Université Paris-Nord — L1
Institut Galilée — Année 2008-2009
Licence 1 - 1er semestre

Travaux dirigés : programmation en mini-assembleur, suite.

Correction. Note aux chargés de TD.

- Dans le texte des corrections se trouvent également quelques explications de correction. Bien entendu, vous n'avez pas à en parler aux étudiants (boucle for, `gcc -S`, etc.).
- Amil est très rudimentaire dans ses possibilités d'écrire des commentaires. Dans cette feuille, il y a des commentaires ligne à ligne, mais aussi une tentative d'indiquer où sont les différents blocs de code dans un style balise xml/html : une ligne avec en commentaire `<bloc1>` signale la première ligne du bloc `bloc1`, une ligne avec en commentaire `</bloc1>` signale la dernière ligne du bloc, et une ligne avec en commentaire `<bloc1/>` signale l'unique ligne du bloc `bloc1`.
- les traces contiennent une colonne *instructions* : cette colonne n'a pas lieu d'être dans les corrections (elle aide juste à relire la trace, qui est générée automatiquement).

Rappel des instructions

`stop`	Arrête l'exécution du programme.
`noop`	N'effectue aucune opération.
`saut i`	Met le compteur ordinal à la valeur i.
`sisaut ri j`	Si la valeur contenue dans le registre i est positive ou nulle, met le compteur ordinal à la valeur j.
`init x ri`	Initialise le registre i avec la valeur x.
`lecture i rj`	Charge, dans le registre j, le contenu de la mémoire d'adresse i.
`lecture *ri rj`	Charge, dans le registre j, le contenu de la mémoire dont l'adresse est la valeur du registre i.
`ecriture ri j`	Écrit le contenu du registre i dans la mémoire d'adresse j.
`ecriture ri *rj`	Écrit le contenu du registre i dans la mémoire dont l'adresse est la valeur du registre j.
`inverse ri`	Inverse le signe du contenu du registre i.
`add x rj`	Ajoute x au contenu du registre j.
`add ri rj`	Ajoute la valeur du registre i à celle du registre j.
`mult`, `div`, `et`	Même syntaxe que pour `add` mais pour la multiplication, la division entière et le et bit à bit.

1 Exécution conditionnelle d'instructions

Soit la donnée x d'adresse 15. On veut écrire la valeur absolue de x à l'adresse 16.

1. Décrire un algorithme réalisant cette tâche.

Instructions	Cycles	CP	r0	15	16
INIT	0	1	?	5	?
`lecture 15 r0`	1	2	5		
`sisaut r0 6`	2	**6**			
`ecriture r0 16`	3	7			5
`stop`	4	8			

FIG. 1 – Trace du calcul de la valeur absolue de 5

Instructions	Cycles	CP	r0	15	16
INIT	0	1	?	-5	?
`lecture 15 r0`	1	2	-5		
`sisaut r0 6`	2	3			
`inverse r0`	3	4	5		
`ecriture r0 16`	4	5			5
`saut 7`	5	**7**			
`stop`	6	8			

FIG. 2 – Trace du calcul de la valeur absolue de -5

Correction.
- Si $x < 0$
- Alors écrire $-x$ à l'adresse 16
- Sinon écrire x à l'adresse 16

2. Écrire un programme réalisant cette tâche.

Correction.
- Si $x < 0$

```
1   lecture 15 r0
2   sisaut r0 6    <-- saute sur le sinon
```

- Alors écrire $-x$ à l'adresse 16

```
3   inverse r0
4   ecriture r0 16
5   saut 7         <-- saute sur le stop
```

- Sinon écrire x à l'adresse 16

```
6   ecriture r0 16
```

- suite du programme

```
7   stop
8   ?
⋮
14  ?
15  5
16  ?
```

3. Construire une trace de votre programme lorsque x vaut 5, puis lorsque x vaut -5.

Correction. Voir les figures 1 et 2.

2 Boucles d'instructions

2.1 Boucles infinies

1. Avec l'instruction `saut`, écrire un programme qui ne termine jamais.

   Correction.

```
1   saut 1
2   stop # <- jamais atteint
```

2. Avec l'instruction `sisaut`, écrire un programme qui ne termine jamais.

   Correction.

```
1   init 0 r0
2   sisaut r0 1
3   stop # <- jamais atteint
```

2.2 Boucles de calculs itératifs

Soit un entier n d'adresse 15.

