ACA - HLS
Final Project Presentation

RISC-V CPU with Tomasulo Algorithm

Team 02
B07902123 蔡奇夆
B07209016 鐘晨瑋

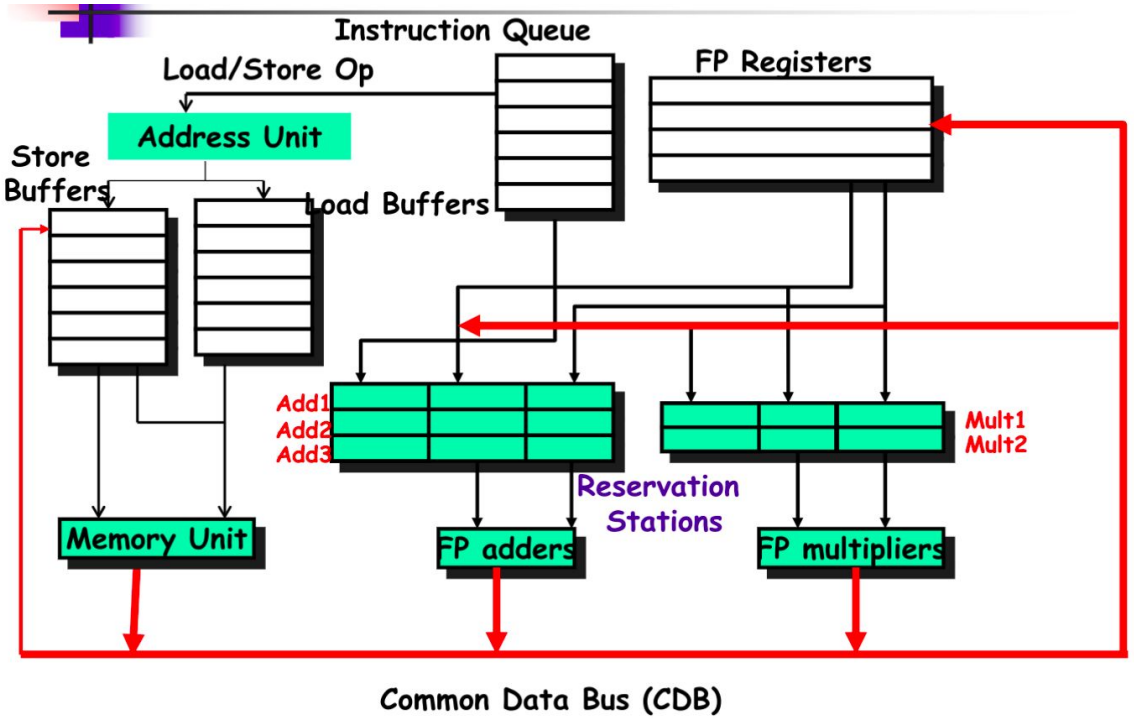
GitHub:
https://github.com/ChungChenWei/ACA-HLS-Final-Team2-Tomasulo-RISC-V

