EDA技术

第11章
16/32 RISC CPU创新设计



《EDA技术实用教程——Verilog HDL版》
（第七版）
配套课件

EDA Lab

处理器典型结构

- 主要部分：
  - 算术逻辑单元ALU
  - 通用寄存器组
  - 程序计数器
  - 数据存储器
  - 指令译码逻辑
  - 指令存储器
  - 控制器
  - 中断控制器（异常处理）

EDA Lab

处理器典型结构

- 处理器中的运算单元一般被称为ALU，即算术逻辑单元
- 主要完成下列功能
  - 加法
  - 逻辑运算
  - 移位

处理器典型结构

- ALU的加
  - 如果数是补码表示的，那么“加”其实可以实现减法
  - 加减法是可以实现乘除法的
  - 各种数制的转换与运算均可实现
  - 如果加的是0，那么功能就是MOVE
  - 如果两个相同数相加，就是乘2，或者算术左移

处理器典型结构

- ALU的逻辑运算
  - 与、或、非、异或
  - 可以实现任何的逻辑运算
  - 异或同样可以实现MOVE功能，只要与0异或
  - 如果异或相同的数，其实完成就是清零
  - 使用“与”可以单独判断某一个位是否为1



EDA Lab

处理器典型结构

- ALU的移位运算
  - 很多处理器结构中，移位是独立于ALU的，使用桶形移位器
  - 分逻辑移位与算术移位
  - 算术移位可以实现2次幂的乘法或除法
- ALU具有基本的算术逻辑运算能力，可以构成任意复杂的计算



EDA Lab

处理器典型结构

- ALU处理数据的字长
  - 数据通路的宽度，决定了处理器的字长，常见的有32位、64位
- ALU一般有两个输入、一个输出
- ALU做双目或者单目运算

HANGZHOU DIANZI UNIVERSITY 1956

处理器典型结构

- 通用寄存器组
  - ALU的输入输出数据一般来自于通用寄存器
  - 一般会构成一个组，有16个或者32个
  - 访问速度最快，但个数有限

EDA Lab

处理器典型结构

- 数据储存器
  - 更大量的数据来自于数据储存器
  - 容量较大
  - 一般先装载如通用寄存器组再送ALU运算，运算结果先放在通用寄存器中，再存储回数据存储器

HANGZHOU DIANZI UNIVERSITY 1956

处理器典型结构



- 指令与指令存储器
  - 通过特定编码的二进制数据，我们称之为指令，它定义了当前使用ALU的功能
  - 多条指令构成一个指令流，就是机器码
  - 存放在指令存储器中



EDA Lab

处理器典型结构



- 程序计数器PC
  - 程序计数器是一个指针，指向下一条将被运行的指令
  - PC或者自动加，或者受分支结果控制，或者受到异常、程序调用控制



EDA Lab

处理器典型结构



- 指令字长
  - 指令字长是指每条指令的位数，可以所有指令都是等长，也可以变长，要看处理器的指令集ISA是如何定义的
  - 32位处理器的指令字长可以是16，典型如ARM Thumb2
  - 不要把指令字长与处理器位数相混淆



EDA Lab

处理器典型结构

- 指令译码器
  - 处理器为了提高程序的代码密度，指令一定是被压缩编码过的，所以需要在执行时进行译码，以便解出诸如ALU的具体控制信号
  - 设计指令集时，会尽量使得译码过程简单，以便提高处理器速度与效率

EDA Lab

处理器典型结构

- 控制器
  - 处理器中关键的时序控制信号来源于控制器
  - 往往是一个状态机
  - 如果处理器采用流水线设计，该控制器往往设计较为复杂，产生复杂控制时序来使得处理器中的各个部件协同工作



EDA Lab

处理器指令集ISA

- 指令集是存储在CPU内部，对CPU运算进行指导和优化的硬程序，用来引导CPU进行加减运算和控制计算机操作系统的一系列指令集合。
- CPU指令集是在CPU设计时固化在里面的“硬程序”，整合在CPU内部的逻辑电路中，是“汇编语言”转换到“机器码”（相当于一个翻译过程）+CPU执行机器码的晶体管和逻辑电路的集合

EDA Lab

处理器指令集ISA分类

CISC(复杂指令集)

- 指令集较丰富，对特殊任务有专用的特殊指令，执行特殊功能，也就是说指令集里面的每个指令较长，每个指令都分成好几个微指令来处理
- 正是因为指令集丰富所以在开发程序时较简单，但在执行时效率较低，处理数据时速度较慢，使得译码器翻译二进制代码时较慢

EDA Lab

处理器指令集ISA分类



- **RISC(精简指令集)**
  - 注重的是指令集的优化
  - RISC的设计者对那些常用的指令进行优化使它们更加简单高效，对于那些不常用的指令会将其组合起来去实现某些特殊的功能
  - 所以RISC指令集位数较短，执行效率高，译码和处理数据较快



EDA Lab

RISC-V处理器指令集ISA



Risc-v基本32位整数指令集RV32I的指令类型与格式

- 共6种指令类型：为R、I、S、B、U、J类型

格式	含义	31~25		24~20	19~15	14~12	11~7		6~0
R-type	Register	funct7		rs2	rs1	funct3	rd		opcode
I-type	Immediate	imm[11:0]			rs1	funct3	rd		opcode
S-type	Store	imm[11:5]		rs2	rs1	funct3	imm[4:0]		opcode
B-type	Branch	imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode
U-type	Upper Immediate	imm[31:12]					rd		opcode
J-type	Jump	imm[20]	imm[10:1]	imm[11]	imm[19:12]		rd		opcode



