Université de Bordeaux — U.F. Informatique
Licence Informatique 2ème année — Année 2017 – 2018

ARCHITECTURE DES ORDINATEURS

Devoir Surveillé 1

1 heure 20 minutes
sans documents

N.B. : - Travaillez calmement. Ne bâclez pas vos réponses : il vaut mieux traiter correctement moins de questions que tout faire de travers.
- Les réponses aux questions doivent être argumentées et aussi concises que possible.
- Le barème est donné à titre indicatif.

Question 1 (5 points)

On considère des entiers codés sur 16 bits, interprétés comme des entiers relatifs codés en complément à deux. Chacun de ces entiers peut (et doit) s'écrire avec quatre chiffres hexadécimaux (on omettra le préfixe « 0x »).

Soient les cinq entiers suivants :

M	N	P	Q	R
4732	B0B0	ABAF	1789	8765

(1.1) (1 point)
Parmi ces 5 entiers, quels sont ceux qui sont négatifs ? Expliquez pourquoi.

(1.2) (1 point)
Classez ces entiers du plus petit au plus grand (par exemple : « M < N < P < Q < R », mais cette réponse choisie arbitrairement n'est probablement pas la bonne...). Justifiez la réponse sur votre copie.

(1.3) (1 point)
Calculez $N + Q$. Pour cela, effectuez l'addition en binaire directement sur la copie — une réponse brute ne rapportera aucun point. Y a-t-il débordement (*overflow*) ?

(1.4) (2 points)
Calculez $-N$ et $M - N$. Y a-t-il débordement (*overflow*) ? Comme pour la question précédente, le détail des calculs doit figurer sur la copie.

Question 2 (6 points)

Les ordinateurs ne savent manipuler que des nombres binaires, alors que les humains expriment les nombres sous forme décimale. Lors d'un calcul interactif avec un ordinateur, il faut donc convertir les nombres décimaux en nombres binaires puis, à la fin du calcul binaire, convertir le résultat en décimal, ce qui est très coûteux.

Pour éviter cela, certains processeurs, utilisés par exemple dans les calculatrices, disposent d'instructions pour travailler directement avec des nombres entiers en base 10, en utilisant une représentation appelée « BCD », pour « *Binary-Coded Decimal* », soit en français « *Décimal Codé en Binaire* ».

L'idée du codage BCD est de coder chaque chiffre décimal sous forme binaire, et de manipuler directement ce codage en machine en modifiant les circuits de calcul pour que leur résultat puisse se lire directement sous forme décimale. Quatre bits sont nécessaires au stockage des chiffres de 0 à 9. On peut donc représenter en BCD le nombre décimal $62_{(10)}$ sur un octet, sous la forme 0110 0010 $= 62_{(16)}$, alors que normalement $62_{(10)} = 3D_{(16)}$, et faire en sorte que, par exemple, $62_{(16)} + 19_{(16)} = 81_{(16)}$, comme en décimal, alors qu'en binaire pur $62_{(16)} + 19_{(16)} = 7B_{(16)}$.

Pour effectuer cet exercice, vous disposez, comme brique de base, d'un additionneur binaire classique à 4 bits. Cet additionneur, appelé `ADD`, possède deux entrées $\mathtt{A} = \mathtt{a_3a_2a_1a_0}$ et $\mathtt{B} = \mathtt{b_3b_2b_1b_0}$ sur 4 bits chacune, codant les deux chiffres hexadécimaux à additionner, plus une entrée de retenue $\mathtt{c_{in}}$ sur 1 bit, et une sortie $\mathtt{S} = \mathtt{s_3s_2s_1s_0}$ sur 4 bits, codant le chiffre hexadécimal de résultat, plus une retenue de sortie $\mathtt{c_{out}}$ sur 1 bit. Le bit 0 est toujours le bit de poids le plus faible.

Vous disposez également d'un multiplexeur `MUX` à deux entrées `A` et `B` sur 4 bits chacune, une entrée de commande `c` sur 1 bit permettant de choisir entre ces deux entrées, et une sortie `X` sur 4 bits renvoyant la valeur de l'entrée sélectionnée : $\mathtt{X} = \mathtt{A}$ si $\mathtt{c} = 0$ et $\mathtt{X} = \mathtt{B}$ si $\mathtt{c} = 1$.

Dans toutes les questions suivantes, vous pourrez utiliser ces deux circuits, ainsi que les portes logiques classiques `AND`, `OR`, `NOT`, `NAND`, etc.

(2.1) (1 point)

Au moyen d'additionneurs `ADD`, créez un additionneur binaire pur permettant d'additionner deux nombres $\mathtt{X} = \mathtt{x_7x_6 \ldots x_1x_0}$ et $\mathtt{Y} = \mathtt{y_7y_6 \ldots y_1y_0}$ sur 8 bits en leur somme $\mathtt{Z} = \mathtt{z_7z_6 \ldots z_1z_0}$ sur 8 bits également.

(2.2) (1 point)

Dans l'exemple ci-dessus, où $62_{(16)} + 19_{(16)} = 81_{(16)}$, alors que normalement $62_{(16)} + 19_{(16)} = 7B_{(16)}$, de combien les deux résultats diffèrent-ils ? Pourquoi ?

(2.3) (1 point)

De même, que vaut $62_{(16)} + 17_{(16)}$ en binaire pur ? Combien cela vaut-il en BCD ? De combien les deux résultats diffèrent-ils ? Pourquoi ?

