COMSM1302 電腦架構概論
四份模擬試卷完整詳解

Test 1 練習卷 A/B（數位邏輯）、Test 2 練習卷 A/B（Hack VM 與編譯）

理論卷逐題解析

2026 年 7 月

使用說明：本文件涵蓋四份理論模擬卷的所有題目：

- **Test 1 練習卷 A**（Theroy1.pdf，12 題）：數字表示法、布林代數、卡諾圖、閂鎖/正反器、時脈、FSM。
- **Test 1 練習卷 B**（Theory 2.pdf，12 題）：DNF/CNF、邏輯電路、R-S 閂鎖、D 正反器、2 補數、傳播延遲。
- **Test 2 練習卷 A**（Test2_Theory1.pdf，12 題）：C 指令、Hack VM、文法與剖析、堆積記憶體配置。
- **Test 2 練習卷 B**（Test2_Theory2.pdf，11 題）：記憶體區段、詞法分析、RISC/CISC、管線危障、CST、記憶體碎片化。

每題先重述題意，接著給出答案，再詳細說明推導過程。原卷部分題目為下拉選單作答，若選項文字與此處敘述略有出入，請依照解題過程中的觀念挑選對應選項。

目錄

1 Test 1 練習卷 A（Theroy1.pdf） 4

第 1 題 4
第 2 題 4
第 3 題 5
第 4 題 5
第 5 題 6
第 6 題 6
第 7 題 7
第 8 題 8

第 9 題 8
第 10 題 8
第 11 題 9
第 12 題 10

2 Test 1 練習卷 B（Theory 2.pdf） 11

第 1 題 11
第 2 題 11
第 3 題 12
第 4 題 12
第 5 題 13
第 6 題 14
第 7 題 14
第 8 題 15
第 9 題 15
第 10 題 16
第 11 題 16
第 12 題 17

3 Test 2 練習卷 A（Test2_Theory1.pdf） 18

第 1 題 18
第 2 題 18
第 3 題 18
第 4 題 19
第 5 題 19
第 6 題 20
第 7 題 20
第 8 題 21
第 9 題 21
第 10 題 22
第 11 題 23
第 12 題 24

4 Test 2 練習卷 B（Test2_Theory2.pdf） 27

第 1 題 27
第 2 題 27
第 3 題 28
第 4 題 28
第 5 題 29
第 6 題 29
第 7 題 30
第 8 題 30
第 9 題 31
第 10 題 32
第 11 題 33

1　Test 1 練習卷 A（Theroy1.pdf）

第 1 題　　（2 分）

題目： 2080 在下列哪些進位表示法中是合法的數字?（二進位／十進位／十六進位／八進位）

答案

二進位：**否**　十進位：**是**　十六進位：**是**　八進位：**否**

解題過程：

判斷方式是檢查「2080」的每一個數字符號是否屬於該進位制允許的數字集合：

- **二進位**只允許 $\{0,1\}$：2080 含有 2 和 8，不合法。
- **十進位**允許 $\{0,\ldots,9\}$：全部合法。
- **十六進位**允許 $\{0,\ldots,9,\mathrm{A},\ldots,\mathrm{F}\}$：2、0、8、0 都合法（其值為 $2\times16^3+0\times16^2+8\times16+0=8320$）。
- **八進位**只允許 $\{0,\ldots,7\}$：含有 8，不合法。

第 2 題　　（4 分）

題目： 下列哪些邏輯等價式是由 De Morgan 定律所定義的?

a) $\neg a \wedge (b \vee c) \equiv (\neg a \wedge b) \vee (\neg a \wedge c)$

b) $\neg a \vee \neg b \vee \neg c \equiv \neg(a \wedge b) \vee \neg c$

c) $\neg(a \vee b) \equiv \neg a \wedge \neg b$

d) $a \vee 0 \equiv a$

答案

a) **否**　b) **是**　c) **是**　d) **否**

解題過程：

De Morgan 定律的兩個形式是：

$$\neg(a \wedge b) \equiv \neg a \vee \neg b \qquad \neg(a \vee b) \equiv \neg a \wedge \neg b$$

- a) 是**分配律**（distributivity）：$\wedge$ 對 $\vee$ 的分配，與 De Morgan 無關。
- b) 把等式右邊的 $\neg(a \wedge b)$ 用 De Morgan 展開成 $\neg a \vee \neg b$，正是左邊，所以這條等價式正是 De Morgan 定律（套用在子式 $a \wedge b$ 上）。
- c) 就是 De Morgan 定律的標準形式之一。
- d) 是**同一律**（identity law）。

第 3 題　　(3 分)

題目：求下列無號二進位數的十進位值。

答案

a) $0b0001\,0000 = \mathbf{16}$　b) $0b1010\,0001 = \mathbf{161}$　c) $0b0010\,1010 = \mathbf{42}$

解題過程：

把每個為 1 的位元乘上對應的 2 的冪次再相加（最右邊是 2^0）：

- a) 只有第 4 位是 1：$2^4 = 16$。
- b) 第 7、5、0 位是 1：$2^7 + 2^5 + 2^0 = 128 + 32 + 1 = 161$。
- c) 第 5、3、1 位是 1：$2^5 + 2^3 + 2^1 = 32 + 8 + 2 = 42$。

第 4 題　　(5 分)

題目：下列敘述哪些為真?

a) SRAM 的存取速度比 DRAM 慢。

b) NOR 是最常用的功能完備（functionally complete）閘。

c) 正反器（flip-flop）是邊緣觸發的。

d) 時脈頻率的設定與傳播延遲無關。

e) 暫存器只需要 load 訊號為高電位就會更新儲存值。

答案

a) **False**　b) **False**　c) **True**　d) **False**　e) **False**

解題過程：

- a) 相反：**SRAM 比 DRAM 快**（SRAM 用 6 電晶體單元、不需刷新；DRAM 用電容、需要刷新且讀取較慢），這也是 SRAM 被用作快取的原因。
- b) NAND 和 NOR 都是功能完備的，但**最常用的是 NAND**（製作成本低、速度快、面積小）。
- c) 正確。閂鎖（latch）是**位準**觸發，正反器（flip-flop）是**邊緣**觸發——只在時脈的上升（或下降）邊緣取樣輸入。
- d) 時脈週期必須**長於**電路中最長的組合邏輯傳播延遲（關鍵路徑），否則訊號來不及穩定，因此頻率絕對受傳播延遲限制。
- e) 暫存器由正反器構成，除了 load 為高之外，還必須等到**時脈邊緣**才會把輸入寫入。

第 5 題　　(4 分)

題目：給定主動低電位（active-low）R-S 閂鎖的波形（輸入為 S' 與 R'），求在標示時間點 a–d 時輸出 Q 的值。

從波形圖讀出（時間軸 0–10）：

$$S' = \begin{cases} 1 & [0,\ 1.25) \\ 0 & [1.25,\ 2.5) \\ 1 & [2.5,\ 6) \\ 0 & [6,\ 7.3) \\ 1 & [7.3,\ 8) \\ 0 & [8,\ 10] \end{cases} \qquad R' = \begin{cases} 0 & [0,\ 0.5) \\ 1 & [0.5,\ 3) \\ 0 & [3,\ 4.75) \\ 1 & [4.75,\ 10] \end{cases}$$

標示點約在 $a \approx 0.4$、$b \approx 2.9$、$c \approx 5.3$、$d \approx 8.4$。

答案

a) $Q = \mathbf{0}$　b) $Q = \mathbf{1}$　c) $Q = \mathbf{0}$　d) $Q = \mathbf{1}$

解題過程：

主動低電位 R-S 閂鎖的規則：**哪個輸入是 0，哪個就「作用」**。

S'	R'	行為
0	1	Set：$Q = 1$
1	0	Reset：$Q = 0$
1	1	Hold：維持前值
0	0	禁止（非法）狀態

逐點分析：

- **a** $(t \approx 0.4)$：$S' = 1,\ R' = 0 \to$ Reset 作用中 $\to Q = 0$。
- **b** $(t \approx 2.9)$：$S' = 1,\ R' = 1 \to$ Hold。往前找最後一次「作用」：$[1.25, 2.5)$ 期間 $S' = 0$（Set），把 Q 設為 1，之後一直保持 $\to Q = 1$。
- **c** $(t \approx 5.3)$：$S' = 1,\ R' = 1 \to$ Hold。最後一次作用是 $[3, 4.75)$ 期間 $R' = 0$（Reset）$\to Q = 0$，保持至此 $\to Q = 0$。
- **d** $(t \approx 8.4)$：$S' = 0,\ R' = 1 \to$ Set 作用中 $\to Q = 1$。（其實 $[6, 7.3)$ 的 Set 早已把 Q 變 1，之後 Hold、再 Set，都維持 1。）

第 6 題　　(4 分)

題目：卡諾圖如下（列為 ab、欄為 cd，皆按格雷碼順序 00,01,11,10），圖上圈了三個群組。問這些群組**直接**產生哪個布林運算式?

