CPU单周期设计文档

目录

[一、 模块设计 2](#_Toc10692)

[1.IFU 2](#_Toc12802)

[2.GRF 3](#_Toc10323)

[3.ALU 3](#_Toc19823)

[4.DM 4](#_Toc26212)

[5.EXT 5](#_Toc24925)

[6.顶层模块（main） 5](#_Toc17111)

[二、 控制器设计 6](#_Toc31589)

[三、测试代码 8](#_Toc32317)

[CPU-text-1: 8](#_Toc26080)

[CPU-text-2: 9](#_Toc15716)

[四、思考题 9](#_Toc11964)

2. 模块设计

**1.IFU**

IFU内部包括 PC、IM(指令存储器)、Adder、选择器和beq跳转值作为输入。PC 用寄存器实现，具有复位功能且有时钟信号。起始地址：0x00000000。IM用ROM 实现。IM的实际地址宽度5位与PC中2-6位相接，ROM输出Instr为指令编码，选择器的选择信号为beq跳转信号，即当指令为beq且置零端为真时选择信号为1.(后续添加J,Jal,Jr指令)

表 1-IFU端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 定义 | | 描述 |
| 1 | input | CLK | 时钟信号，下一个时钟上升沿来到PC寄存器存入当前PC |
| 2 | RESET | 复位信号，当复位信号有效时，PC被设置为0x00000000 |
| 3 | beq | 32位，指令中立即数imm移位且符号扩展后的值 |
| 4 | Zero | 多路选择器选择信号，当为beq指令且ALU置零端为真时为1 |
| 5 | output | Instr | 输出对应PC的指令码 |
| 新增 | 定义 | | 描述 |
| 1 | input | j | 直接跳转到PC[31...28] || instr\_index(指令码后26位) || 00 |
| 2 | jal | 跳转并保存$ra($31=PC+4) |
| 3 | jr | 跳转至$ra($31)保存的地址 |

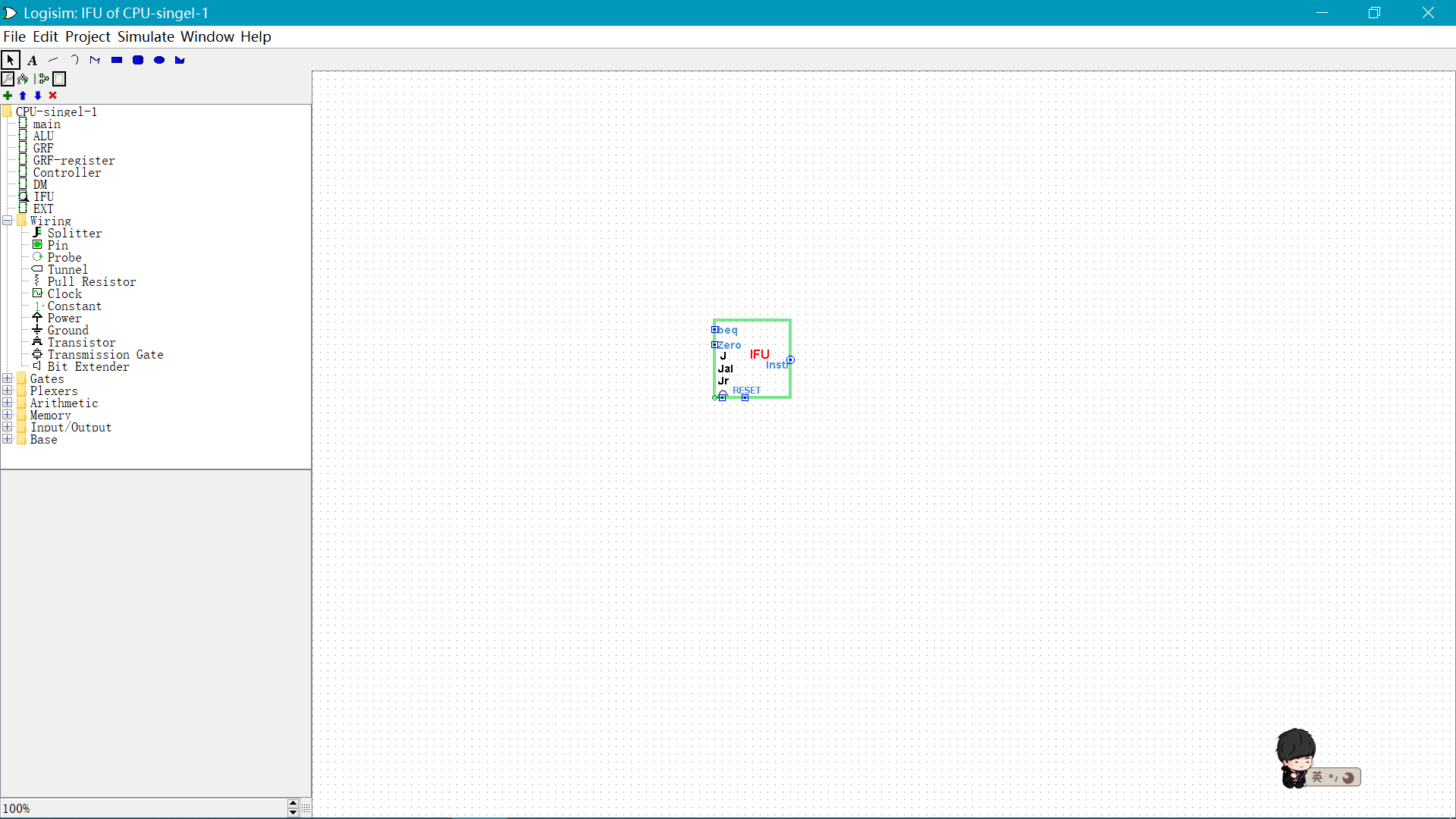


图 1-IFU模块图

**2.GRF**

用具有写使能和复位信号的寄存器实现，寄存器总数为 32 个，第一个寄存器值始终为0x0000，其他寄存器默认为0x0000。输入端有三个地址，一个数据输入，两个数据输出。

