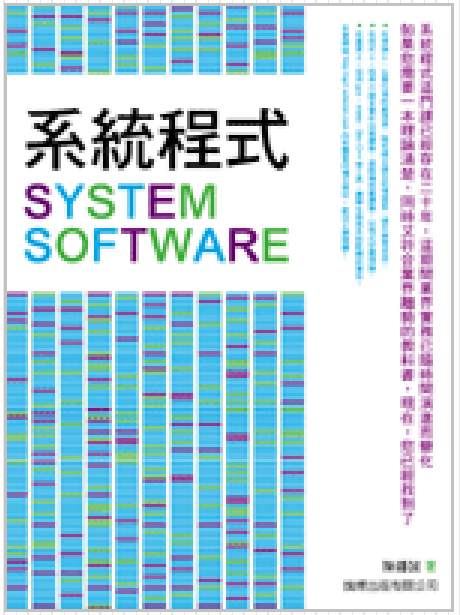
第 2 章、電腦的硬體結構

作者：陳鍾誠



第 2 章、電腦的硬體結構

- 2.1　CPU0 處理器
- 2.2　CPU0 的指令集
- 2.3　CPU0 的運作原理
- 2.4　CPU0 的程式執行
- 2.5　實務案例：IA32 處理器

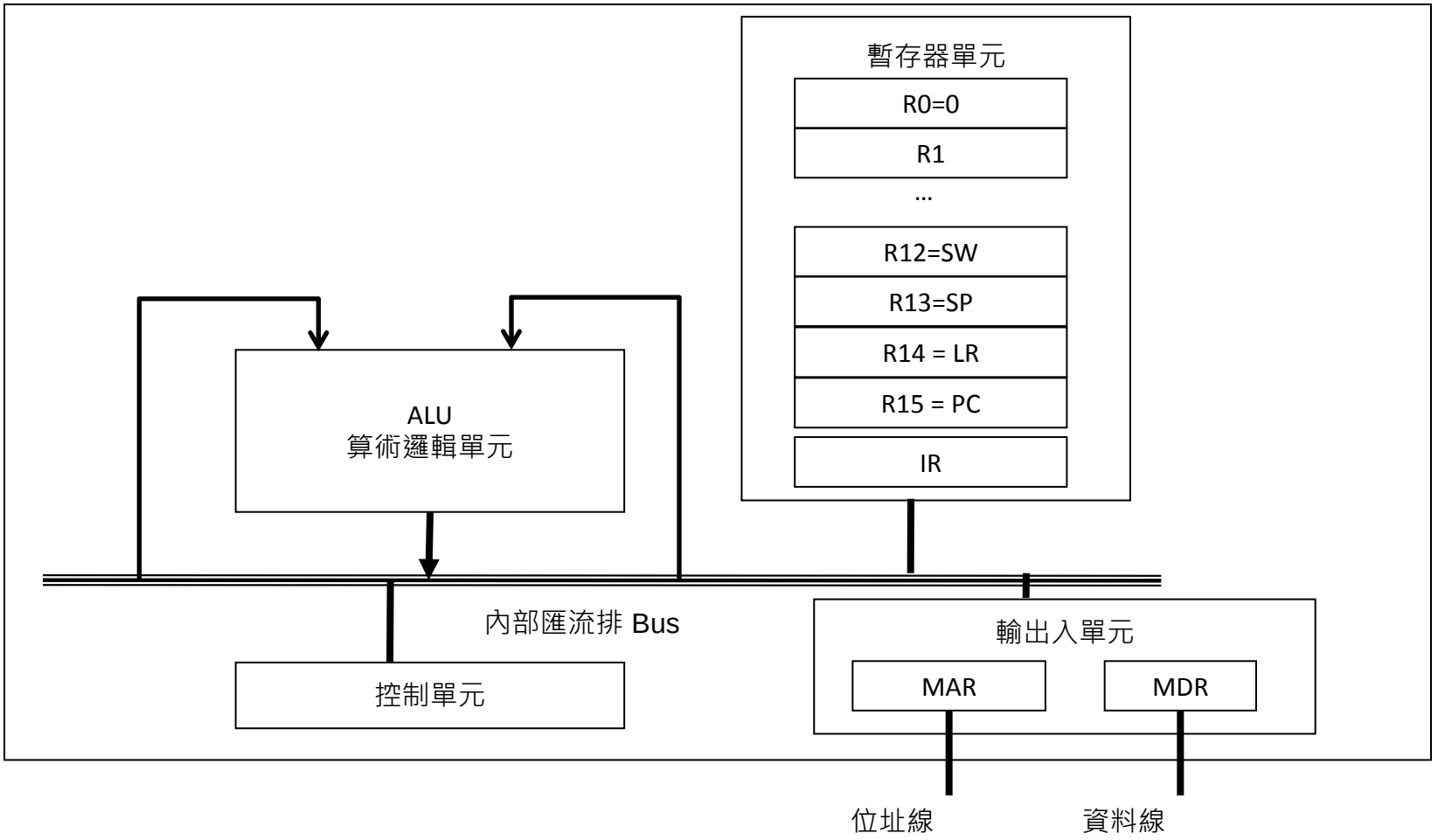
2.1 CPU0 處理器

- CPU0 是筆者所設計的一個簡易的 32 位元處理器, 主要用來說明系統程式的運作原理。

- CPU0 的設計主要是為了教學考量, 設計重點在於簡單、容易理解，因此 CPU0 幾乎不考慮成本與速度的問題。

- 商業上的處理器通常很複雜, 除了考慮成本與速度之外, 有時還會考慮相容性的問題, 因此並不容易理解。

圖 2.1 CPU0 的架構圖



CPU0 的結構

- CPU0 = ALU + 暫存器 + 控制單元

- ALU
  - 採用二補數的方式進行整數運算。
  - 具有加、減、乘、除、邏輯運算與旋轉移位等功能。
  - ALU 可被想像成是 CPU 當中的小型處理器。

- 控制單元 (Control Unit)
  - 根據IR當中的運算碼決定 ALU 的運算類型。
  - 控制資料的傳遞方向。
  - 根據狀態暫存器 SW 的內容，決定是否要跳躍。

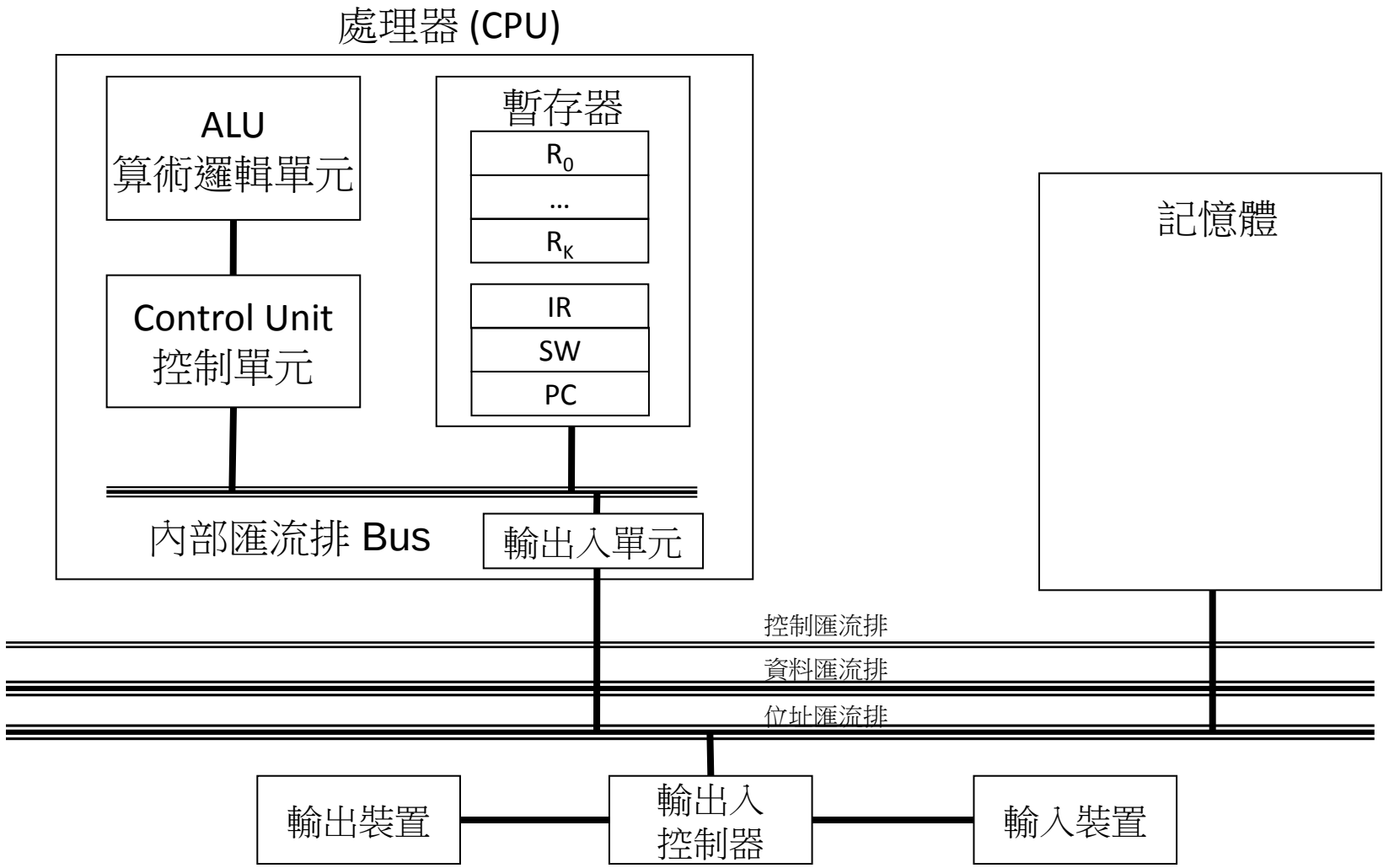
CPU0 的暫存器

- R0
  - 唯讀暫存器。R0的值永遠都是常數零。
- R1-R15
  - 15 個可存取暫存器
  - R12：狀態暫存器 (Status Word, SW)
  - R13：堆疊暫存器 (Stack Pointer Register, SP)
  - R14：連結暫存器 (Link Register, LR)
  - R15：程式計數器 (Program Counter, PC)