EDA Lab

简单16/32位RISC CPU的Verilog设计

EDA Lab

KX9016的结构与特色

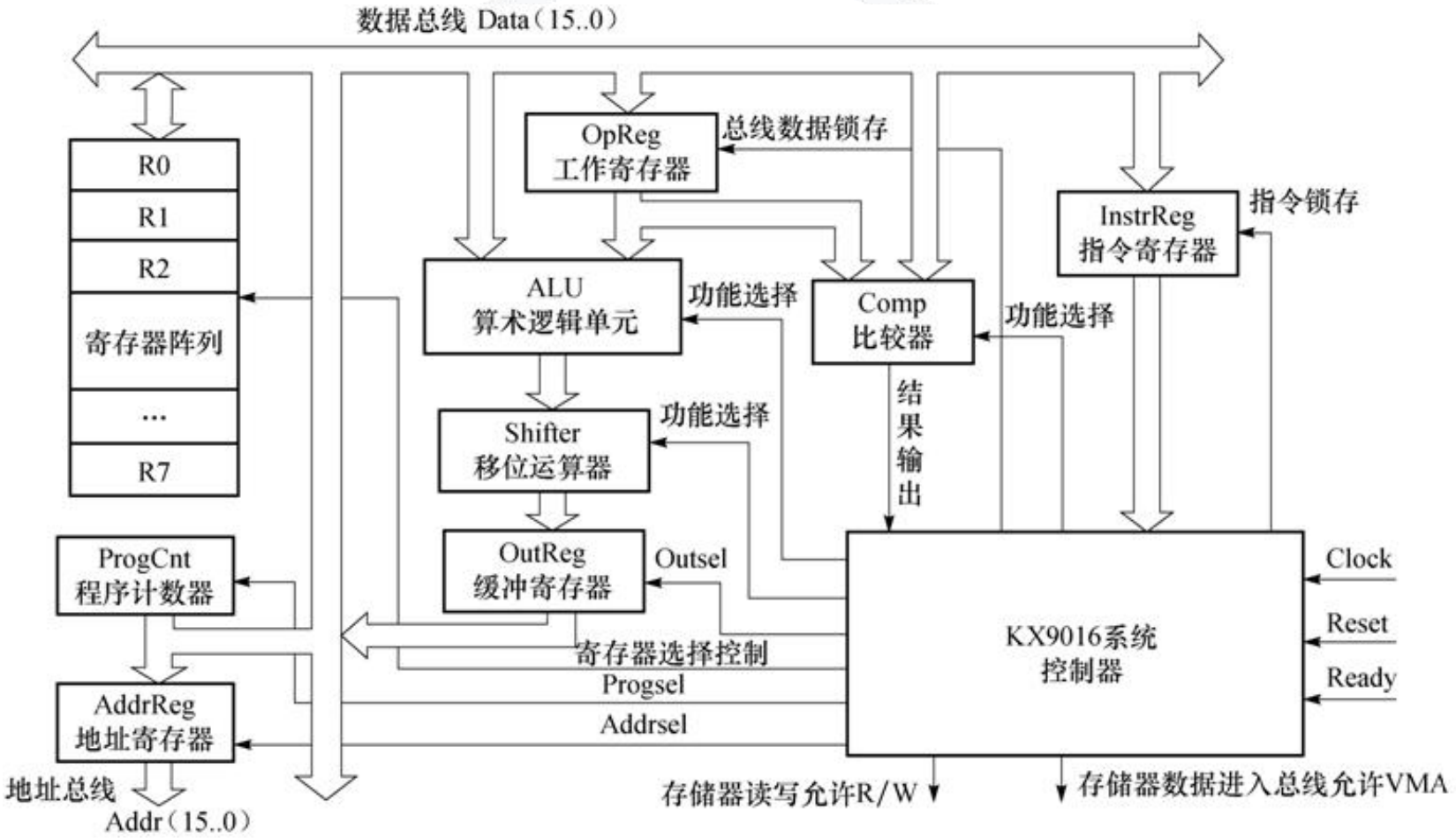
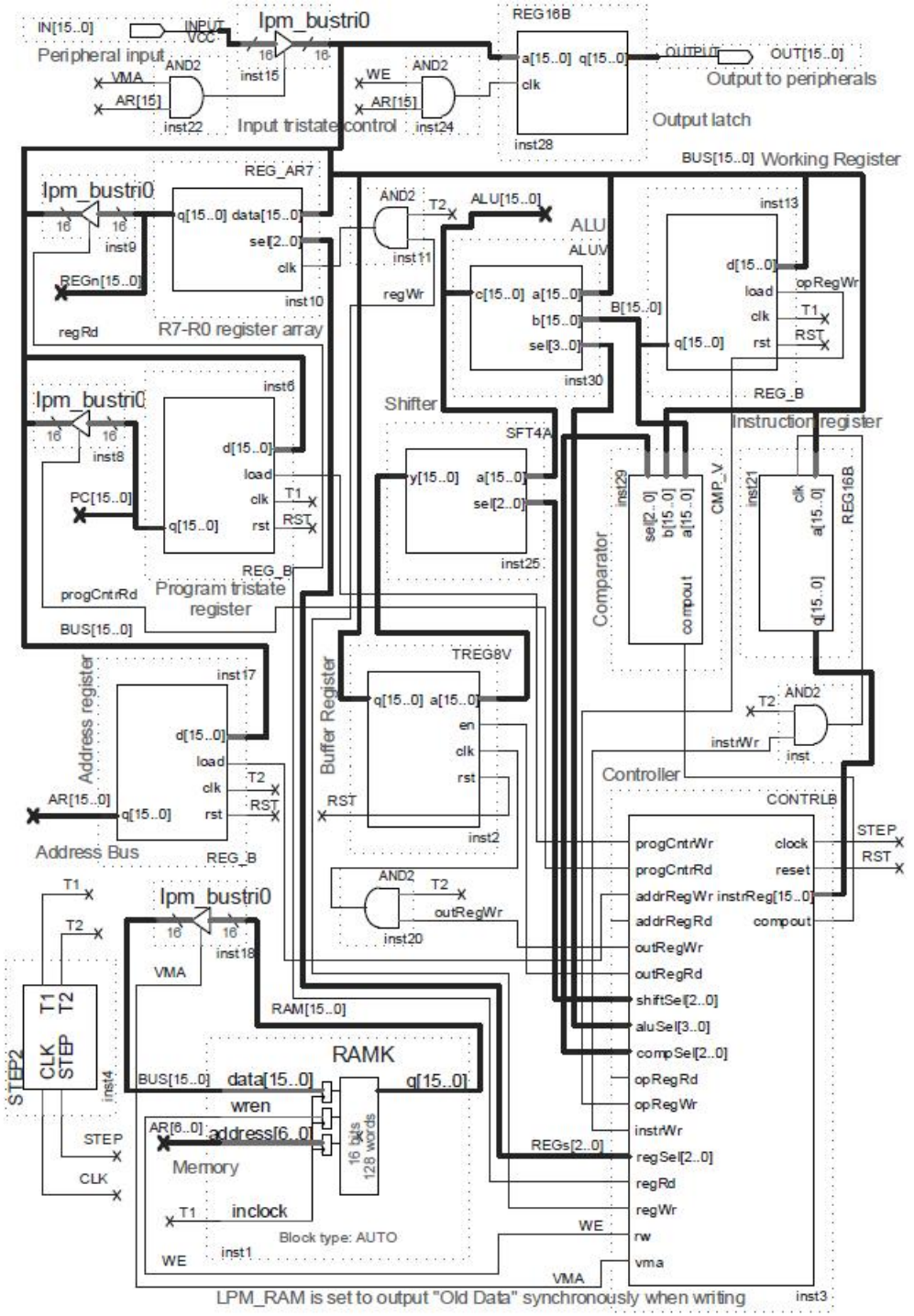
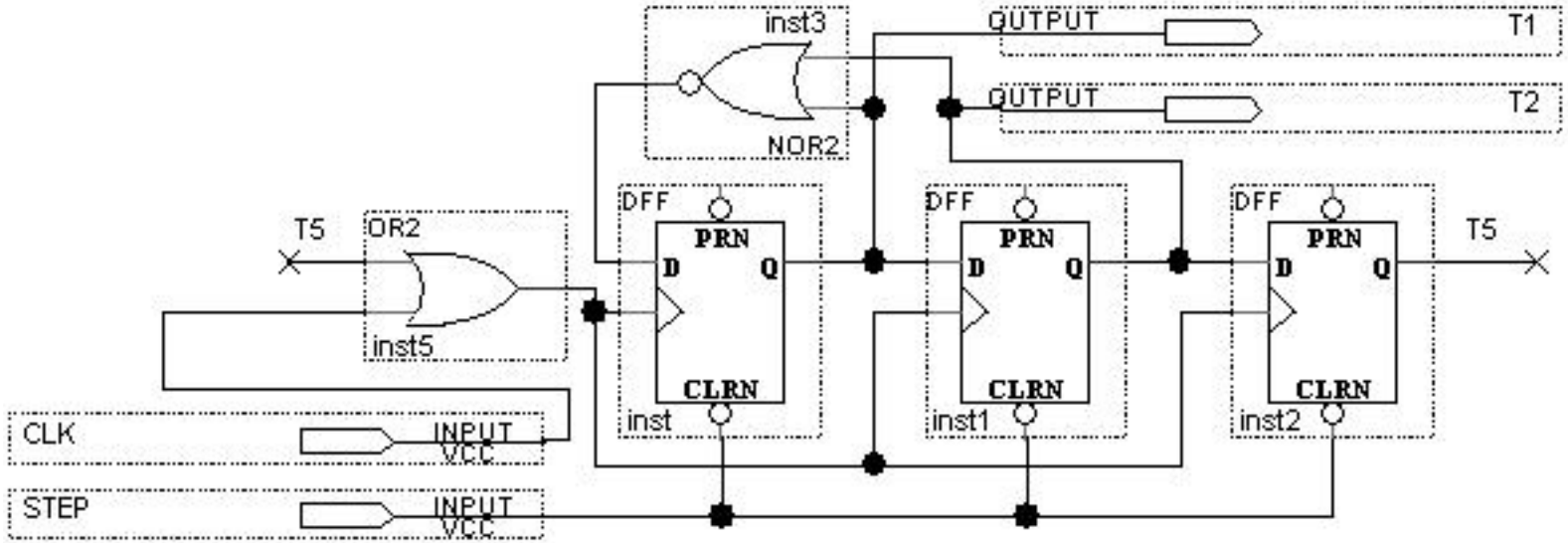


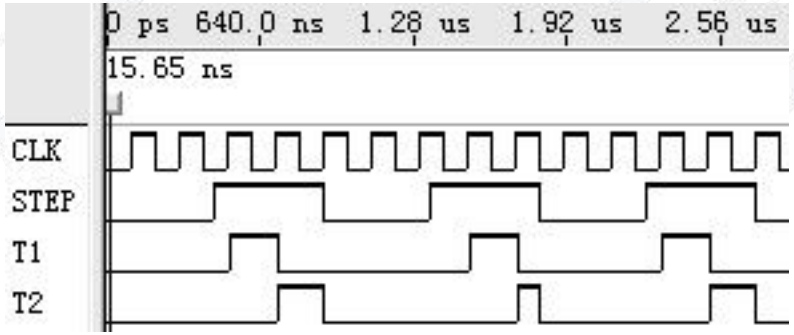
图 11-1　16 位 KX9016v1 CPU 结构框图


IN[15..0]
INPUT
VCC
lpm_bustri0
Peripheral input
VMA
AR[15]
AND2
inst15
inst22
Input tristate control
WE
AR[15]
AND2
inst24
REG16B
a[15..0] q[15..0]
clk
inst28
OUTPUT
OUT[15..0]
Output to peripherals
Output latch
BUS[15..0] Working Register
REG_AR7
lpm_bustri0
q[15..0] data[15..0]
sel[2..0]
clk
inst10
inst9
REGn[15..0]
regRd
R7-R0 register array
AND2
T2
ALU[15..0]
inst11
regWr
ALU
ALUV
c[15..0] a[15..0]
b[15..0]
sel[3..0]
inst30
B[15..0]
inst13
d[15..0]
load
clk
rst
q[15..0]
opRegWr
T1
RST
REG_B
Instruction register
lpm_bustri0
inst6
inst8
PC[15..0]
d[15..0]
load
clk
rst
q[15..0]
T1
RST
REG_B
Program tristate register
progCntrRd
BUS[15..0]
Shifter
SFT4A
y[15..0] a[15..0]
sel[2..0]
inst25
Comparator
inst29
sel[2..0]
b[15..0]
a[15..0]
CMP_V
compout
inst21
clk
a[15..0]
q[15..0]
REG16B
Address register
inst17
d[15..0]
load
clk
rst
q[15..0]
T2
RST
AR[15..0]
Address Bus
REG_B
Buffer Register
TREG8V
q[15..0] a[15..0]
en
clk
rst
RST
inst2
AND2
T2
instrWr
inst
Controller
CONTRLB
progCntrWr
progCntrRd
addrRegWr
addrRegRd
outRegWr
outRegRd
shiftSel[2..0]
aluSel[3..0]
compSel[2..0]
opRegRd
opRegWr
instrWr
regSel[2..0]
regRd
regWr
rw
vma
clock
reset
instReg[15..0]
compout
STEP
RST
AND2
T2
outRegWr
inst20
T1
T2
lpm_bustri0
inst18
VMA
RAM[15..0]
STEP2
CLK
STEP
inst4
RAMK
BUS[15..0]
data[15..0]
q[15..0]
wren
AR[6..0]
address[6..0]
16 bits
128 words
Memory
REGs[2..0]
STEP
CLK
T1
inclock
Block type: AUTO
inst1
WE
WE
VMA
LPM_RAM is set to output "Old Data" synchronously when writing
inst3

KX9016基本硬件系统设计

1 单步节拍发生模块





KX9016基本硬件系统设计

2 ALU模块

【例 11-1】

```
module ALUV (input[15:0] a, b, input[3:0] sel, output reg[15:0] c);
  parameter alupass=0, andOp=1, orOp=2, notOp=3, xorOp=4, plus=5,
  alusub=6, inc=7, dec=8, zero=9;
  always @(a or b or sel)
    case (sel)
      alupass : c <= a;          //总线数据直通 ALU
       andOp : c <= a & b;       //逻辑与操作
        orOp : c <= a | b;       //逻辑或操作
       xorOp : c <= a ^ b;       //逻辑异或操作
       notOp : c <= ~a;          //取反操作
        plus : c <= a + b;       //算术加操作
      alusub : c <= a - b;       //算术减操作
         inc : c <= a + 1;       //加 1 操作
         dec : c <= a - 1;       //减 1 操作
        zero : c <= 0;           //输出清 0
      default : c <= 0;
    endcase
endmodule
```

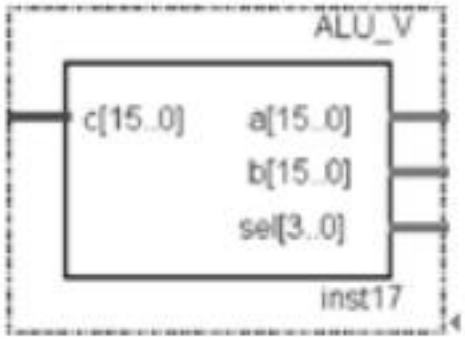


图 11-4 ALU 模块

KX9016基本硬件系统设计

3 比较器模块

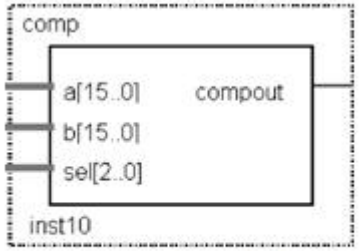


图 11-5 比较器模块符号

【例 11-2】

```
module CMP_V (input[15:0] a, b, input[2:0] sel, output reg compout);
parameter eq=0, neq=1, gt=2, gte=3, lt=4, lte =5;
always @(a or b or sel)
 case (sel)
 eq : if (a==b) compout<=1; else compout<=0; //a 等于 b，输出为 1，负责是 0
 neq : if (a!=b) compout<=1; else  compout<=0; //a 不等于 b，输出为 1
  gt : if (a>b)   compout<=1; else  compout<=0;  //a 大于 b，输出为 1
 gte : if (a>=b)  compout<=1; else  compout<=0; //a 大于等于 b，输出为 1
  lt : if (a<b)   compout<=1; else  compout<=0; //a 小于 b，输出为 1
 lte : if (a<=b)  compout<=1; else  compout<=0; //a 小于等于 b，输出为 1
 default :        compout<=0;
endcase
endmodule
```

