IN 101 - Examen
09/01/2024
François Pessaux

Énoncé

- **Lisez attentivement les consignes et tout le sujet avant de commencer.**

- Les documents (polys, transparents, TDs, livres ...) sont autorisés.
- Sont **absolument interdits** : le WEB, le courrier électronique, les messageries diverses et variées, le répertoire des camarades, le téléphone (même pour avoir l'heure puisque vous l'avez sur votre ordinateur).

- Votre travail sera (en partie) évalué par un mécanisme automatique. Vous **devez** respecter les règles de **nommage** des fichiers et autres **consignes** qui vous sont données.
- La connaissance de C est un pré-requis d'IN101. Ainsi, la présence d'avertissement(s) ou d'erreur(s) à la compilation réduira mécaniquement la note de l'exercice considéré.
- **Sauf indications contraires**, vos programmes doivent **gérer** les cas **d'erreur** pouvant survenir.
- Lorsqu'il vous est demandé que votre programme réponde en affichant « **Yes** » ou « **No** », il ne doit **rien** afficher d'autre, et **pas** « Oui » ou « Yes. » ou « no » ou « La réponse est : no ». Donc, pensez à retirer vos affichages de test / debug.
- La **lisibilité** et l'efficacité / simplicité / complexité de vos programmes seront **prises en compte** dans l'évaluation.

- **À la fin de l'examen**, vous devrez créer une **archive** contenant **tous** les fichiers **sources** que vous avez écrits (`.c`, `.h`). **N'incluez pas** d'exécutables dans l'archive, le mail pourrait la considérer comme un attachement dangereux et le supprimer. Le nom de cette archive devra avoir la structure suivante :
  `nom_prenom.zip` ou `.tgz` (selon l'outil d'archivage que vous utilisez).
  Exemple : `zip -r lagaffe_gaston.zip rep_exam/`
  Exemple : `tar cvzf lagaffe_gaston.tgz rep_exam/`
- Vous devrez **m'envoyer** cette archive par mail. En cas d'envoi incorrect, il vous sera demandé de refaire l'archive et l'envoi. Par contre, vous ne devrez **surtout pas modifier** les fichiers : leurs dates de dernière modification ne devront pas être ultérieures à l'heure de fin de l'épreuve sous peine d'être considérés comme nuls.
- **N'oubliez pas** d'effectuer cet envoi sinon nous devrons considérer que vous n'avez rien rendu !

- Le sujet comporte **4** pages et l'examen dure **3h** heures.

1 Microprocesseur ($\sim 60\%$)

Le microprocesseur `N.A.Z.E`® (*New Architecture for Zero Efficiency*) est constitué de 4 registres `r0`, `r1`, `r2` et `r3` pouvant contenir des entiers **32 bits** (`int`). Il comporte un immense jeu d'instructions constitué de 2 opérations :

- `movi` *constante-entière registre*
  transfère la constante entière dans le registre spécifié
- `add` *registre*$_1$ *registre*$_2$
  stocke dans *registre*$_2$ le résultat de *registre*$_1$ + *registre*$_2$

Il est donc capable d'exécuter des fichiers « exécutables » dont le format est le suivant :

1. Nombre d'instructions du programme
2. Instruction 1
3. Instruction 2
4. etc.

Exemple (`prg1.dat`) :

```
5
movi 10 r0
movi 5 r1
add r1 r0
movi 2 r2
add r2 r0
```

Exemple (`prg2.dat`) :

```
7
movi 1 r0
movi 2 r1
movi 3 r2
movi 4 r3
add r1 r0
add r2 r0
add r3 r0
```

Initialement, au tout début d'un programme, tous les registres du microprocesseur sont **initialisés à 0**.

De manière macroscopique, le but est d'écrire un simulateur pour le processeur `Naze`® qui permette d'exécuter le programme contenu dans le fichier dont le nom sera passé **en argument de ligne de commande** et qui **affiche** finalement les valeurs des 4 registres.

Nommage : Le fichier source de ce programme devra s'appeler `cpu.c`.

Format de sortie : Pour chaque registre, afficher son nom suivi d'un espace, suivi de `=`, suivi d'un espace, suivi de son contenu, un retour chariot terminant chaque ligne.

Exemples et jeux de données : Plusieurs exemples de « programmes » vous sont fournis pour tester votre programme. Le fichier `prg1.dat` contient le « programme » suivant :

```
5
movi 10 r0
movi 5  r1
add  r1  r0
movi 2  r2
add  r2  r0
```

pour lequel la sortie attendue est la suivante :

```
$ ./cpu.x prg1.dat
r0 = 17
r1 = 5
r2 = 2
r3 = 0
```

À chaque fichier `.dat` est associé un fichier `.res` qui contient l'affichage attendu afin de comparaison. Vous pourrez vous en servir pour vérifier votre programme.