表 2-GRF端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 定义 | | 描述 |
| 1 | input | CLK | 时钟信号，下一个时钟上升沿来临且写入信号有效写入数据 |
| 2 | RESET | 复位信号，当复位信号有效时，寄存器被清零 |
| 3 | WE | 写入寄存器使能信号 |
| 4 | A1 | Rs寄存器地址5bit |
| 5 | A2 | Rt寄存器地址5bit |
| 6 | WA | Rt或Rd寄存器地址5bit |
| 7 | output | WD | 需要写入寄存器的值 |
| 8 | RD1 | 输出对应Rs寄存器里的值 |
| 9 | RD2 | 输出对应Rt寄存器里的值 |

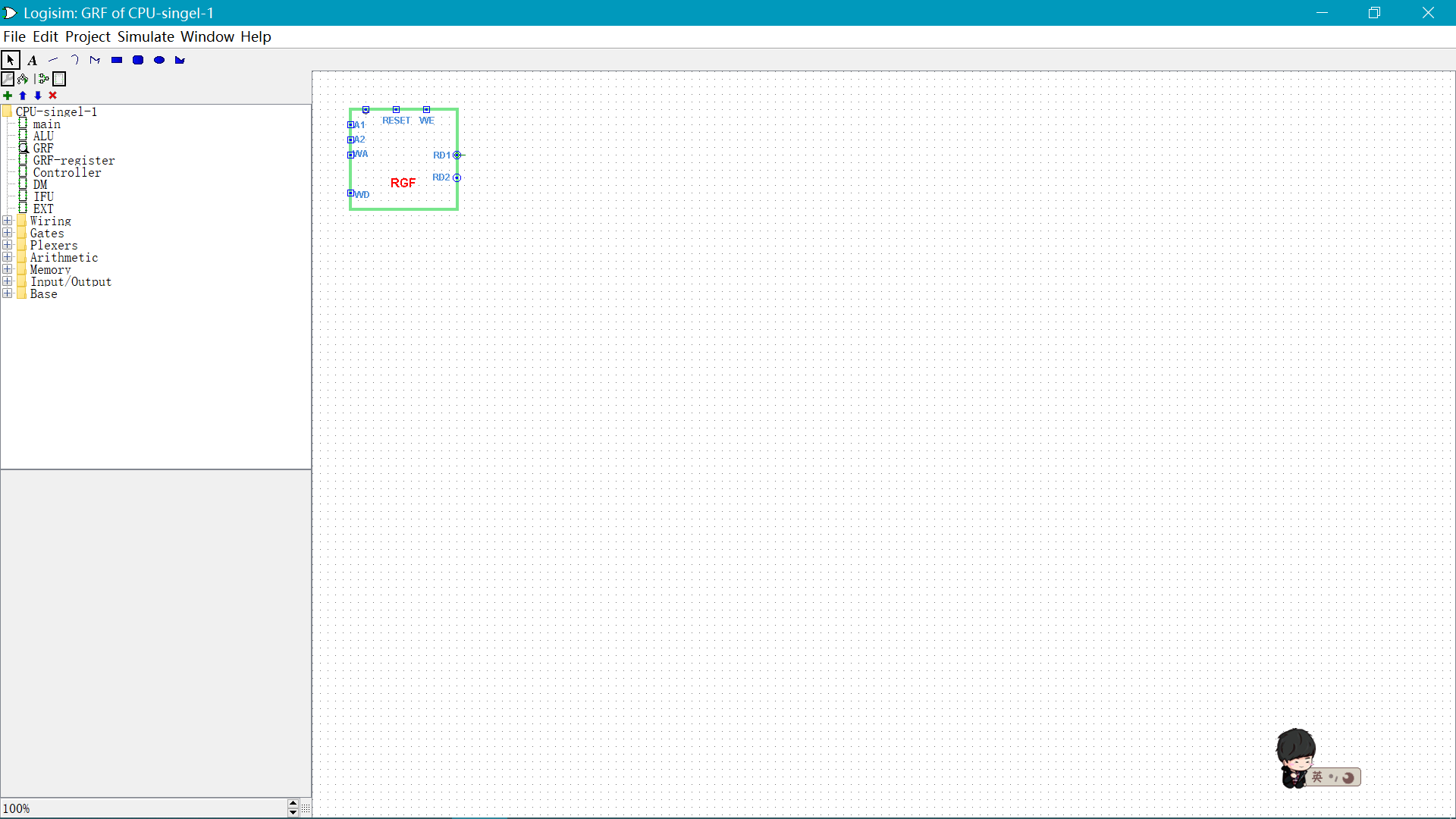


图 2-RGF模块图

**3.ALU**

算术逻辑单元，32位无符号加、无符号减、立即数或运算及大小比较功能。在这里大小比较采用无符号数相减之后的结果与0x0比较，相等则输出1。

表 3-ALU端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 定义 | | 描述 |
| 1 | input | OP | ALU运算控制信号，0001无符号加、0010无符号减、0100立即数或、1000立即数加载至高位 |
| 2 | A | 第一个数，32位 |
| 3 | B | 第二个数，32位 |
| 4 | output | Result | 运算结果，32位 |
| 5 | Zero | A-B=0时Zero=1，置零端 |

