中山大学本科生实验报告

（2016 学年春季学期）

课程名称：嵌入式系统案例分析与设计　　任课教师：**王军**　　助教：杨涵铄

年级&班级	2014 级 8 班	专业（方向）	软件工程（嵌入式软件与系统）
学号	14331062	姓名	方铭
电话	18826072452	Email	fangm7@mail2.sysu.edu.cn
开始日期	2017.3.29	完成日期	2017.4.12

实验题目：Case 1:Pipeline Processor

一、实验目的

1. 认识和掌握基于精简指令集 (RISC) 的简单流水线处理器原理及其设计方法；
2. 掌握流水线处理器的 verilog 代码实现方法及其测试；
3. 分析指令集，明确指令功能、指令在 CPU 中执行各阶段中的行为；
4. 掌握较复杂的设计在 FPGA 实验板上的综合实现，并通过开发板上的 led 或数码管显示执行效果。

二、实验内容

实验步骤

1. 分析并设计处理器的数据通路和控制通路。
2. 编写设计代码并编译。
3. 软件仿真。
4. 进行硬件配置。

实验要求

- RISC CPU
- Data path 16b
- Data memory -$2^8 \times 16$b
- Size of operation set -2^5
- General register -8×16b
- Flags -NF,ZF,CF
- Control -clock, reset, enable, start
- Testing -4 bit selection for 16 bit output

实验原理

类似于 RISC 体系架构依据流水线结构设计，每条指令就被分成五个流水阶段，并且每个阶段占用固定的时间，即一个时钟周期。

处理器在设计时，将处理器的执行阶段划分为以下五个阶段:

IF Instruction Fetch，取指。从指令缓存（I-Cache）中获取下一条指令。

ID Instruction Decode（Read Register），译码（读寄存器）。翻译指令，识别操作码和操作数，从寄存器堆中读取数据到 ALU 输入寄存器。

EX Execute，执行（算术/逻辑运算）。在一个时钟周期内，完成算术或逻辑操作。注意，浮点算术运算和整数乘除运算不能在一个时钟周期内完成。

MEM Memory Access，内存数据读或者写。在该阶段，指令可以从数据缓存（D-Cache）中读/写内存变量。平均来说，大约四分之三的指令在这一阶段没有执行任何操作，为每条指令分配这个阶段是为了保证同一时刻不会有两条指令都访问数据缓存。

WB Write Back，写回。操作完成后，将计算结果从 ALU 输出寄存器写回到通用寄存器中

实验假设

1. 本实验不实现自启动的控制电路，也不提供进程切换所需引脚，不实现进程切换，读、写、内部寄存器的功能。
2. 本实验认为指令和数据有两个独立的缓存，同时读写不会有冲突。
3. 本实验认为指令地址和数据地址均已经被映射到缓存 (cache) 中，读写可在一个时钟周期内完成，从而满足流水线的要求。
4. 本实验暂时未实现自动处理冒险 (hazard)，冒险的解决方法依赖于通过软件层面汇编器的在会导致冒险的指令前插入一定数量的 nop 指令。
5. 本实验运行的时钟频率较低，是为了在实验板 (Xlinx® Spartan6 XC6LX16-CS324) 上观察调试。可以在实验板上正常工作的最大时钟频率暂未测定。

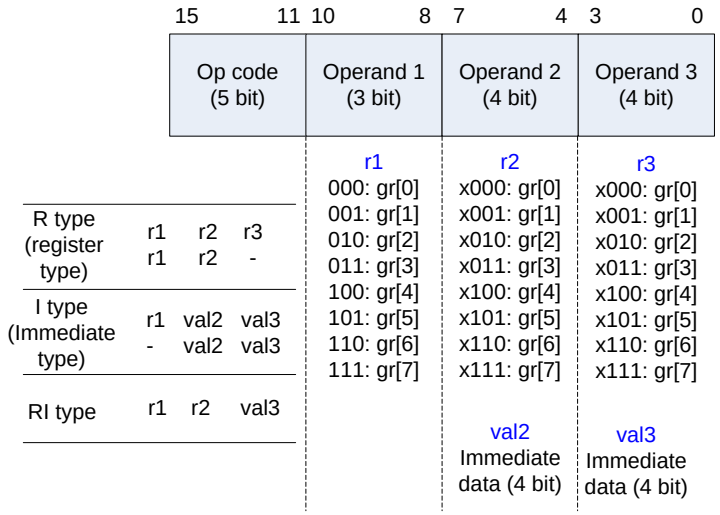


图 1: Operation Field

mnemonic	operand1	operand2	operand3	op code	operation
NOP				00000	no operation
HALT				00001	halt
LOAD	r1	r2	val3	00010	gr[r1]<-m[r2+val]
STORE	r1	r2	val3	00011	m[r2+val]<-r1
LDIH	r1	val2	val3	10000	r1<-r1+{val2,val3,8'b0}
MOV	r1	val2	val3	10100	r1<-{val2,val3}
ADD	r1	r2	r3	01000	r1<-r2+r3
ADDI	r1	val2	val3	01001	r1<-r1+{val2,val3}
ADDC	r1	r2	r3	10001	r1<-r2+r3+CF
SUB	r1	r2	r3	01010	r1<-r2-r3
SUBI	r1	val2	val3	01011	r1<-r1-{val2,val3}
SUBC	r1	r2	r3	10010	r1<-r2-r3+CF
CMP		r2	r3	01100	r2-r3;set CD,ZF and NF

表 1: Data transfer & Arithmetic Operation

mnemonic	operand1	operand2	operand3	op code	operation
AND	r1	r2	r3	01101	r1 <- r2 and r3
OR	r1	r2	r3	01110	r1 <- r2 or r3
XOR	r1	r2	r3	01111	r1 <- r2 xor r3
NOT	r1	r2		10011	r1<- r1
SL(SLL/SLA)	r1	r2	val3	00100	r1 <- r2 « val3
SRL	r1	r2	val3	00101	r1 <- r2 » val3 (logically)
SRA	r1	r2	val3	00110	r1 <- r2 » val3 (arithmetically)

表 2: Logical / shift

mnemonic	operand1	operand2	operand3	op code	operation
JUMP		val2	val3	11000	jump to {val2,val3}
JMPR	r1	val2	val3	11001	jump to r1+{val2,val3}
BZ	r1	val2	val3	11010	if ZF=1 branch to r1+{val2,val3}
BNZ	r1	val2	val3	11011	if ZF=0 branch to r1+{val2,val3}
BN	r1	val2	val3	11100	if NF=1 branch to r1+{val2,val3}
BNN	r1	val2	val3	11101	if NF=0 branch to r1+{val2,val3}
BC	r1	val2	val3	11110	if CF=1 branch to r1+{val2,val3}
BNC	r1	val2	val3	11111	if CF=0 branch to r1+{val2,val3}

