**实验八 单周期CPU中指令控制器的设计与仿真**

**一、实验目的**

1、深入理解单周期CPU中指令控制器的结构和工作原理。

2、掌握使用Verilog HDL设计和实现单周期CPU中的指令控制器。

**二、实验设备**

1、装有vivado的计算机 1台

2、EGO1开发板 1块

**三、实验任务**

1、设计和实现单周期CPU中指令控制器的结构并且进行功能仿真。

**四、实验原理**

1、一个单周期CPU的硬件结构如图8.1所示。其中红色框内部分是控制器。红色的线条是控制器发出的控制信号，控制指令通路上的多个元件工作，从而完成取指令，取操作数，控制运算器运算，运算结果写回等一系列操作。

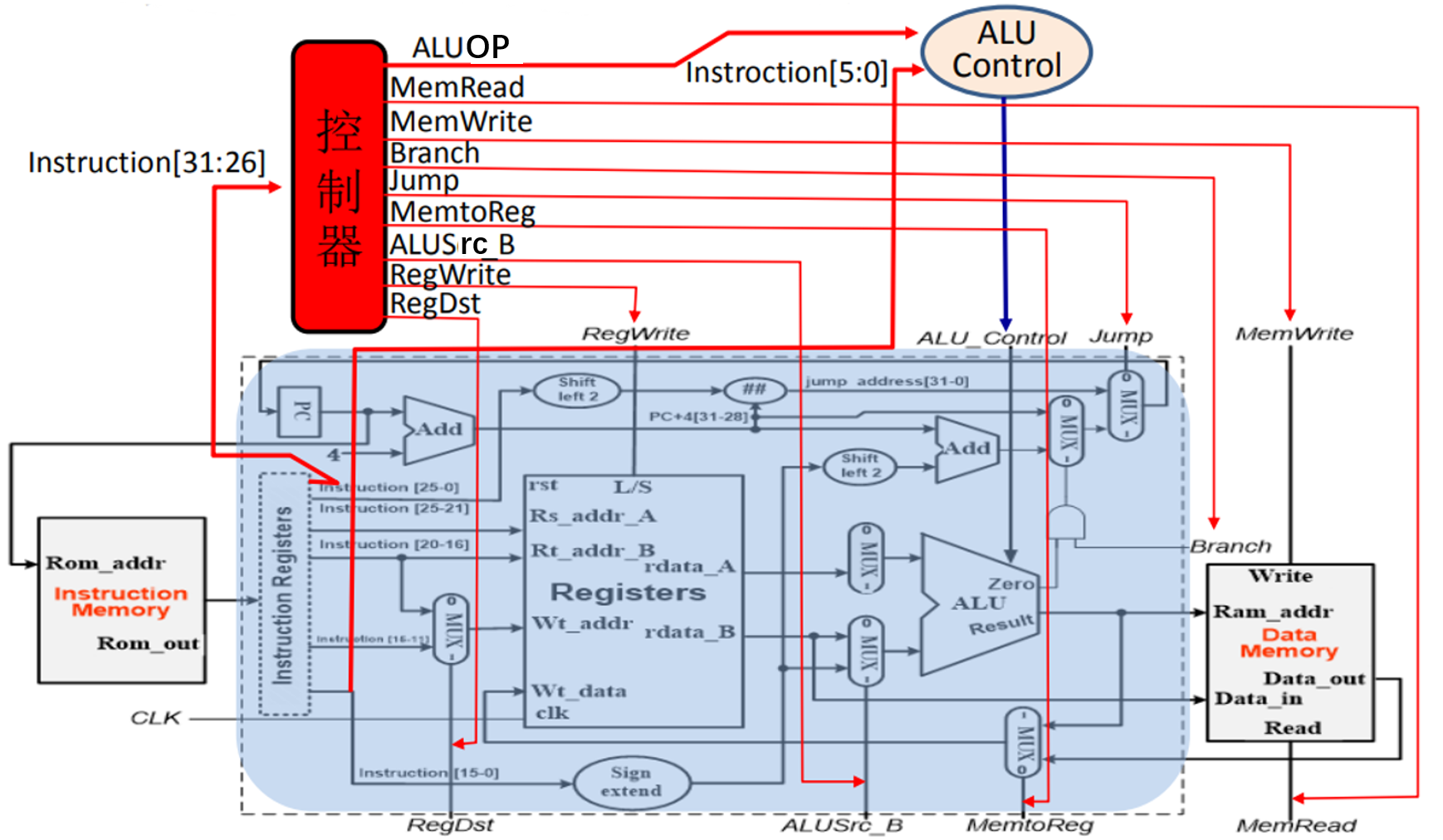


图8.1 单周期CPU的结构

在图8.1中，控制器模块接收从指令存储器读出的指令机器码，根据指令机器码的高6位instruction[31:26]进行译码后，生成控制信号RegDst、RegWrite、ALUSrc\_B、MemtoReg、Jump、Branch、MemWrite、MemRead、ALUOP。其中的控制信号ALUOP与指令机器码的低6位instruction[5:0]联合起来，生成运算器运算类型相对应的控制信号ALU\_Control。

信号MemRead：数据存储器读控制信号。

信号MemWrite：数据存储器写控制信号。

信号Branch：条件跳转控制信号，送给对应的二选一数据选择器。

信号Jump：直接跳转控制信号，送给对应的二选一数据选择器。

信号MemtoReg：将数据存储器读出的结果写入寄存器的控制信号。

信号ALUSrc\_B：操作数B的选择对应的控制信号。

