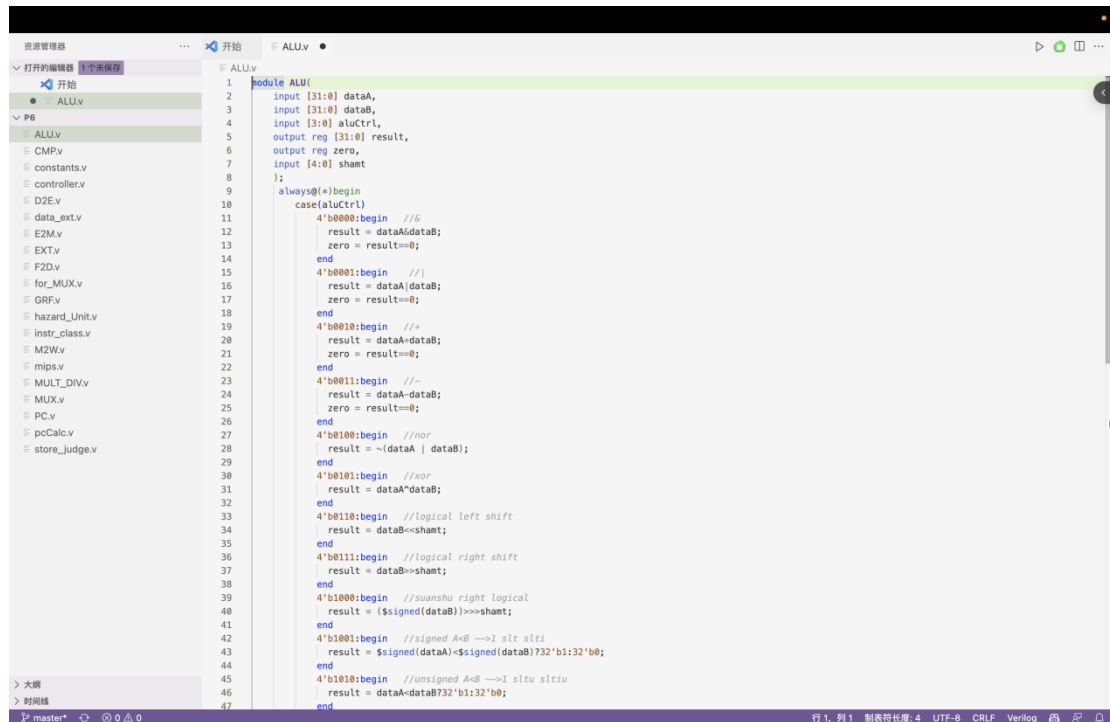


## P6 设计报告-陈伟杰

### 模块设计

#### ALU

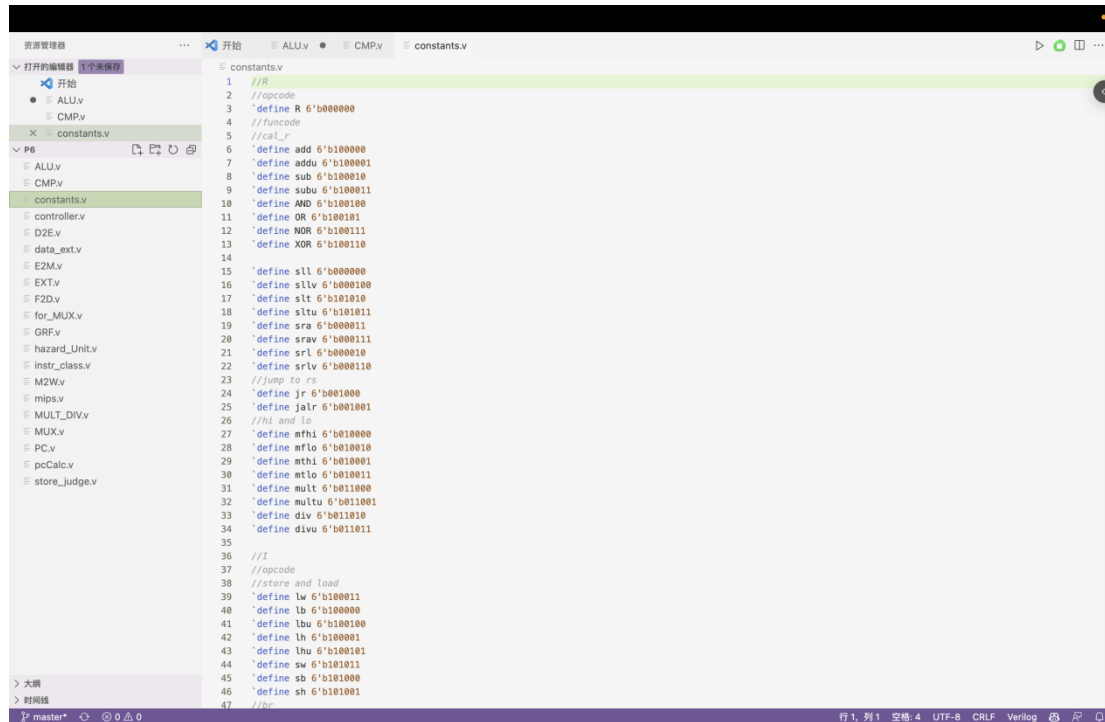


#### CMP

```
module CMP(  
    input [31:0] d1,  
    input [31:0] d2,  
    input [3:0] branch,  
    output eq,  
    output lez,  
    output ltz,  
    output gez,  
    output gtz,  
    output equal  
);  
assign eq=(d1==d2);  
assign lez=($signed(d1))<=0;  
assign ltz=($signed(d1))<0;  
assign gez=($signed(d1))>=0;  
assign gtz=($signed(d1))>0;  
assign equal=(eq&&branch==4'b0001)||(~eq&&branch==4'b0010)|| (gtz&&branch==4'b0011)  
            || (lez&&branch==4'b0100)|| (gez&&branch==4'b0101)|| (ltz&&branch==4'b0110);  
endmodule
```

## 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;
`bgeztz: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

The screenshot shows a Verilog IDE with the 'D2E.v' file open. The left sidebar displays a project tree with various modules, including 'D2E.v'. The main editor area contains the following Verilog code:

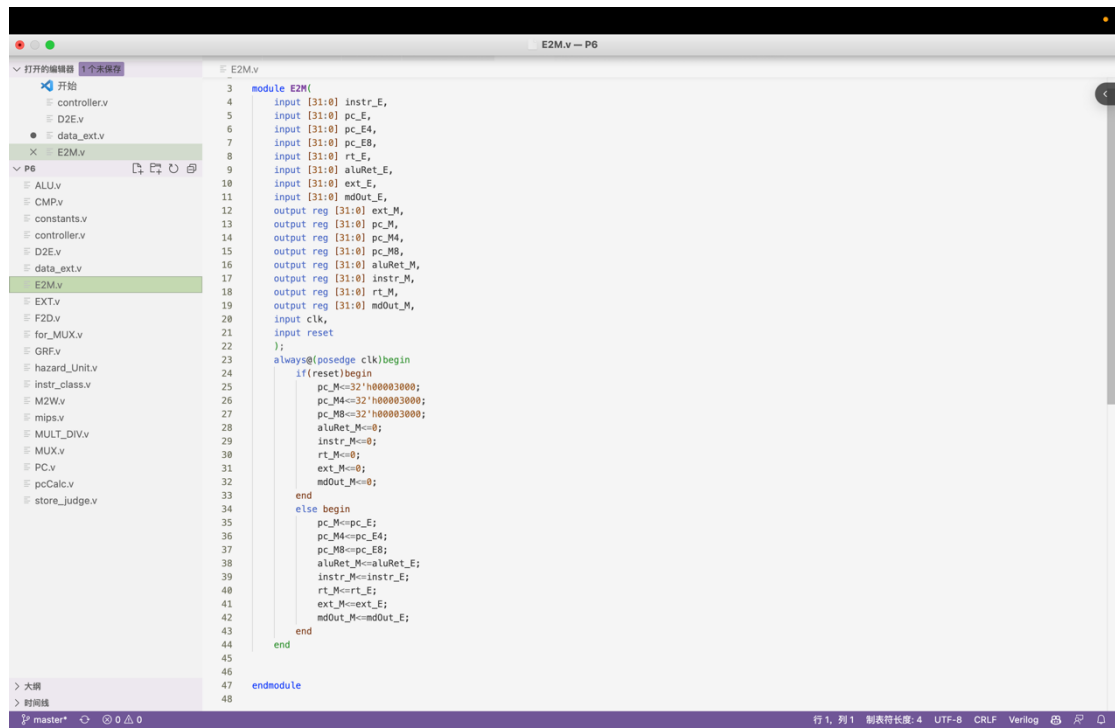
```
1 timescale 1ns / 1ps
2
3 module D2E(
4     input [31:0] instr_D,
5     input [31:0] pc_D,
6     input [31:0] pc_D4,
7     input [31:0] pc_E8,
8     output reg [31:0] pc_E,
9     output reg [31:0] pc_E4,
10    output reg [31:0] pc_E8,
11    input [31:0] grf_RD1,
12    input [31:0] grf_RD2,
13    input [31:0] ext_D,
14    output reg [31:0] ext_E,
15    output reg [31:0] instr_E,
16    output reg [31:0] rs_E,
17    output reg [31:0] rt_E,
18    input clk,
19    input reset
20 );
21 always@(posedge clk)begin
22     if(reset)begin
23         pc_E<=32'h00003000;
24         pc_E4<=32'h00003000;
25         pc_E8<=32'h00003000;
26         ext_E<=0;
27         instr_E<=0;
28         rs_E<=0;
29         rt_E<=0;
30     end
31     else begin
32         pc_E<=pc_D;
33         pc_E4<=pc_D4;
34         pc_E8<=pc_E8;
35         ext_E<=ext_D;
36         instr_E<=instr_D;
37         rs_E<=grf_RD1;
38         rt_E<=grf_RD2;
39     end
40 end
41 endmodule
```

## Data\_ext

The screenshot shows a Verilog IDE with the 'data\_ext.v' file open. The left sidebar displays a project tree with various modules, including 'data\_ext.v'. The main editor area contains the following Verilog code:

```
1 module data_ext(
2     input [1:0] A,
3     input [31:0] Din,
4     input [2:0] op,
5     output reg [31:0] Dout
6 );
7 always@(*)begin
8     case(op)
9         3'b000:begin
10             Dout=Din;
11         end
12         3'b001:begin
13             case(A)
14                 2'b00:Dout={24(1'b0),Din[7:0]};
15                 2'b01:Dout={24(1'b0),Din[15:8]};
16                 2'b10:Dout={24(1'b0),Din[23:16]};
17                 2'b11:Dout={24(1'b0),Din[31:24]};
18                 default:Dout=0;
19             endcase
20         end
21         3'b010:begin
22             case(A)
23                 2'b00:Dout={24(Din[7]),Din[7:0]};
24                 2'b01:Dout={24(Din[15]),Din[15:8]};
25                 2'b10:Dout={24(Din[23]),Din[23:16]};
26                 2'b11:Dout={24(Din[31]),Din[31:24]};
27                 default:Dout=0;
28             endcase
29         end
30         3'b011:begin
31             case(A[1])
32                 1'b0:Dout={16(1'b0),Din[15:0]};
33                 1'b1:Dout={16(1'b0),Din[31:16]};
34                 default:Dout=0;
35             endcase
36         end
37         3'b100:begin
38             case(A[1])
39                 1'b0:Dout={16(Din[15]),Din[15:0]};
40                 1'b1:Dout={16(Din[31]),Din[31:16]};
41                 default:Dout=0;
42             endcase
43         end
44         default:Dout=0;
45     endcase
46 end
47 endmodule
```

