



## 1. 实验目的和要求

---

### Purpose

- Understand the principle of Cache Management Unit (CMU) and State Machine of CMU
- Master the design methods of CMU and Integrate it to the CPU.
- Master verification methods of CMU and compare the performance of CPU when it has cache or not

### Task

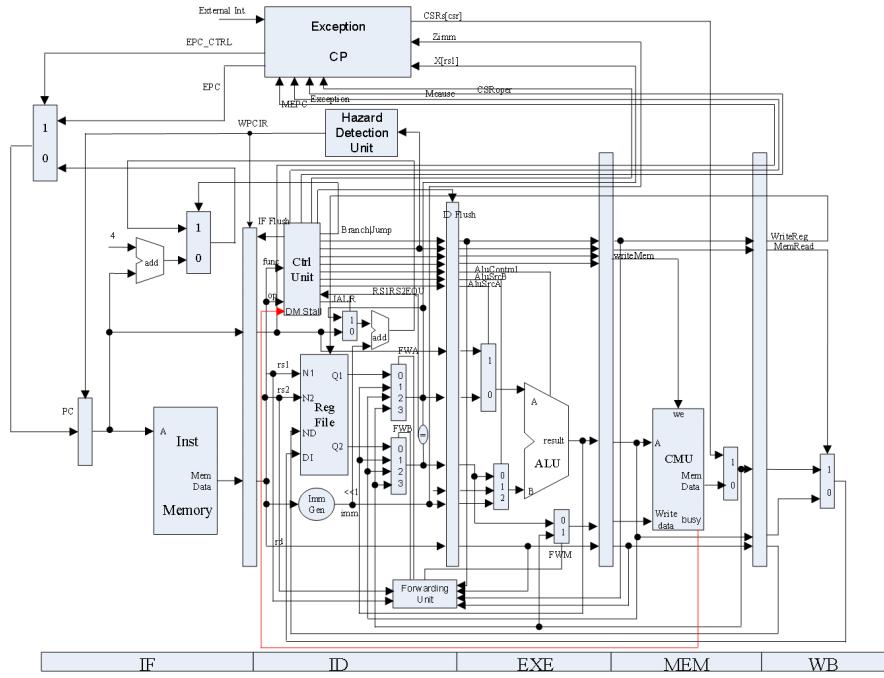
- Design of Cache Management Unit and integrate it to CPU
- Observe and Analyze the Waveform of Simulation
- Compare the performance of CPU when it has cache or not

## 2. 实验内容和原理

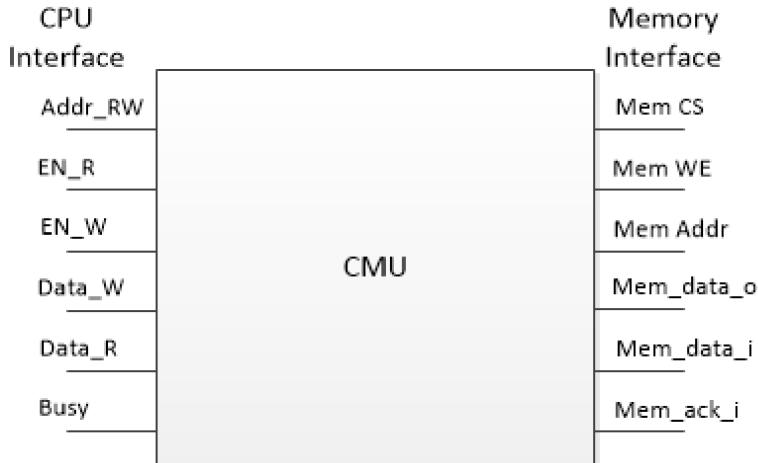
---

### CMU

CMU, cache management unit, is the unit controlling the behavior of cache. The memory in the previous pipelined CPU is replaced by a CMU in this experiment. Exactly, CPU processes data in the cache.



