CPU设计案例解析

CPU设计案例解析

- 1 CPU概述
- 2 简单CPU设计方法
- 3 功能模块设计
- 4 系统调试

什么是CPU

CPU：中央处理单元，是计算机的核心部件

计算机进行信息处理分两个步骤：

- 将数据和程序（指令序列）输入到计算机的存储器中
- 从第一条指令的地址起开始执行程序，得到所需结果，结束运行。

CPU的作用：协调并控制计算机的各部件，使之有条不紊的执行程序

CPU的基本功能

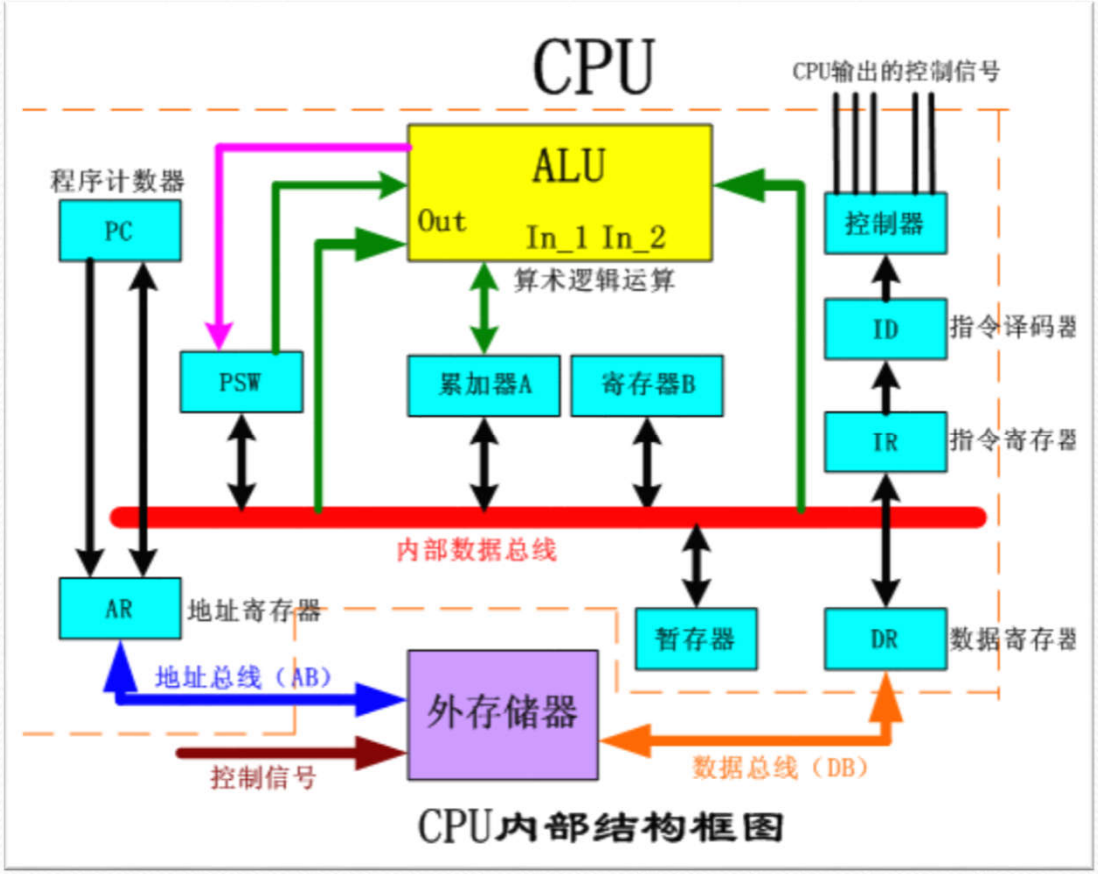
CPU的基本功能：

- 取指令-地址与控制信号
- 分析指令-即指令译码，操作和操作控制信号
- 执行指令-操作控制信号作用于各部件

CPU的基本功能概括：

- 能对指令进行译码并执行规定的动作
- 可以进行算术和逻辑运算
- 能与存储器和外设交换数据
- 提供系统所需的控制

CPU的内部结构



CPU内部结构框图

- 运算器和控制器组成
- 按时钟节拍工作
- 工作的规定是指令
- 指令由操作码和操作数组成
- 过程是取指令和执行指令
- 执行指令是产生各种按一定的时序的控制信号

CPU设计案例解析

- 1 CPU概述
- 2 简单CPU设计方法
- 3 功能模块设计
- 4 系统调试

简单CPU系统结构

CPU的内部结构：

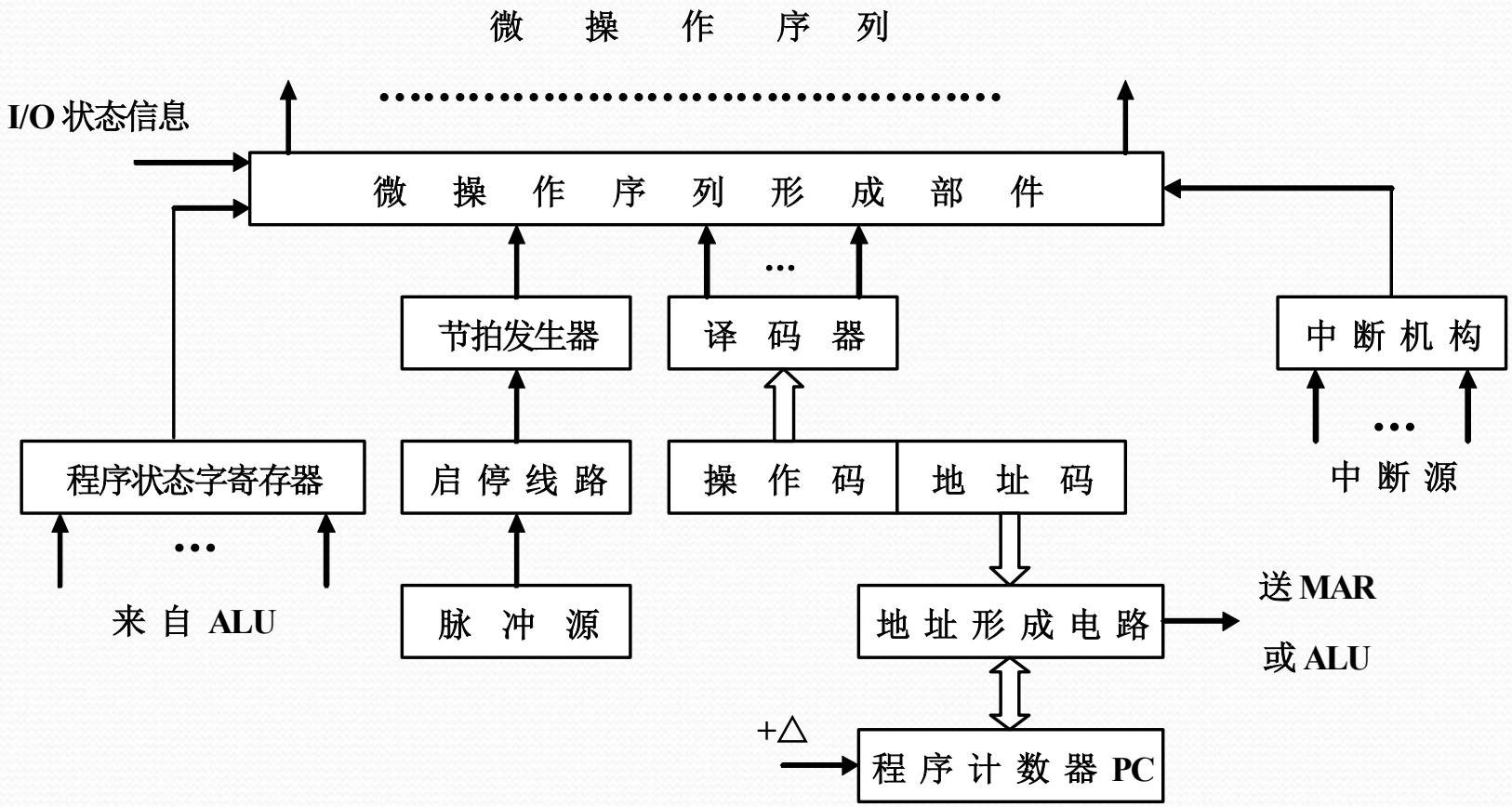
- 算术逻辑运算单元（ALU）
- 累加器
- 程序计数器
- 指令寄存器和译码器
- 时序和控制部件

RISC即精简指令集计算机：

Reduced Instruction Set Computer

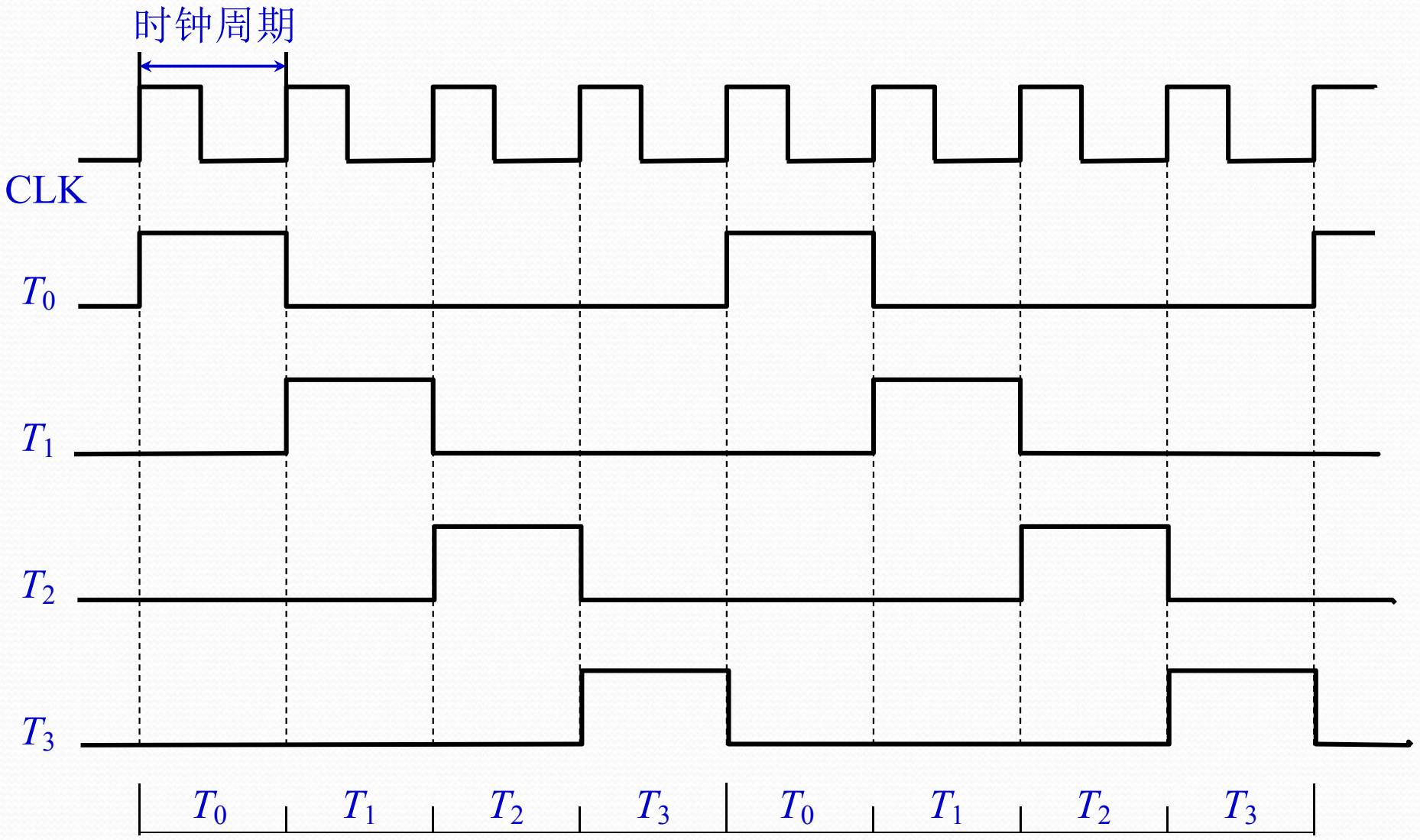
时序控制信号由硬件而不是微程序控制

控制器的基本组成

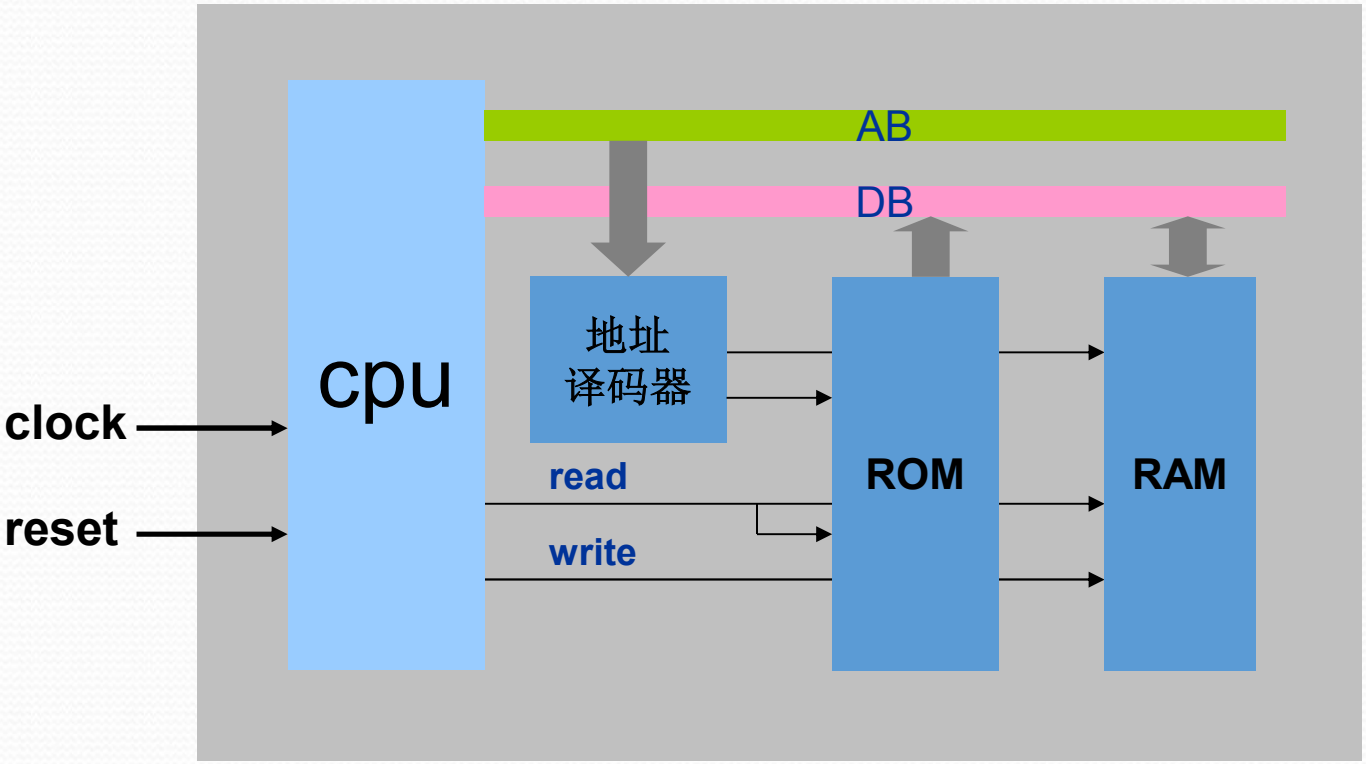


控制器基本组成框图

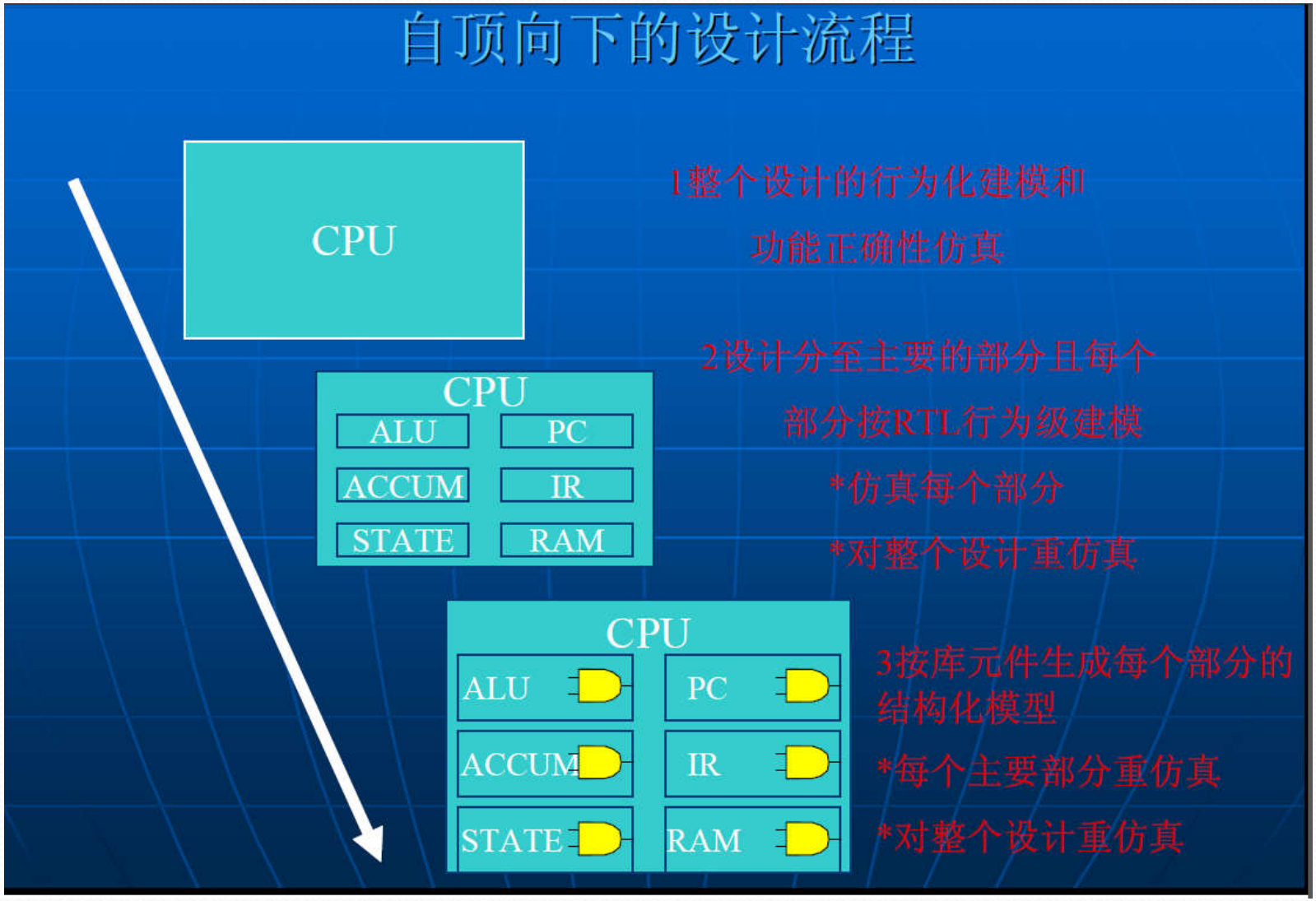
时钟周期(节拍、状态)

