P7设计文档

——MIPS微体系

1. 指令集

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| add | addu | sub | subu | sll | srl | sra | sllv | srlv | srav |
| and | or | xor | nor | slt | sltu | addi | addiu | andi | ori |
| xori | lui | slti | sltiu | beq | bne | blez | bgtz | bltz | bgez |
| j | jal | jalr | jr | lb | lbu | lh | lhu | lw | sb |
| sh | sw | mult | multu | div | divu | mfhi | mflo | mthi | mtlo |
| mtc0 | mfc0 | eret |  |  |  |  |  |  |  |

1. 模块规格
2. PC：指令地址寄存器（F级部件）

模块端口定义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位数 | 方向 | 描述 |
| clk |  | I | 内置时钟信号 |
| reset |  | I | 同步复位信号 |
| en |  | I | 写使能信号 |
| PCI | [31:0] | I | 下一条指令的地址 |
| PC | [31:0] | O | 当前指令的地址 |
| exccode\_PC | [4:0] | O | PC部件的异常编码 |

模块功能说明如下：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名 | 描述 |
| 1 | 读指令地址 | 通过PC端口输出当前指令地址 |
| 2 | 写指令地址 | 时钟上升沿到来时若en信号为1，则更新指令地址 |
| 3 | 同步复位 | 时钟上升沿到来时若reset信号为1，则复位指令地址至初始状态0x00003000 |
| 4 | 输出异常编码 | 指令地址超出范围或未字对齐时输出相应异常编码 |

1. IM：指令存储器（F级部件）

模块端口定义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位数 | 方向 | 描述 |
| IMI | [31:0] | I | 当前指令地址信号 |
| IM | [31:0] | O | 当前指令信号 |

模块功能说明如下：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名 | 描述 |
| 1 | 读指令 | 读出指令存储器中IMI地址对应的指令并通过IM端口输出 |
| 2 | 初始化写指令 | 初始时向指令存储器中读入所有指令 |

1. GRF： 寄存器堆（D级部件，W级部件）

模块端口定义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位数 | 方向 | 描述 |
| clk |  | I | 内置时钟信号 |
| reset |  | I | 同步复位信号 |
| PC | [31:0] | I | 当前指令的地址信号 |
| A1 | [4:0] | I | 第一个读寄存器地址输入信号 |
| A2 | [4:0] | I | 第二个读寄存器地址输入信号 |
| A3 | [4:0] | I | 写寄存器地址输入信号 |
| WD | [31:0] | I | 写入数据信号 |
| WE |  | I | 写使能信号 |
| RD1 | [31:0] | O | A1所对应的寄存器的数据信号 |
| RD2 | [31:0] | O | A2所对应的寄存器的数据信号 |

模块功能说明如下：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名 | 描述 |
| 1 | 读数据 | 读出A1，A2地址对应寄存器数据并通过RD1，RD2端口输出 |
| 2 | 写数据 | 时钟上升沿到来时，若WE信号为1，则向A3地址对应寄存器写入数据WD（0号寄存器不能被写入）并输出一条与PC有关的信息 |
| 3 | 同步复位 | 时钟上升沿到来时，若reset信号为1，则复位所有寄存器至初始状态0x00000000 |

1. EXT：立即数拓展模块（D级部件）

模块端口定义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位数 | 方向 | 描述 |
| EXTIMM | [15:0] | I | 参与拓展的立即数信号 |
| EXTOp | [1:0] | I | 拓展方式的选择信号 |
| EXT | [31:0] | O | 拓展完成后的数据信号 |

模块功能说明如下：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名 | 描述 |
| 1 | 符号拓展 | 若EXTOp==00，将立即数加载至输出信号低位并用其最高位填充输出信号的高16位 |
| 2 | 零拓展 | 若EXTOp==01，将立即数加载至输出信号低位并用0填充输出信号的高16位 |
| 3 | 低位零拓展 | 若EXTOp==10，将立即数加载至输出信号高位并用0填充输出信号的低16位 |

1. NPC：指令地址计算器（D级部件）

模块端口定义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位数 | 方向 | 描述 |
| ADD4 | [31:0] | I | PC+4信号 |
| PC4\_D | [31:0] | I | F/D级流水线寄存器中的PC4信号 |
| imm\_index | [25:0] | I | 计算跳转地址的imm/index数据信号 |
| M\_RD1\_D | [31:0] | I | D级读取rs寄存器时的转发信号 |
| CMP |  | I | 跳转时特殊条件的判断信号 |
| Is\_B |  | I | b类指令的判断信号（beq/bne/blez/bgtz/bltz/bgez） |
| Is\_J |  | I | j类指令的判断信号（j / jal） |
| PCOp | [1:0] | I | 计算下一条指令地址的选择信号 |
| IntReq |  | I | 中断请求信号 |
| EPC | [31:0] | I | CP0中EPC信号 |
| TURE\_NPC | [31:0] | O | 下一条指令地址 |

模块功能说明如下：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名 | 描述 |
| 1 | 计算指令地址 | 若TntReq有效，则输出0x00004180  若PCOp==00，输出PC+4信号  若PCOp==01，输出M\_RD1\_D信号  若PCOp==10且is\_B==1且CMP==1，则计算PC4\_D+imm拓展并输出  若PCOp==10且is\_J==1，则计算PC4\_D与index拓展信号并输出  若PCOp==10且不属于以上情况，则输出PC4\_D+4信号  若PCOp==11，则输出EPC信号 |

1. CMP：数据比较器（D级部件）

模块端口定义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位数 | 方向 | 描述 |
| D1 | [31:0] | I | 参与比较的第一个数据信号 |
| D2 | [31:0] | I | 参与比较的第二个数据信号 |
| CMPOp | [2:0] | I | 比较方式的选择信号 |
| CMP |  | O | 比较结果信号 |

