数字系统II实验

类MIPS单周期微处理器设计

实验目的

- 了解微处理器的基本结构
- 掌握哈佛结构的计算机工作原理
- 学会设计简单的微处理器
- 了解软件控制硬件工作的基本原理

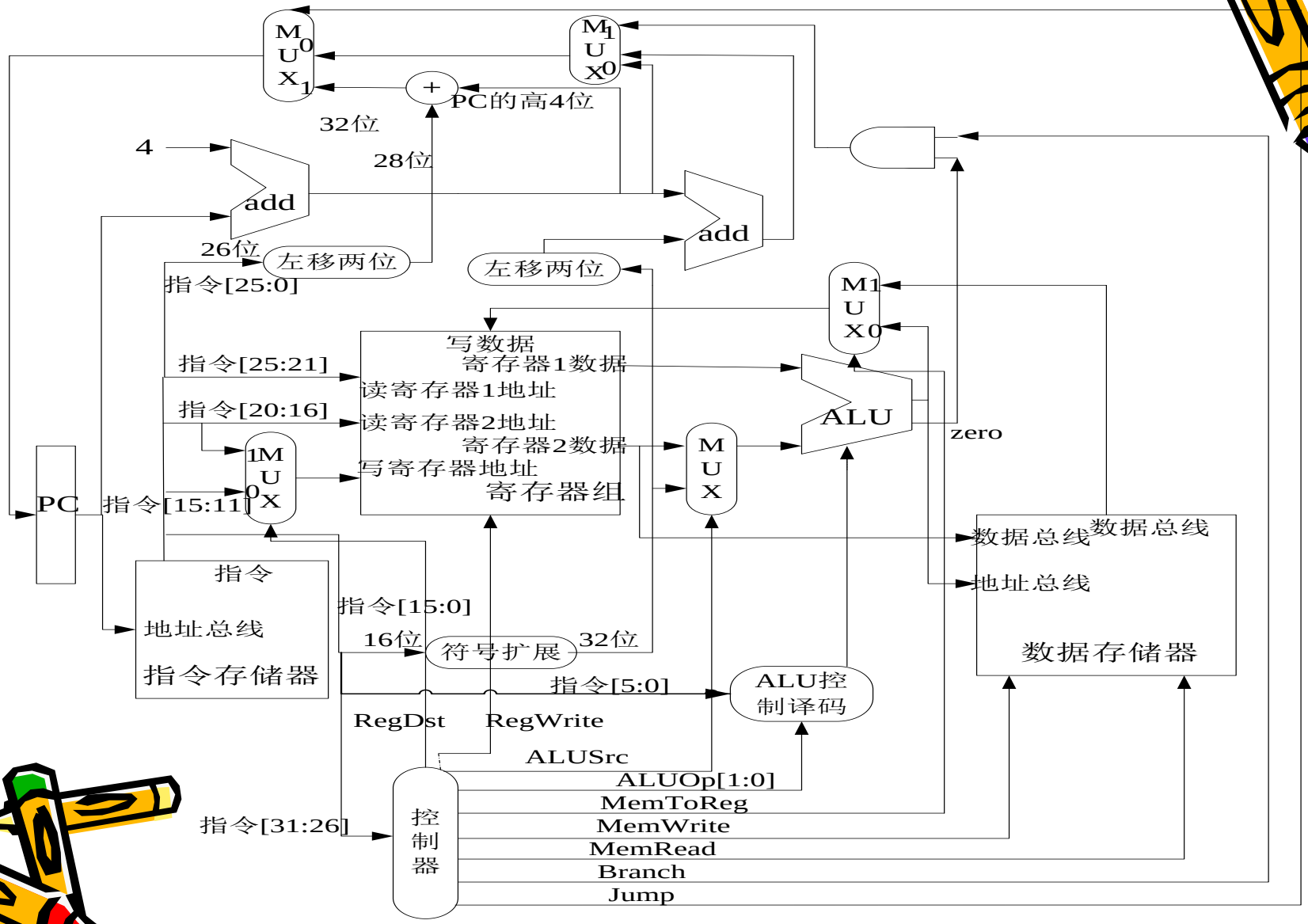
实验任务（第6、7章内容）

- 利用HDL语言，基于Xilinx FPGA nexys4实验平台，设计一个能够执行以下MIPS指令集的单周期类MIPS处理器，要求完成所有支持指令的功能仿真，验证指令执行的正确性，要求编写汇编程序将本人学号的ASCII码存入RAM的连续内存区域
  - 支持基本的内存操作如lw，sw指令
  - 支持基本的算术逻辑运算如add，sub，and，or，slt，andi指令
  - 支持基本的程序控制如beq，j指令

时间安排

- 课内
  - 两次课
- 课外
  - 两周时间内自行添加

基本原理


M U X
M U X
+
PC的高4位
32位
28位
4
add
add
26位
左移两位
左移两位
指令[25:0]
M1 U X0
写数据
寄存器1数据
指令[25:21]
读寄存器1地址
指令[20:16]
读寄存器2地址
寄存器2数据
写寄存器地址
寄存器组
ALU
zero
1M U 0X
M U X
PC
指令[15:11]
数据总线
数据总线
地址总线
指令
地址总线
指令存储器
指令[15:0]
16位
符号扩展
32位
数据存储器
指令[5:0]
ALU控制译码
RegDst
RegWrite
ALUSrc
ALUOp[1:0]
MemToReg
MemWrite
MemRead
Branch
Jump
指令[31:26]
控制器

