《计算机组成原理》实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| **年级、专业、班级** |  | **姓名** | 周梓蕙 |
| **实验题目** | 实验二处理器译码实验 | | |
| **实验时间** | 2021 年 5 月 21 日 | **实验地点** | ds1410 |
| **实验成绩** | 优秀/良好/中等 | **实验性质** | □验证性  □G设计性  □综合性 |
| **教师评价：**  □算法/实验过程正确; □源程序/实验内容提交; □程序结构/实验步骤合理;  □实验结果正确; □语法、语义正确; □报告规范;  其他:  评价教师: | | | |
| **实验目的**   1. 掌握单周期 CPU 控制器的工作原理及其设计方法。 2. 掌握单周期 CPU 各个控制信号的作用和生成过程。 3. 掌握单周期 CPU 执行指令的过程。 4. 掌握取指、译码阶段数据通路执行过程。 | | | |

报告完成时间: 2021 年 5 月 29 日

# 实验内容

* 1. PC。D 触发器结构，用于储存 PC(一个周期)。需实现 2 个输入，分别为 *clk, rst*, 分别连接时钟和复位信号；需实现 2 个输出，分别为 *pc, inst\_ce*, 分别连接指令存储器的 *addra, ena* 端口。其中 *addra* 位数依据coe 文件中指令数定义；
  2. 加法器。用于计算下一条指令地址，需实现 2 个输入，1 个输出，输入值分别为当前指令地址 *PC*、*32*’*h4*；
  3. Controller。其中包含两部分：
     1. main\_decoder。负责判断指令类型，并生成相应的控制信号。需实现 1 个输入，为指令inst 的高 6 位 *op*，输出分为 2 部分，控制信号有多个，可作为多个输出，也作为一个多位输出，具体参照参考指导书进行设计；*aluop*，传输至 alu\_decoder，使 alu\_decoder 配合 *inst* 低 6 位 *funct*，进行 ALU 模块控制信号的译码。
     2. alu\_decoder。负责 ALU 模块控制信号的译码。需实现 2 个输入，1 个输出，输入分别为

*funct, aluop*；输出位 *alucontrol* 信号。

* + 1. 除上述两个组件，需设计 controller 文件调用两个decoder，对应实现 *op,funct* 输入信号，并传入调用模块；对应实现控制信号及 *alucontrol*，并连接至调用模块相应端口。
  1. 指令存储器。使用 Block Memory Generator IP 构造。(参考指导书)

注意：Basic 中 Generate address interface with 32 bits 选项不选中；PortA Options 中 Enable Port Type 选择为Use ENA Pin

* 1. 时钟分频器。将板载 100Mhz 频率降低为 1hz，连接 PC、指令存储器时钟信号 clk。(参考数字逻辑实验)

注意：Xilinx Clocking Wizard IP 可分的最低频率为 4.687Mhz，因而只能使用自实现分频模块进行分频

# 实验设计

## 控制器 (Controller)

### 功能描述

本次实验完成了控制器的取指令、译码部分的功能。

### 接口定义

* + 1. **逻辑控制**

top.v 是顶层文件，将 pc、ram、controller、display 几个模块连接。其输入为 clk 和 rst，输出指令信息。输入的 clk 传入pc、ram、display 模块，rst 传入pc、display 模块。

**表 1:** 接口定义

|  |  |  |  |
| --- | --- | --- | --- |
| **信号名** | **方向** | **位宽** | **功能描述** |
| clk | Input | 1-bit | 时钟信号 |
| rst | Input | 1-bit | 复位信号 |
| ans | Output | 8-bit | 确定哪一个 7 段数码管发光。 |
| seg | Output | 7-bit | 确定一个 7 段数码管的哪几根管发  光。 |
| douta | Output | 31-bit | 从指令存储器里取出的指令值。 |

pc 模块接受 clk、rst 两个输入，产生 pc、inst\_ce 两个输出，其中 pc 为当前指令的地址，inst\_ce为指令存储器的使能信号。时钟的上升沿到来时，若 rst=1，则 pc 的值初始化为 0，且输出的 inst\_ce置为 0，否则 pc=pc+4，inst\_ce=1。

pc 产生的输出送入指令存储器，作为读操作的地址和使能信号。此外，还接受 clk 信号，产生的输出为指令的值。

指令存储器的高 6 位为op，低 6 位为funct，均送入 controller。controller 包括main decoder 和 alu decoder。main decoder 接受op，根据 op 的值，利用 case 语句确定每个控制信号的值。其中，控制信号 aluop 输入 alu decoder，结合 funct 得到 alucontrol，作为 alu 的控制信号。控制器产生的输出送入display 显示。