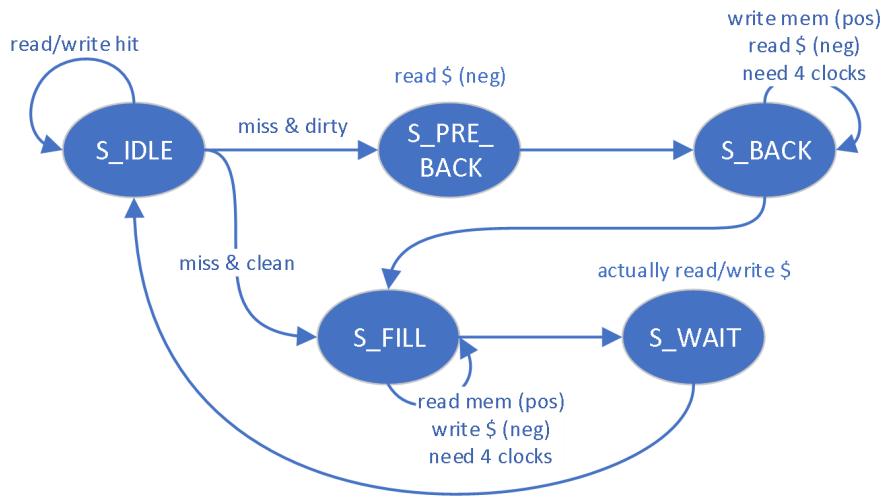
A CMU provides many ports for communicating with CPU and memory. If there is a data miss and CMU is loading data from memory, CPU should stall to wait for the data to be ready.



## Cache Management State Machine

All the cache operations occur at the negative edge and all the memory operations occur at the negative edge. When the cache interacts with memory, CMU should change its state only after receiving acknowledgement signal from memory.

- **S\_IDLE**: no memory operations in this state
- **S\_PRE\_BACK**: read cache once to get the data to be written back
- **S\_BACK**: write data back to memory in several cycles
- **S\_FILL**: load data to cache from memory in several cycles
- **S\_WAIT**: read out the newly arrived data



### 3. 实验过程和数据记录

There is only one module to be filled out.

#### cmu.v

In the sequential block controlling state, there are only a few blanks to be filled out.

Before finishing other code, we should notice that there is a block updating state and the counter at the positive edge.

```

always @ (posedge c1k) begin
    if (rst) begin
        state <= S_IDLE;
        word_count <= 2'b00;
    end
    else begin
        state <= next_state;
        word_count <= next_word_count;
    end
end

```

According to the state diagram, if there is a hit, the machine remains `S_IDLE`. If there is a miss and the data to be replaced is dirty, the dirty line should be written back to memory at first, thus the machine entering `S_PRE_BACK` reading the dirty data. If not, the data is immediately loading in, and the machine enters `S_FILL`.

```

S_IDLE: begin
    if (en_r || en_w) begin
        if (cache_hit)
            next_state = S_IDLE;
        else if (cache_valid && cache_dirty)
            next_state = S_PRE_BACK;
        else
            next_state = S_FILL;
    end
    next_word_count = 2'b00;
end

```

According to the diagram, there is only one destination after this state. The machine enters `S_BACK` writing dirty data back.

```
S_PRE_BACK: begin
    next_state = S_BACK;
    next_word_count = 2'b00;
end
```

According to the diagram, the machine is writing back dirty data. The counter increments only after the acknowledgement is received. If the last acknowledgement is received, the machine turns to `S_FILL` loading new data. If not, the machine remains `S_BACK`.

```
S_BACK: begin
    if (mem_ack_i && word_count == {ELEMENT_WORDS_WIDTH{1'b1}})      // 2'b11 in
default case
    next_state = S_FILL;
else
    next_state = S_BACK;

if (mem_ack_i)
    next_word_count = word_count + 2'b01;
else
    next_word_count = word_count;
end
```

