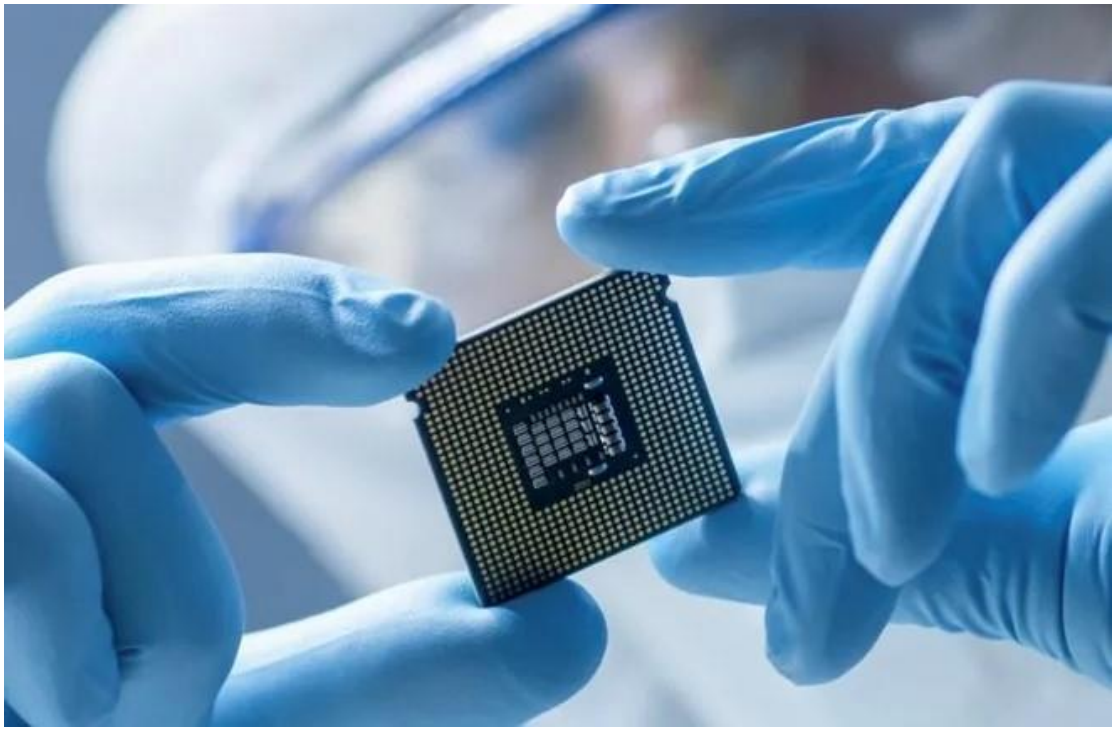


# 數位系統設計作業

# HW10

學號 : 01257027 | 姓名 : 林承羿

# 第一題

前置：

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity Add_32bit is
    port(
            A:in std_logic_vector(31 downto 0);
            B:in std_logic_vector(31 downto 0);
            Result:out std_logic_vector(31 downto 0)
        );
end Add_32bit;
architecture Add_32bit of Add_32bit is

begin

    Result <= A + B;

end Add_32bit;
```

32 bits 加法

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ALU is
    port(
            A:in std_logic_vector(31 downto 0);
            B:in std_logic_vector(31 downto 0);
            Operation:in std_logic_vector(3 downto 0);
            ALUresult:out std_logic_vector(31 downto 0);
            Zero:out std_logic

            );
end ALU;
architecture ALU of ALU is
        signal tmp: std_logic_vector(31 downto 0);
begin
    -- 定義運算規則
    tmp <=      A or B when Operation = "0000" else
                A and B when Operation = "0001" else
                A + B when Operation = "0010" else
                A - B when Operation = "0110" else
                A nor B when Operation = "1100" else
                x"00000001" when Operation = "0111" and A < B else
                x"00000000" when Operation = "0111" else
                x"FFFFFFFF";

    ALUresult <= tmp;
    Zero <= '1' when tmp = x"00000000" else
            '0';



end ALU;
```

定義 ALU 運算規則

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity ALUControl is
    port(
            ALUOp:in std_logic_vector(1 downto 0);
            Inst5_0:in std_logic_vector(5 downto 0);  --func of R type inst
            Operation:out std_logic_vector(3 downto 0)
        );
end ALUControl;
architecture ALUControl of ALUControl is
begin

    Operation <= "0010" when ALUOp = "00" else  -- lw sw
                 "0110" when ALUOp = "01" else  -- beq
                 "0010" when ALUOp = "10" and Inst5_0 = "100000" else -- (+)
                 "0110" when ALUOp = "10" and Inst5_0 = "100010" else -- (-)
                 "0000" when ALUOp = "10" and Inst5_0 = "100100" else -- (or)
                 "0001" when ALUOp = "10" and Inst5_0 = "100101" else -- (and)
                 "0111" when ALUOp = "10" and Inst5_0 = "101010" else -- (slt)
              "1111";

