

# Le langage machine

M. Dubacq

S1D 2009

## 1 Premiers pas en VLTN

**Objectif :** *Savoir simuler une machine, comprendre le fonctionnement d'une machine élémentaire, connaître les instructions VLTN.*

1. Soit le programme suivant (code machine et assembleur sont donnés) :

```
00: 90 50 01    // READ 50 01
01: 90 51 02    // READ 51 02
02: 21 50 03    // LOAD 50 03
03: 40 51 04    // ADD 51 04
04: 31 52 05    // STOR 52 05
05: 91 52 06    // WRIT 52 06
06: 00 00 00    // EJECT 0000
```

Les nombres avant les deux-points sont les adresses mémoires, les chiffres derrière le contenu des cases mémoires (par exemple la case 05 contient 915206). Ce qui est derrière les // sont les instructions en assembleur VLTN correspondantes.

Représentez la machine VLTN au démarrage (ACCU,PC et cases mémoires différentes de 0). En supposant que les entrées sont 000024 et 000048, simulez pas-à-pas la machine VLTN jusqu'à ce qu'elle s'arrête (représentez à chaque cycle l'état complet de la machine).

2. Examinez le programme suivant :

```
00: 90 50 86
10: 40 51 41
12: 21 50 10
41: 31 52 99
86: 90 51 12
91: 00 00 00
99: 91 52 91
```

En face de chaque ligne, mettez la traduction en assembleur VLTN du code machine correspondant.

3. Simulez la machine VLTN si on lui donne en entrée ce même programme. On supposera que les entrées sont 000024 et 000048. Que remarquez-vous ?
4. Que se passe-t-il si on ne fournit pas d'entrées du tout à ce programme ?
5. Que se passe-t-il si on ne met pas le contenu de l'adresse 91 ?
6. Voilà un programme en assembleur VLTN :

```
00: _____ // READ 50 01
01: _____ // TSTE 04 02
02: _____ // READ 51 03
03: _____ // TSTE 04 10
04: _____ // EJECT 0001
10: _____ // LOAD 50 11
11: _____ // SUB 51 12
12: _____ // TSTE 20 13
13: _____ // WRIT 50 14
14: _____ // WRIT 51 15
15: _____ // EJECT 0000
20: _____ // WRIT 51 21
21: _____ // WRIT 50 15
```

Convertissez les instructions en code machine.

7. Dans le programme précédent, que se passe-t-il ? Simulez le programme sur diverses entrées.

## 2 Boucle

**Objectif :** Comprendre les structures de contrôle élémentaires, connaître les instructions VLTN.

1. Faites un court programme qui écrit 5 fois le nombre 42. Puis 120 fois.
2. Modifiez votre programme pour qu'il demande deux nombres  $a$  et  $b$ , puis qu'il affiche  $a$  fois le nombre  $b$ .

```
00: 10 42 01 // SET 42 01
01: 31 50 02 // STOR 50 02
02: 10 05 03 // SET 05 03
03: 01 10 04 // TSTN 10 04
04: 91 50 05 // WRIT 50 05
05: 45 01 03 // SUBI 01 03
10: 00 00 00 // EJECT 0000
```

Pour lire  $a$  et  $b$ , il suffit de faire à la place des deux premières lignes : *READ 51, READ 50, LOAD 51* et de continuer à la ligne 3. On observera qu'on peut aussi répondre à la question 1 en faisant 5 *WRIT* à la suite, mais qu'on ne peut plus pour 120...

## 3 Modes d'adressage

**Objectif :** Comprendre les différents modes d'adressage.

Pour toutes les instructions VLTN, dites dans quel type de mode d'adressage on se trouve.

- *TSTE, TSTN* : code absolu/code absolu
- *RET* : adressage relatif
- Toutes les autres, code absolu (sauf *EJECT*).
- Pour l'opérande : jamais d'adressage par registre puisqu'il n'y a pas le choix du registre (ce n'est donc pas précisé dans l'instruction).
- *MEM[YY]* : adressage direct
- *MEM[PT+YY]* : adressage indexé (offset+base)
- Utilisation directe de *YY* : immédiat
- Pas d'utilisation de *YY* (*SWAP, COPY, SAVE*) : adressage implicite

## 4 Somme d'entiers

**Objectif :** Comprendre la gestion d'entrées/sorties en VLTN, connaître les instructions VLTN.

1. Traduisez le programme suivant en code machine :

```
00: _____ // READ 50 01
01: _____ // TSTE 02 03
02: _____ // EJECT 0001
03: _____ // SET 00 04
04: _____ // STOR 51 05
05: _____ // LOAD 50 06
06: _____ // ADD 51 07
07: _____ // TSTE 08 10
08: _____ // WRIT 50 09
09: _____ // EJECT 0002
10: _____ // STOR 51 11
11: _____ // LOAD 50 12
12: _____ // SUBI 01 13
13: _____ // STOR 50 14
14: _____ // TSTN 15 05
15: _____ // WRIT 51 16
16: _____ // EJECT 0000
```