According to the diagram, the counter increments only after the acknowledgement is received. If the last acknowledgement is received, the machine turns to `S_WAIT`. If not, the machine remains `S_FILL`.

```
S_FILL: begin
    if (mem_ack_i && word_count == {ELEMENT_WORDS_WIDTH{1'b1}})
        next_state = S_WAIT;
else
    next_state = S_FILL;

if (mem_ack_i)
    next_word_count = word_count + 2'b01;
else
    next_word_count = word_count;
end
```

According to the diagram, reset the counter and turn the machine to `S_IDLE`. (This is simplified here according to my comprehension)

```
S_WAIT: begin
    next_state = S_IDLE;
    next_word_count = 2'b00;
end
```

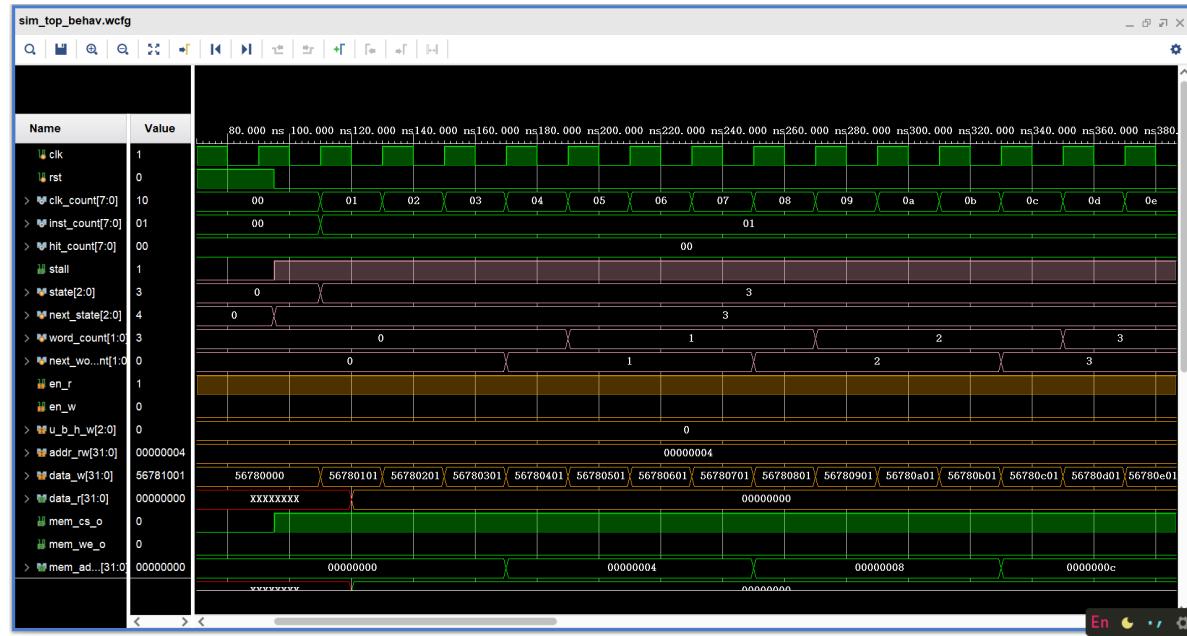
At last, if the machine is not turning into `S_IDLE`, which indicates there is a miss and a latency for required data is necessary, CPU should stall to wait for the missing data.

```
assign stall = (next_state != S_IDLE);
```

