邏輯系統實習
實驗十：期末專題

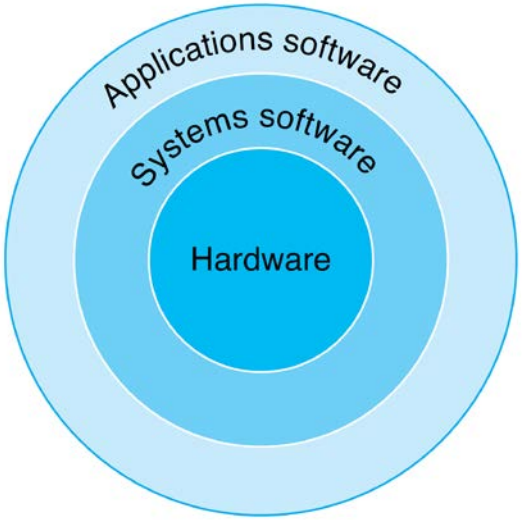
期末專題 : Single cycle ARM32指令集架構實現

國立成功大學 電機系

2014

電腦系統簡介(1/2)

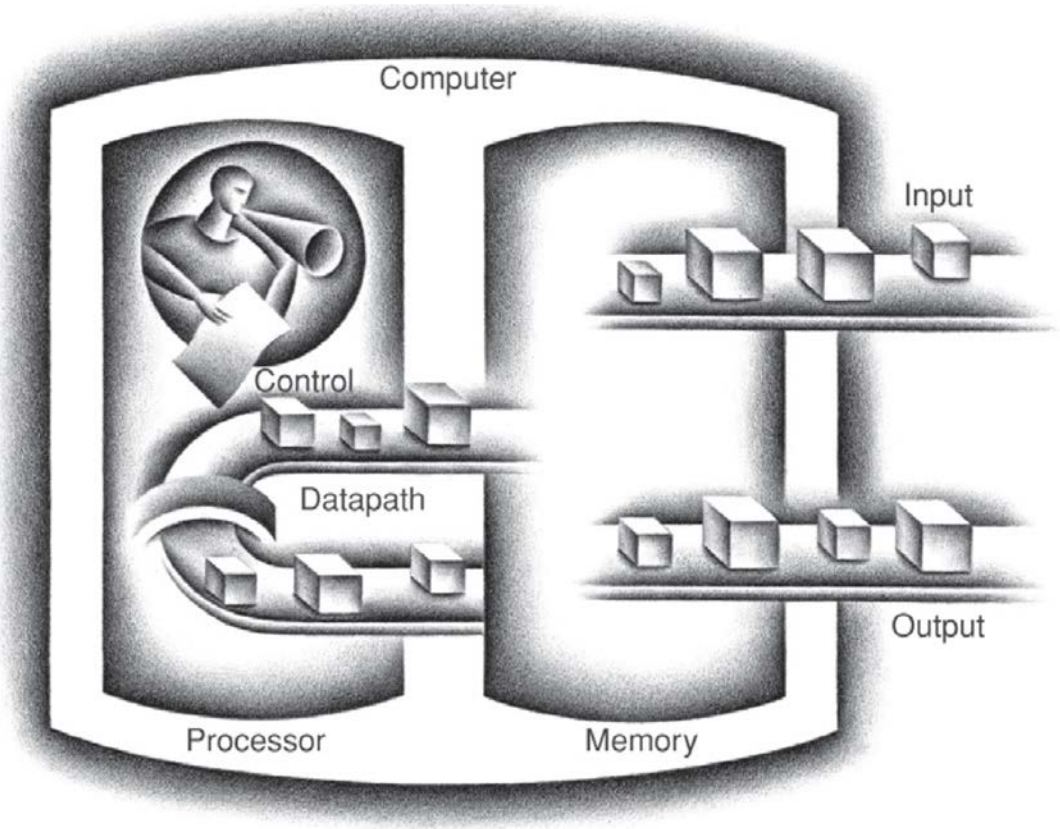
- 電腦系統可簡單分為三層:
  - Applications software :
    - 我們日常使用的應用軟體。Ex : word、Powerpoint…
  - Systems software :
    - 管理和溝通硬體的程式。 Ex : windows、driver…
  - Hardware :
    - 執行指令計算或輸入輸出工具。 Ex : CPU、memory…



A simplified view of hardware and software as hierarchical layers

電腦系統簡介(2/2)

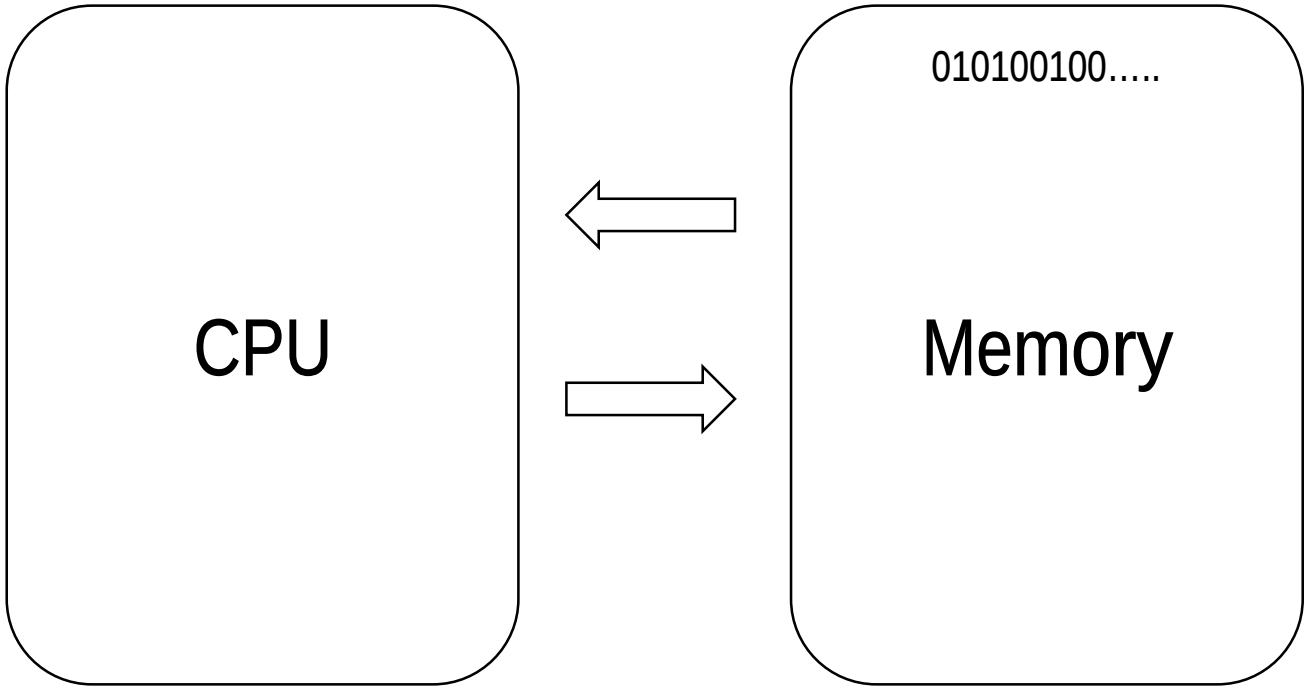
- 組成電腦的五大元素:
  - 輸入、輸出、記憶體、計算單元、控制單元。
  - 計算單元和控制單元又稱處理器。



The organization of a computer, showing the five classic components.

處理器簡介(1/3)

- CPU永遠就做一件事情:
  - 抓取指令 → 執行指令 →抓取指令→ 執行指令……



處理器簡介(2/3)

- CPU只看得懂0101…訊號，
  - 我們常打的高階語言必須轉為二進位的機器碼存入記憶體，CPU才可以抓取指令並且執行。

```
Swap(int v[], int k)
{ int temp;
    temp = v[k];
    v[k]   = v[k+1];
    v[k+1] = temp;
}
```

compiler

```
swap:
    multi $2, $5,4
    add $2, $4,$2
    lw $15, 0($2)
    lw $16, 4($2)
    sw $16, 0($2)
    sw $15, 4($2)
    jr $31
```

assembler

```
00000000101000100000000100011000
00000000100000100001000001000010
10001101111000100000000000000000
10001111000010010000000000000100
10101111000010010000000000000000
10101101111000100000000000000100
00000011111000000000000000001000
```

High-level language program (in C)

Assembly language program (for MIPS)

Binary machine language program (for MIPS)

處理器簡介(3/3)

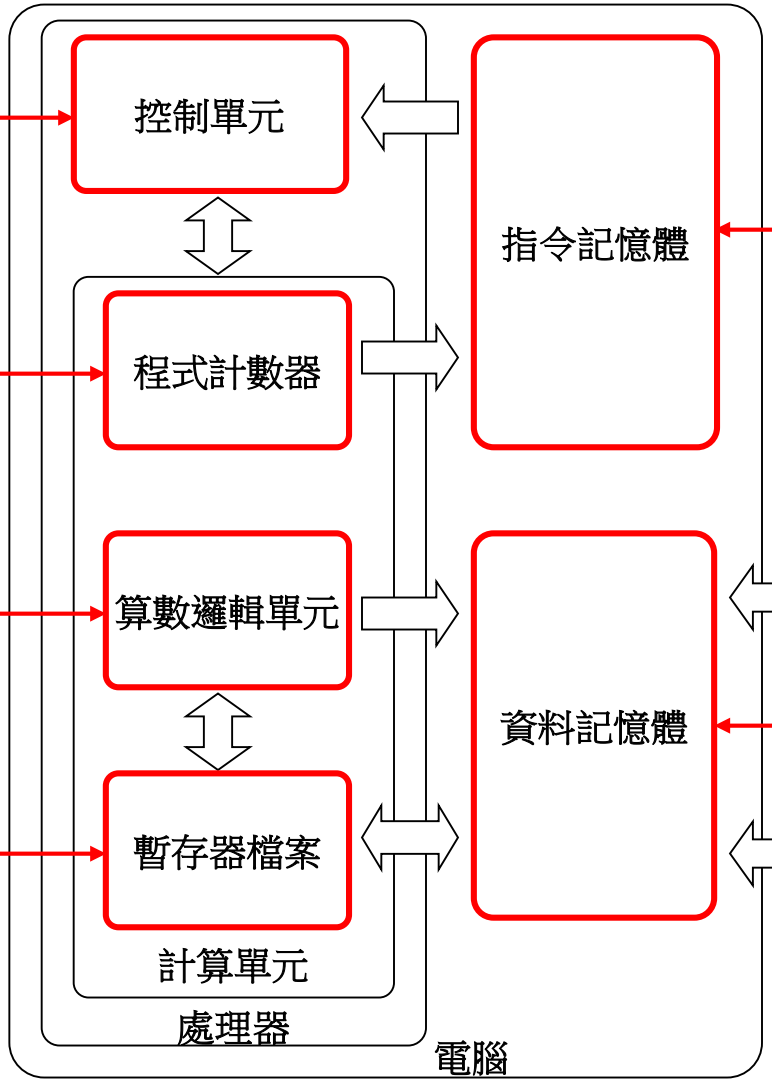
- 處理器架構

解讀指令並對資料路徑發出控制訊號

指向處理器下一個時脈要執行的指令位址

處理器中的核心計算單元，包含算數與邏輯等運算

處理器中存取資料最快的地方，用來存放暫時計算結果



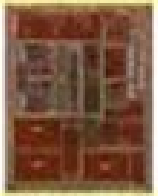
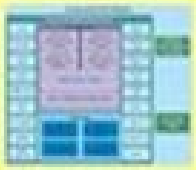
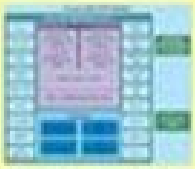
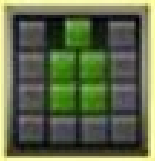
存放程式指令碼的記憶體區段

存放程式計算資料的記憶體區段

ARM微處理器概述(1/2)

- ARM（Advanced RISC Machines）
  - 是一個公司的名字，也是一類微處理器的通稱。
- 1991年ARM公司成立於英國劍橋
  - 主要出售晶片設計技術的授權。
  - 目前採用ARM技術知識產權（IP）的微處理器，即我們通常所說的ARM微處理器。
- ARM現在已遍及
  - 工業控制
  - 消費類電子產品
  - 通信網路系統
- ARM的微處理器應用約佔
  - 32位元RISC微處理器75%以上的市場比例

ARM微處理器概述(2/2)

型號	A6	Exynos 4412	Exynos 4412	APQ 8064	Tegra 3	Tegra 3 AP37
代表手機	iPhone5	SAMSUNG Note2	SAMSUNG S3	LG Optimus G	HTC ONE X	HTC ONE X+
CPU						
處理器結構						
ARM架構	Cortex A9	Cortex A9	Cortex A9	Krait A15	Cortex A9	Cortex A9
核心數	2	4	4	4	4+1	4(1.6Ghz)+1
常規主頻	1.02~1.29GHZ(未證實)	1.6GHZ	1.4GHZ	1.5GHZ	1.4GHz	1.7GHz
同步/異步	同步	同步	同步	異步	異步	異步
製程工藝	32mn	32mn	32mn	28nm	40nm	40nm
GPU	PowerVR SGX 543MP3	Mali400	Mali400	Adreno 320	Geforce 416MHz	Geforce 520MHz
L2二級緩存	1MB	1MB	1MB	2MB	1MB	1MB
支援雙通	雙通	雙通	雙通	雙通	單通	單通?
作業系統	iOS6	Android	Android	Android	Android	Android
DRAM	1GB	2GB	1GB/2GB	2GB	1GB	1GB

