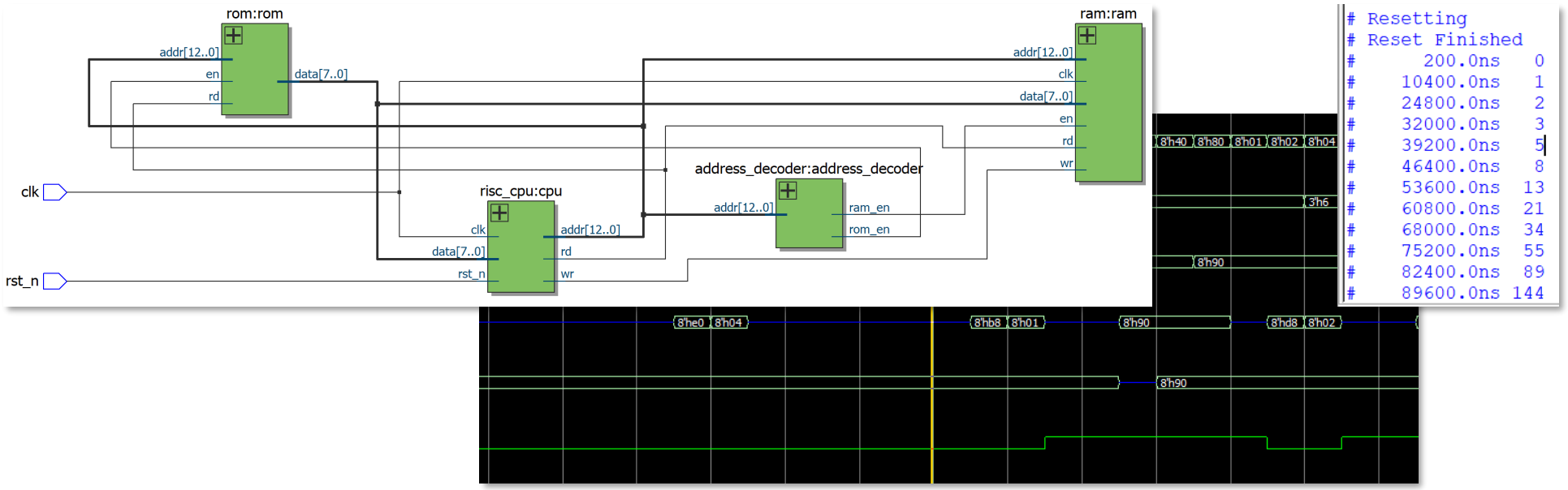
简易精简指令集CPU

Simple Reduced Instruction Set CPU



<你的名字>

2024年秋季

北京郵電大學
Beijing University of Posts and Telecommunications

简易精简指令集CPU

1. **相关概念**

2. 模块设计

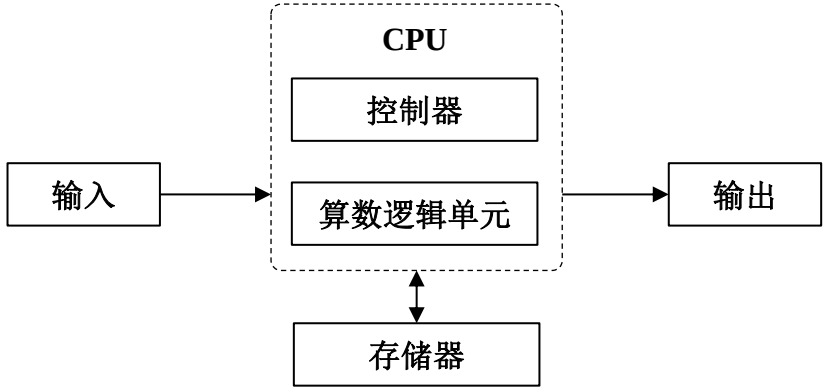
3. 仿真分析

相关概念

➢ 计算机体系结构 (Computer Architecture)

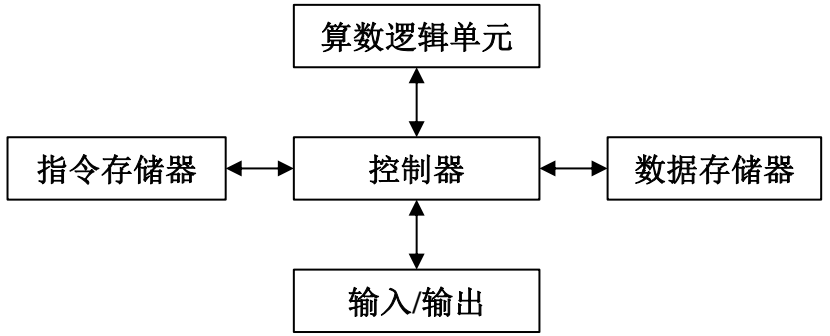
冯·诺依曼体系结构

- ➢ 共享存储器
- ➢ 顺序执行
- ➢ 单总线结构



哈佛体系结构

- ➢ 指令与数据分开存储
- ➢ 并行处理
- ➢ 非顺序执行



相关概念

- **指令集**（或称：指令集架构，Instruction Set Architecture，ISA）

  指令——计算机处理器执行的基本操作，**指令集架构**——处理器所能支持所有指令的集合

  复杂指令集计算机（CISC），**精简指令集**计算机（RISC）

	CISC	RISC
指令系统	指令**种类丰富**，功能复杂	指令**种类较少**，功能简单，复杂功能通过组合指令完成
程序	**因一条指令可完成多项任务**，程序通常较小	**因需要更多指令完成同样功能**，程序通常较大
中断	支持复杂的中断处理机制，中断处理时间较长	中断处理相对简单，响应时间相对较短
CPU特点	设计周期长，规模与功耗均较大	设计周期短，规模与功耗均较小
应用范围	多用于通用机（PC等）	多用于小型电子设备（手机等）及专用机

