**单周期MIPS处理器设计实验报告**

**姓 名： 钱鹏 学 号： 18020100032 指导教师： 邢建国**

# **一、实验目的**

（1）掌握单周期MIPS处理器的数据流图的构成、原理及其设计方法；

（2）掌握单周期MIPS处理器的实现方法，代码实现方法；

（3）认识和掌握指令与处理器的关系；

（4）掌握测试单周期MIPS处理器的方法；

# **二、实验内容**

设计一个单周期的MIPSCPU，使其能实现下列指令：

**==> 算术运算指令**

（1）add rd, rs, rt （**说明：以助记符表示，是汇编指令；以代码表示，是机器指令）**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 000000 | rs(5位) | rt(5位) | rd(5位) | reserved |

功能：rd←rs + rt。**reserved为预留部分，即未用，一般填“0”。**

（2）addi rt, rs, **immediate**

|  |  |  |  |
| --- | --- | --- | --- |
| 000001 | rs(5位) | rt(5位) | **immediate**(16位) |

功能：rt←rs + (sign-extend)**immediate；immediate**符号扩展再参加“加”运算。

（3）sub rd, rs, rt

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 000010 | rs(5位) | rt(5位) | rd(5位) | reserved |

功能：rd←rs - rt

**==> 逻辑运算指令**

（4）ori rt, rs, **immediate**

|  |  |  |  |
| --- | --- | --- | --- |
| 010000 | rs(5位) | rt(5位) | **immediate**(16位) |

功能：rt←rs **|** (zero-extend)**immediate；immediate**做“0”扩展再参加“或”运算。

（5）and rd, rs, rt

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 010001 | rs(5位) | rt(5位) | rd(5位) | reserved |

功能：rd←rs & rt；逻辑与运算。

（6）or rd, rs, rt

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 010010 | rs(5位) | rt(5位) | rd(5位) | reserved |

功能：rd←rs **|** rt；逻辑或运算。

**==>移位指令**

（7）sll rd, rt, sa

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 011000 | 未用 | rt(5位) | rd(5位) | sa | reserved |

功能：rd<－rt<<(zero-extend)sa，左移sa位 ，(zero-extend)sa

**==>比较指令**

（8） slt rd, rs, rt 带符号数

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 011100 | rs(5位) | rt(5位) | rd(5位) | reserved |

功能：if (rs<rt) rd =1 else rd=0, 具体请看表2 ALU运算功能表，带符号

**==> 存储器读/写指令**

（9）sw rt, **immediate(**rs) 写存储器

|  |  |  |  |
| --- | --- | --- | --- |
| 100110 | rs(5位) | rt(5位) | **immediate**(16位) |

功能：memory[rs+ (sign-extend)**immediate**]←rt；**immediate**符号扩展再相加。即将rt寄存器的内容保存到rs寄存器内容和立即数符号扩展后的数相加作为地址的内存单元中。

（10) lw rt , **immediate**(rs) 读存储器

|  |  |  |  |
| --- | --- | --- | --- |
| 100111 | rs(5位) | rt(5位) | **immediate**(16位) |

功能：rt ← memory[rs + (sign-extend)**immediate**]；**immediate**符号扩展再相加。

即读取rs寄存器内容和立即数符号扩展后的数相加作为地址的内存单元中的数，然后保存到rt寄存器中。

**==> 分支指令**

（11）beq rs, rt, **Immediate**

|  |  |  |  |
| --- | --- | --- | --- |
| 110000 | rs(5位) | rt(5位) | **immediate**(16位) |

功能：if(rs=rt) pc←pc + 4 + (sign-extend)**immediate** <<2 else pc ←pc + 4

特别说明：**immediate是从PC+4地址开始和转移到的指令之间指令条数。immediate**符号扩展之后左移2位再相加。为什么要左移2位？由于跳转到的指令地址肯定是4的倍数（每条指令占4个字节），最低两位是“00”，因此将**immediate**放进指令码中的时候，是右移了2位的，也就是以上说的“指令之间指令条数”。

(12）bne rs, rt, **Immediate**

|  |  |  |  |
| --- | --- | --- | --- |
| 110001 | rs(5位) | rt(5位) | **immediate** |

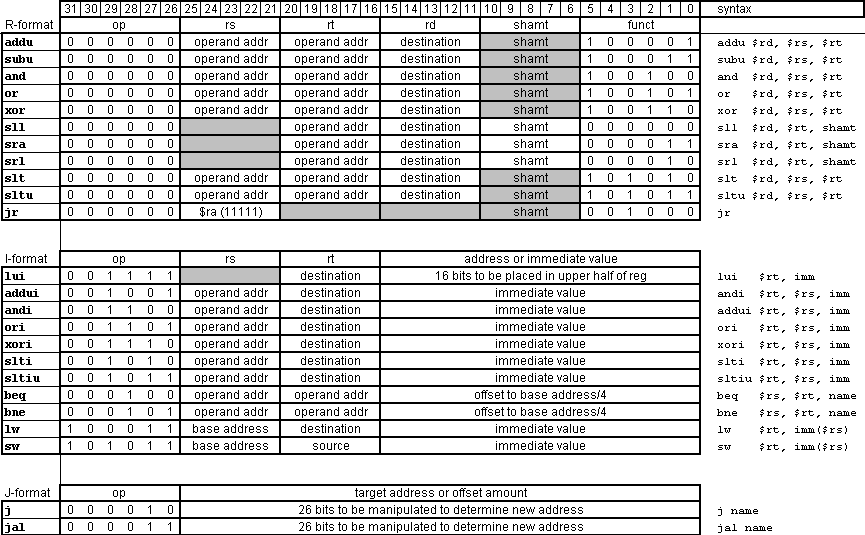
功能：if(rs!=rt) pc←pc + 4 + (sign-extend)immediate <<2 else pc ←pc + 4

特别说明：与beq不同点是，不等时转移，相等时顺序执行。

**==>跳转指令**

（14）j addr

|  |  |
| --- | --- |
| 111000 | addr[27..2] |

****

**图1 实验指导图**

# **三、实验原理**

单周期CPU指的是一条指令的执行在一个时钟周期内完成，然后开始下一条指令的执行，即一条指令用一个时钟周期完成。电平从低到高变化的瞬间称为时钟上升沿，两个相邻时钟上升沿之间的时间间隔称为一个时钟周期，时钟周期一般也称振荡周期。

