常规指令：

I\_1型指令：与立即数的运算

指令结构：

1. ADDI指令：将地址为rs1的通用寄存器的值加上imm存入地址为rd的通用寄存器，imm的最高位为符号位，会扩展为32

指令：0000\_0000\_1001\_00001\_000\_00001\_0010011 即32’h0090\_8093

解释：将reg[1]（初值为0）加上4’b1001存入reg[1]（32’h0000\_0009）

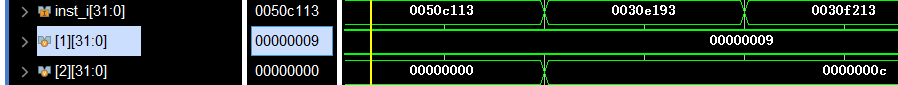


（仿真中的inst\_i取的是写回模块的inst\_i，所以结果会在inst\_i之后一个周期，下同）

1. XORI指令：将rs1与imm做异或运算，结果存入rd

指令：0000\_0000\_0101\_00001\_100\_00010\_0010011 32’h0050\_c113

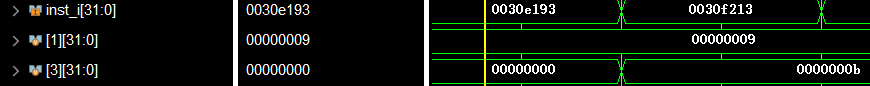
解释：将reg[1]（1001）与0101做异或运算存入reg[2]，结果应为1100（c)



1. ORI指令：将rs1与imm做或运算，结果存入rd

指令：0000\_0000\_0011\_00001\_110\_00011\_0010011 32’h0030\_e193

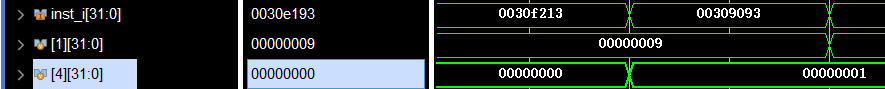
解释：将reg[1](1001)与(0011)做或运算存入reg[3]，结果应为1011（b）



1. ANDI指令：将rs1与imm做与运算，结果存入rd

指令：0000\_0000\_0011\_00001\_111\_00100\_0010011 32’h0030\_f213

解释：将reg[1]（1001）与0011做与运算，结果存入reg[4]（0001）



1. SLLI指令：rd = rs1 << imm [0:4]

指令：0000\_0000\_0011\_00001\_001\_00001\_0010011 32’h0030\_9093

指令解释：将reg[1](1001)逻辑左移3位存入reg[1]，结果为100\_1000(8'h48)



1. SRLI指令：rd = rs1 >> imm[0:4]，逻辑右移，高位补0

指令：0000\_0000\_0011\_00001\_101\_00001\_0010011 32’h0030d093

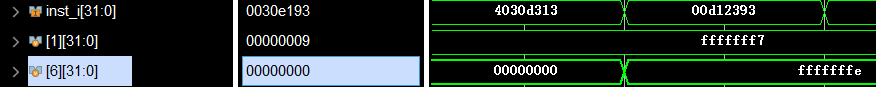
指令解释：将reg[1](100\_1000)逻辑右移3位存入reg[1],结果为1001（9）



1. SRAI指令：rd = rs1 >> imm[0:4]，算术右移，带符号位

指令：0000\_0000\_0011\_00001\_101\_00110\_0010011 32’h4030d313

指令解释:将reg[1]（32’hffff\_fff7）数字右移3位，带符号位存入reg[6]（32’hffff\_ffffe）



1. SLTI 指令：rd = (rs1 < imm)?1:0 有符号数比大小

指令：1111\_1111\_1000\_00001\_010\_00111\_0010011 32’hff80a393

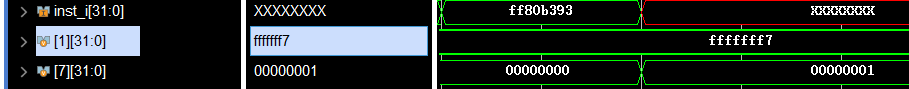
指令解释：将reg[1]（32’hffff\_fff7，即-9）与立即数32’hffff\_fff8（即-8）比较，小于则返回1存入reg[7]（返回1）。