Outline

- Problem description
- Our implementation
- Discussion
- Lessons learned

Problem Description

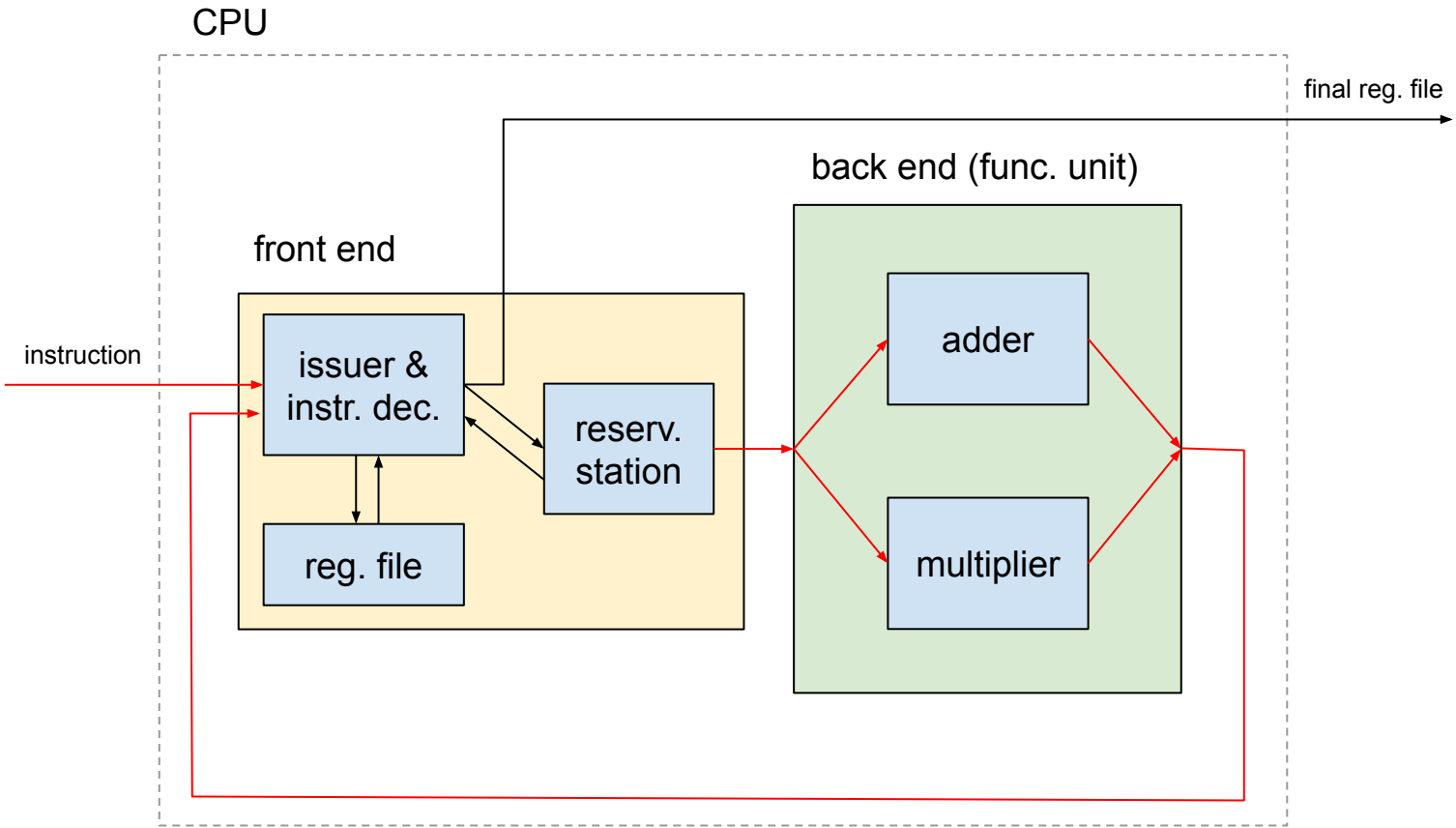
Tomasulo Algorithm



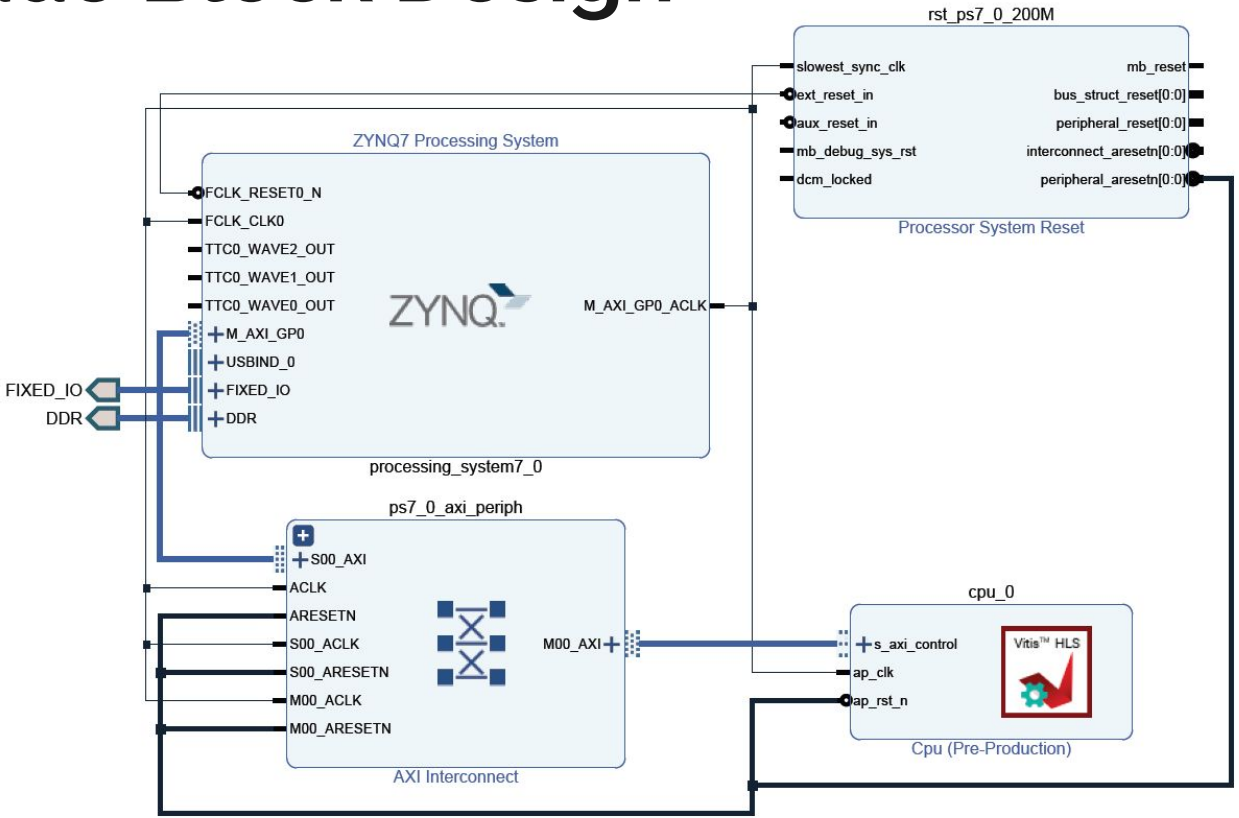
Goal

- Implement a RISC-V CPU with Tomasulo algorithm using Vitis HLS
- Run the above CPU on pynq-z2 board

Our Implementation


CPU
final reg. file
back end (func. unit)
front end
instruction
issuer &
instr. dec.
reserv.
station
reg. file
adder
multiplier

Vivado Block Design



Host program

```python
def main():
    # timeKernelStart = time()

    # write instruction
    for i in range(len(instrBuffer)) :
        cpuIP.write(INSTR_I + i * 4, int(instrBuffer[i]))

    # start signal
    cpuIP.write(AP_CONTR, AP_START)
    # until idle
    while (cpuIP.read(AP_CONTR) & AP_IDLE) == 0x0:
        continue

    # timeKernelEnd = time()
    # print(f"Kernel execution time: {str(timeKernelEnd - timeKernelStart)} s")
```

```
inst = """
ADDI x0, x0, 0
ADDI x1, x1, 10
ADDI x2, x2, 20
MUL x1, x1, x2
MUL x2, x2, x2
ADDI x3, x2, 10
MUL x4, x3, x1
ADD x5, x4, x4
ADDI x6, x6, 5
ADDI x7, x7, 6
MUL x10, x7, x6
ADDI x0, x0, 5
"""
```

