Report for MIPS CPU Simulator
Time: 1/17/15
Author: Bingyu Shen

1. Design Idea

I mainly design the alu, alucontrol, ctr, pc, reg and cpu in the first part.

The cpu is the most difficult part I think. The other difficult point is that the onClock method of reg should add a onChange()， which is the reason of the test point branch's failure, but I learn it from the libother's method.

ALU

```
void ALU::onChange()
{
 // Add your code here
 LineData result = 0;
 switch(in[aluCtr]){
 case 0://and
     result = in[input1]&in[input2];
     setOutput(zero, (result == 0) ? 1 : 0);
     break;
 case 1://or
     result = in[input1]|in[input2];
     setOutput(zero, (result == 0) ? 1 : 0);
     break;
 case 2:
     result = in[input1]+in[input2];
     setOutput(zero, (result == 0) ? 1 : 0);
     break;
 case 6:
     result = in[input1]-in[input2];
     setOutput(zero, (result == 0) ? 1 : 0);
     break;
 case 7:// set on if less than
     if(in[input1]<in[input2])
         result = 1;
     else
         result =0;
     setOutput(zero, (result == 0) ? 1 : 0);
     break;
```

```
case 12:
    result = ~(in[input1]|in[input2]);
    setOutput(zero, (result == 0) ? 1 : 0);
    break;
}
setOutput(aluRes,result);
}
}
```

ALUCONTROL

```
   void ALUControl::onChange()
   {
    // Add your code here
    if(in[aluOp] == 0){
        setOutput(aluCtr,B("0010"));
        return;
    }
    if(in[aluOp] == 1){
        setOutput(aluCtr,B("0110"));
        return;
    }
    if(in[aluOp] == 2){
        if(in[funct] == 0){
            setOutput(aluCtr,B("0010"));
            return;
        }
        if(in[funct] == 2){
            setOutput(aluCtr,B("0110"));
            return;
        }if(in[funct] == 4){
            setOutput(aluCtr,B("0000"));
            return;
        }if(in[funct] == 5){
            setOutput(aluCtr,B("0001"));
            return;
        }
        if(in[funct] == 10){
            setOutput(aluCtr,B("0111"));
            return;
        }

    }
   }
}
```

CPU

```
namespace MIPS {

  CPU::CPU() : pc(-4), instMem("Instruction Memory"), dataMem("Data Memory"),
       muxMem2Reg("muxMem2Reg"), muxAlu("muxAlu"), muxRegDes("muxRegDes")

  {
       // Add your code here
       //IF
       BIND(pc,newPC,instMem,address);

       //ID
       instMem.bind(readData,partialListener(26,31,ctr,opCode));
       instMem.bind(readData,partialListener(21,25,reg,readReg1));
       instMem.bind(readData,partialListener(16,20,reg,readReg2));

       instMem.bind(readData,partialListener(16,20,muxAlu,input1));
       instMem.bind(readData,partialListener(11,15,muxAlu,input2));

       instMem.bind(readData,partialListener(0,3,aluControl,funct));
       instMem.bind(readData,partialListener(0,15,signExtend,immInput));
       //ctr
       BIND(ctr,regDst,muxAlu,muxSel);
       BIND(ctr,branch,pc,branch);
       BIND(ctr,memRead,dataMem,memRead);
       BIND(ctr,memToReg,muxMem2Reg,muxSel);
       BIND(ctr,aluOp,aluControl,aluOp);
       BIND(ctr,memWrite,dataMem,memWrite);
       BIND(ctr,ALUSrc,muxRegDes,muxSel);
       BIND(ctr,regWrite,reg,regWrite);

       BIND(muxAlu,muxOut,reg,writeReg);
       //aluCtr
       BIND(aluControl,aluCtr,alu,aluCtr);
       //signExtend

       BIND(signExtend,immData,muxRegDes,input2);
       BIND(signExtend,immData,pc,immData);
       //reg
       BIND(reg,readData1,alu,input1);
       BIND(reg,readData2,muxRegDes,input1);
       BIND(muxRegDes,muxOut,alu,input2);
```

```
        BIND(reg,readData2,dataMem,writeData);


        //alu
        BIND(alu,aluRes,dataMem,address);
        BIND(alu,aluRes,muxMem2Reg,input1);
        BIND(alu,zero,pc,zero);
        //dataMem
        BIND(dataMem,readData,muxMem2Reg,input2);
        //muxMem2Reg
        BIND(muxMem2Reg,muxOut,reg,writeData);



        instMem.input(memRead,1);
}
```

