电子科技大学

计算机专业类课程

实验报告

课程名称：计算机系统结构实验

学　　院：计算机科学与工程

专　　业：计算机科学与技术

学生姓名：

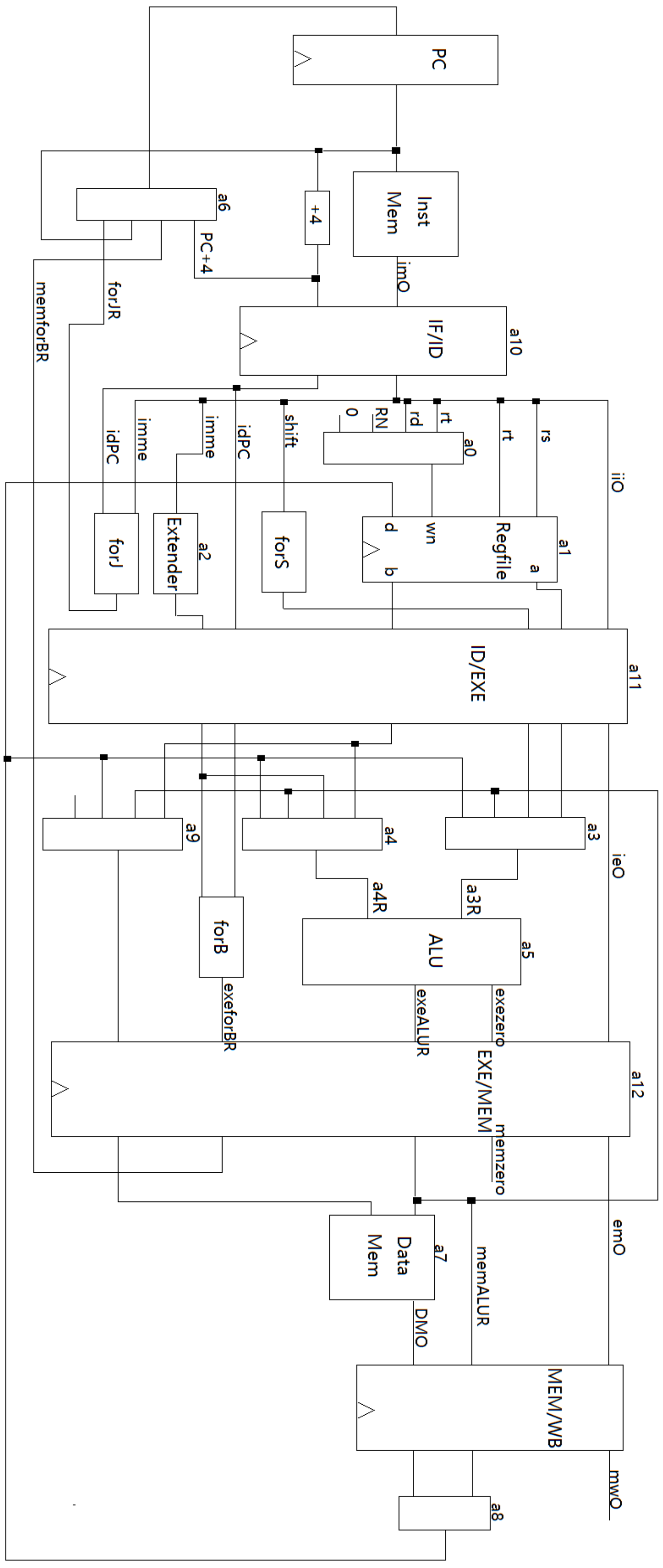
学　　号：

指导教师：

日　　期：　2016 年　5　月　20　日

系统结构第三次试验报告

1. 不含控制器的 CPU 电路图



该 CPU 实现的指令集：

指令	指令意义	Op[31:26]	Op2 [25:20]	[19:15]	[14:10]	[9:5]	[4:0]
add	寄存器加法	000000	000001	00000	rd	rs	rt
and	寄存器与	000001	000001	00000	rd	rs	rt
or	寄存器或	000001	000010	00000	rd	rs	rt
xor	寄存器异或	000001	000100	00000	rd	rs	rt
sra	算术右移	000010	000001	shift	rd	00000	rt
srl	逻辑右移	000010	000010	shift	rd	00000	rt
sll	逻辑左移	000010	000011	shift	rd	00000	rt
addi	立即数加法	000101	16 位 immediate			rs	rt
andi	立即数与	001001	16 位 immediate			rs	rt
ori	立即数或	001010	16 位 immediate			rs	rt
xori	立即数异或	001100	16 位 immediate			rs	rt
load	取整数数据字	001101	16 位 offset			rs	rt
store	存整数数据字	001110	16 位 offset			rs	rt
beq	相等则跳转	001111	16 位 offset			rs	rt
bne	不相等则跳转	010000	16 位 offset			rs	rt
jump	无条件跳转	010010	26 位 address				

Op 和 Op2 为操作码；
shift 保存要移位的位数；
rd、rs、rt 分别为寄存器的寄存器号；
immediate 保存立即数的低 16 位；
offset 为偏移量；
address 为转移地址的一部分。

1、对于 add/sub/mul/and/or/xor　rd,rs,rt 指令　//rd←rs　op　rt
其中 rs 和 rt 是两个源操作数的寄存器号，rd 是目的寄存器号。

2、对于 sll/srl/sra　rd,rt,shift 指令　//rd←rt　移动　shift 位

3、对于 addi/muli　rt,rs,imm 指令　//rt←rs+imm(符号拓展)
rt 是目的寄存器号，立即数要做符号拓展到 32 位。

4、对于 andi/ori/xori　rt,rs,imm 指令　//rt←rs　op　imm(零拓展)
因为是逻辑指令，所以是零拓展。

5、对于 load　rt,offset(rs)　指令　//rt← memory[rs+offset]
load 是一条取存储器字的指令。寄存器 rs 的内容与符号拓展的 offset 想加，得到存储器地址。从存储器取来的数据存入 rt 寄存器。

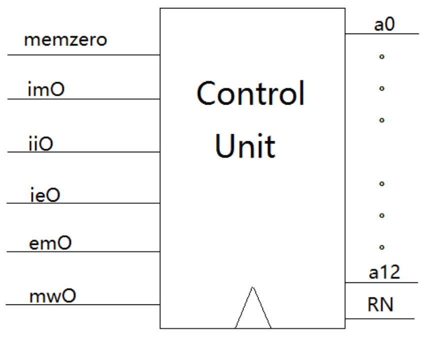
6、对于 store　rt,offset(rs)　指令　// memory[rs+offset] ← rt
store 是一条存字指令。存储器地址的计算方法与 load 相同。

7、对于 beq　rs,rt,label 指令　//if(rs==rt)　PC←label beq 是一条条件转移指令。当寄存器 rs 内容与 rt 相等时，转移到 label。如果程序计数器 PC 是 beq 的指令地址，则 label=PC+4+offset<<2。offset 左移两位导致 PC 的最低两位永远是 0，这是因为 PC 是字节地址，而一条指令要占 4 个字节。offset 要进行无符号拓展。

8、bne 指令去 beq 类似，但是是在寄存器 rs 内容与 rt 不相等时，转移到 label。

9、对于 jump　target 指令　//PC←target
jump 是一条跳转指令。target 是转移的目标地址，32 位，由 3 部分组成：最高 4 位来自于 PC+4 的高 4 位，中间 26 位是指令中的 address，最低两位为 0。

2. 控制器的设计及实现
控制器示意图如下：



控制器的输入端：
imO：指令存储器的输出，32 位；
iiO：流水线寄存器 IF/ID 的输出之一，32 位，属于寄存器型变量，输入端为 imO；
ieO：流水线寄存器 ID/EXE 的输出之一，32 位，属于寄存器型变量，输入端为 iiO；
emO：流水线寄存器 EXE/MEM 的输出之一，32 位，属于寄存器型变量，输入端为 ieO；
mwO：流水线寄存器 MEM/WB 的输出之一，32 位，属于寄存器型变量，输入端为 emO；
memzero：流水线寄存器 EXE/MEM 的输出之一；置 1 表示 ALU 的运算结果为 0；
控制器的输出端：
a0——a12、RN：控制单元的 14 个输出信号：
RN：由于寄存器堆在前半个周期写寄存器，此时对应的寄存器号在 emO 中，且不确定是否为 rd 位置还是 rt 位置，此时就需要 Control Unit 处理，若为 add~srl 指令则打回 rd 寄存器代表位置[14:10]，若为 addi~load 则打回 rt 代表位置[4:0]，其余情况置 0；
a0：4 路 5 位多选器选择信号，结果应随时钟信号变化(时钟为高电平时为前半周期，有可能需要装入 iiO 中 rt 位置或 rd 位置的寄存器号，此时寄存器可写不可读；时钟为低电平时处于后半周期，有可能需要写回 WB 级的多选器结果，此时读寄存器)；
a1：寄存器写/读信号，其中写为高电平，读为低电平；
a2：符号扩展器信号，置 1 时作符号扩展，置 0 时作无符号扩展；