4 基本寄存器与寄存器阵列组

1. 基本寄存器

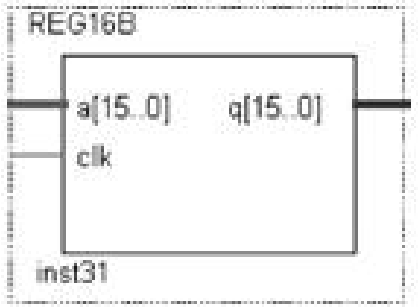


图 11-6 基本寄存器

【例 11-3】

```
module REG16B (input[15:0] a, input clk, output reg[15:0] q);
   always @(posedge clk)     q <= a;   endmodule
```

EDA Lab

4 基本寄存器与寄存器阵列组

1. 基本寄存器

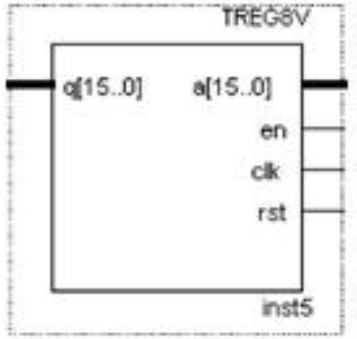


图 11-7　含三态门的寄存器

【例 11-4】

```
module TREG8V (a, en, clk,rst, q);
  input rst,en,clk; input[15:0] a; output[15:0] q; reg[15:0] q,val;
   always @(posedge clk or posedge rst)
     if (rst==1'b1)  val <= {16{1'b0}}; else  val <= a;
   always @(en or val)
     if (en == 1'b1)  q <= val;  else  q <= 16'bzzzzzzzzzzzzzzzz;
endmodule
```

4 基本寄存器与寄存器阵列组

1. 基本寄存器

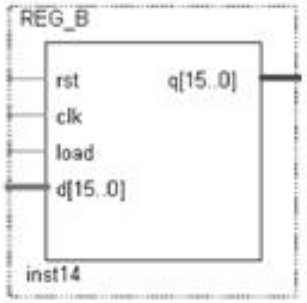


图 11-8 含加载使能的寄存器

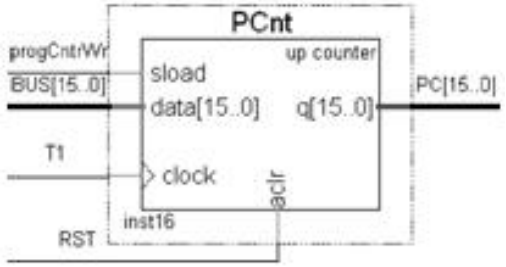


图 11-9 PC 替代电路

【例 11-5】

```
module REG_B (input rst, clk, load, input[15:0]d, output reg[15:0] q);
     always @(posedge clk or posedge rst)
          if (rst==1'b1)  q <= {16{1'b0}};   else
     begin  if (load==1'b1)   q <= d;  end
endmodule
```

4 基本寄存器与寄存器阵列组

2. 寄存器阵列

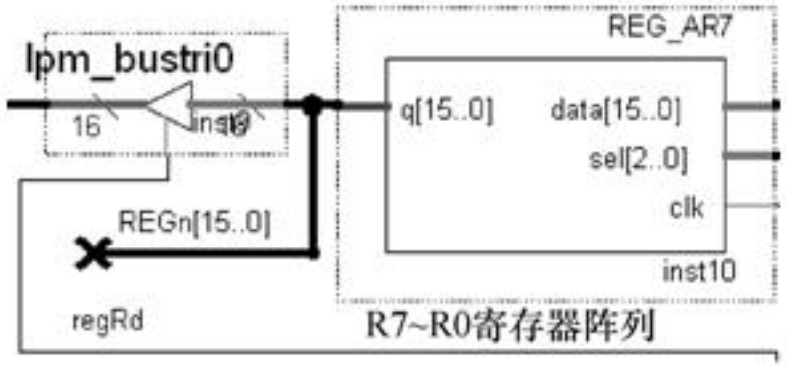


图 11-10 寄存器阵列元件与三态控制门电路

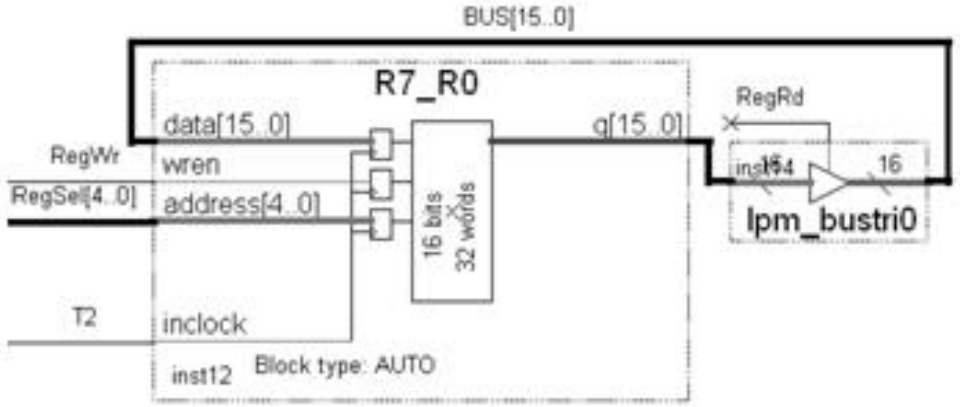


图 11-11 用 LPM_RAM 替代寄存器阵列的电路

【例 11-6】

```
module REG_AR7 (data, sel, clk, q);
   input[15:0] data;  input[2:0] sel; input clk;   output[15:0] q;
   reg[15:0] ramdata[0:7];
   always @(posedge clk)    ramdata[sel] <= data;
   assign q = ramdata[sel] ;
endmodule
```

KX9016基本硬件系统设计

5 移位器模块

【例 11-7】

```
module SFT4A (input[15:0] a,
              input[2:0] sel,
              output reg[15:0] y );
  parameter shftpass=0, sftl=1, sftr=2, rotl=3, rotr=4;
  always @(a or sel)
     case (sel)
     shftpass :  y<=a ;                    //数据直通
        sftl  :  y<={a[14:0], 1'b0};       //左移
        sftr  :  y<={1'b0, a[15:1]};       //右移
        rotl  :  y<={a[14:0], a[15]};      //循环左移
        rotr  :  y<={a[0], a[15:1]};       //循环右移
      default :  y<=0 ;
    endcase
endmodule
```



图 11-12　移位器符号

KX9016基本硬件系统设计

6 程序与数据存储器模块

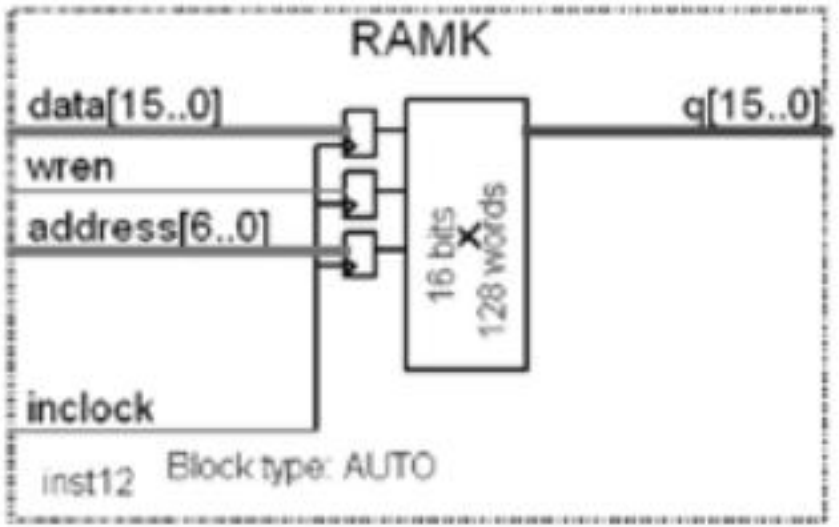


图 11-13　存储器符号

KX9016v1指令系统设计

.1 指令格式

（1）单字指令

操作码						源操作数			目的操作数		
Opcode					.....	SRC			DST		
15	14	13	12	11	.....	5	4	3	2	1	0

ADD R1, R2, R3

将R1 R2的寄存器内容相加后存入寄存器R3

若加法的指令操作码是01101

指令码：01101 00 011 010 001 = 68D1H

KX9016v1指令系统设计

.1 指令格式

（2）双字指令

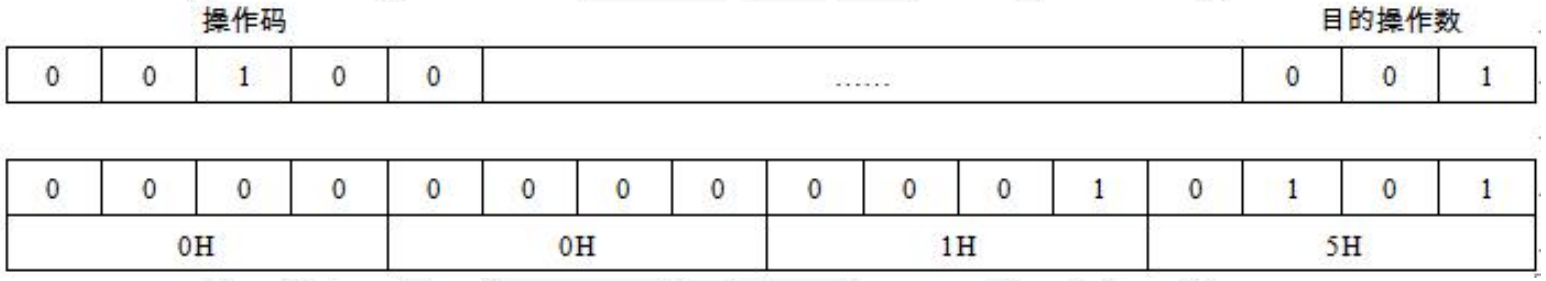


立即装载指令LDR R1, #0015H
将十六进制数0015H装载到寄存器R1中
假设高5位操作码00100，第3位中至少寄存器R1的目的操作数代码001
则指令码：00100 000 0000 0001 0000 000 0001 0101 = 2001H 0015H

