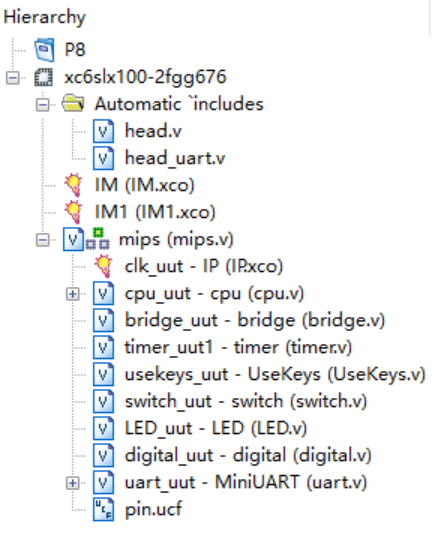
**P8 实验报告**

**第一部分 mips微系统模型架构**

1. **Mips微系统架构视图**



1. **流水线CPU数据通路和控制器**
2. **各指令对应的数据通路表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 指令 | | addu | subu | ori |
| PC | | pc->IM | pc->IM | pc->IM |
| IM | | IM->INS | IM->INS | IM->INS |
| ADD8(4) | | PC+8->PC8 | PC+8->PC8 | PC+8->PC8 |
| D级 | INS | new | new | new |
| PC8 | new | new | new |
| decode | | rd,rs,rt | rd,rs,rt | rt,rs,imm16 |
| EXT | |  |  | ZE(imm16) |
| NPC | |  |  |  |
| GRF | | RF[rs]->RS\_OUT RF[rt]->RT\_OUT | RF[rs]->RS\_OUT RF[rt]->RT\_OUT | RF[rs]->RS\_OUT RF[rt]->RT\_OUT |
| compare | |  |  |  |
| E级 | WB | new(rd) | new(rd) | new(rt) |
| EXT\_OUT |  |  | new(EXT) |
| RS\_OUT | new(GRF) | new(GRF) | new(GRF) |
| RT\_OUT | new(GRF) | new(GRF) | new(GRF) |
| PC8 | old | old | old |
| ALU | | RS\_OUT+RT\_OUT unsigned | RS\_OUT-RT\_OUT unsigned | RS\_OUT|EXT\_OUT |
| M级 | WB | old | old | old |
| ALU\_result | new(ALU) | new(ALU) | new(ALU) |
| DIN |  |  |  |
| DM | |  |  |  |
| W级 | WrData | old(ALU\_result) | old(ALU\_result) | old(ALU\_result) |
| WB | old | old | old |
| GRF | | WrData->RF[WB] | WrData->RF[WB] | WrData->RF[WB] |

|  |  |  |  |
| --- | --- | --- | --- |
| lui | lw | sw | beq |
| pc->IM | pc->IM | pc->IM | pc->IM |
| IM->INS | IM->INS | IM->INS | IM->INS |
| PC+8->PC8 | PC+8->PC8 | PC+8->PC8 | PC+8->PC8 |
| new | new | new | new |
| new | new | new | new |
| rt,imm16 | rt,rs,imm16 | rt,rs,imm16 | rt,rs,imm16 |
| LUI(imm16) | SE(imm16) | SE(imm16) | SE(imm16) |
|  |  |  | B\_type PC8+SE |
| RF[rs]->RS\_OUT (32'b0,rs=00000) | RF[rs]->RS\_OUT | RF[rs]->RS\_OUT RF[rt]->RT\_OUT |  |
|  |  |  | RF[rs]==RF[rt] |
| new(rt) | new(rt) |  |  |
| new(EXT) | new(EXT) | new(EXT) |  |
| new(GRF) | new(GRF) | new(GRF) |  |
|  |  | new(GRF) |  |
| old | old | old | new(npc) |
| RS\_OUT|EXT\_OUT | RS\_OUT+RT\_OUT unsigned | RS\_OUT+RT\_OUT unsigned |  |
| old | old |  |  |
| new(ALU) | new(ALU) | new(ALU) |  |
|  |  | old(RT\_OUT) |  |
|  | read:DM[ALU\_re] | DIN->DM[ALU\_re] |  |
| old(ALU\_result) | new(DM) |  |  |
| old | old |  |  |
| WrData->RF[WB] | WrData->RF[WB] |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| j | jal | jr | jalr |
| pc->IM | pc->IM | pc->IM | pc->IM |
| IM->INS | IM->INS | IM->INS | IM->INS |
| PC+8->PC8 | PC+8->PC8 | PC+8->PC8 | PC+8->PC8 |
| new | new | new | new |
| new | new | new | new |
| imm26 | imm26 | rs | rd,rs |
|  |  |  |  |
| J\_type PC[31:28]|imm26|00 | J\_type PC[31:28]|imm26|00 | JR\_type RF[rs] | JR\_type RF[rs] |
|  | PC8->RF[31] | RF[rs]->RS\_OUT | RF[rs]->RS\_OUT PC8->RF[rd] |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
| new(npc) | new(npc) | new(npc) | new(npc) |

1. **CPU各模块规格**
2. **IFU**
3. 端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 接口名 | 方向 | 位数 | 端口描述 |
| clk | IN | 1 | 时钟信号 |
| Reset | IN | 1 | 同步复位信号 |
| if\_jump | IN | 1 | pc跳转执行信号 |
| stall\_pc | IN | 1 | pc暂停信号 |
| next\_pc | IN | 32 | pc跳转地址 |
| instr | OUT | 32 | 从IM中读取到的指令 |
| PC | OUT | 32 | PC当前值 |
| PC4 | OUT | 32 | PC+4值 |
| PC8 | OUT | 32 | PC+8值 |

1. 模块功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 具体描述 |
| 1 | 同步复位 | 复位信号为1时，在时钟上升沿复位pc |
| 2 | pc顺序执行 | pc执行pc+4 |
| 3 | pc跳转执行 | pc跳转至next\_pc地址 |
| 4 | pc暂停执行 | pc保持当前值不变，即暂停 |
| 5 | 读取指令 | 根据pc所指地址从IM读取响应指令并输出 |

1. **GRF**
2. 端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 端口名 | 方向 | 位数 | 描述 |
| clk | IN | 1 | 时钟信号 |
| WrData | IN | 32 | 回写32位数据 |
| RWAddr | IN | 5 | 回写寄存器地址 |
| rs | IN | 5 | 读寄存器地址1 |
| rt | IN | 5 | 读寄存器地址2 |
| RegWr | IN | 1 | 寄存器堆写使能信号 |
| Reset | IN | 1 | 复位信号 |
| RS\_OUT | OUT | 32 | RS对应输出32位数据 |
| RT\_OUT | OUT | 32 | RT对应输出32位数据 |

（2）模块功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 功能描述 |
| 1 | 同步复位 | 当Reset信号有效时同步复位所有寄存器 |
| 2 | 读寄存器 | 读出rs, rt对应寄存器内存储的数据 |
| 3 | 写寄存器 | 将WrData输入的数据写入RW对应的寄存器 |