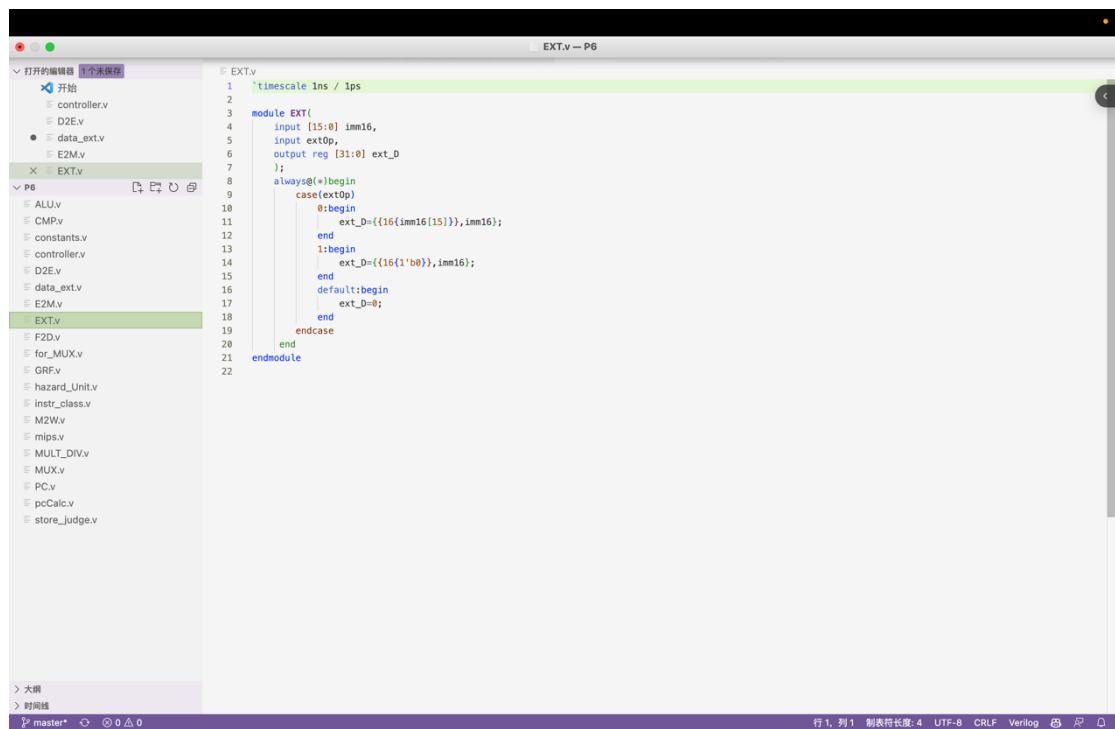
## E2M



The screenshot shows the E2M.v Verilog code in a text editor. The code defines a module E2M with inputs instr\_E, pc\_E, pc\_E4, pc\_E8, rt\_E, aluRet\_E, ext\_E, mdOut\_E, and outputs pc\_M, pc\_M4, pc\_M8, aluRet\_M, instr\_M, rt\_M, mdOut\_M. It includes a reset input and a clock input. The code is structured with an always block for the clock and an if block for the reset. The reset block initializes registers to zero. The clock block updates registers based on the inputs.

```
3 module E2M(
4     input [31:0] instr_E,
5     input [31:0] pc_E,
6     input [31:0] pc_E4,
7     input [31:0] pc_E8,
8     input [31:0] rt_E,
9     input [31:0] aluRet_E,
10    input [31:0] ext_E,
11    input [31:0] mdOut_E,
12    output reg [31:0] ext_M,
13    output reg [31:0] pc_M,
14    output reg [31:0] pc_M4,
15    output reg [31:0] pc_M8,
16    output reg [31:0] aluRet_M,
17    output reg [31:0] instr_M,
18    output reg [31:0] rt_M,
19    output reg [31:0] mdOut_M,
20    input clk,
21    input reset
22 );
23 always@(posedge clk)begin
24     if(reset)begin
25         pc_M<=32'h00003000;
26         pc_M4<=32'h00003000;
27         pc_M8<=32'h00003000;
28         aluRet_M<=0;
29         instr_M<=0;
30         rt_M<=0;
31         ext_M<=0;
32         mdOut_M<=0;
33     end
34     else begin
35         pc_M<=pc_E;
36         pc_M4<=pc_E4;
37         pc_M8<=pc_E8;
38         aluRet_M<=aluRet_E;
39         instr_M<=instr_E;
40         rt_M<=rt_E;
41         ext_M<=ext_E;
42         mdOut_M<=mdOut_E;
43     end
44 end
45
46
47 endmodule
48
```

## EXT

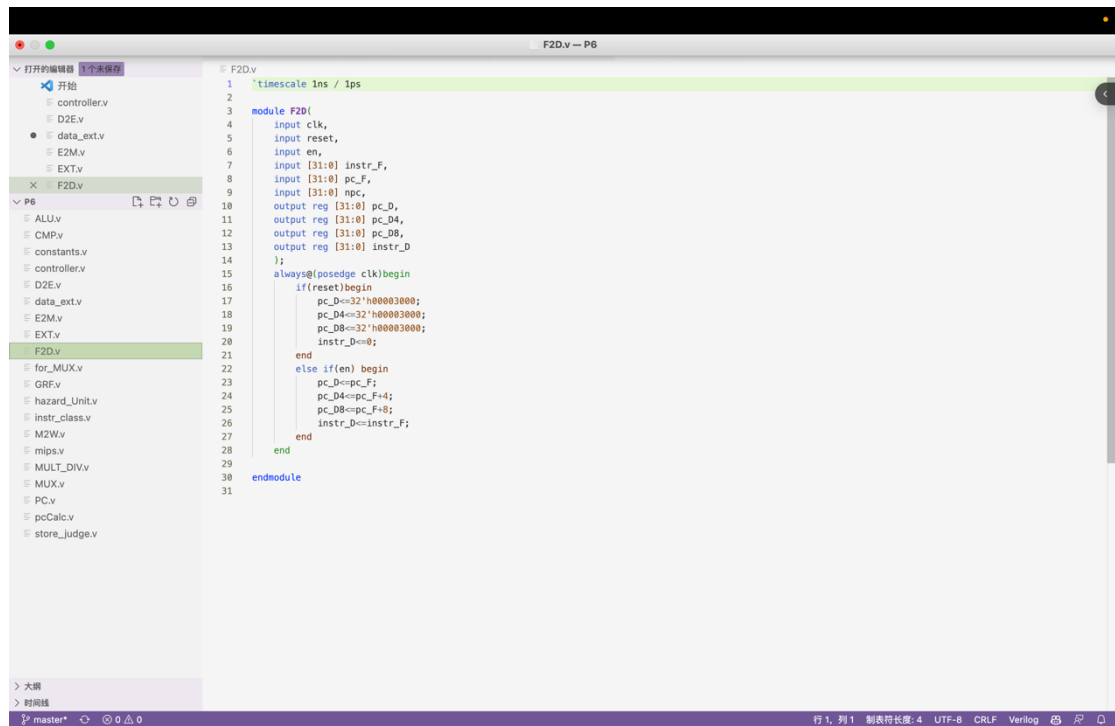


The screenshot shows the EXT.v Verilog code in a text editor. The code defines a module EXT with inputs imm16 and ext0p, and an output ext\_D. It includes a timescale directive and a case statement for the ext0p input. The case statement updates ext\_D based on the value of ext0p.

```
1 `timescale 1ns / 1ps
2
3 module EXT(
4     input [15:0] imm16,
5     input ext0p,
6     output reg [31:0] ext_D
7 );
8 always@(+)begin
9     case(ext0p)
10         0:begin
11             ext_D<={{16{imm16[15]}},imm16};
12         end
13         1:begin
14             ext_D<={{16{1'b0}},imm16};
15         end
16         default:begin
17             ext_D<=0;
18         end
19     endcase
20 end
21 endmodule
22
```