表 3: Control

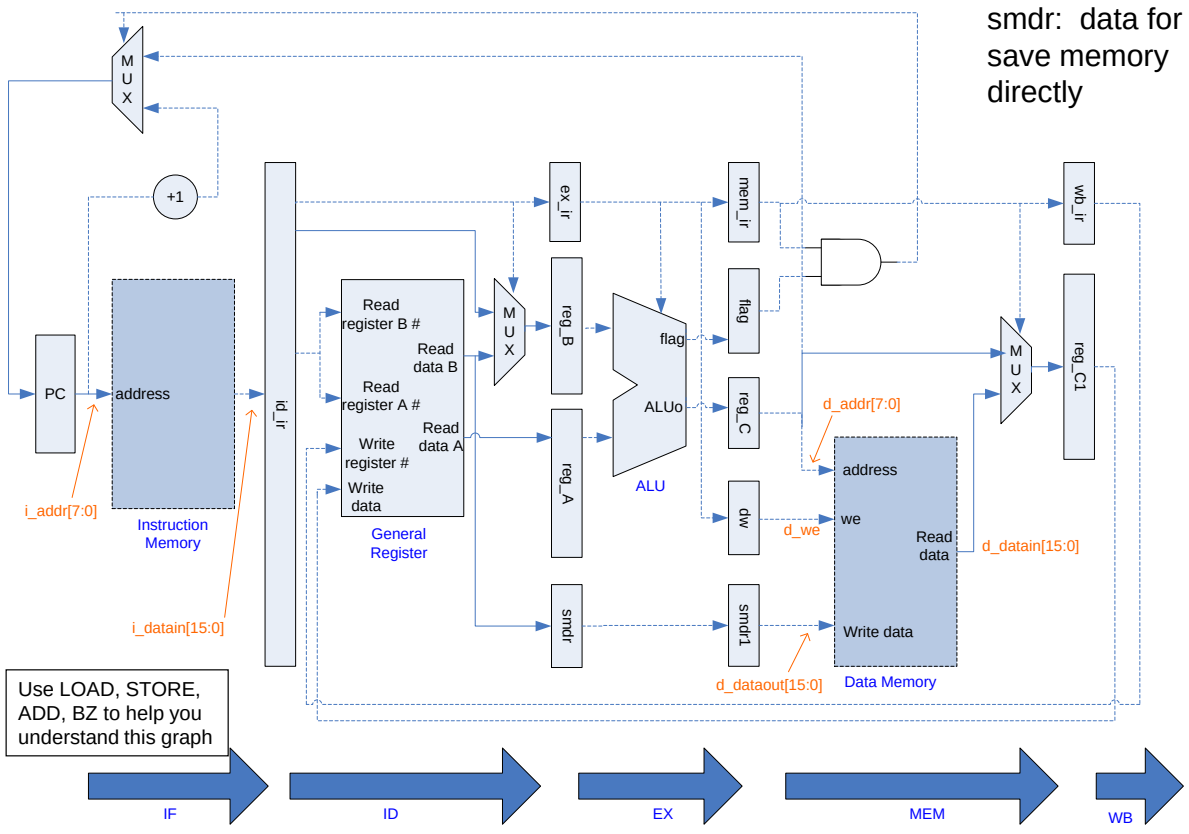


图 2: Block Diagram

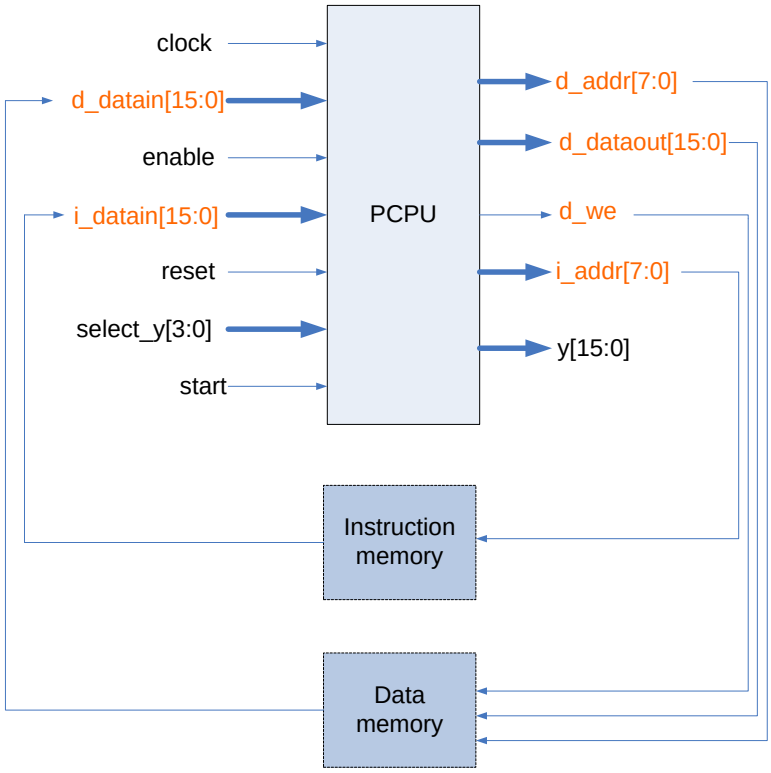


图 3: Top View

三、实验结果

3.1 综合的 RTL 电路图

顶层模块、PCPU 模块的 RTL 图见附录。
时钟分屏、数码管译码模块省略。

手动时钟模块的 RTL 图如下。

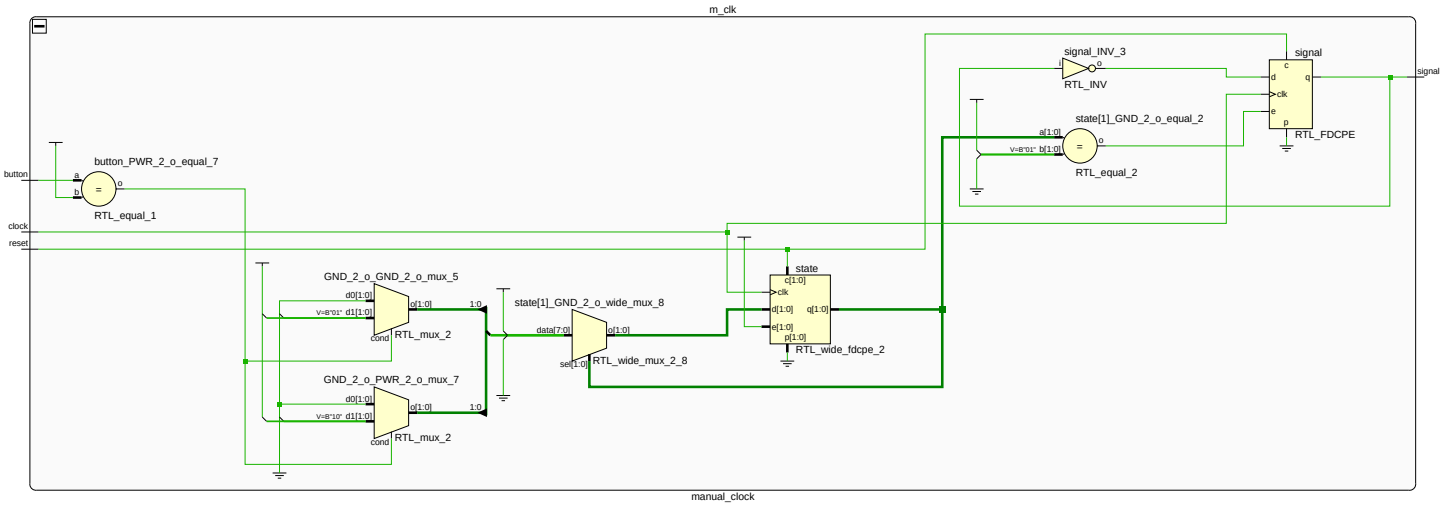


图 4: 手动时钟模块的 RTL 图（Xilinx PlanAhead 14.7 导出）

经验证，各模块综合出的 RTL 图与原理图基本一致，综合正确。

3.2 texture simulation

汇编代码 code.asm

```
0  addi  $1, 60      #0x003c
1  mov   $2, 60      #0x003c
2  load  $6, $0, 3   #0x2369
3  load  $7, $0, 7   #0x5555
4  ldih  $1, 171     #ab3c
5  jr    $0, 15
6  nop
7  nop
8  nop
9  add   $6, $6, $3  #2459
10 xor   $7, $7, $4  #5403
11 j     22
12 nop
13 nop
14 nop
15 sll   $3, $2, 2   #00f0
16 srl   $4, $1, 7   #0156
17 sra   $5, $1, 7   #ff56
18 bn    $0, 9
19 nop
20 nop
21 nop
22 store $6, $0, 0
23 store $7, $0, 1
24 halt
```