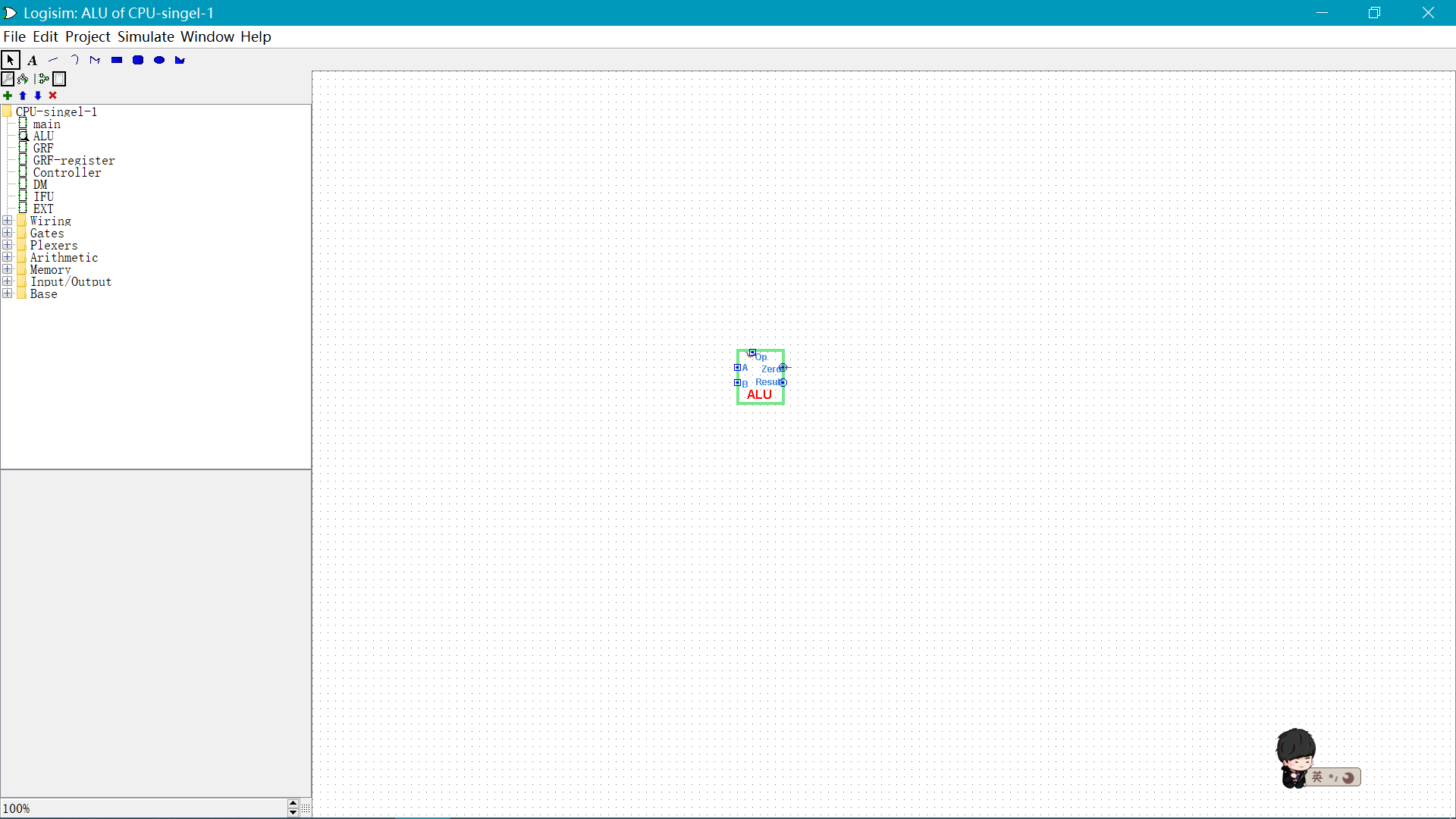


图 3-ALU模块图

**4.DM**

数据存储器，使用RAM实现，容量为32bit \* 32。起始地址：0x00000000。输入端口有5位地址，写入数据32位等，输出对应的32位数据。

表 4-DM端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 定义 | | 描述 |
| 1 | input | clk | 时钟信号，当时钟上升沿来到且写入信号有效则写入数据 |
| 2 | clr | 复位信号 |
| 3 | A | 5位存储器地址 |
| 4 | WD | 写入的数据，32位 |
| 5 | SD | SW指令存数信号 |
| 6 | LD | LW指令取数信号 |
| 7 | sel | 选择信号，暂时不用 |
| 8 | output | RD | 读取数据，32位 |

