**计算机组成原理CPU设计实验报告**

王嘉禾 2193211079 计算机试验班001

**单周期CPU**

(代码见文件夹“单周期cpu”)

1. **指令集设计**

指令集分为四个类型，分别为1/2/3/4型指令，格式如下：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 占用bit | 2b | 5b | 5b | 5b | 11b | 4b |
| 1型指令 | 00 | Rs | Rd | Rt | / | Func |
| 2型指令 | 01 | Rs | Rt | Addr\_16/Imm\_16 | | Func |
| 3型指令 | 10 | Rs | / | Addr\_16/Imm\_16 | | Func |
| 4型指令 | 11 | / | | Addr\_16/Imm\_16 | | Func |

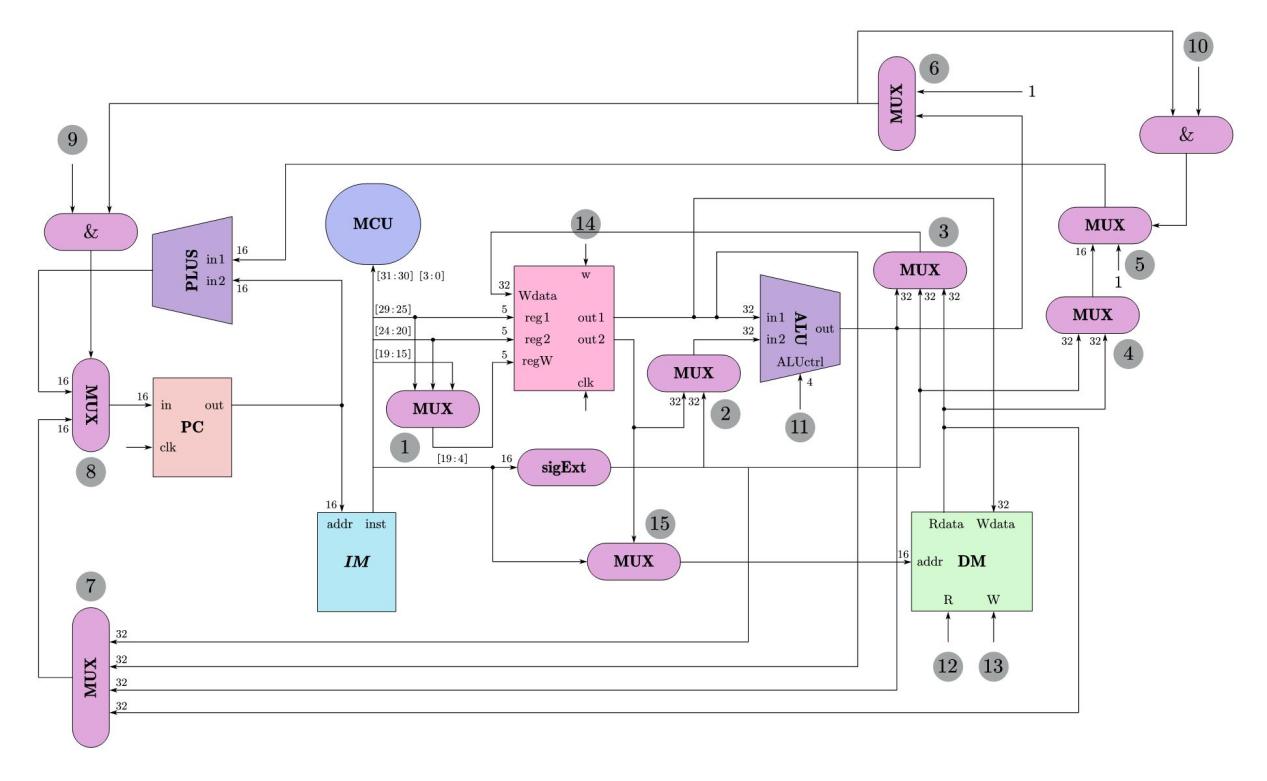
其中Rs/Rd/Rt分别代表两个源寄存器和一个目标寄存器的编号

Addr\_16代表16位地址，Imm\_16代表16位立即数

详细指令设计如下：

|  |  |
| --- | --- |
| 1型指令（共10条） | |
| 操作码 | 功能 |
| 0000 | Rt <- (Rs) + (Rd) |
| 0001 | Rt <- (Rs) - (Rd) |
| 0010 | Rt\_32 <- (Rs\_16low) \*( Rd\_16low) |
| 0011 | Rt\_16low <- (Rs\_32) / (Rd\_16low) |
| 0100 | Rt <- (Rs) & (Rd) |
| 0101 | Rt <- (Rs) | (Rd) |
| 0110 | Rt <- (Rs) ^ (Rd) |
| 0111 | if (Rs) == (Rd) Rt<-1 else Rt<-0 |
| 1000 | Rt <- (Rs) << (Rd) |
| 1001 | Rt <- (Rs) >> (Rd) |
| 2型指令（共6条） | |
| 操作码 | 功能 |
| 0000 | if (Rs) == (Rt) PC <- Addr\_16 else PC <- PC + 1 |
| 0001 | if (Rs) == (Rt) PC = PC + Imm\_16 else PC <- PC + 1 |
| 0010 | if (Rs) == (Rt) PC <- DM[Addr\_16] else PC <- PC + 1 |
| 0011 | if (Rs) == (Rt) PC = PC + DM[Addr\_16] else PC <- PC + 1 |
| 0111 | If (Rs) == Imm\_16 Rt <= 1 else Rt <- 0 |
| 1000 | DM[(Rt)] <- (Rs) |
| 3型指令（共14条） | |
| 操作码 | 功能 |
| 0000 | Rs <- (Rs) + Imm\_16 |
| 0001 | Rs <- (Rs) - Imm\_16 |
| 0010 | Rs <- (Rs\_16low) \* Imm\_16 |
| 0011 | Rs\_16low <- (Rs) / Imm\_16 |
| 0100 | Rs <- (Rs) & Imm\_16 |
| 0101 | Rs <- (Rs) | Imm\_16 |
| 0110 | Rs <- (Rs) ^ Imm\_16 |
| 1000 | Rs <- (Rs) << Imm\_16 |
| 1001 | Rs <- (Rs) >> Imm\_16 |
| 1010 | Rs <- DM[Addr\_16] |
| 1011 | Rs <- Imm\_16 |
| 1100 | DM[Addr\_16] <- Rs |
| 1101 | PC <- (Rs) |
| 1110 | PC <- (Rs) + Imm\_16 |
| 4型指令（共4条） | |
| 操作码 | 功能 |
| 0000 | PC <- Addr\_16 |
| 0001 | PC <- PC + Imm\_16 |
| 0010 | PC <- DM[Addr\_16] |
| 0011 | PC = PC + DM[Addr\_16] |