$ab\backslash cd$	00	01	11	10
00	1	1	0	1
01	1	1	1	0
11	0	1	1	0
10	0	0	0	1

藍色 = 左上 2×2 群、黃色 = 中間 2×2 群、綠色 = $cd=10$ 欄上下兩格（垂直繞回）

答案

E. $(b\wedge d)\vee(\neg b\wedge\neg d\wedge c)\vee(\neg c\wedge\neg a)$

解題過程：

逐一讀出每個群組的乘積項（找出群內所有格子共有、值固定的變數）：

- **左上** 2×2（$ab\in\{00,01\}$，$cd\in\{00,01\}$）：共同點是 $a=0$、$c=0$（b,d 都有 0 有 1，消去）$\to\neg a\wedge\neg c$。
- **中間** 2×2（$ab\in\{01,11\}$，$cd\in\{01,11\}$）：共同點是 $b=1$、$d=1\to b\wedge d$。
- $cd=10$ **欄的第一列與最後一列**（$ab=00$ 與 $ab=10$，垂直繞回成一組）：共同點是 $b=0$、$c=1$、$d=0$（a 消去）$\to\neg b\wedge c\wedge\neg d$。

所以整張圖直接產生

$$(\neg a\wedge\neg c)\vee(b\wedge d)\vee(\neg b\wedge c\wedge\neg d),$$

把各項重排即為選項 E（$\neg b\wedge\neg d\wedge c=\neg b\wedge c\wedge\neg d$、$\neg c\wedge\neg a=\neg a\wedge\neg c$）。選項 A 多了一項 $\neg a\wedge b\wedge\neg c\wedge d$（圖上沒有第四個群組），其餘選項或使用 POS 形式（B、D）、或項不符（C），皆不符「直接讀出」的結果。

第 7 題　　**(6 分)**

題目：求下列二進位加法的結果（答案為 8 位元二進位數）。

答案

a) $0b0000\,1000+0b0101\,0110=0b\mathbf{01011110}$

b) $0b0110\,1001+0b0110\,0101=0b\mathbf{11001110}$

c) $0b1001\,1100+0b1011\,1010=0b\mathbf{01010110}$（產生進位、捨棄第 9 位）

解題過程：

直式加法，逢 2 進位：

- a) $8+86=94$。逐位相加沒有重疊的 1（0000 1000 與 0101 0110 沒有同位皆 1），直接合併得 0101 1110 $=94$。✓
- b) $105+101=206=128+64+8+4+2=1100\,1110$。過程：低 4 位 $1001+0101=1110$（無進位出）；高 4 位 $0110+0110=1100$。合起來 1100 1110。

- c) $156 + 186 = 342$。$342 = 256 + 86$，即 9 位元 $1\,0101\,0110$；在 8 位元暫存器中**最高位進位被丟棄**，留下 $0101\,0110$（$= 86$，$342 \bmod 256 = 86$）。

第 8 題　　(6 分)

題目：時脈週期分別為 a) $50\,\mu s$、b) $25\,ns$、c) $2\,s$，求頻率（Hz）。

答案

a) **20000** Hz　b) **40000000** Hz（4×10^7）　c) **0.5** Hz

解題過程：

頻率與週期互為倒數：$f = 1/T$。

- a) $T = 50\,\mu s = 50 \times 10^{-6}\,s$，$f = \dfrac{1}{50 \times 10^{-6}} = 20{,}000$ Hz（20 kHz）。
- b) $T = 25\,ns = 25 \times 10^{-9}\,s$，$f = \dfrac{1}{25 \times 10^{-9}} = 40{,}000{,}000$ Hz（40 MHz）。
- c) $T = 2\,s$，$f = \dfrac{1}{2} = 0.5$ Hz。

第 9 題　　(4 分)

題目：要用二進位計算 $94_{10} - 75_{10}$，且所有二進位值使用 2 補數表示。求完成算式 $0b01011110 + 0b$_____ 的 8 位元二進位數。

答案

$0b$**10110101**（即 -75 的 2 補數表示）

解題過程：

減法在 2 補數系統中轉換成「加上相反數」：$94 - 75 = 94 + (-75)$。求 -75 的 2 補數：

1. $75 = 64 + 8 + 2 + 1 = 0b0100\,1011$。
2. 逐位取反：$0b1011\,0100$。
3. 加 1：$0b1011\,0101$。

驗算：$0b01011110\,(94) + 0b10110101\,(-75) = 1\,0001\,0011$，捨棄進位後為 $0b0001\,0011 = 19 = 94 - 75$。✓

第 10 題　　(6 分)

題目：(下拉選單完成句子)「For a CPU with 4 bytes of memory split into _____, a multiplexer with a 2-bit select input is required to _____ _____ number.」

答案

For a CPU with 4 bytes of memory split into **four 1-byte words（4 個 1 位元組的字組）**, a multiplexer with a 2-bit select input is required to **read（讀取） the word with a given address（指定位址的字組）** number.

解題過程：

- 2 位元的選擇輸入可以編碼 $2^2 = 4$ 種選擇，所以記憶體必須被切成 **4 個字組**——4 bytes 分成 4 份，每份正好 1 byte。
- **多工器（multiplexer）**把多個輸入中的一個接到單一輸出，對記憶體而言是「**讀取**」：從 4 個字組中選出位址（address number）對應的那一個送到輸出。
- 相對地，**解多工器（demultiplexer）**才是把寫入訊號導向指定字組（見練習卷 B 第 9 題）。

第 11 題　　**（6 分）**

題目：FSM 輸入代表使用者是否成功完成任務（1 = 成功、0 = 失敗），FSM 輸出 1，直到使用者**連續失敗兩次**時輸出 0。輸出 0 之後重新計數：連續第三次失敗輸出 1、連續第四次失敗又輸出 0，依此類推。哪個狀態圖正確表示此系統的 **Moore machine**?

答案

E. None of the other options（其餘選項皆不正確）

解題過程：

先建立正確的 Moore machine 應有的樣子。狀態需要記「目前連續失敗了幾次（模 2）」：

狀態	意義	輸出	轉移
S0	連續失敗 0 次	1	輸入 1 → S0；輸入 0 → S1
S1	連續失敗 1 次	1	輸入 1 → S0；輸入 0 → S2
S2	剛完成一對（第 2、4、6⋯次失敗）	0	輸入 1 → S0；**輸入 0 → S1**

關鍵在 $S2 \xrightarrow{0} S1$：輸出 0 之後再失敗一次（連續第 3 次）應輸出 1，且這次失敗是「新一對的第 1 次」。驗證題目給的例子：輸入 $0,0,0,0$ → 狀態 $S1, S2, S1, S2$ → 輸出 $1,0,1,0$，符合「三連敗輸出 1、四連敗輸出 0」。

檢查各選項：

- **A**：狀態與輸出正確（S0/1、S1/1、S2/0），但 S2 的**兩條**出邊（輸入 0 與輸入 1）都指回 **S0**。模擬 $0,0,0,0$：$S1(1), S2(0), S0(1), S1(1)$——第四次失敗輸出 1，違反題意（應為 0）。錯誤。
- **B**：S2 有一個標 1 的**自迴圈**。這表示連續兩次失敗後即使「成功」也永遠停在輸出 0 的狀態，違反「輸出 0 之後（成功時）回到輸出 1」的行為。錯誤。
- **C、D**：邊上標記為「輸入/輸出」（如 0/1、1/1）形式，是 **Mealy machine** 的畫法（輸出掛在轉移上），不是題目要求的 Moore machine（輸出掛在狀態上）。且行為亦不符。錯誤。

四個都不對，所以答案是 E。

第 12 題 (0 分)