電腦的基本結構

- 電腦的五大組成元件
  - Computer = CPU + BUS + Memory + Input + Output
  - CPU (處理器)
  - Memory (記憶體)
  - Bus (匯流排)
  - Input (輸入)
  - Output (輸出)

圖 2.2馮紐曼電腦的結構



2.2 CPU0 的指令集

- 包含四大類的指令
  - 載入儲存 : LD, ST, …
  - 運算指令 : ADD, SUB, XOR, SHL, ROL, …
  - 跳躍指令 : JMP, JGT, JGE, …
  - 堆疊指令 : PUSH, POP, …

CPU0 的指令編碼表 – 載入儲存指令

表格 2.1 CPU0 的指令編碼表

類型	格式	指令	OP	說明	語法	語義
載入儲存	L	LD[2]	00	載入 word	LD Ra, [Rb+Cx]	Ra←[Rb+ Cx]
	L	ST	01	儲存 word	ST Ra, [Rb+ Cx]	Ra→[Rb+ Cx]
	L	LDB	02	載入 byte	LDB Ra, [Rb+ Cx]	Ra←(byte)[Rb+ Cx]
	L	STB	03	儲存 byte	STB Ra, [Rb+ Cx]	Ra→(byte)[Rb+ Cx]
	A	LDR	04	LD (暫存器版)	LDR Ra, [Rb+Rc]	Ra→(byte)[Rb+ Rc]
	A	STR	05	ST (暫存器版)	STR Ra, [Rb+Rc]	Ra→[Rb+ Rc]
	A	LBR	06	LDB (暫存器版)	LBR Ra, [Rb+Rc]	Ra←(byte)[Rb+ Rc]
	A	SBR	07	STB (暫存器版)	SBR Ra, [Rb+Rc]	Ra→(byte)[Rb+ Rc]
	L	LDI	08	立即載入	LDI Ra, Rb+Cx	Ra← Rb + Cx

CPU0 的指令編碼表 – 運算指令

類型	格式	指令	OP	說明	語法	語義
運算指令	A	CMP[3]	10	比較	CMP Ra, Rb	SW ← Ra >=< Rb
	A	MOV	12	移動	MOV Ra, Rb	Ra ← Rb
	A	ADD	13	加法	ADD Ra, Rb, Rc	Ra ← Rb+Rc
	A	SUB	14	減法	SUB Ra, Rb, Rc	Ra ← Rb-Rc
	A	MUL	15	乘法	MUL Ra, Rb, Rc	Ra ← Rb*Rc
	A	DIV	16	除法	DIV Ra, Rb, Rc	Ra ← Rb/Rc
	A	AND	18	邏輯 AND	AND Ra, Rb, Rc	Ra ← Rb and Rc
	A	OR	19	邏輯 OR	OR Ra, Rb, Rc	Ra ← Rb or Rc
	A	XOR	1A	邏輯 XOR	XOR Ra, Rb, Rc	Ra ← Rb xor Rc
	A	ROL[4]	1C	向左旋轉	ROL Ra, Rb, Cx	Ra ← Rb rol Cx
	A	ROR	1D	向右旋轉	ROR Ra, Rb, Cx	Ra ← Rb ror Cx
	A	SHL	1E	向左移位	SHL Ra, Rb, Cx	Ra ← Rb << Cx
	A	SHR	1F	向右移位	SHR Ra, Rb, Cx	Ra ← Rb >> Cx

CPUO 的指令編碼表 – 跳躍指令

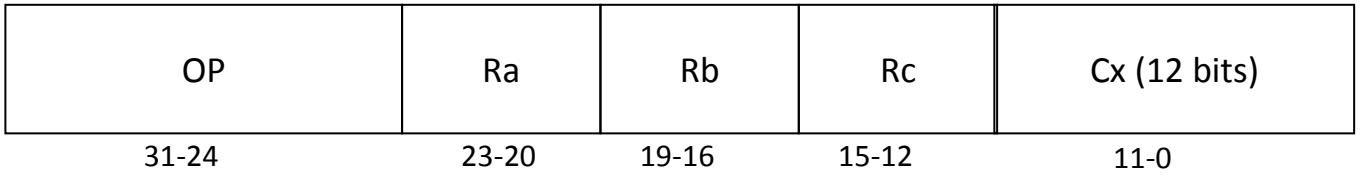
類型	格式	指令	OP	說明	語法	語義
跳躍指令	J	JEQ[5]	20	跳躍 (相等)	JEQ Cx	if SW(=) PC ← PC+Cx
	J	JNE	21	跳躍 (不相等)	JNE Cx	if SW(!=) PC ← PC+Cx
	J	JLT	22	跳躍 (<)	JLT Cx	if SW(<) PC ← PC+Cx
跳躍指令	J	JGT	23	跳躍 (>)	JGT Cx	If SW(>) PC ← PC+Cx
	J	JLE	24	跳躍 (<=)	JLE Cx	if SW(<=) PC ← PC+Cx
	J	JGE	25	跳躍 (>=)	JGE Cx	If SW(>=) PC ← PC+Cx
	J	JMP	26	跳躍 (無條件)	JMP Cx	PC ← PC+Cx
	J	SWI[6]	2A	軟體中斷	SWI Cx	LR ← PC; PC ← Cx
	J	CALL	2B	跳到副程式	CALL Cx	LR← PC; PC ← PC+Cx
	J	RET	2C	返回	RET	PC ← LR

CPU0 的指令編碼表 – 堆疊指令

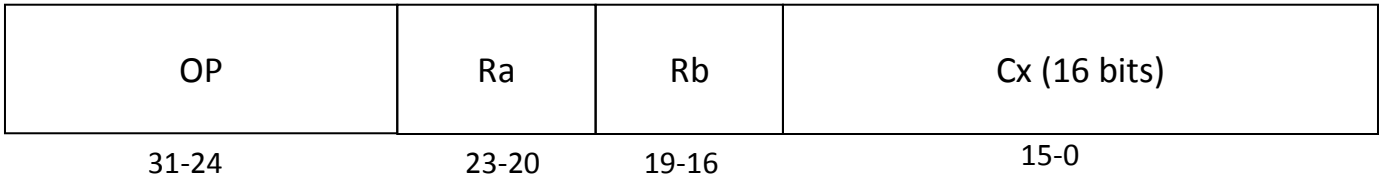
類型	格式	指令	OP	說明	語法	語義
堆疊指令	A	PUSH	30	推入 word	PUSH Ra	SP-=4; [SP] = Ra;
	A	POP	31	彈出 word	POP Ra	Ra = [SP]; SP+=4;
	A	PUSHB	32	推入 byte	PUSHB Ra	SP--; [SP] = Ra; (byte)
	A	POPB	33	彈出 byte	POPB Ra	Ra = [SP]; SP++; (byte)

圖 2.3 CPU0的指令格式

A 型



L 型



J 型

2.3 CPU0 的運作原理

- 移動指令 : MOV R1, R2
- 加法指令 : ADD R1, R2, R3
- 減法指令 : SUB R1, R2, R3
- 邏輯運算 : XOR R1, R2, R3
- 移位指令 : SHL R1, R2, 4
- 比較指令: CMP R1, R2
- 跳躍指令: JMP [0x30]
- 條件跳躍: JLE [0x30]
- 載入指令: LD R1, [0x28]
- 儲存指令: ST R1, [0x32]

