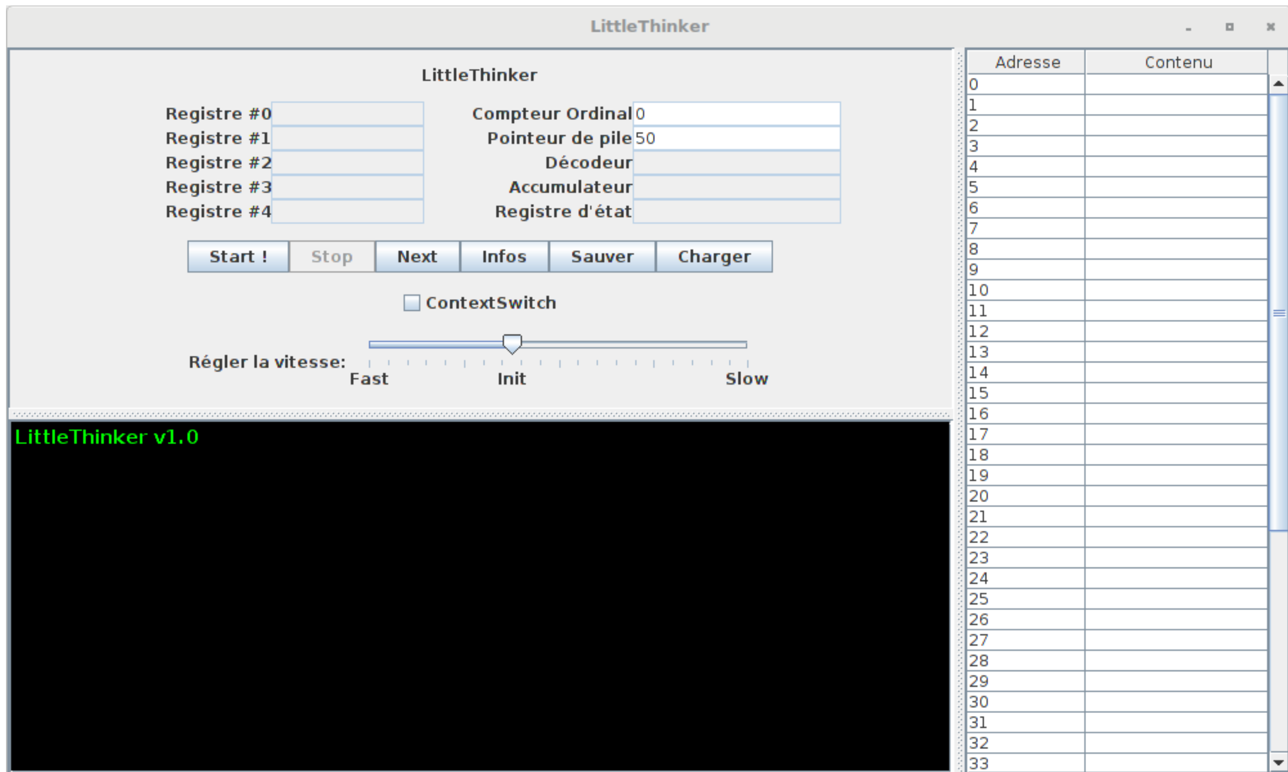
LittleThinker Version 1.0



I. Introduction

Ce logiciel permet de simuler un processeur… C'est une vision simplifiée, avec 5 registres (numérotés de #0 à #4), un accumulateur (pour récupérer le résultat de calculs), un compteur ordinal (pour pointer sur l'adresse mémoire contenant l'instruction à exécuter), un décodeur (qui affiche l'instruction en cours), un registre d'état (qui alerte sur les événements importants, tels que un résultat négatif (N), ou nul (Z)), et un pointeur de pile qui permet de gérer une pile dans la mémoire.

La mémoire est constituée de 51 cases, dans lesquelles on peut saisir des instructions et des données. L'adresse d'une case mémoire est notée avec un $.