信号RegWrite：对寄存器进行写操作的控制信号。

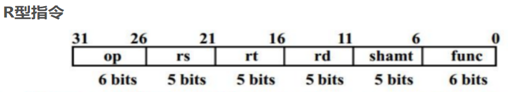
信号RegDst：从二选一选择器中选中对应的值，写入对应的寄存器。具体参看表8.1。

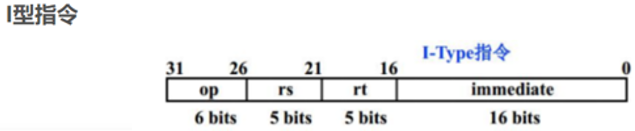
控制器输出信号赋值为0和赋值为1代表的含义如下表8.1所示：

表8.1 控制器输出信号功能定义表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **信号** | **源数目** | **功能定义** | **赋值0时动作** | **赋值1时动作** |
| ALUSrc\_B | 2 | ALU端口B输入选择 | 选择寄存器B数据 | 选择32位立即数(符号扩展后) |
| RegDst | 2 | 寄存器写地址选择 | 选择指令rt域 | 选择指令rs域 |
| MemtoReg | 2 | 寄存器写入数据选择 | 选择存储器数据 | 选择ALU输出 |
| Branch | 2 | Beq指令目标地址选择 | 选择PC+4地址 | 选择转移地址(Zero=1) |
| Jump | 2 | J指令目标地址选择 | 选择J目标地址 | 由Branch决定输出 |
| RegWrite | - | 寄存器写控制 | 禁止寄存器写 | 使能寄存器写 |
| MemWrite | - | 存储器写控制 | 禁止存储器写 | 使能存储器写 |
| MemRead | - | 存储器读控制 | 禁止存储器读 | 使能存储器读 |
| ALU\_Control | 000-111 | 3位ALU操作控制 |  |  |