圖 2.4 移動指令MOV R1, R2 的執行過程

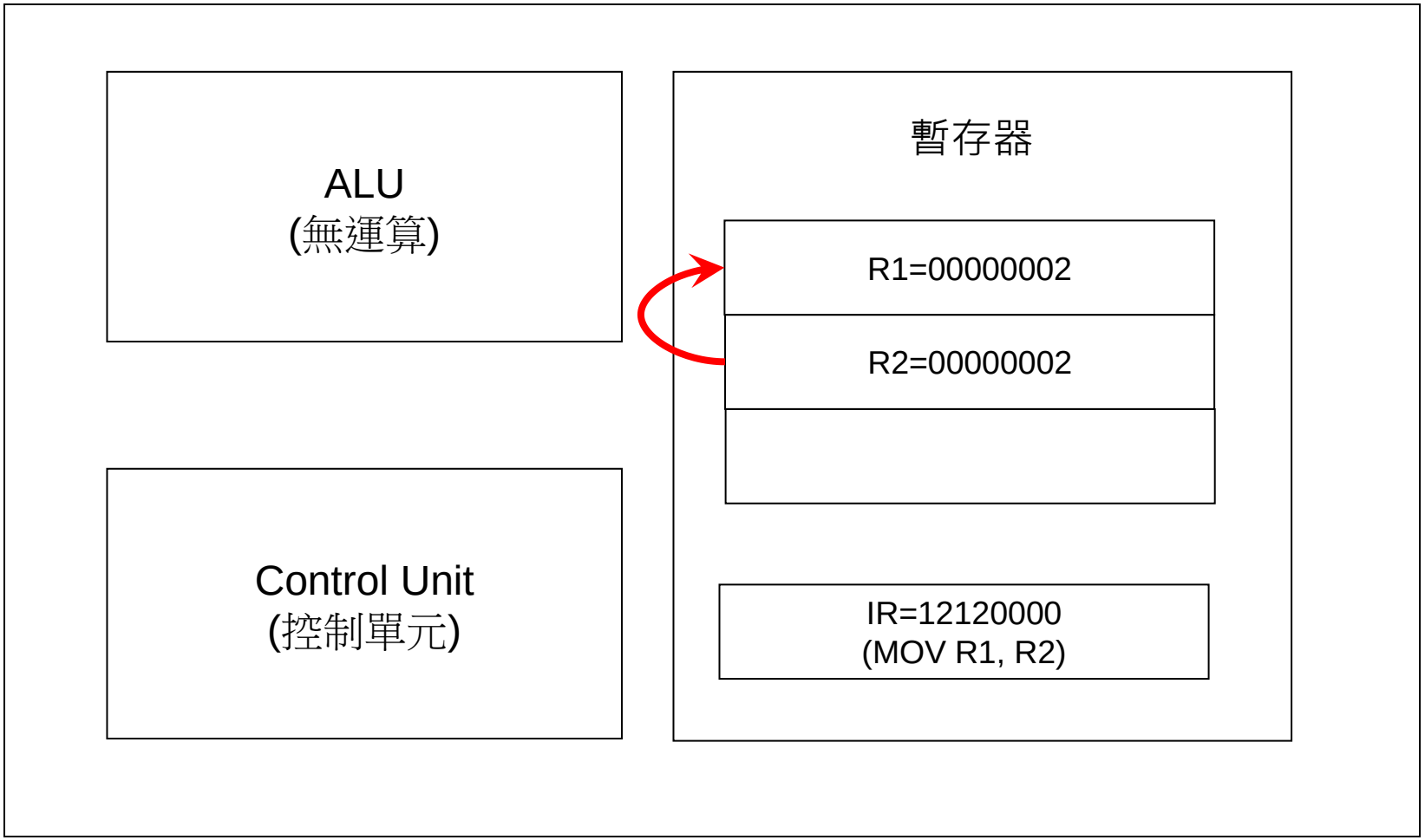


圖 2.5加法指令ADD R1, R2, R3的執行過程

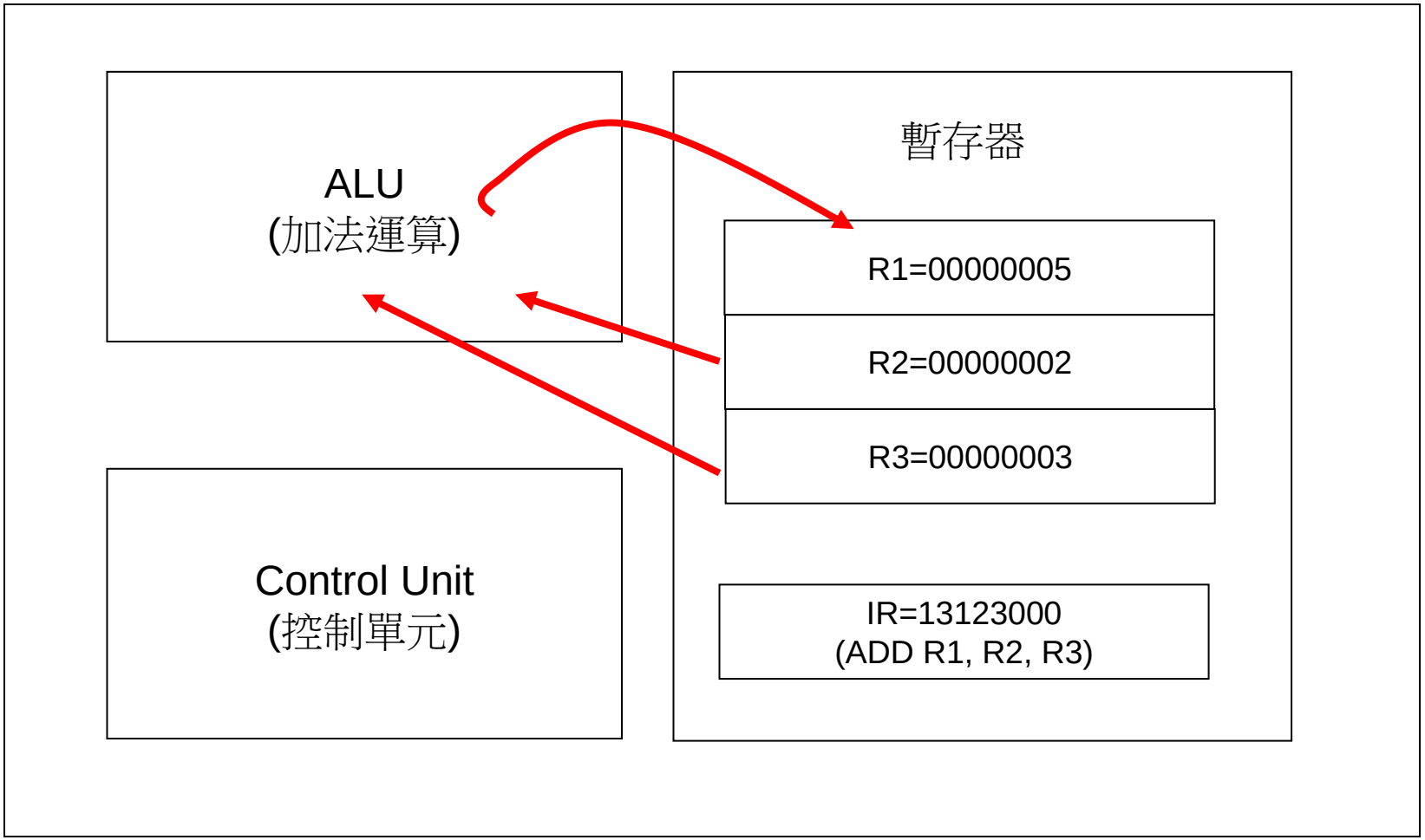


圖 2.6減法指令SUB R1, R2, R3 的執行過程

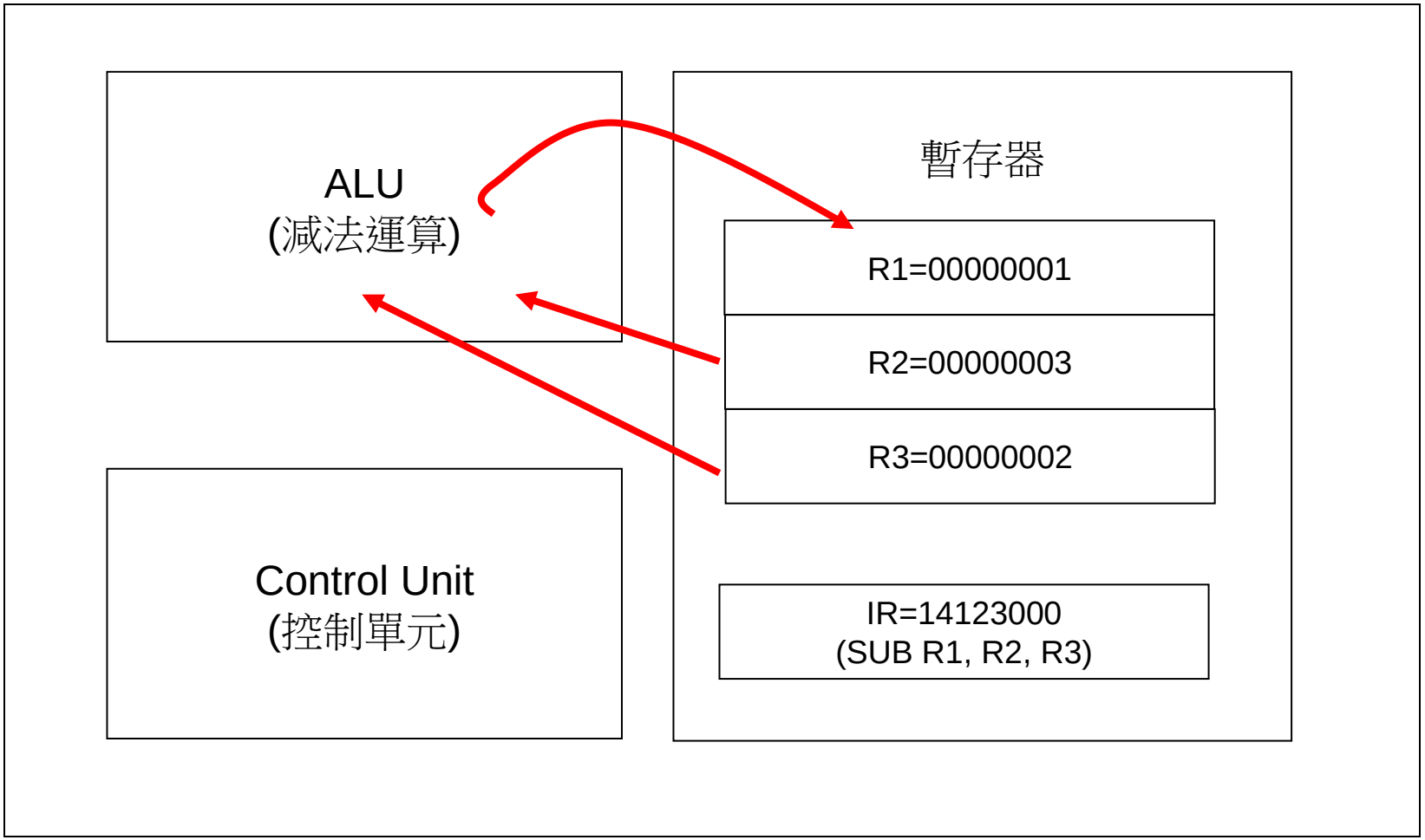
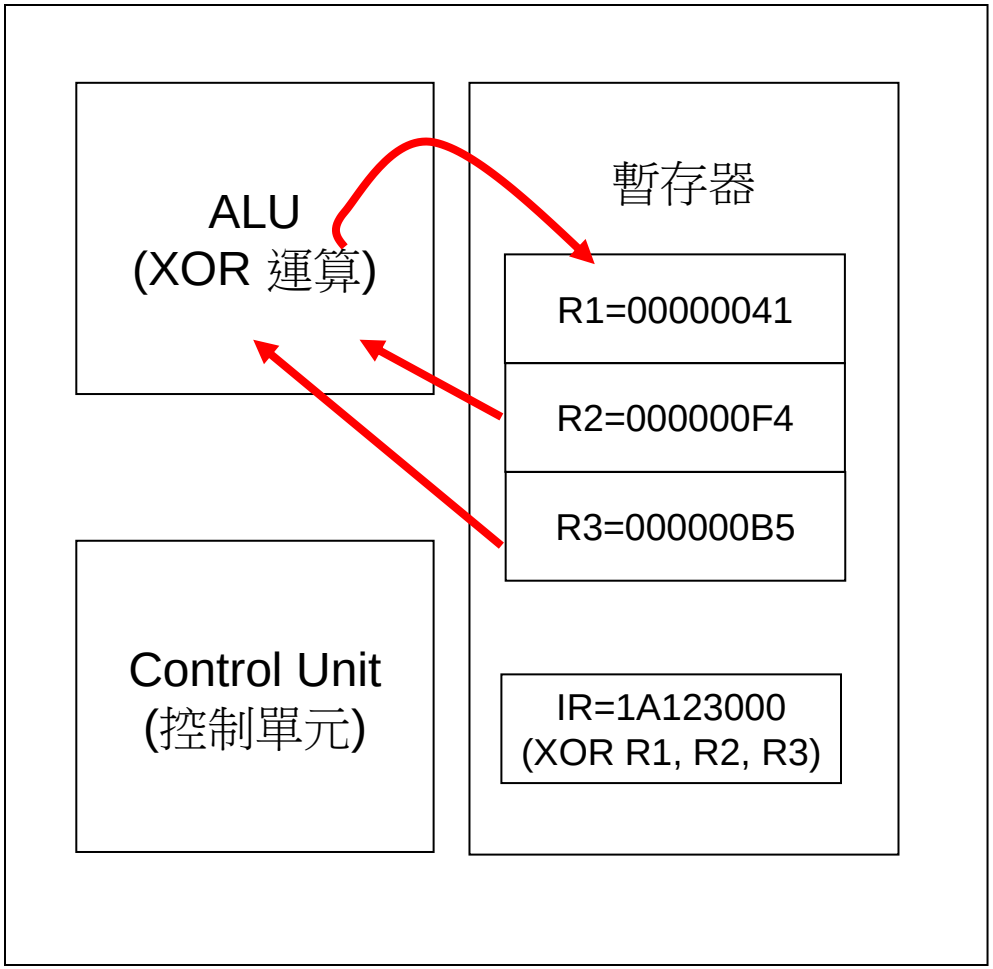