1. **ALU**
2. 端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 端口名 | 方向 | 位数 | 描述 |
| ALUOp | IN | 4 | ALU控制信号，控制进行何种运算 |
| inputA | IN | 32 | ALU的第一个操作数 |
| inputB | IN | 32 | ALU的第二个操作数 |
| shift | IN | 5 | 移位操作移位位数 |
| SHIFTV | IN | 1 | 可变移位选择信号 |
| ALU\_result | OUT | 32 | ALU计算结果 |

（2）模块功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 功能描述 |
| 1 | 加 | 将两个操作数相加 |
| 2 | 减 | 将两个操作数相减 |
| 3 | 或 | 将两个操作数按位或 |
| 4 | 逻辑左移 | 将操作数B逻辑左移shift位 |
| 5 | 逻辑右移 | 将操作数B逻辑左移shift位 |
| 6 | 与 | 将两个操作数按位与 |
| 7 | 或非 | 将两个操作数按位或再取反 |
| 8 | 异或 | 将两个操作数按位异或 |
| 9 | 有符号比较大小 | 对两个操作数进行有符号比较 |
| 10 | 无符号比较大小 | 对两个操作数进行无符号比较 |
| 11 | 算术右移 | 将操作数B算术右移shift位 |

1. **DM**
2. 端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 端口名 | 方向 | 位数 | 描述 |
| clk | IN | 1 | 时钟信号 |
| DIN | IN | 32 | 写入32位数据 |
| DMAddr | IN | 32 | 写入32位地址 |
| DMWr | IN | 1 | DM写使能信号 |
| Reset | IN | 1 | 同步复位信号 |
| L\_S\_SL | IN | 3 | load/store类型选择信号 |
| DOUT | OUT | 32 | 读出32位数据 |
| LoadSel | OUT | 2 | Load类型地址决定值 |

1. 模块功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 功能描述 |
| 1 | 读内存数据 | DMWr信号无效时，读出DMAddr地址存储的数据 |
| 2 | 写内存数据 | DMWr信号有效时，向DMAddr地址写入DIN数据 |
| 3 | 同步复位 | 当Reset信号有效时，同步复位DM |

1. **EXT**
2. 端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 端口名 | 方向 | 位数 | 描述 |
| EXT\_Op | IN | 2 | 扩展类型选择信号 |
| imm16 | IN | 16 | 要执行扩展操作的16位立即数 |
| EXT\_OUT | OUT | 32 | 扩展结果 |

1. 模块功能

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 功能描述 |
| 1 | 符号扩展 | EXT\_Op信号为01时，对16位立即数进行符号扩展 |
| 2 | 无符号（零）扩展 | EXT\_Op信号为00时，对16位立即数进行零扩展 |
| 3 | 加载立即数到高位 | EXT\_Op信号为11时，将16位立即数加载到高位 |

1. **CMP**
2. 端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 接口名 | 方向 | 位数 | 端口描述 |
| RS\_OUT | IN | 32 | RS寄存器值 |
| RT\_OUT | IN | 32 | RT寄存器值 |
| opcode | IN | 6 | 比较指令解码1 |
| rt | IN | 5 | 比较指令解码2 |
| judge | OUT | 1 | 判断结果信号 |

1. 功能描述

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 具体描述 |
| 1 | 相等时跳转 | Rs和RT相等时判断信号为1 |
| 2 | 不等时跳转 | Rs和RT不等时判断信号为1 |
| 3 | 小于0时跳转 | Rs小于0时判断为1 |
| 4 | 小于等于0时跳转 | Rs小于等于0时判断为1 |
| 5 | 大于0时跳转 | Rs大于0时判断为1 |
| 6 | 大于等于0时跳转 | Rs大于等于0时判断为1 |

1. **NPC**
2. 端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 接口名 | 方向 | 位数 | 端口描述 |
| judge | IN | 1 | 比较判断信号 |
| NPC\_SL | IN | 3 | NPC跳转模式选择信号 |
| IMM16 | IN | 16 | 16位立即数 |
| IMM26 | IN | 16 | 26位立即数 |
| RS\_OUT | IN | 32 | RS寄存器值 |
| PC\_4 | IN | 32 | PC+4的值 |
| next\_pc | OUT | 32 | 跳转pc地址 |
| if\_jump | OUT | 1 | 跳转判断信号 |

1. 功能描述

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 具体描述 |
| 1 | B型跳转 | 当比较判断信号为1，NPC\_SL为B型跳转时，相对跳转 |
| 2 | J型跳转 | NPC\_SL为J型时跳转到26位立即数所指地址 |
| 3 | JR跳转 | NPC\_SL为JR型时跳转到RS寄存器所指地址 |

1. **LoadExt**
2. 端口定义

|  |  |  |  |
| --- | --- | --- | --- |
| 接口名 | 方向 | 位数 | 端口描述 |
| mem\_out | IN | 32 | DM读出数据 |
| L\_S\_SL | IN | 3 | Load类型选择信号 |
| DMAddr | IN | 2 | 地址偏移信号 |
| DM\_data | OUT | 32 | 写回数据 |

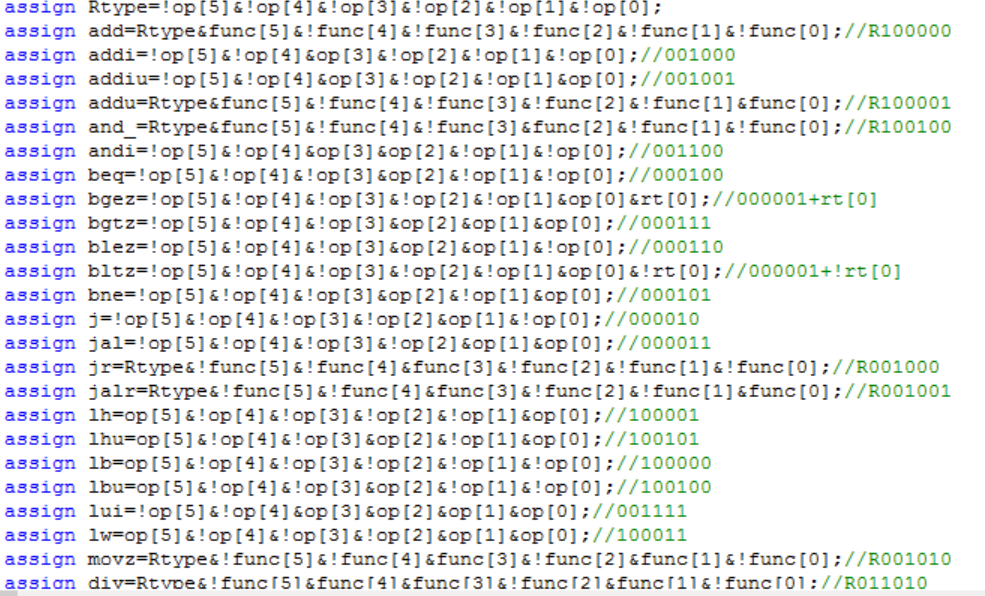
1. 功能描述

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 具体描述 |
| 1 | 按字读取 | 将DM读出数据整字读出 |
| 2 | 按半字读取 | 将DM读出数据取半字读出 |
| 3 | 按字节读取 | 将DM读出数据取字节读出 |

