

大实验 实验报告

计 31（104 组）

陈冲 赖涵光 蔡文静

目录

一、 实验目的	2
二、 完成情况概述	2
三、 指令系统与数据通路	3
四、 冲突解决方法	4
五、 性能测试运行情况	6
六、 扩展指令运行情况	8
七、 扩展（单步分支预测、INT 中断）	10
八、 遇到的问题及解决	13
九、 实验分工	14
十、 实验心得	15
十一、 对本课程的建议	16

一、实验目的

- (1) 加深对计算机系统知识的理解；
- (2) 进一步理解和掌握流水线结构计算机各部件组成及内部工作原理；
- (3) 掌握计算机外部输入输出的设计；
- (4) 培养硬件设计和调试的能力。

二、完成情况概述

本次大实验通过编写 VHDL 代码实现了支持指令流水的计算机系统。使用 10M 时钟，实现延迟槽。最终成果：可以正常运行监控程序；正常运行五条扩展指令，运算结果正确；正常运行测试样例，实现了数据冲突、结构冲突、控制冲突的解决，同时，运行速度较快；完成了 INT 中断与单步分支预测两项扩展功能；代码精简，算上注释，仅用不到 900 行代码就实现了上述所有功能。

同时，为了方便管理三人编写的代码，也为了修改后重新使用此前的代码，我们还使用了 git 工具进行版本控制工作，这对我们的调试工作有很大帮助。

总的来说，本组设计实现的 CPU 以较高的质量完成了实验的所有要求，性能稳定，并实现了一些扩展。

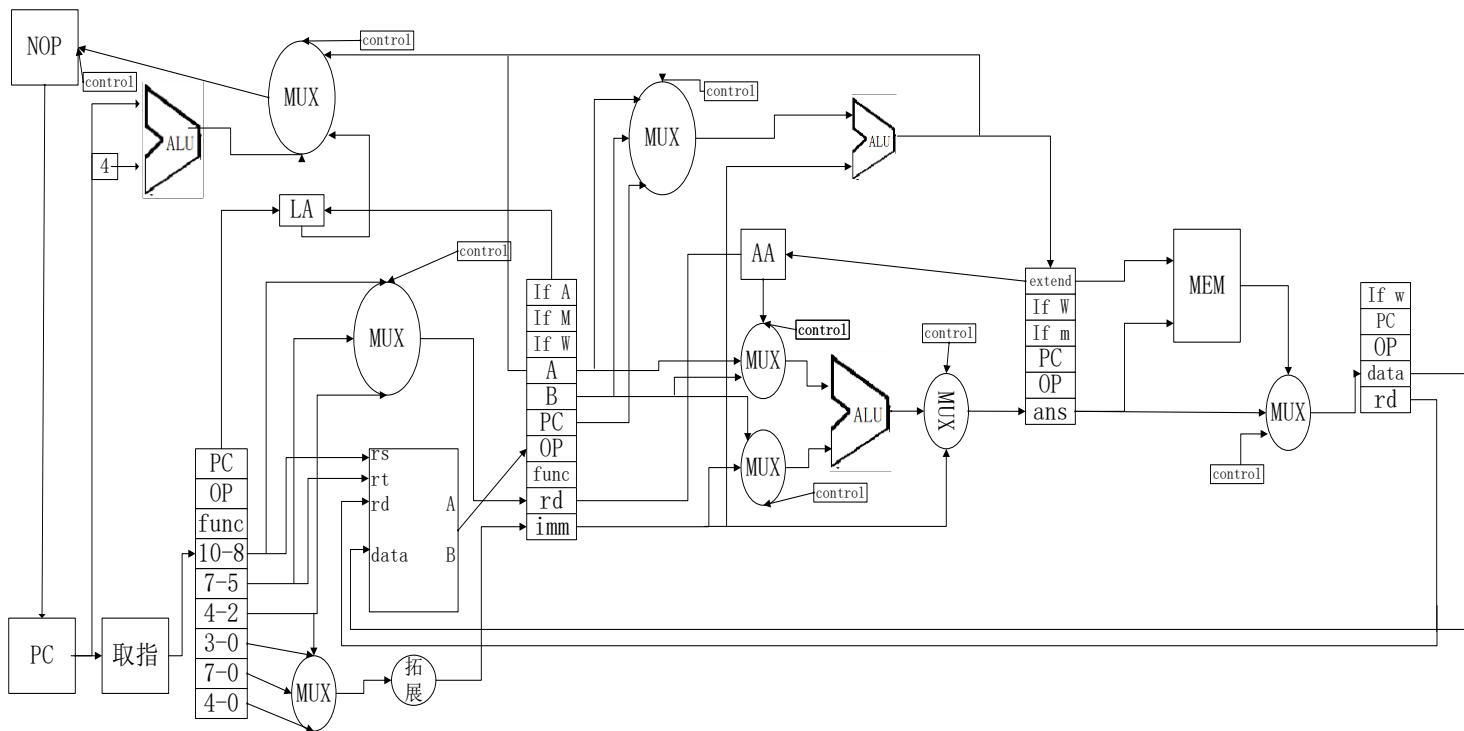
三、指令系统与数据通路

指令系统:

本次 CPU 大实验实现了给定的 25 条基本指令与 5 条扩展指令 (SLLV、SRAV、CMPI、MOVE、SLTU)，指令列表如下:

0	00001				NOP
1	00010				B
2	00100				BEQZ
3	00101				BNEZ
4	00110			00	SLL
5	00110			11	SRA
6	01000				ADDIU
7	01001				ADDIU
8	01100	000			BTEQZ
9	01100	011			ADDSP
10	01100	100			MTSP
11	01101				LI
12	01110				CMPI
13	01111				MOVE
14	10010				LW_SP
15	10011				LW
16	11010				SW_SP
17	11011				SW
18	11100			01	ADDU
19	11100			11	SUBU
20	11101	000	000	00	JR
21	11101	010	000	00	MFPC
22	11101		001	00	SLLV
23	11101		011	00	AND
24	11101		011	01	OR
25	11101		010	10	CMP
26	11101		001	11	SRAV
27	11110			00	MFIH
