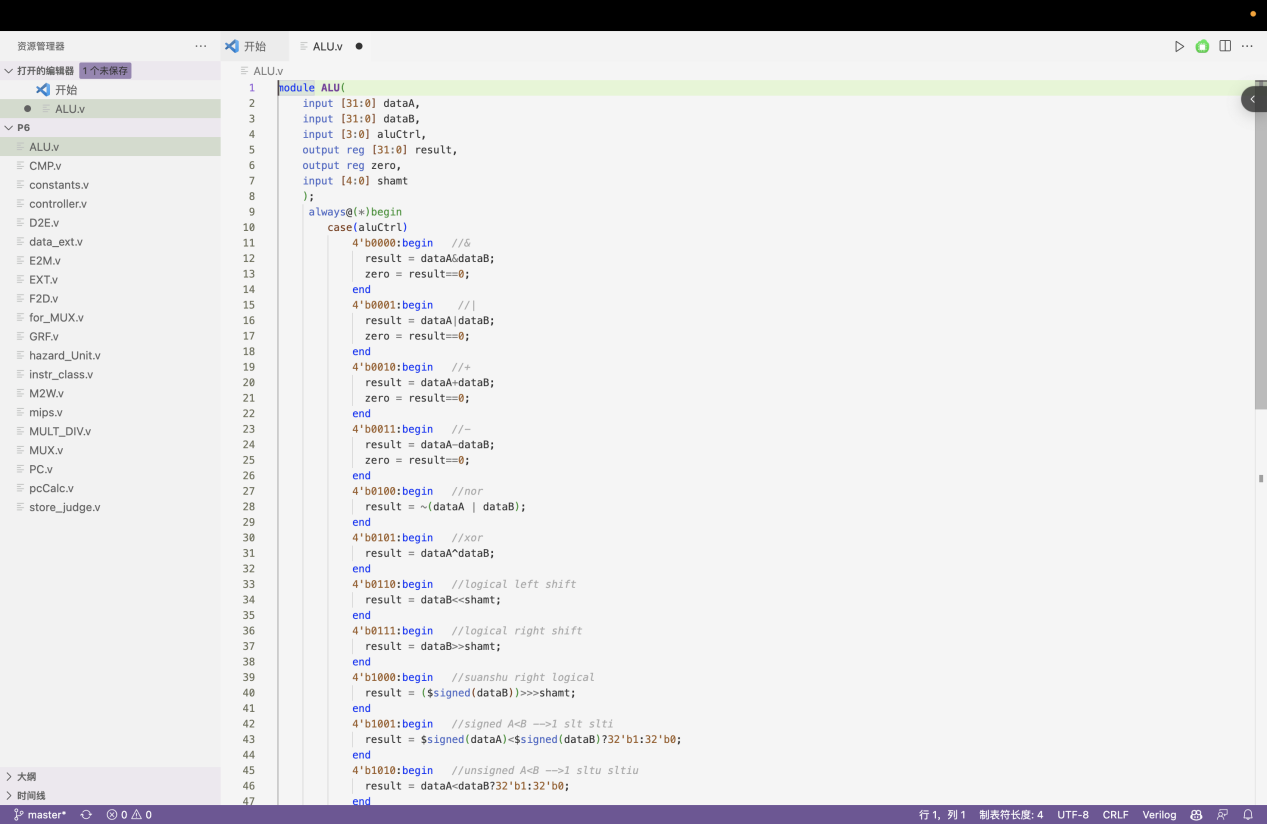
P6 设计报告-陈伟杰

模块设计

ALU

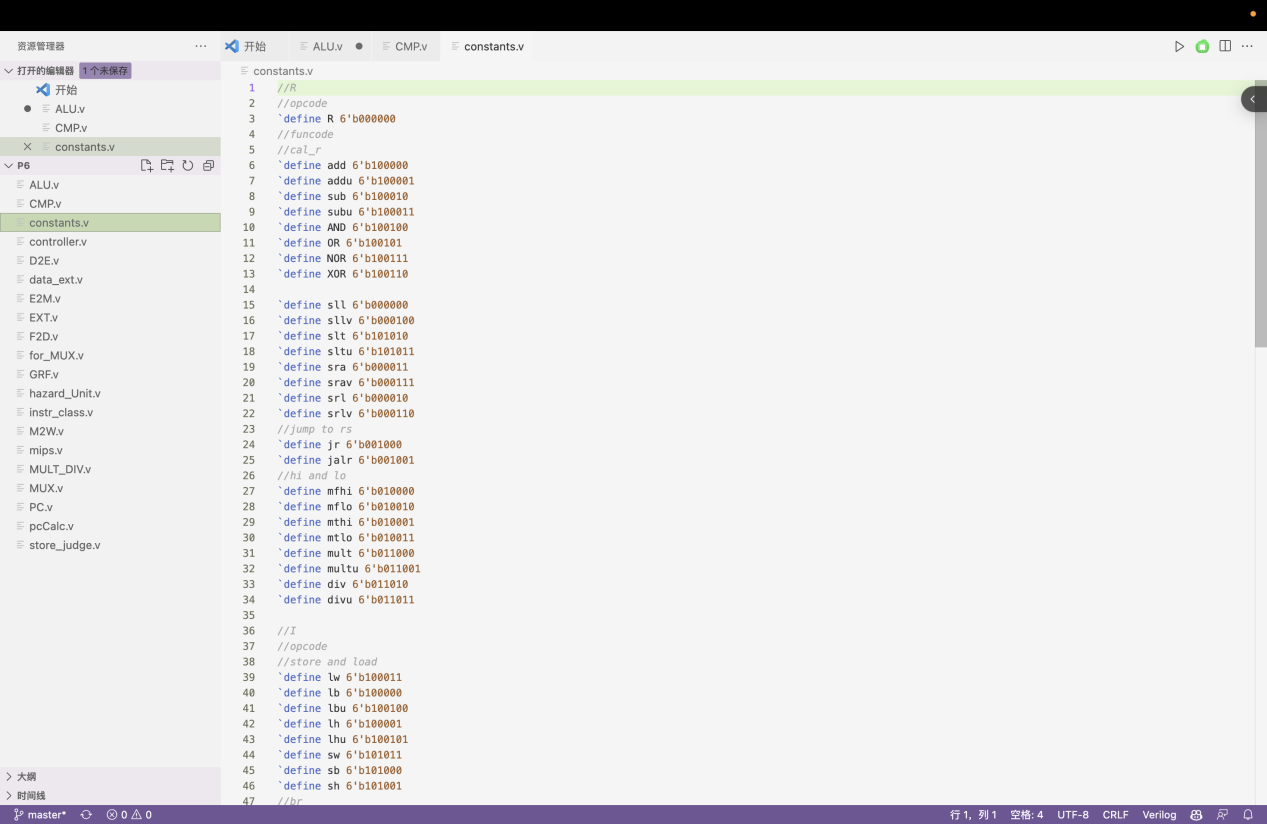


CMP



Constant

这个文件主要定义不同类型opcode和function



Controller

`include "constants.v"

module **controller**(

input [31:0] instr,

input [5:0] op,

input [5:0] func,

output [1:0] regDst,

output aluSrc,

output regWrite,

output memRead,

output memWrite,

output [2:0] memToReg,

output extOp,

output [3:0] branch,

output jump,

output [3:0] aluCtrl,

output pcSrc,

output shamt\_or\_rs,

output mord,

output wHiLo,

output weMD,

output mdStart,

output [2:0] extRdOp,

output [1:0] storeOp,

output signmd,

output rHiLo

);

wire instr\_20\_16;

assign instr\_20\_16=instr[20:16];

reg addu=0,subu=0,sub,add,AND,OR,XOR,NOR,sll,sllv,slt,sltu,sra,srav,srl,srlv;

reg jr=0,jalr,jal=0,j=0;

reg lui,ori,addi,addiu,andi,xori,slti,sltiu;

reg beq=0,bne,bgtz,blez,bgez,bltz;

reg mfhi,mflo,mthi,mtlo,mult,multu,div,divu;

reg sw,sb,sh;

reg lw,lb,lbu,lh,lhu;

reg nop=0;

always@(\*)begin

addu=0;subu=0;sub=0;add=0;AND=0;OR=0;XOR=0;NOR=0;

sll=0;sllv=0;slt=0;sltu=0;sra=0;srav=0;srl=0;srlv=0;

beq=0;bne=0;bgtz=0;blez=0;bgez=0;bltz=0;

jr=0;jalr=0;jal=0;j=0;

sw=0;sb=0;sh=0;

lw=0;lb=0;lbu=0;lh=0;lhu=0;

mfhi=0;mflo=0;mthi=0;mtlo=0;mult=0;multu=0;div=0;divu=0;

lui=0;ori=0;addi=0;addiu=0;andi=0;xori=0;slti=0;sltiu=0;

nop=0;

case(op)