模块功能说明如下：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名 | 描述 |
| 1 | 比较相等 | 若CMPOp==000，二数据信号相等，则输出1，否则输出0 |
| 2 | 比较不等 | 若CMPOp==001，若二数据信号不等，则输出1，否则输出0 |
| 3 | 比较小于0 | 若CMPOp==010，若D1小于0，则输出1，否则输出0 |
| 4 | 比较小于等于0 | 若CMPOp==011，若D1小于等于0，则输出1，否则输出0 |
| 5 | 比较大于0 | 若CMPOp==100，若D1大于0，则输出1，否则输出0 |
| 6 | 比较大于等于0 | 若CMPOp==101，若D1大于等于0，则输出1，否则输出0 |

1. ALU：计算模块（E级部件）

模块端口定义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位数 | 方向 | 描述 |
| A | [31:0] | I | 参与运算的第一个数据信号 |
| B | [31:0] | I | 参与运算的第二个数据信号 |
| ALUOp | [3:0] | I | 运算方式的选择信号 |
| IR\_E | [31:0] | I | E级指令信号 |
| ALU | [31:0] | O | 运算结果的数据信号 |
| exccode\_ALU | [4:0] | O | ALU部件的异常编码 |

模块功能说明如下：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名 | 描述 |
| 1 | 按位与运算 | 若ALUOp==0000，计算A&B并通过ALU端口输出 |
| 2 | 按位或运算 | 若ALUOp==0001，计算A|B并通过ALU端口输出 |
| 3 | 支持溢出加法 | 若ALUOp==0010，计算A+B并通过ALU端口输出 |
| 4 | 支持溢出减法 | 若ALUOp==0011，计算A-B并通过ALU端口输出 |
| 5 | 逻辑左移 | 若ALUOp==0100，计算B逻辑左移A并通过ALU端口输出 |
| 6 | 逻辑右移 | 若ALUOp==0101，计算B逻辑右移A并通过ALU端口输出 |
| 7 | 算术右移 | 若ALUOp==0110，计算B算术右移A并通过ALU端口输出 |
| 8 | 异或 | 若ALUOp==0111，计算A xor B并通过ALU端口输出 |
| 9 | 或非 | 若ALUOp==1000，计算A nor B并通过ALU端口输出 |
| 10 | 小于置1 | 若ALUOp==1001，A < B则输出1，否则输出0 |
| 11 | 无符号小于置1 | 若ALUOp==1010，A无符号 < B则输出1，否则输出0 |
| 12 | 输出异常编码 | 运算溢出时输出相应异常编码 |

1. XALU：乘除法计算模块（E级部件）

模块端口定义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位数 | 方向 | 描述 |
| clk |  | I | 内置时钟信号 |
| reset |  | I | 同步复位信号 |
| XALUA | [31:0] | I | 参与运算的第一个数据信号 |
| XALUB | [31:0] | I | 参与运算的第二个数据信号 |
| XALUOp | [2:0] | I | 运算方式的选择信号 |
| START |  | I | 乘除法计算的开始信号 |
| BUSY |  | O | 正在运算乘除法的判断信号 |
| HIGH | [31:0] | O | HI寄存器的值 |
| LOW | [31:0] | O | LO寄存器的值 |

模块功能说明如下：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名 | 描述 |
| 1 | 同步复位 | 时钟上升沿到来时，若reset信号为1，则复位HI和LO至0x00000000，复位BUSY至0 |
| 2 | 符号乘法 | 时钟上升沿到来时，若START信号为1且XALUOp==000，则计算A\*B并在5个时钟周期后通过HIGH和LOW输出 |
| 3 | 无符号乘法 | 时钟上升沿到来时，若START信号为1且XALUOp==001，则计算无符号A\*B并在5个时钟周期后通过HIGH和LOW输出 |
| 4 | 符号除法 | 时钟上升沿到来时，若START信号为1且XALUOp==010，则计算A/B并在10个时钟周期后通过HIGH和LOW输出 |
| 5 | 无符号除法 | 时钟上升沿到来时，若START信号为1且XALUOp==011，则计算无符号A/B并在10个时钟周期后通过HIGH和LOW输出 |
| 6 | 写HI寄存器 | 时钟上升沿到来时，若XALUOp==100，则将XALUA值写入HI |
| 8 | 写LO寄存器 | 时钟上升沿到来时，若XALUOp==100，则将XALUA值写入LO |

1. DM：数据存储器（M级部件）

模块端口定义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位数 | 方向 | 描述 |
| clk |  | I | 内置时钟信号 |
| reset |  | I | 同步复位信号 |
| PC | [31:0] | I | 当前指令的地址信号 |
| DMA | [31:0] | I | 存取的地址信号 |
| DMD | [31:0] | I | 存取的数据信号 |
| DMWE |  | I | 存数据使能信号 |
| DMOp | [1:0] | I | 存数据方式的选择信号 |
| IR\_M | [31:0] | I | M级指令信号 |
| DM | [31:0] | O | 取出的数据信号 |
| exccode\_DM | [4:0] | O | DM部件的异常编码 |

模块功能说明如下：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名 | 描述 |
| 1 | 存数据 | 时钟上升沿到来时，若DMWE信号为1，则根据DMOp的值向数据存储器DMA对应地址的某一段中写入DMD数据并输出一条与PC有关的信息 |
| 2 | 取数据 | 将数据存储器DMA地址对应的数据通过DM端口输出 |
| 3 | 同步复位 | 时钟上升沿到来时，若reset信号为1，则复位数据存储器至初始状态0x00000000 |
| 4 | 输出异常编码 | 数据地址不对齐或超出范围时输出相应异常编码 |