a3：4 路 32 位多选器选择信号，置 0 时选择 exeQa，置 1 时选择 forSR，置 2 时选择 memALUR，置 3 时选择 a8R；

a4：4 路 32 位多选器选择信号，置 0 时选择 exeQb，置 1 时选择 ExR，置 2 时选择 memALUR，置 3 时选择 a8R；

a5：ALU 控制信号，000 表加法，001 表减法，010 表与运算，011 表或运算，100 表异或运算，101 表算术右移，110 表逻辑右移，111 表逻辑左移（也即算术左移）；

a6: 在执行 beq 指令时, 当 zero 为 0 时 a6 选择 01 处的无符号扩展后的立即数 forBR 进行跳转, 在执行 bne 指令时, 当 zero 不为 0 时 a6 选择 01 处的无符号扩展后的立即数 forBR 进行跳转(条件分支指令跳转后，IF/ID、ID/EXE、EXE/MEM 清零，中间损失了 3 个时钟周期；另外需要注意，beq、bne 指令执行时，立即数作的是无符号扩展而非符号扩展，原因是本 CPU 在设计 PC 时选择 PC 的[7：2]位进行对指令存储器中存储单元的选择，不需要进行符号扩展)，当 iiO 为 jump 指令时，直接转到 PC 选择 forJR，清零 IF/ID、ID/EXE，浪费 2 个时钟周期；

a7：数据存储器写信号，置 0 表示可读不可写，置 1 表示可写不可读；

a8：位于 WB 级的 2 路 32 位多选器选择信号，置 0 时选择 wbALUR，置 1 时选择 wbDMO；

a9：位于 EXE 级的 4 路 32 位多选器选择信号，该多选器用于在 EXE 级的 store 指令的 rt 寄存器与 MEM 级的 memALUR（即 MEM 级指令更改了 EXE 级中 store 指令 rt 寄存器的值）或 WB 级的 a8R（即 WB 级指令更改了 EXE 级中 store 指令 rt 寄存器的值）产生冒险时进行数据前推；当 a9 置 0 时，代表无冒险，正常打入 ID 级中的 idQb；当 a9 置 1 时，代表与 MEM 级指令产生了数据冒险，打入 memALUR；当 a9 置 2 时，代表与 WB 级指令产生了数据冒险，打入 a8R；剩余一个引脚置空；

a10：IF/ID 锁存器控制信号（当 a10 置 0 时锁存器左端的输入信号打入输出端；当 a10 置 1 或 3（本次试验默认置 1，3 未出现）时锁存器保持上个时钟周期的输出；当 a10 置 2 时起到 Reset 作用，锁存器内容清零）；

a11：ID/EXE 锁存器控制信号（当 a11 置 0 时锁存器左端的输入信号打入输出端；当 a11 置 1 或 3（本次试验默认置 1，3 未出现）时锁存器保持上个时钟周期的输出；当 a11 置 2 时起到 Reset 作用，锁存器内容清零）；

a12：EXE/MEM 锁存器控制信号（当 a12 置 0 时锁存器左端的输入信号打入输出端；当 a12 置 1 或 3（本次试验默认置 1，3 未出现）时锁存器保持上个时钟周期的输出；当 a12 置 2 时起到 Reset 作用，锁存器内容清零）；

由于在实际情况下，CPU 不能控制 Reset 信号，Reset 一般为使用者手动控制，故该工程中每个具有保存输入信号功能的器件都有一个统一的 Reset 信号；而对于 IF/ID 锁存器、ID/EXE 锁存器和 EXE/MEM 锁存器，它们在产生 load 暂停时需要清零，且该信号应由 Control Unit 控制，故这三个锁存器即有 Reset 信号，又在对应的控制单元控制信号置 2 时清零，从而也起到了 Reset 的作用；

由于 MEM/WB 锁存器不需要对数据冒险做出反应（与未发生数据冒险时功能完全相同），故该锁存器无控制信号；

控制单元的实现：

```
module ControlUnit(
     input zero,clk,
        input [31:0] iiO,ieO,emO,mwO,
        output a1,a2,a7,a8,a12,
        output [2:0] a5,
        output [1:0] a0,a3,a4,a9,a10,a11,a6,
        output [4:0] RegNumber
    );
        wire a,b,c;
    assign a0 = (clk & (mwO[31:26] < 14)) ? 2 : //A|B|C
                        ((~clk) & (iiO[31:26] < 3)) ? 1 : //A
                                ((~clk) & (iiO[31:26] > 4) & (iiO[31:26] < 14)) ? 0 : 3; //B and C

        assign a1 = (clk & (mwO[31:26] < 14)) ? 1 : //A|B|C
                        ((~clk) & (iiO[31:26] < 17)); //A|B|C|D|E

        assign a2 = (iiO[28] & ~iiO[27] & iiO[26]) | (iiO[28] & iiO[27] & ~iiO[26]); //addi load|store
        assign a7 = (emO[31:26] == 13) ? 1 : 0; //store
        assign a8 = (mwO[31:26] == 14) ? 1 : 0; //load

        assign a5[2] = (~ieO[30] & ~ieO[29] & ieO[27]) | (~ieO[29] & ~ieO[28] & ieO[26] & ieO[22]) | (ieO[28]
& ~ieO[27] & ~ieO[26]); //sra srl sll|xor|xori
        assign a5[1] = (~ieO[30] & ~ieO[29] & ieO[27] & ieO[21]) | (~ieO[29] & ~ieO[28] & ieO[26] &
~ieO[22]) | (ieO[29] & ~ieO[28]); //srl sll|and or|andi ori
    assign a5[0] = (ieO[28] & ieO[27] & ieO[26]) | (ieO[30] & ~ieO[27]) | (ieO[29] & ~ieO[28] & ieO[27]) |
(~ieO[30] & ~ieO[29] & ~ieO[28] & ieO[27] & ieO[20]) |
                            (~ieO[29] & ~ieO[28] & ieO[26] & ieO[21]); //beq|bne|ori|sra sll|or

    assign a = ~(((ieO[31:26] == 1) & (emO[31:26] < 3) & (ieO[9:5] == emO[14:10])) | //AAX and not shift
                                ((ieO[31:26] == 1) & (emO[31:26] > 4) & (emO[31:26] < 13) & (ieO[9:5] ==
emO[4:0])) | //ABX and not shift
                                ((ieO[31:26] > 4) & (ieO[31:26] < 14) & (emO[31:26] < 3) & (ieO[9:5] ==
emO[14:10])) | //BAX|CAX
                                ((ieO[31:26] > 4) & (ieO[31:26] < 14) & ((emO[31:26] > 4) & (emO[31:26] < 13))
& (ieO[9:5] == emO[4:0])) | //BBX|CBX
                                ((ieO[31:26] == 14) & (emO[31:26] < 3) & (ieO[9:5] == emO[14:10])) | //DAX
and ieO(rs)=emO(rd)
                                ((ieO[31:26] == 14) & (emO[31:26] > 4) & (emO[31:26] < 13) & (ieO[9:5] ==
emO[4:0])) | //DBX and ieO(rs)=emO(rt)
                                (((ieO[31:26] == 15) | (ieO[31:26] == 16)) & (emO[31:26] < 3) & (ieO[9:5] ==
emO[14:10])) | //EAX
                                (((ieO[31:26] == 15) | (ieO[31:26] == 16)) & ((emO[31:26] > 4) & (emO[31:26] <
13)) & (ieO[9:5] == emO[4:0]))); //EBX

        assign b = ~(((ieO[31:26] < 3) & (emO[31:26] < 3) & (ieO[4:0] == emO[14:10])) | //AAX
                                 ((ieO[31:26] < 3) & (emO[31:26] > 4) & (emO[31:26] < 13) & (ieO[4:0] ==
emO[4:0])) | //ABX
```