`R:begin

case(func)

`addu:addu=1;

`subu:subu=1;

`sub:sub=1;

`add:add=1;

`AND:AND=1;

`OR:OR=1;

`XOR:XOR=1;

`NOR:NOR=1;

`R:begin

if(instr!=0)sll=1;

else nop=1;

end

`sllv:sllv=1;

`slt:slt=1;

`sltu:sltu=1;

`sra:sra=1;

`srav:srav=1;

`srl:srl=1;

`srlv:srlv=1;

`jr:jr=1;

`jalr:jalr=1;

`mfhi:mfhi=1;

`mflo:mflo=1;

`mthi:mthi=1;

`mtlo:mtlo=1;

`mult:mult=1;

`multu:multu=1;

`div:div=1;

`divu:divu=1;

default:nop=1;

endcase

end

`beq:beq=1;

`bne:bne=1;

`bgtz:bgtz=1;

`blez:blez=1;

`bgeltz:begin

if(instr\_20\_16==0)bltz=1;

else if(instr\_20\_16==1)bgez=1;

end

`lui:lui=1;

`ori:ori=1;

`addi:addi=1;

`addiu:addiu=1;

`andi:andi=1;

`xori:xori=1;

`slti:slti=1;

`sltiu:sltiu=1;

`lw:lw=1;

`lb:lb=1;

`lbu:lbu=1;

`lh:lh=1;

`lhu:lhu=1;

`sw:sw=1;

`sb:sb=1;

`sh:sh=1;

`j:j=1;

`jal:jal=1;

default:nop=1;

endcase

end

wire cal\_i,cal\_r,cal\_md,st,ld,b,jump1,w\_md,r\_md;

assign cal\_i=lui|ori|addi|addiu|andi|xori|slti|sltiu;

assign cal\_r=add|addu|sub|subu|AND|OR|XOR|NOR|sll|sllv|sra|srav|srl|srlv|slt|sltu;

assign cal\_md=mult|multu|div|divu;

assign r\_md=mflo|mfhi;

assign w\_md=mthi|mtlo;

assign jump1=j|jalr|jal|jr;

assign b=beq|bgtz|blez|bgez|bltz|bne;

assign ld=lw|lb|lbu|lh|lhu;

assign st=sw|sb|sh;

assign regDst = (cal\_r|jalr|r\_md)?2'b01:

(jal)?2'b10:2'b00;

assign aluSrc = cal\_i|st|ld;

assign regWrite = cal\_r|cal\_i|r\_md|jal|jalr|ld;

assign memRead = ld;

assign memWrite = st;

assign memToReg = (ld)?3'b001:

(jump1)?3'b010:

(mfhi|mflo)?3'b011:

3'b000;

assign extOp = (xori|andi|ori)?1:0;

assign branch = beq?4'b0001:

bne?4'b0010:

bgtz?4'b0011:

blez?4'b0100:

bgez?4'b0101:

bltz?4'b0110:4'b0000;

assign aluCtrl = (AND|andi)?4'b0000:

(ori|OR)?4'b0001:

(addu||add||addi||addiu||ld||st)?4'b0010:

(subu|sub)?4'b0011:

(NOR)?4'b0100:

(XOR|xori)?4'b0101:

(sll|sllv)?4'b0110:

(srl|srlv)?4'b0111:

(sra|srav)?4'b1000:

(slt|slti)?4'b1001:

(sltu|sltiu)?4'b1010:

(lui)?4'b1011:

4'b0000;

assign jump = (jal|j);

assign pcSrc = (jr|jal|j|jalr);

assign shamt\_or\_rs=srl|sra|sll;

assign extRdOp=lbu?3'b001:

lb?3'b010:

lhu?3'b011:

lh?3'b100:3'b000;

assign storeOp=sh?2'b01:sb?2'b10:2'b00;

assign mdStart=cal\_md;

assign mord=div|divu;

assign wHiLo=mtlo;

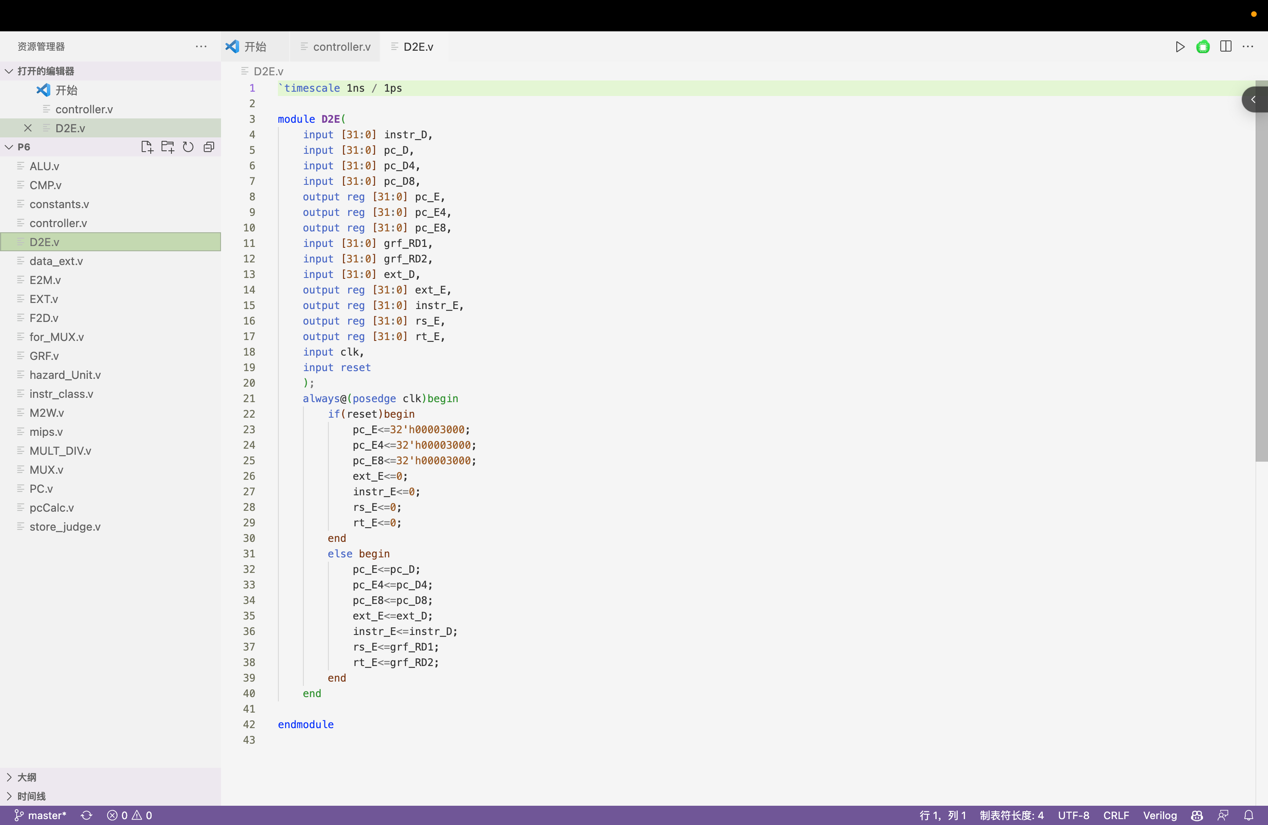
assign weMD=w\_md;

assign signmd=mult|div;