28	11110			01	MTIH
29	11101		000	11	SLTU

数据通路:



本组设计的数据通路如上图所示。在数据通路中，我们不仅明确了五级流水各个阶段的过程，同时还写出了各个阶段需要的所有阶段寄存器，这对我们实验初期的工作有很大帮助。同时，在设计数据通路时，我们已经尝试进行冲突的解决，例如，图中就标出了 AA 型数据冲突和 LA 型数据冲突的解决方式，具体的冲突解决方法下面将详细讲述。

四、冲突解决方法

1.数据冲突

AA 型数据冲突：例如连续两个 ADDU 指令，我们使用数据旁路来解决。在 EXE 的第 0 阶段，判断是否需要从数据旁路中获得操作寄存器的值，如下图。

```

if id_exe_a_exe = '1' then
    exe_a := id_exe_a_from_exe;
elsif id_exe_a_me = '1' then
    exe_a := id_exe_a_from_me;
else
    exe_a := id_exe_a;
end if;
if id_exe_b_exe = '1' then
    exe_b := id_exe_b_from_exe;
elsif id_exe_b_me = '1' then
    exe_b := id_exe_b_from_me;
else
    exe_b := id_exe_b;
end if;

```

在 EXE 的第 4 阶段，判断上一条指令的结果寄存器和该条指令的操作寄存器相同，且没有访存，则使用数据旁路，如下图。

```

if exe_wb = '1' and id_op /= 0 then
    if exe_rz = id_rx then
        id_exe_a_exe <= '1';
        id_exe_a_from_exe <= exe_ans;
    end if;
    if exe_rz = id_ry then
        id_exe_b_exe <= '1';
        id_exe_b_from_exe <= exe_ans;
    end if;
end if;

```

LA 型数据冲突：例如 LW 指令后紧跟一条 ADDU 指令，我们使用插入 NOP 的方法来解决，如下图。

```

if exe_lw = '1' and id_op /= 0 then
    if exe_rz = id_rx or exe_rz = id_ry then
        pre_pc_by_exe <= '1';
        pre_pc_from_exe <= exe_pc;
        if_id_op_nop_exe <= '1';
        id_exe_op_nop_exe <= '1';
    end if;
end if;