1. SLTIU指令：rd = (rs1 < imm)?1:0 无符号数比大小

指令：1111\_1111\_1000\_00001\_011\_00111\_0010011 32’hff80b393

指令解释：将reg[1]（32’hffff\_fff7）与立即数32’hffff\_fff8比较，小于则返回1存入reg[7]（返回1）。



I\_2指令：读取数据，从RAM中读取值写回到rd中

1. LB指令：rd = M[rs1+imm][0:7]

0000\_0000\_0001\_00010\_000\_00011\_0000011 32’h00110183

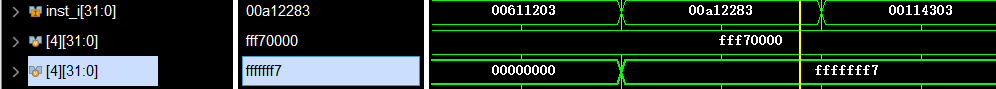
指令解释：ram地址的最后两位用来判断读取该地址寄存器的哪一节字节，这里rs1=reg[2]=1100，imm=1，rs1+imm=1101，即ram[3]的[15:8]，存入到reg[3]中。左边符号位扩展，结果为32’hffff\_fff7



1. LH指令：rd = M[rs1+imm][0:15]

0000\_0000\_0110\_00010\_001\_00100\_0000011 32‘h00611203

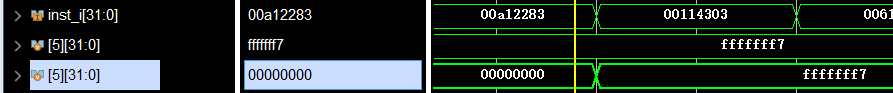
指令解释：同样最后的两位地址用来判断读取那两个字节，这里rs1+imm=01100+00110=10010，即读取ram[4]的[31:16]存入reg[4]。



1. LW指令：rd = M[rs1+imm][0:31]

0000\_0000\_1010\_00010\_010\_00101\_0000011 32’h00a12283

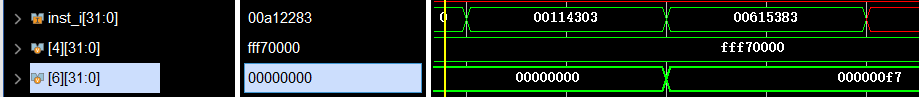
指令解释：读取整个寄存器数据，地址最后两位截断，rs1+imm=01100+01010=10110，即读取ram[5]存入reg[5]



1. LBU指令：rd = M[rs1+imm][0:7]，无符号读取，左扩展0

0000\_0000\_0001\_00010\_100\_00110\_0000011 32’h00114303

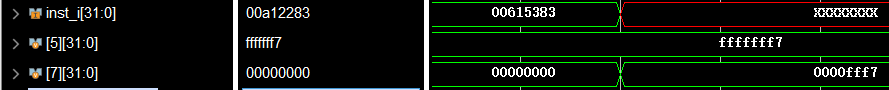
指令解释:与LB指令读取的数字相同，但是是无符号数，所以左边扩展为0。



1. LHU指令：rd = M[rs1+imm][0:15]，无符号读取，左扩展0。

0000\_0000\_0110\_00010\_101\_00111\_0000011 32’h00615838

指令解释：与LH读取数据相同，左扩展0。



R指令：将rs1与rs2之间操作的指令

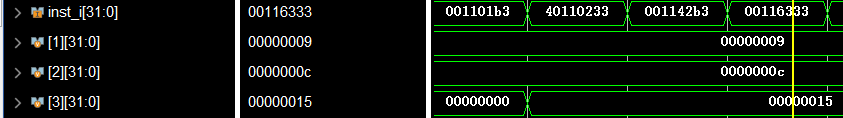
指令结构：



1. ADD: rd = rs1 + rs2

指令：0000000\_00001\_00010\_000\_00011\_0110011 32’h001101b3

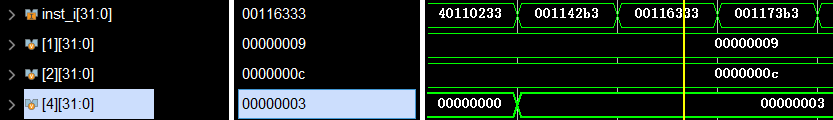
指令解释：reg[1](9)+reg[c](a)存入reg[3](8’h15)



