Verilog单周期CPU设计文档

1. CPU设计方案综述

本CPU为Verilog实现的单周期CPU（32位），支持的指令集包含**{addu, subu, ori, lw, sw, beq, lui, nop}，并进行了适当的指令扩展。**

**该CPU采用层次化，模块化的设计，主要包含IFU，GRF，EXT，ALU，DM，Controller等子模块。处理器顶层包含两个输入端口时钟信号clk和复位信号reset。**

1. 关键模块定义
2. IFU
3. 基本描述

IFU内部主要包括PC，NPC，IM（容量32bit\*1024，起始地址为0x00003000）以及相关逻辑。在NPC中产生下一条指令的地址，当时钟上升沿到来时，PC更新指令地址并将其输出，IM根据地址输出对应指令。

1. 端口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| CLK | I | 时钟信号 |
| Reset | I | 复位信号 |
| NPCOp[1:0] | I | 控制NPC进行相应的操作：  00：当前为顺序执行指令，NPC输出PC+4  01：当前指令为beq，作为决定是否跳转的条件之一  10：当前指令为jal，NPC输出PC31..28||instr\_index||02  11：当前指令为jr，NPC输出GRF[rs] |
| RA[31:0] | I | 将GRF[rs]的值输入IFU |
| Zero | I | 相等标志信号，判断ALU两操作数是否相等 |
| Instr[31:0] | O | 根据地址取到的当前指令 |
| PC4[31:0] | O | 输出PC+4作为地址 |
| PC[31:0] | O | 输出当前执行指令的地址 |

1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 当复位信号有效时，PC被设置为0x00003000 |
| 2 | 取指令 | 根据当前PC的值从IM中取出相应的指令并通过Instr端口输出 |
| 3 | 输出PC+4 | 在PC4输出端口输出PC+4 |
| 4 | 输出PC | 在PC端口输出当前执行指令的地址PC |
| 5 | 计算NPC | NPC根据NPCOp取值确定：  00：PC+4  01：PC+4+sign\_extend(offset||00)  （Zero为真时）；PC+4（Zero为假时）  10：PC||instr\_index||00  11：GRF[rs] |
| 6 | 更新PC | 当时钟上升沿到来时，更新PC为NPC |

1. GRF
   1. 基本描述

GRF模块内部具有32个具有写使能和复位功能的寄存器，0号寄存器内的值始终为0。GRF支持同时读取两个寄存器的值以及写入一个寄存器的操作。

* 1. 端口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| CLK | I | 时钟信号 |
| Reset | I | 复位信号 |
| PC | I | 当前执行指令的地址 |
| A1[4:0] | I | 地址输入信号，将对应地址寄存器的值输出至RD1 |
| A2[4:0] | I | 地址输入信号，将对应地址寄存器的值输出至RD2 |
| A3[4:0] | I | 地址输入信号，指定要进行写入的寄存器 |
| RFWr | I | 写使能信号 |
| WD[31:0] | I | 要写入寄存器的值 |
| RD1 | O | 数据输出信号，输出A1地址对应的寄存器的值 |
| RD2 | O | 数据输出信号，输出A2地址对应的寄存器的值 |

* 1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 当复位信号有效时，GRF中的寄存器全部复位（初值设置为0x00000000） |
| 2 | 读取数据 | 读取A1和A2地址所对应寄存器的数据至RD1和RD2输出端口 |
| 3 | 写入数据 | 当时钟上升沿到来时，如果RFWr信号有效，则将WD输入端口的数据写入A3地址所对应的寄存器中（无视对于0号寄存器的写入），并输出相应的写入信息 |

1. EXT
   1. 基本描述

EXT用于将16位立即数进行符号（无符号）扩展成32位、将十六位立即数加载到高位并输出。

* 1. 端口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| imm16[15:0] | I | 数据输入信号，输入要进行扩展的16位立即数 |
| EXTOp | I | 控制信号：  0：无符号扩展  1：符号扩展 |
| Ext[31:0] | O | 数据输出信号，输出扩展完毕的数据 |
| Imm32[31:0] | O | 数据输出信号，输出加载到高位的立即数 |

