8-bit CPU的Verilog设计

# 一、概况

CPU（Central Processing Unit）的中文全称是“中央处理器”，是计算机中的核心部件。任何计算机程序都是使用编程语言编写的，可以被分为高级语言和低级语言。高级语言采用更易于识别和记忆的字符作为关键字，且更加符合人类的思维逻辑，但是不可以被处理器所理解。[[1]](#footnote-1)因此，需要通过编译变成低级语言。汇编语言就是一种低级语言，它与指令集中的指令一一对应。

本8-bit CPU采用流水线设计，对指令同步进行“取指”“译码”“执行”“写回”四步操作。在指令集的设计上，功能简单，易于电路实现，依赖多步操作以完成复杂的功能。且具有一定的分支预测能力。

寄存器是CPU内部的小存储器，用来暂时存放一些数据。CPU的位数决定了CPU一次能处理的数据大小，也就是CPU的运算能力。CPU的位数越高，其运算能力越强。

本8bit-CPU就是指能够一次处理8个比特的数据的处理器。内部有两个数据寄存器，分别为寄存器0和寄存器1。其中，寄存器1又称为地址寄存器，它的数据不但可以直接被指令赋值，并且可以帮助内存寻址。

由于该指令集的指令允许算术逻辑单元直接从内存中取值，因此并不属于精简指令集（RISC）。

# 二、指令集结构和设计

指令集是一组处理器能够理解和执行的机器指令的集合，通常包含算数操作（加减乘除）、逻辑操作（与或非）、数据传输操作（加载、存储）等，指令集中的每个指令都可以对应到处理器的特定电路，处理器通过解码指令来执行相应的操作。指令集是处理器的重要组成部分，不同的处理器有不同的指令集，指令集的设计直接影响了处理器的性能和功能。

常见的指令集可以被分类为精简指令集（RISC）和复杂指令集（CISC）。RISC指令集的特点是指令长度固定，指令格式简单，指令的执行时间基本相同。CISC指令集的特点是指令长度不固定，指令格式复杂，指令的执行时间不同。RISC指令集的优点是指令执行速度快，硬件实现简单，易于设计和实现。CISC指令集的优点是指令集丰富，编程灵活，可以用较少的指令完成较多的工作。常见的指令集包括x86、ARM、RISCV等，其中x86是英特尔推出的32位复杂指令集，超微半导体（AMD）在此基础上延伸出了64位指令集AMD64，目前广泛应用于家用计算机处理器[[2]](#footnote-2)。而ARM是一种精简指令集（RISC）处理器的指令集，在手机芯片上比较常见。RISCV则是一个新兴的开源免费的RISC指令集。

RISC还有一个特点是不可以直接对内存中的数据进行计算。必须先将数据存储到寄存器中，然后对寄存器之间进行计算操作。这样的好处是可以提高运算速度，但是缺点是需要更多的寄存器。因此，RISC的CPU通常都会有更多的寄存器。