## F2D

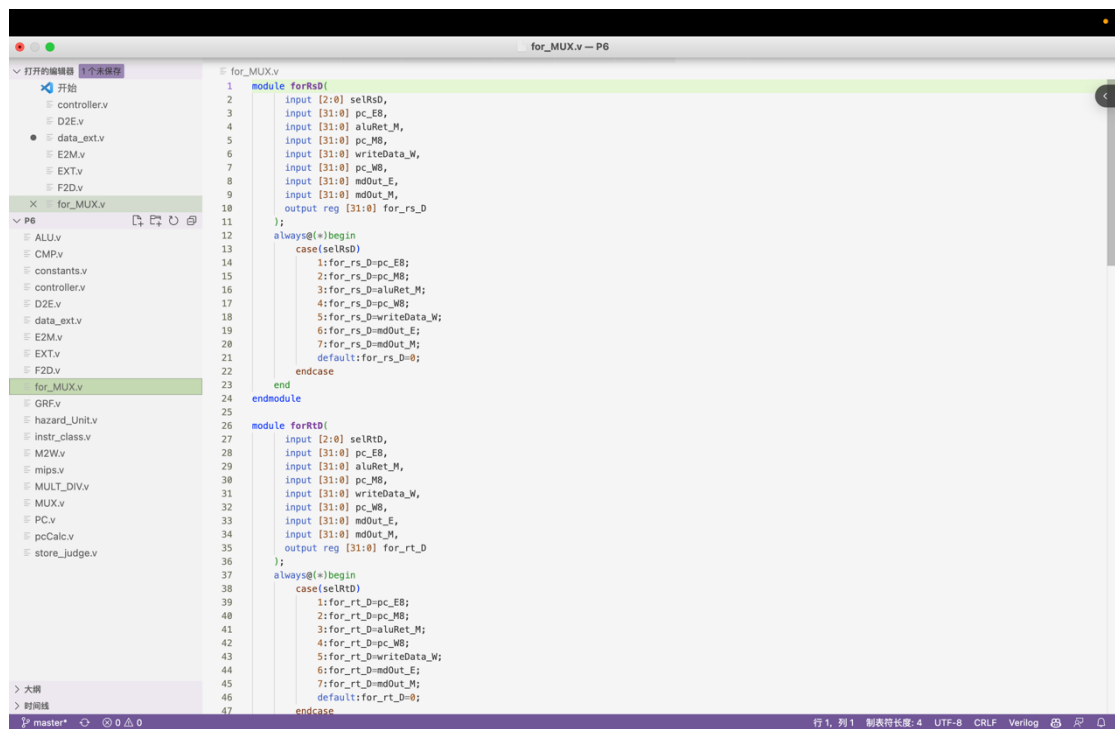


The screenshot shows the F2D module in a Verilog text editor. The left sidebar displays a project tree with files like controller.v, D2E.v, data\_ext.v, E2M.v, EXT.v, and F2D.v. The main editor window shows the F2D.v file with the following code:

```
1 timescale 1ns / 1ps
2
3 module F2D(
4     input clk,
5     input reset,
6     input en,
7     input [31:0] instr_F,
8     input [31:0] pc_F,
9     input [31:0] npc,
10    output reg [31:0] pc_D,
11    output reg [31:0] pc_D4,
12    output reg [31:0] pc_D8,
13    output reg [31:0] instr_D
14);
15    always@(posedge clk)begin
16        if(reset)begin
17            pc_D<=32'h00003000;
18            pc_D4<=32'h00003000;
19            pc_D8<=32'h00003000;
20            instr_D<=0;
21        end
22        else if(en) begin
23            pc_D<=pc_F;
24            pc_D4<=pc_F+4;
25            pc_D8<=pc_F+8;
26            instr_D<=instr_F;
27        end
28    end
29 endmodule
30
31
```

The status bar at the bottom indicates the file is master\*, the editor is in UTF-8 CRLF Verilog mode, and the cursor is at line 1, column 1.

## For-mux

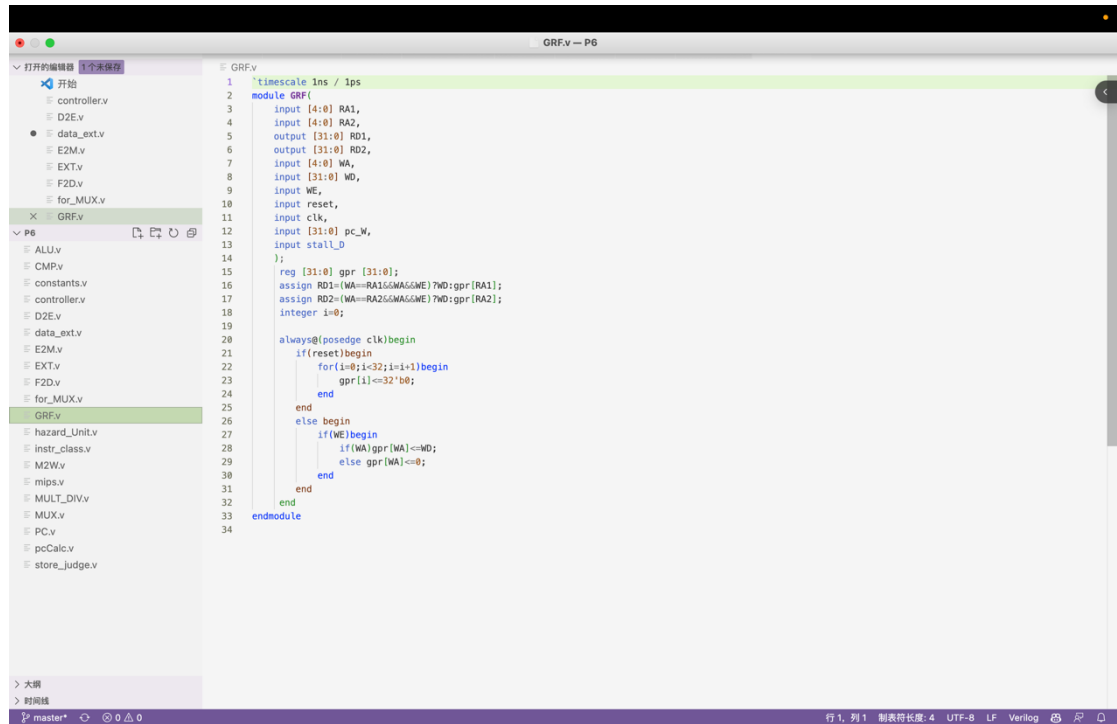


The screenshot shows the for\_MUX.v module in a Verilog text editor. The left sidebar displays a project tree with files like controller.v, D2E.v, data\_ext.v, E2M.v, EXT.v, F2D.v, and for\_MUX.v. The main editor window shows the for\_MUX.v file with the following code:

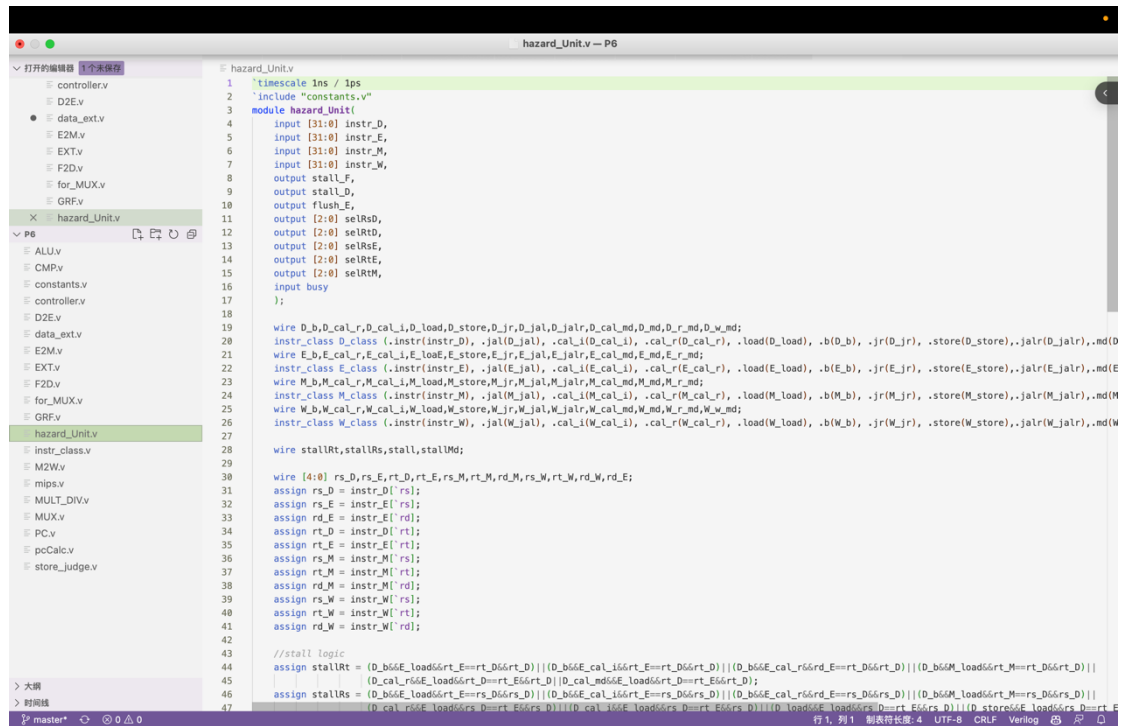
```
1 module forRsD(
2     input [2:0] selRsD,
3     input [31:0] pc_E8,
4     input [31:0] aluRet_M,
5     input [31:0] pc_M8,
6     input [31:0] writeData_W,
7     input [31:0] pc_W8,
8     input [31:0] mdOut_E,
9     input [31:0] mdOut_M,
10    output reg [31:0] for_rs_D
11);
12    always@(*)begin
13        case(selRsD)
14            1:for_rs_D=pc_E8;
15            2:for_rs_D=pc_M8;
16            3:for_rs_D=aluRet_M;
17            4:for_rs_D=pc_W8;
18            5:for_rs_D=writeData_W;
19            6:for_rs_D=mdOut_E;
20            7:for_rs_D=mdOut_M;
21            default:for_rs_D=0;
22        endcase
23    end
24 endmodule
25
26 module forRtD(
27     input [2:0] selRtD,
28     input [31:0] pc_E8,
29     input [31:0] aluRet_M,
30     input [31:0] pc_M8,
31     input [31:0] writeData_W,
32     input [31:0] pc_W8,
33     input [31:0] mdOut_E,
34     input [31:0] mdOut_M,
35     output reg [31:0] for_rt_D
36);
37    always@(*)begin
38        case(selRtD)
39            1:for_rt_D=pc_E8;
40            2:for_rt_D=pc_M8;
41            3:for_rt_D=aluRet_M;
42            4:for_rt_D=pc_W8;
43            5:for_rt_D=writeData_W;
44            6:for_rt_D=mdOut_E;
45            7:for_rt_D=mdOut_M;
46            default:for_rt_D=0;
47        endcase
48    end
49 endmodule
```

The status bar at the bottom indicates the file is master\*, the editor is in UTF-8 CRLF Verilog mode, and the cursor is at line 1, column 1.