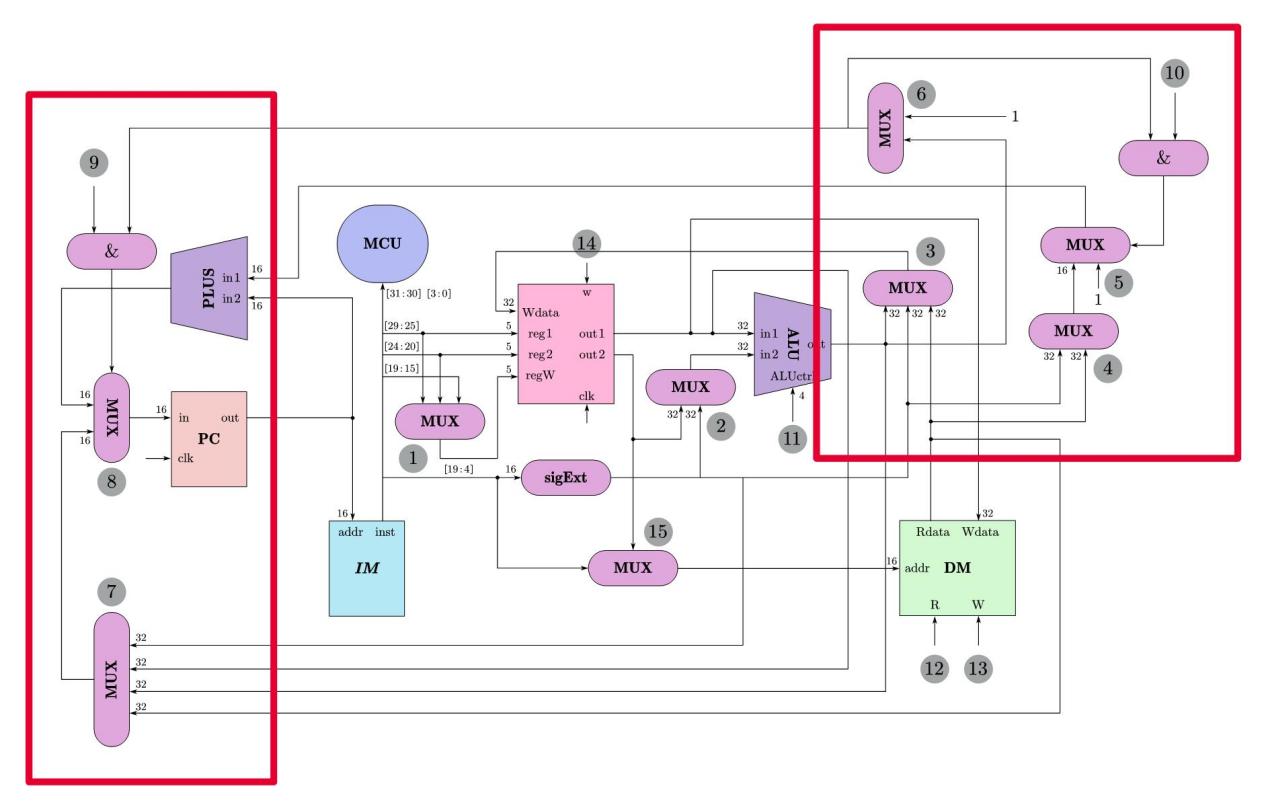
1. **CPU元件和连线设计**



图中的编号代表没有画出的主控制单元生成的主控制信号，以下给出详细解释

|  |  |  |
| --- | --- | --- |
| 编号 | 信号 | 作用解释 |
| 1 | 多路选择器控制信号 | 由MCU根据指令生成，决定指令中那一个字段是要写入数据的寄存器编号。 |
| 2 | 多路选择器控制信号 | 由MCU根据指令生成，决定是两个寄存器的运算还是一个寄存器和一个立即数的运算，对于ALU，in1一定为寄存器，in2可能是寄存器或立即数。 |
| 3 | 多路选择器控制信号 | 由MCU根据指令生成，决定写入寄存器的数据来自指令中的立即数/ALU的运算结果/主存中取出的值。 |
| 4 | 多路选择器控制信号 | 由MCU根据指令生成，当涉及无条件跳转指令且为PC跳转一个特定距离时，决定该距离是一个立即数/存储空间的值。 |
| 5/10 | 多路选择器控制信号来自与门的输出信号 | 多路选择器的控制信号由MCU或MCU和ALU共同控制，当不涉及条件转移时，6号多路选择器输出1，不影响与门的结果，由MCU控制信号控制PC的变化；当涉及条件转移时，mux6输出ALU的结果，若为1，同上，若为0，则不转移，传输1，PC=PC+1（和另一个与门9和多路选择器6联合作用） |
| 6 | 多路选择器控制信号 | 由MCU根据指令生成，ALU是否参与决定PC的变化方式，即判断该指令是否是条件转移指令，当是条件转移指令且ALU结果为0时（不满足条件），该多路选择器输出0，控制两个与门都输出0，此时会选择PC加上值的模式，且该值为1；当不是条件转移指令或条件满足时，多路选择器输出1，由MCU的信号完全控制PC的转移。（与两个与门联合作用） |
| 7 | 多路选择器控制信号 | 由MCU根据指令生成，涉及转移指令时生效，当PC的转移方式为直接赋成特定值时，决定改值来自立即数/寄存器/ALU运算结果/主存内容。 |
| 8/9 | 多路选择器控制信号来自与门的输出信号 | 与6号多路选择器联合作用，8号多路选择器决定PC的转移方式为加上特定的值/被赋成特定的值，信号9位MCU控制信号，当非PC=PC+1的情况下做出上述决定，当存在条件转移且条件不满足时，与门的另一输入为0，强制选择加上特定值的方式且特定值为1，即PC=PC+1。 |
| 11 | ALU控制信号 | 由MCU生成，决定ALU作何种运算，在此cpu设计中，ALU控制信号在绝大多数情况下与指令中的操作码字段一致。 |
| 12 | 主存读信号 | 控制主存读addr字段对应地址的内容并输出。 |
| 13 | 主存写信号 | 控制主存将Wdata端口输入的数据存入主存的addr地址。 |
| 14 | 寄存器堆写信号 | 控制寄存器堆将传入的值写入regW端口输入指定编号的寄存器。 |
| 15 | 多路选择器控制信号 | 控制主存写入的地址时立即数还是目标寄存器中的值（低16位） |

