数字逻辑与处理器基础
多周期处理器

cvxbzn

2021 年 6 月 5 日

目录

1　数据通路设计

1.1　寄存器与多路选择器及其功能

- 寄存器
  - PC：输入下一状态 PC，输出当前 PC，使能信号为 1 时在时钟上升沿写入。
  - 指令寄存器 (IR)：存放从存储器中取出的指令，IRWrite 为 1 时在时钟上升沿写入
  - 数据寄存器 (MDR)：输入存储器的输出 MemData。
  - 临时寄存器 A：输入为寄存器堆中的 ReadData1
  - 临时寄存器 B：输入为寄存器堆中的 ReadData2
  - ALU 寄存器 ALUOut：存放 ALU 输出 ALUOut
- 多路选择器
  - IorD：Instruction or data，选择读取存储器的数据为指令 (0) 或者数据 (1)。
  - RegDst：选择写回寄存器堆中的寄存器位置，rt(00),rd(01) 或者 $ra(10)，其中 $ra 在执行 jal 指令时使用。
  - MemToReg：选择写回寄存器堆的数据来源。ALUOut(00)，内存数据 (01)，PC(10)。
  - ALUSrcA：选择 ALU 操作数 1 的数据来源，PC(00), 临时寄存器 A(01), 以及在移位时用到的 Shamt(10)
  - ALUSrcB：选择 ALU 操作数 2 的数据来源，临时寄存器 B(00)，常数 4(01)，立即数 (10) 以及移位后的立即数 (11)
  - PCSource：选择更新 PC 时 PC 的来源，PC+4(00)，分支指令计算结果 (01) 以及伪直接寻址 (10)

