**数字逻辑与处理器基础大作业**

**单周期cpu设计**

于海粟 无56 2015010901

1. **处理器结构**

1. 控制信号相关问题：

a) RegDst信号控制的多路选择器，输入2对应常数31，这里的31代表寄存器$ra的寄存器号，因为调用jal指令时需要进行$ra = PC+4 的操作，因此执行jal指令时需要RegDst信号为2；

b) 从指令表中可以看出只有sll、srl、sra三个指令用到了[10:6]的指令字段，为shamt，只有在执行这三条指令时才会用到[10:6]字段的参数；

c) MemtoReg信号控制的多路选择器输入2对应的输出为PC+4，所以在执行jal和jalr指令时需要MemtoReg信号为2，因为这些指令需要将下一条指令地址（PC+4）存入$ra或其他寄存器中；

d) PCSrc信号控制的多路选择器输入2对应的输出为寄存器ReadData1，即相应的指令[25:21](rs)，在执行jr和jalr时需要该信号输入2，因为这两个指令都有跳转到寄存器所在地址的功能。

e) ExtOP信号控制符号扩展，有符号数进行扩展时需要输入1，无符号数扩展时需要输入0，ExtOp用于控制到底进行有符号还是无符号的扩展；

f) sll指令中，若rt=rd=shamt=0，也不会进行任何操作，即为nop，不需要更改处理器结构。

2. 真值表：

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | PCSrc  [1:0] | Branch | RegWrite | RegDst  [1:0] | MemRead | Memwrite | MemtoReg[1:0] | ALUSrc1 | ALUSrc2 | ExtOp | LuOp |
| lw | 00 | 0 | 1 | 00 | 1 | 0 | 01 | 0 | 1 | 1 | 0 |
| sw | 00 | 0 | 0 | XX | 0 | 1 | XX | 0 | 1 | 1 | 0 |
| lui | 00 | 0 | 1 | 00 | 0 | 0 | 00 | 0 | 1 | X | 1 |
| add | 00 | 0 | 1 | 01 | 0 | 0 | 00 | 0 | 0 | X | X |
| Addu | 00 | 0 | 1 | 01 | 0 | 0 | 00 | 0 | 0 | X | X |
| Sub | 00 | 0 | 1 | 01 | 0 | 0 | 00 | 0 | 0 | X | X |
| Subu | 00 | 0 | 1 | 01 | 0 | 0 | 00 | 0 | 0 | X | X |
| Addi | 00 | 0 | 1 | 00 | 0 | 0 | 00 | 0 | 1 | 1 | 0 |
| Addiu | 00 | 0 | 1 | 00 | 0 | 0 | 00 | 0 | 1 | 1 | 0 |
| And | 00 | 0 | 1 | 01 | 0 | 0 | 00 | 0 | 0 | X | X |
| Or | 00 | 0 | 1 | 01 | 0 | 0 | 00 | 0 | 0 | X | X |
| Xor | 00 | 0 | 1 | 01 | 0 | 0 | 00 | 0 | 0 | X | X |
| Nor | 00 | 0 | 1 | 01 | 0 | 0 | 00 | 0 | 0 | X | X |
| Andi | 00 | 0 | 1 | 00 | 0 | 0 | 00 | 0 | 1 | 0 | 0 |
| Sll | 00 | 0 | 1 | 01 | 0 | 0 | 00 | 1 | 0 | X | X |
| Srl | 00 | 0 | 1 | 01 | 0 | 0 | 00 | 1 | 0 | X | X |
| Sra | 00 | 0 | 1 | 01 | 0 | 0 | 00 | 1 | 0 | X | X |
| Slt | 00 | 0 | 1 | 01 | 0 | 0 | 00 | 0 | 0 | X | X |
| Sltu | 00 | 0 | 1 | 01 | 0 | 0 | 00 | 0 | 0 | X | X |
| Slti | 00 | 0 | 1 | 00 | 0 | 0 | 00 | 0 | 1 | 1 | 0 |
| Sltiu | 00 | 0 | 1 | 00 | 0 | 0 | 00 | 0 | 1 | 0 | 0 |
| Beq | 00 | 1 | 0 | XX | 0 | 0 | XX | 0 | 0 | 1 | 0 |
| J | 01 | X | 0 | XX | 0 | 0 | XX | X | X | X | X |
| Jal | 01 | X | 1 | 10 | 0 | 0 | 10 | X | X | X | X |
| Jr | 10 | X | 0 | XX | 0 | 0 | XX | X | X | X | X |
| jalr | 10 | X | 1 | 01 | 0 | 0 | 10 | X | X | X | X |

