2023 ASIC Design Mid-Term project **2023.11.06.**

5-stage Risc-V Processor Design

Name: **Jinmo Ahn**

1 Schematic and Explain my design in detail

1.1 Overall CPU design

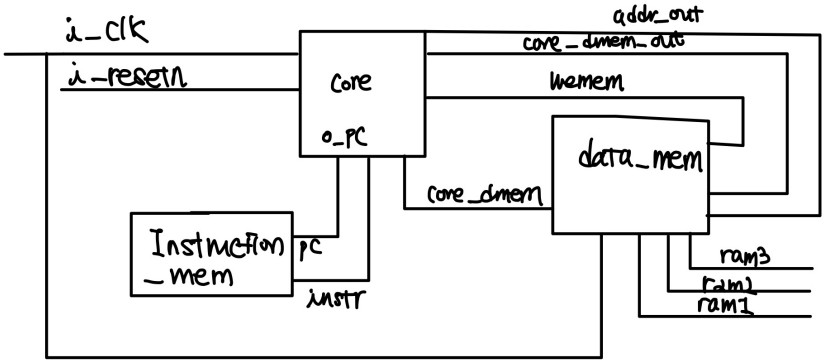

Figure 1: Overall Design

Overall Schematic은 위와 같다. 여기서, Core part는 당연히 I/O signal을 Processing하는 part이고, 우리가 구현하고자 하는 Havard Architecture에 맞추어 D/I mem을 구분하였다. Imem의 경우 주어진 구현에 맞추어, $readmemh 의 구문을 아래처럼 넣어주었다.

```
$readmemh ("test_program2.txt",rom);
```

다음은 data-mem이다. 사실 우리가 RTL Verification 만으로는 실제 RAM에 접근하기 어려워, 이를 Flip-Flop으로 설계하였다. 또한 여기서 ram1 ,ram2 ,ram3는 [31:0] ram [0:31]중 ram[0], ram[1], ram[2]에 wire한 내용으로, 내부에 저장이 잘 되었는지 확인하기 위해 바깥으로 빼서 놓았다. 따라서 향후 저장 시에, 주소를 0,1,2 에만 저장하는 것을 볼 수 있을 것이다. 이를 직접적으로 코드로 확인하면 아래와 같다. (ex. sw x18, 0(x0) / In Risc V, x0 is hardly wired to zero)

```
reg [31:0] ram [0:31];
 assign ram1 = ram[0];
 assign ram2 = ram[1];
 assign ram3 = ram[2];

 always @(posedge i_clk)
 begin
     if(we)
      ram[i_addr[4:0]] = i_data;
 end

 integer i;
 initial
 begin
     for(i=0;i<32;i=i+1)
         ram[i] = 0;
 end
```

Data memory Verilog Source code

마지막으로 Core unit인데, 이는 아래에서 자세히 알아보도록 하자.

1.2 Core design

Core의 component는 우리가 일반적으로 아는 RISC 계열 CPU와 동일하다. PC register와, 각종 signal에 따라 output 값을 바꿔주는 Mux, Control unit이 있다. 또한 CPU의 SRAM과 동일한 역할을 하는 Register file이 존재한다. 해당 Design에서 CLA (carry look-ahead adder)를 사용하여 더욱 빠른 Computation을 하도록 하였으며, Control Hazard를 약간이라도 개선하기 위한 Comparator unit과, Static Branch non-taken mode를 사용하였다. 사실 Dynamic branch prediction과 Static Branch Prediction이 큰 차이가 나지 않는다는 것에 기인해서 앞선 mode를 사용하였다. 하지만, 여전히 1cycle의 Stall을 사용하여야 한다. (Double pumping까지 이용한다면, 가능할 수도 있으리라 생각한다.) 또한 각각 Control signal, Data 등에 맞추어 Pipeline register를 apply해주었다.