Sur l'écran d'exemple, on peut voir un petit programme qui commence à l'adresse $0 (la première colonne indique l'adresse mémoire). Ce petit programme contient quatre instructions, et manipule une adresse mémoire ($10).

Actuellement, LittleThinker accepte l'adressage absolu direct ($NuméroDeLaCaseMémoire) et l'adressage indirect ($#NuméroDuRegistre) par un registre, c'est à dire que l'on met dans un registre un nombre X, et c'est le contenu de la case $X qui sera manipulé…

Adresse	Contenu
0	ld #0 $10
1	ld a 14
2	sub #0
3	st a $11
4	
5	
6	
7	
8	
9	
10	16
11	
12	

II. Commandes de base

II.A) Syntaxe

La syntaxe des commandes est la suivante :

Commande** espace **argument1** espace**…. argumentN

En général, les commandes prennent un ou deux arguments. Les commandes sont en minuscules. L’espace est le séparateur d’argument. On écrit la commande complète dans une case (C’est un petit arrangement : Dans un vrai processeur, on parle de mot mémoire, et les instructions se codent au minimum sur un mot mémoire…)

II.B) Transferts

On ne peut pas directement travailler en mémoire centrale. Si on veut manipuler des données, on doit les transférer dans des registres du processeur, faire des opérations dessus grâce à l’accumulateur, puis remettre les données en mémoire centrale…

II.B.1 – ld (load)

ld cible valeur

Chargement des registres ou de l’accumulateur. Cette commande permet de stocker une **valeur** dans un des registres (de #0 à #4), ou dans l’accumulateur (a) : c’est la **cible**. La valeur peut être donnée directement, ou contenue dans une case mémoire (adressage direct), ou contenue dans une case mémoire dont l’adresse est dans un registre (adressage indirect)…

Exemple :

La première ligne écrit le nombre 25 dans le registre #0.

La seconde écrit le contenu de la case mémoire $4 (ici le chiffre 6) dans le registre #1

La troisième ligne stocke dans l’accumulateur le contenu de la case mémoire pointée par le registre #1 (#1 contient 6 grâce à l’instruction précédente, donc on met dans l’accumulateur le contenu de la case $6, c’est à dire le nombre 43…)

Cette commande peut déclencher le drapeau Z (valeur nulle) ou N (valeur négative) dans le registre d’état…

Adresse	Contenu
0	ld #0 25
1	ld #1 $4
2	ld a $#1
3	
4	6
5	
6	43
7	
8	
9	
10	

II.B.2 – st (store)

st valeur cible

Ecriture dans la mémoire. Cette commande permet de stocker en mémoire une **valeur**. Cette valeur peut être directement indiquée, ou être celle contenue dans l’accumulateur (a), ou l’un des registres (de #0 à #4). Quant à la **cible**, on peut soit directement donner son adresse (adressage direct), soit indiquer le numéro du registre qui contient l’adresse (adressage indirect)

Exemple : (L’écran correspond au résultat, après éxécution)

Ici, la ligne 0 charge l’accumulateur avec la valeur 25, la ligne 1 met la valeur 5 dans le registre #0, et la ligne 2 met 9 dans le registre #1.

La ligne 3 stocke la valeur 10 dans la case mémoire numéro 8 (adressage direct)

La ligne 4 stocke la valeur contenu dans le registre #0 (ici 5, d’après la ligne 1) dans la case pointée par #1 ($9, d’après la ligne 2)

La ligne 5 stocke la valeur contenu dans l’accumulateur (ici 25, d’après la ligne 0) dans la case mémoire 42.