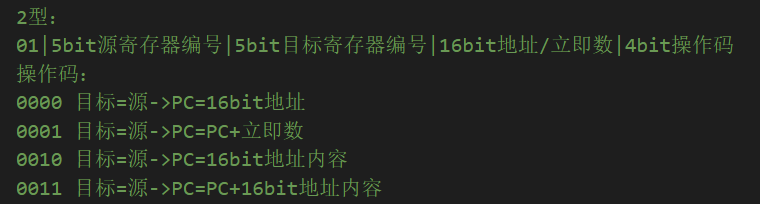
1. **自主设计：cpu指令转移通路**

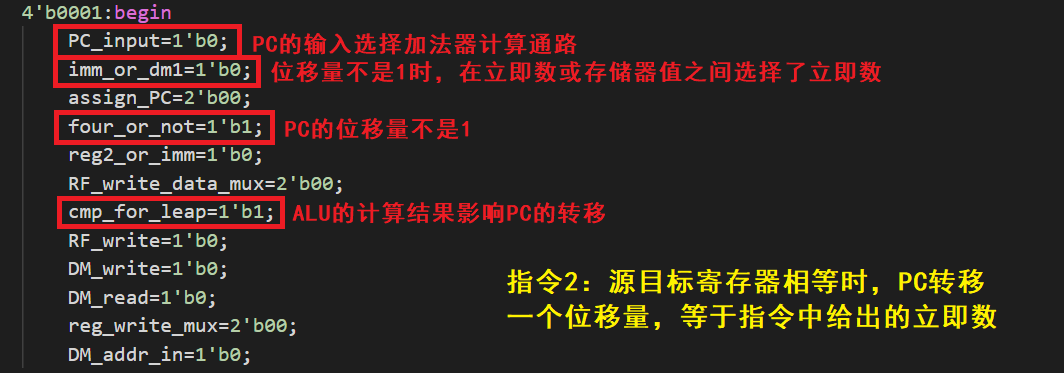
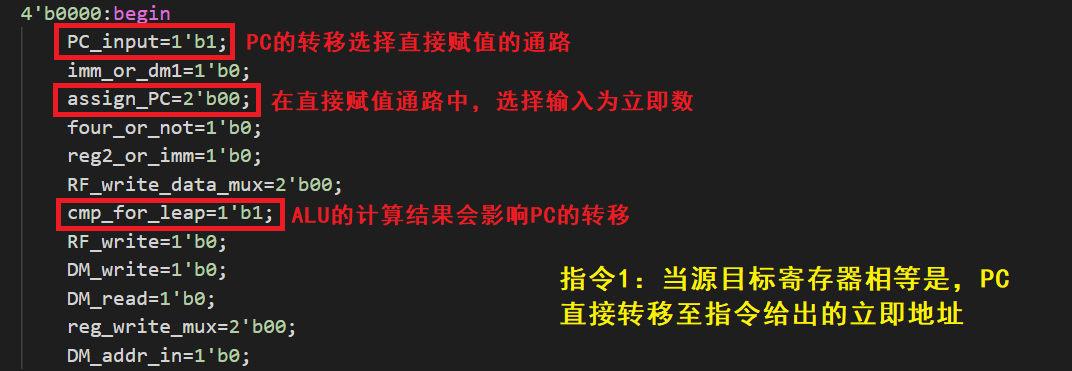


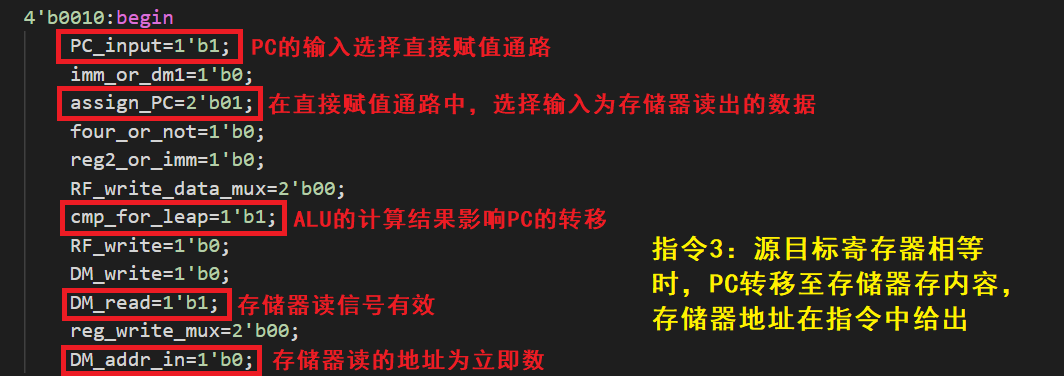
整个指令转移通路的核心部件为五个多路选择器（其中四个二路选择器，一个四路选择器）和两个与门构成。总的来说，**转移通路有两条支路通过一个二路选择器输入进PC，分别对应位移量或直接赋值的转移方式。两条支路各有多路选择器控制相关的量的输入来源。并且，ALU的计算结果通过一个二路选择器和两个与门干预整个转移通路，对应条件转移指令的情况。**以下给出核心部件的作用描述：

