MacCarthy

February 11, 2023

1 Fonction F91 de Mac Carthy

Un exercice du partiel de Mars 2022 d'INF401 portait sur la fonction F91 de Mac Carthy

```
Fonction f91(n : entier) avec résultat entier
loc : entier
1: si n > 100 alors
2: loc = n-10
3: sinon
4: loc = f91(n+11)
5: loc = f91(loc)
6: finsi
7: retourne loc

Programme principal
10: EcrChaine("Entrer un nombre")
11: Lire32(&x)
12: y:=f91(x)
13: EcrNdecimal32(y)
```

L'objectif était de traduire ce code en un programme en assembleur ARM.

Une ébauche était proposée :

```
.data
m: .asciz "Entrer un nombre"
.bss
x: .word
y: .word
.text
.global main
main:
push {lr}
@ partie à compléter
pop {lr}
bx lr
ptr_m: .word m
ptr_x: .word x
ptr_y: .word y
```

Pour la suite, nous utiliserons cette ébauche pour écrire et tester nos programmes.

[1]:
```
%%writefile macCarthy.se
.data
m: .asciz "Entrer un nombre"
.balign 4
x: .word 1948 @initialisation presque aléatoire
y: .word 2048 @initialisation presque aléatoire
.text
.global main
main:
push {lr}
@Prog Principal
#include "macCarthy10-11.se"
#include "macCarthy12.se"
#include "macCarthy13.se"
mov r0,#0
pop {lr}
bx lr
@Fonction 91 de MacCarthy
#include "macCarthyF91.se"
ptr_m: .word m
ptr_x: .word x
ptr_y: .word y
```

```
Writing macCarthy.se
```

Et par défaut on crée des fichiers (vides) pour la suite.

[2]:
```
!touch macCarthy10-11.se
!touch macCarthy12.se
!touch macCarthy13.se
!touch macCarthyF91.se
```

1.1 Questions

1.1.1 Traduire la saisie et l'affichage du programme principal : les lignes 10, 11 et 13

[3]:
```
%%writefile macCarthy10-11.se
@10: EcrChaine("Entrer un nombre")
    ldr R1, ptr_m
    bl EcrChaine
@11: Lire32(&x)
    ldr r1, ptr_x
    bl Lire32
```

```
Overwriting macCarthy10-11.se
```

[4]:
```
%%writefile macCarthy13.se
@13: EcrNdecimal32(y)
  ldr r1, ptr_y
  ldr r1, [r1]
  bl EcrNdecimal32
```

```
Overwriting macCarthy13.se
```

Pour vérifier la syntaxe, et un peu l'exécution, on peut demander au compilateur.

[5]:
```
!arm-linux-gnueabi-cpp macCarthy.se -o macCarthy.s
!arm-linux-gnueabi-gcc -static macCarthy.s es.s -o macCarthy.e
!echo 4567 | qemu-arm  macCarthy.e
```

```
Entrer un nombre
2048
```

La valeur affichée (2048) est la valeur d'initialisation mise pour y. Pour améliorer le test, il faudrait pouvoir faire et afficher la saisie. Cela n'est pas prévu dans le programme, mais si on ajoute une affectation y=x, ce sera déjà mieux.

[6]:
```
%%writefile macCarthy12.se
@12(pour tester, code non définitif): y<-x
  ldr r0, ptr_x
  ldr r1, [r0]
  ldr r0, ptr_y
  str r1, [r0]
```

```
Overwriting macCarthy12.se
```

On peut maintenant tester un peu mieux :

[7]:
```
!arm-linux-gnueabi-cpp macCarthy.se -o macCarthy.s
!arm-linux-gnueabi-gcc -static macCarthy.s es.s -o macCarthy.e
!echo 4567 | qemu-arm  macCarthy.e
```

```
Entrer un nombre
4567
```

Le nombre en sortie est maintenant égal au nombre en entrée, c'est ok (c'est juste ce que l'on voulait)

1.1.2 Traduire l'appel à la fonction F91 du programme principal : la ligne 12

[8]:
```
%%writefile macCarthy12.se
@12 y:=f91(x)
  ldr r1, ptr_x
  ldr r1,[r1]
  push {r1}
```

```
 mov r0, #42 @valeur presque aléatoire pour un pseudo résultat
 push {r0} @ place pour le résultat, on peut aussi juste déplacer sp (sub sp,␣
↪sp, #4)
 bl f91
 pop {r0}
 pop {r1} @ récupération place paramètre, on peut aussi juste déplacer sp (add␣
↪sp, sp, #4)
 ldr r1,ptr_y
 str r0, [r1]
```

```
Overwriting macCarthy12.se
```

Si on lance maintenant la compilation, comme la fonction f91 n'est pas définie, cela ne peut pas marcher.

Mais on peut faire une mini fonction f91.

[9]:
```
%%writefile macCarthyF91.se
f91:
    bx lr
```

```
Overwriting macCarthyF91.se
```

Maintenant on peut tester la syntaxe.