具体情况如下示意图所示，代码见附件。

1.2　示意图

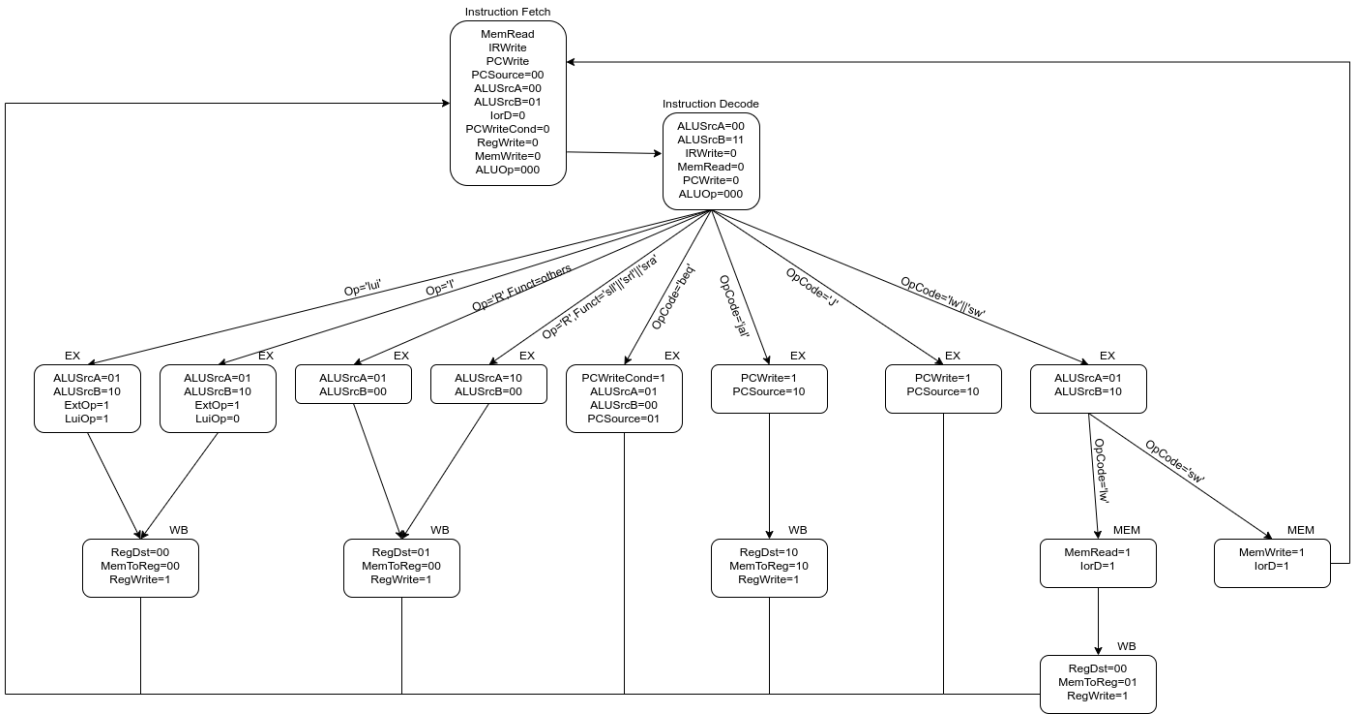
2　控制信号分析与有限状态机实现

2.1　控制信号及具体功能

- PCWrite：PC 寄存器的写使能信号；0-不能写 PC；1-允许写 PC
- PCWriteCond：分支指令 PC 写使能信号，0-无分支指令；1-有分支指令
- IorD：选择读取存储器的数据信号，0-指令，1-数据

- MemWrite：内存的写使能信号；0-不能写内存；1-允许写内存
- MemRead：内存的读使能信号；0-不能读内存；1-允许读内存
- IRWrite：指令寄存器的写使能信号；0-不能写；1-允许写
- MemToReg：写回寄存器堆数据来源选择信号；00-ALUOut，01-内存数据，10-PC
- RegDst：写回寄存器堆的寄存器位置选择信号，00-rt,01-rd 或者 10-$ra
- RegWrite：寄存器堆的写使能信号；0-不能写；1-允许写
- ExtOp：符号扩展信号，0-逻辑扩展；1-算术扩展
- LuiOp：Lui 控制信号，0-不左移 16 位；1-左移 16 位
- ALUSrcA：ALU 操作数 1 数据来源选择信号；00-PC；01-临时寄存器 A；10-Shamt
- ALUSrcB：ALU 操作数 2 数据来源选择信号；00-临时寄存器 B；01-常数 4；10-立即数；11-移位后的立即数
- ALUOp：ALU 控制信号；000-ADD；001-SUB；100-AND；101-SLT；010-取决于 Funct
- PCSource：PC 更新来源选择信号；00-PC+4；01-分支指令计算结果；10-伪直接寻址

2.2 状态转移图



3 ALU 功能拓展

3.1 setsub 类型和机器码字段内容

setsub rd rs rt：{6'h0,rs[4:0],rt[4:0],rd[4:0],5'h0,6'h28}

3.2 ALU verilog 代码修改

对应 ALUConf 设置为 5'b00011，设计执行程序为

```
lui $a0 0xABCD
addi $a0 $a0 0x1234
lui $a1 0xCDEF
addi $a1 $a1 0x3456
setsub $a2 $a0 $a1
Loop:
beq $zero $zero Loop
```

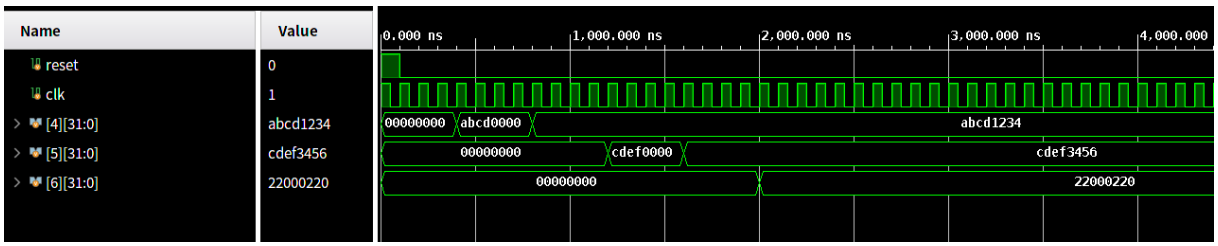
对应机器码为

```
{6'h0f,5'd0,5'd4,16'habcd}
{6'h08,5'd4,5'd4,16'h1234}
{6'h0f,5'd0,5'd5,16'hcdef}
{6'h08,5'd5,5'd5,16'h3456}
{6'h0,5'd4,5'd5,5'd6,5'h0,6'h28}
{6'h04,5'd0,5'd0,16'hffff}
```

