Lab5实验报告 实验要求 本次实验要求我们将一段cpp程序转化为汇编语言,具体cpp程序如下所示: int judge(int r0) { int i = 2; r1 = 1;while (i * i <= r0) { if (r0 % i == 0) { r1 = 0;break; i++; } return r1; } 具体要求的结构如下所示: .ORIG x3000 ...; TO BE DONE **HALT** JUDGE ...; TO BE DONE ...; TO BE DONE **RET** ...; TO BE DONE . END 实验设计思路 首先,我们采取先观察该段cpp程序的作用的方法,将cpp程序补全如下所示: #include <iostream> using namespace std; int judge(int r0) { int i = 2; int r1 = 1; while (i * i <= r0) { if (r0 % i == 0) { r1 = 0;break: i++; } return r1; } int main() { int r0, r1; cin>>r0; r1 = judge(r0);cout<<r1<<endl;</pre> } 在测试数个样例后结果如下所示: ≥ zsh cd "/Users/apple/Desktop/Projects/icsproject/lab5/" && g++ test.cpp -o test && "/Users/apple/Desk top/Projects/icsproject/lab5/"test Code Huff... apple@wjMacBook - ~/Desktop/Projects - cd "/Users/apple/Desktop/Projects/icsproject/lab5/" && g+ + test.cpp -o test && "/Users/apple/Desktop/Projects/icsproject/lab5/"test Code lab5 apple@wjMacBook - ~/Desktop/Projects/icsproject/lab5 cd "/Users/apple/Desktop/Projects/icsproj ect/lab5/" && g++ test.cpp -o test && "/Users/apple/Desktop/Projects/icsproject/lab5/"test ect/lab5/" && g++ test.cpp -o test && "/Users/apple/Desktop/Projects/icsproject/lab5/"test apple@wjMacBook -~/Desktop/Projects/icsproject/lab5 - cd "/Users/apple/Desktop/Projects/icsproj ect/lab5/" && g++ test.cpp -o test && "/Users/apple/Desktop/Projects/icsproject/lab5/"test 聚焦资源管理器中的文件夹 (Cmd + 单击) apple@wjMacBook ~/Desktop/Projects/icsproject/lab5 cd "/Users/apple/Desktop/Projects/icsproj ect/lab5/" && g++ test.cpp -o test && "/Users/apple/Desktop/Projects/icsproject/lab5/"test ect/lab5/" && g++ test.cpp -o test && "/Users/apple/Desktop/Projects/icsproject/lab5/"test 12 ect/lab5/" && g++ test.cpp -o test && "/Users/apple/Desktop/Projects/icsproject/lab5/"test apple@wjMacBook [] ~/Desktop/Projects/icsproject/lab5 [] 由此我们可以看出,这一程序的目的即为判断一个数是否为素数。若为素数,则R1为1,反之,R1为0。 在得知程序的目的后,就很容易得到相应的代码。我们在程序中调用judge函数,采用JSR指令,同时注意在函数内部要维护R7的值。 为了翻译程序,我们对cpp代码一句一句的分析。对于i*i来说,我们运用lab1中的计算乘法的方法即可。判断下列式子时 R0 % i == 0 我们采用,将RO的值反复减去i,如果为出现为O则正确,如果为正则继续循环相减,如果为负则不正确。 通过这一思路,我们最终得到的代码如下所示: .ORIG x3000 AND R1, R1, #0 AND R2, R2, #0 AND R7, R7, #0 JSR JUDGE HALT JUDGE ADD R2,R2,#2 ADD R1,R1,#1 AND R4,R4,#0 AND R3, R3, #0 ;R2*R2 => R3 star ADD R4,R2,#0 ADD R3, R2, #0 AND R5, R5, #0 ;乘法结果放在R3 mul ADD R5,R5,R3 ADD R4,R4,#-1 BRnp mul NOT R5, R5 ADD R5, R5, #1 ADD R5, R5, R0 BRn DONE ; if (R0%i == 0) R1 = 0; break; ;求R0%i **;**i存放在R2 AND R3, R3, #0 AND R4, R4, #0 ADD R5, R2, #0 ADD R6, R0, #0 NOT R5, R5 ADD R5, R5, #1 mod ADD R6, R6, R5 BRp mod BRn neg AND R1, R1, #0 BRz DONE neg ADD R2,R2,#1 BRnzp star DONE RET . END 接下来我们验证该程序的正确性。 首先我们利用LC3的评测网站,测试结果如下所示: lc3web 器中载入汇编代码或者机器码(在左侧的 Assemble 和 Raw 选项中载入),机器码需要 0011 头。 通过: 12 / 12 击评测即 通过: 2. 指令数: 22, 预期: 1, 你的: 1 台时条件 通过: 3. 指令数: 22, 预期: 1, 你的: 1 通过: 4. 指令数: 35, 预期: 0, 你的: 0 团目 通过: 7. 指令数: 55, 预期: 1, 你的: 1 通过: 456. 指令数: 487, 预期: 0, 你的: 0 通过: 993. 指令数: 1712, 预期: 0, 你的: 0 无输出长度 令数: 1 通过: 997. 指令数: 8145, 预期: 1, 你的: 1 逗号分割 通过: 1569. 指令数: 2672, 预期: 0, 你的: 0 56, 993, 通过: 9293. 指令数: 92956, 预期: 1, 你的: 1 化内存, 通过: 121. 指令数: 866, 预期: 0, 你的: 0 通过: 9339. 指令数: 15622, 预期: 0, 你的: 0 stcase) 通过: 1437. 指令数: 2452, 预期: 0, 你的: 0 Prime(n 平均指令数: 10420.5 $= 3) \{ ret$ 关闭 数: r[1]; -步验证我们程序的正确性。 下面分别为R0 = 1,R0 = 7,R0 = 19,R0 = 测评网站测试了12个数据,均通过。 