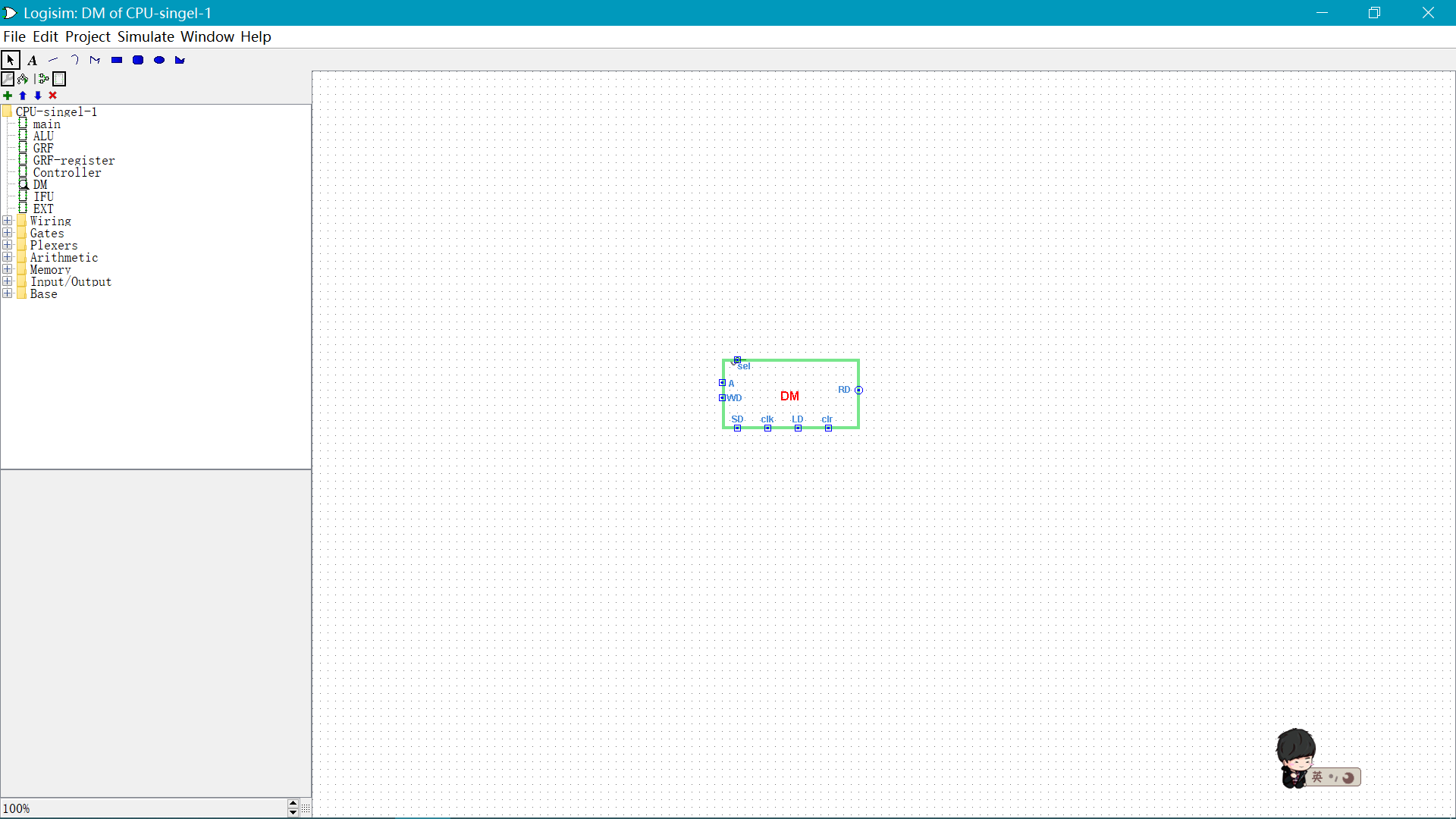


图 4-DM模块图

**5.EXT**

扩展器，该扩展器包括了跳转指令的立即数扩展。

表 5-EXT端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 定义 | | 描述 |
| 1 | input | EXTOp | 扩展类型信号，为0代表符号扩展，为1代表beq跳转(sign(imm||00))左移两位后符号扩展 |
| 2 | imm16 | 16位立即数或offset |
| 3 | output | ext32 | 扩展后的32位数 |