Adresse	Contenu
0	ld a 25
1	ld #0 5
2	ld #1 9
3	st 10 $8
4	st #0 $#1
5	st a $10
6	
7	
8	10
9	5
10	25
11	

II.B.3 – mv (move)

mv origine cible

Copie en interne du processeur. Cette commande permet de copier le contenu d’un registre (ou de l’accumulateur) vers l’accumulateur (ou un registre). Sans cette commande, il faudrait passer par la mémoire centrale.
Exemple :
mv #0 #2 : copie le contenu du registre 0 dans le registre 2
mv a #1 : copie la valeur de l'accumulateur dans le registre 1
mv #3 a : copie le contenu du registre 3 dans l'accumulateur

II.C) Les opérations mathématiques

Dans LittleThinker, on peut appliquer des opérations mathématiques (pour le moment en décimal) dans l’accumulateur. C’est le rôle de l’accumulateur. Il récupère le résultat des opérations. Il est cependant possible de faire une opération très simple d’incrémentation et de décrémentation directement sur les registres (#0 à #4).

II.C.1 – inc (incrémentation)

inc cible

Ajoute 1 au contenu du registre **cible**.
Exemples :
inc #0
inc #4
Cette commande peut déclencher le drapeau Z (valeur nulle) ou N (valeur négative) dans le registre d’état…

II.C.2 – dec (decrémentation)

dec cible

Retire 1 au contenu du registre **cible**.
Exemples :
dec #0
dec #4
Cette commande peut déclencher le drapeau Z (valeur nulle) ou N (valeur négative) dans le registre d’état…

II.C.3 – add (addition)

add valeur

Fait une addition, et stocke le résultat dans l’accumulateur. Cette commande prend le contenu de l’accumulateur, additionne à celui-ci la valeur passée en argument, puis met le résultat obtenu dans l’accumulateur. Celui-ci est donc modifié de façon irréversible. Cette commande accepte les valeurs, un registre ou une case mémoire. Il est possible d’utiliser l’adressage indirect.

Adresse	Contenu
0	ld #0 7
1	ld a 25
2	add 2
3	add #0
4	add $8
5	add $#0
6	
7	20
8	-12
9	

La première instruction met la valeur 7 dans le registre #0, et la seconde charge la valeur 25 dans l'accumulateur.
Ensuite, on ajoute 2 au contenu de l'accumulateur (on obtient donc 27). **Add #0** ajoute le contenu du registre #0 (ici 7, voir la ligne 0) à l'accumulateur, qui valait déjà 27. On arrive à 34. **Add $8** ajoute ensuite le contenu de la case mémoire $8 (-12, qui avait été saisi dans cette case). L'accumulateur revient donc à 22. Enfin, grâce à l'adressage indirect, j'ajoute le contenu de la case pointée par le registre #0 (qui contient toujours 7, voir la ligne 0). Le contenu de la case $7 est 20, le résultat stocké dans l'accumulateur sera de 22+20 donc 42.
Cette commande peut déclencher le drapeau Z (valeur nulle) ou N (valeur négative) dans le registre d'état…

II.C.4 – sub (soustraction)

sub valeur

Fait une soustraction, et stocke le résultat dans l'accumulateur. Cette commande fonctionne exactement comme l'addition, et offre les mêmes caractéristiques d'adressage. Voir donc l'addition (add). NOTA : L'argument est soustrait au contenu de l'accumulateur. Si l'accumulateur contient 10, et que LittleThinker exécute la commande **sub 3**, on obtiendra 10-3 donc 7 dans l'accumulateur.
Cette commande peut déclencher le drapeau Z (valeur nulle) ou N (valeur négative) dans le registre d'état…

II.C.5 – mul (multiplication)

mul valeur

Fait une multiplication, et stocke le résultat dans l'accumulateur. Cette commande fonctionne exactement comme l'addition, et offre les mêmes caractéristiques d'adressage. Voir donc l'addition (add). Cette commande peut déclencher le drapeau Z (valeur nulle) ou N (valeur négative) dans le registre d'état…

