datapath包括IM、PC、Add、Nadd、Shift1、Shift2、RegDst\_MUX、ALUSrc\_MUX、MemtoReg\_MUX、PC1\_MUX、PC2\_MUX、GRF、ALU、DM、EXT等部分

PC:

基本描述

PC模块的主要功能是将PC\_in[31:0]的值保存并输出。PC的各种取值将根据所执行的指令、外部状态（中断）及处理器控制器的当前状态不同，由数据通路其他部件生成。

模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | MIPS-C处理器时钟 |
| reset | I | 复位信号 |
| PC\_in[31:0] | I | 下一个PC值 |
| PC\_out[31:0] | O | PC输出 |

功能定义

PC模块的核心是一个寄存器，该寄存器在clk上升沿将PC\_in[31:0]锁存并输出。

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 初始化 | 当reset信号有效时，PC输出0x00003000 |
| 2 | PC更新 | 当时钟上升沿到来时，将PC\_in写入PC内部，并且从PC端口输出 |

IM:

基本描述

IM模块的主要功能是储存32bit指令并输出。当IM的Read address端口接收到来自PC的指令，则从IM内部取出指令，传给控制器、寄存器堆、Shift和EXT。

模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| IM\_add | I | PC所指向的地址 |
| IM\_Instr[31:0] | O | IM输出的指令 |
| IM\_Rs[4:0] | O | Rs，传输至GRF |
| IM\_Rt[4:0] | O | Rt，传输至GRF |
| IM\_Rd[4:0] | O | Rd，传输至GRF |
| IM\_Op[5:0] | O | Op，传输至主控制器 |
| IM\_func[5:0] | O | func，传输至ALU控制器 |
| IM\_imm[15:0] | O | Imm，传输至EXT |
| IM\_index[25:0] | O | Index，传输至移位器2 |

功能定义

IM由一个32bit\*32的ROM实现，时钟上升沿来临时，读入PC所指定的地址，根据PC所指向的地址，取出对应的指令

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 取出指令 | 读入PC所指定的地址，取出对应的指令 |

Add:

基本描述

是一个二输出的32位加法器。Add模块的主要功能是将当前PC所指的地址加4，其输出作为Nadd和一个由信号 nPC\_Sel和Zero控制的Mux以及Shift2的输入。

模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Add\_A  [31:0] | I | 第一个加数，是当前PC |
| Add\_out  [31:0] | O | Add的输出 |

功能定义

实现PC+4。

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 加法 | 输入当前PC，与立即数4做加法，并输出 |

Nadd：

基本描述

是一个二输出的32位加法器，一个输入来自Add的输出，另一个输入来自Shift1的输出，二者相加后，得到的结果作为Nadd的输出。

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Nadd\_A  [31:0] | I | 第一个加数，是Add的输出 |
| Nadd\_B  [31:0] | I | 第二个加数，Shift1的输出 |
| Nadd\_out  [31:0] | O | Nadd的输出 |

功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 加法 | 输入Add和Shif1的输出，二者做加法并输出 |

实现lw和sw指令时，Nadd的输出指向内存中的一个地址。

Shift1：

基本描述

移位器1。将来自EXT的输出左移两位后输出。

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Shift1\_in  [31:0] | I | Shift1的输入，来自EXT的输出 |
| Shift1\_out  [31:0] | O | Shift1的输出 |

功能定义

实现移位功能，执行beq指令时用到。

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 移位 | 将输入左移两位后输出 |

Shift2：

基本描述

移位器2。将来自IM中取出的指令的后26位左移两位后输出。

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Shift2\_in1[25:0] | I | Shift2的输入，来自IM中取出的指令的后26位 |
| Shift2\_in2[31:0] | I | Shift2的输入，来自Add的输出 |
| Shift2\_out | O | Shift2的输出，为PC<31:28>||Instr<25:0>||00 |

功能定义

实现移位功能。当执行jal、j指令时，PC的输入为Shift2的输出。

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 移位 | 输出等于PC<31:28>||Instr<25:0>||00 |

PC1\_MUX:

基本描述

选择器。通过Zero和nPC\_Sel信号选择Nadd的输出或Add的输出作为该Mux的输出。

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| PC1\_A | I | 该多选器的第一个输入信号，来自Add的输出 |
| PC1\_B | I | 该多选器的第二个输入信号，来自Nadd的输出 |
| nPC\_Sel | I | 该选择器的控制信号，当Zero和nPC\_Sel同时为1时，该选择器选择PC1\_B输出 |
| Zero | I | 该选择器的控制信号，当Zero和nPC\_Sel同时为1时，该选择器选择PC1\_B输出 |
| OUT | O | 该多选器的输出 |

功能定义

实现选择功能。

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 选择 | 通过Zero和nPC\_Sel信号选择Add的输出或Nadd的输出作为该Mux的输出 |

PC2\_MUX:

基本描述

选择器。通过Jump信号选择Mux1的输出或Shift2或GRF\_RD1作为该Mux的输出。

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| PC2\_A  <31:0> | I | 该多选器的第一个输入信号，来自Mux1的输出 |
| PC2\_B  <31:0> | I | 该多选器的第二个输入信号，来自Shift2的输出 |
| PC2\_C  <1:0> | I | 该多选器的第三个输入信号，来自GRF\_RD1 |
| Jump  <1:0> | I | 该多选器的选择信号，为00时，选择PC2\_A  输出，为01时，选择PC2\_B  输出，否则，选择PC2\_C输出 |
| PC2\_out | O | 该多选器的输出 |

功能定义

实现选择功能。当执行jal和j指令时，选择PC2\_B输出，当执行jr指令时，选择PC2\_C输出，否则，选择PC2\_A输出。

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 选择 | 通过Jump信号选择输出 |

GRF：  
 基本描述

32个32位寄存器所组成的通用寄存器组，也称寄存器堆、寄存器文件。寄存器中存放数据其中0号寄存器内的值始终为0，其余31个寄存器的值可根据需要修改。（注意，GRF一直都是可读的，不受时钟的约束）

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号，当时钟上升沿来临时，如果reset有效，PC复位至0x00003000，寄存器中的内容清零 |
| GRF\_WPC  [31:0] | I | 当前PC位置 |
| GRF\_R1  [4:0] | I | 是IM输出的第25~21位，指向第一个储存数据的寄存器 |
| GRF\_R2  [4:0] | I | 是IM输出的第20~16位，指向第二个储存数据的寄存器 |
| GRF\_WR  [4:0] | I | RegDst\_MUX的输出，指向一个即将写入数据的寄存器 |
| GRF\_WD  [31:0] | I | 即将存入寄存器中的数据 |
| RegWrite | I | RegWrite有效时，GRF是可写的 |
| GRF\_RD1  [31:0] | O | 存在GRF\_R1中的数据，可能是jr指令中相应寄存器中的地址，或参与ALU运算 |
| GRF\_RD2  [31:0] | O | 存在GRF\_R2中的数据，可能是sw中写入内存的数据，或参与ALU运算 |

功能描述

从IM取出的指令的第25~21位作为GRF\_R1的输入，第20~16位作为GRF\_R2的输入，执行R型指令时，指令的15~11位作为GRF\_WR的输入，执行lw型指令时，指令的第20~16位作为GRF\_WR的输入,执行jal指令时，选择31作为的输入；GRF\_WD端口的数据来自一个由MemtoReg控制的Mux，当MemtoReg为000时，选择ALU的输出作为输出，为001时，选择DM的输出作为输出，为010时，选择PC+4作为输出。

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 时钟上升沿来临时，如果reset有效，GRF寄存器中的值全部清零 |
| 1 | 读数据 | 根据GRF\_R1和GRF\_R2中读入的数据确定相应的寄存器，并从GRF\_RD1和GRF\_RD2输出相应数据 |
| 2 | 写数据 | 当RegWrite有效时，把从GRF\_WD端口读入的数据写入端口GRF\_WR读入数据指向的寄存器中 |

ALU：

基本描述

逻辑运算单元，支持32位的加法、减法、与、或运算

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| ALU\_A  [31:0] | I | ALU的第一个输入，来自GRF\_RD1  的输出 |
| ALU\_B  [31:0] | I | ALU的第二个输入，来自通过信号ALUSrc\_MUX的输出 |
| ALUoperation  [2:0] | I | 控制ALU的功能 |
| ALU\_Zero | O | 进行减法运算时，如果两个输入相等，ALU\_Zero置1，否则为0 |
| ALU\_Result  [31:0] | O | ALU执行加法、减法、与、或运算的结果，作为DM端口DM\_add的输入和MemtoReg\_MUX的输入 |