Host program

```
inst = """
ADDI x0, x0, 0
ADDI x1, x1, 10
ADDI x2, x2, 20
MUL x1, x1, x2
MUL x2, x2, x2
ADDI x3, x2, 10
MUL x4, x3, x1
ADD x5, x4, x4
ADDI x6, x6, 5
ADDI x7, x7, 6
MUL x10, x7, x6
ADDI x0, x0, 5
"""
```

```
# final register status
    x0 = 0 , x1 = 200 , x2 = 400 , x3 = 410
    x4 = 82000 , x5 = 164000 , x6 = 5 , x7 = 6
    x8 = 0 , x9 = 0 , x10 = 30 , x11 = 0
    x12 = 0 , x13 = 0 , x14 = 0 , x15 = 0
    x16 = 0 , x17 = 0 , x18 = 0 , x19 = 0
    x20 = 0 , x21 = 0 , x22 = 0 , x23 = 0
    x24 = 0 , x25 = 0 , x26 = 0 , x27 = 0
    x28 = 0 , x29 = 0 , x30 = 0 , x31 = 0
```

```
x0  =     0     x1  =   200     x2  =   400     x3  =   410     x4  = 82000
x5  = 164000    x6  =     5     x7  =     6     x8  =     0     x9  =     0
x10 =    30     x11 =     0     x12 =     0     x13 =     0     x14 =     0
x15 =     0     x16 =     0     x17 =     0     x18 =     0     x19 =     0
x20 =     0     x21 =     0     x22 =     0     x23 =     0     x24 =     0
x25 =     0     x26 =     0     x27 =     0     x28 =     0     x29 =     0
x30 =     0     x31 =     0
```

Host program

10000 times	with pipeline off	without pipeline off
mean	0.1183	0.1166
std	0.0003	0.0045

```
void cpu (instr_t instruction_memory_i[INSTR_MEM_SIZE], data_t final_register_file_o[REGISTER_NUM]) {
#pragma HLS INTERFACE s_axilite port=return
    Register_file rf;
    int cycle = 0;
    unsigned PC = 0;

    while (PC < INSTR_MEM_SIZE && cycle < MAX_CYCLE_NUM) {
#pragma HLS PIPELINE off
        bool success = true;
        every_cycle(instruction_memory_i[PC], success, rf);
        cycle += 1;
        if (success) PC += 1;
    }

    rf.get_final_status(final_register_file_o);
}
```

