Hack Computer

Nous allons présenter un ordinateur simple, *The Hack Computer*, tiré du livre *The Elements of Computing Systems*.
Une machine peut être décrite à partir de ses composants élémentaires, en expliquant comment elle est construite à partir de ceux-ci. Elle peut aussi être décrite de manière plus abstraite par ce qu'on appelle son modèle de programmation qui consiste en :

- une collection de registres ;
- un ensemble d'instructions ;
- les spécifications de l'espace d'adressage (mémoire, entrées/sorties).

Nous allons tout d'abord présenter le modèle de programmation de l'ordinateur *Hack*, puis une vue d'ensemble de l'architecture de cette machine que vous devrez réaliser dans la section exercices.

1 Modèle de programmation

La machine *Hack* est une machine basée sur le modèle de *von Neumann*. C'est une machine consituée d'une unité centrale, de deux modules de mémoire séparés – un pour les instructions et un pour les données –, d'un écran et d'un clavier.

- **Espace d'adressage.** L'ordinateur *Hack* possède deux espaces d'adressage séparés : une mémoire pour les instructions et une mémoire pour les données. La mémoire contenant les instructions est une mémoire morte (on peut l'assimiler à une cartouche dans une console de jeux). Chacune de ces mémoires contient $32K$ mots de 16 bits.

- **Entrées/Sorties.** Les données associés à l'écran et au clavier sont contenues dans la mémoire des données. Pour écrire à l'écran, il faut donc modifier certains mots contenus à certaines adresses, et pour lire la dernière touche pressée par l'utilisateur, il faut lire un certain mot en mémoire. Nous ne nous soucierons pas des entrées/sorties dans la suite de ce chapitre.

- **Registres.** Dans l'unité centrale de la machine *Hack*, deux registres, appelés `A` et `D`, sont mis à la disposition du programmeur. Ces registres peuvent être manipulés explicitement par des opérations arithmétiques et logiques comme `A=D-1` ou `D=¬A` ("¬" représente le "non" bit à bit). Tandis que le registre D est utilisé seulement pour stocker des données, le registre `A` sert aussi de registre d'adresse. En fonction du contexte d'utilisation, le contenu de `A` sera interprété comme une donnée, ou comme une adresse dans la mémoire des données ou dans la mémoire des instructions. En effet, les instructions de la machine *Hack* sont codées sur 16 bits et les adresses sont sur 15 bits. Il est donc impossible de mettre dans une même instruction le code

de celle-ci et l'adresse. Les instructions accédant à la mémoire s'effectuent en deux étapes. Par exemple, pour exécuter l'instruction `D=Mémoire[50]-1`, on chargera d'abord la valeur 50 dans le registre `A`, puis on exécutera l'instruction assembleur `D=Mémoire[A]-1`. Les deux instructions en assembleur *Hack* correspondant à ces deux opérations sont notées :

```
@50
D=M-1
```

l'instruction `@50` ayant pour signification de charger la valeur 50 dans le registre `A`. Le registre `A` est aussi utilisé pour faciliter l'accès à la mémoire des instructions. De la même manière que pour accéder à la mémoire de données, l'instruction `jump` de la machine *Hack* ne spécifie pas l'adresse de saut. Là encore, c'est le registre `A` qui contient cette adresse.

Nous allons maintenant décrire les deux types d'instruction de la machine *Hack*.

1.1 Instruction de type A

L'instruction de type A est utilisée pour mettre à jour le contenu du registre `A` avec une donnée de 15 bits. En assembleur *Hack*, cette instruction est notée :

```
@valeur
```

où `valeur` est une chaîne de 15 bits. En language machine, cette instruction est codée :

0	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}

chaque v_i peut prendre la valeur 0 ou 1.

L'instruction de type A a trois usages différents :
- entrer une constante dans l'ordinateur ;
- spécifier l'adresse d'une donnée en mémoire ;
- spécifier l'adresse de saut dans la mémoire des instructions.

1.2 Instruction de type C

Une instruction de type C spécifie :
- quoi calculer ;
- où stocker le résultat du calcul ;
- quelle instruction exécuter après.

En assembleur *Hack*, cette instruction est notée :

```
dest=comp;jump // les champs "dest" ou "jump"
               // peuvent être vides
               // si "dest" est vide, le "=" peut être omis
               // si "jump" est vide, le ";" peut être omis
```

En language machine, cette instruction est codée :

1	?	?	a	c_1	c_2	c_3	c_4	c_5	c_6	d_1	d_2	d_3	j_1	j_2	j_3

Le bit le plus à gauche est le code de l'instruction de type C qui est 1. Les deux bits suivants ne sont pas utilisés et leur valeur n'est pas importante. Les bits qui suivent correspondent aux trois parties de l'instruction assembleur. La sémantique générale de l'instruction assembleur est la suivante :

- le champ `comp` (correspondant à la valeur a et aux valeurs c_i) instruit l'ALU du calcul à effectuer ;
- le champ `dest` (correspondant aux valeurs d_i) indique où stocker la valeur calculée par l'ALU ;
- le champ `jump` (correspondant aux valeurs j_i) indique une condition de saut pour savoir quelle instruction exécuter après celle-ci.