KX9016v1指令系统设计

.1 指令格式

（2）双字指令



KX9016v1指令系统设计

.2 指令操作码

表 11-1 KX9016 预设指令及其功能表

操作码	指令	功能	操作码	指令	功能
00000	NOP	空操作	01111	IN	外设数据输入指令
00001	LD	装载数据到寄存器	10000	JMPLTI	小于时转移到立即数地址
00010	STA	将寄存器的数存入存储器	10001	JMPGT	大于时转移
00011	MOV	在寄存器间传送操作数	10010	OUT	数据输出指令
00100	LDR	将立即数装入寄存器	10011	MTAD	16 位乘法累加
00101	JMPI	转移到由立即数指定的地址	10100	MULT	16 位乘法
00110	JMPGTI	大于转移至立即数地址	10101	JMP	无条件转移
00111	INC	加 1 后放回寄存器	10110	JMPEQ	等于时转移
01000	DEC	减 1 后放回寄存器	10111	JMPEQI	等于时转移到立即地址
01001	AND	两个寄存器间与操作	11000	DIV	32 位除法
01010	OR	两个寄存器间或操作	11001	JMPLTE	小于等于时转移
01011	XOR	两个寄存器间异或操作	11010	SHL	左逻辑移位
01100	NOT	寄存器求反	11011	SHR	右逻辑移位
01101	ADD	两个寄存器加运算	11100	ROTR	循环右移
01110	SUB	两个寄存器减运算	11101	ROTL	循环左移

EDA Lab

KX9016v1指令系统设计

.2 指令操作码

表 11-2 示例程序

指令	机器码	字长	操作码	闲置码	源操作数	目的操作数	功能说明
LDR R1，0025H	2001H 0025H	2	00100	xxxxx	xxx	001	立即数 0025H 送 R1
			0000 0000 0010 0101				
LDR R2，0047H	2002H 0047H	2	00100	xxxxx	xxx	010	立即数 0047H 送 R2
			0000 0000 0100 0111				
LDR R6，0036H	2006H 0036H	2	00100	xxxxx	xxx	110	立即数 0036H 送 R6
			0000 0000 0011 0110				
LD R3，[R1]	080BH	1	00001	xxxxx	001	011	从 R1 指定的 RAM 存储单元取数送 R3
STA [R2]，R3	101AH	1	00010	xxxxx	011	010	将 R3 的内容存入 R2 指定 RAM 单元
JMPGTI R1，R6，[0000]	300EH 0000H	2	00110	xxxxx	001	110	若 R1>R6，则转向地址[0000H]
			0000 0000 0000 0000				
INC R1	3801H	1	00111	xxxxx	xxx	010	R1+1→R1
INC R2	3802H	1	00111	xxxxx	xxx	010	R2+1→R2
JMPI [0006]	2800H 0006H	2	00101	xxxxx	xxx	xxx	绝对地址转移指令：转向地址 0006H
			0000 0000 0000 0110				

EDA Lab

KX9016v1指令系统设计

.3 软件程序设计实例

表 11-3 七条指令的汇编程序示例

地址	机器码	指令	功能说明
0000H 0001H	2001H 0032H	LDR R1, 0032H	将立即数 0032H 送寄存器 R1
0002H 0003H	2002H 0011H	LDR R2, 0011H	将立即数 0011H 送寄存器 R2
0004H	680AH	ADD R1 , R2 , R3	将寄存器 R1 和 R2 的内容相加后送 R3
0005H	1819H	MOV R1 , R3	将寄存器 R3 的内容送入 R1
0006H	3802H	INC R2	R2 + 1→R2
0007H	101AH	STA [R2] , R3	将 R3 的内容存入 R2 指定地址的 RAM 单元
0008H	080BH	LD R3 , [R1]	将 R1 指定地址的 RAM 单元的数据送 R3
0009H	0000H	NOP	空操作

EDA Lab

.3 软件程序设计实例

表 11-4 存储器初始化文件 RAM_16.mif 的内容

WIDTH = 16;	03 : 0011;	0B : 0000;	13 : 0000;
DEPTH = 256;	04 : 680A;	0C : 0000;	.....
ADDRESS_RADIX = HEX;	05 : 1819;	0D : 0000;	41 : 0000;
DATA_RADIX = HEX;	06 : 3802;	0E : 0000;	42 : 0000;
CONTENT BEGIN	07 : 101A;	0F : 0000;	43 : A6C7;
00 : 2001;	08 : 080B;	10 : 0000;	.....
01 : 0032;	09 : 0000;	11 : 0000;	4F : 0000;
02 : 2002;	0A : 0000;	12 : 1524	END;

EDA Lab

KX9016v1指令系统设计

.4 KX9016v1控制器设计

【例 11-8】

```
module CONTRLA (clock, reset, instrReg, compout, progCntrWr, progCntrRd,
 addrRegWr, addrRegRd, outRegWr, outRegRd, shiftSel, aluSel, compSel,
 opRegRd, opRegWr, instrWr, regSel, regRd, regWr, rw, vma);
  input clock;   input reset;    //时钟和复位信号
  input[15:0] instrReg; input compout;//指令寄存器操作码输入及比较器结果输入
  output progCntrWr;              //程序寄存器同步加载允许，但需 T1 的上升沿有效
  output progCntrRd;              //程序寄存器数据输出至总线三态开关允许控制
  output addrRegWr;               //地址寄存器允许总线数据锁入，但需 T2 有效
  output addrRegRd;               //地址寄存器读入总线允许
  output outRegWr;                //输出寄存器允许总线数据写入，但需 T2 有效
  output outRegRd;                //输出寄存器数据进入总线允许，即打开三态门
  output[2:0] shiftSel; output[3:0] aluSel; //移位器功能选择和 ALU 功能选择
  output[2:0] compSel;  output opRegRd; //比较器功能选择和工作寄存器读出允许
  output opRegWr;                 //总线数据允许锁入工作寄存器，但需 T1 有效
  output instrWr;                 //总线数据允许锁入指令寄存器，但需 T2 有效
  output[2:0] regSel;             //寄存器阵列选择
  output regRd;                   //寄存器阵列数据输出至总线三态开关允许控制
  output regWr;                   //总线上数据允许写入寄存器阵列，但需 T2 有效
```

EDA Lab

KX9016v1指令系统设计



.4 KX9016v1控制器设计

```
output rw;                          //rw=1，RAM写允许；rw=0，RAM读允许
output vma;                         //存储器RAM数据输出至总线三态开关允许控制
reg progCntrWr, progCntrRd, addrRegWr, addrRegRd, outRegWr, outRegRd,
    opRegRd, opRegWr, instrWr, regRd, vma, regWr, rw;
reg[2:0] shiftSel, regSel;  wire[2:0] compSel;   reg[3:0] aluSel;
parameter shftpass=0,alupass=0, zero=9, inc=7, plus=5; //参见程序例11-1
parameter reset1=0, reset2=1, reset3=2, execute=3, nop=4, load=5,
store=6,load2=7,load3=8,load4=9,store2=10,store3=11,store4=12,
incPc=13,incPc2=14,incPc3=15,loadI2=19,loadI3=20,loadI4=21,
loadI5=22,loadI6=23,inc2=24,inc3=25,inc4=26,
move1=27,move2=28,add2=29,add3=30,add4=31;
   //在状态机中增加三个作加法微操作的状态变量元素add2, add3, add4
reg[4:0] current_state, next_state;   //定义现态和次态状态变量
always @(current_state or instrReg or compout)  begin : COM //组合过程
progCntrWr<=0; progCntrRd<=0; addrRegWr<=0; addrRegRd<=0; outRegWr<=0;
  outRegRd<=0; shiftSel<=shftpass; aluSel<=alupass; opRegRd<=0;
  opRegWr<=0;instrWr<=0; regSel<=0; regRd<=0; regWr<=0; rw<=0; vma<=0;
    case (current_state)
    reset1 :  begin  aluSel<=zero; shiftSel<=shftpass;
                outRegWr<=1'b1; next_state<=reset2;    end
    reset2 :  begin  outRegRd<=1'b1; progCntrWr<=1'b1;
```

EDA Lab

KX9016v1指令系统设计

.4 KX9016v1控制器设计

```
                addrRegWr<=1'b1; next_state<=reset3;  end
 reset3 :  begin  vma<=1'b1; rw<=1'b0; instrWr<=1'b1;
                next_state<=execute;                  end
execute : begin
  case (instrReg[15:11])                  //不同指令识别分支处理
    5'b00000 : next_state <= incPc;     // 转 nop 指令处理
    5'b00001 : next_state <= load2;     // 转 load 指令处理
    5'b00010 : next_state <= store2;    // 转 store 指令处理
    5'b00100:begin progCntrRd<=1'b1;aluSel<=inc;shiftSel<=shftpass;
                next_state<=loadI2; end   //转 loadI 指令处理
    5'b00111 : next_state <= inc2;        //转 inc 指令处理
    5'b01101 : next_state <= add2;        //增加一个加法 ADD 指令分支
    5'b00011 : next_state <= move1;       // 转 move 指令处理
     default : next_state <= incPc;       //转 PC 加 1
   endcase          end
 load2 :  begin  regSel<=instrReg[5:3]; regRd<=1'b1;
                 addrRegWr<=1'b1; next_state<=load3; end
 load3 :  begin  vma<=1'b1; rw<=1'b0; regSel<=instrReg[2:0];
                regWr<=1'b1; next_state<=incPc; end
add2 :  begin  regSel <= instrReg[5:3];    //选择寄存器阵列的 R1
```

.4 KX9016v1控制器设计

```
                regRd <= 1'b1;                    //允许 R1 寄存器数据进入总线
          next_state<=add3; opRegWr<=1'b1;end  //将此数据锁入工作寄存器
                                                  //以上 4 步在一个 STEP 脉冲完成，以下同样
add3 :  begin  regSel <= instrReg[2:0]; //选择寄存器阵列的 R2
          regRd<=1'b1; aluSel<=plus;//允许 R2 寄存器数据进入总线，同时
                                                  //选择 ALU 作加法
       shiftSel<=shftpass; outRegWr<=1'b1;//使 ALU 输出直通移位器，同时将数据
                                                  //锁入输出寄存器
       next_state<=add4; end  //此时相加结果尚未进入总线。此 5 步在一个 STEP
                                                  //脉冲完成
add4 :   begin  regSel <= 3'b011;//固定选择寄存器阵列的 R3
       outRegRd<=1'b1; regWr<=1'b1;//允许输出寄存器的数据进入总线，将此数据
                                                  //锁入工作寄存器 R3
       next_state <= incPc ; end //加法操作结束，最后转入做 PC 加 1 操作的状态
 move1 : begin  regSel<=instrReg[5:3]; regRd<=1'b1; aluSel<=alupass;
            shiftSel<=shftpass; outRegWr<=1'b1; next_state<=move2; end
 move2 : begin  regSel<=instrReg[2:0]; outRegRd<=1'b1;
                  regWr<=1'b1; next_state<=incPc;  end
```