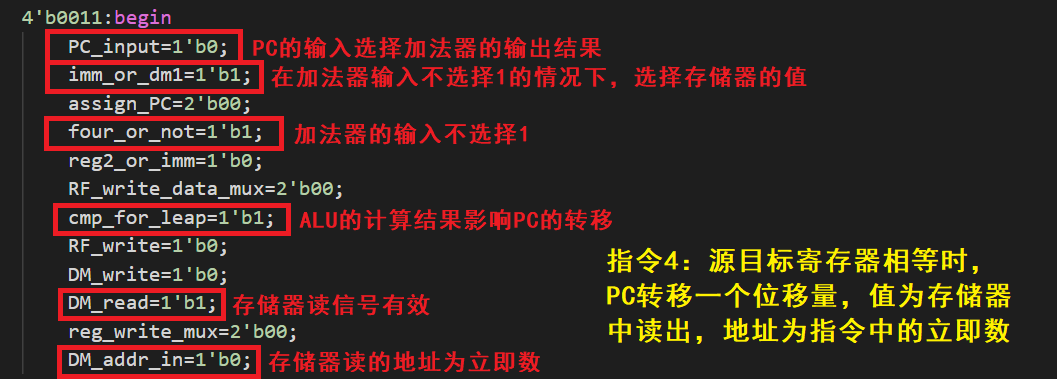
|  |  |  |
| --- | --- | --- |
| 图中编号 | 部件名称 | 作用描述 |
| 8 | 二路选择器 | 控制最终输入PC的值是由加法器做加法运算得出（0），还是直接赋值（1），对应于转移指令中是转移一个位移量，还是直接转移到特定地址 |
| 7 | 四路选择器 | 在PC的转移为直接赋值的情况下，控制输入的值是如何得到的，有四种情况，分别为：立即数（00），存储器内容（01），寄存器内容（10）和ALU计算结果（通过寄存器内容+立即数得到，对应于基址变址寻址）。当8号二路选择器的输入为1时，PC的转移方式为直接赋值，本器件有效运行。 |
| 5 | 二路选择器 | 在PC转移方式为位移量（通过加法器之后赋值）的情况下，控制输入的值是不是1（顺序转移与否）。当8号二路选择器的输入为0时，PC的转移方式为位移量，本器件有效运行。 |
| 4 | 二路选择器 | 在PC转移方式为位移量，且不是直接顺序执行的情况下，控制输入的值是立即数（0）还是存储器读出的值（1）。当8号二路选择器的输入为0时，PC的转移方式为位移量，且5号二路选择器的输入为1时，本器件有效运行。 |
| 6 | 二路选择器 | 控制ALU的计算结果是否影响PC的转移。当为条件转移指令时（选择器控制信号为1），则输出ALU的计算结果的最后一位（此时ALU输出0或1）；不是条件转移指令时，则输出1。与9号10号两个与门配合使用，当该二路选择器输出0时（只可能是条件转移且条件不满足），则5号二路选择器输出1，且8号二路选择器输入为0，表示选择加法器运算结果支路，代表强制执行PC=PC+1；当该二路选择器输出为1时（条件转移且条件满足或非条件转移指令），则两个与门输出9号和10号的信号量，按指令设定的PC转移方式运作。 |
| 9 | 与门 | 输入的信号为指令设定的PC转移方式（位移量或直接赋值），与6号二路选择器配合使用。若另一输入为0，则与门输出为0，这代表此时指令为条件转移指令且指令不满足，强制执行PC=PC+1，因此PC的输入通路为加法器运算结果通路。 |
| 10 | 与门 | 输入的信号为当PC的转移方式为位移量时，是不是1，与6号二路选择器配合使用，当指令为非转移指令时，10号信号为0，与门输出0，代表选择PC的位移量为1,；当为条件转移指令或转移指令时，输出1，代表PC的位移量为立即数或存储器值（当然也可能通过这种方式输出1），在条件转移下是否成功转移由与门的另一输入（6号二路选择器的输出）决定。若为条件转移且条件不满足，则为0，与门输出0，强制选择位移量1。 |

