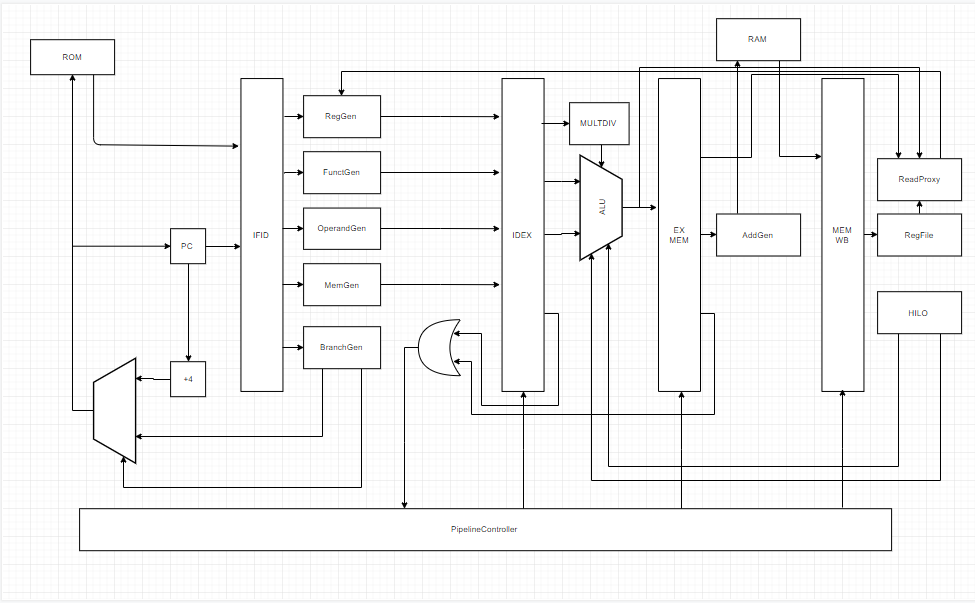
**一、系统总体架构图**

****

**二、组内成员主要工作及贡献比例**

唐挺：主要负责扩展算术运算指令（15%）

吴祉靖：主要负责扩展数据移动指令（20%）

赵佳宾：主要负责扩展移位指令（15%）

薛璞：主要负责扩展逻辑运算指令（15%）

梁逸飞：主要负责扩展分支跳转、访存指令和仿真测试（35%）

**三、设计过程**

六类指令共扩展27个指令。

首先是指令扩展思路。

（1）根据op字段将指令划分为I、J、R型指令

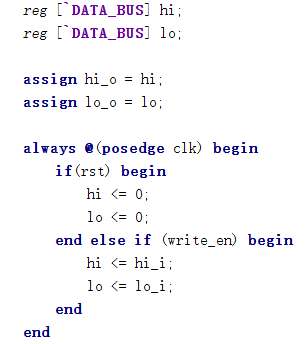
（2）R型指令需补充func字段，I、J需要补充op字段。

（3）根据汇编格式和指令的功能描述分析指令执行过程，在对应级进行相关实现

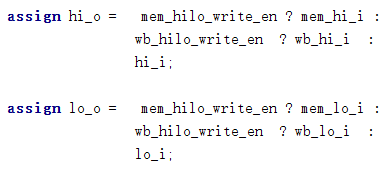
然后开始指令扩展。

1、HILO寄存器

在实现数据移动指令和乘除法指令时都会涉及到HILO寄存器，所以在扩展之前先实现了HILO寄存器。核心部分如下。



对HILO寄存器进行写操作是在WB阶段，而对HILO寄存器内数据进行读取和使用是在 EX阶段，需用代理实现数据前递解决数据冲突。实现如下。



2、算术运算指令

（1）SLTI、SLTIU

|  |
| --- |
| SLTI指令格式：    汇编格式：SLTI rt, rs, imm  功能描述：将寄存器rs的值与**有符号扩展**至32位的立即数imm进行有符号数比较，如果寄存器rs中的值小，则寄存器rt置 1；否则寄存器rt置0。 |

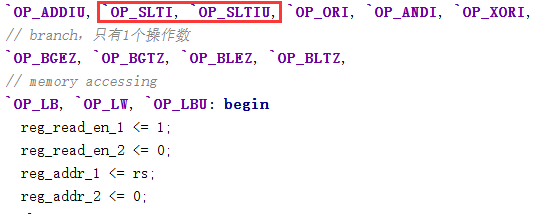
|  |
| --- |
| SLTIU指令格式：    汇编格式：SLTIU rt, rs, imm  功能描述：将寄存器rs的值与**有符号扩展**至32位的立即数imm进行无符号数比较，如果寄存器rs中的值小，则寄存器rt置 1；否则寄存器rt置0。 |

