同濟大學
TONGJI UNIVERSITY

计算机系统结构课程实验
总结报告

实验题目：动态流水线设计与性能定量分析

学号：1951444

姓名：林佳奕

指导教师：秦国锋

日期：2021-12-10

一、实验环境部署与硬件配置说明

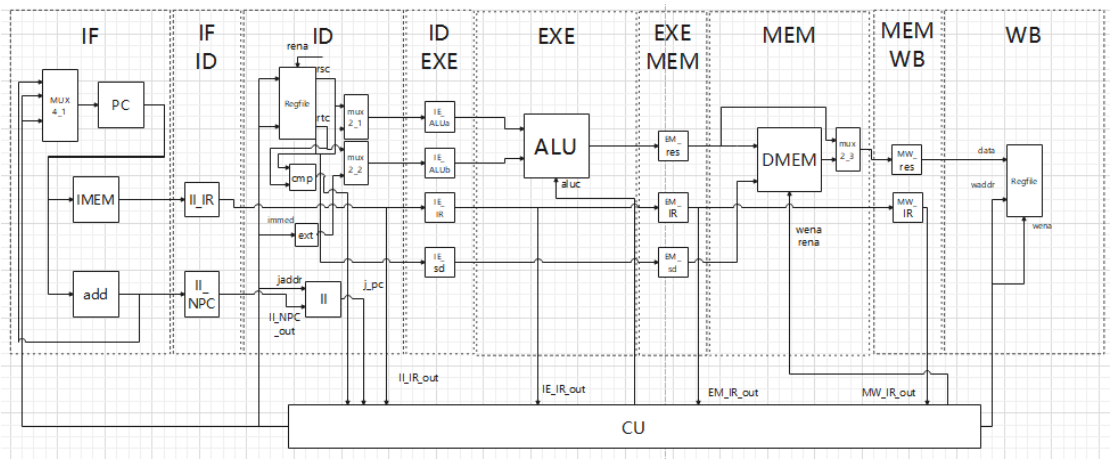
本地操作系统	Windows 10
开发环境	Mars4.5（验证程序） Vivado2016（Verilog）
硬件配置	Xilinx N4 开发板
指令集	MIPS32

二、实验的总体结构

实验要求完成至少 31 条 MIPS 指令的动态流水线 CPU 设计，并支持中断。在 CPU 运行验证程序的过程中，由按键或拨动开关产生一个暂停的中断，再次按键或拨动开关结束中断，继续运行后续的运算，并在数码管上动态显示运算值。

1. 动态流水线的总体结构

本实验中动态流水线的数据通路图如下：



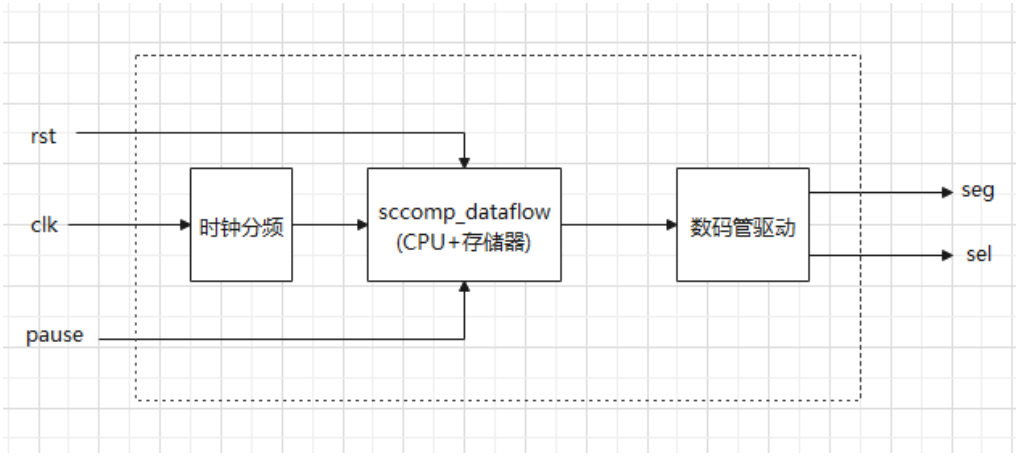
类似静态流水线，这里同样是分为了 5 段，分别为 ID，IF，EXE，MEM，WB。段间流水寄存器用来保存上一级运算结果，并传递给下一级。

2. 顶层模块设计

本实验为验证一数组运算模型，需要检查某一寄存器的值。要求能够从外部

实现中断暂停流水线。

因此顶层模块的设计如下：



此模块实现的功能是启动流水线 CPU 运算，并将结果展示在数码管上。

为保证本实验运行结果正确，我对时钟信号进行了分频处理，将时钟频率降低为原来的 1/3，经过下板测试可得到正确结果。

三、总体架构部件的解释说明

1. 动态流水线总体结构部件的解释说明

(1) IF 段部件说明

IF 段涉及到的部件包括：PC 寄存器，IMEM（指令存储器），NPC（加法器，对 PC+4），一个 MUX2 和一个 MUX4（用于选择写入 PC 的值）。

此部分实现的功能是：取指，修改 PC（正常 PC+4，分支成功和跳转 PC 为新地址）。

修改 PC 寄存器时，需要考虑多种情况。正常情况下，PC 写入 NPC 的值；当流水线暂停时，PC 保持不变；当 ID 段为跳转的时候，PC 写入跳转地址；当 ID 段遇到中断指令时，PC 值写入中断向量（本实验中，中断向量为 0x00400004）。

(2) ID 段部件说明

ID 段涉及到的部件包括：通用寄存器堆 Regfiles，数据扩展器 ext5,ext16（5 位，16 位），数据比较器 cmp，地址合成器 II（用于合成 jump 类型指令的跳转地址），加法器 add（用于计算分支跳转地址），和若干多路选择器。

此段实现的功能为：译码，产生操作数，并对跳转分支指令进行处理。

对于运算类指令，本实验的操作数来源包括寄存器和立即数。对于寄存器操

作数，向寄存器读数即可。对于立即数，需要进行位扩展，并根据指令类型决定有无符号扩展；对于访存指令，需要产生访存地址，写存指令还要找到写入数据（来自寄存器），访存地址为寄存器基址+立即数位移，其处理类似运算类指令。

由于本实验中对于相关产生的冲突的处理方法是数据前推，数据前推的结果不仅要送入流水寄存器，还要送到数据比较器和数据扩展器前（分支需要用到寄存器操作数，sllv 等移位指令也要用到），因此这里会用到一些数据扩展器，其选择信号来源于控制器。

(3) EXE 段部件说明

EXE 段涉及到的部件包括：ALU，乘法器，和多路选择器。

此段实现的功能是：进行算术逻辑运算。

由于本实验中涉及到了乘法运算，因此单独的 ALU 不再适用，需要单独使用一个乘法器。本实验使用了乘法器为纯组合逻辑，因此 1 个周期就能运算完毕，不需要流水线的暂停。

此段的运算结果会前推至 ID 段。

(4) MEM 段部件说明

MEM 段涉及到的部件包括：数据存储器，多路选择器。

此段实现的功能是：访存。

在本实验中，有些指令不访存，也会流入此段，因此在写入流水寄存器前，需要用多路选择器选择存储器数据或者是上一级流水线传来的数据。

涉及到的访存指令包括 load 和 store，其控制信号由控制器产生。

此段同样会将结果前推至 ID 段。

(5) WB 段部件说明

WB 段涉及到的部件包括：通用寄存器堆

此段涉及到的功能包括：写回寄存器。

有些指令不需要写回寄存器堆，但也会流过此段。对此的处理只需要在控制器中令读信号不使能即可。

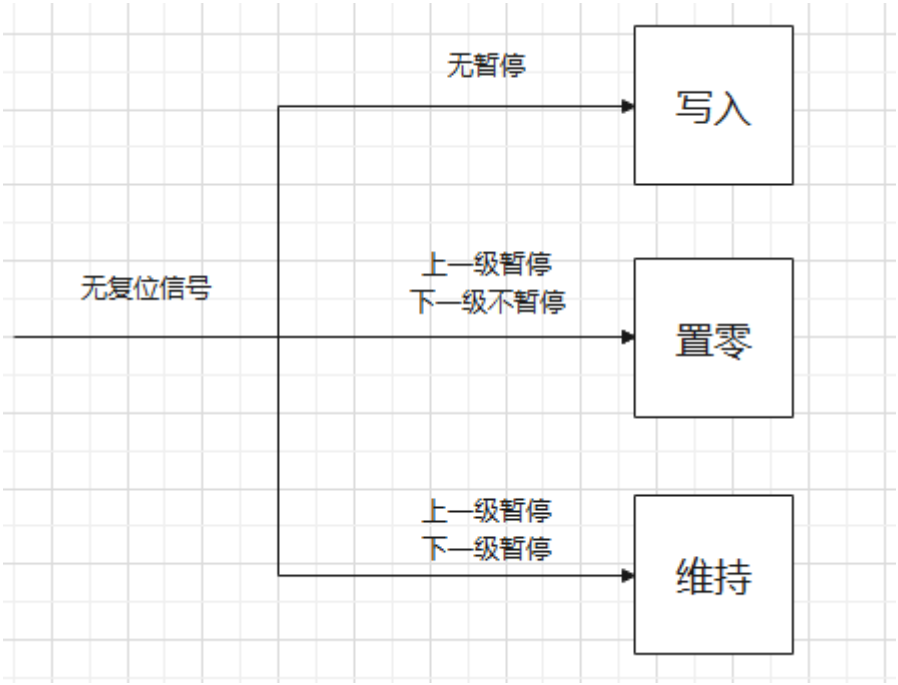
此段的前推在寄存器堆内部实现，即若读地址与写地址相同，则读出的内容为写入的内容，而非寄存器旧值。

(6) 流水寄存器

流水寄存器的作用是保存上一段流水线的结果，并提供给下一段。

本实验中，除 IF 段以外，每段都需要段间流水寄存器提供指令来产生控制信号并进行响应操作，因此指令也需要保存在流水寄存器中。由于本实验使用的指令集中，当指令机器码为 32 位 0 时，此指令不会产生任何的改变，因此需要作废某一条指令时（控制相关时使用），可以直接向保存指令的流水寄存器写入 32 位 0，而不需要进行其他处理。