Discussion

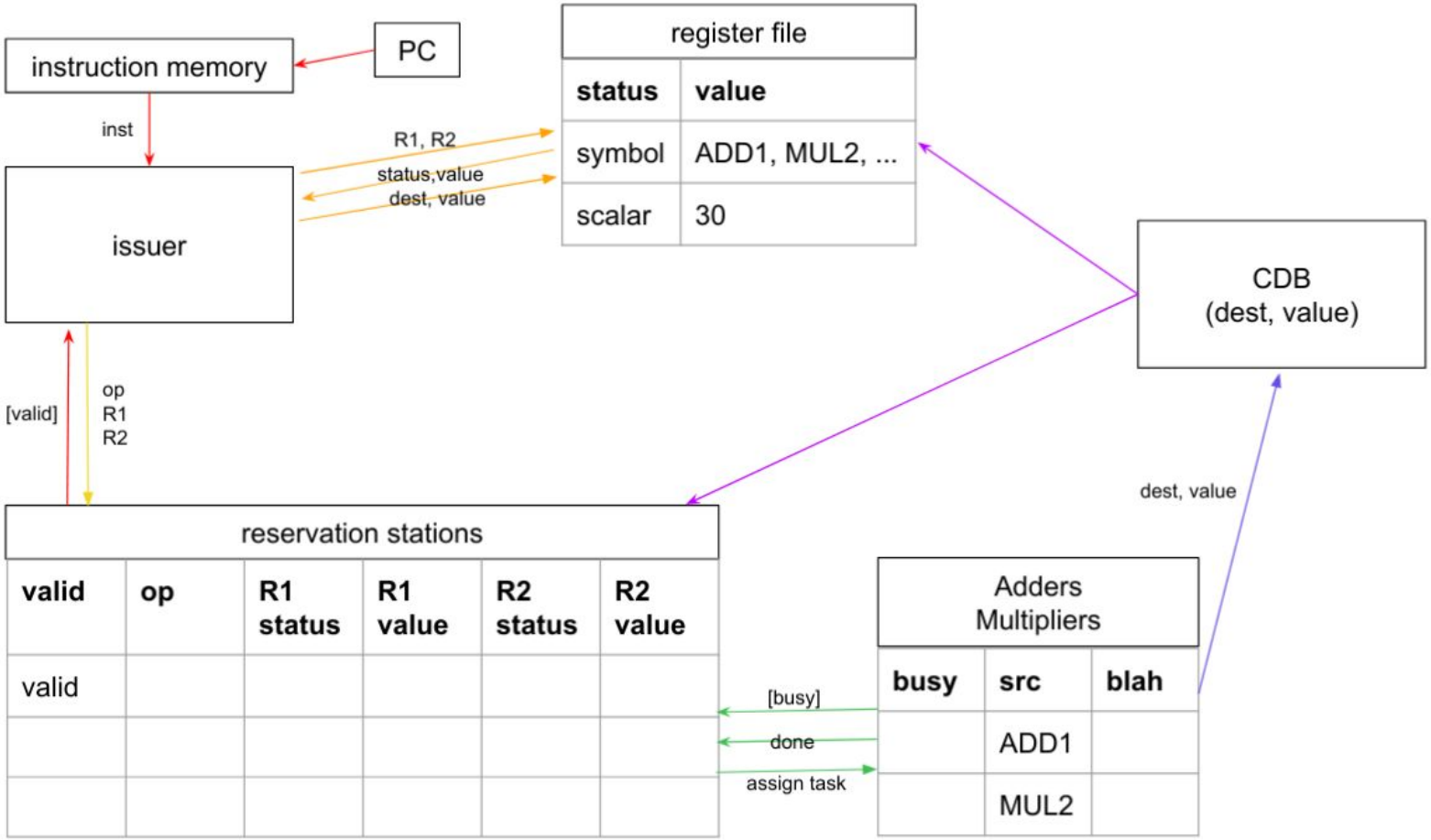
Subtitles

- the HLS directives used in our implementation
- the evolution of our design
- using HLS to do CPU design