两种指令均为I型指令，需在opcode.v中补充对应op字段。

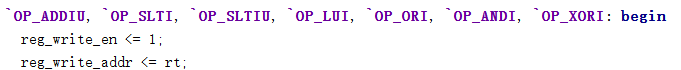


这两种指令涉及读写操作，读取1个寄存器，需要在ID级的RegGen.v中生成寄存器读取信号。

读信号生成：

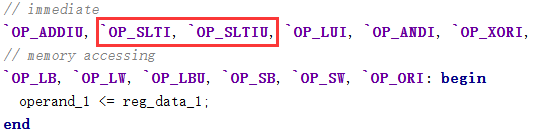


写信号生成：

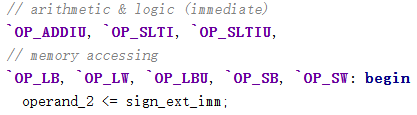


在ID级的时候需读取获取立即数并进行有符号扩展至32位，即在OperandGen.v中生成其操作数。

获取操作数1（寄存器）：



获取操作数2（立即数）：



由于功能和SLT、SLTU一样，所以需在ID级的FunctGen.v中添加对应的func字段。



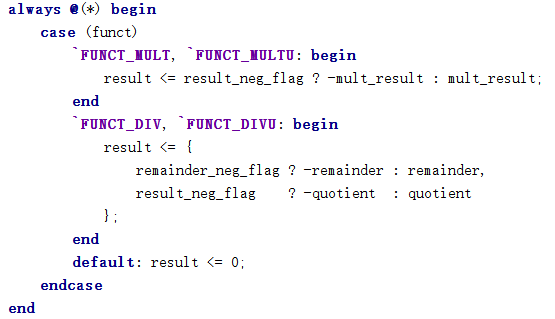
之后在EX级执行之后可以根据其对应的func实现对应功能。

（2）乘除法指令

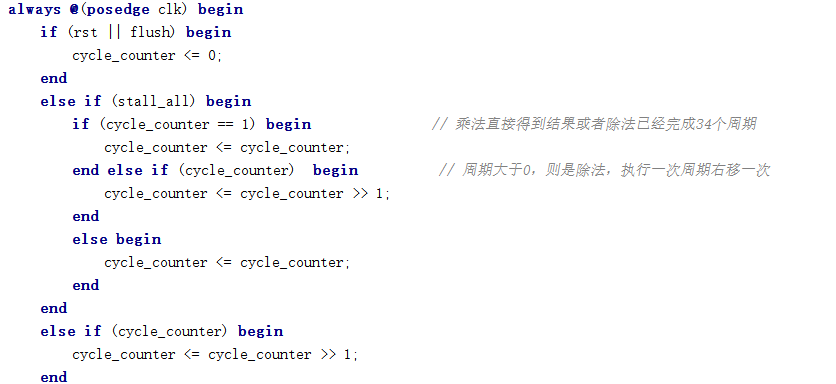
需要用到乘除法器，通过vivado的ip核实现。

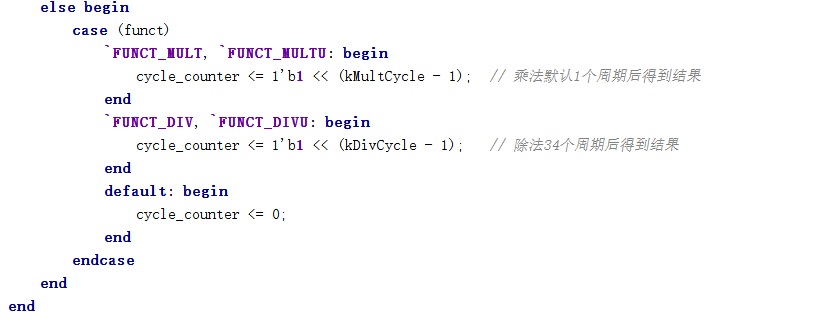
在MultDiv.v中，实现了无符号的乘除法器，还需要添加其他细节处理。

如果结果为负，需要手动取补。

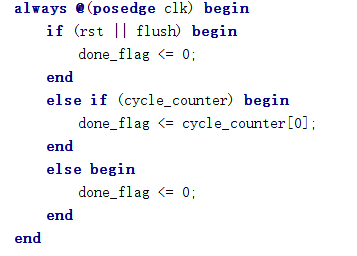


计算乘除法执行周期。乘法默认1个周期得到结果，除法默认34个周期得到结果





乘除法器无法在当前周期得到结果，因此生成一个完成信号done\_flag用于提示当前运算已经完成。



以DIV指令为例：

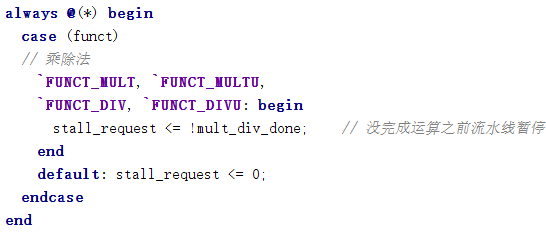
|  |
| --- |
| 指令格式：    汇编格式：DIV rs, rt  功能描述：有符号除法，寄存器rs的值除以寄存器rt的值，商写入LO寄存器中，余数写入HI寄存器中。 |