1. Écrire un programme qui lorsque $n \geq 0$, écrit la valeur n à l'adresse 16, puis, toujours à l'adresse 16, écrit la valeur $n-1$, puis $n-2$, et ainsi de suite jusqu'à écrire 0, après quoi le programme termine. Si n est négatif, le programme ne fait rien.

   Correction.
   – x vaut n
   – Tant que $x \geq 0$: écrire x puis enlever 1 à x
   Pour la traduction en assembleur de l'algorithme, on imite le schéma de traduction standard du `while (...) { ... }` . Dans cette traduction le test décidant de (la répétition de) l'exécution de la boucle, vient juste après le corps de la boucle plutôt qu'avant. Et il y a juste avant le corps de boucle un saut vers ce test. Ainsi la condition de saut traduit la condition d'exécution (non sa négation) en un minimum de lignes.

```
1   lecture 15 r0
2   saut 5          <-- saut sur le test d'execution de boucle
3   ecriture r0 16 ::  <corps de boucle>
4   add -1 r0      :: </corps de boucle>
5   sisaut r0 3    :: <condition de boucle/>, saut sur le corps de boucle
6   stop
7   ?
8   ?
9   ?
10  ?
11  ?
12  ?
13  ?
14  ?
```

Instructions	Cycles	CP	r0	15	16
INIT	0	1	?	3	?
`lecture 15 r0`	1	2	3		
`saut 5`	2	**5**			
`sisaut r0 3`	3	**3**			
`ecriture r0 16`	4	4			3
`add -1 r0`	5	5	2		
`sisaut r0 3`	6	**3**			
`ecriture r0 16`	7	4			2
`add -1 r0`	8	5	1		
`sisaut r0 3`	9	**3**			
`ecriture r0 16`	10	4			1
`add -1 r0`	11	5	0		
`sisaut r0 3`	12	**3**			
`ecriture r0 16`	13	4			0
`add -1 r0`	14	5	-1		
`sisaut r0 3`	15	6			
`stop`	16	7			

FIG. 3 – Décrément pour $n = 3$

```
15   3
16   ?
```

2. Décrire un algorithme calculant la somme des entiers 0, 1, ..., n. Par convention cette somme, $\sum_{i=0}^{n} i = 0+1+\ldots+n$ est nulle lorsque $n < 0$. Écrire le programme correspondant. Le résultat de la sommation sera écrit à l'adresse 16.

Correction. Attention plusieurs réponses correctes sont possibles, mais il faudra donner la correction suivante.

La correction suivante reprend la traduction en assembleur (`gcc -S`) d'un `while (...) ...`, ou d'une boucle `for( i = 0 ; i <= n ; i = i + 1)` en C. L'algorithme consiste en copier la formule de la somme :
– la somme vaut 0, i vaut 0.
– Tant que $i \leq n$, ajouter x_i à la somme puis ajouter 1 à i.
Le programme est alors :

```
1   lecture 15 r0   <---------------------------------------- r0 vaut n
2   init 0 r1       <----------------------- accumulateur pour la somme
3   init 0 r2       <------------- initialisation de l'indice de boucle
4   saut 7          <------ saut sur la condition d'execution de boucle
5   add r2 r1       :: <corps_de_boucle>
6   add 1 r2        :: </corps_de_boucle>
7   init 0 r3       :: <condition_de_boucle>
8   add r2 r3
9   inverse r3
10  add r0 r3       <---------------------------------- r3 vaut n - r2
11  sisaut r3 5     :: </condition_de_boucle>, saut sur corps_de_boucle
```

```
12   ecriture r1 16
13   stop
14   ?
15   3               <-- n
16   ?               <-- valeur de retour
```

La boucle incrémente i. Dans amil, à cause de la difficulté à écrire des tests tels que $i \leq n$, il serait plus rapide sur cet exercice de décrémenter n jusqu'à 0 comme à la question précédente :

– x vaut n, la somme vaut 0
– Tant que $x \geq 0$, ajouter x à la somme et décrementer x.

Cette réponse est correcte et on peut encourager dans un premier temps les étudiants qui cherchent dans cette direction, mais, pour les préparer à la suite du cours, il faut leur donner la correction qui incrémente l'indice.

Parmi les autres programmes corrects possibles, des étudiants peuvent même penser à la formule de Gauss $\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$.

3. (Optionnel) Tester votre programme en construisant sa trace pour $n = 3$.

Correction. Cette question n'est à traitée qu'au cas où les étudiants auraient eu de grosses difficultés avec la précédente trace. Dans un groupe ou les traces passent mal, la similitude avec l'exercice précédent peut aider à leur faire refaire une trace (dans ce cas on donne la correction). La trace est donnée figure 4.