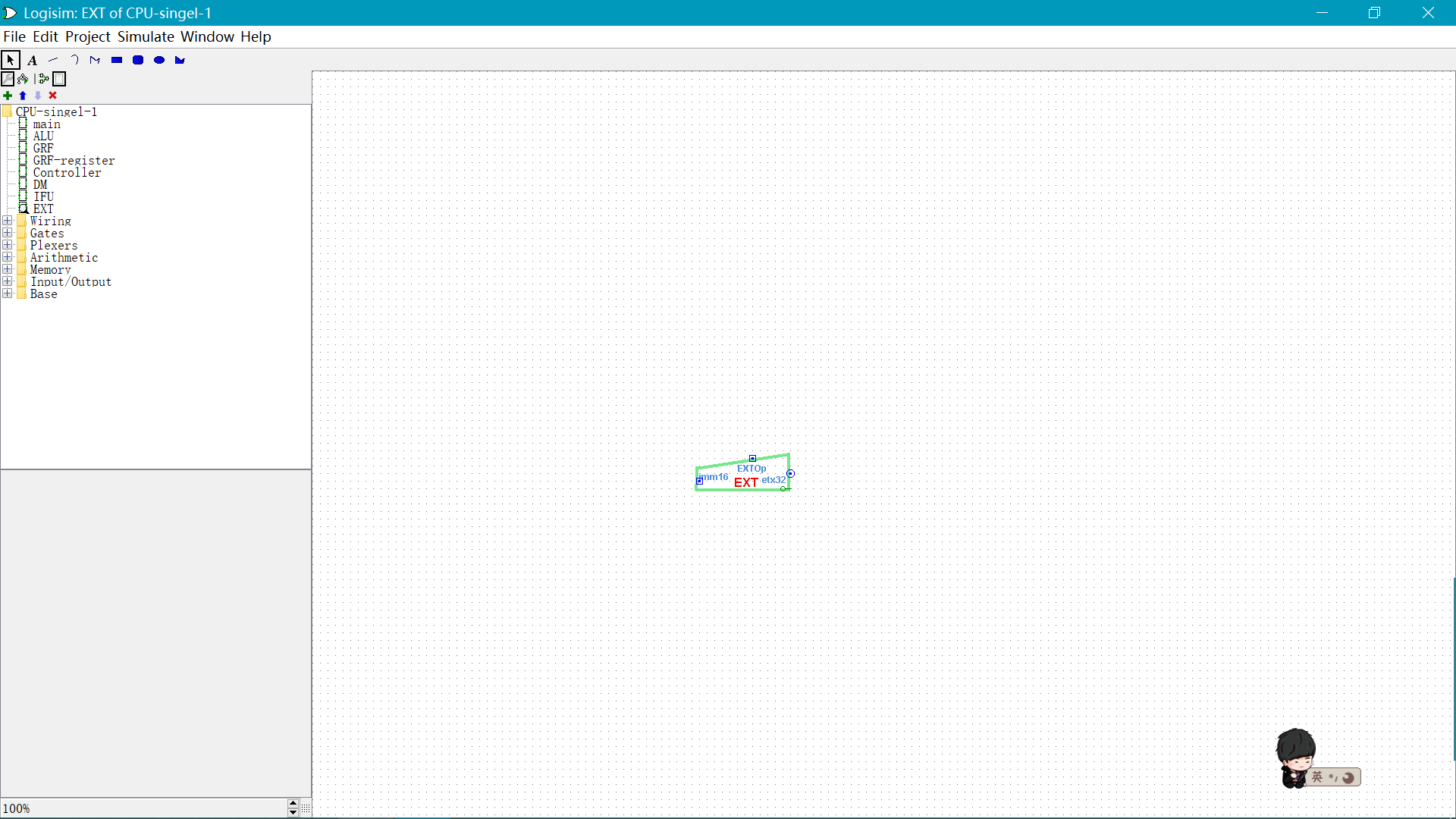


图 5-EXT模块图

**6.顶层模块（main）**

这里是几个模块连接而成的单周期CPU顶层块，右上角为输入输出信息。

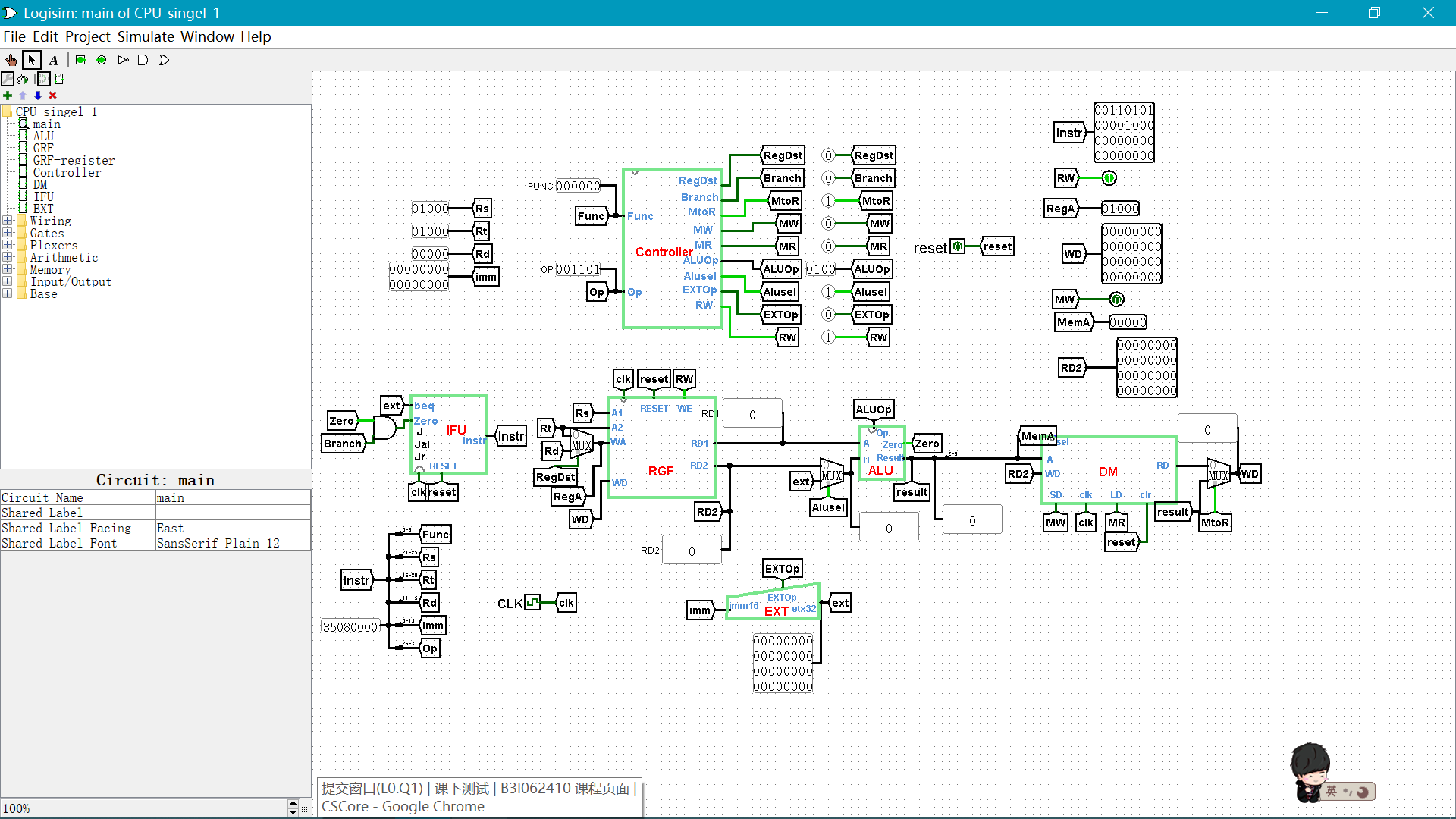


图 6-顶层模块图

1. 控制器设计

此控制器ALUOp采用独热编码，0001无符号加、0010无符号减、0100立即数或、1000立即数加载至高位；在DM（数据存储器）模块增加了MW,MR作为写入使能信号、读取信号；EXTOp信号用来选择立即数扩展的类型，扩展器选择信号，0选择符号扩展，1选择beq跳转。

表 6-控制器端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 定义 | | 描述 |
| 1 | input | Func | 指令码后6位 |
| 2 | Op | 指令码前6位操作码 |
| 3 | output | RegDst | Rt与Rd选择信号，0选择Rt，1选择Rd |
| 4 | Branch | 若是分支指令（当前指beq）为1 |
| 5 | MtoR | 写入寄存器数据的选择信号，0选择存储器读取的数据，1选择ALU运算的result |
| 6 | MW | 存储器写入信号 |
| 7 | MR | 存储器读取信号 |
| 8 | ALUOp | ALU控制信号，0001无符号加、0010无符号减、0100立即数或、1000立即数加载至高位 |
| 9 | Alusel | ALU数据B端口的选择信号，0选择RD2，1选择扩展数 |
| 10 | EXTOp | 扩展器选择信号，0选择符号扩展，1选择beq跳转 |
| 11 | RW | 寄存器文件写入信号 |
| 新增 | 定义 | | 描述 |
| 1 | output | J | 若是跳转指令J则为1 |
| 2 | Jal | 若是跳转指令Jal则为1 |
| 3 | Jr | 若是跳转指令Jr则为1 |

表 7-控制器功能真值表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Func | 100001 | 100011 | x | x | x | x | x |