題目:（回饋題）如果對試題有疑義請在此說明，否則留白。此題不計分。

答案

不計分，若無疑義**留白**即可。

2　Test 1 練習卷 B（Theory 2.pdf）

第 1 題　　(5 分)

題目：在不做任何化簡的前提下，判斷下列各式是 DNF（析取正規形）、CNF（合取正規形）還是兩者皆非。

a) $(A \wedge B) \vee C$

b) $(A \vee \neg B) \wedge (C \vee A)$

c) $\neg(A \wedge B) \wedge (\neg B \vee C)$

d) $(\neg B \wedge \neg C) \vee \neg(B \wedge A)$

e) $(C \wedge \neg A) \vee (B \wedge \neg C)$

答案

a) **DNF**　b) **CNF**　c) **皆非**　d) **皆非**　e) **DNF**

解題過程：

定義（其中「文字（literal）」指變數或其否定，如 A、$\neg B$）：

- **DNF** = 「文字的 AND」再用 OR 連接：$(\cdots \wedge \cdots) \vee (\cdots \wedge \cdots) \vee \cdots$
- **CNF** = 「文字的 OR」再用 AND 連接：$(\cdots \vee \cdots) \wedge (\cdots \vee \cdots) \wedge \cdots$
- 單一文字可視為只有一個元素的 AND 或 OR 子句。
- **否定符號只能作用在單一變數上**；一旦出現 $\neg(\ldots)$ 包住複合式，就既不是 DNF 也不是 CNF。

- a) $(A \wedge B) \vee C$：合取子句 $A \wedge B$ 與單一文字 C 的析取 → DNF。
- b) 兩個析取子句 $(A \vee \neg B)$、$(C \vee A)$ 的合取 → CNF。
- c) $\neg(A \wedge B)$ 的否定包住複合式 → 皆非。
- d) $\neg(B \wedge A)$ 同樣包住複合式 → 皆非。
- e) 兩個合取子句的析取 → DNF。

第 2 題　　(3 分)

題目：邏輯式 $Out = (A \vee B) \wedge (\neg A \vee C)$。下列哪個電路圖是**不含任何化簡或代數操作**的直接實作?

答案

A

解題過程：

「直接實作」表示電路結構要與式子逐字對應：

1. 一個 **OR 閘**計算 $A \vee B$；
2. 一個 **NOT 閘**計算 $\neg A$，接著另一個 **OR 閘**計算 $\neg A \vee C$；
3. 最後一個 **AND 閘**把兩個 OR 的輸出接起來得到 Out。

選項 A 正是這個結構（NOT 在 A 分支進入第二個 OR 之前；最後是 AND）。其他選項的錯誤：

- B：把兩個 OR 換成 AND、最後用 OR——計算的是 $(A \wedge B) \vee (\neg A \wedge C)$。
- C：NOT 放在下方 OR 的**輸出**上，計算 $(A \vee B) \wedge \neg(A \vee C)$。
- D：閘的種類與位置皆不符（最後是 OR）。

第 3 題　　**(4 分)**

題目：下列各組運算式是否邏輯等價?

a) $A \wedge (B \vee C)$ 與 $(A \wedge B) \vee (A \wedge C)$

b) $(A \wedge B) \vee C$ 與 $(A \wedge C) \vee (B \wedge C)$

c) $A \wedge (B \wedge C)$ 與 $(A \wedge C) \wedge (B \wedge C)$

d) $(A \vee B) \vee C$ 與 $A \vee (B \vee C)$

答案

a) **等價**　b) **不等價**　c) **等價**　d) **等價**

解題過程：

- a) 這是 $\wedge$ 對 $\vee$ 的**分配律**，成立。
- b) 取 $A = B = 1,\ C = 0$：左式 $= (1 \wedge 1) \vee 0 = 1$；右式 $= (1 \wedge 0) \vee (1 \wedge 0) = 0$。找到反例，不等價。
- c) 右式展開為 $A \wedge B \wedge C$（C 重複出現由**冪等律** $C \wedge C \equiv C$ 吸收），與左式相同，等價。
- d) $\vee$ 的**結合律**，成立。

第 4 題　　**(4 分)**

題目：完成主動高電位（active-high）R-S 閂鎖的真值表（Q_{prev}、Q'_{prev} 為前一刻的值）。

答案

R	S	Q	Q'
0	0	Q_{prev}	Q'_{prev}
0	1	1	0
1	0	0	1
1	1	0	0

解題過程：

主動高電位 R-S 閂鎖由兩個交叉耦合的 **NOR** 閘組成，輸入為 1 時「作用」：

- $R=0, S=0$：兩者都不作用 → **保持**（$Q=Q_{prev}$，$Q'=Q'_{prev}$）。
- $R=0, S=1$：**Set** → $Q=1$、$Q'=0$。
- $R=1, S=0$：**Reset** → $Q=0$、$Q'=1$。
- $R=1, S=1$：**非法狀態**。兩個 NOR 閘各自有一個輸入為 1，輸出都被壓成 0，所以 $Q=0$ 且 $Q'=0$（注意此時 Q' 不再是 Q 的反相；且兩輸入同時放開會產生競態）。

第 5 題　　(5 分)

題目：下列敘述哪些為真?

a) 閂鎖（latch）是位準觸發的。

b) 存取 RAM 中的資料比 ROM 快。

c) Mealy machine 需要的狀態數比等價的 Moore machine 多。

d) 有 2 個選擇位元的解多工器有 4 個輸入。

e) 給定低電位輸入，one-shot 會輸出恰好一個時脈週期的高電位訊號。

答案

a) **True**　b) **True**　c) **False**　d) **False**　e) **False**

解題過程：

- a) 正確：閂鎖是**位準（level）**觸發（致能為高時輸入直通），正反器才是邊緣觸發。
- b) 正確：一般而言 RAM 的讀取速度比 ROM（如快閃記憶體）快，這也是開機時把程式載入 RAM 執行的原因之一。
- c) 相反：**Mealy 機通常需要的狀態數少於或等於** Moore 機，因為輸出掛在轉移上，可以把「同狀態、不同輸出」的情況合併。
- d) 解多工器是 1 個輸入、2^n 個**輸出**：2 個選擇位元 → 1 個輸入、4 個輸出。「4 個輸入」錯誤。
- e) one-shot 是在輸入出現（高電位／觸發）時輸出一個時脈週期的脈衝；輸入保持**低電位**時輸出維持低電位，不會產生脈衝。

第 6 題　　(4 分)

題目：給定**下降緣觸發** D 正反器的波形（資料輸入 D、時脈 en），求標示點 a–d 的輸出 Q。

從波形圖讀出：

$$D=\begin{cases}1 & [0,\,0.6)\\ 0 & [0.6,\,2.2)\\ 1 & [2.2,\,2.9)\\ 0 & [2.9,\,3.1)\\ 1 & [3.1,\,6.5)\\ 0 & [6.5,\,7.1)\\ 1 & [7.1,\,8.9)\\ 0 & [8.9,\,10]\end{cases}\qquad en \text{ 的下降緣：} t=1,\ 3,\ 5,\ 7,\ 9$$

標示點約在 $a\approx 1.7$、$b\approx 3.6$、$c\approx 6.7$、$d\approx 7.7$。

答案

a) $Q=\mathbf{0}$　b) $Q=\mathbf{0}$　c) $Q=\mathbf{1}$　d) $Q=\mathbf{0}$

解題過程：

下降緣觸發 D 正反器的規則：**只在時脈（en）由 1 變 0 的瞬間取樣** D，其餘時間輸出保持不變。所以對每個標示點，找出它**之前最近的一次下降緣**，看當時 D 的值：

- **a**（$t\approx 1.7$）：最近的下降緣在 $t=1$，當時 $D=0$（D 在 $[0.6, 2.2)$ 為 0）$\to Q=0$。
- **b**（$t\approx 3.6$）：最近下降緣在 $t=3$，當時 $D=0$——注意 D 恰好在 $t=3$ 附近有一個短暫的低谷 $[2.9, 3.1)$，這是本題的陷阱 $\to Q=0$。
- **c**（$t\approx 6.7$）：最近下降緣在 $t=5$（$t=7$ 的下降緣還沒到），當時 $D=1\to Q=1$。
- **d**（$t\approx 7.7$）：最近下降緣在 $t=7$，當時 $D=0$（D 在 $[6.5, 7.1)$ 為 0）$\to Q=0$。

第 7 題　　(3 分)

題目：8 位元二進位數 $0b10110010$ 在下列表示法中的值是多少?