2.3 Boucles de parcours

Soient n entiers $x_1, \ldots, x_n$ rangées aux adresses $30+1, \ldots, 30+n$. La valeur $n > 0$ est elle même rangée à l'adresse 30.

1. Décrire un algorithme qui effectue la somme $\sum_{i=1}^{n} x_i = x_1 + \ldots + x_n$.

Correction.
– La somme vaut 0
– Pour i allant de 1 à n, ajouter x_i à la somme.

2. Écrire un programme réalisant cet algorithme. (Vous aurez besoin des instructions `lecture *ri rj` et `ecriture ri *rj`). Le résultat sera écrit à l'adresse $30 + n + 1$.

Correction.

```
1    lecture 30 r1    <---------------------------------------- r1 vaut n
2    init 0 r0        <---------------------------------------- r0 vaut 0
3    init 1 r2        <------------- initialisation de l'indice de boucle
4    saut 10          <------------------ saut sur la condition de boucle
5    init 30 r4       :: <corps_de_boucle>
6    add r2 r4
7    lecture *r4 r5
8    add r5 r0        <---- r0 contient la somme des r2 premiers elements
9    add 1 r2         :: </corps_de_boucle>
10   init 0 r3        :: <condition_de_boucle>
```

Instructions	Cycles	CP	r0	r1	r2	r3	15	16
INIT	0	1	?	?	?	?	3	?
`lecture 15 r0`	1	2	3					
`init 0 r1`	2	3		0				
`init 0 r2`	3	4			0			
`saut 7`	4	**7**						
`init 0 r3`	5	8				0		
`add r2 r3`	6	9				0		
`inverse r3`	7	10				0		
`add r0 r3`	8	11				3		
`sisaut r3 5`	9	**5**						
`add r2 r1`	10	6		0				
`add 1 r2`	11	7			1			
`init 0 r3`	12	8				0		
`add r2 r3`	13	9				1		
`inverse r3`	14	10				-1		
`add r0 r3`	15	11				2		
`sisaut r3 5`	16	**5**						
`add r2 r1`	17	6		1				
`add 1 r2`	18	7			2			
`init 0 r3`	19	8				0		
`add r2 r3`	20	9				2		
`inverse r3`	21	10				-2		
`add r0 r3`	22	11				1		
`sisaut r3 5`	23	**5**						
`add r2 r1`	24	6		3				
`add 1 r2`	25	7			3			
`init 0 r3`	26	8				0		
`add r2 r3`	27	9				3		
`inverse r3`	28	10				-3		
`add r0 r3`	29	11				0		
`sisaut r3 5`	30	**5**						
`add r2 r1`	31	6		6				
`add 1 r2`	32	7			4			
`init 0 r3`	33	8				0		
`add r2 r3`	34	9				4		
`inverse r3`	35	10				-4		
`add r0 r3`	36	11				-1		
`sisaut r3 5`	37	12						
`ecriture r1 16`	38	13						6
`stop`	39	14						

FIG. 4 – Somme des entiers de 0 à 3

```
11  add r2 r3
12  inverse r3
13  add r1 r3       <---------------------------------- r3 vaut n - r2
14  sisaut r3 5     :: </condition_de_boucle>, saut sur corps_de_boucle
15  init 31 r4
16  add r1 r4
17  ecriture r0 *r4
18  stop
19  ?
20  ?
21  ?
22  ?
23  ?
24  ?
25  ?
26  ?
27  ?
28  ?
29  ?
30  3
31  100
32  1
33  10
34  ? <-- resultat
```

3. Écrire un algorithme qui trouve le plus petit des entiers parmi $x_1, \ldots, x_n$. En déduire un programme réalisant cette recherche du minimum. Le résultat sera écrit à l'adresse $30 + n + 1$.

Correction.
- Prendre x_1 comme étant le minimum.
- Pour i allant de 2 à n :
  - Si x_i est plus petit que le minimun alors prendre x_i comme étant le minimum.

```
1   lecture 31 r0   <--------- r0 contient le minimum, initialisé a x1
2   lecture 30 r1   <------------------------------------ r1 vaut n
3   init 2 r2       <------------ initialisation de l'indice de boucle
4   SAUT 14         <----------------- saut sur la condition de boucle
5   init 30 r4      :: <corps_de_boucle>
6   add r2 r4
7   lecture *r4 r5  <--------------- r5 contient l'element suivant, xi
8   add r0 r6       :: <condition_du_si>              (xi < minimum ?)
9   inverse r6
10  add r5 r6       <---------------------------- r6 vaut xi - minimum
11  SISAUT r6 13    :: </condition_du_si>    saute apres le bloc alors
12  lecture *r4 r0  :: <alors/>
13  add 1 r2        :: </corps_de_boucle>
14  init 0 r3       :: <condition_de_boucle>
```