圖 2.7 邏輯運算 XOR R1, R2, R3 的執行過程



R1 (二進位) = * 0100 0001

R2 (二進位) = * 1111 0100

R3 (二進位) = * 1011 0101

圖 2.8 移位運算 SHL R1, R2, 4 的執行過程

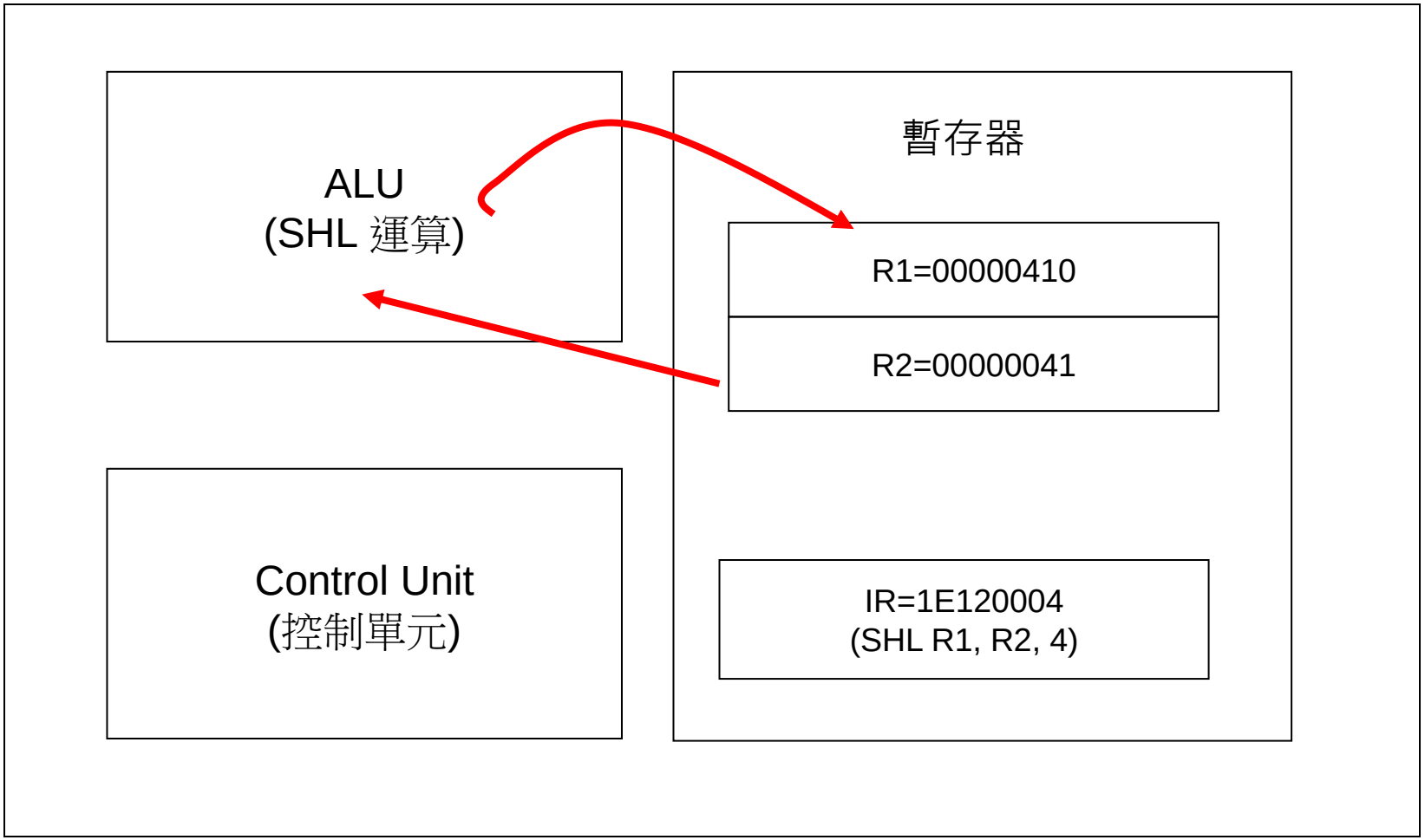
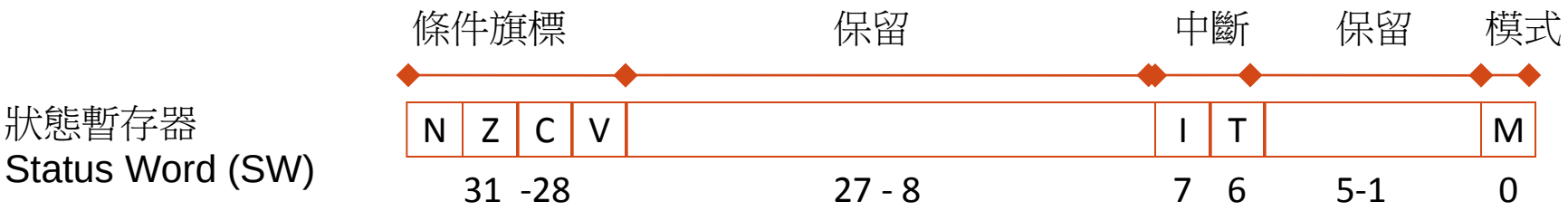


圖 2.9 CPU0 當中的狀態暫存器 SW (R12)



N : 負旗標 (Negative)

Z : 零旗標 (Zero)

C : 進位旗標 (Carry)

V : 溢位旗標 (Overflow)

I : 中斷位元 (Interrupt)

T : 軟體中斷位元 (Trap)

M : 處理器模式 (Mode)

