计算机组成与嵌入式

Project

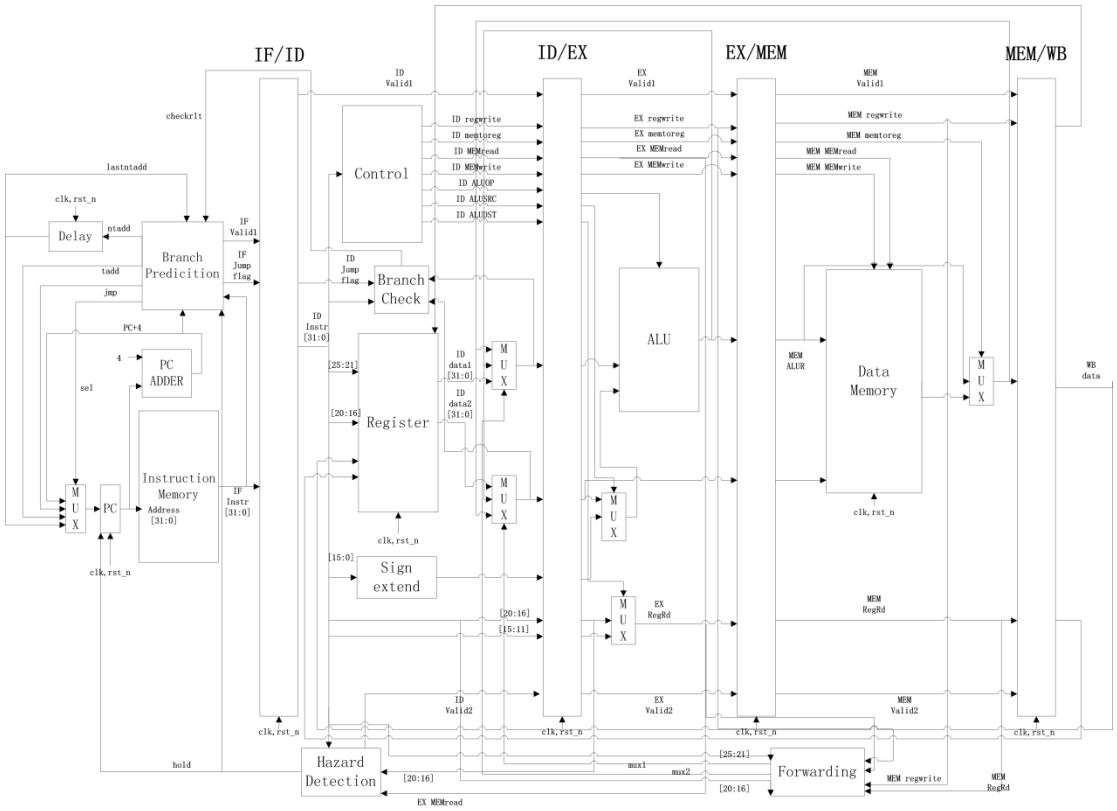
MIPS 架构仿真实验

第 21 组
组长：胡翰彬　5092119008
组员：刘　云　5092119011
陈　诚　5092119019

2011/12/8

一、MIPS 架构总体设计思路

1. 总体结构图



2. 设计思路

本实验采用《Computer Organization and Design: The Hardware/Software Interface》中所介绍的 5 级流水线 MIPS 处理器架构，并对本架构进行部分改动，加入分支预测功能，实现 Load-Use 的异常处理模块，及在 Forwarding 技术基础上作出时序改动以实现分支预测功能。

本实验共实现了 add，sub，and，or，xor，nor，slt，sllv，srlv，srav，addi，lw，sw，beq，bne，j 等指令，并通过 Forwarding 和 Hazard detection 模块解决了大部分本指令集会造成的 Data Hazard。同时由 Branch Prediction 模块实现了 jump 指令，处理了 Control Hazard 的问题。

本 MIPS 架构运行思路为：通过 PC 和指令寄存器取指令，经过 ID-stage 译码获得控制信号，并从寄存器中读取所需值，并通过 Forwarding 得到 forward 值进入 EX 栈，EX 栈经过 ALU 计算得出 ALU 结果，进入 MEM 读写内存，并通过 WB 写回寄存器的值。Branch Prediction 在 IF 栈得到预测结果并在 ID 栈验证结果准确性。