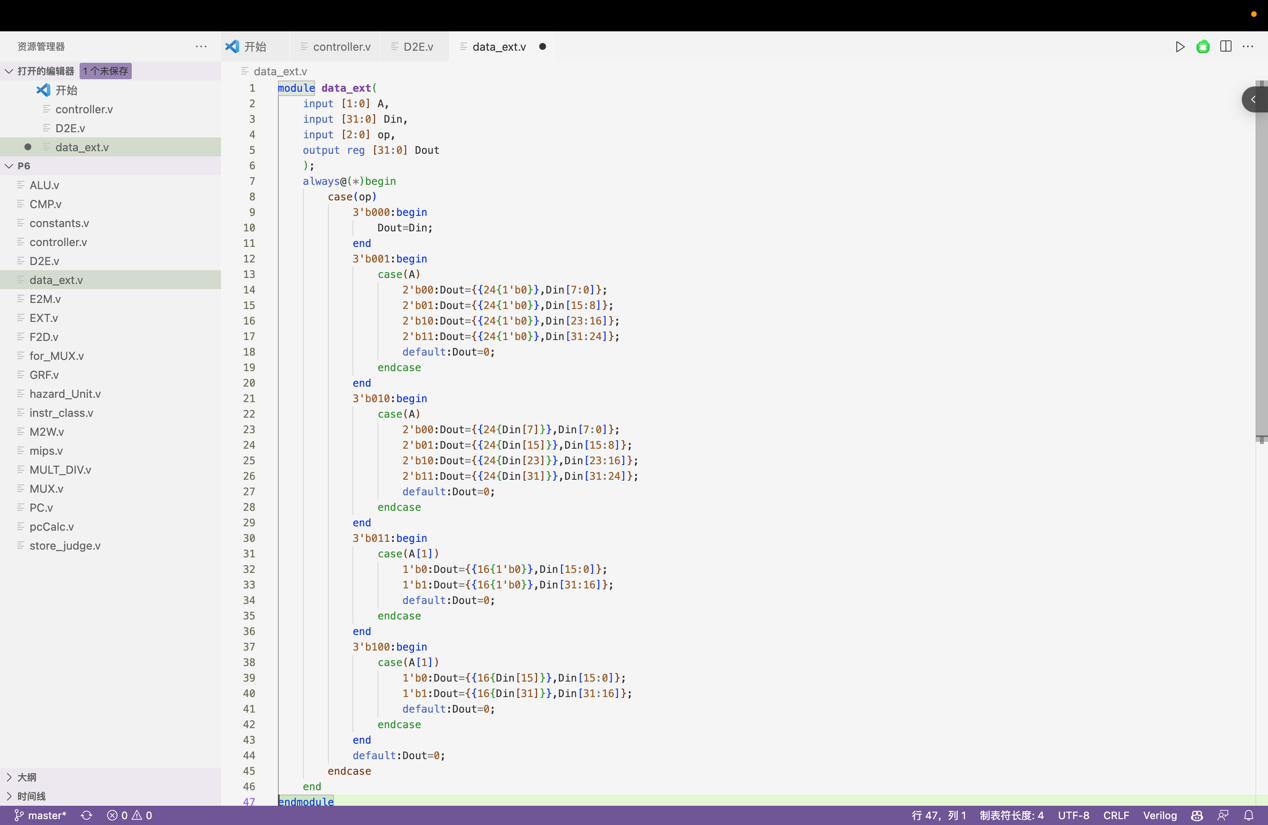
assign rHiLo=mflo;