1. SUB: rd = rs1 - rs2

指令：0100000\_00001\_00010\_000\_00100\_0110011 32’h40100233

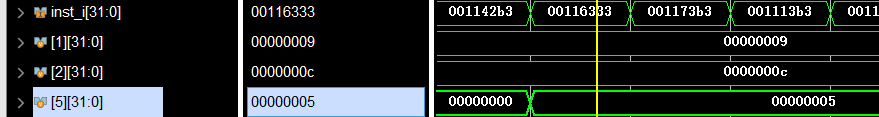
指令解释：reg[2](c)-reg[1](9)=reg[4](3)



1. XOR: rd = rs1 ˆ rs

指令：0000000\_00001\_00010\_100\_00101\_0110011 32’h001142b3

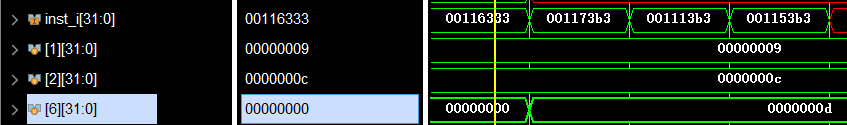
指令解释：reg[1](1001)^reg[2](1100)=reg[5](0101)



1. OR: rd = rs1 | rs2

指令：0000000\_00001\_00010\_110\_00110\_0110011 32’h00116333

指令解释：reg[1](1001)|reg[2](1100)=reg[6](1101)



1. AND: rd = rs1 & rs2

指令：0000000\_00001\_00010\_111\_00111\_0110011 32’h001173b3

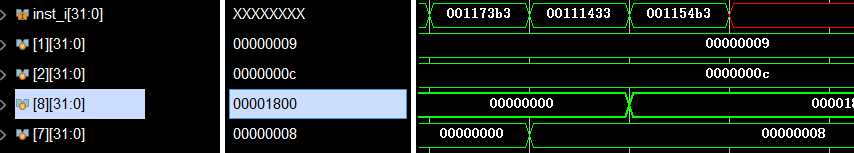
指令解释：reg[1](1001)&reg[2](1100)=reg[7](1000)



1. SLL: rd = rs1 << rs2

指令：0000000\_00001\_00010\_001\_01000\_0110011 32’h00111433

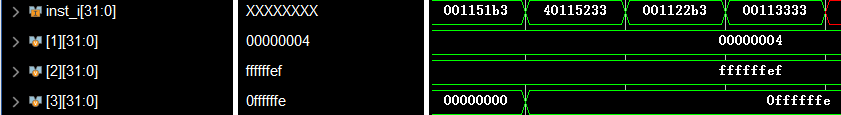
指令解释：将reg[2](32’h0000\_000c)左移reg[1](9)位，存入reg[8](32’h0000\_1800)



1. SRL: rd = rs1 >> rs2 ， 逻辑右移，高位补0

指令：0000000\_00001\_00010\_101\_00011\_0110011 32’h001151b3

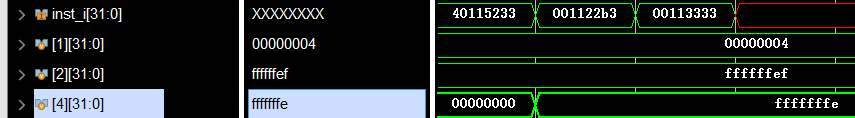
指令解释：将reg[2]（32’hffff\_ffef）右移reg[1](4)位，高位补0，存入reg[3]（32’h0fff\_fffe）



1. SRA: rd = rs1 >> rs2 ， 数字右移，高位补符号位

指令：0100000\_00001\_00010\_101\_00100\_0110011 32’h40115233

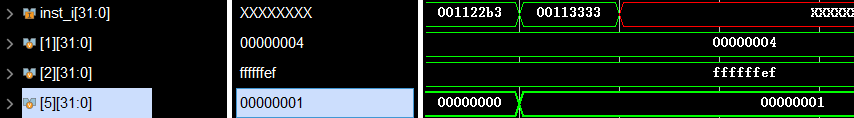
指令解释：将reg[2]右移reg[1]（4）位，高位补1，存入reg[4](32’hffff\_fffe)