CPU在处理指令时，一般需要经过以下几个步骤：

(1) 取指令(**IF**)：根据程序计数器PC中的指令地址，从存储器中取出一条指令，同时，PC根据指令字长度自动递增产生下一条指令所需要的指令地址，但遇到“地址转移”指令时，则控制器把“转移地址”送入PC，当然得到的“地址”需要做些变换才送入PC。

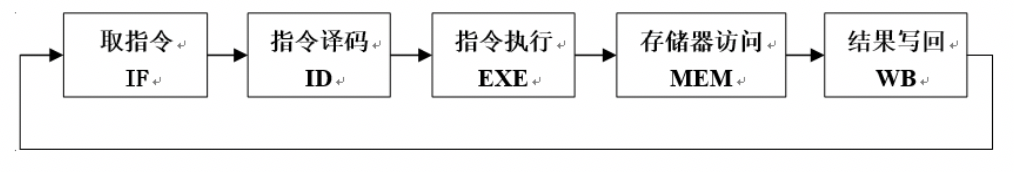
(2) 指令译码(**ID**)：对取指令操作中得到的指令进行分析并译码，确定这条指令需要完成的操作，从而产生相应的操作控制信号，用于驱动执行状态中的各种操作。

(3) 指令执行(**EXE**)：根据指令译码得到的操作控制信号，具体地执行指令动作，然后转移到结果写回状态。

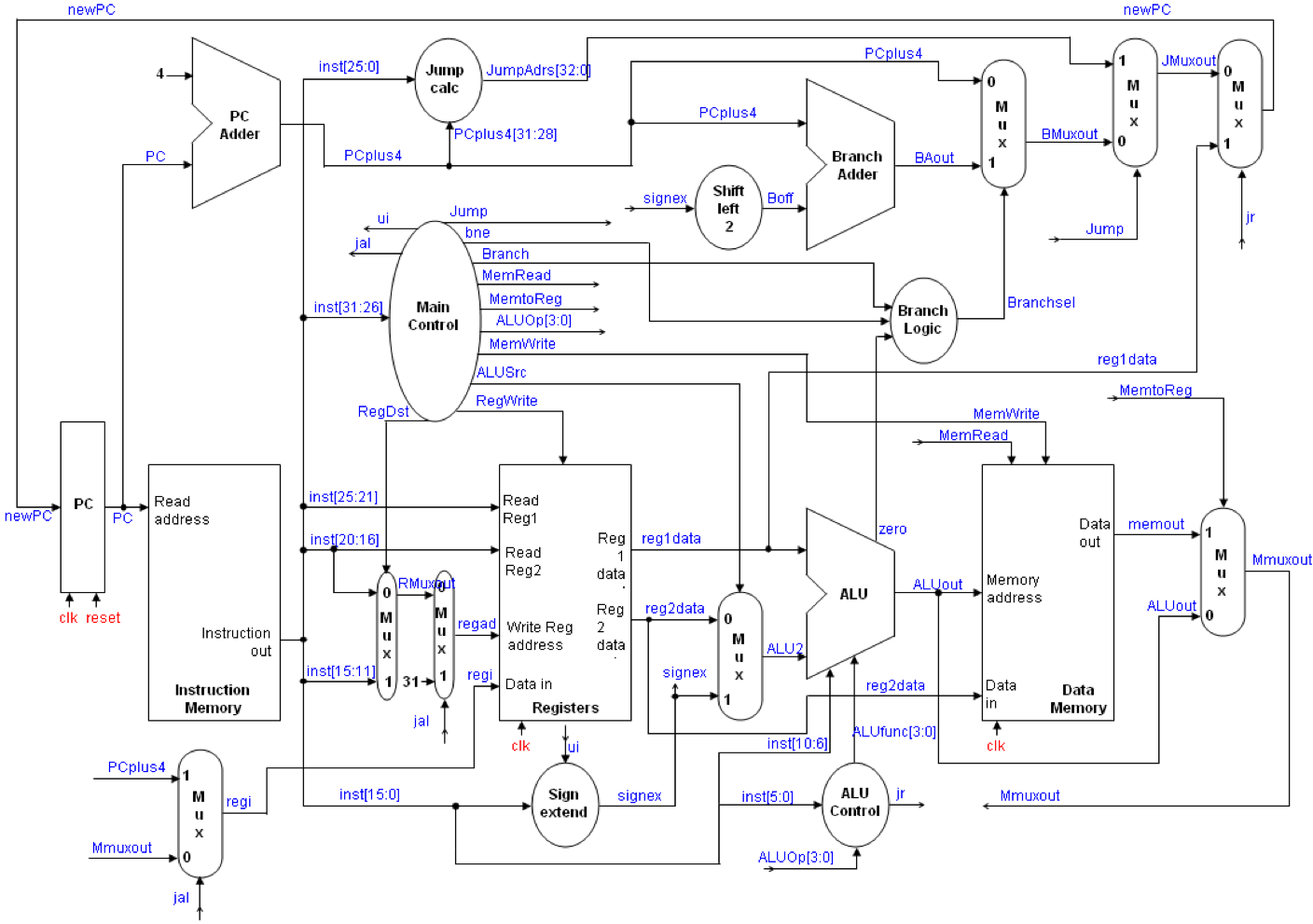
(4) 存储器访问(**MEM**)：所有需要访问存储器的操作都将在这个步骤中执行，该步骤给出存储器的数据地址，把数据写入到存储器中数据地址所指定的存储单元或者从存储器中得到数据地址单元中的数据。

