LINFO 1140
Rapport projet bonus - Assembleur

Cheffert Théo
NOMA : 25882000

UCLouvain

Professeur : O. Bonaventure

epl

1 Description de la fonction

Afin de choisir un bon exercice en assembleur en vue de me préparer à l'interrogation, j'ai décidé de partir sur la programmation d'une fonction. Celle-ci calcule et retourne le nombre d'éléments pairs d'un tableau d'entiers donné en argument.

La fonction prend comme premier argument, dans le registre D, **l'adresse** du tableau sur lequel on va appliquer la fonction. Les éléments du tableau doivent être des entiers positifs (0 inclus). Ensuite, comme deuxième argument, cette fonction prend **la longueur** du tableau (étiquette *len*). Enfin, comme dernier argument, elle prend **la fonction "parity"** qui retourne 1 si D (l'argument qu'elle reçoit) est *pair* et 0 sinon (*impair*). Ces 2 arguments sont placés sur la *pile* (cf. schéma annexe).

J'ai donc codé la fonction "*parity*". Celle-ci ne fait cependant pas partie de l'exercice. En effet, le but est de programmer une fonction "**func**" qui renvoie dans le *registre A* le nombre d'éléments pairs au sein du tableau (placé dans D).

2 Code en Python

Voici une implémentation en **Python** d'une telle fonction (*CodeProjetBonus.py* dans le zip).

Code Python

```
#--Fonction qui compte le nombre d'elements pairs d'un tableau--#
def count_even_elem(tab):
    '''
    pre: tab est un tableau d'entiers.
    post: retourne le nombre d'elements pairs de ce tab
    (0 est pris comme un nombre pair).
    '''
    count = 0
    for i in range(len(tab)):
        if tab[i] % 2 == 0:
            count += 1
    return count


#--Quelques tests--#
tab1 = [0, 2, 4, 6, 8]
tab2 = [0, 2, 5, 6]
tab3 = [1, 3, 5, 7]
tab4 = []
tab5 = [-1, -2, 3, 4]

assert (count_even_elem(tab1) == 5)
assert (count_even_elem(tab2) == 3)
assert (count_even_elem(tab3) == 0)
assert (count_even_elem(tab4) == 0)
assert (count_even_elem(tab5) == 2)

#-----------------------------------------------------------------#
```

3 Code en Assembleur

Je mets dans cette section la fonction "**func**" uniquement. La totalité du code fonctionnel, se trouve en annexe (version écrite) ainsi que dans le zip (fichier ASM).

Code Assembleur

```
func: PUSH B     ;On sauvegarde les 2 registres B et C
      PUSH C
      PUSH 0     ;Accumulateur qui va stocker le nombre d'elems pairs
      MOV C, 0   ;L'index i

loop: MOV B, D          ;On place dans B l'adresse du tableau D
      MOV D, [B]        ;On place dans D l'elem tab[i]

      MOV A, [SP+8]     ;On place dans A la fonction "parity"
      CALL A            ;On appelle "parity"

      CMP A, 0          ;On compare si tab[i] est pair
      JE not            ;Si impair -> on va à "not"

      ADD B, 2         ;Sinon, on se deplace d'un élem dans le tab
      MOV D, B         ;L'adresse du prochain élem se trouve dans D

      POP A            ;tab[i] est donc pair -> on recupere l'acc
      ADD A, 1         ;On lui ajoute 1 car il y a 1 élem pair
      PUSH A           ;On le remet sur la pile

suite: INC C             ;On continue la fonction -> i++
       CMP C, [SP+10]    ;i ==? len(tab) ?
       JNE loop          ;Si on est pas à la fin: recommencer la loop

fin: POP A         ;On recupere les valeurs
     POP C
     POP B
     RET           ;On retourne à l'appel de la fonction (res dans A)

not: ADD B, 2      ;tab[i] est impair -> on n'ajoute rien à l'acc,
     MOV D, B      ;et on se deplace au prochain élem de tab
     JMP suite     ;On se deplace à "suite"
```