1. SLT: rd = (rs1 < rs2)?1:0

指令：0000000\_00001\_00010\_010\_00101\_0110011 32’h001122b3

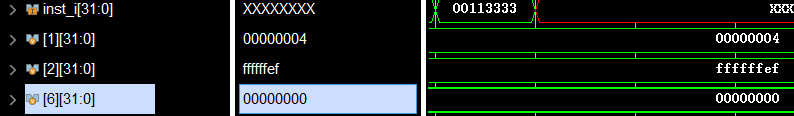
指令解释：在有符号数下，reg[2]（32’hffff\_ffef）为一负数，所以reg[2]<reg[1]，返回1存入reg[5]



1. SLTU: rd = (rs1 < rs2)?1:0 无符号比较 32’h00113333

指令：0000000\_00001\_00010\_010\_00101\_0110011

指令解释：无符号数下,reg[2]>reg[1]，返回0存入reg[6]。



S型：写回指令，将rs2的值写回到对应地址的ram中

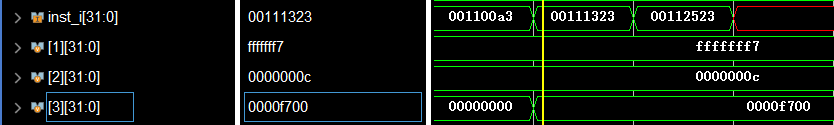
指令结构：



1. SB：M[rs1+imm][0:7] = rs2[0:7]

指令：0000000\_00001\_00010\_000\_00001\_0100011 即32’h001100a3

指令解释：在写回指令中，地址的后两位作为判断写回到ram中地址的哪几位。在SB指令中，如果rs1+imm的最后两位为00，则将rs2[0:7]写回到对应地址的最后8位，即[7:0]而其余位置上的结果不变。在测试指令中，rs1=reg[2]=4‘b1100，imm=4’b0001，所以rs1+imm=4’b1101，即写回到ram[3]的[15：8]位上。此处的rs2=reg[1]=32’hffff\_fff7，所以rs2[0:7]=8’hf7。所以ram[3]=32’h0000\_f700。

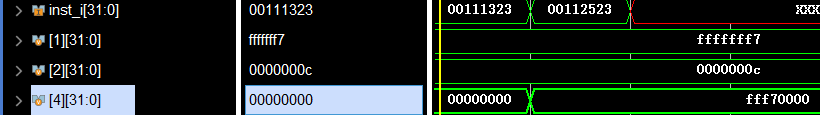


上图中1和2为reg[1]与reg[2]，3为ram[3]

1. SH: M[rs1+imm][0:15] = rs2[0:15]

指令: 0000000\_00001\_00010\_001\_00110\_0100011 32’h00111323

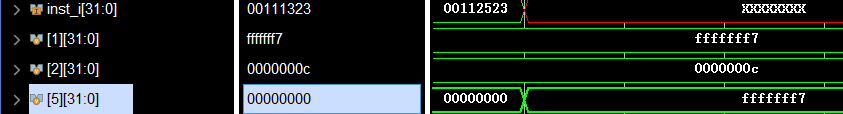
指令解释：与SB指令一样，rs1+imm的后两位选择写回的具体位置，如果rs1+imm[1:0]=2’b00，则写回到对应ram的后16位，其余写回到前十六位。这里reg[2]+imm=5’b01100+5’b00110=5’b10010，所以写回到ram[4]的前十六位。



1. SW: M[rs1+imm][0:31] = rs2[0:31]

指令：0000000\_00001\_00010\_010\_01010\_0100011 32’h00112523

指令解释：将整个rs2=reg[1]写回到rs1+imm=reg[2]+4’b1010=5‘b10110中，地址后两位没有意义直接截断。即写回到ram[5]中。



J型指令：无条件跳转指令：

指令结构:



1. JAL: rd = PC+4; PC += imm

指令：0\_0000000100\_0\_00000000\_00010\_1100111 32’h00800167

指令解释：pc跳转8，即跳过下一条指令。并将下一条指令的地址存到reg[2]

