CPU单周期设计文档

目录

[一、 模块设计 1](#_Toc17975)

[1.IFU 1](#_Toc460)

[2.GRF 2](#_Toc28926)

[3.ALU 2](#_Toc5385)

[4.DM 3](#_Toc3506)

[5.EXT 4](#_Toc2126)

[6.顶层模块（mips） 4](#_Toc29154)

[二、 控制器设计 4](#_Toc21344)

[三、测试代码 6](#_Toc5356)

[CPU-text-1: 6](#_Toc7202)

[CPU-text-2: 7](#_Toc26910)

[CPU-text-3: 8](#_Toc18371)

[四、思考题 8](#_Toc21174)

1. 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：0x00003000。IM用ROM 实现。IM的实际地址宽度10位与PC中2-11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。添加了J,Jal,Jr指令，分别有jump(1bit),jr(1bit),ra(32bit),index(26)输出端口，还外输出PC+4的值。IFU里调用了一个ROM块，ROM采用readmemh函数读取指令码。

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0x00003000 |
| 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 | jump | 1bit，为1时跳转到PC[31...28] || instr\_index(指令码后26位) || 00 |
| 2 | ra | 32位$ra的值($31=PC+4) |
| 3 | jr | 1bit,为1时跳转至$ra($31)保存的地址 |
| 4 | index | instr\_index(指令码后26位) |
| 5 | output | PC\_ | PC+4的值，32bit |