MIPS处理器基本架构

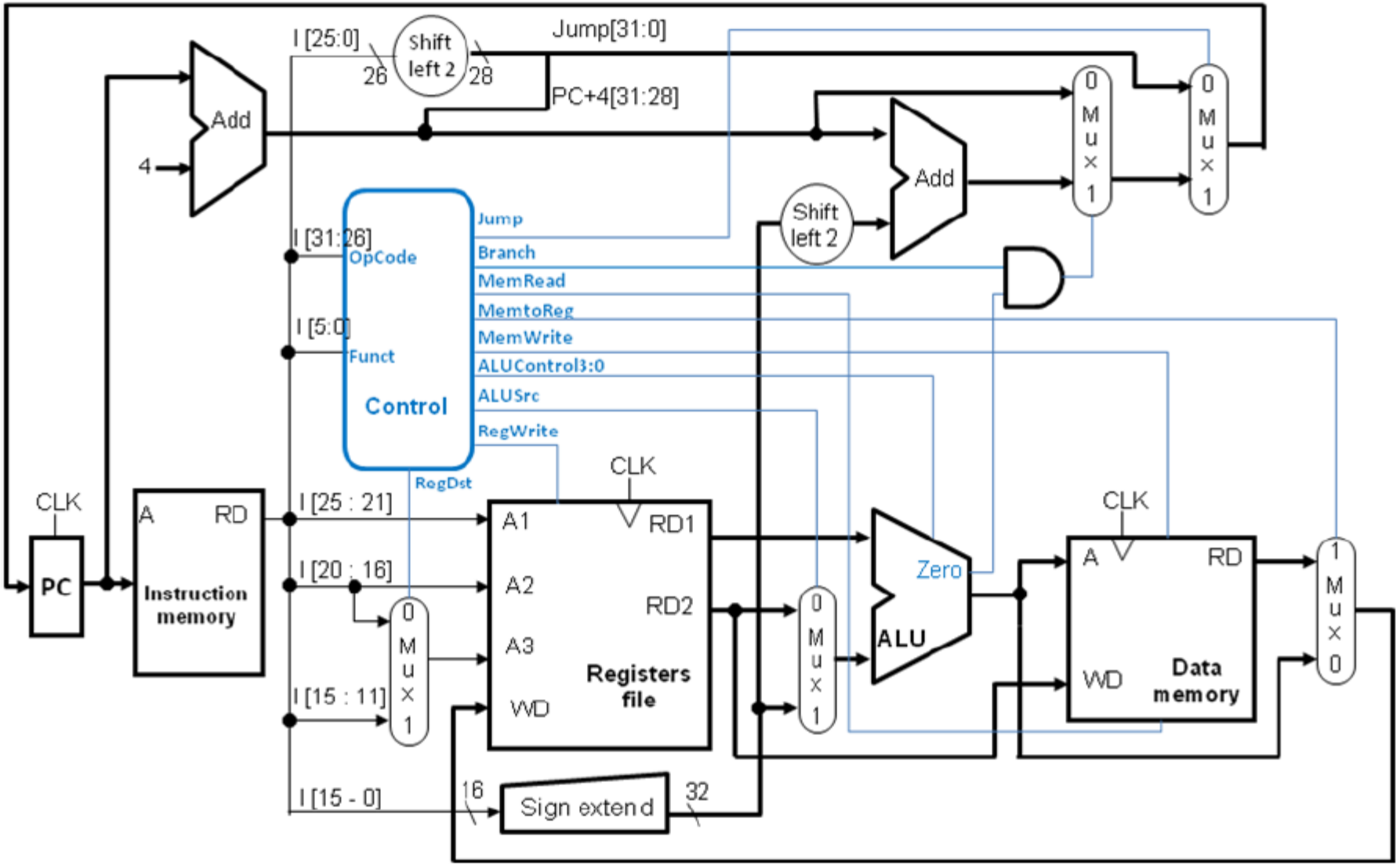
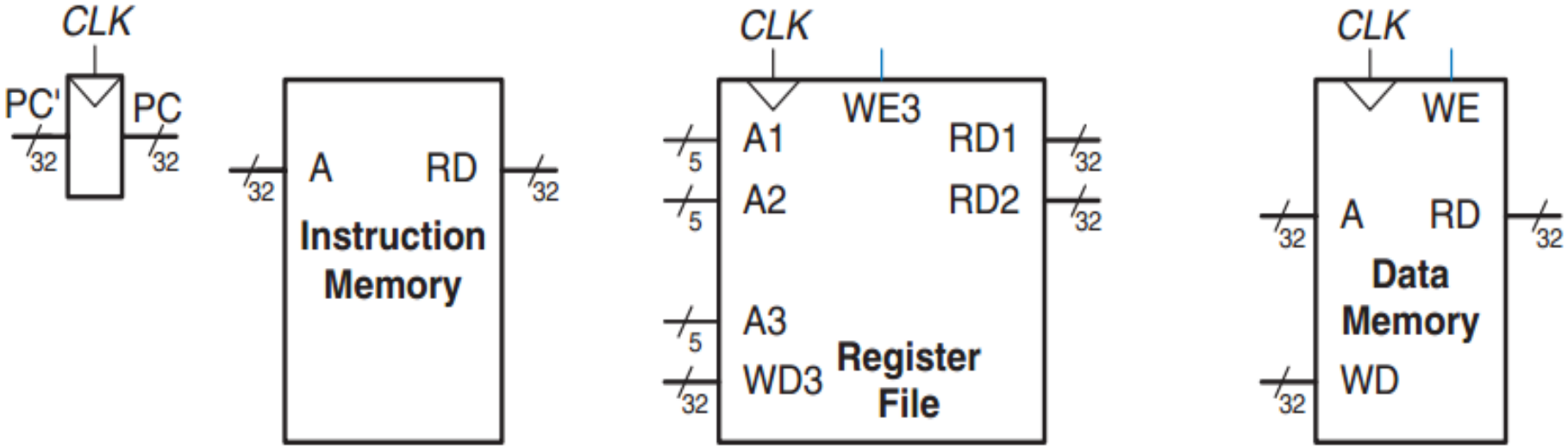
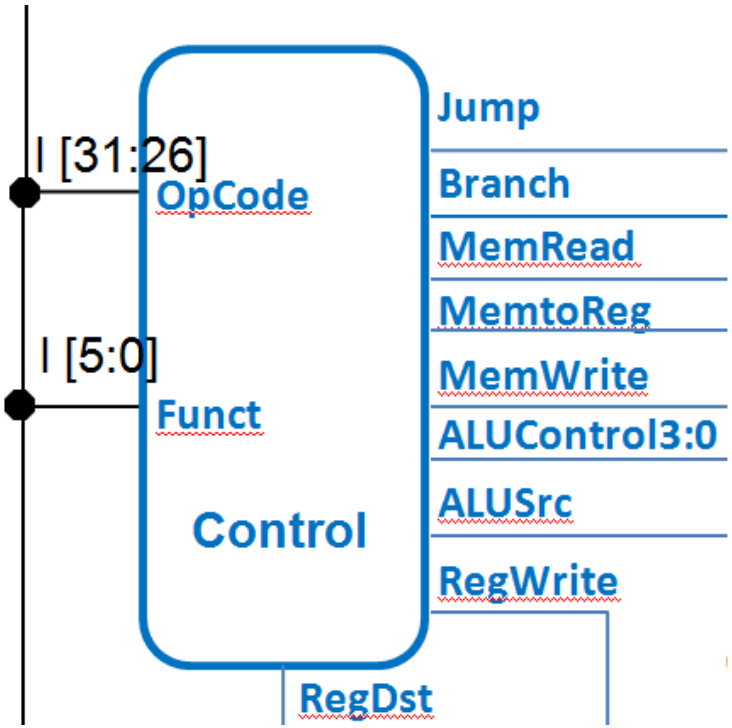