3. 外部接口

接口名	功能
clk	系统时钟，实现各模块同步
rst_n	重置信号，读入初值，并寄存器清零
imemdata.txt	指令寄存器初值，8 个 bit 为一行，32bit 为一条指令
mem.txt	内存初值，8 个 bit 为一行，32bit 为一条数据

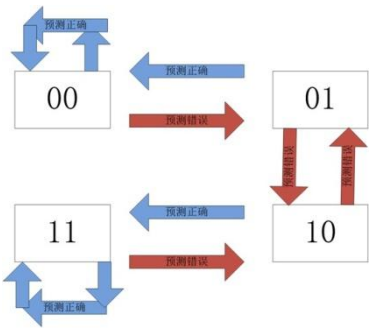
4. 实现指令集

MIPS 指令集(共 31 条)							
助记符	指令格式						操作及其解释
Bit #	31..26	25..21	20..16	15..11	10..6	5..0	
R-type	op	rs	rt	rd	shamt	func	
add	000000	rs	rt	rd	00000	100000	rd <- rs + rt;
sub	000000	rs	rt	rd	00000	100010	rd <- rs - rt;
and	000000	rs	rt	rd	00000	100100	rd <- rs & rt;
or	000000	rs	rt	rd	00000	100101	rd <- rs | rt;
xor	000000	rs	rt	rd	00000	100110	rd <- rs xor rt; (异或)
nor	000000	rs	rt	rd	00000	100111	rd <- not(rs | rt); (或非)
slt	000000	rs	rt	rd	00000	101010	if (rs < rt) rd=1 else rd=0 ;
sllv	000000	rs	rt	rd	00000	000100	rd <- rt << rs;
srlv	000000	rs	rt	rd	00000	000110	rd <- rt >> rs; (logical)
srav	000000	rs	rt	rd	00000	000111	rd <- rt >> rs; (arithmetic)符号位保留
I-type	op	rs	rt	immediate			
addi	001000	rs	rt	immediate			rt <- rs + (sign-extend)immediate;
lw	100011	rs	rt	immediate			rt <- memory[rs + (sign-extend)immediate];
sw	101011	rs	rt	immediate			memory[rs + (sign-extend)immediate] <- rt;
beq	000100	rs	rt	immediate			if (rs == rt) PC <- PC+4 + (sign-extend)immediate<<2
bne	000101	rs	rt	immediate			if (rs != rt) PC <- PC+4 + (sign-extend)immediate<<2
J-type	op	address					
j	000010	address					PC<-(PC+4)[31..28],address,0,0;

二、MIPS 架构设计特色

1. 2bit 分支预测

2bit 分支预测有两个状态位，状态位是 00 与 01 时都认为该地址跳转，状态位是 10 和 11 时，否认为该地址不跳转，各个状态位的转换方式如上图所示。



检测到分支指令时，首先判断分支方向来

确定该目标地址的状态初值：若是分支地址在 PC+4 之前（即认为在进行某个循环），那么进行分支跳转（初值为 01），反之则不跳转（初值为 10）。

0	NULL	0	0
1	NULL	0	0
2	NULL	0	0
3	NULL	0	0
4	NULL	0	0
5	NULL	0	0
6	NULL	0	0
7	NULL	0	0

BranchPrediction 模块在进行分支时，向后传递一个 JumporNot 信号，为了 ID 栈的 BranchCheck（通过 JumporNot 信号，两个寄存器的值，分支指令类型）可以进行分支预测的验证。

若是 BranchCheck 检测到错误，则通过 CheckRlt 信号反应给 BranchPrediction，BranchPrediction 模块找到上一次分支错误的条目并按状态位的变化规则对相应目标地址的状态位进行修改，同时 sel 输出改变为 10（选择上一次分支时未被选择的地址），并把输出 valid1 信号改为 0，Flush 掉下一栈不该被执行的语句。

2. Forwarding 修改

由于分支预测提前至 IF-stage，并在 ID-stage 需实现验证工作，故如仍按照大部分在 EX-stage 实现 Forwarding，将导致在 ID-stage 验证时从寄存器中读出的数据并不为所需进行验证的数据，故有必要对 Forwarding 进行提前，并作出修改。