end ALUControl;
```

經初步份類後細部規範指令運算模式

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity Control is
    port(
            Inst31_26:in std_logic_vector(5 downto 0);
            Jump:out std_logic;
            Halt:out std_logic;
            RegImport:out std_logic;
            RegOutport:out std_logic;
            RegDst:out std_logic;
            Branch:out std_logic;
            MemRead:out std_logic;
            MemtoReg:out std_logic;
            ALUOp:out std_logic_vector(1 downto 0);
            MemWrite:out std_logic;
            ALUSrc:out std_logic;
            RegWrite:out std_logic
        );
end Control;
architecture Control of Control is
    signal tmp:std_logic_vector(12 downto 0);
begin
    -- op code
    tmp <= "0000100100010" when inst31_26 = 0 else      --R
           "0000011110000" when inst31_26 = 35 else     --lw
            "0000010001000" when inst31_26 = 43 else    --sw
            "0000000000101" when inst31_26 = 4 else     --beq
            "0010000100011" when inst31_26 = 63 else    --RegImport
            "0001000000011" when inst31_26 = 62 else    --RegOutportout
            "0100000000011" when inst31_26 = 61 else    --Halt
            "1000000000011" when inst31_26 = 2 else     --Jump
            "0000000000000";
    Jump <= tmp(12);
    Halt <= tmp(11);
    RegImport <= tmp(10);
    RegOutport <= tmp(9);
```

```
    RegDst <= tmp(8);
    ALUSrc <= tmp(7);
    MemtoReg <= tmp(6);
    RegWrite <= tmp(5);
    MemRead <= tmp(4);
    MemWrite <= tmp(3);
    Branch <= tmp(2);
    ALUOp <= tmp(1 downto 0);
end Control;
```

規定好各種控制線

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity Data_memory is                                   -- 記憶體(64 words)
    port(
            clk:in std_logic;
            Address:in integer range 0 to 63;
            Write_data:in std_logic_vector(31 downto 0);
            MemWrite:in std_logic;
            MemRead:in std_logic;
            Read_data:out std_logic_vector(31 downto 0)
        );
end Data_memory;
architecture Data_memory of Data_memory is
    type mem_array is array (0 to 63)of std_logic_vector( 31 downto 0 );
    signal memory : mem_array;
begin
    -- 讀入記憶體
    Read_data <= memory(Address) when MemRead = '1' else
              "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";  --(others => 'Z')
    process(clk)
    begin
        if clk'event and clk = '1' then
            if MemWrite = '1' then
                memory(Address) <= Write_data;  -- 寫入記憶體
            end if;
        end if;
    end process;



end Data_memory;
```

管理記憶體，包括 讀、寫

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity Registers is                                    -- 暫存器(MIPS 有 32 個)
    port(
            clk:in std_logic;
            Reset:in std_logic;
            RegWrite:in std_logic;
            Read_register1:in integer range 0 to 31;
            Read_register2:in integer range 0 to 31;
            Write_register:in integer range 0 to 31;
            Write_data:in std_logic_vector(31 downto 0);
            Read_data1:out std_logic_vector(31 downto 0);
            Read_data2:out std_logic_vector(31 downto 0)
        );
end Registers;

architecture Registers of Registers is

    type reg_array is array (0 to 31)of std_logic_vector( 31 downto 0 );
    signal reg_files : reg_array;
begin

    Read_data1 <= reg_files(Read_register1);
    Read_data2 <= reg_files(Read_register2);

    process(clk,Reset)
    begin
        if Reset = '0' then
            for i in 0 to 31 loop
                reg_files(i) <= (others => '0');        -- 全部歸零
            end loop;
        elsif clk'event and clk = '1' then
            if RegWrite = '1' then
                reg_files(Write_register) <= Write_data; -- 寫入暫存器
            end if;
        end if;
    end process;
```

暫存器管理，包括讀、寫暫存器

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity Sign_extend is                                   -- 立即值延伸MSB
    port(
            Inst15_0:in std_logic_vector(15 downto 0);
            Extendout:out std_logic_vector(31 downto 0)
        );
end Sign_extend;
architecture Sign_extend of Sign_extend is

begin

    Extendout <= "0000000000000000"&Inst15_0 when Inst15_0(15) = '0' else
                 "1111111111111111"&Inst15_0;

end Sign_extend;
```

I type 指令延伸

主要程式：

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;

entity MIPS is
    port(
            clk:in std_logic;
            Reset:in std_logic;
            --Input:in std_logic_vector(31 downto 0);
            PCout:out std_logic_vector(7 downto 0);
            ALUout:out std_logic_vector(31 downto 0);
            Instout:out std_logic_vector(31 downto 0);
            ALUopout:out std_logic_vector(3 downto 0);
            Output:out std_logic_vector(31 downto 0)
        );
end MIPS;
architecture MIPS of MIPS is

            component Registers is
                port(
                        clk:in std_logic;
                        Reset:in std_logic;
                        RegWrite:in std_logic;
                        Read_register1:in integer range 0 to 31;
                        Read_register2:in integer range 0 to 31;
                        Write_register:in integer range 0 to 31;
                        Write_data:in std_logic_vector(31 downto 0);
                        Read_data1:out std_logic_vector(31 downto 0);
                        Read_data2:out std_logic_vector(31 downto 0)
                    );
            end component;