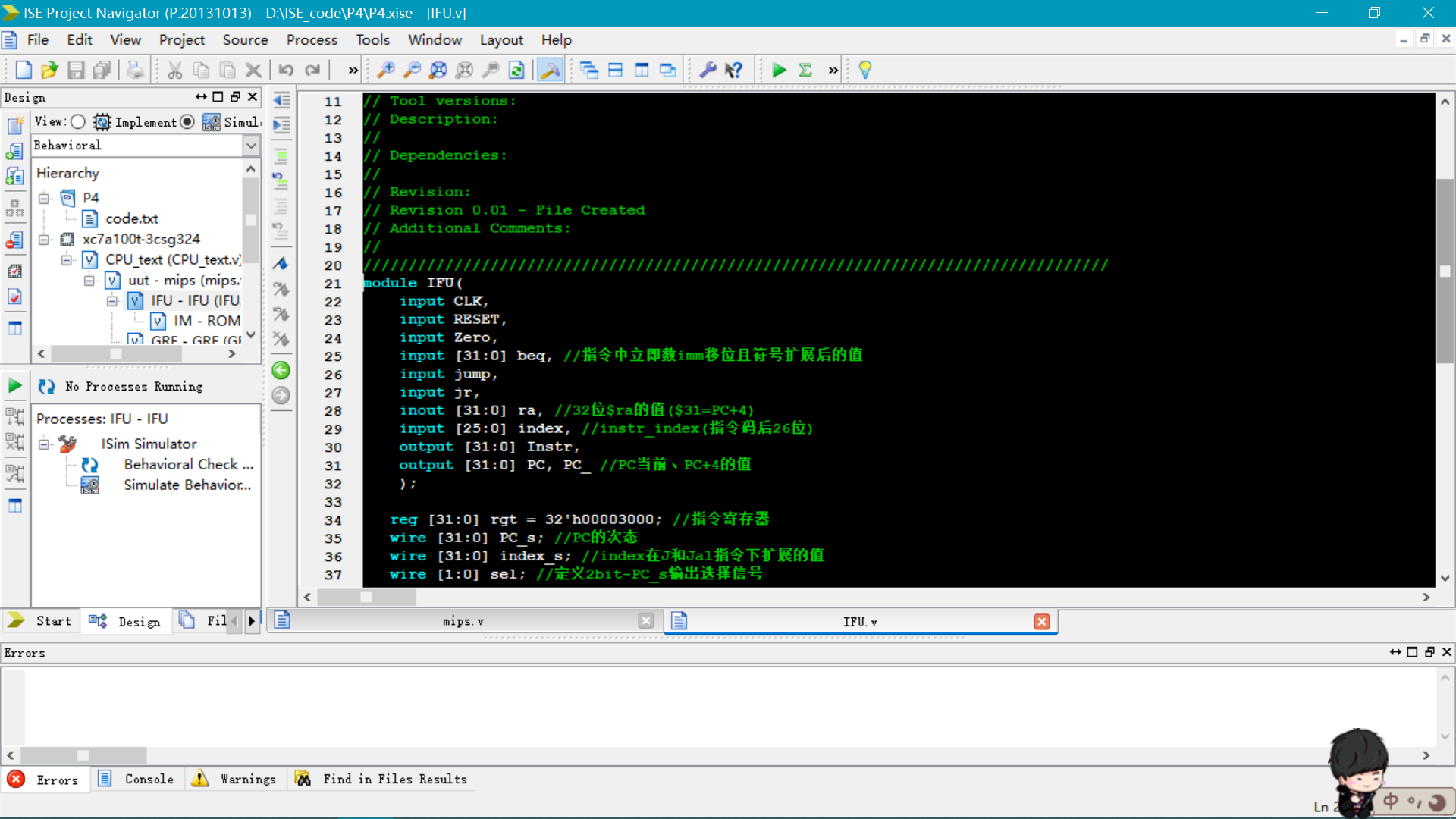


图 1-IFU模块图

**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0x00000000，其他寄存器默认为0x00000000。输入端有三个地址，一个数据输入，两个数据输出。增加一个输入端口PC，用于接收来自IFU的PC的值。

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 | WD | 需要写入寄存器的值 |
| 8 | PC | 接收来自IFU的当前PC值 |
| 9 | output | RD1 | 输出对应Rs寄存器里的值 |
| 10 | RD2 | 输出对应Rt寄存器里的值 |