1. **CPU控制器设计**
2. **端口定义和功能描述**

|  |  |  |  |
| --- | --- | --- | --- |
| 端口名 | 方向 | 位数 | 描述 |
| opcode | IN | 6 | 指令高六位[31:26]，用于判断指令类型 |
| func | IN | 6 | 指令低六位[5:0]，用于判断R指令 |
| RegDst | OUT | 2 | 写入寄存器选择信号,00选择rd，01选择rt，10选择ra |
| NPC\_SL | OUT | 2 | PC跳转选择信号，00顺序执行，01为B型指令，10为J型指令，11为JR指令 |
| RegData | OUT | 2 | 寄存器写入数据选择信号，00为ALU，01为DM，10为PC\_4 |
| RegWr | OUT | 1 | 寄存器写使能信号 |
| ExtOp | OUT | 2 | 扩展类型选择信号，01是符号扩展，00是零扩展，10是加载立即数到高位 |
| ALUsrc | OUT | 1 | ALU输入端2选择信号，为1选择EXT数据，为0选择rt寄存器数据 |
| DMWr | OUT | 1 | DM模式选择信号，为1向DM写入数据，为0从DM读出数据 |
| ALUOp | OUT | 3 | ALU功能选择信号，ADD:000，SUB:001，OR:010，SLL:011 |
| L\_S\_SL | OUT | 3 | Load/Store类型选择信号 |
| newdatatype | OUT | 2 | 新产生数据来源信号，目前有三种来源ALU,DM,PC |
| rt\_tuse | OUT | 2 | D级指令还有几个周期使用rt寄存器 |
| rs\_tuse | OUT | 2 | D级指令还有几个周期使用rs寄存器 |
| M\_D\_Read | OUT | 1 | HI和LO寄存器读取选择信号 |
| M\_D\_Cal | OUT | 2 | 乘除计算信号 |
| is\_signed | OUT | 1 | 有符号计算信号 |
| SHIFTV | OUT | 1 | 可变移位信号 |