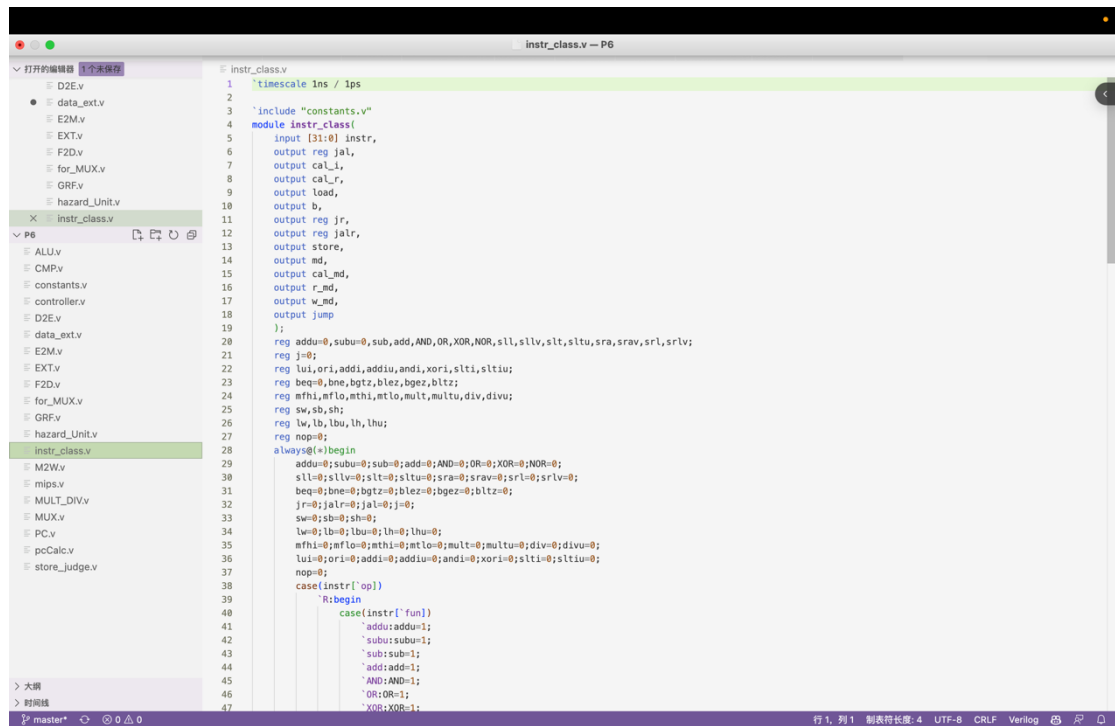
## GRF



## Hazard\_Unit

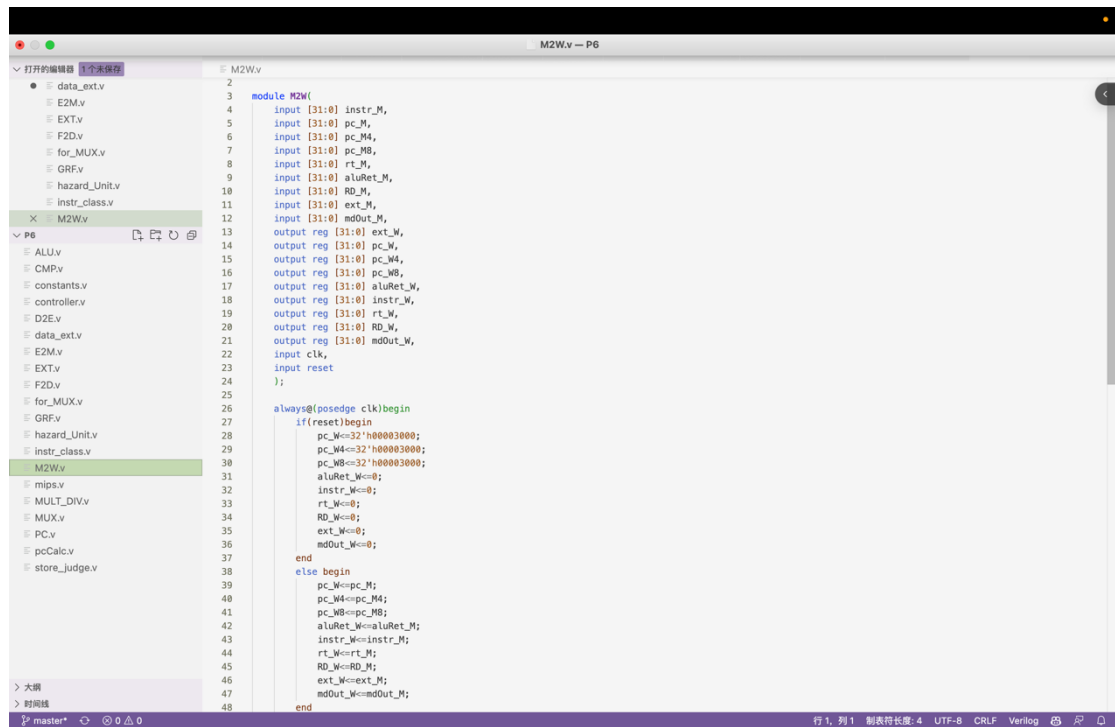


## Instr\_class



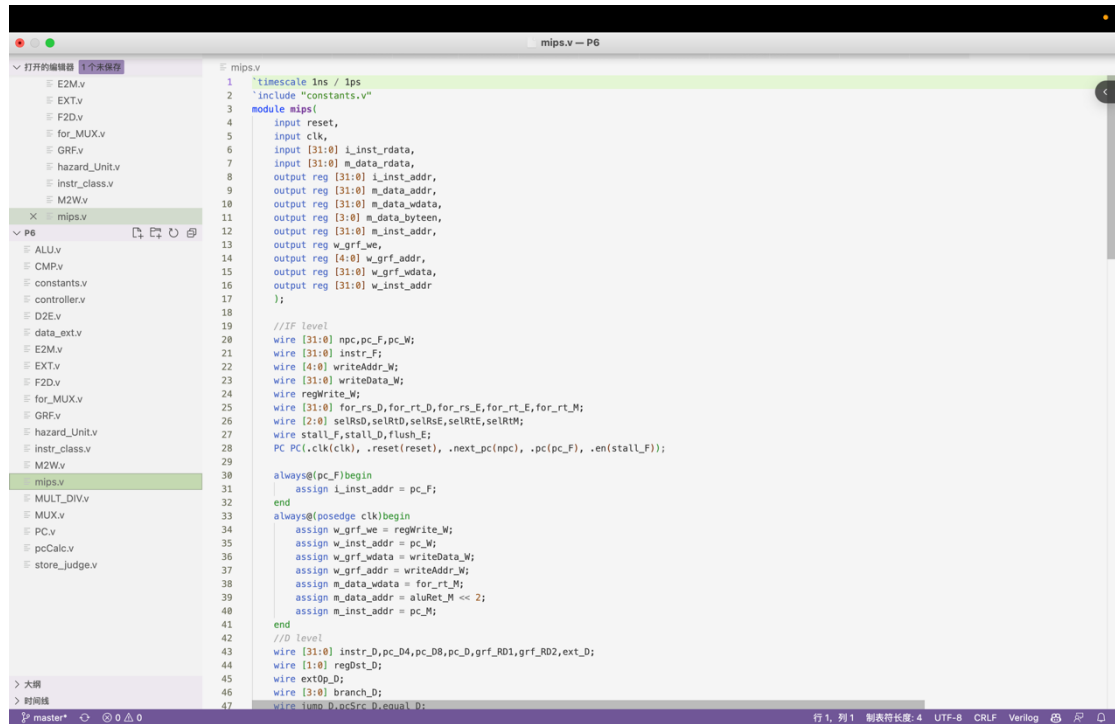
```
1 `timescale 1ns / 1ps
2
3 `include "constants.v"
4 module instr_class
5     input [31:0] instr,
6     output reg jal,
7     output cal_l,
8     output cal_r,
9     output load,
10    output b,
11    output reg jr,
12    output reg jalr,
13    output store,
14    output md,
15    output cal_md,
16    output r_md,
17    output w_md,
18    output jump
19 );
20 reg addu=0,subu=0,sub,add,AND,OR,XOR,NOR,sll,sllv,slt,sltu,sra,srav,srl,srlv;
21 reg j=0;
22 reg lui,ori,addi,addiu,andi,xori,slti,sltiu;
23 reg beq=0,bne,bgtz,blez,bgez,bltz;
24 reg mfh,mflo,mthi,mto,mult,multu,div,divu;
25 reg sw,sh,sh;
26 reg lw,lb,lbu,lb,lbh;
27 reg nop=0;
28
29 always@(*)begin
30     addu=0;subu=0;sub=0;add=0;AND=0;OR=0;XOR=0;NOR=0;
31     sll=0;sllv=0;slt=0;sltu=0;sra=0;srav=0;srl=0;srlv=0;
32     beq=0;bne=0;bgtz=0;blez=0;bgez=0;bltz=0;
33     jr=0;jalr=0;jal=0;j=0;
34     sw=0;sh=0;sh=0;
35     lw=0;lb=0;lbu=0;lb=0;lbh=0;
36     mfh=0;mflo=0;mthi=0;mto=0;mult=0;multu=0;div=0;divu=0;
37     lui=0;ori=0;addi=0;addiu=0;and=0;xori=0;slti=0;sltiu=0;
38     nop=0;
39     case(instr[7:0])
40     `R:begin
41         case(instr[4:0])
42             `addu: addu=1;
43             `subu: subu=1;
44             `sub: sub=1;
45             `add: add=1;
46             `AND: AND=1;
47             `OR: OR=1;
48             `XOR: XOR=1;
```