[10]:
```
!arm-linux-gnueabi-cpp macCarthy.se -o macCarthy.s
!arm-linux-gnueabi-gcc -static macCarthy.s es.s -o macCarthy.e
!echo 4567 | qemu-arm  macCarthy.e
```

```
Entrer un nombre
42
```

Le nombre en sortie est la valeur par défaut mise pour le résultat de la fonction (et comme il n'y a pas encore de fonction, c'est ce nombre qui reste)

Pour la suite, on utilisera l'ébauche suivant pour la fonction

[11]:
```
%%writefile macCarthyF91.se
f91:
#include "macCarthyF91Prologue.se"
#include "macCarthyF91Corps.se"
#include "macCarthyF91Epilogue.se"
    bx lr
```

```
Overwriting macCarthyF91.se
```

Avec quelques fichiers (vides) pour éviter les bugs.

[12]:
```
!touch macCarthyF91Prologue.se
!touch macCarthyF91Corps.se
!touch macCarthyF91Epilogue.se
```

```
!touch macCarthy4-5.se
```

1.2 Fonction 91 de mac Carthy

1.2.1 Prologue, ligne 7 et épilogue

[13]:
```
%%writefile macCarthyF91Prologue.se
prologue:
  push {lr}
  push {fp}
  mov fp,sp
  mov r0, #65  @valeur presque aléatoire pour loc
  push {r0} @pour loc, on peut aussi juste déplacer sp (sub sp, sp, #4)
  push {r0}
  push {r1}
  push {r2}
```

```
Overwriting macCarthyF91Prologue.se
```

[14]:
```
%%writefile macCarthyF91Epilogue.se
l.7:   @ retourne loc
  ldr r0,[fp,#-4]
  str r0,[fp,#8]
epilogue:
  pop {r2}
  pop {r1}
  pop {r0}
  add sp, sp, #4
  pop {fp}
  pop {lr}
  bx lr
```

```
Overwriting macCarthyF91Epilogue.se
```

On peut tester.

[15]:
```
!arm-linux-gnueabi-cpp macCarthy.se -o macCarthy.s
!arm-linux-gnueabi-gcc -static macCarthy.s es.s -o macCarthy.e
!echo 4567 | qemu-arm  macCarthy.e
```

```
Entrer un nombre
65
```

Cela donne la valeur initiale (presque aléatoire) attribuée à loc, c'est normal, notre fonction est vide.

1.2.2 Traduire un calcul et une affectation de F91 : la ligne 2

[16]:
```
%%writefile macCarthy2.se
@2: loc = n-10
  ldr r1,[fp,#12]
  sub r1,r1,#10
  str r1,[fp,#-4]
```

```
Writing macCarthy2.se
```

Pour tester, il faut au moins prévoir de mettre cette ligne dans le corps de F91.

[17]:
```
%%writefile macCarthyF91Corps.se
#include "macCarthy2.se"
```

```
Overwriting macCarthyF91Corps.se
```

Si on teste, pour comprendre le résultat, il faut se rappeler que le calcul se fait sur une variable locale, mais que la conditionnelle n'est pas encore traduite (donc que l'entrée soit >100 ou pas le résultat sera -10), dans tous les cas on vérifiera que l'on n'a pas introduit d'erreur.

[18]:
```
!arm-linux-gnueabi-cpp macCarthy.se -o macCarthy.s
!arm-linux-gnueabi-gcc -static macCarthy.s es.s -o macCarthy.e
!echo 4567 | qemu-arm  macCarthy.e
```

```
Entrer un nombre
4557
```

1.2.3 Traduire la terminaison de F91 : les lignes 1, 3 et 6

[19]:
```
%%writefile macCarthyF91Corps.se
corps:
@1: si n > 100 alors
  ldr r1,[fp,#12]
  cmp r1,#100
  ble l.4
#include "macCarthy2.se"
@3: sinon
  b l.6
l.4:
#include "macCarthy4-5.se"
l.6: @ finsi
  nop
```

```
Overwriting macCarthyF91Corps.se
```

On peut maintenant faire une exécution avec la conditionnelle (même si la fonction n'est pas encore complètement traduite)

[20]:
```
!arm-linux-gnueabi-cpp macCarthy.se -o macCarthy.s
!arm-linux-gnueabi-gcc -static macCarthy.s es.s -o macCarthy.e
!echo 4567 | qemu-arm  macCarthy.e
```

```
Entrer un nombre
4557
```

[21]:
```
!echo 15 | qemu-arm  macCarthy.e
```

```
Entrer un nombre
65
```

Si le nombre est plus grand que 100 on lui enlève 10, sinon on ne fait rien (au départ il y a une valeur par défaut pour loc), ce sont bien les résultats observés. Tout va bien.

On peut maintenant traduire la dernière partie de F91.