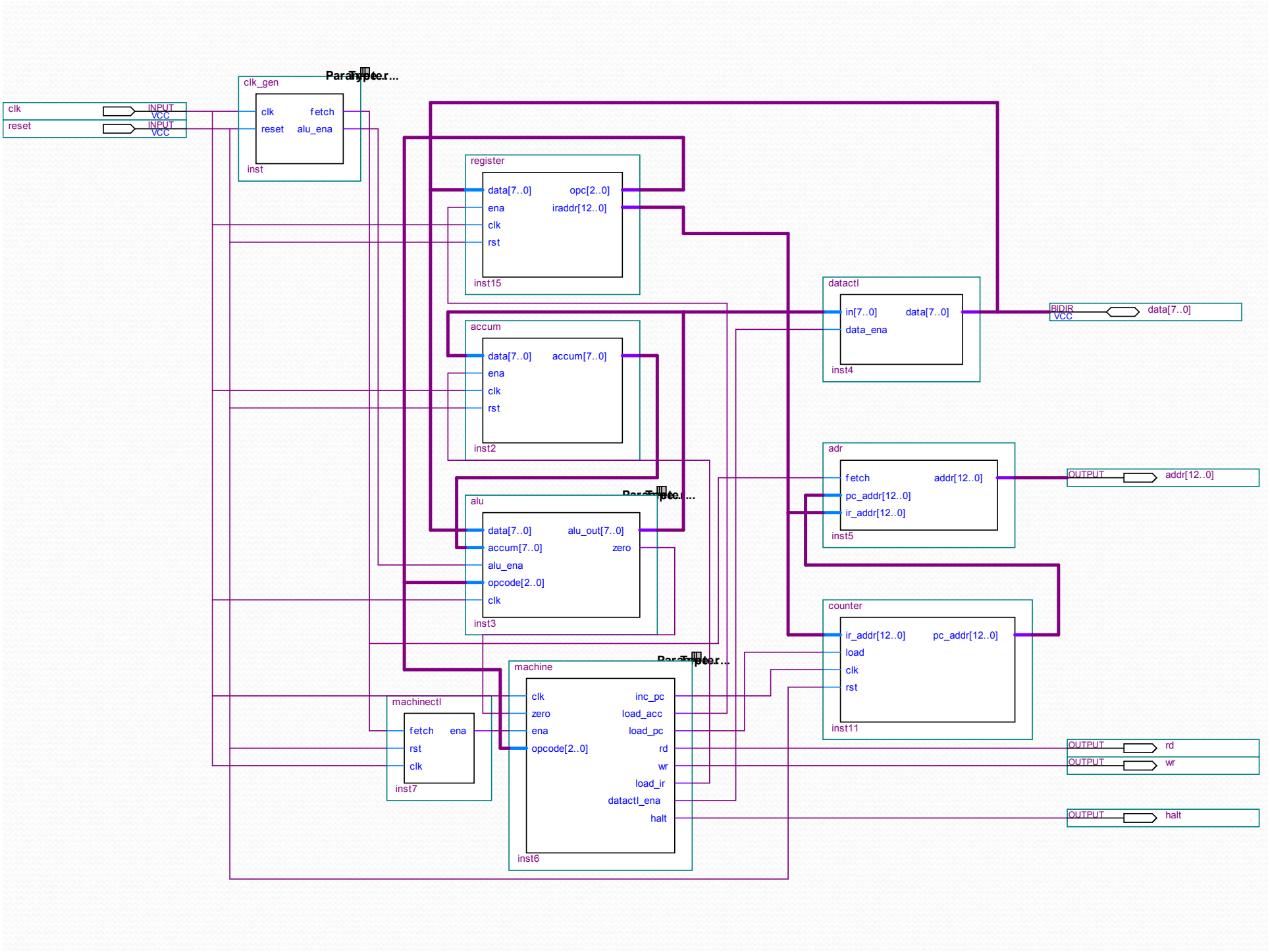


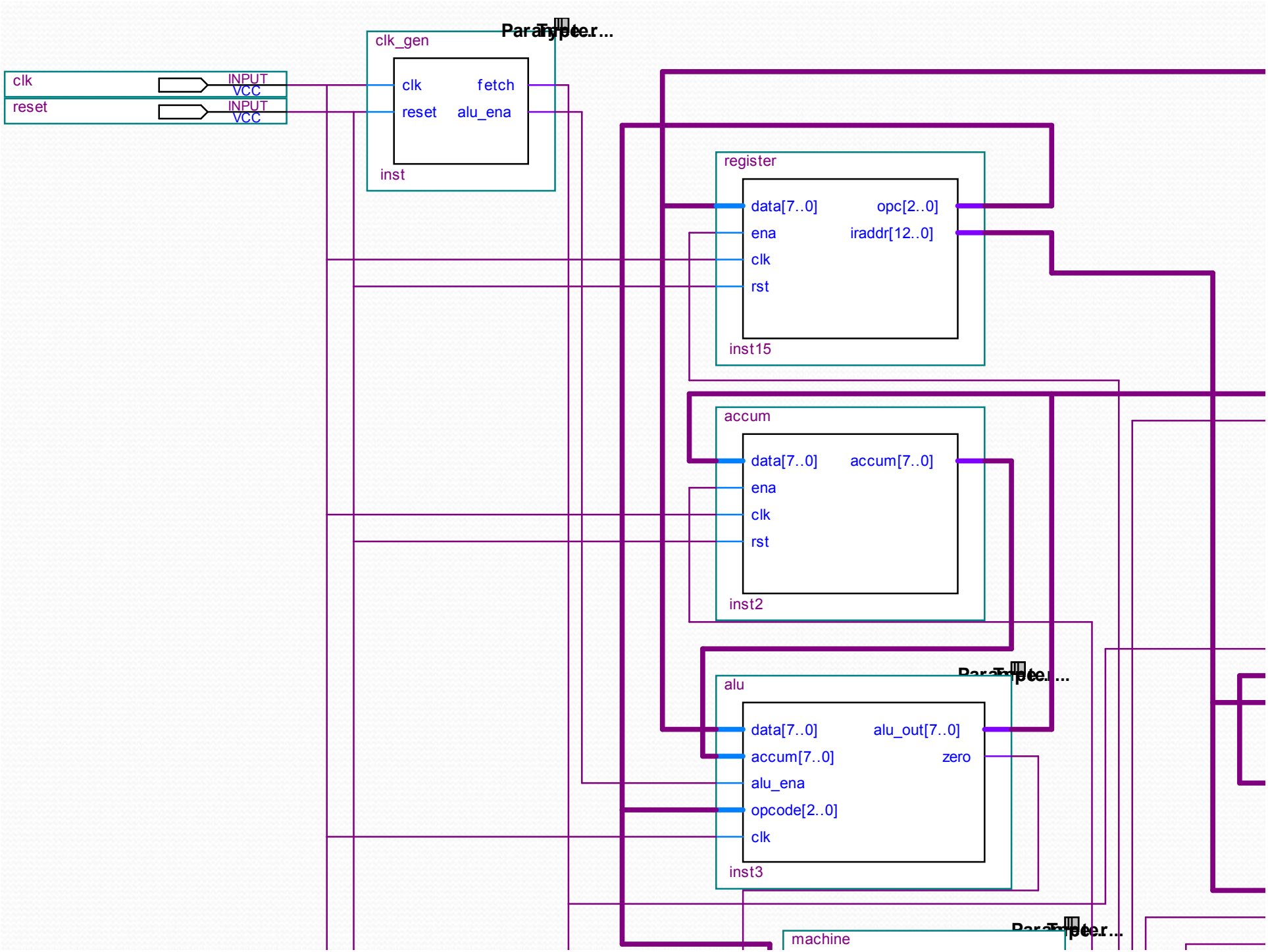
典型的计算机系统



自顶而下的设计流程


自顶向下的设计流程
CPU
1整个设计的行为化建模和
功能正确性仿真
CPU
ALU
PC
ACCUM
IR
STATE
RAM
2设计分至主要的部分且每个
部分按RTL行为级建模
*仿真每个部分
*对整个设计重仿真
CPU
ALU
PC
ACCUM
IR
STATE
RAM
3按库元件生成每个部分的
结构化模型
*每个主要部分重仿真
*对整个设计重仿真

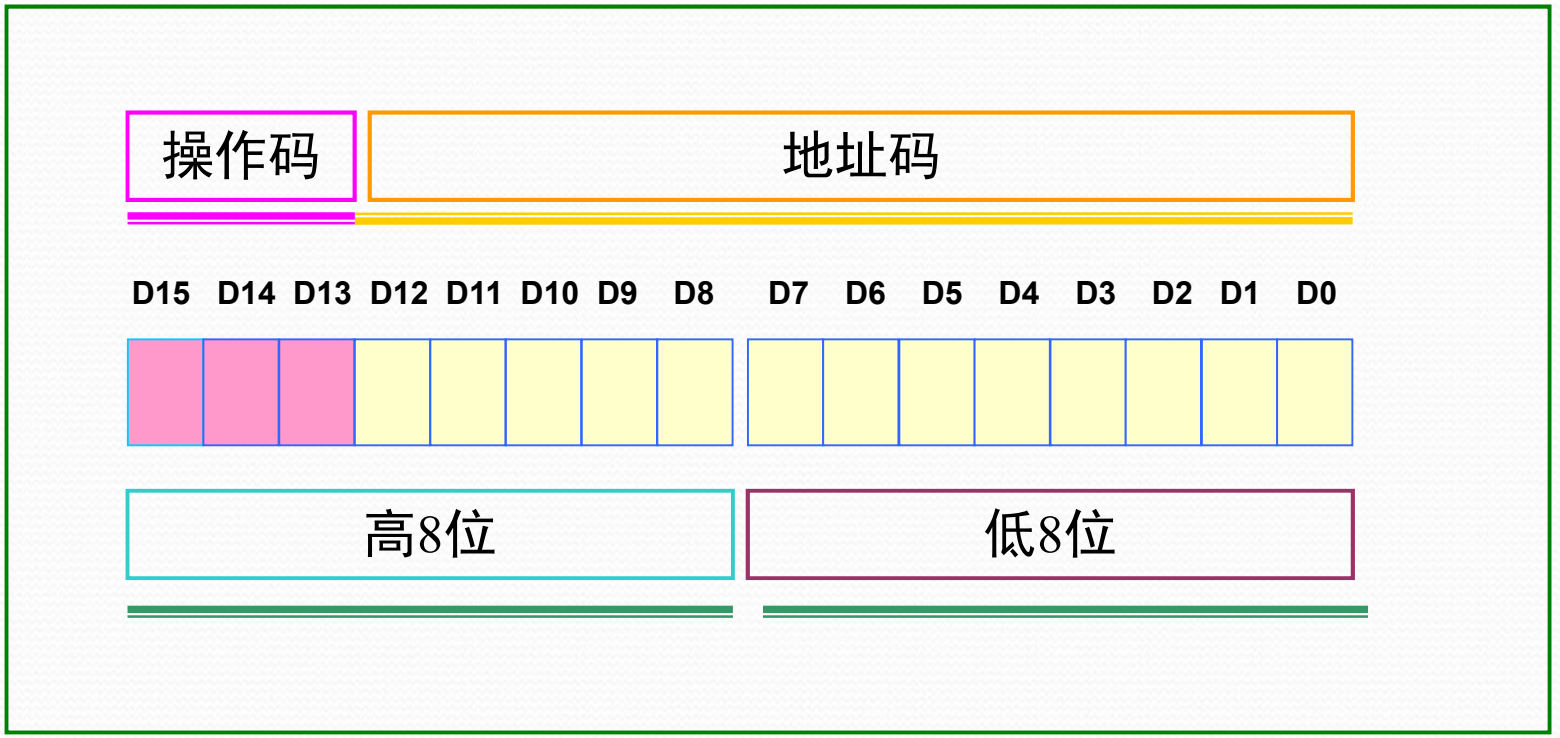

clk
INPUT
VCC
reset
INPUT
VCC
clk_gen
clk
reset
fetch
alu_ena
inst
register
data[7..0]
ena
clk
rst
opc[2..0]
iraddr[12..0]
inst15
accum
data[7..0]
ena
clk
rst
accum[7..0]
inst2
alu
data[7..0]
accum[7..0]
alu_ena
opcode[2..0]
clk
alu_out[7..0]
zero
inst3
machinectl
fetch
rst
clk
ena
inst7
machine
clk
zero
ena
opcode[2..0]
inc_pc
load_acc
load_pc
rd
wr
load_ir
datactl_ena
halt
inst6
datactl
in[7..0]
data_ena
data[7..0]
inst4
BIDIR
VCC
data[7..0]
adr
fetch
pc_addr[12..0]
ir_addr[12..0]
addr[12..0]
inst5
OUTPUT
addr[12..0]
counter
ir_addr[12..0]
load
clk
rst
pc_addr[12..0]
inst11
OUTPUT
rd
OUTPUT
wr
OUTPUT
halt


clk
INPUT
VCC
reset
INPUT
VCC
clk_gen
clk
reset
fetch
alu_ena
inst
register
data[7..0]
ena
clk
rst
opc[2..0]
iraddr[12..0]
inst15
accum
data[7..0]
ena
clk
rst
accum[7..0]
inst2
alu
data[7..0]
accum[7..0]
alu_ena
opcode[2..0]
clk
alu_out[7..0]
zero
inst3
machine

CPU设计案例解析

- 1 CPU概述
- 2 简单CPU设计方法
- 3 功能模块设计
- 4 系统调试

指令



- 指令数目：8条
- 地址空间：$2^{13}=8K$

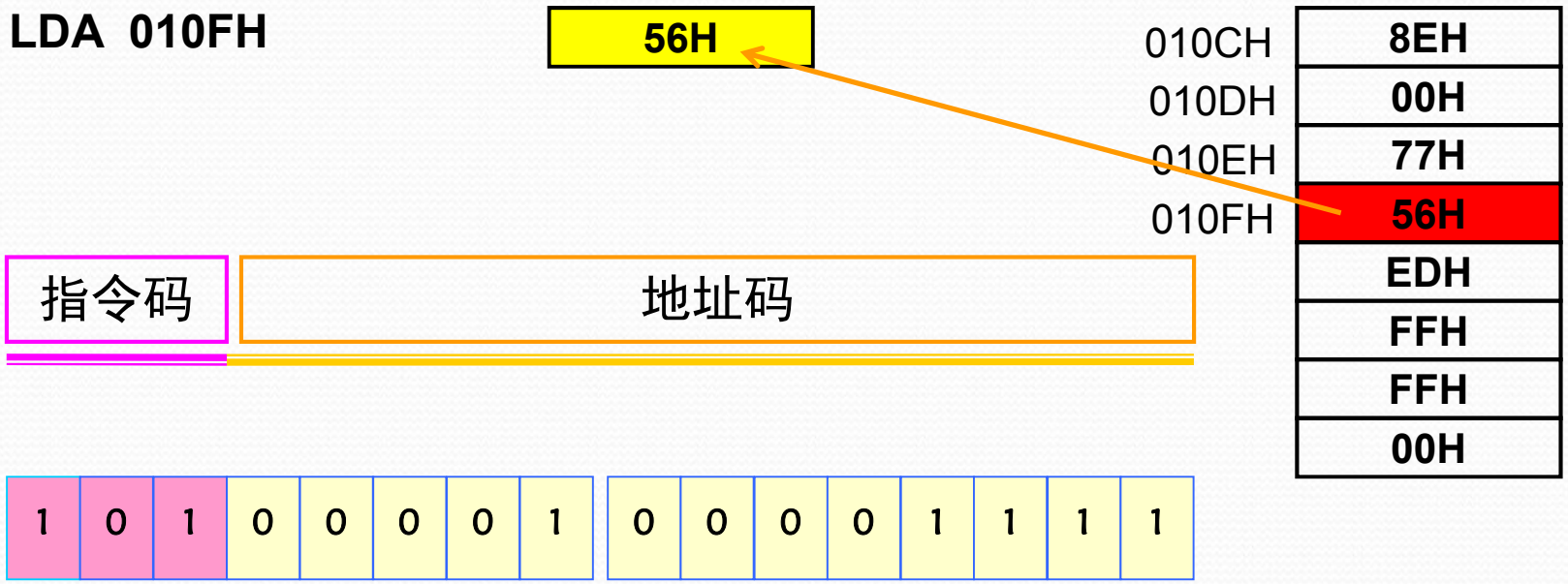
指令集

助记符	操作码	功能
HLT	3'b000	暂停
SKZ	3'b001	累加器为零转移
ADD	3'b010	加
ANDD	3'b011	与
XORR	3'b100	或
LDA	3'b101	取数
STO	3'b110	存数
JMP	3'b111	转移

