CPU Report

1. Original operations

1.1 MUL

The function of this instruction is to multiply the value of r2 register by the value of r3 register and store it in r1 register.

When a bit of the multiplier a is 1 (0 can be ignored, because the result of multiplying 0 and b is also 0), the result of multiplying b is b (16 bits). After adding 16 zeros in front, the result is 32 bits, and then moving i-1 bits to the left, that is, moving the i-1 zeros in front of the result to the last complement.

MUL | r1(3 bit) | 1'b0, r2(3 bit) | 1'b0, r3(3 bit) |

1.2 MOVE

The function of this instruction is to copy the value of r2 register to r1 register.

The structure of the MOVE instruction and the CMP instruction are similar. It is just that the operation registers are different, and an additional step of writing back to register r1 is added.

MOVE | r1(3 bit) | 1'b0, r2(3 bit) | 4'bxxxx |

2. Machine programs

2.1 Greatest common divisor

2.1.1 Algorithm process

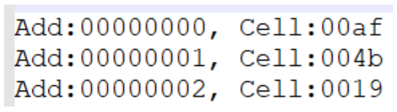
I use Euclidean algorithm to find the GCD of two numbers, and it works by repeatedly subtracting the smaller number from the larger number until the difference is 0. The process is shown in algorithm 1.

Algorithm 1: The process of finding greatest common divisor

```
Input: a, b
Output: The greatest common divisor of the input
$t \leftarrow abs(a-b)$
while $t != 0$ do
    $a \leftarrow b$
    $b \leftarrow t$
    $t \leftarrow abs(a-b)$
end
return b
```

2.2.2 Simulation result

As shown in Figure 1. The first and second lines are the two numbers input(175, 75). The third row is the greatest common divisor of two numbers(25).



```
Add:00000000, Cell:00af
Add:00000001, Cell:004b
Add:00000002, Cell:0019
```

Figure 1: Simulation result of GCD

2.2 Factorial

2.2.1 Algorithm process

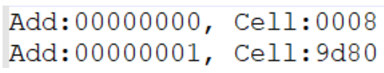
The process is shown in algorithm 2. This program can only calculate factorials within 8.

Algorithm 2: The process of factorial

```
Input: a
Output: The factorial of the input
$result \leftarrow 1$
for $i = 1$ to $a$ do
    $result \leftarrow result * i$
end
return result
```

2.2.2 Simulation result

As shown in Figure 2. The first line is the input number(8), and the second line is the calculation result(40320).



```
Add:00000000, Cell:0008
Add:00000001, Cell:9d80
```

Figure 2: Simulation result of factorial

2.3Bubble sort

2.3.1 Algorithm process

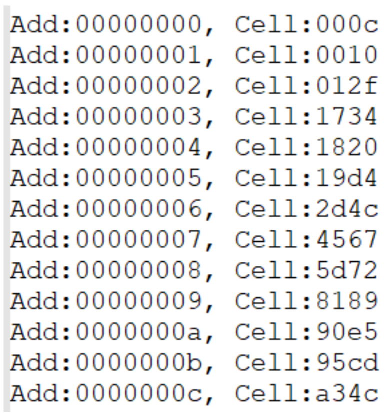
I use the bubble sorting algorithm to get the ascending array for the input array. The process is shown in algorithm 3.

Algorithm 3: The process of bubble sort

```
Input: n(The number of input array), array
Output: The sorted array
for $i = 0$ to $n - 1$ do
    for $j = 0$ to $n - i - 1$ do
        if $arr[j] > arr[j + 1]$: swap $arr[j]$ and $arr[j + 1]$
    end
end
```

2.3.2 Simulation result

As shown in Figure 3. The first line is the number of numbers entered, followed by an array sorted in ascending order.



```
Add:00000000, Cell:000c
Add:00000001, Cell:0010
Add:00000002, Cell:012f
Add:00000003, Cell:1734
Add:00000004, Cell:1820
Add:00000005, Cell:19d4
Add:00000006, Cell:2d4c
Add:00000007, Cell:4567
Add:00000008, Cell:5d72
Add:00000009, Cell:8189
Add:0000000a, Cell:90e5
Add:0000000b, Cell:95cd
Add:0000000c, Cell:a34c
```

Figure 3: Simulation result of bubble sort

3. Consideration and impression

3.1 Consideration

- Data forwarding: first of all, the premise of data forwarding is that the previous instruction can write to the register, and the read register of the next register conflicts with the register written by the above instruction. So we can push the data forward according to the types of registers written and read. Like $RegA \leftarrow r1$, $RegA \leftarrow r2$, $RegB \leftarrow r3$. I've grouped the reads from the registers into these three categories and defined a function to better organize. If the two registers are consistent before the pipeline stops, they can be assigned directly without waiting for the final write step.
- Branch prediction: when writing the imi.ini program, sometimes I did not pay attention to the NOP after the branch instruction, which caused bugs. I have considered branch prediction to replace the software's three-step NOP instruction, but the ability is limited and it is too complicated, so I gave up.

3.2 Impression

- Multiplication instructions can only be simulated in VCS, and FPGA cannot be integrated. Therefore, the understanding of hardware algorithm is not enough, and the hardware thinking cannot be converted to software thinking.

4. Attachment

1. pcpu.v
2. bubble: files for bubble sort algorithm
3. gcd: files for greatest common divisor algorithm
4. factorial: files for factorial algorithm
5. RTL pcpu.png: images for RTL views of top and pcpu