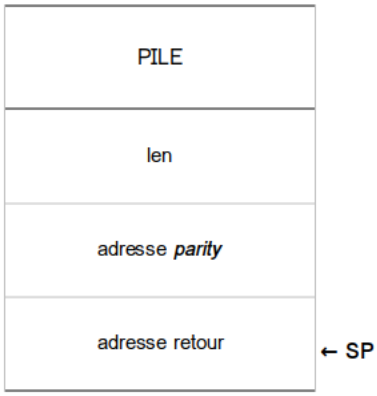
4 Liste de tests

J'ai testé mon code en **Assembleur** sur le simulateur avec les mêmes tableaux qu'en Python et à chaque fois, les résultats étaient identiques (ils renvoient tous les deux la bonne réponse). J'ai testé plusieurs cas. Tout d'abord, un tableau où tous les éléments sont pairs. Ensuite, la même chose avec que des éléments impairs. Et enfin, un mixte des 2 ainsi qu'un tableau vide. On peut retrouver **en annexe** différents tests effectués sur le simulateur (images (des tableaux, des registres et des résultats) et explications).

A Schéma montrant l'évolution de la pile

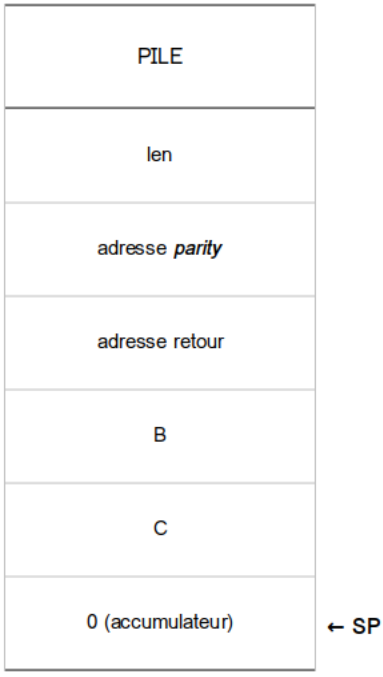
A.1 Schéma de la pile à l'appel de *CALL func*

Ceci est un schéma qui montre l'état de la pile à l'appel de la fonction "**func**".



A.2 Schéma de la pile au début de la boucle *loop*

L'accumulateur va être modifié si un nombre pair est détecté. Il restera néanmoins au même endroit sur la pile.



B Code entier en Assembleur

Un fichier **ASM** se trouve dans le **zip** et peut-être chargé dans le simulateur (tab, len et parity déja initalisé ! Il n'y a plus qu'à *RUN*).

Code entier en Assembleur

```
;---Les registres B et C sont utilises---;
MOV B, [rb]
MOV C, [rc]
MOV D, tab
PUSH [len]       ;La longueur du tableau est placee sur la pile
PUSH parity      ;La fonction "parity" est placee sur la pile
CALL func        ;Appel à la function "func"
HLT

func: PUSH B     ;On sauvegarde les 2 registres B et C
      PUSH C
      PUSH 0     ;Accumulateur qui va stocker le nombre d'elems pairs
      MOV C, 0   ;L'index i

loop: MOV B, D         ;On place dans B l'adresse du tableau D
      MOV D, [B]       ;On place dans D l'elem tab[i]

      MOV A, [SP+8]    ;On place dans A la fonction "parity"
      CALL A           ;On appelle "parity"

      CMP A, 0         ;On compare si tab[i] est pair
      JE not           ;Si impair -> on va à "not"

      ADD B, 2        ;Sinon, on se deplace d'un élem dans le tab
      MOV D, B        ;L'adresse du prochain élem se trouve dans D

      POP A           ;tab[i] est donc pair -> on recupere l'acc
      ADD A, 1        ;On lui ajoute 1 car il y a 1 élem pair
      PUSH A          ;On le remet sur la pile

suite: INC C             ;On continue la fonction -> i++
       CMP C, [SP+10]    ;i ==? len(tab) ?
       JNE loop          ;Si on est pas à la fin: recommencer la loop

fin: POP A         ;On recupere les valeurs
     POP C
     POP B
     RET           ;On retourne à l'appel de la fonction (res dans A)

not: ADD B, 2      ;tab[i] est impair -> on n'ajoute rien à l'acc,
     MOV D, B      ;et on se deplace au prochain élem de tab
     JMP suite     ;On se deplace à "suite"
```