下面是流水寄存器对不同情况的处理



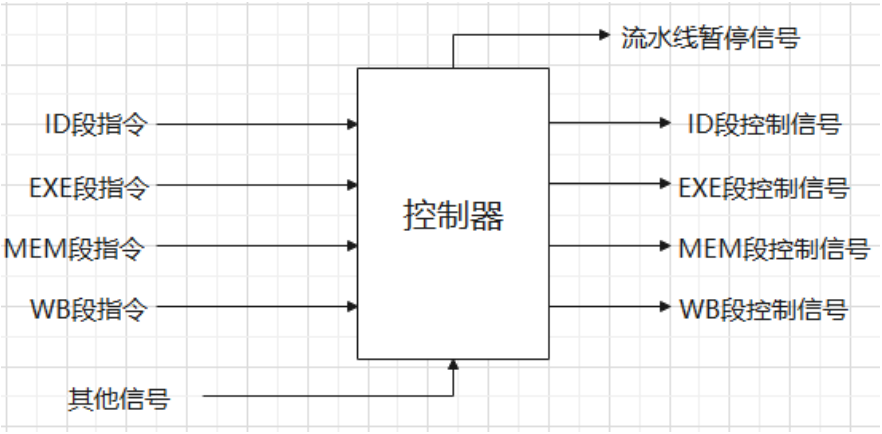
本实验中，除了存储器取数带来的写后读冲突和外部中断带来的暂停以外，没有需要处理的暂停。

(7) 控制器

控制器负责译码，并产生控制信号。

在本实验的 5 级流水线 CPU 中，除 IF 段以外的 4 段都对应着一条指令，因此控制器需要对这四条指令进行译码，并分别产生控制信号。此外还需要根据相关和冲突，决定是否要暂停流水线。

本实验的控制器为组合逻辑，因为每级流水线都在一个时钟周期内完成操作，不需要分周期进行状态转移，因此也不需要加入时钟和复位信号。



(8) 数据前推相关部件

本实验中，对于数据相关带来的冲突，采用定向技术进行处理，即数据前推。

某条指令经过 ID 段译码并准备好操作数后，需要经过两个时钟周期才能写回寄存器堆，经过 EXE 和 MEM 段的时候，都需要视具体的指令进行前推。

具体的实现在于控制器和多路选择器。

在控制器中，需要比较 EXE 段和 MEM 段和 ID 段的指令类型、操作数地址和目标寄存器地址，若确认发生了冲突，需要修改相应多路选择器的选择信号。

多路选择器需要选择的原数据和前推数据。其中，若存在着 3 条连续指令的相关（例如第一条写$1，第二条也写$1，第三条读$1），不考虑暂停，当第三条处于 ID 段的时候，第一条位于 MEM，第二条位于 EXE，此时我们应该选择前推最新的数据，即 EXE 段数据。因此多路选择器要先选择前推的数据，再进行前推。

2. 顶层模块结构部件说明

(1) sccomp_dataflow 模块

实际上就是 CPU+存储器的模块，这里沿用了计算机组成原理的命名，以便统一管理。

(2) 分频模块

为确保实验正确，本实验将时钟周期降低为 1/3，具体的实现是用寄存器计数实现，这里不再过多叙述。

(3) 数码管驱动模块

数码管驱动沿用了计算机组成原理实验的下板用模块，主要原理是对输入的

数据，产生相应的信号，输出到开发板中，使相应的数码管变亮。对于多个数码管，采用动态扫描的方法，实现不同位不同数字的输出。由于扫描频率很快，人眼无法分辨出扫描的过程，因此像是同一时刻多个管显示了不同的数字。具体的实现不是本实验重点，不做详细描述。

(4) 外部中断

本实验要求外部干涉实现暂停，采用拨动开关的方法进行处理。

CPU 会传入一个信号 stall 用来表示是否中断，并会将这个信号传进控制器中，当 stall 信号使能的时候，暂停 5 段流水线。

四、实验仿真过程

1. 动态流水线的仿真过程

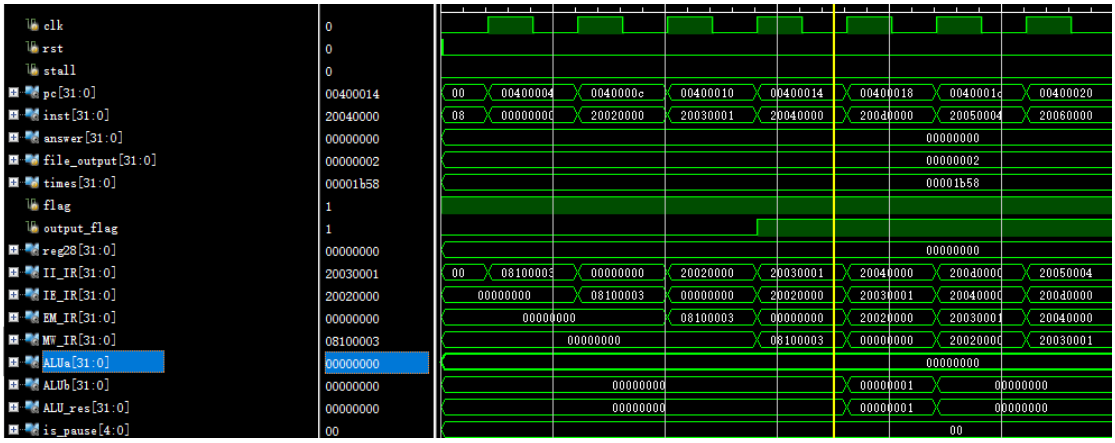
编写完 CPU 后，对 sccomp_dataflow 模块编写测试模块进行验证，并利用层级访问检查寄存器的值。

这里沿用了计算机组成原理使用的测试文件，将通用寄存器值、pc 值、instr 值写入到文件中，并选择重要的寄存器值展示在波形图中。

编写完毕后，点击 Run Simulation-Run Behaviour Simulation 即可开始仿真。

五、实验仿真的波形图及某时刻寄存器值的物理意义

1. 动态流水线的波形图及某时刻寄存器值的物理意义

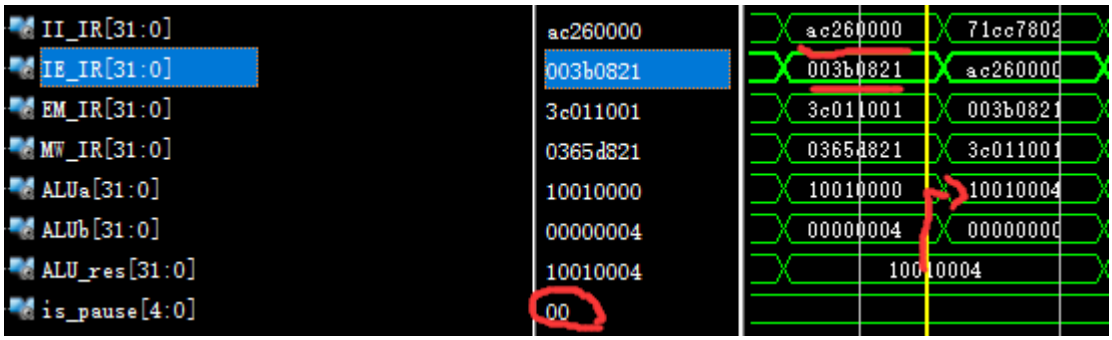


（因波形图过长，截不全）

上图中，II_IR 为 ID 段执行的指令，IE_IR 为 EXE 段执行的指令，EM_IR 为 MEM 段执行的指令，MW_IR 为 WB 段执行的指令，ALUa 为送入 ALU 的操作数 1，

ALUb 为送入 ALU 的操作数 2，ALU_res 为 ALU 的运算结果，is_pause。为是否暂停的标志。

在某一时刻，产生了相关



上图中，指令 ac260000 是 addu $1,$1,$27，指令 003b0821 是 sw $6,0($1)，显然发生了数据相关，本实验中，使用了定向技术，addu 指令的运算结果被前推到 sw 指令中，因而没有暂停。

六、实验验算数学模型及算法程序

int a[m],b[m],c[m],d[m];

a[0]=0;

b[0]=1;

a[i]=a[i-1]+i;

b[i]=b[i-1]+3i;

$$c[i]=\begin{cases} a[i], & 0\le i\le 19 \\ a[i]+b[i], & 20\le i\le 39 \\ a[i]*b[i], & 40\le i\le 59 \end{cases}$$

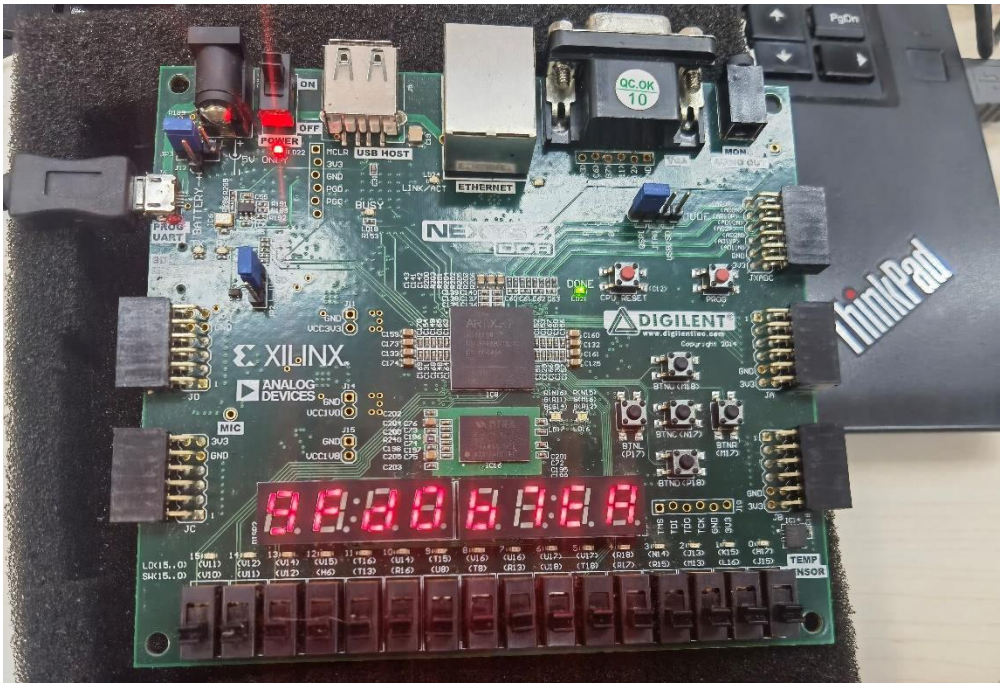
$$d[i]=\begin{cases} b[i], & 0\le i\le 19 \\ a[i]*c[i], & 20\le i\le 39 \\ c[i]*b[i], & 40\le i\le 59 \end{cases}$$