(5) 结果写回(**WB**)：指令执行的结果或者访问存储器中得到的数据写回相应的目的寄存器中。

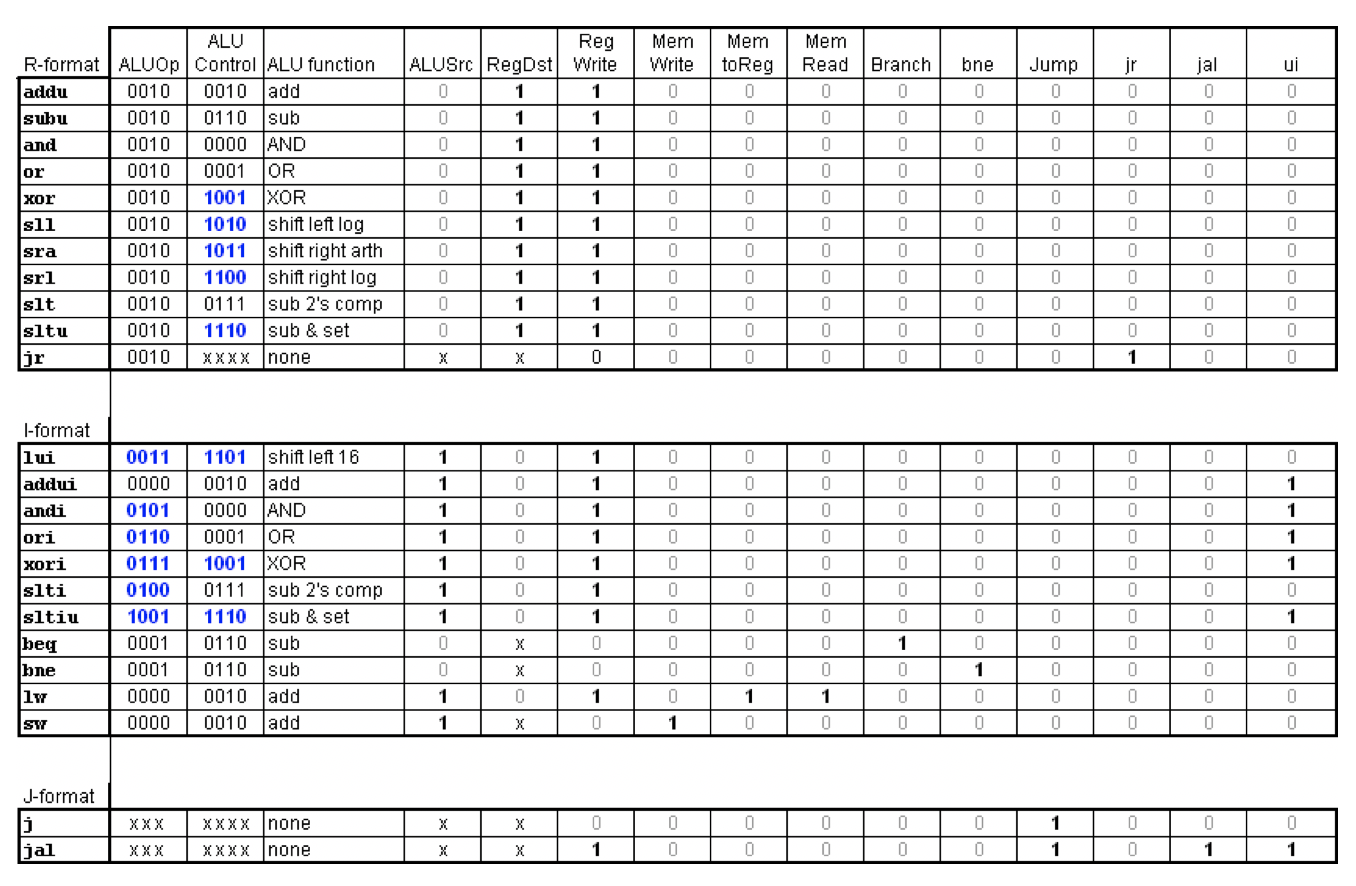
单周期CPU，是在一个时钟周期内完成这五个阶段的处理。



**图2 单周期CPU指令处理过程图**

对于不同的指令，需要执行的步骤是不同的，其中取字指令（lw）需要执行全部五个步骤。因此，CPU的时间周期由取字指令决定。

**图3 控制路线和熟路通路**

上图为CPU的数据通路和必要的控制线路图，其中Ins.Mem为指令存储器，Data.Mem为数据存储器。访问存储器时，先给出内存地址，然后由读或写信号控制操作。对于寄存器组，先给出寄存器地址，读操作时，输出端就直接输出相应数据；而在写操作时，在 WE使能信号为1，在时钟边沿触发将数据写入寄存器。

