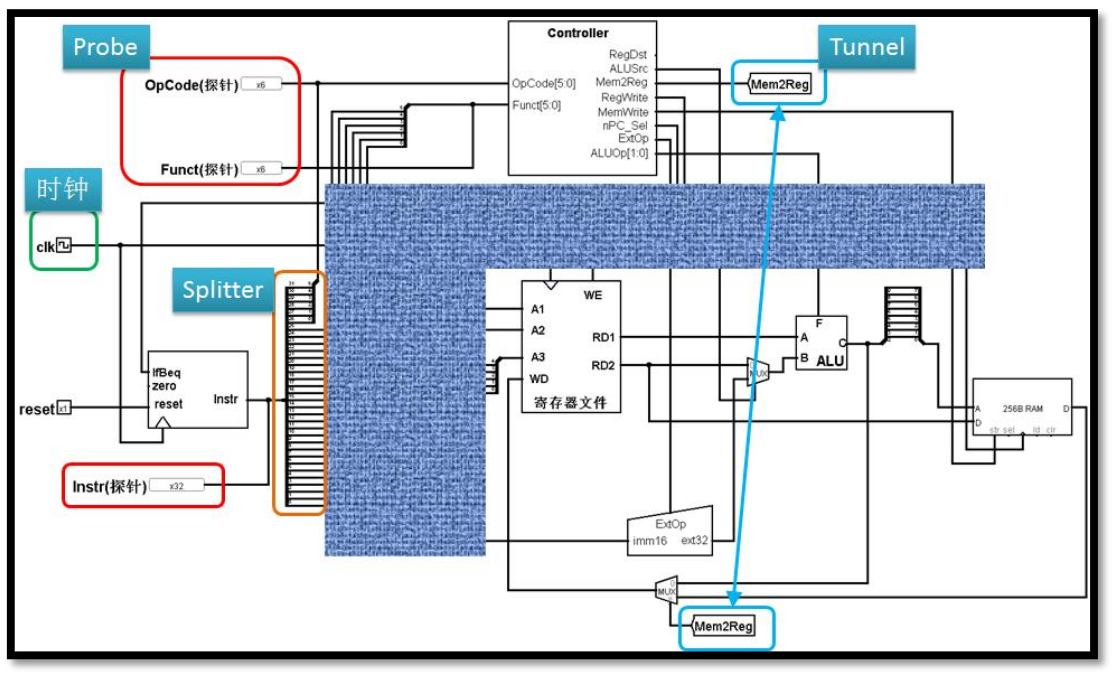
**Single-Cycle CPU Design Instructions**

# **Overview**

        Through the study of P0, P1, P2, you must have mastered the use of the tool set of this course. It is time to take a small test. In this section, you will use Logisim to develop a simple MIPS single-cycle processor and use Mars to write your own test program to verify the correctness of the CPU design. The experiments in this section will take you into the world of CPUs, a glimpse into the internal structure of the CPU, and lay the foundation for future experimentation. In this process, we hope that students can read the relevant chapters of the theoretical textbooks for their own reference. In the process, we also hope that students can write their own CPU design documents from scratch, which is our guide for circuit construction. It is worth mentioning that we will check the contents of the document (design documents, thinking questions) under the class and in the class test, please be prepared!

# **The** **Basic** **Idea**

        In this section, we design the CPU to include Controller (controller), IFU (instruction unit), GRF (general register group, also known as register file, register file), ALU (arithmetic logic unit), DM (data memory) Basic components such as EXT (bit expander) are connected to a data path through a combination of built-in devices such as MUX and Splitter. A possible top-level design reference illustration is:



# **Design** **and** **Test** **Instructions**

* The processor is a 32-bit processor.
* The instruction set that the processor should support is: {addu, subu, ori, lw, sw, beq, lui, nop}.
* The nop machine code is 0x00000000, which is a null instruction, and does not perform any valid action (modify the register, etc.)
* Addu, subu may not support overflow.
* The processor is designed for a single cycle.
* Modular and hierarchical design is required. The top-level valid driver signal requirements include and only include: reset (clk use the built-in clock module).
* You need to construct a test set yourself to verify the correctness of the design. (Automatic testing through class does not mean that your design is completely free of problems)
* After each section, the explanatory content is accompanied by document writing suggestions and related thinking questions. Please complete the relevant thinking questions and attach them to the CPU design document!
* Finally, you need to submit the contents: CPU design document (including thinking questions), Logisim circuit source files.

Friendly Tip: Please read through all the contents of this Lab before designing to avoid unnecessary modifications!

# **Module** **Specification**

      Module specifications are an important part of the design documentation and include port descriptions and functional definitions for each module of the CPU. A good module specification allows others to quickly understand and implement the functionality of the module. This is equivalent to a "manual" of the important parts of the CPU.