ARM 指令集架構簡介(1/14)

- 指令集架構是CPU所能看懂的指令格式。
  - EX：同樣ADD R2,R4,R2 (R2+R4→R2)
    - MIPS的ISA為0000_0000_1000_0010_0001_0000_0100_0010
    - ARM的ISA為1110_0000_1000_0010_0100_0000_0000_0000
- 我們主要實現ARM ISA三種TPYE指令
  - Branch Instructions
    - 改變program counter(PC)的值，即可作指令的跳躍。
  - Data Processing Instructions
    - 根據不同的opcode做對暫存器做運算。
  - Single Data Transfer instructions
    - 將記憶體的內容讀到暫存器或將暫存器的內容存到記憶體。

ARM 指令集架構簡介(2/14)

- ARM Register
  - ARM 主要擁有16個暫存器 R0~R15個暫存器和CPSR暫存器。
    - R0~R12通用資料暫存器(32bit)
    - R13 堆疊暫存器(32bit)
    - R14 連結暫存器(32bit)
      - 當執行到跳躍指令時，可以用此暫存器來儲存跳躍位置，方便將指令跳回來。
    - R15(PC) 程式計數暫存器(32bit)
      - 用來儲存指令的位置。
      - PC暫存器為特殊的暫存器
        - 若指令沒有對PC暫存器做寫入的動作，PC暫存器會自行+4。
          - 因此我們沒有將PC暫存器設計到register file內。
    - CPSR(current processor state register) 處理器狀態暫存器(4bit)
      - 用來儲存現在處理器的狀況，並不在register file內。
      - 在傳統ARM的架構為32bit，但我們這次Final project只會用到CPSR內的4bit NZCV因此只需4bit。

ARM 指令集架構簡介(3/14)

- Branch Instructions
  - 改變PC(r15)的內容即可跳躍到下一個想要執行的指令。
    - PC = (PC+4) + (24 bits signed immediate value)*4
      - PC+4+imm*4→PC
  - ISA :

    - Cond : 條件判斷，根據CPSR的NZCV決定是否執行這道指令。
    - L : Link bit
      - 判斷跳躍時是否要將CP+4的位置存入R14(link register)。
        - L=0 : R14不變 ; PC=(PC+4)+immediate*4
        - L=1 : R14=PC+4 ; PC=(PC+4)+immediate*4
    - 24 bits Signed Immediate Value : 和PC相加得到跳躍的PC位置。

ARM 指令集架構簡介(4/14)

- ARM Condition:
  - ARM 在執行每個指令前都必須經過NZCV暫存器內容來判斷要不要執行這道指令。
    - EX : ADDEQ R1,R1,R2 (R1+R2➡R1)
      - 在執行ADD前必續判斷Z=1才會去執行ADD這道指令，若Z=0則就不會執行。
  - 透過運算或比較指令去更新negative zero carry overflow(nzcv)
    - 判斷現在運算的結果是否有negative zero carry overflow(nzcv)
      - 若有則將有的狀態設為1；若無則設為0
    - EX : CMP R1,R2 (假設R1=0x00000001, R2=0x00000001)
      - 運算完的結果為R1-R2=0，因為運算結果為0，因此Zero bit就會被設置為1(其餘NCV都沒有發生引此設定為0)。
    - EX: ADDS R1,R2,R3 (R2+R3➡R1)
      - 假設R2為0xffffffff, R3為0x00000001
      - ADD後面加S就是代表會根據運算結果更改NZCV；若沒加S則代表不會更改nzcv。
      - 運算結果 R1 = 0x0000000 Z=0 ，因為有進位 C=1 (其餘NV都為0)

ARM 指令集架構簡介(5/14)

Code	Suffix	Flags	Meaning
0000	EQ	Z=1	Equal
0001	NE	Z=0	Not equal
0010	CS	C=1	Unsigned higher or same
0011	CC	C=0	Unsigned lower
0100	MI	N=1	Negative
0101	PL	N=0	Positive or zero
0110	VS	V=1	Overflow
0111	VC	V=0	No overflow
1000	HI	C = 1 and Z = 0	Unsigned higher
1001	LS	C = 0 or Z = 1	Unsigned lower or same
1010	GE	N = V	Greater or equal
1011	LT	N ≠ V	Less than
1100	GT	Z = 0 and N = V	Greater than
1101	LE	Z =1 or N ≠V	Less than or equal
1110	AL	(Ignored)	Always

Condition code

ARM 指令集架構簡介(6/14)

- Branch Instructions example:
  - BLEQ #16 (PC+4+16*4➡PC、PC+4➡R14)

Cond	101	L	24 bits Signed Immediate Value(offset)

    - 0000_101_1_00000000000000010000
    - 若Z=1 則執行PC=(PC+4)+16*4、R14 =PC+4
  - BEQ #16 (PC+4+16*4➡PC)

Cond	101	L	24 bits Signed Immediate Value(offset)

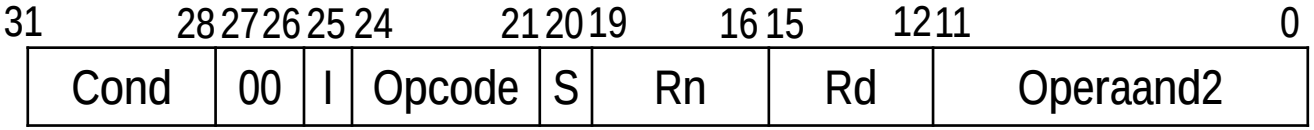
    - 0000_101_0_00000000000000010000
    - 若Z=1 則執行PC=(PC+4)+16*4
  - BGT #-4 (PC+4-4*4➡PC)

Cond	101	L	24 bits Signed Immediate Value(offset)

    - 1100_101_0_11111111111111111100
    - 若Z=0 且 N=V 則執行 PC=(PC+4)-4*4

ARM 指令集架構簡介(7/14)

- Data Processing Instructions
  - 根據不同的opcode做 Rd = Rn op Operand2
    - Rn op Op2➡Rd
  - ISA:



    - Cond : 條件判斷，根據CPSR的NZCV決定是否執行這段指令。
      - 同上頁的branch指令的condition。
    - I : Immediate bit。
      - I=0 : Operand2=register ; I=1 : Operand2=Immediate
    - S : 決定是否根據運算結果更動CPSR的NZCV。
    - Rn : Operand1。
    - Rd : Destination register。
    - Operand2 : 第二運算元

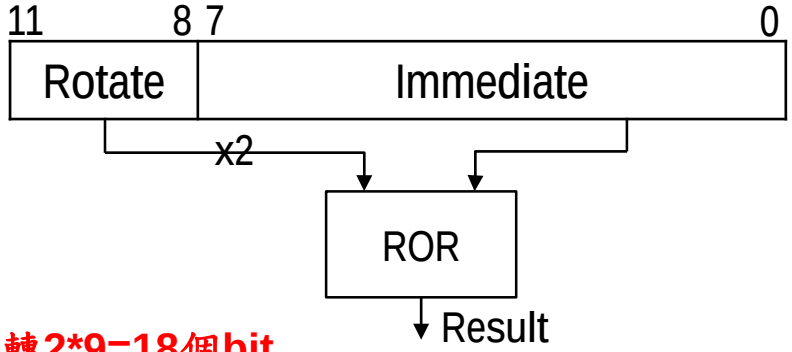
ARM 指令集架構簡介(8/14)

Assembler Mnemonic	Opcode	Action
AND	0000	Operand1 and Operand2
EOR	0001	Operand1 xor Operand2
SUB	0010	Operand 1 – Operand2
RSB	0011	Operand2 – Operand1
ADD	0100	Operand1 + Operand2
ADC	0101	Operand1+ Operand2 + carry
SBC	0110	Operand1 – Operand2 + carry -1
RSC	0111	Operand2 – Operand1 + carry -1
TST	1000	As AND, but result is not written
TEQ	1001	As XOR, but result is not written
CMP	1010	As SUB, but result is not written
CMN	1011	As ADD, but result is not written
OR	1100	Operand1 or Operand2
MOV	1101	Operand2(Operand1 is ignored)
BIC	1110	Operand1 and not Operand2 (Bit clear)
MVN	1111	not operand2 (operand1 is ignored)

opcode

ARM 指令集架構簡介(9/14)

- Immediate bit = 1 → Operand2 = #rot + 8-bit Immediate。
  - Immed_8的規範必須式8-bits可以表示，或是以向右偶次數旋轉之後可以用8-bit表式的數字。



向右旋轉2*9=18個bit

- EX: 12b'1001_1111_1111 =

  1 2 3 4 5 6 7 8 18

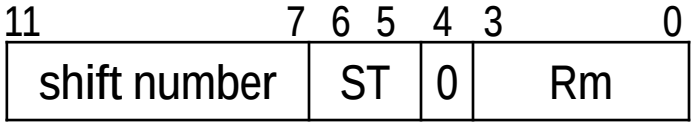
  32'b0000_0000_0011_1111_1100_0000_0000_0000
- EX: 12b'0001_1001_1001 =

  32'b0100_0000_0000_0000_0000_0000_0010_0110
- EX: 12b'1110_1111_1111 =

  32'b0000_0000_0000_0000_0000_1111_1111_0000

ARM 指令集架構簡介(10/14)

- Immediate bit = 0 → Operand2 = #shift+ST+0+Rm
  - 將Rm的內容做位移後當作運算元。



    - Shift number :
      - 位移數量的大小5bit不帶符號整數。
    - ST : shift type
      - 2'b00 : 邏輯左移(乘法)
      - 2'b01 : 邏輯右移
      - 2'b10 : 算數右移(除法)
      - 2'b11 : 旋轉右移
    - Rm :
      - 被位移的暫存器。

ARM 指令集架構簡介(11/14)

- Data Processing Instructions example:
  - ADDS R1,R2,R3 (R2+R3→R1)

Cond	00	I	Op	S	Rn	Rd	SN	St	0	Rm

    - 1110_00_0_0100_1_0010_0001_00000_00_0_0011
  - EOR R1,R2,R3>>2 (R2 xor R3→R1)

Cond	00	I	Op	S	Rn	Rd	SN	St	0	Rm

    - 1110_00_0_0001_0_0010_0001_00010_01_0_0011
  - ADD R1,R2,0x00000001 (R2+#1→R1)

Cond	00	I	Op	S	Rn	Rd	ROR	Imm

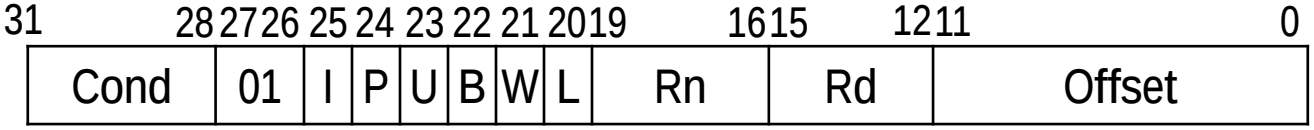
    - 1110_00_1_0100_0_0010_0001_0000_00000001
  - ADDNE R1,R2,0x01000000(R2+#64→R1)

Cond	00	I	Op	S	Rn	Rd	ROR	Imm

    - 0001_00_1_0100_0_0010_0001_0100_00000001
    - 有condition判斷，必須Z=0才執行這道指令。

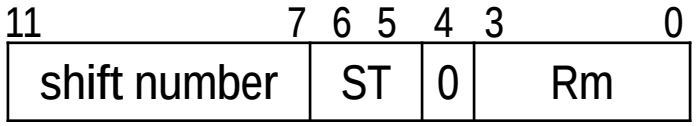
ARM 指令集架構簡介(12/14)

- Single Data Transfer instructions
  - 透過此指令做單一資料的記憶體存取。



  - Store : Rd➔MEM[Rn+Offset]；Load :MEM[Rn+Offset]➔Rd
  - Offset :
    - 用於和Base暫存器相加算出address。
      - Unsigned 12-bit immediate offset

      - Shift operation to register



  - Rd : Destination register
    - 想要儲存或讀取的暫存器。

ARM 指令集架構簡介(13/14)

- Rn : Base register
  - 用於和Offset相加算出address
- L : Load/store bit
  - L=0 : store to memory ; L=1 : load to register
- W : write-back bit (Our final project always is '0')
  - W=0 : No write-back ; W=1 : write address to base register
- B : Byte/Word bit (Our final project always is '0')
  - B=0 : transfer word quantity ; B=1 : transfer byte quantity
- U : Up/Down bit
  - U=0 : subtract offset from base ; U=1 : add offset to base
- P : Pre/Post indexing bit (Our final project always is '1')
  - P=0 : Post ,add offset after transfer ;
  - P=1 : Pre, add offset before transfer
- I : Immediate bit
  - I=1 : Offset=register ; I=0 : Offset=Immediate