1. DMEXT：数据存储器拓展器（M级部件）

模块端口定义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位数 | 方向 | 描述 |
| DM | [31:0] | I | 所需要拓展的数据信号 |
| DMA | [31:0] | I | 读取数据存储器的地址信号 |
| DMEXTOp | [2:0] | I | 拓展方式选择信号 |
| DMEXT | [31:0] | O | 拓展后的数据信号 |

模块功能说明如下：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名 | 描述 |
| 1 | 拓展数据 | 根据DMA末两位值与DMEXTOp值对DM数据进行拓展后通过DMEXT端口输出 |

1. CP0：协处理器（M级部件）（在下文中断异常处理一节中叙述）
2. 流水线寄存器规格
3. FDreg：F/D级流水线寄存器

模块端口定义如下：

|  |  |  |
| --- | --- | --- |
| 控制信号 | 输入信号 | 输出信号 |
| clk | PC | IR\_D |
| reset | IM | PC\_D |
| en | exccode\_PC | PC4\_D |
|  |  | PC8\_D |
|  |  | exccode\_D |

1. DEreg：D/E级流水线寄存器

模块端口定义如下：

|  |  |  |
| --- | --- | --- |
| 控制信号 | 输入信号 | 输出信号 |
| clk | IR\_D | IR\_E |
| reset | PC\_D | PC\_E |
| en | PC4\_D | PC4\_E |
|  | PC8\_D | PC8\_E |
|  | RD1 | RD1\_E |
|  | RD2 | RD2\_E |
|  | EXT | EXT\_E |
|  | GRFWE | GRFWE\_E |
|  | Tnew | Tnew\_E |
|  | exccode\_D | exccode\_E |
|  | exccode\_controller |  |

1. EMreg：E/M级流水线寄存器

模块端口定义如下：

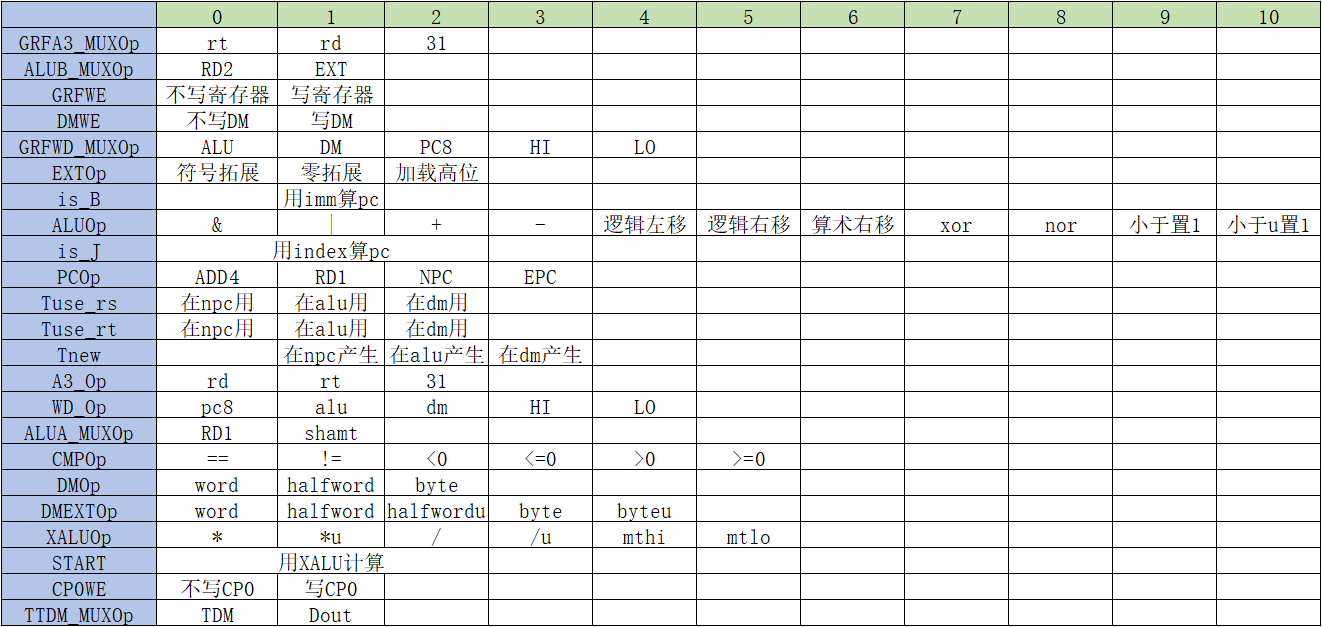
|  |  |  |
| --- | --- | --- |
| 控制信号 | 输入信号 | 输出信号 |
| clk | IR\_E | IR\_M |
| reset | PC\_E | PC\_M |
| en | PC4\_E | PC4\_M |
|  | PC8\_E | PC8\_M |
|  | ALU | ALU\_M |
|  | RD2\_E | RD2\_M |
|  | GRFWE\_E | GRFWE\_M |
|  | Tnew\_E | Tnew\_M |
|  | exccode\_E | exccode\_M |
|  | exccode\_ALU |  |