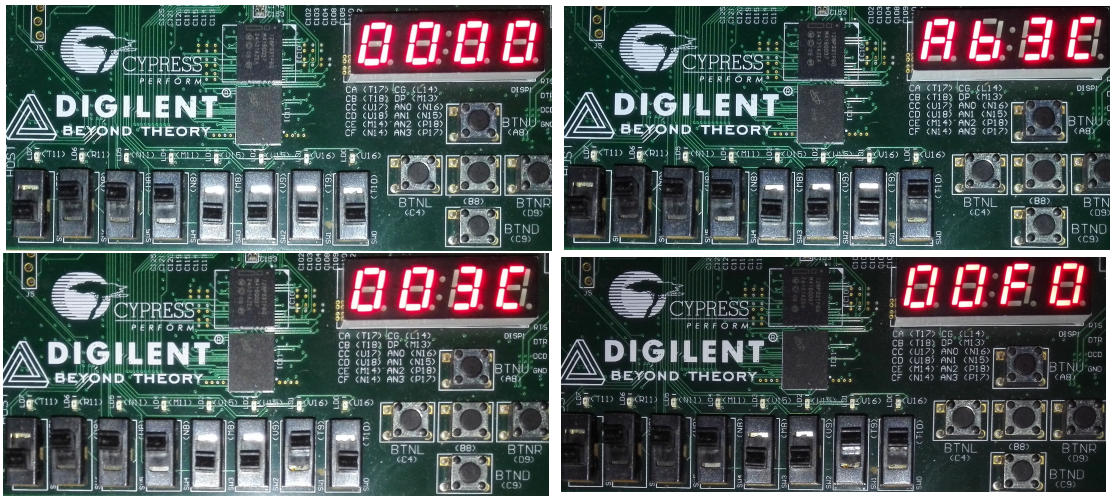
```
Console
ISim P.20131013 (signature 0x7708f090)
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
pc:     id_ir      :reg_A:reg_B:reg_C:da:dd  :w:reC1:gr1 :gr2 : gr3
00:0000000000000000:xxxx :xxxx :xxxx :xx:xxxx:x:xxxx:0000:0000:0000
01:0100100100111100:0000 :0000 :xxxx :xx:xxxx:0:xxxx:0000:0000:0000
02:1010001000111100:0000 :003c :0000 :00:0000:0:xxxx:0000:0000:0000
03:0001011000000011:0000 :003c :003c :3c:0000:0:0000:0000:0000:0000
04:0001011100000111:0000 :0003 :003c :3c:0000:0:003c:0000:0000:0000
05:1000000110101011:0000 :0007 :0003 :03:0000:0:003c:003c:0000:0000
06:1100100000001111:003c :ab00 :0007 :07:0000:0:2369:003c:003c:0000
07:0000000000000000:0000 :000f :ab3c :3c:003c:0:5555:003c:003c:0000
08:0000000000000000:0000 :0000 :000f :0f:0000:0:ab3c:003c:003c:0000
0f:0000000000000000:0000 :0000 :0000 :00:0000:0:000f:ab3c:003c:0000
10:0010001100100010:0000 :0000 :0000 :00:0000:0:0000:ab3c:003c:0000
11:0010110000010111:003c :0002 :0000 :00:0000:0:0000:ab3c:003c:0000
12:0011110100010111:ab3c :0007 :00f0 :f0:0000:0:0000:ab3c:003c:0000
13:1110000000001001:ab3c :0007 :0156 :56:0000:0:00f0:ab3c:003c:0000
14:0000000000000000:0000 :0009 :ff56 :56:0000:0:0156:ab3c:003c:00f0
15:0000000000000000:0000 :0000 :0009 :09:0000:0:ff56:ab3c:003c:00f0
09:0000000000000000:0000 :0000 :0000 :00:0000:0:0009:ab3c:003c:00f0
0a:0100011001100011:0000 :0000 :0000 :00:0000:0:0000:ab3c:003c:00f0
0b:0111111101110100:2369 :00f0 :0000 :00:0000:0:0000:ab3c:003c:00f0
0c:1100000000010110:5555 :0156 :2459 :59:2369:0:0000:ab3c:003c:00f0
0d:0000000000000000:0000 :0016 :5403 :03:5555:0:2459:ab3c:003c:00f0
0e:0000000000000000:0000 :0000 :0016 :16:0000:0:5403:ab3c:003c:00f0
16:0000000000000000:0000 :0000 :0000 :00:0000:0:0016:ab3c:003c:00f0
17:0001111000000000:0000 :0000 :0000 :00:0000:0:0000:ab3c:003c:00f0
18:0001111100000001:0000 :0000 :0000 :00:0000:0:0000:ab3c:003c:00f0
19:0000100000000000:0000 :0001 :0000 :00:2459:1:0000:ab3c:003c:00f0
1a:0000000000000000:0000 :0000 :0001 :01:5403:1:0000:ab3c:003c:00f0
1b:0000000000000000:0000 :0000 :0000 :00:0000:0:0001:ab3c:003c:00f0
1c:0000000000000000:0000 :0000 :0000 :00:0000:0:0000:ab3c:003c:00f0
ISim>
Console | Compilation Log | Breakpoints | Find in Fil
```

图 5: Simulation results

经分析对比，pc 寄存器和其他寄存器结果、控制信合结果均正确，仿真测试通过。

3.3 开发板的显示效果

管脚约束定义见附录。烧入以上程序，全部运行结束后，通用寄存器的结果显示如下：



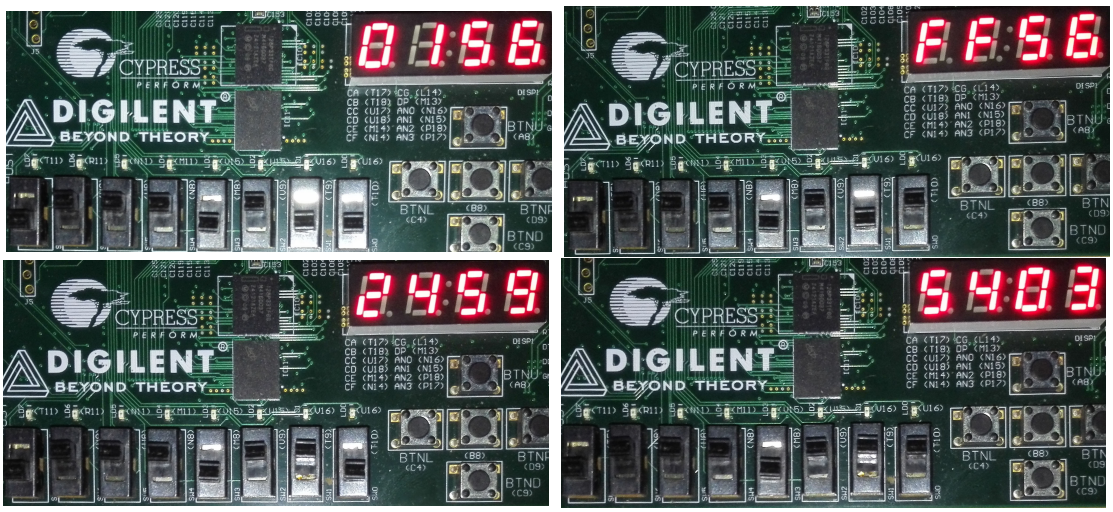


图 6: 通用寄存器 gr[0:7] 的最终结果

四、实验感想

本次实验初步实现了一个基于精简指令集的流水线 CPU，进一步提高了我设计并实现较复杂硬件系统的能力。

我原先计划先译码得到各个阶段的控制信号，再对控制信号进行移位，从而并发地控制各级流水单元。后来参考了老师给出的设计图，发现对指令进行移位保存也能实现，而且不容易出错，更适合高级硬件设计语言来描述。虽然理论上，前者消耗的寄存器，门电路会更少，但是通过 ISE 实现时，还有着一些时间、面积、功耗的约束影响，ISE 在特定的约束条件下，对原始的 verilog 代码优化后，只要在语言描述上是等价的，那么综合出的设计就是几乎一样的，因此我还是认同了第二种方案。虽然类似于 smdr，smdr1 这种寄存器的声明，看似耗费的资源较多，但实际上，是会被优化掉的。

以往设计的 CPU 的经验在本次实验中也得到的了应用，即尽可能少的增加 if 条件判断，对一些赋不赋值均可的寄存器，如果能减少条件判断，就应该赋值，减少寄存器写使能信号的组合逻辑的复杂度。某些指令与 ALU 无关，与读写内存无关，只要控制好读写使能信号，就不必要对的这些寄存器清零。

本次实验基本上没有用到新的知识点，但对逻辑左移右移和算术左移右移之间的区别，以及加减法进位借位的实现不是很有把握，为了减少以后的测试复杂性，我还是事先新建了测试文件，先实现了这些运算的功能，仿真成功后，再写入到 PCPU 模块去。经过预先测试，发现算术右移一定要声明为 signed 类型或者通过 $signed() 系统函数进行强制类型转换才能实现，否则右移补零和逻辑右移一样。

经过实验验证并结合我的分析，我认为：使用 always @* 语句生成组合电路时，不完整的条件语句会导致 latch，但是 always @(posedge clk) 结合非阻塞赋值 <= 生成的触发器可以没有完整的条件分支。因为这些语句生成的是触发器，分支中赋值的语句的右值作为数据选择器的输入，判断条件通过编码变成一个个选择信号，如果有不完整的条件分支就需要为触发器加一个使能端，为以上条件使能；如果条件分支完整，那么这个触发器不需要使能端，或者令使能端一直处于使能状态。本实验中，我在设计触发器部分时，由于需要保持原来的数据，条件分支没有写完整，但最后也没有任何 warning 提示生成了 latch；相反在利用 always @* 设计的组合逻辑部分，如果在某个条件分支里写了类似 x = x 这样的代码，即使条件完整，也会有 warning 提示生成了 latch。