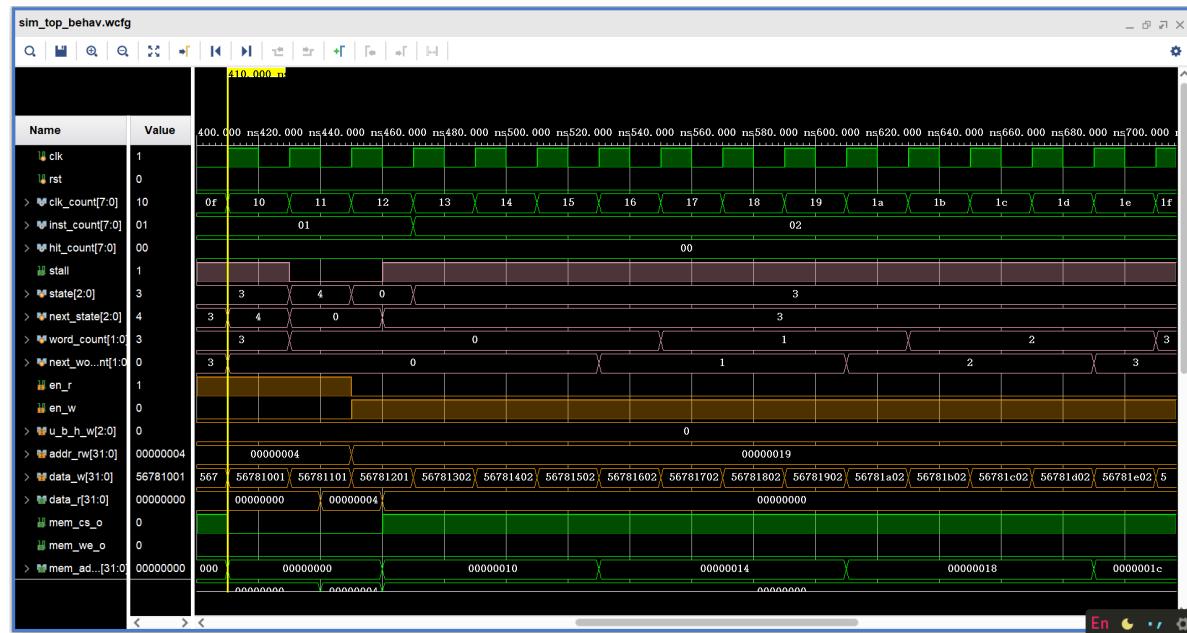
## 4. 实验结果分析

### Simulation

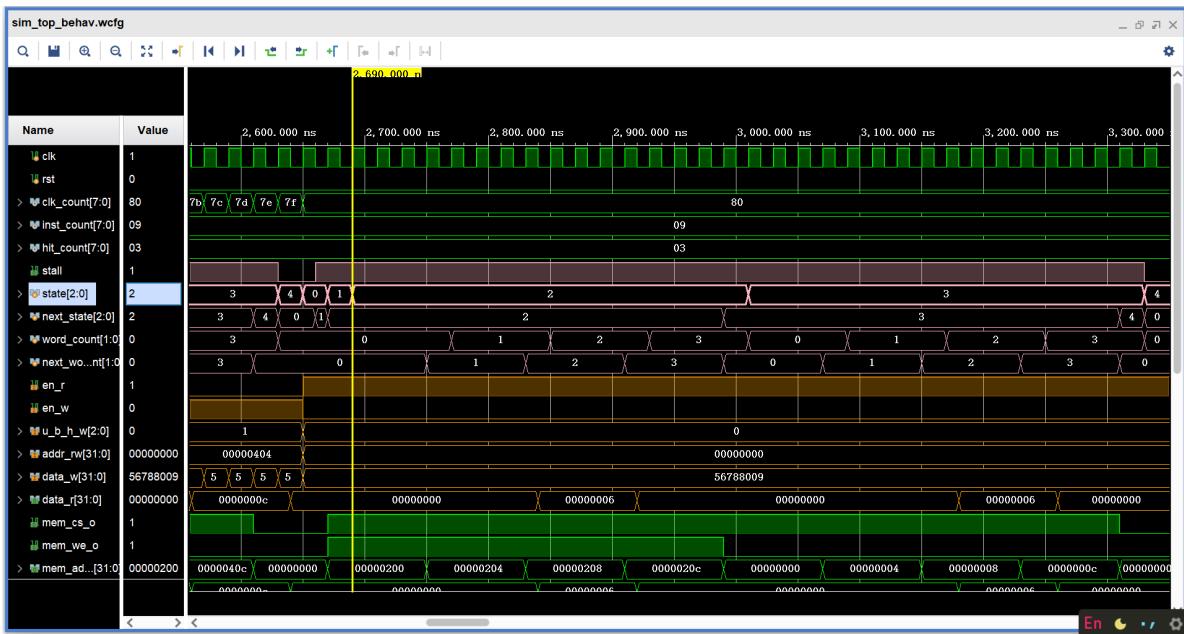
Run the simulation and we can see the result below. The first `l1b` instruction causes a miss and we can see `stall` signal is set for the miss. Since this is the first miss and there is no data in the cache, we can see the state changes from `S_IDLE` to `S_FILL` immediately. The counter increments per cycle, which indicates data is being loaded from memory.