II.C.6 – div (division)

div valeur

Fait une division, et stocke le résultat dans l'accumulateur. Cette commande fonctionne exactement comme l'addition, et offre les mêmes caractéristiques d'adressage. Voir donc l'addition (add). NOTA : L'argument divise le contenu de l'accumulateur par l'argument. Si l'accumulateur contient 10, et que LittleThinker exécute la commande div 2, on obtiendra 10/2 donc 5 dans l'accumulateur.
Cette commande peut déclencher le drapeau Z (valeur nulle), D (la division est à virgule) ou N (valeur négative) dans le registre d'état…

II.C.7 – mod (modulo)

mod valeur

Fait une division entière, et stocke le reste dans l'accumulateur. Cette commande fonctionne exactement comme l'addition, et offre les mêmes caractéristiques d'adressage. Voir donc l'addition

(add). NOTA : L’argument divise le contenu de l’accumulateur. Si l’accumulateur contenait 10, et que LittleThinker exécute la commande mod 3, on obtiendra 10/3-le diviseur est 3 et le reste est 1-donc 1 dans l’accumulateur.
Cette commande peut déclencher le drapeau Z (valeur nulle) ou N (valeur négative) dans le registre d’état…

II.C.8 – cmp (comparaison)

cmp valeur

Cette commande compare l'accumulateur à la valeur passée en argument à cmp. Cet argument peut être une valeur directement exprimée (2 par exemple), un registre (#1 par exemple), une case mémoire ($17, ou $#3 par exemple). Si l'accumulateur est plus grand que la valeur, les registres d'état sont inchangés. Si l'accumulateur est égal à la valeur, alors le Z apparaît (Zéro). Et enfin, si l'accumulateur est plus petit que la valeur, le drapeau N est levé...

II.D) Les ruptures

Les ruptures sont des commandes qui permettent de modifier le déroulement du programme. Selon certains événements (mise à zéro, valeur négative, etc), on va forcer le compteur ordinal à contenir une autre adresse que celle de l’instruction suivante. C’est ce que l’on appelle un saut ou branchement. Les commandes commencent toutes par b (branchement), et dépendent du registre d’état (le registre d’état est modifié automatiquement par certaines opérations mathématiques, voir II.C). Les commandes b* vont permettre de tester ces drapeaux, et de dérouter le programme sur l’adresse de l’instruction voulue.

II.D.1 – beq (branchement si égal à zéro)

beq adresse

Saut à l’adresse indiquée si le drapeau Z est levé. Les commandes **ld inc dec add sub mul div mod cmp** peuvent déclencher le drapeau Z si on obtient un zéro. La commande **beq** permet de dérouter vers l’adresse passée en argument dans ce cas.

II.D.2 – bnz (branchement différent de zéro)

bnz adresse

Saut à l’adresse indiquée si le drapeau Z *n’est pas* levé. Les commandes **ld inc dec add sub mul div mod cmp** peuvent déclencher le drapeau Z si on obtient un zéro.

II.D.3 – bde (branchement si décimal)

bde adresse

Saut à l’adresse indiquée si le drapeau D est levé. La commande **div** peut déclencher le drapeau D si le reste de la division entière est différent de zéro.

II.D.4 – bnd (branchement si pas décimal)

bnd adresse

Saut à l’adresse indiquée si le drapeau D **n’est pas** levé. La commande **div** peut déclencher le drapeau D si le reste de la division entière est différent de zéro.

II.D.5 – bmi (branchement si négatif)

bmi adresse

Saut à l’adresse indiquée si le drapeau N est levé. Les commandes **ld inc dec add sub mul div cmp** peuvent déclencher le drapeau N si on obtient un nombre négatif.

II.D.6 – bpl (branchement si positif)

bpl adresse

Saut à l’adresse indiquée si le drapeau N *n’est pas* levé. Les commandes **ld inc dec add sub mul div cmp** peuvent déclencher le drapeau N si on obtient un nombre négatif.

II.D.7 – brn (branchement inconditionnel)

brn adresse

Saut automatique à l’adresse indiquée, sans aucune condition.