     In the module specification section, we do not directly give detailed port descriptions and functional specifications, only give a brief request, I hope students in the process of using Logisim to build the CPU, complete the corresponding design, build a more complete port and function description and upload. In the class test, we will check this section, please ensure that you will be able to let others understand your design!

     Below we give a few requirements for each module:

1. IFU (instruction unit): Internally includes PC (program counter), IM (instruction memory) and related logic.

* The PC is implemented with registers and should have a reset function.
* Start address: 0x00000000.
* IM is implemented in ROM and has a capacity of 32bit \* 32.
* Since the actual address width of the IM is only 5 bits, it is necessary to use an appropriate method to associate the address stored in the PC with the IM.

1. GRF (General Register Group, also known as register file, register file)

* Implemented with a write-enabled register, the total number of registers is 32.
* The value of register 0 always remains at 0. The other registers have an initial value of 0 and do not require special settings.

1. ALU (Arithmetic Logic Unit)

* Provides 32-bit addition, subtraction, or operation and size comparison.
* Overflow may not be supported (no overflow detected).

1. DM (Data Storage/Data Memory)

* Implemented in RAM with a capacity of 32bit \* 32.
* Start address: 0x00000000.
* The RAM should use dual port mode, which sets the Data Interface property of RAM to Separate load and store ports.

1. EXT:

* You can use the built-in Bit Extender in logisim.

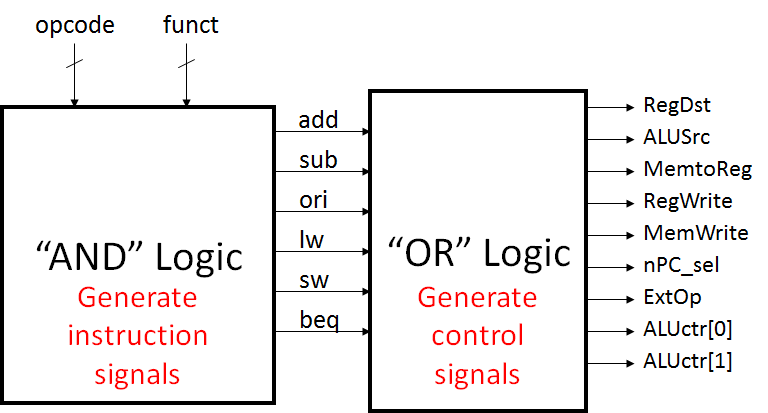
1. Controller

* Constructing control signals using an OR gate array,
* The specific method is described later.

