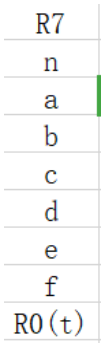
题目：**HUMANOID COMPILER**

```
typedef int i16;
typedef unsigned int u16;
i16 func(i16 n, i16 a, i16 b, i16 c, i16 d, i16 e, i16 f){ //Lots of arguments
 i16 t = GETC() - '0' + a + b + c + d + e + f;
 if(n > 1){
 i16 x = func(n - 1, a, b, c, d, e, f);
 i16 y = func(n - 2, a ,b, c, d, e, f);
 return x + y + t - 1;
 }else{
 return t;
 } }
i16 main(void){
 i16 n = GETC() - '0';
 return func(n, 0, 0, 0, 0, 0, 0);
}
_Noreturn void __start(){
 /*
 Here is where this program actually starts executing.
 Complete this function to do some initialization in your compiled assembly.
 TODO: Set up C runtime.
 */
 u16 __R0 = main(); //The return value of function main() should be moved to R0.
 HALT();
}
```

1 思路讲解

因为要严格翻译C代码，只能使用栈来实现递归（而非BR指令）。每调用func()前，先把t、f-a、n按顺序压栈，进入func之后，立刻把R7压栈（RET前取出）。

如图：



按照原程序，对n进行判断，要么继续递归，要么释放子程序的栈空间后存入返回值。

OVERFLOW和EMPTY是上溢出和下溢出的判断。

要求中提到，程序会被随机地放在x3000到xC000的地方，所以我把栈设置在xE000~xEFFF (4096个空间)。

2 具体实现

2.1 初始化过程

由于要求中提到，

```
At first the registers and memory are in random status. So, you should design
and complete the initialization process yourself.
```

写出这样一个模块_START

```
.ORIG x3000
_START  LD  R6, BASE
        JSR MAIN
        HALT
```

2.2 错误处理

检查上溢出、下溢出。

```
EMPTY    LD  R5, BASE
         NOT R5, R5
         ADD R5, R5, #1
         ADD R5, R5, R6
         BRn END
         LD  R0, EM
         PUTS
         HALT
OVERFLOW     LD  R5, TOP
             NOT R5, R5
             ADD R5, R5, #1
             ADD R5, R5, R6
             BRp END
             LD  R0, OVER
             PUTS
             HALT

BASE .FILL xF000 ; assume the stack is over there
TOP  .FILL xE000 ; the highest
EM   .STRINGZ "stack empty"
OVER .STRINGZ "stack overflow"
```

栈指针R6下移则检查EMPTY，上移检查OVERFLOW

2.3 调用、传参、callee保存、设计标准

MAIN压栈后JSR FUNC

```
MAIN     ADD R6, R6, #-1
         STR R7, R6, #0
         ADD R6, R6, #-1
         JSR OVERFLOW
         STR R0, R6, #0 ; push R0
         AND R0, R0, #0 ; clean R0
         AND R2, R2, #0
         ADD R2, R2, #6
L0       ADD R6, R6, #-1
         JSR OVERFLOW
         STR R0, R6, #0
         ADD R2, R2, #-1
         BRp L0
```

```
        GETC
        LD  R2, ASC
        ADD R0, R0, R2
        ADD R6, R6, #-1
        JSR OVERFLOW
        STR R0, R6, #0
        JSR FUNC
        LDR R0, R6, #0
        ADD R6, R6, #1
        JSR EMPTY
        LDR R7, R6, #0
        RET
```

FUNC立刻把R7压栈

```
FUNC    ADD R6, R6, #-1
        STR R7, R6, #0  ; push R7
        JSR OVERFLOW
```

计算t，判断n

```
        GETC
        LD  R1, ASC    ;-'0'
        ADD R0, R0, R1
        LDR R1, R6, #2 ; a
        ADD R0, R0, R1
        LDR R1, R6, #3 ; b
        ADD R0, R0, R1
        LDR R1, R6, #4 ; c
        ADD R0, R0, R1
        LDR R1, R6, #5 ; d
        ADD R0, R0, R1
        LDR R1, R6, #6 ; e
        ADD R0, R0, R1
        LDR R1, R6, #7 ; f
        ADD R0, R0, R1
        LDR R1, R6, #1 ; n
        ADD R1, R1, #-1 ; n-1
        BRnz FINAL

ASC     .FILL x-30  ; -'0'
```

n<=1 直接跳转到FINAL：callee处理R0，R7的存入和取出

```
FINAL   ADD R6, R6, #8 ; point to R0 now
        JSR EMPTY
        LDR R7, R6, #-8 ; R7 back
        LDR R2, R6, #0 ; save the return R0
        STR R0, R6, #0 ; push t
        ADD R0, R2, #0 ; R0 back
END     RET
```