II.E) La pile

Dans LittleThinker, une pile est simulée par le pointeur de pile. Les valeurs stockées dans la mémoire au-delà de l’adresse du pointeur de pile peuvent être récupérées une par une avec la commande **pp**. On peut ajouter une valeur dans la pile avec la commande **ps**.

II.E.1 – ps (push)

ps valeur

Ajoute une **valeur** en haut de la pile, à l’adresse du pointeur de pile. Cette valeur peut être directement indiquée, ou être celle contenue dans l’accumulateur (a), ou l’un des registres (de #0 à #4). Décrémente ensuite le pointeur de pile.
Fonctionne comme un **st** donc la cible est l’adresse contenue dans le pointeur de pile.

II.E.2 – pp (pop)

pp cible

Récupère la valeur en haut de la pile et la place dans le registre **cible**. Incrémente ensuite le pointeur de pile.
Fonctionne comme un ld dont la valeur est la valeur contenue à l’adresse spécifiée par le pointeur de cible – 1.

III. Exemple complet de programme

Voici une petite version d'un programme qui convertit un nombre décimal en binaire. Le nombre en décimal se range en $13, et le résultat est lu en partant du plus loin possible et en remontant jusqu'à $14 (donc ici, de $19 à $14). On obtient donc de la conversion de $42_{(10)}$ en $101010_{(2)}$.
La première instruction stocke le contenu de la case $13 dans le registre #0, qui servira de référence pendant tout l'algorithme. On stocke dans #1 l'adresse contenant le bit de poids faible de la conversion. En incrémentant ce registre, on pourra ranger au fur et à mesure tous les bits de la conversion. En 2, je commence ma boucle. La valeur décimale est mise dans l'accumulateur, puis divisée par 2 (inst. 3). Selon que l'opération tombe juste ou non, je mets un 1 ou un 0 dans la case pointée par le registre #1 (Remarquez comment le si-alors-sinon est codé, grâce à deux sauts). Puis, j'incrémente le registre #1, afin de passer au bit suivant. Ensuite, comme je ne peux pas tester le contenu de l'accumulateur, je déplace le contenu dans le registre #0, ce qui me permettra de détecter le passage à 0, signe que la conversion est terminée…

Adresse	Contenu
0	ld #0 $13
1	ld #1 14
2	mv #0 a
3	div 2
4	bnd 7
5	st 1 $#1
6	brn 8
7	st 0 $#1
8	inc #1
9	mv a #0
10	bnz 2
11	
12	
13	42
14	0
15	1
16	0
17	1
18	0
19	1
20	
21	

IV. Utiliser LittleThinker

IV.A) Les fichiers

LittleThinker permet la sauvegarde et le chargement de vos programmes. LittleThinker ne peux charger que des fichiers d'extension .lt. Lors de la sauvegarde, si vous ne spécifiez pas l'extension .lt, LittleThinker la rajoutera de lui même.

Les fichiers sont cependant modifiable comme des fichiers texte. Ainsi, vous pouvez écrire vos programme directement dans les fichier, puis les charger dans LittleThinker afin de les tester.

IV.B) ContextSwitch

Si la case ContextSwitch est cochée, alors l'état du processeur sera sauvegardé après chaque instruction lue dans un fichier context.txt qui sera créé dans le répertoire courant.

Cela peut permettre de débugguer plus rapidement un programme...

