《计算机组成原理》第三次作业

网络空间安全学院 信息安全 陆皓喆 2211044

2.22

假如程序计数器被设置为`0x20000000`。

2.22.1

如果使用了MIPS的跳转链接（`jal`）指令，可以转移到的地址范围是多少?

跳转链接的地址变化范围是最后的26位，因为跳转指令的address占了26位，所以变成3FFFFFF，左移两位后，变成FFFFFFC。

所以，跳转的范围是，**0x10000000-0x2FFFFFFC**

2.22.2

如果使用了MIPS的相等则分支（`beq`）指令，可以转移到的地址范围是多少?

`beq`指令的地址变化范围只有16位，因为分支指令的address只占16位。我们寻址的地址应该是，PC地址加上4，然后分别最小的地址是，减去2的17次，最大的地址是，加上2的17次再减去4，这样就构成了我们能够跳转的地址范围。

所以，跳转的范围是，**0x1FFE0004-0x20020000**

2.29

使用MIPS汇编语言实现下面的C代码。指示：栈指针必须保持为16的整数倍。

```
int fib(int n){
  if (n==0)
    return 0;
  else if (n == 1)
    return 1;
  else
    return fib(n-1) + fib(n-2);
}
```

我们使用MIPS汇编语言来编写代码。

```
fib:
  addi $sp,$sp,-8
  sw $ra,4($sp)
  sw $a0,0($sp)
  slt $t0,$a0,2
  beq $t0,$zero,L1
  addi $t1,$zero,1
  beq $a0,$t1,L2
```

```
  addi $v0,$zero,0
  addi $sp,$sp,8
  jr $ra

L2:
  addi $v0,$zero,1
  addi $sp,$sp,8
  jr $ra

L1:
  addi $a0,$a0,-1
  jal fib
  lw $a0,0($sp)
  sw $a0,0($sp)
  addi $a0,$a0,-2
  jal fib

  lw $t1,0($sp)
  add $v0,$v0,$t1
  lw $ra,4($sp)

  addi $sp,$sp,8
  jr $ra
```

2.31

Translate function f into MIPS assembly language. If you need to use registers $t0 through $t7, use the lower-numbered registers first. Assume the function declaration for func is "int f(int a, int b);". Th e code for function f is as follows:

```
int f(int a, int b, int c, int d){
   return func(func(a,b),c+d);
}
```

我们使用MIPS汇编语言来编写代码。

```
f:
  addi $sp,$sp,-4
  sw $ra,0($sp)
  jal func
  addi $a0,$v0,0
  add $a1,$a2,$a3
  jal func

  lw $ra,0($sp)
  addi $sp,$sp,4
  jr $ra
```

2.35

对于如下代码：

```
lbu $t0, 0($t1)
sw $t0, 0($t2)
```

假设寄存器$t1中存放地址0x10000000，该地址存放的数据为0x11223344。

2.35.1

在大端地址的机器中，存储在0x10000004处的数据是多少?

大端存储的形式为：11存储在0x10000000，22存储在0x10000004,33存储在0x10000008,44存储在0x1000000C

第一句代码的意思是，将$t1的第一个位置的值，即0x00000011存储到$t0

所以，存储在0x10000004的位置应该是**0x22**

2.35.2

在小端地址的机器中，存储在0x10000004处的数据是多少?

跟上面是一样的意思，只不过存储的内容是0x33，所以存储在0x100000004的位置应该是**0x33**

2.39

Assume for a given processor the CPI of arithmetic instructions is 1,the CPI of load/store instructions is 10, and the CPI of branch instructions is 3.Assume a program has the following instruction breakdowns: 500 million arithmetic instructions, 300 million load/store instructions, 100 million branch instructions.

2.39.1

Suppose that new, more powerful arithmetic instructions are added to the instruction set. On average, through the use of these more powerful arithmetic instructions, we can reduce the number of arithmetic instructions needed to execute a program by 25%, and the cost of increasing the clock cycle time by only 10%. Is this a good design choice? Why?

这应该不是一个好的选择。

原来的情况下，CPUtime是：

$$CPUtime = (500 \times 1 + 300 \times 10 + 100 \times 3) \times 10^6 \times cycletime = 3.8 \times 10^9 \times cycletime$$

在修改之后，我们的算术指令变成了375millon条，所以现在的CPUtime变成了3675millon乘上cycletime。

题目说，cycletime变成了原来的1.1倍，那么现在的CPUtime应该变成了：

$$CPUtime = (375 \times 1 + 300 \times 10 + 100 \times 3) \times 10^6 \times cycletime \times 1.1 = 4.0425 \times 10^9 \times cycletime$$

所以相比于原来的CPUtime，现在的CPUtime相对来说是比较长的

所以，这不是一个好的选择。

2.39.2

Suppose that we find a way to double the performance of arithmetic instructions. What is the overall speedup of our machine? What if we find a way to improve the performance of arithmetic instructions by 10 times?

按照上一问的公式，我们分别计算出将算术指令的时钟时长。

两倍：

$$\frac{CPUtime}{cycletime} = (500 \times 0.5 + 300 \times 10 + 100 \times 3) \times 10^6 = 3.55 \times 10^9$$

可以计算出，变化前后，两者之比为3550与3800之比，为0.9342

所以性能实际上只提升了**6.58%**

十倍：

$$\frac{CPUtime}{cycletime} = (500 \times 0.1 + 300 \times 10 + 100 \times 3) \times 10^6 = 3.35 \times 10^9$$

两者的比值为3350与3800之比，结果为0.8816

所以性能实际上只提升了**11.84%**

2.41

假设MIPS ISA中包含了比例偏移的寻址方式，该寻址方式与 2.19节中给出的x86相应的寻址方式相同。如果要使用“比例变址取数指令”进一步减少实现练习题2.4中功能的汇编指令数量，请给出你的思路。

2.4中的指令为

```
sll $t0, $s0, 2          # $t0 = f * 4
add $t0, $s6, $t0        # $t0 = &A[f]
sll $t1, $s1, 2          # $t1 = g * 4
add $t1, $s7, $t1        # $t1 = &B[g]
lw $s0, 0($t0)           # f = A[f]
addi $t2, $t0, 4         #
lw $t0, 0($t2)           #
add $t0, $t0, $s0        #
sw $t0, 0($t1)           #
```

我们使用x86中的比例偏移寻址方式

我们使用x86的汇编语言，假设EDI存储着f，ESI中存储着数组A的基址

```
MOV EAX,ESI[4*EDI]
MOV EBX,ESI[4*EDI+4]
ADD EAX,EAX,EBX
```

这样就可以通过比例偏移的寻址方式来调用A[f]和A[f+1]

然后使用ADD语句，完成A[f]和A[f+1]的加法即可，极大的缩短了汇编指令的数量