ARM 指令集架構簡介(14/14)

- Single Data Transfer Instructions example:
  - ST R5,[R4+R3] (R5→MEM[R4+R3])

Cond	01	I	P	U	B	W	L	Rn	Rd	SN	St	0	Rm

    - 1110_01_1_1_1_0_0_0_0100_0101_00000_00_0_0011
  - LD R5,[R4+R3<<2] (MEM[R4+R3<<2]→R5)

Cond	01	I	P	U	B	W	L	Rn	Rd	SN	St	0	Rm

    - 1110_01_1_1_1_0_0_1_0100_0101_00010_00_0_0011
  - LD R5,[R4+#8] (MEM[R4+8]→R5)

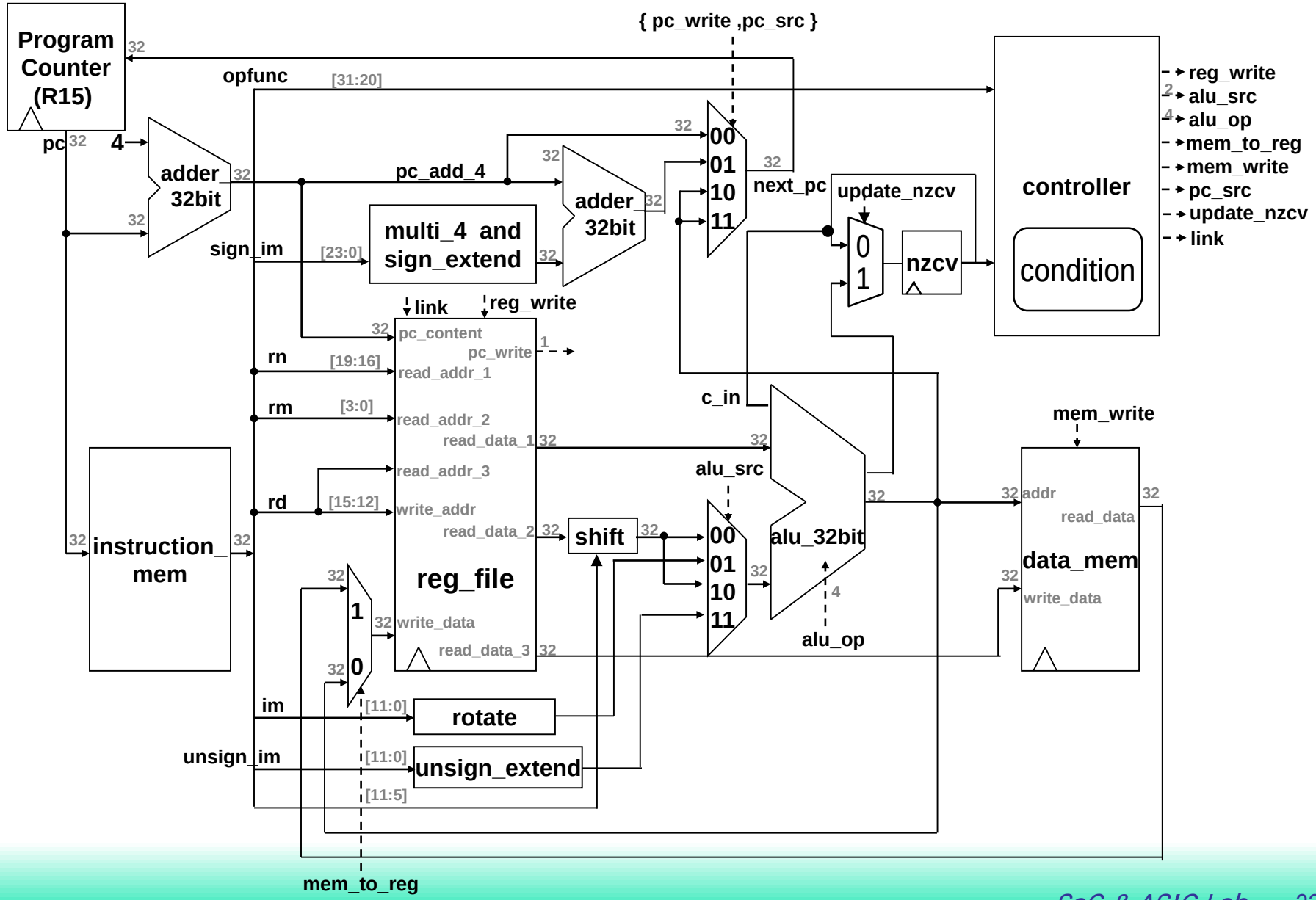
Cond	01	I	P	U	B	W	L	Rn	Rd	Immediate

    - 1110_01_0_1_1_0_0_1_0100_0101_000000001000
  - LDGT R5,[R4-#8] (MEM[R4-8]→R5)

Cond	01	I	P	U	B	W	L	Rn	Rd	Immediate

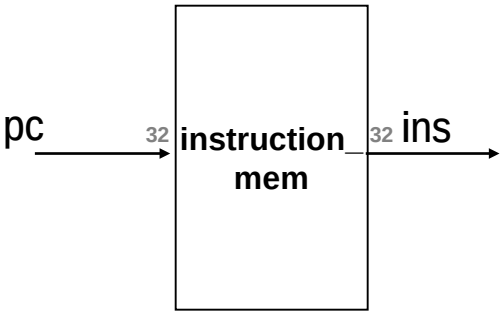
    - 1100_01_0_1_0_0_0_1_0100_0101_000000001000
    - 有condition判斷，必須Z=0且N=V才執行這道指令。

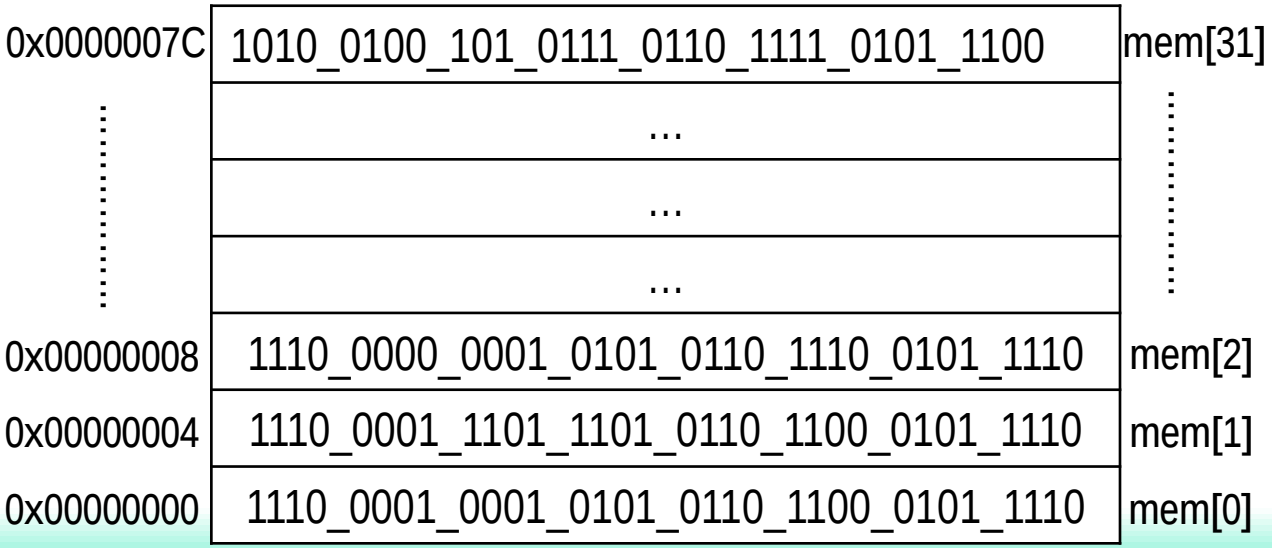
Single cycle ARM implementation(1/25)



Single cycle ARM implementation(2/25)

- Instruction memory :
  - 組合邏輯電路
    - 輸入
      - 32_bit : pc。
    - 輸出
      - 32_bit : ins
    - 合成面積考量，我們只實現存放32個指令的記憶體。
  - 將pc所選擇的記憶體位址中的一個word(32bit)大小的指令給讀出





Single cycle ARM implementation(3/25)

請務必使用ins_mem做為名子，這樣testbench才不會錯。

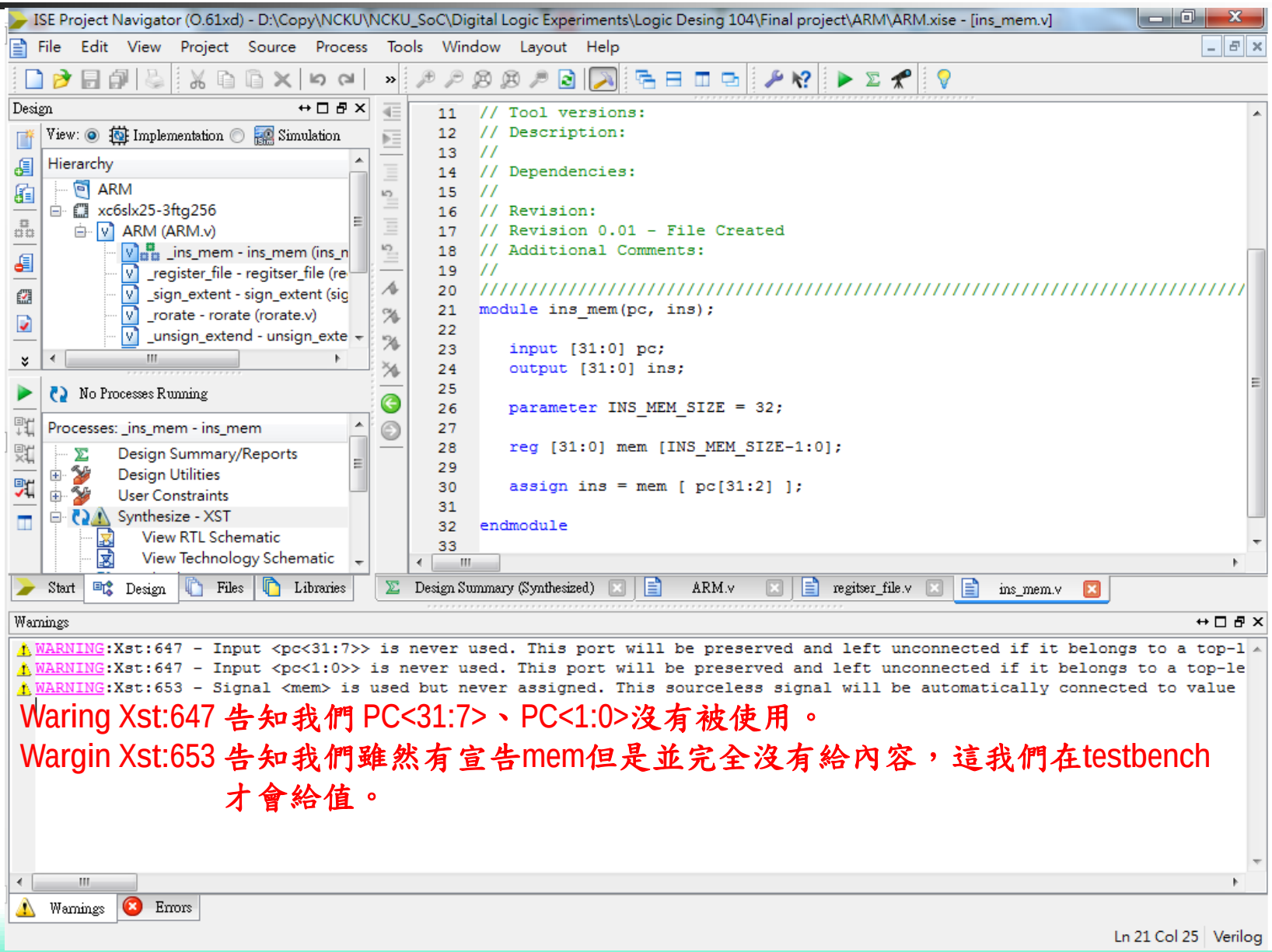
```
module ins_mem(pc, ins);

   input [31:0] pc;
   output [31:0] ins;

   parameter INS_MEM_SIZE = 32;

   reg [31:0] mem [INS_MEM_SIZE-1:0];

   assign ins = mem [ pc[31:2] ];

endmodule
```

memory的大小為32個word (1_word=32_bit=4_byte)

宣告reg二為陣列，大小為32bit的32個reg，
請務必使用mem做為名子，這樣testbench才不會錯。

因為我們ins_mem是以word(4_byte)為一單位，所以取pc[31:2]，
EX:PC=0000_0000_0000_0000_0000_0000_0000_0100 (PC=4)
則抓取mem[1]的指令；
EX:PC=0000_0000_0000_0000_0000_0000_0000_1100 (PC=12)
則抓取mem[3]的指令。

Single cycle ARM implementation(4/25)