寻址方式

直接寻址方式

- 数据放在存储器中，寻址单元的地址由指令直接给出。



指令系统

HLT: 停机操作

- 该操作空一个指令周期，即8个时钟

SKZ: 为零跳过下一条语句

- 累加器结果为零，跳过下一语句；
- 累加器结果不为零，顺序执行。

ADD: 相加

- 累加器的值+地址所知的存储器中的数据，结果送回累加器。

指令系统

ANDD: 相与

- 累加器的值&地址所知的存储器中的数据，结果送回累加器。

XORR: 相异或

- 累加器的值^地址所知的存储器中的数据，结果送回累加器。

LDA: 读数据

- 地址所知的存储器中的数据送累加器。

指令系统

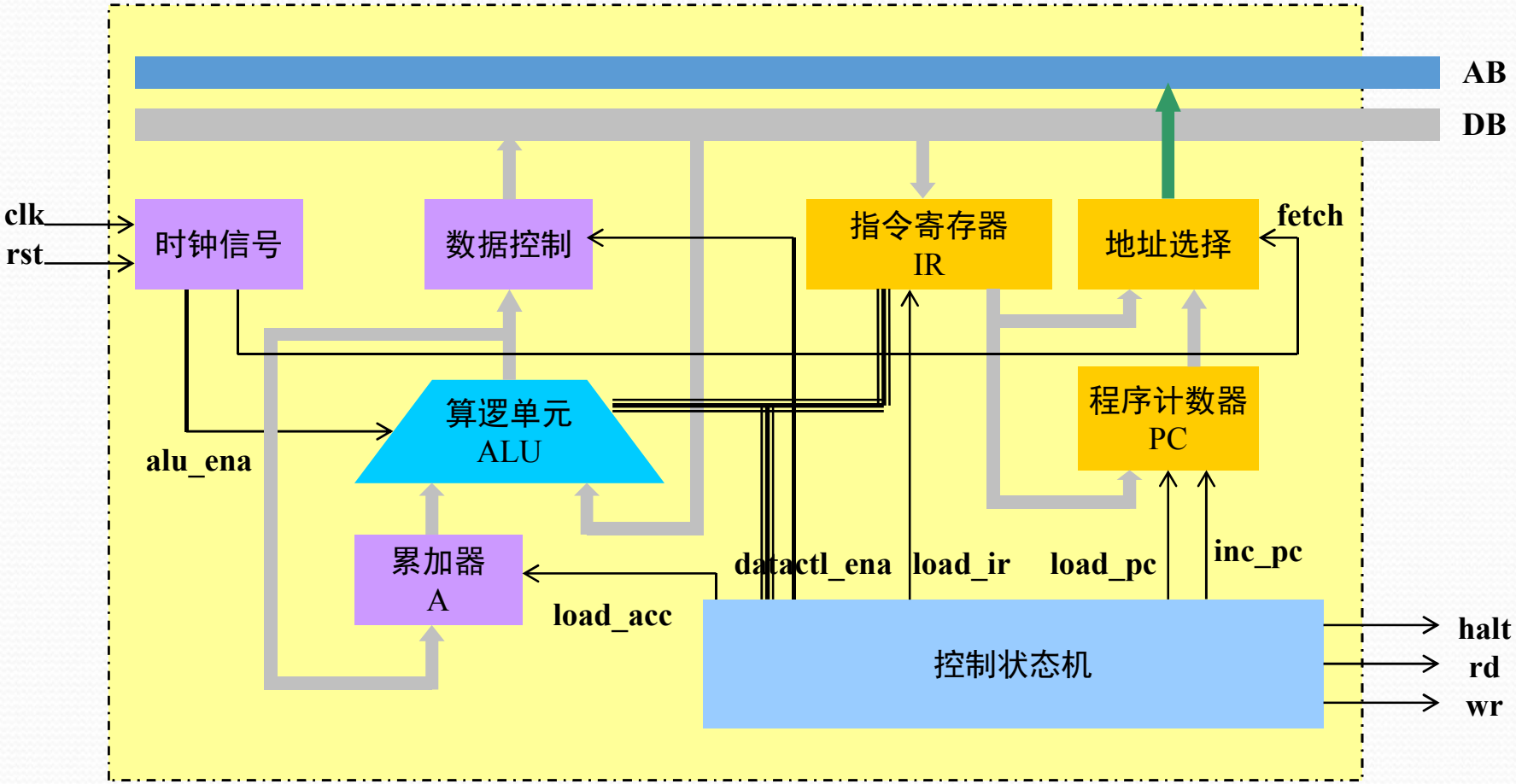
STO: 写数据

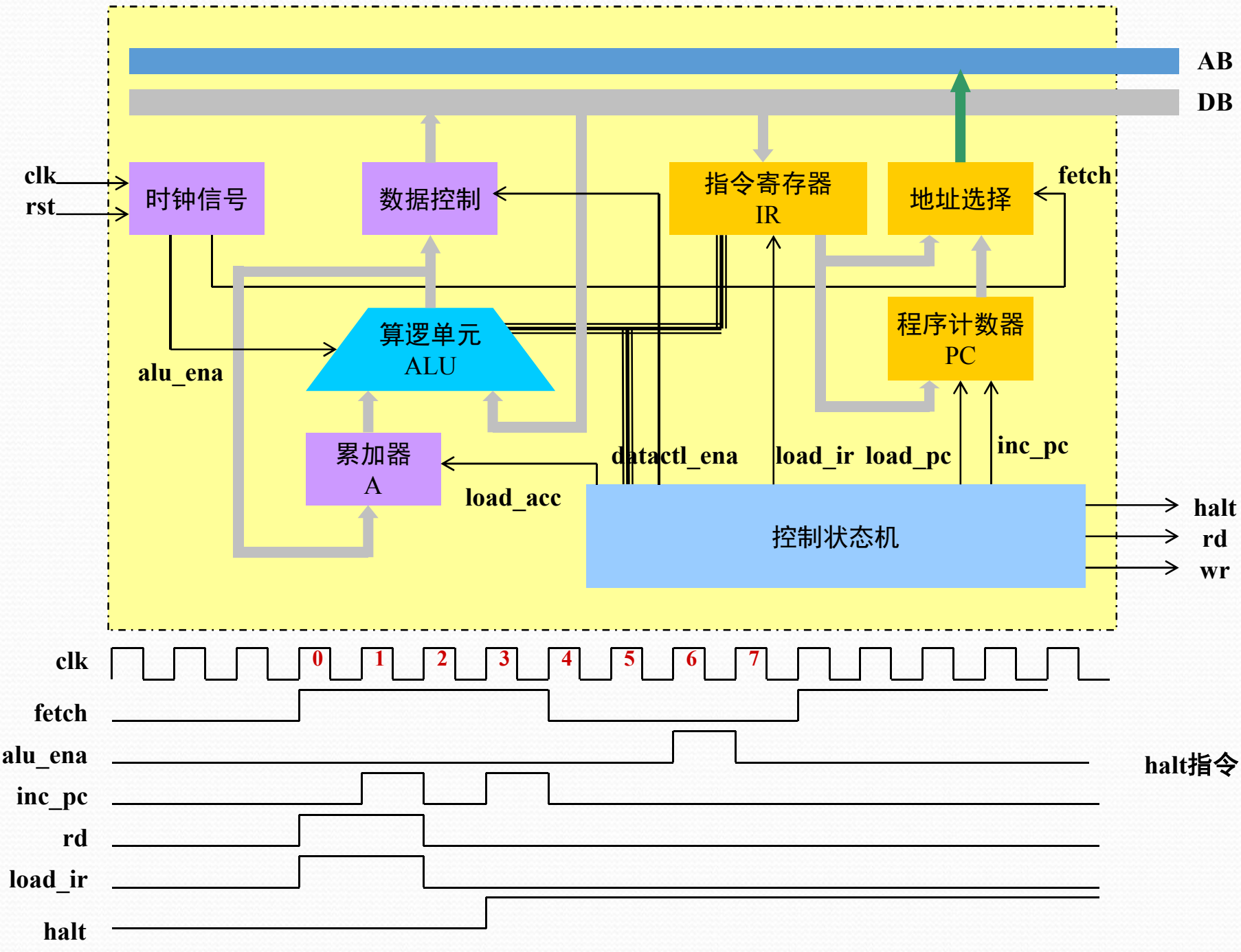
- 累加器中的数据送地址所指的存储器中。

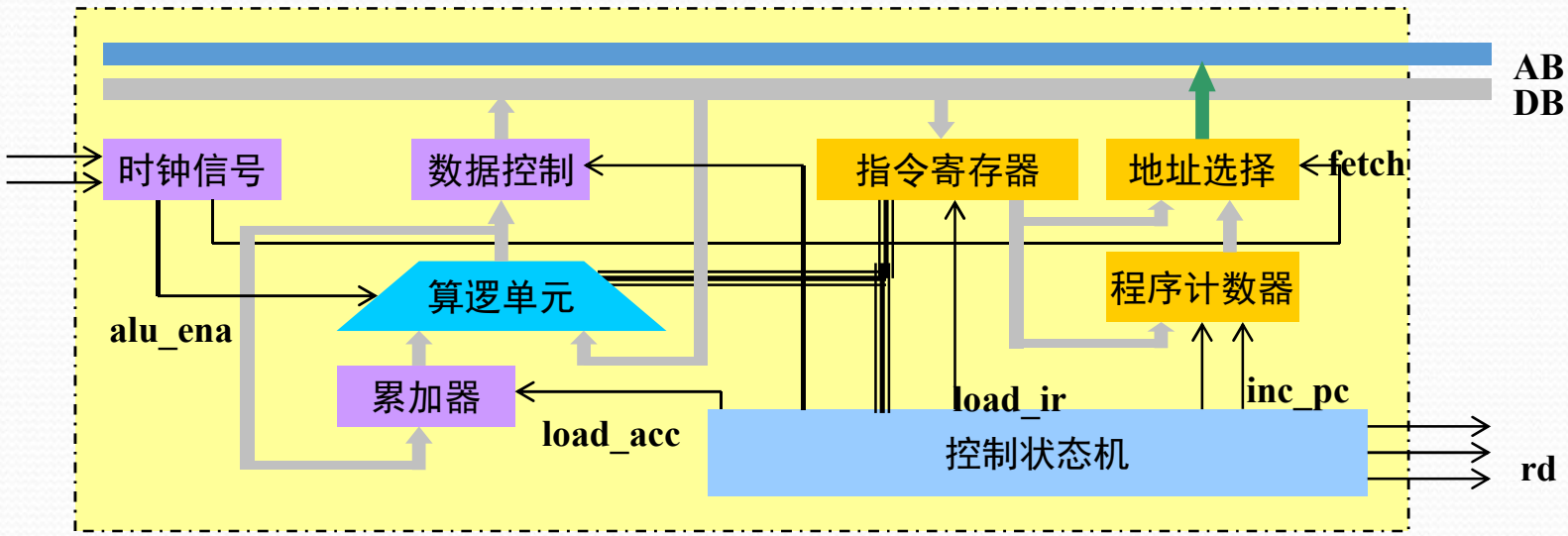
JMP: 无条件跳转语句

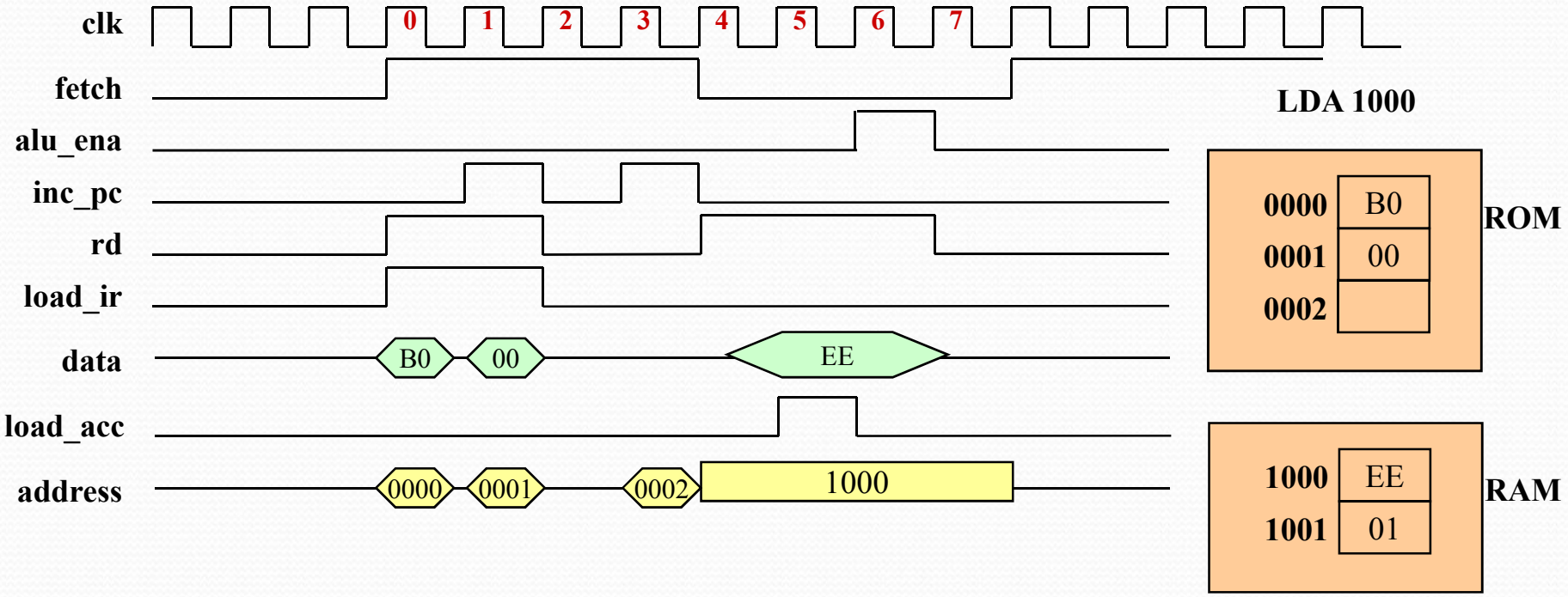
- 该操作跳转至指令给出的目的地址，继续执行。

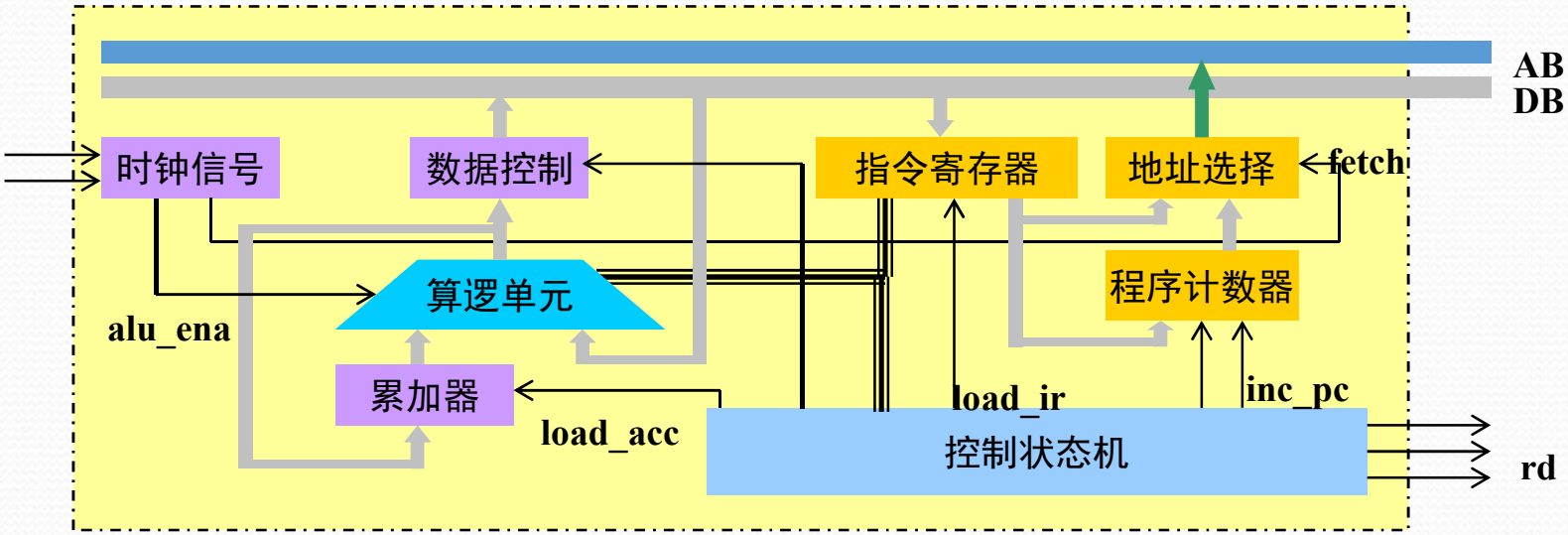
CPU功能结构图

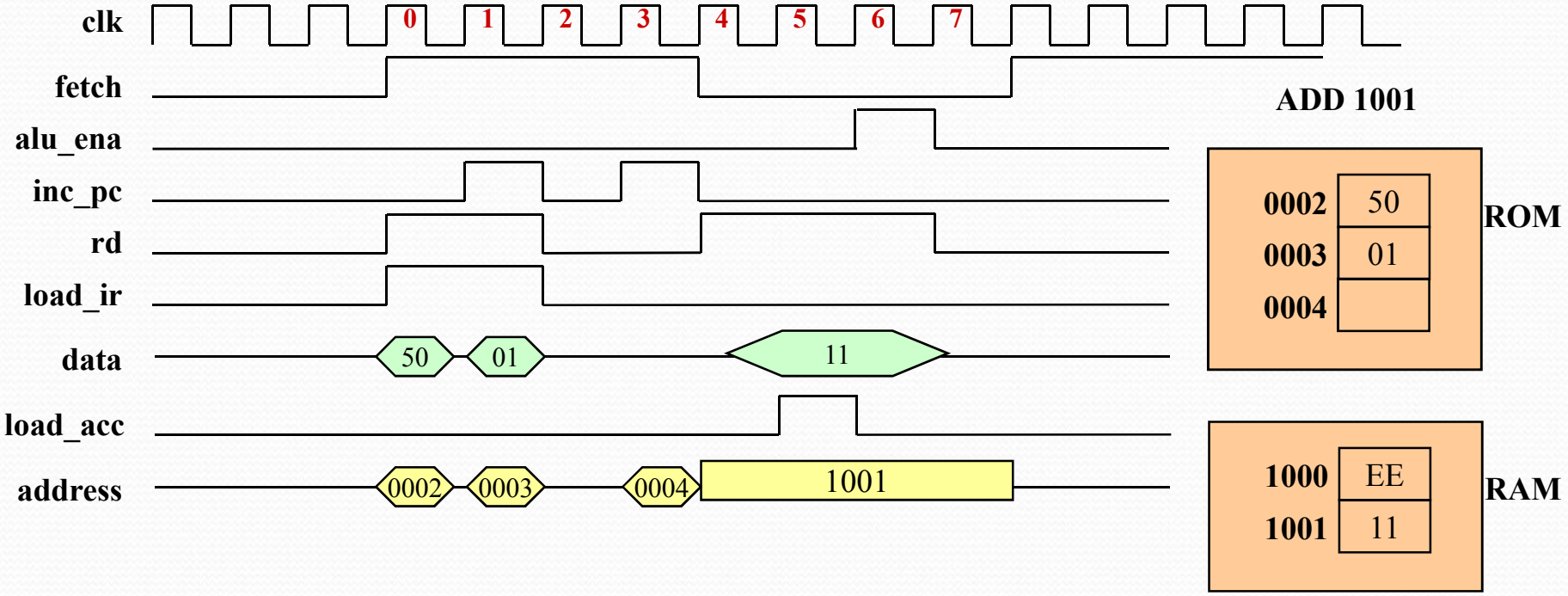

AB
DB
clk
rst
时钟信号
数据控制
指令寄存器
IR
地址选择
fetch
算逻单元
ALU
程序计数器
PC
alu_ena
累加器
A
datactl_ena
load_ir
load_pc
inc_pc
load_acc
控制状态机
halt
rd
wr

