IFT6800–Atelier en Informatique
(TP #1, Automne 2016)

Louis Salvail[1]

Université de Montréal (DIRO), QC, Canada
salvail@iro.umontreal.ca
Bureau: Pavillon André-Aisenstadt, #3369

1 TP #1

Il s'agit du premier TP pour le cours. La date de remise est:

Vendredi, 14 octobre 2016 avant 12:30.

La remise doit être faite par courriel à l'adresse: `phil.lamontagne@gmail.com`. Vous pouvez faire votre TP en équipe d'au plus 3 participants.

1.1 Le problème

Vous devez fournir un programme X-TOY qui répond au problème suivant.

Votre programme accepte un programme en entrée (qui fonctionne si le programme est logé à partir de l'adresse 10_{16}), le place à partir de l'adresse 0_{16} en le modifiant pour qu'il fasse la même chose que s'il était placé à l'adresse 10_{16} avant de l'exécuter.

Nous supposons que le programme donné en entrée (sur `stdin`) est un programme qui devrait être logé à l'adresse 10_{16}, comme les programmes habituels sur la X-TOY. Nous supposons que le programme termine avant l'adresse 20_{16} (i.e. la dernière instruction est toujours à une adresse qui précède $1F_{16}$, l'adresse maximale qui peut contenir une instruction est donc $1E_{16}$), il contient donc strictement moins de 15 instructions. Nous supposons également que le programme donné en entrée n'écrit jamais et ne charge jamais la mémoire aux adresses qui précèdent l'adresse 50_{16}. Les adresses 50_{16} à FE_{16} contiennent les données du programme qui est fourni sur `stdin`. Votre programme doit placer le programme donné en entrée à partir de l'adresse 0_{16}. La plage mémoire 0_{16} à F_{16} est donc réservée pour contenir le programme donné en entrée. Pour que le programme donné en entrée et placé à l'adresse 0_{16} (au lieu de 10_{16}) puisse se comporter de la même façon que s'il était logé à l'adresse 10_{16}, votre programme doit convertir chaque instruction de branchement (dont la destination est toujours considérée être dans la plage mémoire de 10_{16} à $1E_{16}$) en une instruction de branchement dont la destination est dans la plage 0_{16} à E_{16}.

Une fois que le programme lu sur `stdin` est reformatté (les adresses de branchement sont mises à jour) et placé à partir de l'adresse 0_{16}, vous l'exécutez (en branchant à l'adresse 0_{16}).

Plus précisemment, votre programme (celui que vous devez écrire) est logé à l'adresse 10_{16} et accepte un programme sur `stdin`.

– Le programme est fourni en entrée de la façon suivante:

1. Les instructions sur 16 bits sont données une à la suite de l'autre sur le `stdin`.
2. Finalement, un code spécial sur 16 bits est donné pour indiquer la fin du programme. Ce code est $00FF_{16}$. Un programme ne peut donc pas avoir une instruction $00FF_{16}$ dans son code. Ceci n'est pas une restriction, car $00FF_{16}$ est la même instruction que 0000_{16} qui est la commande `HALT`.

– Votre programme convertit les instructions de branchement du programme donné en entrée (les adresses de branchement sont toujours entre 10_{16} et $1E_{16}$ dans le programme original) en instructions de branchement dont les adresses de destination sont entre 0_{16} à E_{16}. Vous ne convertissez que les branchements C_{16} et D_{16} (i.e. le brachement `if (R[d] == 0) pc <- imm` et le branchement `if (R[d] > 0) pc <- imm`) dans le programme donné sur `stdin`. Ignorez les autres.
– Le programme donné sur `stdin` et dont les instructions de branchement ont été modifiées est placé à partir de l'adresse 0_{16}.
– Le programme dont la première instruction est logée à l'adresse 0_{16} est maintenant exécuté.

Rappel: La première instruction de votre programme (celui que vous écrivez) doit être à l'adresse 10_{16}, car cette adresse est l'adresse standard pour la première instruction des programmes sur la machine X-TOY. C'est la raison pourquoi le programme donné en entrée doit être logé ailleur (à partir de l'adresse 0_{16} dans notre cas) pour faire le même travail.