endmodule

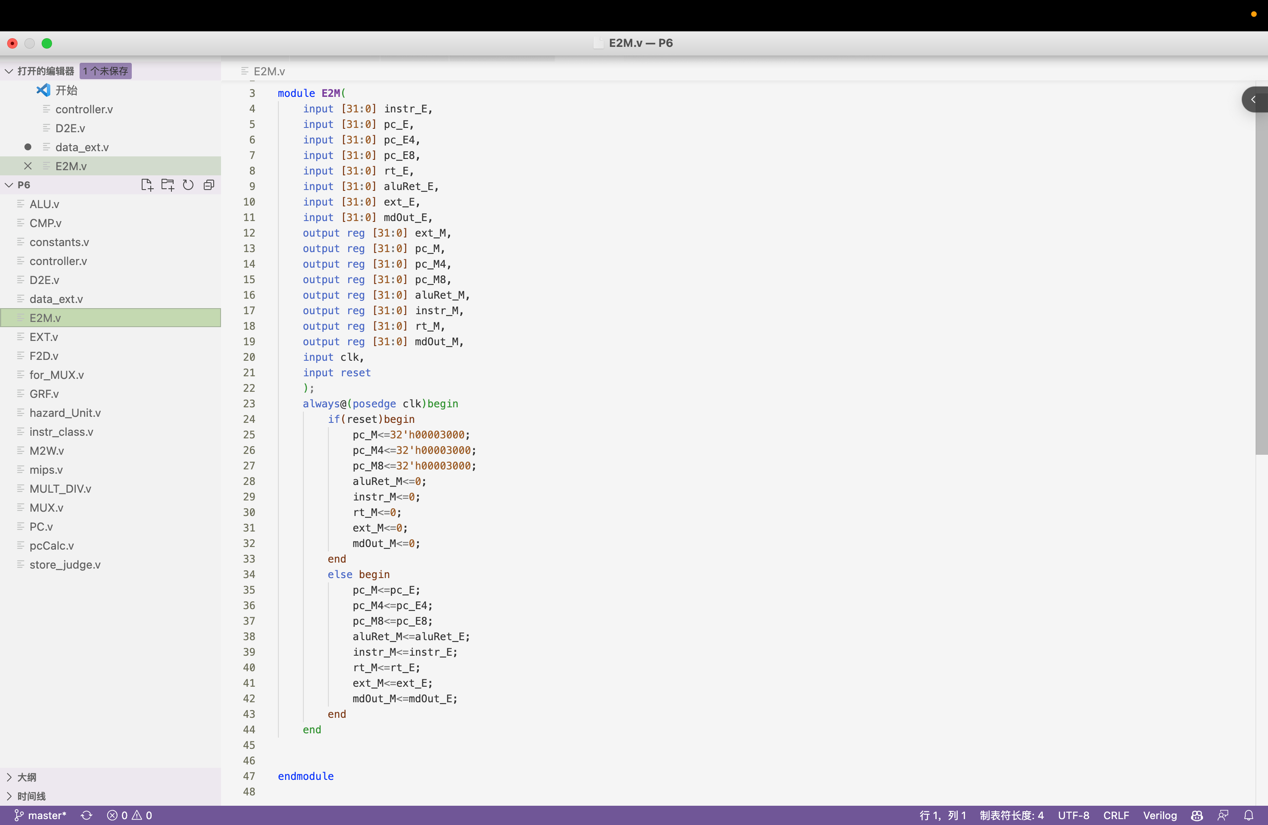
D2E



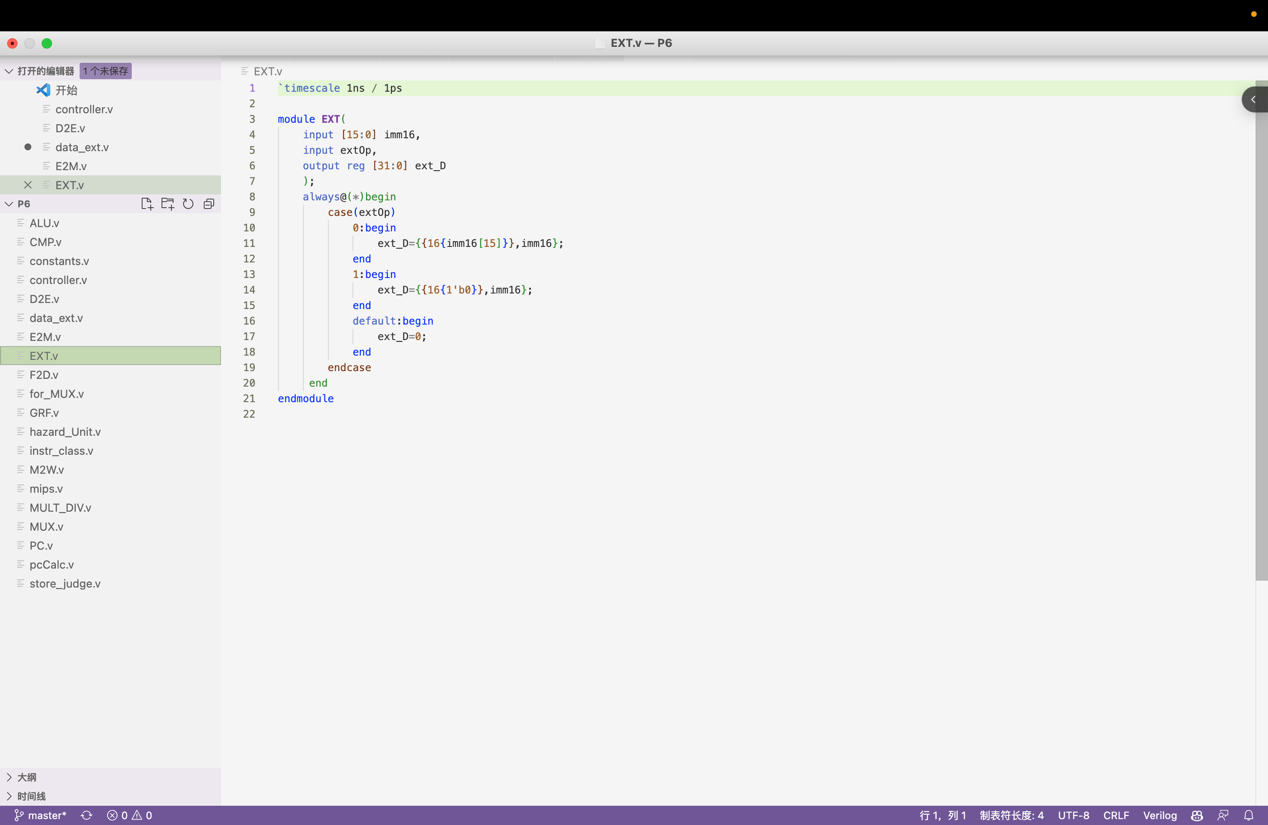
Data\_ext



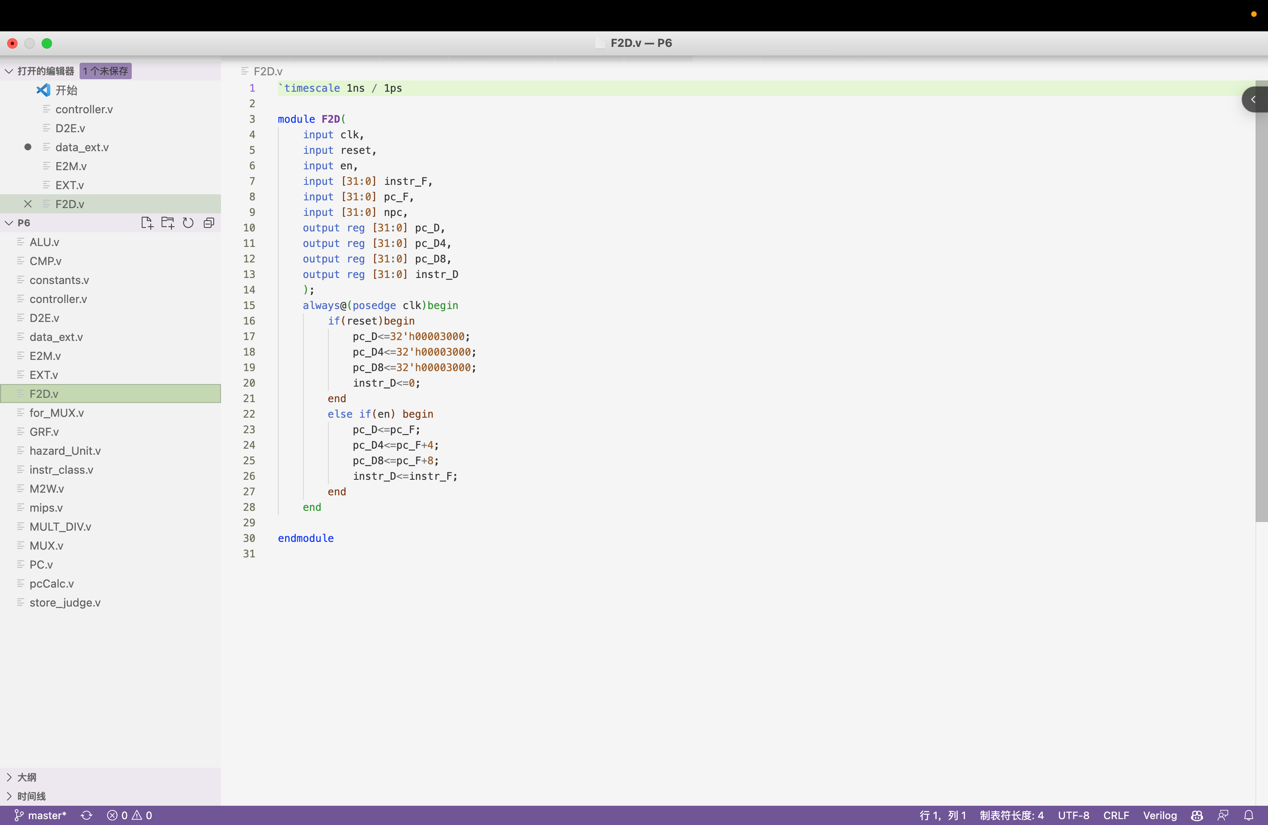
E2M



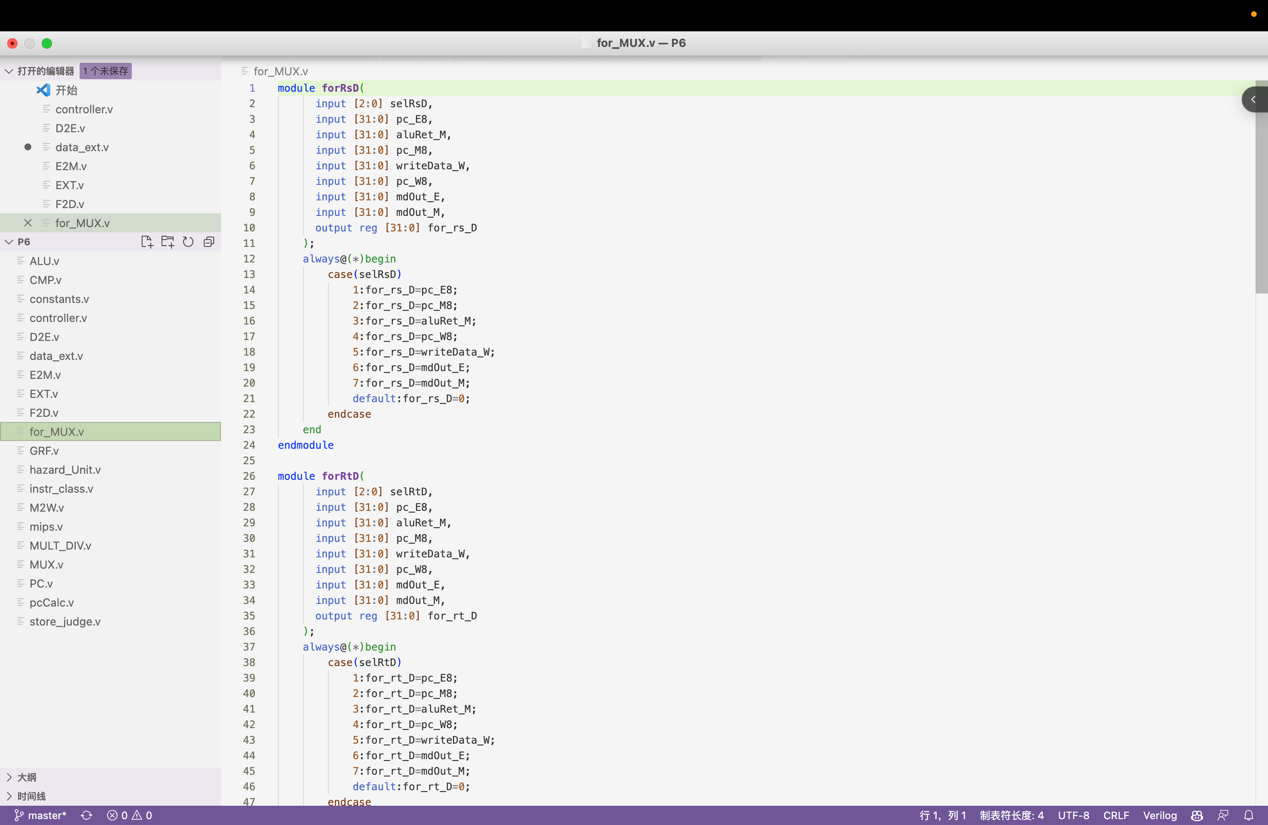
EXT



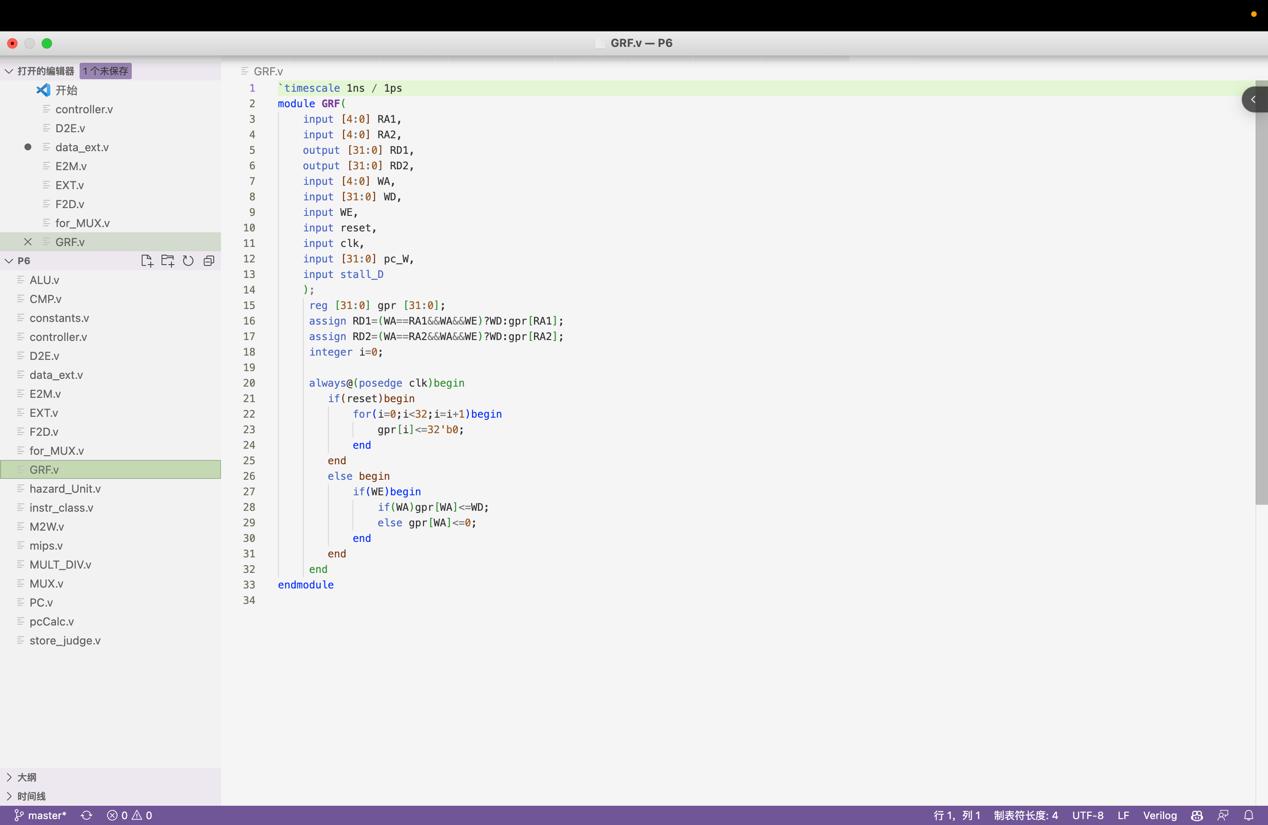
F2D



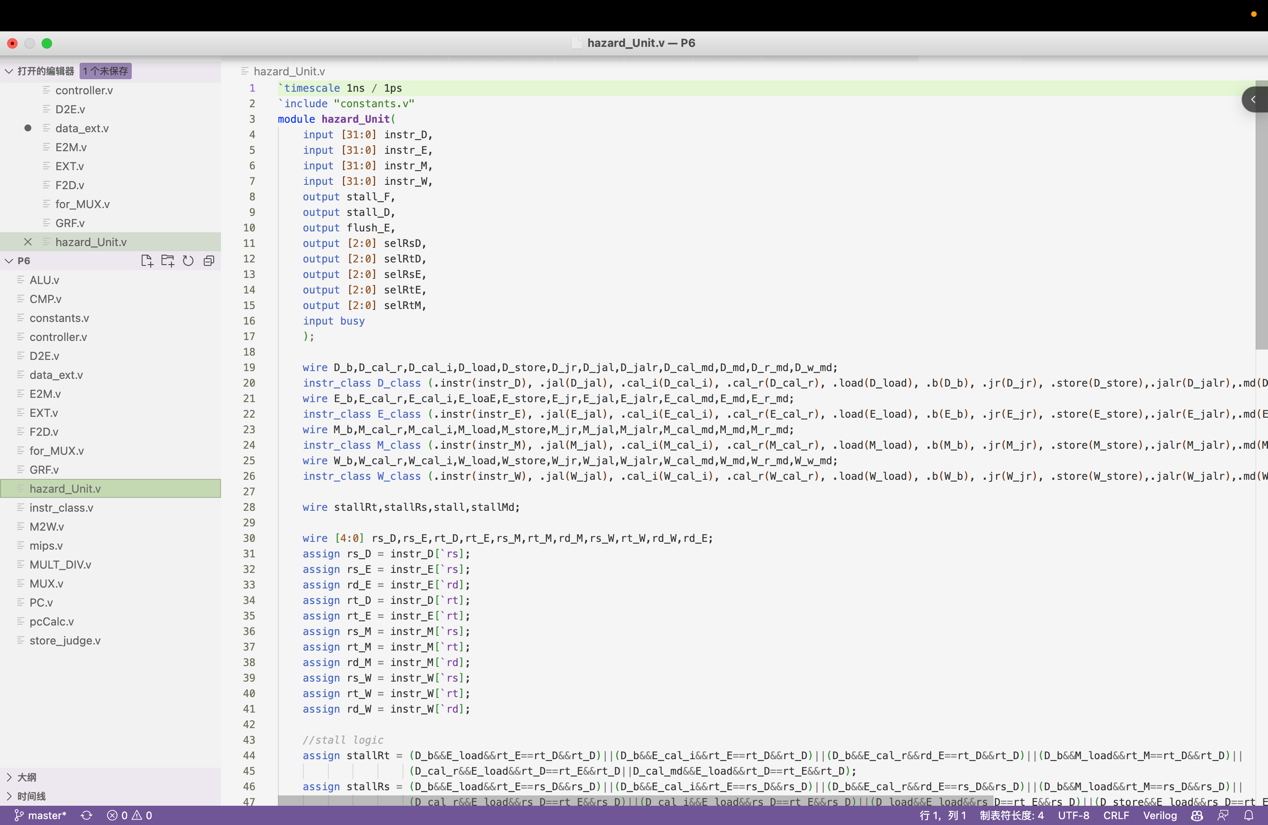
For-mux



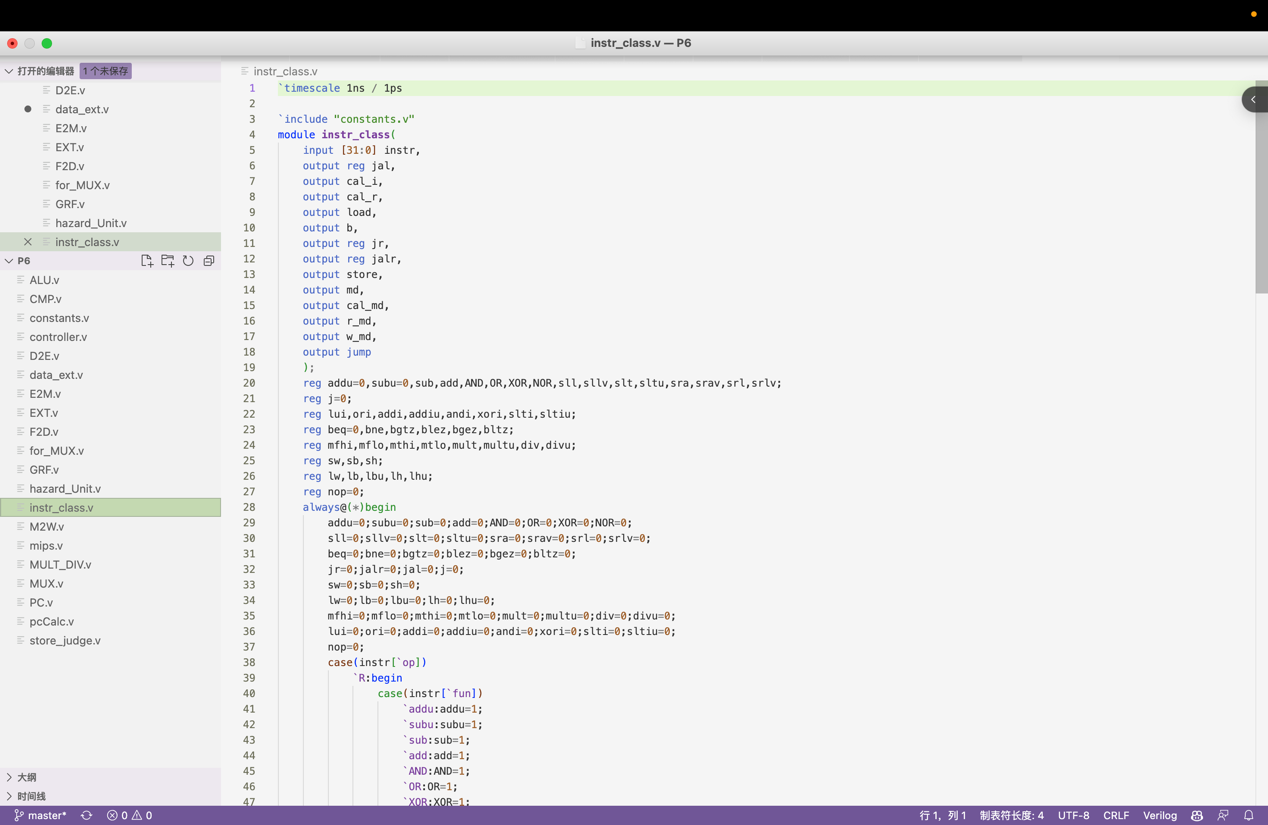
GRF