七、实验验算程序下板测试过程与实现

获得实验验算程序的汇编代码，并导出为十六进制机器码，整合成用于 IP 核的 coe 文件。

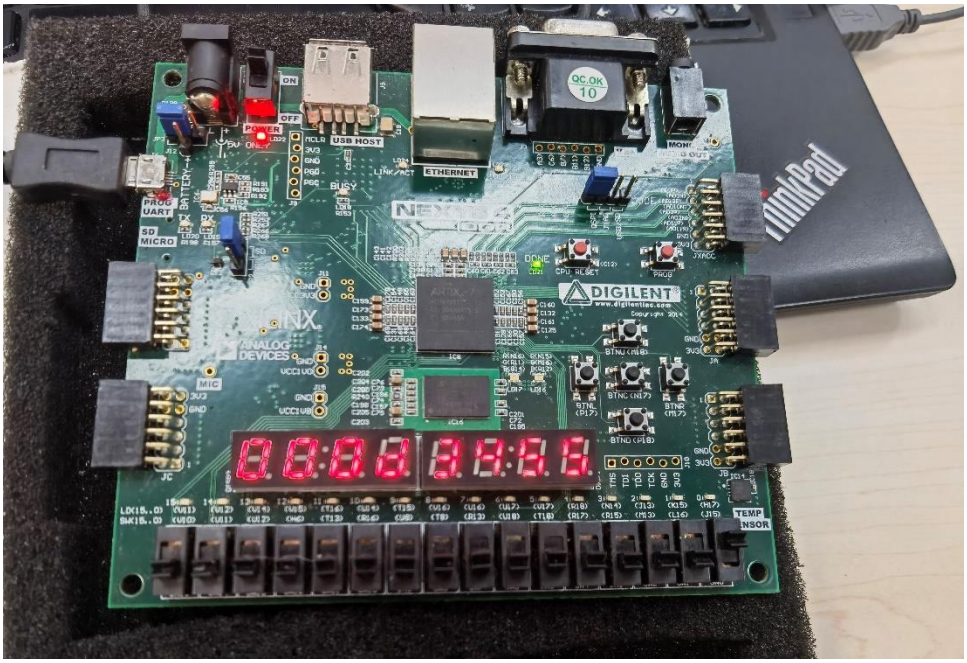
之后在 vivado 中点击 Synthesis（综合），综合成功后配置管脚，并生成比特流文件。将其输入到开发板上。

根据往届验收标准，将 28 号寄存器的值输出到数码管上，得到以下结果。



数字是 9FD0B7EA，符合验收标准。

将流水线暂停，得到如下结果，数字是 D3455，说明暂停成功



八、流水线的性能指标定性分析（包括：吞吐率、加速比、效率及相关与冲突分析）

1. 动态流水线的性能指标定性分析

(1) 吞吐率

本实验得到正确结果共执行了 1843 条指令，包括 159 条分支成功造成的延迟槽失效的指令，因此任务数为 1843-159=1684

由于在本实验中，没有出现存储器读数而必须暂停的情况，因此完成此验证程序不存在暂停，故执行周期为 1843-1+5=1847

因此吞吐率为：

$$TP = \frac{1684}{1847} = 0.912\ 时钟周期^{-1}$$

(2)加速比

若不采用流水线完成本验证程序，考虑到 sw 指令只需要 4 个时钟周期，j，break 指令只需要 2 个时钟周期，beq，bne 指令只需要 3 个时钟周期，因此一共需要 $1179 \times 4 + 304 \times 3 + 201 \times 2 + 159 \times 2 = 6348$个时钟周期。

因此加速比为：

$$S = \frac{6348}{1847} = 4.56$$

(3)效率

时空图中，5 段有效执行周期为 6348 个周期，而总执行周期为$5 \times 1847 = 9235$个时钟周期

因此效率为：

$$\eta = \frac{6348}{9235} \times 100\% = 68.74\%$$

2. 相关和冲突分析

(1)数据相关

本实验中，数据相关出现的情况为上一条指令的目的寄存器是下一条指令的操作数寄存器，例如：

```
addu $1,$1,$27，
sw $6,0($1)
```

对此的处理是进行定向。

addu	IF	ID	EXE	MEM	WB	
sw		IF	ID	EXE	MEM	WB

这样可避免冲突

出现与存储器读数无关的多条冲突时，需要注意定向的正确性，例如：

lui $27,0x0000

addu $27,$27,$5

sw $16,D($27)

此时定向的方式应该是

lui	IF	ID	EXE	MEM	WB		
addu		IF	ID	EXE	MEM	WB	
sw			IF	ID	EXE	MEM	WB

(2)控制相关

本实验中，对于分支跳转的处理，均采用延迟槽的方式实现并在 ID 段完成分支的判断。当分支成功的时候，作废延迟槽的指令

以 beq 为例，分支失败时，正常执行。

beq	IF	ID	EXE	MEM	WB	
next		IF	ID	EXE	MEM	WB

分支成功时，作废延迟槽指令

beq	IF	ID	EXE	MEM	WB		
next		IF	drop	drop	drop	drop	
new			IF	ID	EXE	MEM	WB

其他出现跳转的指令（例如 j，break）指令按照上述情况中的分支成功处理。

(3)寄存器堆写后读冲突

在寄存器堆中，若出现写入地址和读数地址一致，则将读数结果赋值为写入结果，而非寄存器旧值，如此可解决寄存器堆写后读冲突。

需特别注意，0 号寄存器的值恒为 0，因此 0 号寄存器不参与上述冲突的检测。

九、总结与体会

这次实验我设计了 MIPS 33 条指令的动态流水线 CPU，并成功运行了数组运算的汇编程序，证明了流水线 CPU 设计正确。

在上一个实验的基础上，我实现了更多条指令，并做了定向操作，使得流水线 CPU 的吞吐率，加速比和效率有所提升，暂停情况减少。

遗憾在于最开始的设计没有考虑周全，以至于在实现定向技术的时候遇到了很多的问题，并花费了大量的时间修改，这说明我应该更加认真的考虑所有的情况，再完成代码的编写。

总的来说这次实验让我进一步了解了动态流水线 CPU 的工作原理和相关冲突处理，令我收获丰富。

十、附件（所有程序）

1. 验证程序

```
.data
A:.space 240
B:.space 240
C:.space 240
D:.space 240
E:.space 240
.text
j main
exc:
nop
j exc

main:
addi $2,$0,0       #a[i]
addi $3,$0,1       #b[i]
addi $4,$0,0       #c[i]
addi $13,$0,0      #d[i]
addi $5,$0,4       #counter
addi $6,$0,0       #a[i-1]
addi $7,$0,1       #b[i-1]
addi $10,$0,0      #flag for i<20 || i<40
addi $11,$0,240 #sum counts
addi $14,$0,3
addi $30,$0,0

# 把 0 1 0 0 ($2,...,$13) 分别存入 A B C D
lui $27,0x0000
addu $27,$27,$0
sw $2,A($27)
lui $27,0x0000
addu $27,$27,$0
sw $3,B($27)
lui $27,0x0000
addu $27,$27,$0
sw $2,C($27)
lui $27,0x0000
addu $27,$27,$0
sw $3,D($27)

# 循环
loop:
## $5(4) 除以 4 (=i) 存入 $12
srl $12,$5,2
# $6 加 i. 自此, $6 就是 a[i] 而不是 a[i-1]了
add $6,$6,$12
# 把 a[i] 的内容存入 A[i] 中
lui $27,0x0000
addu $27,$27,$5
sw $6,A($27)
# $14 (3) 乘以 $5/4 (i) ( = 3i )
mul $15,$14,$12
# 把 $7 (b[i-1]) 的内容加上 3i, 存入 B[i]. 自此, $7 就是 b[i] 而不是 b[i-1]
add $7,$7,$15
lui $27,0x0000
addu $27,$27,$5
sw $7,B($27)
# $5 是否小于 80 (i 是否小于 20)? 记入 $10
slti $10,$5,80
# 若不是, 跳转
bne $10,1,c1

# (0<=i<=19)
# 把 $6 的内容存入 C[i] 中 (c[i] = a[i])
lui $27,0x0000
addu $27,$27,$5
sw $6,C($27)
# 把 $7 的内容存入 D[i] 中 (d[i] = b[i])
lui $27,0x0000
addu $27,$27,$5
sw $7,D($27)
addi $15,$6,0 # $15 $16 分别赋值为 c[i] d[i]
addi $16,$7,0
j endc
c1: # (20<=i<=39)
# i 是否小于 40 ？ 若不是，跳转到 c2
slti $10,$5,160
addi $27,$0,1
```

```
bne $10,$27,c2
# C[i] = a[i] + b[i]
add $15,$6,$7
lui $27,0x0000
addu $27,$27,$5
sw $15,C($27)
# D[i] = a[i] * b[i]
mul $16, $15,$6
lui $27,0x0000
addu $27,$27,$5
sw $16,D($27)
j endc
c2: # (i>=40)
# C[i] = a[i] * b[i]
mul $15,$6,$7
lui $27,0x0000
addu $27,$27,$5
sw $15,C($27)
# D[i] = c[i] * b[i]
mul $16,$15,$7
lui $27,0x0000
addu $27,$27,$5
sw $16,D($27)

endc:
add $28,$15,$16 #$28 = c[i] + d[i]
lui $27,0x0000
addu $27,$27,$5
sw $28,E($27) # 将 c[i] + d[i] 存入 E[i]
addi $5,$5,4 # i = i + 1
bne $5,$11,loop # i = 60 不跳转
#break
# 最后 E[i] = c[i] + d[i], 可通过验证 E[59] 的正确性来验证 c[i] 和 d[i] 正确性
```

