17.

```
addi a0, x0, 0    # a0=0
addi a1, x0, 1    # a1=1
addi a2, x0, 30   # a2=30
loop:
beq  a0, a2, done # a0≥a2, →done
slli a1, a1, 1    # a1=2a1
addi a0, a0, 1    # a0=a0+1
j    loop
done: #exit code
```

```
int S=1
for (int i=0; i<30; i++)
    S=2*S
```

功能：计算 2^{30}.

结果保存于 a1 寄存器中.

3.29 第6周 chapter two

9. (1) jal 指令包含20位的有符号立即数编码(J-type)，该指令相较当前PC可以跳转的地址空间范围是：$-2^{20} \sim (2^{20}-1)$ 即有1MB的范围.

(2) 条件转移指令(如bne)包含12位有符号立即数编码(B-type)，这类指令相较当前PC可以跳转的地址空间范围为：$-2^{11} \sim (2^{11}-1)$，即有4KB的范围.

(3) 是否可以使用一条lui指令和一条jalr指令的组合完成任意32位绝对地址的跳转操作？

答：是可以使用此方法实现的。首先，使用lui指令将需要跳转的32位绝对地址的高16位加载入寄存器中，再使用jalr指令跳转到该寄存器中存储的地址，从而实现32位任意绝对地址的跳转.

10. 调查什么是RVC压缩指令集编码，说明一条常用的32位指令能够被压缩为16位RVC指令的条件是什么？RVC中有类型的指令是否都能使用完整的32个通用整型寄存器？

答：RCV指令集是对ISCV的一个压缩指令集，可以将常用的32位指令压缩为16位指令。下面是一个常用的32位指令和对应的16位RVC指令：

32位指令 addi x1, x1, 1 → 16位指令RCV指令：addi4spn x1, 1

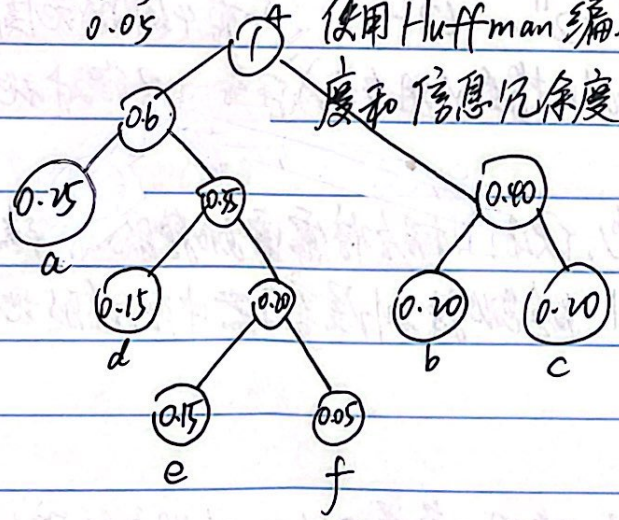
该指令可被压缩的条件为：① 该指令不能使用立即即条件转移指令

② 该指令的操作数必须是16位立即数或16位寄存器

③ 该指令不能使用CSR指令（即访问控制和状态寄存器的指令）。RVC中各类型的指令都可以使用完整型寄存器，这是因为RVC的指令格式中包含了寄存器编号，可以使用任意一个通用整型寄存器。但是在一个16位指令中，只能有一个寄存器编号，因此对于需要使用两个寄存器的指令，需要通过两条16位的RCV指令来实现。

18、有一组操作码，它们出现的几率如下图所示。

a_i	P_i	l_i
a	0.25	2
b	0.20	2
c	0.20	2
d	0.15	3
e	0.15	4
f	0.05	4



进行编码

使用Huffman编码对这组操作码，计算操作码的平均长度和信息冗余度。

操作码平均长度：

$$H=\sum_{i=1}^{6} P_i l_i = 2\times0.25+0.20\times2+0.20\times2+0.15\times3+0.15\times4+0.05\times4$$

$$=0.5+0.4+0.4+0.45+0.6+0.2$$

$$=2.55$$

信息冗余度 $R=1-\frac{\sum_{i=1}^{6} P_i l_i}{\lceil \log_2 6 \rceil}=1-\frac{2.55}{3}=0.15$.

19. (1) 当函数嵌套调用层数过多(例如递归陷入死循环时)，可能会造成栈溢出，请简述其原理；

(2) 有什么办法可以缓解或避免特定情况下的栈溢出问题。

答：(1) 函数嵌套调用时，每次调用都会将一些信息(如当前函数的返回地址、参数、局部变量等)保存在栈中，随着函数嵌套调用层数的增加，栈中存储的信息也会随之积累增加，当栈的空间不足以保存这么多信息时，就会发生栈溢出。栈溢出的原因主要是由于递归调用的层数过多，导致栈空间被耗尽，无法继续保存新的函数的调用信息，从而导致程序崩溃。(2) 方法：①尽量避免使用递归函数，可以使用循环体替代递归，或者使用尾递归优化，减少函数调用的层数。②增加栈的大小，可以通过修改操作系统配置文件或者编译器选项来增加栈的空间大小。③减少函数中局部变量的使用，可以使用全局变量or静态变量代替局部变量，减少栈空间的使用。④使用动态内存分配，将需要大量空间的数据存储在堆中，而不是栈中，避免栈空间被耗尽。⑤使用异常处理机制，当发生栈溢出时，可以使用异常处理机制来捕获异常，释放栈空间并进行相应处理，避免程序崩溃。

20. $F_1 \to F_2 \to F_3$ 三个函数嵌套

F_1函数含有一个输入参数，过程中使用寄存器 t0、S0

F_2 … 2个 … … t0、t1、S0、S1

F_3

在表中填入当F_2首次调用F_3前栈保存的内容

ra(F_1) 返回地址	上一个栈帧RBP/EBP(栈底)(F_2)
上一个栈帧RBP(栈底)(F_1)	t0 (F_2)
t0(F_1) 局部变量	t1 (F_2)
S0 (F_1) 局部变量	S0 (F_2)
参数 (F_1)	S1 (F_2)
ra(F_2) 返回地址	第二个参数 (F_2)
	第一个参数(F_2)