(2.4) (1 point)

Donnez la formule logique d'un circuit `TEST` possédant une entrée $\mathtt{A} = \mathtt{a_3a_2a_1a_0}$ sur 4 bits et une sortie `r` sur 1 bit, tel que $\mathtt{r} = 1$ si $\mathtt{A} \geq 10_{(10)}$. Pour ce faire, considérez le codage binaire de toutes les valeurs d'entrée telles que $\mathtt{r} = 1$ et déduisez-en une formule logique simplifiée.

(2.5) (1 point)

Créez un circuit `PLUS`, possédant une entrée `A` sur 4 bits et une entrée `p` sur 1 bit, et une sortie `X` sur 4 bits, complétée d'une sortie de retenue $\mathtt{c_{out}}$ sur 1 bit, tel que $\mathtt{X} = \mathtt{A}$ si $\mathtt{p} = 0$ et $\mathtt{X} = \mathtt{A} + 6$ si $\mathtt{p} = 1$ (avec possible retenue).

(2.6) (1 point)

En combinant les circuits précédents, construisez un circuit additionneur BCD sur 8 bits.

Question 3 (9 points)

Voici le texte d'un programme écrit en langage y86. Tous les commentaires ayant malencontreusement été effacés, il vous faut partir de zéro pour comprendre ce que fait ce programme, en répondant aux questions ci-dessous.

Attention : les réponses qui, au lieu d'expliquer, paraphrasent simplement le code (telles que « on soustrait %edx à %ebx ») ne rapporteront aucun point.

```
0x000:                | .pos 0
0x000: 308400020000   | main:     irmovl pile,%esp
0x006: 308000010000   |           irmovl m,%eax
0x00c: a008           |           pushl  %eax
0x00e: 308004000000   |           irmovl 4,%eax
0x014: a008           |           pushl  %eax
0x016: 308008010000   |           irmovl t,%eax
0x01c: a008           |           pushl  %eax
0x01e: 802a000000     |           call   mystere
0x023: c0840c000000   |           iaddl  12,%esp
0x029: 10             |           halt
                      |
0x02a: 501408000000   | mystere:  mrmovl 8(%esp),%ecx
0x030: 6211           |           andl   %ecx,%ecx
0x032: 73a7000000     |           je     et4
0x037: a038           |           pushl  %ebx
0x039: a068           |           pushl  %esi
0x03b: a078           |           pushl  %edi
0x03d: 507418000000   |           mrmovl 24(%esp),%edi
0x043: 506410000000   |           mrmovl 16(%esp),%esi
0x049: 500600000000   |           mrmovl (%esi),%eax
0x04f: 400700000000   |           rmmovl %eax,(%edi)
0x055: 400704000000   |           rmmovl %eax,4(%edi)
0x05b: c08604000000   | et1:      iaddl  4,%esi
0x061: c18101000000   |           isubl  1,%ecx
0x067: 73a1000000     |           je     et3
0x06c: 500600000000   |           mrmovl (%esi),%eax
0x072: 2002           |           rrmovl %eax,%edx
0x074: 503700000000   |           mrmovl (%edi),%ebx
0x07a: 6103           |           subl   %eax,%ebx
0x07c: 7187000000     |           jle    et2
0x081: 402700000000   |           rmmovl %edx,(%edi)
0x087: 2020           | et2:      rrmovl %edx,%eax
0x089: 503704000000   |           mrmovl 4(%edi),%ebx
0x08f: 6103           |           subl   %eax,%ebx
0x091: 755b000000     |           jge    et1
0x096: 402704000000   |           rmmovl %edx,4(%edi)
0x09c: 705b000000     |           jmp    et1
0x0a1: b078           | et3:      popl   %edi
0x0a3: b068           |           popl   %esi
0x0a5: b038           |           popl   %ebx
0x0a7: 90             | et4:      ret
                      |
0x100:                |           .pos   0x100
0x100: 00000000       | m:        .long  0
0x104: 00000000       | n:        .long  0
0x108: 01000000       | t:        .long  1
0x10c: fdffffff       |           .long  -3
0x110: 02000000       |           .long  2
0x114: fbffffff       |           .long  -5
                      |
0x200:                |           .pos   0x200
0x200: 00000000       | pile:     .long  0
```

(3.1) (1 point)

Quels sont les paramètres de la fonction `mystere` ? Expliquer quelle partie du code permet de répondre à cette question.

(3.2) (1 point)

Quel est le rôle de l'instruction d'adresse `0x030` ?

(3.3) (1 point)

Pourquoi la fonction `mystere` a-t-elle besoin des instructions situées en mémoire aux adresses allant de `0x037` à `0x03c` ?

(3.4) (4 points)

Que calcule la fonction `mystere`, et que renvoie-t-elle ? Pour cela :

- Expliquez le rôle de chacun des registres utilisés au sein de cette fonction et leur évolution au cours du déroulement de la fonction ;
- Expliquez le rôle des étiquettes `et1`, `et2`, `et3` et `et4`.

(3.5) (1 point)

Si ce code avait été produit par un compilateur C, quel aurait été le prototype de la fonction `mystere` ?

(3.6) (1 point)

Quelle est la valeur contenue à l'adresse `0x104` à la fin du programme ?