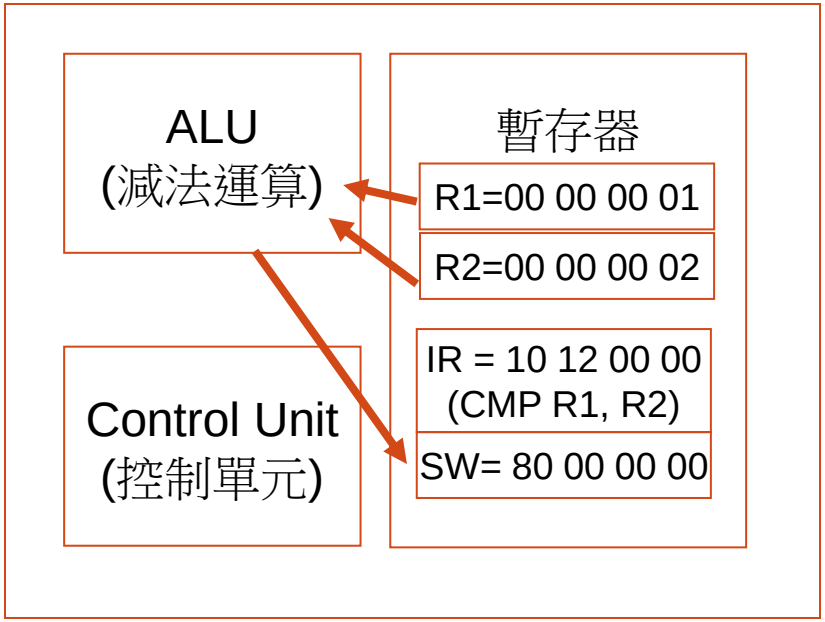
M=0 : 使用者模式

M=1 : 特權模式

比較指令

- 條件旗標的 N、Z 旗標值, 可以用來代表比較結果, 當執行 CMP Ra, Rb 動作後, 可能會有下列三種情形。
  - 1. 若 Ra > Rb, 則 N=0, Z=0。
  - 2. 若 Ra < Rb, 則 N=1, Z=0。
  - 3. 若 Ra = Rb, 則 N=0, Z=1。
- 如此, 用來進行條件跳躍的 JGT、JGE、JLT、JLE、JEQ、JNE 等指令, 就可以根據 N、Z 旗標決定是否進行跳躍。

圖 2.10 比較指令CMP R1, R2 的執行過程



使用 LD 指令進行跳躍的動作

▶範例 2.1 使用 LD 指令進行跳躍的動作

組合語言程式	說明
ADDR: WORD 0xFF000000	宣告一個整數變數以存放 0xFF000000
...	
LD R15, ADDR	將 0xFF000000 放入到 PC，也就是跳躍到該處
...	