We can see state `S_FILL` lasts for 16 cycles. This is because the bandwidth between memory and cache is one word per 4 cycles, and one cache line consists of 4 words, which is 16 bytes. Cache needs 16 cycles to complete loading a cache line from memory or writing a cache line to memory.



The following figure shows that if the data to be replaced is dirty, the design also takes 16 cycles to write back dirty data. The other part of the result is the same as that given in the guidance, so is omitted here, the analysis of which is similar to this part.



## Program Device

The first instruction causing a miss just enters phase MEM. At this time, the state of CMU is still `S_IDLE`.

```

Zhejiang University Computer Organization Experimental
SOC Test Environment (With RISC-V)

x0:zero 00000000  x01: ra 00000000  x02: sp 00000000  x03: gp 00000000
x04: tp 00000000  x05: t0 00000000  x06: t1 00000000  x07: t2 00000000
x8:fps0 00000000  x09: s1 00000000  x10: a0 00000000  x11: a1 00000000
x12: a2 00000000  x13: a3 00000000  x14: a4 00000000  x15: a5 00000000
x16: a6 00000000  x17: a7 00000000  x18: s2 00000000  x19: s3 00000000
x20: s4 00000000  x21: s5 00000000  x22: s6 00000000  x23: s7 00000000
x24: s8 00000000  x25: s9 00000000  x26:s10 00000000  x27:s11 00000000
x28: t3 00000000  x29: t4 00000000  x30: t5 00000000  x31: t6 00000000
PC---IF 00000010  INST-IF 01C04203  rs1Data 00000000  rs2Data 00000000
PC---ID 0000000C  INST-ID 01C02183  rs1Addr 00000000  rs2Addr 0000001C
PC---EXE 00000008  INST-EX 01C01103  OMU-RAM 00000000  PCJumpA 00000028
PC---MEM 00000004  INST--M 01C00083  B/PCE-S 00000000  D/C-Hzd 00000000
PC---WB 00000000  INST-WB 00000013  I/ABSel 00010001  PCIFNxt 00000014
ALU-Ain 00000000  ALU-Out 0000001C  CPUAddr 00000000  ALUCtrl 00000001
ALU-Bin 0000001C  WB-Data 00000000  CPU-Dai 00000000  WR-MIO 00000001
Imm32 ID 0000001C  WB-Addr 00000000  CPU-DAO 00000000  RegW/DR 00010000
CODE-00 00000013  lbu x04,x00,01CH  CODE-03 01C02183
CODE-04 01C04203  lw x03,x00,01CH  CODE-07 00000000
CODE-08 00000000  1h x02,x00,01CH  CODE-0B 00000000
CODE-0C 00000000  lb x01,x00,01CH  CODE-0F 00000000
CODE-10 00000000  nop JStall:addi0  CODE-13 00000000
CODE-14 00000000  CODE-15 00000000  CODE-16 00000000  CODE-17 00000000
CODE-18 00000000  CODE-19 00000000  CODE-1A 00000000  CODE-1B 00000000
CODE-1C 00000000  CODE-1D 00000000  CODE-1E 00000000  CODE-1F 00000000
CODE-20 00000000  CODE-21 00000000  CODE-22 00000000  CODE-23 00000000
CODE-24 00000000  CODE-25 00000000  CODE-26 00000000