1. MWreg：M/W级流水线寄存器

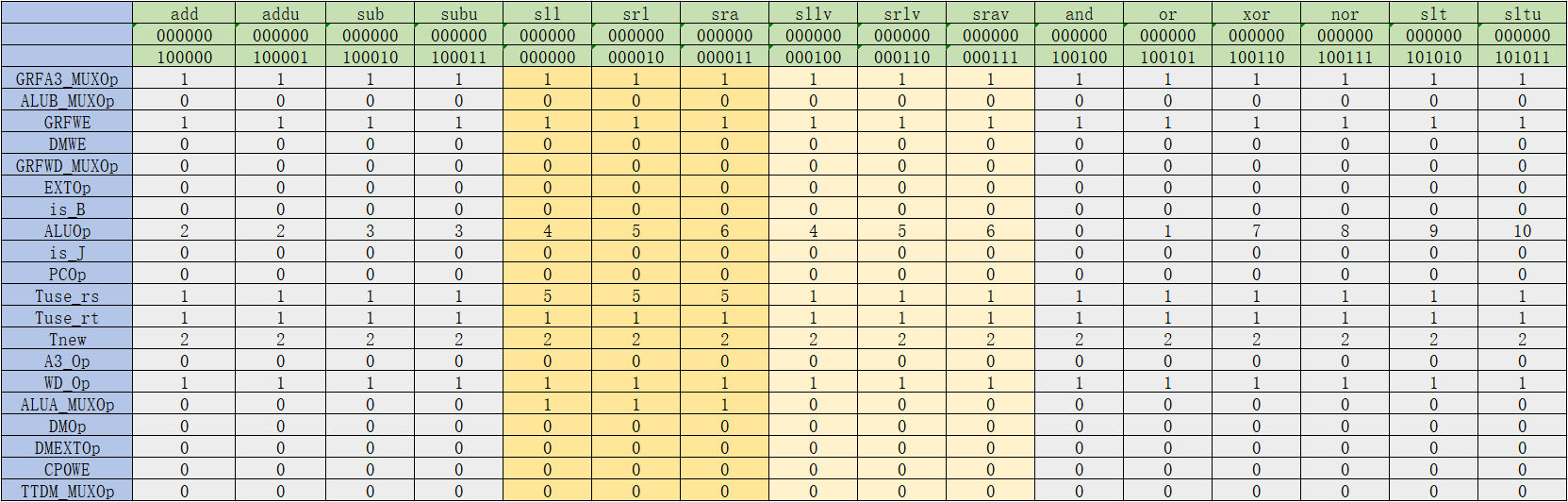
模块端口定义如下：

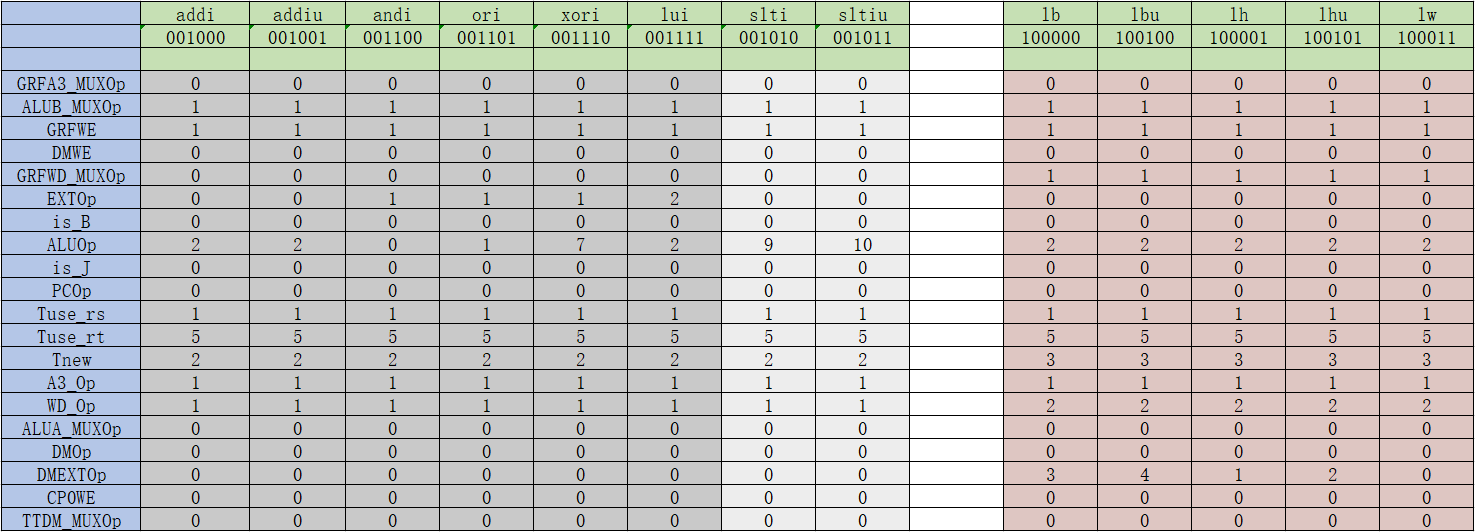
|  |  |  |
| --- | --- | --- |
| 控制信号 | 输入信号 | 输出信号 |
| clk | IR\_M | IR\_W |
| reset | PC\_M | PC\_W |
| en | PC4\_M | PC4\_W |
|  | PC8\_M | PC8\_W |
|  | ALU\_M | ALU\_W |
|  | DM | DM\_W |
|  | GRFWE\_M | GRFWE\_W |