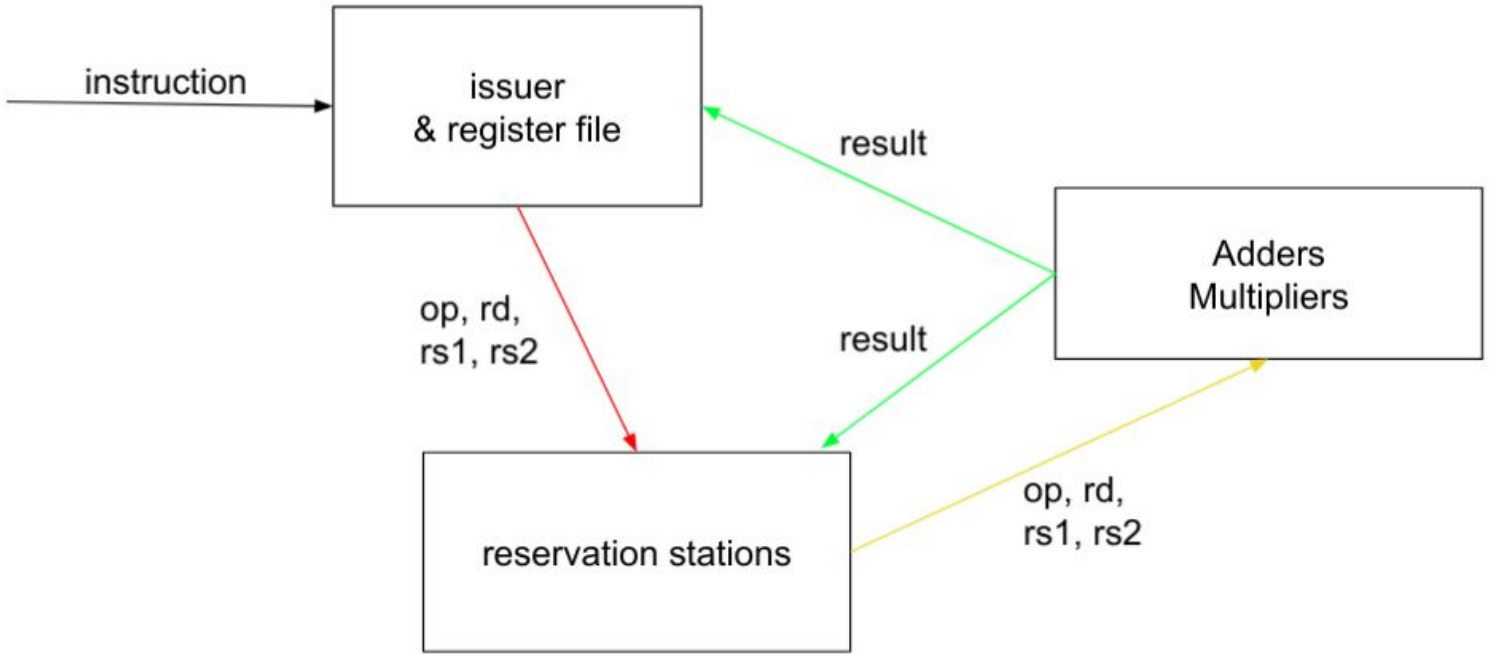
HLS Directives

- loop unroll & array partition
  - parallel access of independent registers
- function inlining
  - enables more scheduling optimization
- pipeline off
  - make sure our design sticks to Tomasulo alrogithm
- dataflow
  - give HLS compiler hints about parallelization