Suite du code

```
;---Fonction parity---;
;Pre: D est un nombre positif (>= à 0)
;Post: retourne dans A la valeur 1 si D est pair et 0 sinon (impair)

parity: MOV A, D
        DIV 2
        MUL 2       ;Compare (D//2)*2 == D :
        CMP A, D    ;(si reste à la div alors D impair)
        JE even

        MOV A, 0
        RET

even: MOV A, 1
      RET

;---Variables---;
tab: DB "?"
     DB "?"
      ...
     DB "?"

len: DB "?"
rb: DB "?"
rc: DB "?"
```

C Tests effectués sur le simulateur

C.1 Test où tous les éléments sont pairs

```
;---Variables---;
tab: DB 0
     DB 2
     DB 4
     DB 6
     DB 8

len: DB 5
rb: DB 0
rc: DB 0
```

Labels

Name	Address	Value
even	00 | B8	00 | 06
fin	00 | 7A	00 | 36
func	00 | 20	00 | 32
len	00 | CA	00 | 05
loop	00 | 32	00 | 01
not	00 | 88	00 | 0D
parity	00 | 98	00 | 01
rb	00 | CC	00 | 00
rc	00 | CE	00 | 00
suite	00 | 6C	00 | 12
tab	00 | C0	00 | 00

CPU & Memory

Registers / Flags

A	B	C	D	PC	SP	Z	C	F
05	00	00	CA	1F	398	TRUE	FALSE	FALSE

La réponse attendue de 5 éléments se trouve bien dans le registre A.

C.2 Test où tous les éléments sont impairs

```
;---Variables---;
tab: DB 1
     DB 3
     DB 5
     DB 7
     DB 9

len: DB 5
rb: DB 0
rc: DB 0
```

Labels

Name	Address	Value
even	00 | B8	00 | 06
fin	00 | 7A	00 | 36
func	00 | 20	00 | 32
len	00 | CA	00 | 05
loop	00 | 32	00 | 01
not	00 | 88	00 | 0D
parity	00 | 98	00 | 01
rb	00 | CC	00 | 00
rc	00 | CE	00 | 00
suite	00 | 6C	00 | 12
tab	00 | C0	00 | 01

CPU & Memory

Registers / Flags

A	B	C	D	PC	SP	Z	C	F
00	00	00	CA	1F	398	TRUE	FALSE	FALSE

La réponse attendue de 0 éléments se trouve bien dans le registre A.

C.3 Test où il y a 3 éléments pairs et 2 impairs

```
;---Variables---;
tab: DB 1
     DB 0
     DB 2
     DB 4
     DB 61

len: DB 5
rb: DB 0
rc: DB 0
```

Labels

Name	Address	Value
even	00 | B8	00 | 06
fin	00 | 7A	00 | 36
func	00 | 20	00 | 32
len	00 | CA	00 | 05
loop	00 | 32	00 | 01
not	00 | 88	00 | 0D
parity	00 | 98	00 | 01
rb	00 | CC	00 | 00
rc	00 | CE	00 | 00
suite	00 | 6C	00 | 12
tab	00 | C0	00 | 01

CPU & Memory

Registers / Flags

A	B	C	D	PC	SP	Z	C	F
03	00	00	CA	1F	398	TRUE	FALSE	FALSE

La réponse attendue de 3 éléments se trouve bien dans le registre A.