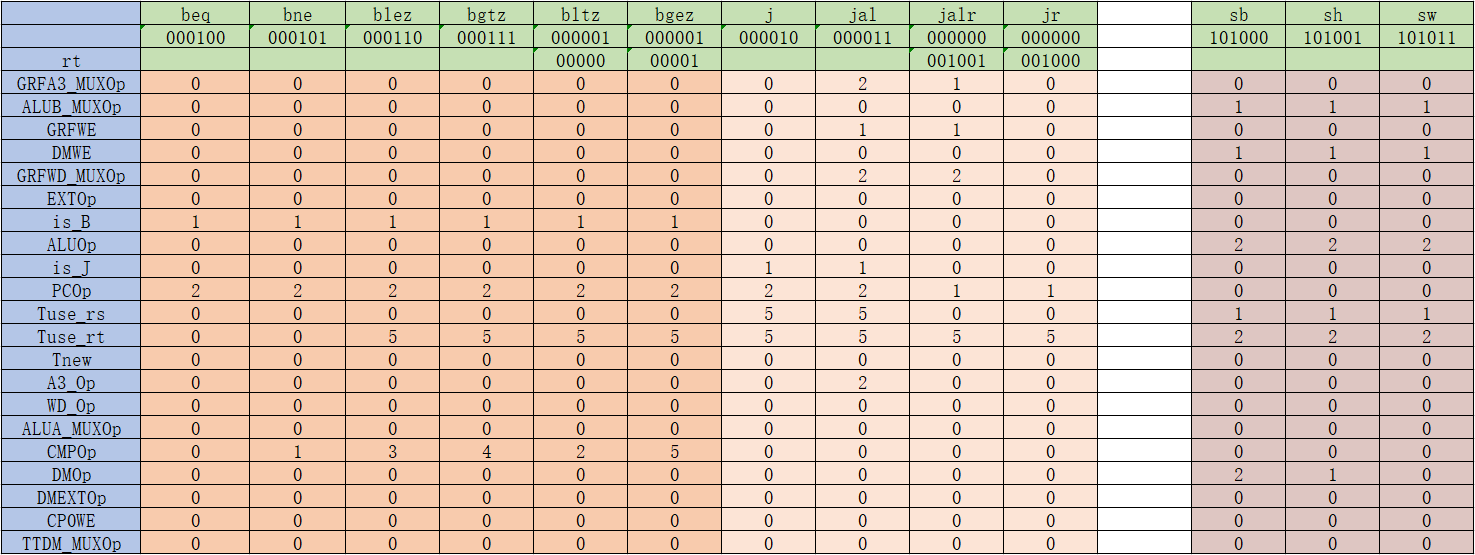
1. 控制器设计
2. 控制信号说明

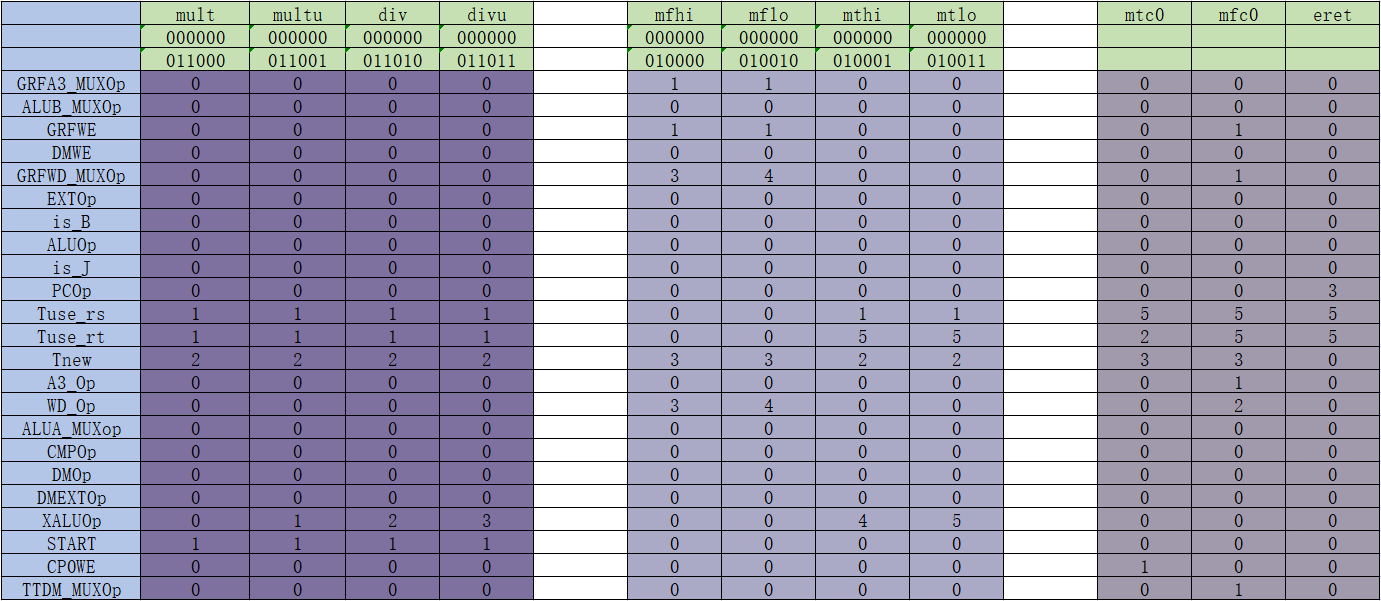


1. 指令与控制信号真值表









1. 冒险处理策略

在P6中共有50条指令，仍使用P5中所采取的分类表格法将十分繁杂且易于出错，因此，本设计采取暴力转发的方法。其主要实现方式为，在每一级流水线寄存器中都存入当前指令的Tnew（距离产生结果所剩的时钟周期数）以及A3（所将要写入的5位寄存器地址）和WD（所将要写入的数据），同时D级控制器产生D级指令的Tuse\_rs（距需要使用RD1所剩的时钟周期数）和Tuse\_rt（距需要使用RD2所剩的时钟周期数）。每一次时钟上升沿到来时，若检测到D级指令需要用到的RD1或RD2仍未产生，则产生一次暂停，若检测到其他已产生结果的数据冒险，则进行转发。这种设计保证了每一条指令所使用的值均是正确的。此外，乘除法的暂停也由BUSY信号决定。