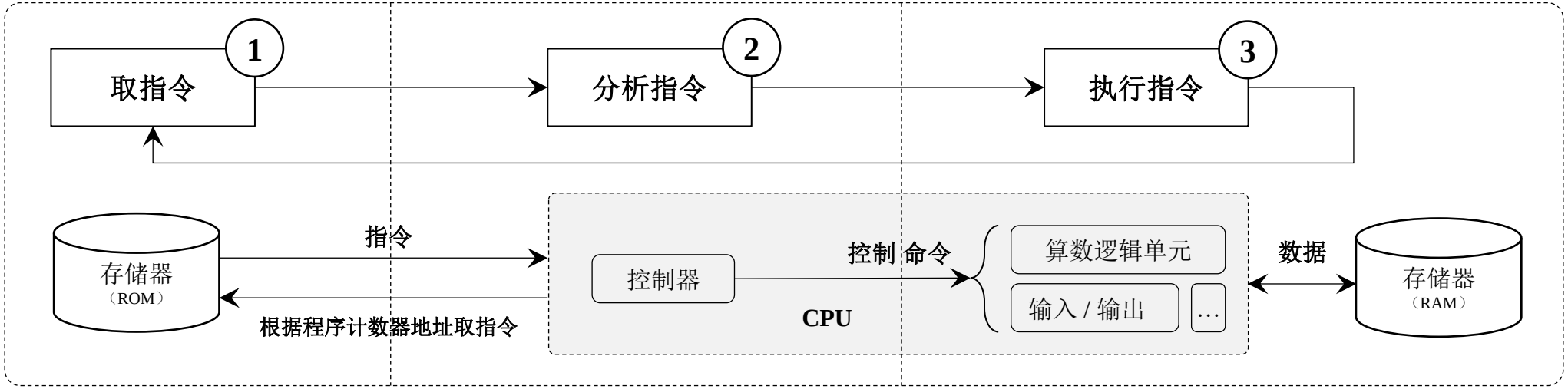
相关概念

➢ CPU的基本功能（最简单的功能：取指-译码-执行）

（1）取指令：根据程序计数器地址，取出存储器中相应位置的指令

（2）分析指令：分析取得的指令，得到操作码和指令地址（或立即数）

（3）执行指令：按照操作数，生成相应的控制信号序列，控制各模块有序工作，实现指令功能



相关概念

- **实现CPU基本功能，我们需要：**（时序电路，必有时钟、复位信号）
  - **程序计数器**（Program Counter，PC）

    提供指令地址
  - **指令寄存器**（Instruction Register，IR）

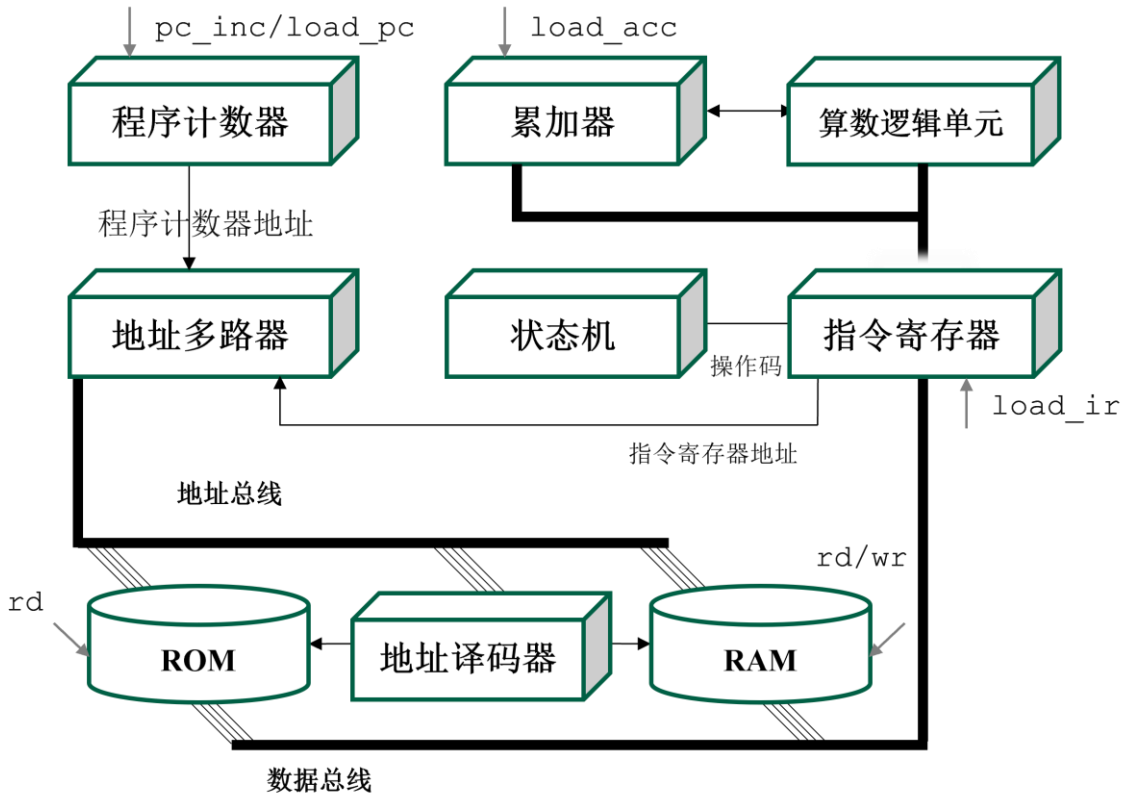
    寄存当前指令
  - **算数逻辑单元**（Arithmetic Logic Unit，ALU）