```

```
component ALU is
    port(
            A:in std_logic_vector(31 downto 0);
            B:in std_logic_vector(31 downto 0);
            Operation:in std_logic_vector(3 downto 0);
            ALUresult:out std_logic_vector(31 downto 0);
            Zero:out std_logic

            );
end component;

component Data_memory is
    port(
            clk:in std_logic;
            Address:in integer range 0 to 63;
            Write_data:in std_logic_vector(31 downto 0);
            MemWrite:in std_logic;
            MemRead:in std_logic;
            Read_data:out std_logic_vector(31 downto 0)
        );
end component;

component Instruction_memory is
    port(
            Read_address:in integer range 0 to 63;
            Instruction:out std_logic_vector(31 downto 0)
        );
end component;

```

```
component Sign_extend is
    port(
            Inst15_0:in std_logic_vector(15 downto 0);
            Extendout:out std_logic_vector(31 downto 0)
        );
end component;

component Control is
    port(
            Inst31_26:in std_logic_vector(5 downto 0);
            Jump:out std_logic;
            Halt:out std_logic;
            RegImport:out std_logic;
            RegOutport:out std_logic;
            RegDst:out std_logic;
            Branch:out std_logic;
            MemRead:out std_logic;
            MemtoReg:out std_logic;
            ALUOp:out std_logic_vector(1 downto 0);
            MemWrite:out std_logic;
            ALUSrc:out std_logic;
            RegWrite:out std_logic
        );
end component;

component ALUControl is
    port(
            ALUOp:in std_logic_vector(1 downto 0);
            Inst5_0:in std_logic_vector(5 downto 0);
            Operation:out std_logic_vector(3 downto 0)
        );
end component;
```

```
            component Add_32bit is
                port(
                        A:in std_logic_vector(31 downto 0);
                        B:in std_logic_vector(31 downto 0);
                        Result:out std_logic_vector(31 downto 0)
                    );
            end component;

        signal Instruction:std_logic_vector(31 downto 0);
        signal Extendout:std_logic_vector(31 downto 0);
        signal Read_data1:std_logic_vector(31 downto 0);
        signal Read_data2:std_logic_vector(31 downto 0);

        signal ALUResult:std_logic_vector(31 downto 0);
        signal ALU_B:std_logic_vector(31 downto 0);
        signal Operation:std_logic_vector(3 downto 0);
        signal Zero:std_logic;


        signal Read_mem_data:std_logic_vector(31 downto 0);
        signal PC:std_logic_vector(31 downto 0);
        signal Jump,Halt,RegImport,RegOutport:std_logic;
        signal Branch,RegDst,MemRead,MemtoReg,MemWrite,ALUSrc,RegWrite:std_logic;
        signal ALUop:std_logic_vector(1 downto 0);

        signal Read_register1,Read_register2,Write_register:integer range 0 to 31;
        signal Write_data:std_logic_vector(31 downto 0);

        signal Add_32bit_A:std_logic_vector(31 downto 0);
        signal Add_32bit_B:std_logic_vector(31 downto 0);
        signal AddResult:std_logic_vector(31 downto 0);

begin

--Program Counter
    process(clk,reset)
    begin
        if reset = '0' then
            PC <= (others => '0');
        elsif clk'event and clk = '1' then
            if Branch = '1' and Zero = '1' then
                PC <= AddResult;
            elsif Jump = '1' then
            --else
                PC <= PC(31 downto 28)&Instruction(25 downto 0)&"00";   -- pc+26 bits 位置(指令4的倍數，無條件左移二)
            elsif Halt = '0' then
            --else
                PC <= PC + 4;
            end if;
        end if;
    end process;

--Instruction Memory

Inst_Memory_part:
Instruction_memory port map (conv_integer(PC(7 downto 2)),Instruction);              -- 讀出指令
```

```
--Registers
Read_register1 <= conv_integer(Instruction(20 downto 16)) when RegOutport = '1' else  -- (OUTP)/(R type、I type)
                    conv_integer(Instruction(25 downto 21));

Read_register2 <= conv_integer(Instruction(20 downto 16));

Write_register <= conv_integer(Instruction(20 downto 16)) when RegImport = '1' else  --MUX (INP)/(I type/R type)
                    conv_integer(Instruction(20 downto 16)) when RegDst = '0' else  --MUX
                    conv_integer(Instruction(15 downto 11));

Write_data <=  x"0000"&Instruction(15 downto 0) when RegImport = '1' else  --INP(直接讀數字)
                  ALUResult when MemtoReg = '0' else
                  Read_mem_data;

Output <= Read_data1 when RegOutport = '1' else  --OUTP
          x"FFFFFFFF";

Register_part:
Registers port map(clk,Reset,RegWrite,Read_register1,Read_register2,Write_register, Write_data,Read_data1,Read_data2);


--ALU
ALU_B <= Read_data2 when ALUSrc = '0' else Extendout; -- 圖表規定

ALU_part:
ALU port map (Read_data1,ALU_B,Operation,ALUresult,Zero); -- 做運算


--Data Memory

Data_mem: -- 讀寫記憶體
Data_memory port map (clk,conv_integer(ALUResult(7 downto 2)),Read_data2,MemWrite,MemRead,Read_mem_data);

--Control   翻譯指令(初步分類)

Control_part:
Control port map (Instruction(31 downto 26),Jump,Halt,RegImport,RegOutport,RegDst,Branch,MemRead,MemtoReg,ALUOp,MemWrite,ALUSrc,RegWrite);

--ALUControl    翻譯指令(細部計算)

ALUControl_part:
ALUControl port map (ALUOp,Instruction(5 downto 0),Operation);


--Sign_extend    立即值延伸

Sign_extend_part:
Sign_extend port map (Instruction(15 downto 0),Extendout);


--Add_32bit
Add_32bit_A <= PC + 4;
Add_32bit_B <= Extendout(29 downto 0) & "00";

Add_32bit_part:
Add_32bit port map (Add_32bit_A,Add_32bit_B,AddResult);

PCout <= PC(7 downto 0); -- 指令行數
ALUout <= ALUResult;
Instout <= Instruction;
ALUopout <= Operation;

end MIPS;
```