之后我们再在LC3上进 20的结果: ivediareia R0 x0001 R1 x0001 x0002 R2 2 R3 x0002 x0000 R4 0 R5 -3 xFFFD x0000 R6 0 x0000 R7 0 **PSR** x8004 -32764CC: N 12315 PC x301B x0000 **MCR** 0 × Console (click to focus) - Halting the LC-3 ---Negisters x0007 R0 R1 x0001 1 Run x0003 3 x0003 R3 3 R4 x0000 0 R5 -2 ×FFFE R6 -1 ×FFFF R7 x0000 0 CC: N **PSR** x8004 -32764PC x301B 12315 x0000 0 MCR × Console (click to focus) Halting the LC-3 --ivediareia x0013 R0 19 R1 x0001 1 R2 x0005 5 R3 x0005 5 R4 x0000 0 xf'f'f'A R6 -1 ×FFFF R7 x0000 0 -32764 CC: N **PSR** x8004 x301B 12315 PC **MCR** x0000 × Console (click to focus) --- Halting the LC-3 ---R0 x0014 20 R1 x0000 0 R2 x0002 2 R3 x0000 0 R4 x0000 R5 -2 ×FFFE R6 x0000 0 R7 x0000 0 x8002 -32766 CC: Z **PSR** x301B PC 12315 MCR x0000 0 × Console (click to focus) --- Halting the LC-3 ---综上,该程序的正确性在LC3上得到了验证。 实验优化 对于该实验来说,并未要求对指令数的优化,但是对于该程序来说,根据LC3测评网站的数据,平均指令数达到了10000左右,对于一 个程序来说指令数较大,应该有较大的优化空间。 我们首先注意到该程序中运用到的乘法,我们可以采取lab1中的p版本,得到程序如下所示: •ORIG x3000 AND R1, R1, #0 AND R2, R2, #0 AND R7, R7, #0 JSR JUDGE HALT JUDGE ST R7, save7 ADD R2, R2, #2 ADD R1,R1,#1 AND R4, R4, #0 AND R3, R3, #0 ;R2*R2 => R3 star ADD R4,R2,#0 ADD R3, R2, #0 AND R5, R5, #0 ADD R6, R5, #1 AND R7, R7, #0 ;乘法结果放在R5 mul AND R5,R3,R6 BRz Skip ADD R7, R7, R4 ADD R4,R4,R4 Skip ADD R6, R6, R6 BRnp mul ADD R5, R7, #0 NOT R5, R5 ADD R5, R5, #1 ADD R5, R5, R0 BRn DONE ; if (R0%i == 0) R1 = 0; break; ;求R0%i **;**i存放在R2 AND R3, R3, #0 AND R4, R4, #0 ADD R5, R2, #0 ADD R6, R0, #0 NOT R5, R5 ADD R5, R5, #1 mod ADD R6, R6, R5 BRp mod BRn neg AND R1, R1, #0 BRz DONE neg ADD R2,R2,#1 BRnzp star DONE LD R7, save7 RET save7 .BLKW 1 . END 在经过这样的优化后,所需的指令数如下所示: 閰 无输出长度 通过: 12 / 12 通过: 2. 指令数: 102, 预期: 1, 你的: 1 通过: 3. 指令数: 102, 预期: 1, 你的: 1 通过: 4. 指令数: 115, 预期: 0, 你的: 0 通过: 7. 指令数: 211, 预期: 1, 你的: 1 stcase) 通过: 456. 指令数: 567, 预期: 0, 你的: 0 Prime(n 通过: 993. 指令数: 1868, 预期: 0, 你的: 0 = 3) { ret 通过: 997. 指令数: 9219, 预期: 1, 你的: 1 通过: 1569. 指令数: 2828, 预期: 0, 你的: 0 数: 通过: 9293. 指令数: 86978, 预期: 1, 你的: 1 通过: 121. 指令数: 1522, 预期: 0, 你的: 0 r[1]; 通过: 9339. 指令数: 15778, 预期: 0, 你的: 0 通过: 1437. 指令数: 2608, 预期: 0, 你的: 0 平均指令数: 10158.16666666666 设 预设 评测 关闭 我们通过这里可以看出,虽然乘法进行了一定的优化,但是最终所需的指令数的减少并不是非常的可观。 对于这份代码而言,如果想尽可能的减少指令数,可以先预处理一些特殊情况。首先我们知道,在0到8中,除了0以外,其余的奇数都 是素数。所以我们可以预处理一些特殊的情况。再者,如果这个数为偶数,对应了它二进制最低位为0,如果这个数并不是0或2,那它 一定是素数。 所以由此我们得到的优化代码如下所示: .ORIG x3000 AND R1, R1, #0 AND R2, R2, #0 AND R7, R7, #0 JSR JUDGE HALT JUDGE ST R7, save7 ADD R1,R1,#1 ADD R0, R0, #0 BRz DONE AND R4, R0, #1 BRz ou ADD R4, R0, #-9 BRn DONE BRzp keep ADD R4,R0,#-2 BRz DONE AND R1, R1, #0 BRnzp DONE keep AND R1,R1,#0 ADD R2, R2, #2 ADD R1,R1,#1 AND R4, R4, #0 AND R3, R3, #0 ;R2*R2 => R3 star ADD R4,R2,#0 ADD R3,R2,#0 AND R5, R5, #0 ADD R6, R5, #1 AND R7, R7, #0 ;乘法结果放在R5 mul AND R5,R3,R6 BRz Skip ADD R7, R7, R4 Skip ADD R4,R4,R4 ADD R6, R6, R6 BRnp mul ADD R5, R7, #0 NOT R5, R5 ADD R5, R5, #1 ADD R5, R5, R0 BRn DONE ;if(R0%i == 0) R1 = 0;break; **;**求R0%i **;**i存放在R2 AND R3, R3, #0 AND R4, R4, #0 ADD R5, R2, #0 ADD R6, R0, #0 NOT R5, R5 ADD R5, R5, #1 mod ADD R6,R6,R5 BRp mod BRn neg AND R1,R1,#0 BRz DONE neg ADD R2,R2,#1 BRnzp star DONE LD R7, save7 RET save7 **.**BLKW #1 . END 该部分代码优化出的指令数结果如下图所示: 谋模拟器中载入汇编代码或者机器码(在左侧的 Assemble 和 Raw 选项中载入),机器码需要 00 开头。 后点击评测即 通过: 12 / 12 测开始时条件 通过: 2. 指令数: 14, 预期: 1, 你的: 1