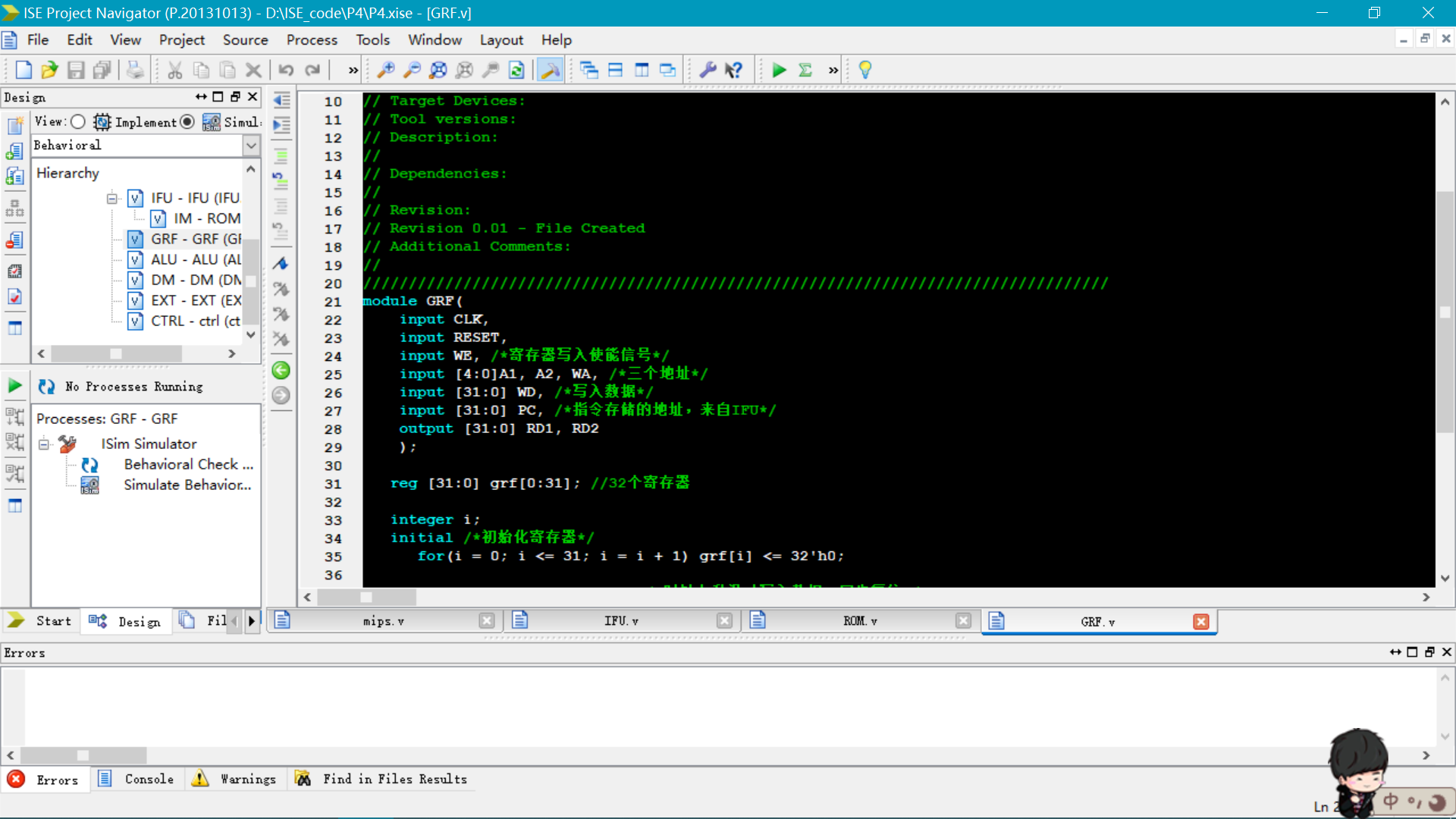


图 2-RGF模块图

**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32’h0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 |

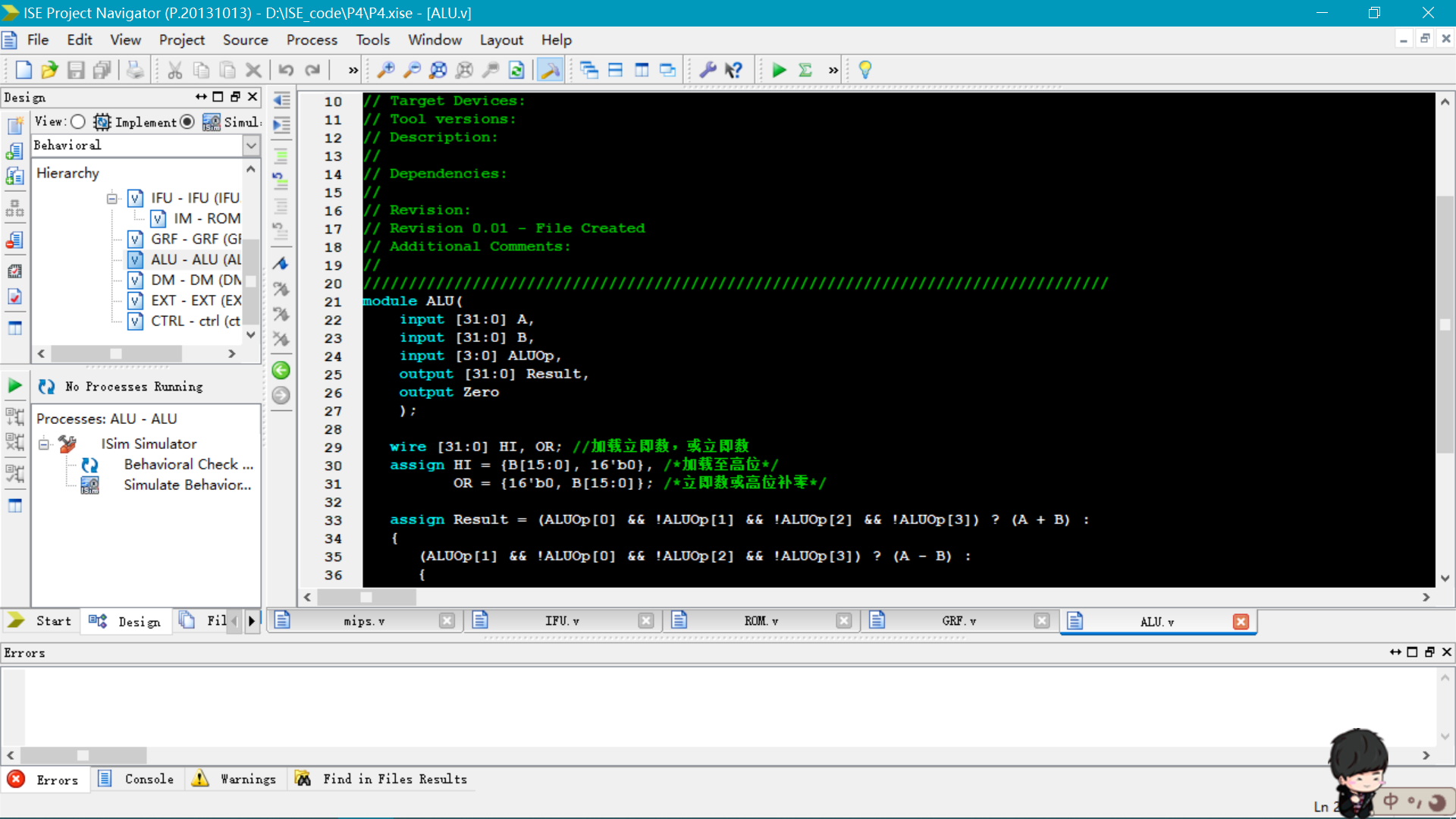


图 3-ALU模块图

**4.DM**

数据存储器，使用RAM实现，容量为32bit \* 1024。起始地址：0x00000000。输入端口有10位地址，写入数据32位等，输出对应的32位数据。增加一个输入端口PC，用于接收来自IFU的PC的值。

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 | 10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 | PC | 接收来自IFU的当前PC值 |
| 8 | output | RD | 读取数据，32位 |