    在执行指令时，进行基本的算数运算、逻辑操作
  - **累加器**（Accumulator）

    存放当前指令执行的结果
  - **控制器/状态机**（Finite State Machine）

    指令译码，并产生相应的控制信号
  - **总线**（Bus）

    寻址（地址总线）、传输数据（数据总线）



简易精简指令集CPU

1. 相关概念

2. 模块设计

3. 仿真分析

模块设计

➢ 简易精简指令集CPU参数

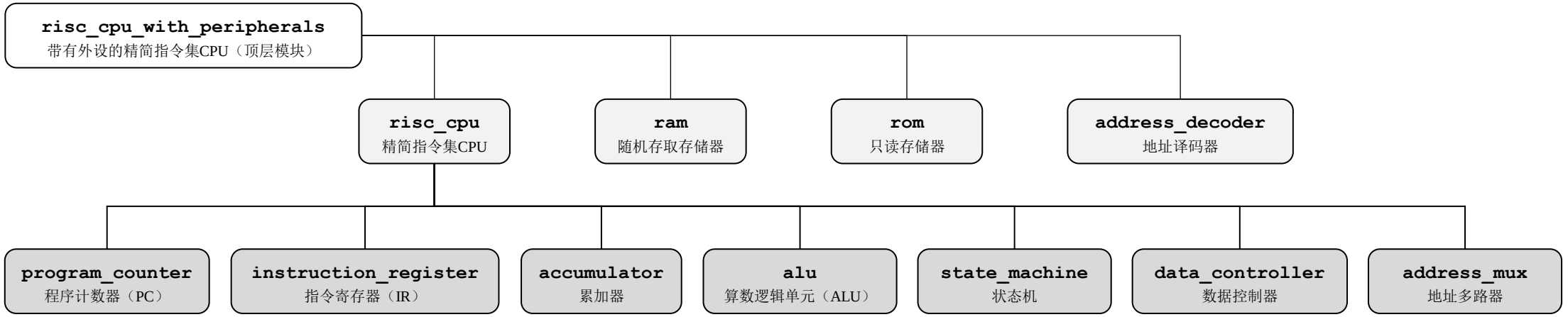
架构：8位，精简指令集架构

寻址：直接寻址，指令低13位为数据在存储器中的地址，寻址空间为8Kbits（2 ^ 13 = 8192）

指令：每条指令长16位，支持8条指令，8个时钟周期——1个指令周期

➢ 简易精简指令集CPU模块

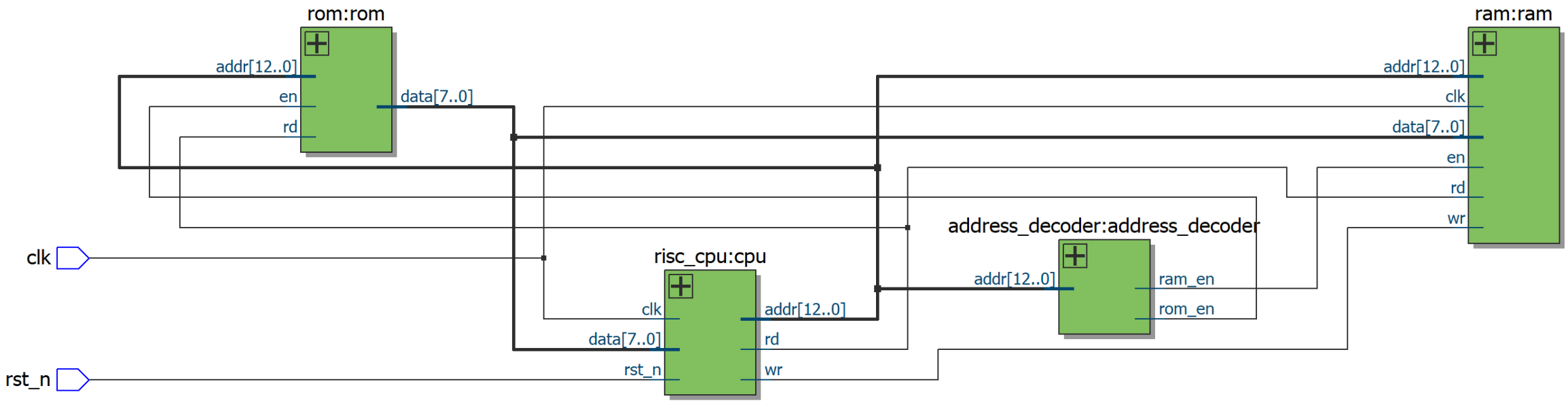
实现CPU功能的`risc_cpu`模块包含7个基本模块



模块设计

➢ 带有外设的精简指令集CPU模块（risc_cpu_with_peripherals）

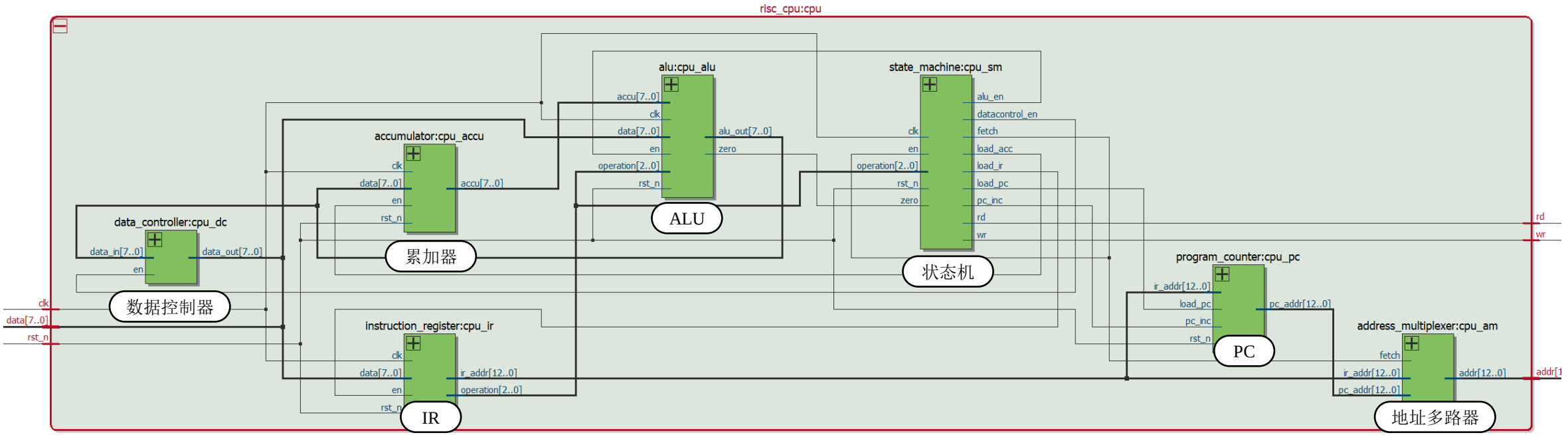
由CPU顶层模块、RAM、ROM与地址译码器构成，只对外暴露时钟（clk）、复位（rst_n）两个输入端口



模块设计

➢ CPU顶层模块 (risc_cpu)

由PC、IR、累加器、ALU、状态机、数据控制器、地址多路器构成，实现RISC CPU基本功能（取指-译码-执行）

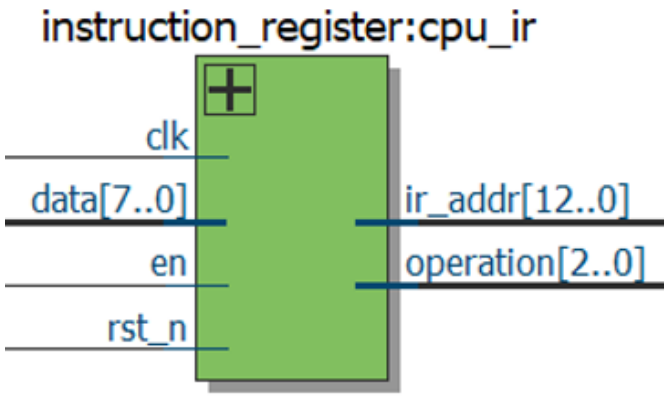


模块设计

➢ 指令寄存器模块 (instruction_register)