功能描述

ALU支持32位的加法、减法、或运算和或运算，根据ALUoperation信号的不同进行不同的运算。

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 加法 | 当ALUoperation为0001时，进行加法运算 |
| 2 | 减法 | 当ALUoperation为0010时，进行减法运算 |
| 3 | 与运算 | 当ALUoperation为0100时，进行与运算 |
| 4 | 或运算 | 当ALUoperation为1000时，进行与运算 |

DM：

基本描述

数据存储器，容量为32bit\*1024，可读可写。注意，读DM不受时钟约束。

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| DM\_add[31:0] | I | 第[11:2]位指向内存中的一个地址，来自于ALU端口ALU\_Result的输出 |
| DM\_WD[31:0] | I | 即将写入DM的数据，来自GRF\_RD2的输出 |
| DM\_NPC[31:0] | I | PC当前所指地址 |
| MemRead | I | 实际上此信号没有用，从DM中读出相应数据不受MemRead和clk的约束 |
| MemWrite | I | 时钟上升压来临时，如果MemWrite有效，可以将数据写入内存中相应地址下的空间 |
| DM\_RD[31:0] | O | 输出内存中相应地址下的值 |

功能描述

DM可读可写，当MemRead有效时，可以从DM中读出相应数据，当MemWrite有效时，可以将数据写入内存中相应地址空间

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读数据 | 当MemRead有效时，可在内存中找到Address对应的地址，并读出相应数据，从Read data中输出 |
| 2 | 写数据 | 当MemWrite有效时，可以将数据写入内存中Write data  相应地址下的空间 |

EXT

基本描述

扩展器，将16位的立即数扩展至32位，当选择信号为01时，进行符号扩展，当选择信号为00时，把原数据移入高16位，低位补0，当选择信号为10时，进行符号扩展。

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| EXT\_in  [15:0] | I | EXT的输入 |
| EXTOp  [1:0] | I | 选择EXT的功能 |
| EXT\_out  [31:0] | O | EXT的输出 |

功能描述

当执行lui指令时（即选择信号为00时），把原数据移入高16位，低位补0；当执行lw、sw、beq指令时（即选择信号为01），进行有符号扩展，当执行ori指令时（即选择信号为10），进行无符号扩展。

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 加载至高位，低位补零 | 当选择信号为00时，把原数据移入高16位，低位补0 |
| 2 | 符号扩展 | 当选择信号为01时，进行有符号扩展 |
| 3 | 无符号扩展 | 当选择信号为10时，进行无符号扩展 |

RegDst\_MUX

基本描述

选择器。通过信号RegDst选择该选择器的输出。当RegDst为00时，选择RegDst\_A作为输出，当RegDst为01时，选择RegDst\_B作为输出，RegDst为10时，选择3’b11111作为输出。

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| RegDst\_A  [4:0] | I | 该多选器的第一个输入信号，来自IM\_Rt的输出 |
| RegDst\_B  [4:0] | I | 该多选器的第二个输入信号，来自IM\_Rd的输出 |
| RegDst  [1:0] | I | 该多选器的选择信号 |
| RegDst\_out  [4:0] | O | 该多选器的输出 |

功能描述

实现选择功能。当执行R型运算类指令时，选择RegDst\_A输出，当执行lw、ori、lui指令时，选择RegDst\_B输出，当执行jal指令时，选择5’b11111输出。

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 选择 | 通过RegDst选择输出 |

ALUSrc\_MUX

基本描述

选择器。通过信号ALUSrc选择该选择器的输出。当ALUSrc为00时，选择ALUSrc\_A作为输出，当ALUSrc为01时，选择ALUSrc\_B作为输出。

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| ALUSrc\_A  [31:0] | I | 该多选器的第一个输入信号，来自GRF\_RD2的输出 |
| ALUSrc\_B  [31:0] | I | 该多选器的第二个输入信号，来自EXT\_out的输出 |
| ALUSrc  [1:0] | I | 该多选器的选择信号 |
| ALUSr\_out  [31:0] | O | 该多选器的输出 |

