基于C0文法的编译器V6.0

10011603 2016302731 冯宇

**一.主要功能**

对MIPS代码进行优化

1.添加addi,subi等指令，相较于原来的加减指令减少对内存的访问

2.对于程序开始的变量声明引起的$sp地址变化，进行累计后一次性变换，减少指令数

3.对于乘除法中的2的幂的数的运算，可以使用移位操作，指令的效率更高

**二.代码实现**

1.首先在void funcContent()中判断加减号两边是数字还是变量，然后进行不同处理，如果是数字，那么经过以下处理

insertCode("addi $t0,$zero," + toString(notation.operand1));

insertCode("add $t0,$t0,$t2");

storeThirdOperand(notation.operand3, "$t0");

通过addi指令减少对于内存的访问

2.在变量定义的MIPS转化函数void mipsVarDef(bool isGlobal)中设置一个变量cout ，记录地址变化的量，退出函数的时候统一进行地址的变换

3.添加函数

int log\_2(int value) //非递归判断一个数是2的多少次方

{

int x = 0;

while (value > 1)

{

value >>= 1;

x++;

}

return x;

}

在void muli\_insertcode(infixNotation notation)中对于乘数为2的幂的数使用sll指令，而不是mul指令

if (num & (num - 1)) { //operand1不是2的幂次方

insertCode("mul $t0,$t2," + toString(notation.operand1));

}

else{

i = log\_2(num);

insertCode("sll $t0,$t2," + intToString(i));

}

storeThirdOperand(notation.operand3, "$t0");