答案

a) 無號表示：**178**　b) 符號—大小表示：**-50**　c) 2 補數表示：**-78**

解題過程：

$0b1011\,0010$：

- a) **無號**：$2^7+2^5+2^4+2^1=128+32+16+2=178$。

- b) **符號—大小（signed-magnitude）**：最高位 1 表示負號，其餘 7 位 0110010 = 50 是大小 → −50。
- c) **2 補數**：最高位權重為 -2^7 → $-128+32+16+2=-78$。（或用 $178-256=-78$。）

第 8 題　　**(4 分)**

題目：卡諾圖如下（含未知格 w、x、y、z）。若此圖產生的公式是 $(b\wedge c)\vee(a\wedge\neg d)$，求 w、x、y、z。

$ab\backslash cd$	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	1	0	w	x
10	1	0	y	z

答案

w = **1**　x = **1**　y = **0**　z = **1**

解題過程：

公式的每個乘積項覆蓋的格子必須全為 1，未被任何項覆蓋的格子必須為 0：

- $b\wedge c$ 覆蓋 $b=1$（列 $ab\in\{01,11\}$）且 $c=1$（欄 $cd\in\{11,10\}$）的 8 格中……本圖已知 $(01,11)=(01,10)=1$ ✓，而 $(11,11)=w$、$(11,10)=x$ 也在其中 → $w=1$、$x=1$。
- $a\wedge\neg d$ 覆蓋 $a=1$（列 $\{11,10\}$）且 $d=0$（欄 $\{00,10\}$）：已知 $(11,00)=(10,00)=1$ ✓，且 $(11,10)=x$、$(10,10)=z$ 在其中 → $z=1$（與 $x=1$ 一致）。
- $(10,11)=y$：此格 $a=1,b=0,c=1,d=1$，**不**被 $b\wedge c$（需 $b=1$）也**不**被 $a\wedge\neg d$（需 $d=0$）覆蓋 → $y=0$。

第 9 題　　**(6 分)**

題目：（下拉選單完成句子）「For a RAM circuit with 8 bytes of memory split into _____, a demultiplexer with a 4-bit select input is required to _____ _____ number.」

答案

For a RAM circuit with 8 bytes of memory split into **sixteen 4-bit words（16 個 4 位元／半位元組的字組）**, a demultiplexer with a 4-bit select input is required to **write to（寫入） the word with a given address（指定位址的字組）** number.

解題過程：

- 4 位元選擇輸入可定址 $2^4=16$ 個字組。8 bytes = 64 位元，分成 16 份，每份 64/16 = 4 位元（半位元組，nibble）。

- **解多工器**把單一輸入訊號（如寫入致能）分配到多個輸出中的一個，對 RAM 而言用來把「寫入」導向指定位址的字組。
- 與練習卷 A 第 10 題成對比：多工器 = 讀取（多選一），解多工器 = 寫入（一分多）。

第 10 題　　(6 分)

題目：下列 8 位元 2 補數十六進位數的十進位值為何?

答案

a) $0xA1 =$ **-95**　b) $0x2F =$ **47**　c) $0xD6 =$ **-42**

解題過程：

先轉成無號值，若 ≥ 128（最高位為 1）則減 256：

- a) $0xA1 = 10 \times 16 + 1 = 161 \geq 128 \rightarrow 161 - 256 = -95$。
- b) $0x2F = 2 \times 16 + 15 = 47 < 128 \rightarrow$ 就是 $+47$。
- c) $0xD6 = 13 \times 16 + 6 = 214 \geq 128 \rightarrow 214 - 256 = -42$。

第 11 題　　(6 分)

題目：電路有三個輸入 A, B, C、兩個輸出。閘延遲：NOT = 5 ps、AND = 10 ps、OR = 15 ps、XOR = 20 ps、導線可忽略。若輸入在時脈上升緣改變，要保證下個上升緣前 $Out1$、$Out2$ 都正確，時脈頻率最高可達多少 MHz?

電路連接（由圖讀出）：

$$Out1 = \underbrace{(\neg A \vee B)}_{\text{NOT+OR}} \wedge \underbrace{\big(B \wedge (A \oplus C)\big)}_{\text{AND}} \qquad Out2 = (A \oplus C) \oplus \neg B$$

答案

最長路徑 = 40 ps → 最高頻率 = $1/(40\,\text{ps}) = 25\,\text{GHz} =$ **25000 MHz**

解題過程：

把每條「輸入 → 輸出」路徑的閘延遲加總，找出**關鍵路徑**（最長者）：

Out1 的路徑（最後一級是 AND，10 ps）：

- $A \rightarrow$ NOT(5) $\rightarrow$ OR(15) $\rightarrow$ AND(10) = 30 ps
- $B \rightarrow$ OR(15) $\rightarrow$ AND(10) = 25 ps
- $A, C \rightarrow$ XOR(20) $\rightarrow$ AND(10) $\rightarrow$ AND(10) = **40** ps
- $B \rightarrow$ AND(10) $\rightarrow$ AND(10) = 20 ps

Out2 的路徑（最後一級是 XOR，20 ps）：

- $A, C \to \mathrm{XOR}(20) \to \mathrm{XOR}(20) = \mathbf{40}$ ps
- $B \to \mathrm{NOT}(5) \to \mathrm{XOR}(20) = 25$ ps

關鍵路徑為 40 ps，時脈週期至少 40 ps：

$$f_{\max} = \frac{1}{40 \times 10^{-12}\,\mathrm{s}} = 2.5 \times 10^{10}\,\mathrm{Hz} = 25\,\mathrm{GHz} = 25000\,\mathrm{MHz}.$$

第 12 題　　**(0 分)**

題目：（回饋題，同卷 A 第 12 題）

答案

不計分，若無疑義**留白**即可。

3　Test 2 練習卷 A（Test2_Theory1.pdf）

第 1 題　　（2 分）

題目：一條 C 指令（C-instruction）有幾個運算元（operand）?

答案

3（comp、dest、jump）

解題過程：

Hack 的 C 指令格式為 `dest = comp ; jump`，機器碼編碼為

$$\underbrace{111}_{\text{op}}\ \underbrace{a\,cccccc}_{\text{comp}}\ \underbrace{ddd}_{\text{dest}}\ \underbrace{jjj}_{\text{jump}}$$

本課程把 comp、dest、jump 稱作 C 指令的三個運算元（卷 B 第 3 題的用語「the comp, dest, and jump operands」即為佐證），其中 dest 與 jump 可省略（編碼為全 0），但欄位仍然存在，所以答案是 3。

第 2 題　　（2 分）

題目：Hack VM 內部用什麼值表示「true」?（以有號十進位作答）

答案

-1

解題過程：

Hack VM 的比較運算（`eq`、`gt`、`lt`）把 true 表示成**全部位元為 1** 的字，即 $0xFFFF$；以 16 位元 2 補數解讀就是 -1。false 則是 0。這樣設計的好處是 true/false 可以直接與位元運算（`and`、`or`、`not`）搭配使用。

第 3 題　　（2 分）

題目：下列哪個 ISA 最可能出現在手機裡?（ARM ／ Hack ／ MIPS ／ x64 ／ x86-64）

答案

A. ARM

解題過程：

ARM 是行動裝置的主流 ISA：RISC 設計、能源效率高，適合電池供電的裝置。x64/x86-64 主要用於桌機、筆電與伺服器；MIPS 曾用於路由器與嵌入式系統；Hack 是本課程的教學用架構，不存在於真實產品。

第 4 題　　(5 分)

題目：下列各項是 ISA 的性質還是微架構（microarchitecture）的性質?

a) 字組長度與記憶體位址空間

b) 能源效率

c) 時脈速度

d) 使用的電晶體數量

e) 支援的定址模式

答案

a) **ISA**　b) **微架構**　c) **微架構**　d) **微架構**　e) **ISA**

解題過程：

判斷準則：**ISA 是程式設計師看得到的「合約」**（指令集、暫存器、位址空間、定址模式——寫組合語言時必須知道的事）；**微架構是硬體如何實作這份合約**（管線、快取、電晶體數、時脈、功耗——同一個 ISA 可以有很多不同實作）。