原以为这次实验仿真效果可能会与综合效果不同，所以我比较谨慎地消除了几乎所有的 warning。初步设计完后，我直接在板子上综合了，综合出现的问题，后来一仿真也有同样的问题。因此，如果设计代码在综合后没有大量 warning 的话，仿真的效果是和综合效果基本是同步的，仿真的结果是可以信赖的。

实验中使用了 Core_Gernerate 生成了 IP，使用了 coe 文件初始化内存。根据我的理解，IP 可以生成一些高级模块的代码，从而进一步完成更加复杂的设计，比如生成一些浮点运算的模块。这可以帮助设计人员更加快速设计一些硬件系统。

附录

部分图片受版面限制，打印后可能导致阅读不便。此份报告使用的图片均为矢量图，可浏览电子版文档放大后阅读。本次实验的项目文件、汇编指令解释器及 coe 生成程序及实验报告可访问 https://github.com/SimonFang1/Pipeline-RISC-CPU/查看。

A 较复杂的 RTL 电路图

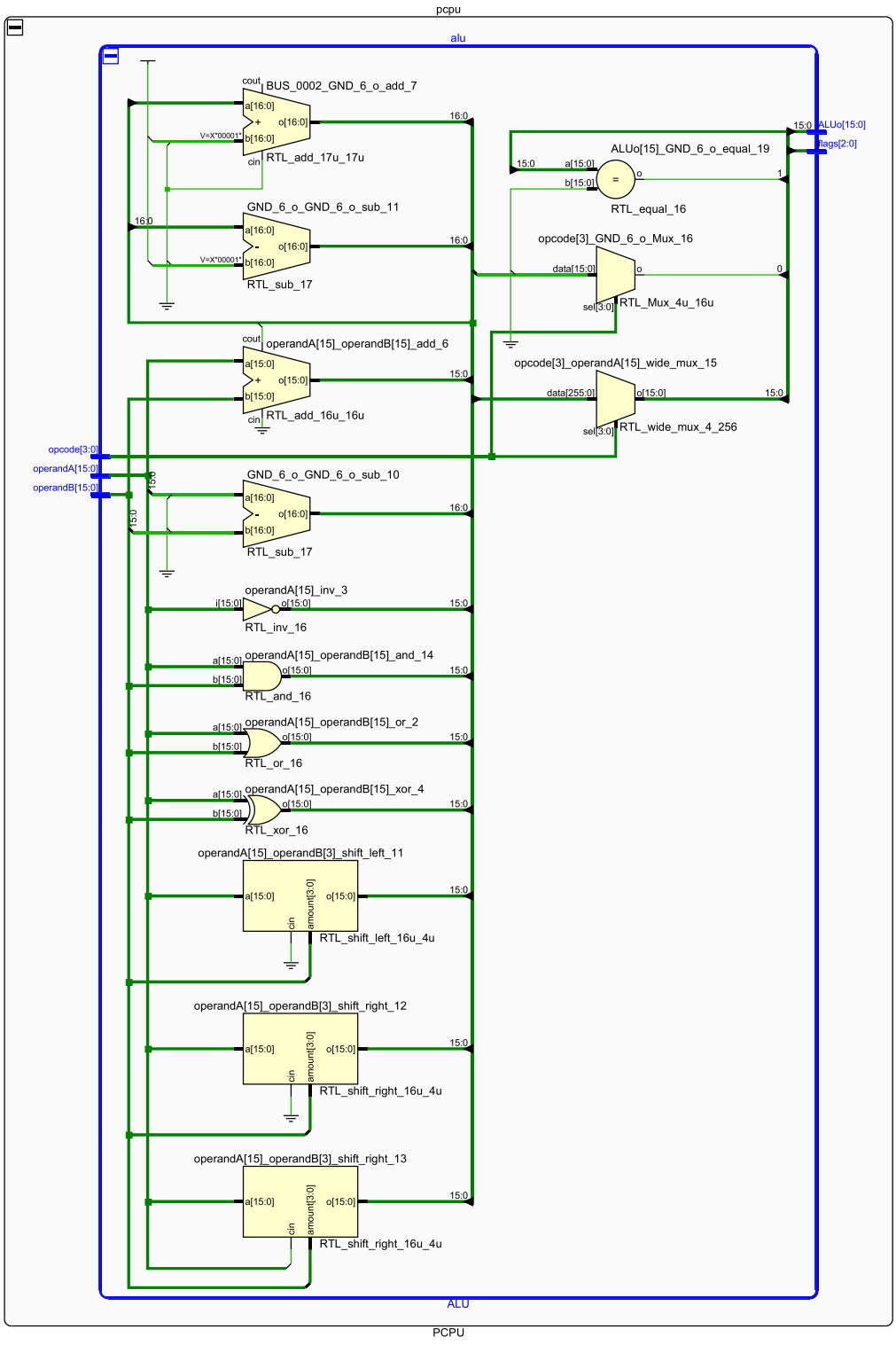


图 7: ALU 模块的 RTL 图（Xilinx PlanAhead 14.7 导出）

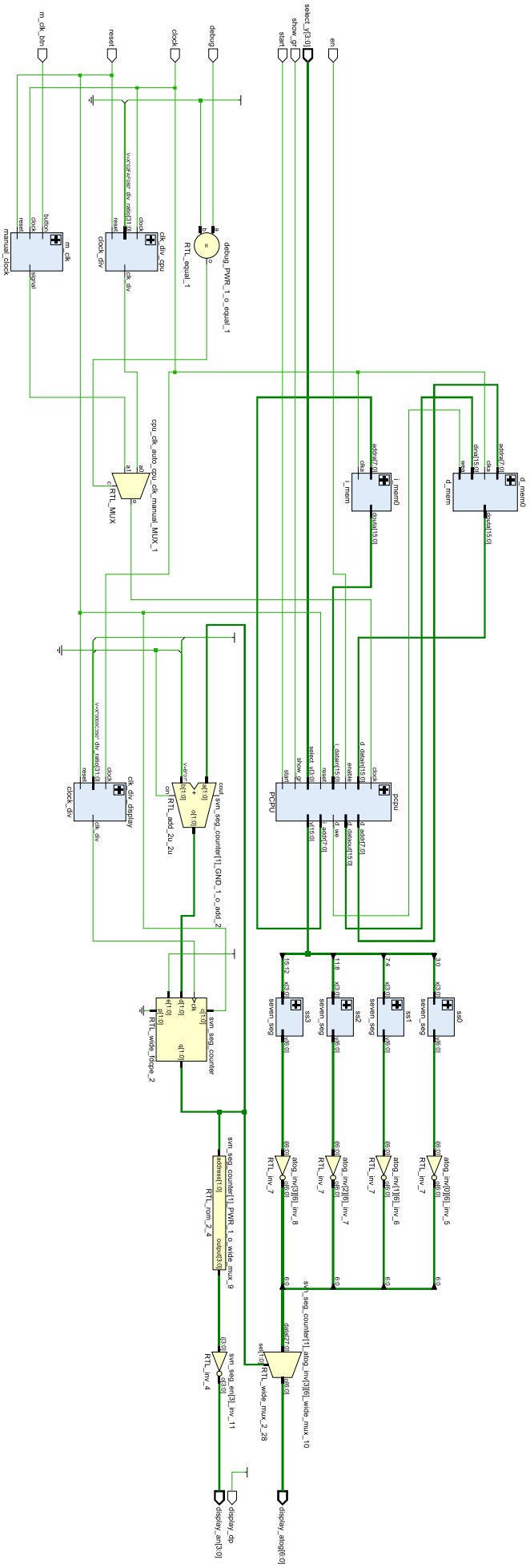


图 8: 顶层模块的 RTL 图（Xilinx PlanAhead 14.7 导出）

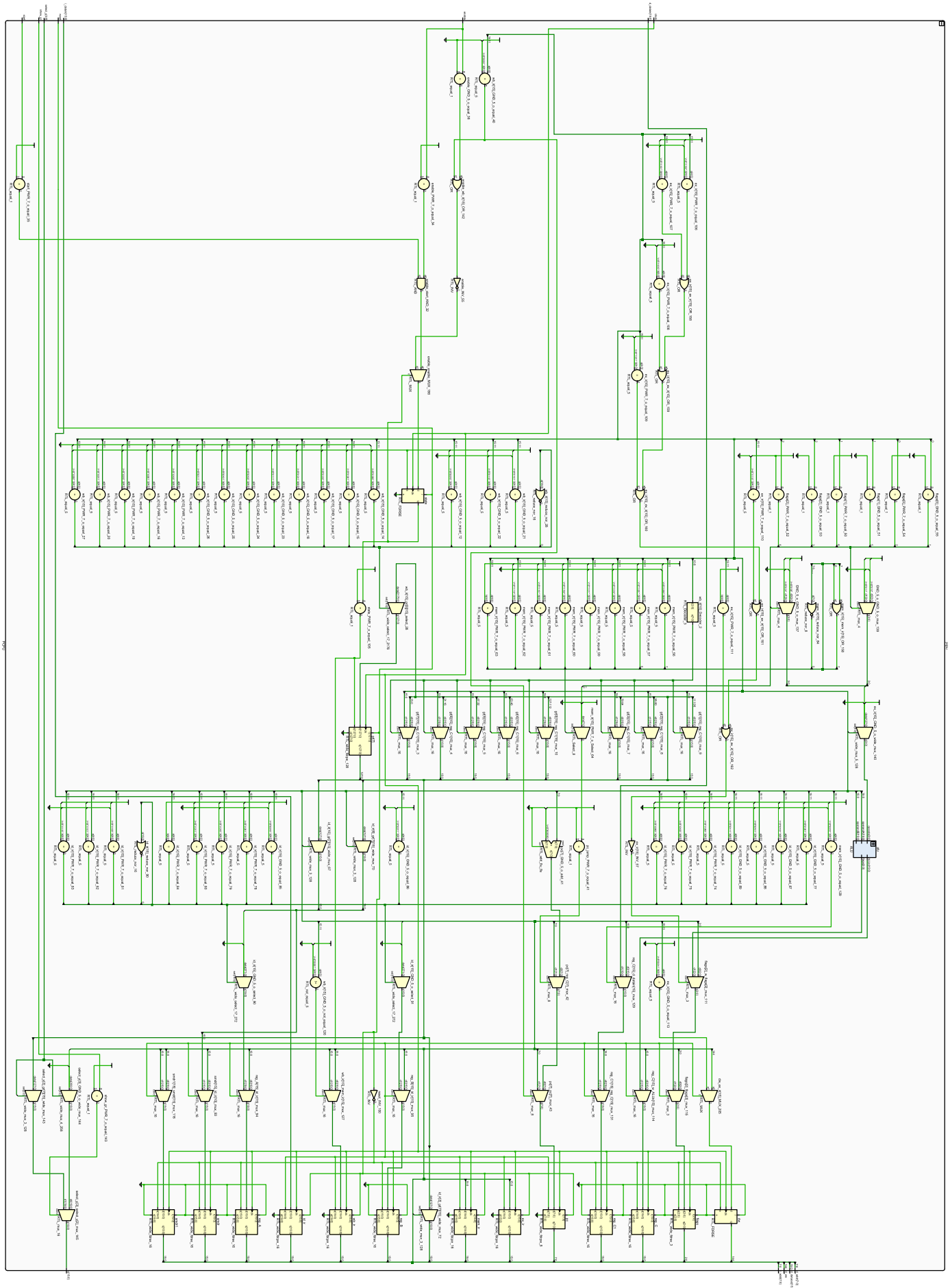


图 9: PCPU 模块的 RTL 图（Xilinx PlanAhead 14.7 导出）

B 实验设计代码

header.v

```
`ifndef __HEADER_V__
`define __HEADER_V__

// CPU state
`define idle        1'b0
`define exec        1'b1

// operation code
//Data transfer & Arithmetic
`define NOP         5'b00000
`define HALT        5'b00001
`define LOAD        5'b00010
`define STORE       5'b00011
`define LDIH        5'b10000
`define MOV         5'b10100  // *
`define ADD         5'b01000
`define ADDI        5'b01001
`define ADDC        5'b10001
`define SUB         5'b01010  // *
`define SUBI        5'b01011  // *
`define SUBC        5'b10010  // *
`define CMP         5'b01100
//Logical / shift
`define AND         5'b01101
`define OR          5'b01110  // *
`define XOR         5'b01111  // *
`define NOT         5'b10011  // * bitwise inverse
`define SL          5'b00100
`define SLL         5'b00100  // pseudo-instruction
`define SRL         5'b00101  // *
`define SLA         5'b00100  // pseudo-instruction
`define SRA         5'b00111  // *
//Control
`define JUMP        5'b11000
`define JMPR        5'b11001
`define BZ          5'b11010
`define BNZ         5'b11011
`define BN          5'b11100
`define BNN         5'b11101  // *
`define BC          5'b11110
`define BNC         5'b11111  // *


// ALU operations
`define A_OR        4'b0000
`define A_AND       4'b0001
`define A_NOT       4'b0010
`define A_XOR       4'b0011
`define A_ADD       4'b0100
`define A_ADDPLS    4'b0101
`define A_SUB       4'b0110
`define A_SUBMNS    4'b0111
`define A_SL        4'b1000
`define A_SRL       4'b1001
`define A_SRA       4'b1010

//instruction segment
`define I_OP        15:11
`define I_R1        10:8
`define I_R2        6:4
`define I_R3        2:0
`define I_VAL2      7:4
```

```
63 `define I_VAL3        3:0
64 `define I_IMDT        7:0
65
66 //gr
67 `define gr0 3'b000
68 `define gr1 3'b001
69 `define gr2 3'b010
70 `define gr3 3'b011
71 `define gr4 3'b100
72 `define gr5 3'b101
73 `define gr6 3'b110
74 `define gr7 3'b111
75
76 `endif
```

top.v

```
1  `timescale 1ns / 1ps
2  module top(
3      input clock,
4       input en,
5       input reset,
6       input start,
7       input debug,
8       input show_gr,
9       input m_clk_btn,
10      input [3:0] select_y,