在P7中新加入了eret指令，为了防止其后的非nop指令生效，也需要暂停。

有关模块：

1. Analysis\_MUX（A3及WD分析模块）

功能：根据每一级控制器产生的A3\_MUX和WD\_MUX信号产生每一级指令的A3与WD

1. StopUnit（暂停分析与产生模块）

功能：检测到D级指令所需数据不能及时得出时进行一次暂停，D级为乘除法相关指令且XALU忙碌时进行一次暂停

1. ForwardUnit（转发来源分析与产生模块）

功能：在D级rs、D级rt、E级rs、E级rt、M级rt五个位点检测到数据冒险时根据各个流水线寄存器的WD和A3值进行转发

1. 桥与IO设计

桥的端口定义如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位数 | 方向 | 描述 |
| PrAddr | [31:0] | I | CPU发出的数据地址信号 |
| PrWD | [31:0] | I | CPU发出的写入数据信号 |
| PrWE |  | I | CPU发出的数据写使能信号 |
| DEV0RD | [31:0] | I | Timer0发出的读出数据信号 |
| DEV1RD | [31:0] | I | Timer1发出的读出数据信号 |
| PrRD | [31:0] | O | 外设总读出数据信号 |
| DEVAddr | [31:0] | O | 外设数据地址信号 |
| DEVWD | [31:0] | O | 外设写入数据信号 |
| DEV0WE |  | O | Timer0写使能信号 |
| DEV1WE |  | O | Timer0写使能信号 |

桥的功能说明如下：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名 | 描述 |
| 1 | 译码 | 将CPU发出的地址、写使能、数据信号转换成相应外设的地址、写使能、数据信号，将外设发出的读数据信号转换成总读数据信号 |

1. CP0设计

在本设计中，CP0实现的内部寄存器有12号SR读写寄存器（im[15:10], exl, ie）、13号Cause只读寄存器（bd, hwint\_pend[15:10], exccode[6:2]）、14号EPC读写寄存器、15号PRId只读寄存器。

注：在本设计中，CP0处于M级。

CP0端口定义如下：

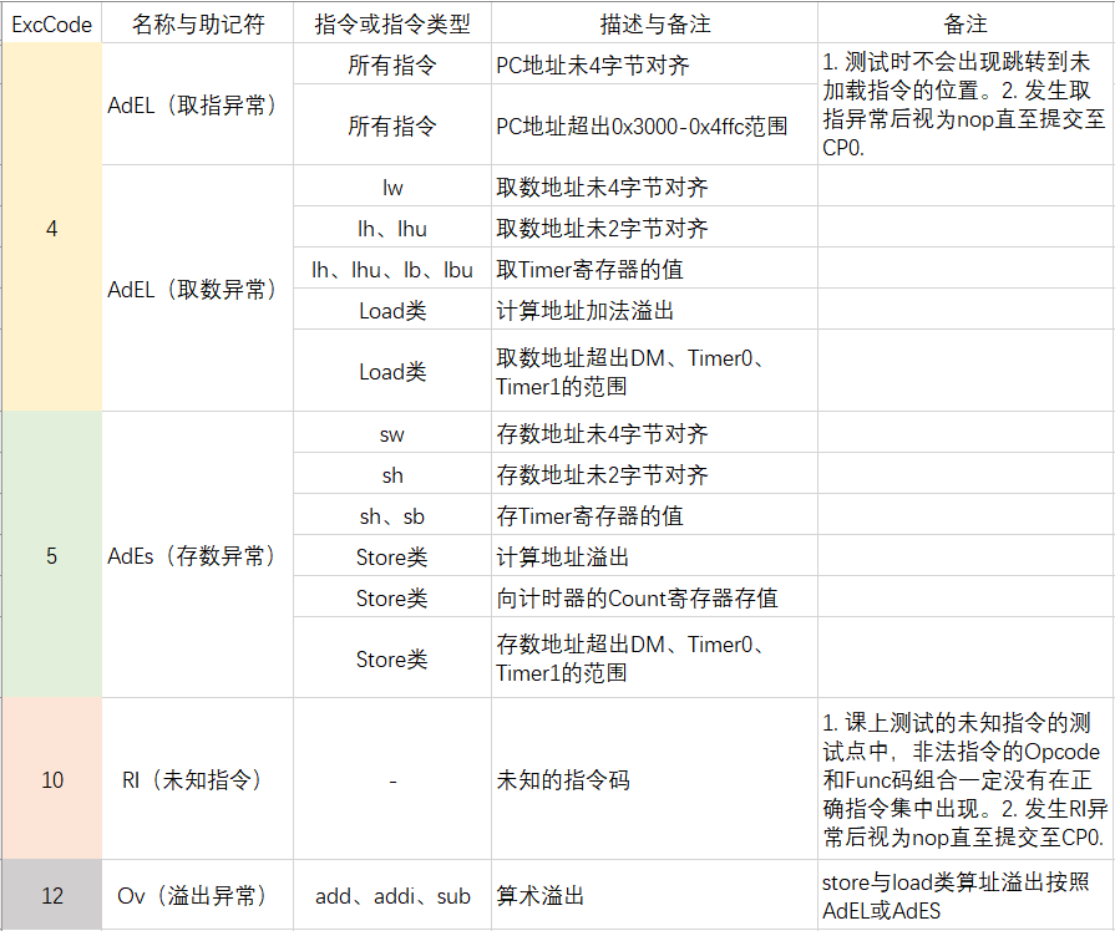
|  |  |  |  |
| --- | --- | --- | --- |
| 信号名 | 位数 | 方向 | 描述 |
| A1 | [4:0] | I | 读CP0寄存器编号 |
| A2 | [4:0] | I | 写CP0寄存器编号 |
| Din | [31:0] | I | 写入数据信号 |
| PC | [31:2] | I | 指令地址信号 |
| ExcCode | [6:2] | I | 当前指令异常类型信号 |
| HWInt | [5:0] | I | 外部中断信号 |
| WE |  | I | 写使能信号 |
| EXLSet |  | I | 用于置位SR的EXL位 |
| ExlClr |  | I | 用于清除SR的EXL位 |
| clk |  | I | 内置时钟信号 |
| reset |  | I | 同步复位信号 |
| IR\_W | [31:0] | I | 流水线W级指令信号 |
| IntReq |  | O | 中断请求信号 |
| EPC | [31:2] | O | 异常处理结束后返回的指令地址信号 |
| Dout | [31:0] | O | 读出数据信号 |