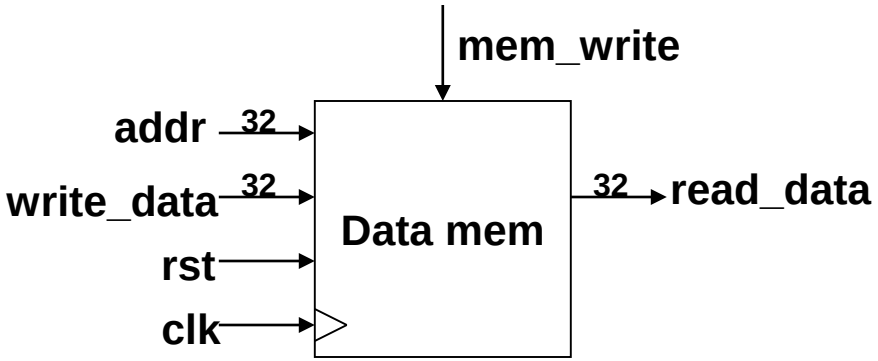


Waring Xst:647 告知我們 PC<31:7>、PC<1:0>沒有被使用。

Wargin Xst:653 告知我們雖然有宣告mem但是並完全沒有給內容，這我們在testbench才會給值。

Single cycle ARM implementation(5/25)

- Data memory :
  - 循序邏輯電路
    - 輸入：
      - 1_bit : clk, rst, mem_write
      - 32_bit : addr, write_data
    - 輸出：
      - 32_bit : read_data



    - 合成面積考量，我們只實現存放64個資料的記憶體。
  - 將資料寫入記憶體時(store word) :
    - 動作如同序向電路。
    - 當clk正緣觸發且mem_write =1時，將一個word大小的資料(write_data)給寫入addr所選擇的記憶體位址
  - 將資料讀出記憶體時(load word) :
    - 動作如同組合電路。
    - 將addr所選擇的記憶體位址中的一個word大小的資料給讀出(read_data)。

Single cycle ARM implementation(6/25)

請務必使用data_mem做為名子，這樣testbench才不會錯。

```
module data_mem(clk,rst,addr,write_data,mem_write,read_data
    );
   input clk,rst,mem_write;
   input [31:0]addr,write_data;
   output [31:0]read_data;

   parameter DATA_MEM_SIZE = 64;

   //register
   reg [31:0] mem [DATA_MEM_SIZE-1:0];

   integer i;
   assign read_data = mem[ addr[31:2] ];

   always@(posedge clk or posedge rst)
   begin
      if( rst == 1'b1 )
         for( i=0 ; i<DATA_MEM_SIZE ; i=i+1)
            mem[i] <= 0;
      else if ( mem_write == 1'b1)
         mem[ addr[31:2] ] <= write_data;
   end

endmodule
```

memory的大小為64個word (1_word=32_bit=4_byte)

宣告reg二為陣列，大小為32bit的64個reg

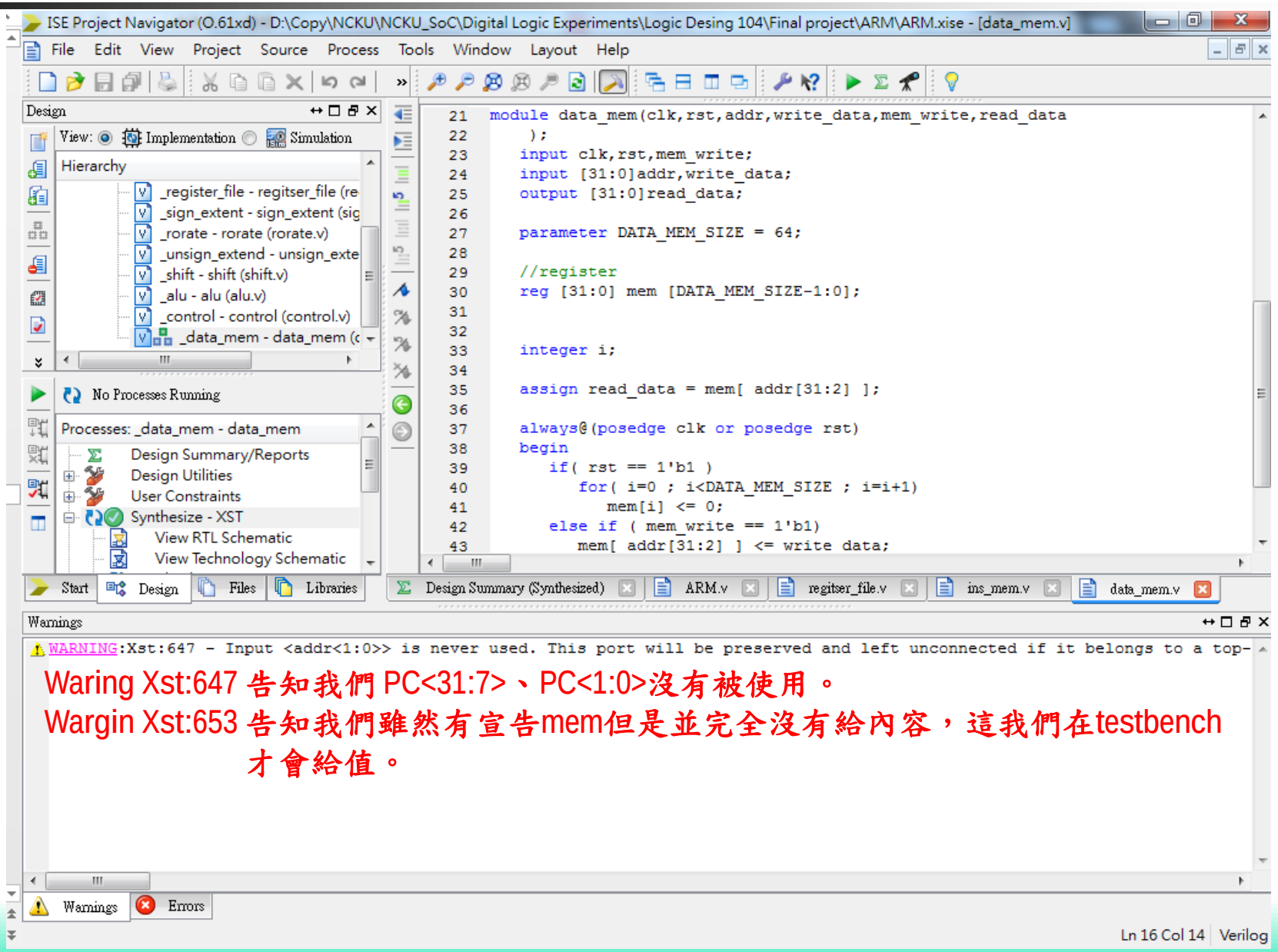
請務必使用mem做為名子，這樣testbench才不會錯。

將addr所選擇的記憶體位址中的一個word大小的資料給讀出

rst訊號對記憶體做清除

mem_write訊號對ddr所選擇的記憶體位址做寫入

Single cycle ARM implementation(7/25)

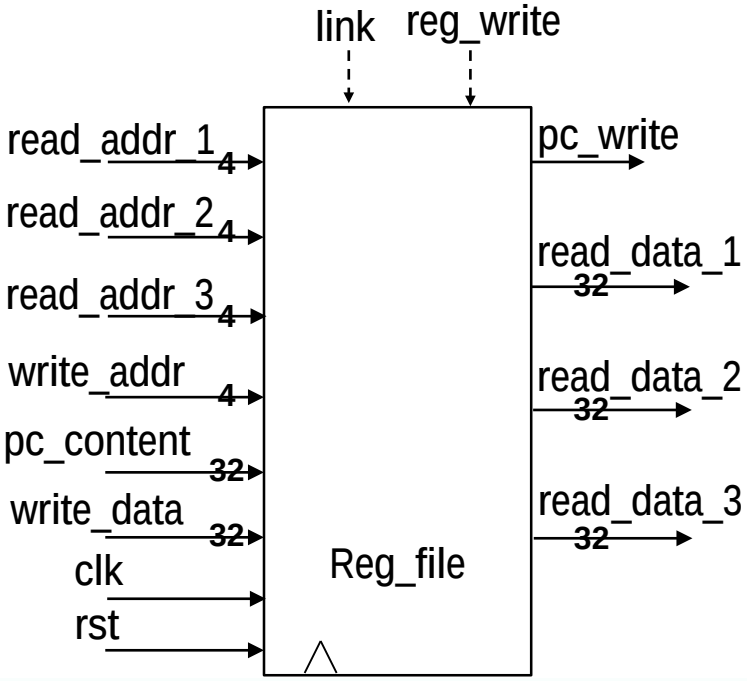


Waring Xst:647 告知我們 PC<31:7>、PC<1:0>沒有被使用。

Wargin Xst:653 告知我們雖然有宣告mem但是並完全沒有給內容，這我們在testbench才會給值。

Single cycle ARM implementation(8/25)

- register file :
  - 循序邏輯電路
  - 輸入：
    - 1_bit : clk, rst, reg_write, link
    - 4_bit : read_addr_1, read_addr_2, read_addr_3, write_addr
    - 32_bit : write_data, pc_content
  - 輸出：
    - 1_bit : pc_wirte
    - 32_bit : read_data_1, read_data_2, read_data_3



Single cycle ARM implementation(9/25)

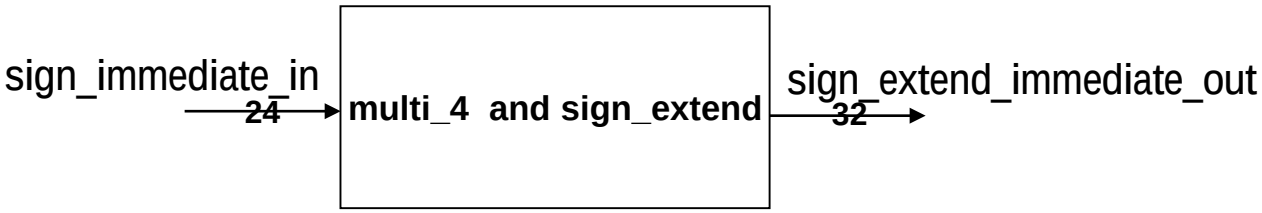
- register file 讀取行為:
  - 將read_addr_1所選擇的暫存器中的資料讀出到read_data_1 。
    - read_addr_1 可能為R0~R15，
      - R15為特別的PC暫存器，因此我們沒有設計到register file裡，
        - 若要讀取R15 則將pc_content的值輸出到read_data_1
  - 將read_addr_2所選擇的暫存器中的資料讀出到read_data_2。
    - read_addr_2 可能為R0~R15，
      - R15為特別的PC暫存器，因此我們沒有設計到register file裡
        - 若要讀取R15 則將pc_content的值輸出到read_data_2
  - 將read_addr_3所選擇的暫存器中的資料讀出到read_data_3 。
    - read_addr_3 可能為R0~R15，
      - R15為特別的PC暫存器，因此我們沒有設計到register file裡，
        - 若要讀取R15 則將pc_content的值輸出到read_data_3

Single cycle ARM implementation(10/25)

- register file 寫入行為:
  - reg_write 訊號:
    - 當reg_write = 1時，將write_data寫入write_addr所選擇的暫存器中。
      - write_addr 可能為R0~R15，
        - R15為特別的PC暫存器，因此我們沒有設計到register file裡，
          - 若要寫入R15 則將pc_write=1。(pc_write只是一條線，用組合電路描述)
    - 當reg_write=0時，則不做寫入動作。
  - link 訊號:
    - 當link = 1時，將pc_content寫入R14。
    - 當link = 0時，則不做寫入動作。

Single cycle ARM implementation(11/25)

- multi_4 and sign extend
  - 組合邏輯電路
  - 輸入：
    - 24_bit : sign_immediate_in
  - 輸出：
    - 32_bit : sign_extend_immediate_out
  - 將輸入的值乘4(左移兩bit)並且做有號數擴展到32bit
    - EX : in = 0000_0000_0000_0000_0000_0001 (#1)
      out = 0000_0000_0000_0000_0000_0000_0000_0100 (#4)
    - EX : in =1111_ 1111_ 1111_ 1111_ 1111_1100 (#-4)
      out = 1111_1111_1111_1111_11111_1111_111_0000 (#-16)

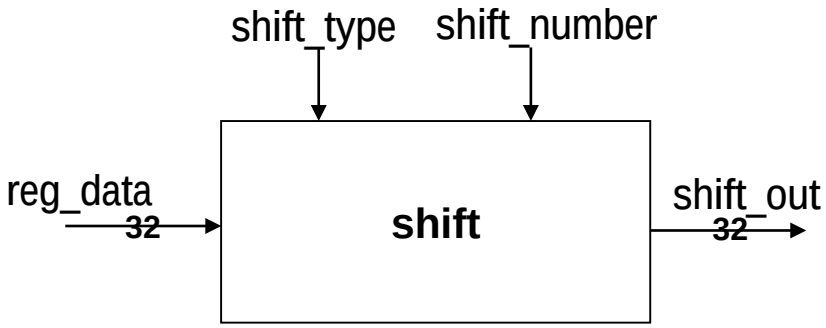


Single cycle ARM implementation(12/25)