具体代码见附件。

3.3 仿真结果

0xabcd1234 setsub 0xcdef5678
= 1010_1011_1100_1101_0001_0010_0011_0100 setsub 1100_1101_1110_1111_0011_0100_0101_0110
= 0010_0010_0000_0000_0000_0010_0010_0000 = 0x2200_0220
仿真结果如图



与预期一致。

4 汇编程序分析-1

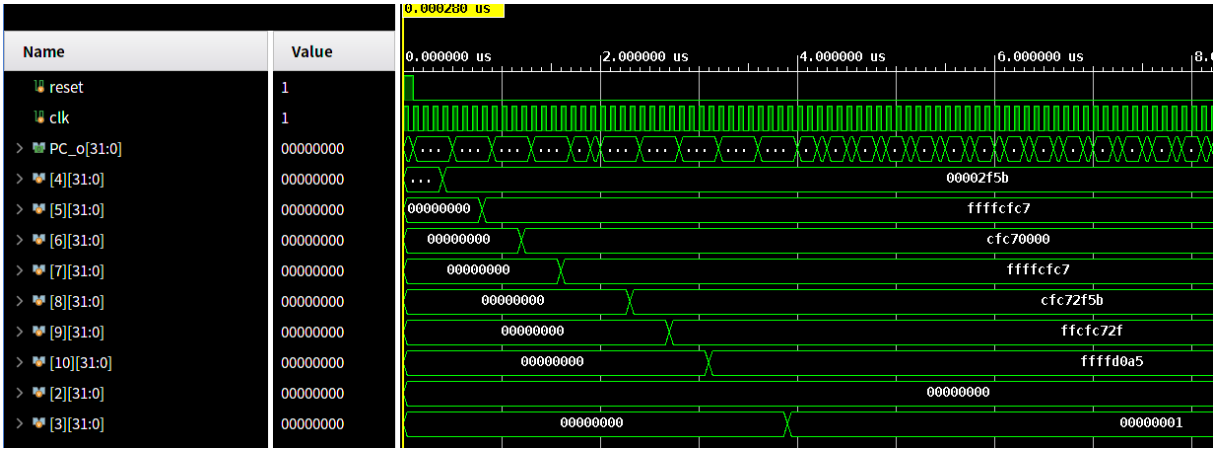
4.1 计算寄存器值

```
addi $a0, $zero, 12123      # $a0 = 0x0000_2f5b
addiu $a1, $zero, -12345    # $a1 = 0xffff_cfc7
sll $a2, $a1, 16            # $a2 = 0xcfc7_0000
sra $a3, $a2, 16            # $a3 = 0xffff_cfc7
beq $a3, $a1, L1            # 跳转至L1
lui $a0, 22222              # 不执行

L1:
add $t0, $a2, $a0           # $t0 = 0xcfc7_2f5b
sra $t1, $t0, 8             # $t1 = 0xffcf_c72f
addi $t2, $zero, -12123     # $t2 = 0xffff_d0a5
slt $v0, $a0, $t2           # 有符号情况下 $a0 > $t2 ,$v0=1
sltu $v1, $a0, $t2          # 无符号情况下 $a0 < $t2 ,$v1=0

Loop:
j Loop
```