以2型指令中的四条条件转移指令为例，分析CU部件的控制信号



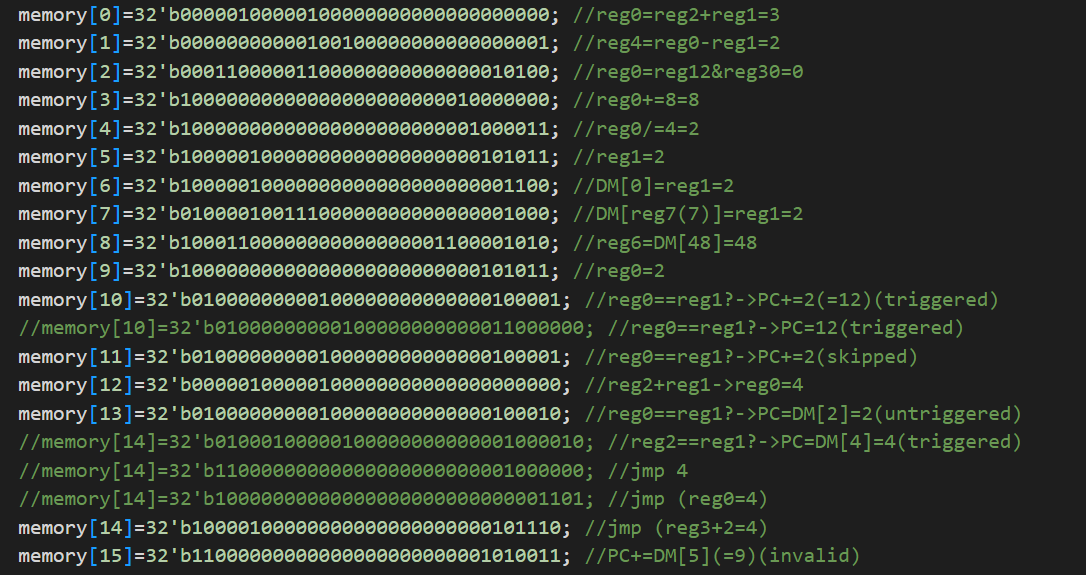




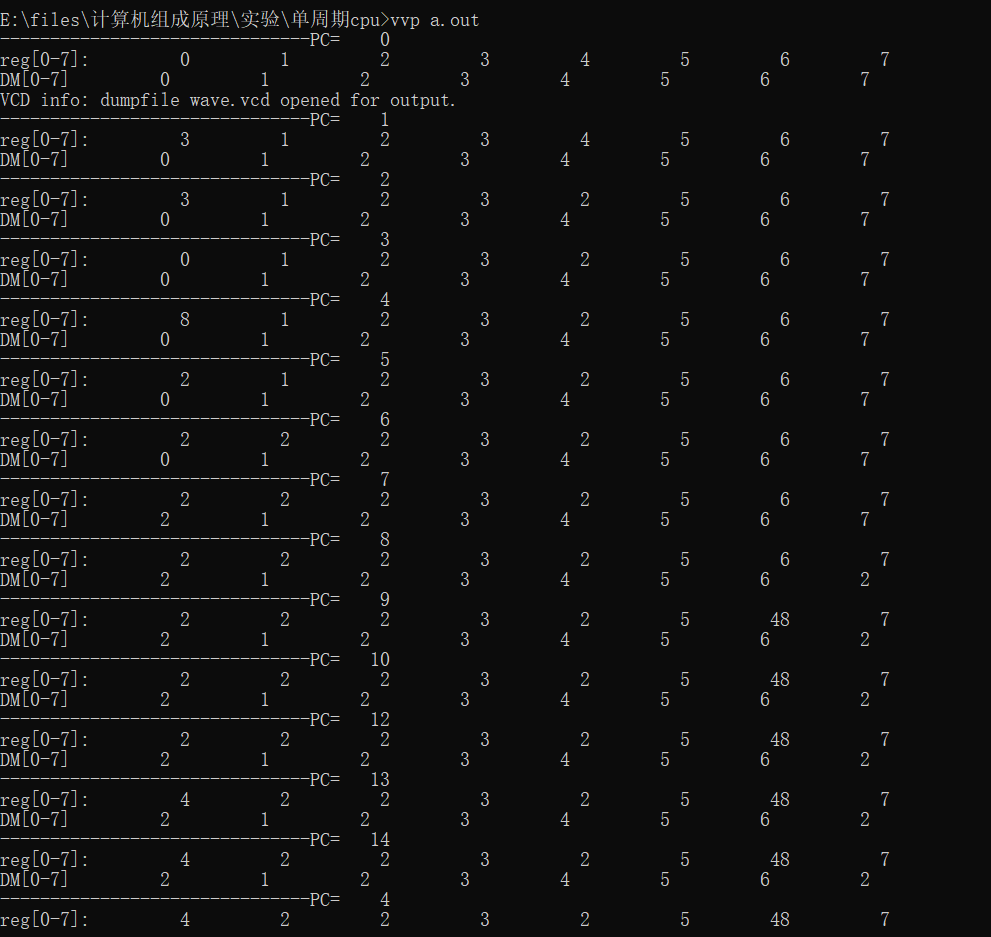
****

1. **实验测试**

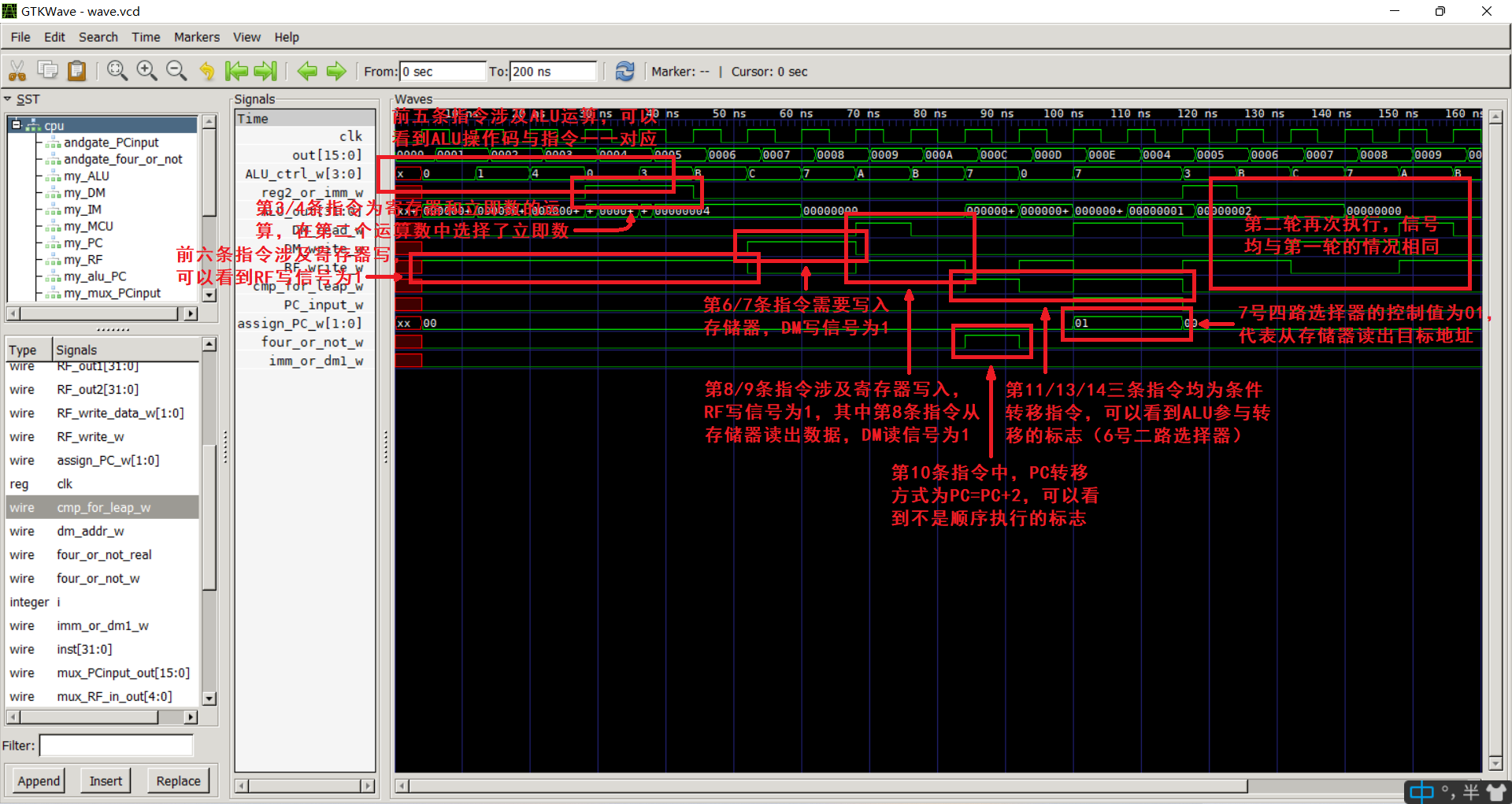
在程序存储器中初始化一端小程序，如下所示，程序的作用在注释中给出



在每个时钟周期打印0-7号寄存器的值，0-7号主存的值和PC的值，如下图所示



部分波形如下图所示，可以看到，第10/11/13/14条指令为条件转移指令，第10条指令条件成立，转移至第12条指令，因而第11条指令没有执行，而第13条质指令条件不成立，第14条指令条件成立，转移至第4条指令，因而第15条指令没有执行。而波形中第10/13条指令执行时，波形显示ALU计算结果影响PC转移,即多路选择器mux输出了ALU的计算结果。第3/4条指令调用立即数参与运算，从波形中可看出ALU的第二个输入in2选择了立即数。第6/7条指令涉及写入主存内容，可以看到主存的写信号DM\_w变为1，但是寻址方式不同，可以看到15号多路选择器在第6条指令中选择了立即数而在第7条中选择了寄存器。在所有的寄存器运算指令中，也可以看到寄存器堆RF的写指令变为1。其余一些波形也均符合程序调用，说明程序调用正确。（图中还给出了另外两种使PC转移到位置4的方法，也均测试正确）以下以部分关键信号我为例分析波形：