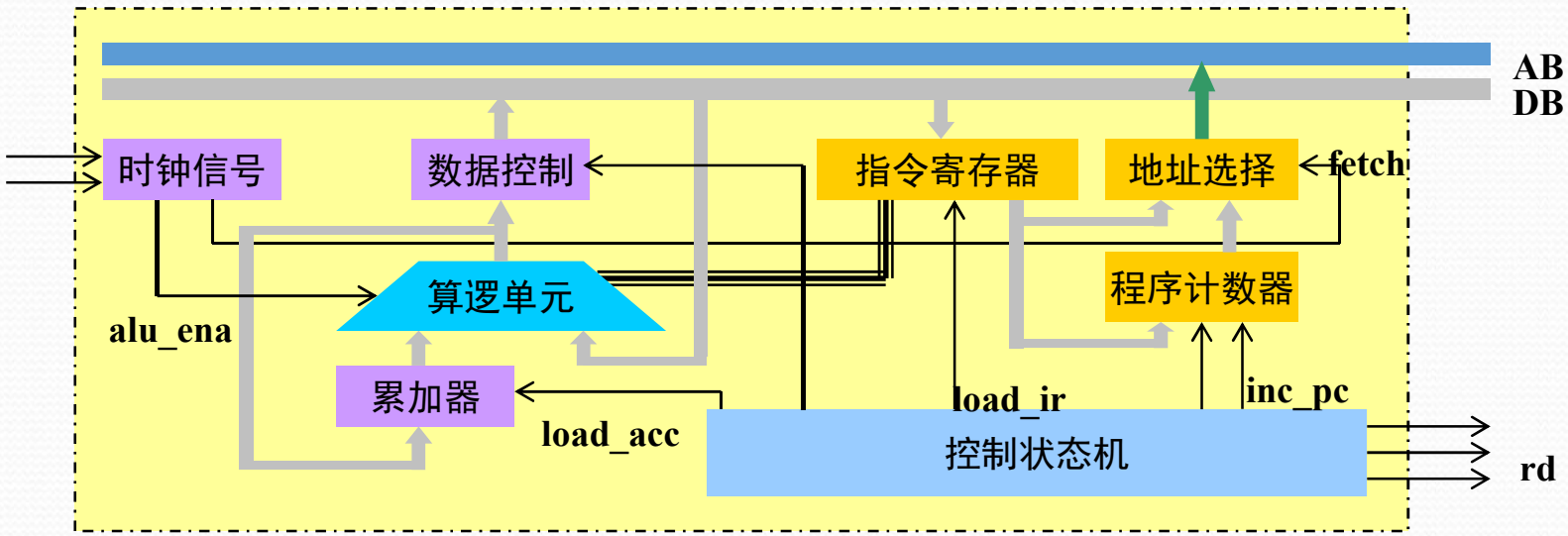

AB
DB
clk
rst
时钟信号
数据控制
指令寄存器
IR
地址选择
fetch
程序计数器
PC
算逻单元
ALU
alu_ena
累加器
A
load_acc
datactl_ena
load_ir
load_pc
inc_pc
控制状态机
halt
rd
wr
clk
0
1
2
3
4
5
6
7
fetch
alu_ena
inc_pc
rd
load_ir
halt
halt指令

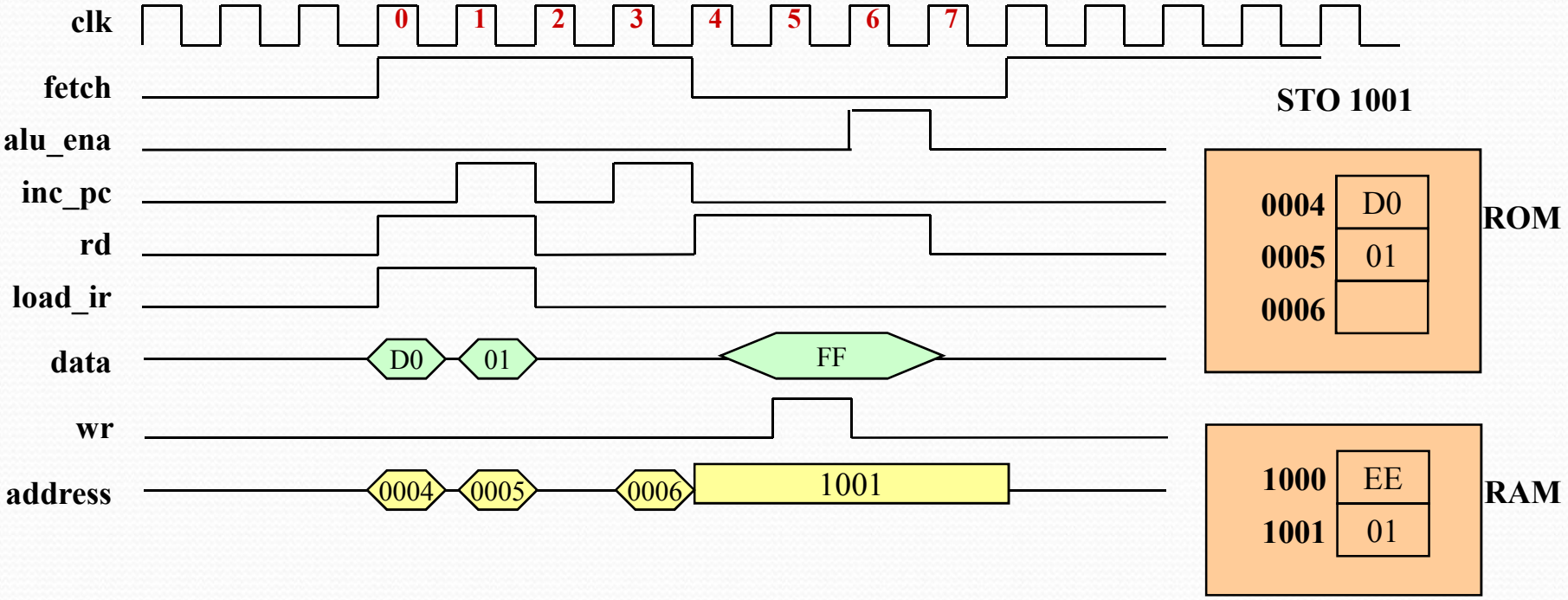

AB
DB
时钟信号
数据控制
指令寄存器
地址选择
fetch
算逻单元
程序计数器
alu_ena
累加器
load_acc
load_ir
inc_pc
控制状态机
rd


clk
0
1
2
3
4
5
6
7
fetch
alu_ena
inc_pc
rd
load_ir
data
B0
00
EE
load_acc
address
0000
0001
0002
1000
LDA 1000
0000
B0
0001
00
0002
ROM
1000
EE
1001
01
RAM


AB
DB
时钟信号
数据控制
指令寄存器
地址选择
fetch
算逻单元
程序计数器
alu_ena
累加器
load_acc
load_ir
inc_pc
控制状态机
rd

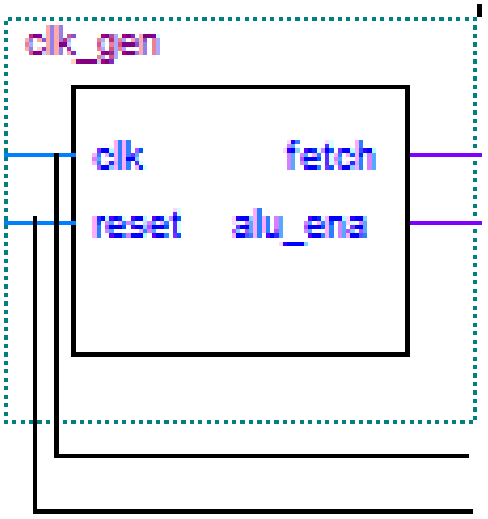

clk
0
1
2
3
4
5
6
7
fetch
alu_ena
inc_pc
rd
load_ir
data
50
01
11
load_acc
address
0002
0003
0004
1001
ADD 1001
0002
50
0003
01
0004
ROM
1000
EE
1001
11
RAM

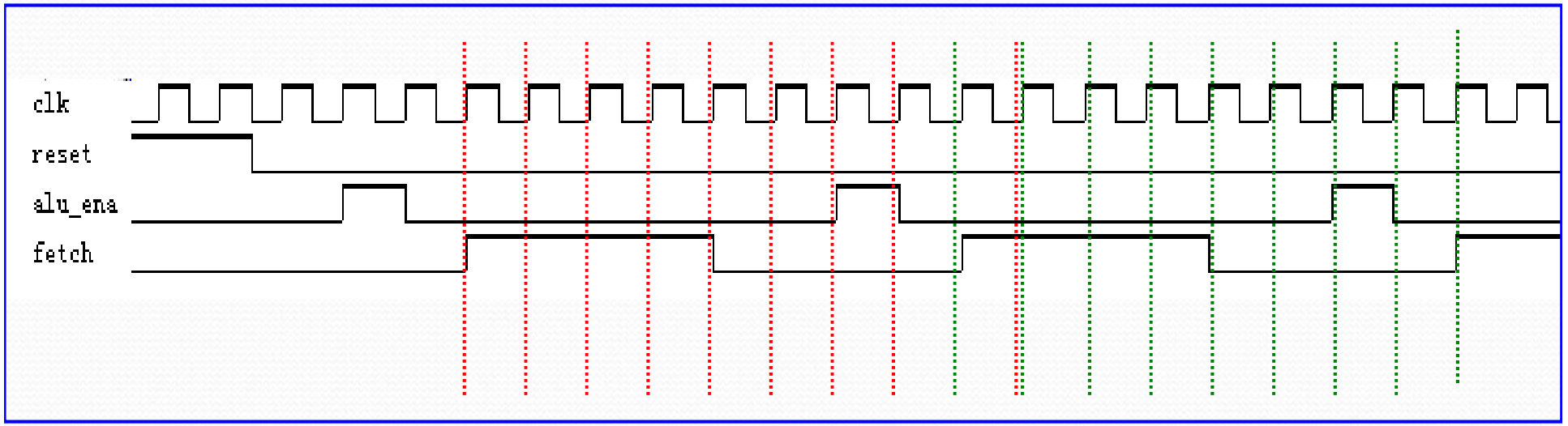

AB
DB
时钟信号
数据控制
指令寄存器
地址选择
fetch
算逻单元
程序计数器
alu_ena
累加器
load_ir
inc_pc
load_acc
控制状态机
rd


clk
0
1
2
3
4
5
6
7
fetch
STO 1001
alu_ena
inc_pc
rd
load_ir
data
D0
01
FF
wr
address
0004
0005
0006
1001
0004
D0
0005
01
0006
ROM
1000
EE
1001
01
RAM

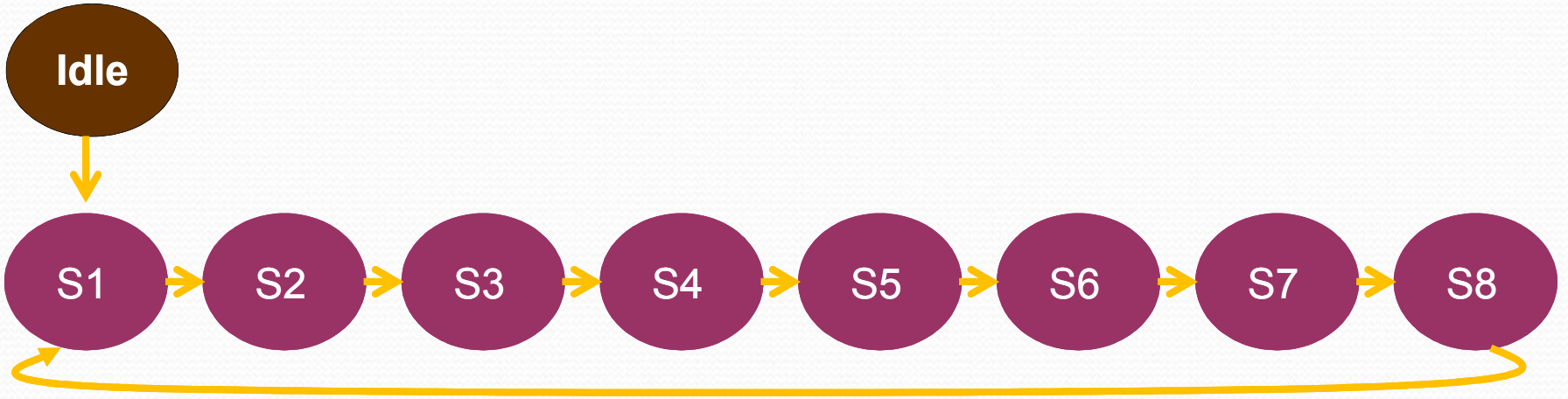
时钟发生器

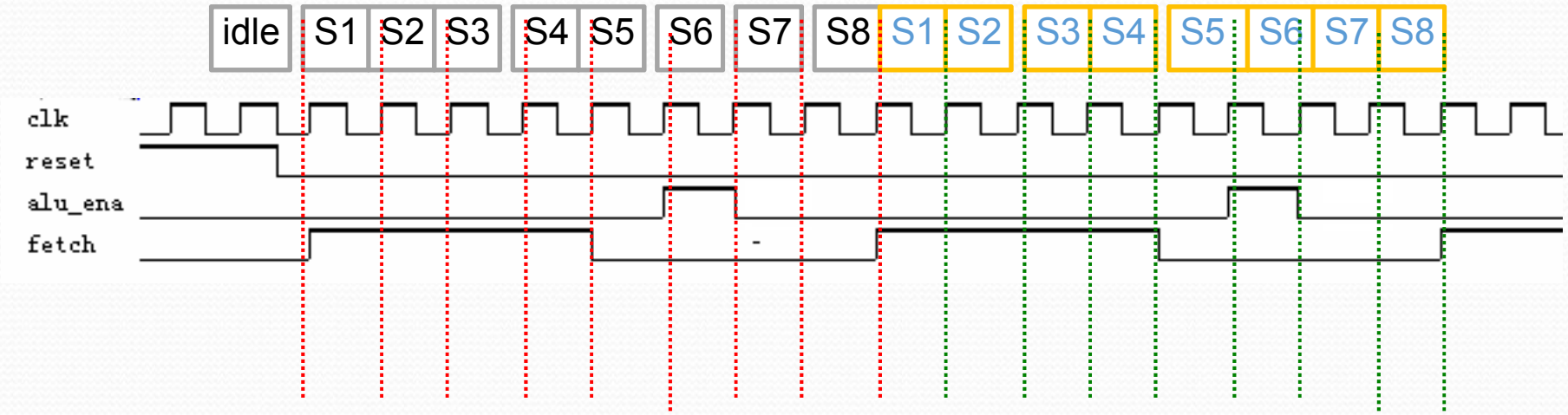
- fetch是控制信号，clk的8分频信号
- alu_ena用于控制算术逻辑运算单元
- clk是时钟信号
- reset是复位信号