Avant de détailler chaque partie de l'instruction de type C, nous donnons dans la table 1 un programme en assembleur *Hack* permettant de calculer la somme des nombres de 1 à 100.

```
     @i     // i représente une adresse en mémoire
     M=1    // i=1
     @sum   // sum représente une adresse en mémoire
     M=0    // sum=0
(LOOP)
     @i
     D=M    // D=i
     @100
     D=D-A  // D=i-100
     @END
     D;JGT  // si (i-100)>0 aller en END
     @i
     D=M    // D=i
     @sum
     M=D+M  // sum=sum+i
     @i
     M=M+1  // i=i+1
     @LOOP
     0;JMP  // aller en LOOP
(END)
```

TABLE 1 – Programme en assembleur *Hack* permettant de calculer la somme de 1 à 100

1.2.1 Partie calcul de l'instruction

Afin de comprendre les spécifications de la partie `comp` d'une instruction de type C, nous présentons l'unité arithmétique et logique de la machine *Hack* à la figure 1. Les spécifications de cette UAL sont données dans la table 2 et le fonctionnement interne et décrit par l'algorithme 1.

La table 3 donne la signification des différentes valeurs autorisées pour a et les c_i. Par exemple :

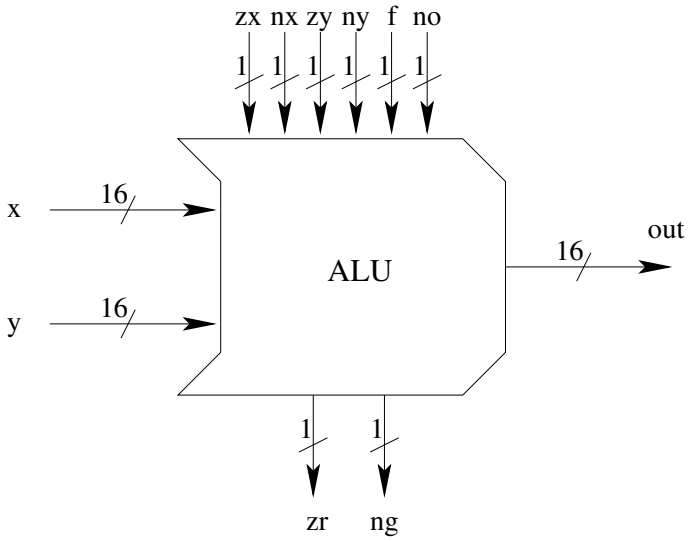

FIGURE 1 – L'unité arithmétique et logique de la machine *Hack*

zx	nx	zy	ny	f	no	out
1	0	1	0	1	0	0
1	1	1	1	1	1	1
1	1	1	0	1	0	-1
0	0	1	1	0	0	x
1	1	0	0	0	0	y
0	0	1	1	0	1	$\neg x$
1	1	0	0	0	1	$\neg y$
0	0	1	1	1	1	$-x$
1	1	0	0	1	1	$-y$
0	1	1	1	1	1	$x+1$
1	1	0	1	1	1	$y+1$
0	0	1	1	1	0	$x-1$
1	1	0	0	1	0	$y-1$
0	0	0	0	1	0	$x+y$
0	1	0	0	1	1	$x-y$
0	0	0	1	1	1	$y-x$
0	0	0	0	0	0	$x \wedge y$
0	1	0	1	0	1	$x \vee y$

TABLE 2 – Table de vérité de l'unité arithmétique et logique

Algorithme 1 : Spécification de l'unité arithmétique et logique de la machine *Hack*

si $zx = 1$ **alors**
  $x \leftarrow 0$
si $nx = 1$ **alors**
  $x \leftarrow \neg x$
si $zy = 1$ **alors**
  $y \leftarrow 0$
si $ny = 1$ **alors**
  $y \leftarrow \neg y$
si $f = 1$ **alors**
  $out \leftarrow x + y$
sinon
  $out \leftarrow x \wedge y$
si $no = 1$ **alors**
  $out \leftarrow \neg out$
si $out = 0$ **alors**
  $zr \leftarrow 1$
sinon
  $zr \leftarrow 0$
si $out < 0$ **alors**
  $ng \leftarrow 1$
sinon
  $ng \leftarrow 0$

- à la ligne 4, lorsque $a = 0$, la valeur calculée par l'instruction est le contenu du registre `D` ;
- à la ligne 5, lorsque $a = 1$, la valeur calculée par l'instruction est le contenu de `Mémoire[A]` ;
- à la ligne 14, lorsque $a = 1$, la valeur calculée par l'instruction est le contenu de `Mémoire[A]` plus le contenu de `D`.