## 存储器 (Block Memory)

### 类型选择

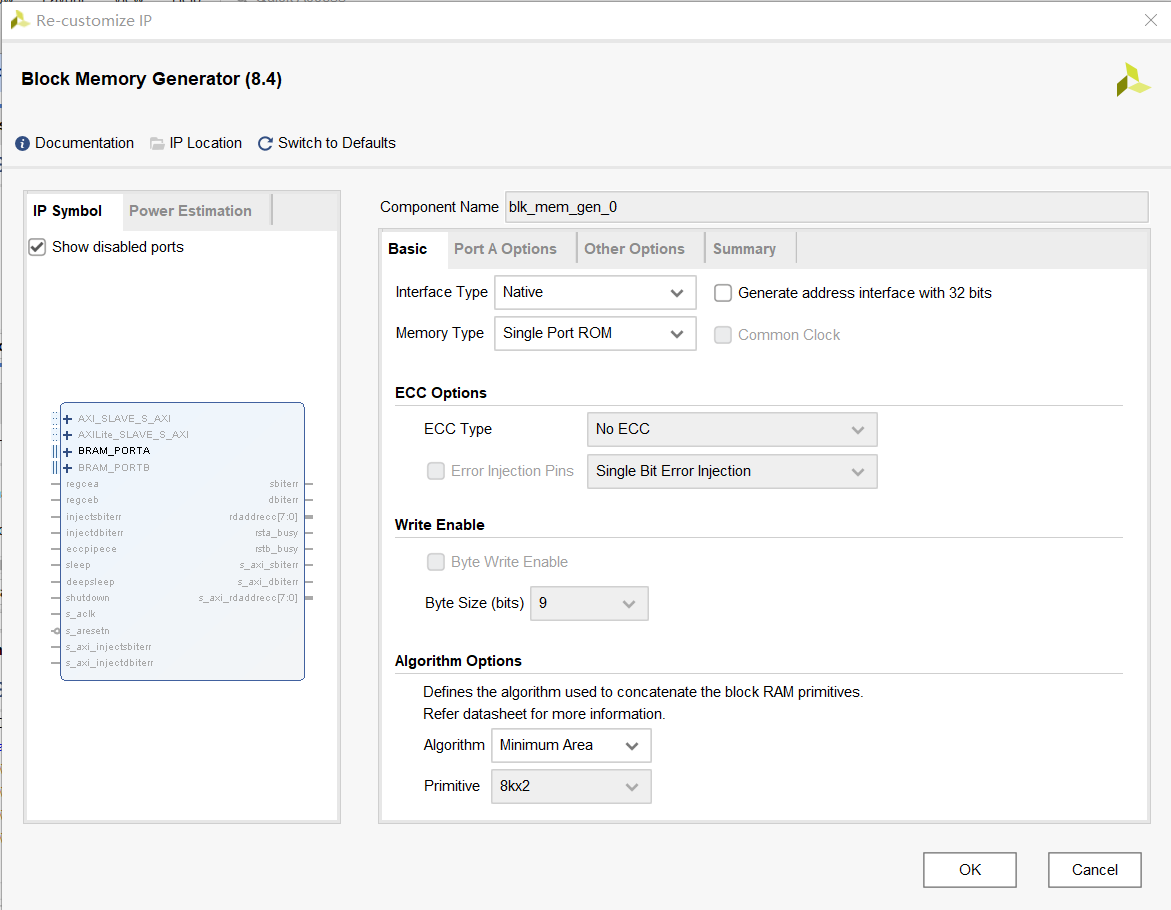
选择的存储器类型为Single Port ROM。

* + 1. **参数设置**

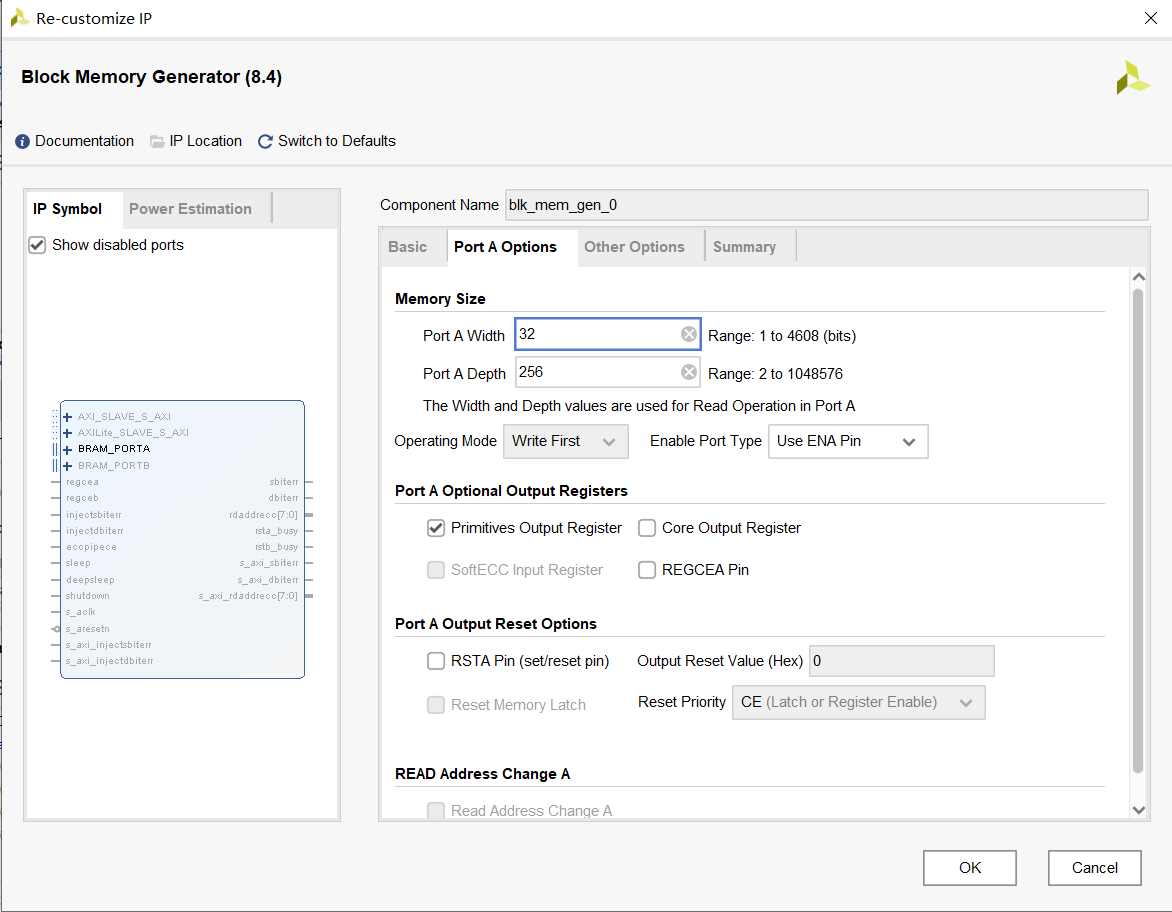
# 实验过程记录

### 完成的工作

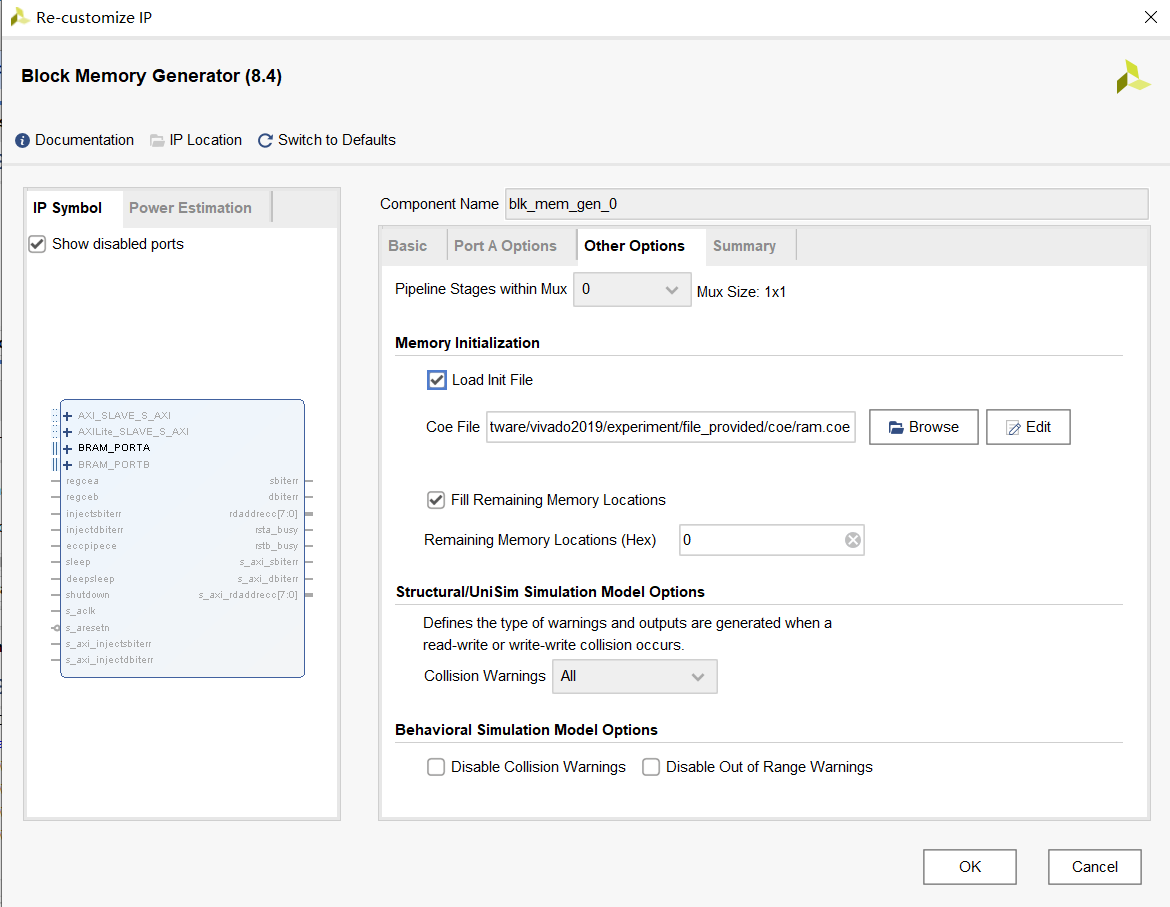
基本实现了控制器的取指令、译码的功能，使 pc 能够正确地自增，成果访问指令存储器，产生响应的控制信号。



**图 1**



**图 2**



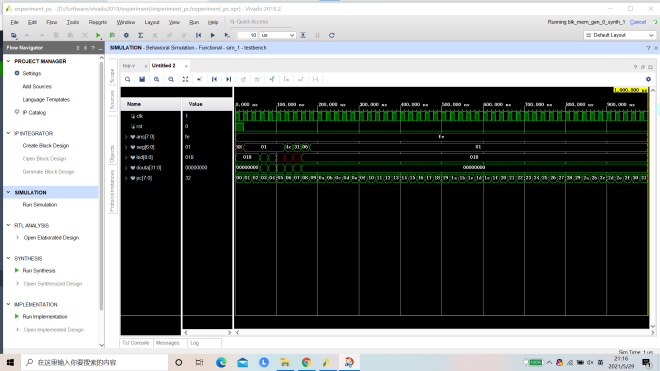
**图 3**

### 遇到的问题

实验过程中，不了解 block memory generator ip 的使用。解决方法：向已经完成实验的同学请教。

# 实验结果及分析

## 仿真图



**图 4**

## 控制台输出

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00a42820 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 010

