```
                        r0          r1       r2        r3       Mem[sp]
                  Mem[fp-8]  Mem[fp-12] Mem[fp-16] Mem[fp-20]    Mem[fp+8]
void int32_t f (int32_t n, int32_t *t, int *min, int *max, int *somme) {
   int32_t s;         // Mem[fp - 4]
   register int i;    // r7
   ...

   s = plus (s,3);
   ...
   t[i]=5;
   ...
}

   str  fp,[sp,#-8]
   str  lr,[sp,#-4]
   sub  fp,sp,#8
   sub  sp,sp,#TAILLE_LOCAUX
                               @ Mem[fp-4] : var locale s
   str  r0,[fp,#-8]            @ r0 : n
   str  r1,[fp,#-12]           @ r1 : t
@   str  r2,[fp,#-16]           @ r2 : min  si autres appels à 4 args
@   str  r3,[fp,#-20]           @ r3 : max
   str  ...                    @ au moins r6, r8, r9


   ldr  r0,[fp,#-4]            @ x_de_plus = s
   mov  r1,#3
   bl   plus
   str  r0,[fp,#-4]            @ s = plus (s,3)

   ldr  r6,[fp,#-12]           @ tmp_r6 = t (qui n'est plus dans r1)
   mov  r8,r7,LSL #2           @ r8 = i*sizeof(int32_t)
   mov  r9,#5
   str  r9,[r6,r8]             @ t[i] = 5

   ldr  r0[fp,#-4]             @ r0 = s
   ldr  r8,[fp,#8]             @ tmp_r8=somme
   str  r0,[r8]                @ *somme =s

   sub  sp,sp,#TAILLE_LOCAUX   @ restauration
   ldr  ... [fp,#-...]         @ des registres
   ldr  fp,[sp,#-8]
   ldr  lr,[sp,#-4]            @ ou directement ldr pc,[sp,#-4]
   bx   lr                     @ ou mov pc,lr


Appel dans main :

   mov  r0,#4
   ldr  r1,= tab
   ldr  r2,= mini
   ldr  r3,=maxi

   ldr  r7,=sigma              @ empiler somme=sigma
   sub  sp,sp,#4
   str  r7,[sp]
   bl   f
   add  sp,sp,#4               @ liberer place argument somme
```