4.2 仿真结果



可见与计算结果一致。

5 汇编程序分析-2

5.1 程序功能以及代码注释

```
addi $a0, $zero, 5      # $a0 = 5
xor $v0, $zero, $zero   # $v0 = 0
jal sum                 # 跳转到sum
Loop:
beq $zero, $zero, Loop  # 程序结束
sum:
addi $sp, $sp, -8       # 入栈前栈指针-8
sw $ra, 4($sp)          # $ra 入栈
sw $a0, 0($sp)          # $a0 入栈
slti $t0, $a0, 1        # if $a0 < 1 $t0 = 1
beq $t0, $zero, L1      # if $a0 >= 1, goto L1
addi $sp, $sp, 8        # 出栈恢复栈指针
jr $ra                  # 回到上一个过程$ra存的位置
L1:
add $v0, $a0, $v0       # $v0 = $a0 + $v0
addi $a0, $a0, -1       # $a0 = $a0 - 1
jal sum                 # 迭代
lw $a0, 0($sp)
lw $ra, 4($sp)
addi $sp, $sp, 8        # 出栈恢复栈指针
add $v0, $a0, $v0       # 又加一次, $v0 = $a0 + $v0
jr $ra
```

可见每次循环相当于加了两次 $a0，因此如果第一行的 5 是任意正整数 n，程序实现了 $2\sum_{i=1}^{n} i$ 的功能。此外由于 lw，$a0 最终值仍为 n。

5.2 将汇编翻译成机器码并写出

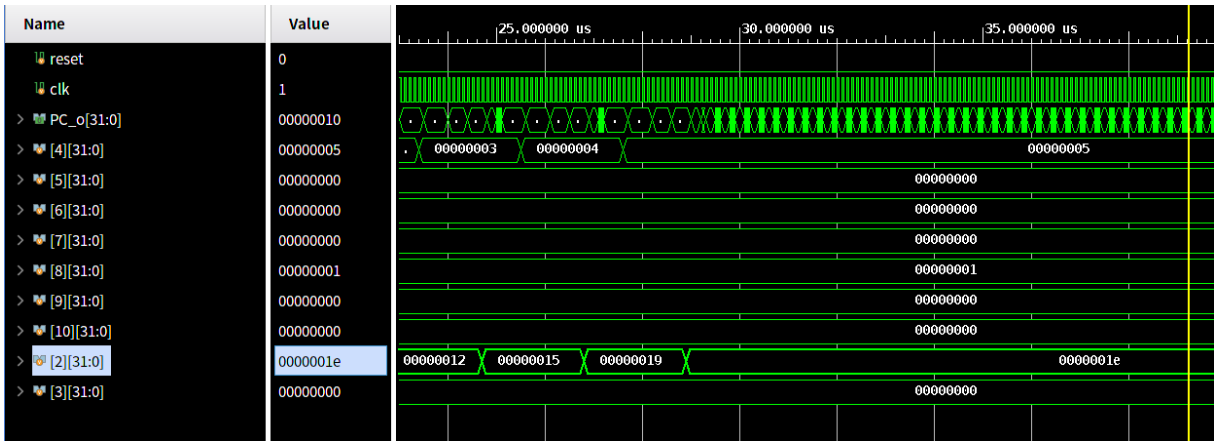
```
{6'h08,5'd0,5'd4,16'd5}
{6'h0,5'd0,5'd0,5'd2,5'h0,6'h26}
{6'h03,26'd4}
{6'h04,5'd0,5'd0,16'hffff}
{6'h08,5'd29,5'd29,16'dfff8}
{6'h2b,5'd29,5'd31,16'd4}
{6'h2b,5'd29,5'd4,16'd0}
```

```
{6'h0a,5'd4,5'd8,16'd1}
{6'h04,5'd8,5'd0,16'd2}
{6'h08,5'd29,5'd29,16'd8}
{6'h0,5'd31,15'd0,6'h08}
{6'h0,5'd4,5'd2,5'd2,5'd0,6'h20}
{6'h08,5'd04,5'd04,16'hffff}
{6'h03,26'h4}
{6'h23,5'd29,5'd4,16'd0}
{6'h23,5'd29,5'd31,16'd4}
{6'h08,5'd29,5'd29,16'd8}
{6'h0,5'd4,5'd2,5'd2,5'd0,6'h20}
{6'h0,5'd31,15'd0,6'h08}
```