## M2W

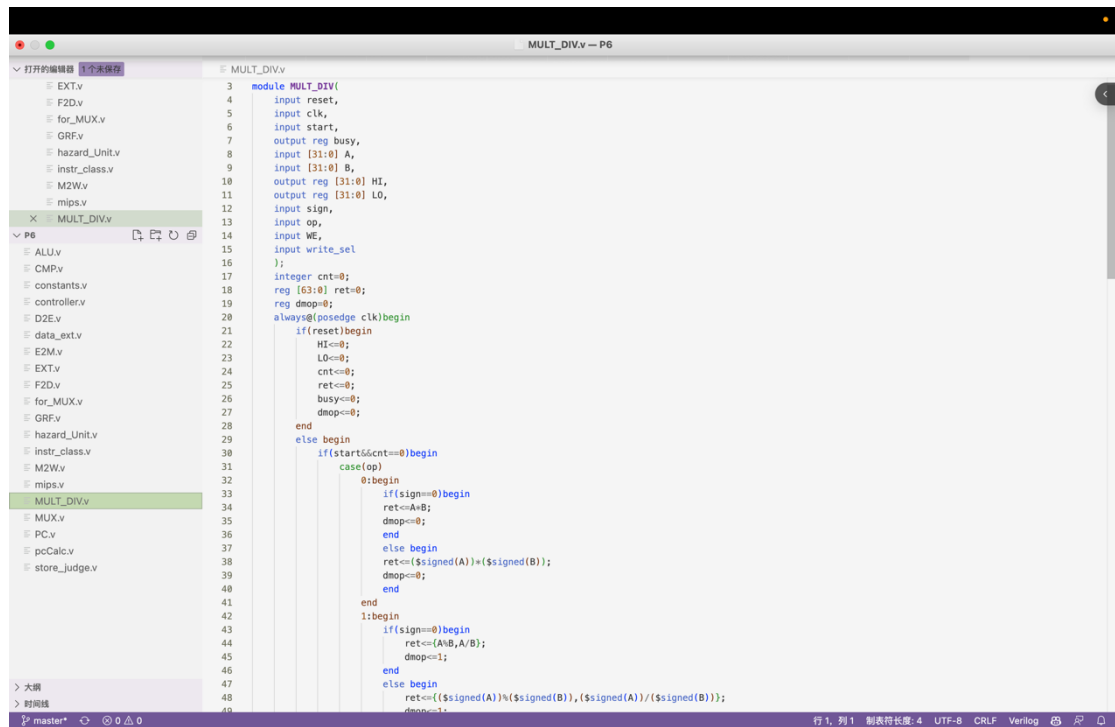


```
1 module M2W
2     input [31:0] instr_M,
3     input [31:0] pc_M,
4     input [31:0] pc_M4,
5     input [31:0] pc_M8,
6     input [31:0] rt_M,
7     input [31:0] aluRet_M,
8     input [31:0] RD_M,
9     input [31:0] ext_M,
10    input [31:0] mdOut_M,
11    output reg [31:0] ext_W,
12    output reg [31:0] pc_W,
13    output reg [31:0] pc_M4,
14    output reg [31:0] pc_W8,
15    output reg [31:0] aluRet_W,
16    output reg [31:0] instr_W,
17    output reg [31:0] rt_W,
18    output reg [31:0] RD_W,
19    output reg [31:0] mdOut_W,
20    input clk,
21    input reset
22 );
23
24 always@(posedge clk)begin
25     if(reset)begin
26         pc_W<=32'h00003000;
27         pc_M4<=32'h00003000;
28         pc_W8<=32'h00003000;
29         aluRet_W<=0;
30         instr_W<=0;
31         rt_W<=0;
32         RD_W<=0;
33         ext_W<=0;
34         mdOut_W<=0;
35     end
36     else begin
37         pc_W<=pc_M;
38         pc_M4<=pc_M4;
39         pc_W8<=pc_M8;
40         aluRet_W<=aluRet_M;
41         instr_W<=instr_M;
42         rt_W<=rt_M;
43         RD_W<=RD_M;
44         ext_W<=ext_M;
45         mdOut_W<=mdOut_M;
46     end
47 end
```