2. Simulez ce programme avec l'entrée 000004.
3. Que fait ce programme s'il n'a aucune entrée ? *Il sort avec un code d'erreur 0001.*
4. Que fait ce programme, dans le cas général ? Réécrivez-le en langage algorithmique (ou en C)...
5. Dans quel cas ce programme donne-t-il le code d'erreur 0002 ? *Il sort avec un code d'erreur 0002 lorsque l'addition atteint ou dépasse 1000000, c'est à dire quand on essaye de calculer la somme d'entiers trop nombreux.*

## 5 Suite récurrente

**Objectif :** Comprendre les structures de contrôle élémentaires, comprendre les manipulations arithmétiques élémentaires, connaître les instructions VLTN.

1. Soit la suite suivante :  $U_0 = 1$  et  $U_{n+1} = 2U_n + 5$ . Écrivez un programme VLTN qui demande un entier, et calcule le terme  $U_n$  de la suite. *Dans cette correction, pas de numéro de ligne... Commencer à introduire la notation avec étiquettes.*

```

READ 50
SET 01
STOR 51
_bcl: LOAD 51
TSTN _fin
LOAD 50
ADD 50
ADDI 05
STOR 50
GOTO bcl
_fin: WRIT 50
EJECT 0000

```

2. Comment, avec nos moyens limités peut-on tester si un nombre est pair ? *Si on multiplie un nombre par 5, et qu'on le divise par 10, TSTE est nul si le nombre est pair, et non-nul sinon. Donc quelque chose comme ça :*

```

LOAD 50
ADD 50
ADD 50
ADD 50
ADD 50
SHFR 01
TSTE _impair,_pair
...

```

3. Même exercice, avec la suite suivante ( $U_0$  et la valeur de  $n$  à laquelle on s'arrête sont demandées au clavier) :

$$U_{n+1} = \begin{cases} 3U_n + 1 & \text{si } U_n \text{ est impair} \\ U_n/2 & \text{sinon} \end{cases}$$

## 6 Lecture dans un tableau

**Objectif :** Comprendre les différents modes d'adressage.

1. Faites un programme qui demande 10 nombres, et les met successivement dans la case mémoire 50.

```

SET 10
_test: TSTN _fin,_suite
_suite: READ 50
SUBI 01,_test
_fin: EJECT 0000

```

2. Faites un programme qui remplit les adresses 59 à 50 par 10 nombres, en utilisant l'adressage *indexé*.

```

SET 10
STOR 60
_test: LOAD 60
TSTN _fin,_suite
_suite: READ 61
SUBI 01
STOR 60
SWAP
LOAD 61
STIX 50 // Stocke dans MEM[PT+50]
GOTO _test
_fin: EJECT 0000

```

3. Faites un programme qui remplit les adresses 50 à 59 par 10 nombres, en utilisant l'adressage *indirect*. *...à faire... TODO XXX*

## 7 Multiplication

**Objectif :** Comprendre les structures de contrôle élémentaires, comprendre les manipulations arithmétiques élémentaires, connaître les instructions VLTN.

1. Donnez le segment de données tel qu'on dispose de trois adresses `Multiplicateur`, `Multiplicande` et `Produit`, avec des valeurs initiales de 6 et de 7 pour les deux premières.

```
.data
Multiplicateur: .word 6
Multiplicande: .word 7
Produit:        .word 0 # ou n'importe quoi d'autre
```

2. Faites le segment de texte complet qui permet de multiplier les nombres situés aux deux premières adresses en faisant des additions successives, qui écrive le résultat dans la troisième.

```
.text
.globl main
main:      Load R0,Multiplicateur
           Load R1,Multiplicande
           Set R3,0

boucle:    Branch=0 R1,finboucle
           Add R3,R3,R2
           Addi R1,R1,-1
           Jump boucle

finboucle: Store R3, Produit
           Set R0,3
           Syscall
```

## 8 Fibonacci

**Objectif :** Comprendre les structures de contrôle élémentaires, comprendre les manipulations arithmétiques élémentaires, connaître les instructions VLTN.