instrument: 8c020050 memtoReg: 1 memWrite: 0 alusrc: 1 regdst: 0 regwrite: 1 branch: 0

alucontrol: 010

instrument: ac020054 memtoReg: x memWrite: 1 alusrc: 1 regdst: x regwrite: 0 branch: 0 alucontrol: 010

instrument: 10a7000c memtoReg: x memWrite: 0 alusrc: 0 regdst: x regwrite: 0 branch: 1 alucontrol: 011

instrument: 08000013 memtoReg: x memWrite: 0 alusrc: x regdst: x regwrite: 0 branch: x alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

instrument: 00000000 memtoReg: 0 memWrite: 0 alusrc: 0 regdst: 1 regwrite: 1 branch: 0

alucontrol: 000

# A Controller 代码

controller.v

module controller(

op , regwrite , regdst , alusrc , branch , memWrite , memtoReg , funct , alucontrol

);

input [5 : 0] op; input [5 : 0] funct; output regwrite ; output regdst; output alusrc ; output branch ; output mem Write ;

output memto Reg ;

output [2 : 0] alucontrol; wire [1 : 0] aluop ;

maindec md (. op( op), . regwrite ( regwrite ), . regdst( regdst), . alusrc ( alusrc ),

. branch ( branch ), . mem Write ( mem Write ), . memto Reg ( memto Reg ), . aluop ( aluop )

);

aludec ad (. funct( funct), . aluop ( aluop ), . alucontrol( alucontrol)); endmodule

maindec.v

module maindec (

op , regwrite , regdst , alusrc , branch , memWrite , memtoReg , aluop

);

input [5 : 0] op; output regwrite ; output regdst; output alusrc ; output branch ; output mem Write ; output memto Reg ; output [1 : 0] aluop ;

reg [7 : 0] code ;

assign regwrite = code [7]; assign regdst = code [6]; assign alusrc = code [5]; assign branch = code [4]; assign mem Write = code [3]; assign memto Reg = code [2]; assign aluop [1] = code [1]; assign aluop [0] = code [0];

always@ (\*) begin

case ( op)

6 ’ b 000000 : code = 8 ’ b 11000010 ;

6 ’ b 100011 : code = 8 ’ b 10100100 ;

6 ’ b 101011 : code = 8 ’ b 0 x 101 x 00 ;

6 ’ b 000100 : code = 8 ’ b 0 x 010 x 01 ;

6 ’ b 001000 : code = 8 ’ b 10100000 ;

6 ’ b 000010 : code = 8 ’ b 0 xxx 0 xxx ; default: code = 8 ’ b 00000000 ;

endcase

end

endmodule

aludec.v

module aludec (

funct , aluop , alucontrol

);

input [5 : 0] funct; input [1 : 0] aluop ;

output [2 : 0] alucontrol; reg [2 : 0] alucontrol;

always@ (\*) begin

if( aluop == 2 ’ b00 ) begin

alucontrol = 3 ’ b010 ; end // if( aluop == 2 ’ b00 ) else if( aluop == 2 ’ b01 ) begin

alucontrol = 3 ’ b110 ;

end // else if( aluop == 2 ’ b01 ) else if( aluop == 2 ’ b10 )

begin

case ( funct)

6 ’ b 100000 : alucontrol = 3 ’ b010 ; 6 ’ b 100010 : alucontrol = 3 ’ b110 ; 6 ’ b 100100 : alucontrol = 3 ’ b000 ; 6 ’ b 100101 : alucontrol = 3 ’ b001 ; 6 ’ b 101010 : alucontrol = 3 ’ b111 ; default: alucontrol = 3 ’ b000 ;

endcase

end // else if( aluop == 2 ’ b10 ) else

begin

alucontrol = 3 ’ b000 ; end // else

end // always endmodule