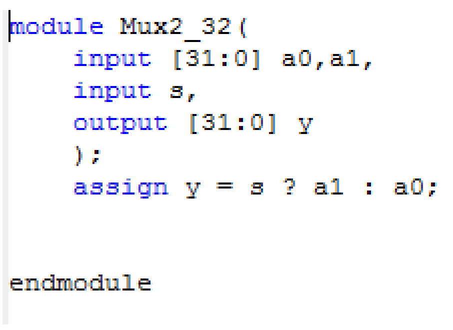
```
                                        (((ieO[31:26] == 15) | (ieO[31:26] == 16)) & (emO[31:26] < 3) & (ieO[4:0] ==
emO[14:10])) | //EAX
                                        (((ieO[31:26] == 15) | (ieO[31:26] == 16)) & ((emO[31:26] > 4) & (emO[31:26]
< 13)) & (ieO[4:0] == emO[4:0])));

         assign c = ~(((ieO[31:26] == 14) & (emO[31:26] < 3) & (ieO[4:0] == emO[14:10])) | //DAX and
ieO(rt)=emO(rd)
                                        ((ieO[31:26] == 14) & ((emO[31:26] > 4) & (emO[31:26] < 13)) & (ieO[4:0] ==
emO[4:0]))); //DBX and ieO(rt)=emO(rt)

         assign a6 = (iiO[30] & iiO[27]) ? 3 : //jump

                                       (((ieO[31:26] < 3) & (emO[31:26] == 13) & ((ieO[9:5] == emO[4:0]) | (ieO[4:0]
== emO[4:0]))) | //ACX
                                       (((ieO[31:26] > 4) & (ieO[31:26] < 14)) & (emO[31:26] == 13) & (ieO[9:5] ==
emO[4:0])) | //BCX|CCX
                                       ((ieO[31:26] == 14) & (emO[31:26] == 13) & ((ieO[9:5] == emO[4:0]) | (ieO[4:0]
== emO[4:0]))) | //DCX
                                       (((ieO[31:26] == 15) | (ieO[31:26] == 16)) & (emO[31:26] == 13) & ((ieO[9:5] ==
emO[4:0]) | (ieO[4:0] == emO[4:0]))) | //ECX
                                      (iiO[31:26] == 15) | (iiO[31:26] == 16) | (ieO[31:26] == 15) | (ieO[31:26] == 16)) ?
2 : //possible to branch

                                       ((zero & (emO[31:26] == 15)) | (~zero & (emO[31:26] == 16))) ? 1 : 2'b00;
//enable to branch

         assign a3 = a & (((ieO[31:26] < 3) & (mwO[31:26] < 3) & (ieO[9:5] == mwO[14:10])) | //AXA and not
shift
                                ((ieO[31:26] < 3) & ((mwO[31:26] > 4) & (mwO[31:26] < 14)) & (ieO[9:5] == mwO[4:0]))
| //(AXB or AXC) and not shift
                                       ((ieO[31:26] > 4) & (ieO[31:26] < 14) & (mwO[31:26] < 3) & (ieO[9:5] ==
mwO[14:10])) | //BXA or CXA
                                      ((ieO[31:26] > 4) & (ieO[31:26] < 14) & ((mwO[31:26] > 4) & (mwO[31:26] < 14))
& (ieO[9:5] == mwO[4:0])) | //BXB|BXC|CXB|CXC
                                       ((ieO[31:26] == 14) & (mwO[31:26] < 3) & (ieO[9:5] == mwO[14:10])) | //DXA
and ieO(rs)=mwO(rd)
                                       ((ieO[31:26] == 14) & ((mwO[31:26] > 4) & (mwO[31:26] < 14)) & (ieO[9:5] ==
mwO[4:0])) | //(DXB|DXC) and ieO(rs)=mwO(rt)
                                       (((ieO[31:26] == 15) | (ieO[31:26] == 16)) & (mwO[31:26] < 3) & (ieO[9:5] ==
mwO[14:10])) | //EXA
                                       (((ieO[31:26] == 15) | (ieO[31:26] == 16)) & (mwO[31:26] > 4) & (mwO[31:26] <
14) & (ieO[9:5] == mwO[4:0]))) ? 3 : //EXB|EXC

                                       (((ieO[31:26] == 1) & (emO[31:26] < 3) & (ieO[9:5] == emO[14:10])) | //AAX
and not shift
                                        ((ieO[31:26] == 1) & (emO[31:26] > 4) & (emO[31:26] < 13) & (ieO[9:5] ==
emO[4:0])) | //ABX and not shift
                                        ((ieO[31:26] > 4) & (ieO[31:26] < 14) & (emO[31:26] < 3) & (ieO[9:5] ==
emO[14:10])) | //BAX|CAX
                                        ((ieO[31:26] > 4) & (ieO[31:26] < 14) & ((emO[31:26] > 4) & (emO[31:26] < 13))
& (ieO[9:5] == emO[4:0])) | //BBX|CBX
                                        ((ieO[31:26] == 14) & (emO[31:26] < 3) & (ieO[9:5] == emO[14:10])) | //DAX
and ieO(rs)=emO(rd)
                                        ((ieO[31:26] == 14) & (emO[31:26] > 4) & (emO[31:26] < 13) & (ieO[9:5] ==
emO[4:0])) | //DBX and ieO(rs)=emO(rt)
                                        (((ieO[31:26] == 15) | (ieO[31:26] == 16)) & (emO[31:26] < 3) & (ieO[9:5] ==
emO[14:10])) | //EAX
                                        (((ieO[31:26] == 15) | (ieO[31:26] == 16)) & ((emO[31:26] > 4) & (emO[31:26] <
13)) & (ieO[9:5] == emO[4:0]))) ? 2 : //EBX

                                        (ieO[31:26] == 2) ? 1 : 0; //shift

      assign a4 = b & (((ieO[31:26] < 3) & (mwO[31:26] < 3) & (ieO[4:0] == mwO[14:10])) | //AXA
                                  ((ieO[31:26] < 3) & ((mwO[31:26] > 4) & (mwO[31:26] < 14)) & (ieO[4:0] == mwO[4:0]))
| //AXB|AXC
                                         (((ieO[31:26] == 15) | (ieO[31:26] == 16)) & (mwO[31:26] < 3) & (ieO[4:0] ==
mwO[14:10])) | //EXA
                                         (((ieO[31:26] == 15) | (ieO[31:26] == 16)) & ((mwO[31:26] > 4) & (mwO[31:26]
< 14)) & (ieO[4:0] == mwO[4:0]))) ? 3 : //EXB|EXC

                                         (((ieO[31:26] < 3) & (emO[31:26] < 3) & (ieO[4:0] == emO[14:10])) | //AAX
                                         ((ieO[31:26] < 3) & ((emO[31:26] > 4) & (emO[31:26] < 13)) & (ieO[4:0] ==
emO[4:0])) | //ABX
                                         (((ieO[31:26] == 15) | (ieO[31:26] == 16)) & (emO[31:26] < 3) & (ieO[4:0] ==
emO[14:10])) | //EAX
                                         (((ieO[31:26] == 15) | (ieO[31:26] == 16)) & ((emO[31:26] > 4) & (emO[31:26]
< 13)) & (ieO[4:0] == emO[4:0]))) ? 2 : //EBX

                                         ((ieO[31:26] > 2) & (ieO[31:26] < 15)) ? 1 : 0; //instructions which include an
imme

      assign a9 = c & (((ieO[31:26] == 14) & (mwO[31:26] < 3) & (ieO[4:0] == mwO[14:10])) | //DXA and
ieO(rt)=mwO(rd)
                                   ((ieO[31:26] == 14) & ((mwO[31:26] > 4) & (mwO[31:26] < 14)) & (ieO[4:0] ==
mwO[4:0]))) ? 2 : //(DXB|DXC) and ieO(rt)=mwO(rt)

                                         (((ieO[31:26] == 14) & (emO[31:26] < 3) & (ieO[4:0] == emO[14:10])) | //DAX
and ieO(rt)=emO(rd)
```

```
                                        ((ieO[31:26] == 14) & ((emO[31:26] > 4) & (emO[31:26] < 13)) & (ieO[4:0] ==
emO[4:0]))) ? 1 : 0; //DBX and ieO(rt)=emO(rt)

          assign a10 = ((zero & (emO[31:26] == 15)) | (~zero & (emO[31:26] == 16)) | //enable to branch
                              (iiO[30] & iiO[27])) ? 2 : //jump

                                            (((ieO[31:26] < 3) & (emO[31:26] == 13) & ((ieO[9:5] == emO[4:0]) |
(ieO[4:0] == emO[4:0]))) | //ACX
                                            (((ieO[31:26] > 4) & (ieO[31:26] < 14)) & (emO[31:26] == 13) & (ieO[9:5]
== emO[4:0])) | //BCX|CCX
                                            ((ieO[31:26] == 14) & (emO[31:26] == 13) & ((ieO[9:5] == emO[4:0]) |
(ieO[4:0] == emO[4:0]))) | //DCX
                                           (((ieO[31:26] == 15) | (ieO[31:26] == 16)) & (emO[31:26] == 13) & ((ieO[9:5]
== emO[4:0]) | (ieO[4:0] == emO[4:0])))) ? 1 : 0; //ECX

          assign a11 = ((zero & (emO[31:26] == 15)) | (~zero & (emO[31:26] == 16)) | (iiO[30] & iiO[27])) ? 2 :
//enable to branch|jump

                                            (((ieO[31:26] < 3) & (emO[31:26] == 13) & ((ieO[9:5] == emO[4:0]) |
(ieO[4:0] == emO[4:0]))) | //ACX
                                            (((ieO[31:26] > 4) & (ieO[31:26] < 14)) & (emO[31:26] == 13) & (ieO[9:5]
== emO[4:0]))    | //BCX|CCX
                                            ((ieO[31:26] == 14) & (emO[31:26] == 13) & ((ieO[9:5] == emO[4:0]) |
(ieO[4:0] == emO[4:0]))) | //DCX
                                           (((ieO[31:26] == 15) | (ieO[31:26] == 16)) & (emO[31:26] == 13) & ((ieO[9:5]
== emO[4:0]) | (ieO[4:0] == emO[4:0])))) ? 1 : 0;//ECX

          assign a12 = ((zero & (emO[31:26] == 15)) | (~zero & (emO[31:26] == 16))) ? 1 : 0;//enable to
branch

          assign RegNumber = (mwO[31:26] < 3) ? mwO[14:10] : //A
                                        (mwO[31:26] < 14) ? mwO[4:0] : 0; //B|C

endmodule
```