写回reg[2]的仿真图



跳过下一条指令的仿真图：

在执行模块接收到跳转指令后，由于pc\_id已经读取了跳转指令的下一条指令32’h00118193，这条指令需要被清零，所以在pc\_id模块受到的指令被插入一条空指令清零，下反应在id\_ex上就是在跳转之后空两个周期直接进行跳转之后的指令。

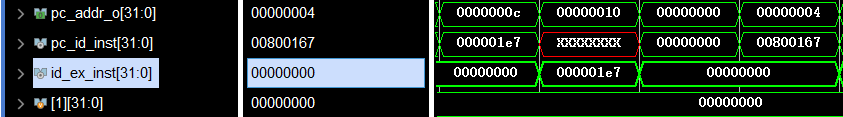


1. JALR: rd = PC+4; PC = rs1 + imm

指令：0000\_0000\_0000\_00000\_000\_00011\_1100111 32’h000001e7

指令解释：该指令为I型指令，格式与I型指令一样。将下一条指令地址存入reg[3]并且跳转到初始位置。

指令跳转



在id\_ex接收到跳转指令32’h000001e7之后，pc计数被归零

写回寄存器：



B型指令：有条件跳转指令：

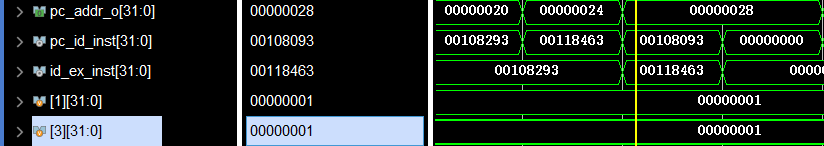
指令结构：



1. BEQ: if(rs1 == rs2) PC += imm

指令：0000000\_00001\_00011\_000\_01000\_1100011 32’h00118463

指令解释：rs1=reg[3]=1，rs2=reg[1]=1，所以rs1==rs2，会跳过下一条指令

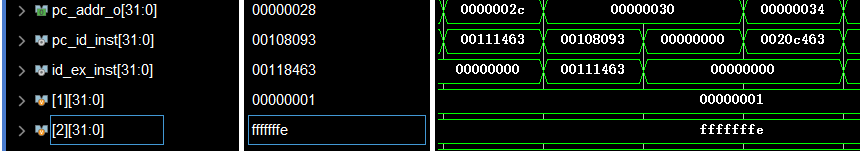


由于满足跳转条件，BEQ指令的地址为32’h00000020，加上imm=8后等于32’h00000028，由于当ex执行到跳转指令时pc正好记到32’h00000028，所以相当于下一次计数回到0028，即0028计数两次，而在跳转指令的下一条指令32’h00108093被跳过。

1. BNE: if(rs1 != rs2) PC += imm

指令: 0000000\_00001\_00010\_001\_01000\_1100011 32’h00111463

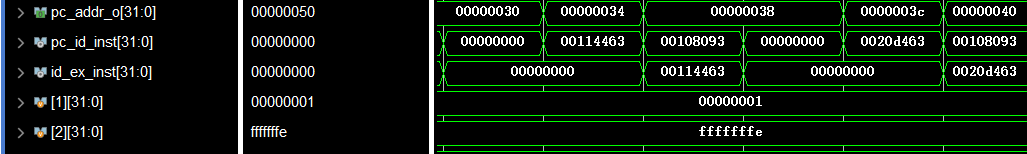
指令解释：rs2=reg[2]=32’hffff\_fffe，即-2，rs1=reg[1]=1，rs1！=rs2，PC+8，跳过下一条指令。



1. BLT: if(rs1 < rs2) PC += imm

指令：0000000\_00001\_00010\_100\_01000\_1100011 32’h00114463

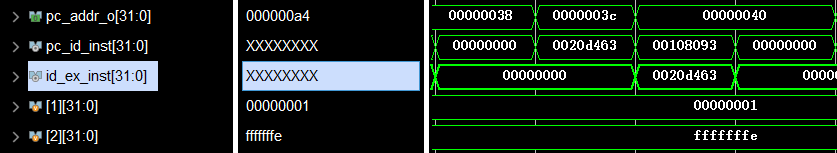
指令解释：rs1=reg[2]=32’hffff\_fffe=-2，rs2=reg[1]=1，所以rs1<rs2，应该跳转。