* 1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 无符号扩展 | 将16位立即数无符号扩展为32位 |
| 2 | 符号扩展 | 将16位立即数符号扩展为32位 |
| 3 | 立即数加载到高位 | 将16位立即数加载到高位成为32位输出数据 |

1. ALU
   1. 基本描述

ALU对输入的两个操作数（32bit）进行加、减、或、大小比较功能，输出运算的结果以及比较结果。

* 1. 端口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A[31:0] | I | 数据输入信号，输入ALU的第一个操作数 |
| B[31:0] | I | 数据输入信号，输入ALU的第二个操作数 |
| ALUOp[1:0] | I | 控制信号：  00：A+B  01：A-B  10：A|B |
| C[31:0] | O | 数据输出信号，输出ALU的计算结果 |
| Zero | O | 数据输出信号，输出两操作数进行相等比较的结果 |

* 1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能定义 |
| 1 | 加法 | 将两操作数相加 |
| 2 | 减法 | 将两操作数相减 |
| 3 | 或运算 | 将两操作数按位或 |
| 4 | 相等比较 | 判断两操作数是否相等，相等则Zero为真，反之为假 |

1. DM
   1. 基本描述

DM用于数据存储（容量为32bit\*1024，起始地址为0x00000000）。DM支持复位功能，采用单向双端口设计。每当时钟上升沿到来时，如果写使能有效则能将数据写入对应地址，每时每刻根据地址信号读出相应数据。

* 1. 端口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| CLK | I | 时钟信号 |
| Reset | I | 复位信号 |
| PC | I | 当前执行指令的地址 |
| DMWr | I | 写使能信号 |
| A[31:0] | I | 地址信号，指定要进行操作的存储单元的地址 |
| WD[31:0] | I | 数据输入信号，输入要写入存储单元的数据 |
| RD[31:0] | O | 数据输出信号，输出地址对应的存储单元的数据 |

* 1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 当复位信号有效时，每一个存储单元都被复位为0x00000000 |
| 2 | 读取 | 根据A地址信号输出对应存储单元的数据至RD输出端口 |
| 3 | 写入 | 当时钟上升沿到来时，如果DMWr有效，则根据A地址信号将WD输入端口中的数据写入对应的存储单元，同时输出写入信息 |

1. DC
   1. 基本描述

DC用于将Instr中的各位按照不同编码规则进行处理，方便之后各个子模块接收相应的指令信息

* 1. 端口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Instr | I | 数据输入信号，当前执行指令 |
| opcode[5:0] | O | 数据输出信号，当前指令的opcode |
| funct[5:0] | O | 数据输出信号，当前指令的funct |
| rs[4:0] | O | 数据输出信号，当前指令的rs |
| rt[4:0] | O | 数据输出信号，当前指令的rt |
| rd[4:0] | O | 数据输出信号，当前指令的rd |
| imm16[15:0] | O | 数据输出信号，当前指令的imm |
| index26[25:0] | O | 数据输出信号，当前指令的index |

* 1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 指令解析 | 将输入的指令进行处理，输出相应的信息，方便其他子模块接收信息 |

7、DC

* 1. 基本描述

MUX（多路选择器），当有多个信号同时输入一个端口时需要根据控制信号进行选择，使用时根据输入信号的个数选择不同的MUX

* 1. 端口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Ini[?:0] | I | 第i个数据的输入信号 |
| Sel[?:0] | I | 数据输出信号，选择相应的输入数据 |
| Out[?:0] | O | 数据输出信号，输出对应的数据 |

* 1. 功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 选择输入信号 | MUX功能部件的集合，根据需要实例化即可 |