| Op | 000000 | | 001101 | 100011 | 101011 | 000100 | 001111 |
|  | Addu | Subu | Ori | Lw | Sw | Beq | Lui |
| RegDst | 1 | 1 | 0 | 0 | x | x | 0 |
| Branch | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| MtoR | 1 | 1 | 1 | 0 | x | x | 1 |
| MW | x | x | x | 0 | 1 | x | x |
| MR | x | x | x | 1 | 0 | x | x |
| ALUOp | 0001 | 0010 | 0100 | 0001 | 0001 | 0010 | 1000 |
| Alusel | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| EXTOp | x | x | 0 | 0 | 0 | 1 | 0 |
| RW | 1 | 1 | 1 | 1 | x | x | 1 |
| Func | x | x | 001000 |  |  |  |  |
| Op | 000010 | 000011 | x |  |  |  |  |
| 新增 | J | Jal | Jr |  |  |  |  |
| RegDst | x | x | x |  |  |  |  |
| Branch | x | x | x |  |  |  |  |
| MtoR | x | x | x |  |  |  |  |
| MW | x | x | x |  |  |  |  |
| MR | x | x | x |  |  |  |  |
| ALUOp | x | x | x |  |  |  |  |
| Alusel | x | x | x |  |  |  |  |
| EXTOp | x | x | x |  |  |  |  |
| RW | x | 1 | x |  |  |  |  |
| J | 1 | 0 | 0 |  |  |  |  |
| Jal | 0 | 1 | 0 |  |  |  |  |
| Jr | 0 | 0 | 1 |  |  |  |  |

