Travaux dirigés 1 : Assembleur, affectation en C, if else.

1 Langage machine

L'objectif de cette première partie est de vous familiariser avec le cycle d'exécution d'un processeur et avec la notion de flux d'instructions. Pour cela, il vous est demandé d'écrire de petits programmes dans le langage assembleur `amil` (*assembleur miniature pour l'informatique de licence*) et de simuler leur exécution soit à la main en écrivant une trace soit en utilisant le simulateur en ligne : `http://www-lipn.univ-paris13.fr/~boudes/amilweb/`.

Jeu d'instructions (simplifié)

`stop`	Arrête l'exécution du programme.
`noop`	N'effectue aucune opération.
`saut i`	Met le compteur de programme à la valeur i.
`sautpos ri j`	Si la valeur contenue dans le registre i est positive ou nulle, met le compteur de programme à la valeur j.
`valeur 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.
`ecriture ri j`	Écrit le contenu du registre i dans la mémoire d'adresse j.
`inverse ri`	Inverse le signe du contenu du registre i.
`add ri rj`	Ajoute la valeur du registre i à celle du registre j (la somme obtenue est placée dans le registre j).
`soustr ri rj`	Soustrait la valeur du registre i à celle du registre j (la différence obtenue est placée dans le registre j).
`mult ri rj`	Multiplie par la valeur du registre i celle du registre j (le produit obtenu est placé dans le registre j).

1.1 Écriture de programmes simples

Écrire les programmes répondant aux problèmes suivants :

1. Soit la valeur 8 contenue dans la case mémoire d'adresse 10. Recopier cette valeur à l'adresse 11.
2. Soit une valeur quelconque, a, contenue à l'adresse 10. Écrire la valeur de $a \times 2 + 1$ dans la case d'adresse 11.

1.2 Trace de programme assembleur

Nous allons désormais produire une représentation de l'exécution pas à pas de nos programmes. Une *trace* d'un programme assembleur sera un tableau dont chaque ligne correspond à l'exécution d'une instruction par le processeur. Une colonne contiendra le cycle d'horloge du processeur et une autre colonne le compteur de programme. Il y aura également une colonne par registre utilisé dans le programme et par case mémoire où des données sont lues ou écrites par les instructions du programme. Une première ligne, d'initialisation, montrera l'état de ces registres et cases mémoires avant le début du programme. Ensuite, chaque exécution d'instruction sera représentée par une nouvelle ligne du tableau, jusqu'à exécution de l'instruction `stop`. Dans cette ligne nous représenterons le cycle d'horloge du processeur (en commençant à 0 pour la ligne d'initialisation), la valeur du compteur de programme après exécution de l'instruction, et, s'il y a lieu, la valeur du registre ou de la case mémoire modifiée par le programme.

1.2.1 Exemple de trace et première trace

Soit le programme ci-dessous :

```
1   lecture 10 r0
2   valeur 5 r2
3   soustr r2 r0
4   sautpos r0 8
5   valeur 0 r0
6   ecriture r0 11
7   saut 9
8   ecriture r0 11
9   stop
10  14
11  ?
```

On représentera alors sa trace comme ceci :

Instructions	Cycles	CP	r0	r2	10	11
Initialisation	0	1	?	?	14	?
`lecture 10 r0`	1	2	14			
`valeur 5 r2`	2	3		5		
`soustr r2 r0`	3	4	9			
`sautpos r0 8`	4	**8**				
`ecriture r0 11`	5	9				9
`stop`	6	10				

La colonne *Instructions* n'est pas nécessaire, elle sert juste ici à la compréhension, sans cette colonne le mot-clé *initialisation* pourra être placé dans la colonne *Cycles*. Remarquez par exemple qu'il n'y a pas de colonne pour le registre 1, puisque celui-ci n'est pas utilisé.

1. Refaire la trace du programme où la valeur 14 à l'adresse 10 est remplacée par 2.
2. Quelles sont les valeurs contenues dans les registres 0, 1 et 2 après arrêt sur `stop` ?

1.3 Exécution conditionnelle d'instructions

À l'aide de l'instruction `sautpos`, écrire les programmes correspondant aux algorithmes suivants et les exécuter sur un exemple (trace ou simulateur) afin de tester leur correction :

1. Soient la valeur a à l'adresse 15, b à l'adresse 16. Si $a >= b$ alors écrire a à l'adresse 17 sinon écrire b à l'adresse 17.
2. Soit l'âge d'une personne à l'adresse 15. Si cette personne est majeure alors écrire 1 à l'adresse 16 sinon écrire 0 à l'adresse 16.
3. Soient trois cases mémoires contenant trois entiers, a, b et c. Calculer le minimum de ces trois entiers et l'écrire dans une autre case mémoire. *Commencer par réflechir à l'algorithme.*

1.4 Boucles infinies

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

2 Premier programme C et affectation

Voici, figure 1, un premier programme C. Nous donnons ici la traduction de ce code source en amil. Il s'agit d'un artifice pédagogique, la traduction réelle en code binaire exécutable est plus compliquée. Par analogie avec la musique, le source est la partition, et le fichier exécutable est le morceau musical (codé sur le support adapté au système de lecture : un fichier mp3, un CD, etc.). La traduction est effectuée par un ensemble de programmes, le source doit donc obéir à des règles syntaxiques précises.

Les textes entre `/*` et `*/` sont des *commentaires*, ils ne feront pas partie du programme exécutable, ils servent aux humains qui manipulent les programmes. Les commentaires du programme figure 1 vous serviront au cours du semestre à structurer tous vos programmes C.

