# *Simple CPU design*

Sida zhong
Computer science dept. Class 247
San Jose State University
San jose, United States

sida.zhong@sjsu.edu

***Abstract*—This project will design a simple CPU. The CPU memory is loaded with a loop program. This program will calculate the sum of 0 to a number N. For example, if the input number is N, the output is 0+1+2+...(N-2)+(N-1). The program is expressed in high-level language as for (i=0;i<N;i++;){result+=i;}. The purpose of this project is to understand how the high-level language is interpreted and run in the machine. The CPU is designed according to the von Neumann structure[1] and contains two main modules, the control unit and the processing unit. The processing unit contains Arithmetic Logic Unit (ALU[2]), a 2-bit address Register File (RF[3]), and a "magic" JUMP instruction module. The control unit contains a ROM[4] memory, the ROM contains the compiled binary code, which is the loop program, an Instruction Register[5] (IR) with a 4-bit address, and a Program Counter[6] (PC). The simulator uses Xilinx Vivado, the operating system is Linux Ubuntu, and the hardware description language is verilog. Initially the project planned to use VHDL, but later found that verilog is obviously more friendly than VHDL and closer to the high-level language syntax, so verilog was used instead. The simulated hardware chip model is basys3 XC7A35T-1CPG236C, which belongs to the Artix family, 28nm technology, and the market price is US$149.**

***Keywords*—Arithmetic Logic Unit*;* Register File*;* ROM*;* Instruction Register*;* Program Counter*;* Vivado*;* verilog; CPU cycle.**

[1] also known as the von Neumann model or Princeton architecture — is a computer architecture based on a 1945 description by John von Neumann and others in the First Draft of a Report on the EDVAC.

[2] In computing, an arithmetic logic unit (ALU) is a combinational digital circuit that performs arithmetic and bitwise operations on integer binary numbers.

[3] A register file is an array of processor registers in a central processing unit. Register banking is the method of using a single name to access multiple different physical registers depending on the operating mode.

[4] Read-only memory (ROM) is a type of non-volatile memory used in computers and other electronic devices. Data stored in ROM cannot be electronically modified after the manufacture of the memory device.

[5] In computing, the instruction register or current instruction register is the part of a CPU's control unit that holds the instruction currently being executed or decoded.

[6] The program counter, commonly called the instruction pointer in Intel x86 and Itanium microprocessors, and sometimes called the instruction address register, the instruction counter, or just part of the instruction sequencer, is a processor register that indicates where a computer is in its program sequence.

## I. Compiler and Instructions

In the early design of the chip, the program and the data were completely separated, and the program was based on the hardware. This makes the computer very inflexible. A computer can only do one thing at the same time. If the circuit is designed as a printer, it cannot be a calculator. The von Neumann structure completely changed this structure. The program was directly imported into the memory as data instead of being designed in the circuit, which directly led to the birth of software and programmers. Therefore, the first thing to do is to parse the high-level programming language into OPCODE. Because the program is very simple, it does not need to use any specific high-level programming language and parser, pseudo code is good enough for this project.

Fig. 1. High-level programming language to OPCODE

```
R3=10;
R0=0;
R1=0;
FOR (R0=0;R0<R3;R0++)
   R1+=R0;
```

```
ASSIGN INPUT TO R3
INIT RO=0
INIT R1=0
R1+=R0
R0++
IF RO<R3 THEN FALSE
IF FALSE THEN GO ADDR 03
OUTPUT R1
OVER
```

Obviously, this program needs 3 registers as variables for ALU operation, R0, R1, R3, which indicates that the address bit of RF is 2. It also needs the operation of ASSIGN, ADDITION, INCREMENT, JUMP, IF. After having

OPCODE, an instruction table is needed to summarize these OPCODE in binary code.

Fig. 2. instruction table

| INSTRUCTION | ENCODING |
|---|---|
| Raa=INPUT | 00001000aa |
| ASSIGN Raa=nnnnnnnn | 1aannnnnnn |
| ADD Rcc=Raa+Rbb | 0100ccbbaa |
| INC Raa++ | 0010aa0001 |
| IF Raa<Rbb THEN FALSE | 000111aabb |
| IF FALSE JUMP nnnn | 000101nnnn |
| OUTPUT=Raa | 00001001aa |
| OVER | 0000000000 |

Finally, these OPCODEs are merged into a compiled binary file according to the instruction table, ready to be loaded in the ROM later.