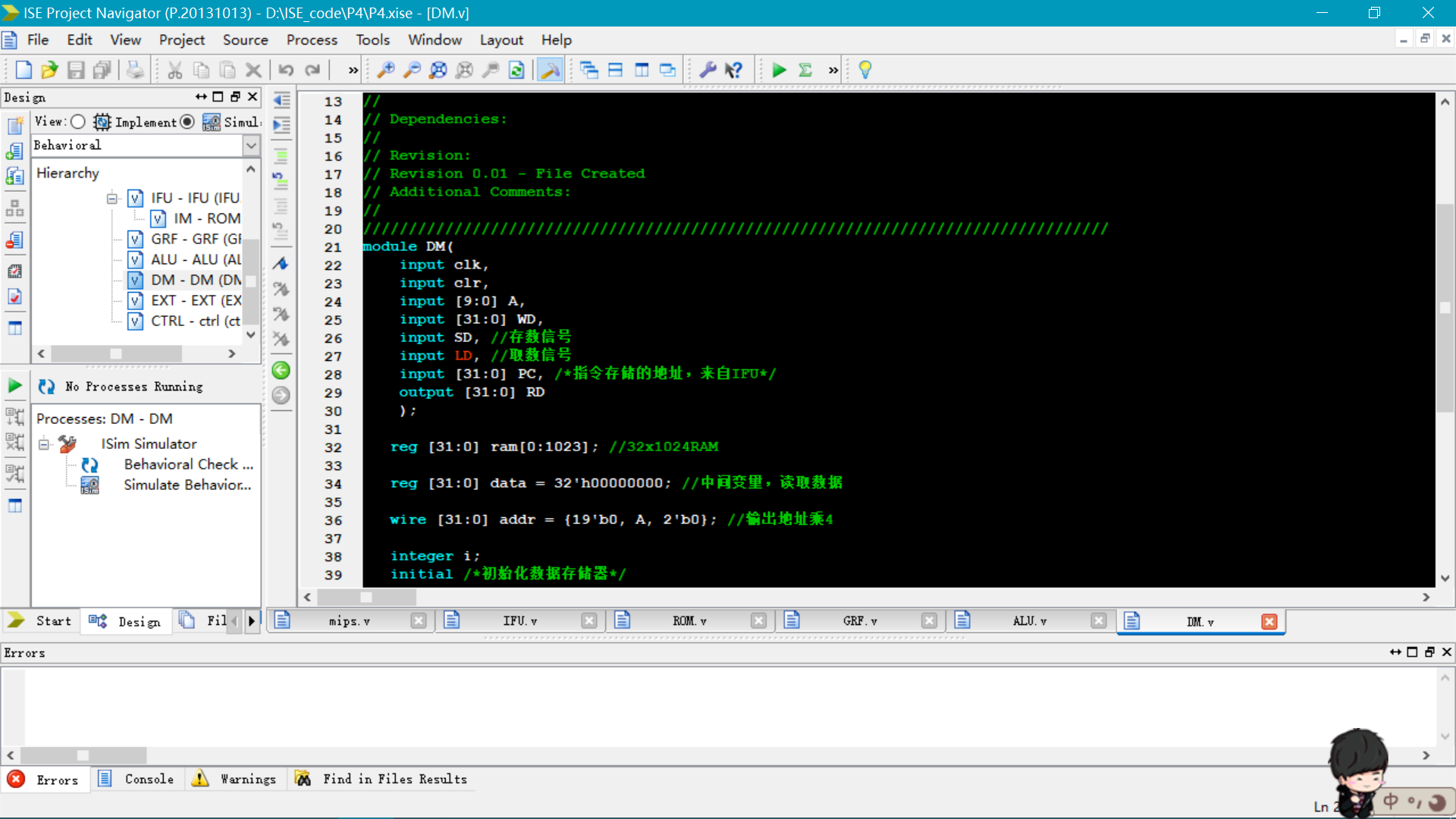


图 4-DM模块图

**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 |

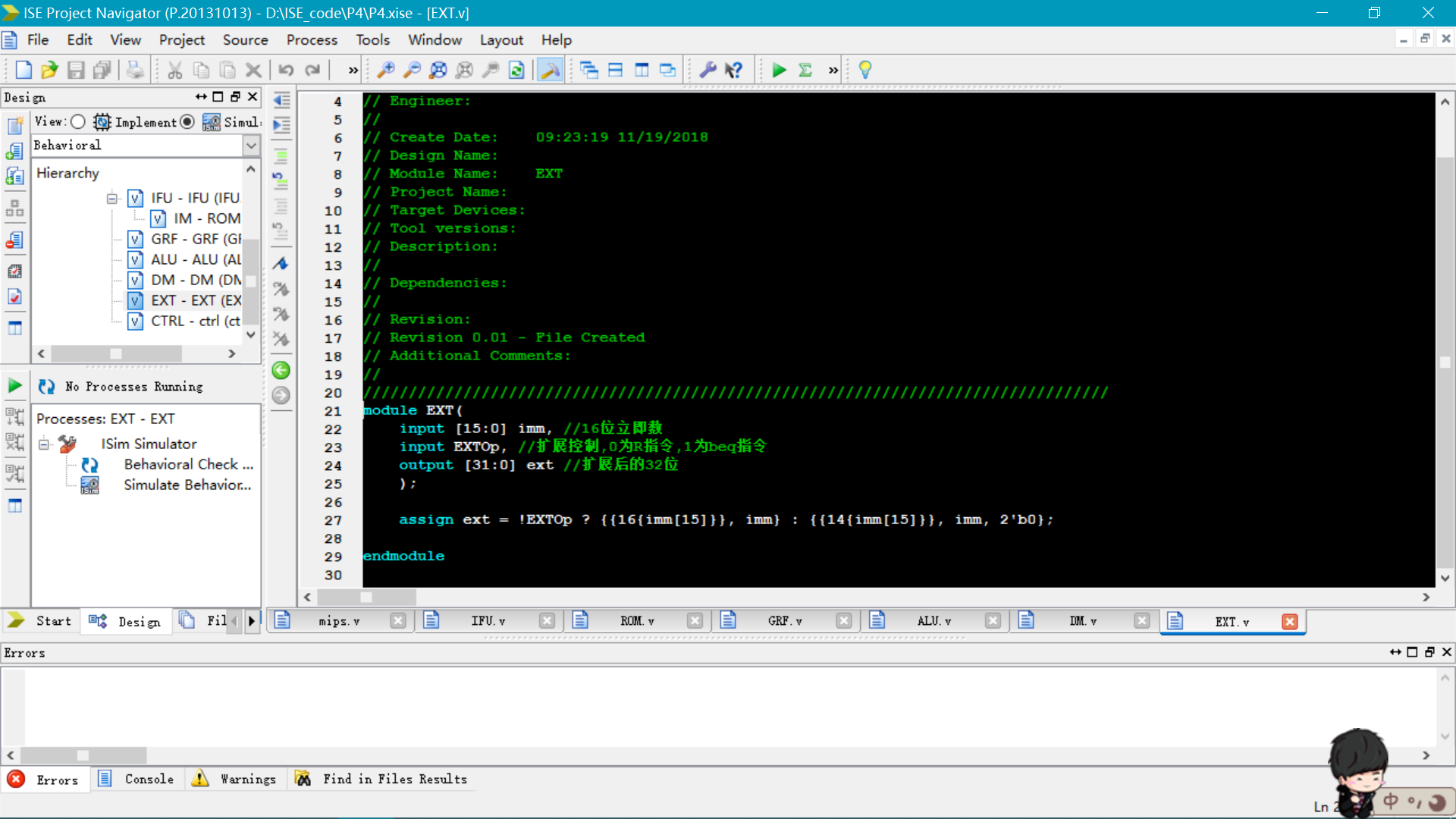


图 5-EXT模块图

**6.顶层模块（mips）**

这里是几个模块连接而成的单周期CPU顶层块，顶层模块也是用来调用各个模块，定义wire型变量将各模块的信号相连。利用assign对变量进行链接。

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。之后添加了J，Jal, Jr输出信号，代表相应的跳转信号。

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 |
| 12 | output | J | 若是跳转指令J则为1 |
| 13 | Jal | 若是跳转指令Jal则为1 |
| 14 | 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 | 000000 |  |  |  |  |
| 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 |  |  |  |  |