将数据总线送来的指令存入其内部的寄存器，并将指令拆分为操作码与指令地址 / 立即数

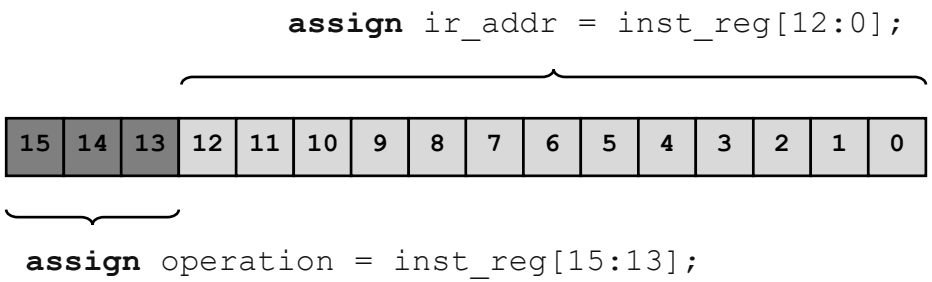
➢ 指令结构



每条指令共16位，高3位为操作码，低13位为地址

由于数据总线为8bit，每条指令需要取两次

取回指令后，拆分[15:13]为操作数，[12:0]为地址 / 立即数

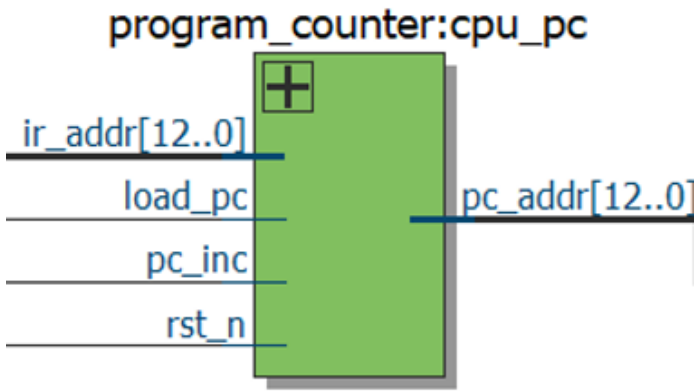


模块设计

➢ 程序计数器模块（program_counter）

维持指令地址的正确变化，并提供指令地址，CPU由此访问存储器的相应地址，读取按顺序存放的指令

➢ 程序计数器地址（pc_addr）的累加



➢ 正常情况下

pc_addr在一个指令周期中增加2，表示指向2个字节后的下一条指令

➢ 无条件跳转指令（JMP）

pc_addr将更新为JMP指令中的目标地址

➢ 为零跳过指令（SKZ）

若算数逻辑运算器输出为0，pc_addr将在一个指令周期中增加4

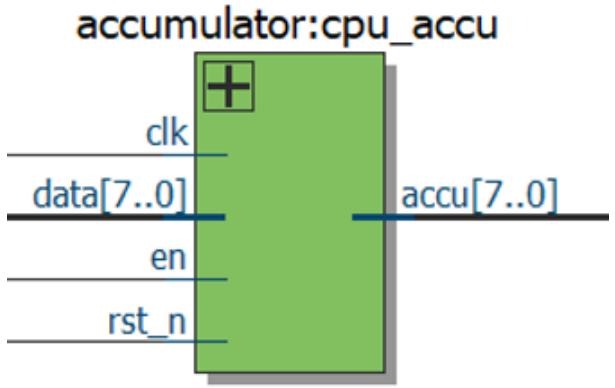
表示指向4个字节之后的下两条指令

模块设计

➢ 累加器模块 (accumulator)