- a) 字組長度、位址空間：組語程式依賴它們 → ISA。
- b)(c)(d) 能源效率、時脈、電晶體數：同一 ISA 的不同晶片可以完全不同（例如各代 Intel 處理器都是 x86-64）→ 微架構。
- e) 定址模式是指令語意的一部分 → ISA。

第 5 題　　(4 分)

題目：給定 Hack 組合語言文法（節錄）：

```
<instruction>   ::= (<aInstruction> | <cInstruction>), newline;
<aInstruction>  ::= "@", (integerLiteral | identifier | <memoryKeyword>);
<memoryKeyword> ::= "SCREEN" | "KBD" | "SP" | "LCL" | "ARG" | "THIS" | "THAT";
```

對輸入 `@loopstart\n` 完成剖析樹：根節點 `<instruction>` 的兩個子節點為 a、b；a 的兩個子節點為 c、d。

答案

a = **<aInstruction>**　b = **newline**　c = **"@"**　d = **identifier**

解題過程：

- 規則 `<instruction> ::= (<aInstruction> | <cInstruction>), newline` 表示 instruction 有兩個子節點：第一個是 aInstruction 或 cInstruction，第二個是 newline。輸入以 `@` 開頭，是 A 指令 → a = **<aInstruction>**、b = newline。

- 規則 `<aInstruction> ::= "@", (integerLiteral | identifier | <memoryKeyword>)` → c = `"@"`。
- `loopstart` 不是整數，也不是七個保留的記憶體關鍵字之一（那些全是大寫），所以是 **identifier** → d = identifier。

第 6 題　　(7 分)

題目： 判斷下列敘述的真假。

答案

a) **True**　b) **False**　c) **True**　d) **False**　e) **True**　f) **True**　g) **True**

解題過程：

a)「在標準記憶體映射下，Hack VM 程式的遞迴呼叫絕不可能深達 360 層而不發生堆疊溢位。」**True**。標準映射的堆疊區是 RAM[256]–RAM[2047]，共 $2048 - 256 = 1792$ 個字。每次函式呼叫至少要推入 5 個字（回傳位址 + 儲存的 LCL、ARG、THIS、THAT），$360 \times 5 = 1800 > 1792$，還沒算引數、區域變數與工作堆疊，所以 360 層一定溢位。

b)「為了編譯標籤，典型的 Hack VM 轉譯器會仔細追蹤產生的組合碼目前的 ROM 位址。」**False**。VM 轉譯器輸出的是**符號化**的組語標籤（如 `(label)`、`@label`），把標籤解析成 ROM 位址是**組譯器**第二階段的工作，轉譯器不需要知道 ROM 位址。

c)「手動堆積配置函式（malloc/free、Memory.alloc/deAlloc）通常無法防止記憶體碎片化。」**True**。碎片化來自配置與釋放的**順序**；配置器可以緩解（合併、bins）但無法阻止碎片化發生，這是講義第 10-4 講的核心觀念。

d)「程式可以藉由 free 掉所有 malloc 的變數、再重新 malloc 它們，手動整理（defragment）記憶體。」**False**。free 之後**資料就沒了**——重新 malloc 拿到的是未定義內容的新記憶體；而且指標會懸空。講義的作法是「依序 deAlloc 並**立刻** reAlloc、同時把資料從舊位置搬到新位置」，並且強調 alloc/deAlloc 本身無法替你做這件事（呼叫端手上的指標配置器動不得）。單純「全部 free 再全部 malloc」並不能達成搬移資料的整理。

e)「多種不同程式語言的編譯器可能使用同一種中間表示法。」**True**。經典例子是 LLVM IR：C、C++、Rust、Swift 的編譯器都輸出 LLVM IR。

f)「一種中間表示法可能編譯到多種不同的組合語言。」**True**。同一份 IR 可由不同後端輸出 x86-64、ARM 等；這正是 IR 的「$M + N$ 而非 $M \times N$」優勢。

g)「Hack VM 的函式名稱應以檔名（去掉.vm）開頭，接一個點，再接描述性名稱。」**True**。慣例是 `FileName.functionName`，例如 `String.vm` 內的函式叫 `String.appendChar`。

第 7 題　　(3 分)

題目： 填空：Hack 指令 $0xE564$ 計算 _____，把結果存入 _____，並且 _____。

答案

計算 **D | A**（D 與 A 的位元 OR），存入 **A**，且**若結果 < 0 則跳躍（JLT）**。

解題過程：

把 $0xE564$ 展開成二進位並按 C 指令欄位切割：

$$0xE564 = \underbrace{111}_{\text{op}}\ \underbrace{0}_{a}\ \underbrace{010101}_{comp}\ \underbrace{100}_{dest}\ \underbrace{100}_{jump}$$

（$0xE564 = 1110\,0101\,0110\,0100$。）

- $a = 0$ 且 $comp = 010101$：查 comp 表得 **D | A**。
- $dest = 100$：三個位元依序對應 A、D、M → 只有 **A** 被寫入。
- $jump = 100$：三個位元依序對應 < 0、$= 0$、> 0 → **JLT**，即結果小於 0 時跳到 ROM[A]。

整條指令等同組語 `A = D|A ; JLT`。

第 8 題　　**（6 分）**

題目：填空（堆疊與堆積的配置規則）。

答案

Memory allocated on the **heap** remains in use until **it is explicitly freed（被明確釋放，如呼叫 free / deAlloc）**. Variables can be allocated on the heap if their size is known **at run time（執行期）**. Memory allocated on the **stack** remains in use until **the function that allocated it returns（配置它的函式返回）**. Variables can be allocated on the stack if their size is known **at compile time（編譯期）**. In C, we should normally expect variables to be allocated on **the stack** unless an explicit call to malloc or free is made, in which case they will be allocated on **the heap**.

解題過程：

- **堆積（heap）**：生命週期由程式設計師控制——直到明確呼叫 free/deAlloc 才釋放；大小只要在**執行期**（呼叫 malloc 的那一刻）知道即可。
- **堆疊（stack）**：跟著函式呼叫框架走——函式返回時自動釋放；編譯器要在編譯期算出框架大小，所以變數大小必須在**編譯期**已知。
- C 的區域變數預設放**堆疊**；只有 malloc（配套 free）明確要求的記憶體放**堆積**。

第 9 題　　**（4 分）**

題目：John 想寫一個文法來匹配「成對括號」字串（如 `()`、`(())`、`((())())` 合法；`(`、`(()` 不合法；不考慮空字串）：

```
<pair>       ::= '(' ')'
<expression> ::= <pair> | '(' <expression> ')' | '(' <expression> ')' <expression>
```

下列何者正確?

答案

C. 文法生成的字串都是成對括號字串，但**並非所有**成對括號字串都能由文法生成（文法把某些合法字串誤判為「不匹配」）。

解題過程：

分兩個方向檢查：

- **健全性（soundness）**：對產生式做結構歸納——`<pair>` 是成對的；若 E 成對，則 (E) 與 $(E)E'$（E' 也成對）都成對。所以文法生成的每個字串都是成對括號字串。✓
- **完備性（completeness）**：找反例——`()()` 是成對括號字串，但無法生成：
  - `<pair>` 只能是 `()`（2 字元）；
  - `'(' <expression> ')'` 生成的字串首尾括號互相匹配，但 `()()` 的第 1 個 `(` 在第 2 個字元就閉合了；
  - `'(' <expression> ')' <expression>` 的第一部分 `( E )` 中 E 至少 2 字元，所以第一部分至少 4 字元，無法只是 `()`。

  三條產生式都失敗 → `()()` 不在語言中。✗

生成的都合法（不會誤收），但漏掉一些合法字串（誤拒）→ 選 C。

第 10 題　　(5 分)

題目： RAM 中有一串連續存放的字組清單，起始位址存放在變數 `list_start` 中。哪段程式碼會把清單的**第 50 個**元素載入 M?