2 Validation Scenarios

2.1 Scenario 1 - Tagetting a Control Hazard

먼저, c언어로, 변수 초기화를 진행하고, ALU operation을 진행한 후, if문+함수를 통해 프로시저 콜을 하고, 받은 value를 store하는 시나리오를 생각하자. 간단하게 Assembly로 생각하면 register에 0를 apply하고 간단한 ALU 이후 beq/bne로 분기를 한다. 이후, 해당 branch 내부에서 ALU를 진행한 후, 다시 Unconditional branch로 돌아온 후 store를 하면 된다. (control Hazard에 Reaction)

1. C language Source code

```c
int main (void){
        int a = 0;
        int b = 0;
        a = a +3;
        b = b +2;
            int c=0;
                if (a != b){ // bne instruction
                  c = jinmo(a,b); // add a,b
    }
    return c;
}

int jinmo (int a,int b){
    return a+b; // store value & unconditional branch
}
```

2. Compile to Risc-V Assembly

```
MAIN:

 0 : add x19, x0, x0  /// a=0
 4 : add x20, x0, x0  /// b = 0
 8 : addi x19, x19, 3 /// a = a+3
12 : addi x20, x20, 2  /// b = b+2
16 : add x18, x0, x0 ///c =0
20 : bne x19, x20, L1 /// if (a != b)
24 : sw x18, 0(x0) // store to ram1

EXIT:

L1:
40 (Just In my instruction memory) : add x18 x19 x20
44 beq x0 x0 24 (Loop 24)
```

3. Change to machine language

1. 000009B3
2. 00000A33
3. 00398993
4. 002A0A13
5. 00000E33
6. 013A1A63
7. 01202023
8. 00000063
9. 00000063
10. 00000063
11. 01498933
12. FE0006E3

이때, (Branch destination = Pc + sign-ext(imm[12:1] , 0) 이므로, 앞에 6번째 instruction인 013A1A63은 +20, 12번째 instruction에서는 -20을 sign-extention을 수행해주었다.
위와 같은 Test bench를 돌려보자. 모듈의 설명은 앞에서 진행했으므로, 생략한다. 따라서 txt file에 위의 instruction을 그대로 얹어주고, 총 cycle은 20사이클로 설정한 후, testbench를 돌리고 필요한 정보만 사진으로 캡처하면, 위와 같이,

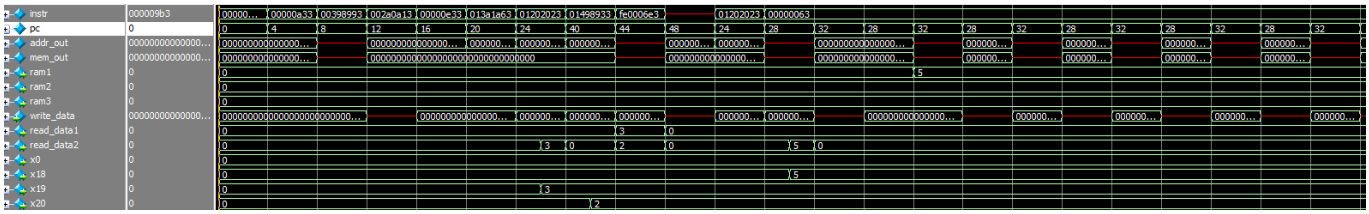

Figure 2: S-1 Testbench

우리가 생각한대로, PC shift를 하고, ALU를 진행한 후, ram1에 store하는 모습을 볼 수 있었다. (사진/tb file은 zip file로 같이 첨부하겠습니다.)

2.2 Scenario 2 - Targetting a Data Hazard and verifying every instruction in PPT

이번에는 C언어 없이 간단하게 assembly로만 생각하자. 먼저 어떠한 register에 하나의 값을 저장하고, 이후 그것을 ram에 저장, 이후 그것을 load하는 과정을 생각해보자. 그렇게 어렵지 않다. 이때,
(Data Hazard를 check 하기위해, ALU operation을 연속으로 해보자.)