本实验采用将 EX-stage 中用于 Forwarding 的两个 MUX 提前至 ID-stage，其输入的两个 Forward 的值从 ALU 结果和提前至 MEM-stage 的 MUX 得到，已获得正确的比较值。并将所有 Forwarding 模块的输入信号线提前一个周期使得其得到正确的 Forwarding 控制信号。具体控制信号逻辑如下：

```
if(RWEX && (RdEX != 0) && (RdEX == Rs)) mux1 = 2'b01;
else if(RWMEM && (RdMEM != 0) && (RdMEM == Rs)) mux1 = 2'b10;
else mux1 = 2'b00;

if(RWEX && (RdEX != 0) && (RdEX == Rt)) mux2 = 2'b01;
else if(RWMEM && (RdMEM != 0) && (RdMEM == Rt)) mux2 = 2'b10;
else mux2 = 2'b00;
```

3. 内存，指令寄存器 8 位实现

考虑到在大部分实际的处理器实现中以 8bit 作为 1 个 byte 为内存的单位，故本 MIPS 架构也实现了这种 8bit 为单位的 Data_Memory 模块和 Instruction_Memory 模块。通过将位合并和拆分的方式对两个 Memory 模块进行读写。

三、实验中遇到的问题和解决方案

本组在实验过程中主要发生的问题在总体架构设计阶段，由于考虑加入分支预测模块，经过小组讨论发现会在总体时序上出现较严重的问题。所以本组对不同情况的分支情况进行了严密论证。

由于考虑可能出现 Forwarding 和 beq/bne 同时作用的情况，故将 Forwarding 整体前移。考虑到在预测错误后需要将预测错误的 PC 值储存加入了 Delay。由于出现 Hazard Detection 和分支预测共同作用，故将 hold 信号接入 Branch Prediction 模块。

在完成了上述架构设计后，本组对一些特殊情况进行了讨论论证，如 add 后 beq/bne、lw 后有 beq/bne、接连多条 beq/bne 等情况，发现皆可正常工作。上述过程得到了助教老师的大力帮助。

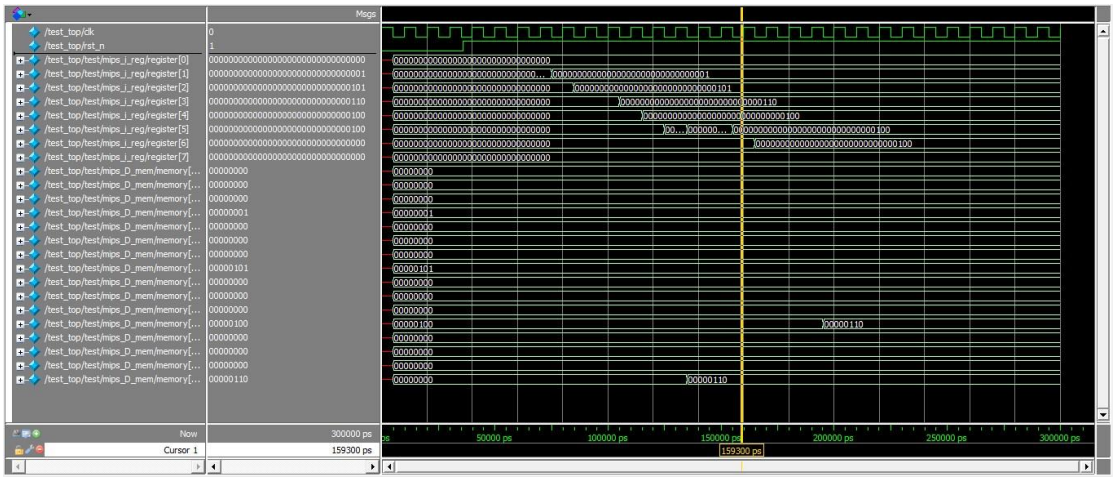
设计过程中，由于总体架构设计较为清晰，故在后续过程中没有发生太过严重的问题。在最终连线过程中，发现对于部分 MUX 的 sel 信号取值没有统一，出现了一些错误。经过适当的修改，以及部分逻辑的重新调整，得到了正确的结果。

四、部分测试结果

首先对于老师给出的测试程序进行测试。内存初始值如实验指导书上设定的初值进行设定。由于若运行正确，则最后的 sw 指令从波形中看不出效果，故改为 sw $3, 44($4)指令。

```
10001100000000010000000000101000    // lw $1, 40($0)
10001100000000010000000000101100    // lw $2, 44($0)
00000000001000100001100000100000    // add $3, $1, $2
00000000010000010010000000100010    // sub $4, $2, $1
00000000011001000010100000100101    // or $5, $3, $4
00000000101000010010100000101010    // slt $5, $5, $1
10101100101000110000000000110100    // sw $3, 52($5)
00000000011001000010100000100100    // and $5, $3, $4
10001100000001100000000000110000    // lw $6, 48($0)
00010000101001100000000000000001    // beq $5, $6, end
00000000101001010010100000100000    // add $5, $5, $5
10101100100000110000000000101100    // sw $3, 44($4)
```