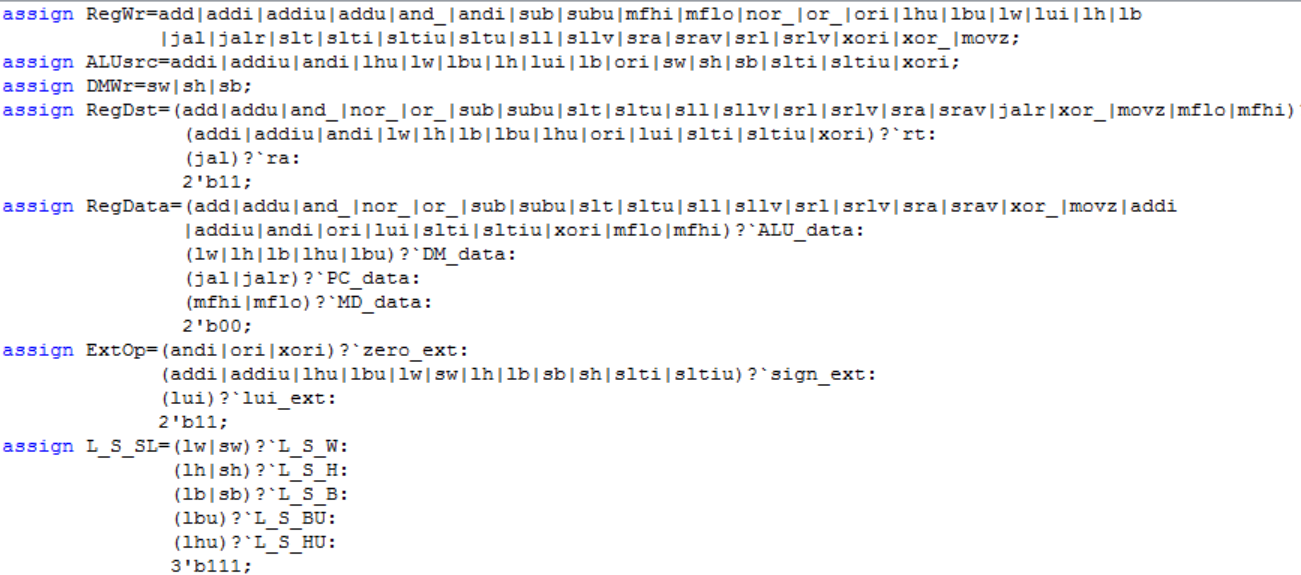
1. **控制器的指令解码（与逻辑AND）**

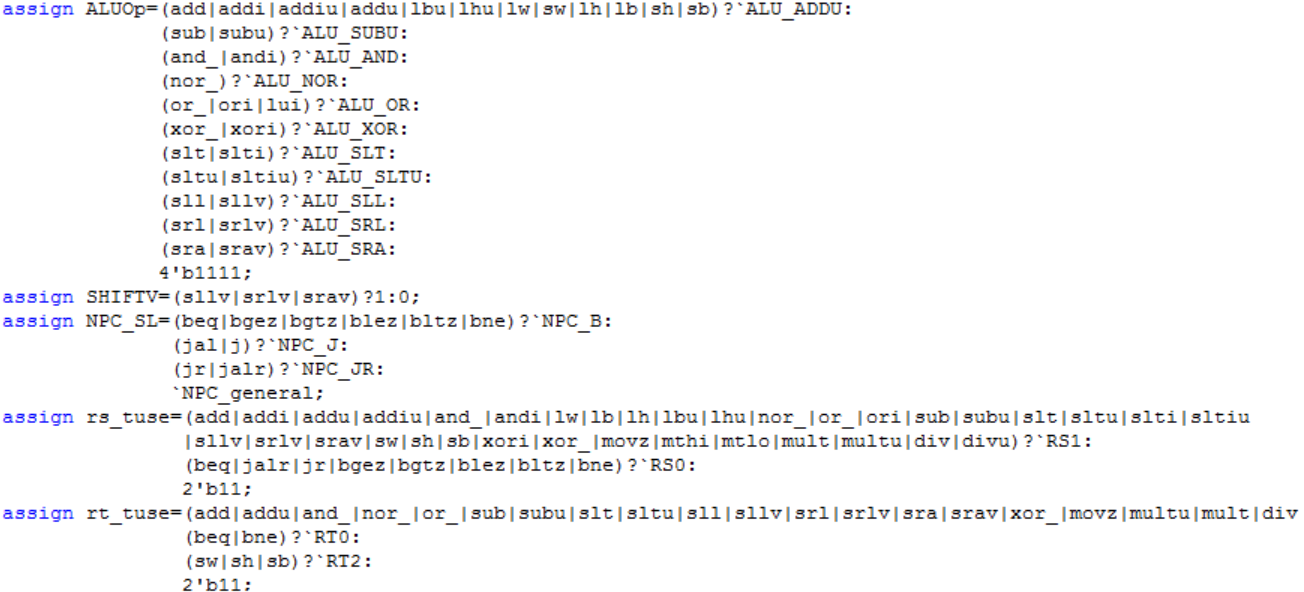


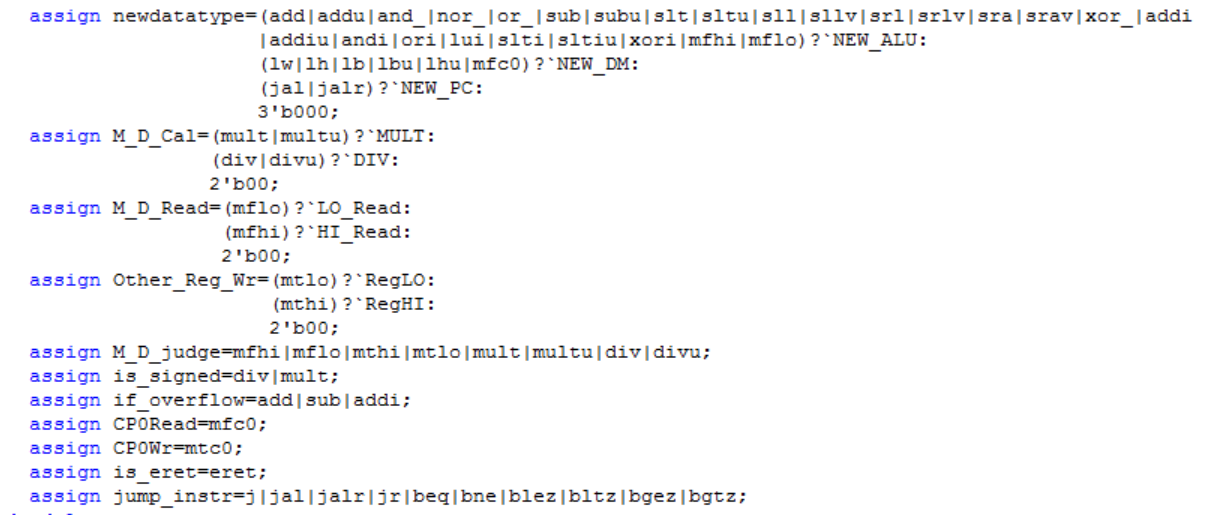
1. **控制器指令与控制信号真值表**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | addu | subu | ori | lw | sw | beq | lui | jal | jr | j |
| RegDst | 00 | 00 | 01 | 01 | X | X | 01 | 10 | X | X |
| NPC\_SL | 00 | 00 | 00 | 00 | 00 | 01 | 00 | 10 | 11 | 10 |
| RegData | 00 | 00 | 00 | 10 | X | X | 00 | 10 | X | X |
| RegWr | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| ExtOp | X | X | 00 | 01 | 01 | X | 00 | X | X | X |
| ALUsrc | 0 | 0 | 1 | 1 | 1 | 0 | 1 | X | X | X |
| DMWr | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| ALUOp | 000 | 001 | 010 | 000 | 000 | 001 | 010 | X | X | X |

**(4) 控制器的控制信号生成（或逻辑OR）**







1. **外部设备**

**1.Bridge**

1. 模块接口：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| PrAddr | I | PC传入地址 |
| PrWD | I | PC传入数据 |
| PrWe | I | PC写使能 |
| DEV\_RD1 | I | 设备1读入数据 |
| DEV\_RD2 | I | 设备2读入数据 |
| DEV\_RD3 | I | 设备3读入数据 |
| DEV\_RD4 | I | 设备4读入数据 |
| DEV\_RD5 | I | 设备5读入数据 |
| DEV\_RD6 | I | 设备6读入数据 |
| DEV\_WE1 | O | 设备1写使能 |
| DEV\_WE2 | O | 设备2写使能 |
| DEV\_WE4 | O | 设备4写使能 |
| DEV\_WE5 | O | 设备5写使能 |
| PrRD | O | PC读数据 |
| DEV\_ADDR | O | 设备写入地址 |
| DEV\_RD | O | 设备写入数据 |

（2）功能定义：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 地址匹配 | 根据addr\_in和所有设备的地址范围进行匹配译码，为每个设备产生一个使能信号。 |
| 2 | CPU读 | 根据地址匹配产生的使能信号，选择一个设备的数据输出。 |
| 3 | CPU写 | 根据地址匹配产生的使能信号和CPU送入的写使能信号，产生对应设备的写使能信号。 |

**2.Timer**

（1）模块接口：

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号。 |
| reset | I | 复位信号。 |
| PrAddr[3:2] | I | 地址输入。 |
| Wr\_en | I | 写使能。 |
| Data\_in[31:0] | I | 32位数据输入。 |
| Data\_out[31:0] | O | 32位数据输出。 |
| IRQ | O | 中断请求。 |

（2）功能定义：

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 计数 | 可以通过设置寄存器的初值来完成计数。 |
| 2 | 中断产生 | 在计数为0时产生中断请求（模式0和模式1两种中断） |

**3.UART**

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| CLK\_I | I | 时钟信号 |
| RST\_I | I | 复位信号 |
| DAT\_I | I | 写入数据 |
| WE\_I | I | 写使能 |
| STB\_I | I | SLAVE选通 |
| ADD\_I | I | 写入地址 |
| RxD | I | 串口接收数据 |
| DAT\_O | O | 读出数据 |
| TxD | O | 串口发送数据 |
| IRQ | O | UART中断信号 |

**4.SWITCH**

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号 |
| ReadAddr | I | 读地址 |
| dip\_switch0 | I | 0组开关 |
| dip\_switch1 | I | 1组开关 |
| dip\_switch2 | I | 2组开关 |
| dip\_switch3 | I | 3组开关 |
| dip\_switch4 | I | 4组开关 |
| dip\_switch5 | I | 5组开关 |
| dip\_switch6 | I | 6组开关 |
| dip\_switch7 | I | 7组开关 |
| data\_out | O | 数据输出 |

**5.LED**

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号 |
| enable | I | 写使能 |
| data\_in | I | 写入数据 |
| led\_light | O | 传输给LED外设的数据 |
| data\_cpu | O | 传输给CPU的数据 |

**6.DIGIT**

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号 |
| Addr | I | 写入地址 |
| data\_in | I | 写入数据 |
| enable | I | 写使能 |
| sel0 | O | 符号位选择信号 |
| sel1 | O | 高4位选择信号 |
| sel2 | O | 低4位选择信号 |
| code0 | O | 符号位编码值 |
| code1 | O | 高4位编码值 |
| code2 | O | 低4位编码值 |
| data\_cpu | O | 传回CPU数据 |

**7.USER\_KEY**

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号 |
| user\_key | I | 用户按键输入 |
| data\_out | O | 输出数据 |