Bonus (10%). Vous obtiendrai des points bonis si suite à l'exécution du programme donné sur `stdin` vous affichez à l'écran $FFFF_{16}$ pour indiquer la fin de l'exécution à l'usager. Vous pouvez supposer que le dernière instruction du programme donné sur `stdin` est la dernière instruction que le programme exécute lorsqu'il termine (sans inclure le `HALT` final, comme expliqué plus bas). Pour répondre à cette question, il vous suffit donc d'ajouter un branchement juste après la dernière instruction du programme logé à partir de 0_{16}, ce branchement retourne à quelque part dans votre code placé à partir de 10_{16} pour qu'il puisse afficher $FFFF_{16}$. Vous pouvez supposer que la dernière instruction n'est pas une instruction `HALT` (i.e. 0000_{16}), que le programme donné sur `stdin` stoppe lorsqu'il rencontre 0000_{16} à l'adresse qui suit la dernière instruction. Vous pouvez donc supposer que le `HALT` final est exécuté lorsque l'adresse mémoire qui suit la dernière instruction du programme est atteinte et cette adresse est supposée contenir 0000_{16} (i.e. un `HALT`), comme c'est habituellement le cas puisque la mémoire est initialement mise à 0000_{16}. Avec cette supposition, vous pouvez reconnaître la fin du programme en remplaçant l'adresse qui suit la dernière adresse du programme donné sur `stdin` par un branchement vers un endroit de votre programme où l'impression de $FFFF_{16}$ sera exécutée.

1.2 Exemple

Voici un programme qui pourrait être donné en entrée à votre programme selon le bon format. Il s'agit du programme de multiplication vu au cours sans le `HALT` final qui sera de toutes façons exécuté puisque l'adresse qui suit l'instruction $9CFF_{16}$ sera un `HALT` (i.e.

0000_{16}) puisque la mémoire est initialisée à 000_{16} au départ:

```
8AFF
8BFF
7C00
7101
CA18
1CCB
2AA1
C014
9CFF
00FF
```

La dernière instruction du programme est l'instruction 0000_{16} qui est un `HALT`. Le mot $00FF_{16}$ indique la fin des instructions programme, $00FF_{16}$ lui-même n'est pas une instruction du programme. Le programme en tant que tel est donc:

```
8AFF
8BFF
7C00
7101
CA18
1CCB
2AA1
C014
9CFF
```

Votre programme devra maintenant modifier les branchements pour qu'ils soient consistants avec l'emplacement du programme à partir de l'adresse 0_{16}. Il faut donc modifier les instructions $CA18_{16}$ et $C014_{16}$ (qui sont les seuls branchements du programme) pour qu'elles branchent à la 19ième et la à la 5ième instructions du programme une fois placée à partir l'adresse 0_{16}. Pour ce faire, il suffit de soustraire 10_{16} aux instructions $CA18$ et $C014_{16}$: $CA18_{16} - 10_{16} = CA08_{16}$ et $C014_{16} - 10_{16} = C004_{16}$ respectivement. Un branchement à l'adresse 14_{16} lorsque le programme est logé à partir de l'adresse 10_{16} correspond à un branchement à l'adresse 04_{16} lorsque le programme est logé à partir de l'adresse 0_{16}. Le programme ainsi modifié est:

```
8AFF
8BFF
7C00
7101
CA08
1CCB
2AA1
C004
9CFF
```

Votre programme peut maintenant être placé à partir de l'adresse 0_{16}:

```
00 : 8AFF
01 : 8BFF
02 : 7C00
03 : 7101
04 : CA08
05 : 1CCB
06 : 2AA1
07 : C004
08 : 9CFF
```

Le programme résultant branche maintenant aux adresses 08_{16} et 04_{16} pour le premier et deuxième branchement. Il fera alors le travail, c'est-à-dire qu'il multipliera les deux mots donnés en entrée sur `stdin` de la même façon que le programme original (celui donné sur `stdin`) le ferait s'il était logé à l'adresse 10_{16}.