1. 数据通路设计

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 部件 | IFU | GRF | | | | EXT | ALU | | DM | |
| 输入信号 | RA | A1 | A2 | A3 | WD | imm | A | B | A | WD |
| addu |  | IFU.Instr[25:21] | IFU.Instr[20:16] | IFU.Instr[15:11] | ALU.C |  | GRF.RD1 | GRF.RD2 |  |  |
| subu |  | IFU.Instr[25:21] | IFU.Instr[20:16] | IFU.Instr[15:11] | ALU.C |  | GRF.RD1 | GRF.RD2 |  |  |
| ori |  | IFU.Instr[25:21] |  | IFU.Instr[20:16] | ALU.C | IFU.Instr[15:0] | GRF.RD1 | GRF.RD2 |  |  |
| lw |  | IFU.Instr[25:21] |  | IFU.Instr[20:16] | DM.RD | IFU.Instr[15:0] | GRF.RD1 | EXT.Ext | ALU.C |  |
| sw |  | IFU.Instr[25:21] | IFU.Instr[20:16] |  |  | IFU.Instr[15:0] | GRF.RD1 | EXT.Ext | ALU.C | GRF.RD2 |
| beq |  | IFU.Instr[25:21] | IFU.Instr[20:16] |  |  |  | GRF.RD1 | GRF.RD2 |  |  |
| lui |  |  |  | IFU.Instr[20:16] | IFU.Instr[15:0]|| |  |  |  |  |  |
| jal |  |  |  | 0x1f | IFU.PC4 |  |  |  |  |  |
| jr | GRF.RD1 | IFU.Instr[25:21] |  |  |  |  |  |  |  |  |
| nop |  | | | | | | | | | |
| 综合 | RA | IFU.Instr[25:21] | IFU.Instr[20:16] | IFU.Instr[15:11]  IFU.Instr[20:16]  0x1f | ALU.C,DM.RD,IFU.PC4,  IFU.Instr[15:0]|| | IFU.Instr[15:0] | GRF.RD1 | GRF.RD2  EXT.Ext | ALU.C | GRF.RD2 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 输出 | 0端口 | 1端口 | 2端口 | 3端口 |
| GRF.A3 | IFU.Instr[15:11] | IFU.Instr[20:16] | 0x1f |  |
| GRF.WD | ALU.C | DM.D | IFU.PC4 | IFU.Imm32 |
| ALU.B | GRF.RD2 | EXT.Ext |  |  |

1. 控制器设计
   1. 基本思路

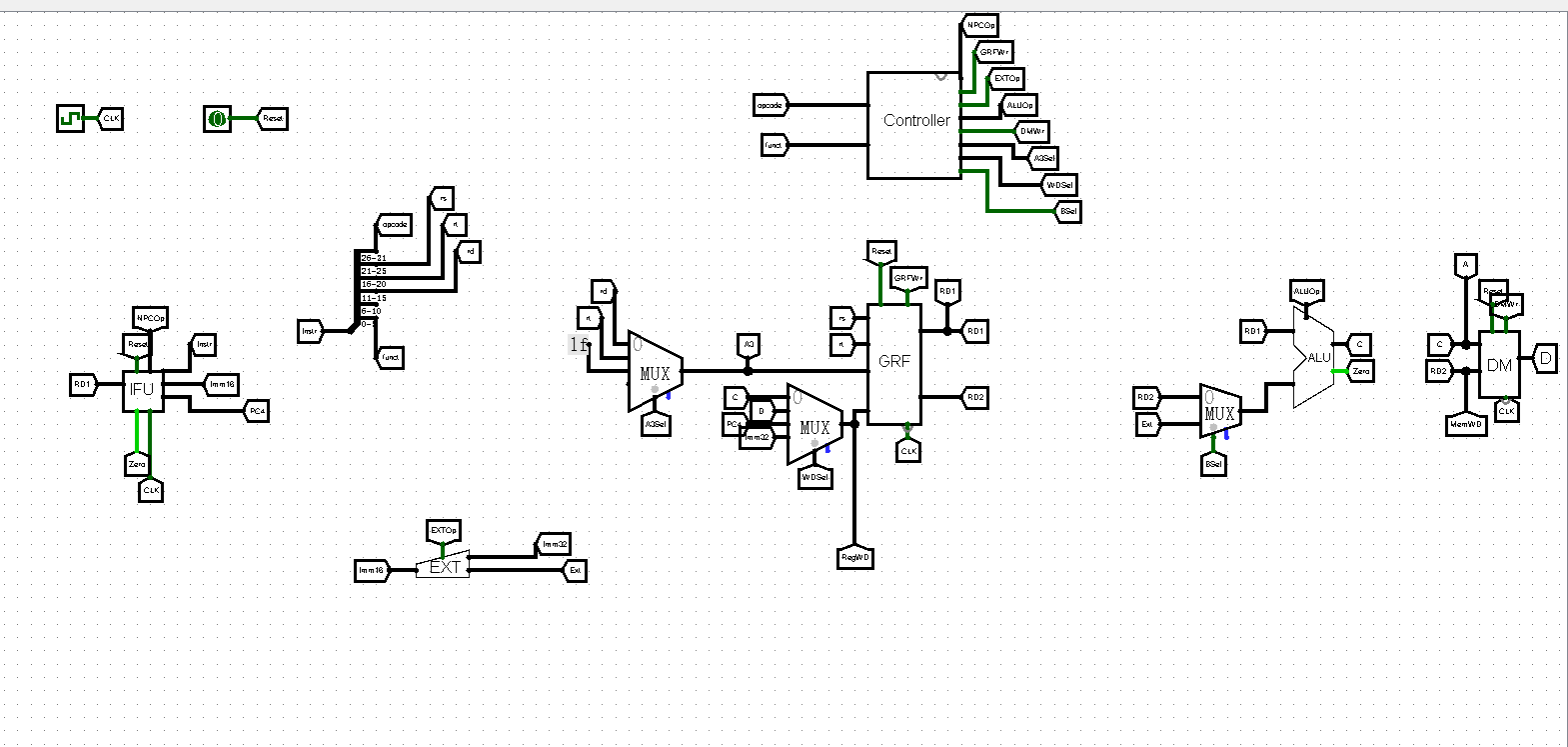
通过指令的opcode和funct产生数据通路所需要的控制信号，具体操作为先通过与阵列得到指令变量，再通过或阵列得到各控制信号的取值。