以上為此次作業的主要程式

## 程式碼

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity Instruction_memory is            -- 指令存放地
    port(
            Read_address:in integer range 0 to 63;
            Instruction:out std_logic_vector(31 downto 0)
         );
end Instruction_memory;
architecture Instruction_memory of Instruction_memory is
    type mem_array is array (0 to 63)of std_logic_vector( 31 downto 0 );
    signal memory : mem_array := (
                                    x"fc010001",x"fc020000",x"fc03000a",x"fc040013",x"fc070004",x"ac430000", x"10640003", x"00611820",x"00471020",x"08000005",
                                    x"fc060000",x"fc020000",x"8c450000",x"f8050000",x"f8060000",x"00c53020",x"00471020",x"10a40001",
                                    x"0800000c",x"f8060000",x"f4000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
                                    x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
                                    x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
                                    x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
                                    x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
                                    x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000");
begin

    Instruction <= memory(Read_address);



end Instruction_memory;
```

以上為此基礎題的機械碼

$1:每次加 1(存入值)

$2:記憶體位置

$3:存入值

$4:停止位置

$5:載入值

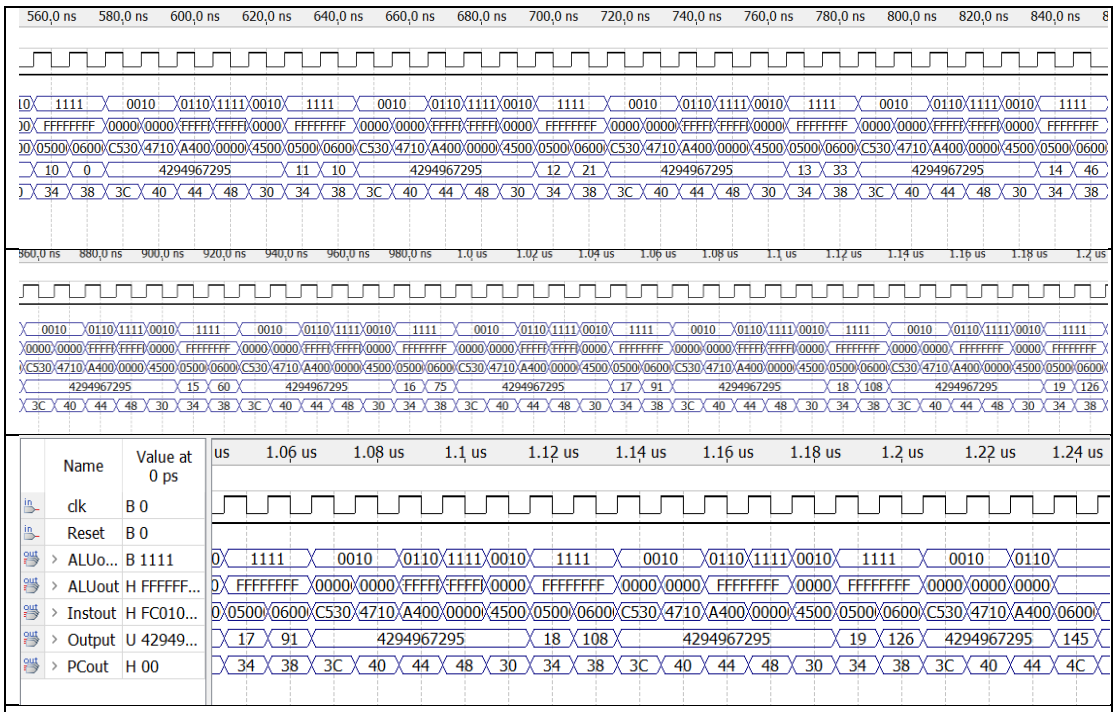
$6:累加值

$7:每次加 4(記憶體)

| | | | | |
|---|---|---|---|---|
| 00 | INP $1, 1 | fc010001 | 111111 00000 00001 0000000000000001 | 存入值+1 |
| 04 | INP $2, 0 | fc020000 | 111111 00000 00010 0000000000000000 | 記憶體位置 |
| 08 | INP $3, 10 | fc03000a | 111111 00000 00011 0000000000001010 | 存入值 |
| 0C | INP $4, 19 | fc040013 | 111111 00000 00100 0000000000010011 | 停止位置 |
| 10 | INP $7, 4 | fc070004 | 111111 00000 00111 0000000000000100 | 記憶體+4 |

| | | | | |
|---|---|---|---|---|
| 14<br>loop1 | SW $3, 0($2) | ac430000 | 101011 00010 00011<br>0000000000000000 | 值存入記憶體 |
| 18 | Beq $3, $4, next1 | 10640003 | 000100 00011 00100<br>0000000000000011 | 跳出迴圈 |
| 1C | Add $3, $1, $3 | 00611820 | 000000 00011 00001 00011<br>00000100000 | 存入值+1 |
| 20 | Add $2, $7, $2 | 00471020 | 000000 00010 00111 00010<br>00000100000 | 記憶體位置+4 |
| 24 | J loop1 | 08000005 | 000010<br>00000000000000000000000001<br>01 | 迴圈 2 |