KX9016v1指令系统设计



.4 KX9016v1控制器设计

```
store2 : begin  regSel<=instrReg[2:0]; regRd<=1'b1;
                addrRegWr<=1'b1;  next_state<=store3; end
store3 : begin  regSel<=instrReg[5:3]; regRd<=1'b1;
              rw<=1'b1;  next_state<=incPc;  end
loadI2 : begin  progCntrRd<=1'b1;  aluSel<=inc ;  shiftSel<=shftpass;
              outRegWr <= 1'b1;  next_state <= loadI3;   end
loadI3 : begin  outRegRd<=1'b1;  next_state<=loadI4;       end
loadI4 : begin  outRegRd <= 1'b1;  progCntrWr <= 1'b1;
              addrRegWr <= 1'b1;   next_state <= loadI5; end
loadI5 : begin  vma<=1'b1; rw<=1'b0;  next_state<=loadI6;  end
loadI6 : begin  vma <= 1'b1;  rw <= 1'b0 ;  regSel <= instrReg[2:0];
              regWr <= 1'b1;  next_state <= incPc;       end
 inc2 : begin  regSel <= instrReg[2:0] ; regRd <= 1'b1; aluSel <= inc;
            shiftSel<=shftpass; outRegWr<=1'b1; next_state<=inc3; end
 inc3 : begin  outRegRd <= 1'b1; next_state <= inc4;  end
 inc4 : begin  outRegRd<=1'b1; regSel<=instrReg[2:0];
             regWr<=1'b1; next_state<=incPc; end
```

EDA Lab

.4 KX9016v1控制器设计

```
 incPc : begin progCntrRd<=1'b1; aluSel<=inc; shiftSel<=shftpass;
                outRegWr <= 1'b1; next_state <= incPc2; end
 incPc2 : begin outRegRd <= 1'b1; progCntrWr <= 1'b1;
                addrRegWr <= 1'b1;  next_state <= incPc3; end
 incPc3 : begin outRegRd <= 1'b0; vma <= 1'b1; rw <= 1'b0;
               instrWr <= 1'b1; next_state <= execute;  end
 default :     next_state <= incPc;
 endcase  end
always @(posedge clock or posedge reset)      //时序过程
   if (reset==1) current_state<=reset1;
        else current_state<=next_state;
endmodule
```