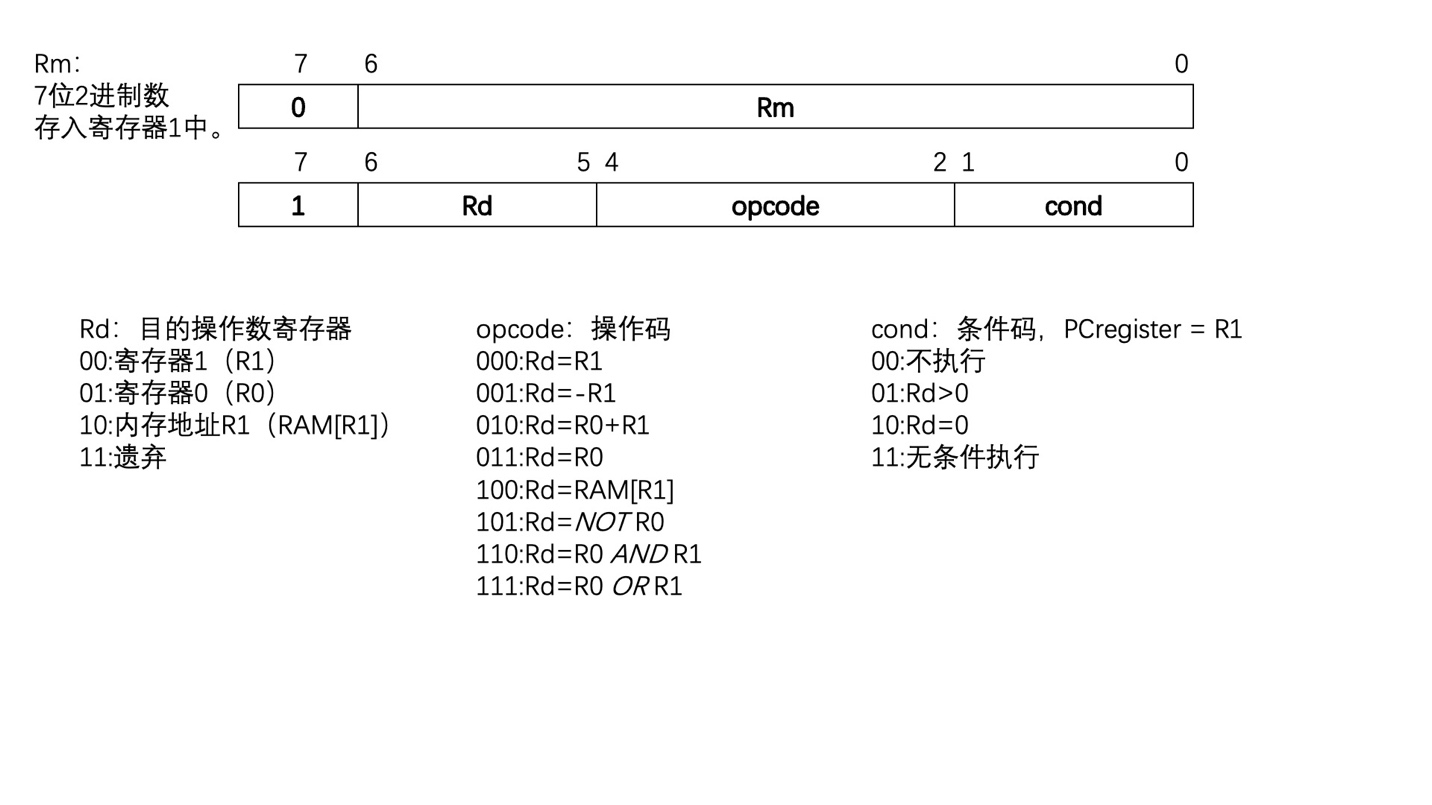
在本8bit-CPU的设计中，所采用的指令集的指令长度和数据长度一样，均为8bit，所以能够执行的指令种类非常受限。因此指令集只设计了2个数据寄存器供CPU调用。在这种条件下，寄存器的数量远远不够，必须允许CPU直接从内存中取值，所以这不是RISC。

指令集由8位构成，分为“数据指令”和“控制指令”。

数据指令：当第7位（最左侧的一位）为0时，将整个指令作为数据存入寄存器1中。

控制指令：当第7位为1时，将指令进行正常的译码以控制整个CPU。

指令集的具体规则如下图所示。

  
图1

当第7位为0时，立即数RM将会存入寄存器1中。立即数只有7位，因此最高位将会用0填充。

当第7位为1时，指令被用来控制CPU的运行。ALU算术逻辑单元是CPU运行时的主要单元，它受到opcode操作码的控制，对输入的数据进行指定运算。Rd目的操作数寄存器，指明了的ALU的输出结果将会被存到什么地方。

Cond条件码是对ALU的输出进行条件判断。通常在CPU执行指令的时候，会按照存储器中的顺序顺次执行指令。这是通过一个会自增的PC计数器（Program counter，程序计数器）实现的，在每个时钟周期，CPU将计数器中的数值作为地址从存储器的对应位置读取指令。所以当需要进行特别的跳转时，根据条件码的约束，在符合条件的情况下，PC计数器不再自增，而是加载寄存器1中存储的数值。

下面给出了一个事例，编写用于计算斐波那契数列的第10个数的程序。

譬如首先先写一个直观的伪代码。

\*a = 0

a[0] = 1

a[1] = 1

i = 0

T = 8

while i<T:

a[i+2] = a[i+1] + a[i]

i = i + 1

上面代码中的a是一个数组，并且数组的初始地址被规定为了0。数组的第一个数和第二个数都等于1。接下来的代码其实是for(i=0;i<T;i+=1)循环的另一种写法。然后我们让每个数都等于前面两个数的和。

在把它翻译为汇编语言时，需要把复杂的代码改写为可以由指令集处理的简单代码。举个例子，a[0]=1， a是一个数组，但是根据C语言的知识，a的本身其实是一个指针，而a[0]其实指向的是\*(a+0)。即\*(a+0)=1。其实就是要把内存RAM[a+0]等于1。那么我们首先要让寄存器1的值R1=a+0，因为指令集只支持读或取RAM[R1]的数据。但是，由于指令集里面唯一的立即数操作只能向寄存器1中写入数据，这将会与上面的RAM[R1]冲突。解决的方法是充分利用寄存器0，凭借指令RAM[R1]=R0来写入数据。过程略有繁琐，如下所示。