时钟发生器


Idle
S1
S2
S3
S4
S5
S6
S7
S8


idle
S1
S2
S3
S4
S5
S6
S7
S8
S1
S2
S3
S4
S5
S6
S7
S8
clk
reset
alu_ena
fetch

状态转移图

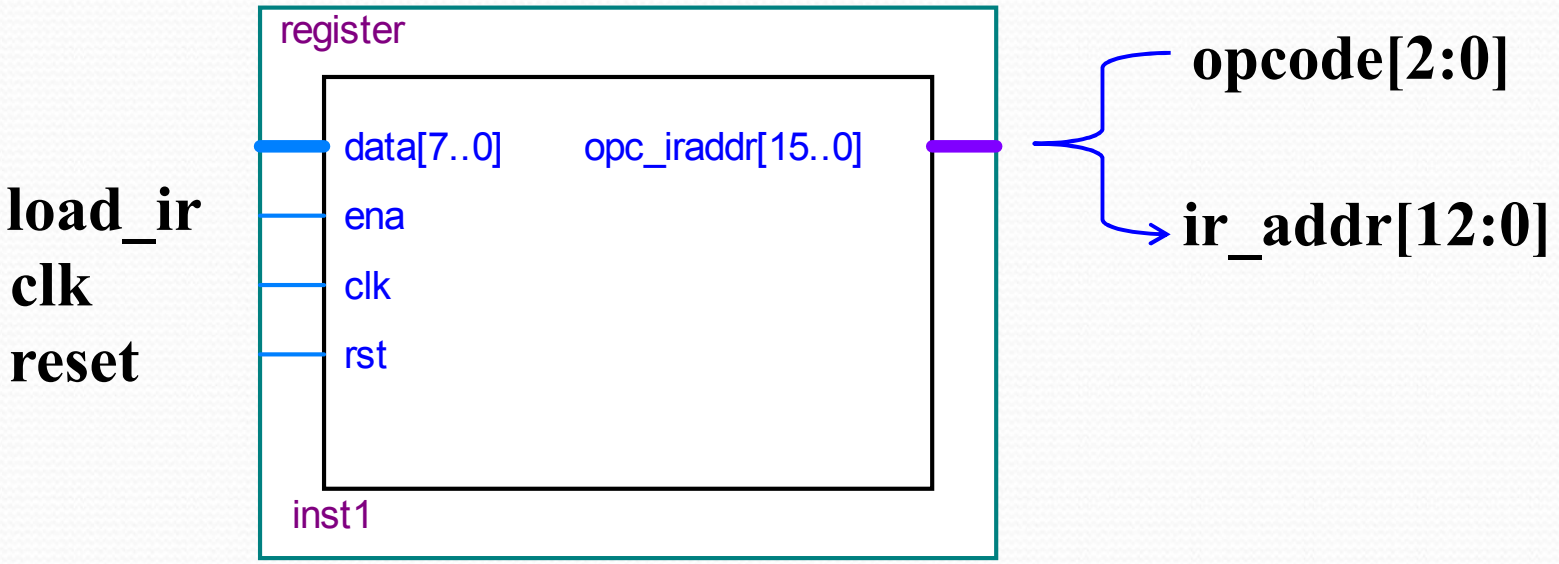
时钟发生器

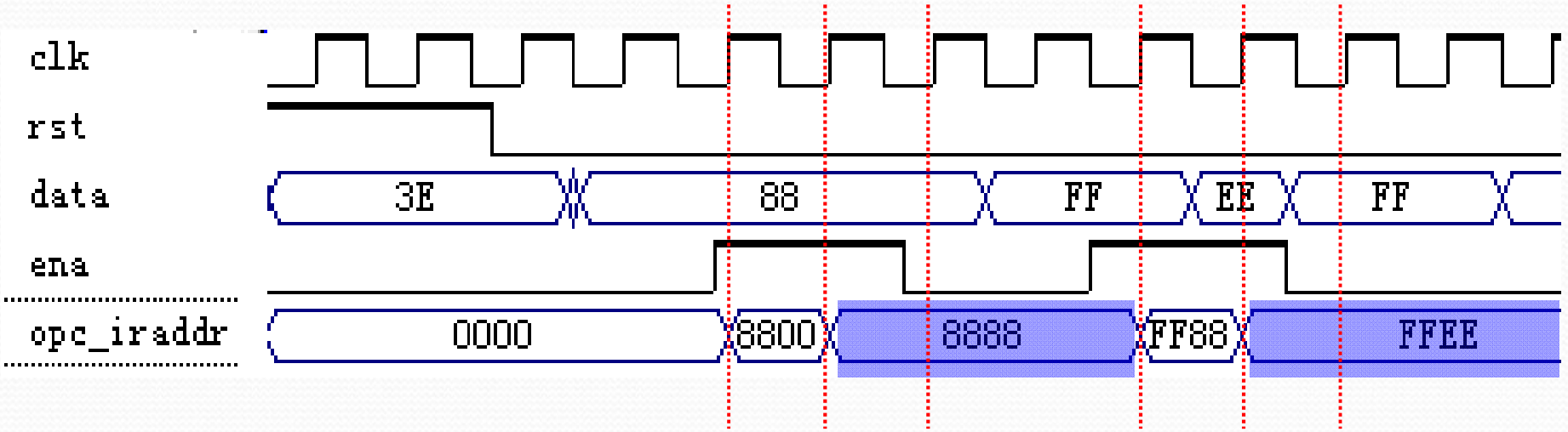
```
//-------clk_gen.v--------
`timescale 1ns/1ns
module  clk_gen(clk,reset,fetch,alu_ena);
   input    clk, reset;
   output  fetch, alu_ena;
   wire      clk, reset;
   reg       fetch, alu_ena;
   reg       [7:0]state;
   parameter         S1=8'b00000001,
                     S2=8'b00000010,
                     S3=8'b00000100,
                     S4=8'b00001000,
                     S5=8'b00010000,
                     S6=8'b00100000,
                     S7=8'b01000000,
                     S8=8'b10000000,
                     idle=8'b00000000;
```

```
always @(posedge clk)
   if(reset)
        begin
          fetch<=0;
          alu_ena<=0;
          state<=idle;
        end
   else
        begin
          case (state)
          …
default:state<=idle;
          endcase
             end
```

```
case (state)
        S1:  begin
             fetch<=1;
             state<=S2;   end
        S2:    state<=S3;
        S3:    state<=S4;
        S4:    state<=S5;
        S5:  begin  state<=S6;
             fetch=0;  end
        S6:  begin
             state<=S7;
             alu_ena<=1; end
        S7:   begin
              alu_ena<=0;
              state<=S8;    end
        S8:   state<=S1;
        idle: state<=S1;
        default:state<=idle;
endcase
```

指令寄存器

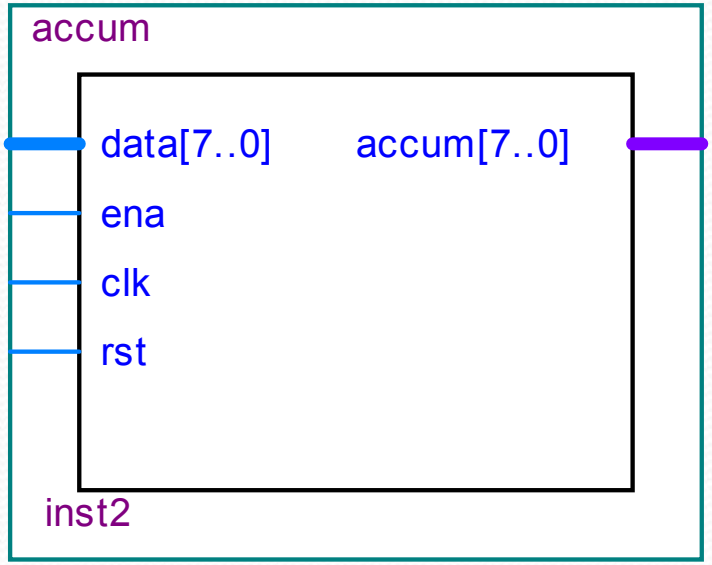

register
data[7..0]
opc_iraddr[15..0]
ena
clk
rst
inst1
load_ir
clk
reset
opcode[2:0]
ir_addr[12:0]

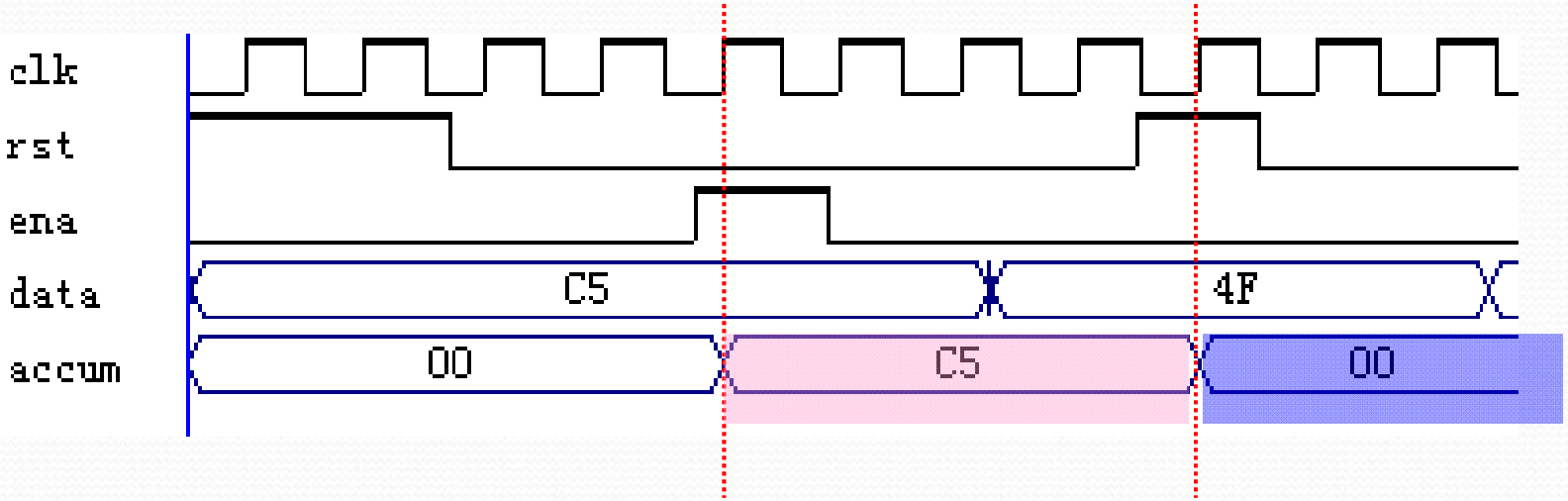

clk
rst
data
ena
opc_iraddr
3E
88
FF
EE
FF
0000
8800
8888
FF88
FFEE

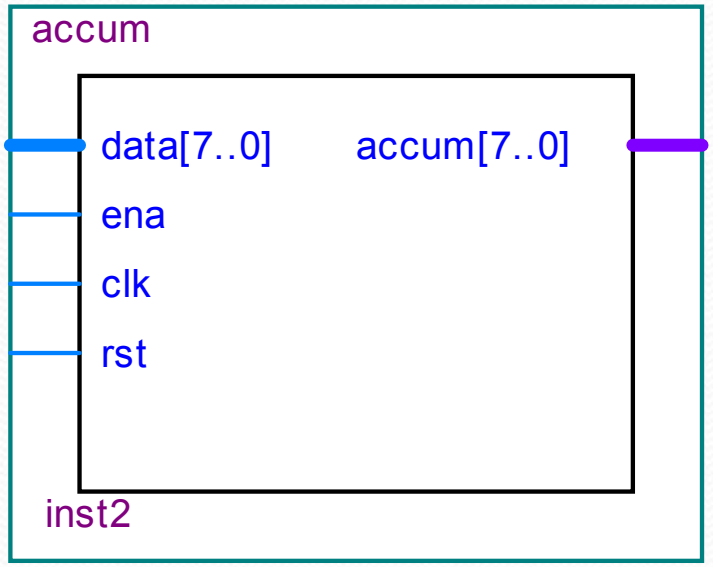
```
`timescale 1ns/1ns
module register(opc_iraddr,data,ena,clk,rst);
  output  [15:0]opc_iraddr;
  input   [7:0]data;
  input   ena,clk,rst;
  reg     [15:0]opc_iraddr;
  reg   state;
  always @(posedge clk)
    begin
      if (rst)
        begin
        opc_iraddr<=16'b0000_0000_0000_0000;
        state<=1'b0;
        end
      else
    if (ena)
        casex(state)
            1'b0:begin
                opc_iraddr[15:8]<=data;
                state<=1;
                end
             1'b1:begin
                opc_iraddr[7:0]<=data;
                state<=0;
                end
              default:begin
                opc_iraddr[15:0]<=16'bx;
                state<=1'bx;
                end
          endcase
      else       state<=1'b0;
   end
endmodule
```

累加器

alu_out[7:0]
load_acc
clk
reset



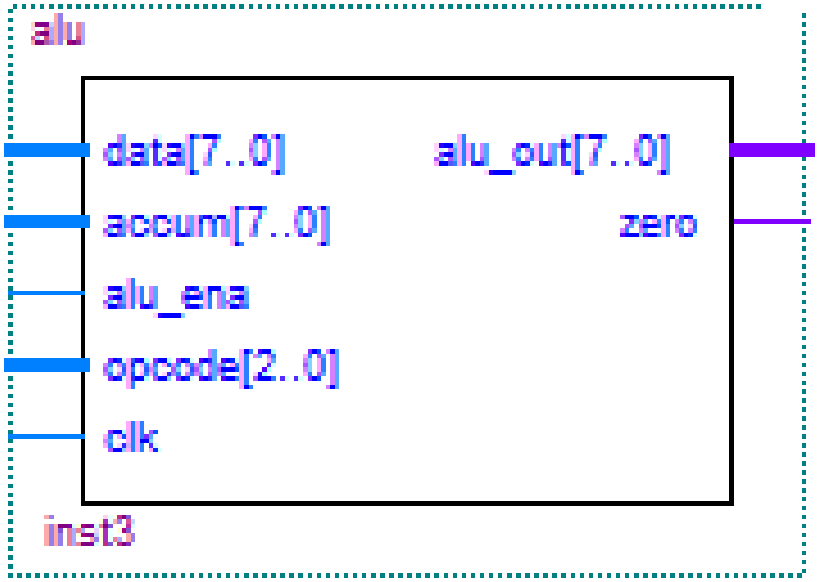


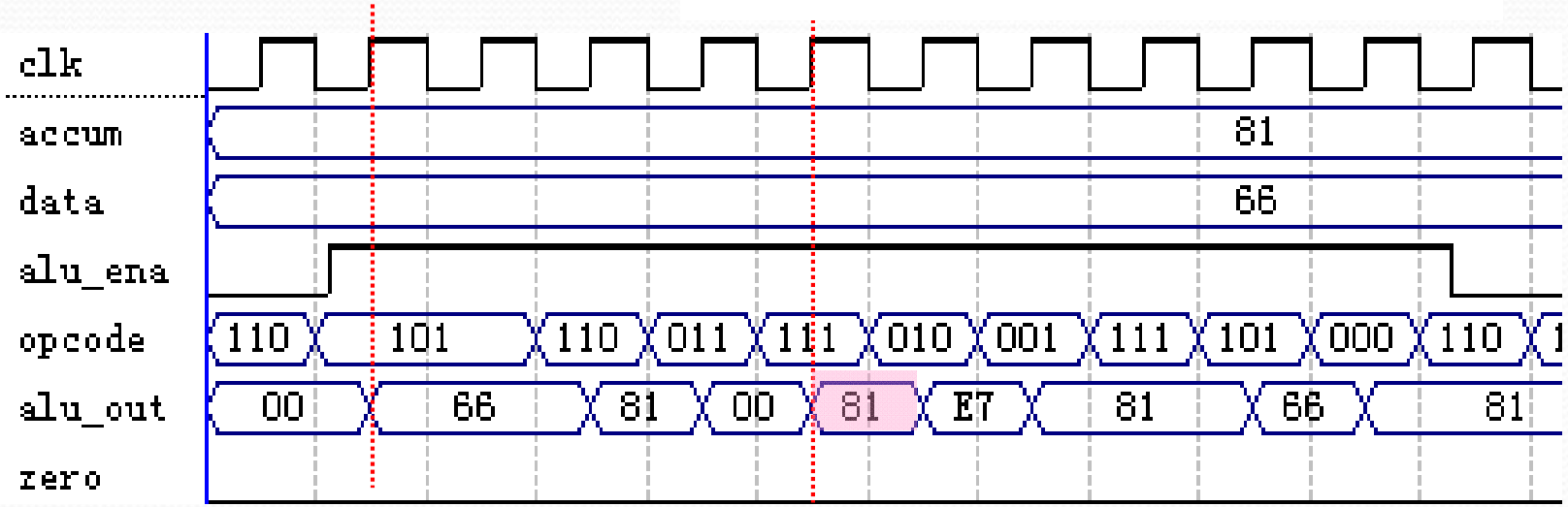



```
module accum(accum,data,ena,clk,rst);
  output  [7:0]accum;
  input   [7:0]data;
  input   ena,clk,rst;
  reg     [7:0]accum;
  always @(posedge clk)
    begin
      if(rst)
        accum<=8'b0000_0000;
      else
        if(ena)
          accum<=data;
    end
endmodule
```

算术逻辑运算器

助记符	操作码	功能
HLT	3'b000	暂停
SKZ	3‘b001	累加器为零转移
ADD	3’b010	加
ANDD	3‘b011	与
XORR	3‘b100	或
LDA	3‘b101	取数
STO	3‘b110	存数
JMP	3‘b111	转移