1.2.4 Traduire les appels récursifs de F91 : Lignes 4 et 5

[22]:
```
%%writefile macCarthy4-5.se
@4:loc = f91(n+11)
  ldr r0,[fp,#12]
  add r0, r0, #11
  push {r0}
  push {r0}  @ place pour le résultat, on peut aussi juste déplacer sp (sub sp,␣
 ↪sp, #4)
  bl f91
  pop {r0}
  str r0,[fp,#-4]
  pop {r0}  @ récupère la place argument, on peut aussi juste déplacer sp (ou␣
 ↪ne rien faire et ne pas faire les 2 instructions suivantes)
@5:loc = f91(loc)
  ldr r0,[fp,#-4]
  push {r0}
  push {r0}  @ place pour le résultat, on peut aussi juste déplacer sp (sub sp,␣
 ↪sp, #4)
  bl f91
  pop {r0}
  str r0,[fp,#-4]
  pop {r0}  @ récupère la place argument, on peut aussi juste déplacer sp (add␣
 ↪sp, sp, #4)
```

```
Overwriting macCarthy4-5.se
```

Et maintenant les tests !

[23]:
```
!arm-linux-gnueabi-cpp macCarthy.se -o macCarthy.s
!arm-linux-gnueabi-gcc -static macCarthy.s es.s -o macCarthy.e
```

```
!echo 1 | qemu-arm  macCarthy.e
```

```
Entrer un nombre
91
```

[24]:
```
!echo 10 | qemu-arm  macCarthy.e
```

```
Entrer un nombre
91
```

[25]:
```
!echo 100 | qemu-arm  macCarthy.e
```

```
Entrer un nombre
91
```

[26]:
```
!echo 1000 | qemu-arm  macCarthy.e
```

```
Entrer un nombre
990
```

C'est bon, tous les tests donnent les valeurs attendues.

1.3 Code entier

Pour voir le code entier :

[27]:
```
!cat macCarthy.s | grep -v '^#'
```

```
.data
m: .asciz "Entrer un nombre"
.balign 4
x: .word 1948 @initialisation presque aléatoire
y: .word 2048 @initialisation presque aléatoire
.text
.global main
main:
push {lr}
@Prog Principal
@10: EcrChaine("Entrer un nombre")
    ldr R1, ptr_m
    bl EcrChaine
@11: Lire32(&x)
    ldr r1, ptr_x
    bl Lire32
@12 y:=f91(x)
  ldr r1, ptr_x
  ldr r1,[r1]
  push {r1}
```

```
  mov r0, #42 @valeur presque aléatoire pour un pseudo résultat
  push {r0} @ place pour le résultat, on peut aussi juste déplacer sp (sub sp,
sp, #4)
  bl f91
  pop {r0}
  pop {r1} @ récupération place paramètre, on peut aussi juste déplacer sp (add
sp, sp, #4)
  ldr r1,ptr_y
  str r0, [r1]
@13: EcrNdecimal32(y)
  ldr r1, ptr_y
  ldr r1, [r1]
  bl EcrNdecimal32
mov r0,#0
pop {lr}
bx lr
@Fonction 91 de MacCarthy
f91:
prologue:
  push {lr}
  push {fp}
  mov fp,sp
  mov r0, #65 @valeur presque aléatoire pour loc
  push {r0} @pour loc, on peut aussi juste déplacer sp (sub sp, sp, #4)
  push {r0}
  push {r1}
  push {r2}
corps:
@1: si n > 100 alors
  ldr r1,[fp,#12]
  cmp r1,#100
  ble l.4
@2: loc = n-10
  ldr r1,[fp,#12]
  sub r1,r1,#10
  str r1,[fp,#-4]
@3: sinon
  b l.6
l.4:
@4:loc = f91(n+11)
  ldr r0,[fp,#12]
  add r0, r0, #11
  push {r0}
  push {r0} @ place pour le résultat, on peut aussi juste déplacer sp (sub sp,
sp, #4)
  bl f91
  pop {r0}
  str r0,[fp,#-4]
```

```
  pop {r0} @ récupère la place argument, on peut aussi juste déplacer sp (ou ne
rien faire et ne pas faire les 2 instructions suivantes)
@5:loc = f91(loc)
  ldr r0,[fp,#-4]
  push {r0}
  push {r0} @ place pour le résultat, on peut aussi juste déplacer sp (sub sp,
sp, #4)
  bl f91
  pop {r0}
  str r0,[fp,#-4]
  pop {r0} @ récupère la place argument, on peut aussi juste déplacer sp (add
sp, sp, #4)
l.6: @ finsi
  nop
l.7: @ retourne loc
  ldr r0,[fp,#-4]
  str r0,[fp,#8]
epilogue:
  pop {r2}
  pop {r1}
  pop {r0}
  add sp, sp, #4
  pop {fp}
  pop {lr}
  bx lr
    bx lr
ptr_m: .word m
ptr_x: .word x
ptr_y: .word y
```