修改文件位于 InstAndDataMemory_1.v

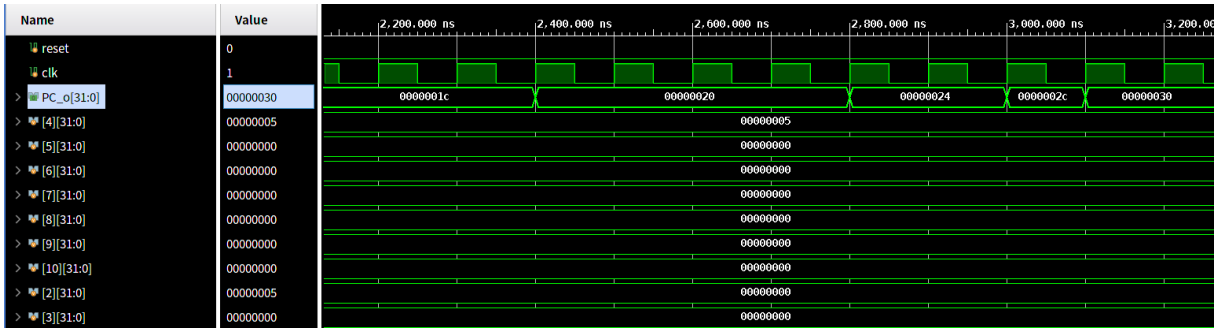
5.3 $a0,$v0 值

根据上述分析，$v0 值为 30，即 0x1e，$a0 值为 5，仿真结果如图



5.4 观察、描述并解释寄存器如何变化

PC 在 sIF 状态结束时 +4，在 beq、j、jal、jr 的 EX 阶段结束时 PC 跳转到对应的地址，如下图所示。



$a0 从 5 减小至 0 再增加至 0，

```
addi $a0, $a0, -1
```

使 $a0 减小，减至 0 后 lw 指令装载之前 $a0 值。
$v0 是 $a0 累加，

$$\$v0 = 5+4+3+2+1+1+2+3+4+5$$

入栈时 $sp 减 8，当 $a0 等于 0 时开始出栈，$sp 加 8。
$ra 在执行 jal 指令时改变，第一次 jal 时 $ra 为 0x0c，之后为 0x38，最后一次 lw 后 $ra 为 0x0c。

6 异常处理

6.1 控制信号与功能

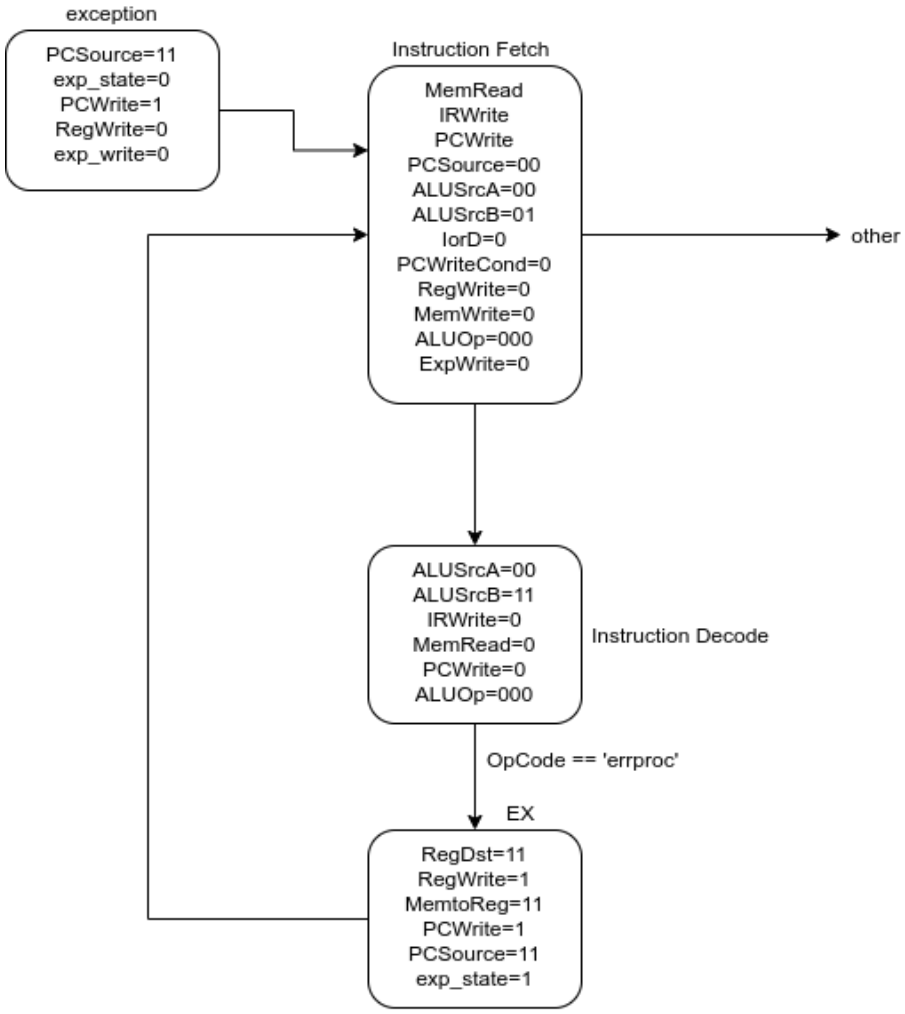
- exp_flag：异常判断信号，0-无异常，1-异常
- exp_write：异常寄存器 (EPC,ErrorTarget) 写信号，0-不能写，1-允许写
- exp_state：异常 PC 跳转选择信号，0-0x7c，1-EPC