寄存器和内存的仿真波形如下：

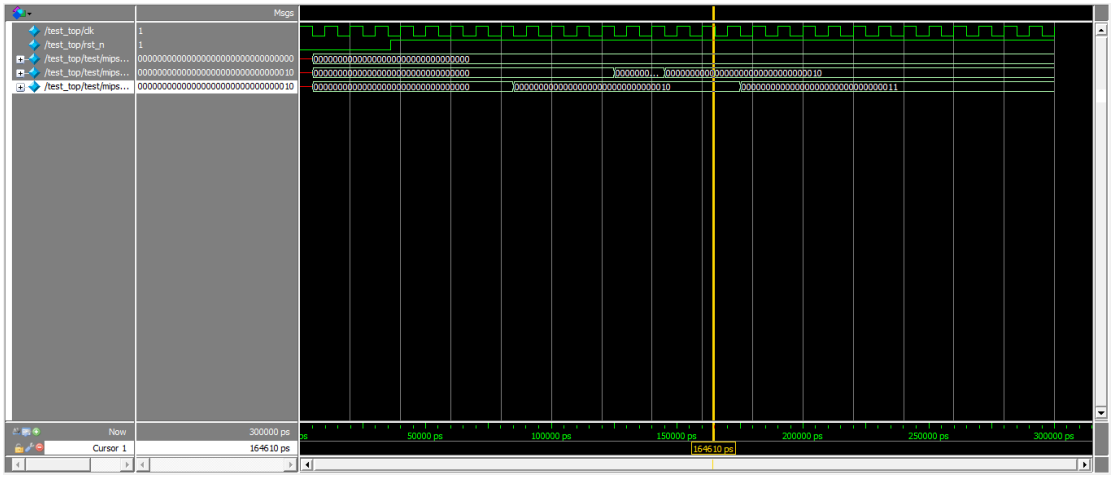


从图中可以看出仿真波形时序及其仿真值都运行正确。

由于考虑过连续出现两次 beq 的情况，故对该情况进行测试，测试如下 MIPS 指令：

```
00100000000000010000000000000000   //addi $1, $0, 0
00100000000000010000000000000010   //addi $2, $0, 2
00010000000000000000000000000001   //beq $0, $0, 1
00100000001000010000000000000001   //addi $1, $1, 1
00010100001000101111111111111110   //bne $1, $2, -2
00100000000000010000000000000011   // addi $2, $0, 3
```

寄存器和内存的仿真波形如下：



从图中可以看出仿真波形时序及其仿真值都运行正确。

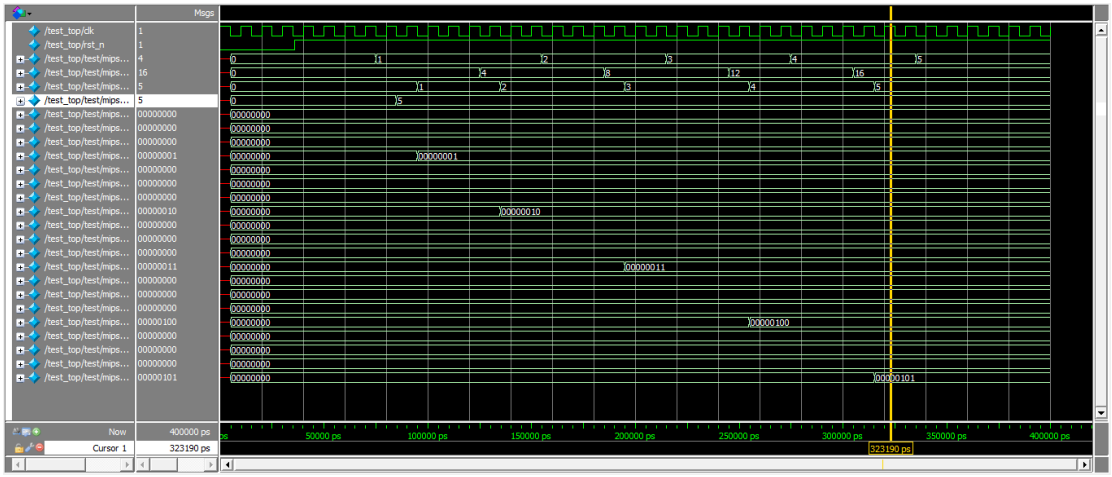
对于本程序进行一个简单 C++循环的测试，C++代码如下：

```
int a[5];
a[0] = 1;
for(int i = 1; i < 5; i++) a[i] = a[i - 1] + 1;
```