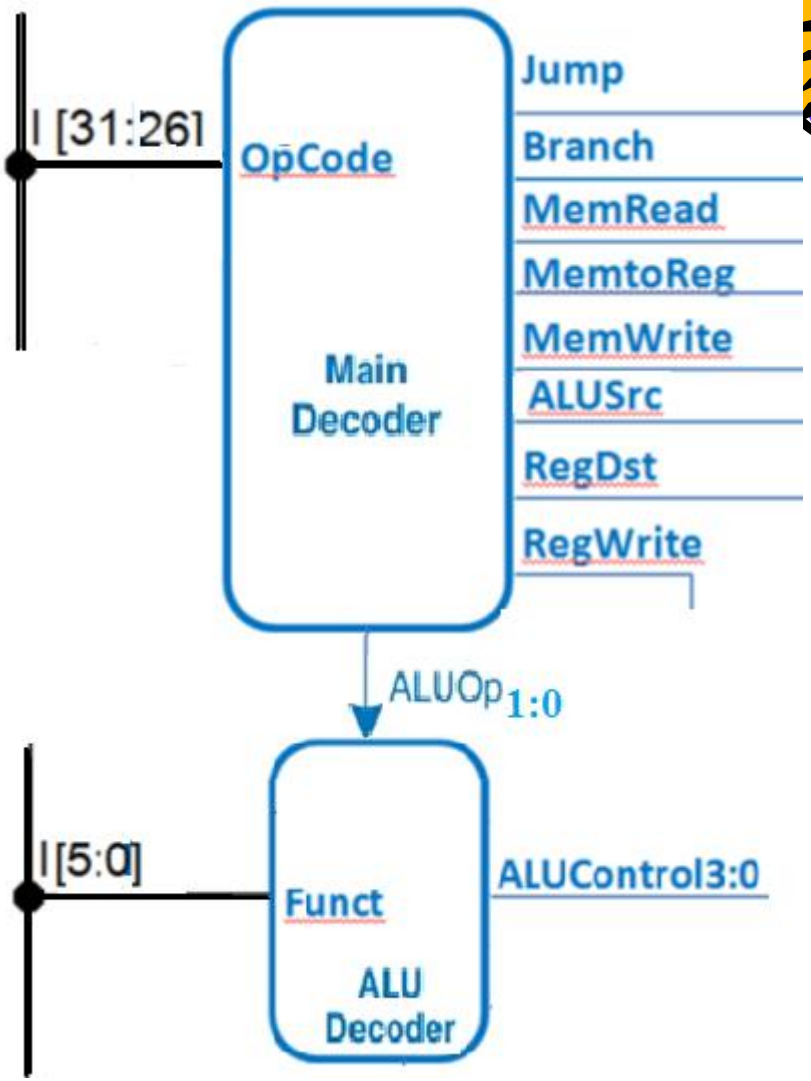
图 1. MIPS 处理器基本架构

Datapath 单元电路设计

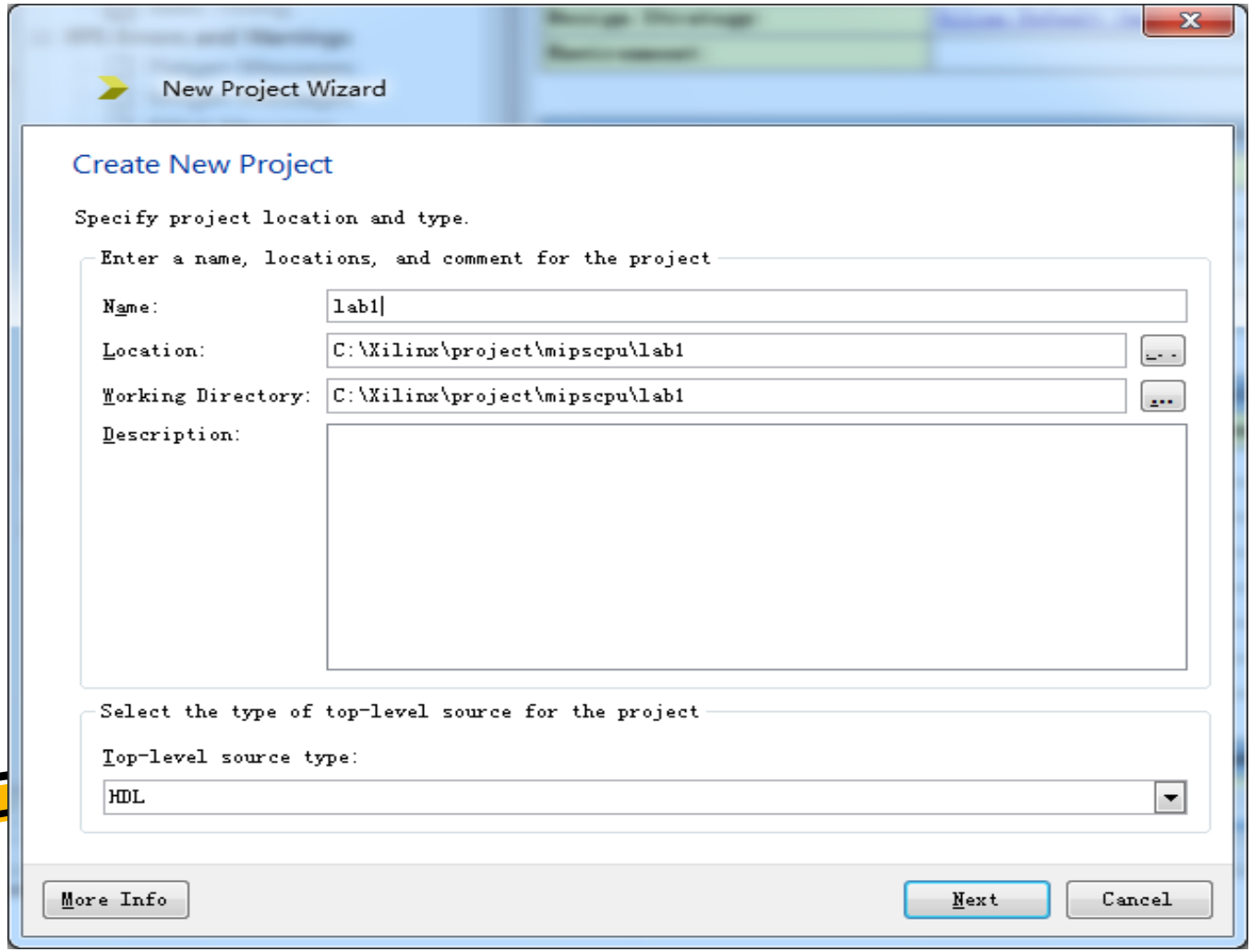


CPU控制器 和 ALU 单元电路设计





ISE新建工程


New Project Wizard
Create New Project
Specify project location and type.
Enter a name, locations, and comment for the project
Name:
lab1
Location:
C:\Xilinx\project\mipscpu\lab1
Working Directory:
C:\Xilinx\project\mipscpu\lab1
Description:
Select the type of top-level source for the project
Top-level source type:
HDL
More Info
Next
Cancel

New Project Wizard

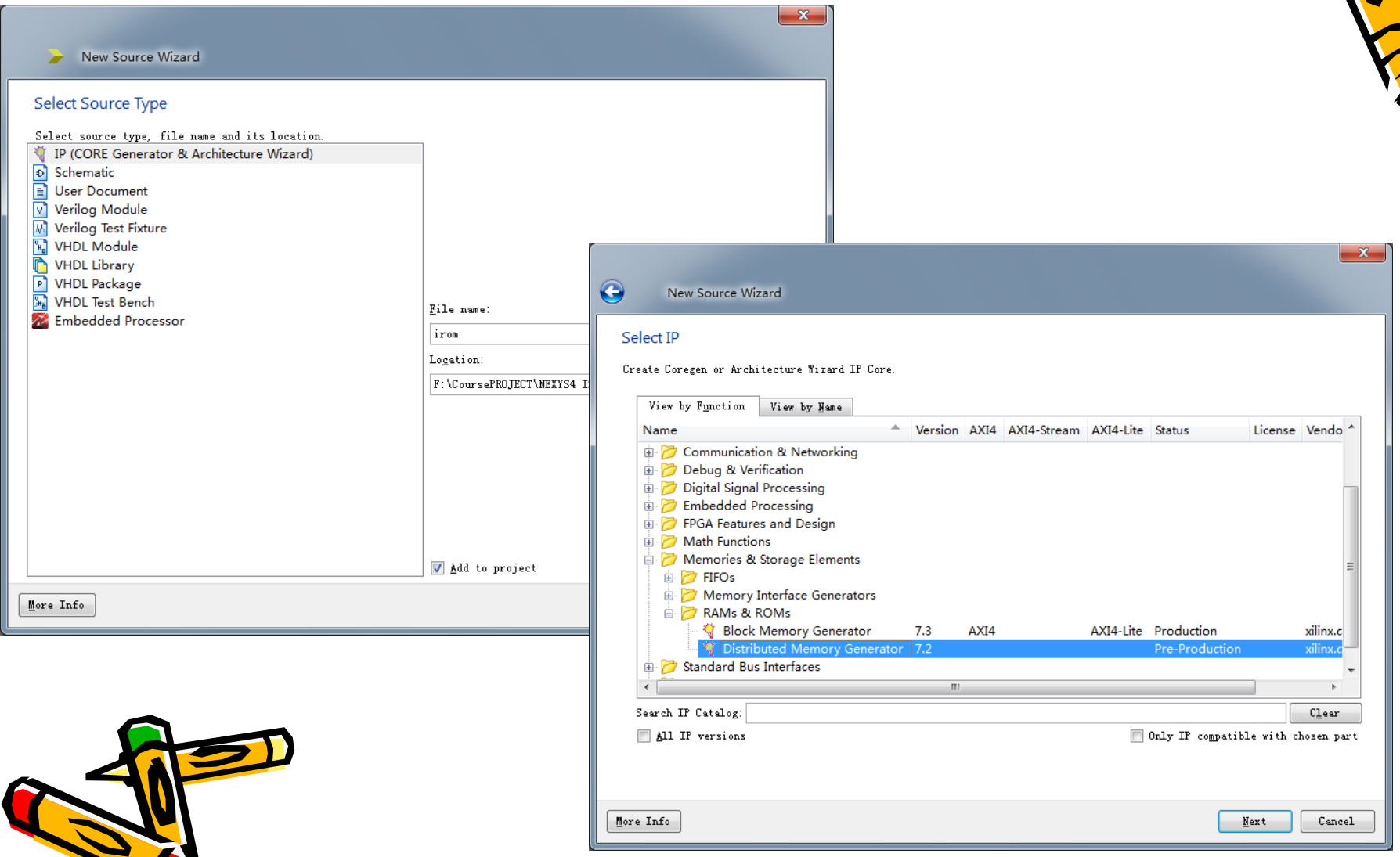
Project Settings

Specify device and project properties.
Select the device and design flow for the project