- roratate
  - 組合邏輯電路
  - 輸入：
    - 12_bit : immediate_in
  - 輸出：
    - 32_bit : rorate_immediate_out
  - 將immediate_in[7:0]向右旋轉immediate_in[11:8]*2後輸出
    - Ex : in = 0001_1001_1001 (將1001_1001往右旋轉2個bit)
      out = 0100_0000_0000_0000_0000_0000_0010_0110
    - Ex : in = 1110_1111_1111 (將1111_1111往右旋轉28個bit)
      out = 0000_0000_0000_0000_0000_1111_1111_0000

Single cycle ARM implementation(13/25)

- shift
  - 組合邏輯電路
  - 輸入：
    - 2_bit : shift_type (位移的種類)
    - 5_bit : shift_number (位移的bit數)
    - 32_bit : reg_data (暫存器讀出來的值)
  - 輸出：
    - 32_bit : shift_out



Single cycle ARM implementation(14/25)

- 根據shift_type做四種位移的運算，根據shift_number當做要位移的量。
  - shift_type
    - 2'b00:邏輯左移 (即往左移，右邊補零，有乘法效果)
    - 2'b01:邏輯右移 (即往右移，左邊補零)
    - 2'b10:算數右移 (即往右移，左邊補signed_bit，有除法效果)
    - 2'b11:旋轉右移 (即往右移，左邊補右移過去的值)
  - Ex: shift_type = 2'b00 , shift_number = 00010, (邏輯左移2bit)

    in = 0000_000_0000_0000_0000_0000_0111_0000 (#112)

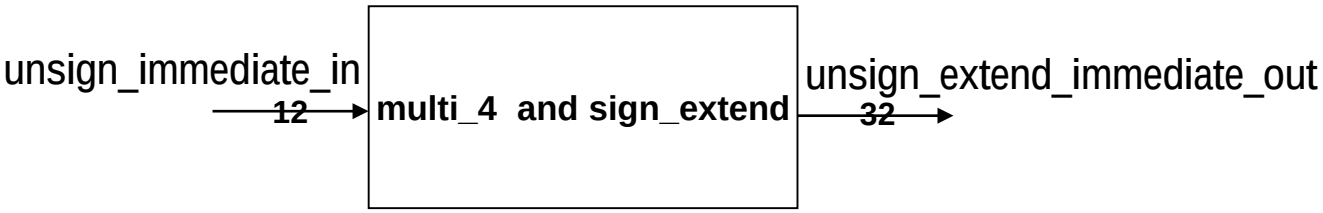
    out = 0000_000_0000_0000_0000_0001_1100_0000 (#448)
  - Ex: shift_type = 2'b10 , shift_number = 00100, (算數右移4bit)

    in = 1111_ 1111_ 1111_ 1111_ 1111_ 1111_ 1100_0000 (#-64)

    out = 1111_ 1111_ 1111_ 1111_ 1111_ 1111_ 1111_1100 (#-4)

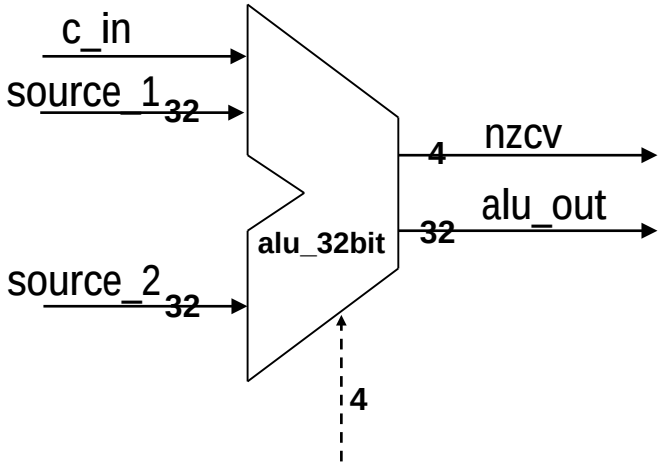
Single cycle ARM implementation(15/25)

- unsign_extend
  - 組合邏輯電路
  - 輸入：
    - 12 bit : unsign_immediate_in
  - 輸出：
    - 32 bit : unsign_extend_immediate_out
  - 將輸入做無號數擴展到32bit
    - EX: in = 1110_1100_1000
      out = 0000_0000_0000_0000_0000_1110_1100_1000
    - EX: in = 0001_1010_1011
      out = 0000_0000_0000_0000_0000_0001_1010_1011



Single cycle ARM implementation(16/25)

- alu (arithmetic logic unit)
  - 組合邏輯電路
    - 輸入：
      - 32_bit : sourc_1, source_2
      - 4_bit : alu_op
      - 1_bit : c_in
    - 輸出：
      - 4_bit : nzvc(negative zero overflow carry)
      - 32_bit : alu_out



  - 根據alu_op做alu_out = soruc_1 op ource_2的運算。
  - 根據alu_out改變nzcv。
    - alu_out == 0 則 z=1 ；alu_out !=0 則 z=1
    - alu_out = 負數 則 n=1；alu_out = 正數 則 n=0
    - Alu_out 有進位或無借位 則 c=1；alu_out 沒進位或有借位則 c=0
    - alu_out 有溢位 則 v=1；alu_out 沒溢位 則 v=0

Single cycle ARM implementation(17/25)

- alu_opcode

alu_pcode	Action
0000	Operand1 and Operand2
0001	Operand1 xor Operand2
0010	Operand 1 – Operand2
0011	Operand2 – Operand1
0100	Operand1 + Operand2
0101	Operand1 + Operand2 + carry
0110	Operand1 – Operand2 + carry -1
0111	Operand2 – Operand1 + carry -1
1000	As AND, but result is not written
1001	As XOR, but result is not written
1010	As SUB, but result is not written
1011	As ADD, but result is not written
1100	Operand1 or Operand2
1101	Operand2(Operand1 is ignored)
1110	Operand1 and not Operand2 (Bit clear)
1111	not operand2 (operand1 is ignored)

Single cycle ARM implementation(18/25)

- EX : alu_op = 0000(and) , c_in=0

source_1 = 1000_ 0000_ 0110_ 0010_ 0110_ 0000_ 0000_ 0110

source_2 = 1100_ 1110_ 0111_ 0110_ 0111_ 0000_ 0110_ 1111

alu_out = 1000_ 0000_ 0110_ 0010_ 0110_ 0000_ 0000_ 0110

nzcv = 1000

(輸出結果為負數 n=1，C=0且V=0邏輯運算沒進位或溢位)

- EX : alu_op = 0100(add) , c_in=0

source_1 = 1111_ 1111_ 1111_ 1111_ 1111_ 1111_ 1111_ 1111

source_2 = 0000_ 0000_ 0000_ 0000_ 0000_ 0000_ 0000_ 0001

alu_out = 0000_ 0000_ 0000_ 0000_ 0000_ 0000_ 0000_ 0000

nzcv = 0110

(若無號數運算是4294967295+1=0，有進位 c=1；

若有號數運算是-1+1=0，沒有overflow v=0；

alu_out 結果為0 ，zero=0)

Single cycle ARM implementation(19/25)

- EX : alu_op = 0010(sub) , c_in = 0

source_1 = 0000_ 0000_ 0000_ 0000_ 0000_ 0000_ 0000_ 0000

source_2 = 0000_ 0000_ 0000_ 0000_ 0000_ 0000_ 0000_ 0001

alu_out = 1111_ 1111_ 1111_ 1111_ 1111_ 1111_ 1111_ 1111

nzcv = 1000

(若無號數運算是0-1=-1，有借位 c=0；

若有號數運算是0-1=-1，沒有overflow v=0；)

- EX : alu_op = 0100(add) , c_in=0

source_1 = 1000_0000_0000_0000_0000_0000_0000_0000

source_2 = 1000_0000_0000_0000_0000_0000_0000_0000

alu_out = 0000_ 0000_ 0000_ 0000_ 0000_ 0000_ 0000_ 0000

nzcv = 0111

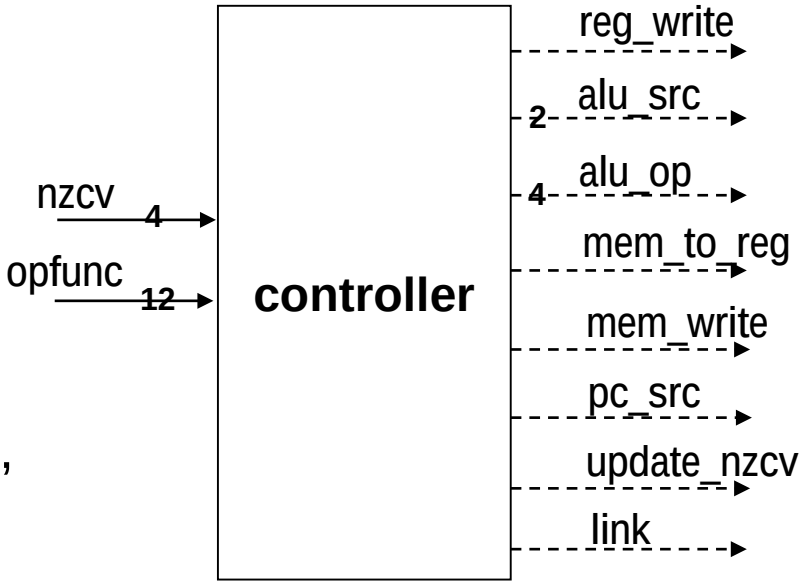
(若無號數運算是2147483648+2147483648=0，有進位 c=1；

若有號數運算是-2147483648-2147483648=0，有overflow v=1；

alu_out 結果為0 ，zero=1)

Single cycle ARM implementation(20/25)

- controller
  - 組合邏輯電路
  - 輸入:
    - 4_bit : nzcv
    - 12_bit : opfunc
  - 輸出:
    - 1_bit : reg_write, mem_to_reg, mem_write, pc_src, update_nzcv, link
    - 2_bit : alu_src
    - 4_bit : alu_op
  - 根據輸入的opfunc判斷要做輸出的控制訊號



Single cycle ARM implementation(21/25)

- 根據opfunc[11:8]比較nzcv，條件判斷成立才會執行指令。

opfunc[11:8]	條件判斷
0000	Z=1
0001	Z=0
0010	C=1
0011	C=0
0100	N=1
0101	N=0
0110	V=1
0111	V=0
1000	C=1 and Z=0
1001	C=0 or Z=1
1010	N = V
1011	N ≠ V
1100	Z=0and N = V
1101	Z set or N ≠V
1110	(Ignored)

Single cycle ARM implementation(22/25)

- 條件判斷失敗：
  - reg_write = 1'b0 ; alu_src = 2'b00 ; alu_op = 4'b0000 ;
  - mem_to_reg = 1'b0 ; mem_write = 1'b0 ; pc_src = 1'b0 ;
  - update_nzcv = 1'b0 ; link = 0;
- 條件判斷成功：
  - 根據 opfunc[7:0] 判斷是何種type的指令輸出的控制訊號。
    - Branch_type :
      - reg_write = 1'b0 ; alu_src = 2'b00 ; alu_op = 4'b0000 ;
      - mem_to_reg = 1'b0 ; mem_write = 1'b0 ; pc_src = 1'b1 ;
      - update_nzcv = 1'b0 ; link = opfunc[4];
    - Data_process_type :
      - Opfunc[4:1] = 1000 或 1001 或 1010 或 1011，則Reg_write = 1’b0;
      - Opfunc[4:1] != 1000 或 1001 或 1010 或 1011 ，則Reg_write = 1’b1;

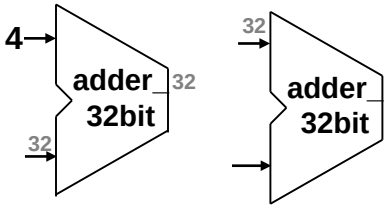
Single cycle ARM implementation(23/25)

- opfunc[5] == 0 ，則 alu_src = 2’b00
- opfunc[5] == 1 ，則alu_src = 2’b01
- alu_op = opfunc[4:1] ; mem_to_reg = 1'b0 ; mem_write = 1'b0 ;
- pc_src = 1'b0 ; update_nzcv = opfunc[0]; link = 1’b0;

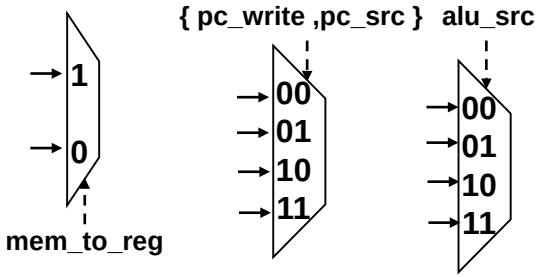
- Data transfer type:
  - reg_write = opfunc[0] ;
  - opfunc[5] == 1’b1，則alu_src = 2'b10;
  - opfunc[5] == 1’b0，則alu_src = 2'b11;
  - opfunc[3] == 1’b1，則alu_op = 4'b0100 ;
  - opfunc[3] == 1’b0，則alu_op = 4'b0010 ;
  - mem_to_reg = 1’b1;
  - mem_write = ~ opfunc[0] ; pc_src = 1'b0 ; update_nzcv = 1'b0 ; link = 1’b0;

Single cycle ARM implementation(24/25)