1. **完成控制器**
2. control.v代码补充：
3. 理解汇编指令：

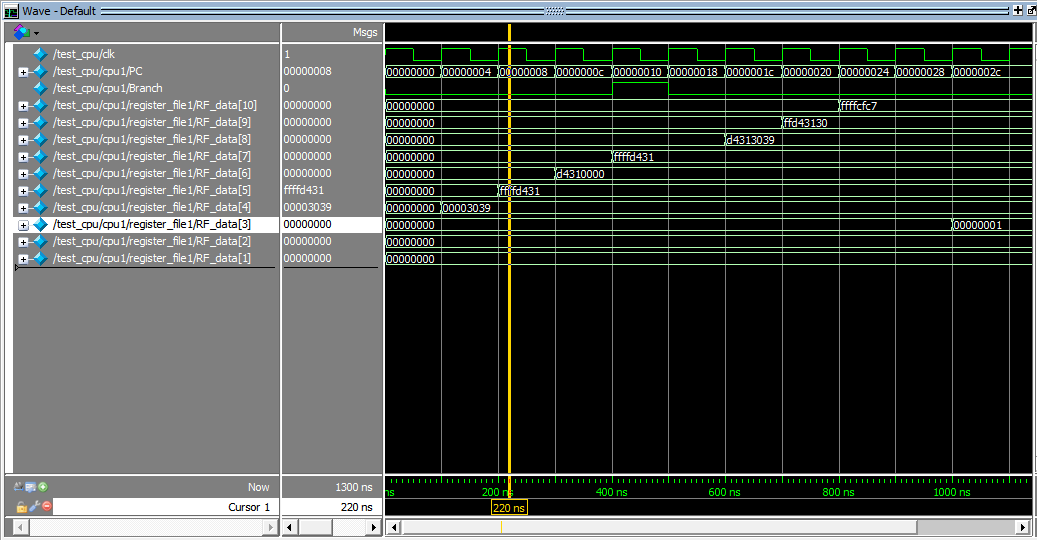
这段程序执行足够长时间后会进入死循环状态保证程序不会结束；此时各寄存器对应值如下图所示：

|  |  |  |
| --- | --- | --- |
| 寄存器 | 对应数值 | 备注 |
| $a0 | 0x00003039 | a0=0x0+0x3039 |
| $a1 | 0xffffd431 | a1=0x0+(按符号位扩展)0xd431 |
| $a2 | 0xd4310000 | a2=a1<<16 |
| $a3 | 0xffffd431 | a3=a2>>16 |
| $t0 | 0xd4313039 | t0=a1+a2 |
| $t1 | 0xffd43130 | t1=t0>>8(符号位扩展) |
| $t2 | 0xffffcfc7 | t2=0x0+0xffffcfc7 |
| $v0 | 0x00000000 | v0=(a0(12345)<t2(-12345)?1:0 |
| $v1 | 0x00000001 | v1=(a0(12345)<t2(2^32-12345)?1:0 |

已知某一时刻在某寄存器中存放着数0xffffcfc7，不可以判断其是否有符号，因为addi、lui等操作也会产生相应的数值；

1. 仿真情况：

仿真结果如下图所示：



1. PC在指令顺序执行无跳转的情况下每次增加4，出现beq时会根据偏移量增加；由于offset=1，PC共增加8；
2. Branch在400-500ns时为1，由于该周期内执行beq指令需要产生分支跳转；引起了PC=PC+8；
3. 100-200ns时PC=0x04，对应的指令为addiu $a1, $zero, -11215，由于还未到达下一个时钟上升沿寄存器还未写入，此时$a1=0x0；200-300ns（即下一个周期内）$a1=0xffffd431为正确的计算结果，因为只有到达下一个时钟上升沿时寄存器写入才会生效；即使下一周期用到了$a1的值也不会报错，因为单周期CPU相邻周期不会互相干扰；
4. 运行足够长时间后各寄存器数值如图所示([4]-[7]对应$a0-$a3，[8]-[10]对应$t0到$t2，[2]-[3]对应$v0和$v1)，与预期一致。
5. **执行汇编程序**