**图4 控制信号**

主要模块接口说明：

Instruction Memory：指令存储器，

address，指令存储器地址输入端口

DataIn，指令存储器数据输入端口（指令代码输入端口）

DataOut，指令存储器数据输出端口（指令代码输出端口）

InsMemRW，指令存储器读写控制信号，为0写，为1读

Data Memory：数据存储器，

address，数据存储器地址输入端口

DataOut，数据存储器数据输出端口

/RD，数据存储器读控制信号，为0读

/WR，数据存储器写控制信号，为0写

Register File：寄存器组

Read Reg1，rs寄存器地址输入端口

Read Reg2，rt寄存器地址输入端口

Write Reg，将数据写入的寄存器端口，其地址来源rt或rd字段

Write Data，写入寄存器的数据输入端口

Read Data1，rs寄存器数据输出端口

Read Data2，rt寄存器数据输出端口

WE，写使能信号，为1时，在时钟边沿触发写入

RST，寄存器清零信号，为0时寄存器清零

ALU： 算术逻辑单元

result，ALU运算结果

zero，运算结果标志，结果为0，则zero=1；否则zero=0

sign，运算结果标志，结果最高位为0，则sign=0，正数；否则，sign=1，负数

# **四、实验过程与结果**

**主要代码与仿真：**

**（1）控制单元**

**module main\_control (**

**input[5:0] inst,**

**output reg[3:0] ALUOp,**

**output reg ALUSrc,RegDst,RegWrite,MemWrite,MemtoReg,**

**MemRead,Branch,bne,Jump,jal,ui);**

**always @ (inst)**

**begin**

**case (inst)**

**6'b000000:**

**begin**

**ALUOp=4'b0010; ALUSrc=0; RegDst=1; RegWrite=1; MemWrite=0;**

**MemtoReg=0; MemRead=0; Branch=0; bne=0; Jump=0; jal=0; ui=0;**

**end**

**6'b001111:**

**begin**

**ALUOp=4'b0011; ALUSrc=1;RegDst=0;RegWrite=1;MemWrite=0;**

**MemtoReg=0; MemRead=0;Branch=0;bne=0;Jump=0;jal=0;ui=0;**

**end**

**6'b001001:**

**begin**

**ALUOp=4'b0000; ALUSrc=1;RegDst=0;RegWrite=1;MemWrite=0;**

**MemtoReg=0; MemRead=0;Branch=0;bne=0;Jump=0;jal=0;ui=1;**

**end**

**6'b001100:**

**begin**

**ALUOp=4'b0101; ALUSrc=1;RegDst=0;RegWrite=1;MemWrite=0;**

**MemtoReg=0; MemRead=0;Branch=0;bne=0;Jump=0;jal=0;ui=1;**

**end**

**6'b001100:**

**begin**

**ALUOp=4'b0101; ALUSrc=1;RegDst=0;RegWrite=1;MemWrite=0;**

**MemtoReg=0; MemRead=0;Branch=0;bne=0;Jump=0;jal=0;ui=1;**

**end**

**6'b001101:**

**begin**

**ALUOp=4'b0101; ALUSrc=1;RegDst=0;RegWrite=1;MemWrite=0;**

**MemtoReg=0; MemRead=0;Branch=0;bne=0;Jump=0;jal=0;ui=1;**

**end**

**6'b001110:**

**begin**

**ALUOp=4'b0101; ALUSrc=1;RegDst=0;RegWrite=1;MemWrite=0;**

**MemtoReg=0; MemRead=0;Branch=0;bne=0;Jump=0;jal=0;ui=1;**

**end**

**6'b001010:**

**begin**

**ALUOp=4'b0101; ALUSrc=1;RegDst=0;RegWrite=1;MemWrite=0;**

**MemtoReg=0; MemRead=0;Branch=0;bne=0;Jump=0;jal=0;ui=0;**

**end**

**6'b001011:**

**begin**

**ALUOp=4'b0101; ALUSrc=1;RegDst=0;RegWrite=1;MemWrite=0;**

**MemtoReg=0; MemRead=0;Branch=0;bne=0;Jump=0;jal=0;ui=1;**

**end**

**6'b000100:**

**begin**

**ALUOp=4'b0101; ALUSrc=0;RegDst=0;RegWrite=0;MemWrite=0;**

**MemtoReg=0; MemRead=0;Branch=1;bne=0;Jump=0;jal=0;ui=0;**

**end**

**6'b000101:**

**begin**

**ALUOp=4'b0101; ALUSrc=0;RegDst=0;RegWrite=0;MemWrite=0;**

**MemtoReg=0; MemRead=0;Branch=0;bne=1;Jump=0;jal=0;ui=0;**

**end**

**6'b000011:**

**begin**

**ALUOp=4'b0101; ALUSrc=1;RegDst=0;RegWrite=1;MemWrite=0;**

**MemtoReg=0; MemRead=1;Branch=0;bne=0;Jump=0;jal=0;ui=0;**

**end**

**6'b101011:**

**begin**

**ALUOp=4'b0000;ALUSrc=1;RegDst=0;RegWrite=0;MemWrite=1;**

**MemtoReg=0;MemRead=0;Branch=0;bne=0;Jump=0;jal=0;ui=0;**

**end**

**6'b000010:**

**begin**

**ALUOp=4'b0000;ALUSrc=0;RegDst=0;RegWrite=0;MemWrite=0;**

**MemtoReg=0;MemRead=0;Branch=0;bne=0;Jump=0;jal=0;ui=0;**

**end**

**6'b000011:**

**begin**

**ALUOp=4'b0000;ALUSrc=0;RegDst=0;RegWrite=1;MemWrite=0;**

**MemtoReg=0;MemRead=0;Branch=0;bne=0;Jump=1;jal=1;ui=0;**

**end**

**default:**

**begin**

**ALUOp=4'b0000;ALUSrc=0;RegDst=0;RegWrite=0;MemWrite=0;**

**MemtoReg=0;MemRead=0;Branch=0;bne=0;Jump=0;jal=0;ui=0;**

**end**

**endcase**

**end**

**endmodule**

**（2）程序计数器（PC）**

**module PC (**

**input clk, reset,**

**input[31:0] newPC,**

**output reg[31:0] PC);**

**always @ (posedge clk or posedge reset)**

**begin**

**if (reset == 1 )**

**PC <=0;**

**else**

**PC <= newPC;**

**end**

**endmodule**

**module PC\_adder(**

**input [31:0] PC,**

**output[31:0] PCplus4);**

**assign PCplus4 = PC + 4;**

**endmodule**

**（3）程序存储器（memory）**

**module Instruction\_Memory (**

**input [31:0] PC,**

**output reg [31:0] inst);**

**reg [7:0] memory[199:0];**

**initial begin**

**memory[0] = 0;**

**memory[1] = 1;**

**memory[2] = 2;**

**memory[3] = 3;**

**memory[4] = 4;**

**memory[5] = 5;**

**memory[6] = 6;**

**memory[7] = 7;**

**memory[8] = 8;**

**memory[9] = 9;**

**end**

**always @ (PC)**

**inst = {memory[PC],memory[PC+1],memory[PC+2],memory[PC+3]};**

**endmodule**

**（4）程序寄存器（register）**

**module registers(**

**input RegWrite,clk,**

**input[4:0] reg1ad,reg2ad,writead,**

**input[31:0] data\_in,**

**output wire[31:0] reg1data,reg2data);**

**reg[31:0] register[31:0];**

**initial begin register[0]=0; end**

**assign reg1data = register[reg1ad];**

**assign reg2data = register[reg2ad];**

**always @ (posedge clk)**

**begin**

**if(RegWrite==1) register[writead]<=data\_in;**

**end**

**endmodule**

**（5）ALU**

**module ALU (**

**input[31:0] reg1data, ALU2,**

**input[3:0] ALUfunc,**

**input[4:0] shamt,**

**output reg [31:0] ALUout,**

**output reg zero,overflow);**

**reg[5:0] i;**

**always @ (\*)**

**begin**

**case (ALUfunc)**

**4'b0010: ALUout = reg1data + ALU2;**

**4'b0110: ALUout = reg1data - ALU2;**

**4'b0000: ALUout = reg1data & ALU2;**

**4'b0001: ALUout = reg1data | ALU2;**

**4'b1001: ALUout = reg1data ^ ALU2;**

**4'b1010: ALUout = ALU2 << shamt;**

**4'b1100: ALUout = ALU2 >> shamt;**

**4'b0111:begin**

**if (reg1data[31] == ALU2[31])**

**begin**

**if (reg1data < ALU2) ALUout = 1;**

**else ALUout = 0;**

**end**

**else**

**begin**

**if (reg1data > ALU2) ALUout = 1;**

**else ALUout = 0;**

**end**

**end**

**4'b1101: ALUout =ALU2 << 16;**

**4'b1110: begin**

**if (reg1data <ALU2) ALUout = 1;**

**else ALUout = 0;**

**end**

**4'b1011: begin**

**ALUout = ALU2;**

**for (i=0;i<=31;i=i+1)**

**begin**

**if (i < shamt) ALUout = {ALUout[31],ALUout[31:1]};**

**end**

**end**

**default: ALUout = 0;**

**endcase**

**if (ALUout == 0) zero = 1;else zero = 0;**

**end**

**endmodule**

**（6）数据存储单元（Data\_Memory）**

**module Data\_Memory (**

**input clk, MemRead, MemWrite,**

**input [31:0] ALUout, reg2data,**

**output reg [31:0] memout);**

**reg [31:0] memory[199:0];**

**initial begin**

**memory[8] = 32'b11110000111100001111000011110000;**

**memory[10] = 32'b00000000000000000000000000000101;**

**end**

**always @ (ALUout, MemRead)**

**begin**

**if (MemRead == 1) memout = memory[ALUout];**

**end**

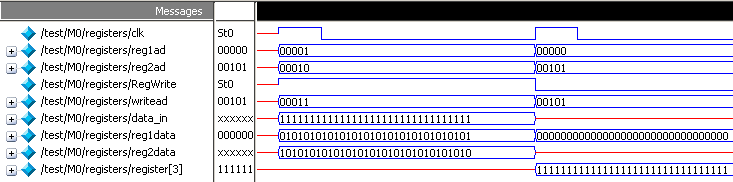
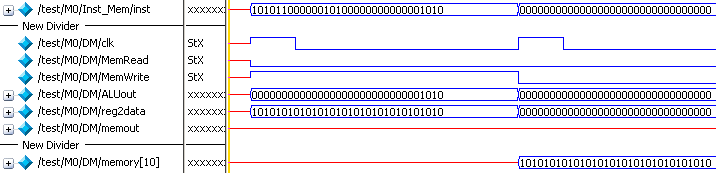
**always @ (posedge clk)**