1. **冒险单元（HAZARD）**
2. **Tuse-Tnew时间模型**
3. **Tuse**

我们定义某条指令位于流水线D级的时候，再经过Tuse个时钟周期就必须要使用相应的数据。而在D级只有rs和rt两个使用数据的寄存器地址，由此得到指令集的Tuse表如下：

|  |  |  |  |
| --- | --- | --- | --- |
| 指令 | Tuse | | 功能部件 |
| rs | rt |
| Addu | 1 | 1 | ALU |
| Add | 1 | 1 | ALU |
| Subu | 1 | 1 | ALU |
| Sub | 1 | 1 | ALU |
| And\_ | 1 | 1 | ALU |
| Nor\_ | 1 | 1 | ALU |
| Or\_ | 1 | 1 | ALU |
| Xor\_ | 1 | 1 | ALU |
| Movn | 1 | 1 | ALU |
| Movz | 1 | 1 | ALU |
| Andi | 1 |  | ALU |
| Addiu | 1 |  | ALU |
| Addi | 1 |  | ALU |
| Slti | 1 |  | ALU |
| Sltiu | 1 |  | ALU |
| Ori | 1 |  | ALU |
| xori | 1 |  | ALU |
| Lui |  |  | ALU |
| Lw | 1 |  | DM |
| Lh | 1 |  | DM |
| Lhu | 1 |  | DM |
| Lb | 1 |  | DM |
| Lbu | 1 |  | DM |
| Sw | 1 | 2 | NW |
| Sh | 1 | 2 | NW |
| Sb | 1 | 2 | NW |
| Beq | 0 | 0 | NW |
| Beql | 0 | 0 | NW |
| Bne | 0 | 0 | NW |
| Bgez | 0 |  | NW |
| Bgezal | 0 |  | PC |
| Bgtz | 0 |  | NW |
| Blez | 0 |  | NW |
| Bltz | 0 |  | NW |
| Bltzal | 0 |  | PC |
| J |  |  | NW |
| Jal |  |  | PC |
| Jalr | 0 |  | PC |
| Jr | 0 |  | NW |
| Sll |  | 1 | ALU |
| Sllv | 1 | 1 | ALU |
| Sra |  | 1 | ALU |
| Srav | 1 | 1 | ALU |
| Srl |  | 1 | ALU |
| Srlv | 1 | 1 | ALU |
| Slt | 1 | 1 | ALU |
| Sltu | 1 | 1 | ALU |
| Mult | 1 | 1 | NW |

1. **Tnew**

对于那些能产生新数据的指令，经过分析发现他们产生数据的来源目前有三种：ALU,DM,PC。我们定义对于这些这令在某一流水级还需经过Tnew个时钟周期便可将新数据写入下级流水寄存器，由此得到Tnew表如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 指令 | 功能部件 | E | M | W |
| addu | ALU | 1 | 0 | 0 |
| subu | ALU | 1 | 0 | 0 |
| ori | ALU | 1 | 0 | 0 |
| lui | ALU | 1 | 0 | 0 |
| sll | ALU | 1 | 0 | 0 |
| slav | ALU | 1 | 0 | 0 |
| lw | DM | 2 | 1 | 0 |
| jal | PC8 | 0 | 0 | 0 |
| jalr | PC8 | 0 | 0 | 0 |

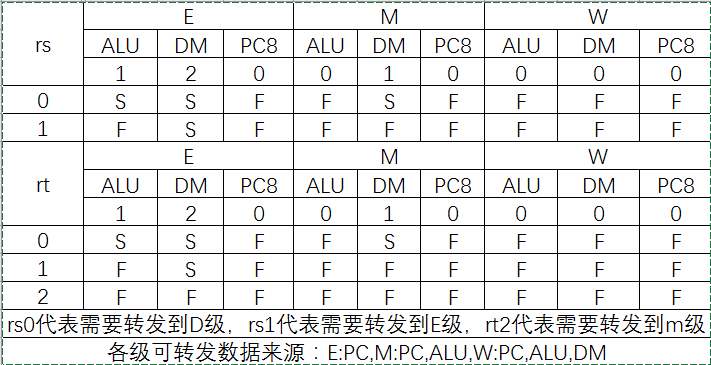
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| E | | | M | | | W | | |
| ALU | DM | PC8 | ALU | DM | PC8 | ALU | DM | PC8 |
| 1 | 2 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

1. **暂停和转发策略**

得到Tuse和Tnew表之后，便可以比较两个时间确定暂停和转发策略。

1. 当Tnew<=Tuse时：后续指令执行需要的数据已经在数据通路中产生，可以全部采用显式转发策略进行解决。
2. 当Tnew>Tuse时：后续指令执行需要的数据未在数据通路中产生，需要暂停流水线cpu运行，等待满足Tnew<=Tuse时再通过相应的转发进行解决。

根据上述原则得到策略矩阵如下：



使用数据：

见上文Tuse图

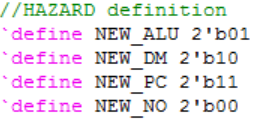
产生数据：

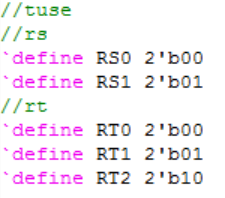
在E级流水线寄存器可转发的数据:PC

在M级流水线寄存器可转发的数据:PC,ALU

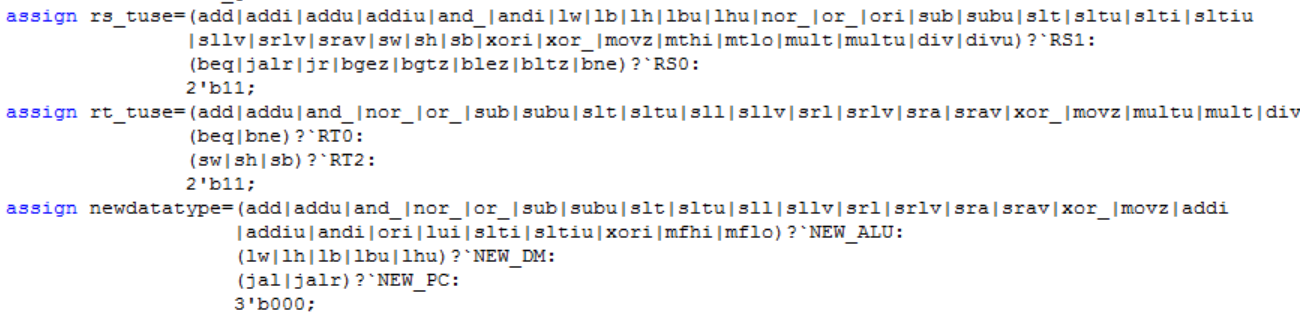
在E级流水线寄存器可转发的数据:PC,ALU,DM

对应的Verilog宏定义如下：





对应的控制器实现如下：

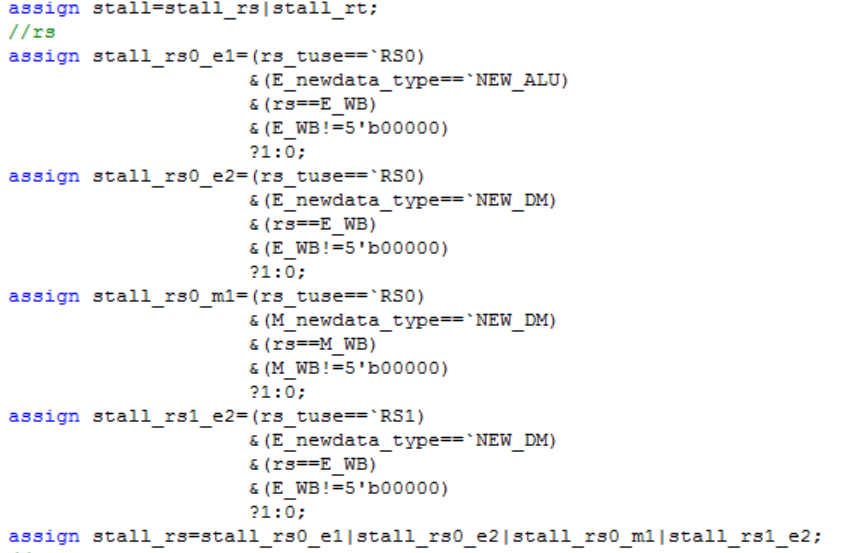


1. **暂停功能具体实现**
2. 暂停策略
3. 使pc暂停，不让其值改变；
4. 使D级流水线暂停，不让其值改变；
5. 清零E级流水线，相当于插入nop指令
6. 暂停情况实现