addi $a0, $zero, 3 # a0 = 3

jal sum # goto sum;$ra = PC+4 = Loop

Loop:

beq $zero, $zero, Loop # loop forever to make sure not end

sum:

addi $sp, $sp, -8 # sp -= 8

sw $ra, 4($sp) # stack\_push($ra)

sw $a0, 0($sp) # stack\_push($a0)

slti $t0, $a0, 1 # t0 = (a0 < 1) ? 1 : 0

beq $t0, $zero, L1 # if (a0 >= 1) goto L1

xor $v0, $zero, $zero # v0 = 0

addi $sp, $sp, 8 # sp += 8

jr $ra # return Loop

L1:

addi $a0, $a0, -1 # a0 -= 1

jal sum # goto sum;$ra = PC+4

lw $a0, 0($sp) # stack\_pop($a0)

lw $ra, 4($sp) # stack\_pop($ra)

addi $sp, $sp, 8 # sp += 8

add $v0, $a0, $v0 # v0 += a0

jr $ra # jump to Register: $ra

1. 这段程序执行计算1+2+……+n的功能；Loop为主程序结束后死循环保证程序不退出；sum为递归函数，等价于

int sum(int n){

if(n<1) return 0;

else return n+sum(n-1);

}

L1则反复调用sum函数完成v0的累加和循环变量a0的自减；

1. 机器码如下：

|  |  |  |
| --- | --- | --- |
| 指令 | 指令地址 | 机器码 |
| addi $a0, $zero, 3 | 00400000 | *20040003* |
| jal sum | 00400004 | *0c100003* |
| beq $zero, $zero, Loop | 00400008 | *1000ffff* |
| addi $sp, $sp, -8 | 0040000c | *23bdfff8* |
| sw $ra, 4($sp) | 00400010 | *afbf0004* |
| sw $a0, 0($sp) | 00400014 | *afa40000* |
| slti $t0, $a0, 1 | 00400018 | *28880001* |
| beq $t0, $zero, L1 | 0040001c | *11000003* |
| xor $v0, $zero, $zero | 00400020 | *00001026* |
| addi $sp, $sp, 8 | 00400024 | *23bd0008* |
| jr $ra | 00400028 | *03e00008* |
| addi $a0, $a0, -1 | 0040002c | *2084ffff* |
| jal sum | 00400030 | *0c100003* |
| lw $a0, 0($sp) | 00400034 | *8fa40000* |
| lw $ra, 4($sp) | 00400038 | *8fbf0004* |
| addi $sp, $sp, 8 | 0040003c | *23bd0008* |
| add $v0, $a0, $v0 | 00400040 | *00821020* |
| jr $ra | 00400044 | *03e00008* |