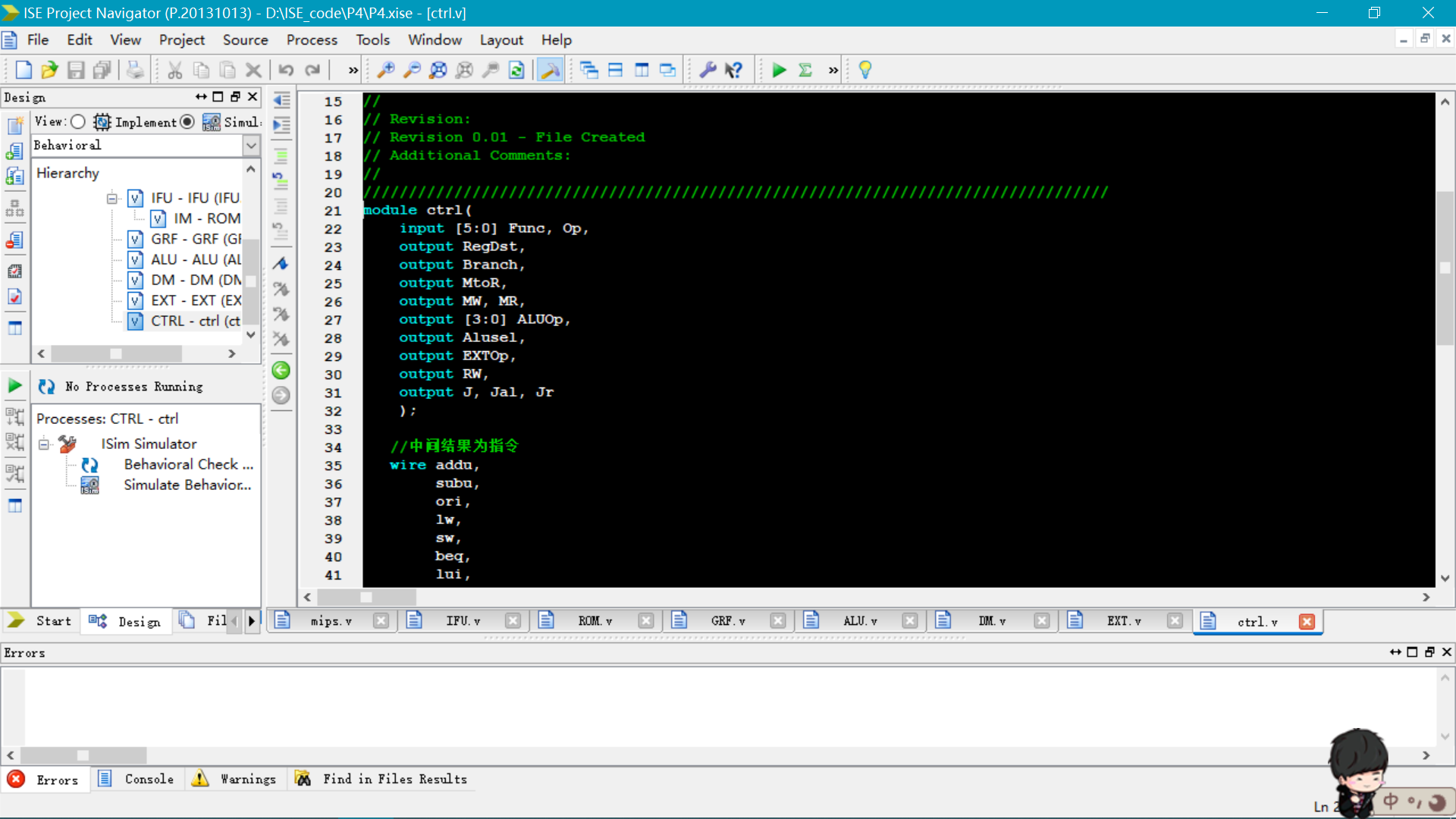


图 7-Controller模块图

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) #no execute

for\_2:

sw $a1, 4($t0) #$t0所存地址加4后的位置存储$a1

CPU-text-3:

# CPU-text file-3

# j,jal,jr

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

ori $a0, $0, 0xffff #$a0=0x0000ffff

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

j for\_2 #jump to for\_2

addu $a0, $0, $0 #no execute

for\_1:

sw $a0, 0($t0) #$t0所存地址加4后的位置存储$a0

jr $ra #return $31所存的地址

for\_2:

sw $a1, 4($t0) #$t0所存地址加4后的位置存储$a1

jal for\_1 #jump to for\_1,and store $31

lw $t1, 4($t0) #取出$t0所存地址加4后位置的数据到$t1

四、思考题

1.根据你的理解，在下面给出的DM的输入示例中，地址信号addr位数为什么是[11:2]而不是[9:0]？这个addr信号又是从哪里来的？

答：MIPS是按照字节存储，而我们所写的DM是按照字存储的。addr来自于ALU的输出口Result.