DIV、DIVU、MULT、MULTU四个指令op字段均为0为R型指令，需在func.v中添加对应功能码和在EX级进行实现，但在进行乘除法运算时，需暂停流水线。

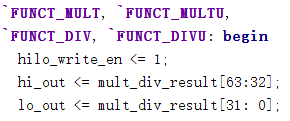
func.v中功能码添加：



EX级中实现流水线暂停：



乘除法器的运算结果为64位，EX级中将结果高32位赋给HI寄存器，低32位赋给LO寄存器。



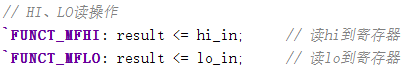
3、数据移动指令

|  |
| --- |
| 以MFHI指令为例：  指令格式：    汇编格式：MFHI rd  功能描述：将 HI 寄存器的值写入到寄存器 rd 中。 |

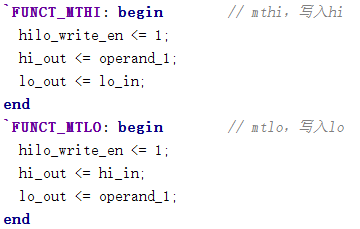
MFHI、MFLO、MTHI、MTLO四个指令op字段均为0，为R型，需在func.v中添加对应功能码。



MFHI、MFLO为读指令，可在EX级将HI、LO的值直接赋给寄存器。



MTHI、MTLO为写指令，写使能无法区分写入的具体为哪一个寄存器，所以需要对HI、LO同时进行写入。



4、逻辑运算指令

（1）NOR

|  |
| --- |
| 指令格式：    汇编格式：NOR rd, rs, rt  功能描述：寄存器rs中的值与寄存器rt中的值按位逻辑或非，结果写入寄存器rd中。 |

该指令的op字段为0，为R型指令，需在func.v中添加对应功能码。



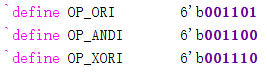
在EX级进行功能实现。



（2）ANDI、ORI、XORI

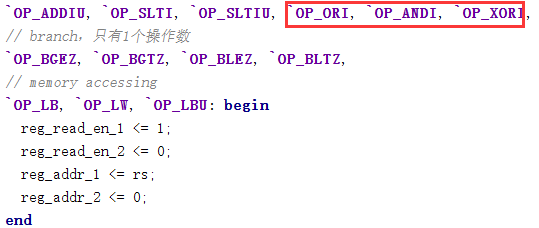
|  |
| --- |
| 以ANDI为例：  指令格式：    汇编格式：ANDI rt, rs, imm  功能描述：寄存器rs中的值**与0扩展**至32位的立即数imm按位逻辑与，结果写入寄存器rt中。 |

三个指令均为为I型指令，需在opcode.v中补充对应op字段。

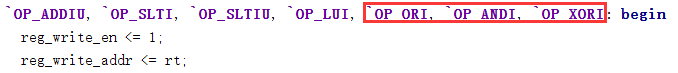


涉及读写操作，读取1个寄存器，并且还要写入寄存器，需在ID级的RegGen.v中生成寄存器读写信号。

读信号生成：

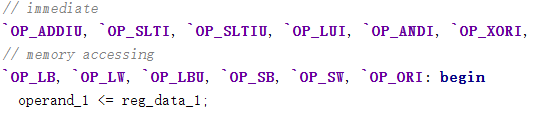


写信号生成：



在ID级OperandGen.v中生成操作数，立即数需进行0扩展。

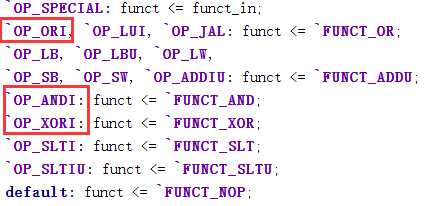
操作数1（寄存器）：



操作数2（立即数）：



三个指令实现的功能是AND、OR、XOR，则需要在ID级的FuncGen.v中添加对应功能码。



之后在EX级执行后可以根据对应的func实现相应功能。

5、移位指令

SRA、SRL。

|  |
| --- |
| SRA指令格式：    汇编格式：SRA rd, rt, sa  功能描述：由立即数sa指定移位量，对寄存器rt的值进行**算术右移**，结果写入寄存器rd中。 |
| SRL指令格式：    汇编格式：SRL rd, rt, sa  功能描述：由立即数sa指定移位量，对寄存器rt的值进行**逻辑右移**，结果写入寄存器rd中。 |