2. 动态流水线 CPU 的 verilog 代码

add.v

```
module add(
    input    [31:0]   a,
    input    [31:0]   b,

    output   [31:0]   r
    );
    assign r=a+b;
endmodule
```

ALU.v

```
module ALU(
        input [31:0] a,
        input [31:0] b,
        input [3:0] aluc,
        output [31:0] r,
        output zero,
        output carry,
        output negative,
        output overflow
        );
parameter
        addu=4'd0,
        add =4'd2,
        subu=4'd1,
        sub =4'd3,
        _and=4'd4,
        _or =4'd5,
        _xor=4'd6,
        _nor=4'd7,
        lui1=4'd8,
        lui2=4'd9,
        sltu=4'd10,
        slt =4'd11,
        sra =4'd12,
        srl =4'd13,
        sll =4'd14,
        slr =4'd15;
reg signed    [31:0]res;
reg                 [32:0]sres;
wire signed [31:0]sa=a,sb=b;
always @ (*)begin
        sres=33'b0;
        case(aluc)
                add:begin

res<=a+b;sres<={sa[31],sa}+{sb[31],sb};
                end
                addu:begin
                        res<=sa+sb;
                end
                sub:begin
res<=a-b;sres<={sa[31],sa}-{sb[31],sb};
                end
                subu:begin
                        res<=sa-sb;
                end
                _and:begin
                        res<=a&b;
                end
                _or:begin
                        res<=a|b;
                end
                _xor:begin
                        res<=a^b;
                end
                _nor:begin
                        res<=~(a|b);
                end
                lui1:begin
                        res<={b[15:0],16'b0};
                end
                lui2:begin
                        res<={b[15:0],16'b0};
                end
                slt:begin
                        res<=(sa<sb)?32'b1:32'b0;
                end
                sltu:begin
                        res<=(a<b)?32'b1:32'b0;
                end
                sra:begin
                        res<=sb>>>a[4:0];
                end
                sll,slr:begin
                        res<=b<<a[4:0];
                end
                srl:begin
                        res<=b>>a[4:0];
                end
                default:begin
                        res<=32'b0;
                end
        endcase
```

```
end
assign r=res[31:0];
assign
zero=(aluc==slt||aluc==sltu)?((a==b)?1'b1:1'b
0):((r==32'b0)?1'b1:1'b0);
assign
carry=(aluc==addu||aluc==subu||aluc==sltu|
|aluc==sra||aluc==sll||aluc==srl)?res[31]:1'b
0;
assign
negative=(aluc==slt)?((res==32'b1)?1'b1:1'b0)
:(res[31]);
assign
overflow=(aluc==add||aluc==sub)?(sres[32]^
sres[31]):1'b0;
endmodule
```

cmp.v

```
module cmp(
    input    [31:0]   num1,
    input    [31:0]   num2,
    input             cmp_equal,

    output            is_branch
    );
    reg       res;
    always @ * begin
        //beq use it
        if(cmp_equal) begin
            res=(num1==num2);
        end
        //bne use it
        else begin
            res=(num1!=num2);
        end
    end
    assign is_branch=res;
endmodule
```