答案

F. None of the other options（其餘選項皆錯）

解題過程：

正確作法：第 1 個元素在位址 $base+0$，所以第 50 個在 $base+49$；且 `list_start` 是**變數**，清單起始位址是它的**內容**，必須先 `A=M` 解參考。正確片段應為：

```
@list_start
A=M         // A = 清單起始位址（變數的內容）
D=A
@49         // 第 50 個元素的偏移量是 49
D=D+A
A=D         // A = base+49，此時 M 即為第 50 個元素
```

逐一檢查選項：

- **A**（`A=M;D=A;@50;D=D+A;M=D`）：偏移用 50（差一錯誤），且最後 `M=D` 是**寫入**記憶體而非讀取。✗
- **B**（`A=M;D=A;@50;D=D+A;A=D`）：結尾正確但偏移 50 指到第 51 個元素。✗
- **C**（`D=A;@49;…;A=D`）：少了 `A=M`，用的是變數 `list_start` 本身的**位址**而不是它儲存的起始位址。✗
- **D**：同 C 的錯誤，還用了 50。✗
- **E**（`A=M;D=A;@49;D=D+A;M=D`）：偏移正確，但最後 `M=D` 把位址**寫進** RAM[49]，破壞記憶體且沒有載入。✗

沒有任何選項正確 → F。

第 11 題　　(6 分)

題目：補完下列 Hack VM 程式（`String.appendChar`：把一個字元附加到字串物件尾端。引數 0 為字串位址、引數 1 為字元。欄位依序為：長度、最大長度、字元陣列的基底位址）。

答案

六個空格依序為：`pop pointer 0`、`if-goto nocrash`、`call Sys.error 1`、`pop local 0`、`pop that 0`、`return`

解題過程：

完整程式與逐行說明：

```
function String.appendChar 1
// 檢查字串是否還有空間，否則呼叫 Sys.error(52) 當機
push argument 0
pop pointer 0        // (1) THIS = 字串物件位址，讓 this 區段對準物件欄位
push this 0          // 長度 length
push this 1          // 最大長度 maxLength
lt                   // length < maxLength ？ （true = 還有空間）
if-goto nocrash      // (2) 有空間就跳過當機碼
push constant 52
call Sys.error 1     // (3) 沒空間：以錯誤碼 52 呼叫 Sys.error
label nocrash
// 把「新的最後一個字元」的位址存入 local 0
push this 2          // 字元陣列基底位址
push this 0          // 目前長度
add                  // 基底 + 長度 = 新字元要放的位址
pop local 0          // (4) 存入 local 0
// 把字元寫入字串
push local 0
```

```
pop pointer 1          // THAT = 該位址
push argument 1        // 要附加的字元
pop that 0             // (5) 寫入 RAM[THAT]
// 更新長度
push this 0
push constant 1
add
pop this 0             // length = length + 1
// 函式結尾
push constant 0        // 回傳值（虛設）
return                 // (6) 返回
```

關鍵觀念：

- `pointer 0/pointer 1` 分別設定 `this/that` 區段的基底；把物件位址 pop 進 `pointer 0` 後，`this 0..2` 就是物件的三個欄位。
- `lt` 比較「次頂 < 頂端」：先推 length 再推 maxLength，算的是 $length < maxLength$。
- Hack VM 的函式一定要以 `return` 結束，且返回前堆疊頂端要有回傳值（此處推 0）。

第 12 題　　(4 分)

題目：考慮課堂上講的 first-fit 配置演算法，含釋放區段的合併（coalescing）但不含 bins（「attempt 3」）。此演算法中 true 代表空閒（free）區段、false 代表使用中（used）區段。給定三個記憶體區段的部分內容（下表），求 (a)–(h)。

區段 1		區段 2		區段 3	
RAM[0x759]	(a)	RAM[0x1420]	___	RAM[0x2000]	___
RAM[0x75A]	0xFFFF	RAM[0x1421]	0x0000	RAM[0x2001]	0xFFFF
RAM[0x75B]	(b)	RAM[0x1422]	(c)	RAM[0x2002]	(e)
RAM[0x75C]	0x2003	RAM[0x1423]	0x75C	RAM[0x2003]	(f)
RAM[0x75D]	0x1422	RAM[0x1424]	___	RAM[0x2004]	___
…		…		…	
RAM[0x1105]	0x9AB	RAM[0x1500]	0xDE	RAM[0x22FF]	(g)
RAM[0x1106]	___	RAM[0x1501]	(d)	RAM[0x2300]	(h)

答案

(a) **0x9AB**　(b) **Null（0x0000）**　(c) **無法判定（使用中區段的資料）**　(d) **0xDF**
(e) **0x75C（指向區段 1 的指標）**　(f) **Null（0x0000）**　(g) **無法判定**　(h) **0x2FE**

解題過程：

步驟 1：回想 attempt 3 的區段格式。講義（第 10-4 講）中每個區段的配置如下（設基底位址為 $base$、可用大小為 s）：

位址	內容
$base$	可用大小 s（第一個字）
$base+1$	狀態：true（$0xFFFF$）＝空閒、false（$0x0000$）＝使用中
$base+2$, $base+3$	**僅空閒區段**：雙向鏈結串列的兩個指標（next ／ prev）
$base+2 \ldots base+s+1$	可用區域（共 s 個字；使用中區段放使用者資料）
$base+s+2$	可用大小 s 再存一次（最後一個字，方便反向走訪）

整個區段佔 $s+3$ 個字；空閒區段串成雙向鏈結串列，RAM[0x800] 指向串列開頭。合併不變式：**兩個相鄰區段不可能都空閒**（否則早已被合併）。

步驟 2：判讀三個區段。每份列出的內容都從區段基底開始（三個區段的第二個字 0xFFFF ／ 0x0000 ／ 0xFFFF 正好都落在 $base+1$，佐證了這個對齊）：

- **區段 1**：$base = 0x759$，狀態 $= 0xFFFF \rightarrow$ **空閒**。列出的最後一個字 RAM[0x1106] 是尾端大小，所以 $s_1 = 0x1106 - 0x759 - 2 =$ **0x9AB** $\rightarrow$ **(a)** $= 0x9AB$（RAM[0x1106] 也等於 0x9AB）。陷阱：RAM[0x1105] $= 0x9AB$ 是可用區域的**最後一個資料字**（$0x75B + 0x9AA = 0x1105$），不是尾端大小——它的值只是誤導用的垃圾資料。若誤以為 0x1105 是尾端，會得出 $s = 0x9AA$ 而與其值矛盾。
- **區段 2**：$base = 0x1420$，狀態 $= 0x0000 \rightarrow$ **使用中**。列出的最後一個字 RAM[0x1501] 是尾端大小：$s_2 = 0x1501 - 0x1420 - 2 =$ **0xDF** $\rightarrow$ **(d)** $= 0xDF$。陷阱：RAM[0x1500] $= 0xDE$ 同樣是最後一個**資料**字（$0x1422 + 0xDE = 0x1500$），假裝成大小的樣子（剛好比真正的大小少 1），引誘你答 0xDE。
- **區段 3**：$base = 0x2000$，狀態 $= 0xFFFF \rightarrow$ **空閒**。尾端大小在 RAM[0x2300]：$s_3 = 0x2300 - 0x2000 - 2 =$ **0x2FE** $\rightarrow$ **(h)** $= 0x2FE$。而 **(g)** ＝ RAM[0x22FF] 是可用區域的最後一個字（$0x2002 + 0x2FD = 0x22FF$）——空閒區段的可用區域內容（除了前兩個指標字）是沒有意義的殘留值 $\rightarrow$ **無法判定**。

步驟 3：重建空閒串列。空閒區段是區段 1 與區段 3；指標欄位在 $base+2$（next）與 $base+3$（prev），指標值記錄的是對方的**第四個字的位址（$base+3$）**——這可由已知資料反推：

- 區段 1 的 prev（RAM[0x75C]）$= 0x2003 = 0x2000 + 3 \rightarrow$ **指向區段 3**：串列順序是「區段 3 $\rightarrow$ 區段 1」。
- 因此區段 3 的 next ＝ **(e)** 必須指回區段 1：$0x759 + 3 =$ **0x75C**。佐證：使用中的區段 2 在 RAM[0x1423] 留著**過期的**指標 $0x75C$——它曾經空閒、曾指向區段 1，被重新配置後這個值成了無意義的殘留，也順便告訴我們指標的編碼方式。同理 RAM[0x75D] $= 0x1422$ 是區段 1 可用區域開頭的殘留垃圾。
- 已知的空閒區段只有這兩個：區段 3 是串列開頭（RAM[0x800] 指向它），其 prev ＝ **(f)**＝ **Null**；區段 1 是串列結尾，其 next ＝ **(b)**＝ **Null**。
- **(c)** ＝ RAM[0x1422]：区段 2 **使用中**，$base+2$ 起是使用者資料，任何值都有可能 $\rightarrow$ **無法判定**。這是本題另一個測驗點：只有**空閒**區段的 $base+2$、$base+3$ 才是指標。