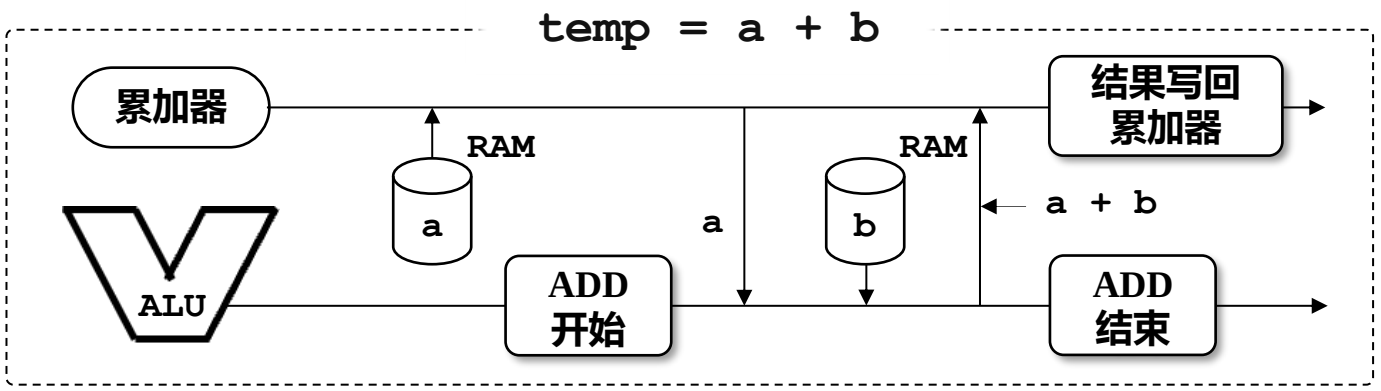
存放当前的结果，该结果可能来自读取的数据、运算的结果等（是双目运算的数据来源）

➢ 双目运算



具有两个运算对象，如相加（ADD）、相与（AND）、异或（XOR）等

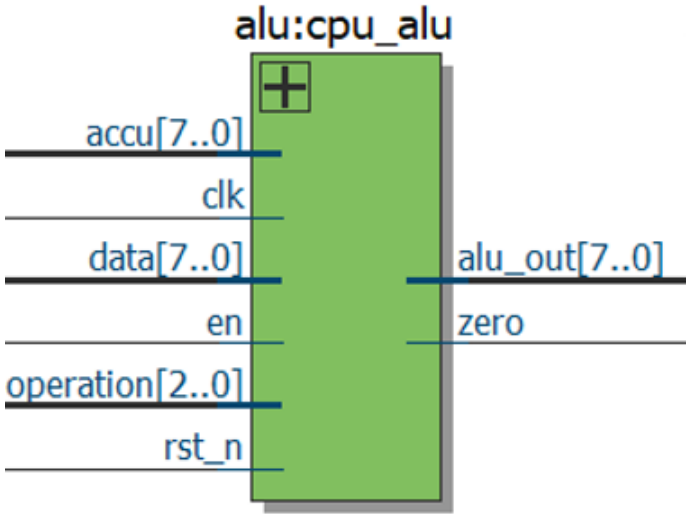
在双目运算中，一个运算对象的数据数据来自存储器，另一个数据即来自累加器



模块设计

➢ 算数逻辑单元模块 (alu)

根据不同的操作数，进行加、与、异或等运算和跳转等操作



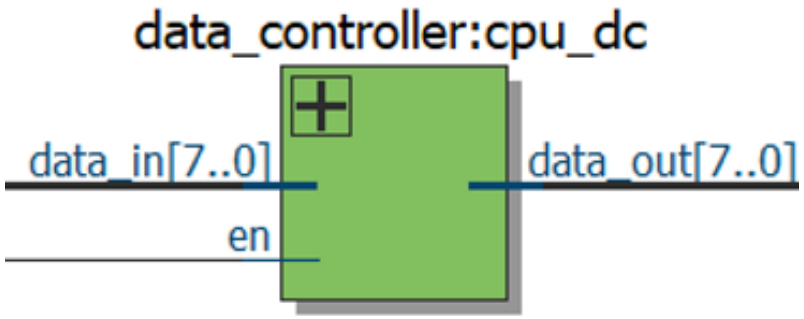
助记符	操作码	指令名称	功能
MOV	000	数据传送	将立即数（当前指令的低8位）写入累加器
SKZ	001	为零跳过	运算结果（alu_out）为0，跳过下一条指令
ADD	010	相加	相加，结果写回累加器
AND	011	相与	相与，结果写回累加器
XOR	100	相异或	相异或，结果写回累加器
LDA	101	读数据	读IR指定地址数据
STO	110	写数据	将累加器数据写入IR指定的存储器地址
JMP	111	无条件跳转	将PC地址设定为IR指定地址，从此继续执行