```

Then we can see, in the next cycle, CPU stalls. CMU changes to state `S_FILL`, and the counter of RAM begins to increment as needed data is loaded to chche.

```

Zhejiang University Computer Organization Experimental
SOC Test Environment (With RISC-V)

x0:zero 00000000  x01: ra 00000000  x02: sp 00000000  x03: gp 00000000
x04: tp 00000000  x05: t0 00000000  x06: t1 00000000  x07: t2 00000000
x8:fps0 00000000  x09: s1 00000000  x10: a0 00000000  x11: a1 00000000
x12: a2 00000000  x13: a3 00000000  x14: a4 00000000  x15: a5 00000000
x16: a6 00000000  x17: a7 00000000  x18: s2 00000000  x19: s3 00000000
x20: s4 00000000  x21: s5 00000000  x22: s6 00000000  x23: s7 00000000
x24: s8 00000000  x25: s9 00000000  x26: s10 00000000 x27: s11 00000000
x28: t3 00000000  x29: t4 00000000  x30: t5 00000000  x31: t6 00000000
PC---IF 00000010  INST-IF 01C04203  rs1Data 00000000  rs2Data 00000000
PC---ID 0000000C  INST-ID 01C02183  rs1Addr 00000000  rs2Addr 0000001C
PC--EXE 00000008  INST-EX 01C01103  CMU-RAM 00030001  PCJumpA 00000028
PC--MEM 00000004  INST-M 01C00083  B/PCE-S 00000000  D/C-Hzd 00000000
PC--WB 00000000  INST-WB 00000013  I/ABSel 00010001  PCIFNxt 00000014
ALU-Ain 00000000  ALU-Out 0000001C  CPUAddr 00000000  ALUCtrl 00000001
ALU-Bin 0000001C  WB-Data 00000000  CPU-Dai 00000000  WR--MIO 00000001
Imm32ID 0000001C  WB-Addr 00000000  CPU-DAo 00000000  RegW/DR 00010000
CODE-00 00000013  lbu x04,x00,01CH
CODE-04 01C04203  lw x03,x00,01CH
CODE-08 00000000  lh x02,x00,01CH
CODE-0C 00000000  lb x01,x00,01CH
CODE-10 00000000  nop JStall: addi0
CODE-14 00000000  CODE-15 00000000  CODE-16 00000000  CODE-13 00000000
CODE-18 00000000  CODE-19 00000000  CODE-1A 00000000  CODE-17 00000000
CODE-1C 00000000  CODE-1D 00000000  CODE-1E 00000000  CODE-1B 00000000
CODE-20 00000000  CODE-21 00000000  CODE-22 00000000  CODE-1F 00000000
CODE-24 00000000  CODE-25 00000000  CODE-26 00000000  CODE-23 00000000
CODE-27 00000000