beq $zero, $zero, Loop中，Loop对应该指令本身，因此设置offset=-1即可；

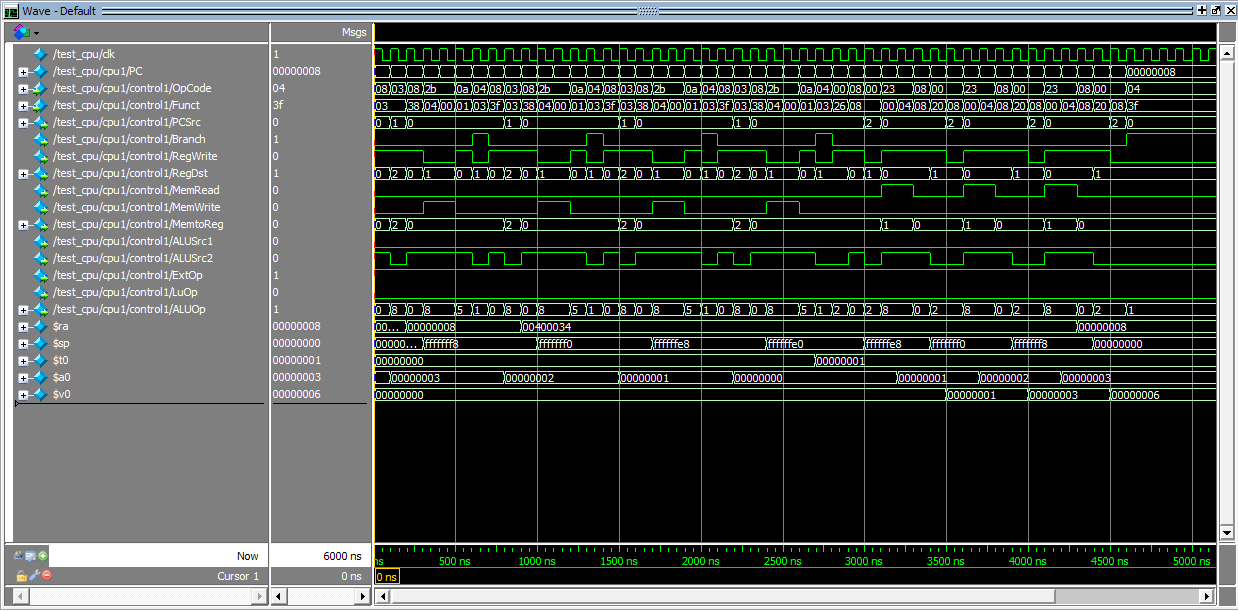
beq $t0, $zero, L1中，L1对应“addi $a0, $a0, -1”，设置offset=3即可；

jal sum将sum翻译为“addi $sp, $sp, -8”该指令的地址00400030/4=0010000c；

立即数-1=0xffff，-8=0xfff8；

1. InstructionMemory.v代码如下：
2. module InstructionMemory(Address, Instruction);
3. input [31:0] Address;
4. output reg [31:0] Instruction;
6. always @(\*)
7. case (Address[9:2])
8. 8'd0:   Instruction <= 32'h20040003;
9. 8'd1:   Instruction <= 32'h0c100003;
10. 8'd2:   Instruction <= 32'h1000ffff;
11. 8'd3:   Instruction <= 32'h23bdfff8;
12. 8'd4:   Instruction <= 32'hafbf0004;
13. 8'd5:   Instruction <= 32'hafa40000;
14. 8'd6:   Instruction <= 32'h28880001;
15. 8'd7:   Instruction <= 32'h11000003;
16. 8'd8:   Instruction <= 32'h00001026;
17. 8'd9:   Instruction <= 32'h23bd0008;
18. 8'd10:  Instruction <= 32'h03e00008;
19. 8'd11:  Instruction <= 32'h2084ffff;
20. 8'd12:  Instruction <= 32'h0c100003;
21. 8'd13:  Instruction <= 32'h8fa40000;
22. 8'd14:  Instruction <= 32'h8fbf0004;
23. 8'd15:  Instruction <= 32'h23bd0008;
24. 8'd16:  Instruction <= 32'h00821020;
25. 8'd17:  Instruction <= 32'h03e00008;
26. default: Instruction <= 32'h00000000;
27. endcase
29. endmodule

仿真结果如下：



可以观察到：  
a）运行足够长时间之后，a0=3，v0=6，符合预期程序功能。

b）各寄存器变化情况：

（PC默认从0开始取址，但不影响程序执行）

PC：00000000->00000004[jal sum]->{0040000c->……（顺序执行[+4]）->00400030[jal sum]->}……（循环：a0=3->2->1）->0040000c->……->00400028->{00400034->……（顺序执行[+4]）->00400044->}（循环：a0=0->1->2->3）->00000008->死循环；

a0：3->2->1->0->1->2->3，即a0先入栈、自减……再出栈，自增的过程，也就是递归调用的变量改变情况；

v0：0->1->3->6 a0出栈后不断累加

$sp：0->-8->-16->-24->-32->-24->-16->-8->0 栈指针先不断减小，即a0和ra不断入栈，再不断出栈的过程；

$ra：0x0 -> 0x8 -> 0x00400034 -> 0x8 ra遇到jal指令时会发生变化，与PC相同，初始的0x8不影响结果。