功能描述

实现选择功能。执行R型运算类指令时，选择ALUSrc\_A，执行lw、sw、ori、lui时，选择ALUSrc\_B输出。

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 选择 | 通过ALUSrc选择输出 |

MemtoReg\_MUX

基本描述

选择器。通过信号MemtoReg选择该选择器的输出。当MemtoReg为000时，选择MemtoReg\_A作为输出，当MemtoReg为001时，选择MemtoReg\_B作为输出。为010时，选择PC+4作为输出。

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| MemtoReg\_A  [31:0] | I | 该多选器的第一个输入信号，来自ALU\_Result的输出 |
| MemtoReg\_B  [31:0] | I | 该多选器的第二个输入信号，来自DM\_RD的输出 |
| MemtoReg\_C  [31:0] | I | 该多选器的第三个输入信号，来自Add的输出 |
| MemtoReg  [2:0] | I | 该多选器的选择信号 |
| MemtoReg\_out  [31:0] | O | 该多选器的输出 |

功能描述

实现选择功能。执行R型运算类指令ori、lui时，选择MemtoReg\_A，执行lw时，选择MemtoReg\_B输出。执行jal时，选择MemtoReg\_C输出。

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 选择 | 通过MemtoReg选择输出 |

control包括control和ALUcontrol两部分

control

基本描述

根据从IM中取出的指令的不同，控制器产生不同的信号控制ALU、GRF等部件的工作

|  |  |  |
| --- | --- | --- |
| 信号名 | 失效时作用 | 有效时作用 |
| RegDst  <1:0> | 无 | 为00时，寄存器堆地址写入端口选择 Rt字段，为01时选择Rd字段，为11时，选择31 |
| RegWrite | GRF不可写 | 把数据写入寄存器堆中对应寄存器 |
| ALUSrc  <1:0> | 无 | ALU的输入端口B选择GRF的输出GRF\_RD2，ALU输入端B选择EXT的输出 |
| MemRead | 无 | 数据存储器DM读数据（输出） |
| MemWrite | DM不可写 | 数据存储器DM写数据（输入） |
| MemtoReg  <2:0> | 无 | 为000时，寄存器堆写入端数据来自ALU的输出Result；  为001时，寄存器堆写入端数据来自DM输出；  为010时，选择PC+4作为寄存器写入端的数据 |
| ALUOp  <2:0> | 无 | 控制ALUcontrol的输出  ALUOp为000时，ALUoperation为0001，ALU执行访存指令和lui指令所需加法；  ALUOp为001时，ALU operation为0010，ALU执行beq指令所需减法；  ALUOp为010时，ALUoperation由R型指令功能码决定；  ALUOp为011时，ALU operation为1000，ALU执行ori指令所需或运算 |
| ALU operation  <3:0> | 无 | 当ALUoperation为0001时，进行加法运算；  当ALUoperation为0010时，进行减法运算；  当ALUoperation为0100时，进行与运算；  当ALUoperation为1000时，进行或运算 |
| EXTOp  <1:0> | 无 | 为00时，立即数加载至高位；  为01时，进行有符号扩展；  为10时，进行无符号扩展； |
| nPC\_sel | 选择Add的输出作为PC1\_MUX的输出 | 与Zero一起作用，选择Nadd的输出作为PC1\_MUX的输出 |
| Jump<1:0> | 无 | 为00时，选择来自PC1\_MUX的输出作为PC2\_MUX的输出；  为01时，选择Shift2作为输出；  为10时，选择GRF\_RD1作为输出 |