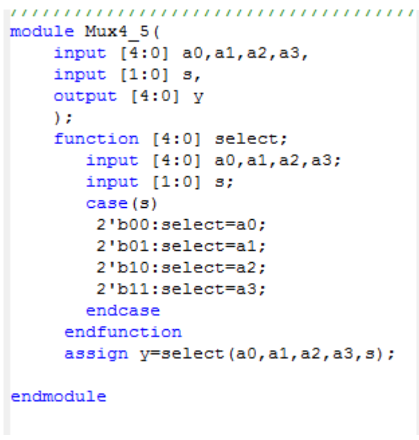
3.CPU 其他部件代码展示

2 路 32 位多选器、4 路 32 位多选器、4 路 5 位多选器:

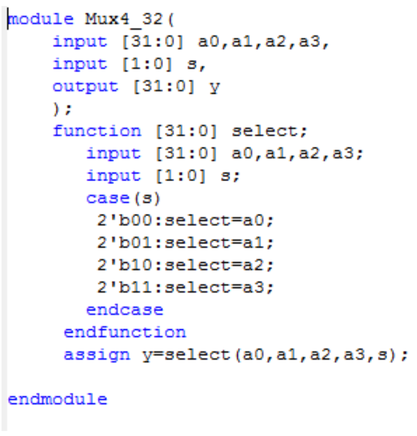
寄存器堆:



```
module Mux2_32(
    input [31:0] a0,a1,
    input s,
    output [31:0] y
    );
    assign y = s ? a1 : a0;


endmodule
```

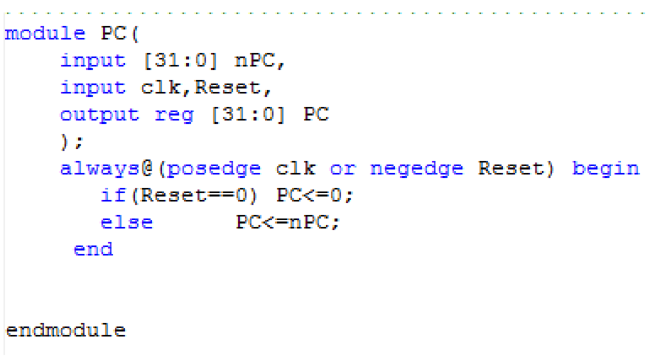


```
module Mux4_5(
    input [4:0] a0,a1,a2,a3,
    input [1:0] s,
    output [4:0] y
    );
    function [4:0] select;
       input [4:0] a0,a1,a2,a3;
       input [1:0] s;
       case(s)
        2'b00:select=a0;
        2'b01:select=a1;
        2'b10:select=a2;
        2'b11:select=a3;
       endcase
     endfunction
     assign y=select(a0,a1,a2,a3,s);

endmodule
```



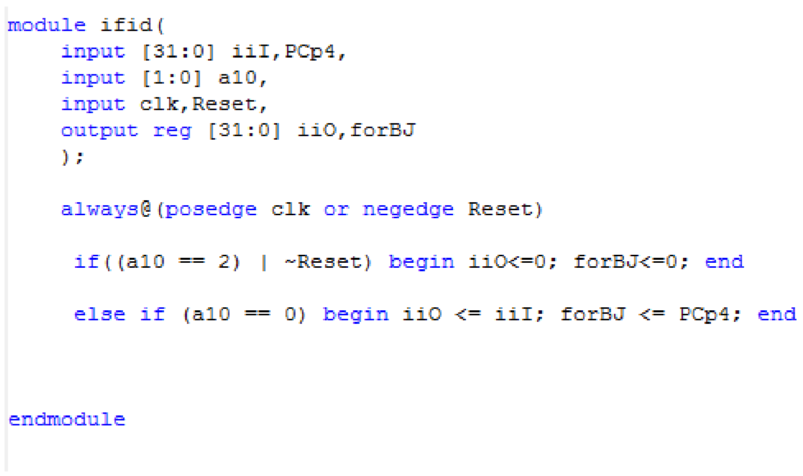
```
module Mux4_32(
    input [31:0] a0,a1,a2,a3,
    input [1:0] s,
    output [31:0] y
    );
    function [31:0] select;
       input [31:0] a0,a1,a2,a3;
       input [1:0] s;
       case(s)
        2'b00:select=a0;
        2'b01:select=a1;
        2'b10:select=a2;
        2'b11:select=a3;
       endcase
     endfunction
     assign y=select(a0,a1,a2,a3,s);

endmodule
```

PC:



```
module PC(
    input [31:0] nPC,
    input clk,Reset,
    output reg [31:0] PC
    );
    always@(posedge clk or negedge Reset) begin
       if(Reset==0) PC<=0;
       else      PC<=nPC;
     end


endmodule
```

IF/ID 锁存器:



```
module ifid(
    input [31:0] iiI,PCp4,
    input [1:0] a10,
    input clk,Reset,
    output reg [31:0] ii0,forBJ
    );

    always@(posedge clk or negedge Reset)

     if((a10 == 2) | ~Reset) begin ii0<=0; forBJ<=0; end

     else if (a10 == 0) begin ii0 <= iiI; forBJ <= PCp4; end



endmodule
```

寄存器堆:

```
module Reg(
          input [4:0]  Rna,Rnb,Wn,//读端口寄存器号a和b以及写端口寄存器号
          input  a1,clk,Reset,
          input [31:0]  Wd,
          output [31:0] Qa,Qb
    );

    reg [31:0] Register[1:31]; //定义31个32位的寄存器
    //Read data
    assign Qa = (Rna == 0) ? 0 : Register[Rna];
    assign Qb = (Rnb == 0) ? 0 : Register[Rnb];
    //Write data
    integer i;

    always @ (negedge clk or negedge Reset)
    if (Reset == 0)
        begin
        for(i = 1 ; i <= 31 ; i = i + 1)
          Register[i] <= 0 ;
        end
    else
         begin
          Register[Wn] <= Wd;
         end

endmodule
```

shift 辅助部件（用于将 shift 信号无条件扩展至 32 位，以适应 4 路 32 位多选器）：

```
module forS(
    input [4:0] shift,
    output [31:0] forSR
    );
    assign forSR={27'b0,shift};


endmodule
```

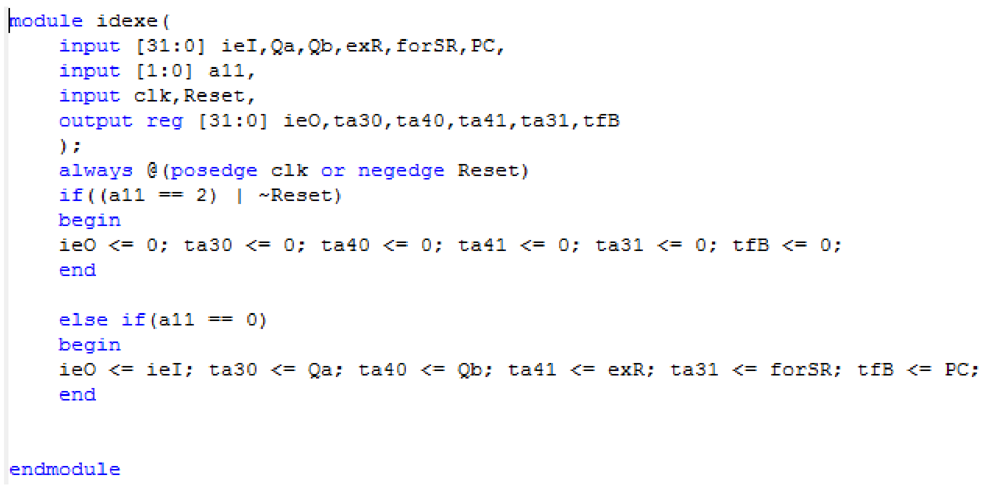
符号扩展器：

```
module Extender(
    input [15:0] imme,
    input a2,
    output [31:0] ximme
    );
    assign ximme = a2 ? {(imme[15] ? 16'hffff : 16'h0) , imme} : {16'h0 , imme};

endmodule
```