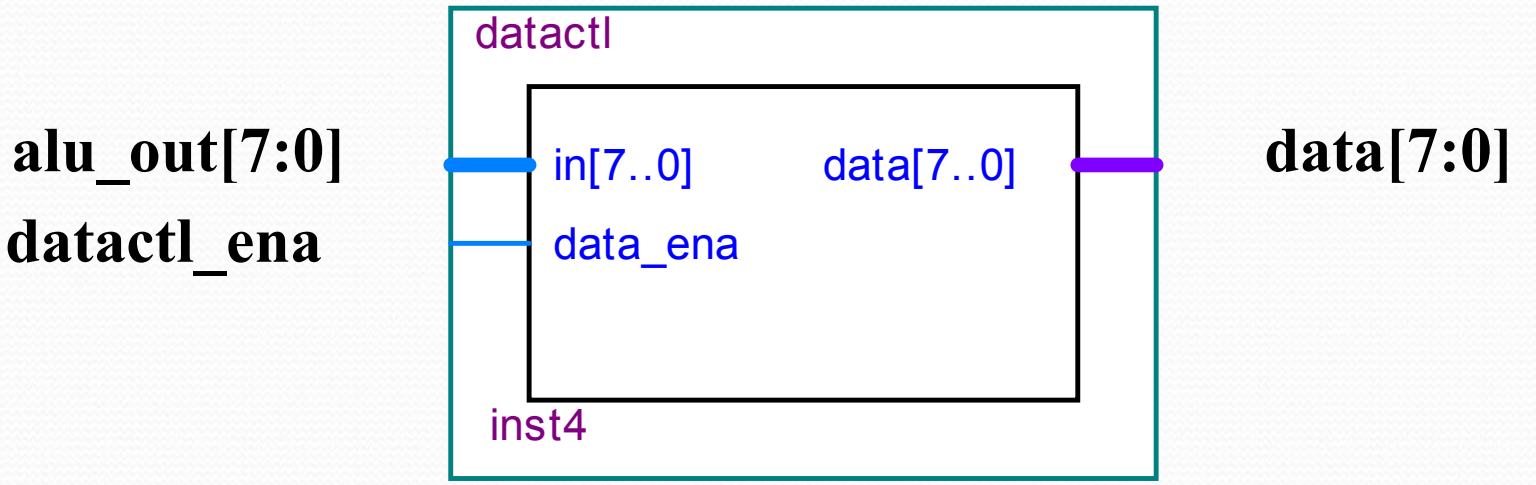


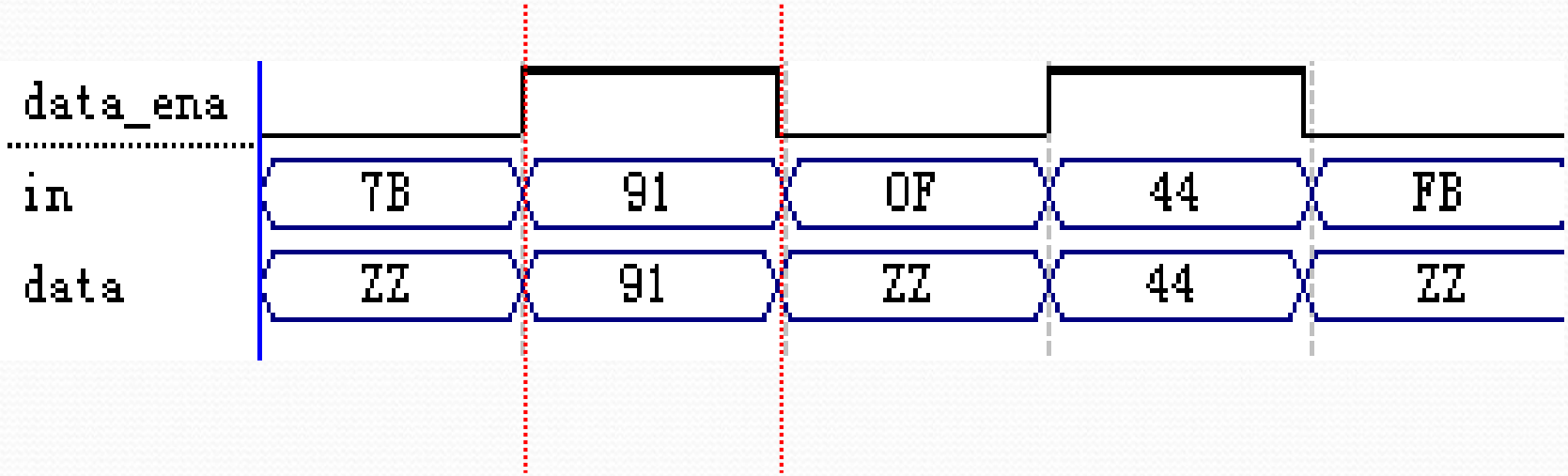
```
`timescale 1ns/1ns
module
    alu(alu_out,zero,data,accum,alu_ena,opcod
    e,clk);
  output  [7:0]alu_out;
  output  zero;
  input   [7:0]data,accum;
  input   [2:0]opcode;
  input   alu_ena,clk;
  reg     [7:0]alu_out;
  parameter HLT=3'b000,
            SKZ=3'b001,
            ADD=3'b010,
            ANDD=3'b011,
            XORR=3'b100,
            LDA=3'b101,
            STO=3'b110,
            JMP=3'b111;
```

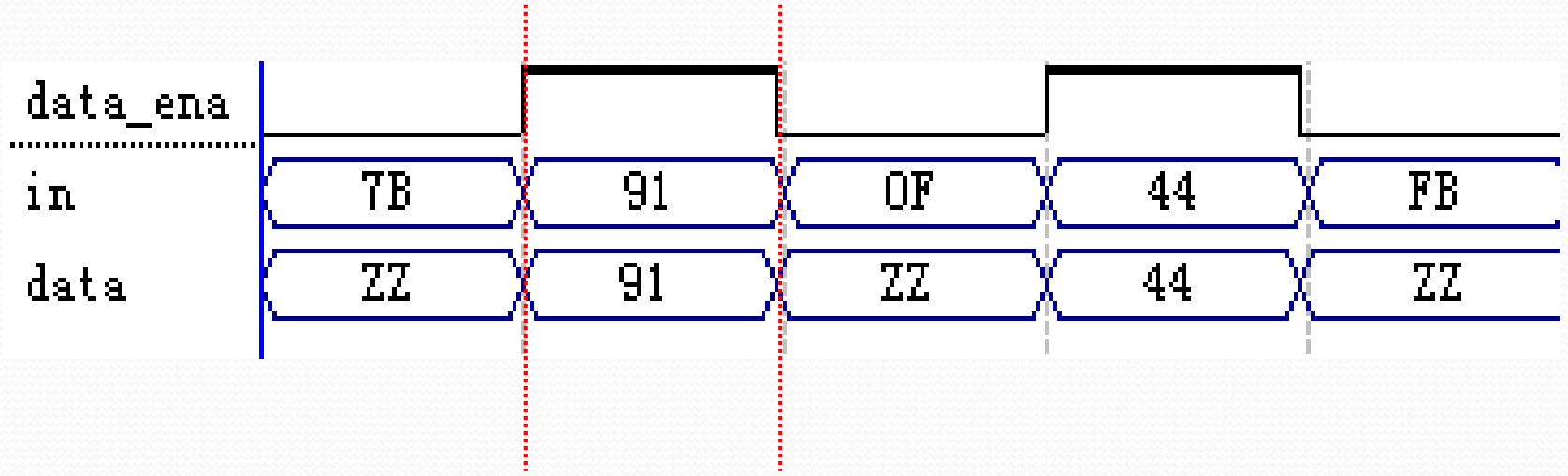
```
 assign  zero=!accum;
  always @(posedge clk)
    if(alu_ena)
      begin
        casex(opcode)
          HLT:alu_out<=accum;
          SKZ:alu_out<=accum;
          ADD:alu_out<=data+accum;
          ANDD:alu_out<=data&accum;
          XORR:alu_out<=data^accum;
          LDA:alu_out<=data;
          STO:alu_out<=accum;
          JMP:alu_out<=accum;
          default:alu_out<=8'bxxxx_xxxx;
        endcase
      end

endmodule
```

数据控制器

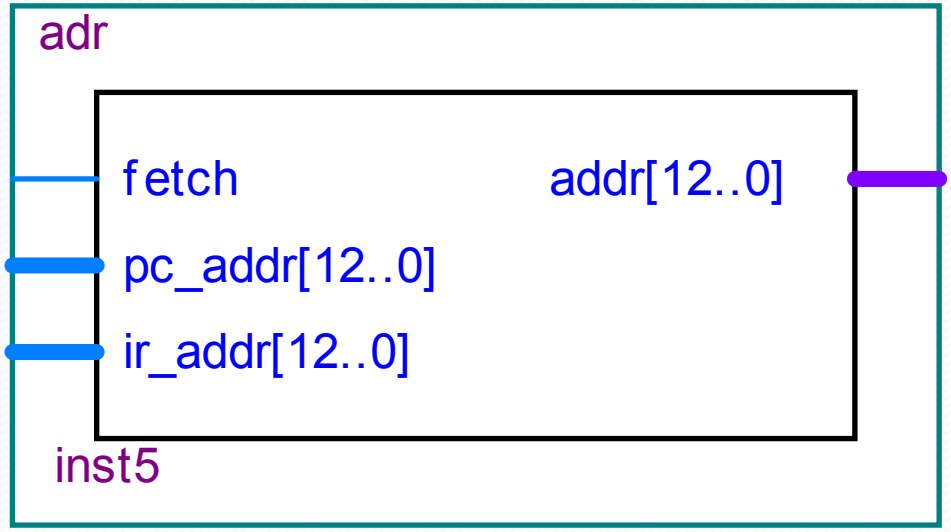

alu_out[7:0]
datactl_ena
datactl
in[7..0]
data_ena
data[7..0]
inst4
data[7:0]

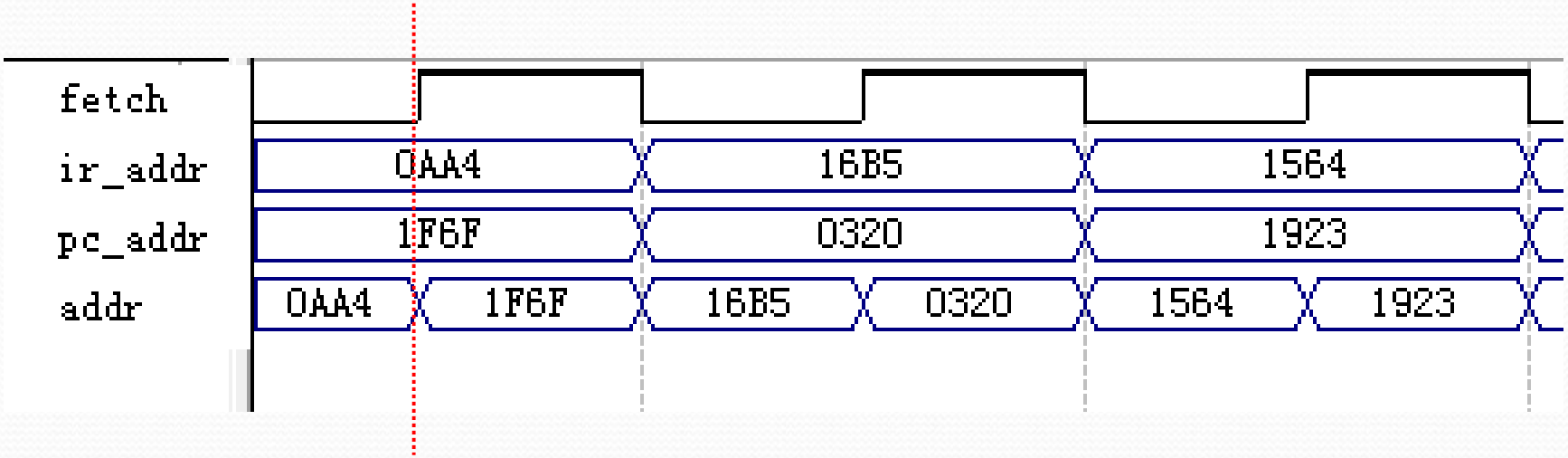

data_ena
in
data
7B
91
0F
44
FB
ZZ
91
ZZ
44
ZZ

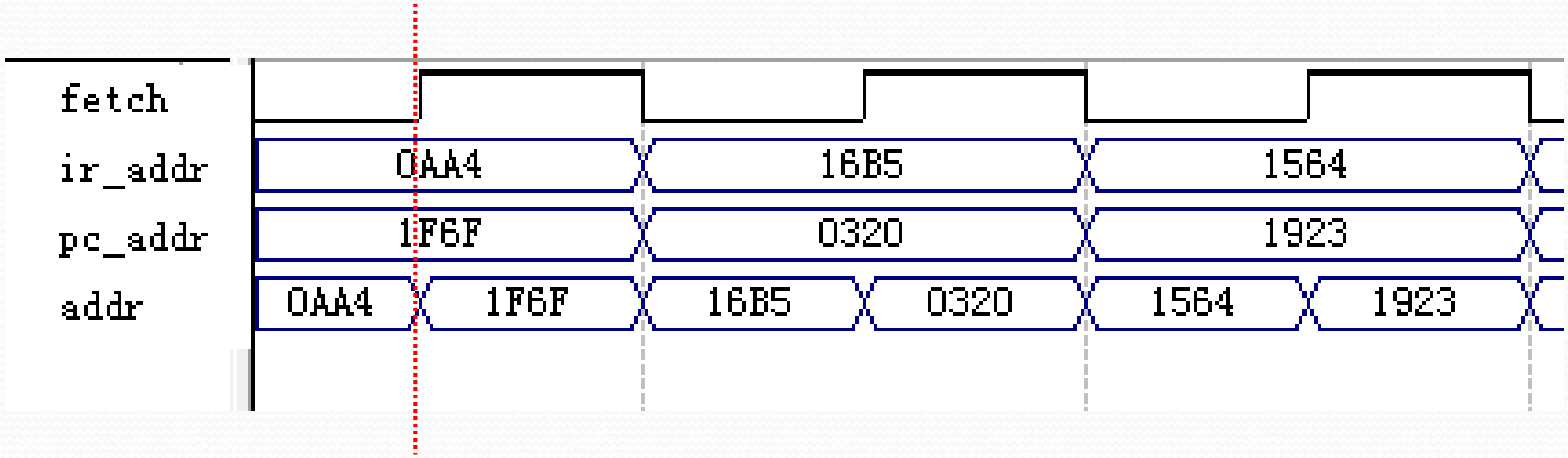



```
module datactl(data,in,data_ena);
  output  [7:0]data;
  input   [7:0]in;
  input   data_ena;
  assign  data=(data_ena)?in:8'bz;
endmodule
```

地址多路器

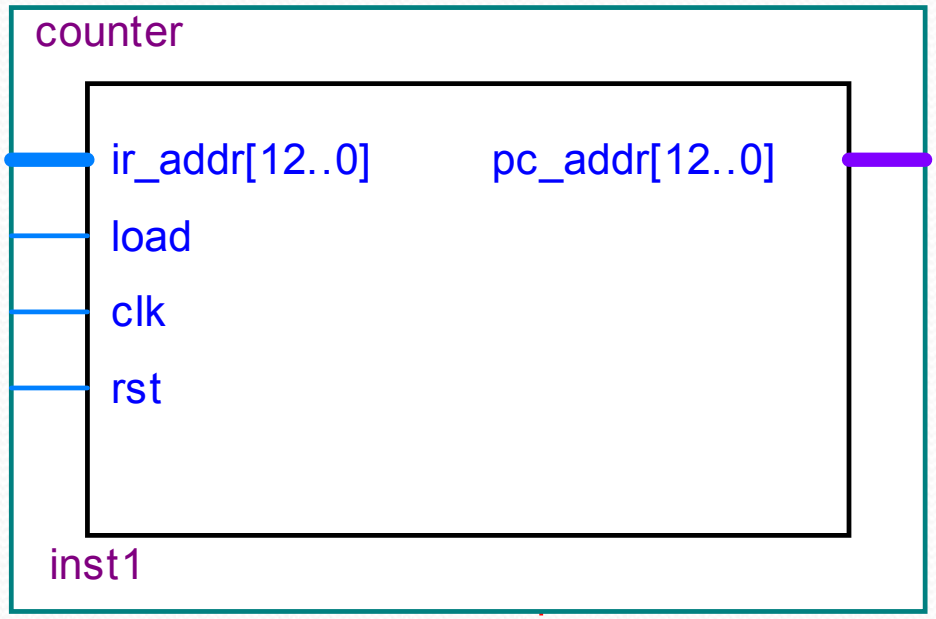

adr
fetch
addr[12..0]
pc_addr[12..0]
ir_addr[12..0]
inst5

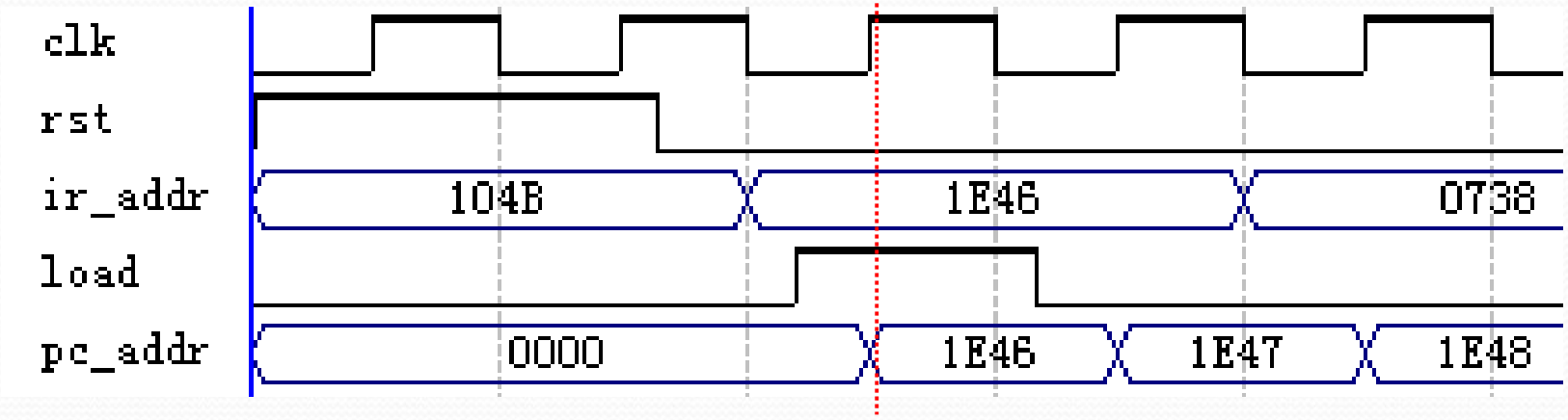

fetch
ir_addr
pc_addr
addr
0AA4
16B5
1564
1F6F
0320
1923
0AA4
1F6F
16B5
0320
1564
1923