Controller.v

```
module Controller(
    input    [31:0]   II_IR_out,
                      IE_IR_out,
                      EM_IR_out,
                      MW_IR_out,
    input             ID_is_branch,
    input             stall,
```

```
    output              RF_wena,
                        RF_rena,
                        DMEM_rena,
                        DMEM_wena,
                        ID_equal,
                        ext16_sext,
                        MUX2_jpc_s,
                        MUX2_jump_s,
                        MUX2_II_IR_s,

MUX2_cmp_pusha_s,

MUX2_cmp_pushb_s,
                        MUX2_ext5_s,
                        MUX2_pusha_s,
                        MUX2_pushb_s,
                        MUX2_EM_res_s,
                        MUX2_MW_res_s,

    output  [1:0]   MUX4_PC_s,
                        MUX4_IE_ALUa_s,
                        MUX4_IE_ALUb_s,
    output  [3:0]   aluc,
    output  [4:0]   is_pause,
                        WB_waddr,
                        ID_rsc,
                        ID_rtc,
                        ID_sa,
    output  [15:0]  ID_immed,
    output  [25:0]  ID_jaddr
    );
    wire    [`CODE_NUM:0]   Code_type
[1:4];
    wire        [4:0]               rsc
[1:4];
    wire        [4:0]               rtc
[1:4];
    wire        [4:0]               rdc
[1:4];
    wire        [4:0]               sa
[1:4];
    wire        [15:0]              immed
[1:4];
    wire        [25:0]              j_addr
```

```
[1:4];
    /* 译码器 */
    Decoder     ID_Decoder(
        .instr      (II_IR_out),
        .rsc        (rsc[`ID_STATE]),
        .rtc        (rtc[`ID_STATE]),
        .rdc        (rdc[`ID_STATE]),
        .sa         (sa[`ID_STATE]),
        .immed      (immed[`ID_STATE]),
        .j_addr     (j_addr[`ID_STATE]),
        .code       (`IDC)
    );
    Decoder     EXE_Decoder(
        .instr      (IE_IR_out),
        .rsc        (rsc[`EXE_STATE]),
        .rtc        (rtc[`EXE_STATE]),
        .rdc        (rdc[`EXE_STATE]),
        .sa         (sa[`EXE_STATE]),
        .immed
(immed[`EXE_STATE]),
        .j_addr     (j_addr[`EXE_STATE]),
        .code       (`EXEC)
    );
    Decoder     MEM_Decoder(
        .instr      (EM_IR_out),
        .rsc        (rsc[`MEM_STATE]),
        .rtc        (rtc[`MEM_STATE]),
        .rdc        (rdc[`MEM_STATE]),
        .sa         (sa[`MEM_STATE]),
        .immed
(immed[`MEM_STATE]),
        .j_addr
(j_addr[`MEM_STATE]),
        .code       (`MEMC)
    );
    Decoder     WB_Decoder(
        .instr      (MW_IR_out),
        .rsc        (rsc[`WB_STATE]),
        .rtc        (rtc[`WB_STATE]),
        .rdc        (rdc[`WB_STATE]),
        .sa         (sa[`WB_STATE]),
        .immed
(immed[`WB_STATE]),
        .j_addr     (j_addr[`WB_STATE]),
```

```
            .code           (`WBC)
        );
        //conflict judge
        wire
IDCrRsc=~(`IDC[`_SLL]|`IDC[`_SRL]|`IDC[`_SRA
]|

`IDC[`_LUI]|`IDC[`_J]|`IDC[`_JAL]|`IDC[`_BRK])
;
        wire
IDCrRtc=~(`IDC[`_ADDI]|`IDC[`_ADDIU]|`IDC[`
_ANDI]|`IDC[`_ORI]|`IDC[`_XORI]|

`IDC[`_SLTI]|`IDC[`_SLTIU]|`IDC[`_LUI]|`IDC[`_
LW]|`IDC[`_SW]|

`IDC[`_J]|`IDC[`_JAL]|`IDC[`_JR]|`IDC[`_BRK]);
        wire
EXECwRF=~(`EXEC[`_SW]|`EXEC[`_BEQ]|`EXEC
[`_BNE]|`EXEC[`_J]|`EXEC[`_JR]|
                                `EXEC[`_BRK]);
        wire
MEMCwRF=~(`MEMC[`_SW]|`MEMC[`_BEQ]|
`MEMC[`_BNE]|`MEMC[`_J]|`MEMC[`_JR]|
                                `MEMC[`_BRK]);
        wire        [4:0]
                    EXEC_waddr=     (`EXEC[`_JAL])?
5'd31:

(`EXEC[`_ADDI]|`EXEC[`_ADDIU]|`EXEC[`_AND
I]|`EXEC[`_ORI]|`EXEC[`_XORI]|

`EXEC[`_SLTI]|`EXEC[`_SLTIU]|`EXEC[`_LUI]|`E
XEC[`_LW])?rtc[`EXE_STATE]:

rdc[`EXE_STATE],
                    MEMC_waddr=
(`MEMC[`_JAL])? 5'd31:

(`MEMC[`_ADDI]|`MEMC[`_ADDIU]|`MEMC[`
_ANDI]|`MEMC[`_ORI]|`MEMC[`_XORI]|

`MEMC[`_SLTI]|`MEMC[`_SLTIU]|`MEMC[`_LU
I]|`MEMC[`_LW])?rtc[`MEM_STATE]:
rdc[`MEM_STATE];
        //前推不考虑 nop
        wire                        EXEC_pusha=
(II_IR_out!=32'b0) && (IE_IR_out!=32'b0) &&
                                IDCrRsc        &&
EXECwRF &&

(rsc[`ID_STATE]==EXEC_waddr)&&
(EXEC_waddr!=5'b0)                          &&
(rsc[`ID_STATE]!=5'b0);
        wire                        EXEC_pushb=
(II_IR_out!=32'b0) && (IE_IR_out!=32'b0) &&
                                IDCrRtc        &&
EXECwRF &&

(rtc[`ID_STATE]==EXEC_waddr)&&
(EXEC_waddr!=5'b0)                          &&
(rtc[`ID_STATE]!=5'b0);
        wire                        MEMC_pusha=
(II_IR_out!=32'b0)   &&   (EM_IR_out!=32'b0)
&&
                                IDCrRsc        &&
MEMCwRF &&

(rsc[`ID_STATE]==MEMC_waddr)&&
(MEMC_waddr!=5'b0)                          &&
(rsc[`ID_STATE]!=5'b0);
        wire                        MEMC_pushb=
(II_IR_out!=32'b0)   &&   (EM_IR_out!=32'b0)
&&
                                IDCrRtc        &&
MEMCwRF &&

(rtc[`ID_STATE]==MEMC_waddr)&&
(MEMC_waddr!=5'b0)                          &&
(rtc[`ID_STATE]!=5'b0);
        wire        ID_MustStop=(`EXEC[`_LW]) &&
(II_IR_out!=32'b0) &&(
                                        (IDCrRsc
&&(rsc[`EXE_STATE]==rsc[`ID_STATE]))||
                                        (IDCrRtc
&&(rsc[`EXE_STATE]==rtc[`ID_STATE]))
                                );
```

```
    reg         [4:0]     pause;
    always @(*) begin
        if(stall)begin
            pause=`HALT;
        end
        else if(ID_MustStop) begin
            pause=`ID_PAUSE;
        end
        else begin
            pause=`NO_PAUSE;
        end
    end
    assign    is_pause=pause;
    wire
II_IR_drop=`IDC[`_J]|`IDC[`_JAL]|`IDC[`_JR]|((`
IDC[`_BEQ]|`IDC[`_BNE])& ID_is_branch);
    assign
RF_wena=~(`WBC[`_SW]|`WBC[`_BEQ]|`WBC
[`_BNE]|`WBC[`_J]|`WBC[`_JR]|`WBC[`_BRK])
;
    assign    RF_rena=1'b1;
    assign    DMEM_rena=`MEMC[`_LW];
    assign    DMEM_wena=`MEMC[`_SW];
    assign    ID_equal=`IDC[`_BEQ];
    assign
ext16_sext=~(`IDC[`_ADDIU]|`IDC[`_ANDI]|`I
DC[`_ORI]|`IDC[`_XORI]|

`IDC[`_SLTIU]|`IDC[`_LUI]|`IDC[`_LW]|`IDC[`_S
W]);
    assign    WB_waddr=(`WBC[`_JAL])     ?
5'd31     :

(`WBC[`_ADDI]|`WBC[`_ADDIU]|`WBC[`_ANDI
]|`WBC[`_ORI]|`WBC[`_XORI]|`WBC[`_SLTI]|`
WBC[`_SLTIU]|`WBC[`_LUI]|`WBC[`_LW])?rtc[
`WB_STATE]:
                                rdc[`WB_STATE];

    assign
aluc[0]=`EXEC[`_SUB]|`EXEC[`_SUBU]|`EXEC[`
_OR]|`EXEC[`_NOR]|`EXEC[`_SLT]|`EXEC[`_SR
L]|
`EXEC[`_SRLV]|`EXEC[`_ORI]|`EXEC[`_SLTI];
    assign
aluc[1]=`EXEC[`_ADD]|`EXEC[`_SUB]|`EXEC[`_
XOR]|`EXEC[`_NOR]|`EXEC[`_SLT]|`EXEC[`_SLT
U]|

`EXEC[`_SLL]|`EXEC[`_SLLV]|`EXEC[`_ADDI]|`E
XEC[`_XORI]|`EXEC[`_LW]|`EXEC[`_SW]|

`EXEC[`_SLTI]|`EXEC[`_SLTIU];
    assign
aluc[2]=`EXEC[`_AND]|`EXEC[`_OR]|`EXEC[`_X
OR]|`EXEC[`_NOR]|`EXEC[`_SLL]|`EXEC[`_SRL]
|`EXEC[`_SRA]|

`EXEC[`_SLLV]|`EXEC[`_SRLV]|`EXEC[`_SRAV]|`
EXEC[`_ANDI]|`EXEC[`_ORI]|`EXEC[`_XORI];
    assign
aluc[3]=`EXEC[`_SLT]|`EXEC[`_SLTU]|`EXEC[`_S
LL]|`EXEC[`_SRL]|`EXEC[`_SRA]|`EXEC[`_SLLV]
|`EXEC[`_SRLV]|

`EXEC[`_SLTI]|`EXEC[`_SLTIU]|`EXEC[`_LUI];

    assign    ID_rsc=rsc[`ID_STATE];
    assign    ID_rtc=rtc[`ID_STATE];
    assign    ID_sa=sa[`ID_STATE];
    assign    ID_immed=immed[`ID_STATE];
    assign    ID_jaddr=j_addr[`ID_STATE];
    // MUX2_jpc_s: 0 j_pc from II,1
rdata1(jr)
    assign    MUX2_jpc_s=`IDC[`_JR];
    // MUX2_jump: 0 MUX2_jpc_out,1
branch_addr
    assign
MUX2_jump_s=(`IDC[`_BEQ]|`IDC[`_BNE])&ID
_is_branch;
    //MUX2_II_IR: 0 instr_in,1 32'b0(drop)
    assign MUX2_II_IR_s=II_IR_drop;
    //MUX2_ext5_s 0:ID_sa 1: RF_data1
    assign
MUX2_ext5_s=`IDC[`_SLLV]|`IDC[`_SRLV]|`IDC
[`_SRAV];
    assign
```

```
MUX2_cmp_pusha_s=EXEC_pusha || MEMC_pusha;
    assign MUX2_cmp_pushb_s=EXEC_pushb || MEMC_pushb;
    //MUX2_pusha: 0 ALU.r 1:MUX2_MW_res_out(MEM.res)
    //MUX2_pushb similar
    //both push,choose nearest EXE data
    assign MUX2_pusha_s=~EXEC_pusha && MEMC_pusha;
    assign MUX2_pushb_s=~EXEC_pushb && MEMC_pushb;
    //MUX2_EM_res: 0 ALU.r,1 MULT.lo
    assign MUX2_EM_res_s=`EXEC[`_MUL];
    //MUX2_MW_res: 0 EM_res_out,1 DMEM_data_out
    assign MUX2_MW_res_s=`MEMC[`_LW];
    //MUX4_PC_s[0] 0:NPC_out|MUX2_jump_out 1:PC_out|int_entry
    //MUX4_PC_s[1] 0:NPC_out|PC_out 1:MUX2_jump_out|int_entry
    assign MUX4_PC_s[0]=is_pause[`IF_STATE] | `IDC[`_BRK];
    assign MUX4_PC_s[1]=`IDC[`_J]|`IDC[`_JAL]|`IDC[`_JR]|((`IDC[`_BEQ]|`IDC[`_BNE])& ID_is_branch)| `IDC[`_BRK];
    //MUX4_IE_ALUa[0] 0:RF.data1|MUX2_pusha_out 1:ext5_out|NPC_out
```

```
    //MUX4_IE_ALUa[1] 0:RF.data1|ext5_out 1:MUX2_pusha_out|NPC_out
    assign MUX4_IE_ALUa_s[0]=(`IDC[`_SLL]|`IDC[`_SRL]|`IDC[`_SRA]|
    `IDC[`_SLLV]|`IDC[`_SRLV]|`IDC[`_SRAV]|
    `IDC[`_JAL]) && !EXEC_pusha && !MEMC_pusha;
    assign MUX4_IE_ALUa_s[1]=EXEC_pusha|MEMC_pusha|
    `IDC[`_JAL];
    //MUX4_IE_ALUb[0]: 0:RF.data2|MUX2_pushb_out 1:ext16_out|32'd4
    //MUX4_IE_ALUb[1]: 0:RF.data2|ext16_out 1:MUX2_pushb_out|32'd4
    assign MUX4_IE_ALUb_s[0]=(`IDC[`_ADDI]|`IDC[`_ADDIU]|`IDC[`_ANDI]|`IDC[`_ORI]|`IDC[`_XORI]|
    `IDC[`_SLTI]|`IDC[`_SLTIU]|`IDC[`_LUI]|`IDC[`_LW]|`IDC[`_SW]|
    `IDC[`_JAL])&& !EXEC_pushb && !MEMC_pushb;
    assign MUX4_IE_ALUb_s[1]=EXEC_pushb|MEMC_pushb|
    `IDC[`_JAL];
endmodule
```

Decoder.v

```
module Decoder(
    input [31:0] instr,
    output [4:0] rsc,
    output [4:0] rtc,
    output [4:0] rdc,
    output [4:0] sa,
    output [15:0] immed,
    output [25:0] j_addr,
    output reg [`CODE_NUM:0] code
```

```
    );
    wire [5:0]opcode = instr[31:26];
    wire [5:0]func = instr[5:0];
    assign rsc=instr[25:21];
    assign rtc=instr[20:16];
    assign rdc=instr[15:11];
    assign sa =instr[10: 6];
    assign immed=instr[15:0];
    assign j_addr =instr[25:0];
```

```
    always @ * begin
        if(opcode==6'b0)begin
            case(func)
                12'h020:code=`ADD;
//add
                12'h021:code=`ADDU;
//addu
                12'h022:code=`SUB;
//sub
                12'h023:code=`SUBU;
//subu
                12'h024:code=`AND;
//and
                12'h025:code=`OR;
//or
                12'h026:code=`XOR;
//xor
                12'h027:code=`NOR;
//nor
                12'h02a:code=`SLT;
//slt
                12'h02b:code=`SLTU;
//sltu
                12'h000:code=`SLL;
//sll
                12'h002:code=`SRL;
//srl
                12'h003:code=`SRA;
//sra
                12'h004:code=`SLLV;
//sllv
                12'h006:code=`SRLV;
//srlv
                12'h007:code=`SRAV;
//srav
                12'h008:code=`JR;
//jr

//6'b001101:code=56'h0000_0000_4000_00;
//break
                12'h00d:code=`BRK;
//break
                default:code=32'h0;
            endcase
        end
        else begin
            case(opcode)
                6'b001000:code=`ADDI;
//addi
                6'b001001:code=`ADDIU;
//addiu
                6'b001100:code=`ANDI;
//andi
                6'b001101:code=`ORI;
//ori
                6'b001110:code=`XORI;
//xori
                6'b100011:code=`LW;
//lw
                6'b101011:code=`SW;
//sw
                6'b000100:code=`BEQ;
//beq
                6'b000101:code=`BNE;
//bne
                6'b001010:code=`SLTI;
//slti
                6'b001011:code=`SLTIU;
//sltiu
                6'b001111:code=`LUI;
//lui
                6'b000010:code=`J;    //j
                6'b000011:code=`JAL;
//jal
                6'b011100:code=`MUL;
//mul
                default:    code=32'h0;
            endcase
        end
    end
endmodule
```

Divider.v

```
module Divider(
	input clk_in,
	input reset,
	output clk_out
	);
	reg clk=1'b0;
	reg [2:0]cnt;
	always @(posedge clk_in or posedge reset)begin
		if(reset)begin
			clk<=1'b0;
			cnt<=2'b0;
		end
		else begin
			if(cnt==2'b11) begin
				clk<=~clk;
				cnt<=2'b0;
			end
			else begin
				cnt<=cnt+1;
			end
		end
	end
	assign clk_out=clk;
endmodule
```