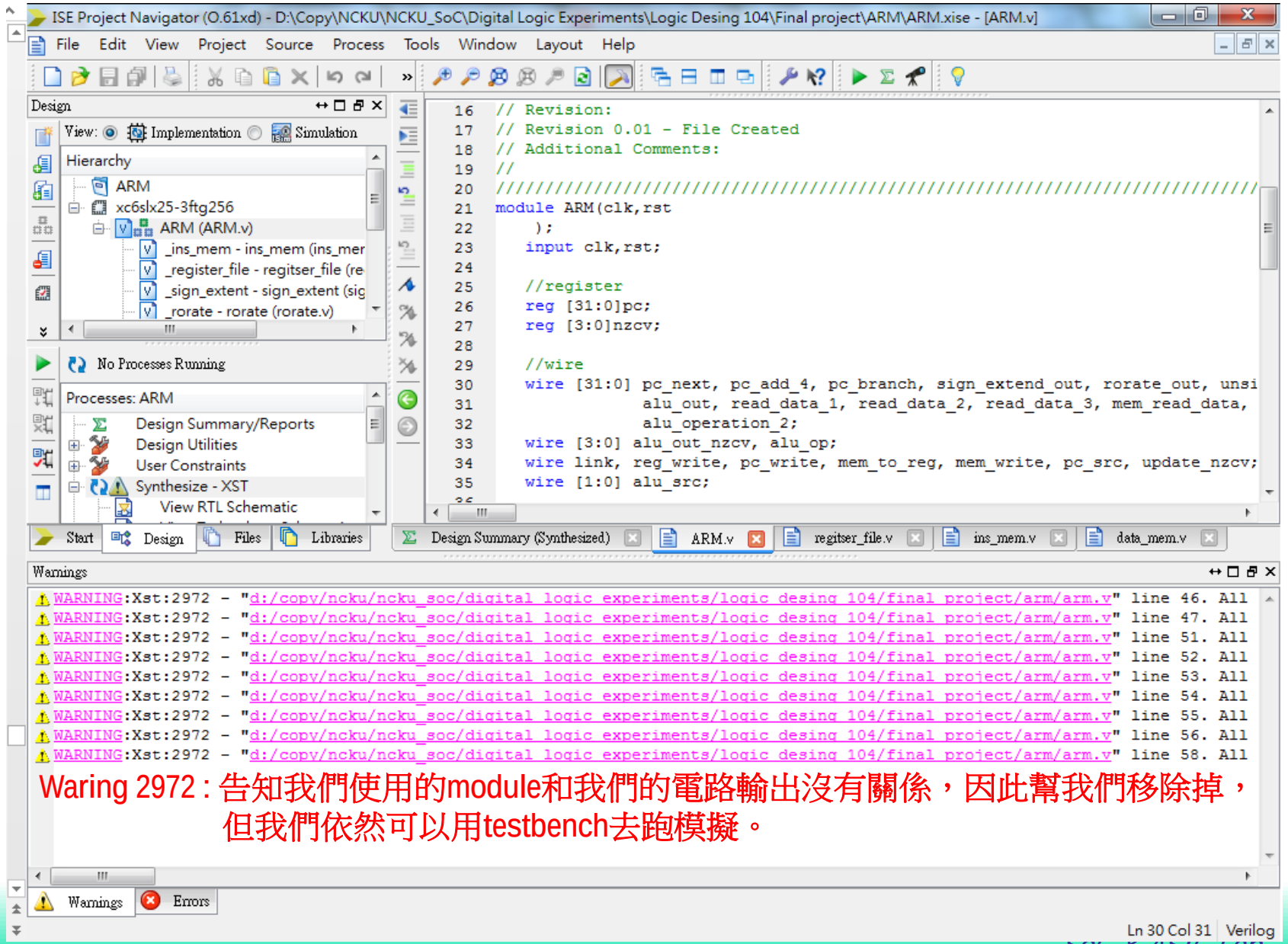
- ARM top module:
  - 除了上述9個 module外
  - 還需要兩個adder
    - assign pc_add_4 = pc+32'd4;
    - assign pc_branch = sign_extend_out + pc_add_4;



  - 三個mux
    - assign reg_write_data = mem_to_reg?(mem_read_data):(alu_out);
    - assign pc_next = pc_write? (pc_src? (alu_out):(alu_out)):
      (pc_src? (pc_branch):(pc_add_4));
    - assign alu_operation_2 = alu_src[1]?
      (alu_src[0]? unsign_extend_out):(shift_out)):
      (alu_src[0]? rorate_out):(shift_out));



Single cycle ARM implementation(25/25)

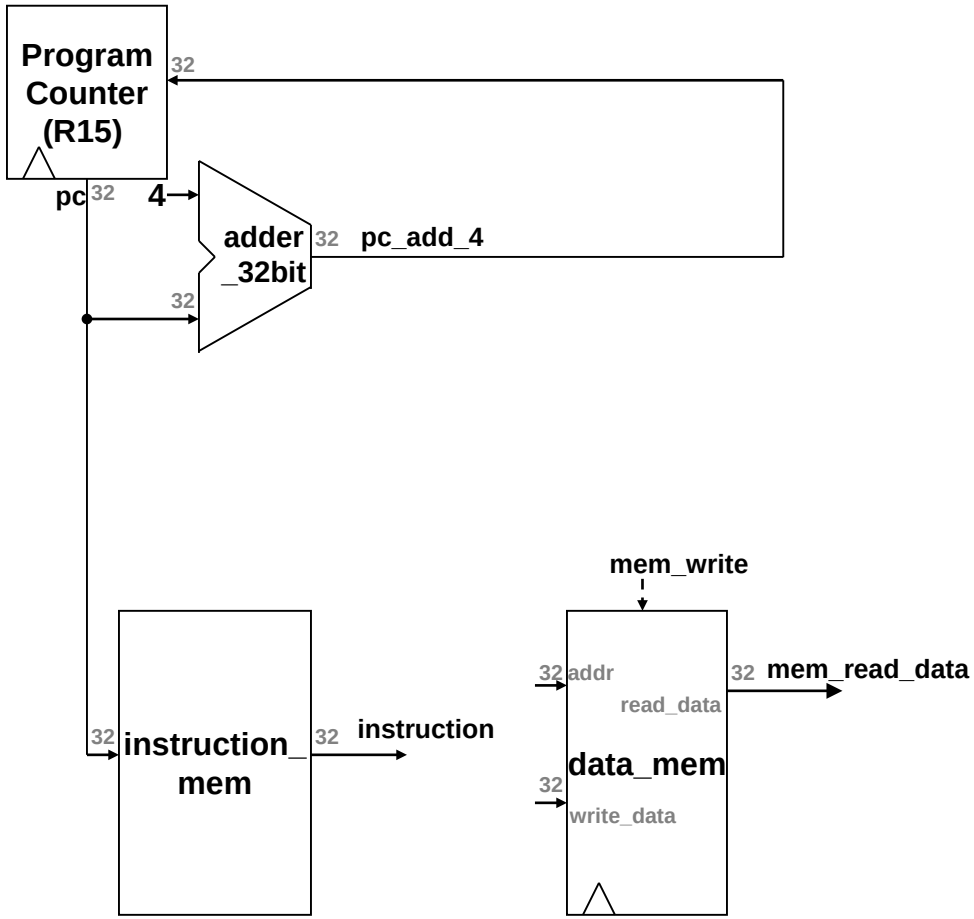


Waring 2972 : 告知我們使用的module和我們的電路輸出沒有關係，因此幫我們移除掉，但我們依然可以用testbench去跑模擬。

Final project code(1/3)

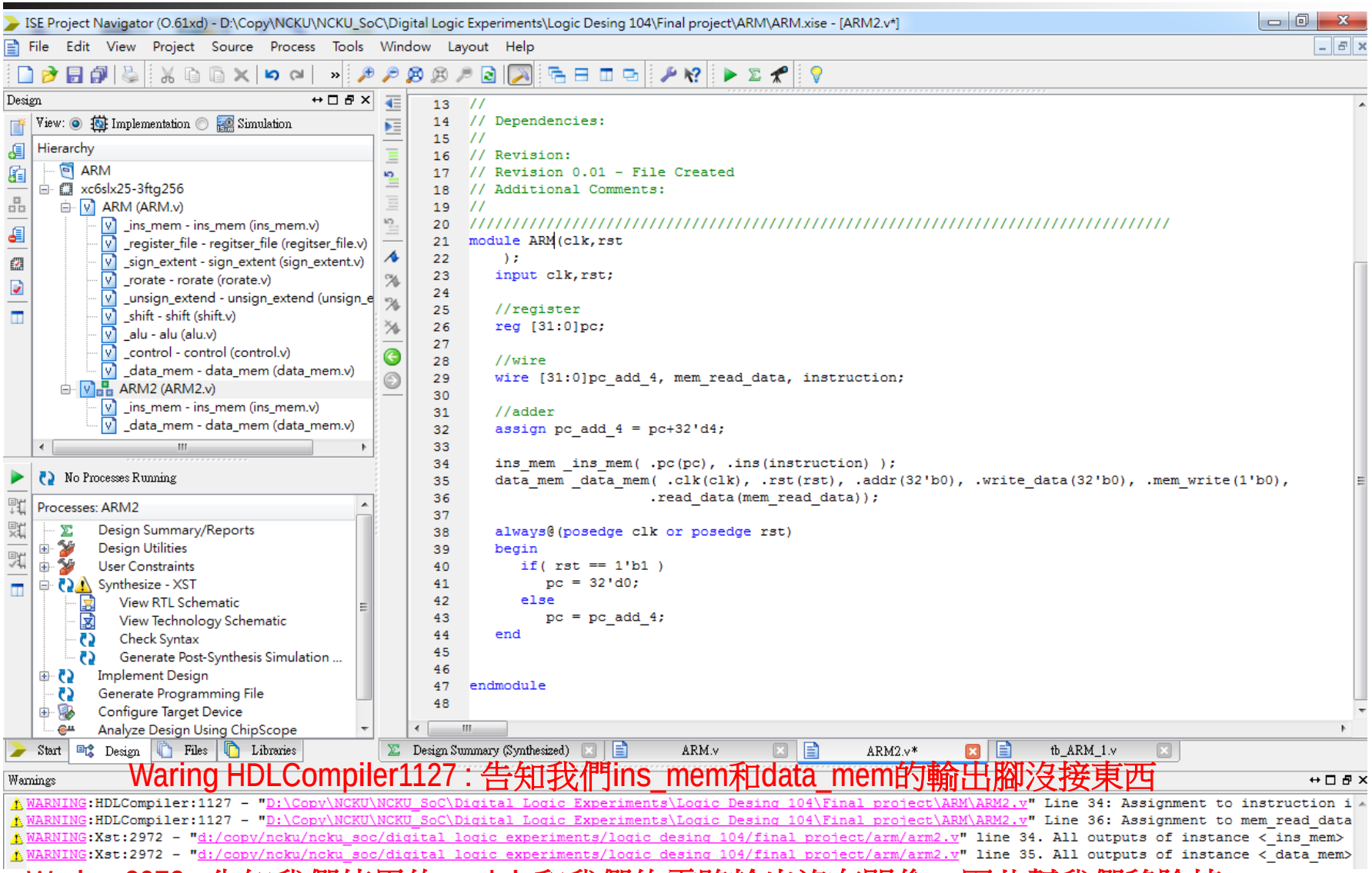
- 在FTP上面的Final project code下載testbench黨
  - ARM：請務必使用此檔案做top module。
  - tb_ARM_1~4.v：
    - test_bench_1：記憶體測試
    - test_bench_2：指令測試
    - test_bench_3：bubble sort程式
    - test_bench_4：指令測試
  - arm_tb_mem_data_1~4.txt：
    - data_memory的內容
  - arm_tb_program_1~4.txt：
    - ins_memory的內容(即是指令)
  - arm_tb_answer_1~4.txt：
    - testbench的答案(data_memory執行完後的內容)

Final project code(2/3)



ARM.v 目前已經實現的電路
(打好inst_mem和data_mem就可以直接跑testbench1)