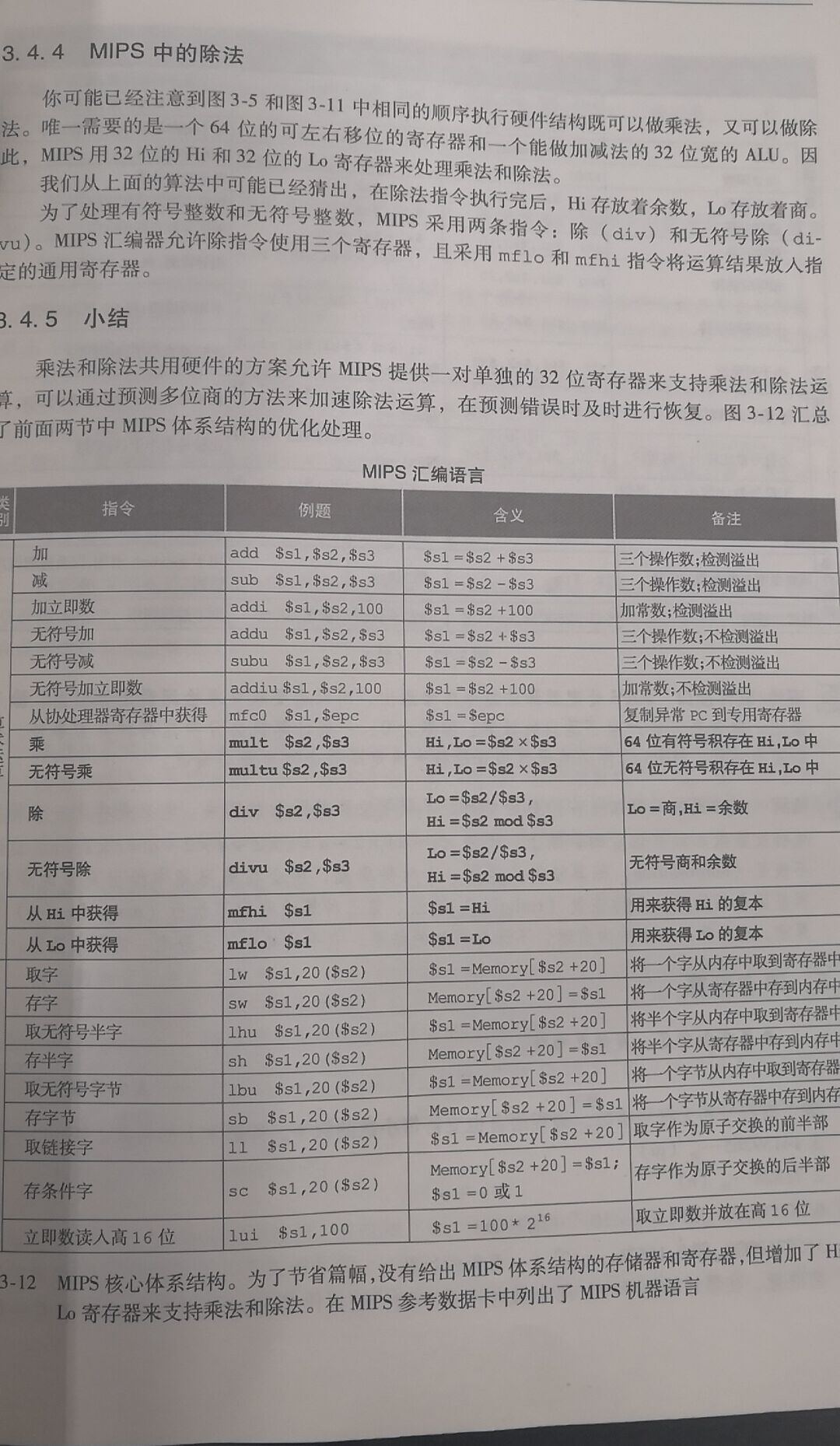
除法同理，如果是除数是2的幂次方，使用sra指令，而不是div指令

**三.测试结果**

|  |  |  |
| --- | --- | --- |
| Input code | 原MIPS代码 | 优化后的MIPS代码 |
| int main(){  int x,y,z;  x=1+6;  y=2;  z=3;  x=y+2;  x=x-20;  z=y\*4;  z=z/2;  return;  } | .data  .text  j main  main:  sw $fp, ($sp)  move $fp, $sp  sub $sp, $sp, 8  sub $sp, $sp, 4  sub $sp, $sp, 4  sub $sp, $sp, 4  sub $sp, $sp, 4  sub $sp, $sp, 4  sub $sp, $sp, 4  sub $sp, $sp, 4  li $t0, 2  sw $t0, -8($fp)  lw $t1, -8($fp)  li $t2, 4  mul $t0, $t1, $t2  sw $t0, -24($fp)  lw $t1, -24($fp)  li $t2, 2  div $t0, $t1, $t2  sw $t0, -32($fp)  lw $t1, -8($fp)  li $t2, 2  add $t0, $t1, $t2  sw $t0, -16($fp)  lw $t1, -16($fp)  li $t2, 20  sub $t0, $t1, $t2  sw $t0, -20($fp)  li $t0, 3  sw $t0, -28($fp)  li $t1, 1  li $t2, 6  add $t0, $t1, $t2  sw $t0, -12($fp)  move $v1, $t0  j mainTail  mainTail: | .data  .text  j main  main:  sw $fp, ($sp)  move $fp, $sp  sub $sp, $sp, 8  sub $sp, $sp, 28  li $t0, 2  sw $t0, -8($fp)  lw $t1, -8($fp)  sll $t0,$t1,2  sw $t0, -24($fp)  lw $t1, -24($fp)  sra $t0,$t1,1  sw $t0, -32($fp)  lw $t1, -8($fp)  addi $t0,$zero,2  add $t0,$t0,$t1  sw $t0, -16($fp)  lw $t1, -16($fp)  subi $t0,$t1,20  sw $t0, -20($fp)  li $t0, 3  sw $t0, -28($fp)  addi $t0,$zero,1  addi $t0,$zero,6  sw $t0, -12($fp)  move $v1, $t0  j mainTail  mainTail: |

**四. Debug以及C++知识点**

**1.优化的时候主要考虑的是添加addi，subi等带常数的指令，来减少对于内存的访问**



在Mars程序中查找得到结果如下：所以只有加减有针对常数添加新的指令，而乘除没有。

**2.乘除法优化**

在乘除法中，想通过使用效率更快的移位指令，来取代对于2的幂级数的操作。

算​术左移和逻辑左移一样都是右边补0：

比如 00101011​

算术左移一位:01010110

逻辑左移一位​:01010110

对于二进制的数值来说左移n位等于原来的数值乘以2的n次方

比如00011010十进制是26，左移两位后是011010​00转成十进制是104恰好是26的4倍。

ps：这种倍数关系只适用于左移后被舍弃的高位不含1的情况，否则会溢出。​

​2.算术右移，逻辑右移

逻辑右移很简单，只要将二进制数整体右移，左边补0即可

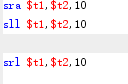
如10101101逻辑右移一位为01010110

算术右移符号位要一起移动，并且在左边补上符号位，也就是如果符号位是1就补1符号位是0就补0

比如：11100算术右移一位为11110（符号位1跟着一起移动并且左边补了1）

对于二进制的数值来说右移n位等于原来的数值除以2的n次方

比如10110100十进制是76（需要先将这个补码转换成原码之后再转换成十进制），右移两位后是11101101转成十进制是19恰好是76的4倍。

而我们需要的是算术左右移指令，即上两条指令。

**3.2的幂级数的快速判断**

我们可以通过(number & number - 1) == 0来快速判断number是否是2的幂级数

      原因：因为2的N次方换算是二进制为10……0这样的形式(0除外)。与上自己-1的位数，这们得到结果为0。例如。8的二进制为1000；8-1=7，7的二进制为111。两者相与的结果为0。比如：

        1000  
     & 0111  
        -------  
        0000