Hw1_109511314_鄭旭恩

Central Processing Unit (CPU) - ALU & ID unit

1.Sketch:

```cpp
#include "ALU.h"

void ALU::alu_process()
{

    switch (opcode.read())
    {
    case 0:
        ALU_out.write(rs_data.read() + rt_data.read());
        break;
    case 1:
        ALU_out.write(rs_data.read() * rt_data.read());
        break;
    case 2:
        ALU_out.write(rs_data.read() & rt_data.read());
        break;
    case 3:
        ALU_out.write(~rs_data.read());
        break;
    case 4:
        if (rs_data.read()[15] == 1)
            ALU_out.write(~rs_data.read() + 1);
        else
            ALU_out.write(rs_data.read());
        break;
    case 5:
        if (rs_data.read() > rt_data.read())
            ALU_out.write(rt_data.read());
        else
            ALU_out.write(rs_data.read());
        break;
    case 6:
        ALU_out.write(rs_data.read() << immediate.read());
        break;
    case 7:
        sc_uint<16> IMM = immediate.read();
        int IMM_length = immediate.read().length();
        if (IMM[IMM_length-1] == 1)
        {
            // 0b = binary補滿
            IMM.range(15, IMM_length) = 0b1111111111111111;
            // 2's complement
            IMM = -(~IMM + 1);
            ALU_out.write(rs_data.read() + IMM);
        }
        else
            ALU_out.write(rs_data.read() + IMM);
        break;
    }
}
```

```cpp
#ifndef ALU_H
#define ALU_H

#include "systemc.h"
#include <bitset>

SC_MODULE(ALU){
    sc_in < sc_uint<3> > opcode;//opcode from decoder, to decide which operation ALU unit do calculate
    sc_in < sc_int<16> > rs_data;//rs_data from register, data from register[rs] to calculate in ALU unit
    sc_in < sc_int<16> > rt_data;//rt_data from register, data from register[rt] to calculate in ALU unit
    sc_in < sc_uint<5> > immediate;//immediate from decoder, data from instruction to calculate in ALU unit
    sc_out < sc_int<16> > ALU_out;//after your calculation, put your result in this port

    void alu_process();

    SC_CTOR(ALU)
    {
        SC_METHOD(alu_process);
        sensitive << opcode << rs_data << rt_data << immediate;
    }

};

#endif
```

```cpp
#include "decoder.h"

void decoder::decoder_process()
{
    opcode.write(instruction.read().range(15, 13));
    rs.write(instruction.read().range(12, 9));
    rt.write(instruction.read().range(8, 5));
    immediate.write(instruction.read().range(4, 0));
}
// void decoder::decoder_process() {
//     while (true) {
//         wait(); // 等待instruction信号的变化
//         sc_uint<16> inst = instruction.read();

//         sc_uint<3> opcode_field = inst.range(15, 13);
//         sc_uint<4> rs_field = inst.range(12, 9);
//         sc_uint<4> rt_field = inst.range(8, 5);
//         sc_uint<5> immediate_field = inst.range(4, 0);

//         opcode.write(opcode_field);
//         rs.write(rs_field);
//         rt.write(rt_field);
//         immediate.write(immediate_field);
//     }
// }
```

```cpp
#ifndef DECODER_H
#define DECODER_H

#include "systemc.h"

SC_MODULE(decoder)
{
    sc_in<sc_uint<16>> instruction; // instruction from pattern
    sc_out<sc_uint<3>> opcode;      // to ALU unit, decide which operation ALU unit do calculate
    sc_out<sc_uint<4>> rs;          // an address to register unit, register unit will output the corresponding data to ALU unit
    sc_out<sc_uint<4>> rt;          // an address to register unit, register unit will output the corresponding data to ALU unit
    sc_out<sc_uint<5>> immediate;   // this signal represent the Memory address where your ALU output result need to go, and your ALU unit also need this to do some of the operation

    void decoder_process();

    SC_CTOR(decoder)
    {
        SC_METHOD(decoder_process);
        sensitive << instruction;
    }
};
#endif
```