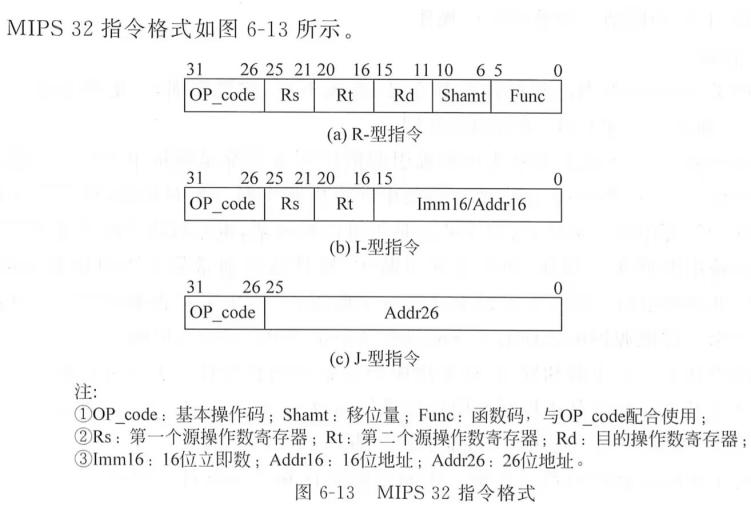
**多周期CPU（单总线模式）**

1. **指令集设计（MIPS32）**



**指令格式描述如下**

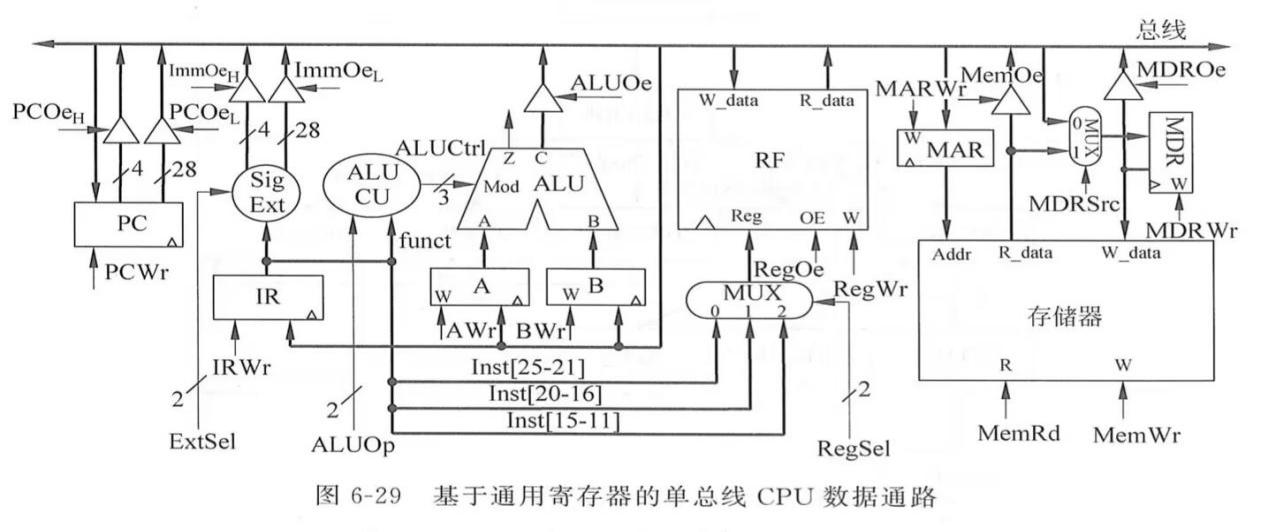
|  |  |
| --- | --- |
| OP\_code | 对应指令 |
| 100011 | Lw |
| 101011 | rw |
| 000000 | beq |
| 000010 | jmp |



**ALU操作码如下表所示**

|  |  |  |
| --- | --- | --- |
| 100 | 加法（判断溢出） | C <- (A) + (B) (save Z) |
| 110 | 减法（判断溢出） | C <- (A) - (B) (save Z) |
| 101 | 加法（不判断溢出） | C <- (A) + (B) |
| 000 | 与运算 | C <- (A) & (B) |
| 001 | 或运算 | C <- (A) | (B) |
| 011 | 异或运算 | C <- (A) ^ (B) |
| 010 | 判断大小 | If (A) < (B) C <- 1 else C <- 0 |
| 111 | 判断相等 | If (A) == (B) C <- 1 else C <- 0 |

1. **数据通路如下图所示**



1. **CU单元控制信号表如下所示**



以下给出控制信号随始终周期变化的逻辑

**取指令阶段**

**第一个时钟周期**，送指令地址：MAR<-(PC);PCOeH,PCOeL,MARWr/A<-(PC);AWr

**第二个时钟周期**，取指令：IR<-M[MAR];MemRd,MemOe,IRWr

**第三个时钟周期**，送PC修正量：B<-1;ExtSel=10,ImmOeH,ImmOeL,BWr

**第四个时钟周期**，修正PC：PC<-(PC)+4;ALUOp=00,ALUCtrl=100,ALUOe,PCWr

**第五个时钟周期**，指令译码和读寄存器：A<-(RF[IR[25-21]]);RegSel=00,RegOe,AWr

**指令执行阶段**

1. 取数指令(lw)

**第六个时钟周期**，计算存储器地址：B<-(SigExt(IR[15-0]));ExtSel=00,ImmOeH,ImmOeL,BWr

**第七个时钟周期**，送存储器地址：MAR<-(A)+(B);ALUOp=00,ALUCtrl=100,ALUOe,MARW

**第八个时钟周期**，读存储器：MDR<-M[MAR];MemRd,MDRSrc,MDRWr

**第九个时钟周期**，写寄存器：RF[IR[20-16]]<-(MDR);MDROe,RegSel=01,RegWr

1. 存数指令(sw)

**第六个时钟周期**，计算存储器地址：B<-(SigExt(IR[15-0]));ExtSel=00,ImmOeH,ImmOeL,BWr

**第七个时钟周期**，送存储器地址：MAR<-(A)+(B);ALUOp=00,ALUCtrl=100,ALUOe,MARWr

**第八个时钟周期**，读寄存器MDR<-(RF[IR[20-16]];RegSel=01,RegOe,MDRSrc=0,MDRWr

**第九个时钟周期**，写存储器：M[MAR]<-(MDR); MemWr

1. 算数/逻辑运算指令(R型)

**第六个时钟周期**，读寄存器：B<-(RF[IR[20-16]]);RegSel=01,RegOe,BWr

**第七个时钟周期**，运算并写寄存器：RF[IR[15-11]]<-(A)OP(B);ALUOp=10,ALUCtrl=xxx,ALUOe, RegSel=10,RegWr

1. 分支指令(beq)

