**32bit RISC-V参考文档**

**1.alu module：**

32bit datapath，操作数为2个32bit数，rs1及rs2，目标寄存器为32bit的 rd， 实现如下32bit数据的算术以及逻辑功能：

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 逻辑 | 功能说明 | 操作 |
| 1 | Shift Left Logic: | 逻辑左移 | Rd = rs1 << rs2 |
| 2 | Shift right Logic | 逻辑右移 | Rd = rs1 >> rs2 |
| 3 | Shift right Arithmetci | 算术右移 | Rd = rs1 >> rs2 (arithmetic) |
| 4 | add | 加，如果溢出给出overflow信号 | Rd = rs1 + rs2 |
| 5 | Sub | 减，如果溢出给出overflow信号 | Rd = rs1 - rs2 |
| 6 | Load upper imm | 高位加载立即数 | Rd = {20bitImm, 12'd0} |
| 7 | Signed set if less than | 有符号数rs1和rs2的比较 | Rd = (signed rs1 < signed rs2) ? 1 : 0 |
| 8 | Unsigned set if less than | 无符号数rs1和rs2的比较 | Rd = unsigned rs1 < unsigned rs2 ? 1 : 0 |
| 9 | xor | 异或操作 | Rd = rs1 ^ rs2 |
| 10 | Or | 或操作 | Rd = rs1 | rs2 |
| 11 | and | 与操作 | Rd = rs1 & rs2 |
| 12 | Mul | 乘操作，忽略溢出 | Rd = (rs1 \* rs2)[31:0] |

模块的参考接口如下

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位宽 | I/O | 描述 |
| op | 4 | I | ALU的操作码定义 |
| rs1 | 32 | I | 第1操作数 |
| rs2 | 32 | I | 第2操作数 |
| rd | 32 | O | ALU运算结果操作数 |
| overflow | 1 | o | 操作溢出标志 |

**2.regfile module：**

2读1写的通用寄存器堆

寄存器堆数量32个，采用二维数组实现，读和写有可能同时发生，其中0号寄存器为常0（无论写什么数据，寄存器值都为0）

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位宽 | I/O | 描述 |
| clk | 1 | I | 时钟 |
| nrst | 1 | I | 全局异步时钟，低电平有效，有效时所有寄存器清零 |
| stall | 1 | I | 暂停信号，该信号为1时，寄存器所有值保持不变 |
| ren | 1 | I | 读使能，高电平有效 |
| radd1/radd2 | 5 | I | 两个读数据端口地址 |
| wen | 1 | I | 写使能，高电平有效 |
| wadd | 5 | I | 写数据端口地址 |
| wdata | 32 | I | 写端口数据 |
| rs1/rs2 | 32 | O | 从寄存器堆读出两个操作数 |

寄存器堆的接口结构如下图：



**3.memory module：**

存储器读写模块，模块功能如下：

存储模块总容量16KB，位宽32bit，考虑到节省功耗，将存储空间划分为4块memory实现，每块存储器4KB即1024\*32大小的存储块。4块memory依次编址

|  |  |
| --- | --- |
| 地址空间(字对齐地址) |  |
| 0xc00 ~ 0xfff | Mem3 |
| 0x800 ~ 0xbff | Mem2 |
| 0x400 ~ 0x7ff | Mem1 |
| 0x000 ~ 0x3ff | Mem0 |

存储器读写模块需要实现的功能如下：

|  |  |  |
| --- | --- | --- |
| 序号 | 操作名称 | 功能说明 |
| 1 | Load Byte | Byte数据装载，根据byte地址读取出相应8bit字节，经过符号位扩展后形成一个32bit数输出 |
| 2 | Load half word | 半字数据装载，根据半字地址（地址最低位为0）读取出相应16bit半字，经过符号位扩展后形成一个32bit数输出 |
| 3 | Load word | 整字数据装载，根据字地址（地址最低2bit为0）读取出一个32bit数输出 |
| 4 | Store Byte | Byte数据存储，将输入32bit数据的低8位根据byte地址存入存储器 |
| 5 | Store half word | 半字数据存储，将输入32bit数据的低16位根据半字地址存入存储器 |
| 6 | Store word | 整字数据装载，将输入32bit数据存入存储器 |

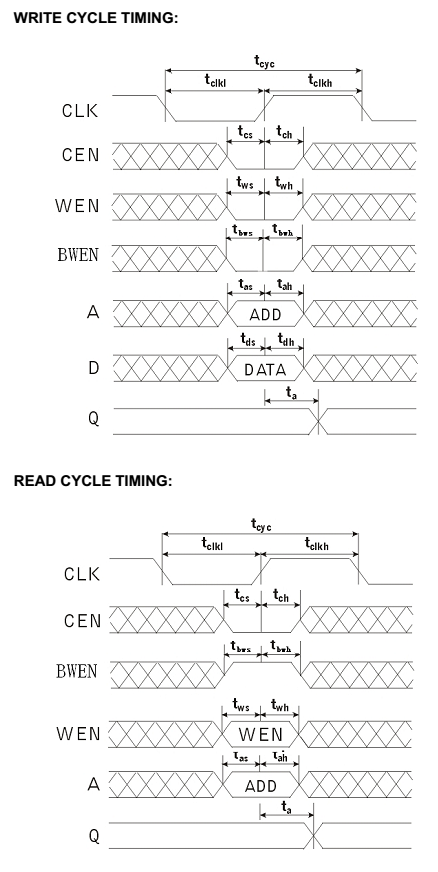
存储器读写模块的接口如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位宽 | I/O | 描述 |
| clk | 1 | I | 时钟 |
| nrst | 1 | I | 全局异步时钟，低电平有效，有效时所有寄存器清零 |
| stall | 1 | I | 暂停信号，该信号为1时，不进行任何读写操作 |
| op\_code | 3 | I | 读写操作类型，对应功能表格各个功能 |
| RWaddr | 14 | I | Byte对齐地址 |
| wdata | 32 | I | 写端口数据 |
| rdata | 32 | O | 从memory读出的32bit数据 |

用verilog搭建memory模型后进行模块例化调用，4KB存储器规模为1024\*32bit，其接口及功能定义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位宽 | I/O | 描述 |
| CLK | 1 | I | 时钟 |
| CEN | 1 | I | mem使能信号，低电平有效 |
| WEN | 1 | I | 写使能信号，低电平有效 |
| BWEN | 32 | I | Bit-write enable，按bit写使能 |
| A | 10 | I | 读写地址 |
| D | 32 | I | 写数据 |
| Q | 32 | O | 读数据 |

mem的读写波形如下：



4.**fetch module：**

**指令取指模块**



指令取指模块结构如上图，PC\_reg是一个32bit位宽寄存器，用于寄存PC地址，PC初始地址采用参数化定义以便后续修改调用，PC地址来源于两个通路：一为跳转地址，当遇到跳转使能br\_en有效时，PC选择br\_addr进行指令存储器的读取，否则PC顺序加一个字地址 后续指令访问。

IMEM模型访问波形同作业3中的存储器模型，总体大小为16KB分2块实现，从IMEM读出的32位数据 即为所需执行的指令

指令取指模块的接口如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位宽 | I/O | 描述 |
| clk | 1 | I | 时钟 |
| nrst | 1 | I | 全局异步时钟，低电平有效，有效时所有寄存器清零 |
| stall | 1 | I | 暂停信号，该信号为1时，不进行任何操作 |
| br\_en | 1 | I | 跳转使能信号 |
| br\_addr | 32 | I | 跳转地址 |
| PC | 32 | O | PC地址输出 |
| ins\_out | 32 | O | 从指令memory读出的32bit数据 |