真值表

主控单元

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| func | 100001 | 100011 | 100101 | 001000 |  |  |  |  |  |  |
| op | 000000 | 000000 | 000000 | 000000 | 001101 | 100011 | 101011 | 000100 | 001111 | 000011 |
|  | addu | subu | or | Jr | ori | lw | sw | beq | lui | Jal |
| RegDst(2) | 01 | 01 | 01 | x | 00 | 00 | x | x | 00 | 10 |
| RegWrite | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| ALUSrc(2) | 00 | 00 | 00 | x | 01 | 01 | 01 | 00 | 01 | x |
| MemRead | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| MemWrite | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| MemtoReg(3) | 000 | 000 | 000 | x | 000 | 001 | x | x | 000 | 010 |
| EXTOp(2) | X | X | x | x | 10 | 01 | 01 | 01 | 00 | x |
| nPC\_sel(Branch) | 0 | 0 | 0 | x | 0 | 0 | 0 | 1 | 0 | x |
| ALUOp(3) | 010 | 010 | 010 | x | 011 | 000 | 000 | 001 | 000 | x |
| Jump(2) | 00 | 00 | 00 | 10 | 00 | 00 | 00 | 00 | 00 | 01 |

ALU控制单元：

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ALUOp | 010 | 010 | 010 | 011 | 000 | 000 | 001 | 000 | x | x |
| Func | 100001 | 100011 | 100101 |  |  |  |  |  |  | 001000 |
|  | addu | subu | or | ori | lw | sw | beq | lui | jal | jr |
| ALUoperation | 0001 | 0010 | 1000 | 1000 | 0001 | 0001 | 0010 | 0001 | x | x |

1. 测试文档

测试文档：

Ori:

ori $0, $zero, 1

ori $8, $zero, 2

ori $9, $zero, 3

ori $10, $zero, 4

ori $11, $zero, 0

ori $12, $zero, -6

ori $13, $8, 0

ori $14, $12, 0

ori $15, $11, 0

ori $16, $8, 3

ori $17, $9, 2

ori $18, $9, -6

ori $19, $12, 2

ori $20, $zero, -13

ori $21, $12, -13

ori $22, $20, -6

Lui:

lui $0, 8

lui $1, 16

lui $2, 15

lui $3, 15

lui $4, 16

lui $5, 15

lui $6, 15

lui $7, 16

lui $8, 15

lui $9, 15

lui $10, 16

lui $11, 15

lui $12, 15

lui $13, -0

lui $14, 0

Addu:

ori $8, $0, 8

ori $9, $0, 9

ori $10, $0, -8

ori $11, $0, -9

lui $22, 9

addu $12, $9, $8

addu $13, $8, $9

addu $14, $8, $11

addu $15, $11, $8

addu $16, $9, $10

addu $17, $10, $9

addu $18, $10, $11

addu $19, $11, $10

addu $20, $8, $10

addu $21, $10, $8

addu $23, $22, $8

Subu:

ori $8, 12

ori $9, 16

ori $10, 0

ori $11, -4

ori $12, -8

subu $13, $8, $9

subu $14, $9, $8

subu $15, $8, $11

subu $16, $11, $8

subu $17, $9, $zero

subu $18, $zero, $9

subu $19, $9, $10

subu $20, $10, $9

subu $0, $9, $10

subu $21, $12, $10

subu $22, $10, $12

subu $23, $zero, $12

subu $24, $12, $zero

subu $25, $zero, $10

subu $4, $11, $12

subu $5, $12, $11

Sw:

ori $8, $zero, 8

ori $9, $zero, 4

ori $10, $zero, 12

ori $11, $zero, 16

ori $12, $zero, 0

ori $13, $zero, -16

sw $11, 0($8)

sw $12, 4($8)

sw $13, 0($10)

sw $11, -4($11)

Lw:

ori $8, $zero, 8

ori $9, $zero, 4

ori $10, $zero, 12

ori $11, $zero, 16

ori $12, $zero, 0

ori $13, $zero, -16

sw $11, 0($8)

sw $12, 4($8)

sw $13, 0($10)

sw $11, -4($11)

sw $9, 0($11)

lw $11, 0($8)

lw $10, 4($8)

lw $13, 0($10)

lw $12, -4($11)