Fig. 3. instruction OPCODE

| ADDRESS | INSTRUCTION | OPCODE |
|---|---|---|
| 0000 | 0000100011 | ASSIGN INPUT TO R3 |
| 0001 | 1000000000 | INIT RO=0 |
| 0010 | 1010000000 | INIT R1=0 |
| 0011 | 0100010100 | R1+=R0 |
| 0100 | 0010000001 | R0++ |
| 0101 | 0001110011 | IF RO<R3 THEN FALSE |
| 0110 | 0001010011 | IF FALSE THEN GO ADDR 03 |
| 0111 | 0000100101 | OUTPUT R1 |
| 1000 | 0000000000 | OVER |

The suffix of the binary compiled file recognized by Vivado is .coe, and the format is as follows.

Fig. 4. Vivado instruction.coe

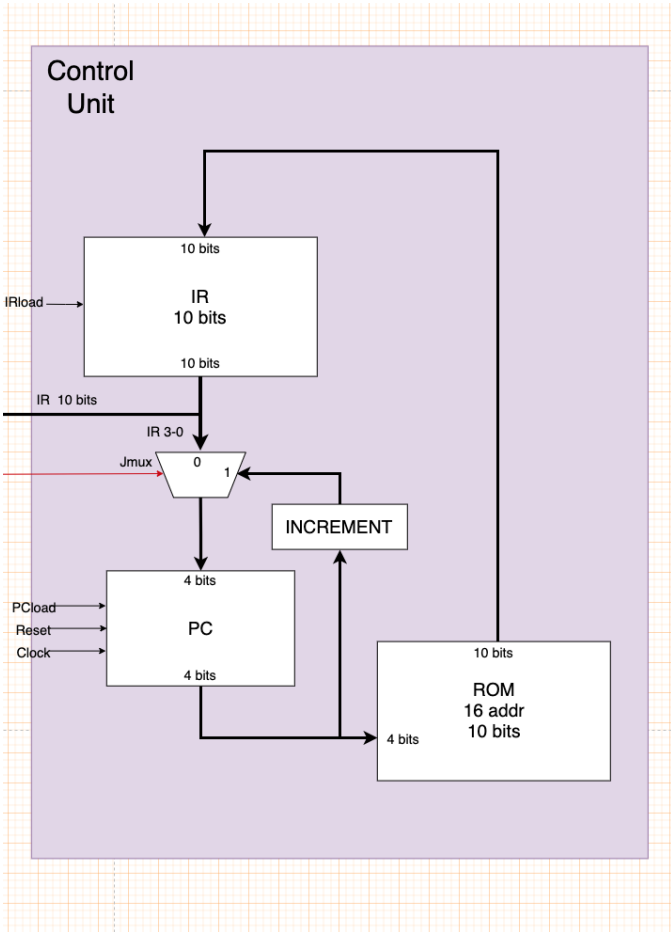
```
memory_initialization_radix=2;
memory_initialization_vector=
0000100011
1000000000
1010000000
0100010100
0010000001
0001110011
0001010011
0000100101
0000000000;

```

## II. CONTROL UNIT

The 3 main components of the control unit are IR, PC and ROM. The output is a 10-bit instruction, and the input is Jmux. Jmux is a signal that can control the instruction jump.

