计算机组成原理

Homework Solution 3

Made by TA

2023 年 4 月 8 日

> 每小问扣0.5分，按时完成了所有题目的提交不少于5分。

> 由于大家错误比较多，且基本都错在同一个地方，所以改作业基本就只标了错误的题号，错误的点写在了下面的解析中。

题目 1. 假设寄存器 t_0 中初始状态下保存的值为 0x00002023。请回答下面的问题:

1. 对于指令 sub t2, t0, t1, 导致结果溢出的 t1 的值的范围？
2. 假设 PC（程序计数器）当前值为 0x0D000000, 则 jal 指令可以到达的地址范围是多少？如果是 blt 呢？

提示: jal 和 blt 的跳转是连续的吗？

> **开幕雷击**:这道题基本一眼望过去全是-1，全对的人不超过20，主要错误如下：
>
> 1. 第一问的问题如下：
>
>   - 未使用补码计算,基本0x7开头的0xf开头的都是这个错误，寄存器中存储的是补码范围为$[-2^{31}, 2^{31}-1]$,如果按有符号原码表示的话范围是$[-(2^{31}-1), 2^{31}-1]$相比补码表示范围更少。
>   - 注意边界能否取到的问题,这个基本就是细心的问题了。
>   - 各种计算错误。

2. • 第二小问所给的提示主要强调两个信息：
   - jal和blt跳转指令是在原offset的基础上最低位补0,这使得jal跳转的立即数偏移需要乘以2的倍数。
   - 由于最低位只能是0，所以jal和blt的跳转范围是不连续的，间隔的，对应这道题就是只能跳到偶数地址。

   > 这两个点都有人注意到，但都对了寥寥无几，甚至于两个都注意到的人中还有不少有计算错误。

   • 未注意偏移量为补码表示（是可正可负的），如jal的20位立即数的范围是$[-2^{19}, 2^{19}-1]$而非$[-(2^{20}-1), 2^{20}-1]$（还有各种各样的位数，基本上把位数弄对的就不超过一半）.

解答:

1. 考虑不溢出的情况$t0-t1 \in [-2^{31}, 2^{31}-1]$,$t1$本身也在这个范围内，从而$t0-t1 \geqslant -2^{31}$是恒成立的，只需要保证$t0-t1 \leqslant 2^{31}-1$即可，即$t1 \in [8228-2^{31}, 2^{31}-1]$ 故导致结果溢出的t1的值的范围是$[-2^{31}, 8227-2^{31}]$，也就是$[0x80000000, 0x80002023]$。
2. jal offset的范围是$[-2^{20}, 2^{20}-2]$(也就是$[-2^{19}, 2^{19}-1] \times 2$), blt offset的范围是$[-2^{12}, 2^{12}-2]$(同理)

> 值得注意的是jalr使用寄存器加立即数的偏移方式，可以访问“全部”地址空间，但由于对齐要求会将计算得到的偏移的最低位置0，同样跳转范围是不连续的。

所以jal可以到达的地址范围是$[0x0D000000-2^{20}, 0x0D000000+2^{20}-2]$, blt可以到达的地址范围是$[0x0D000000-2^{12}, 0x0D000000+2^{12}-2]$。即$jal \in [0x0CF00000, 0x0D0FFFFE]$, $blt \in [0x0CFFF000, 0x0D000FFE]$，**同时均只能到达第0位为0的地址**。

imm[20|10:1|11|19:12]				rd	1101111	J jal
imm[11:0]		rs1	000	rd	1100111	I jalr
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	B beq
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	B bne
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	B blt
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	B bge
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	B bltu
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	B bgeu

题目 2. 本题目中所述的 int 整型变量都是 32 位的。

1. 将 $-a*2-(b+c)-(d+b+c)+200$ 转换为 RV32I 指令（a, b, c, d 均为 int 整型数值，且已经分别保留在寄存器 t0, t1, t2, t3 中。不考虑溢出问题）。请尝试使用尽可能少的寄存器和尽可能少的指令。
2. 将 $A[2*j]=B[i-8]$ 转换为 RV32I 指令，其中 A, B 为 int 整型数组。它们的基址分别保存在寄存器 a0, a1 中。 i, j 均为 int 整型变量，且已保存在寄存器 t0, t1 中（不考虑溢出以及非法访问问题，所有数据都已经四字节对齐）。

> 尽可能为你的指令添加注释，使得助教能够更好的理解你的代码。

主要错误如下：

1. 用了8条及以上指令的都扣了分，主要就是忽略了x0寄存器恒为0，导致使用了额外的指令。还有一些错误原因如下：
   - 用了mul指令，而rv32i没有mul这条指令
   - xori/not 取反后忘了加1，或加1没有合并到常数200中导致指令数增加。
2. 这道题集中于指令格式错误，和偏移量计算错误，错误如下：
   - lw，sw指令格式错误，lw t0 t1(t2)这种写法是错误的，只有lw R IMM(R),这种格式（R代表寄存器，IMM代表立即数）
   - 用了ld指令，ld指令是64位的，而这道题数据说明了是32位的，所以不能用ld指令

> lh:load half byte, lb: load byte, lw: load word, ld: load double word

> - 用了mul指令，而rv32i没有mul这条指令
> - **int为32位也就是4字节，应该在原数组偏移基础上乘4**,RISC-V是按字节寻址的，所以*32的，*8的，没乘的都是错误的。
> - 2次幂的乘法很容易通过移位来实现，ics应该经常使用这点，不知道大家为什么在这里基本都是用一长串加法来实现的。（不是扣分项）

解答:

1. 容易注意到b+c是不必重复计算的，可以如果旧数据不需要了，可以直接覆盖，所以可以使用如下指令：

```
sub t0, x0, t0 // t0 = -a
slli t0, t0, 1 // t0 = -2a
add t1, t1, t2 // t1 = b + c
sub t0, t0, t1 // t0 = -2a - (b + c)
add t3, t3, t1 // t3 = d + b + c
sub t0, t0, t3 // t0 = -2a - (b + c) - (d + b + c)
addi t0, t0, 200  // t0 = -2a - (b + c) - (d + b + c) + 200
```

   或者将算式展开$-2(a+b+c)-d+200$

```
sub t0 , x0 , t0 //t0 = -a
sub t0 , t0 , t1 //t0 = -a-b
sub t0 , t0 , t2 //t0 = -a-b-c
slli t0, t0, 1    //t0 = -2(a+b+c)
sub t0 , t0 , t3 //t0 = -(a+b+c)-d
addi t0 , t0 , 200 //t0 = -2(a+b+c)-d+200
```

2. 需要注意数组下标每一位对应4个字节,因此需要在数组原偏移基础上乘4.

```
addi t0, t0, -8 // t0 = i - 8
slli t0, t0, 2 // t0 = 4 * (i - 8) get the offset of B
add t0, t0, a1 // & B[i-8]
slli t1, t1, 3 // t1 = 4 * 2j get the offset of A
```

```
add t1, t1, a0 // & A[2*j]
lw t2, 0(t0) // t2 = B[i-8]
sw t2, 0(t1) // A[2*j] = B[i-8]
```

题目 3. 我们知道 RISC-V 存储是小端序的，即低地址存储低位，高地址存储高位。阅读如下代码：

```
lb t1, 1(t0);
sw t1, 4(t0);
```

初始条件下，t0 的内容为 0x2023, 地址 0x2023 的内容为 0x20881124。请问:

1. 该代码执行后，地址 0x202A 的内容是什么？
2. 如果 RISC-V 是大端序存储的，那么该代码执行后，地址 0x202A 的内容是什么？

> 本以为大小端是最阴间的题，没想到反而是这道题对的人最多，这种题一般只要画个图就很容易，注意不要掉了前面的0x

解答:

1. t1：0x00000011, 0x202A：0x00

0x2023	0x2024	0x2025	0x2026
0x24	0x11	0x88	0x20
0x2027	0x2028	0x2029	0x202A
0x11	0x00	0x00	0x00

2. t1：0xffffff88, 0x202A：0x88

0x2023	0x2024	0x2025	0x2026
0x20	0x88	0x11	0x24
0x2027	0x2028	0x2029	0x202A
0xff	0xff	0xff	0x88

在最新版的RISC-V手册中(群文件中有)，不再支持访存指令的非对齐访问，所以本题中访存在最新版的手册中是不合法的。

0	4	Load address misaligned
0	5	Load access fault
0	6	Store/AMO address misaligned
0	7	Store/AMO access fault

它们会引起page fault。至于原因，非对齐的访问可能会非常慢，并且无法保证原子性, 需要额外的同步机制。

> The problem you are having is that you are trying to use lw with the address (t1 << 2) + t0. T0 is equal to 1 on first use (unless it has been modified in the meantime) which gives an unaligned word address. Not being supported (or at least not by default) this gives the error you saw.
>
> May be misaligned access are disabled by default. There what the specification tell:
>
> > For best performance, the effective address for all loads and stores should be naturally aligned for each data type (i.e., on a four-byte boundary for 32-bit accesses, and a two-byte boundary for16-bit accesses). The base ISA supports misaligned accesses, but these might run extremely slowly depending on the implementation. Furthermore, naturally aligned loads and stores are guaranteed to execute atomically, whereas misaligned loads and stores might not, and hence require additional synchronization to ensure atomicity.
>
> To avoid the error you are facing there is two possibilities:
> You really need to do misaligned access and you have to enable this feature.
> What you want is read the first word , second ... then what you need to do is to shift the elements you get from the array (by 2 to the left).

所以RISC-V的设计者决定不再支持非对齐访问。

题目 4. 现在我们需要使用 RV32I 指令求解斐波那契数列的前 n 项，其中 n 为 int 整型变量，保存在内存地址 place 中。斐波那契数列的第一项和第二项分别保存在内存地址 first 和 second 中。请根据以上信息编写 RV32I 指令，将从第一项开始的结果依次保存在从内存地址 save 开始的连续内存中。

这道题出现的错误主要是大家把la（load address）当作加载数据来用了，其实它是加载地址，所以要先把地址加载到寄存器中，然后再进行访存操作。还有不少同学创造了各种奇奇怪怪的访存指令。

> 一些同学使用了lw t0 place这样的伪指令，助教后来在rars上试了一下是可以的，重新补回了这样写的同学的分数，如果有遗漏请及时联系助教。

> lw t0 0(place)这样的写法还是非法的。

解答:

```
.text
    la a0 first
    la a1 second
    la a2 place
    la s1 save #get save
    lw s5 0(a2)#get n
    beq s5 x0 end
    lw s2 0(a0)#ld fisrt element
    sw s2 0(s1)
    addi s1 s1 0x4
    addi s5 s5 -1
    beq s5 x0 end
    lw s3 0(a1)#ld second element
    sw s3 0(s1)
    addi s1 s1 0x4
    addi s5 s5 -1
    beq s5 x0 end
loop:          #loop for next element
    add s4 s2 s3
    add s2 x0 s3
    add s3 x0 s4
    sw s4 0(s1)
    addi s1 s1 0x4
    addi s5 s5 -1
    beq s5 x0 end
```

```
        j   loop
end :
        nop
```

实验题 1. lab0 中的实验题