快速檢核表：使用中區段只有三個可信欄位（頭尾的大小、狀態字）；空閒區段多兩個指標欄位；其餘統統是垃圾。看到「像大小、像指標」的值先算位置再下結論——本題的 0x9AB ／ 0xDE ／ 0x75C（在區段 2 中）全部都是設計好的誘餌。

4　Test 2 練習卷 B（Test2_Theory2.pdf）

第 1 題　(3 分)

題目：把 C 之類的高階語言編譯到 Hack VM 時，**全域變數**的欄位最適合存放在哪個記憶體區段？（this ／ that ／ pointer ／ local ／ static ／視情況而定）

答案

E. static

解題過程：

各區段的用途：

- `local`：函式的區域變數，函式返回即消失。
- `this/that + pointer`：存取堆積上的物件與陣列（基底可移動）。
- `static`：**整個檔案共享、程式執行期間永遠存在**——正符合全域變數「生命週期為整個程式」的需求。

第 2 題　(5 分)

題目：在組譯器中，下列哪些工作通常屬於**詞法分析（lexing）**階段？

a) 把程式從字串轉換成一串 token。

b) 建立標籤 → 記憶體位址的符號表。

c) 建立變數 → 記憶體位址的符號表。

d) 把每行組語轉成機器碼。

e) 對產生的機器碼做最佳化。

答案

a) **是**　b) **否**　c) **否**　d) **否**　e) **否**

解題過程：

編譯／組譯的典型階段分工：

階段	工作
詞法分析（lexing）	字元流 → token 流（只認「字」，不管結構與意義）
語法分析（parsing）	token 流 → 剖析樹
語意分析	建符號表（標籤、變數 → 位址）、檢查意義
程式碼產生	輸出機器碼
最佳化	改善產出碼

只有 a) 是詞法分析的工作；b)、c) 屬於語意分析（組譯器的第一遍掃描），d) 是程式碼產生，e) 是最佳化。

第 3 題　　(4 分)

題目： C 指令 `AM=!M` 的 comp、dest、jump 運算元的二進位值為何？（需含前導 0）

答案

comp = **1110001**（$a = 1$ 加上 $cccccc = 110001$；若答案格只要 6 位元的 $cccccc$ 則為 **110001**）
dest = **101**　jump = **000**

解題過程：

- **comp**：`!M` 在 comp 表中屬於「用 M 的那一欄」，需要 $a = 1$，而 $cccccc = 110001$（與 `!A` 同編碼、只差 a 位元）。合起來 comp 欄位為 1110001。
- **dest**：三個位元依序代表 A、D、M。`AM=` 寫入 A 與 M，不寫 D → 101。
- **jump**：沒有跳躍 → 000。

整條指令的完整編碼為 $111\,1110001\,101\,000 = 0xFF15\ldots$ 依欄位串接：$1111\,1100\,0110\,1000 = 0xFC68$。

第 4 題　　(5 分)

題目： 在記憶體配置的情境下，下列哪些是**碎片化（fragmentation）**可能的後果？（假設演算法為含合併、不含 bins 的「attempt 3」）

a) 記憶體中既有的資料損毀、無法再存取。

b) 配置一個大的新區段失敗，即使記憶體中有足夠的可用空間。

c) 配置一個兩個字的區段失敗，即使一半的記憶體是空的。

d) 配置新區段花的時間比平常長很多。

e) 釋放區段花的時間比平常長很多。

答案

a) **否**　b) **是**　c) **是**　d) **是**　e) **否**

解題過程：

- a) 碎片化只影響**尚未配置**的空間的形狀，不會動到已配置區段內的資料。✗
- b) 這是碎片化的**定義性**後果：總空閒空間夠大，但被切成多個不相鄰小塊，沒有單一區段裝得下大請求。✓

- c) 可能。例如「使用中 (1 字) ／空閒 (1 字)」交錯排列：每個空閒區段可用大小都只有 1 個字，兩個字的請求全數失敗，但空閒區段合計佔了約一半的記憶體。（合併不變式只保證**相鄰**空閒區段會合併，交錯排列不違反。）✓
- d) first-fit 每次都要沿著空閒串列掃描；碎片化讓串列變得又長又碎，配置變慢。✓
- e) attempt 3 的 deAlloc 是**常數時間**：改狀態字、看前後相鄰區段（靠頭尾的大小欄位定位）、雙向串列 $O(1)$ 插入／刪除——都與碎片程度無關。✗

第 5 題　　(4 分)

題目：下列 VM 運算序列執行後，堆疊頂端留下的有號十進位值是多少?

```
push constant 4
push constant 5
sub
not
push constant 0
eq
```

答案

-1

解題過程：

逐步追蹤堆疊（頂端在右）：

指令	動作	堆疊
`push constant 4`	推入 4	[4]
`push constant 5`	推入 5	[4, 5]
`sub`	$x - y = 4 - 5$	$[-1]$
`not`	**位元**取反：$\neg 0xFFFF = 0x0000$	[0]
`push constant 0`	推入 0	[0, 0]
`eq`	$0 = 0 \rightarrow \text{true} = -1$	$[-1]$

兩個關鍵：`sub` 是「次頂 − 頂」= $4-5=-1$；`not` 是位元 NOT（不是邏輯「非」），$-1 = 0xFFFF$ 全部取反得 0。最後 `eq` 比較 0 與 0 相等，推入 true $= -1$。

第 6 題　　(5 分)

題目：完成句子：

a) 指令長度可變的情況較可能出現在 _____ 架構。

b) 與 ARM 相比，MIPS 更偏 _____ ISA，而 x86-64 更偏 _____ ISA。

c) 同一段 C 程式分別編譯到 CISC 與 RISC 的機器碼，預期 _____ 的機器碼指令較多。

d) _____ ISA 通常對記憶體的使用效率較差。

答案

a) **CISC**　b) MIPS 更偏 **RISC**、x86-64 更偏 **CISC**　c) **RISC**　d) **RISC**

解題過程：

- a) CISC（如 x86-64）指令長度可變（1–15 位元組）；RISC 通常固定長度（如 32 位元）。
- b) ISA 光譜：MIPS 是教科書級的純 RISC，ARM 居中，x86-64 是典型 CISC。
- c) RISC 指令每條做的事少，同樣的程式需要**更多**條指令。
- d) RISC 指令固定長度、每條又做得少，程式碼密度較低，**記憶體使用效率較差**；CISC 一條抵多條、長度可變，密度較高。

第 7 題　(4 分)

題目：下列哪段 Hack 組語片段就管線化而言可能含有**資料危障（data hazard）**？

答案

D. `A=A+1 / M=M+1 / D=D+A`

解題過程：

資料危障（RAW，read-after-write）：後面的指令要**讀**前面指令**寫**的暫存器，在管線中前一條還沒寫回、後一條就要讀。逐選項檢查：

- A（`M=!M / A=!A / D=!D`）：三條各自讀寫 M、A、D，互不相依。✗
- B（`D=M+1 / A=M+1`）：第二條讀 M 與寫 A，都不是第一條寫的 D。✗
- C（`M=A-D / A=A+D / D=0`）：第二條讀 A、D（皆非第一條寫的 M）；第三條不讀任何東西。✗
- **D**（`A=A+1 / M=M+1 / D=D+A`）：第一條**寫 A**；第二條 `M=M+1` 讀寫 RAM[A]，**隱含讀 A** 來定址；第三條也讀 A。兩處 RAW 相依 → 資料危障。✓

Hack 的陷阱在於 `M` 其實是 RAM[A]，任何用到 M 的指令都隱含依賴 A。

第 8 題　(5 分)

題目：給定 Jack 文法（節錄）：

```
<expression>     ::= <term>, {('+'|'-'|'*'|'/'|'&'|'|'|'<'|'>'|'='), <term>};
<term>           ::= integerLiteral | stringLiteral | 'true' | 'false' | 'null' | 'this
    '
                   | identifier, ['[', <expression>, ']']
                   | '(', <expression>, ')'
                   | (('-'|'~'), <term>) | <subroutineCall>;
```

```
<subroutineCall> ::= identifier, ['.', identifier], '(', <expressionList>, ')';
<expressionList> ::= [<expression>, {',', <expression>}];
```

下列哪一個**不是**合法 `<expression>` 的合法 CST?