Exemple :

Fichier context.txt après éxécution du programme de la partie **III.**

```
1  Compteur Ordinal: 1 | Décodeur: ld #0 $13 | Accumulateur:  | Registre d'état:  | Registres: 42 Vide Vide Vide Vide
2  Compteur Ordinal: 2 | Décodeur: ld #1 14 | Accumulateur:  | Registre d'état:  | Registres: 42 14 Vide Vide Vide
3  Compteur Ordinal: 3 | Décodeur: mv #0 a | Accumulateur: 42 | Registre d'état:  | Registres: 42 14 Vide Vide Vide
4  Compteur Ordinal: 4 | Décodeur: div 2 | Accumulateur: 21 | Registre d'état:  | Registres: 42 14 Vide Vide Vide
5  Compteur Ordinal: 7 | Décodeur: bnd 7 | Accumulateur: 21 | Registre d'état:  | Registres: 42 14 Vide Vide Vide
6  Compteur Ordinal: 8 | Décodeur: st 0 $#1 | Accumulateur: 21 | Registre d'état: Z | Registres: 42 14 Vide Vide Vide
7  Compteur Ordinal: 9 | Décodeur: inc #1 | Accumulateur: 21 | Registre d'état:  | Registres: 42 15 Vide Vide Vide
8  Compteur Ordinal: 10 | Décodeur: mv a #0 | Accumulateur: 21 | Registre d'état:  | Registres: 21 15 Vide Vide Vide
9  Compteur Ordinal: 2 | Décodeur: bnz 2 | Accumulateur: 21 | Registre d'état:  | Registres: 21 15 Vide Vide Vide
10 Compteur Ordinal: 3 | Décodeur: mv #0 a | Accumulateur: 21 | Registre d'état:  | Registres: 21 15 Vide Vide Vide
11 Compteur Ordinal: 4 | Décodeur: div 2 | Accumulateur: 10 | Registre d'état: D | Registres: 21 15 Vide Vide Vide
12 Compteur Ordinal: 5 | Décodeur: bnd 7 | Accumulateur: 10 | Registre d'état: D | Registres: 21 15 Vide Vide Vide
13 Compteur Ordinal: 6 | Décodeur: st 1 $#1 | Accumulateur: 10 | Registre d'état:  | Registres: 21 15 Vide Vide Vide
14 Compteur Ordinal: 8 | Décodeur: brn 8 | Accumulateur: 10 | Registre d'état:  | Registres: 21 15 Vide Vide Vide
15 Compteur Ordinal: 9 | Décodeur: inc #1 | Accumulateur: 10 | Registre d'état:  | Registres: 21 16 Vide Vide Vide
16 Compteur Ordinal: 10 | Décodeur: mv a #0 | Accumulateur: 10 | Registre d'état:  | Registres: 10 16 Vide Vide Vide
17 Compteur Ordinal: 2 | Décodeur: bnz 2 | Accumulateur: 10 | Registre d'état:  | Registres: 10 16 Vide Vide Vide
18 Compteur Ordinal: 3 | Décodeur: mv #0 a | Accumulateur: 10 | Registre d'état:  | Registres: 10 16 Vide Vide Vide
19 Compteur Ordinal: 4 | Décodeur: div 2 | Accumulateur: 5 | Registre d'état:  | Registres: 10 16 Vide Vide Vide
20 Compteur Ordinal: 7 | Décodeur: bnd 7 | Accumulateur: 5 | Registre d'état:  | Registres: 10 16 Vide Vide Vide
21 Compteur Ordinal: 8 | Décodeur: st 0 $#1 | Accumulateur: 5 | Registre d'état: Z | Registres: 10 16 Vide Vide Vide
22 Compteur Ordinal: 9 | Décodeur: inc #1 | Accumulateur: 5 | Registre d'état:  | Registres: 10 17 Vide Vide Vide
23 Compteur Ordinal: 10 | Décodeur: mv a #0 | Accumulateur: 5 | Registre d'état:  | Registres: 5 17 Vide Vide Vide