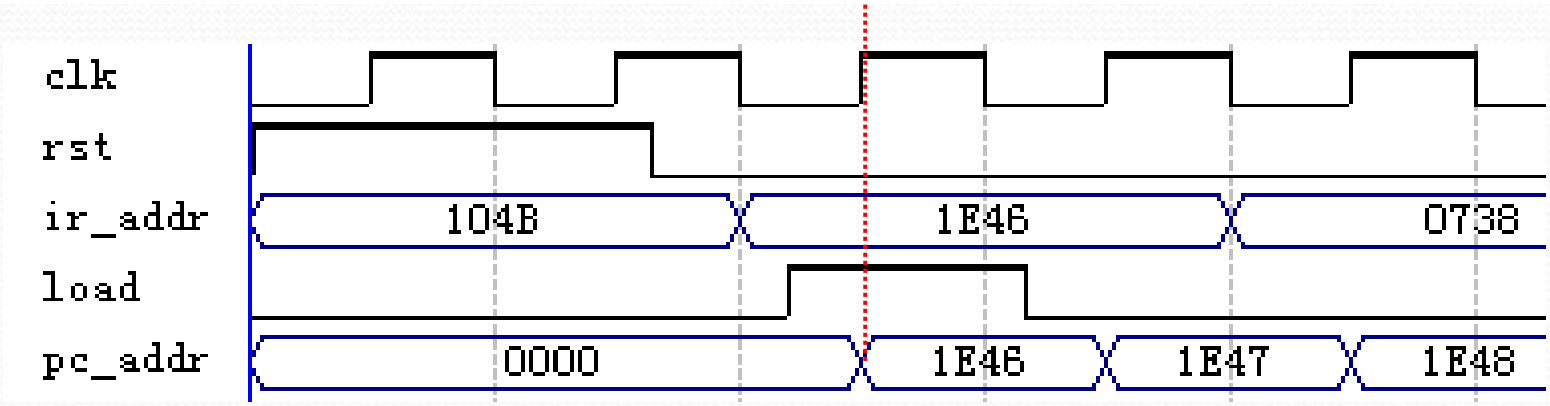



```
module adr(addr, fetch, pc_addr, ir_addr);
  input   fetch;
  input [12:0]pc_addr,ir_addr;
  output  [12:0]addr;
  assign  addr=fetch? pc_addr:ir_addr;
endmodule
```

程序计数器

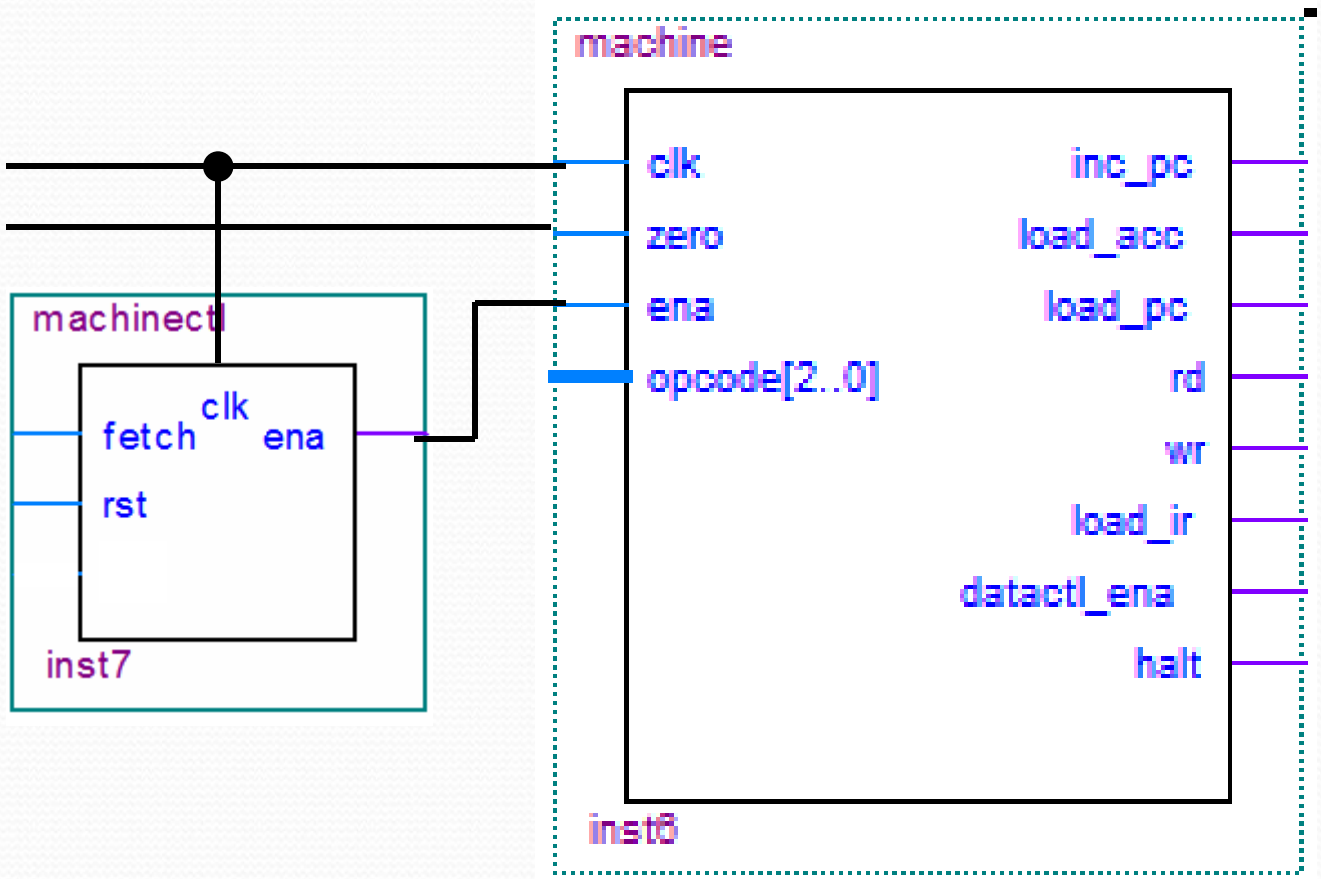

counter
ir_addr[12..0]
pc_addr[12..0]
load
clk
rst
inst1


clk
rst
ir_addr
load
pc_addr
104B
1E46
0738
0000
1E46
1E47
1E48




```
module counter(pc_addr,ir_addr,load,clk,rst);
  input   load,clk,rst;
  input [12:0]ir_addr;
  output  [12:0]pc_addr;
  reg [12:0]pc_addr;
  always @(posedge clk or posedge rst)
     if(rst)
       pc_addr<=13'b0_0000_0000_0000;
     else
       if(load)
         pc_addr<=ir_addr;
       else
         pc_addr<=pc_addr+1;
endmodule
```

状态控制器


machine
clk
zero
ena
opcode[2..0]
inc_pc
load_acc
load_pc
rd
wr
load_ir
datactl_ena
halt
inst6
machinectl
clk
fetch
ena
rst
inst7

状态控制器

状态控制器两部分：

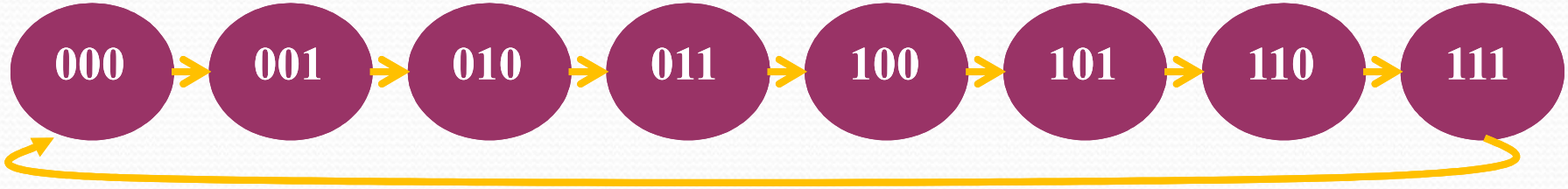
- 状态机（machine）
- 控制器machinectl

  rst有效，ena=0，

  状态机停止工作。

```
`timescale 1ns/1ns
module machinectl(ena,fetch,rst,clk);
  input fetch,rst,clk;
  output  ena;
  reg   ena;
  always @(posedge clk)
    begin
      if(rst)
        ena<=0;
      else
        if(fetch)
          ena<=1;
    end
endmodule
```

状态机



状态机是CPU的核心部件，用于产生控制信号：

- 启动、停止某些部件
- CPU执行读指令来读写接口、存储器
- 状态变量state的值，是指令周期已经过的时钟数

各指令周期完成的操作

第0个时钟：rd和load_ir为高电平，ROM中高八位指令代码-->指令寄存器

第1个时钟：rd和load_ir为高电平，inc_pc加1，故PC+1，ROM中低八位指令代码-->指令寄存器

第2个时钟：空操作

第3个时钟：PC+1，指向下一条指令；若HLT指令，halt=1；其它指令控制线输出为0

各指令周期完成的操作

第4个时钟：若AND ADD XOR LDA，读相应地址数据；若JMP，目的地址-->PC；若STO，输出累加器数据。

第5个时钟：若AND ADD XOR，进行相应运算；若LDA，数据-->运算器-->累加器;若JMP，锁存目的地址；若STO，数据到地址处。

第6个时钟：空操作

第7个时钟：若SKZ指令， PC+1跳过一条指令；否则PC无变化

```
`timescale 1ns/1ns
Module
  machine(inc_pc,load_acc,load_pc,rd,wr,load_ir,datactl_ena,halt,clk,zero,ena,opcode);
  output  inc_pc,load_acc,load_pc,rd,wr,load_ir;
  output  datactl_ena,halt;
  input clk,zero,ena;
  input [2:0]opcode;
  reg inc_pc,load_acc,load_pc,rd,wr,load_ir;
  reg datactl_ena,halt;
  reg [2:0]state;
  parameter HLT =3'B000,
            SKZ =3'b001,
            ADD =3'b010,
            ANDD=3'b011,
            XORR=3'b100,
            LDA =3'b101,
            STO =3'b110,
            JMP =3'b111;
```

```
always @(negedge clk)
    begin
      if(!ena)
        begin
          state<=3'b000;
          {inc_pc,load_acc,load_pc,rd}<=4'b0000;
          {wr,load_ir,datactl_ena,halt}<=4'b0000;
        end
      else
          ctl_cycle;
    end
```

```
//-------begin of task ctl_cycle-------
   task  ctl_cycle;
     begin
       casex(state)
         3'b000: begin
             {inc_pc,load_acc,load_pc,rd}<=4'b0001;
             {wr,load_ir,datactl_ena,halt}<=4'b0100;
             state<=3'b001;
           end
         3'b001: begin
             {inc_pc,load_acc,load_pc,rd}<=4'b1001;
             {wr,load_ir,datactl_ena,halt}<=4'b0100;
             state<=3'b010;
           end
        3'b010: begin
             {inc_pc,load_acc,load_pc,rd}<=4'b0000;
             {wr,load_ir,datactl_ena,halt}<=4'b0000;
             state<=3'b011;
           end
```

```
3'b011:
        begin
          if(opcode==HLT)
            begin
              {inc_pc,load_acc,load_pc,rd}<=4'b1000;
              {wr,load_ir,datactl_ena,halt}<=4'b0001;
              state<=3'b100;
            end
          else
            begin
              {inc_pc,load_acc,load_pc,rd}<=4'b1000;
              {wr,load_ir,datactl_ena,halt}<=4'b0000;
              state<=3'b100;
            end
        end
```

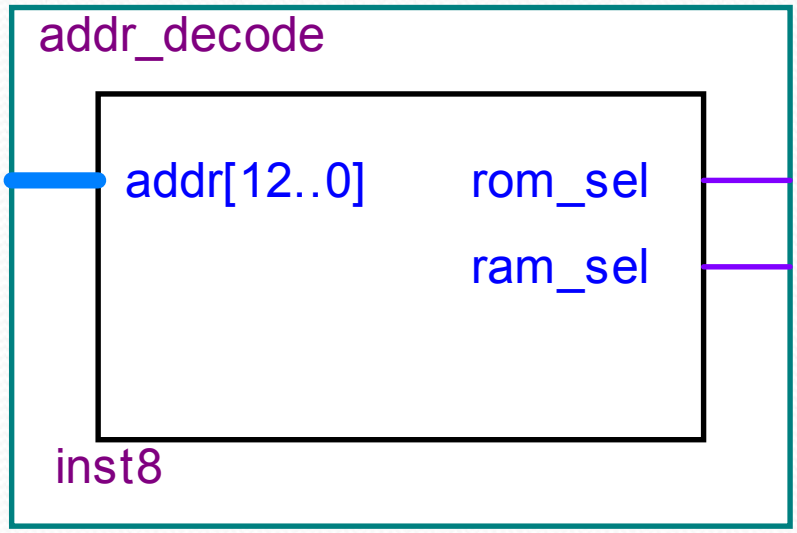
```
3'b100:  begin
          if(opcode==JMP)
            begin
              {inc_pc,load_acc,load_pc,rd}<=4'b0010;
              {wr,load_ir,datactl_ena,halt}<=4'b0000;              end
          else if(opcode==ADD||opcode==ANDD||opcode==XORR||opcode==LDA)
            begin
              {inc_pc,load_acc,load_pc,rd}<=4'b0001;
              {wr,load_ir,datactl_ena,halt}<=4'b0000;              end
          else if(opcode==STO)
            begin
              {inc_pc,load_acc,load_pc,rd}<=4'b0000;
              {wr,load_ir,datactl_ena,halt}<=4'b0010;              end
          else
            begin
              {inc_pc,load_acc,load_pc,rd}<=4'b0000;
              {wr,load_ir,datactl_ena,halt}<=4'b0000;              end
          state<=3'b101;
    end
```

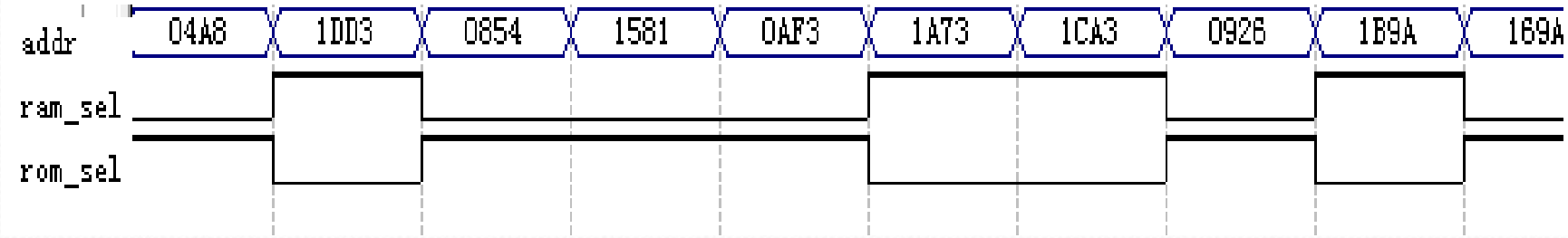
```
3'b101: begin          //operation
        if(opcode==ADD||opcode==ANDD||opcode==XORR||opcode==LDA)
          begin
            {inc_pc,load_acc,load_pc,rd}<=4'b0101;
            {wr,load_ir,datactl_ena,halt}<=4'b0000;              end
        else if(opcode==SKZ && zero==1)
          begin
            {inc_pc,load_acc,load_pc,rd}<=4'b1000;
            {wr,load_ir,datactl_ena,halt}<=4'b0000;               end
        else if(opcode==JMP)
          begin
            {inc_pc,load_acc,load_pc,rd}<=4'b1010;
            {wr,load_ir,datactl_ena,halt}<=4'b0000;              end
        else if(opcode==STO)
          begin
            {inc_pc,load_acc,load_pc,rd}<=4'b0000;
            {wr,load_ir,datactl_ena,halt}<=4'b1010;              end
        else
          begin
            {inc_pc,load_acc,load_pc,rd}<=4'b0000;
            {wr,load_ir,datactl_ena,halt}<=4'b0000;              end
        state<=3'b110;
      end
```

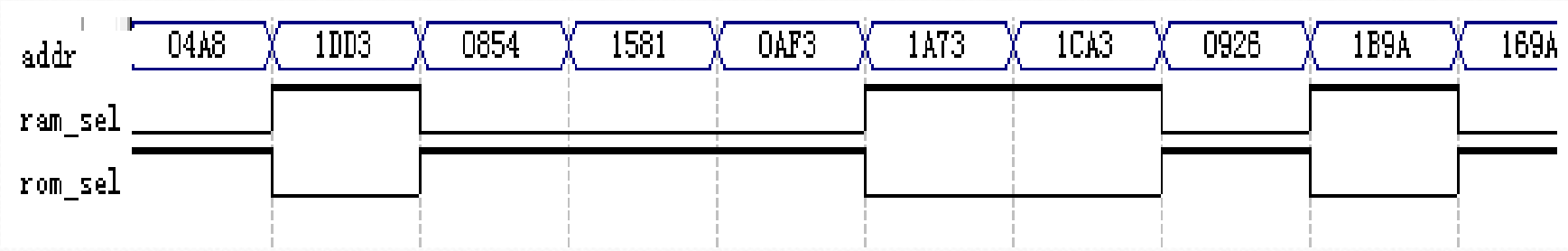
```
3'b110://idle
        begin
          if(opcode==STO)
            begin
              {inc_pc,load_acc,load_pc,rd}<=4'b0000;
              {wr,load_ir,datactl_ena,halt}<=4'b0010;
            end
          else
if(opcode==ADD||opcode==ANDD||opcode==XORR||opcode==LDA)
            begin
              {inc_pc,load_acc,load_pc,rd}<=4'b0001;
              {wr,load_ir,datactl_ena,halt}<=4'b0000;
            end
          else
            begin
              {inc_pc,load_acc,load_pc,rd}<=4'b0000;
              {wr,load_ir,datactl_ena,halt}<=4'b0000;
            end
          state<=3'b111;
        end
```

```
    3'b111:  begin
             if(opcode==SKZ && zero==1)
               begin
                 {inc_pc,load_acc,load_pc,rd}<=4'b1000;
                 {wr,load_ir,datactl_ena,halt}<=4'b0000;                end
             else
               begin
                 {inc_pc,load_acc,load_pc,rd}<=4'b0000;
                 {wr,load_ir,datactl_ena,halt}<=4'b0000;                end
             state<=3'b000;
            end
          default:  begin
                 {inc_pc,load_acc,load_pc,rd}<=4'b0000;
                 {wr,load_ir,datactl_ena,halt}<=4'b0000;
                 state<=3'b000;                                                 end
        endcase
      end
    endtask
  //----------end of task ctl_cycle----------
endmodule
```