CP0功能说明如下：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名 | 描述 |
| 1 | 写数据 | WE有效时向A2寄存器中写入Din值 |
| 2 | 读数据 | 将A1寄存器值通过Dout端口输出 |
| 3 | 同步复位 | reset信号有效时复位所有寄存器至0x00000000 |
| 4 | 锁存外部中断 | 将HWInt信号不断记录至hwint\_pend位 |
| 5 | 进入内核态 | 出现外部中断或M级指令异常时，进入内核态置exl为1，记录EPC、bd、exccode |
| 6 | 从内核态返回 | EXLClr有效时从内核态返回 |

1. 异常与中断设计
2. 异常行为规范

本设计中，对所有异常都遵循精确异常的处理规则：即受害指令的前序指令的期望执行结果都应执行，并且受害指令及其后序指令的执行效果不影响异常处理程序返回后所执行指令的执行效果，具体异常分类与exccode编码如下：



注：分支跳转指令无论跳转与否，延迟槽指令为受害指令时BD均需要置位。

如果jr跳转到一个不对齐的地址，那么EPC的值应该为该地址，而非jr指令的PC值。

1. 异常判断方式

在本设计中，PC、D级controller、ALU、DM部件会判断当前指令的异常类型，记录其exccode（无异常则即为0）并随流水线向下传递，传递时流水线寄存器优先取前一级流水线寄存器记录的exccode值，若为0则取该级功能部件发出的exccode值，并在M级交由CP0处理。

1. 中断行为规范

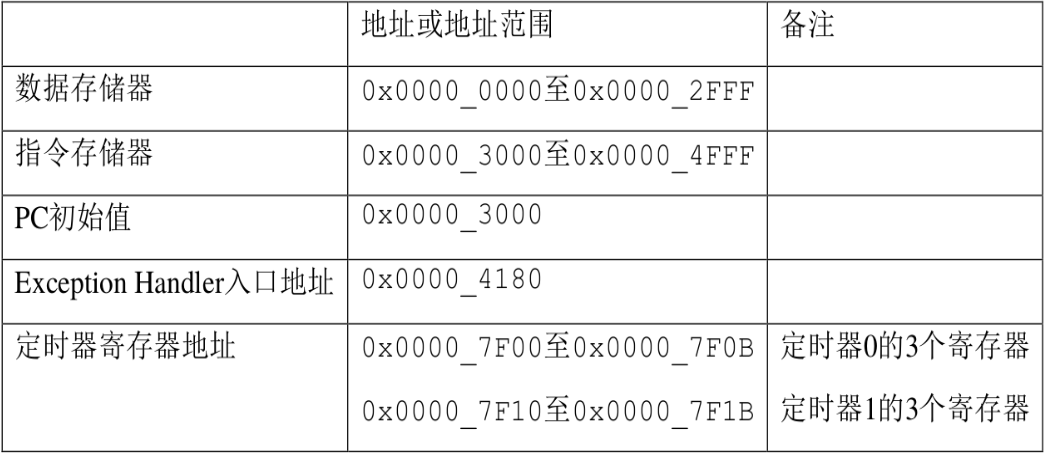
本实验不要求在中断发生时指定具体受害指令，但需要保证中断的处理是精确的——内核态前后每条指令的执行效果不变。CPU共需接收来自两个Timer模拟的外部中断和一个来自mips模块外的外部中断信号。

1. 异常与中断处理

在M级CP0接收到外部中断或内部M级指令异常后，经内部逻辑处理，记录各类异常中断信息（exccode、bd、hwint\_pend、epc）并发出IntReq信号，清除仍在流水线中的指令并进入内核态，屏蔽外部中断。当M级指令为eret时从内核态返回。

1. 其他

地址范围声明如下：





1. 测试程序

命令行导出：

java -jar E:\Mars4\_5.jar a db mc CompactDataAtZero dump 0x00004180-0x00004ffc HexText E:\code\_handler.txt E:\source.asm

测试主要从以下四方面入手：CPU，异常，中断，IO

1. CPU测试主要由P6的自动化测试完成，并手动构造关于CP0的三条指令的样例
2. 异常测试主要是手动构造遍历各种异常情况

例如下：

.ktext 0x4180

\_entry:

mfc0 $k0, $14

mfc0 $k1, $13

ori $k0, $0, 0x1000

sw $sp, -4($k0)

addiu $k0, $k0, -256

move $sp, $k0

j \_save\_context

nop

\_main\_handler:

mfc0 $k0, $13

ori $k1, $0, 0x007c

and $k0, $k1, $k0

beq $0, $k0, \_restore\_context

nop

mfc0 $k0, $14

addu $k0, $k0, 4

mtc0 $k0, $14

j \_restore\_context

nop

\_restore:

eret

\_save\_context:

sw $1, 4($sp)

sw $2, 8($sp)

sw $3, 12($sp)

sw $4, 16($sp)

sw $5, 20($sp)

sw $6, 24($sp)

sw $7, 28($sp)

sw $8, 32($sp)

sw $9, 36($sp)

sw $10, 40($sp)

sw $11, 44($sp)

sw $12, 48($sp)