```

2.结构冲突

通过分层把取指和 MEM 访存分割在不同阶段，以此来避免结构冲突。

3.控制冲突

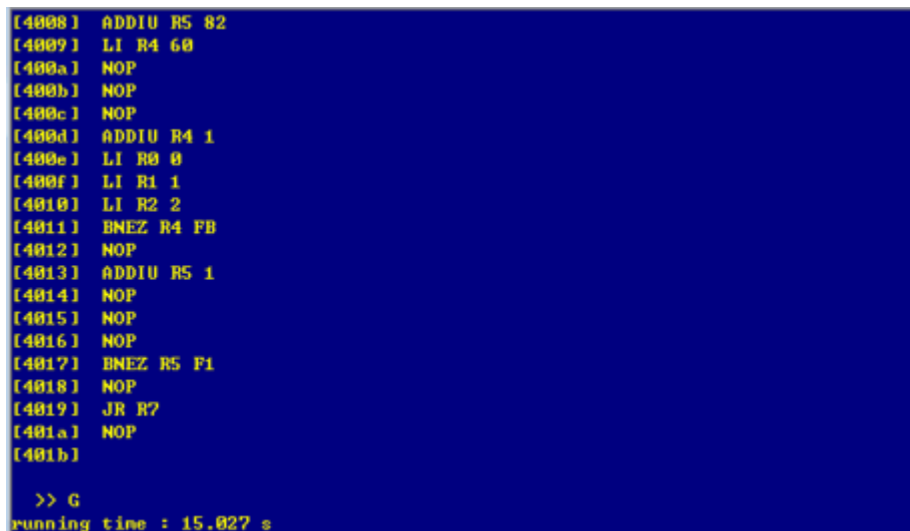
B 型控制冲突：使用[分支预测](#)的方法，在扩展中会详细描述。

J 型控制冲突：使用插入 NOP 的方法来实现，如下图。

```
when 20 => -- JR
    id_exe_wb <= '0';
    pre_pc_by_id <= '1';
    get_regbin("0" & id_ins(10 downto 8), pre_pc_from_id);
    if_id_op_nop_id <= '1';
    if id_ins(10 downto 8) = "110" and int_jr = '1' then
        int_jr <= '0';
        int_recover <= '1';
    end if;
```

五、性能测试运行情况

1.性能标定



```
[4008] ADDIU R5 82
[4009] LI R4 60
[400a] NOP
[400b] NOP
[400c] NOP
[400d] ADDIU R4 1
[400e] LI R0 0
[400f] LI R1 1
[4010] LI R2 2
[4011] BNEZ R4 FB
[4012] NOP
[4013] ADDIU R5 1
[4014] NOP
[4015] NOP
[4016] NOP
[4017] BNEZ R5 F1
[4018] NOP
[4019] JR R2
[401a] NOP
[401b]

>> G
running time : 15.027 s
```

运行时间 15.027 秒

2.运算数据冲突的效率测试

```
[4005] ADDU R1 R1 R2
[4006] ADDU R2 R1 R3
[4007] SUBU R3 R2 R2
[4008] CMP R1 R2
[4009] ADDU R2 R3 R2
[400a] BEQZ R4 3
[400b] ADDIU R4 1
[400c] BTEQZ F8
[400d] NOP
[400e] ADDIU R5 1
[400f] BNEZ R5 F5
[4010] NOP
[4011] JR R7
[4012] NOP
[4013]

>> G
running time : 27.565 s
```

运行时间 27.565 秒

3.控制指令冲突测试

```
>> A
[4000] LI R1 1
[4001] LI R5 FF
[4002] SLL R5 R5 0
[4003] ADDIU R5 83
[4004] LI R4 60
[4005] CMP R4 R1
[4006] BTEQZ 3
[4007] ADDIU R4 1
[4008] BNEZ R4 FD
[4009] CMP R4 R1
[400a] BNEZ R5 F9
[400b] ADDIU R5 1
[400c] JR R7
[400d] NOP
[400e]

>> G
running time : 15.027 s
```

运行时间 15.027 秒

4.访存数据冲突性能测试

```

>> A
[4000] LI R2 FF
[4001] LI R3 C0
[4002] SLL R3 R3 0
[4003] LI R5 FF
[4004] SLL R5 R5 0
[4005] ADDIU R5 83
[4006] LI R1 61
[4007] SW R3 R1 2
[4008] LW R3 R4 2
[4009] SW R3 R4 1
[400a] LW R3 R1 1
[400b] BNEZ R4 FB
[400c] ADDIU R1 1
[400d] BNEZ R5 F8
[400e] ADDIU R5 1
[400f] JR R7
[4010] NOP
[4011]

>> G
running time : 20.025 s