11      output reg [6:0] display_atog,
12      output display_dp,
13      output [3:0] display_an
14     );
15
16      wire cpu_clk_auto, cpu_clk_manual, cpu_clk;
17      wire display_clk;
18
19      wire [15:0] d_datain, i_datain;
20     wire [7:0] i_addr, d_addr;
21     wire [15:0] d_dataout;
22     wire d_we;
23     wire [15:0] pcpu_y;
24
25
26      assign cpu_clk = debug == 1'b1 ? cpu_clk_manual : cpu_clk_auto;
27     manual_clock m_clk(
28         .clock(clock),
29          .reset(reset),
30          .button(m_clk_btn),
31          .signal(cpu_clk_manual)
32     );
33      clock_div clk_div_cpu(
34         .clock(clock),
35          .reset(reset),
36          .div_ratio(32'd50_000_000), // 1s
37          .clk_div(cpu_clk_auto)
38     );
39      clock_div clk_div_display(
40         .clock(clock),
41          .reset(reset),
42          .div_ratio(32'd50_000), // 1ms
43          .clk_div(display_clk)
44     );
45
46      // 7-segment
47      reg [1:0] svn_seg_counter;
48      reg [3:0] svn_seg_en;
49      wire [6:0] atog_inv [3:0];
```

```
50      always @(posedge display_clk or posedge reset) begin
51          if (reset)
52              svn_seg_counter <= 0;
53          else
54              svn_seg_counter <= svn_seg_counter + 1'b1;
55      end
56      always @(*) begin
57          case(svn_seg_counter)
58              2'd0: begin svn_seg_en = 4'b0001; display_atog = ~atog_inv[0]; end
59              2'd1: begin svn_seg_en = 4'b0010; display_atog = ~atog_inv[1]; end
60              2'd2: begin svn_seg_en = 4'b0100; display_atog = ~atog_inv[2]; end
61              2'd3: begin svn_seg_en = 4'b1000; display_atog = ~atog_inv[3]; end
62           endcase
63      end
64      assign display_an = ~svn_seg_en;
65      assign display_dp = 1'b1;
66      seven_seg ss0(
67          .x(pcpu_y[3:0]),
68         .y(atog_inv[0])
69      );
70      seven_seg ss1(
71          .x(pcpu_y[7:4]),
72         .y(atog_inv[1])
73      );
74      seven_seg ss2(
75          .x(pcpu_y[11:8]),
76         .y(atog_inv[2])
77      );
78      seven_seg ss3(
79          .x(pcpu_y[15:12]),
80         .y(atog_inv[3])
81      );
82
83      //
84      PCPU pcpu(
85          .clock(cpu_clk),
86            .reset(reset),
87           .d_datain(d_datain),
88           .enable(en),
89           .i_datain(i_datain),
90            .show_gr(show_gr),
91           .select_y(select_y),
92           .start(start),
93           .d_addr(d_addr),
94           .d_dataout(d_dataout),
95            .d_we(d_we),
96            .i_addr(i_addr),
97            .y(pcpu_y)
98      );
99
100     i_mem i_mem0(
101       .clka(clock),
102       .addra(i_addr),
103       .douta(i_datain)
104    );
105
106     d_mem d_mem0 (
107         .clka(clock),
108           .wea(d_we),
109           .addra(d_addr),
110           .dina(d_dataout),
111           .douta(d_datain)
112     );
113
114 endmodule
```

