第五次作业

T1

> What is the problem with using the string AND as a label?

与 AND 指令重名，会引发冲突

T2

> What is the purpose of the .END pseudo-op? How does it differ from the HALT instruction?

.END 仅仅是一个分隔符，表示源程序结束了，出现在 .END 之后的任何字符都会被汇编器丢弃；HALT 是程序结束指令，程序运行到 HALT 指令就停止，而 .END 并不会停止程序的执行，事实上最后的程序中不会出现 .END

T3

> Whether multiple .END pseudo-ops can exist in one file? And please explain your answer.

可能存在，但没有意义，因为出现在 .END 之后的任何字符都会被汇编器丢弃，包括后面的 .END

T4

> The following program fragment has an error in it.
>
> ```
> ADD R3, R3, #30
> ST R3, A
> HALT
> A .BLKW 1
> ```
>
> (a) Identify the error and explain how to fix it.
>
> (b) Will this error be detected when this code is assembled or when this code is run on the LC-3?

(a)

ADD 指令立即数只有 5 位，表示不到 30；

(b)

在汇编时就会报错

T5

> Suppose you write two separate assembly language modules that you expect to be combined by the linker. Each module uses the label AGAIN , and neither module contains the pseudo-op .EXTERNAL AGAIN .
>
> Is there a problem using the label AGAIN in both modules? Why or why not?

不会有问题，多个独立的代码区段分别独立汇编，创建独立的符号表。

T6

> When the following LC-3 program is executed,
>
> ```
> .ORIG x3005
> LEA R2, DATA
> LDR R4, R2, #0
> LOOP ADD R4, R4, #-3
> BRzp LOOP
> TRAP x25
> DATA .FILL x8002
> .END
> ```
>
> a) how many times will the instruction at the memory address labeled LOOP execute?
>
> b) Now if we replace the instruction in x300A with DATA .FILL x8003 , will your answer be the same?

(a)

R2 = x300a；R4 = x8002 = 1000_0000_0000_0010

R4 = R4 - 3 = 2^15 - 1 = 32767

所以会执行 $1 + \lfloor \frac{32767}{3} \rfloor + 1 = 10924$ 次

(b)

结果一样。

T7

> Fill in the missing blanks so that the subroutine below implements a stack multiply. That is it pops the top two elements off the stack, multiplies them, and pushes the result back on the stack. You can assume that the two numbers will be non-negative integers (greater than or equal to zero) and that their product will not produce an overflow

```
MUL ______________
ST R0, SAVER0
ST R1, SAVER1
ST R2, SAVER2
ST R5, SAVER5
AND R2, R2, #0
JSR POP
ADD R1, R0, #0
JSR POP
ADD R1, R1, #0
BRz DONE    ;ans2
AGAIN ADD R2, R2, R0
ADD R1, R1, #-1     ;ans3
BRp AGAIN
DONE ADD R0, R2, #0
JSR PUSH
______________
LD R0, SAVER0
LD R1, SAVER1
LD R2, SAVER2
LD R5, SAVER5
RET
```

T8

> Figure 8.18 in P286 describes a FIB subroutine as shown below:
>
> ```
> ;FIB subroutine
> ; + FIB(0) = 0
> ; + FIB(1) = 1
> ; + FIB(n) = FIB(n-1) + FIB(n-1)
> ;
> ; Input is in R0
> ; Return answer in R1
> ```

```
;
FIB ADD R6, R6, #-1
STR R7, R6, #0 ; Push R7, the return linkage
ADD R6, R6, #-1
STR R0, R6, #0 ; Push R0, the value of n
ADD R6, R6, #-1
STR R2, R6, #0 ; Push R2, which is needed in the subroutine
; Check for base case
AND R2, R0, #-2
BRnp SKIP ; Z=0 if R0=0,1
ADD R1, R0, #0 ; R0 is the answer
BRnzp DONE
; Not a base case, do the recursion
SKIP ADD R0, R0, #-1
JSR FIB ; R1 = FIB(n-1)
ADD R2, R1, #0 ; Move result before calling FIB again
ADD R0, R0, #-1
JSR FIB ; R1 = FIB(n-2)
ADD R1, R2, R1 ; R1 = FIB(n-1) + FIB(n-2)
; Restore registers and return
DONE LDR R2, R6, #0
ADD R6, R6, #1
LDR R0, R6, #0
ADD R6, R6, #1
LDR R7, R6, #0
ADD R6, R6, #1
RET
```

a) Is R0 caller save or callee save? What about R2 and R7 ?

b) The following table shows the instruction cycles used to execute this subroutine for corresponding input n:

n	2	3	4	5	6	7	8	9	10
$f(n)$	55	93	169	283	473	777	1271	2069	3361

Can you define recursively(递归地) and explain your definition?

c) How can you improve the efficiency of this recursive subroutine? Just describe your idea briefly.

(a)

R0 R2 R7 都为 callee save，因为在子程序中已经备份了 R0 的值，并在最后重新赋回；

(b)

$f(n) = f(n-1) + f(n-2) + 21$

递归调用的指令数，加上剩下的 21 条指令

(c)

将递归改为循环，从下往上计算，$f(n-2) + f(n-1) = f(n)$，计算 n - 1 次(不包含特例 n = 0)；

T9

> Rewrite the PUSH and POP routines such that the stack on which they operate holds elements that take up two memory locations each. Assume we are writing a program to simulate a stack machine that manipulates 32-bit integers with the LC-3. We would need PUSH and POP routines that operate with a stack that holds elements that take up two memory locations each. Rewrite the PUSH and POP routines for this to be possible.

```
PUSH ADD R6, R6, #-2
STR R0, R6, #0
STR R1, R6, #1

POP LDR R0, R6, #0
LDR R1, R6, #1
ADD R6, R6, #2
```

T10

> The input stream of a stack is a list of all the elements we pushed onto the stack, in the order that we pushed them. The output stream is a list of all the elements that are popped off the stack in the order that they are popped off.
>
> a) If the input stream is ABCD, create a sequence of pushes and pops such that the output stream is BDCA.
>
> b) If the input stream is ABCD, is it posible to create a sequence of pushes and pops such that the output stream is DBCA? Why?
>
> c) If the input stream is ABCDE, how many different output streams can be created? Only consider output streams that are 5 characters long?

(a)

$push\ A \to push\ B \to pop\ B \to push\ C \to push\ D \to pop\ D \to pop\ C \to pop\ A$

(b)

不可能，pop D 时 B，C 都在栈内，且 C 在上层，所以一定是 C 比 B 先 pop

(c)

考虑出栈时 A 的位置。

若 A 在第一个，则剩下四个进出栈为子问题，即 $f(4)$,

若 A 在第二个，则有一个(B)比 A 先出栈，为子问题 $f(1)$，剩下三个，即 $f(3)$,

若 A 在第三个，则有两个(BC)比 A 先出栈，为子问题 $f(2)$，剩下两个，即 $f(2)$,

。。。

所以递推公式为:

$$f(n) = \sum_{i=0}^{n-1} f(i) * f(n-i-1) = \frac{C_{2n}^{n}}{n+1}$$

所以 $f(5) = 42$

T11

不会