rs和rt都是相同的四种暂停情况，故此处以rs为例。

根据策略矩阵中S情况对应的Tuse和Tnew，构造如图所示的暂停条件：

1. 满足此时对应的Tuse和Tnew条件
2. 当前指令使用数据的寄存器和前级产生新数据要写入的寄存器相同
3. 为了避免0号寄存器带来的误转发，还要保证产生数据的寄存器不是0号寄存器。

当暂停条件满足时，HAZARD单元使stall\_pc,stall\_D,reset\_E三个信号为1，执行相应操作完成暂停。

1. **转发功能具体实现**
2. 转发条件：

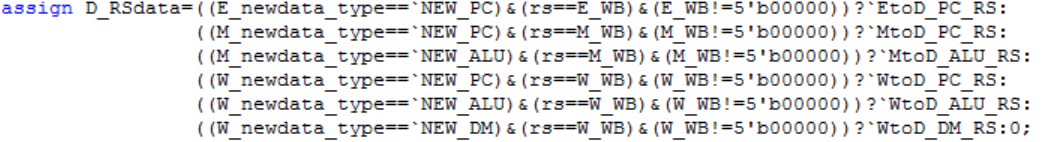
i.下一级寄存器确实产生了新数据；

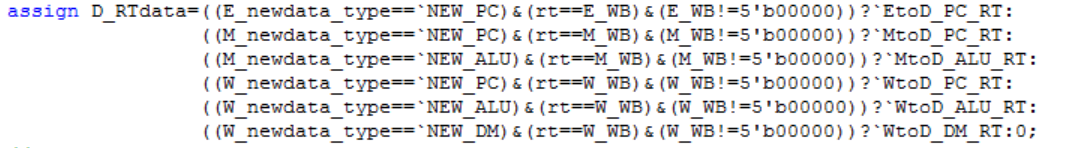
ii.当前指令使用数据的寄存器和前级产生新数据要写入的寄存器相同.

iii.为了避免0号寄存器带来的误转发，还要保证产生数据的寄存器不是0号寄存器。

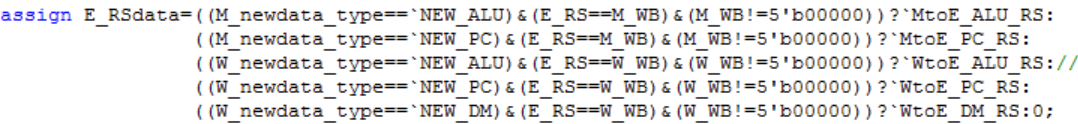
(2) 各级的转发情况

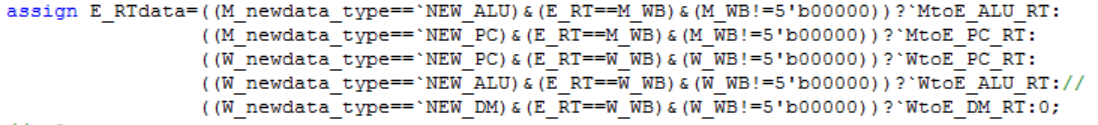
D级：rs和rt位置都需要转发，D级可以接受来自E,M,W三个级的转发数据，E级的转发来源PC,M级的转发来源PC,ALU，W级的转发来源PC,ALU,DM，故总共有可能的六个转发条件，Verilog实现如下：



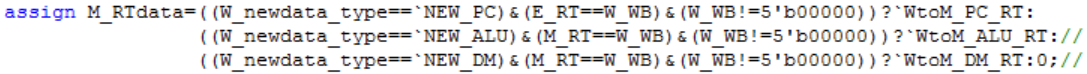


E级：rs和rt位置都需要转发，E级可以接受来自M,W两个级的转发数据，M级的转发来源PC,ALU，W级的转发来源PC,ALU,DM，故总共有可能的五个转发条件，Verilog实现如下：





M级：目前指令集只有rt位置需要转发，M级可以接受来自W级的转发数据， W级的转发来源PC,ALU,DM，故总共有可能的三个转发条件，Verilog实现如下：



新增加的指令，mfc0在M级产生新数据，故Tnew为2，

mtc0在M级使用新数据，故Tuse为2

eret 前序若有mtc0指令修改EPC,或者sw指令修改设备寄存器值，则需要产生新的暂停情况。

**第二部分 测试验证**

1. **中断和异常测试**
2. 新增加三条指令mfc0,mtc0,eret，指令本身的行为，与这三条指令有关的转发和暂停。
3. 计时器的基础操作，通过load和store指令访问设备寄存器来管理设备。
4. 异常类错误，PC错误，最后两位未对齐或PC跳出指定指令区范围。
5. 异常类错误，非法指令，经控制器检测未在指令集53条指令范围内的指令。
6. 异常类错误，ALU算术溢出，针对add,sub,addi三条指令
7. 异常类错误，Load错误，load地址超出规定范围，lw,lh,lhu未对齐，对计时器进行lh,lhu,lb,lbu操作。
8. 异常类错误，Store错误，store地址超出规定范围，sw,sh未对齐，对计时器进行sh,sb操作，写入只读型寄存器。
9. 异常类错误，发生异常的指令为延迟槽指令。
10. 中断错误，计时器模式0和模式1分别触发的中断情况。
11. 中断错误，延迟槽指令发生中断，尤其注意保持jal和jalr的写入
12. 中断错误，暂停时发生中断，能否正确修改PC跳入异常处理程序。
13. 中断错误，暂停时硬件插入的nop发生中断，能否正确将其修改为被阻塞指令的PC值。
14. 中断和异常错误同时发生时，优先进入中断
15. 处于异常处理程序中时，屏蔽一切新产生的中断和异常，即不支持嵌套操作。
16. eret后面硬件插入nop
17. PC错误和非法指令异常产生后视为nop
18. **测试程序**
19. **异常处理程序**