1. BGE: if(rs1 >= rs2) PC += imm

指令：0000000\_00010\_00001\_101\_01000\_1100011 32’h0020d463

指令解释：rs1=reg[1]=1，rs2=reg[2]=-2，rs1>rs2，跳转



1. BLTU: if(rs1 < rs2) PC += imm

指令：0000000\_00010\_00001\_110\_01000\_1100011 32’h0020e463

指令解释：rs2=reg[2]=32’hffff\_fffe，rs1=reg[1]=1，无符号数下rs1<rs2，所以应该跳转



1. BGEU: if(rs1 >= rs2) PC += imm

指令：0000000\_00001\_00010\_111\_01000\_1100011 32’h00117463

指令解释：rs1=reg[2]=32’hffff\_fffe，rs2=reg[1]=1，无符号数下rs1>=rs2，所以应该跳转



U型指令：立即数位移指令

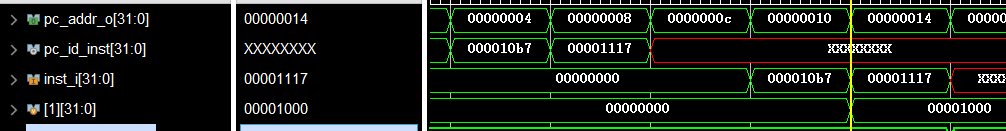
指令结构：



1. LUI: rd = imm << 12

指令：0000\_0000\_0000\_0000\_0001\_00001\_0110111 32’h000010b7

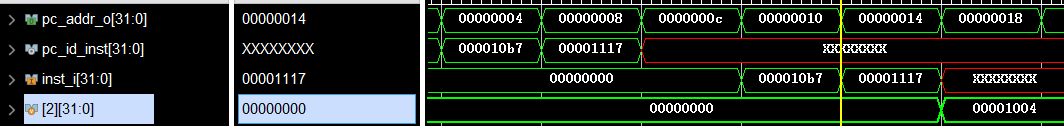
指令解释：将1左移12位存入reg[1],结果为32’h00001000



1. AUIPC: rd = PC + (imm << 12)

指令：0000\_0000\_0000\_0000\_0001\_00010\_0010111 32’h00001117

指令解释：将这条指令的地址（4）加上左移十二位的1，存入reg[2]，结果为32’h00001004



扩展指令：

1. 乘法指令：

0000001\_00011\_00010\_000\_10000\_0110011

MULU（无符号乘法）:将reg[2](c)乘以reg[3](b)结果存入reg[16](8‘h84)

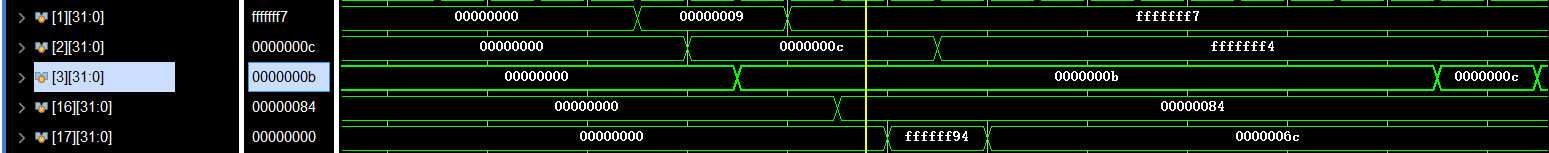
0000001\_00001\_00010\_001\_10001\_0110011

MUL（有符号乘法）:将reg[1](-9)乘以reg[2](c)结果存入reg[17](-8'h94,即32'hffff\_ff94)