Beq:

ori $8, $zero, 8

ori $9, $zero, 12

ori $10, $zero, 8

change1:

ori $11, $zero, 1

beq $8, $9, change1

ori $12, $zero, 2

beq $8, $10, change1

ori $8, $zero, 8

ori $9, $zero, 12

ori $10, $zero, 8

ori $11, $zero, 1

beq $8, $9, change1

ori $12, $zero, 2

beq $8, $10, change1

ori $13, $zero, 3

change1:

ori $14, $zero, 4

ori $8, $zero, -8

ori $9, $zero, -12

ori $10, $zero, -8

ori $11, $zero, 1

beq $8, $9, change1

ori $12, $zero, 2

beq $8, $10, change1

ori $13, $zero, 3

change1:

ori $14, $zero, 4

Jal:

ori $8, $zero, 1

ori $9, $zero, 2

change2:

ori $10, $zero, 3

jal change1

ori $11, $zero, 4

change1:

ori $12, $zero, 5

jal change2

Jr:

ori $8, $zero, 1

ori $9, $zero, 2

ori $10, $zero, 3

ori $11, $zero, 4

ori $12, $zero, 5

ori $13, $zero, 6

ori $15, $zero, 0x00003008

jr $15

ori $8, $zero, 1

ori $9, $zero, 2

ori $10, $zero, 3

jal change

ori $11, $zero, 4

ori $12, $zero, 5

change:

ori $13, $zero, 6

ori $14, $zero, 7

jr $ra

测试文档

ori $t0, $zero, 8

ori $t1, $zero, 4

addu $t2, $t0, $t1

subu $t3, $t1, $t0

sw $t1, 0($t0)

lw $t4, 0($t0)

ori $t5, $zero, 8

beq $t0, $t5, change1

ori $t6, $zero, 12

change1:

ori $t6, $zero, 8

beq $t1, $t5, change2

ori $t6, $zero, 4

change2:

ori $t6, $zero, 3

jal change3

lui $t2, 4

change3:

ori $t2, $zero, 8

jr $ra

期望输出：

@00003000: $ 8 <= 00000008

@00003000: $ 8 <= 00000008

@00003004: $ 9 <= 00000004

@00003008: $10 <= 0000000c

@0000300c: $11 <= fffffffc

@00003010: \*00000008 <= 00000004

@00003014: $12 <= 00000004

@00003018: $13 <= 00000008

@00003024: $14 <= 00000008

@0000302c: $14 <= 00000004

@00003030: $14 <= 00000003

@00003034: $31 <= 00003038

@0000303c: $10 <= 00000008

@00003038: $10 <= 00040000

@0000303c: $10 <= 00000008

@00003038: $10 <= 00040000

@0000303c: $10 <= 00000008

@00003038: $10 <= 00040000

@0000303c: $10 <= 00000008

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？



答案：因为MIPS是按字节寻址的，一个字等于四个字节，为了和MIPS匹配，因此DM中地址一定是4的倍数，因此是[11：2]，这个addr的信号是ALU的输出ALU\_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？

答案：针对PC、GRF。

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。

答案：

assign编码：

Assign RegDst[1] = ~op[5] && ~op[4] && ~op[3] && ~op[2] && ~op[1] && ~op[0];

Case语句：

Case(op)

6’b000000: case(func)

6’b100001: begin

ALUtoMem = 2’bxx;

RegDst = 2’b01;

ALUSrc = 2’b00;

MemtoReg = 3’b000;

nPC\_Sel = 1’b0;

Jump = 2’b00;

EXTOp = 2’bxx;

ALUOp = 3’b010;

MemWrite = 1’b0;

MemRead = 1’b0;

RegWrite = 1’b1;

end

endcase

Endcase

Define：

`define addu\_op 6’b100001;

`define addu\_func 6’b000000;

If(control\_in1==`addu\_op && control\_in2==`addu\_func) begin

RegDst = 2’b01;

ALUSrc = 2’b00;

end

4.根据你所列举的编码方式，说明他们的优缺点。

答案：

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 。

答案：由是手册可知，add是在不溢出的情况下，把加法运算结果的第31到0位放到对应寄存器中，如果溢出，则报出错误信息；addu是直接把加法运算的结果放到相应寄存器中，因此，如果不考虑溢出，addu和add是等价的。

1. 根据自己的设计说明单周期处理器的优缺点。

答案：优点：构造简单，能够处理addu subu lw sw ori lui jal jr nop指令，逻辑清晰，每条指令都在前面的指令执行完后才开始执行，因此不存在冒险；

缺点：由于每条指令都等前面的指令时行完后才开始执行，因此部件的利用率很低，完成一定数目的指令需要较长时间。