KX9016v1指令系统设计



.4　KX9016v1控制器设计

1、程序结构

（1）CPU复位模块。

（2）指令辨认分支模块。

（3）指令处理状态序列。

2、指令的语句结构

EDA Lab

KX9016v1指令系统设计



.5 指令设计实例详解

（1）确定功能。

（2）确定指令的操作码。

（3）设定相关常数。

（4）增加状态元素。

（5）加入指令操作码译码语句。

（6）加入完成实际指令功能的状态转换语句。

（7）处理PC。

EDA Lab

KX9016的时序仿真与硬件测试

.1 时序仿真与指令执行波形分析

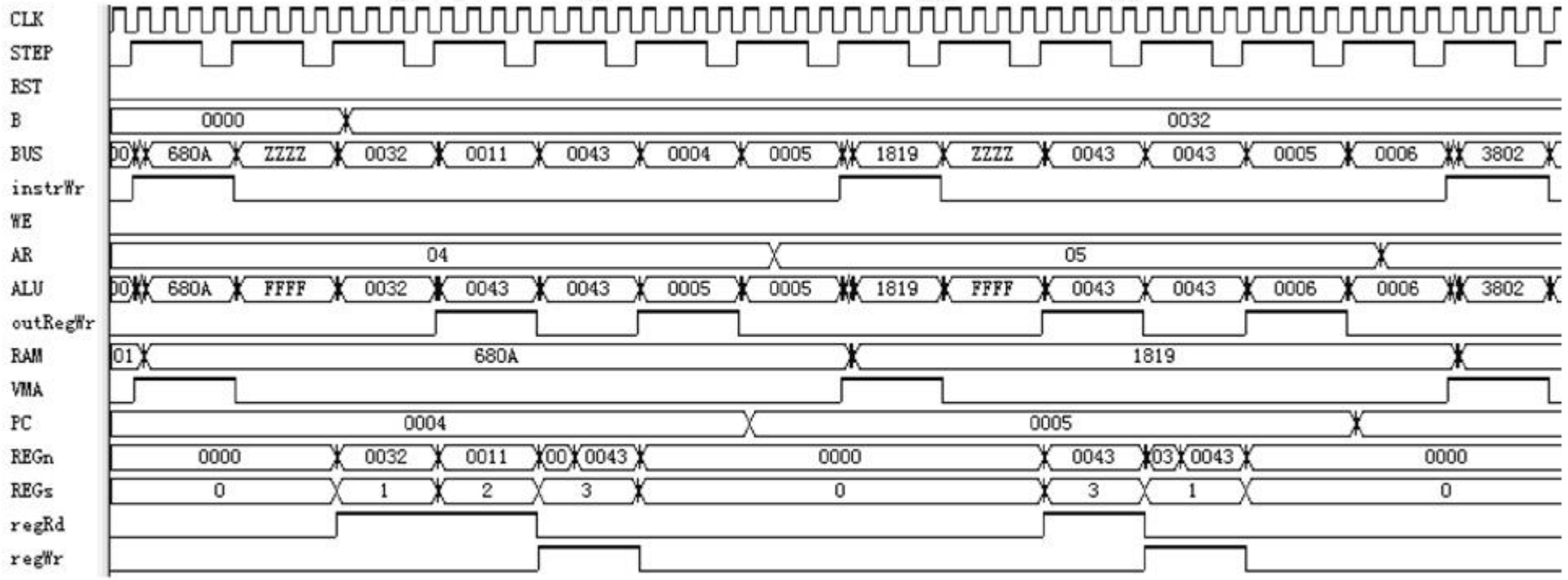


图 11-14　KX9016 的仿真波形，含 ADD 指令和 MOV 指令的时序

KX9016的时序仿真与硬件测试

.1 时序仿真与指令执行波形分析

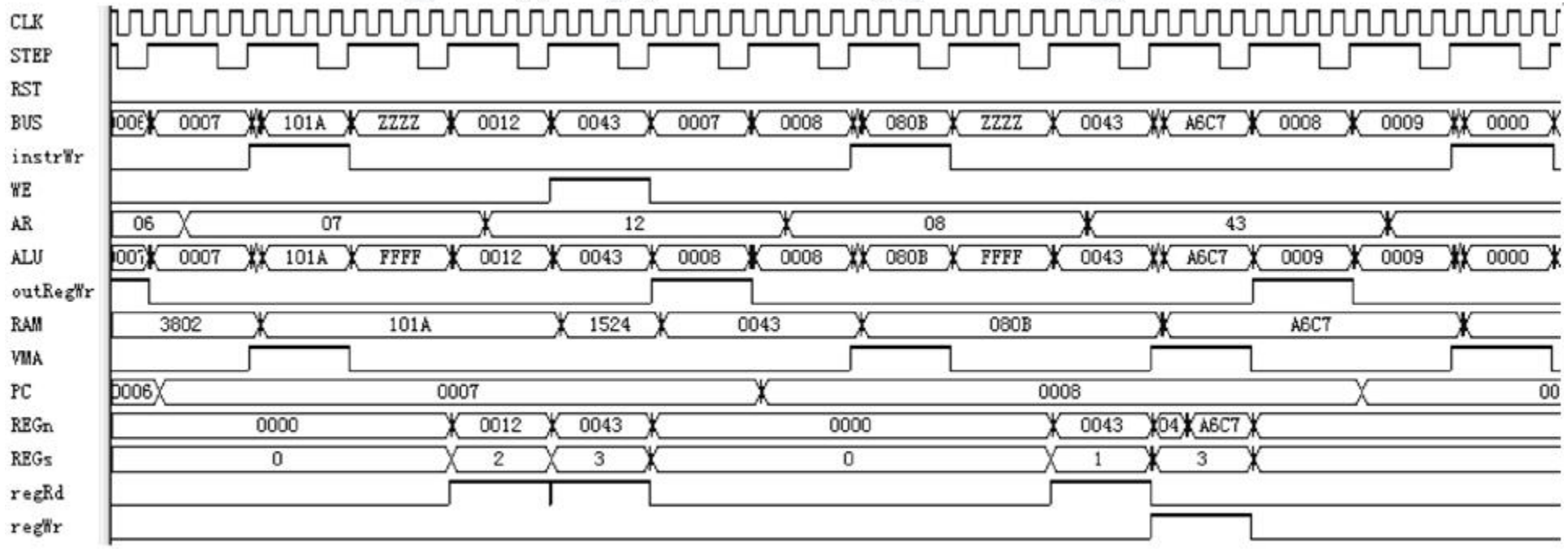


图 11-15　KX9016 的仿真波形，含 STA 指令和 LD 指令的时序

KX9016的时序仿真与硬件测试

.2 CPU工作情况的硬件测试

1. SignalTap II测试与分析

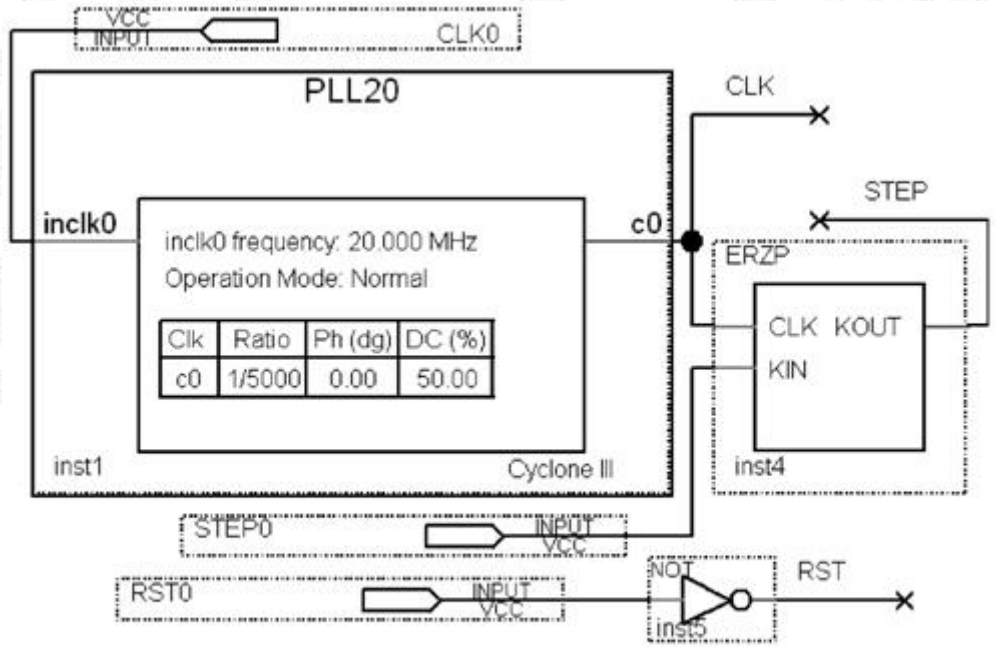


图 11-16　加入锁相环的电路方案

EDA Lab

KX9016的时序仿真与硬件测试

.2 CPU工作情况的硬件测试

1. SignalTap II测试与分析

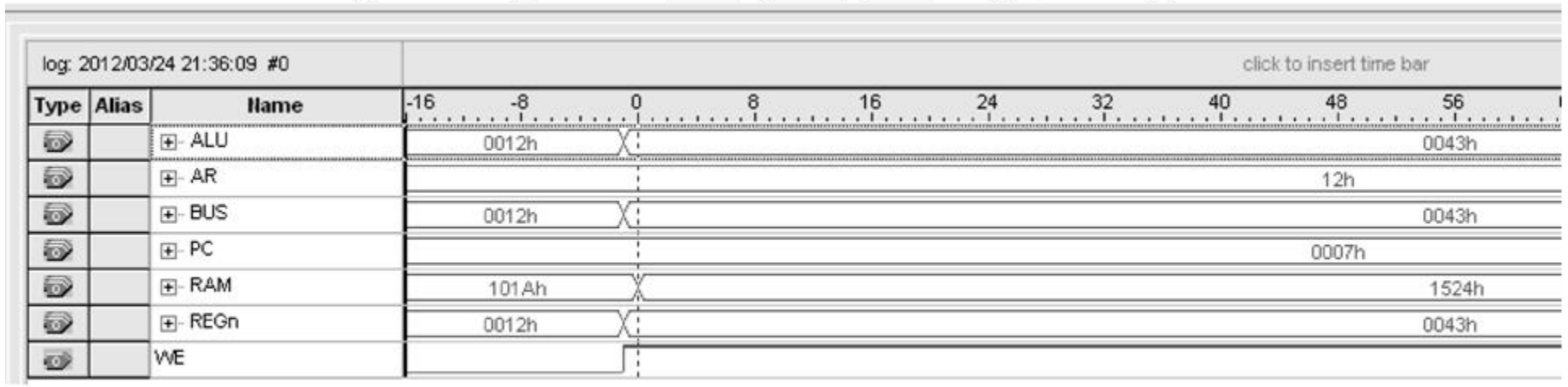


图 11-17　嵌入式锁相环对 KX9016 执行从 RAM 写数据指令的实时测试波形

KX9016的时序仿真与硬件测试

.2 CPU工作情况的硬件测试

2．利用In-System Memory Content Editor进行实时测试

0 RAM8:

000000	20	01	00	32	20	02	00	11	68	0A	18	19	38	02	10	1A	08	0B	00	00	00	00	00	00	00	00	00	00	00	00	00	00
000010	00	00	00	00	00	43	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	B3	B4	B5	B6	B7	B8	D1	D2	D3	D4
000020	D5	D6	D7	D8	D9	DA	E1	E2	E3	E4	E5	E6	20	08	20	09	20	0A	20	0B	20	0C	20	0D	20	0E	20	0F	20	10	20	11
000030	D5	D6	D7	D8	D1	23	E1	E2	E3	E4	E5	E6	20	08	20	09	20	0A	20	0B	20	0C	00	00	00	00	00	00	00	00	00	00
000040	00	00	00	00	00	00	A6	C7	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