圖 2.11 跳躍指令JMP [0x30] 的執行過程

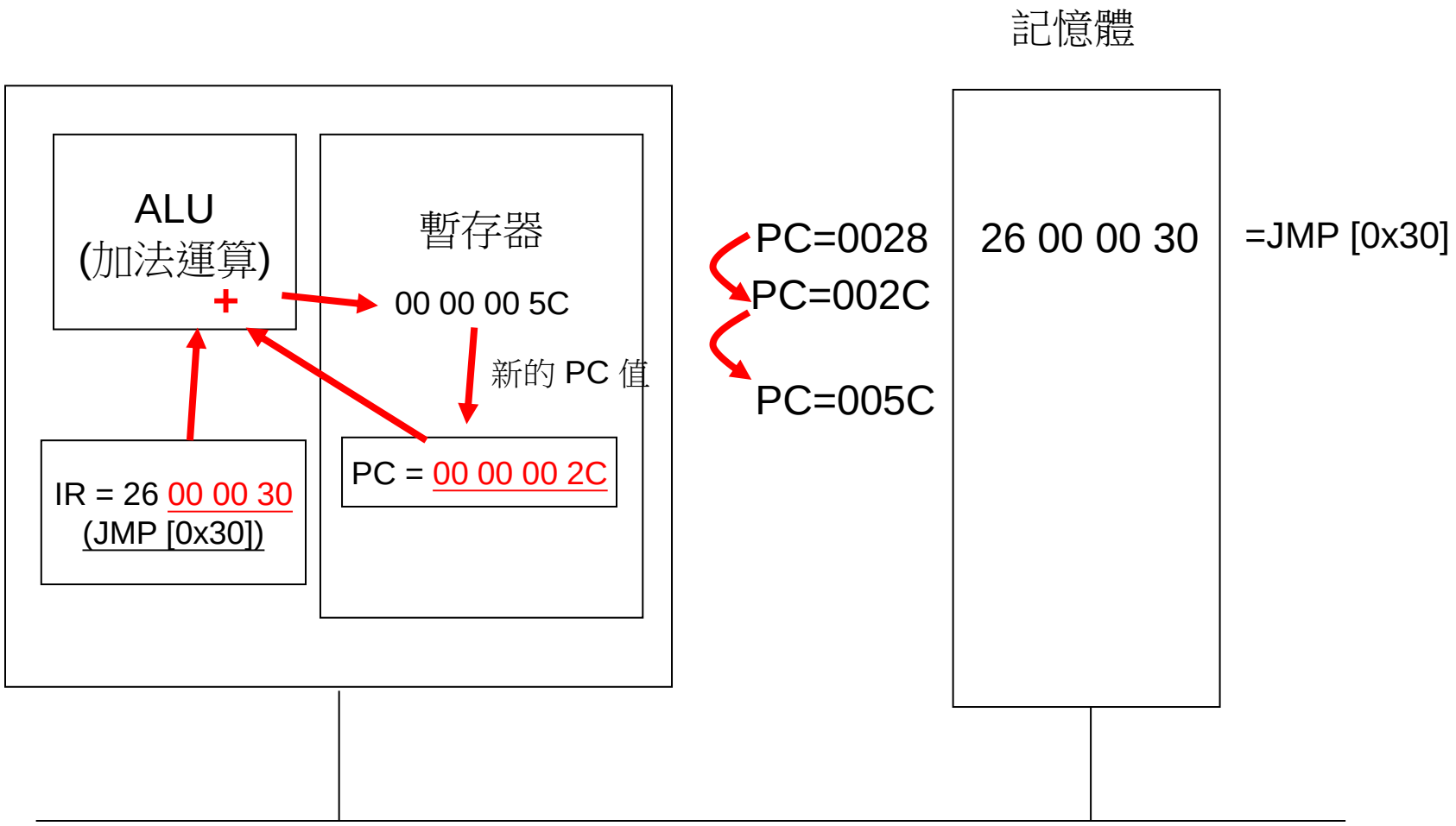


圖 2.12 條件式跳躍指令JLE [0x30] 的執行過程

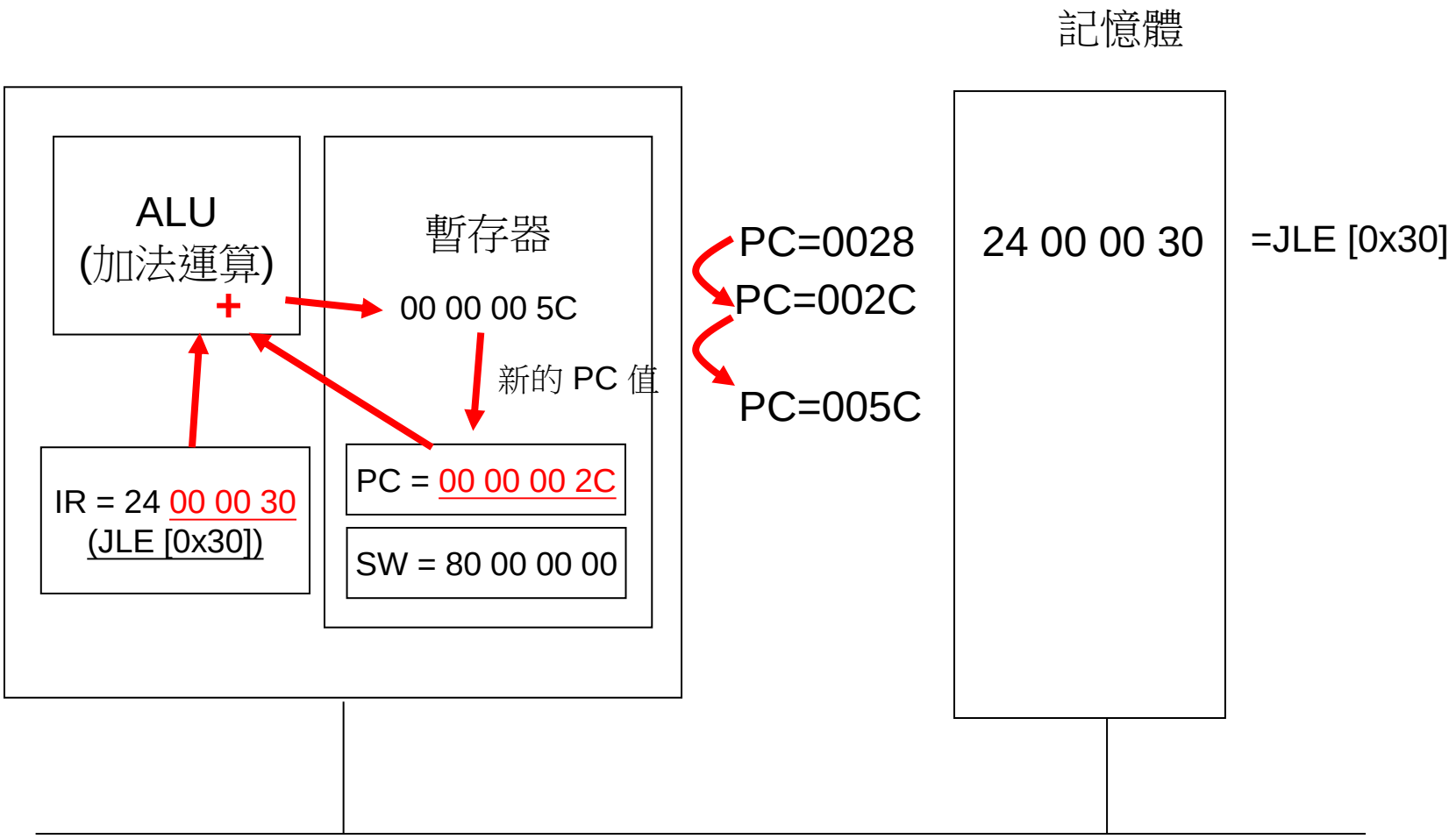


圖 2.13載入指令LD R1, [0x28] 的執行過程

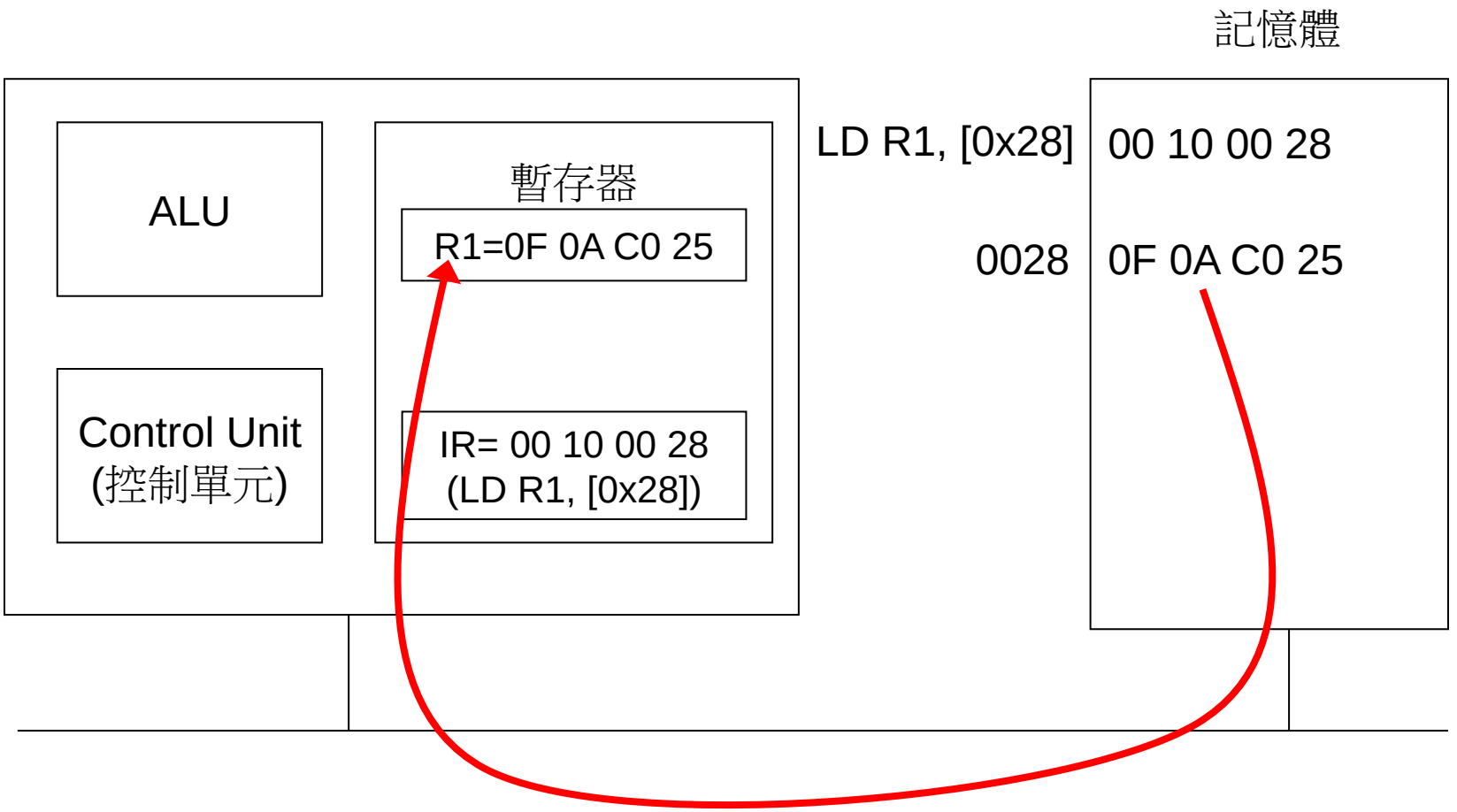
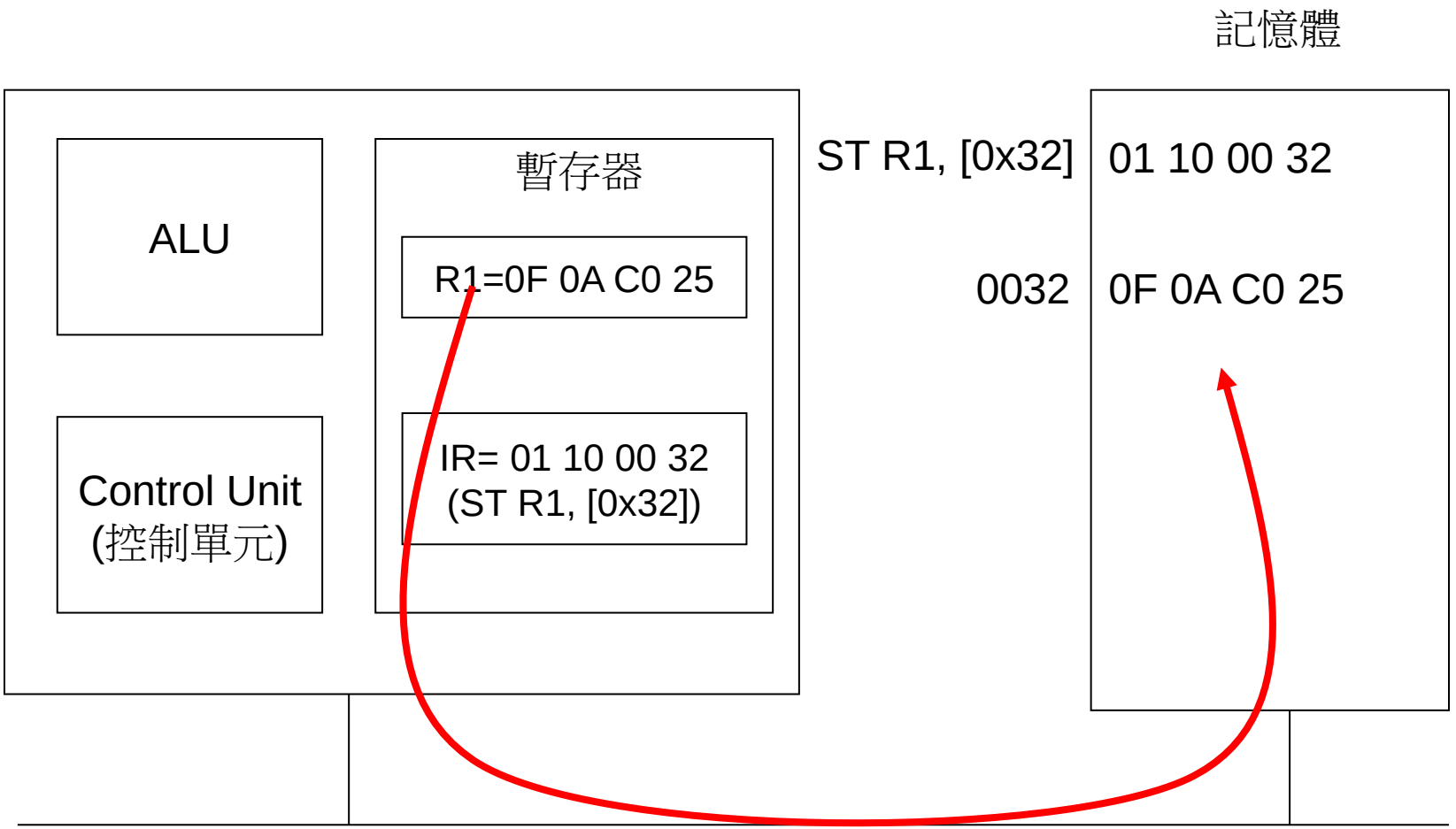


圖 2.14 儲存指令ST R1, [0x32] 的執行過程



定址模式

- CPU0 的定址模式可分為三種
  - 立即定址: LDI
  - 相對定址: LD、ST、LDB、STB
  - 索引定址: LDR、STR、LBR、SBR

立即定址

- 格式：LDI Ra, Rb+Cx

- 範例1：
  - LDI R1, R2+100
  - 說明：
    - 將 R2 + 100 的放入 R1，相當於 R1=R2+100

- 範例2：
  - LDI R1, R0+100

  - 說明：
    - R1=R0+100=100，可以簡寫為LDI R1, 100

相對定址

- 指令：LD, ST, LDB, STB

- 範例1：
  - LD R1, [R2+100]
  - 說明：
    - R1 = [R2+100], 取出記憶體位址 R2+100 的內容，放入 R1 當中。

- 範例2：
  - ST R1, [R2+100]
  - 說明：
    - 將 R1 的內容存入 R2+100 的記憶體位址中。

索引定址

- 指令：LDR, STR, LBR, SBR

- 範例 1
  - LDR R1, [R2+R3]

絕對定址

- 說明：
  - 直接存取記憶體絕對位址內容的方法
  - 範例：LD R1, [100]

- CPU0 中欲使用絕對定址法，可用 R0 作為基底
  - 範例： LD R1, [R0+100]
  - 說明：
    - 因為 R0 暫存器永遠為 0，因此上述指令相當於 LD R1, [100]。

2.4 CPU0 的程式執行

- 在整個程式的執行過程中, 指令會一個接著一個被取出後執行, 直到出現跳躍指令為止。

- 一個指令的執行可分為三個階段
  - 1. 提取
  - 2. 解碼
  - 3. 執行

三大階段：提取、解碼、執行

圖 2.16 指令執行的三大階段 - 提取、解碼、執行

```
階段 (a)：提取階段
  動作 1、提取指令          ：IR = [PC]
  動作 2、更新計數器        ：PC = PC + 4
階段 (b)：解碼階段
  動作 3、解碼              ：控制單元對 IR 進行解碼後，
                              設定資料流向開關與 ALU 的運算模式
階段 (c)：執行階段
  動作 4、執行              ：資料流入 ALU，經過運算後，流回指定的暫存器
```