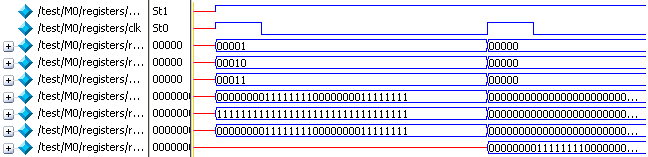
**begin**

**if (MemWrite == 1) memory[ALUout] <= reg2data;**

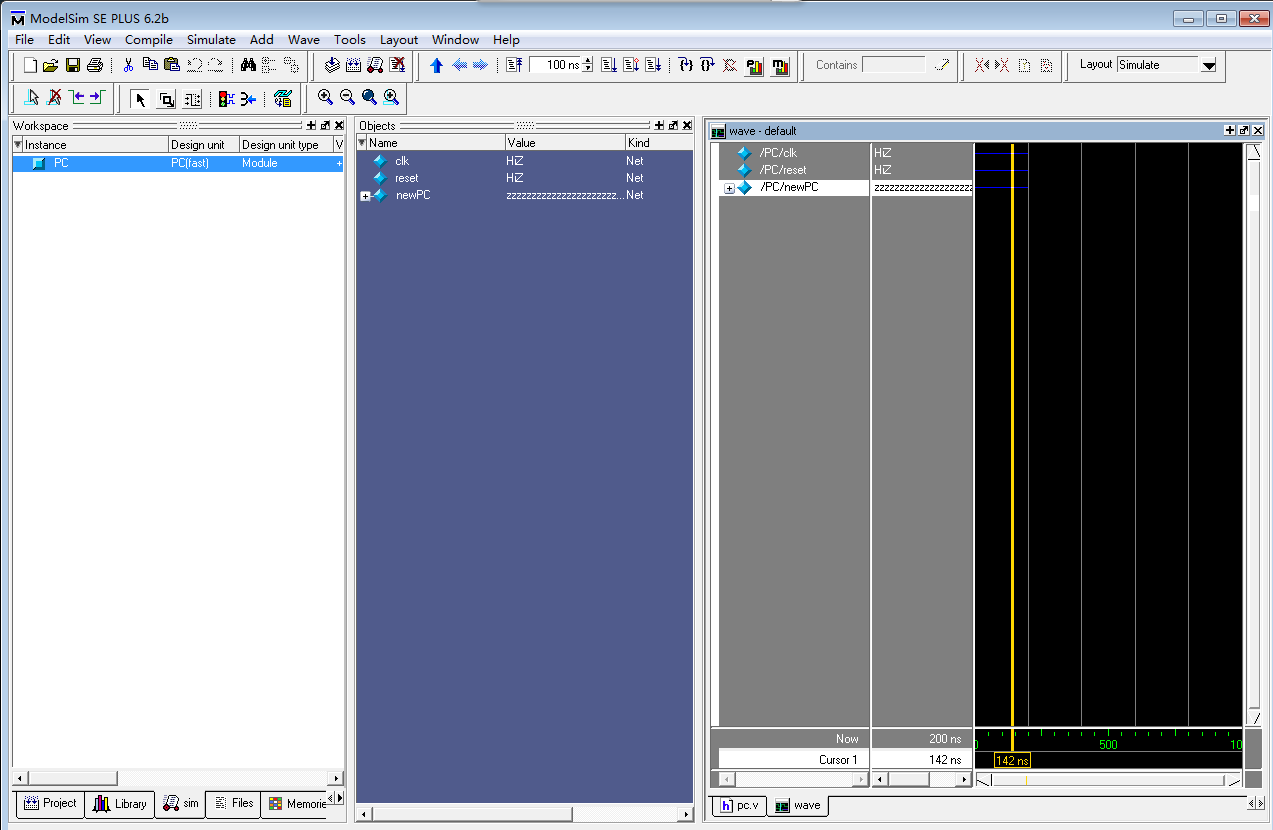