| 28<br>next1 | INP $6, 0 | fc060000 | 111111 00000 00110<br>0000000000000000 | 累加值(答案) |
| 2C | INP $2, 0 | fc020000 | 111111 00000 00010<br>0000000000000000 | 記憶體位置 |
| 30<br>loop2 | LW $5, 0($2) | 8c450000 | 100011 00010 00101<br>0000000000000000 | 載入記憶體值 |
| 34 | OUPT $5 | f8050000 | 111110 00000 00101<br>0000000000000000 | 輸出載入值 |
| 38 | OUPT $6 | f8060000 | 111110 00000 00110<br>0000000000000000 | 輸出累加結果 |
| 3C | Add $6, $5, $6 | 00c53020 | 000000 00110 00101 00110<br>00000100000 | 累加值更新 |
| 40 | Add $2, $7, $2 | 00471020 | 000000 00010 00111 00010<br>00000100000 | 記憶體位置+4 |
| 44 | Beq $5, $4, next2 | 10a40001 | 000100 00101 00100<br>0000000000000001 | 跳出迴圈 |
| 48 | J loop2 | 0800000c | 000010<br>00000000000000000000001100 | 迴圈 2 |
| 4C<br>next2 | OUPT $6 | f8060000 | 111110 00000 00110<br>0000000000000000 | 輸出結果 |
| 50 | HALT | f4000000 | 111101<br>0000000000000000000000000<br>00 | 停止 |

## 波形圖




以上為 x”34”輸出的表示，證明確實有把值存入記憶體，符合需求

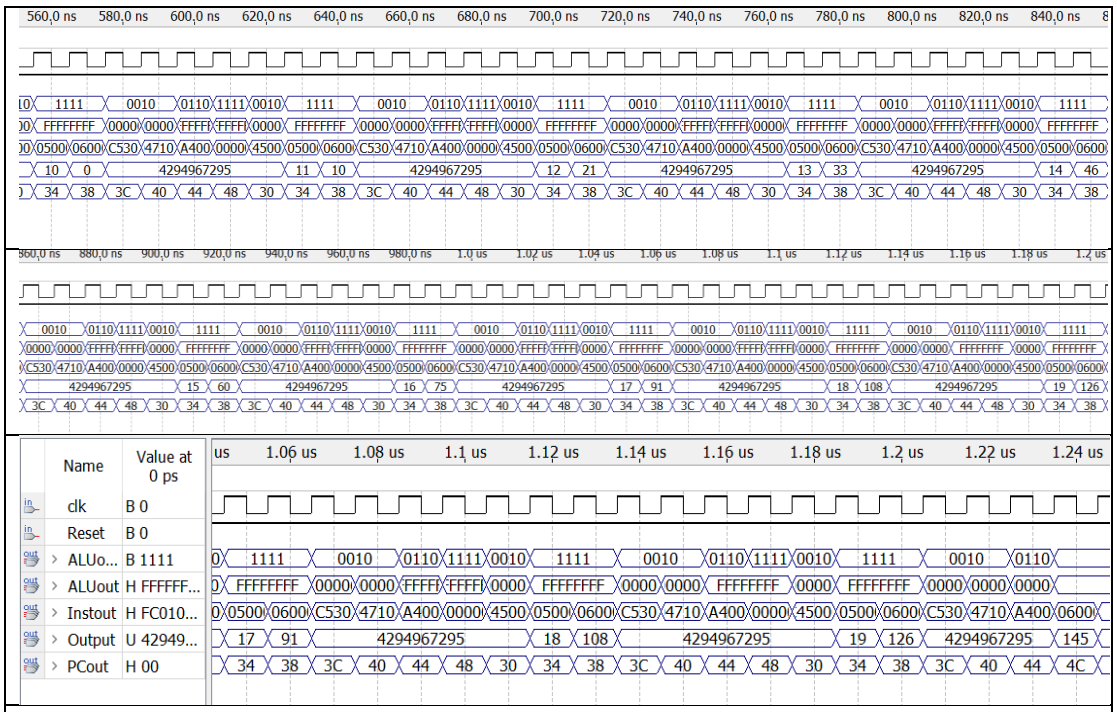
# 第二題

## 程式碼

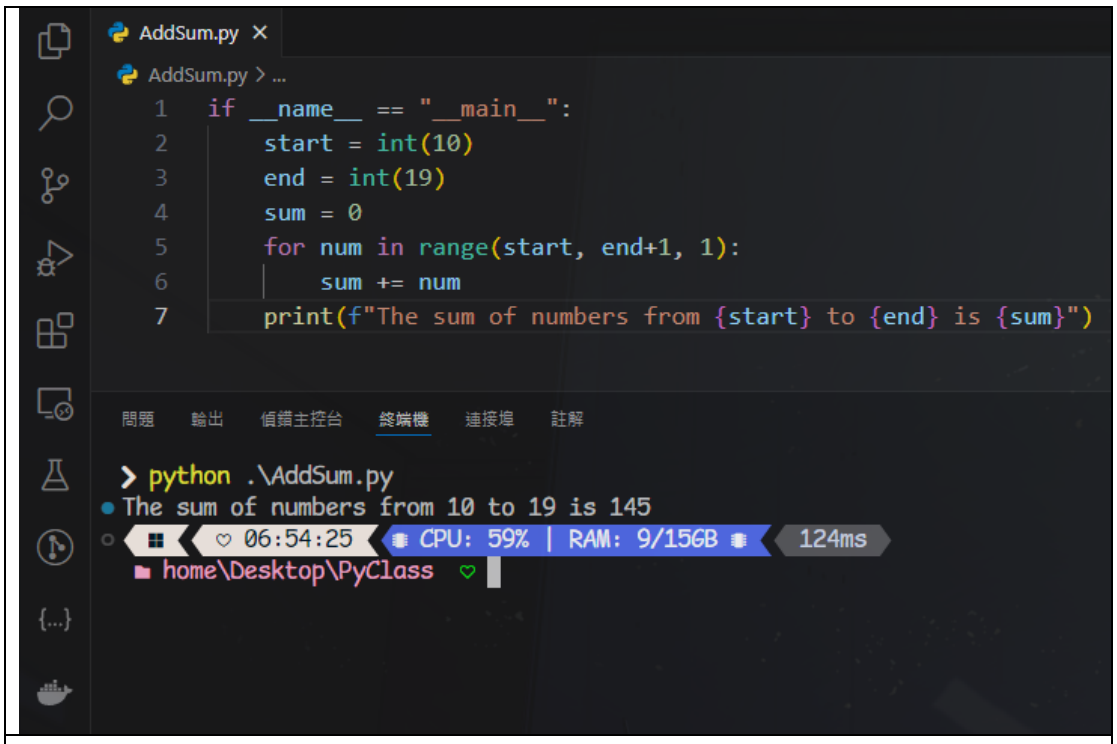
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity Instruction_memory is            -- 指令存放地
    port(
            Read_address:in integer range 0 to 63;
            Instruction:out std_logic_vector(31 downto 0)
        );
end Instruction_memory;
architecture Instruction_memory of Instruction_memory is
    type mem_array is array (0 to 63)of std_logic_vector( 31 downto 0 );
    signal memory : mem_array := (
                                    x"fc010001",x"fc020000",x"fc03000a",x"fc040013",x"fc070004",x"ac430000", x"10640003", x"00611820",x"00471020",x"08000005",
                                    x"fc060000",x"fc020000",x"8c450000",x"f8050000",x"f8060000",x"00c53020",x"00471020",x"10a40001",
                                    x"0800000c",x"f8060000",x"f4000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
                                    x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
                                    x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
                                    x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
                                    x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
                                    x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000");
begin

    Instruction <= memory(Read_address);



end Instruction_memory;
```