Property Name	Value
Evaluation Development Board	None Specified
Product Category	All
Family	Artix7
Device	XC7A100T
Package	CSG324
Speed	-1
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	Verilog
Property Specification in Project File	Store all values
Manual Compile Order	☐
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	☐

More Info | Next | Cancel

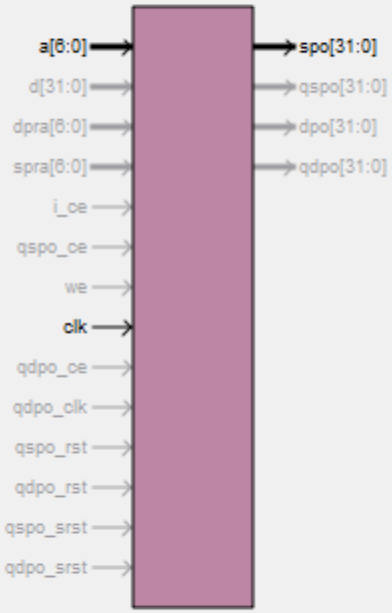
新建存储器模块


New Source Wizard
Select Source Type
Select source type, file name and its location.
IP (CORE Generator & Architecture Wizard)
Schematic
User Document
Verilog Module
Verilog Test Fixture
VHDL Module
VHDL Library
VHDL Package
VHDL Test Bench
Embedded Processor
File name:
irom
Location:
F:\CoursePROJECT\NEXYS4 I
Add to project
More Info
New Source Wizard
Select IP
Create Coregen or Architecture Wizard IP Core.
View by Function
View by Name
Name
Version
AXI4
AXI4-Stream
AXI4-Lite
Status
License
Vendo
Communication & Networking
Debug & Verification
Digital Signal Processing
Embedded Processing
FPGA Features and Design
Math Functions
Memories & Storage Elements
FIFOs
Memory Interface Generators
RAMs & ROMs
Block Memory Generator 7.3 AXI4 AXI4-Lite Production xilinx.c
Distributed Memory Generator 7.2 Pre-Production xilinx.c
Standard Bus Interfaces
Search IP Catalog:
Clear
All IP versions
Only IP compatible with chosen part
More Info
Next
Cancel

Distributed Memory Generator

View Documents

IP Symbol



Distributed Memory Generator

xilinx.com:ip:dist_mem_gen:7.2

Component Name irom

Options

Depth 128 Range: 16..65536

Data Width 32 Range: 1..1024

Memory Type

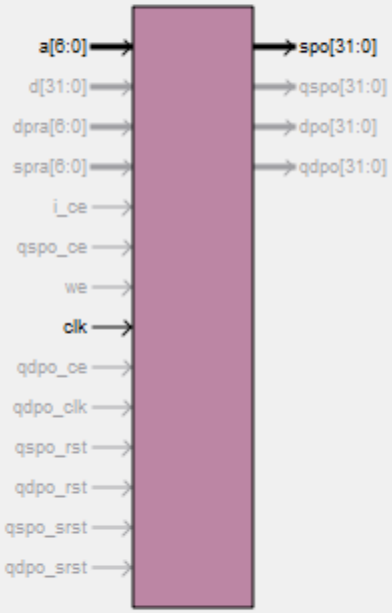
- (●) ROM
- ( ) Dual Port RAM
- ( ) Simple Dual Port RAM
- ( ) Single Port RAM
- ( ) SRL16-based Memory

Datasheet | < Back | Page 1 of 3 | Next > | Generate | Cancel | Help

Distributed Memory Generator

View Documents

IP Symbol



Distributed Memory Generator

xilinx.com:ip:dist_mem_gen:7.2

Input Options

- (○) Non Registered
- (●) Registered
- [ ] Input Clock Enable
- [ ] Qualify WE with I_CE

Dual Port Address

- (●) Non Registered
- (○) Registered

Output Options

- (●) Non Registered
- (○) Registered
- (○) Both
- [ ] Common Output CLK
- [ ] Single Port Output CE
- [ ] Common Output CE
- [ ] Dual Port Output CE

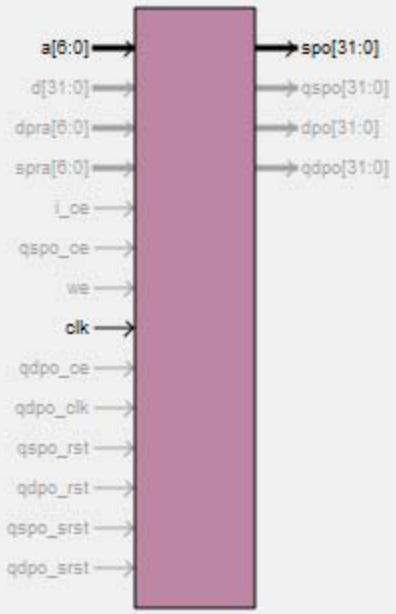
Pipelining Options

Pipeline Stages: 0

Datasheet | < Back | Page 2 of 3 | Next > | Generate | Cancel | Help

View Documents

IP Symbol



LogiCORE

Distributed Memory Generator

xilinx.com:ip:dist_mem_gen:7.2

Load COE File

If desired the initial memory content can be set by using a COE file. Initialisation File (MIF).

Coefficients File : C:\Xilinx\projectNexys3 Browse... Show...

COE Options

Default Data : 0 Radix : 16

Reset Options

Reset QSPO

Synchronous Reset QSPO

CE Overrides Sync Controls

COE File Contents

Radix: 16

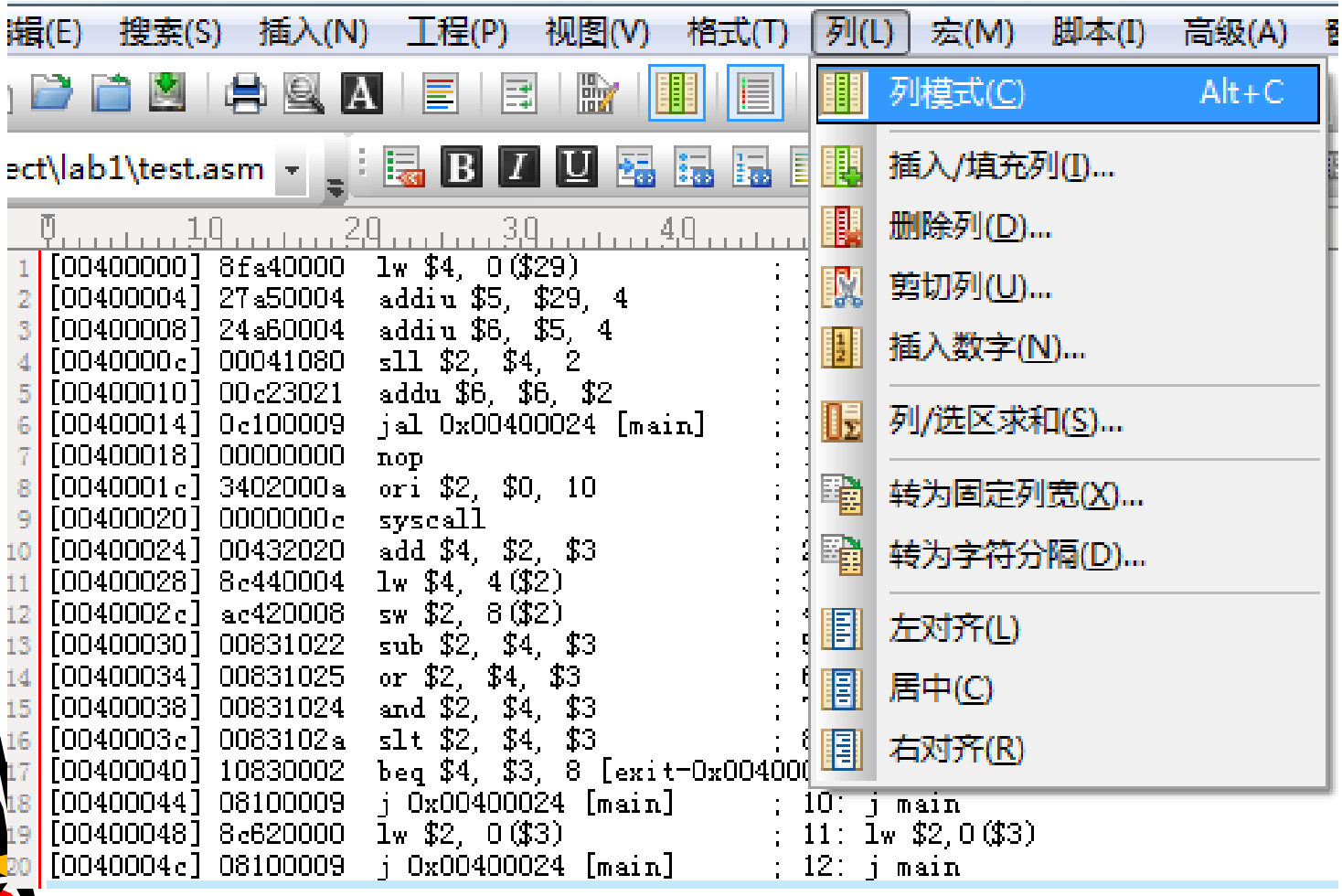
COE Vector: memory_initialization_vector

