

3/8: 9. (1) JAL指令: 无条件跳转的J型指令, 跳转范围为 $PC \pm 1MB (-2^{20} \sim 2^{20}-1)$

(2) BNE指令: 有条件跳转的I型指令, 跳转范围为 $PC \pm 4KB (-2^{12} \sim 2^{12}-1)$.

(3) 可以, 先用LUI加载高20位的立即数, 后用JALR加载低12位跳转地址, 共同合成32位跳转地址

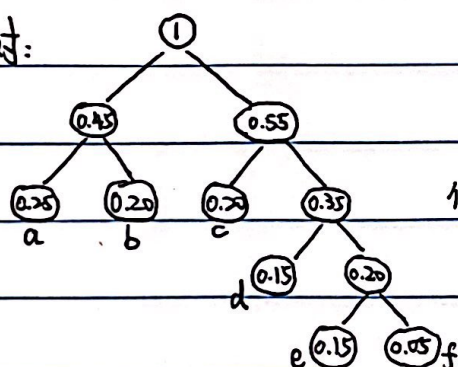
10. 能被压缩为16位RVC指令的条件: ① 立即数或地址偏移量是较小的值。

② 其中的一个寄存器为 $x_0 \sim x_7$ ③ 目的寄存器和第一个源寄存器是相同的。

④ 在指令中和过程中所用到的寄存器均为8个最常用的寄存器

RVC指令集中的CR, CI, CSS类型指令均可以使用完整的32个寄存器, 只是在编码过程中将寄存器编号压缩至5位, 但CIW, CL, CS, CA和CB指令只能用到其中的8个寄存器。

18. 霍夫曼树:



$$\text{平均长度: } \sum_{i=1}^6 p_i l_i = 0.65 \times 2 + 0.15 \times 3 + 0.2 \times 4 = 2.55$$

$$\text{信息冗余: } R = 1 - \frac{\sum_{i=1}^6 p_i \log_2 p_i}{2.55}$$

$$\approx 0.033$$

19. 栈溢出原理: 栈是一种数据结构, 而其大小是有限的, 而每次函数递归调用时系统均要在栈中不断保存函数调用时的现场和过程中产生的变量, 即在栈中加一层栈帧, 占用一部分栈的资源, 故当函数嵌套调用层数过多时, 使用的栈资源会超过栈可用资源的最大值, 造成栈溢出。

缓解/避免方法: ① 在程序开始时设置栈大小, 从而增加可用的栈的空间。

② 采用动态内存分配: 使用 malloc 和 free 等函数动态分配和释放内存。

③ 优化递归算法: 采用尾递归 (在函数返回时调用函数本身, 多次调用只会使用一个栈帧) 优化。

④ 改变算法: 用迭代、动态规划等算法代替递归算法。

⑤ 在编写程序时控制递归的深度。

20. 见下页图片。

20. 假设有三个函数：F1、F2 和 F3。其中 F1 包含 1 个输入参数，计算过程使用寄存器 t0 和 s0；F2 包含 2 个输入参数，计算过程使用寄存器 t0-t1 及 s0-s1，返回一个 int 值。F1 执行过程中会调用 F2，F2 执行过程中会调用 F3。下表模拟了 F1 执行过程中栈的内容，其中第一行为 F1 函数被首次调用时 sp 寄存器指向的位置。请在表中填入当 F2 函数首次调用 F3 前栈内保存的可能内容，并在每行的括号内标注该值是被哪个函数所保存的。第一行的内容已经给出。（可根据需要增删行数）

ra (F1)
a0(F1)
t0(F1)
s0(F1)
ra (F2)
a0(F2)
a1(F2)
t0(F2)
t1(F2)
s0(F2)
s1(F2)
ra(F3)