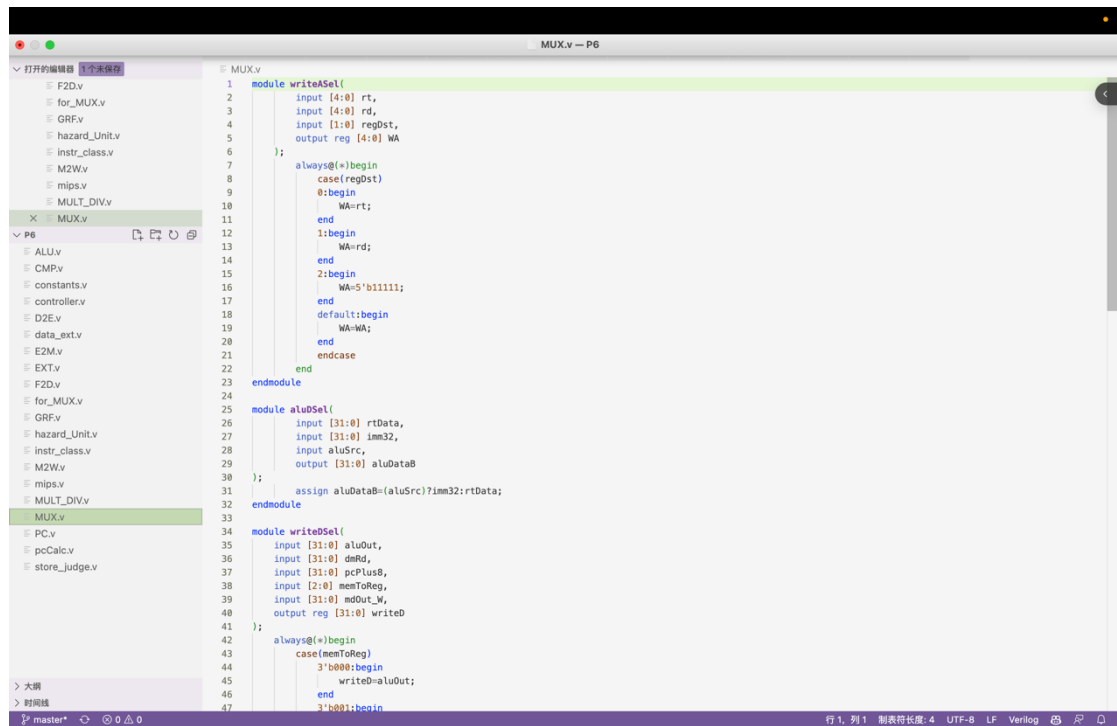
## mips



## MULT\_DIV

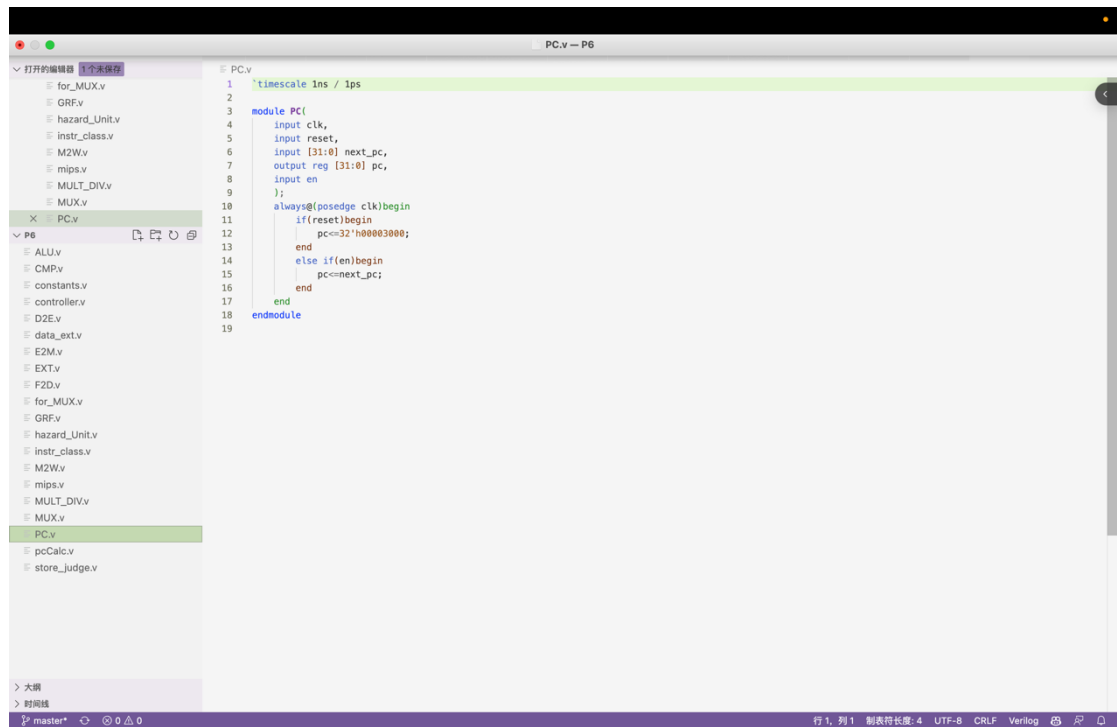


## MUX



```
1 module writeAtoSel(
2     input [4:0] rt,
3     input [4:0] rd,
4     input [1:0] regDst,
5     output reg [4:0] WA
6 );
7     always@(*)begin
8         case(regDst)
9             0:begin
10                 WA=rt;
11             end
12             1:begin
13                 WA=rd;
14             end
15             2:begin
16                 WA=5'b11111;
17             end
18             default:begin
19                 WA=WA;
20             end
21         endcase
22     end
23 endmodule
24
25 module aluSel(
26     input [31:0] rtData,
27     input [31:0] imm32,
28     input aluSrc,
29     output [31:0] aluData8
30 );
31     assign aluData8=(aluSrc)?imm32:rtData;
32 endmodule
33
34 module writeBtoSel(
35     input [31:0] aluOut,
36     input [31:0] memRd,
37     input [31:0] pcPlus8,
38     input [2:0] memToReg,
39     input [31:0] mdoOut_W,
40     output reg [31:0] wrote0
41 );
42     always@(*)begin
43         case(memToReg)
44             3'b000:begin
45                 wrote0=aluOut;
46             end
47             3'b001:begin
```

## PC



```
1 `timescale 1ns / 1ps
2
3 module PC(
4     input clk,
5     input reset,
6     input [31:0] next_pc,
7     output reg [31:0] pc,
8     input en
9 );
10     always@(posedge clk)begin
11         if(reset)begin
12             pc<=32'h00003000;
13         end
14         else if(en)begin
15             pc<=next_pc;
16         end
17     end
18 endmodule
19
```

pcCalc

打开的编辑器1个未保存

GRF.v

hazard\_Unit.v

instr\_class.v

M2W.v

mips.v

MULT\_DIV.v

MUX.v

PC.v

pcCalc.v

store\_judge.v

pcCalc.v

1 timescale 1ns / 1ps

2 module pcCalc(

3 input [31:0] pc,

4 input [31:0] pc\_D4,

5 input [31:0] imm,

6 input [31:0] rsData,

7 output reg [31:0] npc,

8 input [3:0] branch,

9 input jump,

10 input zero,

11 input pcSrc,

12 input [25:0] imm26

13 );

14 always@(\*)begin

15 case(pcSrc)

16 0:begin

17 if( zero&&(branch!=0) )begin

18 npc=pc\_D4+(imm<<2);

19 end

20 else begin

21 npc=pc+4;

22 end

23 end

24 1:begin

25 if(jump)begin

26 npc={pc\_D4[31:28],imm26,{2{1'b0}}};

27 end

28 else begin

29 npc=rsData;

30 end

31 end

32 default:npc=32'h00003000;

33 endcase

34 end

35 endmodule

> 大纲

> 时间线

master\* 0 0 0

行 1, 列 1 制表符长度: 4 UTF-8 LF Verilog

Store\_judge

打开的编辑器1个未保存

hazard\_Unit.v

instr\_class.v

M2W.v

mips.v

MULT\_DIV.v

MUX.v

PC.v

pcCalc.v

store\_judge.v

store\_judge.v

1 timescale 1ns / 1ps

2

3 module store\_judge(

4 input [1:0] op,

5 output reg [3:0] BE,

6 input [1:0] A\_1\_0

7 );

8 always@(\*)begin

9 BE=4'b0000;

10 case(op)

11 2'b00:begin //sw

12 BE=4'b1111;

13 end

14 2'b01:begin //sh

15 BE=(A\_1\_0[1]==1'b1)?4'b1100:4'b0011;

16 end

17 2'b10:begin //sb

18 BE=(A\_1\_0==2'b00)?4'b0001:

19 (A\_1\_0==2'b01)?4'b0010:

20 (A\_1\_0==2'b10)?4'b0100:

21 (A\_1\_0==2'b11)?4'b1000:

22 4'b0000;

23 end

24 default:begin

25 BE=4'b0000;

26 end

27 endcase

28 end

29 endmodule

30

31

> 大纲

> 时间线

master\* 0 0 0

行 1, 列 1 制表符长度: 4 UTF-8 CRLF Verilog

## 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。

2. 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

3. 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

4. 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。

5. 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。

6. 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。

7. 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

8. 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