Cycles	CP	instruction	r0	r1	r2	r3	r4	r5	30	31	32	33	34
INIT	1		?	?	?	?	?	?	3	100	1	10	?
1	2	lecture 30 r1		3									
2	3	init 0 r0	0										
3	4	init 1 r2			1								
4	**10**	saut 10											
5	11	init 0 r3				0							
6	12	add r2 r3				1							
7	13	inverse r3				-1							
8	14	add r1 r3				2							
9	**5**	sisaut r3 5											
10	6	init 30 r4					30						
11	7	add r2 r4					31						
12	8	lecture *r4 r5						100					
13	9	add r5 r0	100										
14	10	add 1 r2			2								
15	11	init 0 r3				0							
16	12	add r2 r3				2							
17	13	inverse r3				-2							
18	14	add r1 r3				1							
19	**5**	sisaut r3 5											
20	6	init 30 r4					30						
21	7	add r2 r4					32						
22	8	lecture *r4 r5						1					
23	9	add r5 r0	101										
24	10	add 1 r2			3								
25	11	init 0 r3				0							
26	12	add r2 r3				3							
27	13	inverse r3				-3							
28	14	add r1 r3				0							
29	**5**	sisaut r3 5											
30	6	init 30 r4					30						
31	7	add r2 r4					33						
32	8	lecture *r4 r5						10					
33	9	add r5 r0	111										
34	10	add 1 r2			4								
35	11	init 0 r3				0							
36	12	add r2 r3				4							
37	13	inverse r3				-4							
38	14	add r1 r3				-1							
39	15	sisaut r3 5											
40	16	init 31 r4					31						
41	17	add r1 r4					34						
42	18	ecriture r0 *r4											111
43	19	stop											

FIG. 5 – Somme des entiers $x_1, \ldots, x_n$

```
15  add r2 r3
16  inverse r3
17  add r1 r3       <---------------------------------- r3 vaut n - r2
18  SISAUT r3 5     :: </condition_de_boucle> saut sur corps_de_boucle
19  init 31 r4
20  add r1 r4       <------------------------------ r4 vaut 30 + n + 1
21  ecriture r0 *r4
22  stop
23  ?
24  ?
25  ?
26  ?
27  ?
28  ?
29  ?
30  3
31  10
32  100
33  1
34  ? <-- resultat
```

Instructions	Cycles	CP	r0	r1	r2	r3	r4	r5	r6	30	31	32	33	34
INIT	0	1	?	?	?	?	?	?	?	3	10	100	1	?
`lecture 31 r0`	1	2	10											
`lecture 30 r1`	2	3		3										
`init 2 r2`	3	4			2									
`saut 14`	4	**14**												
`init 0 r3`	5	15				0								
`add r2 r3`	6	16				2								
`inverse r3`	7	17				-2								
`add r1 r3`	8	18				1								
`sisaut r3 5`	9	**5**												
`init 30 r4`	10	6					30							
`add r2 r4`	11	7					32							
`lecture *r4 r5`	12	8						100						
`add r0 r6`	13	9							10					
`inverse r6`	14	10							-10					
`add r5 r6`	15	11							90					
`sisaut r6 13`	16	**13**												
`add 1 r2`	17	14			3									
`init 0 r3`	18	15				0								
`add r2 r3`	19	16				3								
`inverse r3`	20	17				-3								
`add r1 r3`	21	18				0								
`sisaut r3 5`	22	**5**												
`init 30 r4`	23	6					30							
`add r2 r4`	24	7					33							
`lecture *r4 r5`	25	8						1						
`add r0 r6`	26	9							100					
`inverse r6`	27	10							-100					
`add r5 r6`	28	11							-99					
`sisaut r6 13`	29	12												
`lecture *r4 r0`	30	13	1											
`add 1 r2`	31	14			4									
`init 0 r3`	32	15				0								
`add r2 r3`	33	16				4								
`inverse r3`	34	17				-4								
`add r1 r3`	35	18				-1								
`sisaut r3 5`	36	19												
`init 31 r4`	37	20					31							
`add r1 r4`	38	21					34							
`ecriture r0 *r4`	39	22												1
`stop`	40	23												

FIG. 6 – Minimum parmi $x_1, \ldots, x_n$