1111\_1110\_1000\_00010\_000\_00010\_0010011

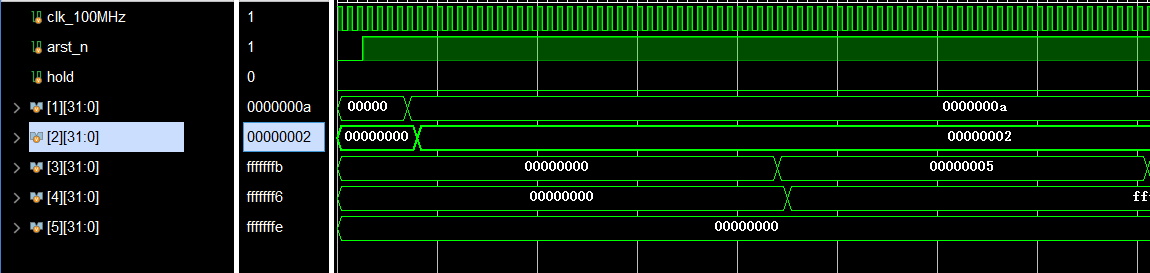
ADDI:将reg[2](c)减去2倍的c，得到-reg[2](-c,即32‘hffff\_fff4),并存入reg[2]

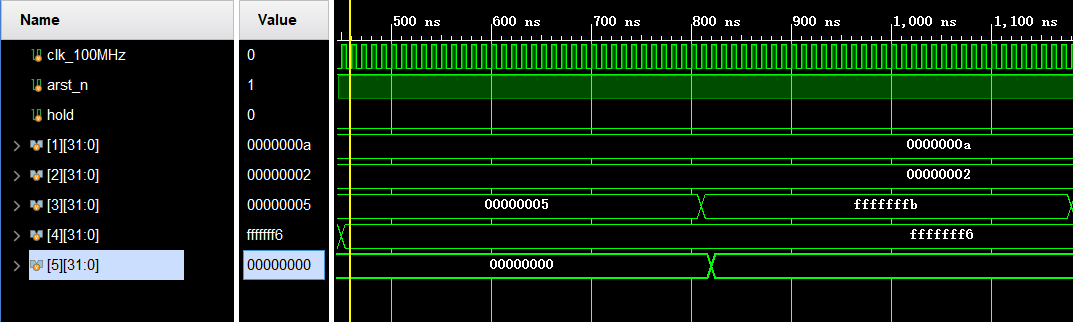
0000001\_00001\_00010\_001\_10001\_0110011

MUL（有符号乘法）:将reg[2](-c)与reg[1](-9)有符号相乘存入reg[17]，得到结果(8'h6c)

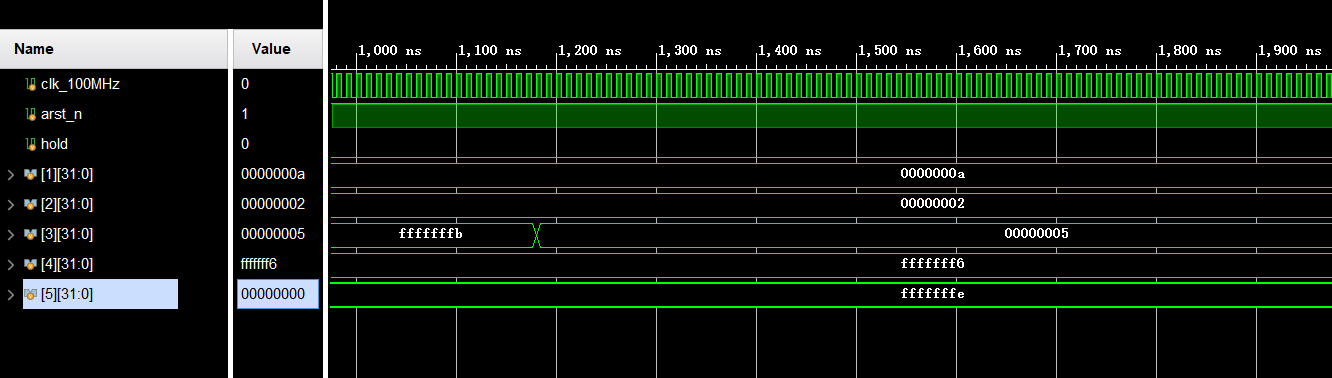
2. 除法指令：

DIVU（无符号除法）: 除法reg[1]（a）除以reg[2](2)，结果存入reg[3](5)，需要执行33个周期



DIV（有符号除法）：除法reg[4](-10)除以reg[2](2)结果存入reg[3](32’hffff\_fffb，即-5)