sw $13, 52($sp)

sw $14, 56($sp)

sw $15, 60($sp)

sw $16, 64($sp)

sw $17, 68($sp)

sw $18, 72($sp)

sw $19, 76($sp)

sw $20, 80($sp)

sw $21, 84($sp)

sw $22, 88($sp)

sw $23, 92($sp)

sw $24, 96($sp)

sw $25, 100($sp)

sw $26, 104($sp)

sw $27, 108($sp)

sw $28, 112($sp)

sw $29, 116($sp)

sw $30, 120($sp)

sw $31, 124($sp)

mfhi $k0

sw $k0, 128($sp)

mflo $k0

sw $k0, 132($sp)

j \_main\_handler

nop

\_restore\_context:

lw $1, 4($sp)

lw $2, 8($sp)

lw $3, 12($sp)

lw $4, 16($sp)

lw $5, 20($sp)

lw $6, 24($sp)

lw $7, 28($sp)

lw $8, 32($sp)

lw $9, 36($sp)

lw $10, 40($sp)

lw $11, 44($sp)

lw $12, 48($sp)

lw $13, 52($sp)

lw $14, 56($sp)

lw $15, 60($sp)

lw $16, 64($sp)

lw $17, 68($sp)

lw $18, 72($sp)

lw $19, 76($sp)

lw $20, 80($sp)

lw $21, 84($sp)

lw $22, 88($sp)

lw $23, 92($sp)

lw $24, 96($sp)

lw $25, 100($sp)

lw $26, 104($sp)

lw $27, 108($sp)

lw $28, 112($sp)

lw $29, 116($sp)

lw $30, 120($sp)

lw $31, 124($sp)

lw $k0, 128($sp)

mthi $k0

lw $k0, 132($sp)

mtlo $k0

j \_restore

nop

.text

ori $2, $0, 0x1001

mtc0 $2, $12

ori $28, $0, 0x0000

ori $29, $0, 0x0000

lui $8, 0x7fff

lui $9, 0x7fff

add $10, $8, $9

or $10, $8, $9

end:

beq $0, $0, end

nop

1. 中断测试使用P6自动生成的text代码并在TB中随机加入中断信号

TB例如下：

module mips\_txt;

// Inputs

reg clk;

reg reset;

reg interrupt;

// Outputs

wire [31:0] addr;

// Instantiate the Unit Under Test (UUT)

mips uut (

.clk(clk),

.reset(reset),

.addr(addr),

.interrupt(interrupt)

);

parameter exception\_pc = 32'h00003018;

integer exception\_count;

integer interrupt\_counter;

integer needInterrupt;

initial begin

exception\_count = 0;

interrupt = 0;

needInterrupt = 0;

interrupt\_counter = 0;

// Initialize Inputs

clk = 0;

reset = 1;

#20 reset = 0;

// Wait 100 ns for global reset to finish

// Add stimulus here

end

always #2 clk = ~clk;

always @(negedge clk) begin

if (reset) begin

interrupt\_counter = 0;

needInterrupt = 0;

interrupt = 0;

end else begin

if (interrupt) begin

if (interrupt\_counter == 0) begin

interrupt = 0;

end else begin

interrupt\_counter = interrupt\_counter - 1;

end

end else if (needInterrupt) begin

needInterrupt = 0;

interrupt = 1;

interrupt\_counter = 5;

end else begin

case (addr)

exception\_pc:

begin

if (exception\_count == 0) begin

exception\_count = 1;

interrupt = 1;

interrupt\_counter = 5;

end

end

endcase

end

end

end

endmodule

1. IO构造少许样例即可

思考题

1. 我们计组课程一本参考书目标题中有“硬件/软件接口”字样，那么到底什么是“硬件/软件接口”？

硬件/软件接口就是计算机中硬件与软件的交互手段。硬件接口既包括物理上的接口，也包括逻辑上的数据传送协议，软件接口通常是驱动程序。

1. 在我们设计的流水线中，DM处于CPU内部，请你考虑现代计算机中它的位置应该在何处。

DM应该是与CPU直接相连的cache，CPU通过cache与主存交换数据。

1. BE部件对所有的外设都是必要的吗？

不全是必要的，某些外设（如本实验中的Timer）只支持按字读写，不支持按半字或字节读写，因此BE部件不必要。

1. 请阅读官方提供的定时器源代码，阐述两种中断模式的异同，并分别针对每一种模式绘制状态转移图。

见《计时器说明文档》。

1. 请开发一个主程序以及定时器的exception handler。整个系统完成如下功能：
   1. 定时器在主程序中被初始化为模式0；
   2. 定时器倒计时数至0产生中断；
   3. handler设置使能Enable为1从而再次启动定时器的计数器，2及3被无限重复；
   4. 主程序在初始化时将定时器初始化为模式0，设定初值寄存器的初值为某个值，如100或1000。（注意，主程序可能需要涉及对CP0.SR的编程，推荐阅读过后文后再进行）

.ktext 0x4180

ori $t9,0x7f00

ori $t8,0x0009

sw $t8,0($t9) ## ctrl[3:0] = 4'b1001

eret

.text

ori $t1,0x7f00

ori $t2,0x0009

sw $t2,0($t1) ## ctrl[3:0] = 4'b1001

ori $t3,100

sw $t3,4($t1) ## preset = 100

1. 请查阅相关资料，说明鼠标和键盘的输入信号是如何被CPU知晓的？

外设包括设备主体和接口控制器两部分，设备主体通过接口控制器与主机连接。鼠标和键盘的输入信号相当于中断，当鼠标和键盘产生输入信号时，会产生一个中断然后中断例程会从端口读入数据到寄存器，CPU接收到中断后进入中断处理程序获取鼠标和键盘的输入信息。