Fig. 5. Control Unit



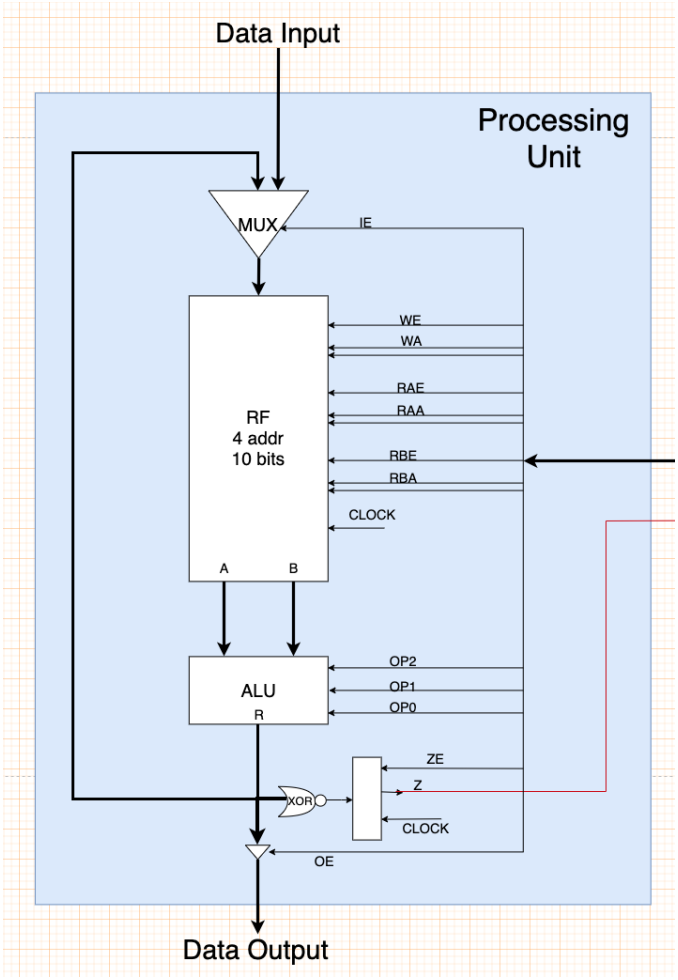

OPCODE will first be directly imported into the 4-bit address ROM. The address of the instruction will be recorded in the PC and driven by an increment module. The instruction from ROM will be recorded in IR to hold the instruction currently being executed. Because the program contains a JUMP instruction, and the 2-bit address of the JUMP instruction is determined by the last 4 bits of the IR instruction. For example, the 7th line of the instruction table is the JUMP command "IF FALSE JUMP nnnn: 00_0101_nnnn", which corresponds to "00_0101_0011 : IF FALSE THEN GO ADDR 03" on the 7th line of the OPCODE. The last 4 bits are 0011 (ADDR 03), which means that the instruction address in the PC needs to jump to 0011. The 0011 instruction address in OPCODE is "01_0001_0100: R1+=R0", R1+=R0 which is the body of the loop. This is how

loop operation is executed in the machine. Therefore, the Jmux signal determines whether the instruction address in the PC is incremented or jumps to the last 4 bits of the instruction address. Jmux is generated from the ALU module according to the instructions output by the Memory, which will be introduced in the next chapter.

## III. Processing unit

The two main components of the processing unit are RF and ALU. RF is a register storage that can be fast read and written. The control signals passed by the Control unit like WE(write enable), WA(write address), RAE(read A enable), RAA(read A address).... The output is two 10-bit data, A and B, as the input of ALU.

Fig. 6. Processing Unit



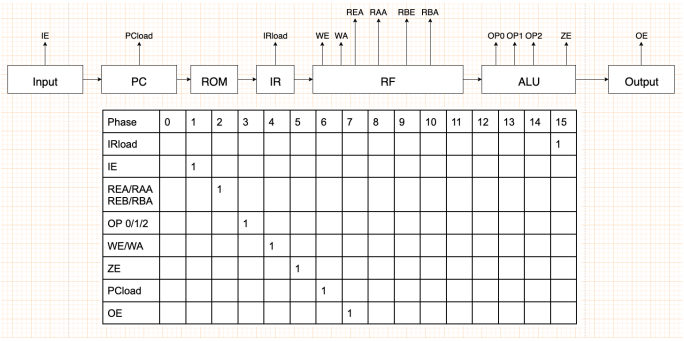

ALU is a combinational logic unit without a clock. The output is a pure function of the present input. When input A, B data comes in, it will directly generate output data R according to the input signals OP1, OP2, OP3. These 3 signals are output by the control unit. For example, the third row of the instruction table is the addition command "ADD Rcc=Raa+Rbb: 0100ccbbaa", which corresponds to the fourth row of OPCODE "0100010100: R1+=R0". The last 4 digits of 0100 are the signals OP2, OP1, OP0. Because there is only addition in ALU, 3 bits are enough. The first 6 bits 01_01_00 represent addresses R1, R1, and R0 respectively, which correspond exactly to R1,R1,R0. Actually the ALU only does the addition of R1+R0, the input result R is returned to RF and assigned to R1. These are the operations given by the instruction returned by the Control unit, and the details will be described again in the CPU chapter.

There is also a special operation in ALU. When the input A>B, the output data R will be 0. This logic corresponds to the XOR gate at the bottom of the circuit diagram, and a signal Z (Jmux) will be output as the PC address jump signal of the Control Unit. In the truth table of the XOR gate, all input signals are 0 and the output signal will be 1. When Jmux is 1, the PC of the Control Unit will carry out the increment command, which means that the JMP command will not be executed, and the OPCODE command will continue to advance, thus jumping out of the loop. In other words, the IF condition that jumps out of the loop in OPCODE is implemented with XOR gates. In a high-level language, this is like (IF A): return A, A can be any type of value. (ELSE A==0) return False, then only False can be returned.