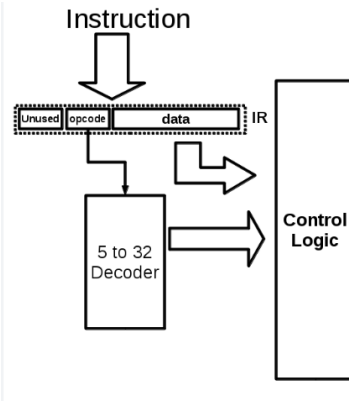
2.Result:

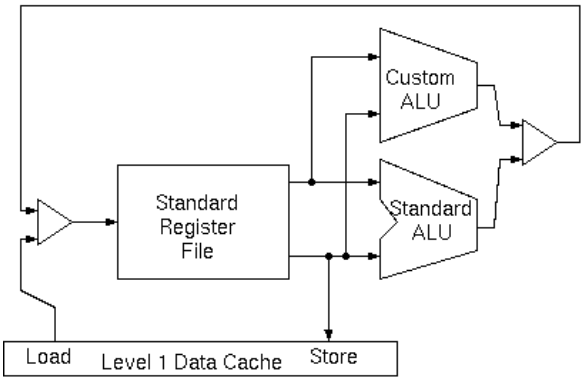
```
-32256  -32256          9               9                PASS
9011    9011            26              26               PASS
8894    8894            6               6                PASS
268     268             18              18               PASS
3333    3333            13              13               PASS
0       0               23              23               PASS
0       0               17              17               PASS
4537    4537            24              24               PASS
17636   17636           3               3                PASS
0       0               26              26               PASS
0       0               5               5                PASS
5678    5678            29              29               PASS
3333    3333            5               5                PASS
0       0               23              23               PASS
4545    4545            24              24               PASS
168     168             9               9                PASS
4545    4545            30              30               PASS
1111    1111            20              20               PASS
8888    8888            11              11               PASS
1111    1111            18              18               PASS
-24576  -24576          13              13               PASS
5       5               6               6                PASS
-6667   -6667           27              27               PASS
10000   10000           20              20               PASS
13185   13185           20              20               PASS

Info: /OSCI/SystemC: Simulation stopped by user.
Number of pass cases: 53
Congratulation! Your design is correct!
17:33 DCS121@ee25[~/HW01]$
```

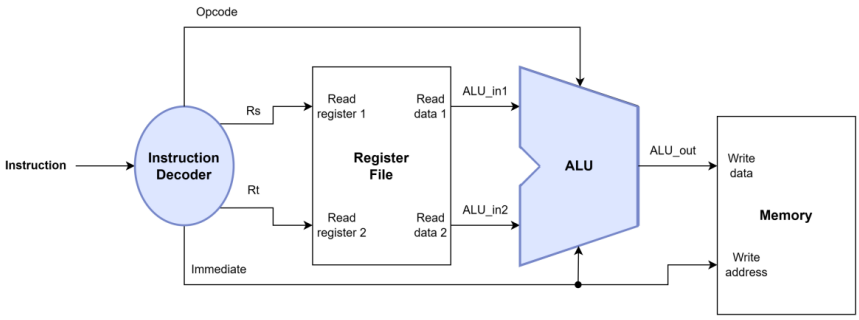
3.Think:

而 ID 是負責在前期負責解析及識別指令，讓處理器可以準確執行指令，並根據指令的要求動態配置資源。透過使用地址，可以更有系統地組織和管理資料，同時也有助於使系統設計更加簡潔。這種設計方式有助於提高系統的效率和可擴展性，同時也使得程式碼更易於理解和維護。如右下圖。





這次實驗中所用到的 ALU 和 ID 都是在設計 CPU 以及 GPU 等等運算單元時很重要的基本構造。ALU 負責執行各種算術和邏輯運算，包括加法、減法、位元運算等。ALU 非常重要因為其直接影響 CPU 或 GPU 等等的處理效能以及運算能力。如左上圖。



這次實驗的 block diagram 如上圖，在這份作業中，需要設計一個 16 位元的 CPU，並設計一個算術邏輯單元（ALU）和一個指令解碼器（ID）單元。每個指令將根據指令集體系結構（ISA）執行不同的操作。最終，將結果將被存儲在特定的記憶體位置。

Rs 和 Rt 表示特定 register 的地址。使用 4 位元的 register address，共有 16 個可用 register。每個寄存器可以容納 16 位元的數據。指令解碼器（ID）單元必須將相關位元分配給 register file，使算術邏輯單元（ALU）能夠訪問其操作所需的數據。