2.在相应的部件中，reset的优先级比其他控制信号（不包括clk信号）都要高，且相应的设计都是同步复位。清零信号reset是针对哪些部件进行清零复位操作？这些部件为什么需要清零？

答：PC寄存器复位0x00003000，GRF中31个寄存器，DM数据存储器。PC恢复到开始值，GRF寄存器和DM数据存储器清零，相当于程序重新开始。

1. 列举出用Verilog语言设计控制器的几种编码方式（至少三种），并给出代码示例。

答：（1）.case语句 （2）.if...else语句 （3）.assign三目运算符语句 （4）.define 或者parameter

**（1）**

**case(Op)**

**6'b000000:begin**

**case(Func)**

**6'b100001: addu = 'b1, subu = 'b0;**

**6'b100011: subu = 'b1, addu = 'b0;**

**default: begin addu = 'b0; subu = 'b0; end**

**end**

**6'b001101:**

**Ori = 'b1,addu = 'b0,subu = 'b0;**

**default: begin addu = 'b0; subu = 'b0, Ori = 'b0; end**

**endcase**

**（2）**

**If(Op == 6'b000000) begin**

**If(Func == 6'b100001) addu = 'b1, subu = 'b0;**

**Else if(Func == 6'b100011) subu = 'b1, addu = 'b0;**

**Else addu = 'b0; subu = 'b0;**

**End**

**Else if(Op == 6'b001101)**

**Ori = 'b1,addu = 'b0,subu = 'b0;**

**Else begin addu = 'b0; subu = 'b0, Ori = 'b0; end**

**（3）**

**assign addu = (Op == 6'b000000 && Func == 6'b100001) ? 'b1 : 'b0,**

**subu = (Op == 6'b000000 && Func == 6'b100011) ? 'b1 : 'b0,**

**ori = (Op == 6'b001101) ? 'b1 : 'b0；**

**（4）**

**`define Opaddu 6'b000000,**

**Opsubu 6'b000000,**

**Opori 6'b001101,**

**Funcaddu 6'b100001,**

**Funcsubu 6'b100011;**

1. 根据你所列举的编码方式，说明他们的优缺点。

答：if...else和case的优点：句义清晰易懂，方便阅读调试，缺点是代码冗长，容易混乱。

Assign 和 define代码较短，有较强的格式规范性，但不容易阅读调试。

5.C语言是一种弱类型程序设计语言。C语言中不对计算结果溢出进行处理，这意味着C语言要求程序员必须很清楚计算结果是否会导致溢出。因此，如果仅仅支持C语言，MIPS指令的所有计算指令均可以忽略溢出。 请说明为什么在忽略溢出的前提下，addi与addiu是等价的，add与addu是等价的。提示：阅读《MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set》中相关指令的Operation部分 。

答：加法执行后结果为32位，若向第33位进位则代表溢出，若忽略溢出的情况下，add,addu，addi,addiu指令不判断是否溢出，但0-31位的结果是一致的，也即运算方法是一样的。

6.根据自己的设计说明单周期处理器的优缺点。

答：优点：单周期的优点是每条指令逐步执行，各条指令之间不会发生寄存器和控制冒险，设计流程较为简单；缺点：执行周期太长，指令数非常多的时候CPU性能非常低。

1. 简要说明jal、jr和堆栈的关系。

答：jal指令跳转并保存当前指令后一条指令的地址至31号寄存器中，jr指令是跳转到31号寄存器所保存的地址。在MIPS代码编写中，为了保证之前31号寄存器所保存的地址不会发生改变，则需要一个栈来存储之前31号寄存器所保存的地址，当需要jr返回时，再取出来，这是一个进栈出栈的操作。