## IV. CPU Pipeline

After the control unit and the processing unit are completed, the two modules can be connected together with the CPU, and a pipeline process is required. The pipeline of the CPU mainly has 3 steps. First, get the instruction from the IR, then decode the instruction corresponding to the control signal of each module, and finally connect the signal to the module to perform the operation.

Fig. 7. Pipeline




| Phase | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| IRload | | | | | | | | | | | | | | | | 1 |
| IE | | 1 | | | | | | | | | | | | | | |
| REA/RAA REB/RBA | | | 1 | | | | | | | | | | | | | |
| OP 0/1/2 | | | | 1 | | | | | | | | | | | | |
| WE/WA | | | | | 1 | | | | | | | | | | | |
| ZE | | | | | | 1 | | | | | | | | | | |
| PCload | | | | | | | 1 | | | | | | | | | |
| OE | | | | | | | | 1 | | | | | | | | |

As shown in the figure above, the first step is to get the input data, and then start the PC, so that the instructions in the

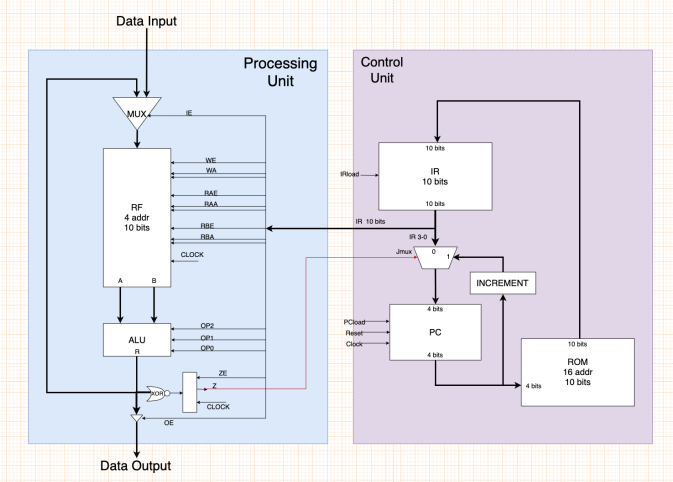
ROM can be input into the IR, and then the RF will get the instructions from the IR. The instruction will tell the ALU to perform the operation, and finally the ALU will output the result. In the specific implementation process, a 4-bit CPU cycle needs to be designed first, with a total of 16 phases. 1 phase has 1 clock time, and it will control the signal switching of a certain module. When the 16 phases are over, one cycle of the CPU ends, which represents the completion of one line of instruction in the IR. For example, as shown in the table in Figure 7, The first signal to be turned on is actually the IRload executed in phase 15. IRload controls the instruction to RF, which means that if IRload is turned off, the CPU will not execute any instructions. The intention of the design is to make the first CPU cycle do nothing, and execute IR after the first cycle is stable, which can improve the fault tolerance rate.

Fig. 8. Incorrect IE signal due to IR delay


IE means input enable and is executed in phase 1 instead of phase 0. The reason why IE skips in phase 0 is that there may be a delay between the instruction load between IR and RF. The earliest design was in phase 0, but Simulation found that IE never had the correct signal.

After phase 1, the RF address will be read (REA/RAA...) in phase 2 first, and perform ALU operation according to the instruction (OP 0/1/2) in phase 3. Then rewrite (WE/WA) the ALU result into RF in phase 4. These 3 phases are a logical operation process, which is expressed as R1+=R0 in OPCODE. The logic operation gives 3 phases of time. Phase 5 is ZE which means enable jump instruction, this signal is Jmux in Memory unit. Phase 6 is PCLoad, which means to execute the next instruction. Finally, phase 7 represents the output result. Only 8 of the 16 phases are used, and the remaining half are in the idle state, but 16 is a magic number. The complete circuit diagram is shown below in fig 9.