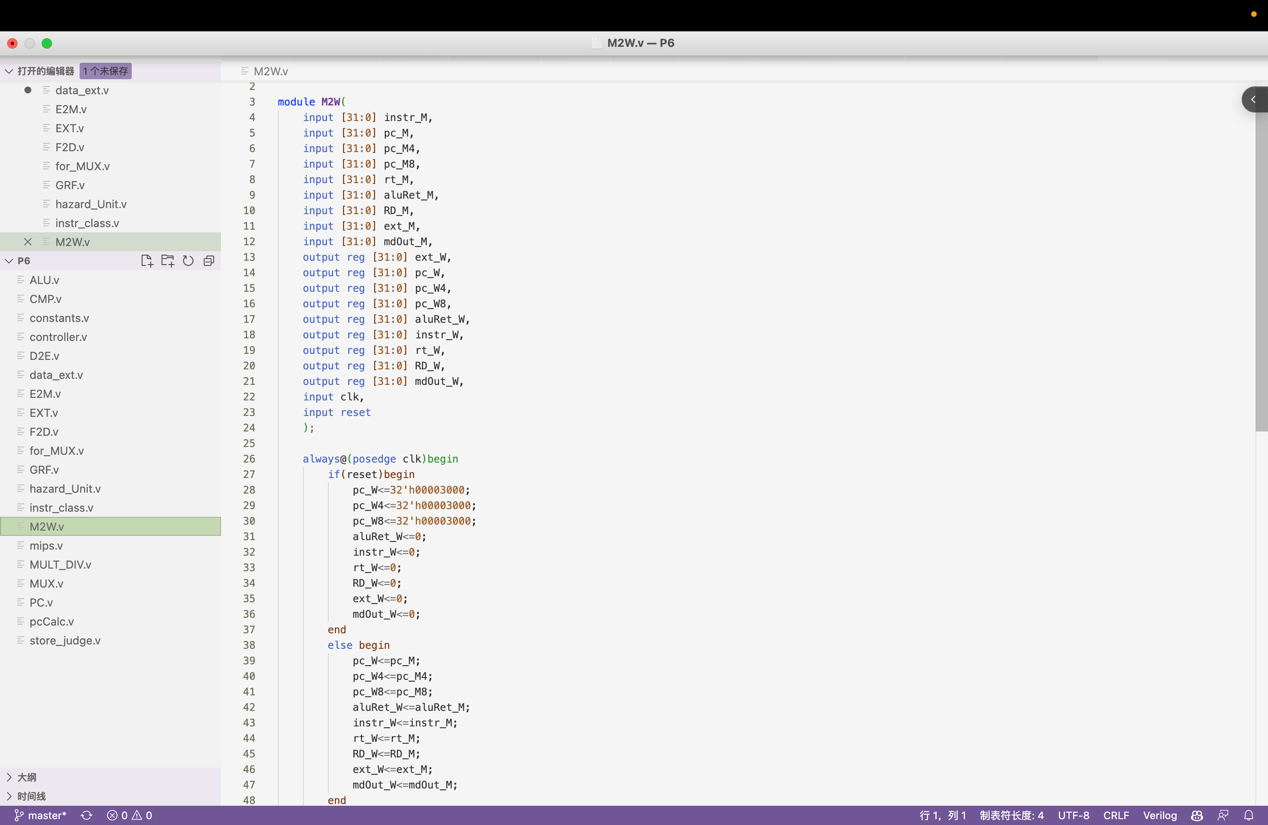
Hazard\_Unit



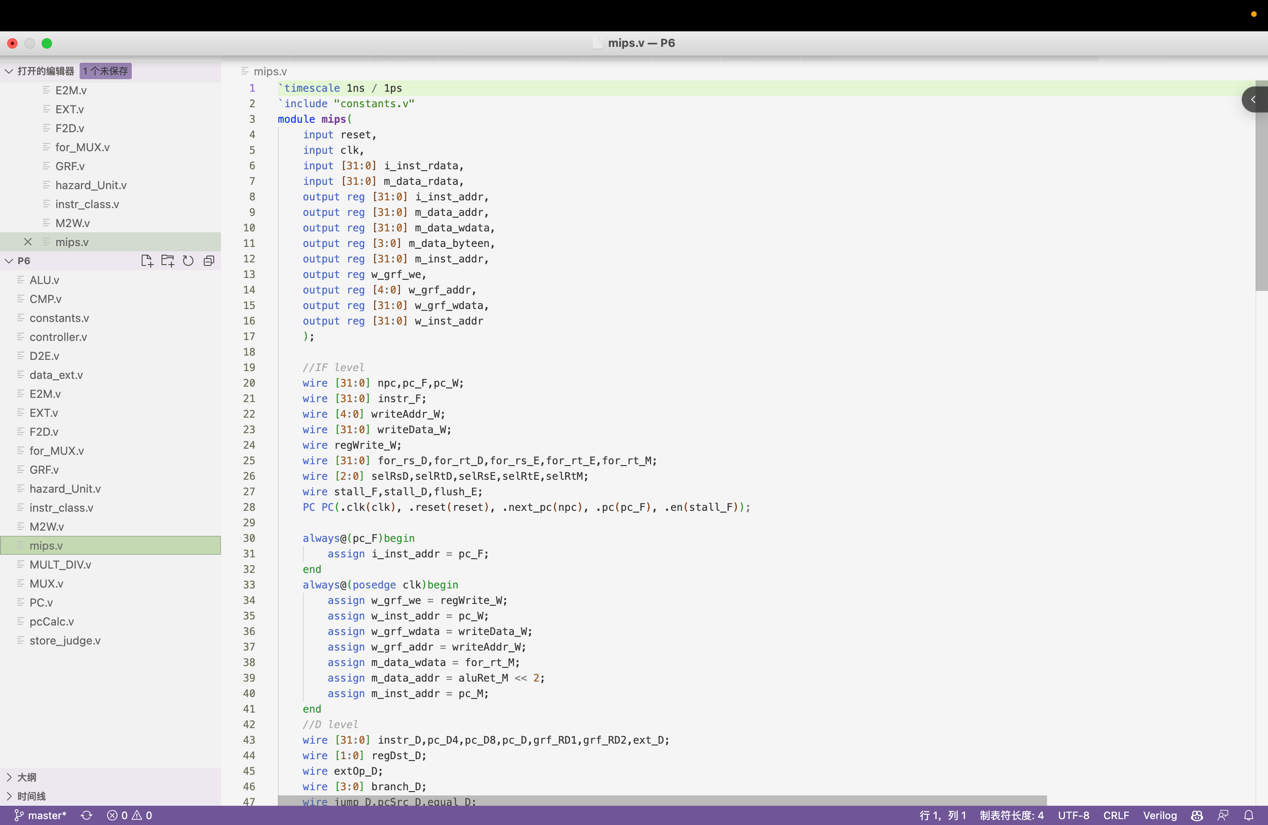
Instr\_class



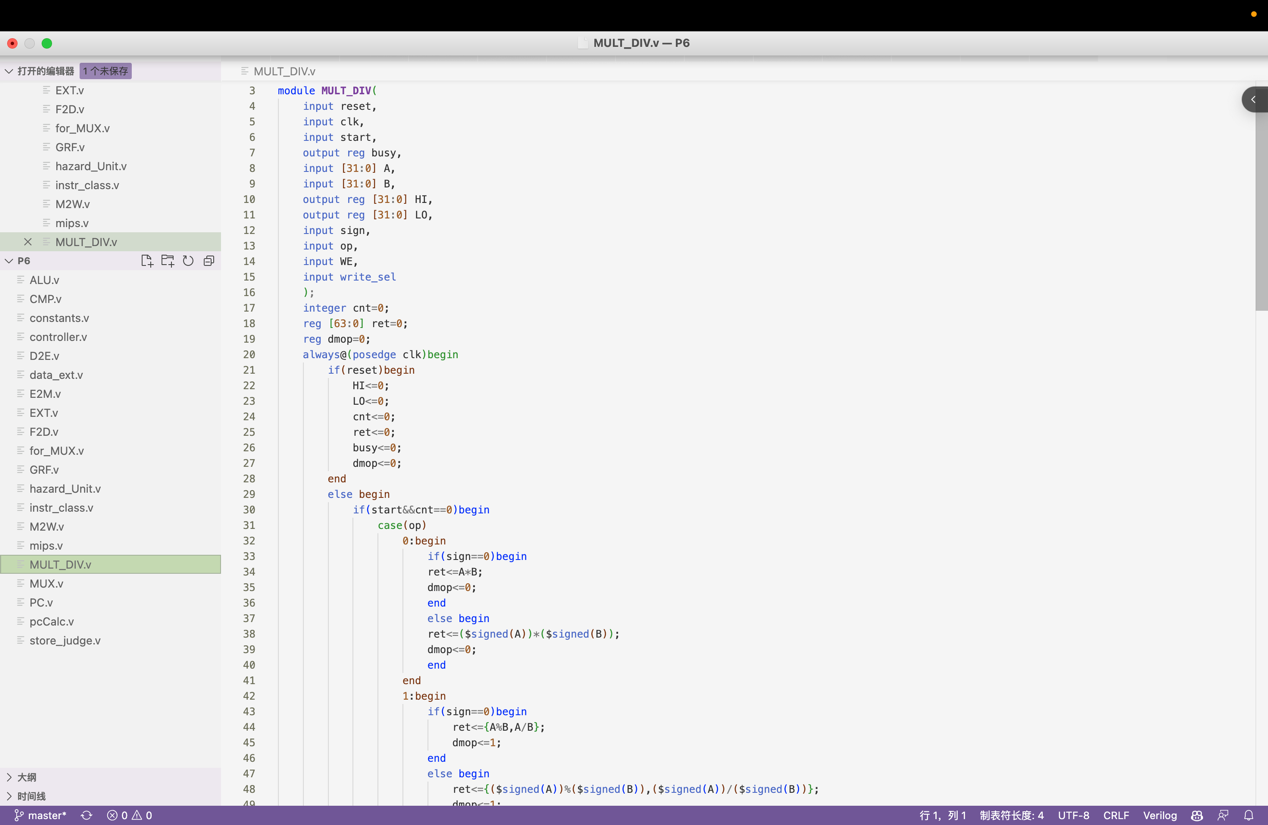
M2W



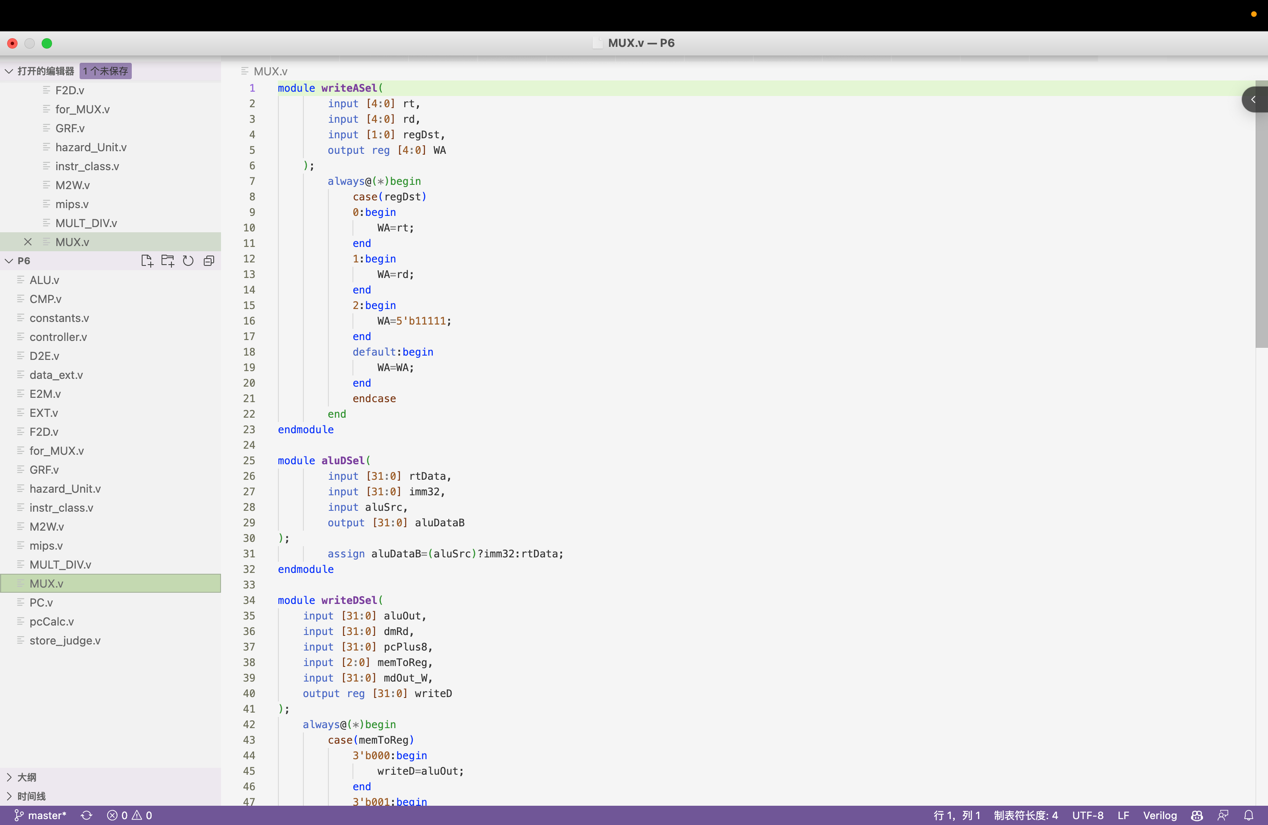
mips



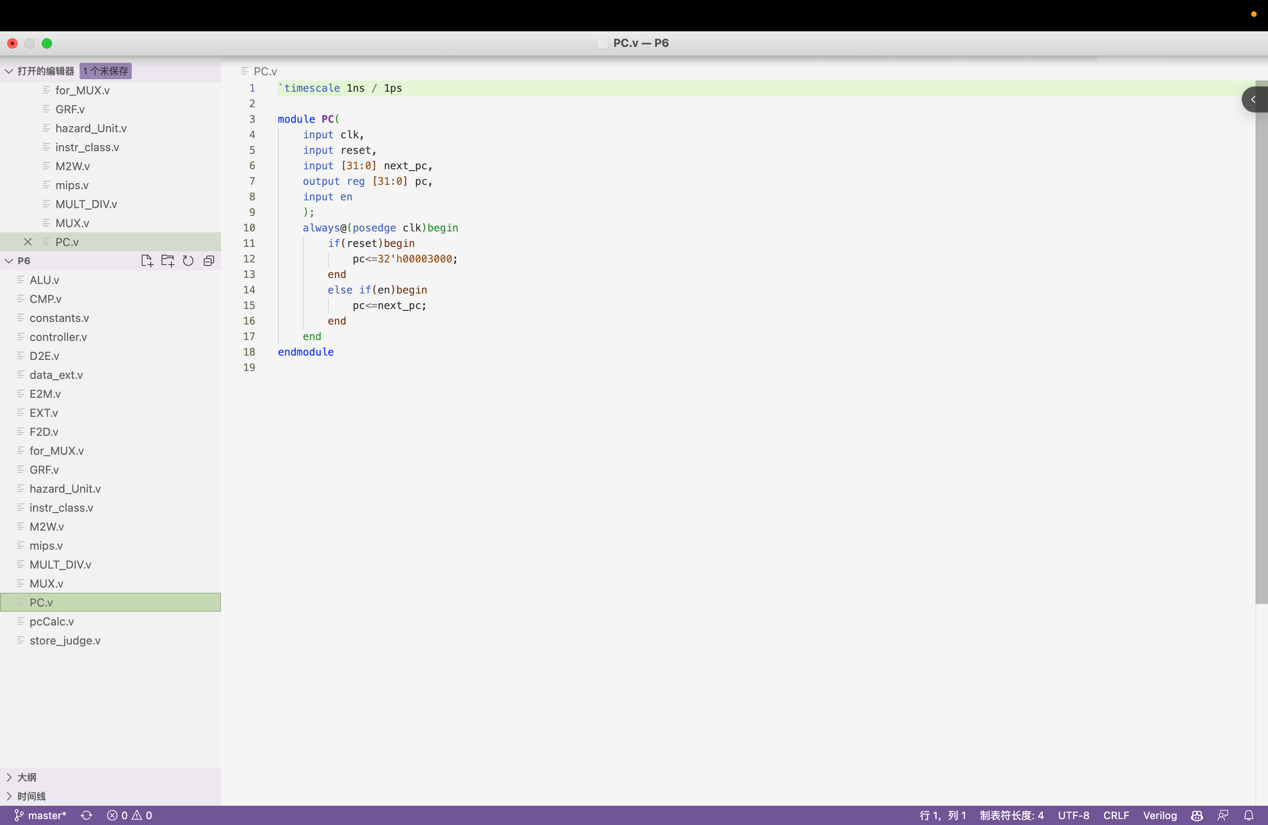
MULT\_DIV



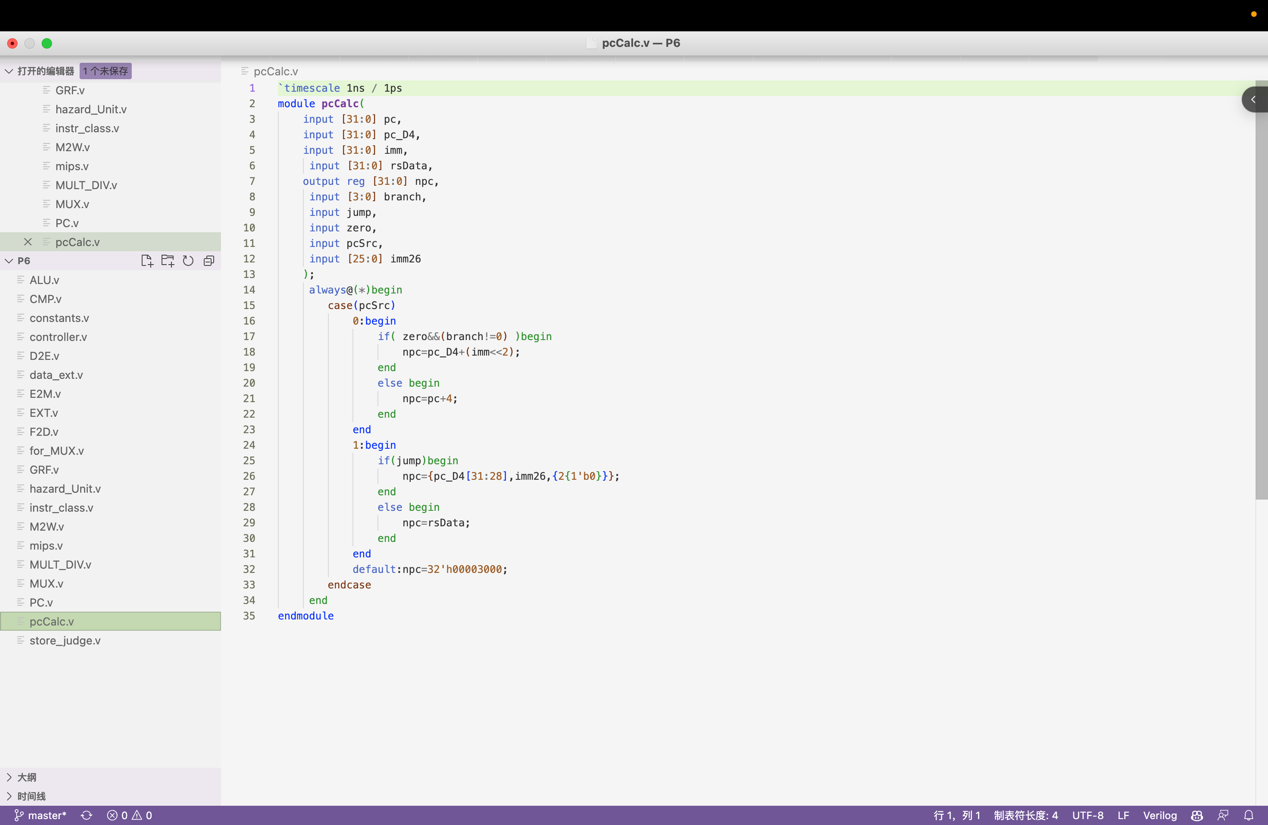
MUX



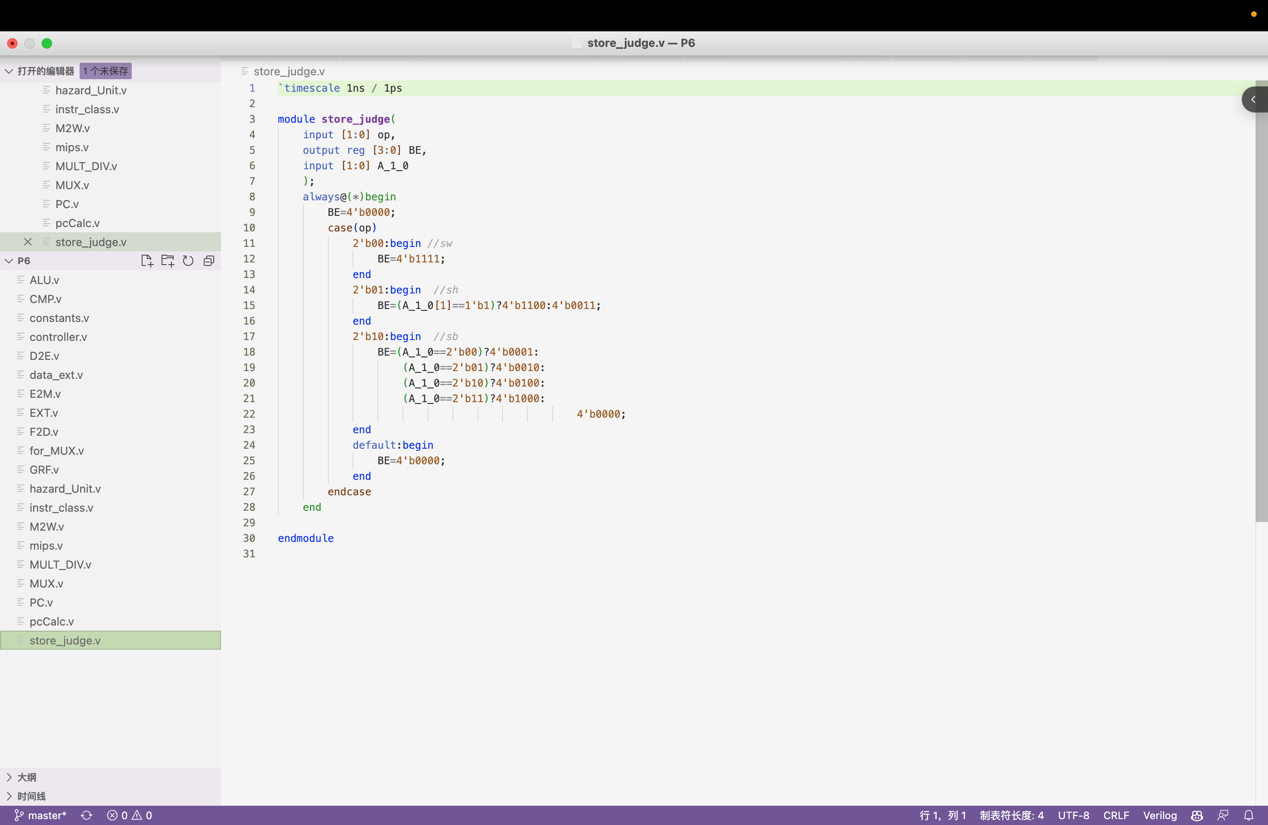
PC



pcCalc



Store\_judge



