣**更新。要正緣觸發，在最前面再加一個 NOT 即可。

(b) 考慮 en 上升的瞬間（負緣觸發版本）：此時領導者「開門」，若 D 已改變，領導者的輸出將開始更新。若跟隨者的 en（$=\neg en$）**還沒**變低，新值就會「漏」過跟隨者，破壞邊緣觸發語意。因為 NOT 只有一級閘延遲、領導者要好幾級，跟隨者必定**先**關門、領導者輸出**後**更新——時序天然安全。

練習 7：時脈換算

(a) 頻率 2.5 GHz 的 CPU，時脈週期是多少？(b) 週期 50 μs 的時脈，頻率是多少？(c) 某電路的最大傳播延遲是 0.8 ns，時脈頻率最高可設多少？

解答

(a) $T = 1/f = 1/(2.5 \times 10^9) = 4 \times 10^{-10}$ 秒 $= 0.4$ ns $= 400$ ps。

(b) $f = 1/T = 1/(50 \times 10^{-6}) = 20{,}000$ Hz $= 20$ kHz。

(c) 需要週期 $\geq$ 最大傳播延遲：$T \geq 0.8$ ns $\Rightarrow f \leq 1/(0.8 \times 10^{-9}) = 1.25 \times 10^9$ Hz $= 1.25$ GHz。這正是「傳播延遲限制時脈上限」的具體計算。

練習 8：為什麼不能 gate 時脈？

有人把暫存器設計成「$load \wedge CLK$ 接到 DFF 的時脈腳」。畫出以下情境並指出錯誤：CLK 在 $t \in [2, 4]$ 為高；$load$ 在 $t = 2.5$ 由 0 升高、$t = 3$ 落下、$t = 3.5$ 又升高。

解答

$load \wedge CLK$ 的波形：在 $[2.5, 3]$ 為高、$[3.5, 4]$ 又為高——產生了 $t = 2.5$ 與 $t = 3.5$ **兩個假上升緣**，DFF 在這兩個時刻各取樣一次，而**真正的** CLK 上升緣（$t = 2$）反而沒有取樣。輸出的更新時機由 $load$ 的抖動決定，完全失控。

正解：時脈直通 DFF；$load$ 改去控制**資料路徑**上的 MUX——$out_{\text{new}} = (\neg load \wedge out_{\text{old}}) \vee (load \wedge in)$。每個上升緣都正常取樣，但 $load = 0$ 時取樣的是自己的舊值，等效於保持。

練習 9：暫存器行為追蹤

一個 1 位元暫存器初始 $out = 0$，連續五個時脈上升緣時的輸入為：

上升緣	1	2	3	4	5
in	1	0	1	1	0
$load$	1	0	0	1	0

寫出每個上升緣之後的 out。

解答

- 緣 1：$load = 1 \Rightarrow$ 載入 $in = 1$，$out = 1$；
- 緣 2：$load = 0 \Rightarrow$ 保持，$out = 1$（$in = 0$ 無效）；
- 緣 3：$load = 0 \Rightarrow$ 保持，$out = 1$；
- 緣 4：$load = 1 \Rightarrow$ 載入 $in = 1$，$out = 1$（值恰好相同）；
- 緣 5：$load = 0 \Rightarrow$ 保持，$out = 1$。

最終 $out = 1$。序列：$1, 1, 1, 1, 1$。

練習 10：地址位元計算

(a) Hack 的 RAM16K 有 16,384 個 16 位元的字，需要幾個地址位元？總共有多少個記憶胞？(b) 一個記憶體有 100 個字，需要幾個地址位元？(c) 64 KiB 的記憶體、字長 16 位元，需要幾個地址位元？

解答

(a) $\log_2 16384 = 14$（$2^{14} = 16384$）⇒ **14 個地址位元**。記憶胞數 $= 16384 \times 16 = 262{,}144$ 個。
(b) $\log_2 100 \approx 6.64$，向上取整 ⇒ **7 個位元**（$2^6 = 64$ 不夠、$2^7 = 128$ 才夠；有 28 個地址沒用到）。
(c) 64 KiB $= 2^{16}$ 位元組 $= 2^{19}$ 位元；字長 16 位元 $= 2^4$ 位元 ⇒ 字數 $= 2^{19}/2^4 = 2^{15} = 32{,}768$ ⇒ **15 個地址位元**。

練習 11：儲存單位陷阱

(a) 32 GiB 是多少位元組（寫成 2 的冪次）？(b) 某廠商的「1 TB」硬碟實際是 10^{12} 位元組，換算成 TiB 是多少？少了幾 percent？