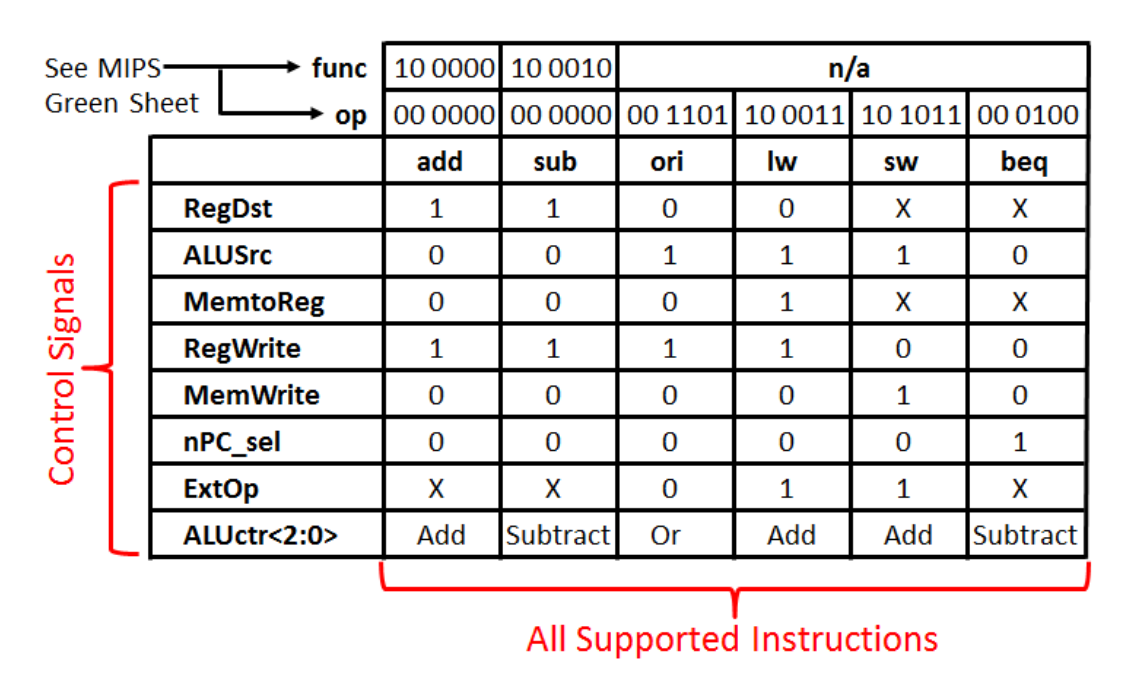
# **Controller** **Design**

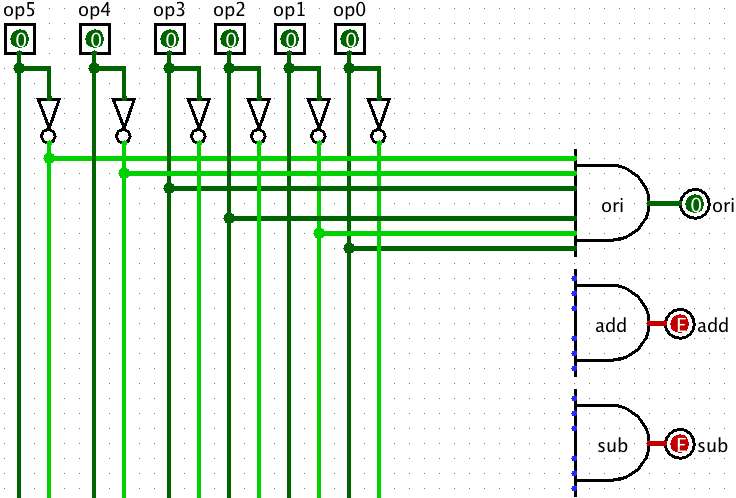
       The design of the controller, at the most basic level, is a decoding process that converts the information contained in each machine instruction into control signals for various parts of the CPU (RegDst, ALUSrc, etc.), I believe the students also learned about this part in the theory class. At the practical level, how to make the decoding process engineering is an important issue. The most violent way is to force the generation of the truth table, which is not scalable and easy to debug in the actual operation. It is very easy to make mistakes when the number of instructions is too large. The clever predecessors have created the following methods for us.

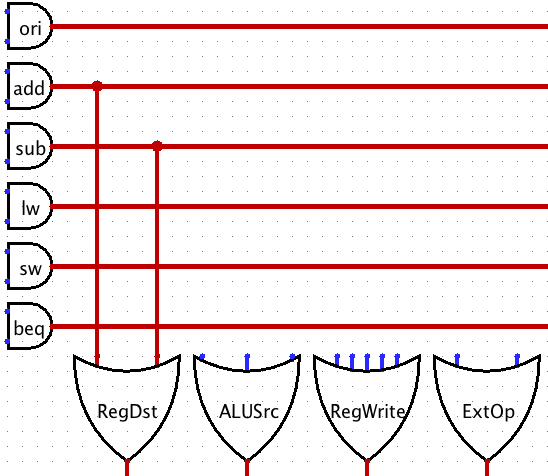
       We decompose the decoding logic into two parts: “AND” logical and “OR” logic: the function of the “AND” logic is to identify the input machine code as the corresponding instruction; the function of the “OR” logic is to generate different control according to the input instruction. signal. This kind of split makes the two parts of the logic clear, which is a kind of simple abstraction and modularity. I hope the students can learn from other fields as well. (The master of this kind of thinking is UNIX, interested students can search for "UNIX philosophy" to deepen understanding)



       In the process of designing these two sets of logic, the logic is very natural. “OR” logic requires us to establish a mapping from instruction to control signals. In order to avoid errors, we hope to use the truth table to complete the corresponding design tasks, and hope that the corresponding logic can be simplified by the truth table. A typical truth table is shown below:



The following figure is a concrete example of the above logic in Logisim - the OR gate array (not complete), of course you can also have your own circuit design.



# **Test** **CPU**

        After the CPU is built, testing the CPU is a necessary part. It can check out the errors in our CPU design and construction. For the design of the test program, we have the following recommendations:

* All instructions should be tested adequately.
* Write the test program in MARS and debug it.
* Note that "Settings->Memory Configuration" in MARS can only configure the instruction memory start address to be 0 address, and the start address of the instruction memory and data memory cannot be configured as 0 address!

Since the DM start address in the Logisim design is 0, please carefully observe the instructions used. After exporting the binary code passed in MARS, you may need to manually modify the data offset in the script. (Hint: In fact, in modern mainstream computers, the starting addresses of the data memory and the instruction memory should not overlap. However, in this design, this can be temporarily ignored due to the separate memory design.)