以上為此基礎題的機械碼

## 波形圖




以上為 x”38”輸出的表示，證明值確實的被累加，並於最後得到 145




證明答案確實是如此

# 加分題

## 程式碼

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity Instruction_memory is
    port(
            Read_address:in integer range 0 to 63;
            Instruction:out std_logic_vector(31 downto 0)
        );
end Instruction_memory;
architecture Instruction_memory of Instruction_memory is
    type mem_array is array (0 to 63)of std_logic_vector( 31 downto 0 );
    signal memory : mem_array := (
                                    x"fc000000",x"fc010007",x"ac010000",x"fc010004",x"ac010004",x"fc010058", x"ac010008", x"fc010040",x"ac01000c",x"fc010022
                                    x"ac010010",x"fc01000a",x"ac010014",x"fc01005b",x"ac010018",x"fc010008",x"ac01001c",x"fc01003e",
                                    x"ac010020",x"fc010009",x"ac010024",x"fc020004",x"fc030000",x"fc040024", x"fc070000",x"8c050000",
                                    x"0065302a",x"10c70001",x"00a71820",x"f8030000",x"10040002",x"00020020",x"08000018",x"f8030000",
                                    x"f4000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
                                    x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
                                    x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",
                                    x"00000000",x"00000000",x"00000000",x"00000000",x"00000000",x"00000000");
begin

    Instruction <= memory(Read_address);