24 Compteur Ordinal: 2 | Décodeur: bnz 2 | Accumulateur: 5 | Registre d'état:  | Registres: 5 17 Vide Vide Vide
25 Compteur Ordinal: 3 | Décodeur: mv #0 a | Accumulateur: 5 | Registre d'état:  | Registres: 5 17 Vide Vide Vide
26 Compteur Ordinal: 4 | Décodeur: div 2 | Accumulateur: 2 | Registre d'état: D | Registres: 5 17 Vide Vide Vide
27 Compteur Ordinal: 5 | Décodeur: bnd 7 | Accumulateur: 2 | Registre d'état: D | Registres: 5 17 Vide Vide Vide
28 Compteur Ordinal: 6 | Décodeur: st 1 $#1 | Accumulateur: 2 | Registre d'état:  | Registres: 5 17 Vide Vide Vide
29 Compteur Ordinal: 8 | Décodeur: brn 8 | Accumulateur: 2 | Registre d'état:  | Registres: 5 17 Vide Vide Vide
30 Compteur Ordinal: 9 | Décodeur: inc #1 | Accumulateur: 2 | Registre d'état:  | Registres: 5 18 Vide Vide Vide
31 Compteur Ordinal: 10 | Décodeur: mv a #0 | Accumulateur: 2 | Registre d'état:  | Registres: 2 18 Vide Vide Vide
32 Compteur Ordinal: 2 | Décodeur: bnz 2 | Accumulateur: 2 | Registre d'état:  | Registres: 2 18 Vide Vide Vide
33 Compteur Ordinal: 3 | Décodeur: mv #0 a | Accumulateur: 2 | Registre d'état:  | Registres: 2 18 Vide Vide Vide
34 Compteur Ordinal: 4 | Décodeur: div 2 | Accumulateur: 1 | Registre d'état:  | Registres: 2 18 Vide Vide Vide
35 Compteur Ordinal: 7 | Décodeur: bnd 7 | Accumulateur: 1 | Registre d'état:  | Registres: 2 18 Vide Vide Vide
36 Compteur Ordinal: 8 | Décodeur: st 0 $#1 | Accumulateur: 1 | Registre d'état: Z | Registres: 2 18 Vide Vide Vide
37 Compteur Ordinal: 9 | Décodeur: inc #1 | Accumulateur: 1 | Registre d'état:  | Registres: 2 19 Vide Vide Vide
38 Compteur Ordinal: 10 | Décodeur: mv a #0 | Accumulateur: 1 | Registre d'état:  | Registres: 1 19 Vide Vide Vide
39 Compteur Ordinal: 2 | Décodeur: bnz 2 | Accumulateur: 1 | Registre d'état:  | Registres: 1 19 Vide Vide Vide
40 Compteur Ordinal: 3 | Décodeur: mv #0 a | Accumulateur: 1 | Registre d'état:  | Registres: 1 19 Vide Vide Vide
41 Compteur Ordinal: 4 | Décodeur: div 2 | Accumulateur: 0 | Registre d'état: D | Registres: 1 19 Vide Vide Vide
42 Compteur Ordinal: 5 | Décodeur: bnd 7 | Accumulateur: 0 | Registre d'état: D | Registres: 1 19 Vide Vide Vide
43 Compteur Ordinal: 6 | Décodeur: st 1 $#1 | Accumulateur: 0 | Registre d'état:  | Registres: 1 19 Vide Vide Vide
44 Compteur Ordinal: 8 | Décodeur: brn 8 | Accumulateur: 0 | Registre d'état:  | Registres: 1 19 Vide Vide Vide
45 Compteur Ordinal: 9 | Décodeur: inc #1 | Accumulateur: 0 | Registre d'état:  | Registres: 1 20 Vide Vide Vide
46 Compteur Ordinal: 10 | Décodeur: mv a #0 | Accumulateur: 0 | Registre d'état: Z | Registres: 0 20 Vide Vide Vide
47 Compteur Ordinal: 11 | Décodeur: bnz 2 | Accumulateur: 0 | Registre d'état: Z | Registres: 0 20 Vide Vide Vide
48 Compteur Ordinal: 11 | Décodeur: STOP | Accumulateur: 0 | Registre d'état: Z | Registres: 0 20 Vide Vide Vide
49
```