**第六个时钟周期**，程序转移：B<-(RF[IR[20-16]]);RegSel=01,RegOe BWr

**第七个时钟周期**，送 PC：If Z==0 then指令周期结束;ALUOp=01,ALUCtrl=110

A<-(PC);PCOeH,PCOeL,AWr

**第八个时钟周期**，送PC修正量：B<-(SigExt(IR[15-0])<<2);ExtSel=01.ImmOeH,ImmOeL,BWr

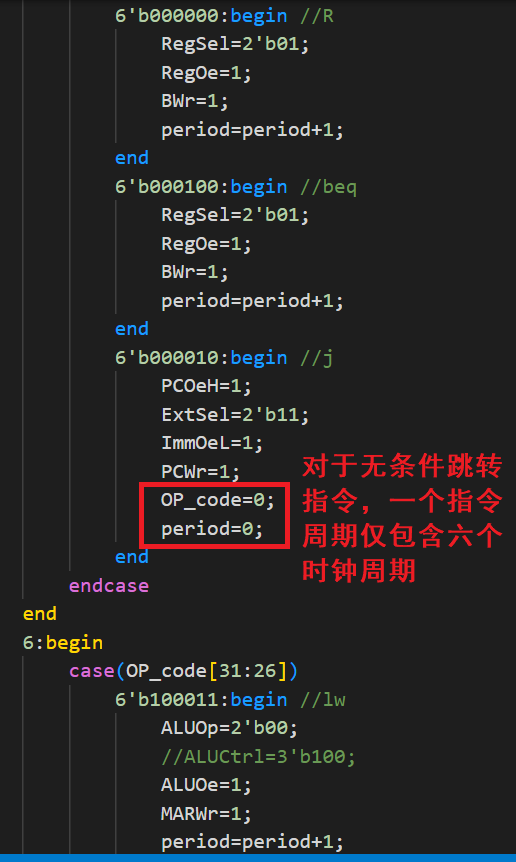
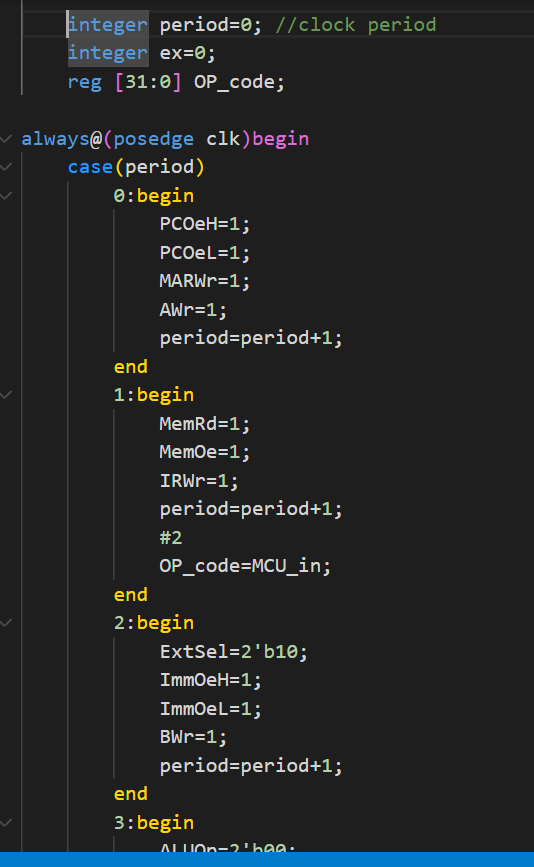
**第九个时钟周期**，计算转移地址：PC<-(A)+(B);ALUOp=00,ALUCtrl=100,ALUOe,PCWr

1. 跳转指令(j)

**第六个时钟周期**，转移：PC<-(PC[31-28])||(IR[25-0]);PCOeH,ExtSel=11,ImmOeL,PCWr

1. **CU组件的组织形式**

在CU组件中，设置一个period变量，控制当前所处的时钟周期数，初始值设置为0，每遇到CLK信号上沿，则根据当前指令的操作码输出相应的控制信号，并进行period（时钟周期）的转移。当遇到CLK信号下沿时，则将所有控制信号清零，程序片段如下图所示：