jump 辅助部件（用于将 jump 指令的低 26 位与 PC 高四位组合成 target）：

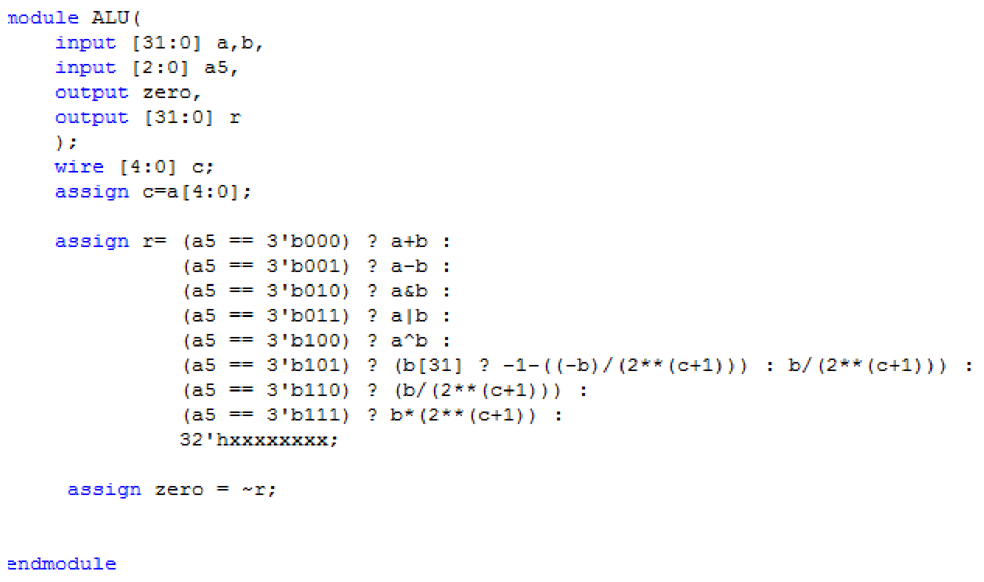
```
module forJ(
    input [25:0] j,
    input [31:0] npc,
    output [31:0] r
    );
    assign r={npc[31:28] , j , 2'b00};


endmodule
```

ID/EXE 锁存器：



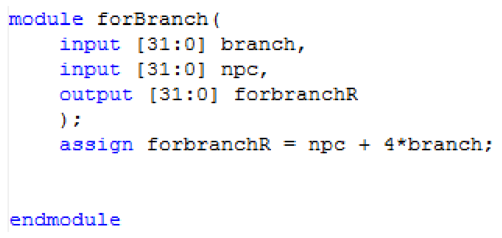
```
module idexe(
    input [31:0] ieI,Qa,Qb,exR,forSR,PC,
    input [1:0] a11,
    input clk,Reset,
    output reg [31:0] ieO,ta30,ta40,ta41,ta31,tfB
    );
    always @(posedge clk or negedge Reset)
    if((a11 == 2) | ~Reset)
    begin
    ieO <= 0; ta30 <= 0; ta40 <= 0; ta41 <= 0; ta31 <= 0; tfB <= 0;
    end

    else if(a11 == 0)
    begin
    ieO <= ieI; ta30 <= Qa; ta40 <= Qb; ta41 <= exR; ta31 <= forSR; tfB <= PC;
    end


endmodule
```

ALU:



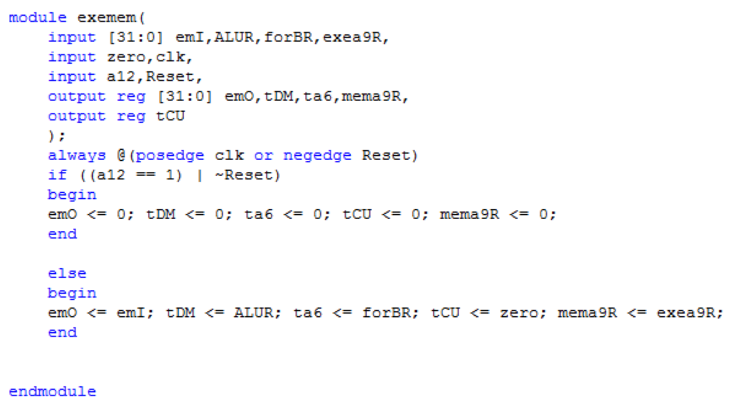
```
module ALU(
    input [31:0] a,b,
    input [2:0] a5,
    output zero,
    output [31:0] r
    );
    wire [4:0] c;
    assign c=a[4:0];

    assign r= (a5 == 3'b000) ? a+b :
              (a5 == 3'b001) ? a-b :
              (a5 == 3'b010) ? a&b :
              (a5 == 3'b011) ? a|b :
              (a5 == 3'b100) ? a^b :
              (a5 == 3'b101) ? (b[31] ? -1-((-b)/(2**(c+1))) : b/(2**(c+1))) :
              (a5 == 3'b110) ? (b/(2**(c+1))) :
              (a5 == 3'b111) ? b*(2**(c+1)) :
              32'hxxxxxxxx;

     assign zero = ~r;


endmodule
```

branch 辅助部件（用于产生 label；label=PC+4+offset<<2）:



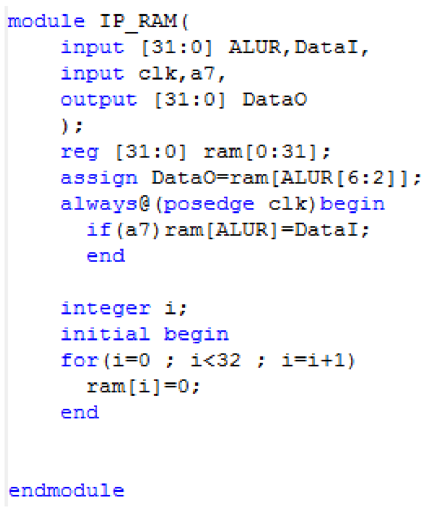
```
module forBranch(
    input [31:0] branch,
    input [31:0] npc,
    output [31:0] forbranchR
    );
    assign forbranchR = npc + 4*branch;


endmodule
```

EXE/MEM 锁存器:



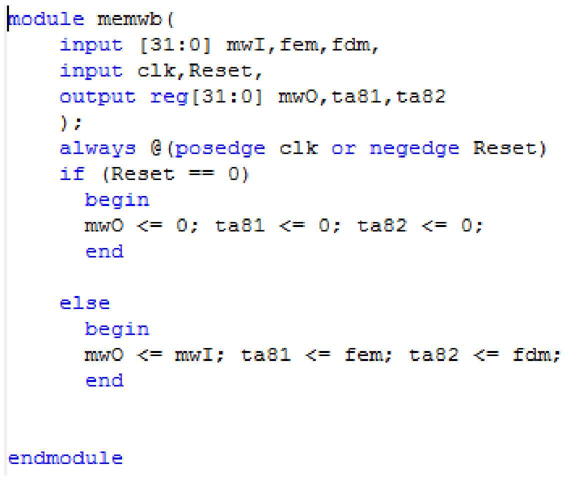
```
module exemem(
    input [31:0] emI,ALUR,forBR,exea9R,
    input zero,clk,
    input a12,Reset,
    output reg [31:0] emO,tDM,ta6,mema9R,
    output reg tCU
    );
    always @(posedge clk or negedge Reset)
    if ((a12 == 1) | ~Reset)
    begin
    emO <= 0; tDM <= 0; ta6 <= 0; tCU <= 0; mema9R <= 0;
    end

    else
    begin
    emO <= emI; tDM <= ALUR; ta6 <= forBR; tCU <= zero; mema9R <= exea9R;
    end


endmodule
```

数据存储器:



```
module IP_RAM(
    input  [31:0] ALUR,DataI,
    input clk,a7,
    output [31:0] DataO
    );
    reg [31:0] ram[0:31];
    assign DataO=ram[ALUR[6:2]];
    always@(posedge clk)begin
      if(a7)ram[ALUR]=DataI;
      end

    integer i;
    initial begin
    for(i=0 ; i<32 ; i=i+1)
      ram[i]=0;
    end


endmodule
```

MEM/WB 级锁存器:



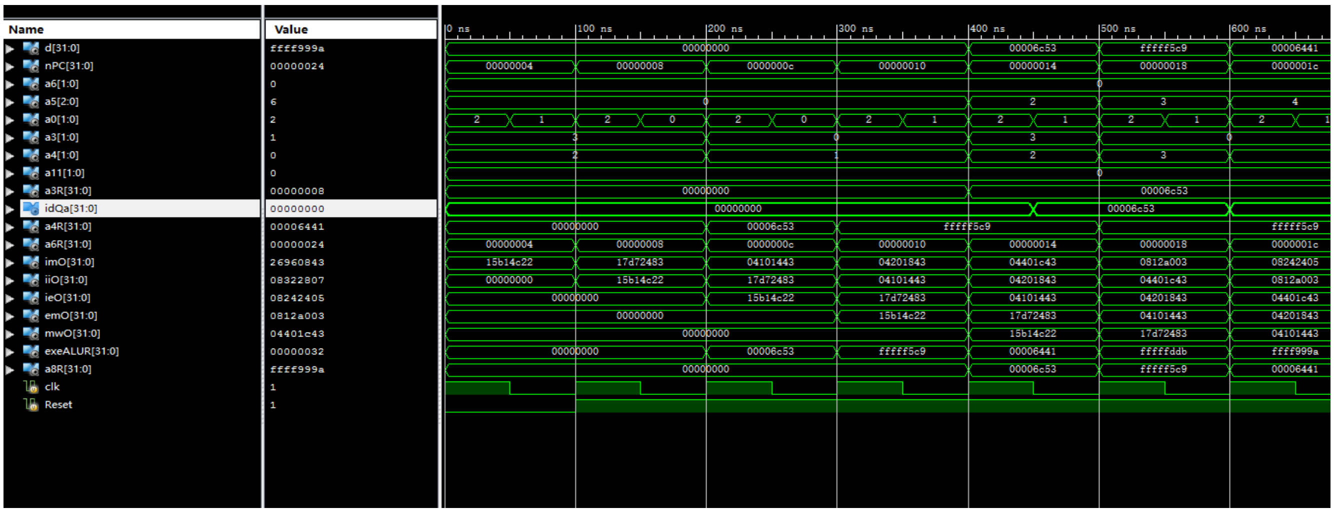
```
module memwb(
    input  [31:0] mwI,fem,fdm,
    input clk,Reset,
    output reg[31:0] mwO,ta81,ta82
    );
    always @(posedge clk or negedge Reset)
    if (Reset == 0)
      begin
      mwO <= 0; ta81 <= 0; ta82 <= 0;
      end

    else
      begin
      mwO <= mwI; ta81 <= fem; ta82 <= fdm;
      end


endmodule
```

4. CPU 仿真执行(1)

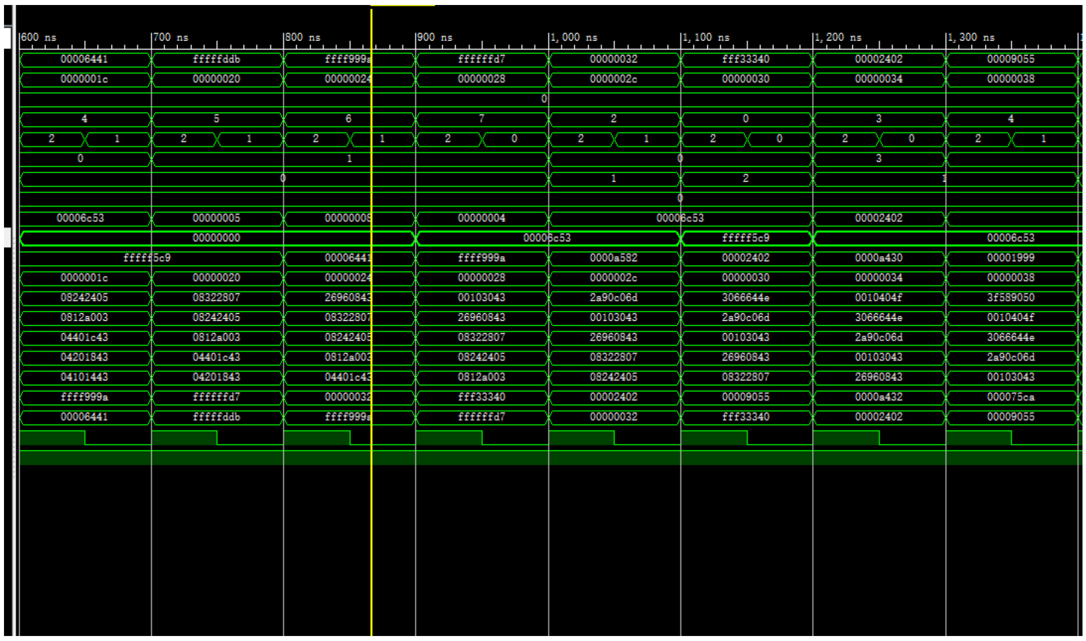
assign rom[6'h0] = 32'h15b14c22; //addi rt(2) rs(1)(6c53)   4
assign rom[6'h1] = 32'h17d72483; //addi   rt(3) rs(4)(f5c9)   8
assign rom[6'h2] = 32'h04101443; //and rd(5) rs(2) rt(3)   12
assign rom[6'h3] = 32'h04201843; //or rd(6) rs(2) rt(3) 16
assign rom[6'h4] = 32'h04401c43; //xor rd(7) rs(2) rt(3) 20
assign rom[6'h5] = 32'h0812a003; //sra(5) rd(8) rt(3) 24
assign rom[6'h6] = 32'h08242405; //srl(8) rd(9) rt(5) 28
assign rom[6'h7] = 32'h08322807; //sll(4) rd(10) rt(7) 32
assign rom[6'h8] = 32'h26960843; //andi   rt(11) rs(2)(a582) 36
assign rom[6'h9] = 32'h00103043; //add rd(12) rs(2) rt(3) 40
assign rom[6'ha] = 32'h2a90c06d; //ori rt(13) rs(3)(a430) 44
assign rom[6'hb] = 32'h3066644e; //xori rt(14) rs(2)(1999) 48
assign rom[6'hc] = 32'h0010404f; //add rd(16) rs(2) rt(15) 52
assign rom[6'hd] = 32'h3f589050; //beq rs(2) rt(16)(d624) (jump to 32h)358c8
assign rom[6'h32] = 32'h380000c7; //store rt(7) 0000(rs(6)) 358cc

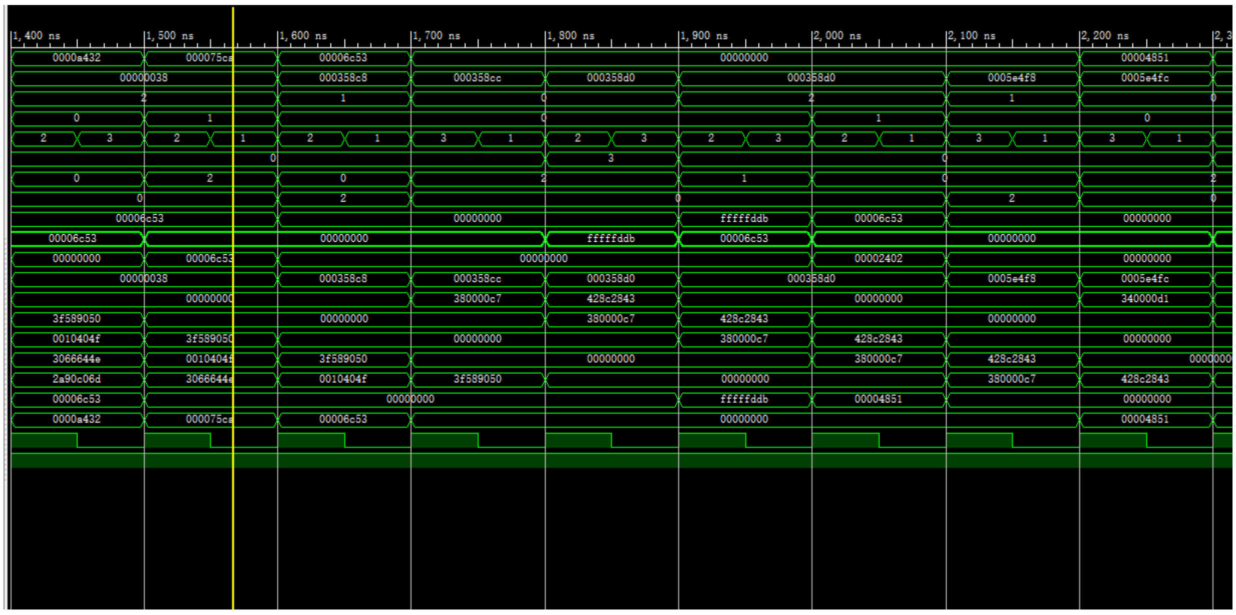
```
assign rom[6'h33] = 32'h428c2843; //bne rs(2) rt(3)(a30a) (jump to 3eh)5e4f8
assign rom[6'h3e] = 32'h340000d1; //load rt(17) 0000(rs(6)) 5e4fc
assign rom[6'h3f] = 32'h49159d16; //jump 1159d16 (jump to 16h) 4567458
assign rom[6'h16] = 32'h00104843; //add rd(18) rs(2) rt(3) 456745c
```