1. Risc-V Assembly

```
0 : add x19, x0, x0
4 : add x20, x0, x0
8 : addi x19, x19, 13
12 : addi x20, x20, 5
16 : sub x19,x19,x20
20 : add x19,x19,x20
24 : sw x19,0(x0)
28 : lw x18,0(x0)
32 : and x18, x18,x20
36 : or x17,x18,x19
40 : sll x16,x18,x19
44 : srl x16,x18,x19
48 : sra x16,x18,x19
52 : andi x16,x18,2
56 : ori x16, x18, 2
60 : sw x16,1(x0)
```

2. Change to machine language

1. 000009B3
2. 00000A33
3. 00D98993

4. 005A0A13
5. 414989B3
6. 014989B3
7. 01302023
8. 00002903
9. 01497933
10. 013968B3
11. 01391833
12. 01395833
13. 41395833
14. 00297813
15. 00296813
16. 010020A3

와 같고, 이에 대하여 test bench를 진행하면,

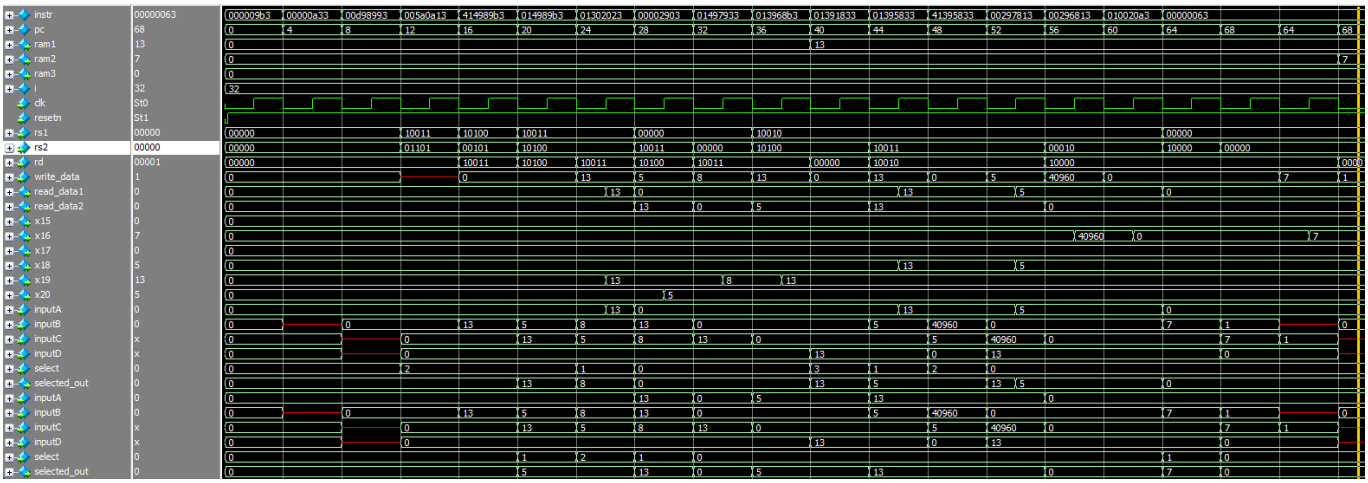

Figure 3: S-2 Testbench

와 같이 아주 정확하게 맞아 떨어지며, Forwarding unit이 정확하게 작동하여 위와 같이 CPU가 잘 동작하는 것을 확인할 수 있다. 또한 ram1, ram2에 정확히 저장이 된다. 이를 통해 모든 functionality와 data-Hazard 를 targeting한 processor에 대한 Validation이 완료되었다.

3 Reference

1. T.W.Seo (2021). Computer Architecture. Seoul, Korea: Hong pup science.
2. J.H. Kung (2023) Asic design Course materials