1.2.2 Partie destination de l'instruction

La valeur calculée par la partie *comp* de l'instruction de type C peut être stockée dans différentes destinations spécifiées par les 3 bits de la partie *dest* de l'instruction. La table 4 donne la signification des différentes combinaisons des d_i. Par exemple, la valeur `011` pour les bits d_1, d_2 et d_3 a pour effet de mettre le résultat du calcul dans le registre `D` et `Mémoire[A]`.

1.2.3 Partie saut de l'instruction

Le champ *jump* de l'instruction de type C indique à l'ordinateur quoi faire après l'exécution de l'instruction courante. L'ordinateur peut chercher et exécuter l'instruction suivante (ce qu'il fait par défaut) ou aller chercher et exécuter une instruction se trouvant à un autre endroit dans le programme. Dans ce dernier cas, le registre `A` contient l'adresse, dans la *ROM*, où chercher la nouvelle instruction. Le saut va dépendre de la valeur des j_i bits de l'instruction et de la sortie de l'*UAL*. La table 5 donne la signification des différentes combinaisons des j_i. Par exemple :

Partie *comp* de l'instruction de type C							
$a = 0$ (mnémonique)	c_1	c_2	c_3	c_4	c_5	c_6	$a = 1$ (mnémonique)
0	1	0	1	0	1	0	
1	1	1	1	1	1	1	
-1	1	1	1	0	1	0	
D	0	0	1	1	0	0	
A	1	1	0	0	0	0	M
$\neg D$	0	0	1	1	0	1	
$\neg A$	1	1	0	0	0	1	$\neg M$
$-D$	0	0	1	1	1	1	
$-A$	1	1	0	0	1	1	$-M$
$D + 1$	0	1	1	1	1	1	
$A + 1$	1	1	0	1	1	1	$M + 1$
$D - 1$	0	0	1	1	1	0	
$A - 1$	1	1	0	0	1	0	$M - 1$
$D + A$	0	0	0	0	1	0	$D + M$
$D - A$	0	1	0	0	1	1	$D - M$
$A - D$	0	0	0	1	1	1	$M - D$
$D \wedge A$	0	0	0	0	0	0	$D \wedge M$
$D \vee A$	0	1	0	1	0	1	$D \vee M$

TABLE 3 – Spécification de la partie *comp* d'une instruction de type C

Partie *dest* de l'instruction de type C				
d_1	d_2	d_3	mnémonique	destination
0	0	0	*null*	la valeur n'est pas stockée
0	0	1	M	Mémoire[A]
0	1	0	D	le registre D
0	1	1	MD	Mémoire[A] et registre D
1	0	0	A	registre A
1	0	1	AM	registre A et Mémoire[A]
1	1	0	AD	registres A et D
1	1	1	AMD	registres A et D et Mémoire[A]

TABLE 4 – Spécification de la partie *dest* d'une instruction de type C

- si j_1, j_2 et j_3 valent `001`, la prochaine instruction sera celle dont l'adresse est dans le registre `A` si la sortie de l'*UAL* est strictement positive ;
- si j_1, j_2 et j_3 valent `000`, la prochaine instruction sera celle dont l'adresse est la valeur du compteur ordinal plus un.

Partie *jump* de l'instruction de type C				
j_1 ($out < 0$)	j_2 ($out = 0$)	j_3 ($out > 0$)	mnémonique	effet
0	0	0	*null*	pas de saut
0	0	1	JGT	saut si $out > 0$
0	1	0	JEQ	saut si $out = 0$
0	1	1	JGE	saut si $out \geq 0$
1	0	0	JLT	saut si $out < 0$
1	0	1	JNE	saut si $out \neq 0$
1	1	0	JLE	saut si $out \leq 0$
1	1	1	JMP	saut inconditionnel

TABLE 5 – Spécification de la partie *jump* d'une instruction de type C

2 Architecture

L'architecture générale de la machine *Hack* est décrite dans la figure 2. La mémoire des données contient des mots de 16 bits et ne charge au front montant de l'horloge que si `load` est à `1`. L'unité centrale est à construire dans l'exercice de la section 3.3. La mémoire des instructions est une mémoire morte qui présente en sortie le contenu de l'adresse spécifiée en entrée. Enfin, l'unité centrale et la mémoire des données sont reliées à la même horloge.