两个指令均为R型指令，需在func.v中添加对应功能码。



在EX级进行功能实现。

逻辑右移高位补0，算数右移高位补符号位。

**注意：rt在汇编格式看来是操作数1，但是在指令格式里看来是操作数2。**



6、分支跳转指令

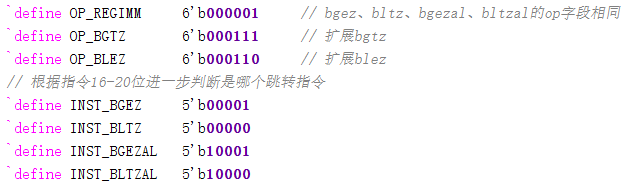
（1）BGEZ、BGTZ、BLEZ、BLTZ、BGEZAL、BLTZAL

|  |
| --- |
| 以BGEZ指令格式：    汇编格式：BGEZ rs, offset  功能描述：如果寄存器rs的值大于等于0则转移，否则顺序执行。转移目标由立即数 offset左移2位并进行有符号扩展的值加上该分支指令对应的延迟槽指令的PC计算得到。 |
| BGTZ指令格式：    汇编格式：BGTZ rs, offset  功能描述：如果寄存器rs的值大于0则转移，否则顺序执行。转移目标由立即数offset 左移2位并进行有符号扩展的值加上该分支指令对应的延迟槽指令的PC计算得到。 |
| BLEZ指令格式：      汇编格式：BLEZ rs, offset  功能描述：如果寄存器rs的值小于等于0则转移，否则顺序执行。转移目标由立即数 offset左移2位并进行有符号扩展的值加上该分支指令对应的延迟槽指令的PC计算得到。 |
| BLTZ指令格式：    汇编格式：BLTZ rs, offset  功能描述：如果寄存器rs的值小于0则转移，否则顺序执行。转移目标由立即数offse左移2位并进行有符号扩展的值加上该分支指令对应的延迟槽指令的 PC 计算得到。 |
| BGEZAL指令格式：    汇编格式：BGEZAL rs, offset  功能描述：如果寄存器rs的值大于等于0则转移，否则顺序执行。转移目标由立即数offset左移2位并进行有符号扩展的值加上该分支指令对应的延迟槽指令的PC计算得到。无论转移与否，将该分支对应延迟槽指令之后的指令的PC值保存至第31号通用寄存器中。 |
| BLTZAL指令格式：    汇编格式：BLTZAL rs, offset  功能描述：如果寄存器 rs 的值小于0则转移，否则顺序执行。转移目标由立即数offset左移2位并进行有符号扩展的值加上该分支指令对应的延迟槽指令的PC计算得到。无论转移与否，将该分支对应延迟槽指令之后的指令的PC值保存至第31号通用寄存器中。 |

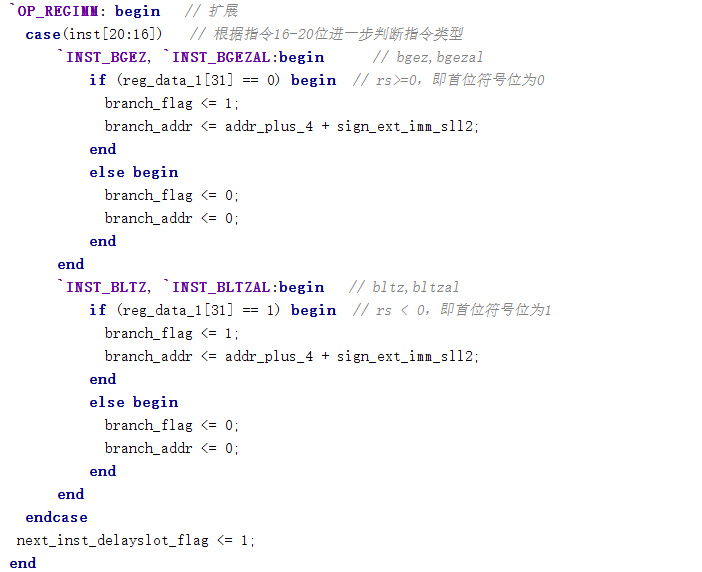
六个指令均为I型分支跳转指令。

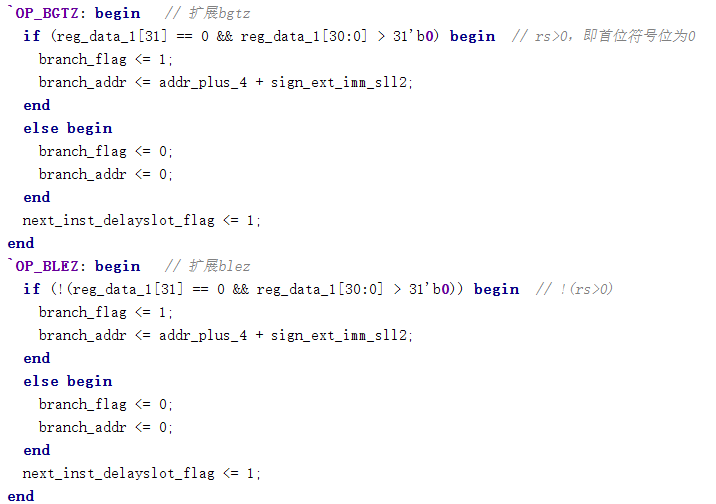
**特别注意：由于BGEZ、BLTZ、BGEZAL、BLTZAL的op字段相同，所以应该根据指令的16-20位进一步判断是哪一种跳转指令。**

