USTL

INFO 202 – **2009/2010**

Architecture des ordinateurs : Fiche de TD 4

Microprocesseurs : la pile

novembre 2009

1 Piles et sous-routines

On considère le microprocesseur traité dans les TD précédents.

Les instructions données dans les exercices précédents ne comportent pas de gestion de piles. Nous allons essayer de palier ce manque du microprocesseur. Par exemple, on va considérer que le registre $RX3$ est le pointeur de pile, l'adresse FFFF étant considérée comme la base de pile (voir Figure 1). Le registre $RX3$ est donc initialisé à la valeur FFFF. Ceci signifie que les registres $R6$ et $R7$ (constituant $RX3$) ne peuvent plus être utilisés pour autre chose que la pile.

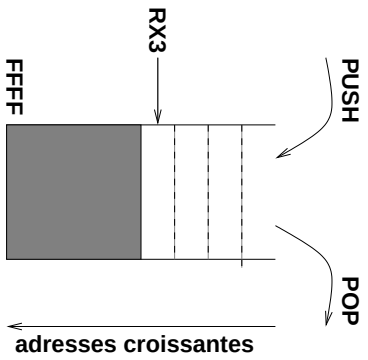

Figure 1: Pile avec $RX3$ comme pointeur de pile

Question 1.1 : *Écrire les séquences d'instructions assembleur correspondant à une* `PUSH Rn` *et à un* `POP Rn`.

Solution 1.1 :

- `PUSH Rn`

```
    SWP R0, Rn
    ST R0, RX3
    DEC R6
    JC dec_R7
    JMP suite
dec_R7:
    dec R7
suite:
    SWP R0, Rn
```

Notons que si `Rn`=`R0`, on peut supprimer les `SWP`. De plus, si les labels `dec_R7` ou `suite` existe déjà, il faut générer des nouveaux labels.

- `POP R`*n*

```
    INC R6
    JC inc_R7
    JMP suite
inc_R7:
    inc R7
suite:
    SWP R0, Rn
    LD R0, RX3
    SWP R0, Rn
```

Même remarque que précédemment.

Autre manque du jeu d'instructions : il n'y a pas possibilités d'avoir un appel à une sous-routine.

Question 1.2 : *Donnez les suites d'instructions assembleur correspondant à une* `CALL HHLL` *(l'adresse est donnée en "dur") et à un* `RET`.

Rappel. *Un* `CALL` *sauvegarde l'adresse de retour dans la pile et effectue un branchement à l'adresse de la sous-routine. Un* `RET` *récupère l'adresse de retour dans la pile et effectue un branchement à cette adresse.*

Lors du `CALL`*, le compilateur peut calculer l'adresse de retour (l'adresse de l'instruction qui suit le* `CALL`*), il est donc inutile de récupérer l'adresse de retour à partir de PC (ce qui est d'ailleurs impossible avec notre architecture). On va supposer que cette adresse de retour est* `MMNN` *(*`NN` *étant l'octet de poids faible).*

Solution 1.2 :

- `CALL HHLL`

```
    ; push NN
    MOV R0, NN
    ST R0, RX3
    DEC R6
    JC dec_R7
    JMP suite
dec_R7:
    dec R7
suite:
    ; push MM
    MOV R0, MM
    ST R0, RX3
    DEC R6
    JC dec_R7_1
    JMP suite_1
dec_R7_1:
    dec R7
suite_1:
    ; saut à la sous-routine
    JMP HHLL
```

L'adresse `MMNN` est donc l'adresse de l'instruction suivant le `JMP`.

- `RET`

  On doit mettre l'adresse de retour dans $RX0$ qui est le seul registre permettant de faire un saut indirect (comprendre dont l'adresse est donnée dans un registre et non pas directement).

```
        ; pop R1
        INC R6
        JC inc_R7
        JMP suite
    inc_R7:
        inc R7
    suite:
        LD R0, RX3
        SWP R0, R1
        ; pop R0
        INC R6
        JC inc_R7_1
        JMP suite_1
    inc_R7_1:
        INC R7
    suite_1:
        LD R0, RX3
        ; saut à RX0
        JMP RX0
```

Ça veut donc dire que lors du retour, les valeurs de $R0$ et $R1$ sont modifiées ce qui peut être gênant (on a déjà supprimé $R6$ et $R7$, manquerait plus qu'on supprime aussi $R0$ et $R1$:-). On va donc modifier le `CALL` pour qu'il fasse la sauvegarde et la restauration de $R0$ et $R1$:

```
    ; push R0
    ST R0, RX3
    DEC R6
    JC dec_R7
    JMP suite
dec_R7:
    DEC R7
suite:
    ; push R1
    SWP R0, R1
    ST R0, RX3
    SWP R0, R1    ; on re-permute pour les remettre comme il faut
    DEC R6
    JC dec_R7_1
    JMP suite_1
dec_R7_1:
    DEC R7
suite_1:
    ; push NN
    MOV R0, NN
    ST R0, RX3
    DEC R6
    JC dec_R7_2
    JMP suite_2
dec_R7_2:
    dec R7
```

```
suite_2:
    ; push MM
    MOV R0, MM
    ST R0, RX3
    DEC R6
    JC dec_R7_3
    JMP suite_3
dec_R7_3:
    dec R7
suite_3:
    ; saut à la sous-routine
    JMP HHLL
    ; ici MMNN doit correspondre à l'adresse du label "retour"
retour:
    ; pop R1
    INC R6
    JC inc_R7
    JMP suite_4
inc_R7:
    inc R7
suite_4:
    LD R0, RX3
    SWP R0, R1
    ; pop R0
    INC R6
    JC inc_R7_1
    JMP suite_5
inc_R7_1:
    INC R7
suite_5:
    LD R0, RX3
```