end Instruction_memory;
```

以上為此加分題的機械碼

$0:記憶體位置

$1:記憶體值

$2:記憶體位置+4

$3:最大值

$4:停止值

$5:載入值

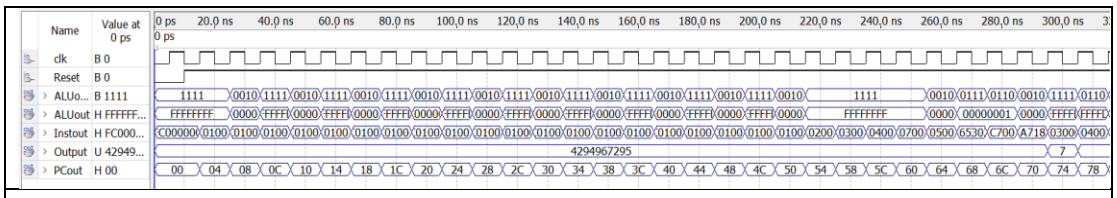
$6:比較結果值

$7:比較後跳躍?

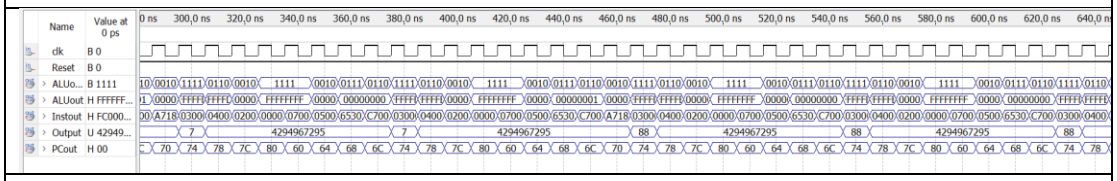
| | | | | |
|---|---|---|---|---|
| 00 | INP $0, 0 | fc000000 | 111111 00000 00000 0000000000000000 | 記憶體位置 |
| 04 | INP $1, 7 | fc010007 | 111111 00000 00001 0000000000000111 | 記憶體值 7 |
| 08 | SW $1, 0($0) | ac010000 | 101011 00000 00001 0000000000000000 | 存值 |
| 0C | INP $1, 4 | fc010004 | 111111 00000 00001 0000000000000100 | 記憶體值 4 |
| 10 | SW $1, 4($0) | ac010004 | 101011 00000 00001 0000000000000100 | 存值 |
| 14 | INP $1, 88 | fc010058 | 111111 00000 00001 0000000001011000 | 記憶體值 88 |
| 18 | SW $1, 8($0) | ac010008 | 101011 00000 00001 0000000000001000 | 存值 |
| 1C | INP $1, 64 | fc010040 | 111111 00000 00001 0000000001000000 | 記憶體值 64 |
| 20 | SW $1, 12($0) | ac01000c | 101011 00000 00001 0000000000001100 | 存值 |
| 24 | INP $1, 34 | fc010022 | 111111 00000 00001 0000000000100010 | 記憶體值 34 |
| 28 | SW $1, 16($0) | ac010010 | 101011 00000 00001 0000000000010000 | 存值 |
| 2C | INP $1, 10 | fc01000a | 111111 00000 00001 0000000000001010 | 記憶體值 10 |
| 30 | SW $1, 20($0) | ac010014 | 101011 00000 00001 0000000000010100 | 存值 |
| 34 | INP $1, 91 | fc01005b | 111111 00000 00001 0000000001011011 | 記憶體值 91 |
| 38 | SW $1, 24($0) | ac010018 | 101011 00000 00001 0000000000011000 | 存值 |
| 3C | INP $1, 8 | fc010008 | 111111 00000 00001 0000000000001000 | 記憶體值 8 |
| 40 | SW $1, 28($0) | ac01001c | 101011 00000 00001 0000000000011100 | 存值 |
| 44 | INP $1, 62 | fc01003e | 111111 00000 00001 0000000000111110 | 記憶體值 62 |

| | | | | |
|---|---|---|---|---|
| 48 | SW $1, 32($0) | ac010020 | 101011 00000 00001 0000000000100000 | 存值 |