- allocation
  - limit the resource usage



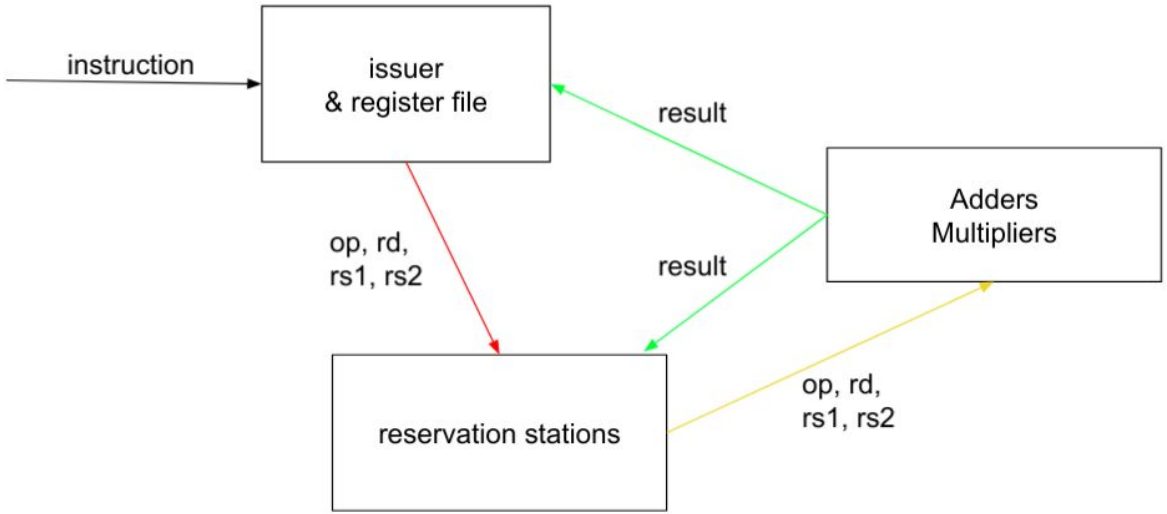
The Evolution of Our Design - 1st



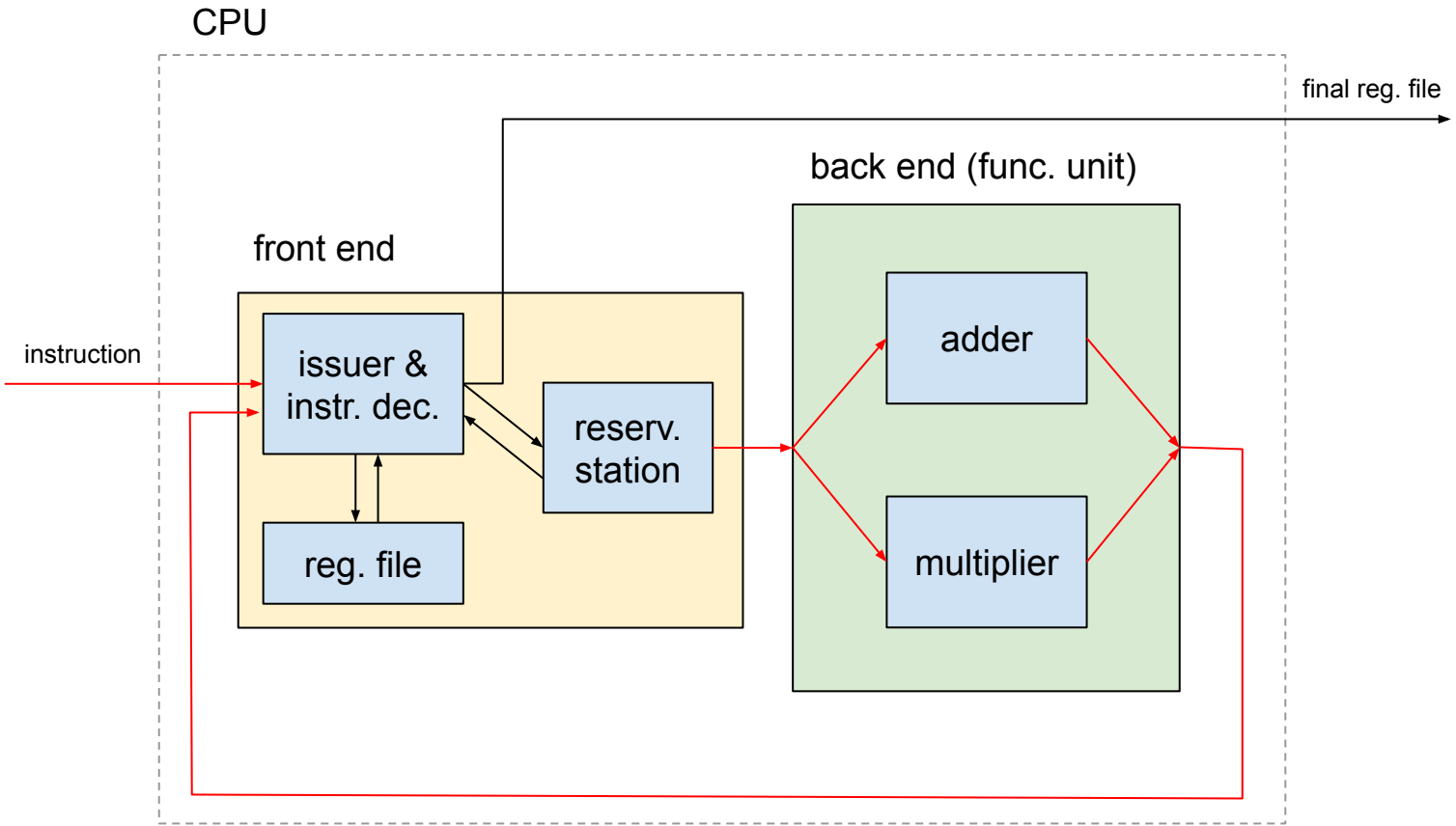
The Evolution of Our Design - 2nd

```
void issuer(hls::stream instr, data_o to_res_sta, data_i from_func_unit) {
    // this?
    if (instr.read()) {
        ...
    }

    // or this?
    while (instr.read()) {
        ...
    }
}
```



The Evolution of Our Design - 2nd (cont'd)



The Evolution of Our Design - 3rd

Use HLS to Design CPU

- In normal case, we can let HLS compiler do as much optimization as possible.
- In implementing a algorithm, HLS compiler may break the intended structure.

Lessons Learned

Lessons Learned

- HLS is not suitable for designing architecture with specific algorithmic requirements.
- When using HLS, think “high level”. Too much “verilog logic” will make the design infeasible.
- HLS can synthesize a “kernel function”.

Thanks for listening!