```

运行时间 20.025 秒

5.读写指令存储器测试

```

>> A
[4000] LI R2 FF
[4001] LI R3 55
[4002] SLL R3 R3 0
[4003] LI R5 FF
[4004] SLL R5 R5 0
[4005] ADDIU R5 83
[4006] LI R4 61
[4007] SW R3 R4 1
[4008] BNEZ R4 FE
[4009] ADDIU R4 1
[400a] BNEZ R5 FB
[400b] ADDIU R5 1
[400c] JR R7
[400d] NOP
[400e]

>> G
running time : 7.528 s

```

运行时间 7.528 秒

六、扩展指令运行情况

针对五条扩展指令，我们自己编写了测试程序对其进行测试，运行后通过 R 命令查看寄存器的值，结果全部正确。

1. SLLV


```

>> A
[4000] LI R1 2
[4001] SLL R2 R1 0
[4002] SLL R3 R1 1
[4003] LI R4 1
[4004] SLLU R1 R4
[4005] JR R7
[4006]

>> G
running time : 0.028 s

>> R
R0=0000 R1=0002 R2=0200
R3=0004 R4=0004 R5=8007

```

2. SRAV

```

>> A
[4000] LI R1 2
[4001] SLL R1 R1 0
[4002] SRA R2 R1 0
[4003] SRA R3 R2 1
[4004] LI R4 1
[4005] SRAU R3 R4
[4006] JR R7
[4007]

>> G
running time : 0.028 s

>> R
R0=0000 R1=0200 R2=0002
R3=0001 R4=0000 R5=8007

```

3. CMPI

```

[4000] LI R1 1
[4001] CMPI R1 1
[4002] BTEQZ 3
[4003] LI R3 3
[4004] LI R4 1
[4005] JR R7
[4006]

>> G
running time : 0.028 s

>> R
R0=0000 R1=0001 R2=0002
R3=0003 R4=0000 R5=8007

```

CMPI 涉及对 T 寄存器的赋值，因此紧跟一条 BTEQZ——根据 T 寄存器的跳转指令，来检验是否正确对 T 寄存器进行赋值。由于使用了延迟槽，因此 BTEQZ 的下一条指令“LI R3 3”依然会执行，之后正常跳转，即“LI R4 1”不会执行。

4. MOVE

```
>> A
[4000] LI R1 1
[4001] LI R2 2
[4002] MOVE R1 R2
[4003] JR R2
[4004]

>> G
running time : 0.028 s

>> R
R0=0000 R1=0002 R2=0002
R3=0003 R4=0000 R5=8007
```

5. SLTU

```
>> A
[4000] LI R1 1
[4001] LI R2 0
[4002] SLTU R1 R2
[4003] BTEQZ 2
[4004] LI R3 3
[4005] LI R4 1
[4006] JR R2
[4007]

>> G
running time : 0.027 s

>> R
R0=0000 R1=0001 R2=0000
R3=0003 R4=0000 R5=8007
```

SLTU 涉及对 T 寄存器的赋值，因此紧跟一条 BTEQZ 来验证正确性，方法同 CMPI。

七、扩展

1. 单步分支预测

实现方法：

记录上次 B 型指令（不包含无条件跳转）是否跳转，以此来预测下一 B 型指令的目标地址。

首先在 ID 阶段译码出是一条 B 型指令，则根据 [predict](#) 的值预测是继续执行还是跳转。

```

if predict = '1' then
    pre_pc_by_id <= '1';
    pre_pc_from_id <= id_pc + sign_extend8( id_ins(7 downto 0) );

```

在 EXE 阶段，已经可以得到计算的结果，此时就可以判断是否预测错误。以 BEQZ 为例，如下图，`exe_a="0000000000000000"` and `predict='0'` 表示应该跳转但没有跳转；`exe_a/= "0000000000000000"` and `predict='1'` 表示不该跳转却跳转了，需要在后面进行补救。`exe_br` 为 ‘1’ 表示预测错误，同时计算出正确的地址 `exe_pc2`。

```

when 2 | 8 => -- BEQZ
if (exe_a = "0000000000000000" and predict = '0' ) or (exe_a /= "0000000000000000" and predict = '1') then
    exe_br := '1';
    exe_pc2 := exe_pc + exe_imm;
end if;