| 4C | INP $1, 9 | fc010009 | 111111 00000 00001 0000000000001001 | 記憶體值9 |
| 50 | SW $1, 36($0) | ac010024 | 101011 00000 00001 0000000000100100 | 存值 |
| 54 | INP $2, 4 | fc020004 | 111111 00000 00010 0000000000000100 | 記憶體間隔 |
| 58 | INP $3, 0 | fc030000 | 111111 00000 00011 0000000000000000 | 最大值 |
| 5C | INP $4, 36 | fc040024 | 111111 00000 00100 0000000000100100 | 停止值 |
| 60 | INP $7, 0 | fc070000 | 111111 00000 00111 0000000000000000 | 載入必較值 |
| 64 loop | LW $5, 0($0) | 8c050000 | 100011 00000 00101 0000000000000000 | 載入記憶體值 |
| 68 | Slt $6, $3, $5 | 0065302a | 000000 00011 00101 00110 00000101010 | 比較最大值 |
| 6C | Beq $6, $7, cg | 10c70001 | 000100 00110 00111 0000000000000001 | 比較最大值 |
| 70 | Add $3, $5, $7 | 00a71820 | 000000 00101 00111 00011 00000100000 | 改值 |
| 74 cg | OUPT $3 | f8030000 | 111110 00000 00011 0000000000000000 | 輸出目前最大 |
| 78 | Beq $0, $4, next | 10040002 | 000100 00000 00100 0000000000000010 | 跳出迴圈 |
| 7C | Add $0, $2, $0 | 00020020 | 000000 00000 00010 00000 00000100000 | 記憶體位置+4 |
| 80 | J loop | 08000018 | 000010 00000000000000000000011000 | 迴圈 |
| 84 next | OUPT $3 | f8030000 | 111110 00000 00011 0000000000000000 | 輸出最大值 |
| 88 | HALT | f4000000 | 111101 00000000000000000000000000 | 停止 |

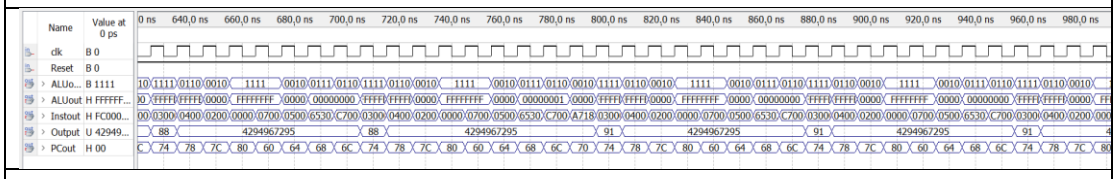
## 波形圖



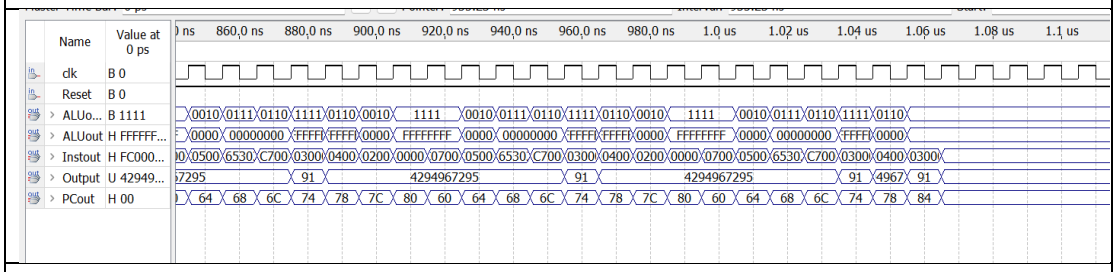

可以看到在經過 x"70"後，第一個數字 7 被選為最大



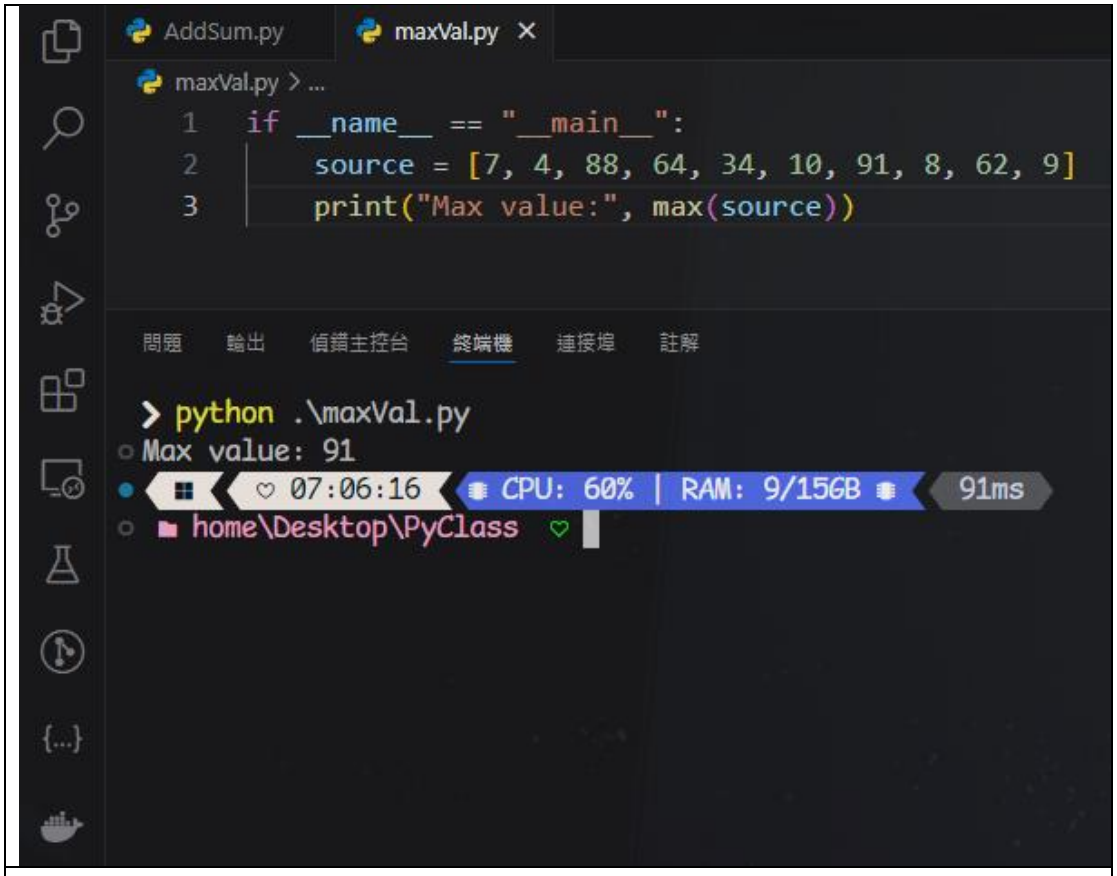

可以看到只有在 7、80 之間會經過 x"70"，符合自己的假設，替換最大值。




保持不斷偵測替換最大值




最終因為 x"88"的 HALT 而停止，找到最大值為 91，符合事實




以上證明答案正確

## 心得

透過此次的實驗，我更了解 MIPS 的運作方式，透過組合語言轉成機械碼，並以此執行程式。也令我了解高階語言的一兩行程式是如此的具有威力，且這噁心的東西十分的不具人性，你怎麼會知道哪個地方自己眼睛花掉少看個一或零？只有當迴圈跑步出來卡死在路上才知道自己又又又得在全部重翻譯一遍，真的是看了一眼不會再想看第二眼，這鬼東西狗都不寫。