Tout programme C comporte une fonction principale, le `main()`, qui sert de point d'entrée au programme. Cette fonction doit se terminer par l'instruction `return EXIT_SUCCESS`. En renvoyant cette valeur, le `main` signale au système d'exploitation la terminaison correcte du programme.

L'instruction `int x = 5` déclare une variable `x` et fixe sa valeur initiale à 5. Le mot clé `int` signifie que cette variable contiendra un entier. Dans le code amil `x` correspond à l'adresse `10` où se trouve intialement la valeur `5`.

Programme C

```
1   /* Declaration de fonctionnalites supplementaires */
2   #include <stdlib.h> /* EXIT_SUCCESS */
3
4   /* Declaration des constantes et types utilisateurs */
5
6   /* Declaration des fonctions utilisateurs */
7
8   /* Fonction principale */
9   int main()
10  {
11      /* Declaration et initialisation des variables */
12      int x = 5;
13      int y;
14
15      y = 2;
16      x = y;
17
18      /* valeur fonction */
19      return EXIT_SUCCESS;
20  }
21
22  /* Definitions des fonctions utilisateurs */
```

Traduction

```
1   valeur 2 r0
2   ecriture r0 11
3   lecture 11 r0
4   ecriture r0 10
5   stop
6
7
8
9
10  5
11  ?
```

FIGURE 1 – Un programme C et sa traduction machine

L'instruction `int y` déclare une variable entière `y` sans l'initialiser. L'effet de cette déclaration est de réserver un espace mémoire pour y stocker un entier.

Le signe égal (`=`) a un sens bien particulier, il dénote une *affectation*. L'objectif de cette partie du TD est de bien comprendre l'affectation. La partie à gauche du signe égal doit désigner une case mémoire, c'est typiquement une variable. La partie à droite du signe égal est une expression dont la valeur sera évaluée et écrite à l'adresse à laquelle renvoie la partie gauche. Par exemple, `y = 2` a été traduit en code machine par une instruction évaluant l'expression `2` dans un registre (ici `valeur 2 r0`), et par une instruction d'écriture de la valeur trouvée dans la mémoire réservée à `y`. Une variable s'évalue comme sa valeur (celle contenue dans la mémoire correspondante, au moment de l'évaluation).

2.1 Questions

1. Quel espace mémoire a été réservé pour `y` dans le code amil ?
2. Comment a été traduite l'instruction d'affectation `x = y` en amil ?
3. Si il y avait `y = x + 2`, ligne 15 dans le programme C, à la place de `y = 2`, quel serait le code amil correspondant ? Et `x = x + 1` ligne 16 ?

2.2 Introduction du problème de l'échange de valeurs

Nous avons deux tableaux anciens, chacun accroché à un clou, et un troisième clou, libre, sur le mur d'une exposition. Pour des critères esthétiques, nous voulons changer de place nos deux tableaux sans les mettre par terre, car cela risquerait d'abîmer nos précieuses toiles, et en ne déplaçant qu'une toile à la fois. Comment faire ?

2.3 Échange des valeurs de deux variables en C

1. Écrire un programme C qui déclare et initialise deux variables entières `x` et `y` et effectue la permutation de ces deux valeurs. Commencer par écrire un algorithme, à l'aide de phrases telles que « Copier la valeur de la variable ... dans la variable ...», en vous inspirant de la question précédente.
2. Traduire ce programme C en un programme amil. On supposera que les deux variables sont stockées aux adresses `10` et `11`.

FIGURE 2 – Échanger les deux tableaux en utilisant le clou libre

3. (Facultatif). Donner d'autres solutions en assembleur à ce problème (la permutation des contenus des adresses `10` et `11`).
4. (Facultatif). Mêmes questions que précédemment mais pour faire une permutation circulaire de 3 valeurs.

3 Exécution conditionnelle d'instructions : *if*

Les programmes suivants réalisent des affichages, pour cela nous utiliserons la fonction `printf`, disponible après avoir inséré `#include <stdio.h>` en début de programme (ligne 3 dans le programme figure 1).

Testez vos programmes sur machine (avec l'aide de votre enseignant).

3.1 Majeur ou mineur ?

Soit la variable `age`, contenant l'âge d'une personne. Écrire un programme qui affiche si cette personne est majeure ou mineure. Indication : `printf("Vous êtes majeur !\n");` affiche `Vous êtes majeur !` et un saut de ligne.

3.2 Exercice type : le minimum de 3 valeurs

Soient 3 variables `a`, `b`, `c`, initialisées à des valeurs quelconques. Écrire un programme qui calcule et affiche à l'écran le minimum des 3 valeurs.

Indication : si `x` est une variable contenant l'entier 42, `printf("Solution : %d\n", x);` affichera `Solution : 42` et un saut de ligne.

3.3 Exercice type : Dans une seconde, il sera exactement...

Écrire un programme qui, étant donnée une heure représentée sous la forme de trois variables : une pour les heures, `h`, une pour les minutes, `m` et une pour les secondes, `s`, affiche l'heure qu'il sera une seconde plus tard. Il faudra envisager tous les cas possibles pour le changement d'heure. Deux exemples de sortie sont :

```
L'heure actuelle est : 23h12m12s
Dans une seconde, il sera exactement : 23h12m13s

L'heure actuelle est : 23h59m59s
Dans une seconde, il sera exactement : 00h00m00s
```

Pour l'affichage : `printf("L'heure actuelle est : %dh%dm%ds\n", h, m, s);`.