CTR

```
namespace MIPS {
void Ctr::onChange()
{
  // Add your code here
  if(in[opCode] == B("000010")){
        //jump
        setOutput(jump,1);
        //setOutput(regDst,0);
        //setOutput(ALUSrc,0);
        //setOutput(memToReg,0);
        setOutput(regWrite,0);
        setOutput(memRead,0);
        setOutput(memWrite,0);
        setOutput(branch,0);
        //setOutput(aluOp,0);
        return;
  }
  else if(in[opCode]==B("000000")){
        // R type
        setOutput(jump,0);
        setOutput(regDst,1);
        setOutput(ALUSrc,0);
        setOutput(memToReg,0);
        setOutput(regWrite,1);
        setOutput(memRead,0);
        setOutput(memWrite,0);
        setOutput(branch,0);
        setOutput(aluOp,2);
        return;
```

```
}
else if(in[opCode]==B("100011")){
     //lw
     setOutput(jump,0);
     setOutput(regDst,0);
     setOutput(ALUSrc,1);
     setOutput(memToReg,1);
     setOutput(regWrite,1);
     setOutput(memRead,1);
     setOutput(memWrite,0);
     setOutput(branch,0);
     setOutput(aluOp,0);
     return;
}
else if(in[opCode]==B("101011")){
     //sw
     setOutput(jump,0);
     //setOutput(regDst,0);
     setOutput(ALUSrc,1);
     //setOutput(memToReg,0);
     setOutput(regWrite,0);
     setOutput(memRead,0);
     setOutput(memWrite,1);
     setOutput(branch,0);
     setOutput(aluOp,0);
     return;
}
else if(in[opCode]==B("000100")){
     //beq
     setOutput(jump,0);
     //setOutput(regDst,0);
     setOutput(ALUSrc,0);
     //setOutput(memToReg,0);
     setOutput(regWrite,0);
     setOutput(memRead,0);
     setOutput(memWrite,0);
     setOutput(branch,1);
     setOutput(aluOp,1);
     return;
}
else if(in[opCode]==B("001000")){
     //addi
     setOutput(jump,0);
     setOutput(regDst,0);
```

```
            setOutput(ALUSrc,1);
            setOutput(memToReg,0);
            setOutput(regWrite,1);
            setOutput(memRead,0);
            setOutput(memWrite,0);
            setOutput(branch,0);
            setOutput(aluOp,0);
            return;
        }
    }
}
```

PC

```
    namespace MIPS {
    void PC::onClock()
    {
      // Add your code here
      if(in[zero] && in[branch])   m_pc = m_pc + 4*in[immData];

      m_pc = m_pc + 4;
      setOutput(newPC,m_pc);
    }
}
```

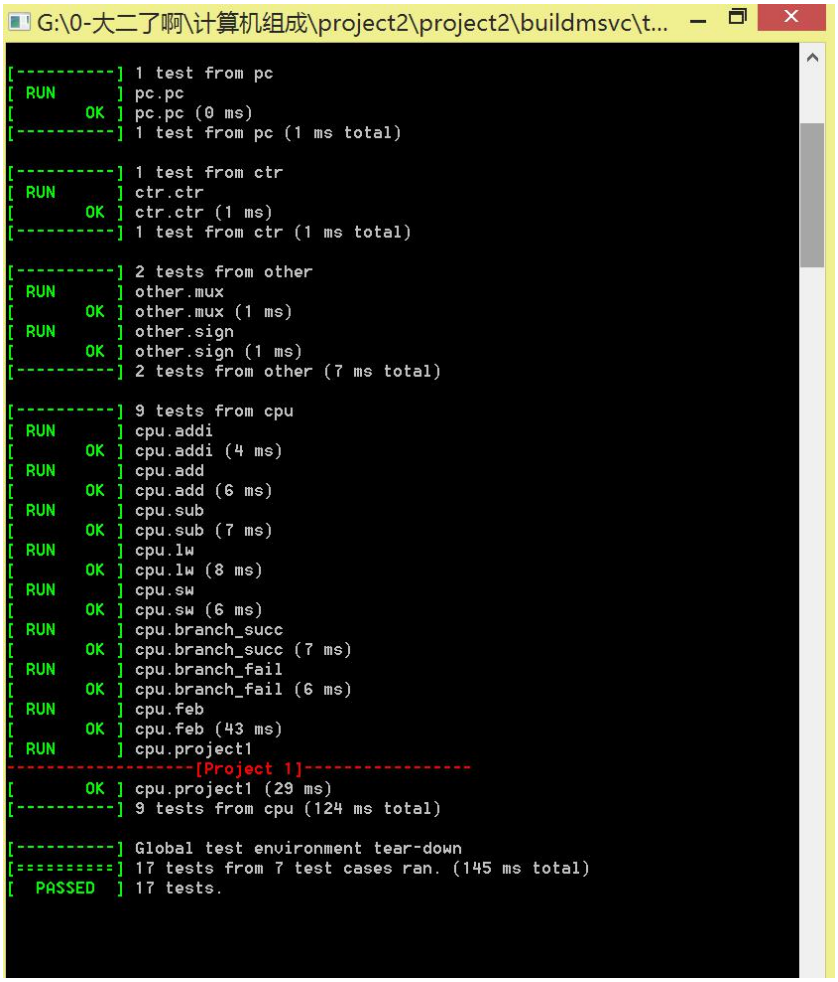
REG

```
    void Reg::onChange()
    {
      // Add your code here

      setOutput(readData1,memory[in[readReg1]]);
      setOutput(readData2,memory[in[readReg2]]);
    }

    void Reg::onClock()
    {
      // Add your code here
      if(in[regWrite]){
            memory[in[writeReg]]=in[writeData];
      }
      onChange();
}
```

2. Test result



The 16 test points all passed. And the additional task is passed.