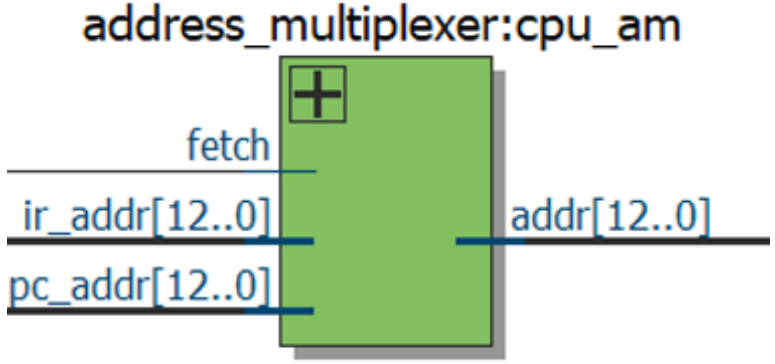
模块设计

- **数据控制器模块**（data_controller）

  控制累加器向数据总线的数据输出

  若不输出，让出数据总线





- **地址多路器模块**（address_multiplexer）

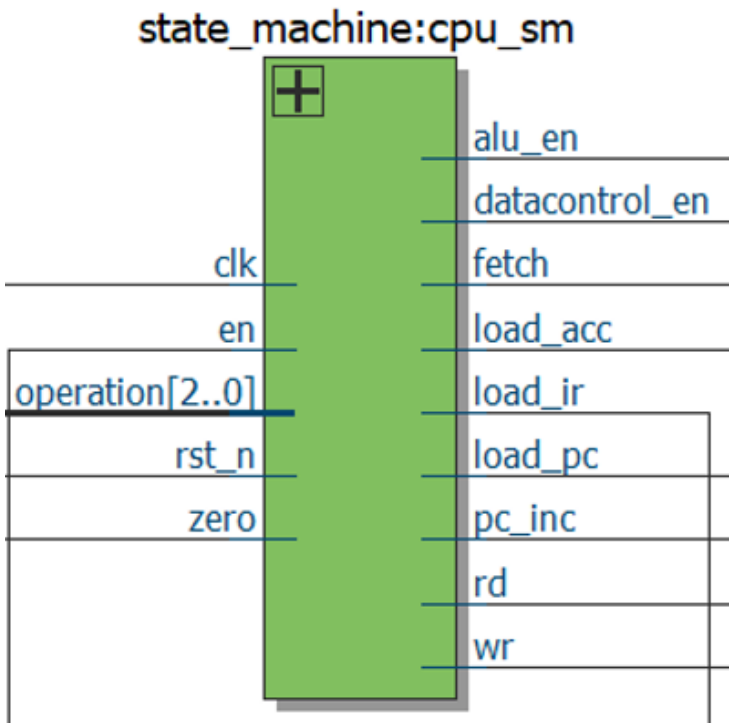
  根据取指令信号，控制输出的地址为**程序地址**或**数据地址**

  pc_addr　ir_addr

模块设计

➢ 状态机模块 (state_machine)

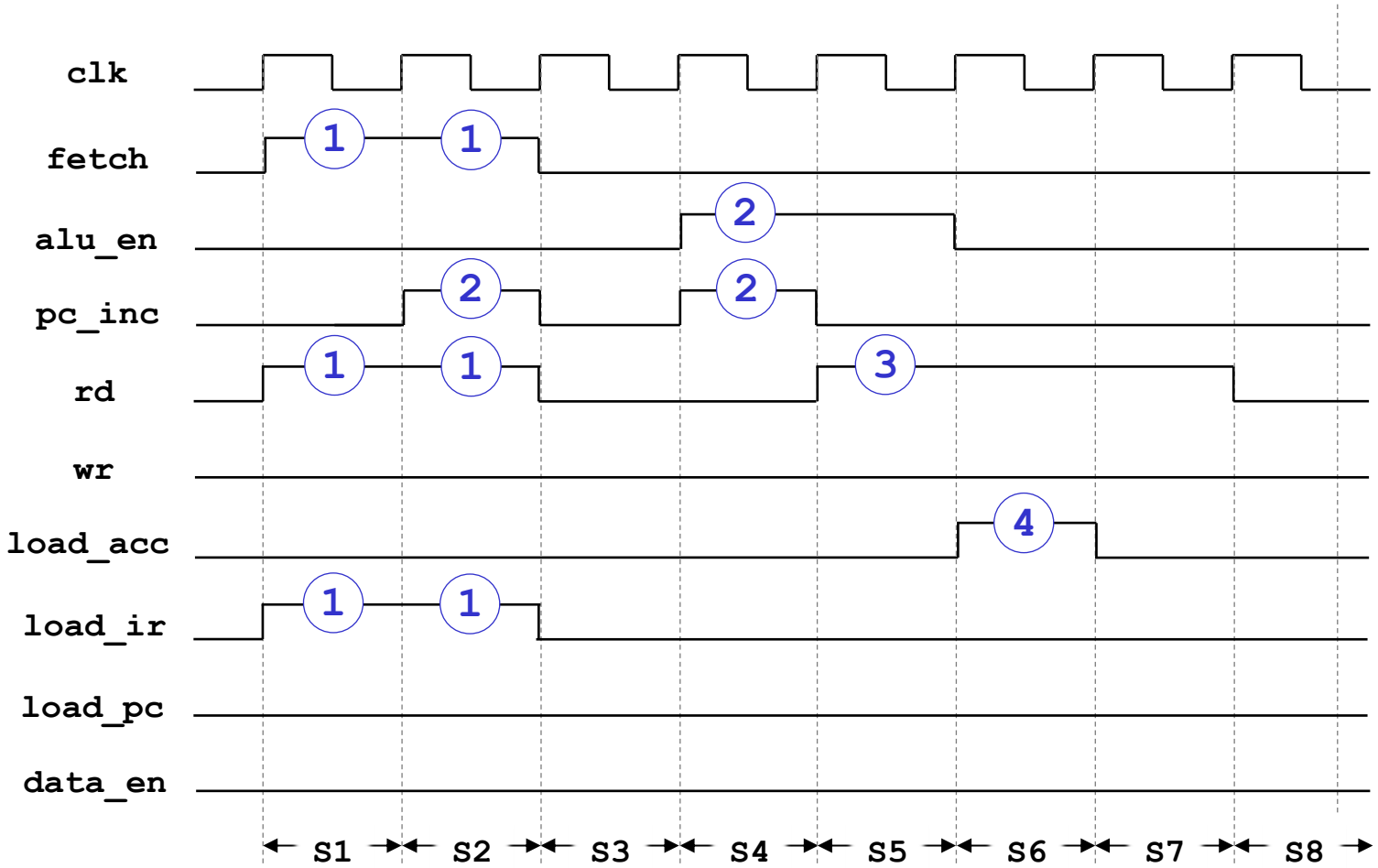
根据当前指令产生的一系列控制信号，令各模块在特定的步骤使能，以实现指令的预期功能