在MIP指令集中，指令格式分为三种，分别为R类型、I类型、J类型指令。如下图8.2。





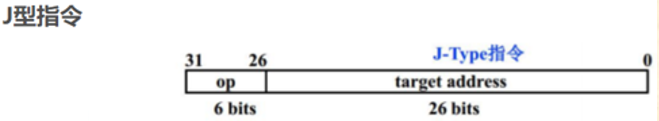


图8.2 MIPS指令集中三种指令格式

图8.2中，op部分字段代表操作码。func部分字段代表功能码。

表8.2列出了MIPS指令集中常见的20条指令的机器码的格式。每条机器指令是32位的二进制数，根据操作码的不同，分成了每段位数固定的若干段。

表8.2 MIPS指令集中20条指令的机器码

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 指令 | [31:26] | [25:21] | [20:16] | [15:11] | [10: 6] | [5:0] | 功能 |
| R型指令 | | | | | | | |
| Add | 000000 | rs | rt | rd | 000000 | 100000 | 寄存器加 |
| Sub | 000000 | rs | rt | rd | 000000 | 100010 | 寄存器减 |
| And | 000000 | rs | rt | rd | 000000 | 100100 | 寄存器与 |
| Or | 000000 | rs | rt | rd | 000000 | 100101 | 寄存器或 |
| Xor | 000000 | rs | rt | rd | 000000 | 100110 | 寄存器异或 |
| Sll | 000000 | 00000 | rt | rd | sa | 000000 | 左移 |
| Srl | 000000 | 00000 | rt | rd | sa | 000010 | 逻辑右移 |
| Sra | 000000 | 00000 | rt | rd | sa | 000011 | 算术右移 |
| Jr | 000000 | rs | rt | rd | 000000 | 001000 | 寄存器跳 |
| I型指令 | | | | | | | |
| Addi | 001000 | rs | rt | immediate | | | 立即数加 |
| Andi | 001100 | rs | rt | immediate | | | 立即数与 |
| Ori | 001101 | rs | rt | immediate | | | 立即数或 |
| Xori | 001110 | rs | rt | immediate | | | 立即数异或 |
| Lw | 100011 | rs | rt | offset | | | 取数据 |
| Sw | 101011 | rs | rt | offset | | | 存数据 |
| Beq | 000100 | rs | rt | offset | | | 相等转移 |
| Bne | 000101 | rs | rt | offset | | | 不等转移 |
| Lui | 001111 | 00000 | rt | immediate | | | 设置高位 |
| J型指令 | | | | | | | |
| J | 000010 | address | | | | | 跳转 |
| Jal | 000011 | address | | | | | 调用 |

控制器输入的操作码与输出信号对应关系，如下表8.3所示。请可以提前填写完成表8.3中有确定值的空白方格。有些方格的值是不确定的，填写X即可。

表8.3 控制器输入的操作码与输出信号对应表

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| OP | 对应的指令 | Reg Dst | ALU Src | Mem toReg | Reg Write | Mem Read | Mem Write | Branch | Jump | ALU op1 | ALU op0 |
| R-格式 000000 | ADD |  |  |  |  |  |  |  |  |  |  |
| R-格式 000000 | SUB |  |  |  |  |  |  |  |  |  |  |
| R-格式 000000 | AND |  |  |  |  |  |  |  |  |  |  |
| R-格式 000000 | OR |  |  |  |  |  |  |  |  |  |  |
| R-格式 000000 | SLL |  |  |  |  |  |  |  |  |  |  |
| I-格式LW | LOAD |  |  |  |  |  |  |  |  |  |  |
| I-格式SW | STORE |  |  |  |  |  |  |  |  |  |  |
| I-格式beq | BEQ |  |  |  |  |  |  |  |  |  |  |
| J-格式指令 | Jump |  |  |  |  |  |  |  |  |  |  |

控制器接口信号如下图8.3所示。由于是单周期CPU的控制器，所以与多周期CPU、流水线CPU相比，结构相对简单一些。图8.3中，左边引脚为输入，右边的引脚为输出。

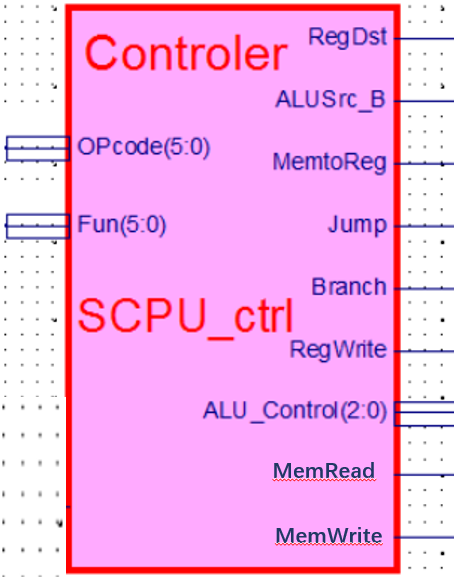


图8.3控制器的接口引脚示意图

CPU部件之控制器SCPU\_ctrl接口代码如下：

module SCPU\_ctrl(

input[5:0] OPcode, //OPcode

input[5:0] func, //function

output reg RegDst,

output reg ALUSrc\_B,

output reg MemtoReg,

output reg Jump,

output reg Branch,

output reg RegWrite,

output reg[2:0] ALU\_Control,

output reg MemRead,

output reg MemWrite );

……

endmodule

根据图81.中控制器需要控制的单元，根据表8.3，可以画出控制器对应的控制电路结构图类似下图8.4。具体推导方法就是根据CPU里面具体单元的数据流控制操作进行推导的。