地址译码器


addr_decode
addr[12..0]
rom_sel
ram_sel
inst8


addr
ram_sel
rom_sel
04A8
1DD3
0854
1581
0AF3
1A73
1CA3
0926
1B9A
169A




```
module  addr_decode(addr,rom_sel,ram_sel);
  output  rom_sel,ram_sel;
  input   [12:0]addr;
  reg     rom_sel,ram_sel;
  always @(addr)
    casex(addr)
      13'b1_1xxx_xxxx_xxxx:{rom_sel,ram_sel}<=2'b01;
      13'b0_xxxx_xxxx_xxxx:{rom_sel,ram_sel}<=2'b10;
      13'b1_0xxx_xxxx_xxxx:{rom_sel,ram_sel}<=2'b10;
      default:{rom_sel,ram_sel}<=2'b00;
    endcase
endmodule
```

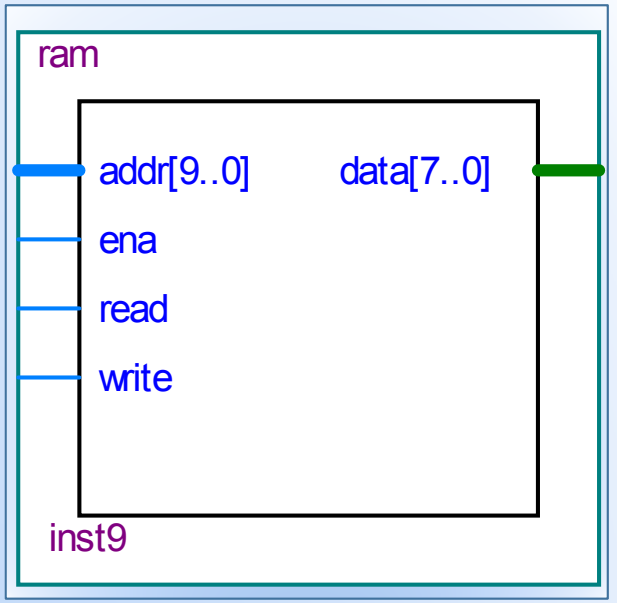
地址译码器用于产生选通信号，选通ROM或RAM。

FFFFH---1800H RAM

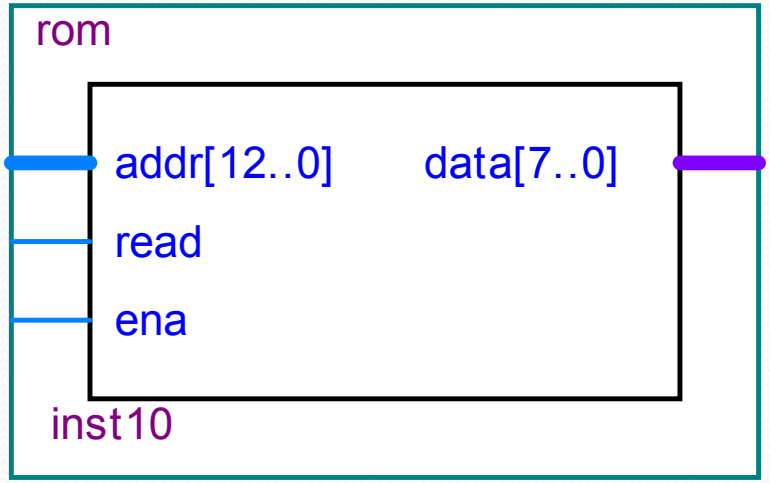
1800H---0000H ROM

RAM

```
module ram(data,addr,read,write,ena);
  output [7:0]data;
  input [12:0]addr;
  input ena,read,write;
  reg [7:0]mem[13'h1fff:0];
  if (read&&ena)
  begin
     assign data=mem[addr];      end
  else if (write&& ena)
  begin
     mem[addr] <= data;    end
  else
  begin
     assign data=8'hzz;     end
endmodule
```



ROM




```
module rom(data,addr,read,ena);
  output [7:0]data;
  input [12:0]addr;
  input ena,read;
  reg [7:0]mem[13'h1fff:0];
  assign  data=(read&&ena)?mem[addr]:8'hzz;
endmodule
```

CPU设计案例解析

- 1 CPU概述
- 2 简单CPU设计方法
- 3 功能模块设计
- 4 系统调试

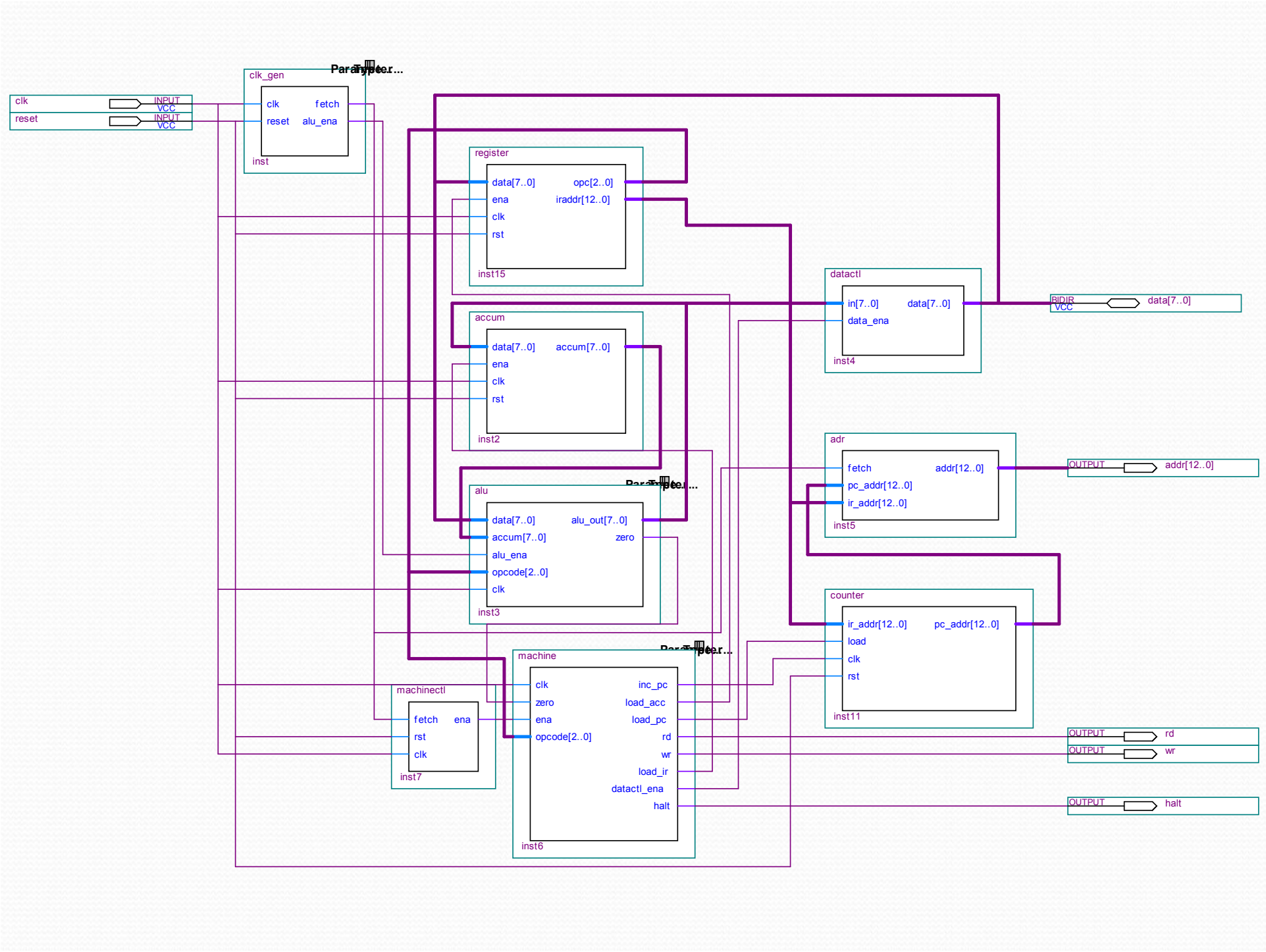
系统复位与启动

复位状态reset=1：

- CPU内部寄存器全为零
- 数据总线为高阻状态
- 地址总线为0000H
- 所有控制信号为无效状态

启动状态reset=0：

- fetch上升沿启动CPU开始工作
- 从ROM的000处读指令并执行


clk
reset
INPUT
VCC
clk_gen
clk
reset
fetch
alu_ena
inst
register
data[7..0]
ena
clk
rst
opc[2..0]
iraddr[12..0]
inst15
accum
data[7..0]
ena
clk
rst
accum[7..0]
inst2
alu
data[7..0]
accum[7..0]
alu_ena
opcode[2..0]
clk
alu_out[7..0]
zero
inst3
machinectl
fetch
rst
clk
ena
inst7
machine
clk
zero
ena
opcode[2..0]
inc_pc
load_acc
load_pc
rd
wr
load_ir
datactl_ena
halt
inst6
datactl
in[7..0]
data_ena
data[7..0]
inst4
BIDIR
VCC
data[7..0]
adr
fetch
pc_addr[12..0]
ir_addr[12..0]
addr[12..0]
inst5
OUTPUT
addr[12..0]
counter
ir_addr[12..0]
load
clk
rst
pc_addr[12..0]
inst11
OUTPUT
rd
OUTPUT
wr
OUTPUT
halt

ROM

```
@00
111_00000     //00 BEGIN:    JMP TEST_JMP
0011_1100
000_00000     //02           HLT
0000_0000
000_00000     //04           HLT
0000_0000
101_11000     //06 JMP_OK:   LDA DATA_1
0000_0000
001_00000     //08           SKZ
0000_0000
000_00000     //0A           HLT
0000_0000
101_11000     //0C           LDA DATA_2
0000_0001
001_00000     //0E           SKZ
0000_0000
111_00000     //10           JMP SKZ_OK
0001_0100
000_00000     //12           HLT
0000_0000
110_11000     //14 SKZ_OK:   STO TEMP
0000_0010
101_11000     //16           LDA DTAT_1
0000_0000
110_11000     //18           STO TEMP
0000_0010
101_11000     //1A           LDA TEMP
0000_0010
```

```
001_00000     //1C           SKZ
0000_0000
000_00000     //1E           HLT
0000_0000
100_11000     //20           XORR DTAT_2
0000_0001
001_00000     //22           SKZ
0000_0000
000_00000     //24           HLT
0000_0000
111_00000     //26           JMP XORR_OK
0010_0100
100_11000     //28 XORR_OK:  XORR DATA_2
0000_0001
001_00000     //2A           SKZ
0000_0000
000_00000     //2C           HLT
0000_0000
000_00000     //2E EDN:      HLT
0000_0000
111_00000     //30           JMP BEGIN
0000_0000
@3C
111_00000     //3C TST_JMP:  JMP JMP_OK
0000_0110
000_00000     //3E           HLT
```

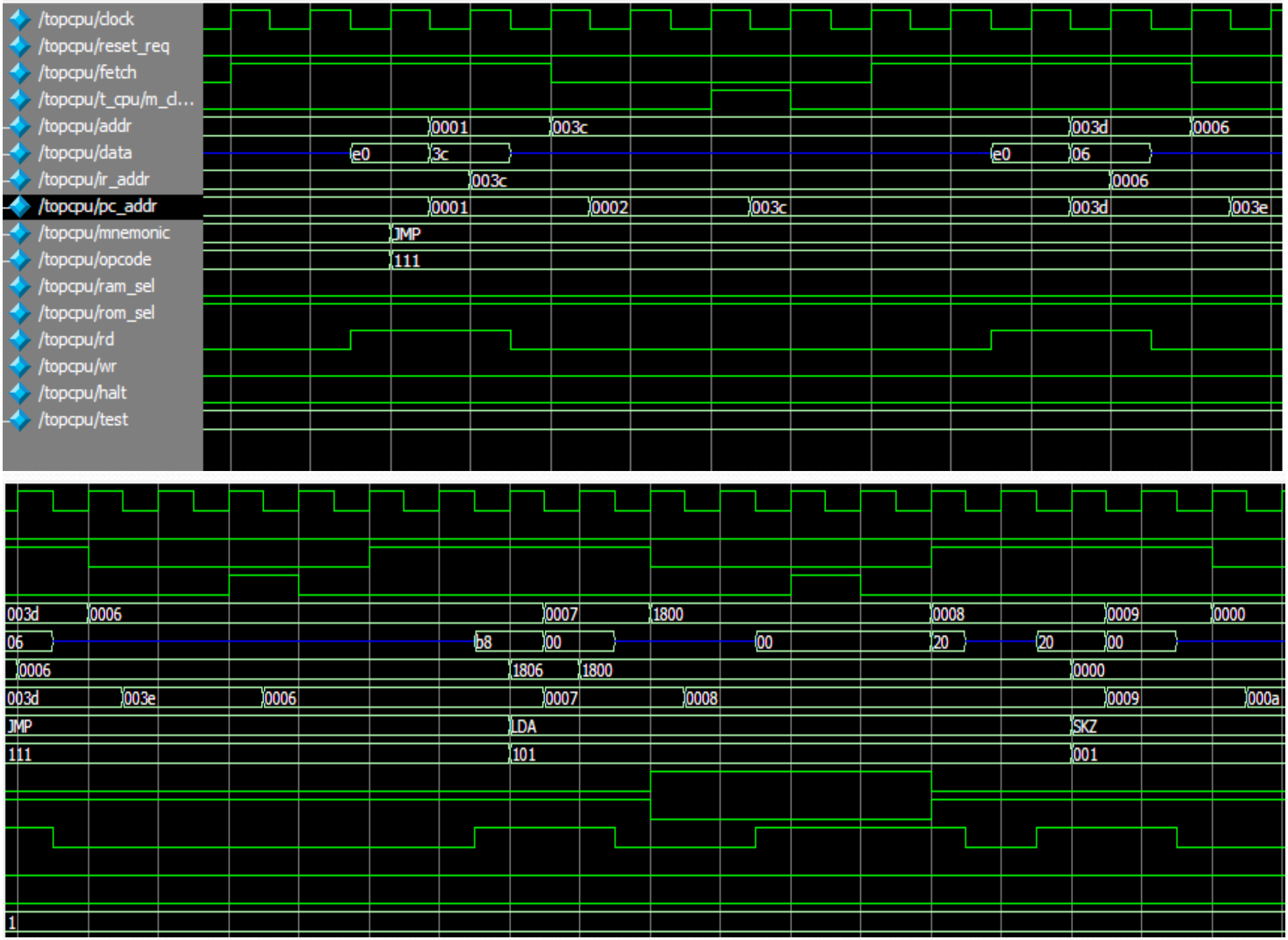
RAM

```
@00
      00000000       //1800  DATA_1
      11111111       //1801  DATA_2
      10101010       //1802  TEMP
```

```
`include "clk_gen.v"
`include "register.v"
`include "accum.v"
`include "adr.v"
`include "alu.v"
`include "machine.v"
`include "machinectl.v"
`include "counter.v"
`include "datactl.v"
`timescale  1ns/1ns
module  cpu(clk,reset,halt,rd,wr,addr,data,opcode,fetch,ir_addr,pc_addr);
  input clk,reset;
  output  rd,wr,halt;
  output  [12:0]addr;
  output  [2:0]opcode;
  output  fetch;
  output  [12:0]ir_addr,pc_addr;
  inout [7:0]data;
  wire  clk,reset,halt;
  wire  [7:0]data;
  wire  [12:0]addr;
  wire  rd,wr;
  wire  fetch,alu_ena;
  wire  [2:0]opcode;
  wire  [12:0]ir_addr,pc_addr;
  wire  [7:0]alu_out,accum;
  wire  zero,inc_pc,load_acc,load_pc,load_ir,data_ena,contrl_ena;
```

```
  ……
  clk_gen m_clkgen(.clk(clk),.reset(reset),.fetch(fetch),.alu_ena(alu_ena));
  register m_register(.data(data),.ena(load_ir),.rst(reset),.clk(clk),.opc_iraddr({opcode,ir_addr}));
  accum  m_accum(.data(alu_out),.ena(load_acc),.clk(clk),.rst(reset),.accum(accum));
  alu
m_alu(.data(data),.accum(accum),.clk(clk),.alu_ena(alu_ena),.opcode(opcode),.alu_out(alu_out),.zero(zero));
  machinectl  m_machinectl(.clk(clk),.rst(reset),.fetch(fetch),.ena(contrl_ena));
  machine     m_machine(.inc_pc(inc_pc),.load_acc(load_acc),.load_pc(load_pc),.rd(rd),.wr(wr),.load_ir(load_ir),
              .clk(clk),.datactl_ena(data_ena),.halt(halt),.zero(zero),.ena(contrl_ena),.opcode(opcode));
  datactl     m_datactl(.in(alu_out),.data_ena(data_ena),.data(data));
  adr         m_adr(.fetch(fetch),.ir_addr(ir_addr),.pc_addr(pc_addr),.addr(addr));
  counter     m_counter(.clk(inc_pc),.rst(reset),.ir_addr(ir_addr),.load(load_pc),.pc_addr(pc_addr));

Endmodule
```


/topcpu/clock
/topcpu/reset_req
/topcpu/fetch
/topcpu/t_cpu/m_cl...
/topcpu/addr
/topcpu/data
/topcpu/ir_addr
/topcpu/pc_addr
/topcpu/mnemonic
/topcpu/opcode
/topcpu/ram_sel
/topcpu/rom_sel
/topcpu/rd
/topcpu/wr
/topcpu/halt
/topcpu/test
0001
003c
003d
0006
e0
3c
e0
06
003c
0006
0001
0002
003c
003d
003e
JMP
111
003d
0006
0007
1800
0008
0009
0000
06
b8
00
00
20
20
00
0006
1806
1800
0000
003d
003e
0006
0007
0008
0009
000a
JMP
LDA
SKZ
111
101
001
1

谢谢！