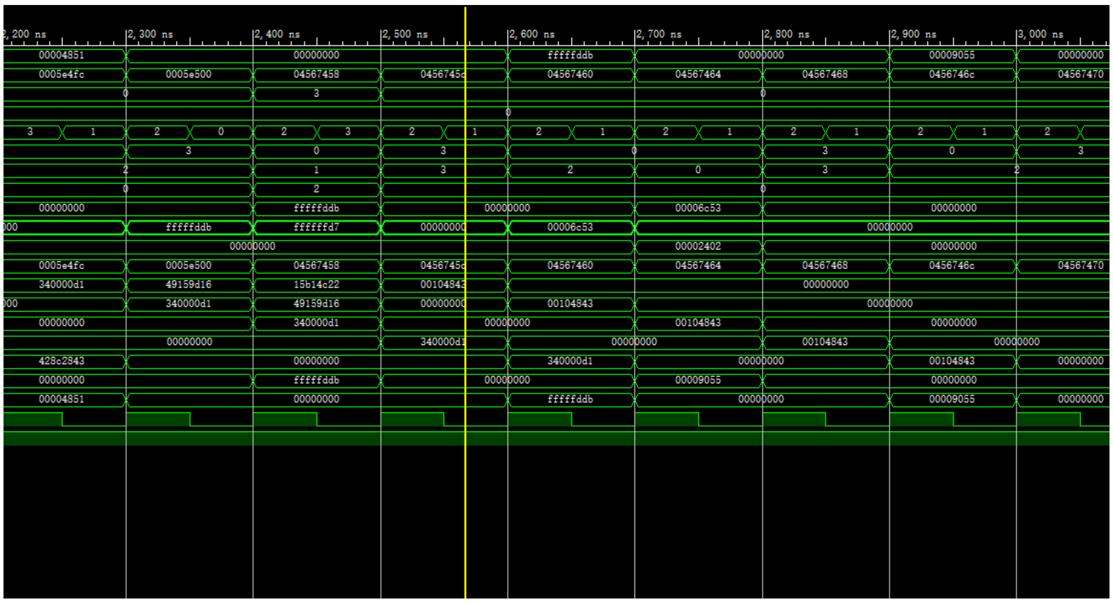
结果截图:



Name	Value
d[31:0]	ffff999a
nPC[31:0]	00000024
a6[1:0]	0
a5[2:0]	6
a0[1:0]	2
a3[1:0]	1
a4[1:0]	0
a11[1:0]	0
a3R[31:0]	00000008
idQa[31:0]	00000000
a4R[31:0]	00006441
a6R[31:0]	00000024
imO[31:0]	26960843
iiO[31:0]	08322807
ieO[31:0]	08242405
emO[31:0]	0812a003
mwO[31:0]	04401c43
exeALUR[31:0]	00000032
a8R[31:0]	ffff999a
clk	1
Reset	1







(2)

assign rom[6'h00]=32'b0;//op6 func6 shift5 rd5 rs5 rt5

assign rom[6'h01]=32'b00110100000000000000000000000001;//001101 000000 00000 00000 00000 00001 load 0+rs0=>rt1

assign rom[6'h02]=32'b00000000000100000000110000100010;//000000 000001 00000 00011 00001 00010 add rd3<=rs1+rs2

assign rom[6'h03]=32'b00110100000000000000000001100100;//001101 000000 00000 00000 00011 00100 load 0+rs3=>rt4

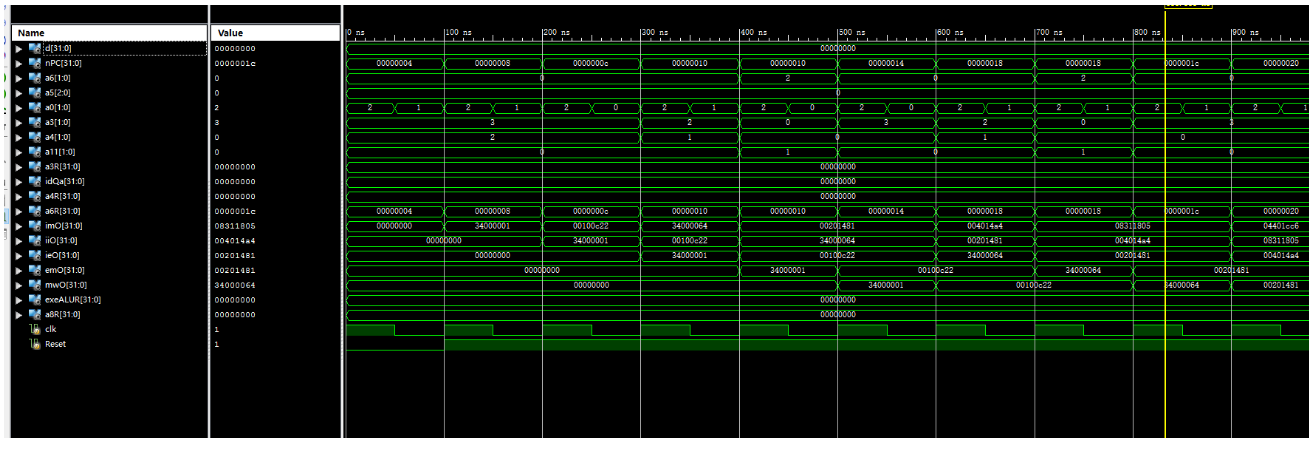
assign rom[6'h04]=32'b00000000001000000000101001000001;//000000 000010 00000 00101 00100 00001 sub rd5<=rs4-rt1

assign rom[6'h05]=32'b00000000010000000000101001010 0100;//000000 000100 00000 00101 00101 00100 mul rd5<=rs5*rt4

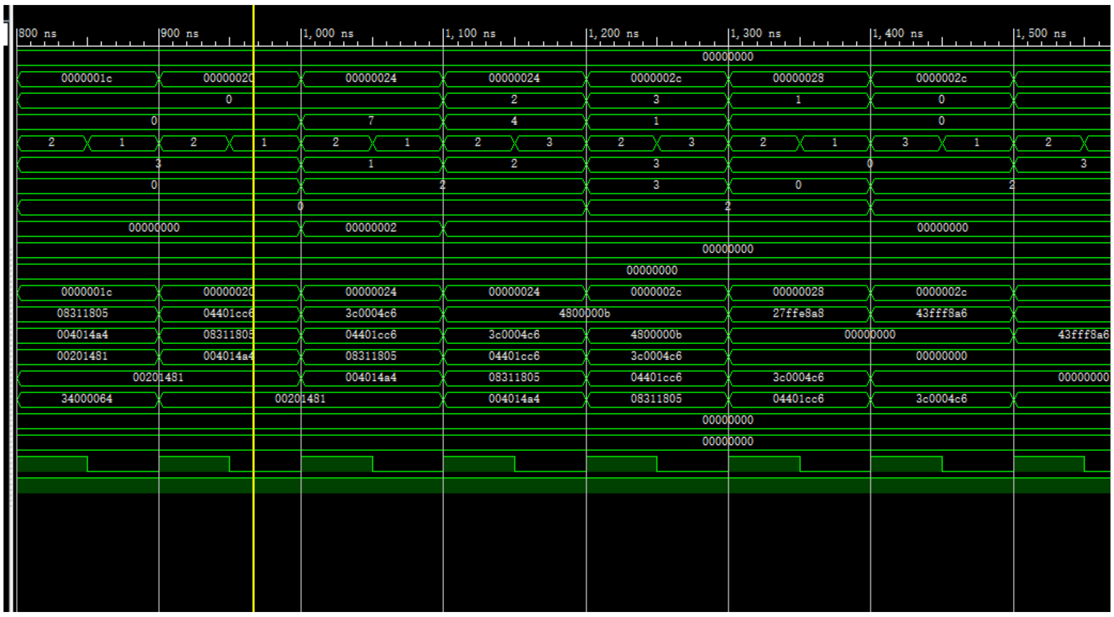
assign rom[6'h06]=32'b00001000001100010001100000000101;//000010 000011 00010 00110 00000 00101 sll 2 rd6<=rt5<<