.ktext 0x00004180

mfc0 $25 $12

mfc0 $27 $13

mfc0 $26 $14

ori $19 $0 0x007c

and $11 $19 $27

beq $11 $0 if\_interrupt

nop

if\_not\_interrupt:

ori $23 $0 4

ori $15 $0 0x3000

slt $16 $26 $15

bne $16 $0 pc\_error

nop

lui $20 0x8000 # BD

and $21 $20 $27

subu $22 $20 $21

bne $22 $0 if\_not\_delay\_slot

nop

#add $26 $26 $23

#add $26 $26 $23

mtc0 $26 $14

eret

if\_not\_delay\_slot:

addu $26 $26 $23

mtc0 $26 $14

eret

pc\_error:

ori $26 $0 0x3000

mtc0 $26 $14

eret

if\_interrupt:

ori $12 $0 0

ori $13 $0 0

ori $12 $0 0x0800

and $12 $27 $12

ori $13 $0 0x0800

beq $12 $13 if\_uart

nop

if\_timer:

ori $12 $0 0

ori $13 $0 0

ori $12 $0 0x7F00

ori $13 $0 0x0006

lw $14 0($12)

and $14 $14 $13

bne $14 $0 mode1

nop

mode0:

ori $15 $0 0

sw $15 0($12)

nop

nop

eret

mode1:

nop

nop

nop

eret

nop

nop

if\_uart:

ori $12 $0 0

ori $13 $0 0

ori $12 $0 0x7F10

lw $13 0($12)

sw $13 0($12)

nop

nop

nop

nop

eret

nop

nop

nop

nop

nop

nop

1. **异常和中断程序**

# new instr test

ori $1 $0 0x1234

mtc0 $1 $14

mfc0 $2 $14

ori $1 $0 0

mtc0 $1 $14

mfc0 $2 $14

ori $1 $0 0xfc01 #1111\_1100\_0000\_0001

mtc0 $1 $12

mfc0 $2 $12

ori $1 $0 0

lui $3 0x8000

ori $1 $0 0x007c #0000\_0000\_0111\_1100

addu $1 $1 $3

mtc0 $1 $13

mfc0 $2 $13

mtc0 $0 $13

ori $1 $0 0

ori $1 $0 0x3054

mtc0 $1 $14

eret

lui $4 0x1234

ori $3 $0 0x5678

# CP0 forward

ori $3 $0 0x007c

ori $1 $0 0x8666

mtc0 $1 $14

ori $2 $0 0x7c01

mtc0 $2 $12

sw $3 0($0)

lw $4 0($0)

mtc0 $4 $13

mfc0 $5 $13

sw $5 4($0)

mfc0 $6 $14

subu $6 $0 $6

mfc0 $7 $12

blez $7 if\_1\_else

nop

if\_1:

j end

nop

if\_1\_else:

sll $8 $1 7

end:

sll $8 $1 3

# PC error

lui $3 0x8000

ori $1 $0 0x007c

addu $1 $1 $3

ori $4 $0 4

jr $4

nop

# illegal instr

lui $3 0x8000

ori $1 $0 0x007c

addu $1 $1 $3

ori $4 $0 0x0008

movz $1 $6 $0 #illegal

add $7 $6 $3

nop

# load\_EXC

#beyond the boundary

ori $1 $0 0x3000

ori $2 $0 3

lw $4 0($1)

addu $5 $4 $2

ori $1 $0 0x7F0c

ori $2 $0 4

lw $4 0($1)

addu $5 $4 $2

ori $1 $0 0x7F00

ori $2 $0 5

lw $4 0($1)

addu $5 $4 $2

# unaligned

ori $1 $0 0x1234

lui $2 0x8765

add $2 $2 $1

sw $2 0($0)

lh $10 0($0)

lh $11 1($0)

lh $12 2($0)

lh $13 3($0)

lw $14 0($0)

lw $15 2($0)

lb $16 3($0)

#timer

ori $1 $0 0x7F00

ori $2 $0 6

lw $4 1($1)

addu $5 $4 $2

lh $16 0($1)

addu $30 $1 $1

lhu $16 0($1)

addu $30 $1 $1

lb $15 0($1)

addu $30 $1 $1

lbu $15 0($1)

addu $30 $1 $1

# store\_EXC

#beyond the boundary

ori $1 $0 0x3004

ori $2 $0 0x1234

lui $3 0x9876

add $4 $3 $2

sw $4 0($1)

ori $1 $0 0x0004

sw $4 0($1)

ori $1 $0 0x7F10

ori $6 $0 5

sw $6 0($1)

sw $6 4($1)

sw $6 8($1)

ori $1 $0 0x7F00

sw $6 8($1)

sw $6 0($1)

#unaligned

ori $1 $0 0x1234

lui $2 0x8765

add $2 $2 $1

sb $2 1($0)

sw $2 1($0)

sb $2 6($0)

sw $2 6($0)

sb $2 11($0)

sw $2 11($0)

sb $2 13($0)

sh $2 13($0)

sb $2 18($0)

sh $2 18($0)

sb $2 23($0)

sw $2 23($0)

sb $2 28($0)

sh $2 28($0)

ori $1 $0 0x7f10

#timer

addu $30 $1 $1

sb $2 0($1)

addu $30 $1 $1

sh $2 0($1)

addu $30 $1 $1

sw $2 8($1)

addu $30 $1 $1

#delay slot exception

lui $1 0x7fff

ori $1 $1 0xffff

blez $1 if\_1\_else

add $2 $1 $1

if\_1:

addu $4 $1 $1

j end1

nop

if\_1\_else:

addu $3 $1 $1

end1:

add $2 $1 $1

addu $3 $1 $1

addu $3 $1 $1

jal func1

movz $5 $4 $3

addu $3 $1 $1

add $2 $1 $1

addu $3 $1 $1

addu $3 $1 $1

la $9 func2

jalr $9

add $2 $1 $1

addu $26 $1 $1

addu $27 $1 $1

jr $ra

sw $5 1($0)

j end2

nop

func1:

addu $6 $1 $1

addu $7 $1 $1

jr $ra

sw $5 1($0)

func2:

addu $16 $1 $1

addu $17 $1 $1

jr $ra

sw $5 3($0)

end2:

#interrupt mode0

ori $1 $0 4 # time-cycle

ori $2 $0 0x7f00

ori $5 $0 0x7f00

ori $3 $0 0xfc01 # SR

ori $4 $0 0x0009 #1001

mtc0 $3 $12

sw $1 4($2)

sw $4 0($2)

lui $1 0x7fff

ori $1 $1 0xffff

addu $2 $2 $1

lui $1 0x7fff

ori $1 $1 0xffff

addu $2 $2 $1

lui $1 0x7fff

ori $1 $1 0xffff

addu $2 $2 $1

lui $1 0x7fff

ori $1 $1 0xffff

sw $4 0($5) #

addu $2 $2 $1

lui $1 0x7fff

ori $1 $1 0xffff

addu $2 $2 $1

addu $2 $2 $1

lui $1 0x7fff

ori $1 $1 0xffff

#interrupt mode1

ori $1 $0 4 # time-cycle

ori $2 $0 0x7f00

ori $3 $0 0xfc01 # SR

ori $4 $0 0x000b #1011

mtc0 $3 $12

sw $1 4($2)

sw $4 0($2)

lui $1 0x7fff

ori $1 $1 0xffff

addu $2 $2 $1

lui $1 0x7fff

ori $1 $1 0xffff

addu $2 $2 $1

lui $1 0x7fff

ori $1 $1 0xffff

addu $2 $2 $1

lui $1 0x7fff

ori $1 $1 0xffff

addu $2 $2 $1

lui $1 0x7fff

ori $1 $1 0xffff

addu $2 $2 $1

#中断异常冲突

ori $1 $0 4 # time-cycle

ori $2 $0 0x7f00

ori $5 $0 0x7f00

ori $3 $0 0xfc01 # SR

ori $4 $0 0x0009 #1001

mtc0 $3 $12

sw $1 4($2)