DMEM.v

```
module DMEM(
	input                          clk,
	input                          rst,
	input                          wena,
	input                          rena,
	input     [`ADDR_BYTES-1:0] addr,
	input     [31:0]               data_in,
	output    [31:0]               data_out
	);
	reg [31:0] ram_array[0:`MAX_MEMORY];
	integer i;
	always@(posedge clk or posedge rst) begin
		if (rst)begin
```

```
for(i=0;i<=`MAX_MEMORY;i=i+1) begin
				ram_array[i]<=32'b0;
			end
		end
		else begin

ram_array[addr]=(wena)?data_in:ram_array[addr];
		end
	end
	assign data_out=(rena)?ram_array[addr]:32'bz;
endmodule
```

DPCPU.v

```
`include       "defines.vh"

module DPCPU(
            input                clk,
            input                rst,
            input     [31:0]   instr_in,
            input     [31:0]   DMEM_out,

            input                stall,

            output    [31:0]   pc,
            output    [31:0]   DMEM_addr,
            output    [31:0]   DMEM_data,
            output               DMEM_wena,
            output               DMEM_rena,

            output    [31:0]   answer
      );
      //

      //连线
      //CU input
      wire                 ID_is_branch;
      wire       [31:0]   II_IR_out,
                           IE_IR_out,
                           EM_IR_out,
                           MW_IR_out;
      //CU output
      wire                 RF_wena,
                           RF_rena,
                           ID_equal,
                           ext16_sext,
                           MUX2_jpc_s,
                           MUX2_jump_s,
                           MUX2_II_IR_s,
                           MUX2_ext5_s,
                           MUX2_pusha_s,
                           MUX2_pushb_s,
                           MUX2_EM_res_s,
                           MUX2_MW_res_s;
      wire                 MUX2_cmp_pusha_s,

MUX2_cmp_pushb_s;
      wire                                [31:0]
MUX2_cmp_pusha_out,

MUX2_cmp_pushb_out;
      wire       [1:0]    MUX4_PC_s,
                           MUX4_IE_ALUa_s,
                           MUX4_IE_ALUb_s;
      wire       [3:0]    aluc;
      wire       [4:0]    is_pause;
      wire       [4:0]    WB_waddr,
                           ID_rsc,
                           ID_rtc,
                           ID_sa;
      wire       [15:0]   ID_immed;
      wire       [25:0]   ID_jaddr;


      //中间连线
      wire                zero,
                          carry,
                          negative,
                          overflow;

      wire       [4:0]    MUX2_ext5_out;

      wire       [31:0]   MUX2_jpc_out;
      wire       [31:0]   MUX4_PC_out,
                          PC_out;
      wire       [31:0]   NPC_out;
      wire       [31:0]   MUX2_jump_out;
      wire       [31:0]   MUX2_II_IR_out;
      wire       [31:0]   II_NPC_out;
      wire       [31:0]   MW_res_out,
                          RF_data1,
                          RF_data2;
      wire       [31:0]   ext5_out;
      wire       [31:0]   ext16_out;
      wire       [31:0]   j_pc;
      wire       [31:0]   b_pc;
      wire       [31:0]   ALU_r,
                          MUX2_pusha_out,
                          MUX2_pushb_out;
      wire       [31:0]   MUX4_IE_ALUa_out;
```

```
    wire    [31:0]  IE_ALUa_out;
    wire    [31:0]  MUX4_IE_ALUb_out;
    wire    [31:0]  IE_ALUb_out;
    wire    [31:0]  IE_wdata_out;
    wire    [31:0]  MUX2_EM_res_out;
    wire    [31:0]  EM_res_out;
    wire    [31:0]  EM_wdata_out;
    wire    [31:0]  MUX2_MW_res_out;
    wire    [31:0]  hi,
                    lo;
    assign pc=PC_out;
    assign  DMEM_addr=EM_res_out;
    assign  DMEM_data=EM_wdata_out;
    //CU
    Controller      CU(
        .II_IR_out      (II_IR_out),
        .IE_IR_out      (IE_IR_out),
        .EM_IR_out      (EM_IR_out),
        .MW_IR_out      (MW_IR_out),
        .ID_is_branch   (ID_is_branch),

        .stall          (stall),

        .RF_wena        (RF_wena),
        .RF_rena        (RF_rena),
        .DMEM_rena      (DMEM_rena),
        .DMEM_wena
(DMEM_wena),
        .ID_equal       (ID_equal),
        .ext16_sext     (ext16_sext),
        .MUX2_jpc_s     (MUX2_jpc_s),
        .MUX2_jump_s
(MUX2_jump_s),
        .MUX2_II_IR_s   (MUX2_II_IR_s),
        .MUX2_ext5_s    (MUX2_ext5_s),
        .MUX2_cmp_pusha_s
(MUX2_cmp_pusha_s),
        .MUX2_cmp_pushb_s
(MUX2_cmp_pushb_s),
        .MUX2_pusha_s
(MUX2_pusha_s),
        .MUX2_pushb_s
(MUX2_pushb_s),
        .MUX2_EM_res_s
```

```
(MUX2_EM_res_s),
        .MUX2_MW_res_s
(MUX2_MW_res_s),
        .MUX4_PC_s      (MUX4_PC_s),
        .MUX4_IE_ALUa_s
(MUX4_IE_ALUa_s),
        .MUX4_IE_ALUb_s
(MUX4_IE_ALUb_s),
        .aluc           (aluc),
        .is_pause       (is_pause),
        .WB_waddr       (WB_waddr),
        .ID_rsc         (ID_rsc),
        .ID_rtc         (ID_rtc),
        .ID_sa          (ID_sa),
        .ID_immed       (ID_immed),
        .ID_jaddr       (ID_jaddr)
    );
    //IF
    PCReg           cpu_PC(
        .clk        (clk),
        .rst        (rst),
        .pc_in      (MUX4_PC_out),
        .pc_out     (PC_out)
    );
    /*IMEM*/
    NPC             cpu_NPC(
        .data_in    (PC_out),
        .data_out   (NPC_out)
    );

    MUX2            MUX2_jump(
        .data0      (MUX2_jpc_out),
        .data1      (b_pc),
        .s          (MUX2_jump_s),
        .out        (MUX2_jump_out)
    );

    MUX4            MUX4_PC(
        .data0      (NPC_out),
        .data1      (PC_out),
        .data2      (MUX2_jump_out),
        .data3      (`INT_ENTRY),
        .s          (MUX4_PC_s),
        .out        (MUX4_PC_out)
```

```
    );
    //IF/ID

    MUX2                MUX2_II_IR(
        .data0      (instr_in),
        .data1      (32'b0),
        .s          (MUX2_II_IR_s),
        .out        (MUX2_II_IR_out)
    );

    SegmentReg          II_IR(
        .clk        (clk),
        .rst        (rst),
        .prev_pause (is_pause[`IF_STATE]),
        .next_pause (is_pause[`ID_STATE]),
        .data_in    (MUX2_II_IR_out),
        .data_out   (II_IR_out)
    );

    SegmentReg          II_NPC(
        .clk        (clk),
        .rst        (rst),
        .prev_pause (is_pause[`IF_STATE]),
        .next_pause (is_pause[`ID_STATE]),
        .data_in    (NPC_out),
        .data_out   (II_NPC_out)
    );
    //ID

    Regfiles            cpu_ref(
        .clk        (clk),
        .rst        (rst),
        .wena       (RF_wena),
        .rena       (RF_rena),
        .wdata      (MW_res_out),
        .waddr      (WB_waddr),
        .raddr1     (ID_rsc),
        .raddr2     (ID_rtc),
        .rdata1     (RF_data1),
        .rdata2     (RF_data2),
        .answer     (answer)
    );

    MUX2                MUX2_cmp_pusha(
        .data0      (RF_data1),
        .data1      (MUX2_pusha_out),
        .s          (MUX2_cmp_pusha_s),
        .out        (MUX2_cmp_pusha_out)
    );
    MUX2                MUX2_cmp_pushb(
        .data0      (RF_data2),
        .data1      (MUX2_pushb_out),
        .s          (MUX2_cmp_pushb_s),
        .out        (MUX2_cmp_pushb_out)
    );
    cmp                 cpu_cmp(
        .num1       (MUX2_cmp_pusha_out),
        .num2       (MUX2_cmp_pushb_out),
        .cmp_equal  (ID_equal),
        .is_branch  (ID_is_branch)
    );

    MUX25               MUX2_ext5(
        .data0      (ID_sa),
        .data1      (RF_data1[4:0]),
        .s          (MUX2_ext5_s),
        .out        (MUX2_ext5_out)
    );

    ext5                CPU_ext5(
        .a          (MUX2_ext5_out),
        .sext       (1'b0),
        .b          (ext5_out)
    );

    ext16               CPU_ext16(
        .a          (ID_immed),
        .sext       (ext16_sext),
        .b          (ext16_out)
    );

    II                  CPU_II(
```

```
        .a              (II_NPC_out[31:28]),
        .b              (ID_jaddr),
        .r              (j_pc)
    );

    MUX2                MUX2_jpc(
        .data0          (j_pc),
        .data1          (RF_data1),
        .s              (MUX2_jpc_s),
        .out            (MUX2_jpc_out)
    );

    add                 branch_add(
        .a              (II_NPC_out),
        .b
({{(14){ID_immed[15]}},ID_immed,2'b0}),
        .r              (b_pc)
    );
    //ID/EXE

    MUX2                MUX2_pusha(
        .data0          (MUX2_EM_res_out),
        .data1
(MUX2_MW_res_out),
        .s              (MUX2_pusha_s),
        .out            (MUX2_pusha_out)
    );
    MUX2                MUX2_pushb(
        .data0          (MUX2_EM_res_out),
        .data1
(MUX2_MW_res_out),
        .s              (MUX2_pushb_s),
        .out            (MUX2_pushb_out)
    );
    MUX4                MUX4_IE_ALUa(
        .data0          (RF_data1),
        .data1          (ext5_out),
        .data2          (MUX2_pusha_out),
        .data3          (II_NPC_out),
        .s              (MUX4_IE_ALUa_s),
        .out            (MUX4_IE_ALUa_out)
    );

    SegmentReg          IE_ALUa(
```

```
        .clk            (clk),
        .rst            (rst),
        .prev_pause (is_pause[`ID_STATE]),
        .next_pause
(is_pause[`EXE_STATE]),
        .data_in        (MUX4_IE_ALUa_out),
        .data_out       (IE_ALUa_out)
    );

    MUX4                MUX4_IE_ALUb(
        .data0          (RF_data2),
        .data1          (ext16_out),
        .data2          (MUX2_pushb_out),
        .data3          (32'd4),
        .s              (MUX4_IE_ALUb_s),
        .out            (MUX4_IE_ALUb_out)
    );

    SegmentReg          IE_ALUb(
        .clk            (clk),
        .rst            (rst),
        .prev_pause (is_pause[`ID_STATE]),
        .next_pause
(is_pause[`EXE_STATE]),
        .data_in        (MUX4_IE_ALUb_out),
        .data_out       (IE_ALUb_out)
    );

    SegmentReg          IE_IR(
        .clk            (clk),
        .rst            (rst),
        .prev_pause (is_pause[`ID_STATE]),
        .next_pause
(is_pause[`EXE_STATE]),
        .data_in        (II_IR_out),
        .data_out       (IE_IR_out)
    );

    SegmentReg          IE_wdata(
        .clk            (clk),
        .rst            (rst),
        .prev_pause (is_pause[`ID_STATE]),
        .next_pause
(is_pause[`EXE_STATE]),
```

```
        .data_in        (RF_data2),
        .data_out       (IE_wdata_out)
    );
    //EXE

    ALU                 cpu_ALU(
        .a              (IE_ALUa_out),
        .b              (IE_ALUb_out),
        .aluc           (aluc),
        .r              (ALU_r),
        .zero           (zero),
        .carry          (carry),
        .negative       (negative),
        .overflow       (overflow)
    );

    MULT                cpu_MULT(
        .a              (IE_ALUa_out),
        .b              (IE_ALUb_out),
        .hi             (hi),
        .lo             (lo)
    );
    //EXE/MEM

    SegmentReg          EM_IR(
        .clk            (clk),
        .rst            (rst),
        .prev_pause
(is_pause[`EXE_STATE]),
        .next_pause
(is_pause[`MEM_STATE]),
        .data_in        (IE_IR_out),
        .data_out       (EM_IR_out)
    );

    MUX2                MUX2_EM_res(
        .data0          (ALU_r),
        .data1          (lo),
        .s              (MUX2_EM_res_s),
        .out            (MUX2_EM_res_out)
    );

    SegmentReg          EM_res(
        .clk            (clk),
        .rst            (rst),
        .prev_pause
(is_pause[`EXE_STATE]),
        .next_pause
(is_pause[`MEM_STATE]),
        .data_in        (MUX2_EM_res_out),
        .data_out       (EM_res_out)
    );

    SegmentReg          EM_wdata(
        .clk            (clk),
        .rst            (rst),
        .prev_pause
(is_pause[`EXE_STATE]),
        .next_pause
(is_pause[`MEM_STATE]),
        .data_in        (IE_wdata_out),
        .data_out       (EM_wdata_out)
    );
    //MEM
    //DMEM

    //wena,rena in CU
    //MEM/WB

    MUX2                MUX2_MW_res(
        .data0          (EM_res_out),
        .data1          (DMEM_out),
        .s              (MUX2_MW_res_s),
        .out            (MUX2_MW_res_out)
    );
    SegmentReg          MW_res(
        .clk            (clk),
        .rst            (rst),
        .prev_pause
(is_pause[`MEM_STATE]),
        .next_pause (is_pause[`WB_STATE]),
        .data_in        (MUX2_MW_res_out),
        .data_out       (MW_res_out)
    );
    SegmentReg          MW_IR(
        .clk            (clk),
        .rst            (rst),
        .prev_pause
```

```
(is_pause[`MEM_STATE]),
            .next_pause (is_pause[`WB_STATE]),
            .data_in       (EM_IR_out),
            .data_out      (MW_IR_out)
      );
      //WB
      //Regfiles
endmodule
```