圖 2.15 程式在CPU0中的執行過程

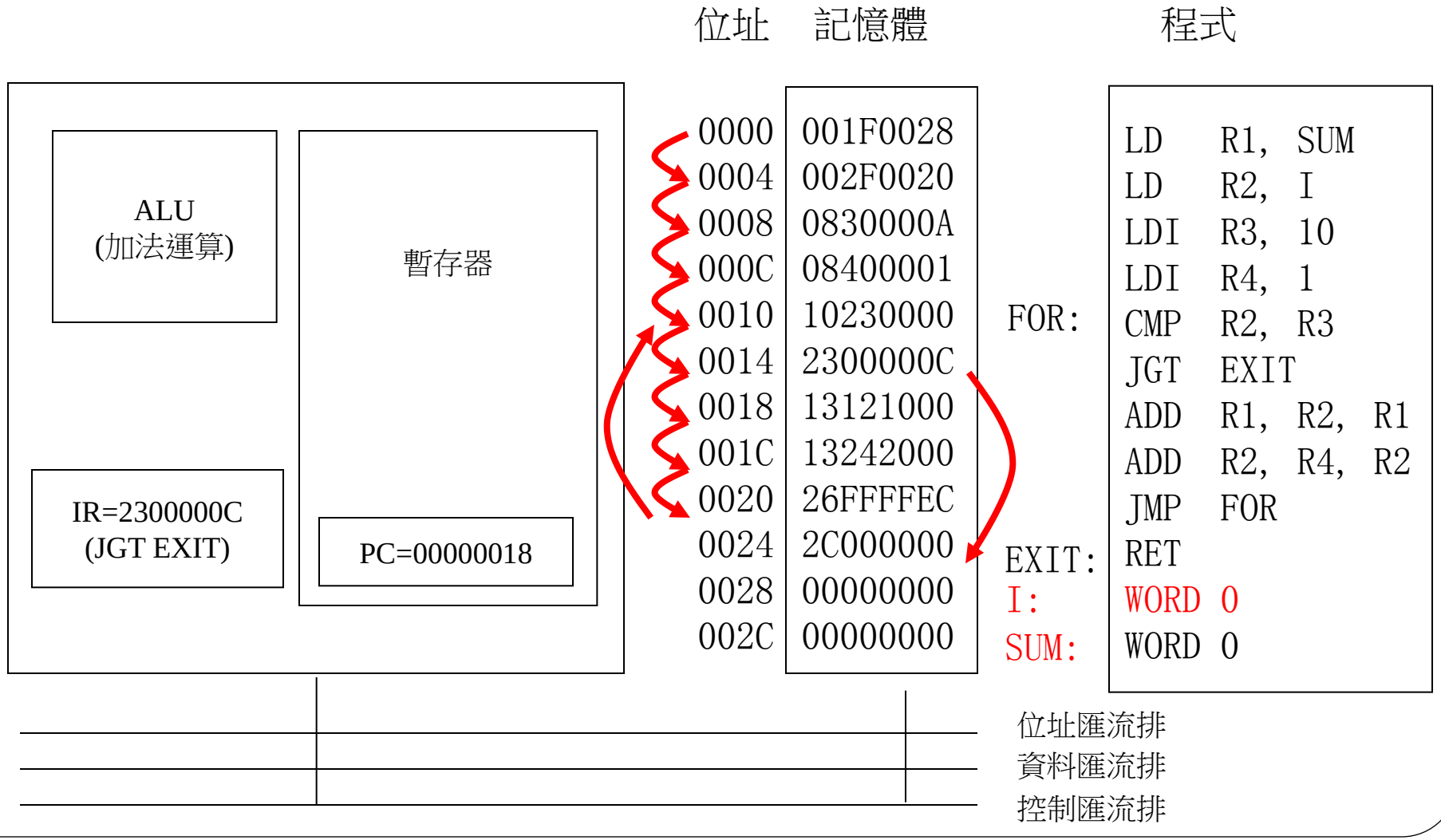
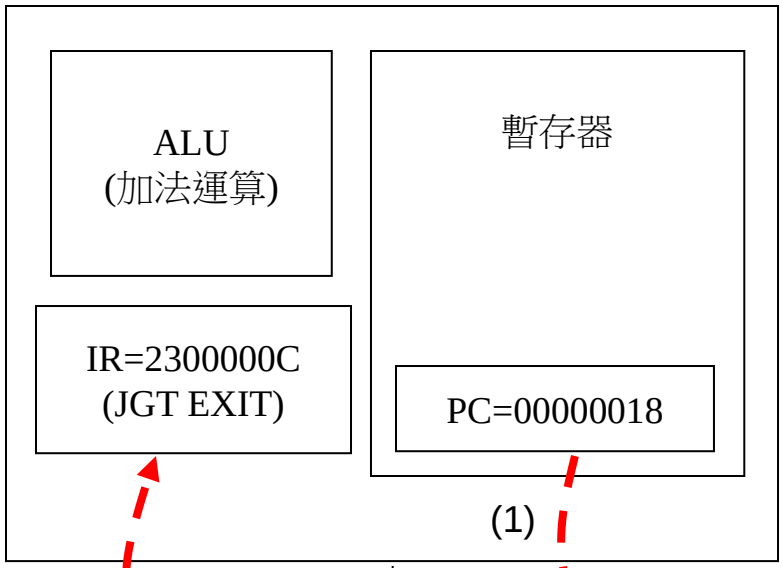


圖 2.17指令提取的細部動作

指令提取階段

(1) CPU 將 PC 傳到位址匯流排。
(2) 設定控制線，請求讀取位於 PC 的記憶體。
(3) 記憶體將指令傳到資料匯流排。
(4) CPU 取得指令碼，放入 IR 暫存器中。



位址	記憶體
0000	001F0028
0004	002F0020
0008	0830000A
000C	08400001
0010	10230000
0014	2300000C
0018	13121000
001C	13242000
0020	26FFFFEC
0024	2C000000
0028	00000000
002C	00000000

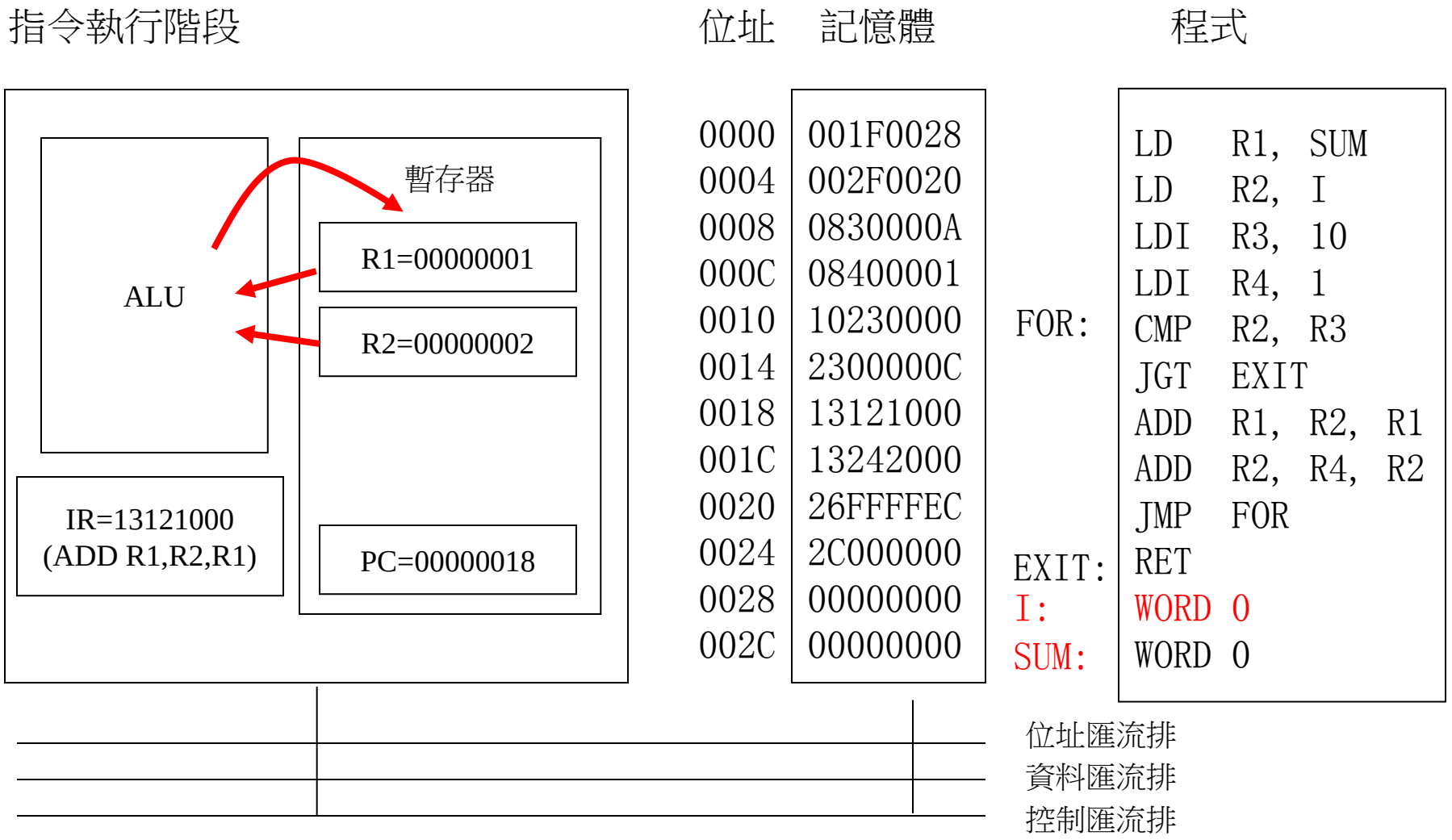
(3)