参考R型指令设计方法，在opcode.v里我将四个op字段相同的指令合为1个REGIMM，再添加对应的指令标识INST（即指令16-20位），其他两个仍然添加对应op字段。

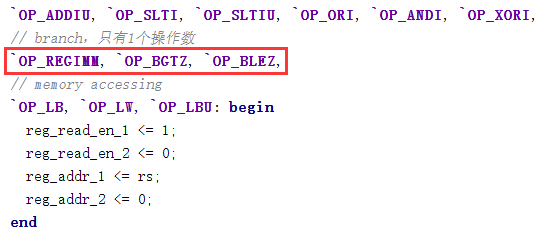


因为是跳转指令，需在ID级的BranchGen.v中生成对应的跳转信号，BGEZ和BGEZAL、BLTZ和BLTZAL跳转功能相同，可以合在一起写。

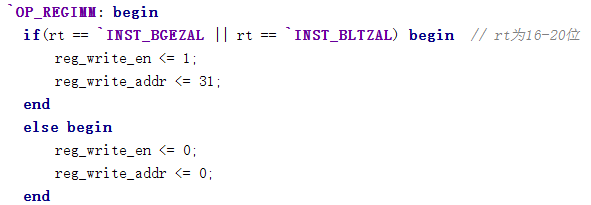




六个指令都涉及读取1个操作，需要在ID级的RegGen.v中生成寄存器读信号。



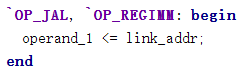
此时BGEZ、BTEZ、BLEZ、BLTZ功能已经实现完成。BGEZAL和BLTZAL还涉及到地址保存的问题，需要进行写操作，所以还需要在ID级的RegGen.v中生成写信号，写入的地址为31号寄存器。



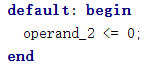
**注意：此时需根据16-20位严格判断BGEZAL和BLTZAL指令，因为其他两个跳转指令不涉及写操作，所以有了if必须要有else，否则会造成数据锁存，导致BGEZ和BLTZ可能会出现误写入的问题。**

由于涉及地址计算，所以需要在ID级的OperandGen.v中生成操作数。

操作数1为地址：



操作数2不必添加，直接进入default为默认值0：



仿照JAL指令扩展方法，地址运算可以用逻辑OR运算（当然也可以用加法或者逻辑与，地址运算可以有多种实现方式），在ID级func.v中生成or功能码，即在EX级会将两个操作数进行或运算生成保存的地址。



由于第二个操作数为0，所以运算结果仍为操作数1，即地址，最终在WB级写入寄存器。

到此BGEZAL和BLTZAL就已经扩展完成。

**问：四个op相同的指令合为一个REGIMM，在生成对应信号的时候也是用REGIMM作为识别标志，都生成了操作数和功能码，那么BGEZ和BLTZ会不会有误写入操作？**

**答：OP\_REGIMM在func.v和OperandGen.v中生成了操作数和功能码，即4个指令都生成了，但是写只有BGEZAL和BLTZAL，BGEZ和BLTZ不会写入。因为在RegGen.v中生成的写信号，根据16-20位严格判断，只有BGEZAL和BLTZAL会生成写信号，BGEZ和BLTZ不满足条件进入else写信号为0，所以尽管生成了操作数和功能码，但都是多余的动作，最终WB级是不会进行写操作。**

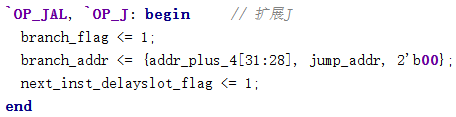
（2）J

|  |
| --- |
| 指令格式：    汇编格式：J target  功能描述：无条件跳转。跳转目标由该分支指令对应的延迟槽指令的PC的最高4位与立即数instr\_index左移2位后的值拼接得到。 |

该指令为J型无条件跳转指令，需在opcode.v中添加对应op字段。



根据功能描述，该指令只是对跳转地址进行运算，没有其他操作，所以无须生成功能码、读写信号，只需在ID级的BranchGen.v中生成跳转信号。



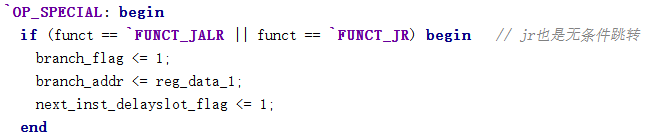
（3）JR

|  |
| --- |
| 指令格式：    汇编格式：JR rs  功能描述：无条件跳转。跳转目标为寄存器rs中的值。 |

JR指令均为R型无条件跳转指令，需在func.v中添加对应的功能码。



同时需在ID级的BranchGen.v中生成跳转信号。



7、访存指令

（1）SH

|  |
| --- |
| 指令格式：    汇编格式：SH rt, offset(base)  功能描述：将base寄存器的值加上符号扩展后的立即数offset得到访存的虚地址，如果地址不是2的整数倍则触发地址错例外，否则据此虚地址将rt寄存器的低半字存入存储器中。 |