```

Loading continues. As the analysis of the simulation, the counter of RAM goes from 0 to 3 for 4 times, which indicates 4 words are loaded at a time and it takes 16 cycles.

```

Zhejiang University Computer Organization Experimental
SOC Test Environment (With RISC-V)

x0:zero 00000000  x01: ra 00000000  x02: sp 00000000  x03: gp 00000000
x04: tp 00000000  x05: t0 00000000  x06: t1 00000000  x07: t2 00000000
x8:fps0 00000000  x09: s1 00000000  x10: a0 00000000  x11: a1 00000000
x12: a2 00000000  x13: a3 00000000  x14: a4 00000000  x15: a5 00000000
x16: a6 00000000  x17: a7 00000000  x18: s2 00000000  x19: s3 00000000
x20: s4 00000000  x21: s5 00000000  x22: s6 00000000  x23: s7 00000000
x24: s8 00000000  x25: s9 00000000  x26: s10 00000000 x27: s11 00000000
x28: t3 00000000  x29: t4 00000000  x30: t5 00000000  x31: t6 00000000
PC---IF 00000010  INST-IF 01C04203  rs1Data 00000000  rs2Data 00000000
PC---ID 0000000C  INST-ID 01C02183  rs1Addr 00000000  rs2Addr 0000001C
PC--EXE 00000008  INST-EX 01C01103  CMU-RAM 00030003  PCJumpA 00000028
PC--MEM 00000004  INST-M 01C00083  B/PCE-S 00000000  D/C-Hzd 00000000
PC--WB 00000000  INST-WB 00000013  I/ABSel 00010001  PCIFNxt 00000014
ALU-Ain 00000000  ALU-Out 0000001C  CPUAddr 00000000  ALUCtrl 00000001
ALU-Bin 0000001C  WB-Data 00000000  CPU-Dai 00000000  WR--MIO 00000001
Imm32ID 0000001C  WB-Addr 00000000  CPU-DAo 00000000  RegW/DR 00010000
CODE-00 00000013  lbu x04,x00,01CH
CODE-04 01C04203  lw x03,x00,01CH
CODE-08 00000000  lh x02,x00,01CH
CODE-0C 00000000  lb x01,x00,01CH
CODE-10 00000000  nop JStall: addi0
CODE-14 00000000  CODE-15 00000000  CODE-16 00000000  CODE-13 00000000
CODE-18 00000000  CODE-19 00000000  CODE-1A 00000000  CODE-17 00000000
CODE-1C 00000000  CODE-1D 00000000  CODE-1E 00000000  CODE-1B 00000000
CODE-20 00000000  CODE-21 00000000  CODE-22 00000000  CODE-1F 00000000
CODE-24 00000000  CODE-25 00000000  CODE-26 00000000  CODE-23 00000000
CODE-27 00000000

```

Finally, CMU reaches state `S_WAIT`.



CMU returns to state S\_IDLE and CPU goes on.



The following figures show that if there is a hit, CPU works continually.

```

Zhejiang University Computer Organization Experimental
SOC Test Environment (With RISC-V)

x0:zero 00000000    x01: ra FFFFFFFF0    x02: sp FFFFF0F0    x03: gp F0F0F0F0
x04: tp 00000000    x05: t0 00000000    x06: t1 00000000    x07: t2 00000000
x8:fps0 00000000    x09: s1 00000000    x10: a0 00000000    x11: a1 00000000
x12: a2 00000000    x13: a3 00000000    x14: a4 00000000    x15: a5 00000000
x16: a6 00000000    x17: a7 00000000    x18: s2 00000000    x19: s3 00000000
x20: s4 00000000    x21: s5 00000000    x22: s6 00000000    x23: s7 00000000
x24: s8 00000000    x25: s9 00000000    x26: s10 00000000   x27: s11 00000000
x28: t3 00000000    x29: t4 00000000    x30: t5 00000000    x31: t6 00000000
PC---IF 0000001C    INST-IF ABCDE0B7    rs1Data 00000000    rs2Data 00000000
PC---ID 00000018    INST-ID 21002003   rs1Addr 00000000    rs2Addr 00000010
PC--EXE 00000014    INST-EX 01C05283   CMU-RAM 00000000    PCJumpA 00000228
PC--MEM 00000018    INST--M 01C04203   B/PCE-S 00000100    D/C-Hzd 00000000
PC---WB 0000000C    INST-WB 01C02183   I/ABSel 00010001    PCIFNxt 00000020
ALU-Ain 00000000    ALU-Out 0000001C    CPUAddr 00000000    ALUCtrl 00000001
ALU-Bin 0000001C    WB-Data F0F0F0F0    CPU-Dai 000000F0    WR-MIO 00000001
Imm32ID 00000210   WB-Addr 00000003    CPU-Dao 00000000    RegW/DR 00010001
CODE-00 00000013    lui x01,ABCDEH
CODE-04 01C04203   lw x00,x00,210H
CODE-08 00000000    lui x05,x00,81CH
CODE-0C 00000000    lwu x04,x00,81CH
CODE-10 00000000    lui x03,x00,81CH
CODE-14 00000000    CODE-15 00000000    CODE-16 00000000    CODE-13 00000000
CODE-18 00000000    CODE-19 00000000    CODE-1A 00000000    CODE-17 00000000
CODE-1C 00000000    CODE-1D 00000000    CODE-1E 00000000    CODE-1B 00000000
CODE-20 00000000    CODE-21 00000000    CODE-22 00000000    CODE-1F 00000000
CODE-24 00000000    CODE-25 00000000    CODE-26 00000000    CODE-23 00000000
                                         CODE-27 00000000