答案

E.

解題過程：

逐一驗證每棵樹是否嚴格遵循產生式：

- **A**：`expression → term '&' term`；左 term = `'(' expression ')'`，其中內層 `expression → term '<' term`（兩個 integer literal）；右 term = `'true'`。每一層都符合產生式。✓
- **B**：`expression → term → subroutineCall`；`subroutineCall → identifier '(' expressionList ')'`（點號部分可省略 ✓）；`expressionList → expression ',' expression`。✓
- **C**：`expression → term '|' term`；右 term = `'~' term`（一元運算子接 term ✓）。✓
- **D**：`expression → term → '(' expression ')'`；內層 `expression → term → subroutineCall`，帶 `identifier '.' identifier '(' expressionList ')'`；`expressionList → expression → term → 'this'`。✓
- **E**：樹根 `expression` 的子節點直接是 `identifier '[' expression ']'`——但依文法，`expression` 的子節點只能是 `term`（與運算子）；「`identifier[expression]`」這種陣列索引結構必須包在一個 `term` 節點底下。此樹**跳過了 term 層** → 不是合法 CST。✗

CST（具體語法樹）必須**一字不差**地反映文法推導的每一層，漏掉中間非終端符號就是錯的。

第 9 題　**(5 分)**

題目： 新語言 Crust 的文件說「在 Crust 中，所有字串都以指向堆積記憶體的指標存放」。由此可以對下列敘述做出什麼判斷?

a) 在函式內定義字串變數並回傳它，程式會壞掉。

b) 定義字串變數時，它使用 ASCII 字元集。

c) 把字串變數傳入函式，函式修改字串後，修改在函式返回後仍然存在。

d) 定義 myString 與 myOtherString 後執行 myOtherString = myString，兩者會是彼此獨立的副本，改一個不影響另一個。

e) 定義字串變數後，與它關聯的記憶體終究需要以類似 C 的 free 或 Jack 的 Memory.deAlloc 釋放（可能是手動或自動）。

答案

a) **False（可判斷為假）**　b) **無法判斷**　c) **True**　d) **False**　e) **True**

解題過程：

「字串 = 指向堆積的指標」告訴我們生命週期與共享語意，但沒說編碼方式：

- a) **假**。堆積記憶體**不**隨函式返回而消失（那是堆疊的行為），回傳指向堆積的指標完全安全——這正是字串放堆積的主要好處。
- b) **無法判斷**。指標與堆積跟字元編碼（ASCII、Unicode…）毫無關係。
- c) **真**。傳入的是指標（的副本），函式透過指標修改的是同一塊堆積記憶體，返回後修改仍在。
- d) **假**。指派只是複製**指標**，兩個變數指向同一塊記憶體，透過任一個修改都會影響另一個（除非語言額外做深拷貝，但文件說的是指標語意）。
- e) **真**。堆積記憶體不會自動隨作用域結束而釋放，終究要有人（手動 free 或自動垃圾回收）釋放，否則就是記憶體洩漏。

第 10 題　　**(2 分)**

題目：給定 EBNF 文法（token 為'A'、'B'）：

```
<a> ::= <b>, 'A' | [<a>], 'A';
<b> ::= <b>, <b>, <b>, {<b>} | <a>, <a> | 'B';
```

下列哪個字串**不是**合法的 `<a>`？（A. A　B. BBBBBAAAAAAA　C. AABBA　D. BABABABABABAA　E. BAABAABAABAABAAA　F. 皆非（都合法）　G. 不只一個不合法）

答案

F. None of the other options——五個字串全部都能由文法生成。

解題過程：

先整理文法的意思：`<a>` 一定以 `A` 結尾，且 `<a>` = `A`、`<a>A` 或 `<b>A`；`<b>` = `B`、兩個 `<a>` 相接、或**三個以上** `<b>` 相接。逐一構造推導：

- **A**：`<a>` → [空], `'A'` ✓
- **BBBBBAAAAAAA**（B^5A^7）：`<a>` → `<b>A`，其中 `<b>` = B^5A^6 拆成 8 個 `<b>`：`B,B,B,B,B,AA,AA,AA`（`AA` = `<a><a>` = `A·A` ✓，$8 \geq 3$ ✓）✓
- **AABBA**：`<a>` → `<b>A`，`<b>` = `AABB` 拆成 3 個 `<b>`：`AA`（=`<a><a>`）、`B`、`B` ✓
- **BABABABABABAA**（$(BA)^6A$）：`<a>` → `<b>A`，`<b>` = `BABABABABABA` 拆成 3 個 `<b>`：`BABA,BABA,BABA`，每個 `BABA` = `<a><a>`（`BA` = `<b>A`，`<b>`=`B` ✓）✓

- **BAABAABAABAABAAA**（$(BAA)^5A$）：先看 `<b>` = `BAABAABAABAAB`（= $BAABAABAABAAB$，9 個 `<b>`：`B` 與 `AA` 交錯 ✓）。然後 `<a>` 逐層包裝：`<b>A` → 再 `[<a>]A` → 再 `[<a>]A`，共補回三個尾端 `A`。✓

五個都合法，所以「哪個不合法」的答案是 F（皆非）。

第 11 題　　(8 分)

題目：補完下列 Hack VM 程式：把使用者每次**新的**按鍵依序存入 RAM[0x1000]–RAM[0x1FFF]，寫到 RAM[0x1FFF] 後繞回 RAM[0x1000]；按住不放只記錄一次。

答案

六個空格依序為：`push constant 24576`、`label main_loop_start`、`pop this 0`、`push pointer 0`、`not`、`if-goto main_loop_start`

解題過程：

完整程式與逐行說明（`pointer 0` = this 基底、`pointer 1` = that 基底；鍵盤映射在 RAM[24576 = $0x6000$]）：

```
// 初始化
push constant 0
pop local 0               // local 0 = 上一次記錄的按鍵值（0 = 無）
push constant 4096
pop pointer 0             // THIS = 0x1000: 下一個按鍵要存的位置
push constant 24576       // (1) 0x6000 = 鍵盤的記憶體映射位址
pop pointer 1             // THAT = 鍵盤
label main_loop_start     // (2) 主迴圈進入點（下方有兩處跳回這裡）
// 測試按鍵是否是新的
push that 0               // 目前鍵盤值 = RAM[0x6000]
push local 0
eq                        // 與上次記錄的相同?
if-goto main_loop_start   // 相同（按住或沒變化）→ 不記錄，繼續輪詢
// 把按鍵存入 local 0 以及 RAM[0x1000]-RAM[0x1FFF] 的下一格
push that 0
pop local 0               // 更新「上一次的按鍵」
push local 0
pop this 0                // (3) 寫入 RAM[THIS]: 記錄按鍵
// 遞增下一個儲存位置
push pointer 0            // (4) 取出目前位置
push constant 1
add
pop pointer 0             // THIS = THIS + 1
// 檢查是否已越過 0x1FFF
push constant 8192        // 0x2000
push pointer 0
```

```
eq                        // THIS == 0x2000 ?
not                       // (5) 取反: 還沒到底 → true
if-goto main_loop_start   // (6) 還沒到底 → 跳過繞回的程式
// 繞回 0x1000
push constant 4096
pop pointer 0
goto main_loop_start
```

重點:

- **(1)**: 鍵盤在標準映射的 RAM[24576]，把它設為 that 的基底後，`push that 0` 就能讀取目前按鍵。
- **(2)**: 程式中有 `if-goto main_loop_start` 與 `goto main_loop_start`，卻沒有宣告標籤，所以空格必為 `label main_loop_start`，且要放在「輪詢測試」開始之前。
- **(3)(4)**: `pointer 0` 存放「寫入位置」，透過 `this 0` 間接寫入該位址；遞增時先 `push pointer 0` 再加一放回去。
- **(5)(6)**: `eq` 之後堆疊上是「到達 0x2000 了嗎」；`not` 把它反轉成「還沒到嗎」，再用 `if-goto` 在**還沒到**時跳回主迴圈，讓程式只有在到達邊界時才執行繞回段。

全卷詳解完