Index	Value
0	00432020
1	8c440004
2	ac420008
3	00831022
4	00831025
5	00831024
6	0083102a
7	10830002
8	08000000
9	8c620000

Close Help

Datasheet < Back Page 3 of 3 Next > Generate Cancel Help

COE文件制作


辑(E) 搜索(S) 插入(N) 工程(P) 视图(V) 格式(T) 列(L) 宏(M) 脚本(I) 高级(A)
列模式(C) Alt+C
插入/填充列(I)...
删除列(D)...
剪切列(U)...
插入数字(N)...
列/选区求和(S)...
转为固定列宽(X)...
转为字符分隔(D)...
左对齐(L)
居中(C)
右对齐(R)
ect\lab1\test.asm
[00400000] 8fa40000 lw $4, 0($29) ;
[00400004] 27a50004 addiu $5, $29, 4 ;
[00400008] 24a60004 addiu $6, $5, 4 ;
[0040000c] 00041080 sll $2, $4, 2 ;
[00400010] 00c23021 addu $6, $6, $2 ;
[00400014] 0c100009 jal 0x00400024 [main] ;
[00400018] 00000000 nop ;
[0040001c] 3402000a ori $2, $0, 10 ;
[00400020] 0000000c syscall ;
[00400024] 00432020 add $4, $2, $3 ;
[00400028] 8c440004 lw $4, 4($2) ;
[0040002c] ac420008 sw $2, 8($2) ;
[00400030] 00831022 sub $2, $4, $3 ;
[00400034] 00831025 or $2, $4, $3 ;
[00400038] 00831024 and $2, $4, $3 ;
[0040003c] 0083102a slt $2, $4, $3 ;
[00400040] 10830002 beq $4, $3, 8 [exit-0x00400
[00400044] 08100009 j 0x00400024 [main] ; 10: j main
[00400048] 8c620000 lw $2, 0($3) ; 11: lw $2,0($3)
[0040004c] 08100009 j 0x00400024 [main] ; 12: j main

- mips 微处理器第一条指令对应的物理地址设计为0x00000000，
- 因此需将0x08100009 修改为0x08000000.

COE文件结构

```
MEMORY_INITIALIZATION_RADIX=16;
MEMORY_INITIALIZATION_VECTOR=
00432020,
8c440004,
ac420008,
00831022,
00831025,
00831024,
0083102a,
10830002,
08000000,
8c620000,
08000000,
```

RAM模块与ROM模块类似方法

符号扩展以及寄存器组

- 采用veriloa实现

```
////////////////////////////////////////////////////////////////////////////////
module signext(
    input [15:0] inst,
    output [31:0] data
    );

assign data=inst[15:15]?{16'hffff,inst}:{16'h0000,inst};

endmodule
```

```
reg [31:0]   regs [0:31];
assign RsData = (RsAddr == 5'b0) ? 32'b0 : regs[RsAddr];
assign RtData = (RtAddr == 5'b0) ? 32'b0 : regs[RtAddr];
 integer i;
 always @ (posedge clk)
begin
   if (!reset)
      begin
      if(regWriteEn==1)
         begin
         regs[regWriteAddr]=regWriteData;
         end
      end
   else
      begin
         for(i=0;i<32;i=i+1)
            regs[i]=0;
      end
   end
```

控制器模块

```
//////////////////////////
module ctr(
    input [5:0] opCode,
    output regDst,
    output aluSrc,
    output memToReg,
    output regWrite,
    output memRead,
    output memWrite,
    output branch,
    output [1:0] aluop,
    output jmp
    );

endmodule
```

```
    reg regDst;
    reg aluSrc;
    reg memToReg;
    reg regWrite;
    reg memRead;
    reg memwrite;
    reg branch;
    reg [1:0] aluop;
    reg jmp;
always @ (opcode)
begin
case (opcode)
   6'b000010://jmp
   begin
       regDst=0;
     aluSrc=0;
    memToReg=0;
     regWrite=0;
     memRead=0;
     memwrite=0;
     branch=0;
     aluop=2'b00;
     jmp=1;
     end
```

指令	RegDst	Jump	Branch	MemToReg	ALUSrc	RegWrite	MemWrite	MemRead	ALUOp1	ALUOp0
R型	1	0	0	0	0	1	0	0	1	0
lw	0	0	0	1	1	1	0	1	0	0
sw	X	0	0	X	1	0	1	0	0	0
beq	X	0	1	X	0	0	0	0	0	1
j	X	1	0	X	X	0	0	0	X	X

ALU控制器

指令	2位操作码	指令功能	6位功能码	ALU的运算	ALU的控制信号
LW	00	取字	XXXXXX	加	0010
SW	00	存字	XXXXXX	加	0010
BEQ	01	相等跳转	XXXXXX	减	0110
R型指令	10	加	100000	加	0010
R型指令	10	减	100010	减	0110
R型指令	10	与	100100	与	0000
R型指令	10	或	100101	或	0001
R型指令	10	小于设置	101010	小于设置	0111

```
module aluctr(
    input [1:0] ALUOp,
    input [5:0] funct,
    output [3:0] ALUCtr
    );
reg[3:0] ALUCtr;
always @(ALUOp or funct)
casex({ALUOp,funct})
   8'b00xxxxxx: ALUCtr=4'b0010;
   8'bx1xxxxxx: ALUCtr=4'b0110;
   8'b1xxx0000: ALUCtr=4'b0010;
   8'b1xxx0010: ALUCtr=4'b0110;
   8'b1xxx0100: ALUCtr=4'b0000;
   8'b1xxx0101: ALUCtr=4'b0001;
   8'b1xxx1010: ALUCtr=4'b0111;
   endcase
endmodule
```

ALU

输入信号	操作类型
0000	与
0001	或
0010	加
0110	减
0111	小于设置

```
reg zero;
reg[31:0] aluRes;

always @(input1 or input2 or aluCtr)
begin
   case(aluCtr)
      4'b0110:
         begin
            aluRes=input1-input2;
               if (aluRes==0)
                  zero=1;
                  else
                  zero=0;
            end
      4'b0010:
         aluRes=input1+input2;
      4'b0000:
         aluRes=input1 & input2;
      4'b0001:
         aluRes=input1 | input2;
      4'b1100:
         aluRes= ~(input1 | input2);
      4'b0111:
         begin
            if(input1<input2)
               aluRes = 1;
         end
      default:
         aluRes = 0;
   endcase
end
endmodule
```