3. Additional task

To run project1 in the cpu I wrote, I translate it into binary code.

Here is my code.

```
TEST(cpu,project1)
{
```

```
   CPU cpu;

//The input is 5!
// You can change the value by changing the hex number.

   // <input:0> addi $v4, $v4, 5
   cpu.instMem.loadMemory(0x00000000, 0x20c60005);
   // <input:1> addi $v0, $v0, 0
   cpu.instMem.loadMemory(0x00000004, 0x20420000);
// <input:2> addi $v1, $v1, 1
cpu.instMem.loadMemory(0x00000008, 0x20630001);
// <input:3> addi $v2, $v2, 2
cpu.instMem.loadMemory(0x0000000c, 0x20840002);
// <input:4> beq $v0, $v4, EXIT0
cpu.instMem.loadMemory(0x00000010, 0x1046000e);
// <input:5> beq $v1, $v4, EXIT1
cpu.instMem.loadMemory(0x00000014, 0x1066000f);
// <input:6> beq $v2, $v4, EXIT2
cpu.instMem.loadMemory(0x00000018, 0x10860010);
// <input:7> addi $v4, $v4, -2
cpu.instMem.loadMemory(0x0000001c, 0x20c6fffe);
//function
// <input:8> add $v3, $v2, $v1
cpu.instMem.loadMemory(0x00000020, 0x00832820);
// <input:9> add $v3, $v3, $v1
cpu.instMem.loadMemory(0x00000024, 0x00a32820);
// <input:10> add $v3, $v3, $v0
cpu.instMem.loadMemory(0x00000028, 0x00a22820);
// <input:11> add $v3, $v3, $v0
cpu.instMem.loadMemory(0x0000002c, 0x00a22820);
// <input:12> add $v3, $v3, $v0
cpu.instMem.loadMemory(0x00000030, 0x00a22820);
   // <input:13> addi $v4, $v4, -1
cpu.instMem.loadMemory(0x00000034, 0x20c6ffff);
// <input:14> beq $v4, $zero, EXIT
cpu.instMem.loadMemory(0x00000038, 0x10c0000a);
// <input:15> addi $v0, $v1, 0
cpu.instMem.loadMemory(0x0000003c, 0x20620000);
// <input:16> addi $v1, $v2, 0
cpu.instMem.loadMemory(0x00000040, 0x20830000);
// <input:17> addi $v2, $v3, 0
cpu.instMem.loadMemory(0x00000044, 0x20a40000);
// <input:18> beq $v0, $v0, function
cpu.instMem.loadMemory(0x00000048, 0x1042fff5);
```

```
    // EXIT0
    // <input:19> addi $v3, $v0, 0
    cpu.instMem.loadMemory(0x0000004c, 0x20450000);
    // <input:20> beq $v0, $v0, EXIT
    cpu.instMem.loadMemory(0x00000050, 0x10420004);
    // EXIT1
    // <input:21> addi $v3, $v1, 0
    cpu.instMem.loadMemory(0x00000054, 0x20650000);
    // <input:22> beq $v0, $v0, EXIT
    cpu.instMem.loadMemory(0x00000058, 0x10420002);
    // EXIT2
    // <input:23> addi $v3, $v2, 0
    cpu.instMem.loadMemory(0x0000005c, 0x20850000);
    // <input:24> beq $v0, $v0, EXIT
    cpu.instMem.loadMemory(0x00000060, 0x10420000);
    // EXIT
    // <input:25> addi $v0, $v3, 0
    cpu.instMem.loadMemory(0x00000064, 0x20a20000);
    // End of the program
    cpu.instMem.loadMemory(0x00000068, 0xFFFFFFFF);

    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),FOREGR
OUND_INTENSITY|FOREGROUND_RED);
    printf("------------------[Project 1]-----------------\n");
    SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),FOREGR
OUND_GREEN|FOREGROUND_RED|FOREGROUND_BLUE);

    EXPECT_EQ(25, cpu.run());
}
```

As you can see, the first is the address of the instruction, the second is the binary code of the instruction. When using the beq instruction, I calculate the immediate by the new address minus (the current address+1).