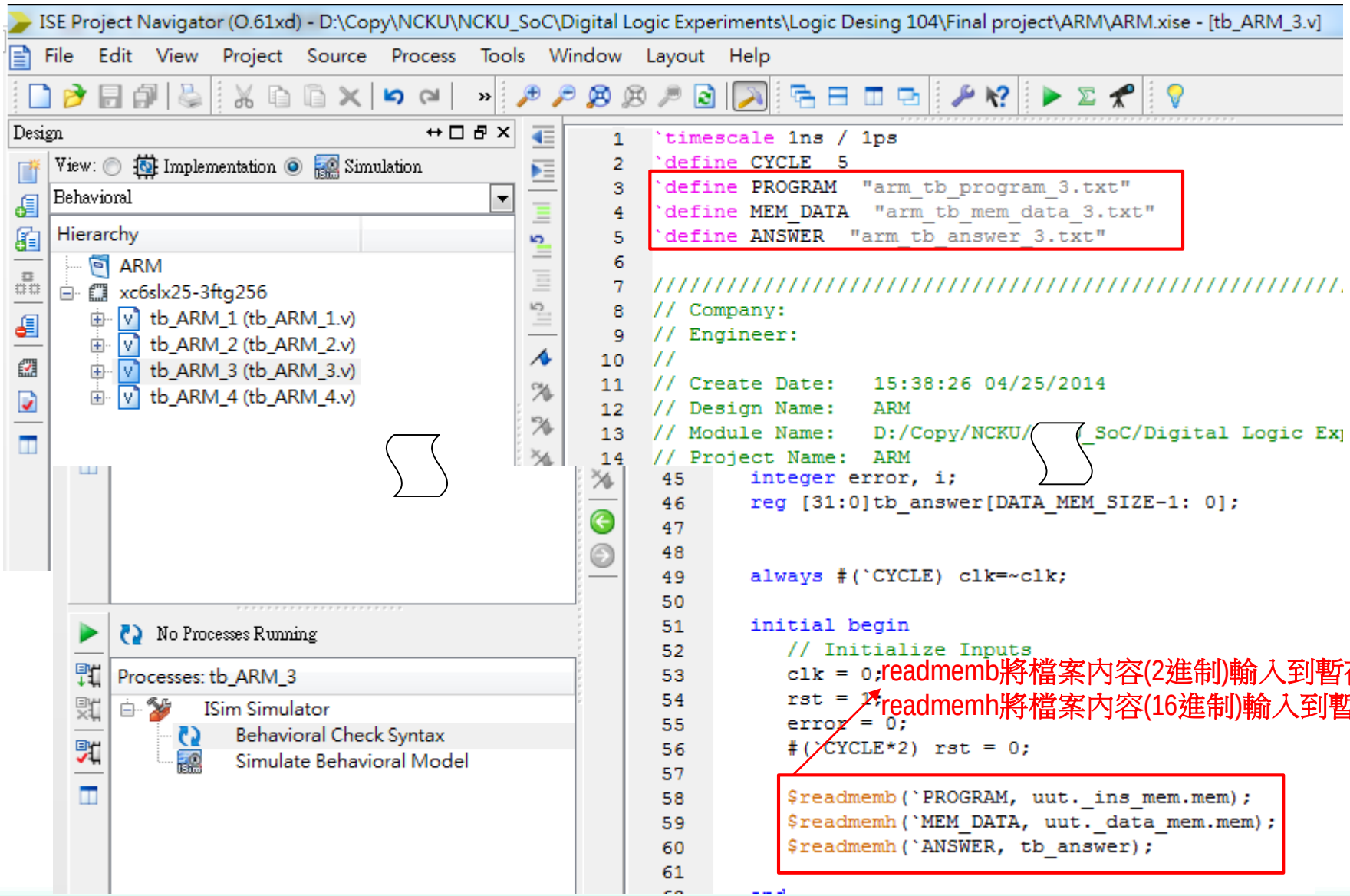
Final project code(3/3)



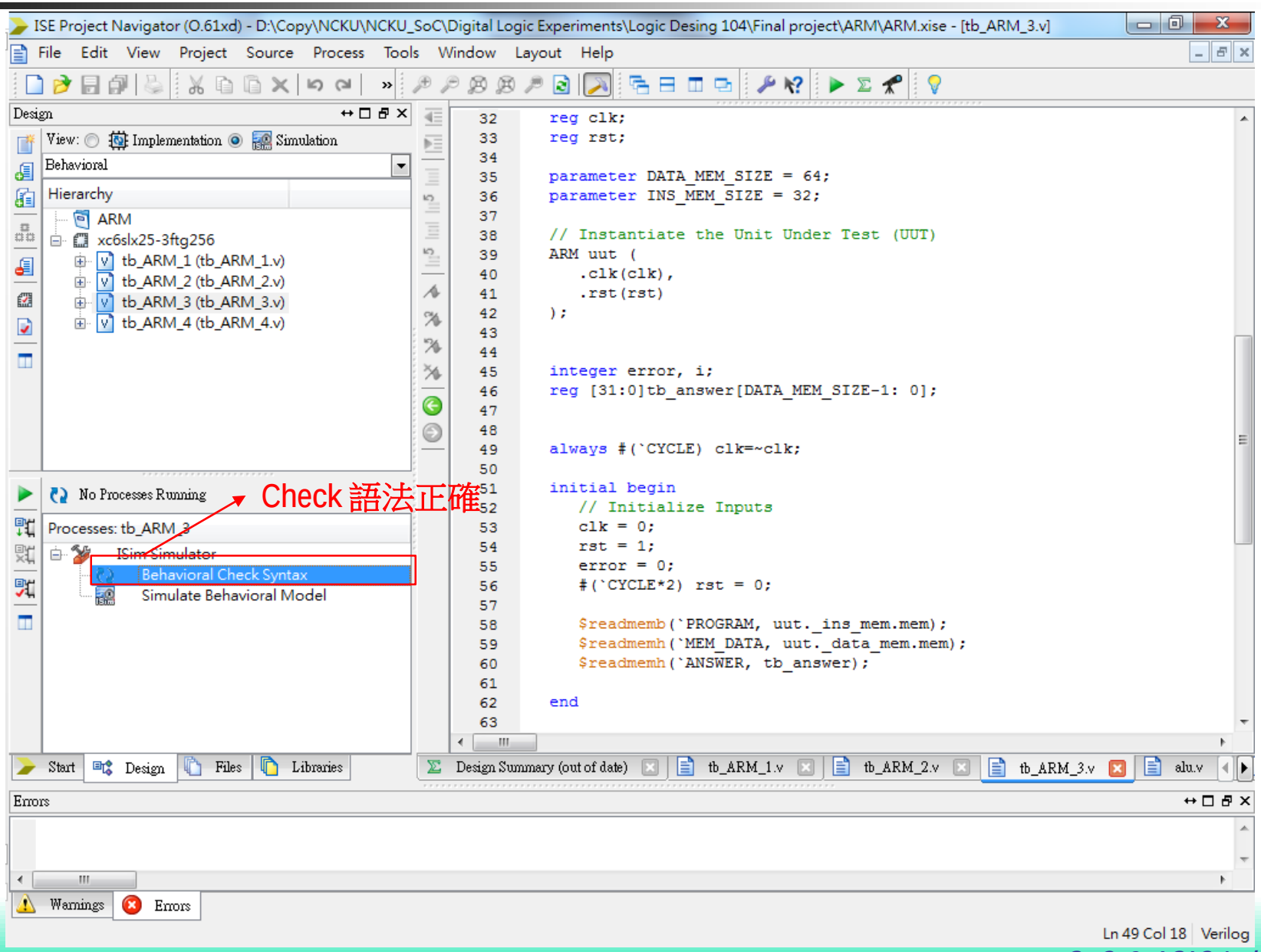
Waring HDLCompiler1127：告知我們ins_mem和data_mem的輸出腳沒接東西

Waring 2972：告知我們使用的module和我們的電路輸出沒有關係，因此幫我們移除掉，
但我們依然可以用testbench去跑模擬。

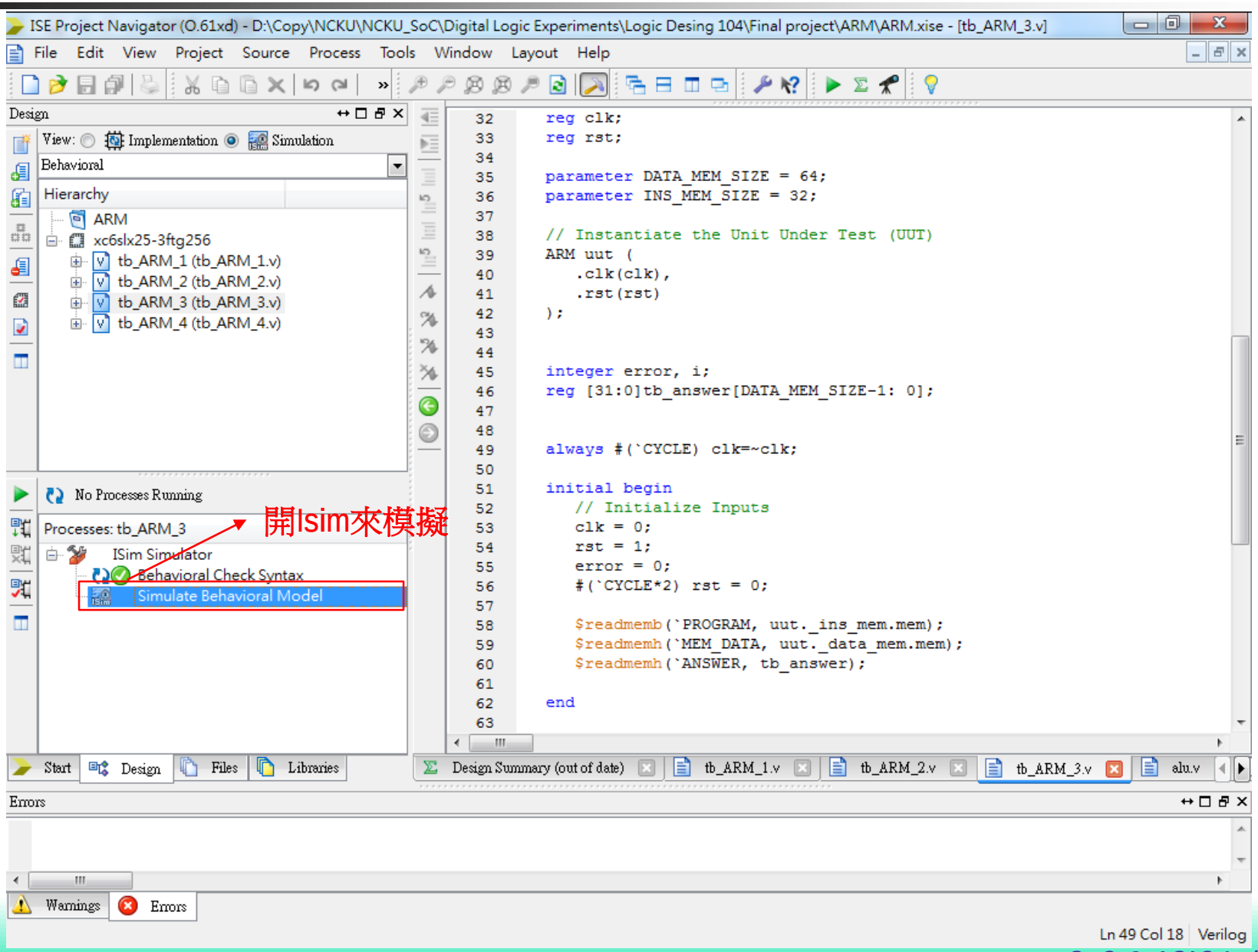
Testbench for ARM(1/7)



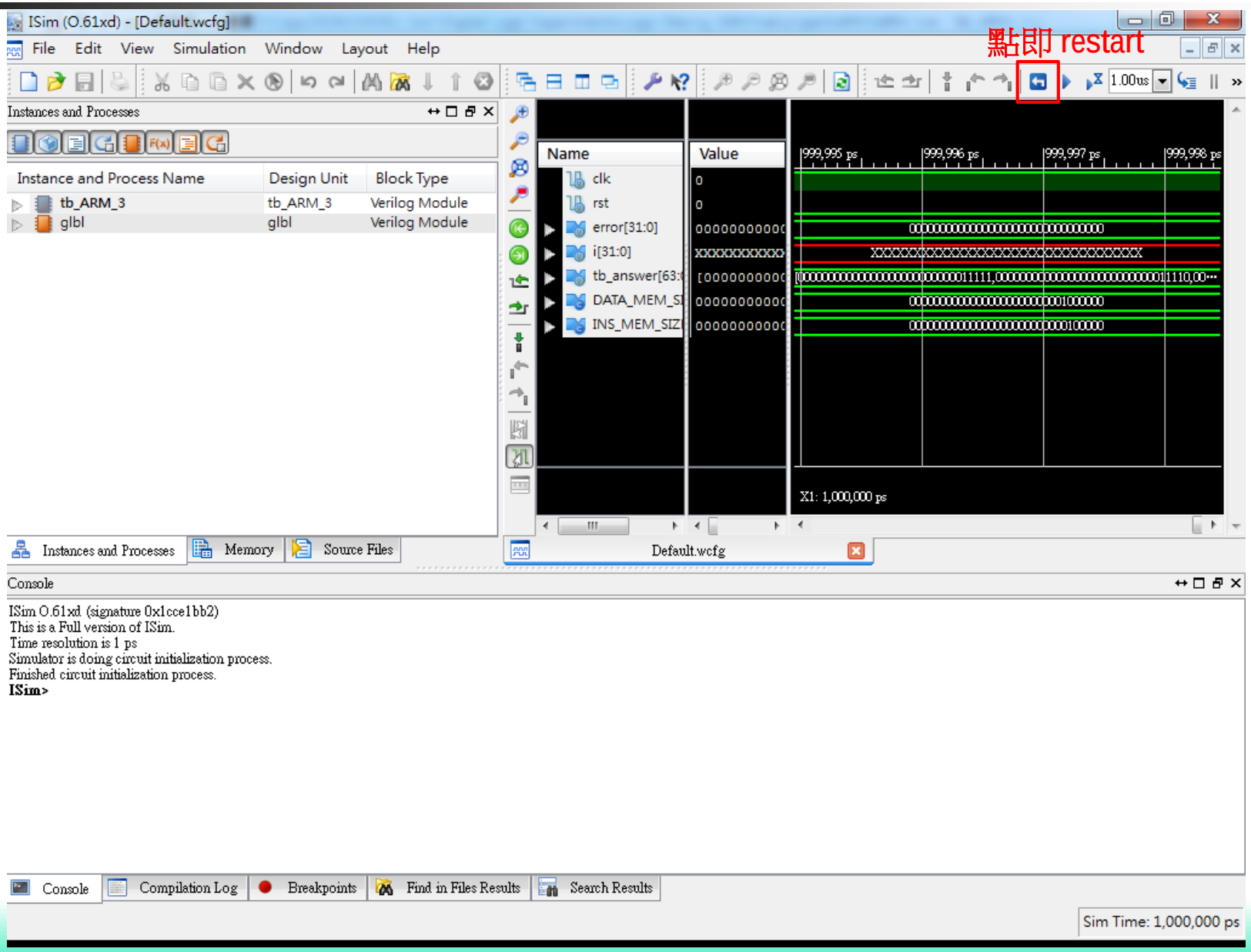
Testbench for ARM(2/7)