顶层

- 实现各个模块之间的连接
- 实例化各个模块

```
reg[31:0] pc;
wire choose4;
wire[31:0] expand2,add4,mux2,mux3,mux4,mux5,address,jmpaddr,inst;
wire[4:0] mux1;
//wire for controller
wire reg_dst,jmp,branch,memread,memwrite,memtoreg;
wire[1:0] aluop;
wire alu_src,regwrite;
//wire for aluunit
wire zero;
wire[31:0] aluRes;
//wire for aluctr
wire[3:0] aluCtr;
//wire for memory
wire[31:0] memreaddata;//memory data
//wire for register
wire[31:0] RsData,RtData;//regfile data
//wireforext
wire[31:0] expand;
wire clkout;
```

```
ctr mainctr(
    .opCode(inst[31:26]),
    .regDst(reg_dst),
    .aluSrc(alu_src),
    .memToReg(memtoreg),
    .regWrite(regwrite),
    .memRead(memread),
    .memWrite(memwrite),
    .branch(branch),
    .aluop(aluop),
    .jmp(jmp));
```

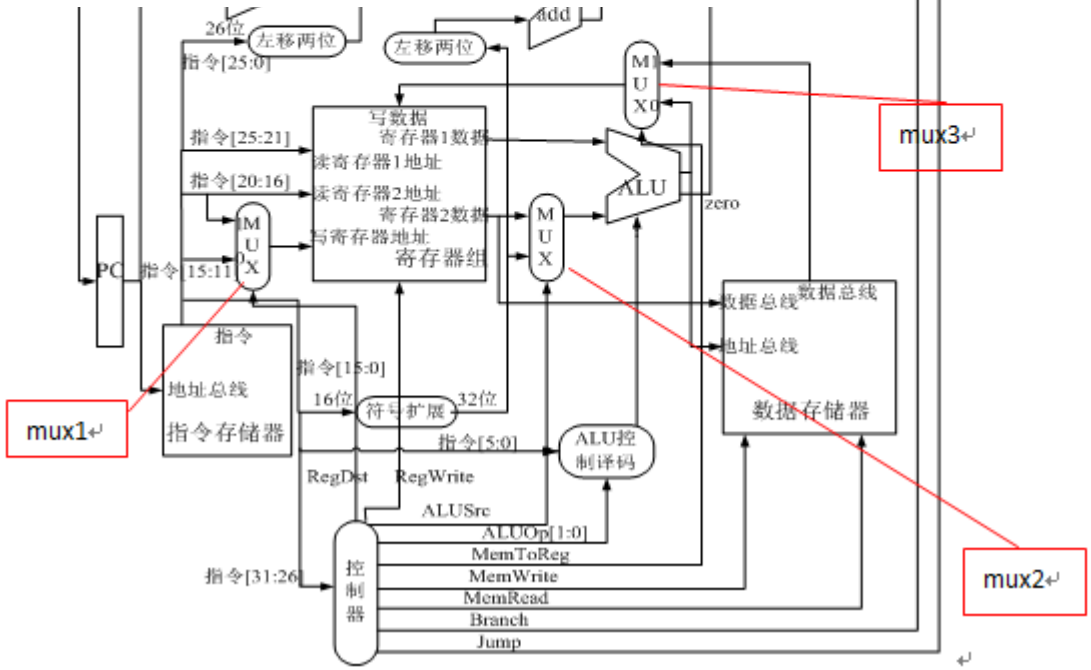
程序计数器PC

- 单周期
  - 执行指令时不能取指令
  - 取指令由ROM输出决定
  - 执行指令包括改变PC的值等等
  - 因此改变PC的值与从ROM读取指令不能同时进行，这里采用时钟的两个不同边沿实现
    - 一个采用上升沿
    - 一个采用下降沿
  - 执行指令时PC的值才进行改变

```
always @(negedge clkin)
    begin
        if(!reset) begin
            pc=mux5;
            add4=pc+4;
            end
            else
                begin
                    pc=32'b0;
                    add4=32'h4;
                    end
    end
```

其他

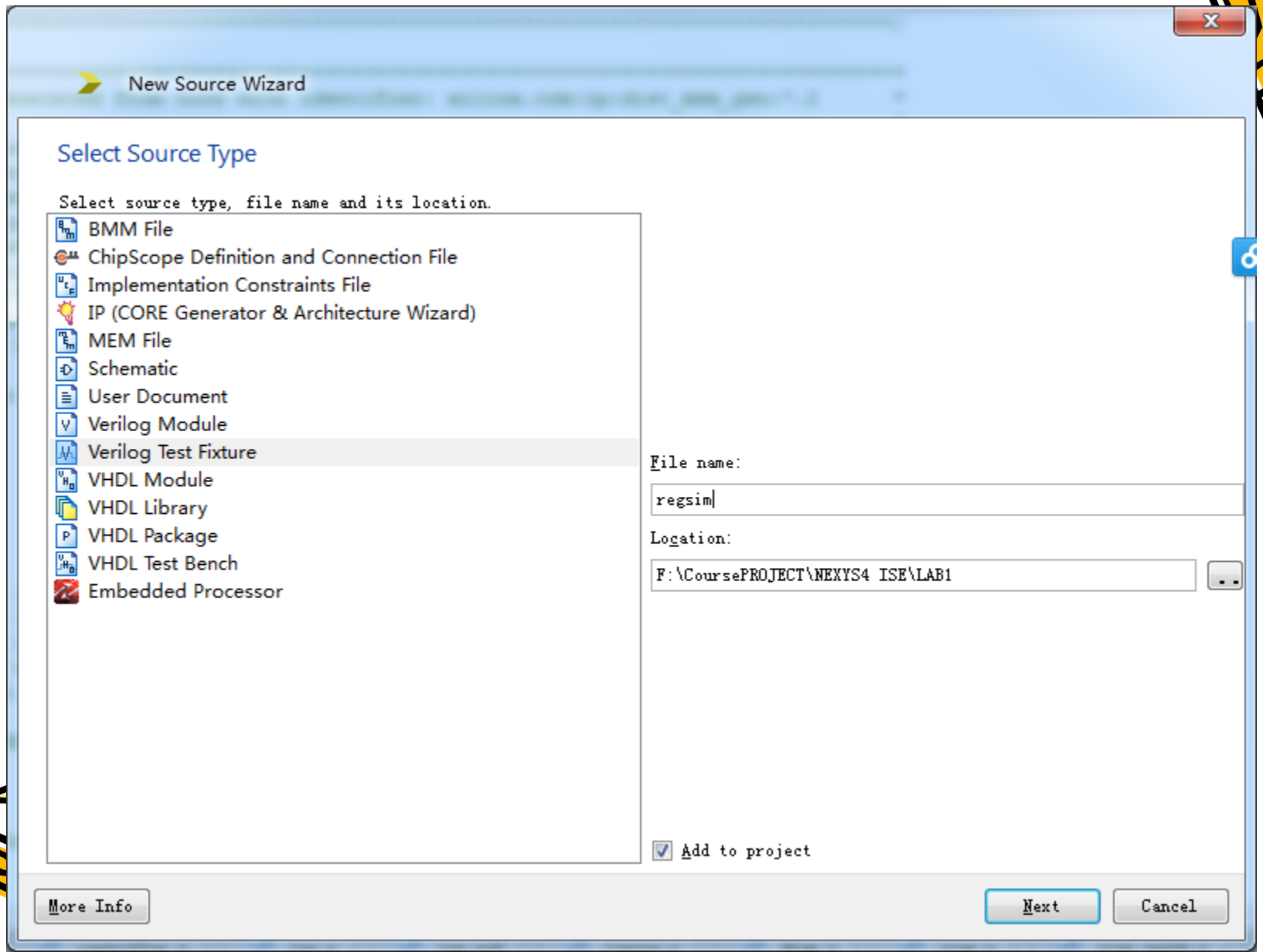
```
assign mux1=reg_dst?inst[15:11]:inst[20:16];
assign mux2=alu_src?expand:RtData;
assign mux3=memtoreg?memreaddata:aluRes;
assign mux4=choose4?address:add4;
assign mux5=jmp?jmpaddr:mux4;
assign choose4=branch&zero;
assign expand2=expand << 2;
assign jmpaddr={add4[31:28],inst[25:0],2'b00};
assign address=pc+expand2;
```