该指令是I型写入寄存器的指令，首先在opcode.v中加入对应op字段。

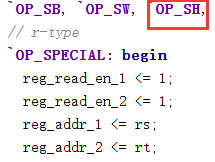


因为涉及到地址的加法运算，所以需在ID级的FunctGen.v中添加对应功能码。



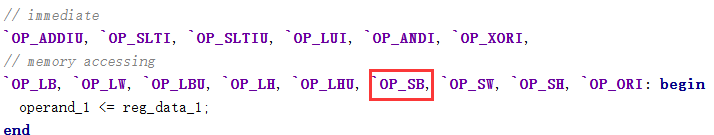
需要读取base和rt寄存器，所以在ID级的RegGen.v中生成对应的寄存器读信号。

读信号：

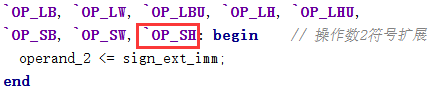


在EX级执行加法操作的时候设计到base寄存器的值和立即数符号扩展后的值，所以在ID级的OperandGen.v中获取对应操作数。

操作数1：



操作数2：

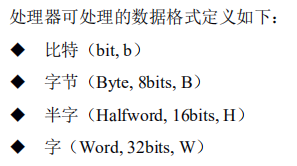


因为是访存指令，需要写入存储器，所以需要在ID级的MemGen.v中生成存储器写信号。

写信号：

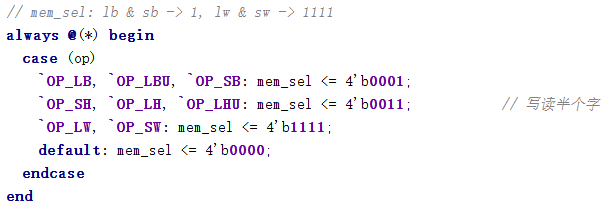


关于数据处理：

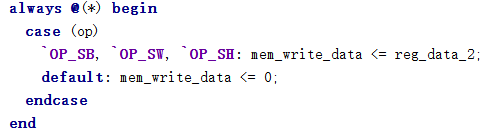


与8086有所不同，1个字等于4个字节，1个半字等于2个字节。

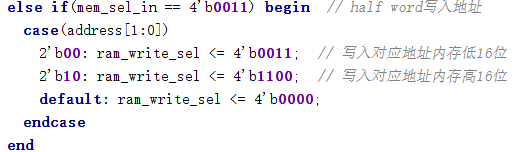
在MemGen.v中生成mem\_sel信号判断写入的字节数，SH是写入半个字，写入1个字节信号是4’b0001，字是4’b1111，则半个字mem\_sel信号应是4’b0011，



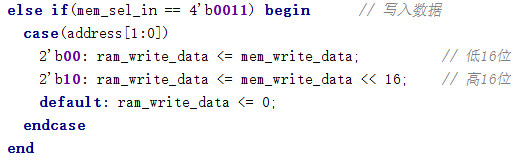
在MemGen.v中生成写入存储器值的来源，为rt寄存器，对应第二个操作数。



ID级的MemGen.v中生成了mem\_sel信号，EX级加法产生了地址，需要在MEM级对mem\_sel（4’b0011）进行处理，根据地址进行判断是写入对应内存的高16位还是低16位并进行数据的写入。



还要在MEM级处理写入存储器的数据。



（2）LH、LHU

|  |
| --- |
| LH指令格式：    汇编格式：LH rt, offset(base)  功能描述：将base寄存器的值加上符号扩展后的立即数offset得到访存的虚地址，如果地址不是2的整数倍则触发地址错例外，否则据此虚地址从存储器中读取连续2个字节的值并进行符号扩展，写入到rt寄存器中。 |
| LHU指令格式：    汇编格式：LHU rt, offset(base)  功能描述：将base寄存器的值加上符号扩展后的立即数offset得到访存的虚地址，如果地址不是2的整数倍则触发地址错例外，否则据此虚地址从存储器中读取连续 2 个字节的值并进行0扩展，写入到rt寄存器中。 |