****

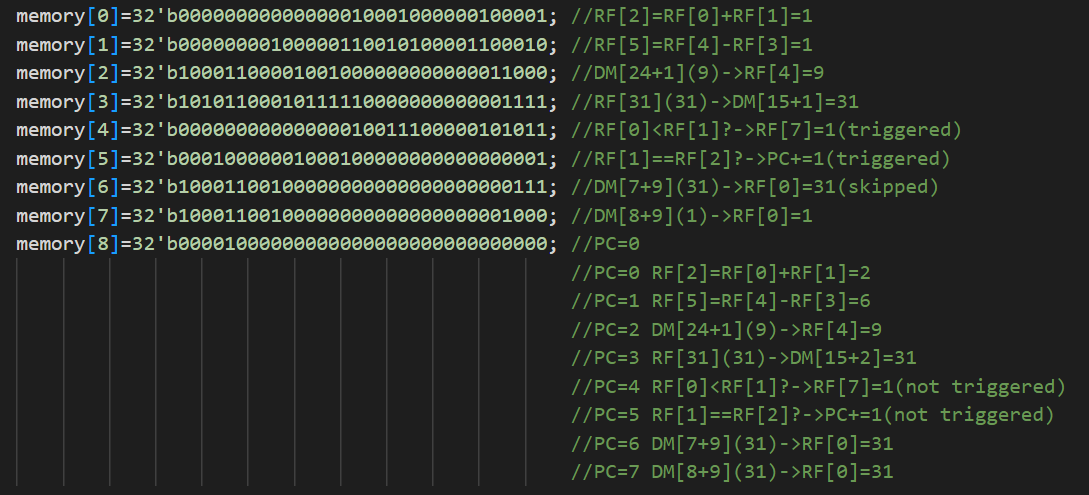
对于每个时钟周期，若不是该指令周期中的最后一个时钟周期，则信号设置结束后

执行period=period+1，否则period直接

跳转回0，开始下一个指令周期

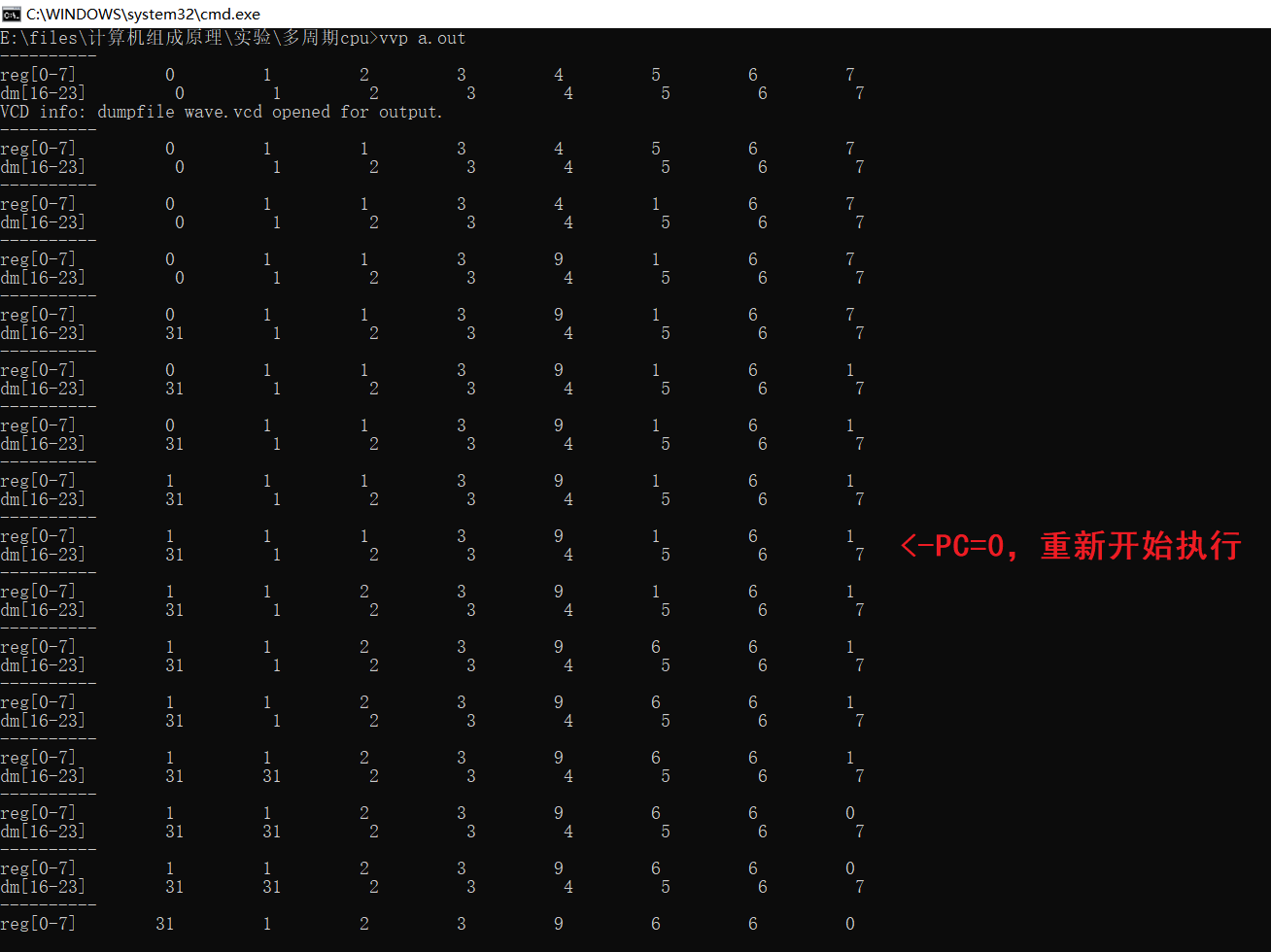
1. **实验测试**

在程序存储器中初始化一端小程序，如下所示，程序的作用在注释中给出，其中，在最后一条指令中PC被设置为1，从指令DM[0]开始重新执行，实验结果中仅给出额外执行一轮的结果。

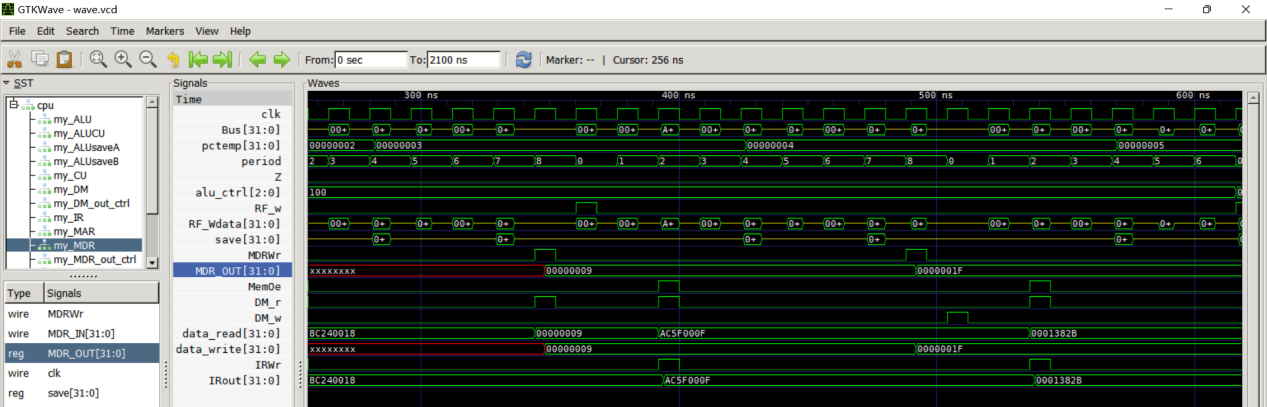


初始化时，将reg0~reg7分别赋值为0~7，将DM[16]~DM[24]分别赋值为0~7

在运行时，每过一个指令周期,打印这8个寄存器和8个主存的内容，运行结果如下图所示



可以看到，两轮执行的算数运算指令由于参与运算的值发生变化，得到的结果有所不同，产生了链锁反应，导致后续的存取数指令的地址有所变化，条件指令（判断大小和条件转移）的结果也有所不同。例如，在第一轮的指令中，reg0和reg1的值分别为0和1，reg2的值计算后为1，这导致第3条指令中的寻址过程发生变化（第一次是DM[16]被写入，这一次是DM[17]被写入）第4和第5条判断指令和条件转移指令都满足条件，而条件转移指令跳过了第6条指令，执行第7条指令，将reg0设置为1，随后在第8条指令中PC被设置为0，重新开始执行。这一次，reg2=reg0+reg1计算后值为2，再加上第一轮最后reg0的值被设为1，导致这一次判断大小指令和条件转移指令均不满足，执行第6条指令，将reg0设置为31，随后在第7条指令中，由于存储器中DM[17]的值已经在之前被设置为31，因此reg0的值仍然为31。其他所有的执行也都正确符合程序，一些关键环节的波形如下图所示。



图中截获了一些关键信息，例如ck信号，PC的值，period表示指令中期中时钟周期的序号，主存的读写时间等等，一下以第一次执行第5条指令为例，分析波形，见下页：