**end**

**endmodule**

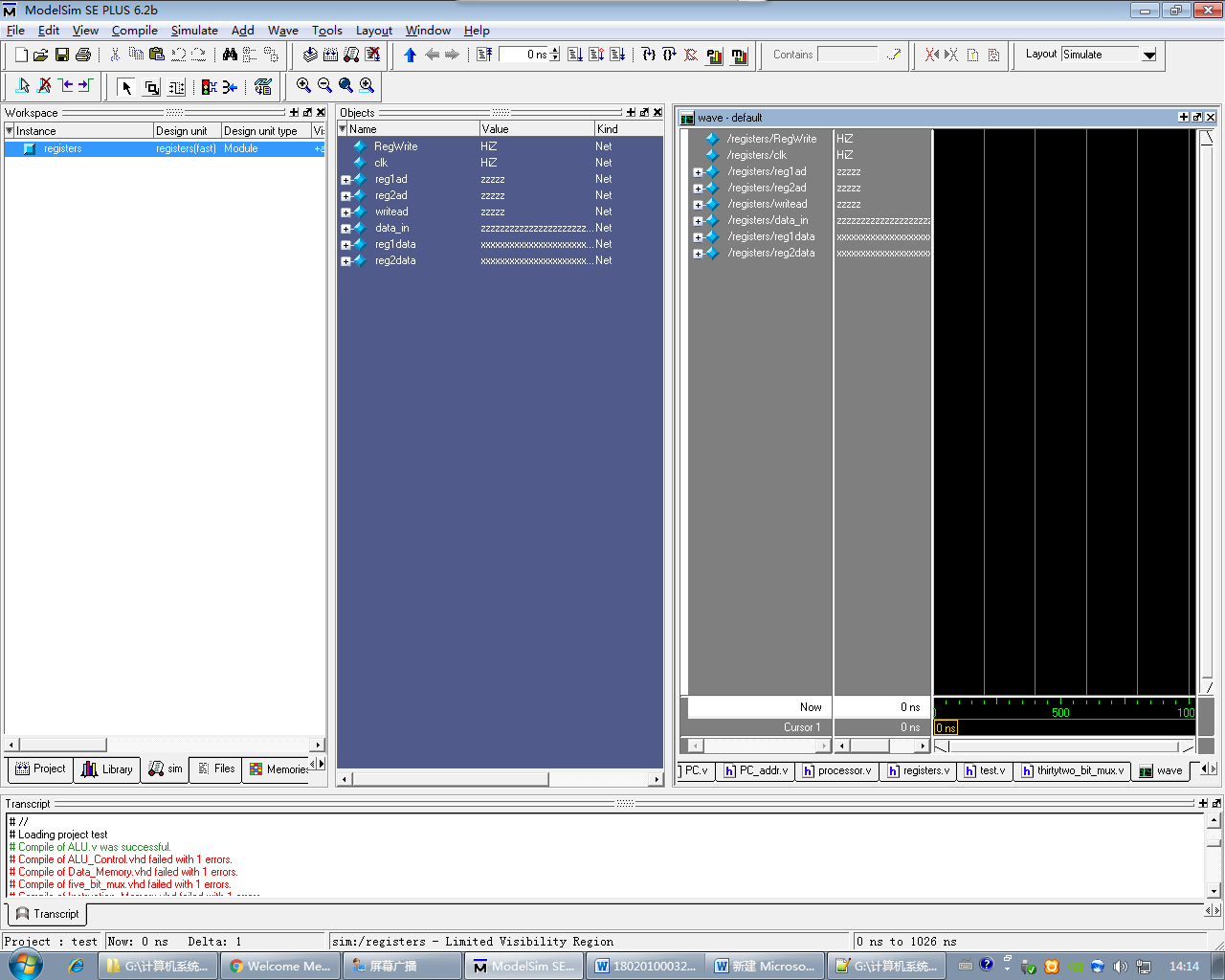
**主要仿真截图：**

****

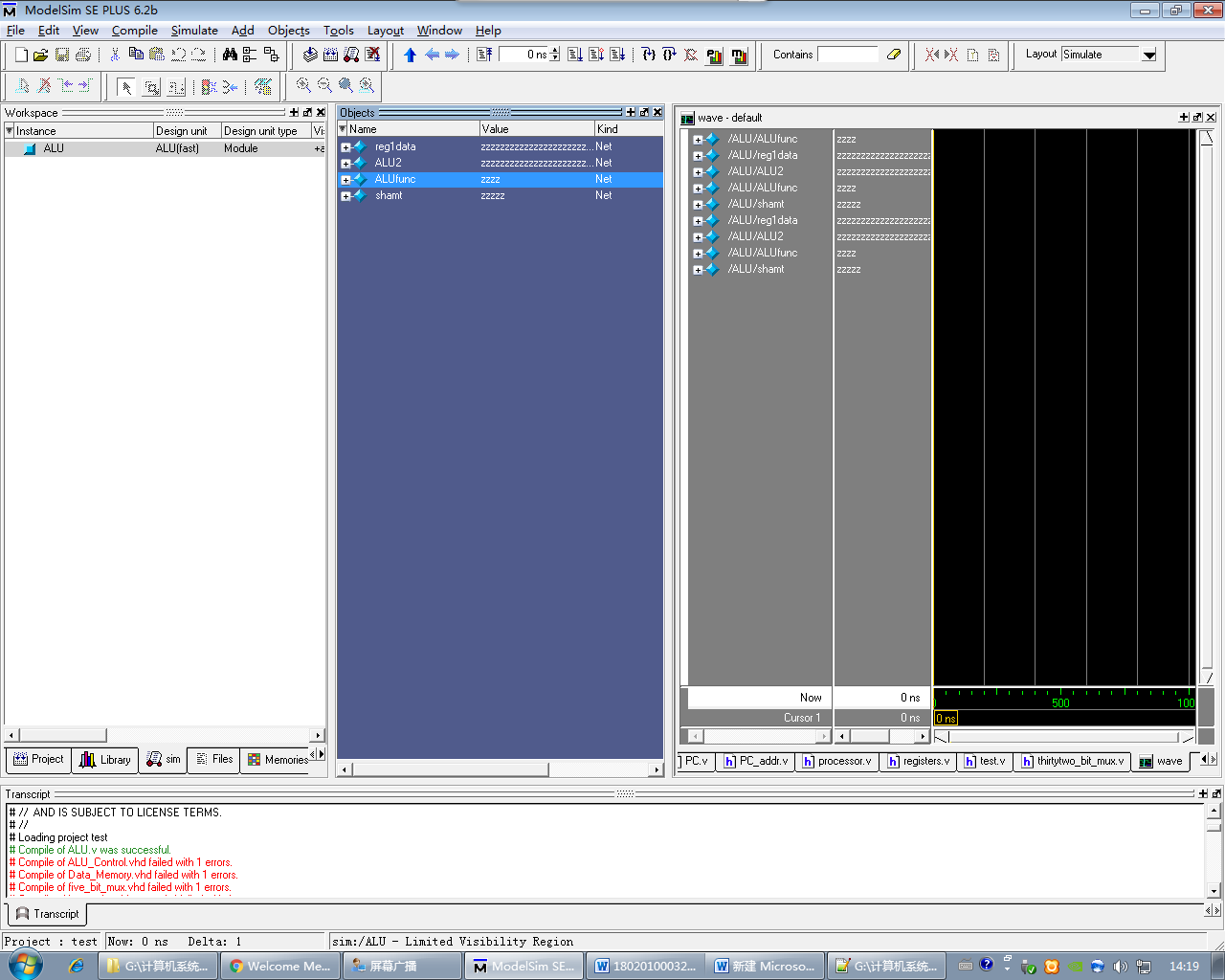
**（1）程序计数器**

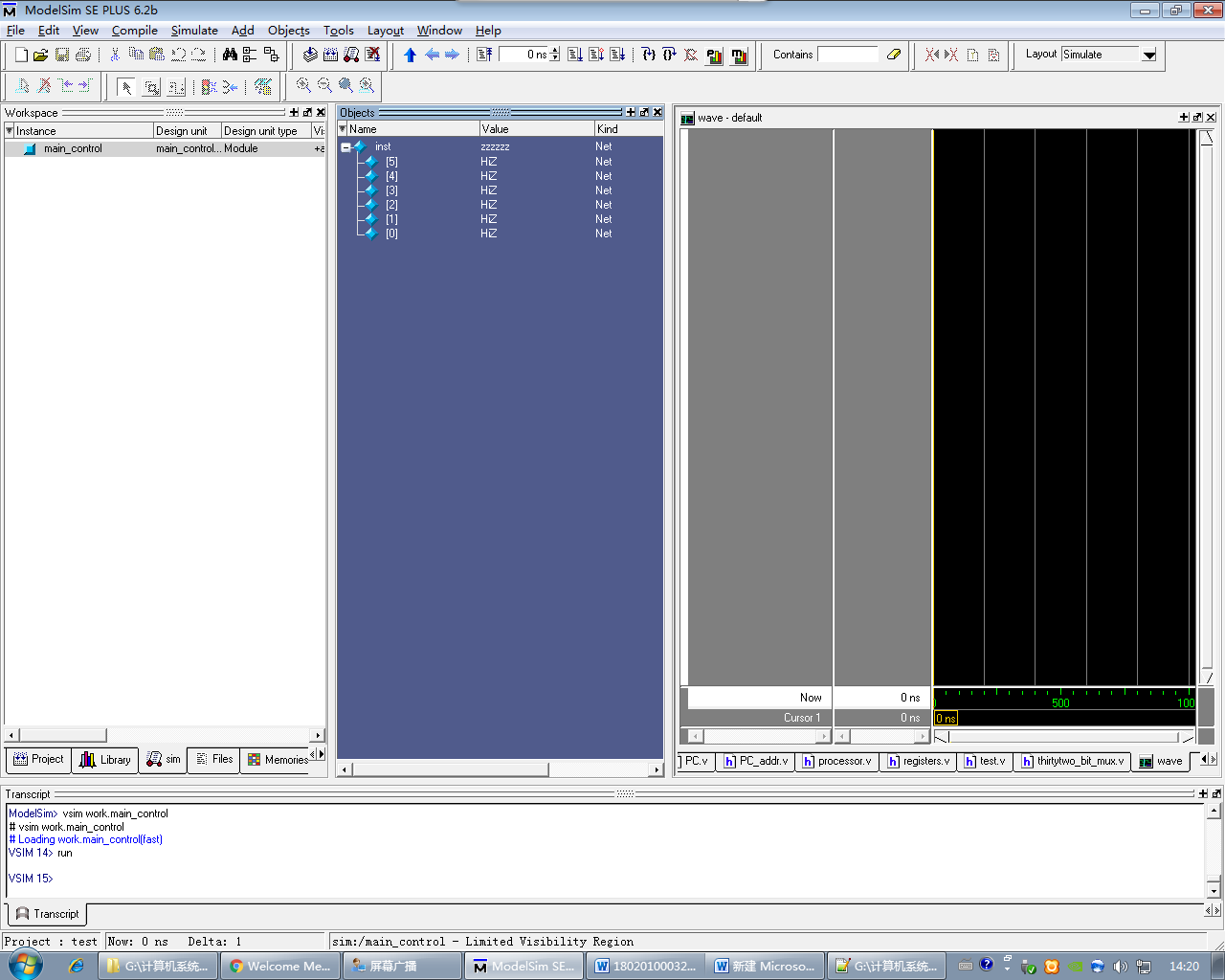