这两个指令均为I型，需在opcode.v中添加对应op字段。

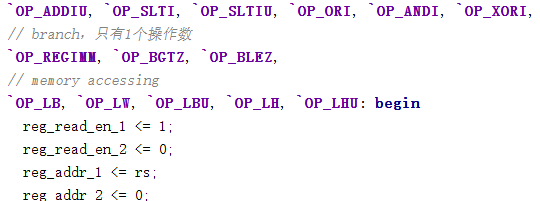


涉及地址加法运算，在ID级的FunctGen.v中添加对应功能码。

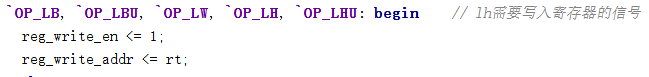


LH、LHU只需要读取base这1个寄存器的值，写入rt寄存器，所以在ID级的RegGen.v中生成寄存器读写信号。

读信号：



写信号：

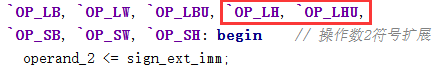


在EX级执行加法操作的时候设计到base寄存器的值和立即数符号扩展后的值，所以在ID级的OperandGen.v中获取对应操作数。

操作数1：

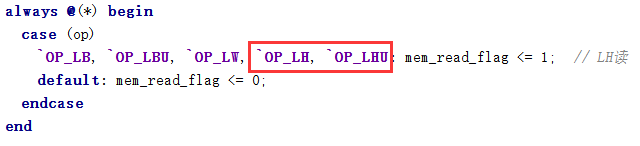


操作数2：

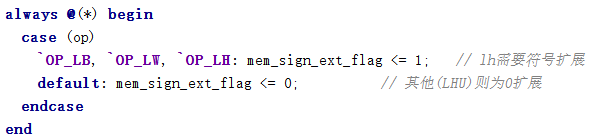


因为是访存指令，读取存储器，需要在ID级的MemGen.v中生成存储器读取信号。

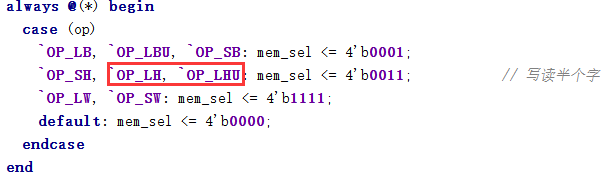
读信号：



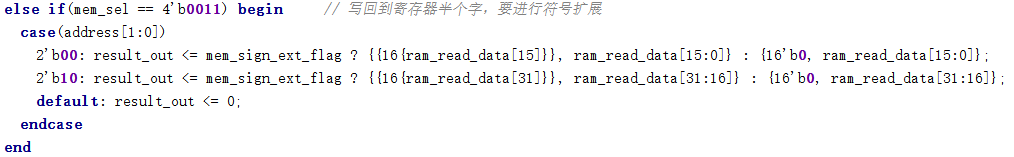
LH和LHU读回的值涉及符号扩展和0扩展，需在ID级的MemGen.v中生成对应的扩展信号。



在MemGen.v中生成mem\_sel信号判断读取的字节数，两个指令都是读取半个字，应是4’b0011。



EX级加法产生了地址，传入MEM级没有操作，再传入WB级进行写回操作，需要在WB级对mem\_sel（4’b0011）进行处理，处理时根据地址的后两位对写回的数据进行判断是写回高16位还是低16位，同时还要对值进行对应的符号或0扩展并进行数据写回。



**四、实验现象及分析**

最终测试是通过trace比对仿真测试来验证所有扩展的指令。

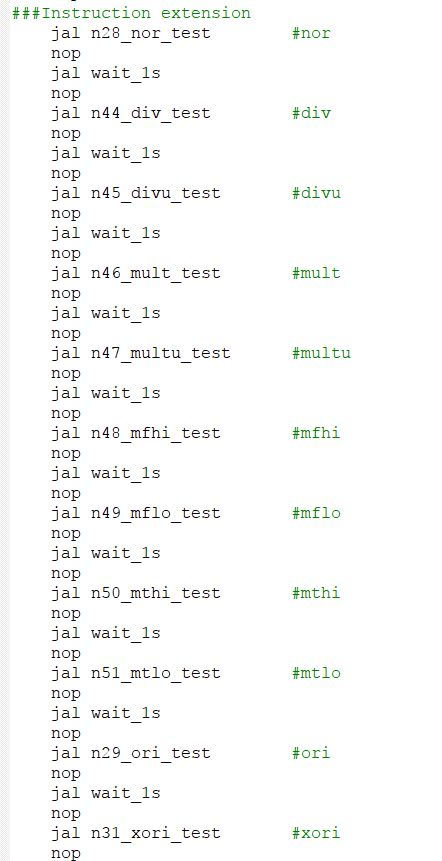
所需环境：Ubuntu16.04（装有gcc-4.3-ls232交叉编译工具)。

1、修改功能测试程序

trace比对的入口程序为start.s文件，只需要在功能测试处添加需要测试的功能点即可。添加了27个功能测试点。



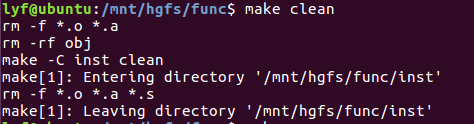
添加的测试功能点（均为扩展的指令）：



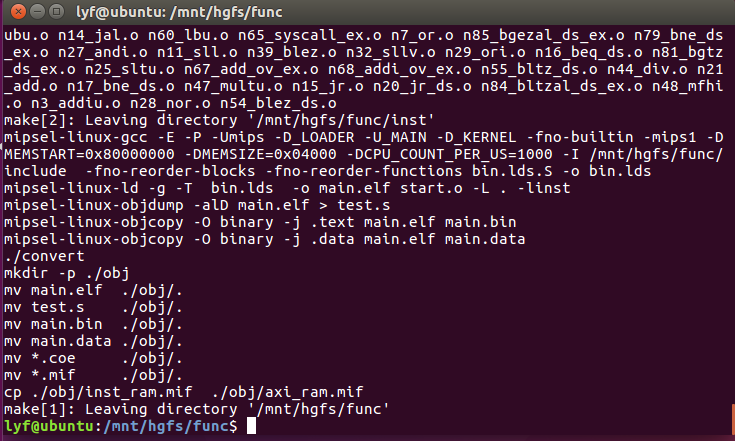
2、编译func

在ubuntu中打开终端命令行，进入func文件夹，先进行make clean操作清除之前编译的文件，再make生成新的编译文件。

make clean:

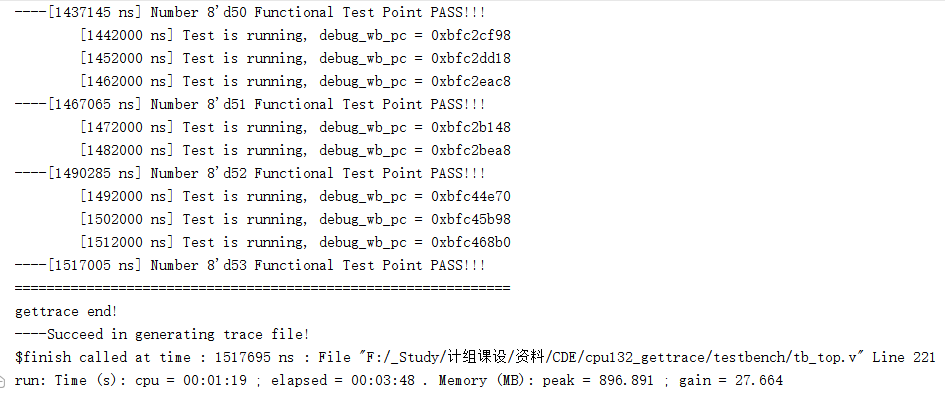


make:



3、生成trace文件

进入cpu132\_gettrace/run\_vivado/cpu132\_gettrace/目录，打开Vivado工程 cpu132\_gettrace，进行仿真，生成参考结果golden\_trace.txt。此时inst\_ram和data\_ram加载的是编译func之后生成的结果。



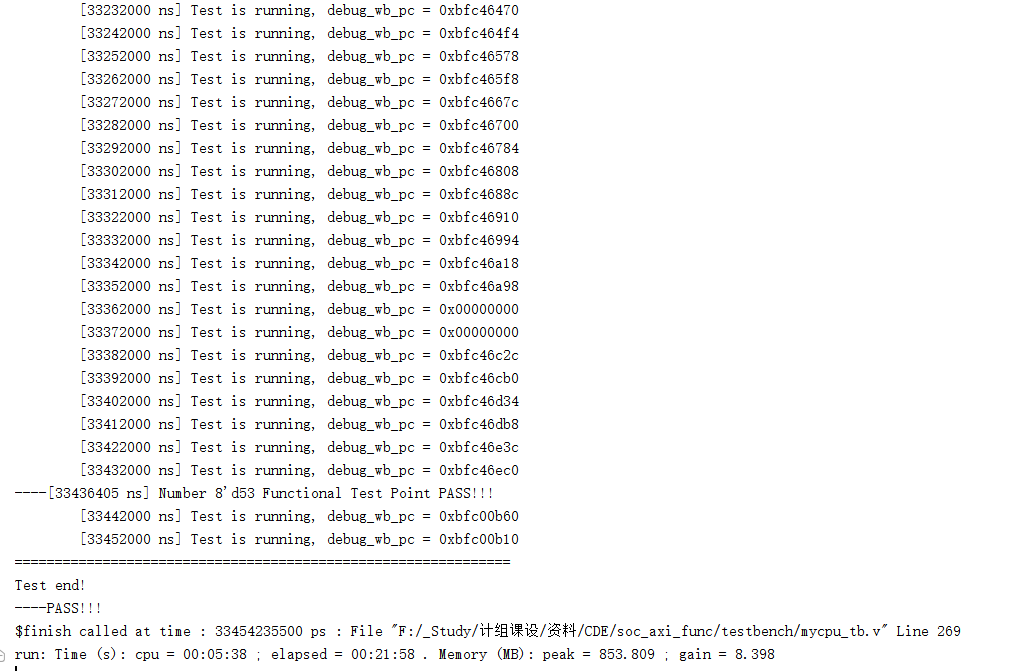
此时已经生成成功，生成了53个功能测试点的比对结果。



最初大小只有986kb，可见参考结果与测试的功能点数量成正比。

4、仿真测试

进入soc\_axi\_func\run\_vivado\mycpu\_prj1目录打开工程mycpu，进行仿真测试。此时inst\_ram和data\_ram加载的是编译func之后生成的结果。观察TCL Console是否有报错信息。



经过漫长的仿真测试，可以看到最终53个功能点全部通过，说明指令扩展全部正确。