* 1. 真值表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令 | NPCOp[1:0] | GRFWr | EXTOp | ALUOp[1:0] | DMWr | A3Sel[1:0] | WDSel[1:0] | BSel |
| addu  (000000/100001) | 00 | 1 | x | 00 | 0 | 00 | 00 | 0 |
| subu  (000000/100011) | 00 | 1 | x | 01 | 0 | 00 | 00 | 0 |
| Xor  (000000/111110) | 00 | 1 | X | 11 | 0 | 00 | 00 | 0 |
| ori  (001101) | 00 | 1 | 0 | 10 | 0 | 01 | 00 | 1 |
| lw  (100011) | 00 | 1 | 1 | 00 | 0 | 01 | 01 | 1 |
| sw  (101011) | 00 | 0 | 1 | 00 | 1 | xx | xx | 1 |
| beq  (000100) | 01 | 0 | x | 01 | 0 | xx | xx | 0 |
| jal  (000011) | 10 | 1 | x | xx | 0 | 10 | 10 | x |
| Bsoal  (111111) | 01 | 0（judge=0）  1（judge=1） | X | Xx | 0 | 10 | 10 | x |
| swrr | 00 | 0 | 1 | 00 | 1 | xx | xx | 1 |
| jr  (000000/001000) | 11 | 0 | x | xx | 0 | xx | xx | x |
| lui  (001111) | 00 | 1 | x | xx | 0 | 01 | 11 | x |