问答问题

1. 为什么需要有单独的乘除法部件而不是整合进 ALU？为何需要有独立的 HI、LO 寄存器？

解：因为乘法和除法操作的计算复杂度比加法和减法操作要高得多。乘法和除法操作需要更多的时间和更多的计算资源来完成，所以为了提高计算机的性能，通常会将它们作为单独的部件来实现。由于乘除法运算的结果通常会比加减法运算的结果要长，所以需要使用独立 HI 和 LO 寄存器来存储乘除法运算的完整结果。

1. 真实的流水线 CPU 是如何使用实现乘除法的？请查阅相关资料进行简单说明。

解：可以使用阵列乘法器

1. 请结合自己的实现分析，你是如何处理 Busy 信号带来的周期阻塞的？

解：需要一个深度为1旁路缓存暂存中间的数据

1. 请问采用字节使能信号的方式处理写指令有什么好处？（提示：从清晰性、统一性等角度考虑）

解：使用字节使能信号处理写指令的一个主要优点是，它可以有效地防止写操作时的干扰。这是因为字节使能信号会指示哪些字节正在被写入，因此其他部分就不会被干扰。这样可以避免写入过程中的冲突，提高系统的可靠性。另外，使用字节使能信号还可以提高写操作的效率，因为它允许在一个时钟周期内写入多个字节。

1. 请思考，我们在按字节读和按字节写时，实际从 DM 获得的数据和向 DM 写入的数据是否是一字节？在什么情况下我们按字节读和按字节写的效率会高于按字读和按字写呢？

解：可能是一字节，要看系统，在一般情况下，按字节读的效率都会比按字节写高，这是因为按字节读写操作可以直接访问内存中的每一个字节，而按字读写操作需要额外的处理步骤来解析字符串。

1. 为了对抗复杂性你采取了哪些抽象和规范手段？这些手段在译码和处理数据冲突的时候有什么样的特点与帮助？

解：在处理复杂性时，一种常用的抽象和规范手段是模型驱动的设计方法。这种方法的基本思想是通过构建模型来抽象和理解系统的复杂结构和功能，并利用模型来验证和优化系统的设计。在译码和处理数据冲突的时候，模型驱动的设计方法可以帮助系统开发人员更好地理解系统的工作原理，并有效地进行系统设计和优化。

1. 在本实验中你遇到了哪些不同指令类型组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？

解：具体看文件夹

1. 如果你是手动构造的样例，请说明构造策略，说明你的测试程序如何保证覆盖了所有需要测试的情况；如果你是完全随机生成的测试样例，请思考完全随机的测试程序有何不足之处；如果你在生成测试样例时采用了特殊的策略，比如构造连续数据冒险序列，请你描述一下你使用的策略如何结合了随机性达到强测的效果。

解：暂无