Question 1.1

Par quelle structure de données pouvez-vous représenter l'état du processeur ?

Question 1.2

Par quelle structure de données pouvez-vous représenter une instruction ?

Question 1.3

Quelles sont les grandes lignes du programme (la structure du futur `main` en quelque sorte) ?

Question 1.4

On s'intéresse maintenant à la fonction qui va permettre d'acquérir le contenu du programme à exécuter. Quels sont ses domaines d'entrée et de sortie ?

Question 1.5

Quel est l'algorithme de la fonction de la question précédente ? On ne vous demande pas de gérer les cas de malformation du fichier à lire (instructions en trop, manquantes, mal formées).

Question 1.6

On s'intéresse maintenant à la fonction qui va permettre d'exécuter un programme préalablement chargé. Quels sont ses domaines d'entrée et de sortie ?

Question 1.7

Décrivez l'algorithme de cette fonction d'exécution.

Question 1.8

Implantez tout cela en C et avec un `main` qui orchestre ces traitements.

Question 1.9

Attention, question bonus : si vous bloquez sur cette question, passez à la suite et revenez-y éventuellement plus tard.

Vous savez que la plage des entiers représentables sur 32 bits est limitée. Lorsque l'on additionne deux entiers positifs et que le résultat de tient pas sur 31 bits (un bit sert pour le signe), il y a débordement (arithmétique modulo 2^{31}) et le résultat devient négatif. Il en va de même en sens inverse lorsque l'on additionne deux entiers négatifs.

Sachant que le processeur (donc votre programme) ne peut effectuer des calculs que sur des `int`, comment votre simulateur pourrait détecter qu'une addition a ou va provoquer un débordement ?

Vous pourrez considérer que vous avez à disposition les deux constantes symboliques `MAX_INT32` et `MIN_INT32` qui représentent les entiers maximal et minimal représentables sur 32 bits. Pour information, ces constantes valent respectivement 2147483647 (càd $2^{31} - 1$) et -2147483648 (càd -2^{31}).

2 Adresse IP ($\sim 40\%$)

Note : Volontairement, cet exercice n'est pas guidé comme l'est le premier. C'est à vous d'exercer vos compétences de décomposition du problème en sous-problèmes plus simples pour concevoir l'algorithme puis le programme répondant au problème posé.

Une adresse IP (v4) textuelle est composée de 4 entiers **positifs inférieurs strictement à 256** séparés par un `'.'` (point). Ces entiers représentent donc chacun un octet de l'adresse (1 octet = 8 bits et $2^8 = 256$). Par exemple, `147.4.200.15` est une adresse valide. Par contre, `147.257.28.6` n'est pas une adresse valide car son second champ est supérieur à 255.

De manière macroscopique, le but est d'écrire un programme qui reçoit une chaîne en argument de ligne de commande et vérifie que cette chaîne représente une adresse IP.

Format d'entrée : Votre programme prendra en argument de **ligne de commande** la chaîne à vérifier.

Format de sortie :

1. Si la chaîne est bien une adresse IP, alors les 4 valeurs entières doivent être affichées sur une même ligne, séparées par **un** espace, avec un **retour à la ligne** final.
2. Si la chaîne **n'est pas** une adresse IP, alors un message d'erreur doit être affiché. Il doit être de la forme `"Bad byte` x`."`, où x est 1, 2, 3 ou 4 et désigne le numéro du premier champ (de gauche à droite) qui est incorrect. Un **retour à la ligne** final devra clôturer ce message d'erreur.
3. Les éventuels autres messages d'erreur sont libres.

Question 2.1

Écrivez le programme C répondant à ce problème. Vous décrirez tout ce que vous jugerez pertinent comme éléments d'analyse en commentaire au début de votre fichier source.

Si vous préférez, vous pouvez nous envoyer une photo de vos brouillons si vous avez bien exercé votre réflexion sur papier (dans un délai **maximum de 30 minutes** après la fin de l'examen).

Respectez les formats d'entrée et de sortie spécifiés en introduction de cet exercice.

Exemples de tests :

- `./ip.x 127.4.5.255` ⟶ `127 4 5 255`↵
- `./ip.x cet-exam-est-pénible` ⟶ `Bad byte 1.`↵
- `./ip.x 127.400.5.255` ⟶ `Bad byte 2.`↵
- `./ip.x 127.4.45.bar` ⟶ `Bad byte 4.`↵
- `./ip.x 127.4.foo.bar` ⟶ `Bad byte 3.`↵

Remarque : La détection d'une chaîne erronée ne doit pas être considérée comme une erreur de votre programme. Il fait correctement son travail s'il détecte correctement qu'une chaîne ne représente pas une adresse IP.

Indication : Souvenez-vous que lorsque l'on doit retourner plusieurs valeurs en C, il est possible d'utiliser le passage d'arguments par adresse.

—— Fin du sujet ——