经过编译后，得到 MIPS 代码如下：

```
00100000000000010000000000000001   //addi $2, $0, 1
00100000000001010000000000000101   //addi $5, $0, 5
00100000000000010000000000000001   //addi $4, $0, 1
10101100000001000000000000000000   //sw $4, 0($0)
00100000000000011000000000000000   //addi $3, $0, 0
00100000011000110000000000000100   //addi $3, $3, 4
00100000100001000000000000000001   //addi $4, $4, 1
10101100011001000000000000000000   //sw $4, 0($3)
00100000010000100000000000000001   //addi $2, $2, 1
00010000010001010000000000000001   //beq $2, $5, 1
00001000000000000000000000000101   //j 20
```

寄存器和内存的仿真波形如下：



从图中可以看出仿真波形时序及其仿真值都运行正确。

五、小组分工

小组成员	完成工作
胡翰彬	担任组长，确定小组分工，完成总体架构设计，编写 hazard detection、Register、ALU、Control、Forwarding、sign extended 及相应 testbench，进行最终结果的测试与仿真，撰写了报告
刘　云	完成总体架构设计，编写了 PC、MUX、Branch Prediction、Branch Check、Delay、Add 模块及相应 testbench，进行最终结果的测试与仿真，并制作了 PPT，绘制了总体结构图
陈　诚	编写各栈间寄存器、Data_memory 及相应 testbench，编写了顶层文件

六、实验心得

完成 MIPS 处理器的 Verilog 实现，我们主要面临了 2 个方面的巨大困难。其一是我们对计算机体系的专业知识尚不了解得非常细致。其次是在总体架构上我们一开始所达成的意见很不一致，为设计结构增添了许多困难。

正式因为我们遇到了如此多的困难，在解决困难后我们的收获颇丰。

首先，为了了解 MIPS 架构的体系结构，我们小组通过阅读书籍和网上查阅资料等方式重新对 MIPS 架构进行了更为深入的认识，巩固了上课所教所学，也为我们在后面的工作中实现带分支预测功能的 MIPS 架构做好了铺垫。

为了实现 MIPS 处理器，我们采用了 Modelsim 进行编写。在编写过程中，我们统一协调代码风格，注意了代码的可综合性，增强了团队协作能力。同时，本项目也深入了我们对于 MIPS 架构的理解，也让我们更加深入地了解到了 MIPS 指令集中各个指令之间存在的联系，深刻地认识了不同 hazard 的类型以及不同的处理方式，了解了系统设计的辛苦与艰辛。我们还增加了分支预测功能，对现有的架构予以了一定的创新与实践，这我们相信也是十分有意义的。

现在回想这个完成 MIPS 处理器的过程，我们仍然感到回味无穷。虽然为了这个项目，我们小组的成员放弃了业余的休息时间用于搜集资料，程序编写和测试，更是牺牲了夜晚的睡眠时间对程序进行了编写和调试，但是在整个过程中，我们也体会到了难以想象的快乐。我们从一群对于处理器和 Verilog 并不十分了

解的学生，到能够完成一个基本能够运行的处理器，仅用了一个月左右的时间，这是我们先前难以想象的。

在编写过程中，我们通过讨论，研究，加深了对微电子学的理解，整个实验，提升了我们的资料收集能力，信息归纳能力，程序编写能力，调试能力，更培养了我们的合作和创新能力，加深了同学之间的友谊，也为我们今后的学习和生活打下了坚实的基础。我们也都十分感谢这个实验提供给我们的提升自我的机会。

七、参考文献

1. David A. Patterson, John L. Hennessy, Computer Organization and Design: The Hardware/Software Interface, Fourth Edition, ARM Edtion, China Machine Press, 2011

2. Samir Palnitkar, Verilog HDL: A Guide to Digital Design and Synthesis, Second Edition, Publishing House of Electronics Industry, 2009

八、致谢

感谢蒋江老师上课的细致讲解，加深了我们对于 MIPS 架构的认识与理解。

感谢助教对我们在总体架构设计及分支预测模块时序问题的认识。

感谢各位同学在我们项目进行过程中对于我们的帮助与指导。