sw $4 0($2)

lui $1 0x7fff

ori $1 $1 0xffff

addu $2 $2 $1

lui $1 0x7fff

ori $1 $1 0xffff

addu $2 $2 $1

lui $1 0x7fff

add $11 $1 $1#### 中断异常冲突

addu $2 $2 $1

lui $1 0x7fff

#延迟槽中断

ori $1 $0 4 # time-cycle

ori $2 $0 0x7f00

ori $5 $0 0x7f00

ori $3 $0 0xfc01 # SR

ori $4 $0 0x0009 #1001

mtc0 $3 $12

sw $1 4($2)

sw $4 0($2)

lui $1 0x7fff

ori $1 $1 0xffff

addu $2 $2 $1

lui $1 0x7fff

addu $2 $2 $1

addu $2 $2 $1

j next

addu $11 $1 $1#### 延迟槽中断

addu $2 $2 $1

lui $1 0x7fff

ori $1 $1 0xffff

next:

ori $1 $1 0xffff

sw $4 0($5) #

addu $2 $2 $1

lui $1 0x7fff

ori $1 $1 0xffff

addu $2 $2 $1

addu $2 $2 $1

#暂停的nop中断

ori $1 $0 4 # time-cycle

ori $2 $0 0x7f00

ori $5 $0 0x7f00

ori $3 $0 0xfc01 # SR

ori $4 $0 0x0009 #1001

mtc0 $3 $12

sw $1 4($2)

sw $4 0($2)

lui $1 0x7fff

ori $1 $1 0xffff

addu $2 $2 $1

lui $1 0x7fff

addu $2 $2 $1

addu $2 $2 $1

lui $1 0xffff

beq $1 $0 if\_1#### 暂停nop中断

addu $2 $2 $1

lui $1 0x7fff

ori $1 $1 0xffff

if\_1:

ori $1 $1 0xffff

sw $4 0($5) #

addu $2 $2 $1

lui $1 0x7fff

#暂停中断

ori $1 $0 4 # time-cycle

ori $2 $0 0x7f00

ori $5 $0 0x7f00

ori $3 $0 0xfc01 # SR

ori $4 $0 0x0009 #1001

mtc0 $3 $12

sw $1 4($2)

sw $4 0($2)

lui $1 0x7fff

ori $1 $1 0xffff

addu $2 $2 $1

nop

nop

lui $1 0x7fff

addu $2 $2 $1

addu $2 $2 $1

lui $1 0xffff

beq $1 $0 if\_1#### 暂停中断

addu $2 $2 $1

lui $1 0x7fff

ori $1 $1 0xffff

if\_1:

ori $1 $1 0xffff

sw $4 0($5) #

addu $2 $2 $1

lui $1 0x7fff

1. **系统功能汇编代码**
2. **简易计算器**

ori $1 $0 0x7F00

ori $2 $0 0x7F10

ori $3 $0 0x7F2C

ori $4 $0 0x7F34

ori $5 $0 0x7F38

ori $6 $0 0x7F40

loop:

lw $11 0($6) # op

lw $12 0($3) # num1

lw $13 4($3) # num2

ori $21 0x1 # add

ori $22 0x2 # or

ori $23 0x4 # and

ori $24 0x8 # xor

ori $25 0x10 # not and

ori $26 0x20 # not nor

ori $27 0x40 # logic left shift 4

ori $28 0x80 # logic right shift 4

andi $15 $11 0xFF

beq $15 $21 add\_

nop

beq $15 $22 or\_

nop

beq $15 $23 and\_

nop

beq $15 $24 xor\_

nop

beq $15 $25 not\_and

nop

beq $15 $26 not\_or

nop

beq $15 $27 sll4

nop

beq $15 $28 srl4

nop

j continue

nop

add\_:

add $16 $12 $13

j continue

nop

or\_:

or $16 $12 $13

j continue

nop

and\_:

and $16 $12 $13

j continue

nop

xor\_:

xor $16 $12 $13

j continue

nop

not\_and:

and $16 $12 $13

not $16 $16

j continue

nop

not\_or:

or $16 $12 $13

not $16 $16

j continue

nop

sll4:

sllv $16 $12 $13

j continue

nop

srl4:

srlv $16 $12 $13

j continue

nop

continue:

sw $16 0($5)

beq $0 $0 loop

nop

1. **计时器和串口回显**

ori $1 $0 0x7F00

ori $2 $0 0x7F10

ori $3 $0 0x7F2C

ori $4 $0 0x7F34

ori $5 $0 0x7F38

ori $6 $0 0x7F40

ori $7 $0 0x0009

ori $8 $0 0x0c01

ori $9 $0 0 #init

mtc0 $8 $12

loop:

lw $10 0($3)

beq $9 $10 end

nop

sw $10 4($1)

sw $7 0($1)

move $9 $10

end:

lw $4 8($1) # 计数值

sw $4 0($5) # 写入数码管

beq $0 $0 loop

nop

**第三部分 思考题**

1. **请查阅相关资料，说一说什么是「FPGA技术」？它有哪些好处和缺陷？**

答：FPGA（Field－Programmable Gate Array），即现场可编程门阵列，它是在PAL、GAL、CPLD等可编程器件的基础上进一步发展的产物。它是作为[专用集成电路](https://baike.baidu.com/item/%E4%B8%93%E7%94%A8%E9%9B%86%E6%88%90%E7%94%B5%E8%B7%AF)（ASIC）领域中的一种半定制电路而出现的。FPGA以并行运算为主，以硬件描述语言来实现。

FPGA器件在结构上 ,由逻辑功能块排列为阵列 ,并由可编程的内部连线连接这些功能块来实现一定的逻辑功能。

它的优点有：既解决了定制电路的不足，又克服了原有可[编程器](https://baike.baidu.com/item/%E7%BC%96%E7%A8%8B%E5%99%A8)件门电路数有限的缺点。而且它可以快速成品，可以被修改来改正程序中的错误，此外还有更便宜的造价。而且FPGA是在逻辑门下编程，使编程有更高的灵活性。FPGA的集成度要比CPLD（复杂可编程逻辑器件）高，具有更复杂的布线结构和逻辑实现。

它的缺陷有：一般来说速度比专用集成电路（ASIC）慢，实现同样的功能面积要比ASIC大。CPLD比FPGA使用起来更方便。CPLD的编程采用E2PROM或FASTFLASH技术，无需外部存储器芯片，使用简单。而FPGA的编程信息需存放在外部存储器上，使用方法复杂。

1. **请简述你的中断实现方案**

答：当UART接受到一次完整的数据后，且停止位为1时，视为发生一次中断，此时将clk\_rf\_av置位1，在clk\_rf\_av上升沿时将rf\_av置为1，rf\_av连接到UART的中断信号IRQ，IRQ连接到外设中断的HWInt[3]位，从而将中断传入CPU，CPU在中断操作中读取UART接受数据寄存器的值，并将其写入到发送数据寄存器，在写入下降沿即over\_head下降沿将rf\_av置为0从而结束中断。