解答

(a) 32 GiB $= 2^5 \times 2^{30} = 2^{35}$ 位元組 $= 34{,}359{,}738{,}368$ 位元組。
(b) 1 TiB $= 2^{40} = 1{,}099{,}511{,}627{,}776$ 位元組。$10^{12}/2^{40} \approx 0.9095$ TiB——比 1 TiB **少了約 9%**。這就是為什麼作業系統顯示的容量總是比包裝盒上的小：廠商用 10^{12}、作業系統多用 2^{40}。本課程中位元組一律用 2 為底。

練習 12：記憶體技術判斷題

判斷並解釋：(a)「SSD 是 ROM 的一種，因為它不是 RAM。」(b)「SDRAM 是 SRAM 的一種。」(c)「DDR5 是一種全新的記憶體技術，與 DRAM 完全不同。」(d)「SRAM 斷電後仍能保住資料。」

解答

(a) ✗。SSD 用 flash（EEPROM 的一種），**非揮發性**但可重寫，絕不是唯讀；而且它也是隨機存取的。「RAM ／ ROM」這組名字是歷史包袱。
(b) ✗。SDRAM = Synchronised **DRAM**（時脈驅動的 DRAM），與 SRAM 毫無關係，只是名字像。
(c) ✗。DDR 1–5 是與 DRAM 之間的**資料傳輸協定**家族，底層仍是 DRAM（1T1C 電容儲存），不是根本不同的技術。
(d) ✗。SRAM 的 S 是 Static（不需 refresh），但它仍是**揮發性**的——斷電資料就沒了。「不用 refresh」≠「不用供電」。

A 附錄：速查表

A.1 本週元件總覽

元件	觸發方式	行為	構造
R-S 閂鎖	位準（主動低）	$S'\downarrow$ 存 1；$R'\downarrow$ 存 0；皆高保持	2 NAND 交叉耦合
D 閂鎖	位準（*en* 主動高）	*en* 高時 Q 跟隨 D；低時保持	R-S + 2 NAND + NOT
D 正反器	邊緣（正緣）	上升緣取樣 D，其餘時間凍結	2 個 D 閂鎖 + NOT
暫存器	邊緣 + *load*	上升緣且 *load* 高才載入	DFF + MUX 回饋
RAM	邊緣 + 地址	讀即時；寫在上升緣（*load* 高）	暫存器陣列 + MUX/DEMUX

A.2 時脈公式

頻率與週期	$f = 1/T$，$T = 1/f$
時脈上限	$T \geq$ 組合邏輯最大傳播延遲
地址位元數	$\lceil \log_2(\text{字數}) \rceil$
換底公式	$\log_2 x = \log_b x / \log_b 2$

A.3 名詞中英對照

英文	中文	英文	中文
combinational logic	組合邏輯	clock cycle / period	時脈週期／週期時間
sequential logic	序向邏輯	frequency (Hz)	頻率（赫茲）
internal state	內部狀態	register	暫存器
latch	閂鎖	bus	匯流排
flip-flop	正反器	word size	字長
active low / high	主動低／高電位	address space	地址空間
timing diagram	時序圖	cell	記憶胞
propagation delay	傳播延遲	volatile	揮發性
level-triggered	位準觸發	RAM / ROM	隨機存取／唯讀記憶體
edge-triggered	邊緣觸發	EEPROM / flash	電子可抹除 ROM ／快閃
leader / follower	領導者／跟隨者	SRAM / DRAM	靜態／動態 RAM
metastability	亞穩態	refresh	更新（再充電）
setup / hold time	建立／保持時間	KiB / MiB / GiB	2 為底的容量單位

A.4 參考資料

- John Lapinskas, *COMSM1302 Week 3 lecture slides*（3-1 至 3-4），University of Bristol.
- Nisan & Schocken, *The Elements of Computing Systems*（nand2tetris），Ch. 3 Sequential Logic（Bit、Register、RAM 系列晶片規格）.

- Wikipedia：*Flip-flop (electronics)*、*Metastability (electronics)*、*Static random-access memory*、*Dynamic random-access memory*.
- CMU 15-213 *Introduction to Computer Systems* 講義：The Memory Hierarchy（SRAM/DRAM 比較）.
- Ginosar, *Fourteen Ways to Fool Your Synchronizer*（IEEE D&T, 2011）——亞穩態與同步器的經典教程.