#实现a[0] = 1的方法

#计算出a+0并存储。

R1 = a # a是一个常数，并且已经定义过了。

R0 = R1 # let R0 = a

R1 = 0

R0 = R0 + R1 # let R0 = a + 0

R1 = &addr # addr也是一个已经定义过的常数。

RAM[R1] = R0 # let RAM[addr] = a + 0

#让R0等于待存入的数

R1 = 1

R0 = R1 # let R0 = 1

#让R1等于目标地址

R1 = &addr

R1 = RAM[R1] # let R1 = a + 0

#向目标地址写入待存入的数

RAM[R1] = R0 # let \*(a+0) = 1

解决了如何向数组a中写入数据，下一个需要解决的是while逻辑。在指令集中，只有大于和等于两种判断，若要实现while i<T的循环，即要在T-i>0时跳转入循环，如下的逻辑所示。

(START)

R1 = T - i # 假设已经通过一系列操作使得R1 = T - i

if R1>0 goto LOOP

...... # 循环结束之后的代码

(LOOP)

...... # 循环内部的代码

goto START

如上面所示，当T-i>0时，跳转到LOOP，并在代码执行结束后回到START；当T-i>0不再为真时，才会运行循环结束后的代码。

最后一个需要考虑的问题是如何实现a[i+2] = a[i+1] + a[i]，短短的一行其实包含了大量的运算，如a+i、a+i+1、a+i+2、Ram[a+i]、……、a[i+1] + a[i]等。这样的操作不是两个寄存器就能实现的，需要不断的向Ram内存中写入数据。在这里我们需要定义三个常数addr、addr2、addr3，作为地址存储a+i、a+i+1和a+i+2的结果存到内存中。最后通过反复的存取来完成计算。

# 过程被简写

RAM[addr] = a + i

RAM[addr1] = a + i + 1

RAM[addr2] = a + i + 2

R1 = RAM[addr]

R1 = RAM[R1] # R1 = a+i

R0 = R1

R1 = RAM[R1]

R0 = R0 + R1

R1 = RAM[addr2]

#定义常量

a = 0

T = 8

#定义变量的指针地址

\*i = 11

\*addr = 12

\*addr2 = 13

\*addr3 = 14

if \_\_name\_\_ == ”\_\_main\_\_”:

addr = 0

a[0] = 1

a[1] = 1

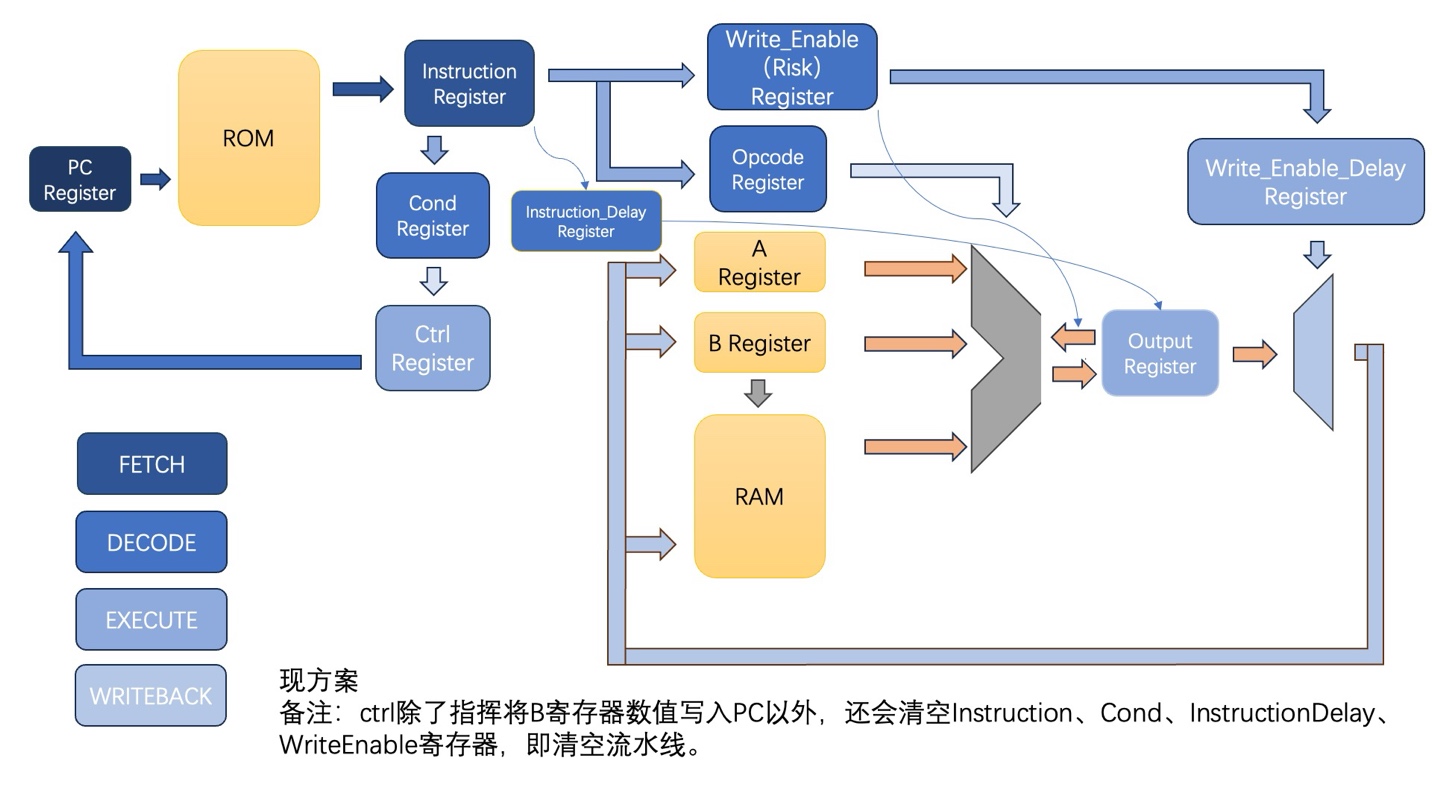
i = 0

while i-T<0:

addr = a + 1

# 三、CPU结构

下图是CPU的流水线逻辑图，展示了CPU的流水线是如何工作的。

  
图2

PC寄存器（Program Counter，程序计数器）是CPU中最基本的控制单元，在默认情况下，它是一个计数器，在每个时钟周期数值加1。

ALU单元（Arithmetic Logic Unit，算术逻辑单元）是CPU的核心单元，是一个非时序的电路，它接受3位操作码，将寄存器0、寄存器1或者内存的数据进行特定的计算并输出。

A Register和B Register 分别为寄存器0和寄存器1。

只读存储器ROM是存放二进制指令的地方，随机存取存储器RAM是保存数据的地方。

虽然直接存储数据的寄存器只有两个，但是实际的CPU设计中还包含了众多寄存器暂存指令。

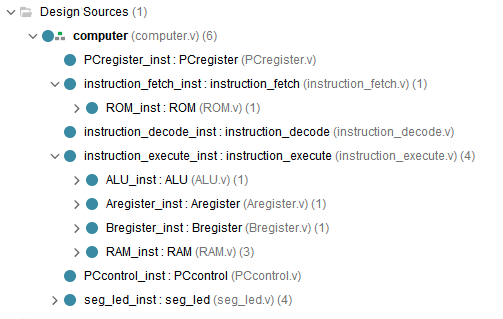
PC寄存器控制着CPU的“取指”操作，从对应的ROM地址中取出二进制指令，取出的指令将会暂存在Instruction Register中。

之后，指令会被“译码”操作。如果是控制指令，那么条件码会进入Cond Register中，操作码会进入Opcode Register中，ALU输出的目的地会写入Write Register中。

最后，指令进行“执行”操作。Write Register相当于控制了一个数据分配器，决定了ALU输出结果的最后走向。

此外，还有一个PCwrite 寄存器，它会基于条件码和ALU输出结果判断是否要进行跳转。如果需要跳转，在“执行”操作的下一个时钟周期，PC寄存器的数据将会被更新为寄存器1的数据。但是在流水线中，跳转指令从取指到执行需要经过3个周期，将会导致不需要的运行的指令也进入了流水线作业。所以，PCwrite还需要拦截下个周期的“取指”“译码”和“执行”操作。

# 四、Verilog实现

  
图3

<https://box.nju.edu.cn/f/274d7195d3554b9aac42/>

1. 低级语言和高级语言的区别 - ps学习的文章 - 知乎https://zhuanlan.zhihu.com/p/65746260 [↑](#footnote-ref-1)
2. 英特尔在家用CPU上同样使用AMD64指令集，因其自研的64位指令集不兼容x86。 [↑](#footnote-ref-2)