manual_clock.v

```
1  `timescale 1ns / 1ps
2  module manual_clock(
3      input clock,
4       input reset,
5       input button,
6       output reg signal
7      );
8
9      parameter RESET = 2'b00, SET = 2'b01, LOCK = 2'b10;
10      reg [1:0] state, nextstate;
11
12     always @(posedge clock or posedge reset) begin
13         if(reset) begin
14               signal <= 0;
15             state <= RESET;
16         end else begin
17             state <= nextstate;
18                 if (state == SET)
19                   signal <= ~signal;
20           end
21     end
22      always @(*) begin
23         case(state)
24             RESET:
25                 if (button == 1'b1) nextstate <= SET;
26                 else  nextstate <= RESET;
27             SET: nextstate <= LOCK;
28             LOCK:
29                 if (button == 1'b1) nextstate <= LOCK;
30                 else nextstate <= RESET;
31             default: nextstate <= RESET;
32         endcase
33     end
34
35 endmodule
```

PCPU.v

```
1  `timescale 1ns / 1ps
2  `include "header.v"
3  module PCPU(
4      input clock,
5      input [15:0] d_datain,
6      input enable,
7      input [15:0] i_datain,
8      input reset,
9      input [3:0] select_y,
10     input start,
11     output [7:0] d_addr,
12     output [15:0] d_dataout,
13     output d_we,
14     output [7:0] i_addr,
15     output reg [15:0] y,
16     input show_gr
17     );
18
19     reg [7:0] pc;
20     reg [15:0] id_ir, ex_ir, mem_ir, wb_ir;
21     reg [15:0] reg_A, reg_B, reg_C, reg_C1;
22     reg [15:0] smdr, smdr1;
23     reg [2:0] flags; // flag[0]-CF, flag[1]-ZF, flag[2]-NF
24     `define CF      flags[0]
25     `define ZF      flags[1]
26     `define NF      flags[2]
27     reg dw;
```

```
28
29      wire [15:0] w_ALUo;
30      wire [2:0] w_flags;
31
32      assign d_dataout = smdr1;
33      assign d_we = dw;
34      assign d_addr = reg_C[7:0];
35      assign i_addr = pc;
36
37       //*********** General Register **********//
38       reg [15:0] gr[0:7];
39       //************ WB ************//
40      always @(posedge clock or posedge reset) begin
41          if (reset) begin
42                gr[0] <= 0; gr[1] <= 0; gr[2] <= 0; gr[3] <= 0;
43                  gr[4] <= 0; gr[5] <= 0; gr[6] <= 0; gr[7] <= 0;
44          end else if (state ==`exec) begin
45              case(wb_ir[`I_OP])
46                  `LOAD, `MOV,
47                  `ADD, `ADDI, `ADDC,
48                  `SUB, `SUBI, `SUBC,
49                  `NOT, `AND, `OR, `XOR,
50                  `SL, `SRL, `SRA,
51                  `LDIH: gr[wb_ir[`I_R1]] <= reg_C1;
52              endcase
53          end
54      end
55
56      //************ CPU control ************//
57      reg state, next_state;
58      always @(posedge clock) begin
59          if (reset)
60              state <= `idle;
61          else
62              state <= next_state;
63      end
64
65      always @(*) begin
66          case (state)
67              `idle :
68                  if ((enable == 1'b1)
69                  && (start == 1'b1))
70                      next_state = `exec;
71                  else
72                      next_state = `idle;
73              `exec :
74                  if ((enable == 1'b0)
75                       || wb_ir[`I_OP] == `HALT)
76                      next_state = `idle;
77                  else
78                      next_state = `exec;
79          endcase
80      end
81
82      //************ IF ************//
83      reg pc_jump;
84      always @(posedge clock or posedge reset) begin
85          if (reset) begin
86              id_ir <= 16'b0000_0000_0000_0000;
87              pc <= 8'b0000_0000;
88          end else if (state ==`exec) begin
89              id_ir <= i_datain;
90                  if (wb_ir[`I_OP] == `HALT)
91                      pc <= pc;
92                  else if (pc_jump == 1'b1)
93                      pc <= reg_C[7:0];
```

```
94                  else
95                      pc <= pc + 1'b1;
96          end
97      end
98      always @(*) begin
99          case(mem_ir['I_OP])
100             'JUMP,'JMPR: pc_jump = 1'b1;
101             'BZ: pc_jump = ('ZF == 1'b1) ? 1'b1 : 1'b0;
102             'BNZ: pc_jump = ('ZF == 1'b0) ? 1'b1 : 1'b0;
103             'BN: pc_jump = ('NF == 1'b1) ? 1'b1 : 1'b0;
104             'BNN: pc_jump = ('NF == 1'b0) ? 1'b1 : 1'b0;
105             'BC: pc_jump = ('CF == 1'b1) ? 1'b1 : 1'b0;
106             'BNC: pc_jump = ('CF == 1'b0) ? 1'b1 : 1'b0;
107             default: pc_jump = 1'b0;
108         endcase
109     end
110
111     //************* ID *************//
112     always @(posedge clock or posedge reset) begin
113         if (reset) begin
114             ex_ir <= 16'd0;
115         end else if (state =='exec) begin
116             ex_ir <= id_ir;
117             smdr <= gr[id_ir['I_R1]];
118             case(id_ir['I_OP])
119                 'JUMP,'MOV: begin
120                     reg_A <= 16'd0;
121                     reg_B <= id_ir['I_IMDT];
122                 end
123                 'LDIH: begin
124                     reg_A <= gr[id_ir['I_R1]];
125                     reg_B <= {id_ir['I_IMDT], 8'd0};
126                     end
127                 'ADDI,'JMPR,'BZ,'BNZ,'BN,
128                 'BNN,'BC,'BNC: begin
129                     reg_A <= gr[id_ir['I_R1]];
130                     reg_B <= id_ir['I_IMDT];
131                 end
132                 'LOAD,'STORE,'SL,
133                 'SRL,'SRA: begin
134                     reg_A <= gr[id_ir['I_R2]];
135                     reg_B <= id_ir['I_VAL3];
136                 end
137                 default: begin
138                     reg_A <= gr[id_ir['I_R2]];
139                     reg_B <= gr[id_ir['I_R3]];
140                 end
141             endcase
142         end
143     end
144
145     //************* EX *************//
146     always @(posedge clock or posedge reset) begin
147         if (reset) begin
148             mem_ir <= 16'd0;
149         end else if (state =='exec) begin
150             mem_ir <= ex_ir;
151             reg_C <= w_ALUo;
152             if (!(ex_ir['I_OP] == 'BN || ex_ir['I_OP] == 'BNN ||
153                   ex_ir['I_OP] == 'BZ || ex_ir['I_OP] == 'BNZ ||
154                   ex_ir['I_OP] == 'BC || ex_ir['I_OP] == 'BNC))
155                 flags <= w_flags;
156                 smdr1 <= smdr;
157             if (ex_ir['I_OP] == 'STORE)
158                 dw <= 1'b1;
159             else
```

```
160                 dw <= 1'b0;
161             end
162         end
163
164         //************* MEM *************//
165         always @(posedge clock or posedge reset) begin
166             if (reset) begin
167                 wb_ir <= 16'd0;
168             end else if (state ==`exec) begin
169                 if (wb_ir[`I_OP] != `HALT)
170                     wb_ir <= mem_ir;
171                 if (mem_ir[`I_OP] == `LOAD)
172                     reg_C1 <= d_datain;
173                 else
174                     reg_C1 <= reg_C;
175             end
176         end
177
178         //ALU
179         reg [3:0] ALUop;
180         always @(ex_ir or flags) begin
181             case(ex_ir[`I_OP])
182                 `LOAD,`STORE,`LDIH,
183                 `BZ,`BNZ,`BN,`BNN,
184                 `BC,`BNC,`ADD,
185                 `ADDI: ALUop = `A_ADD;
186                 `ADDC: ALUop = `CF == 1'b1 ? `A_ADDPLS : `A_ADD;
187                 `SUB,`SUBI,
188                 `CMP: ALUop = `A_SUB;
189                 `SUBC: ALUop = `CF == 1'b1 ? `A_SUBMNS : `A_SUB;
190                 `AND: ALUop = `A_AND;
191                 `XOR: ALUop = `A_XOR;
192                 `NOT: ALUop = `A_NOT;
193                 `SL: ALUop = `A_SL;
194                 `SRL: ALUop = `A_SRL;
195                 `SRA: ALUop = `A_SRA;
196                 `OR,`JUMP,`MOV: ALUop = `A_OR;
197                 default: ALUop = `A_OR;
198             endcase
199         end
200
201         ALU alu(
202             .opcode(ALUop),
203             .operandA(reg_A),
204             .operandB(reg_B),
205             .ALUo(w_ALUo),
206             .flags(w_flags)
207         );
208
209         //*********** Output ***********//
210         always @(*) begin
211             if (show_gr == 1'b1) begin
212                 case (select_y[2:0])
213                     3'b000: y = gr[0];
214                     3'b001: y = gr[1];
215                     3'b010: y = gr[2];
216                     3'b011: y = gr[3];
217                     3'b100: y = gr[4];
218                     3'b101: y = gr[5];
219                     3'b110: y = gr[6];
220                     3'b111: y = gr[7];
221                 endcase
222             end else begin
223                 case (select_y)
224                     4'd0: y = pc;     //{8'b0000_0000, pc};
225                     4'd1: y = id_ir;
```

```
                4'd2: y = ex_ir;
                4'd3: y = mem_ir;
                4'd4: y = wb_ir;
                4'd5: y = reg_A;
                4'd6: y = reg_B;
                4'd7: y = reg_C;
                4'd8: y = flags;
                4'd9: y = dw;
                4'd10: y = smdr;
                4'd11: y = smdr1;
                4'd12: y = reg_C1;
                default: y = pc;
            endcase
        end
    end
endmodule
```