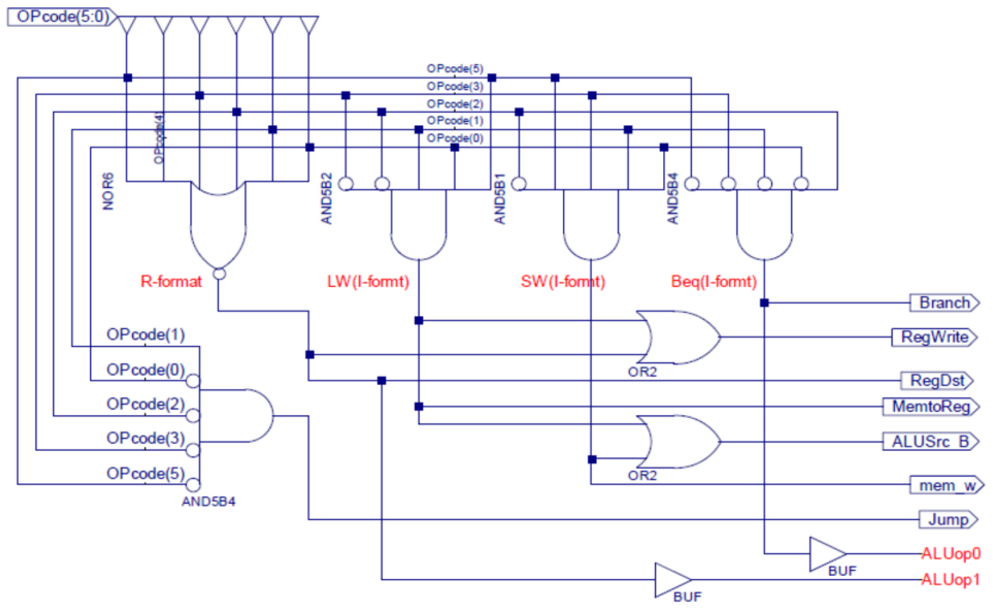


图8.4控制器控制电路结构示意图

图8.1中的控制信号ALUOP与指令机器码的低6位instruction[5:0]联合起来，生成运算器运算类型相对应的控制信号ALU\_Control。控制器输入的操作码与运算器控制信号ALU\_Control对应关系如表8.4所示。

表8.4 控制器输入的操作码与运算器控制信号ALU\_Control对应表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Instruction opcode | ALUOP | Instruction operation | func field | Desired ALU action | ALU\_Control |
| LW | 00 | Load word | xxxxxx | Load word | 010 |
| SW | 00 | Store word | xxxxxx | Store word | 010 |
| Beq | 01 | branch equal | xxxxxx | branch equal | 110 |
| R-type | 10 | add | 10 0000 | add | 010 |
| R-type | 10 | subtract | 10 0010 | subtract | 110 |
| R-type | 10 | AND | 10 0100 | AND | 000 |
| R-type | 10 | OR | 10 0101 | OR | 001 |
| R-type | 10 | Set on less than | 10 1010 | Set on less than | 111 |
| R-type | 10 | NOR | 10 0111 | NOR | 100 |

控制器输入的操作码与运算器控制信号ALU\_Control对应逻辑电路的结构如图8.5所示。在图8.4的右边有两条输出信号线ALUop1和ALUop2，这两个信号连同指令机器码中的后6位（这6位称为func[5:0]，参加图图8.2中的R型指令格式图），作为输入，生成控制运算器的控制码：ALU\_Control[2:0]，由ALU\_Control[2:0]控制运算器进行与、或、加、减、或非、访存（读或者写）、相等则跳转、小于则置1这8种操作中的某一种操作。