6.2 指令格式与状态转移图

由于异常处理程序只涉及将 0xffff_ffff 写回目标寄存器无额外功能，因此只需要分配新 OpCode 加以区分。指令格式

```
{6'h05,26'h0}
```

状态转移图为



6.3 仿真

汇编程序为

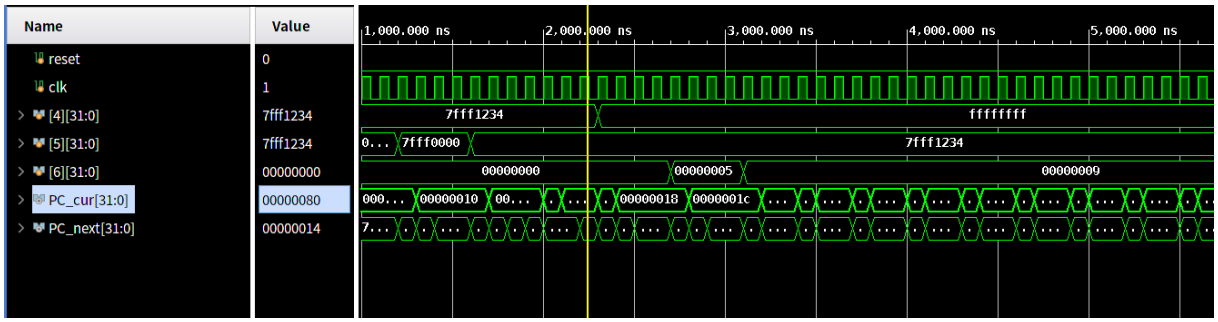
```
lui $a0 0x7fff
addi $a0 $a0 0x1234
lui $a1 0x7fff
addi $a1 $a1 0x1234
add $a0 $a0 $a1
addi $a2 $0 5
addi $a2 $a2 4
Loop:
beq $zero $zero Loop
```

对应机器码为

```
{6'h0f,5'd0,5'd4,16'h7fff}
{6'h08,5'd4,5'd4,16'h1234}
{6'h0f,5'd0,5'd5,16'h7fff}
{6'h08,5'd5,5'd5,16'h1234}
{6'h0,5'd4,5'd5,5'd4,5'h0,6'h20}
{6'h08,5'd0,5'd6,16'd5}
{6'h08,5'd6,5'd6,16'd4}
{6'h04,5'd0,5'd0,16'hffff}
```

程序写在 InstAndDataMemory_exp.v 中，需修改顶层文件对应模块名称。这段程序 $a0 会溢出，最终值应为 ffff_ffff，而 $a2 继续执行，最终值应为 9。

仿真结果为



与预期一致。