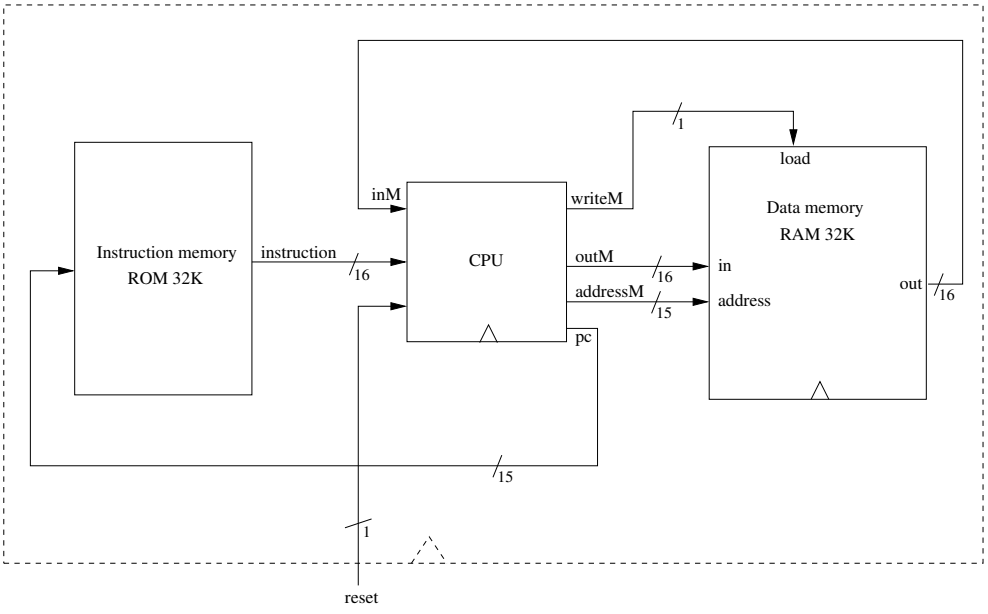

FIGURE 2 – Architecture de la machine *Hack*

3 Exercices

3.1 Traduction de l'assembleur *Hack* vers le langage machine

Traduire en langage machine les 11 premières lignes du programme de la table 1.

3.2 Unité Arithmétique et Logique

Réaliser l'unité arithmétique et logique de la machine *Hack*.

3.3 Unité centrale

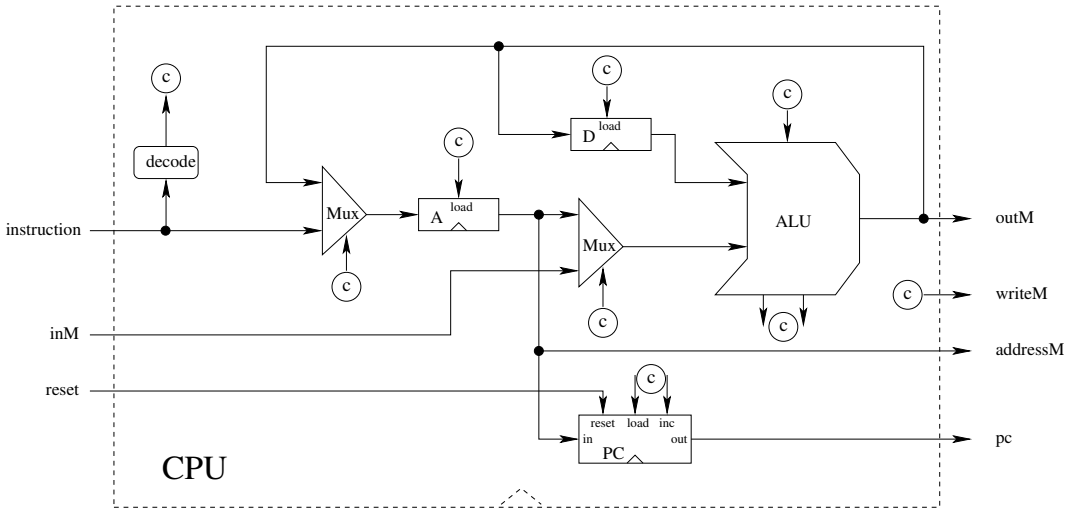

FIGURE 3 – Unité centrale de la machine *Hack*

La figure 3 donne une ébauche de l'unité centrale de la machine *Hack*. Les c entourés d'un cercle représentent le contrôle qu'il vous reste à câbler en fonction du langage machine. Sur cette figure, on peut voir le compteur ordinal (PC) dont le fonctionnement est décrit par l'algorithme 2.

Algorithme 2 : Spécification du compteur ordinal

si $reset(t-1)$ **alors**
  $out(t) = 0$
sinon
  si $load(t-1)$ **alors**
    $out(t) = in(t-1)$
  sinon
    si $inc(t-1)$ **alors**
      $out(t) = out(t-1) + 1$
    sinon
      $out(t) = out(t-1)$