Immediate 表示一個記憶體地址。一個 5 位元的即時值表示有 32 個不同的記憶體塊可用。在你的架構中，ID 單元需要將相關位元分配給這個記憶體，從而使 ALU 能夠正確執行指令。

4.Problem finding:

這次實驗中，在設計 ALU 及 ID 的時分別都遇到一些需要解決的問題，如以下。

ALU

Function Name	Operation	op code
Add	Out = R[rs] + R[rt]	3'b000
Mult	Out = R[rs] x R[rt]	3'b001
And	Out = R[rs] & R[rt]	3'b010
Inverse	Out = ~R[rs]	3'b011
Absolute Value	Out = | R[rs] |	3'b100
Minimum	Out = min(R[rs], R[rt])	3'b101
Left Shift	Out = R[rs] << unsigned(Imm)	3'b110
Addi	Out = R[rs] + sign extended(Imm)	3'b111

如上圖所示，此為在 ALU 中所要運行的程式。在前四個 op code 中並沒有遇到甚麼問題，因為都為基本的運算語法，同時也非常直觀。然而從第五個 op code 開始後開始需要用 Behavior 去描述其邏輯運算。尤其在 op code 的 index[7]的時候。試過許多方式處理 immediate 的值，最後選擇先分類 MSB 是 1 還是 0。若為 1 則先找出 immediate 的長度，並將剩餘的值做 sign extension，接著做 two's complement。以 immediate 24 =(11000)為例，轉成-(00111+1)=-(01000)，便輸出-8，再將其加上 rs_data。若 MSB 為 0 則直接加上 rs_data 便可，如下圖。

```
case 7:
    sc_uint<16> IMM = immediate.read();
    int IMM_length = immediate.read().length();
    if (IMM[IMM_length-1] == 1)
    {
        // 0b = binary補滿
        IMM.range(15, IMM_length) = 0b1111111111111111;
        // 2's complement
        IMM = -(~IMM + 1);
        ALU_out.write(rs_data.read() + IMM);
    }
    else
        ALU_out.write(rs_data.read() + IMM);
    break;
```

ID

再將 instruction 分成 opcode、rs、rt、immediate 的時候，我所用到 systmec 中的.range()函數，直接把 instruction 特定 range 的值取出寫入對應到的資料中。如下圖所示。

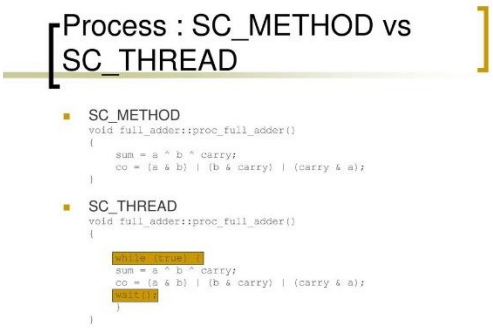
Methods in Arbitrary Width Bit Type

Methods	Description	Usage
range()	Range selection	variable.range(indexl, index2)
and_reduce()	Reduction AND	variable.and_reduce()
or_reduce()	Reduction OR	variable.or_reduce()
xor_reduce()	Reduction XOR	variable.xor_reduce()

```
void decoder::decoder_process()
{
    opcode.write(instruction.read().range(15, 13));
    rs.write(instruction.read().range(12, 9));
    rt.write(instruction.read().range(8, 5));
    immediate.write(instruction.read().range(4, 0));
}
```

SC_METHOD v.s SC_THREAD

由於 CPU 為持續重複運算，只要有新的 instruction 進入，便會執行。因此在 SC_CTOR 時，可以直接用 SC_METHOD，如同 verilog 的 always 語法。若要用 THREAD 則需要用 while(true)迴圈包起來，並在結束時加上 wait()。這也是我一開始在打程式不小心忽略的地方。



5.Suggestion:

Well done overall.

6.Reference:

- SystemC DataTypes Part III (asic-world.com)

DCS_HW1 (2).pdf

-

- Chapter 3 : Combination and Sequential Circuits Modeling - ppt download (slideplayer.com)
- Part II CST SoC D/M Slide Pack 6 (Tools/Tech/Eng): Conservation Cores Approach (cam.ac.uk)
- IAY0340-Digital Systems Modeling and Synthesis (ttu.ee)