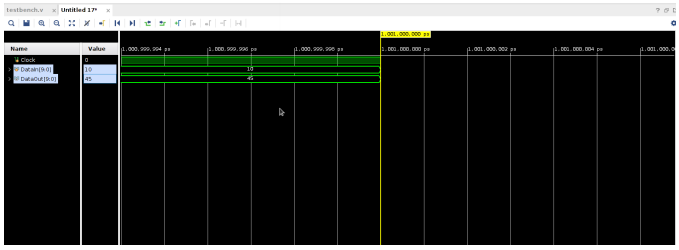
Fig. 9. CPU circuit




## V. SIMULATION

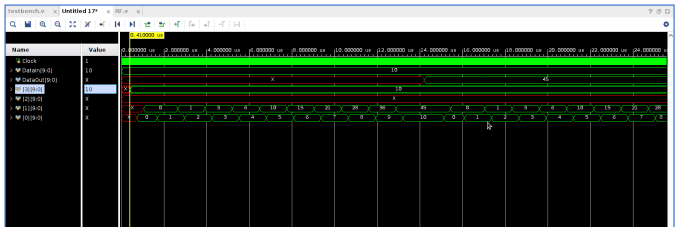
The testbench of the CPU is very simple. The clock is set to switch the signal every 10ns and the input signal is set to 10. Finally, connect the clock, input signal, and output signal to the CPU.

Fig. 10. Input and output waveform



Set the CPU execution time to 1ms, check the input and output signals, The waveform is displayed as expected in fig 10. If the input is 10, then according to the program, the output result should be 0+1+2+3+..+9=45.

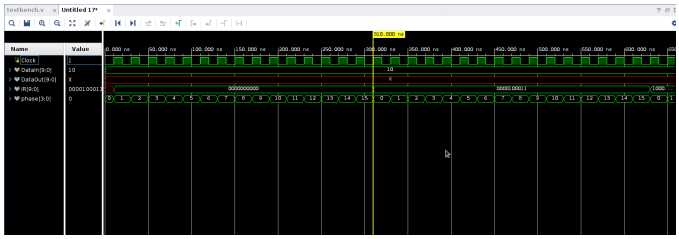
Fig. 11. Input and output waveform



Check the variables R0, R1, and R3 in the program. These three variables are in the register RF. The waveform is displayed as expected in fig 11. As the assigned variable of

input, R3 is always kept at 10. R2 is not used in the program, so the signal is X. R1 outputs the result variable, which is increasing every loop, 0,1,3,6,10,15...45. R0 is used as the IF condition counter of the loop, so in each loop it is incremented as 0,1,2,3,4...10.

Fig. 11. Input and output waveform



Check the status of the phase and instructions in the CPU. As shown in fig 12, in the first cycle, the IR instruction is 00_0000_0000. As mentioned in the previous chapter, PCload is executed from the last phase 15 instead of phase 0, so that the CPU does not perform any actions during the first cycle of startup. Instead, the first instruction is prepared in phase 15 at the end, so the CPU fetches the first instruction of IR from the second cycle. According to the first line of OPCODE, the first IR instruction is 00_0010_0011, and the second OPCODE instruction after the 15th phase.

## VI. Summary

A real CPU is definitely a lot more complicated, but this project shows a process of a program from a high-level language, to OPCODE, to binary code, to a control unit, to a processing unit, and combined into a CPU. Although the program is very simple, it involves many commonly used OPCODE instructions, including IF conditions, loops, variable assignment, logic operations, input and output. In simulation, check the status of each signal, including phase, register file, instruction register, input, output, and observe how these signals change in each instruction. In the future, the designed circuit diagram can also be burned in the chip for hardware testing. Instruction table can also be extended with more functions.

## References


[1] J. D. Carpinelli, "**The Very Simple CPU Simulator,**" 32nd Annual Frontiers in Education, 2002, pp. T2F-T2F, doi: 10.1109/FIE.2002.1157946.

[2] M. H. H. Ichsan and W. Kurniawan, "**Design and implementation 8 bit CPU architecture on Logisim for undergraduate learning support**," 2017 International Conference on Sustainable Information Engineering and Technology (SIET), 2017, pp. 132-137, doi: 10.1109/SIET.2017.8304123.

[3] Y. Chen and P. Cao, "**Toy CPU: An innovative curriculum design**," 2012 7th International Conference on Computer Science & Education (ICCSE), 2012, pp. 1690-1693, doi: 10.1109/ICCSE.2012.6295390.

[4] https://blog.csdn.net/linzhiheng123/article/details/78960163

[5] https://www.bilibili.com/video/BV1pK4y1C7es/

[6] https://www.youtube.com/watch?v=eVRdfl4zxfI&t=164s

[7] https://www.bilibili.com/video/BV1S7411f7sZ/

[8] https://danielmangum.com/posts/vivado-2020-x-ubuntu-20-04/

[9] https://zhuanlan.zhihu.com/p/109574885