通过: 3. 指令数: 14, 预期: 1, 你的: 1 则试题目 通过: 4. 指令数: 16, 预期: 0, 你的: 0 通过: 7. 指令数: 14, 预期: 1, 你的: 1 通过: 456. 指令数: 16, 预期: 0, 你的: 0 无输出 2次指令数: 1 通过: 993. 指令数: 1877, 预期: 0, 你的: 0 7组(逗号分割 通过: 997. 指令数: 9228, 预期: 1, 你的: 1 1, 7, 456, 993, 通过: 1569. 指令数: 2837, 预期: 0, 你的: 0 通过: 9293. 指令数: 86987, 预期: 1, 你的: 1 划始化内存, 通过: 121. 指令数: 1531, 预期: 0, 你的: 0 on (testcase) 通过: 9339. 指令数: 15787, 预期: 0, 你的: 0 tion isPrime(n 通过: 1437. 指令数: 2617, 预期: 0, 你的: 0 num <= 3) { ret 平均指令数: 10078.16666666666 e { **茶函数**: 关闭 on () { n lc3.r[1]; lab 1 预设 lab 2 & 3 预设 lab 5 预设 lab 4 task 1预设 lab 4 task 2预设

本次实验主要是在功能上翻译一段cpp的代码。代码的实现比较简单,但是对于如何将该代码的指令数尽可能的减少比较麻烦。在判断

偶数时已经优化到了非常理想的地步,而在判断奇数时,对于较小的数或者较大的非素数时,算法效率较为客观,但是对于较大的素数

时,该算法的效率略微有一点低,经过分析主要原因可能是在于求平方时用的指令数过多,即使使用移位的快速幂算法时优化的程度仍

实验总结

然不是很好。