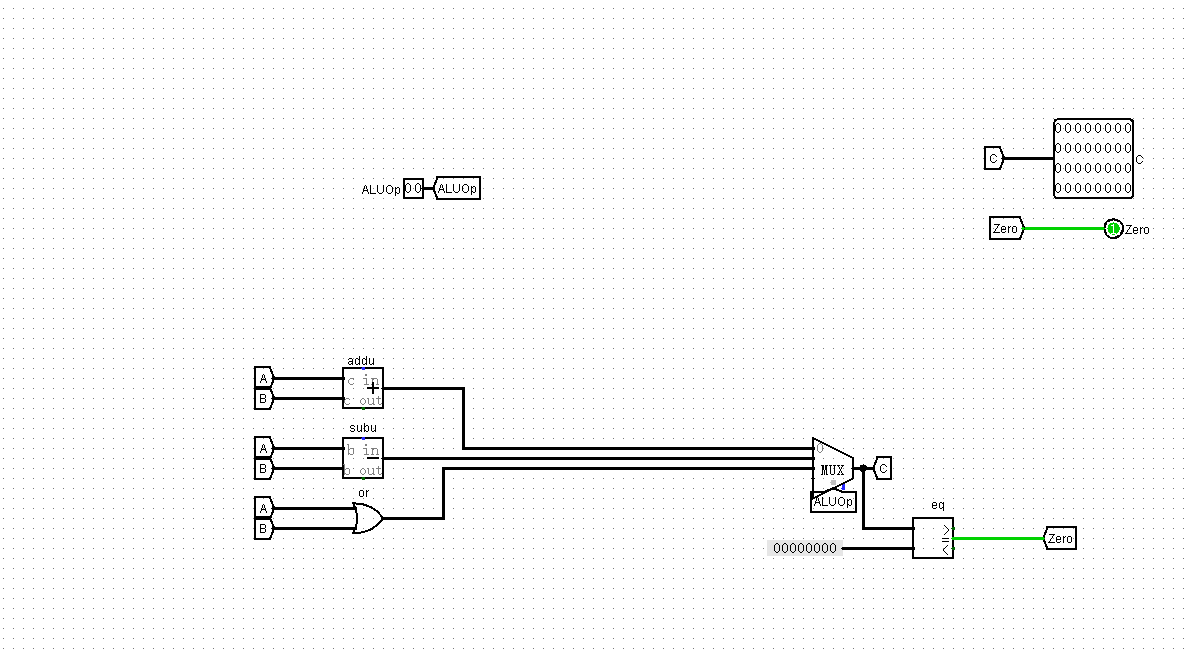
1. 顶层及子模块电路图

MIPS：

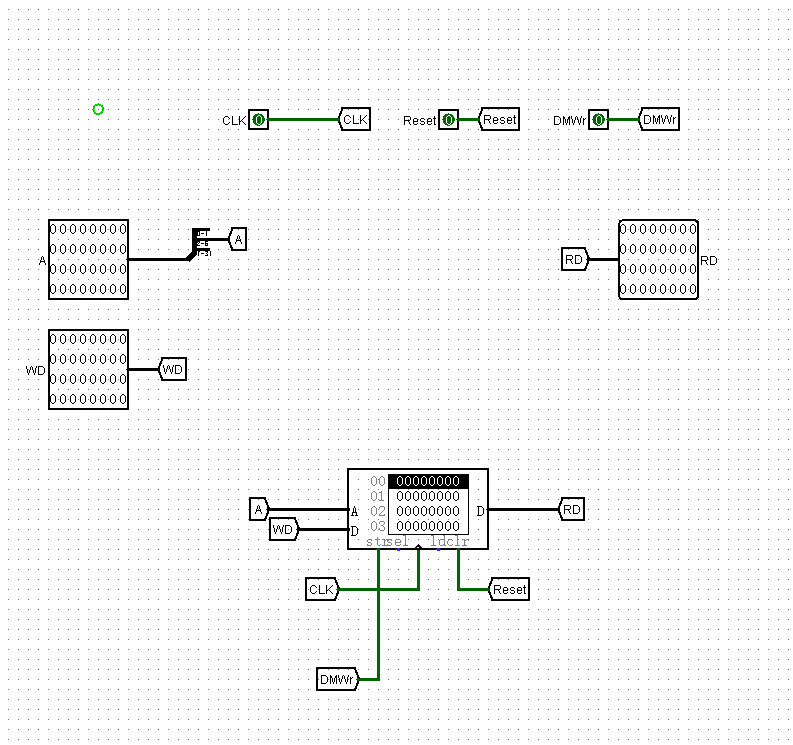


（注意Equ Gre Less 传入CU CU传出Br进行判断）

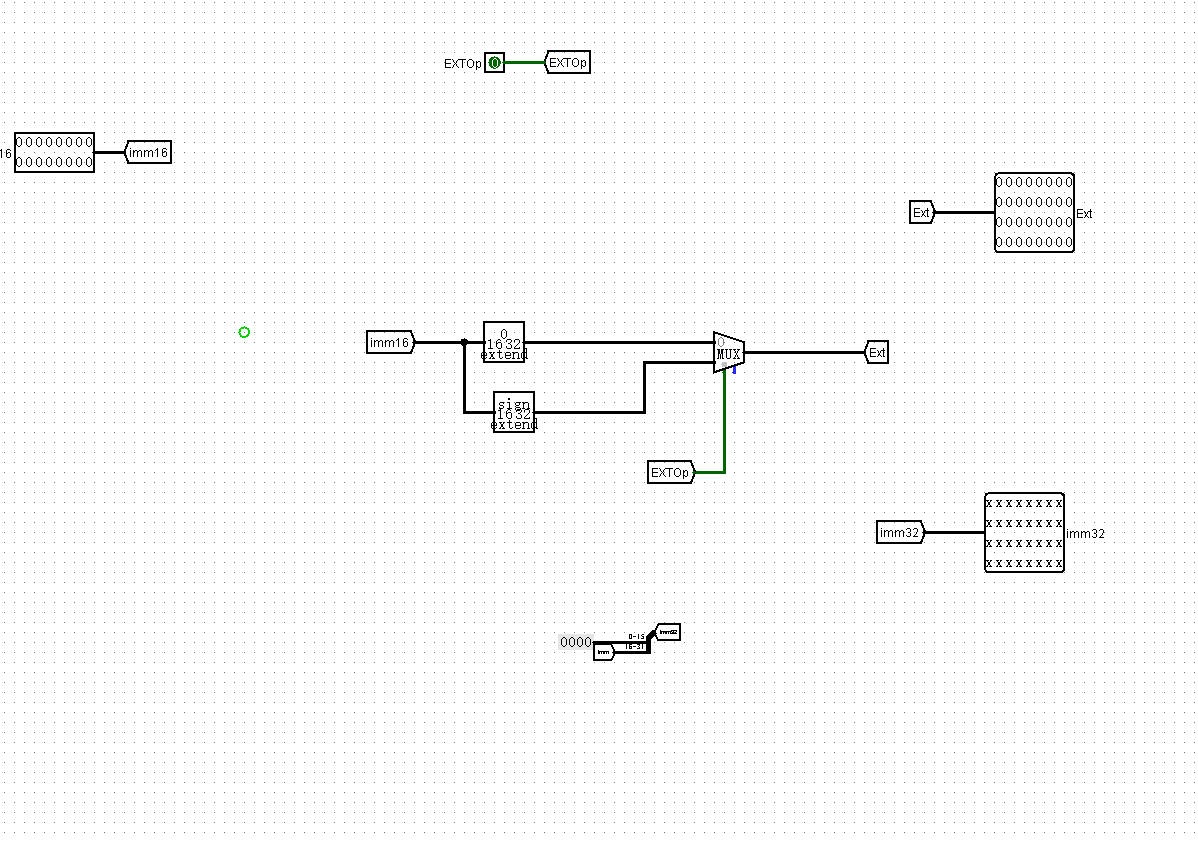
ALU：



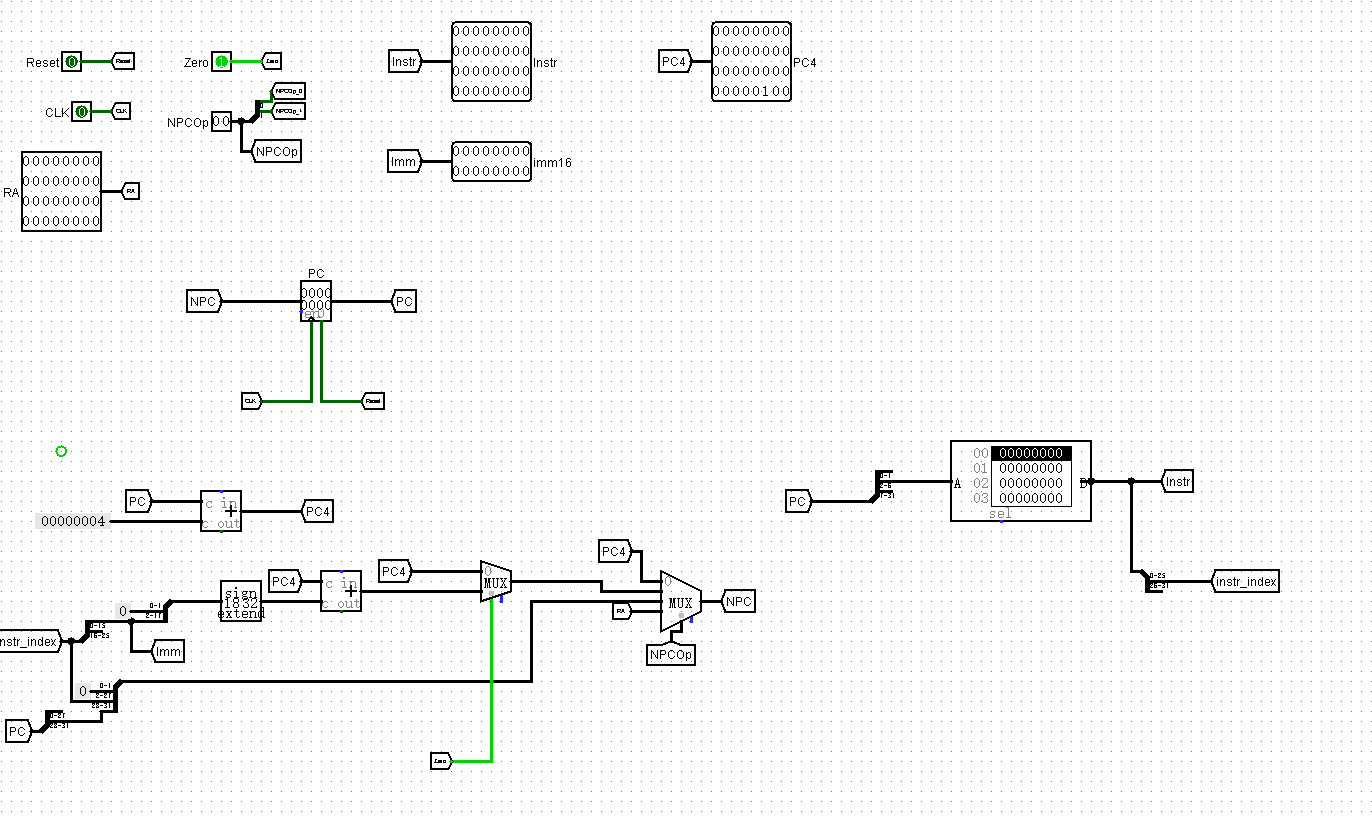
DM：



EXT：