ALU.v

```
`timescale 1ns / 1ps
`include "header.v"
module ALU(
    input [3:0] opcode,
    input [15:0] operandA,
    input [15:0] operandB,
    output reg [15:0] ALUo,
    output reg [2:0] flags
    );

    `define CF      flags[0]
    `define ZF      flags[1]
    `define NF      flags[2]

    always @(opcode or operandA or operandB) begin
        case(opcode)
            `A_AND: begin ALUo = operandA & operandB; `CF = 0; end
            `A_OR: begin ALUo = operandA | operandB; `CF = 0; end
            `A_NOT: begin ALUo = ~operandA; `CF= 0; end
            `A_XOR: begin ALUo = operandA ^ operandB; `CF = 0; end
            `A_ADD: {`CF, ALUo} = operandA + operandB;
            `A_ADDPLS: {`CF, ALUo} = operandA + operandB + 1'b1;
            `A_SUB: {`CF, ALUo} = operandA - operandB;
            `A_SUBMNS: {`CF, ALUo} = operandA - operandB - 1'b1;
            `A_SL: begin ALUo = operandA << operandB[3:0]; `CF = 0; end
            `A_SRL: begin ALUo = operandA >> operandB[3:0]; `CF = 0; end
            `A_SRA: begin ALUo = $signed(operandA) >>> operandB[3:0]; `CF = 0; end
            default: begin ALUo = operandA & operandB; `CF = 0; end
        endcase
    end

    always @(ALUo) begin
        `NF = ALUo[15];
        `ZF = ALUo == 16'd0;
    end

endmodule
```

pcpu_tb.v

```
`timescale 1ns / 1ps
`include "header.v"
module pcpu_tb;
    // Inputs
    reg clock;
    reg enable;
    reg reset;
```

```
8       reg [3:0] select_y;
9       reg start;
10      reg show_gr;
11
12      // Outputs
13      wire [7:0] d_addr;
14      wire [15:0] d_dataout;
15      wire d_we;
16      wire [7:0] i_addr;
17      wire [15:0] i_datain;
18      wire [15:0] d_datain;
19      wire [15:0] y;
20
21      // Instantiate the Unit Under Test (UUT)
22      PCPU pcpu (
23          .clock(clock),
24          .d_datain(d_datain),
25          .enable(enable),
26          .i_datain(i_datain),
27          .reset(reset),
28          .select_y(select_y),
29          .start(start),
30          .d_addr(d_addr),
31          .d_dataout(d_dataout),
32          .d_we(d_we),
33          .i_addr(i_addr),
34          .y(y),
35          .show_gr(show_gr)
36      );
37
38      imem imem0(
39          .address(i_addr),
40          .q(i_datain)
41      );
42
43      dmem dmem0(
44          .clock(clock),
45          .address(d_addr),
46          .we(d_we),
47          .data(d_dataout),
48          .q(d_datain)
49      );
50
51      initial begin
52          // Initialize Inputs
53          clock = 0;
54          enable = 0;
55          reset = 0;
56          select_y = 0;
57          start = 0;
58          show_gr = 0;
59          #10 reset = 1;
60          #10 reset = 0;
61
62          // Wait 100 ns for global reset to finish
63          #100;
64
65          // Add stimulus here
66          $display("pc:     id_ir      :reg_A:reg_B:reg_C:da:dd  :w:reC1:gr1 :gr2 :
                gr3");
67          $monitor(
68                  "%h:%b:%h :%h :%h :%h:%h:%b:%h:%h:%h:%h",
69                  pcpu.pc, pcpu.id_ir, pcpu.reg_A, pcpu.reg_B, pcpu.reg_C,
70                  d_addr, d_dataout, d_we, pcpu.reg_C1, pcpu.gr[1], pcpu.gr[2],
                        pcpu.gr[3]
71          );
```

```
72          start = 1;
73          enable = 1;
74          #10;
75
76      end
77      always #5 clock = ~clock;
78
79  endmodule
80
81  module imem(
82      input [7:0] address,
83       output [15:0] q
84      );
85      reg [15:0] ram[0:255];
86      assign q = ram[address];
87      initial begin
88          $readmemb("ipcore_dir/i_mem.mif", ram);
89      end
90  endmodule
91
92  module dmem(
93       input clock,
94       input [7:0] address,
95       input we,
96       input [15:0] data,
97       output [15:0] q
98      );
99      reg [15:0] ram[0:255];
100     assign q = ram[address];
101     always @(posedge clock) begin
102         if (we)
103             ram[address] <= data;
104     end
105     initial begin
106          $readmemb("ipcore_dir/d_mem.mif", ram);
107     end
108 endmodule
```

pin.ucf

```
1  NET clock               LOC = V10;
2  NET en                  LOC = V8;
3  NET start               LOC = U8;
4  NET show_gr             LOC = N8;
5  NET debug               LOC = T5;
6  NET reset               LOC = A8;
7  NET m_clk_btn           LOC = B8;
8  NET select_y<0>         LOC = T10;
9  NET select_y<1>         LOC = T9;
10 NET select_y<2>         LOC = V9;
11 NET select_y<3>         LOC = M8;
12 NET display_atog<0>     LOC = T17;
13 NET display_atog<1>     LOC = T18;
14 NET display_atog<2>     LOC = U17;
15 NET display_atog<3>     LOC = U18;
16 NET display_atog<4>     LOC = M14;
17 NET display_atog<5>     LOC = N14;
18 NET display_atog<6>     LOC = L14;
19 NET display_dp          LOC = M13;
20 NET display_an<0>       LOC = N16;
21 NET display_an<1>       LOC = N15;
22 NET display_an<2>       LOC = P18;
23 NET display_an<3>       LOC = P17;
```