(2)

位址匯流排
資料匯流排
控制匯流排

```
        LD    R1, SUM
        LD    R2, I
        LDI   R3, 10
        LDI   R4, 1
FOR:    CMP   R2, R3
        JGT   EXIT
        ADD   R1, R2, R1
        ADD   R2, R4, R2
        JMP   FOR
EXIT:   RET
I:      WORD  0
SUM:    WORD  0
```

圖 2.18 加法指令 ADD R1, R2, R1 的執行階段動作



2.5 實務案例：IA32 處理器

- IBM PC個人電腦所用的處理器
- Intel 公司所設計的處理器
- x86 系列處理器的成員
- IA32 是相當複雜的處理器

圖 2.19 個人電腦的結構圖

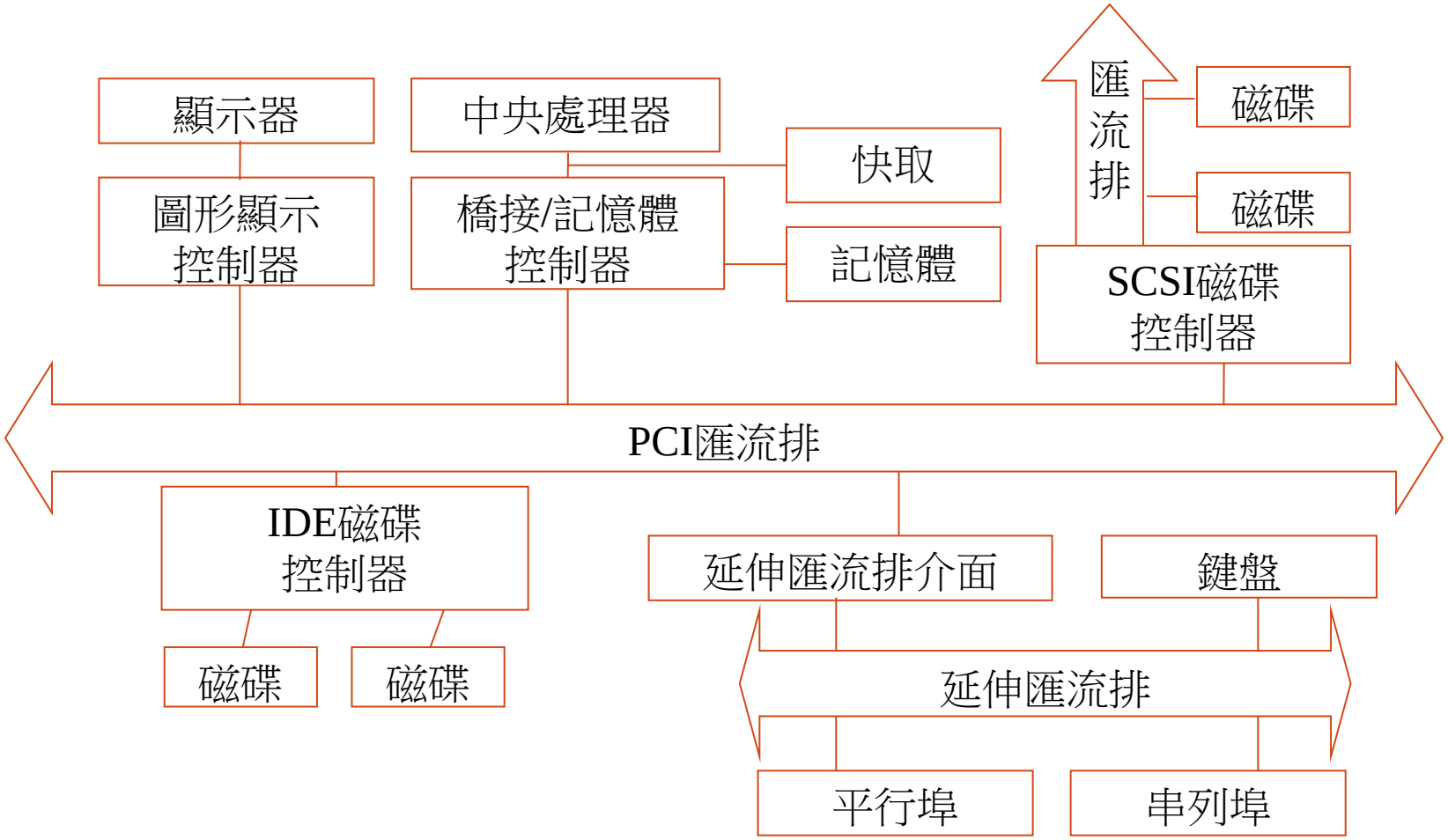


圖 2.20 IA32 的常用暫存器

通用暫存器：EAX
通用暫存器： EBX
通用暫存器： ECX
通用暫存器： EDX

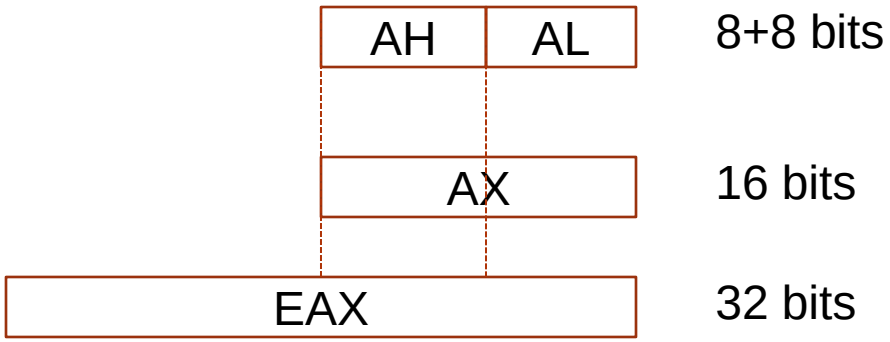
狀態暫存器： EFLAGS

程式計數器：EIP

基底暫存器：EBP
堆疊暫存器：ESP
來源指標：ESI
目的指標：EDI

程式段：CS
堆疊段： SS
資料段： DS
延伸段： ES
延伸段： FS
延伸段： GS

圖 2.21 IA32 的 EAX 暫存器



IA32 的指令格式

- 指令的參數通常可以是暫存器或記憶體，具有多種組合形式

表格 2.2：ADD 指令的六種參數組合形式

ADD reg, reg	ADD reg, mem	ADD mem, imm
ADD mem, reg	ADD reg, imm	ADD accum, imm

表格 2.3：MOV 指令的九種參數組合形式

MOV reg, reg	MOV mem, reg	MOV reg, mem
MOV reg16, segreg	MOV segreg, reg16	MOV reg, imm
MOV mem, imm	MOV mem16, segreg	MOV segreg, mem16

表格 2.4：IA32 的指令分類表

類型	代表指令	屬於本類型的指令
運算	ADD	ADD, SUB, ADC, MUL, DIV, IDIV, IMUL, SBB
邏輯	AND	AND, OR, NOT, XOR, NEG
位元	BT	清除：CLC, CLD, CLI, CMC 設定：SET, STC, STD, STI 測試：BT, BTC, BTR, BTS, TEST
副程式	CALL	CALL, RET, RETN, RETF
轉換	CBW	CBW, CDQ, CWD
比較	CMP	CMP, CMPS, CMPSB, CMPSW, CMPSD, CMPXCHG
字元調整	AAA	AAA, AAD, AAM, AAS, DAA, DAS
交換	SWAP	BSWAP, XCHG, XADD, XLAT, XLATB
增減	INC	INC, DEC
框架	ENTER	ENTER, LEAVE
暫停	HLT	HLT, NOP
輸出入	IN	輸入：IN, INS, INSB, INSW, INSD, 輸出：OUT, OUTS, OUTSB, OUTSW, OUTSD
中斷	INT	INTO, IRET
跳躍	JMP	JA, JNA, JAE, JNAE, JB, JNB, JBE, JNBE, JG, JNG, JGE, JNGE, JL, JNL, JLE, JCXZ, JECXZ
載入儲存	LEA	LEA, LDS, LES, LFS, LGS, LSS, LAHR, LAHF, SAHF
迴圈	LOOP	LOOP, LOOPW, LOOPD, LOOPE, LOOPZ, LOOPNE, LOOPNZ
重複	REP	REP, REPZ, REPE, REPNE, PEPNZ
移動	MOV	MOV, MOVSX, MOVZX

習題

1. 請畫出馮紐曼電腦的基本架構圖。
2. 請說明暫存器在電腦中的用途？
3. 請說明控制單元在電腦中的用途？
4. 請說明ALU在電腦中的用途？
5. 請問CPU0有哪些暫存器？並說明
6. 請問CPU0 的指令可分為哪幾類？並且以範例說明每一類指令的功能？
7. 請問CPU0 當中有哪些定址方式，並以範例加以說明？
8. 請說明 CPU0 程式的執行原理，並說明指令暫存器與程式計數器在程式執行時的作用？
9. 請畫出 ADD R5, R6, R1 指令的資料流向圖，並說明該指令的運作方法。
10. 請畫出 LD R5, [480] 指令的資料流向圖，並說明該指令的運作方法。
11. 請簡要說明 IA32 處理器的特性？