IFU：



1. 测试方案
   1. 测试代码

.data

a: .word 1:32

.text

ori $t0,$t0,7 #$t0=7

ori $t1,$t1,15 #t1=15

ori $t3,$t3,4 #t3=2

ori $t5,$t5,1 #t5=1

nop

ori $t6,$t6,1

beq $t5,$t6,next

ori $t0,$t0,6

ori $t1,$t1,6

ori $t3,$t3,6

next:

addu $t1,$t0,$t0 #t1=14

subu $t2,$t1,$t0 #t2=7

lw $t4,a($t3)

sw $t1,a($t3)

jal test

addu $s0,$s0,$t5

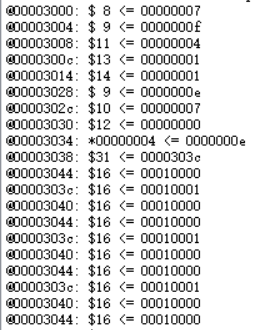
subu $s0,$s0,$t5

test:

lui $s0,1

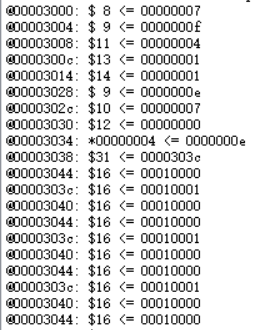
jr $ra

* 1. 期望结果



(…)