```

```

Zhejiang University Computer Organization Experimental
SOC Test Environment (With RISC-V)

x0:zero 00000000    x01: ra FFFFFFFF0    x02: sp FFFFF0F0    x03: gp F0F0F0F0
x04: tp 000000F0    x05: t0 00000000    x06: t1 00000000    x07: t2 00000000
x8:fps0 00000000    x09: s1 00000000    x10: a0 00000000    x11: a1 00000000
x12: a2 00000000    x13: a3 00000000    x14: a4 00000000    x15: a5 00000000
x16: a6 00000000    x17: a7 00000000    x18: s2 00000000    x19: s3 00000000
x20: s4 00000000    x21: s5 00000000    x22: s6 00000000    x23: s7 00000000
x24: s8 00000000    x25: s9 00000000    x26: s10 00000000   x27: s11 00000000
x28: t3 00000000    x29: t4 00000000    x30: t5 00000000    x31: t6 00000000
PC---IF 00000020    INST-IF 71C08093   rs1Data 00000000    rs2Data 00000000
PC---ID 0000001C    INST-ID ABCDE0B7   rs1Addr 00000018    rs2Addr 0000001C
PC--EXE 00000018    INST-EX 21002003   CMU-RAM 00000000    PCJumpA ABCDE01C
PC--MEM 00000014    INST--M 01C05283   B/PCE-S 00000100    D/C-Hzd 00000000
PC---WB 00000018    INST-WB 01C04203   I/ABSel 00050001    PCIFNxt 00000024
ALU-Ain 00000000    ALU-Out 0000001C    CPUAddr 00000000    ALUCtrl 0000000C
ALU-Bin 000000210   WB-Data 000000F0    CPU-Dai 000000F0    WR-MIO 00000001
Imm32ID ABCDE000   WB-Addr 00000004    CPU-Dao 00000000    RegW/DR 00010001
CODE-00 00000013    addi x01,x01,71CH
CODE-04 01C04203   lui x01,ABCDEH
CODE-08 71C08093   lw x00,x00,210H
CODE-0C 00000000    lui x05,x00,81CH
CODE-10 00000000    lwu x04,x00,81CH
CODE-14 00000000    CODE-15 00000000    CODE-16 00000000    CODE-13 00000000
CODE-18 00000000    CODE-19 00000000    CODE-1A 00000000    CODE-17 00000000
CODE-1C 00000000    CODE-1D 00000000    CODE-1E 00000000    CODE-1B 00000000
CODE-20 00000000    CODE-21 00000000    CODE-22 00000000    CODE-1F 00000000
CODE-24 00000000    CODE-25 00000000    CODE-26 00000000    CODE-23 00000000
                                         CODE-27 00000000

```

## 5. 讨论与心得

This experiment is relatively easy since there is not much work to do and all the blanks can be filled out according to the state diagram. It just takes some effort to completely understand how CMU works.