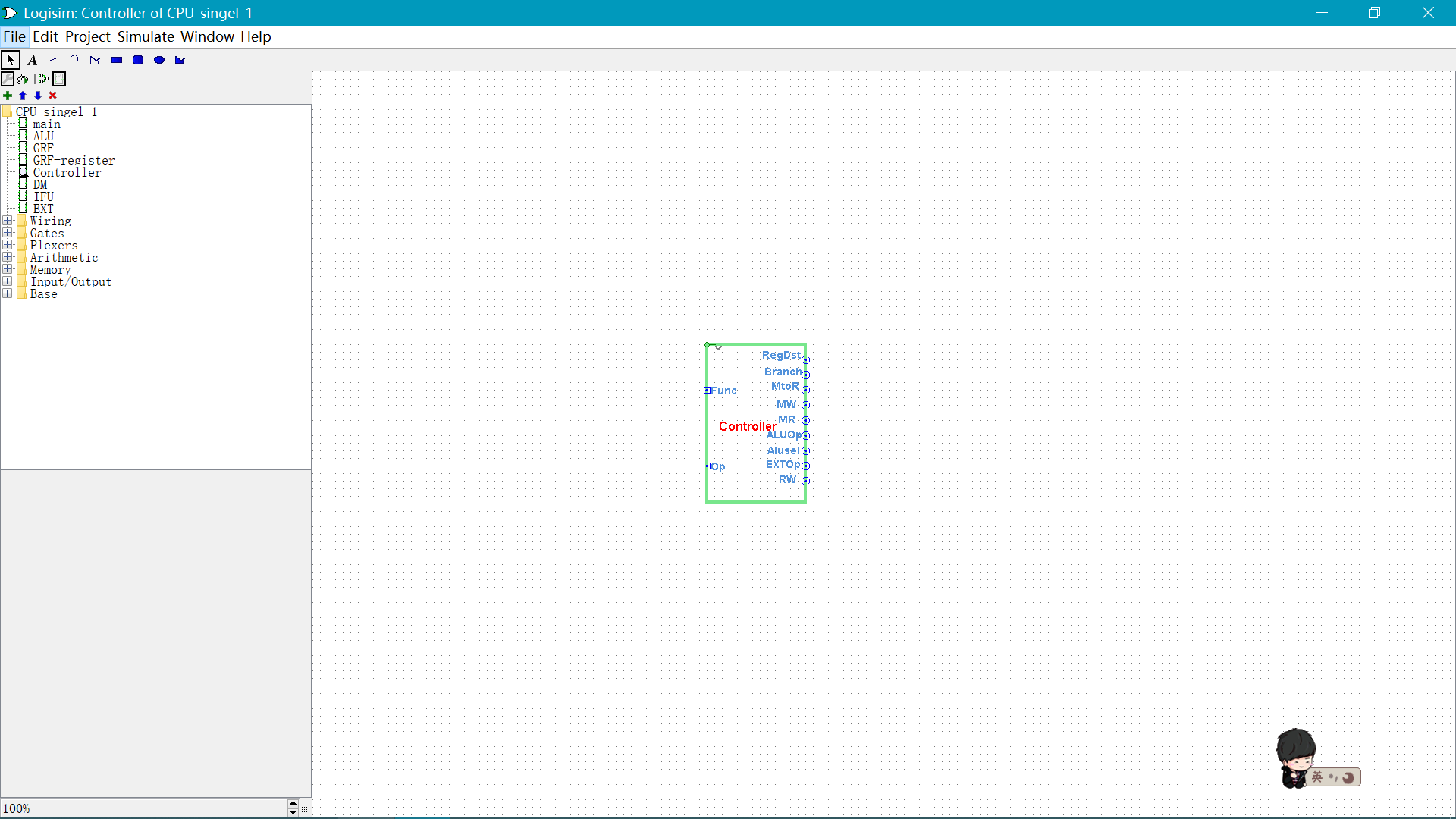


图 7-Controller模块图

三、测试代码

预期测试结果在右侧

CPU-text-1:

# CPU-text file-1

# ori

ori $a0, $0, 123 #$a0=123

ori $a1, $0, 456 #$a1=456

# lui

lui $a2, 234 #$a2=15335424

lui $a3, 0xffff #$a3=-65536

# addu

addu $s0, $a0, $a1 #$s0=579

addu $s1, $a0, $a2 #$s1=15335547

# subu

subu $s2, $a1, $a0 #$s2=333

subu $s3, $a0, $a1 #$s3=-333

# sw

ori $t0, $0, 0x0000 #$t0=0

sw $a0, 0($t0) #$t0所存地址加0后的位置存储$a0

sw $a1, 4($t0) #$t0所存地址加4后的位置存储$a1

# lw

lw $a0, 0($t0) #取出$t0所存地址加0后位置的数据到$a0

lw $a1, 4($t0) #取出$t0所存地址加4后位置的数据到$a1

sw $a0, 4($t0) #$t0所存地址加0后的位置存储$a0

sw $a1, 0($t0) #$t0所存地址加0后的位置存储$a0

CPU-text-2:

# CPU-text file-2

# beq

ori $t0, $0, 0x0000 #$t0=0

ori $a0, $0, 1 #$a0=1

ori $a1, $0, 2 #$a1=2

ori $a2, $0, 1 #$a2=1

beq $a0, $a1, for\_1 #if $a0=$a1, for\_1,不成立

beq $a0, $a2, for\_2 #if $a0=$a2, for\_2,成立

for\_1:

sw $a0, 0($t0) #不执行该语句

for\_2:

sw $a1, 4($t0) #$t0所存地址加4后的位置存储$a1

四、思考题

1.若PC（程序计数器）位数为30位，试分析其与32位PC的优劣。

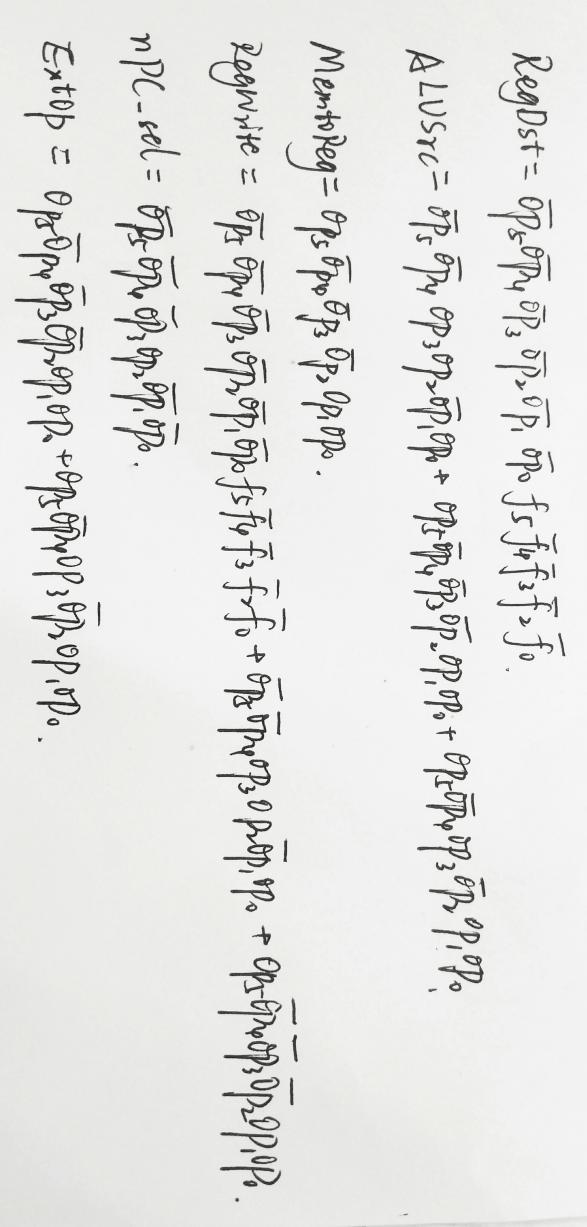
答：缺点是32位PC的话不需要扩展2位0后再加4，而30位PC需要扩展后加4；优点是所用的触发器元件少了。

2.现在我们的模块中 IM使用ROM， DM使用RAM， GRF使用寄存器，这种做法合理吗？ 请给出分析，若有改进意见也请一并给出。

答：在单周期CPU设计中是合理的，ROM是只读存储器，指令存储器只用来读取指令；RAM是读写存储器，数据存储器需要读写功能并且需要足够的存储空间，而不需要很快地访问速度；寄存器是用来保存临时变量的，GRF需要很快的访问速度，而寄存器存取数据速度最快。

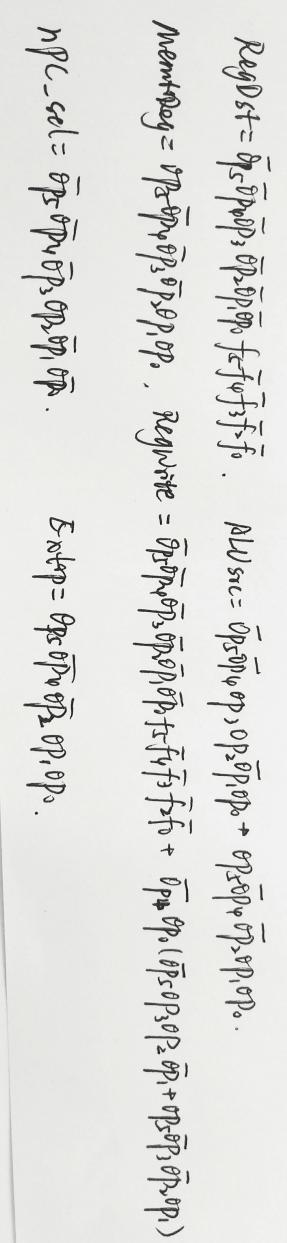
1. 结合上文给出的样例真值表，给出RegDst， ALUSrc， MemtoReg，RegWrite, nPC\_Sel, ExtOp与op和func有关的布尔表达式（表达式中只能使用“与、或、非”3 种基本逻辑运算。）

答：



1. 充分利用真值表中的 X 可以将以上控制信号化简为最简单的表达式， 请给出化简后的形式。

答：



5.事实上，实现nop空指令，我们并不需要将它加入控制信号真值表，为什么？请给出你的理由。

答：nop指令没有任何操作，对控制信号没有作用，对逻辑电路无影响，所以不用加。

6.前文提到，“可能需要手工修改指令码中的数据偏移”，但实际上只需再增加一个 DM片选信号,就可以解决这个问题。请阅读相关资料并设计一个 DM 改造方案使得无需手工修改数据偏移。

答：该单周期CPU的DM是容量为 32bit \* 32的RAM，数据起始地址：0x00000000，所以增加一个片选：选择输入数据的2-6位，即是DM中数据的地址。

1. 除了编写程序进行测试外，还有一种验证CPU设计正确性的办法——形式验证。 形式验证的含义是根据某个或某些形式规范或属性，使用数学的方法证明其正确性或非正确性。请搜索“形式验证（Formal Verification)"了解相关内容后，简要阐述相比与测试，形式验证的优劣。

答：所谓形式验证，是指从数学上完备地证明或验证电路的实现方案是否确实实现了电路设计所描述的功能。形式验证方法分为等价性验证、模型检验和定理证明等。形式验证更直接的验证所得结果是否一致，而不像测试需要每一步仿真获得结果；同时，形式验证更全面（达到100%）、更合理地验证结果所出现的情况，而测试只能针对一部分出现的情况进行验证；形式验证可以确定问题出现在哪一个具体的门级电路，电路功能的等价可利用逻辑一致性来验证，验证时间短，可以很快发现和改正电路设计中的错误，可以缩短设计周期。形式验证的缺点是，不能考虑到电路具体的实现情况，也就是无法判断各电路的延迟、冒险和功耗等，所以无法分析电路性能。