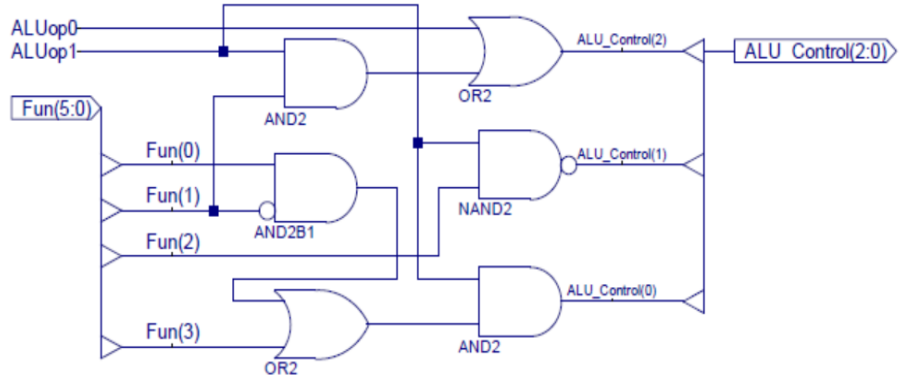


图8.5 控制器输入的操作码与运算器控制信号ALU\_Control对应逻辑电路的结构

**五、实验步骤**

**（一）设计单周期CPU的指令控制器（实验室完成和进行仿真验收）**

1、**新建Vivado 工程 参照实验五的步骤创建新工程。**

参考上面的图8.1、图8.4和图8.5，设计单周期CPU的**指令控制器电路**，接口定义类似如下:

module SCPU\_ctrl(

input[5:0] OPcode, //OPcode

input[5:0] func, //function

output reg RegDst,

output reg ALUSrc\_B,

output reg MemtoReg,

output reg Jump,

output reg Branch,

output reg RegWrite,

output reg[2:0] ALU\_Control,

output reg MemRead,

output reg MemWrite );

……请在此处补充对应的verilog代码，要求在实验前完成。进实验室时会检查。验收时会围绕核心代码进行提问。

endmodule

请补充上面控制器模块中没有完成的verilog代码。

**2、在Vivado中进行仿真测试单周期CPU的控制器模块。**

建立了顶层模块后，如果需要对顶层模块进行测试的话，就需要给顶层模块添加相应的输入，观察顶层模块的输出所具有的特点。例如：将加、减、与、或、跳转等多条指令的机器码，每隔100ns依次送入。观察控制器输出信号对应的波形是否正确。

完成下面控制器激励输入模块：

module testcontrol(…… )

请在此处补充对应的verilog代码，要求在实验前完成。进实验室时会检查。验收时会围绕核心代码进行提问。

endmodule

进行仿真，验证仿真后的波形图是否正确。

在下面粘贴你综合后生成的电路的原理图。

在下面粘贴你的指令执行对应的仿真波形截图，并分析指令执行结果的正确性。

**（三）**、结合EGO1开发板，在Vivado中生成bit文件和下载。将多条指令的机器码存放到instruction.txt文件中，通过readmem（）函数可以加载instruction.txt文件中指令的机器码，在开关的控制下依次将指令送入单周期CPU的控制电路中，将控制电路的输出送到EGO1开发板上LED0、LED1、LED2、LED3…和LED15这些发光二极管上去。要求开关sw0、sw1……sw7中某个开关值为1就执行对应的指令。例如sw0、sw1……sw7中8个开关值为10000000时，执行第一条指令。sw0、sw1……sw7中8个开关值为01000000时，执行第二条指令。sw0、sw1……sw7中8个开关值为00100000时，执行第三条指令……依此类推，观察16个发光二极管的亮灭情况，记入下表8.5。可以用16个开关对应于16条指令的执行。也可以自己定义EGO1开发板上开关与指令之间的对应关系。

表8.5指令执行时控制器输出值

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令 | 对应的指令 | Reg Dst | ALU Src | Mem toReg | Reg Write | Mem Read | Mem Write | Branch | Jump | ALU op1 | ALU op0 |
| 第1条 | ADD |  |  |  |  |  |  |  |  |  |  |
| 第2条 | SUB |  |  |  |  |  |  |  |  |  |  |
| 第3条 | OR |  |  |  |  |  |  |  |  |  |  |
| 第4条 | AND |  |  |  |  |  |  |  |  |  |  |
| 第5条 | NOT |  |  |  |  |  |  |  |  |  |  |
| 第6条 | LOAD |  |  |  |  |  |  |  |  |  |  |
| 第7条 | STORE |  |  |  |  |  |  |  |  |  |  |
| 第8条 | Branch |  |  |  |  |  |  |  |  |  |  |
| 第9条 | Jump |  |  |  |  |  |  |  |  |  |  |