图 11-18　In-System Memory Content Editor 对 KX9016 内 RAM 数据变化情况的实测情况

EDA Lab

KX9016的时序仿真与硬件测试

.2 CPU工作情况的硬件测试

3．利用In-System Sources & Probes进行实时测试

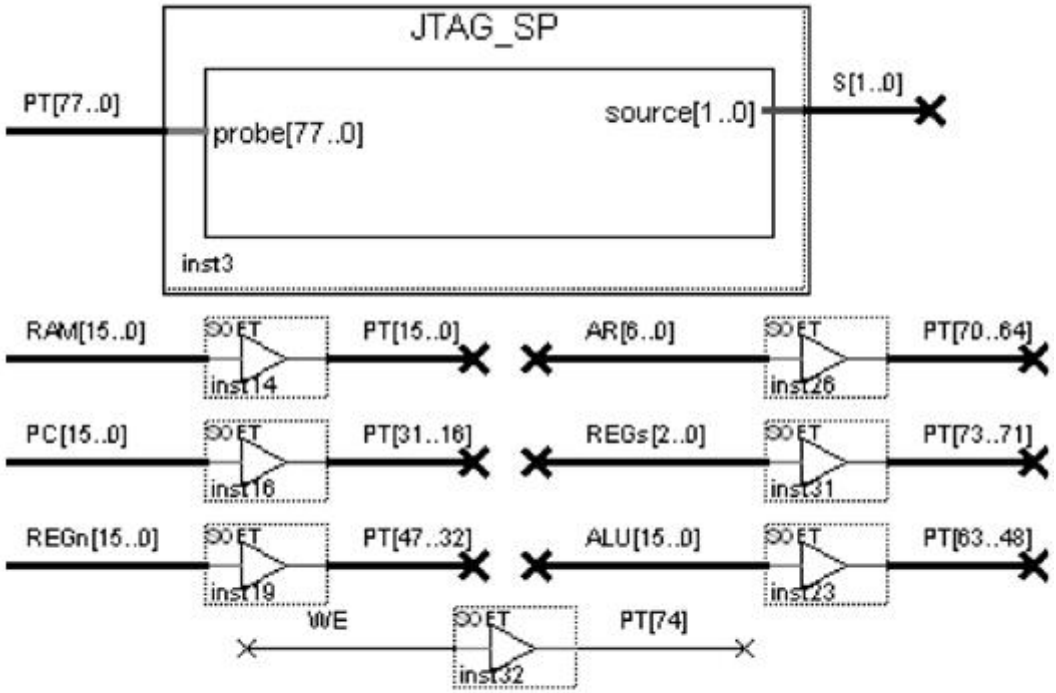


图 11-19　S/P 模块对 KX9016 的端口连接情况

KX9016的时序仿真与硬件测试

.2 CPU工作情况的硬件测试

3．利用In-System Sources & Probes进行实时测试

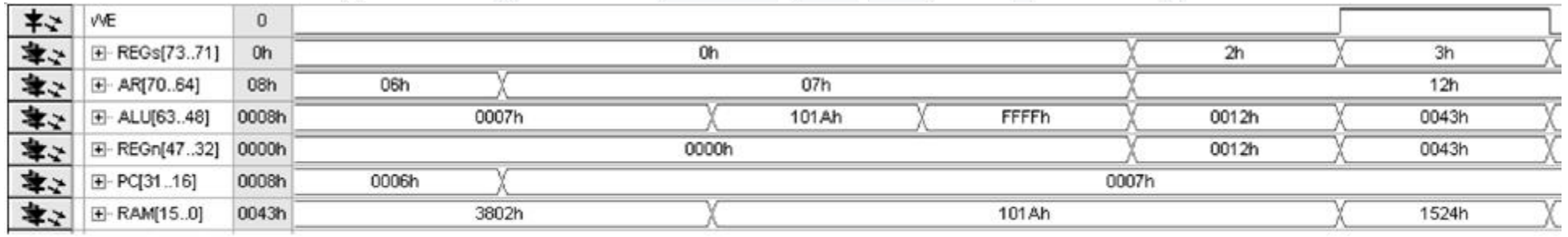


图 11-20　在系统 S/P 模块对 KX9016 执行从 RAM 写数据指令 STA 的实时测试波形

.1　乘法算法及其硬件实现

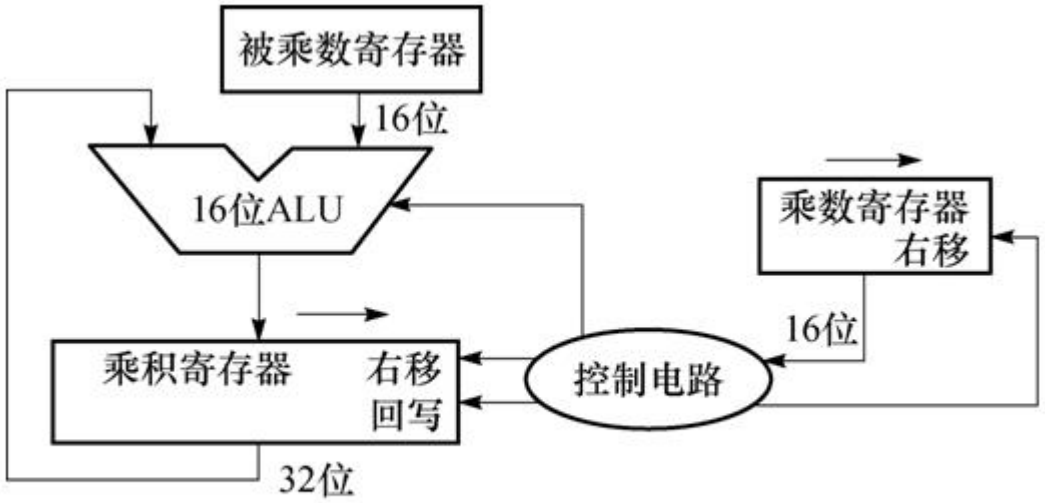


图 11-21　乘法算法 1 的硬件实现

.1 乘法算法及其硬件实现

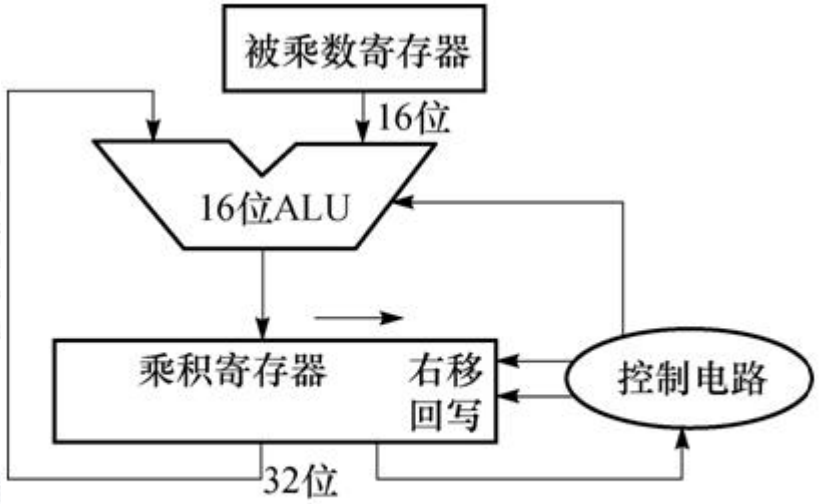


图 11-22 改进后的乘法算法 2 的硬件实现

EDA Lab

.1 乘法算法及其硬件实现

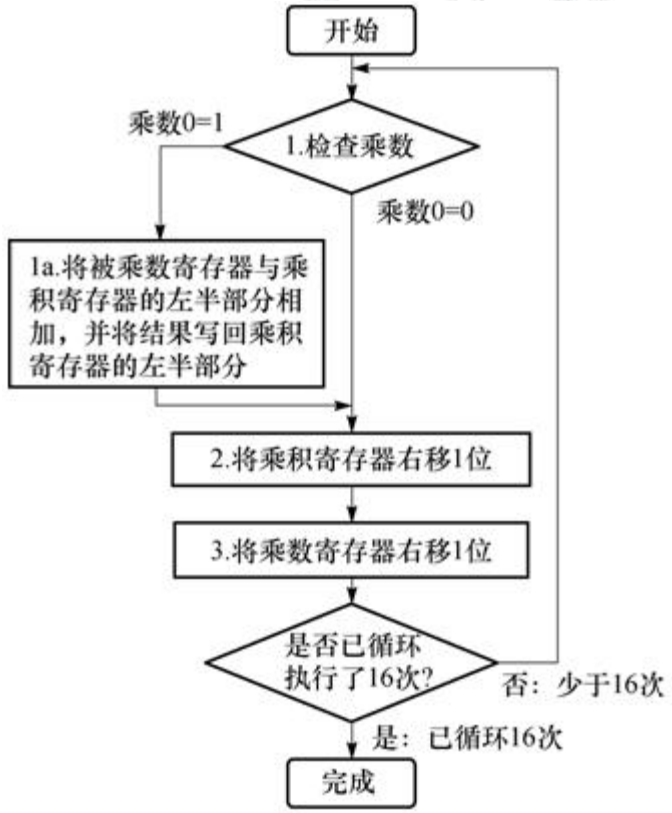


图 11-23 乘法算法 1 的流程图

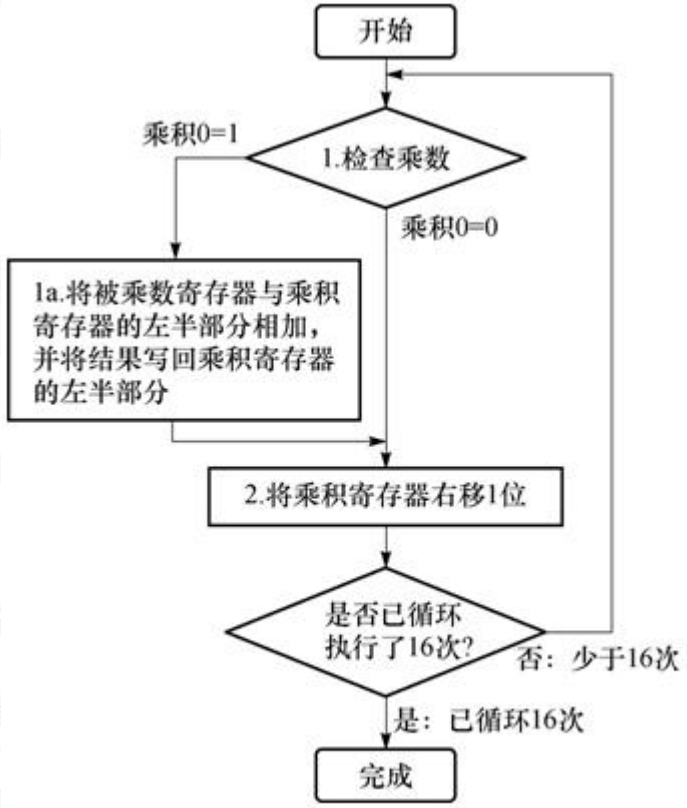


图 11-24 乘法算法 2 的流程图

.1 除法算法及其硬件实现

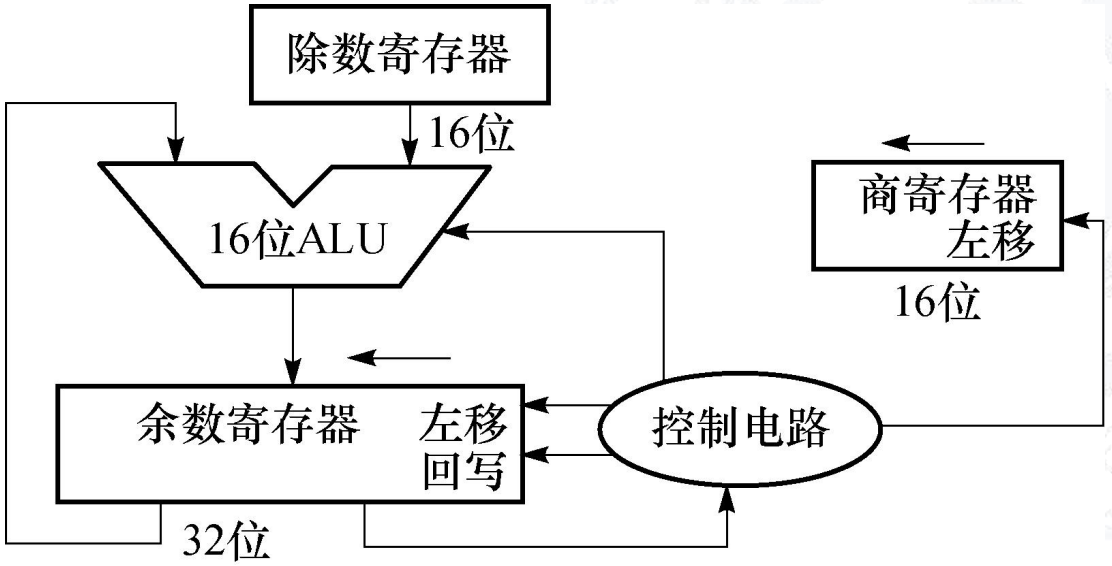


图11-25 除法算法1的硬件结构

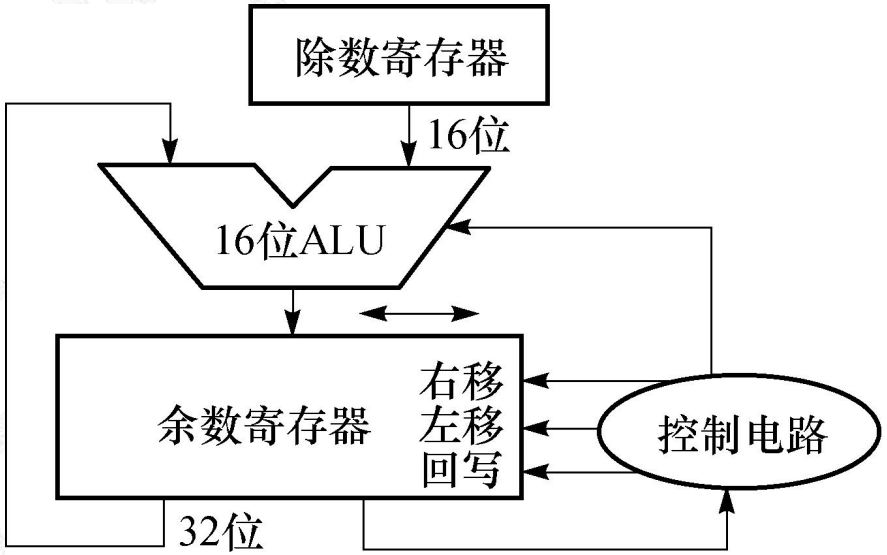


图11-26 除法算法2的硬件结构

RISC-V



- 32位第5代RISC CPU架构
- RV32I
- RV32IM
- RV32IMC



EDA Lab

常见ISA

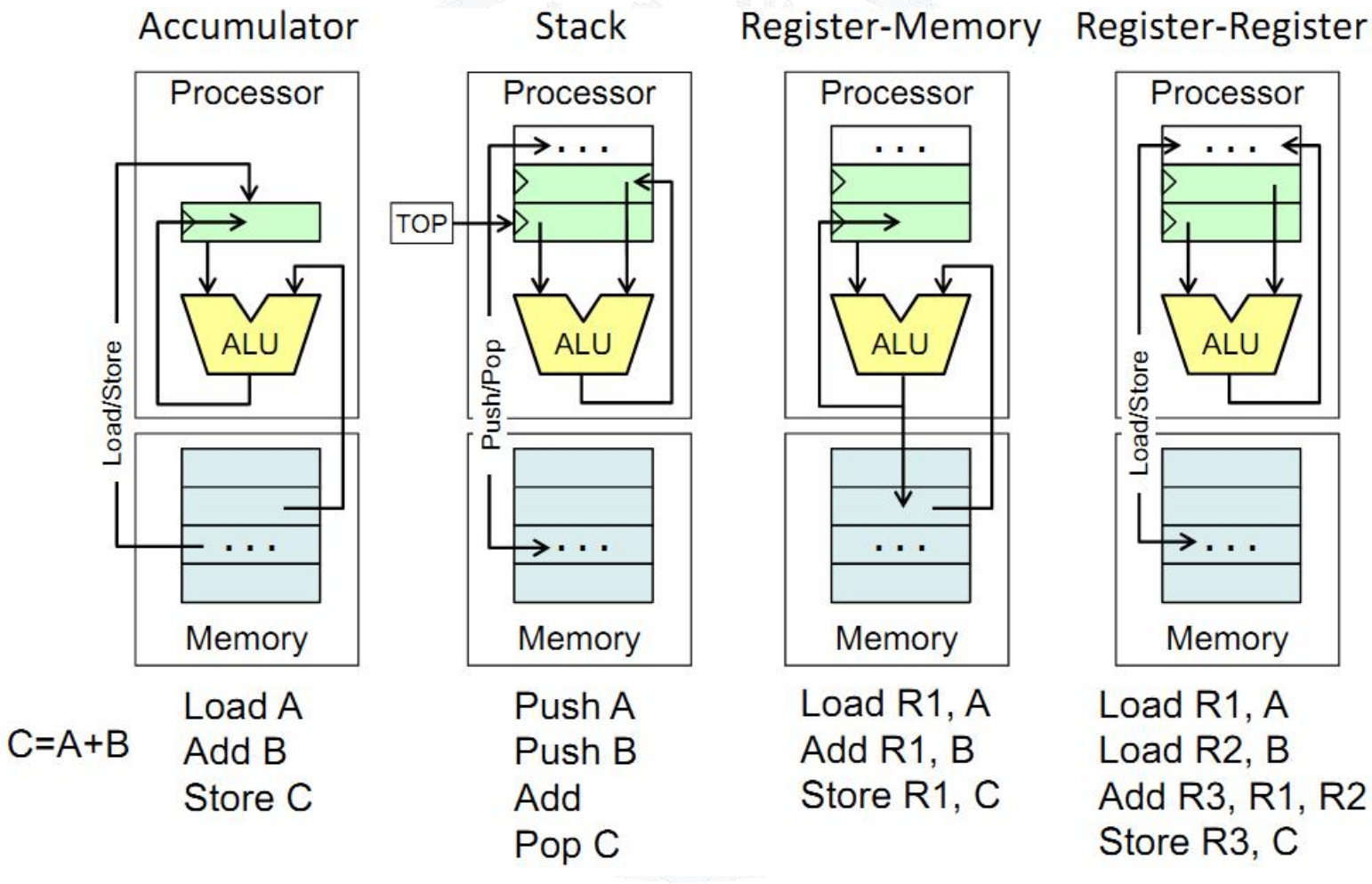


	MIPS	SPARC	Alpha	ARMv7	ARMv8	OpenRISC	80x86
Free and Open		√				√	
64-bit Address	√	√	√		√	√	√
Compressed Instructions	√			√			Partial
Separate Privileged ISA			√				
Position-Indep. Code	Partial			√	√		√
IEEE 754-2008					√		√
Classically Virtualizable	√	√	√		√		