* 1. 测试结果



(…)

1. 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？

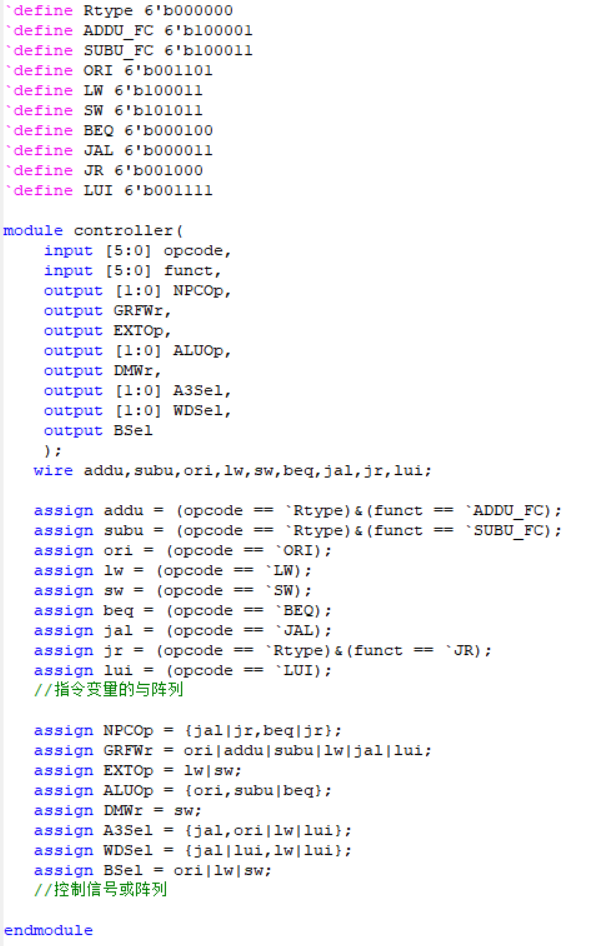


因为本CPU的DM的内部是用1024个（10bit）32位寄存器实现的，最小存储单元（32bit），而不是（8bit），所以首地址应该是4的倍数，所以应该截取最后两位，所以是是[11:2]而不是[9:0]。

这个信号从ALU的C端口来，在ALU中完成了地址的计算。

* 1. 思考Verilog语言设计控制器的译码方式，给出代码示例，并尝试对比各方式的优劣。

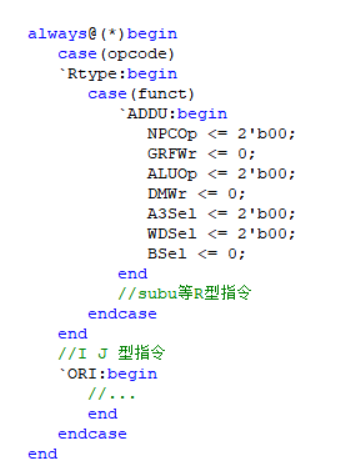
1. 与或阵



优点：接近底层实现，代码短，之后扩展指令的时候很方便

缺点：不太直观，不能一眼看出当前指令下各个控制信号的取值

1. Always case语句



优点：直观简单，能一眼看出当前指令下的控制信号的取值

缺点：代码量大，需要写很多代码，当需要扩展指令的时候，代码量比较大

1. 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所驱动的部件具有什么共同特点？

PC，GRF，DM等都需要有确定的初始值，若不能复位，这些部件内部存储的值不确定，可能使CPU没法正常工作。

1. 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 。

add和addu、addi和addiu的区别仅仅在于溢出时会不会触发异常，C语言在执行加法时即使溢出也不会触发异常，即C语言忽略溢出；MIPS如果仅支持C语言，那么就不用关心异常的触发与否，即忽略溢出的前提下，这些指令是等价的。

1. 根据自己的设计说明单周期处理器的优缺点。

优点：设计简单，结构简单。

缺点：所有指令的时钟周期等长，时钟周期只能由关键路径决定，使得某些执行得很快的指令也得执行相同的时钟周期，造成时间的浪费，部件的空闲，没有得到有效利用。