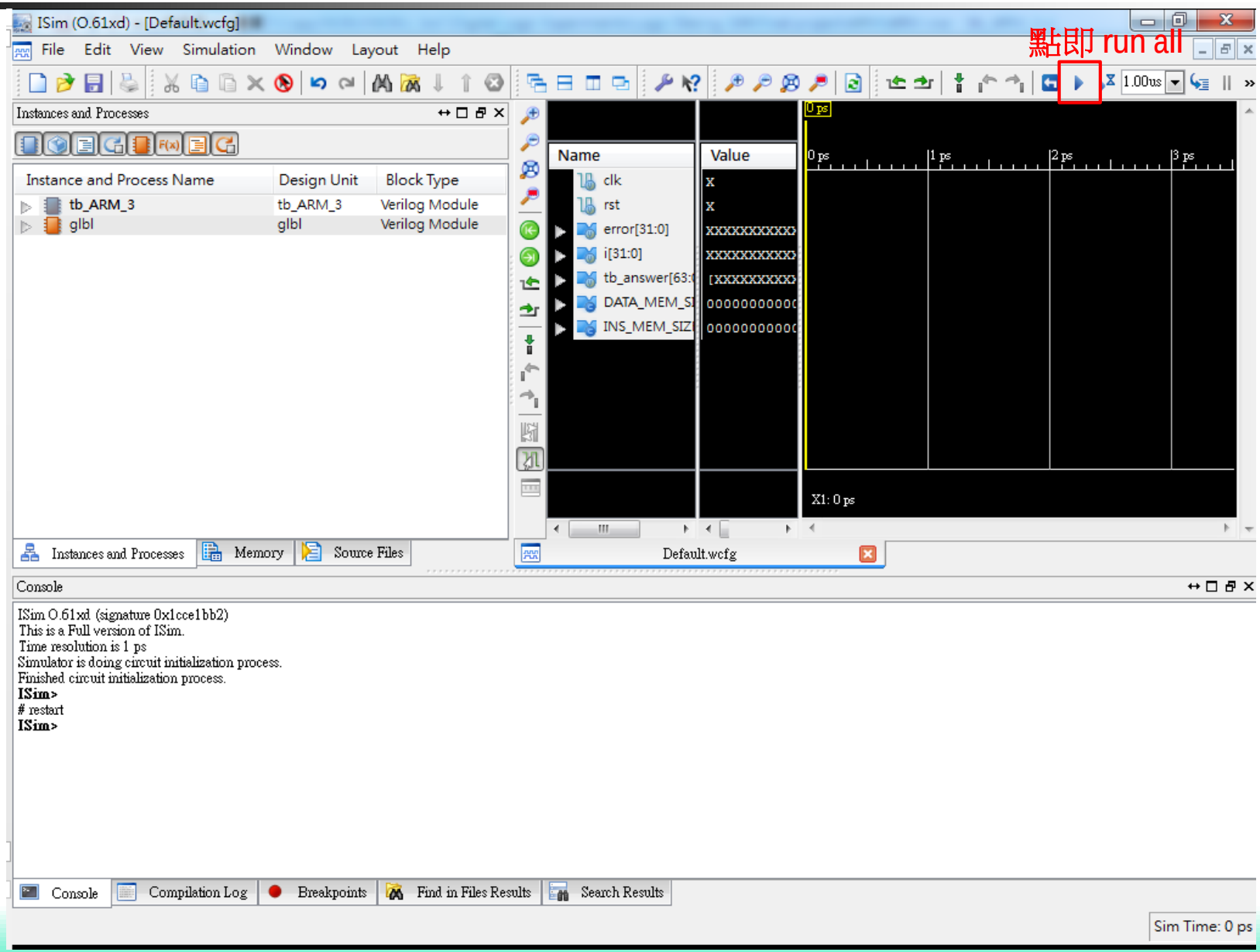
Testbench for ARM(3/7)



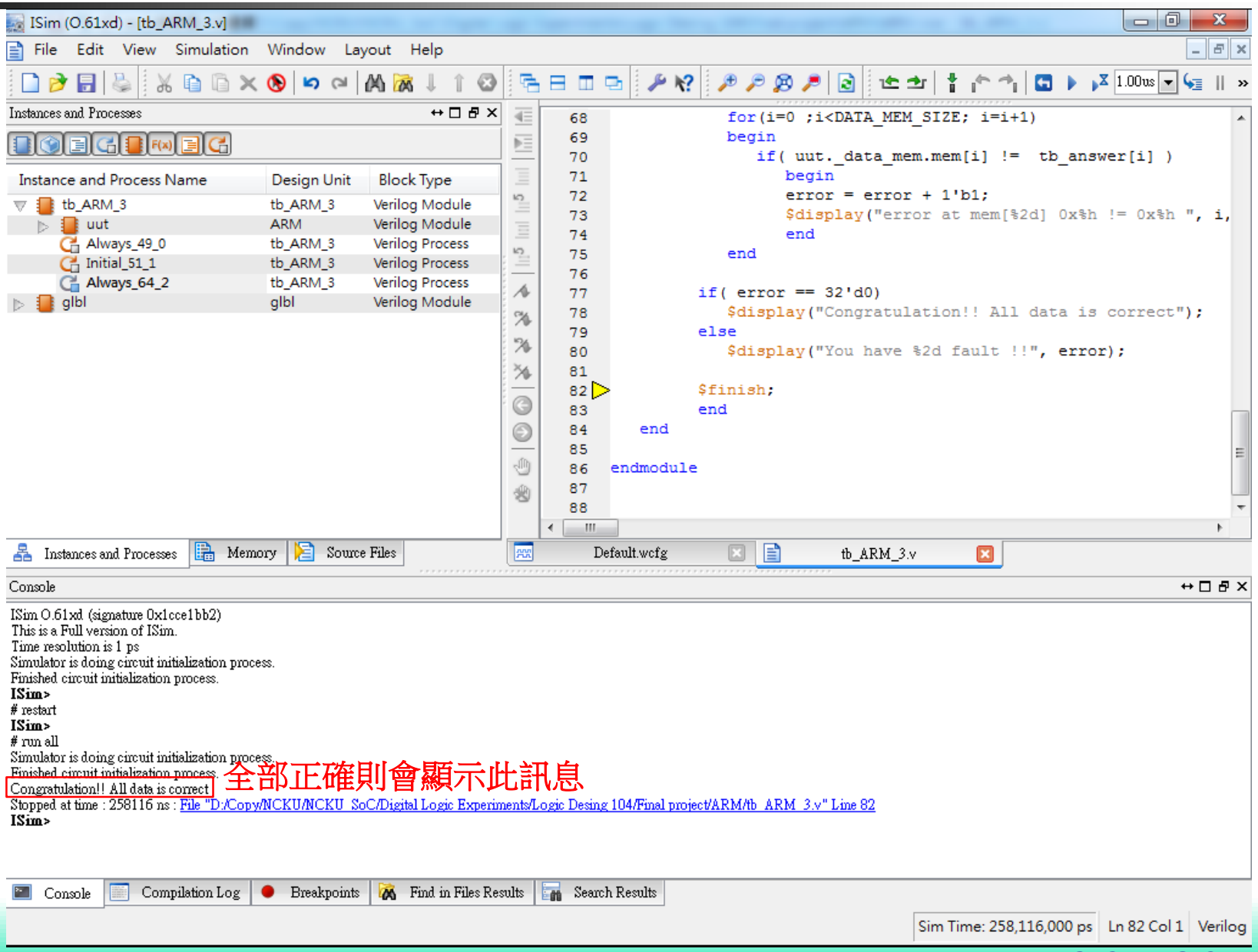
Testbench for ARM(4/7)



Testbench for ARM(5/7)

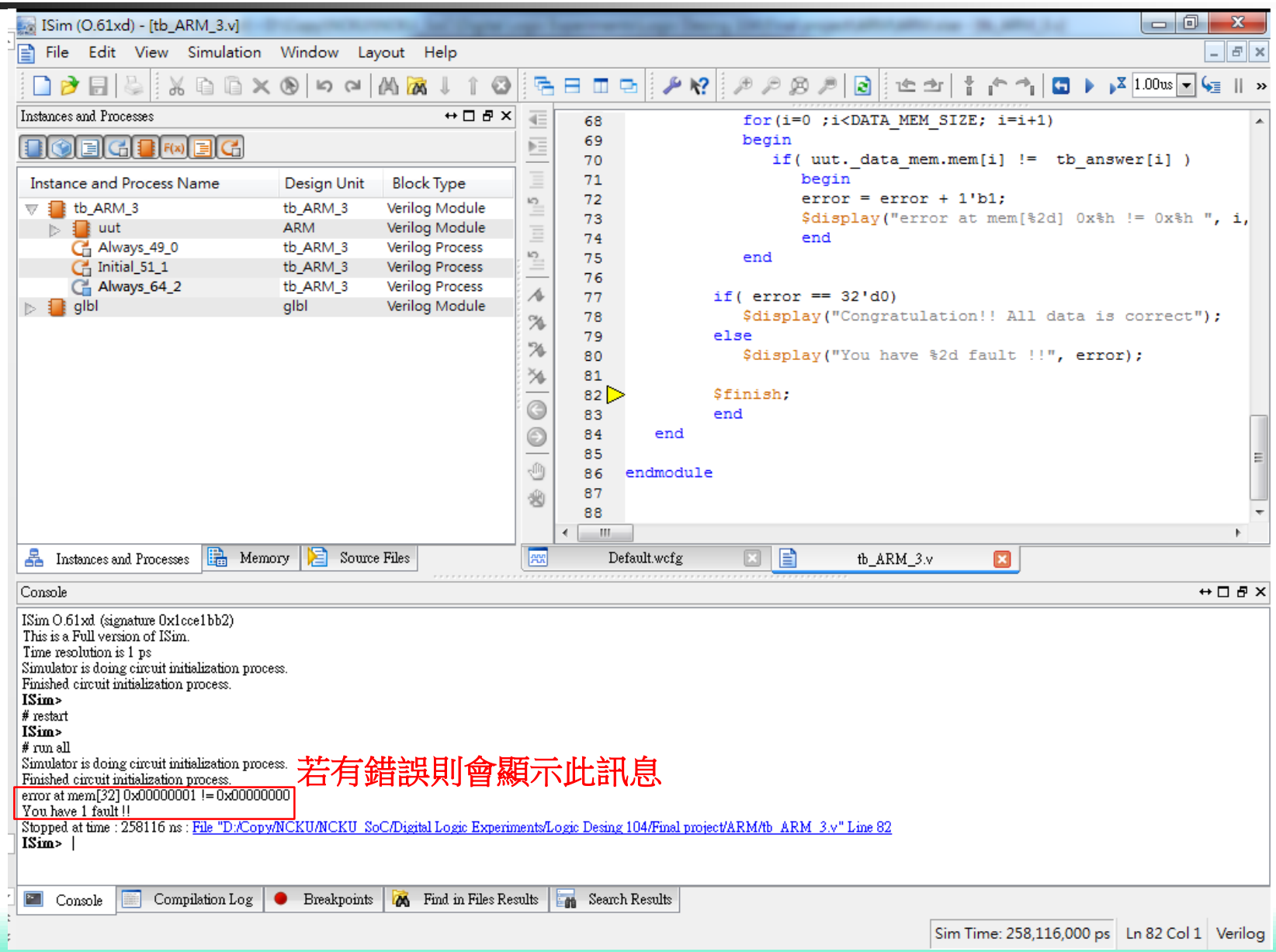


Testbench for ARM(6/7)



全部正確則會顯示此訊息

Testbench for ARM(7/7)



評分方法

- 這次期末project占我們總成績的20%，主要分五階段評分!!

階段	分數
繳交程式碼和報告	4%
通過 test_bench_1	4%
通過 test_bench_2	4%
通過 test_bench_3	4%
通過 test_bench_4	4%