DIV（有符号除法）：除法reg[4](-10)除以reg[5](-2)结果存入reg[3]（5）



冒险的仿真：

数据冒险：

1. 如果上一条指令为对通用寄存器的reg进行了操作，下一条指令需要读取该寄存器的值，但是上一条指令的操作结果还未写回的冒险。此类冒险通过数据前置解决，不需要暂停流水线

测试：

0000\_0000\_1001\_00001\_000\_00001\_0010011 //reg[1]加9

0000\_0000\_1001\_00001\_000\_00001\_0010011 //reg[1]加9

0000\_0000\_1001\_00001\_000\_00001\_0010011 //reg[1]加9

0000\_0000\_1001\_00001\_000\_00001\_0010011 //reg[1]加9

1. 当此刻为访存型指令（如lw），上一条为写回指令（如sw）且写回和访存是同一个地址的Ram时，同样只需要在访存阶段数据前移，不需要暂停流水线。

控制冒险:上一条指令为访存指令，下一指令对访存存入的寄存器操作，此时需要暂停流水线一个周期。

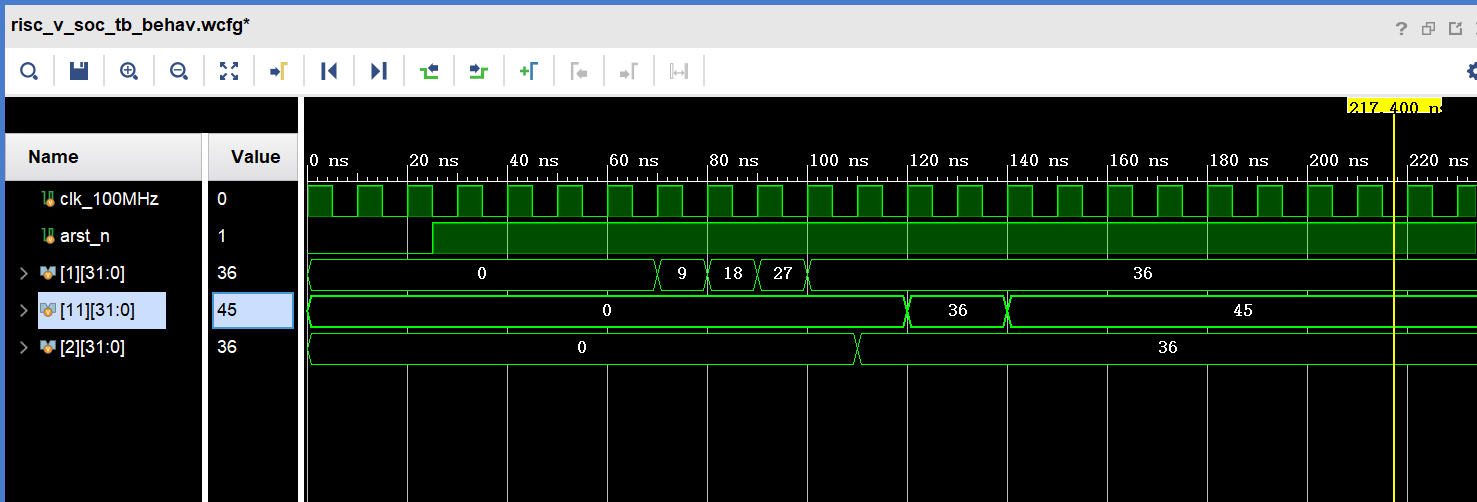
测试：

0000000\_00001\_00010\_010\_01010\_0100011 //把reg[1]的数据写到ram[2]中

0000\_0000\_1010\_00010\_010\_01011\_0000011 //把ram[2]的数据写入reg[11]中

0000\_0000\_1001\_01011\_000\_01011\_0010011 //reg[11]加9

仿真结果：



可以看到对reg[1]的连续加操作以及对Ram[2]的写回读取操作没有暂停流水线，并得到正确结果。而对将ram[2]的值存入reg[11]，然后立即对reg[11]进行加操作的时候，reg[11]在36停留了两个周期，通过暂停流水线解决了冒险。