ext5.v

```
module ext5#(parameter WIDTH = 5)(
      input [WIDTH - 1:0] a,
      input  sext,          //sext 有效为符号扩展，否则 0 扩展
      output [31:0] b //32 位输出数据，
);
assign
b=sext?{{(32-WIDTH){a[WIDTH-1]}},a}:{{(32-WIDTH){1'b0}},a};
endmodule
```

ext16.v

```
module ext16#(parameter WIDTH = 16)(
      input [WIDTH - 1:0] a,
      input  sext,          //sext 有效为符号扩展，否则 0 扩展
      output [31:0] b //32 位输出数据，
);
assign
b=sext?{{(32-WIDTH){a[WIDTH-1]}},a}:{{(32-WIDTH){1'b0}},a};
endmodule
```

II.v

```
module II(
      input     [3:0]     a,
      input     [25:0]    b,
      output    [31:0]    r
      );
      assign r={a,b,2'b0};
endmodule
```

IMEM.v

```
module IMEM(
	input	[10:0]	addr,
	output	[31:0]	instr
	);
	dist_mem_gen_0		imem(
		addr,
		instr
	);
endmodule
```

MULT.v

```
module MULT(
	input [31:0] a,
	input [31:0] b,
	output [31:0] hi,
	output [31:0] lo
	);
	wire [63:0] moved_a[31:0];
	wire [63:0] z;
	assign
moved_a[0]=(b[0])?(-{{32{a[31]}},a}):64'b0;
	assign
moved_a[1]=(b[1]^b[0])?(b[1]?(-{{31{a[31]}},a
,1'b0}):({{31{a[31]}},a,1'b0})):64'b0;
	assign
moved_a[2]=(b[2]^b[1])?(b[2]?(-{{30{a[31]}},a
,2'b0}):({{30{a[31]}},a,2'b0})):64'b0;
	assign
moved_a[3]=(b[3]^b[2])?(b[3]?(-{{29{a[31]}},a
,3'b0}):({{29{a[31]}},a,3'b0})):64'b0;
	assign
moved_a[4]=(b[4]^b[3])?(b[4]?(-{{28{a[31]}},a
,4'b0}):({{28{a[31]}},a,4'b0})):64'b0;
	assign
moved_a[5]=(b[5]^b[4])?(b[5]?(-{{27{a[31]}},a
,5'b0}):({{27{a[31]}},a,5'b0})):64'b0;
	assign
moved_a[6]=(b[6]^b[5])?(b[6]?(-{{26{a[31]}},a
,6'b0}):({{26{a[31]}},a,6'b0})):64'b0;
	assign
moved_a[7]=(b[7]^b[6])?(b[7]?(-{{25{a[31]}},a
,7'b0}):({{25{a[31]}},a,7'b0})):64'b0;
	assign
moved_a[8]=(b[8]^b[7])?(b[8]?(-{{24{a[31]}},a
,8'b0}):({{24{a[31]}},a,8'b0})):64'b0;
	assign
moved_a[9]=(b[9]^b[8])?(b[9]?(-{{23{a[31]}},a
,9'b0}):({{23{a[31]}},a,9'b0})):64'b0;
	assign
moved_a[10]=(b[10]^b[9])?(b[10]?(-{{22{a[31
]}},a,10'b0}):({{22{a[31]}},a,10'b0})):64'b0;
	assign
moved_a[11]=(b[11]^b[10])?(b[11]?(-{{21{a[3
1]}},a,11'b0}):({{21{a[31]}},a,11'b0})):64'b0;
	assign
moved_a[12]=(b[12]^b[11])?(b[12]?(-{{20{a[3
1]}},a,12'b0}):({{20{a[31]}},a,12'b0})):64'b0;
	assign
moved_a[13]=(b[13]^b[12])?(b[13]?(-{{19{a[3
1]}},a,13'b0}):({{19{a[31]}},a,13'b0})):64'b0;
	assign
moved_a[14]=(b[14]^b[13])?(b[14]?(-{{18{a[3
1]}},a,14'b0}):({{18{a[31]}},a,14'b0})):64'b0;
	assign
moved_a[15]=(b[15]^b[14])?(b[15]?(-{{17{a[3
1]}},a,15'b0}):({{17{a[31]}},a,15'b0})):64'b0;
	assign
moved_a[16]=(b[16]^b[15])?(b[16]?(-{{16{a[3
1]}},a,16'b0}):({{16{a[31]}},a,16'b0})):64'b0;
	assign
moved_a[17]=(b[17]^b[16])?(b[17]?(-{{15{a[3
1]}},a,17'b0}):({{15{a[31]}},a,17'b0})):64'b0;
	assign
moved_a[18]=(b[18]^b[17])?(b[18]?(-{{14{a[3
1]}},a,18'b0}):({{14{a[31]}},a,18'b0})):64'b0;
```

```
    assign
moved_a[19]=(b[19]^b[18])?(b[19]?(-{{13{a[3
1]}},a,19'b0}):({{13{a[31]}},a,19'b0})):64'b0;
    assign
moved_a[20]=(b[20]^b[19])?(b[20]?(-{{12{a[3
1]}},a,20'b0}):({{12{a[31]}},a,20'b0})):64'b0;
    assign
moved_a[21]=(b[21]^b[20])?(b[21]?(-{{11{a[3
1]}},a,21'b0}):({{11{a[31]}},a,21'b0})):64'b0;
    assign
moved_a[22]=(b[22]^b[21])?(b[22]?(-{{10{a[3
1]}},a,22'b0}):({{10{a[31]}},a,22'b0})):64'b0;
    assign
moved_a[23]=(b[23]^b[22])?(b[23]?(-{{9{a[31
]}},a,23'b0}):({{9{a[31]}},a,23'b0})):64'b0;
    assign
moved_a[24]=(b[24]^b[23])?(b[24]?(-{{8{a[31
]}},a,24'b0}):({{8{a[31]}},a,24'b0})):64'b0;
    assign
moved_a[25]=(b[25]^b[24])?(b[25]?(-{{7{a[31
]}},a,25'b0}):({{7{a[31]}},a,25'b0})):64'b0;
    assign
moved_a[26]=(b[26]^b[25])?(b[26]?(-{{6{a[31
]}},a,26'b0}):({{6{a[31]}},a,26'b0})):64'b0;
    assign
moved_a[27]=(b[27]^b[26])?(b[27]?(-{{5{a[31
]}},a,27'b0}):({{5{a[31]}},a,27'b0})):64'b0;
    assign
moved_a[28]=(b[28]^b[27])?(b[28]?(-{{4{a[31
]}},a,28'b0}):({{4{a[31]}},a,28'b0})):64'b0;
    assign
moved_a[29]=(b[29]^b[28])?(b[29]?(-{{3{a[31
]}},a,29'b0}):({{3{a[31]}},a,29'b0})):64'b0;
    assign
moved_a[30]=(b[30]^b[29])?(b[30]?(-{{2{a[31
]}},a,30'b0}):({{2{a[31]}},a,30'b0})):64'b0;
    assign
moved_a[31]=(b[31]^b[30])?(b[31]?(-{{1{a[31
]}},a,31'b0}):({{1{a[31]}},a,31'b0})):64'b0;
    assign
z=moved_a[0]+moved_a[1]+moved_a[2]+mo
ved_a[3]+moved_a[4]

+moved_a[5]+moved_a[6]+moved_a[7]+mov
ed_a[8]+moved_a[9]

+moved_a[10]+moved_a[11]+moved_a[12]+
moved_a[13]+moved_a[14]

+moved_a[15]+moved_a[16]+moved_a[17]+
moved_a[18]+moved_a[19]

+moved_a[20]+moved_a[21]+moved_a[22]+
moved_a[23]+moved_a[24]

+moved_a[25]+moved_a[26]+moved_a[27]+
moved_a[28]+moved_a[29]
                +moved_a[30]+moved_a[31];
    assign hi=z[63:32];
    assign lo=z[31:0];
endmodule
```