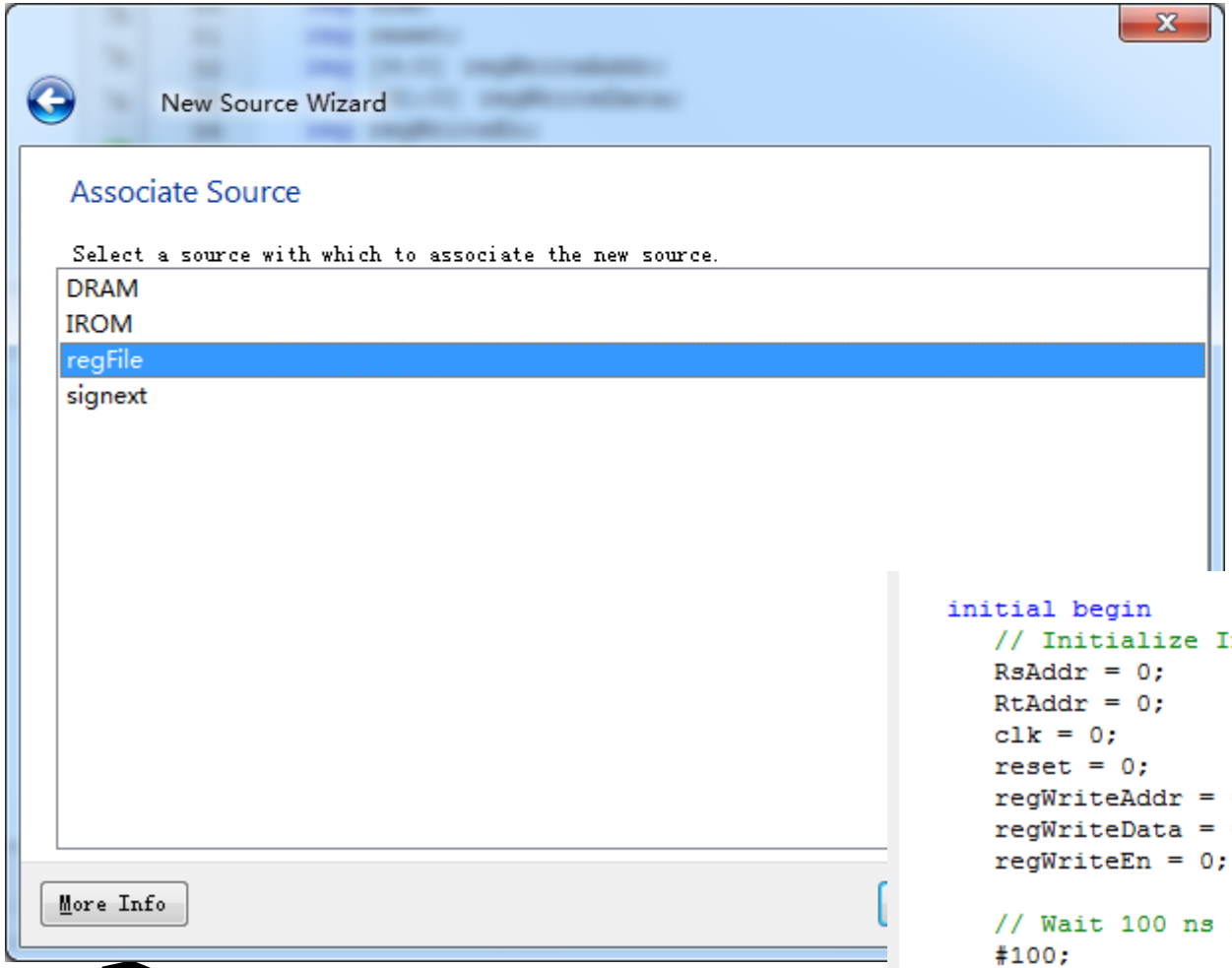
仿真

- 子模块仿真
- 顶层仿真
- 仿真激励文件
  - 模拟产生外部输入信号

新建激励文件


New Source Wizard
Select Source Type
Select source type, file name and its location.
BMM File
ChipScope Definition and Connection File
Implementation Constraints File
IP (CORE Generator & Architecture Wizard)
MEM File
Schematic
User Document
Verilog Module
Verilog Test Fixture
VHDL Module
VHDL Library
VHDL Package
VHDL Test Bench
Embedded Processor
File name:
regsim
Location:
F:\CoursePROJECT\NEXYS4 ISE\LAB1
Add to project
More Info
Next
Cancel

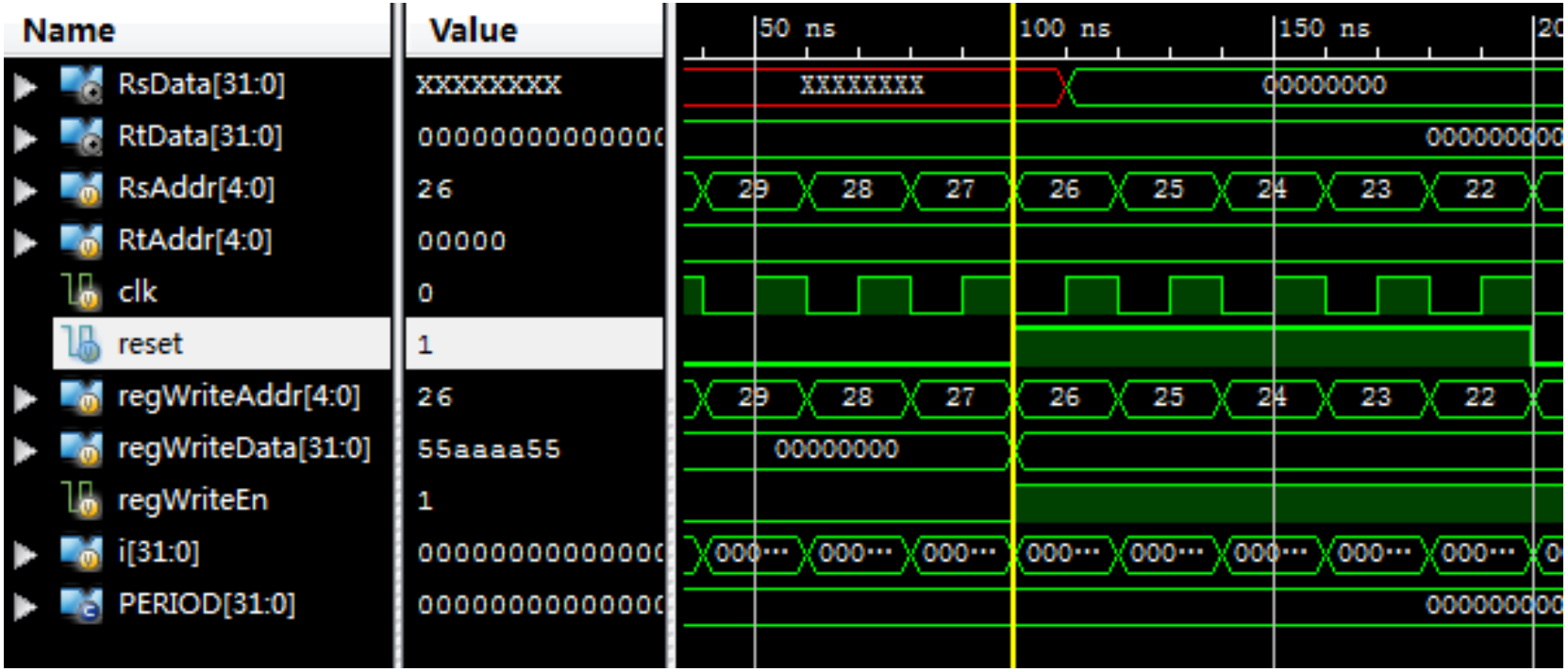
选择相应的模块




```
    initial begin
        // Initialize Inputs
        RsAddr = 0;
        RtAddr = 0;
        clk = 0;
        reset = 0;
        regWriteAddr = 0;
        regWriteData = 0;
        regWriteEn = 0;

        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here

    end

endmodule
```

```
integer i;
   initial begin
      // Initialize Inputs
      RsAddr = 0;
      RtAddr = 0;
      clk = 0;
      reset = 0;
      regWriteAddr = 0;
      regWriteData = 0;
      regWriteEn = 0;

      // Wait 100 ns for global reset to finish
      #100;
      // Add stimulus here
      regWriteData =32'h55aaaa55;
      regWriteEn=1;
      reset=1;
      #100;
      reset=0;
   end
    parameter PERIOD = 20;
   always begin
      clk = 1'b0;
      #(PERIOD/2) clk = 1'b1;
      #(PERIOD/2);
   end
   always begin
   for (i = 31; i >= 1; i=i-1) begin
      regWriteAddr=i;
      RsAddr=i;
       #PERIOD;
   end
   end
endmodule
```