La suite de Fibonacci est la suite définie par  $U_n = U_{n-1} + U_{n-2}$ , et  $U_0 = U_1 = 1$ . Affichez le 25e terme de la suite de Fibonacci (c'est-à-dire  $U_{24}$ ).

```
.text
.globl main
main:      Set R0,1
           Set R1,1
           Set R3,24

boucle:    Branch=0 R3,affichage
           Add R2,R0,R1
           Move R0,R1
           Move R1,R2
           Addi R3,R3,-1
           Jump boucle

affichage: Set R0,1
           Syscall
           Set R0,3
           Syscall
```

## 9 Instructions natives

**Objectif :** Comprendre la différence entre langage assembleur et langage machine.

Un programme en assembleur sur un machine 32 bits comporte 100 000 mnémoniques, dont 14% sont des pseudo-instructions remplacées par 2 instructions natives, 1% des pseudo-instructions remplacées par 3 instructions natives, 1% des pseudo-instructions remplacées par 4 instructions natives.

1. Quel est le nombre d'instructions natives du segment de texte de ce programme ? *Nombre d'instructions natives :  $84000 + 42000 + 3000 + 4000 = 133000$*
2. Quel est le nombre d'octets occupés par le segment de texte de ce programme ? *Soit 432000 octets.*
3. Est-ce que ça changerait pour une machine 64 bits ? *Oui, la taille serait le double. Mais il est probable qu'un certain nombre de pseudo-instructions soient remplacées par moins d'instructions natives.*

## 10 Tri à bulles

**Objectif :** Comprendre un algorithme complexe, comprendre les structures de contrôle élémentaires, connaître les instructions VLTN.

Le tri à bulles est une méthode simple pour trier un tableau d'entier : si on a un tableau de  $n$  entiers, il suffit de répéter  $n - 1$  fois de suite l'opération suivante : pour chaque élément du tableau qui n'est pas le dernier, on compare cet élément et le suivant dans le tableau. Si le premier est plus grand que le deuxième, on les inverse dans le tableau ; sinon, on ne fait rien. Lorsqu'on a parcouru  $n - 1$  fois le tableau, le tableau contient les éléments triés par ordre croissant.

1. Écrire en langage algorithmique le programme de tri à bulles.
2. Analysez le programme suivant. Dites quelles adresses contiennent quoi, ajoutez des commentaires.

```

_main: READ 50
      SETP 55
      LOAD 50
      STOR 51
_bcl1: READ 54
      LOAD 54
      STIX 00
      SWAP
      ADDI 01
      SWAP
      LOAD 51
      SUBI 01
      STOR 51
      TSTN _fbcl _bcl
_fbcl1:LOAD 50
      SUBI 01
      STOR 52
_bcl2: SETP 55
      LOAD 50
      SUBI 01
      STOR 53
_blc3: LDIX 01
      STOR 54
      LDIX 00
      SUB 54
      TSTE _ok _pasok
      _pasok:LDIX 00
              STIX 01
              LOAD 54
              STIX 00
      _ok:   SWAP
              ADDI 01
              SWAP
              LOAD 53
              SUBI 01
              STOR 53
              TSTN _fbcl3 _bcl3
      _fbcl3:LOAD 52
              SUBI 01
              STOR 52
              TSTN _fbcl2 _bcl2
      _fbcl2:SETP 55
      _bcl4: LDIX 00
              STOR 54
              WRIT 54
              SWAP
              ADDI 01
              SWAP
              LOAD 50
              SUBI 01
              STOR 50
              TSTN _fin _bcl4
      _fin:  EJECT 0000

```

```

_main: READ 50
      SETP 55
      LOAD 50
      STOR 51
_bcl1: READ 54 // lecture
      LOAD 54 // stockage au
      STIX 00 // bon endroit
      SWAP
      ADDI 01 // endroit suivant
      SWAP
      LOAD 51 // i=i-1
      SUBI 01
      STOR 51
      TSTN _fbcl _bcl // fin?
_fbcl1:LOAD 50 // ici, tout est entré
      SUBI 01
      STOR 52 // boucles de rangement restantes
_bcl2: SETP 55 // on pointe au début du tableau
      LOAD 50
      SUBI 01
      STOR 53 // nombre de rangements restants
_bcl3: LDIX 01 // on lit le deuxième élément
      STOR 54 // on le stocke
      LDIX 00 // on lit le premier élément
      SUB 54 // on compare les éléments
      TSTE _ok _pasok //
      _pasok:LDIX 00 // inversion
              STIX 01
              LOAD 54
              STIX 00 // inversé!
_ok:      SWAP
              ADDI 01 // endroit suivant
              SWAP
              LOAD 53
              SUBI 01
              STOR 53 // Test dernier rangement
              TSTN _fbcl3 _bcl3
_fbcl3:LOAD 52
              SUBI 01
              STOR 52 // Test dernière répétition
              TSTN _fbcl2 _bcl2
_fbcl2:SETP 55
_bcl4: LDIX 00
      STOR 54
      WRIT 54
      SWAP
      ADDI 01
      SWAP
      LOAD 50
      SUBI 01
      STOR 50
      TSTN _fin _bcl4
_fin: EJECT 0000

```