**（2）寄存器**

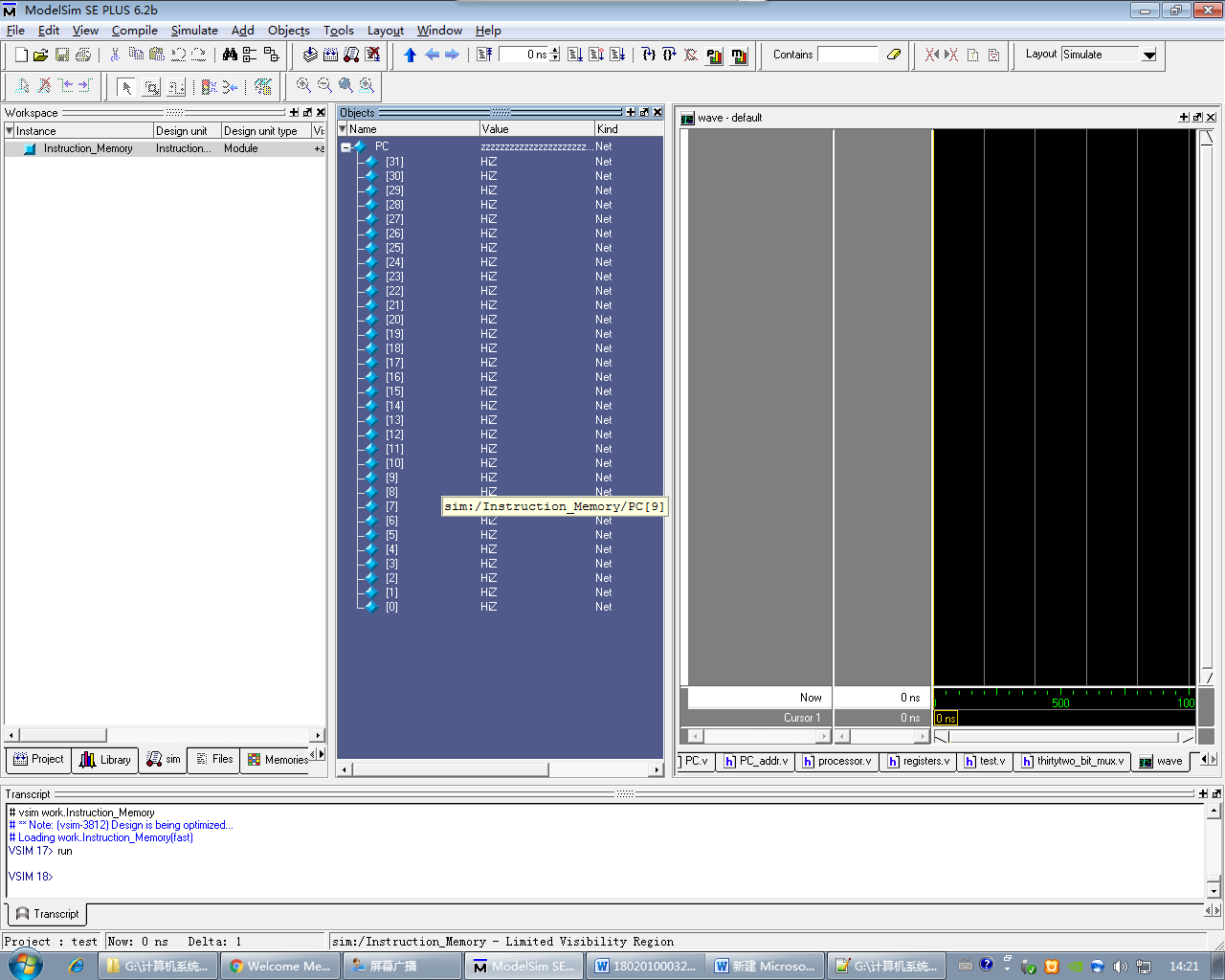
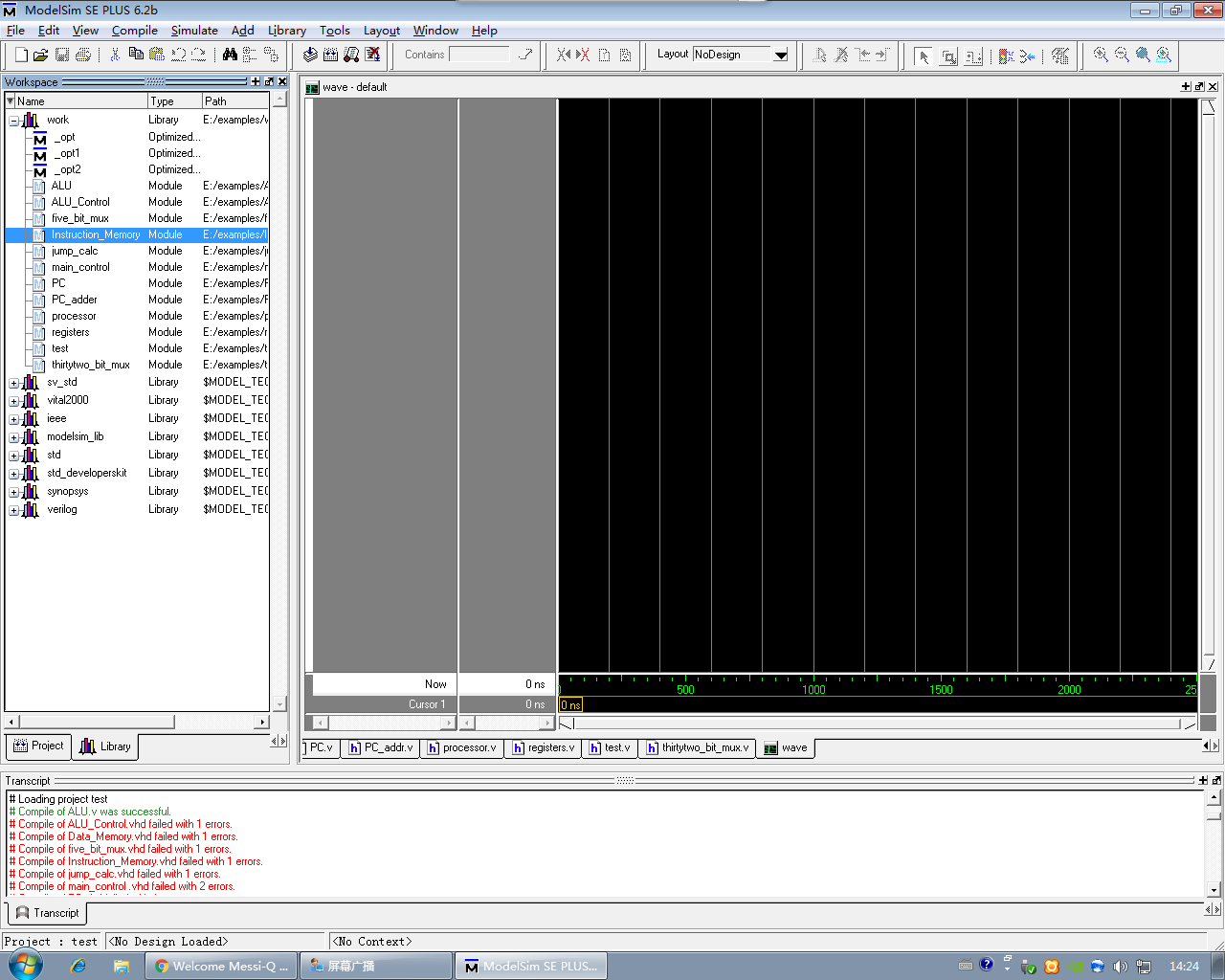


**（3）ALU**



**（4）控制器**

**（5）Memory**

**总体图**

# **五、实验总结**

本实验实现了一个单周期mipscpu，实现了算术运算、逻辑运算、移位、比较、存储器读/写、分支、跳转这些指令。实验的关键在于弄清楚cpu以及每条指令的数据通路，还有相应的控制信号的具体赋值。在具体实现的时候，最为重要的的是确保指令的跳转正确，在此基础上对各个指令的执行debug就轻而易举。需要注意的是，在实例化的时候各个对应的接口较为繁杂，应当注意不要出错，命名简单易懂。