运行仿真，观察输出波形或数据



Memory

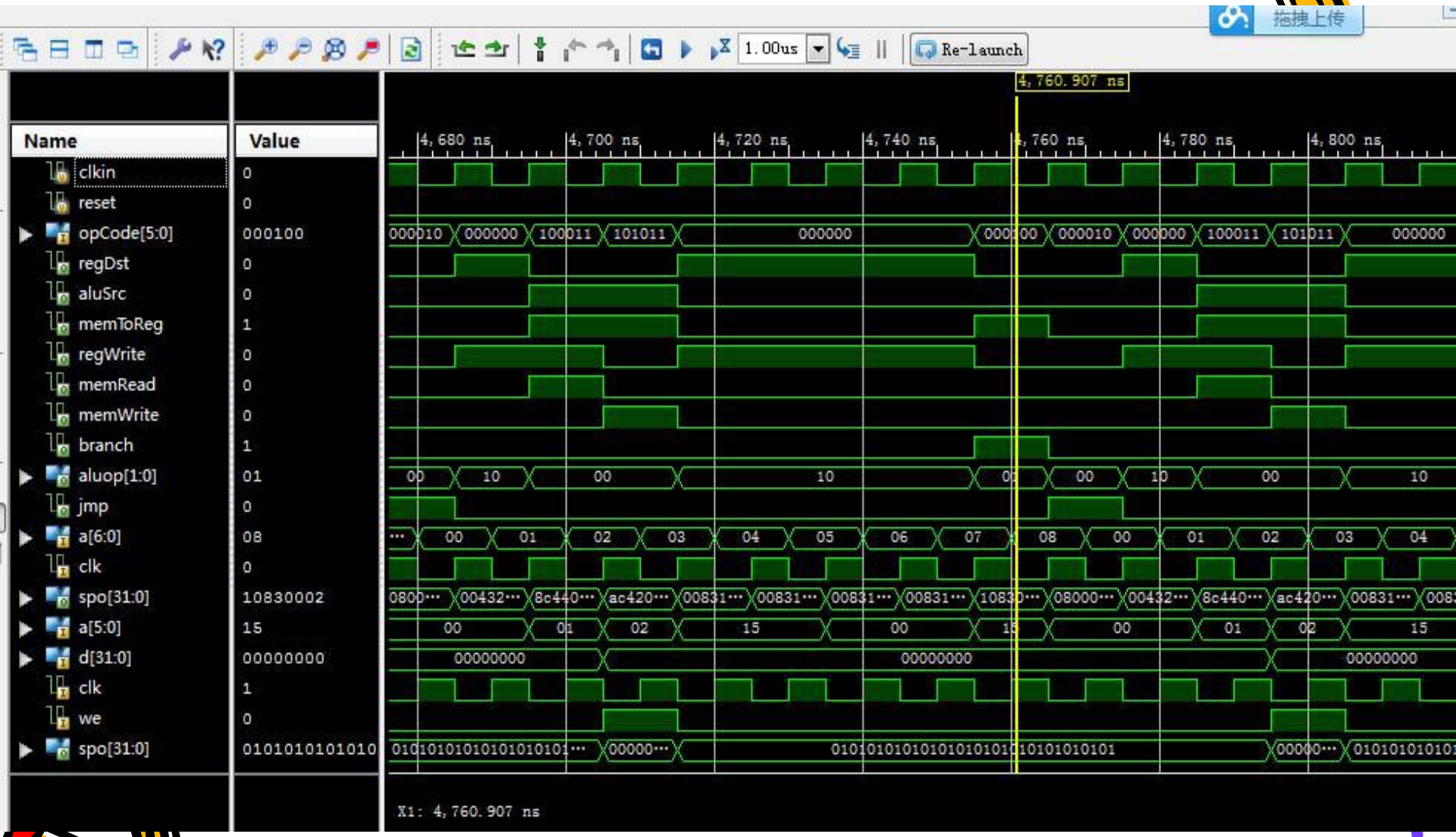
/topsim/uut/dmem/inst/ram_data[63:0,
/topsim/uut/dmem/inst/ram_data_tmp[
/topsim/uut/imem/inst/ram_data[1023:0
/topsim/uut/imem/inst/ram_data_tmp[1
/topsim/uut/regfile/regs[0:31,31:0]

Inst

Address: | Columns: auto | Address Radix: Hexadecimal | Value Radix: Hexadecimal

	0	1
0x0	00000000	00000000
0x2	55555555	00000000
0x4	55555555	00000000
0x6	00000000	00000000
0x8	00000000	00000000
0xA	00000000	00000000
0xC	00000000	00000000
0xE	00000000	00000000
0x10	00000000	00000000
0x12	00000000	00000000
0x14	00000000	00000000
0x16	00000000	00000000
0x18	00000000	00000000
0x1A	00000000	00000000
0x1C	00000000	00000000
0x1E	00000000	00000000

顶层仿真


拖拽上传
1.00us
Re-launch
4,760.907 ns
Name
Value
clkin 0
reset 0
opCode[5:0] 000100
regDst 0
aluSrc 0
memToReg 1
regWrite 0
memRead 0
memWrite 0
branch 1
aluop[1:0] 01
jmp 0
a[6:0] 08
clk 0
spo[31:0] 10830002
a[5:0] 15
d[31:0] 00000000
clk 1
we 0
spo[31:0] 0101010101010
4,680 ns
4,700 ns
4,720 ns
4,740 ns
4,760 ns
4,780 ns
4,800 ns
X1: 4,760.907 ns

添加ANDI指令,实现赋值

- andi指令既不是R型指令，也不是lw，sw，beq指令，因此可以采用11作为其对ALU 2位操作码的编码。

指令	ALUOp1	ALUOp0
R型	1	0
lw	0	0
sw	0	0
beq	0	1
j	X	X

输入信号	操作类型
0000	与
0001	或
0010	加
0110	减
0111	小于设置

```
 6'b001100:
begin
    regDst=0;
  aluSrc=1;
 memToReg=0;
  regWrite=1;
  memRead=0;
  memWrite=0;
  branch=0;
  aluop=2'b11;
  jmp=0;
  end
```

主控制器

```
8'b11xxxxxx: ALUCtr=4'b0000;
```

ALU控制器

实验报告要求

- 实验任务、目标
- 微处理器各个模块硬件设计原理、verilog代码
- Rom汇编程序设计、代码
- 各个模块的仿真激励代码、仿真结果截图以及文字说明如何验证其正确性
- 心得、体会与建议