```

如果预测成功则什么都不做，如果预测错误则进行补救：若是应该跳转但没有跳转，则执行这一跳转；若是不该跳转却跳转了，则让 pc 加一得到正确的 pc。纠正的方法是插入 NOP。同时将 `predict` 取反保证正确性。

```

if exe_br = '1' then
    pre_pc_by_exe <= '1';
    if predict = '0' then
        pre_pc_from_exe <= exe_pc2;
    else
        pre_pc_from_exe <= exe_pc + 1 ;
    end if;
    predict <= not predict ;
    if_id_op_nop_exe <= '1';
    --id_exe_op_nop_exe <= '1';
end if;

```

2.INT 指令中断

实现方法：

通过阅读监控程序，知道了监控程序中处理中断的方法。首先进行保存现场的工作，如下图，

```

signal int_r0      : std_logic_vector( 15 downto 0) := "0000000000000000";
signal int_r1      : std_logic_vector( 15 downto 0) := "0000000000000000";
signal int_imm      : std_logic_vector( 15 downto 0) := "0000000000000000";
signal int_pc       : std_logic_vector( 15 downto 0) := "0000000000000000";
signal int_jr       : std_logic := '0';
signal int_recover  : std_logic := '0';

```

`int_r0`、`int_r1`、`int_imm`、`int_pc` 备份当前的 `r0`, `r1`, 中断号, 当前 PC,

`int_jr` 表示是否跳转, `int_recover` 表示是否恢复。其次, 如下图,

```

when "11111" =>
    if_id_op <= 30;
    pre_pc <= "00000000000000101";

```

在 IF 阶段把 INT 指令编号为 30。再次, 如下图,

```

when 30 =>
    id_exe_wb <= '0';
    int_jr <= '1';
    int_imm <= "0000000000000" & id_ins(3 downto 0);
    int_pc <= id_pc;

```

在 ID 阶段, 当检测到 INT 指令时, 实行跳转, 把 `int_jr` 设为 1, 并备份 PC。然后, 在 WB 阶段, 如下图,

```

if me_wb_op = 30 then
    int_r0 <= r0;
    int_r1 <= r1;
    sp <= "1011111100010000";
    r0 <= int_imm;
    r1 <= int_pc;
end if;

```

当检测到 INT 指令时, 对 R0、R1 进行备份, 令 SP=BF10, 将中断号和当前 PC 存入 R0, R1。

根据监控程序, 当指令运行到“JR R6”时返回原 PC, 因此, 在 ID 阶段处理 JR 指令过程中, 当检测到“JR R6”且 `int_jr`=1 时, 如下图,

```

if id_ins(10 downto 8) = "110" and int_jr = '1' then
    int_jr <= '0';
    int_recover <= '1';
end if;

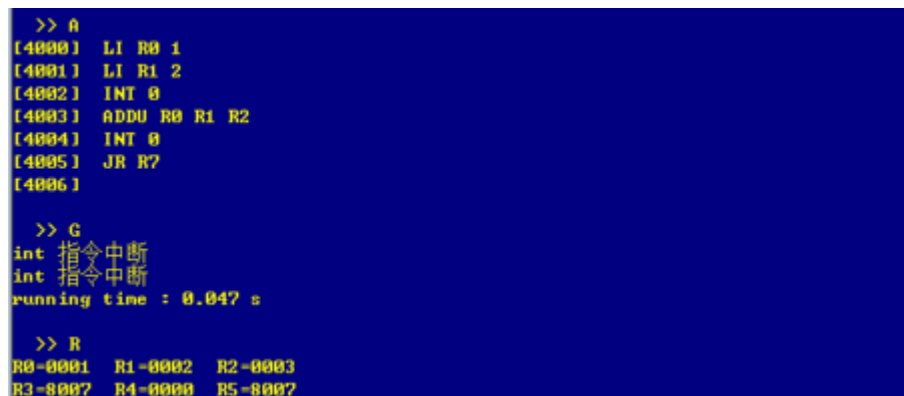
```

设 `int_recover`=1, `int_jr`=0。最后, 在 WB 第 4 阶段, 若检测到

int_recover=1 则把原先保存的 R0、R1 恢复，如下图。

```
if int_recover ='1' then
    r0 <= int_r0;
    r1 <= int_r1;
end if;
```