控制信号	功能
fetch	控制PC输出pc_addr，以便接下来从该地址取出指令
alu_en	使能alu，执行算数或逻辑操作
pc_inc	控制PC将pc_addr在一个时钟周期内增加1
rd	允许CPU通过地址总线访问RAM或ROM，读取数据或指令
wr	允许CPU通过地址总线访问RAM，写入数据
load_acc	控制累加器，从数据总线上更新运算结果或数据，并锁存
load_ir	控制指令寄存器，寄存数据总线传来的指令
load_pc	控制程序计数器将pc_addr替换为ir_addr，实现指令跳转
datacontrol_en	控制数据控制器，以允许累加器向数据总线输出数据

模块设计

➢ 状态机控制下CPU的时序波形——以ADD指令为例



1

S1、S2状态，fetch均为高电平，地址多路器输出pc_addr；load_ir、rd均为高电平，指令高8位，低8位分两次自ROM的pc_addr地址处读取到指令寄存器

S3状态，fetch变为低电平，地址多路器输出ir_addr

2

S4状态，pc_inc已累计2次高电平，指向下一条指令；alu_en为高电平，进行指令译码得到ADD

3

S5状态， rd为高电平，将RAM相应地址的数据读入算数逻辑运算单元

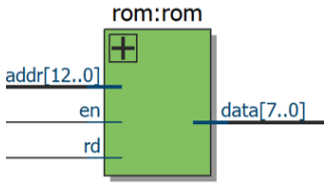
4

S6状态，算数逻辑运算单元对S5状态读入的数据与累加器accu执行相加运算，运算结果给累加器锁存。

模块设计

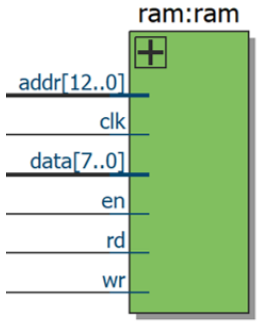
➢ 只读存储器模块（rom）

用于存储指令，只读不写，其大小为8×256



➢ 随机存取存储器模块（ram）

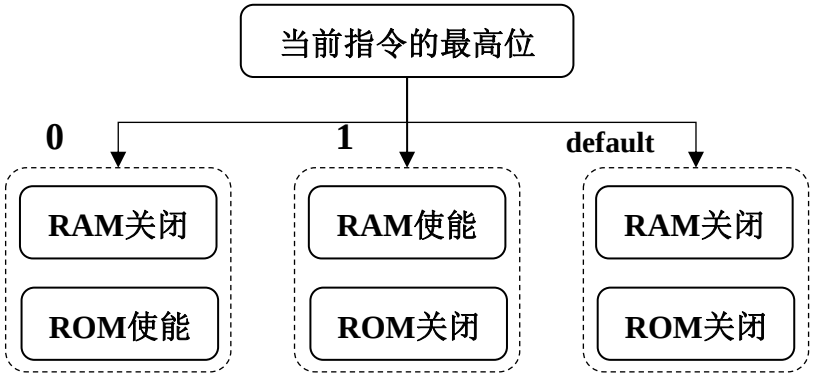
用于存储数据，可读可写，其大小为8×256

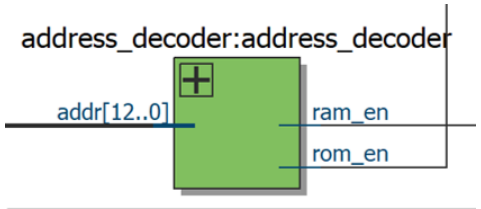


➢ 地址译码器模块（address_decoder）

用于产生ROM或RAM的使能信号

以令ROM、RAM在正确的时间工作





简易精简指令集CPU

1. 相关概念

2. 模块设计

3. 仿真分析

仿真分析

➢ 斐波那契数列（Fibonacci Sequence）

又称黄金分割数列、兔子数列，由意大利数学家**莱昂纳多·斐波那契**于1202年提出

递推公式 $F_n = F_{n-1} + F_{n-2}, n \geq 2, F_0 = 1, F_1 = 1$

前几项：1，1，2，3，5，8，13，21，34，55，89，144，...