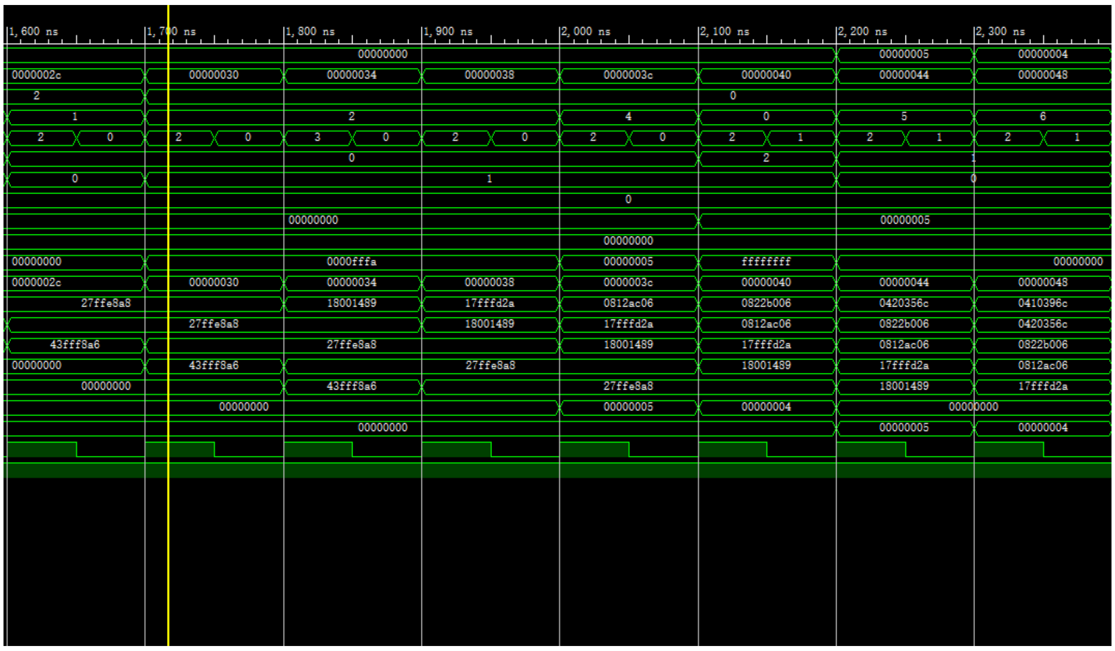
```
    assign rom[6'h07]=32'b00000100010000000000111001100011 0;//000001 000100 00000 00111 00110 00110 xor rd7<=rs6xorrt6
    assign rom[6'h08]=32'b00111100000000000000000100110001 10;//001111 000000 00000 00001 00110 00110 beq 1 rs6 rt6
    assign rom[6'h09]=32'b01001000000000000000000000001011;//010010 000000 00000 00000 00000 01011 jmp 0BH
    assign rom[6'h0A]=32'b01000011111111111111100010100110;//010000 111111 11111 11110 00101 00110 bne -2 rs5 rt6
    assign rom[6'h0B]=32'b00100111111111111111010001010100 0;//001001 111111 11111 11010 00101 01000 andi rs5=>rt8
    assign rom[6'h0C]=32'b00011000000000000000101001000100 1;//000110 000000 00000 00101 00100 01001 muli 5*rs4=>rt9
    assign rom[6'h0D]=32'b00010111111111111111111010010101 0;//000101 111111 11111 11111 01001 01010 addi -1+rs9=>rt10
    assign rom[6'h0E]=32'b00001000000100101010110000000011 0;//000010 000001 00101 01011 00000 00110 sra 5 rd11<=rt6>>
    assign rom[6'h0F]=32'b00001000001000101011000000000011 0;//000010 000010 00101 01100 00000 00110 srl 5 rd12<=rt6>>
    assign rom[6'h10]=32'b00000100001000000011010101101100;//000001 000010 00000 01101 01011 01100 or rd13<=rs11orrt12
    assign rom[6'h11]=32'b00000100000100000011100101101100;//000001 000001 00000 01110 01011 01100 and rd14<=rs11andrt12
    assign rom[6'h12]=32'b00101000000000000000010101101110;//001010 000000 00000 00001 01011 01110 ori 1ors11=?rt14
    assign rom[6'h13]=32'b00111000000000000001000000000111 0;//001110 000000 00000 01000 00000 01110 store 8+rs0<=rt14
    assign rom[6'h14]=32'b0;
    assign rom[6'h15]=32'b0;
    assign rom[6'h16]=32'b0;
    assign rom[6'h17]=32'b0;
    assign rom[6'h18]=32'b0;
```

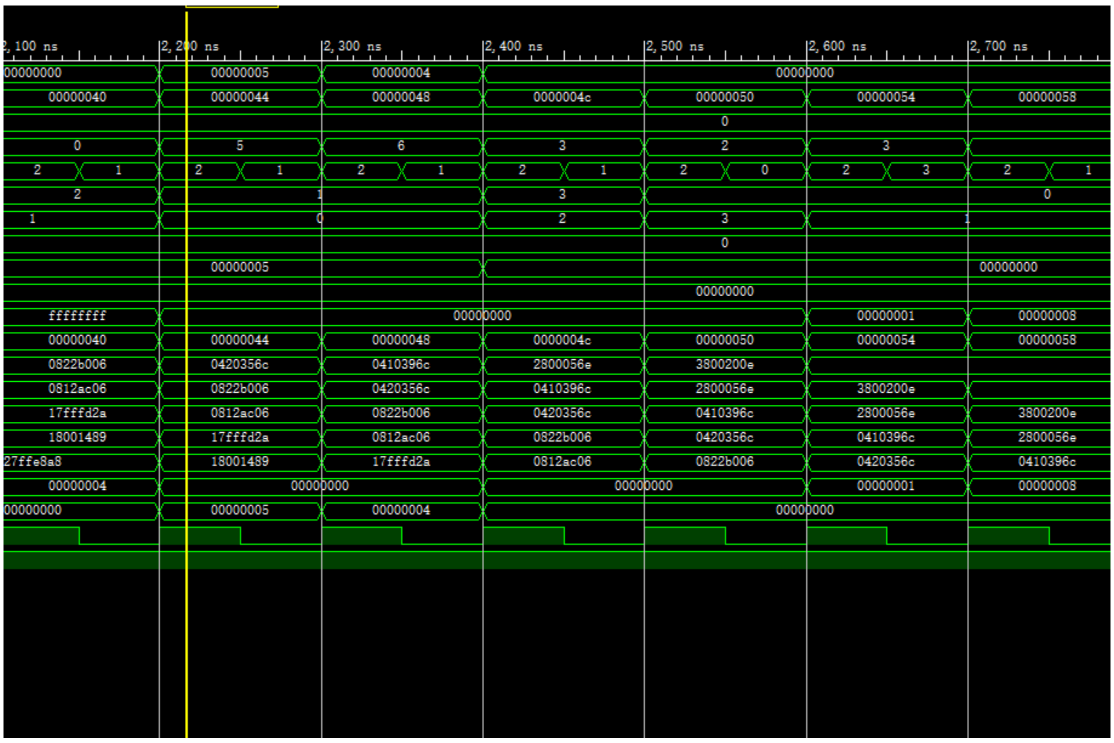
结果截图:



Name	Value
d[31:0]	00000000
nPC[31:0]	0000001c
a6[1:0]	0
a5[2:0]	0
a0[1:0]	2
a3[1:0]	3
a4[1:0]	0
a11[1:0]	0
a3R[31:0]	00000000
idQa[31:0]	00000000
a4R[31:0]	00000000
a6R[31:0]	0000001c
imO[31:0]	08311805
iiO[31:0]	004014a4
ieO[31:0]	00201481
emO[31:0]	00201481
mwO[31:0]	34000064
exeALUR[31:0]	00000000
a8R[31:0]	00000000
clk	1
Reset	1


800 ns
900 ns
1, 000 ns
1, 100 ns
1, 200 ns
1, 300 ns
1, 400 ns
1, 500 ns


1, 600 ns
1, 700 ns
1, 800 ns
1, 900 ns
2, 000 ns
2, 100 ns
2, 200 ns
2, 300 ns


2, 100 ns
2, 200 ns
2, 300 ns
2, 400 ns
2, 500 ns
2, 600 ns
2, 700 ns
00000000 00000005 00000004 00000000
00000040 00000044 00000048 0000004c 00000050 00000054 00000058
0
0 5 6 3 2 3
2 1 2 1 2 1 2 1 2 0 2 3 2 1
2 1 3 0
1 0 2 3 1
0
00000005 00000000
00000000
ffffffff 00000000 00000001 00000008
00000040 00000044 00000048 0000004c 00000050 00000054 00000058
0822b006 0420356c 0410396c 2800056e 3800200e
0812ac06 0822b006 0420356c 0410396c 2800056e 3800200e
17fffd2a 0812ac06 0822b006 0420356c 0410396c 2800056e 3800200e
18001489 17fffd2a 0812ac06 0822b006 0420356c 0410396c 2800056e
27ffe8a8 18001489 17fffd2a 0812ac06 0822b006 0420356c 0410396c
00000004 00000000 00000000 00000001 00000008
00000000 00000005 00000004 00000000