MUX2.v

```
module MUX2(
    input       [31:0]  data0,
    input       [31:0]  data1,
    input               s,
    output      [31:0]  out
    );
    assign out=(s)?data1:data0;
endmodule
```

MUX4.v

```
module MUX4(
    input       [31:0]  data0,
    input       [31:0]  data1,
    input       [31:0]  data2,
    input       [31:0]  data3,
    input       [1:0]   s,
    output      [31:0]  out
    );
    assign out= (s==2'b00)?data0:
                (s==2'b01)?data1:
                (s==2'b10)?data2:
                data3;
endmodule
```

MUX25.v

```
module MUX25(
    input       [4:0]   data0,
    input       [4:0]   data1,
    input               s,
    output      [4:0]   out
    );
    assign out=(s)?data1:data0;
endmodule
```

NPC.v

```
module NPC(
    input   [31:0]  data_in,
    output  [31:0]  data_out
    );
    assign data_out=data_in+32'h4;
endmodule
```

PCReg.v

```
module PCReg(
    input              clk,
    input              rst,
    input   [31:0]   pc_in,
    output  [31:0]   pc_out
    );
    reg [31:0]   pc;
    always @ (posedge clk or posedge rst)begin
        if(rst)
            pc<=`PC_INIT;
         else
            pc<=pc_in;
    end
    assign pc_out=pc;
endmodule
```

Regfiles.v

```
module Regfiles(
    input              clk,
    input              rst,
    input              wena,
    input              rena,
    input   [31:0]   wdata,
    input   [4:0]    waddr,
    input   [4:0]    raddr1,
    input   [4:0]    raddr2,

    output  [31:0]   rdata1,
    output  [31:0]   rdata2,
    output  [31:0]   answer
    );
    reg [31:0]   array_reg    [31:0];
    integer i;
    always @(posedge clk or posedge rst)begin
        if(rst) begin
            for(i=0;i<=31;i=i+1)
                array_reg[i]<=32'b0;
        end
        else begin
            array_reg[waddr]<=(wena && waddr!=5'b0)?wdata:array_reg[waddr];
        end
    end
    assign rdata1=(rena)?((wena && waddr==raddr1 && waddr!=5'b0)?wdata:array_reg[raddr1]):32'bz;
    assign rdata2=(rena)?((wena && waddr==raddr2 && waddr!=5'b0)?wdata:array_reg[raddr2]):32'bz;
    assign answer=array_reg[28];
endmodule
```

sccomp_dataflow.v

```
`include "defines.vh"

module sccomp_dataflow(
    input           clk,
    input           rst,
    input           stall,
    output  [31:0]  pc,
    output  [31:0]  inst,
    output  [31:0]  answer
    );
    /*数据通路*/
    wire                    [31:0]
DMEM_out,DMEM_addr,DMEM_data;
    wire                    [31:0]
IMEM_addr_in=(pc-`PC_INIT)>>2,

DMEM_addr_in=(DMEM_addr-`DMEM_BASE)
>>2;
    wire
DMEM_wena,DMEM_rena;
    DPCPU           sccpu(
        .clk        (clk),
        .rst        (rst),
        .instr_in   (inst),
        .DMEM_out   (DMEM_out),
        .stall      (stall),
        .pc         (pc),
        .DMEM_addr  (DMEM_addr),
        .DMEM_data  (DMEM_data),
        .DMEM_wena  (DMEM_wena),
        .DMEM_rena  (DMEM_rena),
        .answer     (answer)
    );
    DMEM            cpu_DMEM(
        .clk        (clk),
        .rst        (rst),
        .wena       (DMEM_wena),
        .rena       (DMEM_rena),
        .addr
(DMEM_addr_in[`ADDR_BYTES-1:0]),
        .data_in    (DMEM_data),
        .data_out   (DMEM_out)
    );
    IMEM            cpu_IMEM(
        .addr
(IMEM_addr_in[10:0]),
        .instr      (inst)
    );
endmodule
```

seg7x16.v

```
module seg7x16(
        input clk,
        input reset,
        input cs,
        input [31:0] i_data,
        output [7:0] o_seg,
        output [7:0] o_sel
      );

      reg [14:0] cnt;
       always @ (posedge clk, posedge reset)
         if (reset)
            cnt <= 0;
         else
            cnt <= cnt + 1'b1;

      wire seg7_clk = cnt[14];

       reg [2:0] seg7_addr;

       always @ (posedge seg7_clk, posedge reset)
           if(reset)
               seg7_addr <= 0;
            else
               seg7_addr <= seg7_addr + 1'b1;

       reg [7:0] o_sel_r;

       always @ (*)
          case(seg7_addr)
              7 : o_sel_r = 8'b01111111;
              6 : o_sel_r = 8'b10111111;
              5 : o_sel_r = 8'b11011111;
              4 : o_sel_r = 8'b11101111;
              3 : o_sel_r = 8'b11110111;
              2 : o_sel_r = 8'b11111011;
              1 : o_sel_r = 8'b11111101;
              0 : o_sel_r = 8'b11111110;
           endcase

       reg [31:0] i_data_store;
       always @ (posedge clk, posedge reset)
          if(reset)
              i_data_store <= 0;
           else if(cs)
              i_data_store <= i_data;

       reg [7:0] seg_data_r;
       always @ (*)
          case(seg7_addr)
              0       :       seg_data_r      =
i_data_store[3:0];
              1       :       seg_data_r      =
i_data_store[7:4];
              2       :       seg_data_r      =
i_data_store[11:8];
              3       :       seg_data_r      =
i_data_store[15:12];
              4       :       seg_data_r      =
i_data_store[19:16];
              5       :       seg_data_r      =
i_data_store[23:20];
              6       :       seg_data_r      =
i_data_store[27:24];
              7       :       seg_data_r      =
i_data_store[31:28];
           endcase

       reg [7:0] o_seg_r;
       always @ (posedge clk, posedge reset)
          if(reset)
              o_seg_r <= 8'hff;
           else
              case(seg_data_r)
                 4'h0 : o_seg_r <= 8'hC0;
              4'h1 : o_seg_r <= 8'hF9;
              4'h2 : o_seg_r <= 8'hA4;
              4'h3 : o_seg_r <= 8'hB0;
              4'h4 : o_seg_r <= 8'h99;
              4'h5 : o_seg_r <= 8'h92;
              4'h6 : o_seg_r <= 8'h82;
              4'h7 : o_seg_r <= 8'hF8;
              4'h8 : o_seg_r <= 8'h80;
```

```
                4'h9 : o_seg_r <= 8'h90;
                4'hA : o_seg_r <= 8'h88;
                4'hB : o_seg_r <= 8'h83;
                4'hC : o_seg_r <= 8'hC6;
                4'hD : o_seg_r <= 8'hA1;
                4'hE : o_seg_r <= 8'h86;
                4'hF : o_seg_r <= 8'h8E;
                endcase

        assign o_sel = o_sel_r;
        assign o_seg = o_seg_r;

endmodule
```

SegmentReg.v

```
module SegmentReg(
    input               clk,
    input               rst,
    input               prev_pause,
    input               next_pause,
    input   [31:0]  data_in,

    output  [31:0]  data_out
    );
    reg [31:0]  data;
    always@(posedge clk or posedge rst)begin
        if(rst)begin
            data<=32'h0;
        end
        else if(prev_pause&&~next_pause)
            data<=32'h0;
        else if(prev_pause)
            data<=data;
        else begin
            data<=data_in;
        end
    end
    assign data_out=data;
endmodule
```

top.v

```
module top(
    input               clk,
    input               rst,
    input               stall,

    output  [7:0]   seg,
    output  [7:0]   sel
    );
    wire    [31:0]  answer;
    wire    [31:0]  pc;
    wire    [31:0]  inst;
    wire            cpu_clk;
    Divider         clk_Divider(
        .clk_in     (clk),
        .reset      (rst),
        .clk_out    (cpu_clk)
    );
    sccomp_dataflow sccomp(
        .clk        (clk),
        .rst        (rst),
        .stall      (stall),
        .pc         (pc),
        .inst       (inst),
        .answer     (answer)
    );
    seg7x16         seg_driver(
        .clk        (clk),
        .reset      (rst),
        .cs         (1'b1),
        .i_data     (answer),
        .o_seg      (seg),
        .o_sel      (sel)
    );
endmodule
```