C 其他代码及脚本

文件目录

- **tools**
  - **assembler**
    - opcode.h
    - expression.h
    - assembler.cpp
  - **coe_generator**
    - gcoe.cpp
  - **mem**
    - mem.cpp
  - assembler.exe
  - gcoe.exe
  - mem.exe
  - compilerall.bat
  - makeicoe.bat
  - makedcoe.bat
  - code.asm
  - def_d_mem.txt
  - i_mem.mif
  - i_mem.coe
  - d_mem.mif
  - d_mem.coe

opcode.h

```
#ifndef __ASM_HEADER_H__
#define __ASM_HEADER_H__

#define NOP     "00000"
#define HALT    "00001"
#define LOAD    "00010"
#define STORE   "00011"
#define LDIH    "10000"
#define MOV     "10100"
#define ADD     "01000"
#define ADDI    "01001"
#define ADDC    "10001"
#define SUB     "01010"
#define SUBI    "01011"
#define SUBC    "10010"
#define CMP     "01100"

#define AND     "01101"
#define OR      "01110"
#define XOR     "01111"
#define NOT     "10011"
#define SL      "00100"
#define SLL     "00100"
#define SRL     "00101"
#define SLA     "00100"
#define SRA     "00111"

#define JUMP    "11000"
#define JMPR    "11001"
#define BZ      "11010"
#define BNZ     "11011"
#define BN      "11100"
#define BNN     "11101"
#define BC      "11110"
#define BNC     "11111"

#endif
```

expression.h

```
#ifndef __ASM_EXPRESSION_H__
#define __ASM_EXPRESSION_H__

#define reg(x) DOLLAR >> x

#define R1_R2_R3 reg(R1) >> COMMA >> reg(R2) >> COMMA >> reg(R3)
#define R1_R2_V3 reg(R1) >> COMMA >> reg(R2) >> COMMA >> VAL3
// #define R1_V2_V3 reg(R1) >> COMMA >> VAL2 >> COMMA >> VAL3
#define R1_IM    reg(R1) >> COMMA >> IMDT
#define R1_R2    reg(R1) >> COMMA >> reg(R2)
#define R2_R3    reg(R2) >> COMMA >> reg(R3)

#endif
```

assembler.cpp (partly)

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

#define INIT_ALL_MEM
#ifdef INIT_ALL_MEM
#define TOTAL_DEPTH 256
#endif

#include "opcode.h"
#define DOLLAR ch
#define COMMA ch
#define R1 r1
#define R2 r2
#define R3 r3
#define VAL2 val2
#define VAL3 val3
#define IMDT immediate
#include "expression.h"

string getBin(int n, int bits) {
    string bin;
    for (int i = bits - 1; i >= 0; i--) {
        bin += (n & (1 << i)) ? '1' : '0';
    }
    return bin;
}

int main() {
    string asmbl, op, machineCode;
    stringstream ss;
    char ch;
    int r1, r2, r3;
    int val2, val3, immediate;
    int line = 0;
    while (getline(cin, asmbl)) {
        ss << asmbl;
        ss >> op;
        if (op == "nop") {
            machineCode = NOP + getBin(0, 11);
        }else if (op == "load") {
            ss >> R1_R2_V3;
            machineCode = LOAD + getBin(r1, 3) + getBin(r2, 4) + getBin(val3, 4);
        } else if (op == "mov" || op == "ldi") {
            ss >> R1_IM;
            machineCode = MOV + getBin(r1, 3) + getBin(immediate, 8);
        }
        ... // other instructions
```

```
50          else {
51              cerr << line + 1 << ": unknown instruction \"" << asmbl << "\""<< endl;
52              machineCode = NOP + getBin(0, 11);
53          }
54          cout << machineCode << endl;
55          line++;
56          ss.clear();
57          ss.str("");
58      }
59  #ifdef INIT_ALL_MEM
60      int rest = TOTAL_DEPTH - line;
61      string reset = getBin(0, 16) + '\n';
62      while (rest--) {
63          cout << reset;
64      }
65      cout << flush;
66  #endif
67      return 0;
68  }
```

gcoe.cpp

```
1  #include <iostream>
2  #include <cstdlib>
3  using namespace std;
4
5  int main(int argc, char const *argv[]) {
6      int radix;
7      if (argc < 2)
8          radix = 2;
9      else
10         radix = atoi(argv[1]);
11     cout << "MEMORY_INITIALIZATION_RADIX=" << radix << ";\n"
12          << "MEMORY_INITIALIZATION_VECTOR="<< '\n';
13     string line;
14     getline(cin, line);
15     cout << line;
16     while (getline(cin, line))
17         cout << ",\n" << line;
18     cout << ';' << endl;
19     return 0;
20 }
```

mem.cpp

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  #define TOTAL_MEM 256
6
7  int mem[TOTAL_MEM]; // global data will be set to zero
8
9  string getBin(int n, int bits) {
10     string bin;
11     for (int i = bits - 1; i >= 0; i--) {
12         bin += (n & (1 << i)) ? '1': '0';
13     }
14     return bin;
15 }
16
17 int main() {
18     int addr, data;
19     while (cin >> hex >> addr >> data) {
20         mem[addr] = data;
21     }
```

```
22      for (int i = 0; i < TOTAL_MEM; i++) {
23          cout << getBin(mem[i], 16) << '\n';
24      }
25      cout << flush;
26      return 0;
27  }
```

def_d_mem.txt

```
1  00 000a
2  01 0004
3  02 0005
4  03 2369
5  04 69c3
6  05 0060
7  06 0fff
8  07 5555
9  08 6152
10 09 1057
11 0a 2895
```

compileall.bat

```
1 g++ assembler/assembler.cpp -o assembler -O3
2 g++ coe_generator/gcoe.cpp -o gcoe -O3
3 g++ mem/mem.cpp -o mem -O3
```

makeicoe.bat

```
1 assembler < code.asm > i_mem.mif
2 gcoe < i_mem.mif > i_mem.coe
```

makedcoe.bat

```
1 mem < def_d_mem.txt > d_mem.mif
2 gcoe < d_mem.mif > d_mem.coe
```

top Project Status (04/18/2017 - 20:52:00)			
Project File:	PCPU.xise	**Parser Errors:**	No Errors
Module Name:	top	**Implementation State:**	Programming File Generated
Target Device:	xc6slx16-3csg324	• **Errors:**	No Errors
Product Version:	ISE 14.7	• **Warnings:**	2 Warnings (2 new)
Design Goal:	Balanced	• **Routing Results:**	All Signals Completely Routed
Design Strategy:	Xilinx Default (unlocked)	• **Timing Constraints:**	All Constraints Met
Environment:	System Settings	• **Final Timing Score:**	0 (Timing Report)

Device Utilization Summary				[-]
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	379	18,224	2%	
Number used as Flip Flops	378			
Number used as Latches	0			
Number used as Latch-thrus	0			
Number used as AND/OR logics	1			
Number of Slice LUTs	800	9,112	8%	
Number used as logic	799	9,112	8%	
Number using O6 output only	677			
Number using O5 output only	16			
Number using O5 and O6	106			
Number used as ROM	0			
Number used as Memory	0	2,176	0%	
Number used exclusively as route-thrus	1			
Number with same-slice register load	0			
Number with same-slice carry load	1			
Number with other load	0			
Number of occupied Slices	270	2,278	11%	
Number of MUXCYs used	164	4,556	3%	
Number of LUT Flip Flop pairs used	847			
Number with an unused Flip Flop	477	847	56%	
Number with an unused LUT	47	847	5%	
Number of fully used LUT-FF pairs	323	847	38%	
Number of unique control sets	11			
Number of slice register sites lost to control set restrictions	46	18,224	1%	
Number of bonded IOBs	23	232	9%	

Number of LOCed IOBs	23	23	100%	
Number of RAMB16BWERs	0	32	0%	
Number of RAMB8BWERs	2	64	3%	
Number of BUFIO2/BUFIO2_2CLKs	0	32	0%	
Number of BUFIO2FB/BUFIO2FB_2CLKs	0	32	0%	
Number of BUFG/BUFGMUXs	2	16	12%	
Number used as BUFGs	2			
Number used as BUFGMUX	0			
Number of DCM/DCM_CLKGENs	0	4	0%	
Number of ILOGIC2/ISERDES2s	0	248	0%	
Number of IODELAY2/IODRP2/IODRP2_MCBs	0	248	0%	
Number of OLOGIC2/OSERDES2s	0	248	0%	
Number of BSCANs	0	4	0%	
Number of BUFHs	0	128	0%	
Number of BUFPLLs	0	8	0%	
Number of BUFPLL_MCBs	0	4	0%	
Number of DSP48A1s	0	32	0%	
Number of ICAPs	0	1	0%	
Number of MCBs	0	2	0%	
Number of PCILOGICSEs	0	2	0%	
Number of PLL_ADVs	0	2	0%	
Number of PMVs	0	1	0%	
Number of STARTUPs	0	1	0%	
Number of SUSPEND_SYNCs	0	1	0%	
Average Fanout of Non-Clock Nets	5.15			

Performance Summary			**[-]**
Final Timing Score:	0 (Setup: 0, Hold: 0)	**Pinout Data:**	Pinout Report
Routing Results:	All Signals Completely Routed	**Clock Data:**	Clock Report
Timing Constraints:	All Constraints Met		