对于instruction.txt文件中存放的指令的机器码，可以参考和采用下面汇编代码对应的机器码。也可以自己设置对应的机器码。

注意下面的汇编指令与后面注释里机器码的对应关系，从而方便相互转换。下面竖线后面显示的是每条指令对应的执行结果。$1：代表编号为1的寄存器。#19：代表立即数19。

addiu $1 ,$0,#1 //32'h24010001; // 00H: addiu $1 ,$0,#1 | $1 = 0000\_0001H

sll $2 ,$1,#4 // 32'h00011100; // 04H: sll $2 ,$1,#4 | $2 = 0000\_0010H

addu $3 ,$2,$1 // 32'h00411821; // 08H: addu $3 ,$2,$1 | $3 = 0000\_0011H

srl $4 ,$2,#2 // 32'h00022082; // 0CH: srl $4 ,$2,#2 | $4 = 0000\_0004H

subu $5 ,$3,$4 // 32'h00642823; // 10H: subu $5 ,$3,$4 | $5 = 0000\_000DH

sw $5 ,#19($1) //32'hAC250013; // 14H: sw $5 ,#19($1) | Mem[0000\_0014H] = 0000\_000DH

nor $6 ,$5,$2 // 32'h00A23027; // 18H: nor $6 ,$5,$2 | $6 = FFFF\_FFE2H

or $7 ,$6,$3 //32'h00C33825; // 1CH: or $7 ,$6,$3 | $7 = FFFF\_FFF3H

xor $8 ,$7,$6 // 32'h00E64026; // 20H: xor $8 ,$7,$6 | $8 = 0000\_0011H

sw $8 ,#28($0) //32'hAC08001C; // 24H: sw $8 ,#28($0) | Mem[0000\_001CH] = 0000\_0011H

slt $9 ,$6,$7 //32'h00C7482A; // 28H: slt $9 ,$6,$7 | $9 = 0000\_0001H

beq $9 ,$1,#2 //32'h11210002; // 2CH: beq $9 ,$1,#2 | 跳转到指令34H

addiu $1 ,$0,#4 //32'h24010004; // 30H: addiu $1 ,$0,#4 | 不执行

lw $10,#19($1) //32'h8C2A0013; // 34H: lw $10,#19($1) | $10 = 0000\_000DH

bne $10,$5,#3 //32'h15450003; // 38H: bne $10,$5,#3 | 不跳转

and $11,$2,$1 //32'h00415824; // 3CH: and $11,$2,$1 | $11 = 0000\_0000H

sw $11,#28($0) //32'hAC0B001C;//40H: sw $11,#28($0) | Men[0000\_001CH] = 0000\_0000H

sw $4 ,#16($0) //32'hAC040010; // 44H: sw $4 ,#16($0) | Mem[0000\_0010H] = 0000\_0004H

lui $12,#12 // 32'h3C0C000C; // 48H: lui $12,#12 | [R12] = 000C\_0000H

j 00H //32'h08000000; // 4CH: j 00H | 跳转指令00H

**（四）**、结合图8.1单周期CPU的结构，补充完整整个CPU的硬件结构，可以描述PC模块、执行模块、数据存储器模块、指令存储器模块、寄存器组模块、时钟模块以及顶层模块top。然后对整个单周期CPU硬件结构进行激励输入和仿真。（不验收）

**六、实验验收要求和思考**

1、可以使用实验室的电脑进行验收。也可以自己带电脑到实验室，在自己的电脑上完成实验，然后在自己的电脑上进行验收。验收时，要求能简单介绍一下系统能实现的主要功能，核心代码的含义和实验中引脚的分配等等。老师会简单提问，如果回答问题较差，会酌情扣分。所以要求对代码进行提前预习和准备。

2、实验八的验收规则为：只验收到实验步骤2，验收成绩80分。验收到步骤3，验收成绩100分。如果当次实验课没有验收完，可以在其他班的实验课时进行补验收。

3、下载情况下，输出信号还可以送到EGO1开发板上的哪里进行显示？如果输出信号送到七段显示器上进行显示，verilog代码应该怎么写？

4、验收时需要指明顶层模块、子模块的名字，还要在顶层模块中找到调用子模块的位置和调用语句（子模块例化就相当于子模块的调用）。

5、如果验收到开发板下载，需要先指明开发板上的开关与输入信号之间的对应关系，还需要指明输出送到了发光二极管上，还是七段数码管上，然后拨动开关，观察输出绑定的发光二极管或者七段数码管的显示变化。

6、如果AX、BX、CX、DX这4个寄存器对应的寄存器编号为00H、01H、02H、03H，对于汇编语言指令MOV AX,#12；MOV BX，#61；ADD AX,BX；编译后的机器指令怎么写？这3条汇编指令在硬件上怎么完成对应的操作？

七、附录参考（MIPS指令集）