EDA Lab

ISA演化


Accumulator
Stack
Register-Memory
Register-Register
Processor
ALU
Load/Store
TOP
Push/Pop
Memory
C=A+B
Load A
Add B
Store C
Push A
Push B
Add
Pop C
Load R1, A
Add R1, B
Store R1, C
Load R1, A
Load R2, B
Add R3, R1, R2
Store R3, C

指令的问题

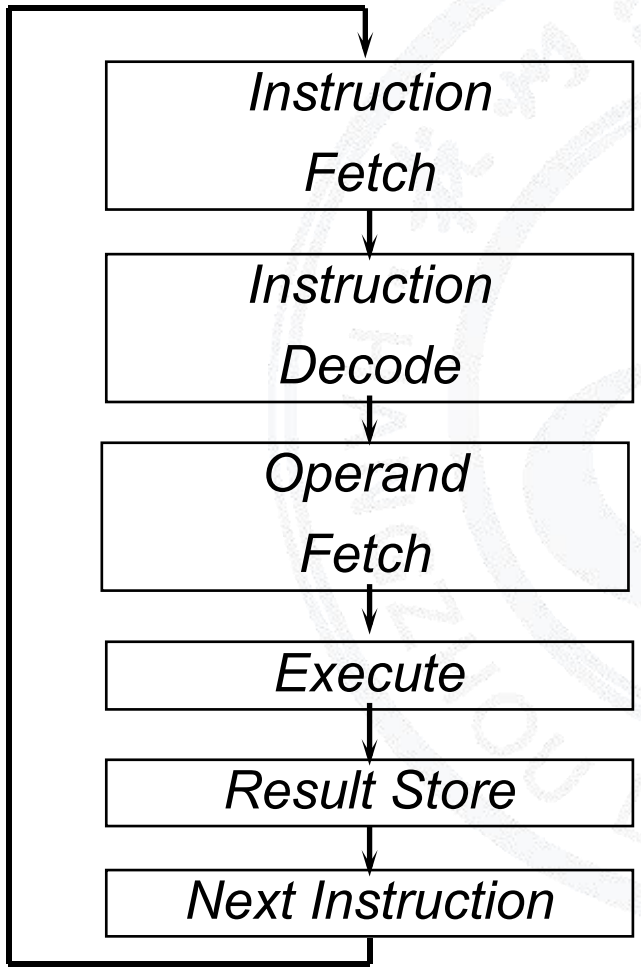


- 存储器寻址
- 操作数的类型与大小
- 所支持的操作
- 控制转移类指令
- 指令格式
- 大小端
- 对齐问题



EDA Lab

执行过程



- Instruction Fetch
- Instruction Decode
- Operand Fetch
- Execute
- Result Store
- Next Instruction

EDA Lab

RISC-V ISA



- UC Berkeley 设计的第5代RISC指令集
- 设计理念（指导思想）：通用的ISA
  - 能适应从最袖珍的嵌入式控制器，到最快的高性能计算机等各种规模的处理器。
  - 能兼容各种流行的软件栈和编程语言。
  - 适应所有实现技术，包括现场可编程门阵列（FPGA）、专用集成电路（ASIC）、全定制芯片，甚至未来的技术。



EDA Lab

RISC-V ISA

- 对所有微体系结构实现方式都有效。例如：
  - 微编码或硬连线控制；顺序或乱序执行流水线；单发射或超标量等等。
- 支持广泛的定制化，成为定制加速器的基础。随着摩尔定律的消退，加速器的重要性日益提高。



EDA Lab

RISC-V ISA

- 基础的指令集架构是稳定的。不能像以前的专有指令集架构一样被弃用，例如AMD Am29000、Digital Alpha、Digital VAX、Hewlett Packard PA-RISC、Intel i860、Intel i960、Motorola 88000、以及Zilog Z8000。
- 完全开源

HANGZHOU DIANZI UNIVERSITY 1956

技术目标

- 将ISA分成基础ISA和可选的扩展部分
  - ISA的基础足够简单，可以用于教学和许多嵌入式处理器，包括定制加速器的控制单元。它足够完整，可以运行现代软件栈。
  - 扩展部分提高计算的性能，并支持多处理机并行
  - 支持32位和64位地址空间
- 方便定制ISA扩展
  - 包括紧耦合功能单元和松耦合协处理器



EDA Lab

技术目标

- 支持变长指令集扩展
  - 既为了提高代码密度，也为了扩展可能的自定义ISA扩展的空间
- 提供对现代标准的有效硬件支持
- 用户级ISA和特权级ISA是正交的（相互独立，互不依赖）
  - 在保持用户应用程序二进制接口(ABI)兼容性的同时，允许完全虚拟化，并允许在特权ISA中进行实验测试

XIDIAN UNIVERSITY

RISC-V ISA的特点



- 完全开源：
  - 它属于一个开放的，非营利性质的RISC-V基金会。
  - 开源采用BSD协议（企业完全自由免费使用，允许企业添加自有指令而不必开放共享以实现差异化发展）



EDA Lab

RISC-V ISA的特点

- 架构简单
  - 没有针对某一种微体系结构实现方式做过度的架构设计
  - 新的指令集，没有向后（backward）兼容的包袱
  - 说明书的页数.....（图1.6）

RISC-V ISA的特点

- 模块化的指令集架构
  - RV32I和RV64I是基础的ISA。可扩展增加其他特性的支持
  - 面向教育或科研，易于扩充或剪裁
  - 支持32位和64位地址空间

HANGZHOU DIANZI UNIVERSITY 1956

RISC-V ISA的特点



- 面向多核并行
- 有效的指令编码方式



EDA Lab

实验与设计



- 11-1 16位CPU验证性设计综合实验
- 11-2 新指令设计及程序测试实验
- 11-3 16位CPU的优化设计与创新
- 11-4 CPU创新设计竞赛

HANGZHOU DIANZI UNIVERSITY 1956

11-2 新指令设计及程序测试实验

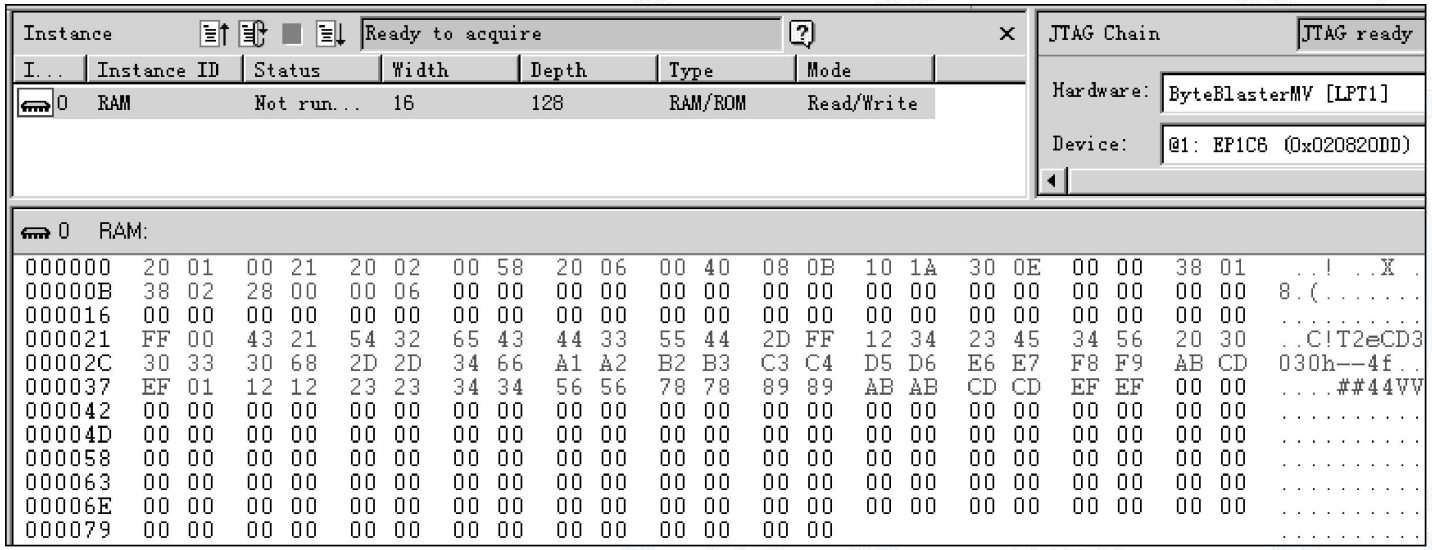


图11-28 用In-System Memory Content Editor读取的数据

RAM_16.mif

Addr	+0	+1	+2	+3	+4	+5	+6	+7
00	2001	0021	2002	0058	2006	0040	080B	101A
08	300E	0000	3801	3802	2800	0006	0000	0000
10	0000	0000	0000	0000	0000	0000	0000	0000
18	0000	0000	0000	0000	0000	0000	0000	0000
20	0000	FF00	4321	5432	6543	4433	5544	2DFF
28	1234	2345	3456	2030	3033	3068	2D2D	3466
30	A1A2	B2B3	C3C4	D5D6	E6E7	F8F9	ABCD	EF01
38	1212	2323	3434	5656	7878	8989	ABAB	CDCD
40	EFEF	0000	0000	0000	0000	0000	0000	0000
48	0000	0000	0000	0000	0000	0000	0000	0000
50	0000	0000	0000	0000	0000	0000	0000	0000
58	0000	0000	0000	0000	0000	0000	0000	0000

图11-27 编辑ram_16.mif文件

EDA Lab