运行结果：



```
>> A
[4000] LI R0 1
[4001] LI R1 2
[4002] INT 0
[4003] ADDU R0 R1 R2
[4004] INT 0
[4005] JR R7
[4006]

>> G
int 指令中断
int 指令中断
running time : 0.047 s

>> R
R0=0001 R1=0002 R2=0003
R3=8007 R4=0000 R5=8007
```

八、遇到的问题及解决

1.处理中断时，当恢复现场时，R3 的值没有恢复，通过读监控程序对于中断部分的代码（如下图）

```
LW_SP R7 0x0000

ADDSP 0x0001
ADDSP 0x0001
NOP
MTIH R3;
JR R6
LW_SP R3 0x00FF

NOP
```

发现对 R3 的处理位于“JR R6”之后，这样 R3 没有恢复这一问题也就得到了解释。

2.处理 INT 指令中断时，我们发现如果中断号不为 0，则不会输

出“int 指令中断”这行字，而是输出 ASCII 码值为中断号的符号。

因此，我们查阅了 Term 程序的 C++源代码，如下图，

```
switch(ch)
{
    case 0x0f:
        return;
    case 0x10:
        cout<<"外部中断"<<endl;
        break;
    case 0x20:
        cout<<"时钟中断"<<endl;
        break;
    case 0x00:
        cout<<"int 指令中断"<<endl;
        break;
    default:
        printf("%c",ch);
        break;
}
break;
```

这表明只有当地址为 0000 时才会输出该行字，且地址为 000F 时是直接 return 的。但是，根据监控程序，如下图，

```
;提示终端，进入中断处理
LI R3 0x000F
```

000F 时进入中断处理，这与 Term 程序有些矛盾。

我们猜测，应当将监控程序改为 `LI R3 0X0000`，即 0000 时进入中断处理，而在“int 指令中断”的下一行输出中断号，且 Term 程序中的 `printf("%c",ch)` 应该改为 `printf("%c",ch+48)`，输出真正的中断号。

九、实验分工

陈冲：负责 IF 阶段、ID 阶段代码，主要冲突的解决，扩展的实现

赖涵光：负责数据通路及阶段寄存器的设计，EXE 阶段代码，测试代码的编写

蔡文静：负责旁路设计，MEM 阶段、WB 阶段代码
调试由三人共同完成。

十、实验心得

此前就听过学长对计原大实验的各种评价——工作量大，难度高，调试费时，需要熬很多夜，“灵魂升华”，等等。但当自己真正面对这一挑战时，其实并没有想那么多，只是想要尽快地上手。

一开始是难度重重的，真正自己需要设计 CPU 时才发现对之前的课程内容，包括五级流水的过程和细节、冲突的解决等，都理解不够透彻，我们只得再翻出课堂的 ppt 仔细研读。数据通路及阶段寄存器都设计完成后，代码的编写还是比较顺畅的。

但调试阶段又是一个极其考验细心和耐心的过程。由于硬件描述语言的特殊性，调试起来比较麻烦，而且耗费时间长，因此我们必须十分细心全面地编写代码、查找 bug，甚至到了检查的前一天，我们还查出 SLL 指令的一个 bug。而陈冲同学在检查前两天由于调试不断出错，一度要放弃实现中断功能，但最终耐心还是战胜了一切。

通过三周的大实验，在课程上，我们对于计算机的组成和 CPU 的工作原理有了更加深入的理解，也能够更加熟练地掌握硬件描述语言。在其他方面，我们体会到了团队的重要性，享受到了团队合作的快乐。同时，这次大实验也是对我们身心的一次大考验，最终我们战

胜了调试过程中产生的焦躁情绪、熬夜带来的困顿、其他科作业的步骤紧逼，高质量完成了此次大实验。

十一、对本课程的建议

1.建议延长大实验的时间，分阶段完成，而不要仅限于三周。

2.前期可以采取适当的方式加深学生对课堂核心内容的理解，避免大实验开始时手忙脚乱。

最后，衷心感谢刘卫东老师、李山山老师和各位助教对我们的悉心指导，让我们能够圆满完成此次大实验，特别是刘卫东老师在小班实验课上对我们组提出的表扬和批评，让我们受益匪浅。