➢ 编写二进制仿真程序

（1）使用RAM保存递推公式中的项，使用ROM保存求解问题所需的指令序列

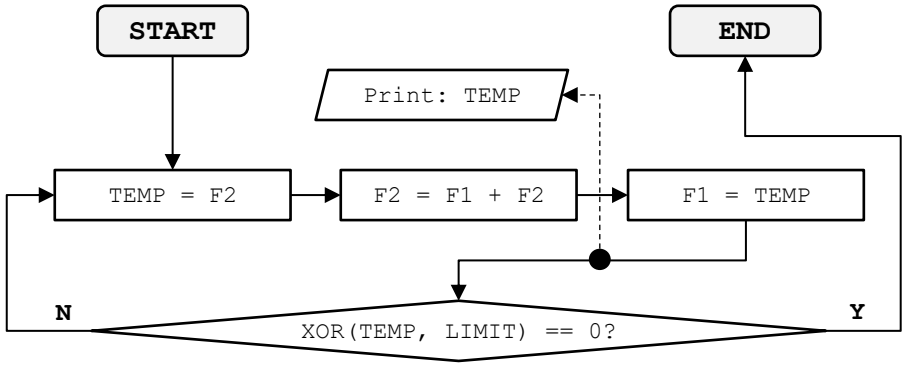
（2）使用MOV指令，为数列前两项赋初值

（3）根据递推公式，使用LDA指令读数，使用ADD指令求和

（4）使用STO指令保存结果，更新数列当前的项

（5）使用XOR指令，比较数列当前的项与规定的上限

（6）安插SKZ、JMP指令，在合适的时机跳出递归，结束程序



仿真分析

➢ 编写二进制仿真程序数据

➢ RAM中的初始数据

00 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

递推公式中的 F_{n-2}，记作F1

01 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

递推公式中的 F_{n-1}，记作F2

10 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

递推公式中的 F_n，记作TEMP

11 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

最大的项（144），记作LIMIT

➢ ROM中的指令

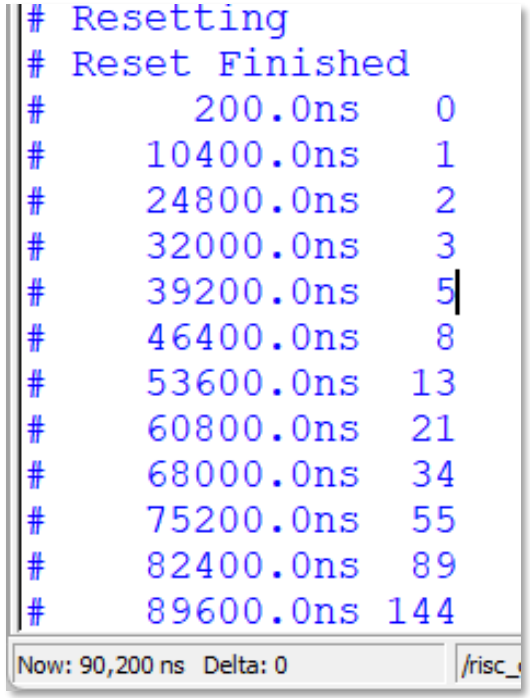
地址	高3bit	低13bit	助记符	功能
00	000	1100000000000	MOV	将立即数0写入累加器
02	110	1100000000001	STO	F2 = 0
04	101	1100000000001	LDA	将F2读入累加器
06	110	1100000000010	STO	TEMP = F2
08	010	1100000000000	ADD	计算F1 + F2，结果写回累加器
0a	110	1100000000001	STO	F2 = F1 + F2
0c	101	1100000000010	LDA	将TEMP读入累加器
0e	110	1100000000000	STO	F1 = TEMP
10	100	1100000000011	XOR	比较TEMP与LIMIT的大小，相等则为0
12	001	0000000000000	SKZ	若上条结果为0，跳过下条，否则不跳过
14	111	0000000000100	JMP	跳转至ROM地址04的指令，继续执行
16	000	0000000000000	MOV	程序结束

仿真分析

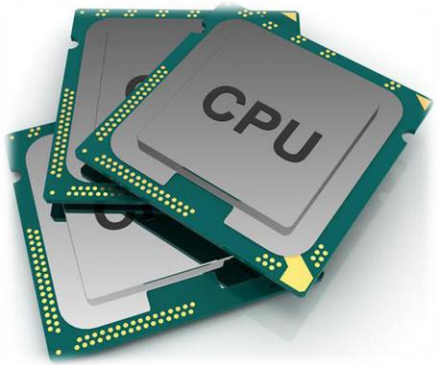
➢ F1、F2与TEMP的变化

在TEMP == LIMIT前，ROM地址14的JMP指令会一直生效，令PC跳回初始地址，程序循环运行

循环次数	F1	F2	TEMP	循环次数	F1	F2	TEMP
0	0	1	**0**	**7**	13	21	**13**
1	1	1	**1**	**8**	21	34	**21**
2	1	2	**1**	**9**	34	55	**34**
3	2	3	**2**	**10**	55	89	**55**
4	3	5	**3**	**11**	89	144	**89**
5	5	8	**5**	**12**	144	233	**144**
6	8	13	**8**				



感谢倾听！



<你的名字>

2024年秋季

北京郵電大學
Beijing University of Posts and Telecommunications