n>1 递归开始（后面也接FINAL）

```
        ADD R6, R6, #-1
```

```
        JSR OVERFLOW
        STR R0, R6, #0 ; push R0 = t
        AND R2, R2, #0 ; counter
        ADD R2, R2, #6 ; abcdef
L1      ADD R6, R6, #-1
        JSR OVERFLOW
        LDR R0, R6, #9 ; f-e-d-c-b-a
        STR R0, R6, #0
        ADD R2, R2, #-1
        BRp L1
        ADD R6, R6, #-1
        JSR OVERFLOW
        STR R1, R6, #0 ; push n-1
        JSR FUNC ; x

        LDR R1, R6, #2  ; n back caller
        ADD R1, R1, #-2 ; n-2
        ADD R6, R6, #-1
        JSR OVERFLOW
        STR R0, R6, #0 ; push R0 = t  callee
        AND R2, R2, #0 ; counter
        ADD R2, R2, #6 ; abcdef
L2      ADD R6, R6, #-1
        JSR OVERFLOW
        LDR R0, R6, #10 ; f-e-d-c-b-a
        STR R0, R6, #0
        ADD R2, R2, #-1
        BRp L2
        ADD R6, R6, #-1
        JSR OVERFLOW
        STR R1, R6, #0 ; n-2
        JSR FUNC ; y
        ADD R0, R0, #-1 ; t-1
        LDR R2, R6, #0
        ADD R0, R0, R2  ; t+y-1
        ADD R6, R6, #1
        JSR EMPTY
        LDR R2, R6, #0
        ADD R0, R0, R2  ; t+x+y-1
        ADD R6, R6, #1
        JSR EMPTY
FINAL   ADD R6, R6, #8 ; point to R0 now
        JSR EMPTY
        LDR R7, R6, #-8 ; R7 back
        LDR R2, R6, #0 ; save the return R0
        STR R0, R6, #0 ; push t
        ADD R0, R2, #0 ; R0 back
END     RET
```

设计递归考虑到（n-1）的递归，R6及以下存储的是R7到n, a~f到R0；而（n-2）时，R6存储的 x, 所以取R7等的位移相比（n-1）的递归要+1。

3 完整代码

```
.ORIG x3000
_START  LD  R6, BASE
```

```
        JSR MAIN
        HALT

MAIN    ADD R6, R6, #-1
        STR R7, R6, #0
        ADD R6, R6, #-1
        JSR OVERFLOW
        STR R0, R6, #0 ; push R0
        AND R0, R0, #0 ; clean R0
        AND R2, R2, #0
        ADD R2, R2, #6
L0      ADD R6, R6, #-1
        JSR OVERFLOW
        STR R0, R6, #0
        ADD R2, R2, #-1
        BRp L0
        GETC
        LD  R2, ASC
        ADD R0, R0, R2
        ADD R6, R6, #-1
        JSR OVERFLOW
        STR R0, R6, #0
        JSR FUNC
        LDR R0, R6, #0
        ADD R6, R6, #1
        JSR EMPTY
        LDR R7, R6, #0
        RET

EMPTY   LD  R5, BASE
        NOT R5, R5
        ADD R5, R5, #1
        ADD R5, R5, R6
        BRn END
        LD  R0, EM
        PUTS
        HALT
OVERFLOW    LD  R5, TOP
            NOT R5, R5
            ADD R5, R5, #1
            ADD R5, R5, R6
            BRp END
            LD  R0, OVER
            PUTS
            HALT

BASE .FILL xF000 ; assume the stack is over there
TOP  .FILL xE000 ; the highest
EM   .STRINGZ "stack empty"
OVER .STRINGZ "stack overflow"


; stack save--R7 R0
; changing--R6(SP)
; save up to down: R7,n,a,b,c,d,e,f,R0

; R0 use as t == should be stored
; R1 changing at first
```

```
; R6 stack pointers
; R7 RET == should be stored
FUNC    ADD R6, R6, #-1
        STR R7, R6, #0  ; push R7
        JSR OVERFLOW
        GETC
        LD  R1, ASC    ;-'0'
        ADD R0, R0, R1
        LDR R1, R6, #2 ; a
        ADD R0, R0, R1
        LDR R1, R6, #3 ; b
        ADD R0, R0, R1
        LDR R1, R6, #4 ; c
        ADD R0, R0, R1
        LDR R1, R6, #5 ; d
        ADD R0, R0, R1
        LDR R1, R6, #6 ; e
        ADD R0, R0, R1
        LDR R1, R6, #7 ; f
        ADD R0, R0, R1
        LDR R1, R6, #1 ; n
        ADD R1, R1, #-1 ; n-1
        BRnz FINAL

        ADD R6, R6, #-1
        JSR OVERFLOW
        STR R0, R6, #0 ; push R0 = t
        AND R2, R2, #0 ; counter
        ADD R2, R2, #6 ; abcdef
L1      ADD R6, R6, #-1
        JSR OVERFLOW
        LDR R0, R6, #9 ; f-e-d-c-b-a
        STR R0, R6, #0
        ADD R2, R2, #-1
        BRp L1
        ADD R6, R6, #-1
        JSR OVERFLOW
        STR R1, R6, #0 ; push n-1
        JSR FUNC ; x

        LDR R1, R6, #2  ; n back caller
        ADD R1, R1, #-2 ; n-2
        ADD R6, R6, #-1
        JSR OVERFLOW
        STR R0, R6, #0 ; push R0 = t  callee
        AND R2, R2, #0 ; counter
        ADD R2, R2, #6 ; abcdef
L2      ADD R6, R6, #-1
        JSR OVERFLOW
        LDR R0, R6, #10 ; f-e-d-c-b-a
        STR R0, R6, #0
        ADD R2, R2, #-1
        BRp L2
        ADD R6, R6, #-1
        JSR OVERFLOW
        STR R1, R6, #0 ; n-2
        JSR FUNC ; y
        ADD R0, R0, #-1 ; t-1
```

```
        LDR R2, R6, #0
        ADD R0, R0, R2  ; t+y-1
        ADD R6, R6, #1
        JSR EMPTY
        LDR R2, R6, #0
        ADD R0, R0, R2  ; t+x+y-1
        ADD R6, R6, #1
        JSR EMPTY

FINAL   ADD R6, R6, #8 ; point to R0 now
        JSR EMPTY
        LDR R7, R6, #-8 ; R7 back
        LDR R2, R6, #0 ; save the return R0
        STR R0, R6, #0 ; push t
        ADD R0, R2, #0 ; R0 back
END     RET

ASC     .FILL x-30  ; -'0'

.END
```