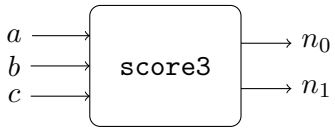
Université Bordeaux 1 **INF155, jeudi 17 mars 2011**
Licence de Sciences et Technologie **Architecture des ordinateurs**

Devoir surveillé numéro 1

Exercice 1

On souhaite construire un circuit combinatoire à trois entrées (de 1 bit chacune) et une sortie (codée sur 2 bits) réalisant la fonction *score* sur 3 bits (notée `score3` sur le schéma), qui est le nombre d'entrées valant 1. Par exemple, si $a = c = 1$ et $b = 0$, il y a 2 entrées qui valent 1, et donc $n_1 = 1$ et $n_0 = 0$ (représentation binaire de l'entier 2).



1. Donnez la table de vérité de la fonction *score*. A quel circuit connu cela vous fait-il penser ?
2. Proposez une implémentation du circuit `score3` à l'aide de portes logiques simples.
3. Expliquez comment combiner deux circuits `score3` et un additionneur pour obtenir un circuit `score7`.

Exercice 2 On rappelle que la division euclidienne de l'entier a (dividende) par l'entier $b > 0$ (diviseur) est définie par $a = bq + r$, avec $0 \leq r < b$; par exemple $22 = 7 * 3 + 1$, donc le quotient de 22 par 7 vaut 3, et le reste de la division vaut 1. Comme le résultat est constitué de deux entiers (le quotient q et le reste r), une fonction C qui effectue ce calcul a pour déclaration :

```
int quo (int a, int b, int *r);
```

elle retourne le quotient q, et place le reste dans la variable r ; le programme ci-dessous propose une réalisation de cette fonction en assembleur y86, et un test qui divise 22 par 7 :

```
0x000: 308400020000 | test:   irmovl  0x200,%esp
0x006: 2045         |         rrmovl  %esp,%ebp
0x008: 30805c000000 |         irmovl  r,%eax
0x00e: a008         |         pushl   %eax
0x010: 308007000000 |         irmovl  7,%eax
0x016: a008         |         pushl   %eax
0x018: 308016000000 |         irmovl  22,%eax
0x01e: a008         |         pushl   %eax
0x020: 8028000000   |         call    quo
0x025: 2054         |         rrmovl  %ebp,%esp
0x027: 10           |         halt
                    |
0x028: a058         | quo:    pushl   %ebp
0x02a: 2045         |         rrmovl  %esp,%ebp
0x02c: 501508000000 |         mrmovl  8(%ebp),%ecx
0x032: 50250c000000 |         mrmovl  12(%ebp),%edx
0x038: 3080ffffffff |         irmovl  -1,%eax
0x03e: c08001000000 | qb:     iaddl   1,%eax
0x044: 6121         |         subl    %edx,%ecx
0x046: 753e000000   |         jge     qb
0x04b: 6021         |         addl    %edx,%ecx
0x04d: 502510000000 |         mrmovl  16(%ebp),%edx
0x053: 401200000000 |         rmmovl  %ecx,(%edx)
0x059: b058         |         popl    %ebp
0x05b: 90           |         ret
0x05c:              | .align 4
0x05c:              | r:
```

Note : cette réalisation de la fonction quo est très sommaire : elle est terriblement inefficace lorsque le quotient est grand, et incorrecte lorsque le dividende a est négatif.

On commence l'exécution du programme avec le simulateur, jusqu'à l'appel de la fonction `quo`, on exécute les deux premières instructions de cette fonction, et à cet instant on prend un cliché :

```
0x28  a058                  quo:    pushl   %ebp
0x2a  2045                          rrmovl  %esp,%ebp
0x2c  501508000000  *        *      mrmovl  8(%ebp),%ecx
0x32  50250c000000                  mrmovl  12(%ebp),%edx
0x38  3080ffffffff                  irmovl  -1,%eax
```

PC: 0000002C

Register File

%eax	%ecx	%edx	%ebx	%esp	%ebp	%esi	%edi
16				lec	lec		

Memory Contents

	0x---0	0x---4	0x---8	0x---c
0x01f-	25	16	7	5c
0x01e-	0	0	0	200

1. Expliquer *en détail* le contenu de la pile. *Attention* : la note attribuée à cette question dépendra beaucoup de la clarté et de la précision des explications ; un schéma sera le bienvenu.
2. Expliquer le codage (`3080ffffffff`) de l'instruction d'adresse 0x38.
3. Poursuivre l'exécution du programme, en indiquant en décimal dans un tableau les valeurs successives prises par les trois registres `%eax`, `%ecx` et `%edx`.

Voici un cliché pris pendant l'exécution du programme :

```
0x3e  c08001000000  qb:     iaddl   1,%eax
0x44  6121                  subl    %edx,%ecx
0x46  753e000000    *       jge     qb
0x4b  6021                  addl    %edx,%ecx
0x4d  502510000000          mrmovl  16(%ebp),%edx
0x53  401200000000          rmmovl  %ecx,(%edx)
```

PC: 00000046

Register File

%eax	%ecx	%edx	%ebx	%esp	%ebp	%esi	%edi
3	fffffffa	7		lec	lec		

Status: AOK Condition Codes: Z 0 S 1 O 0

4. Le saut vers l'étiquette `qb` aura-t-il lieu ? Pourquoi ? Combien vaut l'entier contenu dans le registre `%ecx` ?
5. Expliquer le rôle des instructions d'adresses 0x04d et 0x053, en sortie de boucle, et quel est le résultat de leur exécution.

On prend un nouveau cliché du contenu (partiel) de la mémoire à la fin de l'exécution du programme :

Memory Contents

	0x---0	0x---4	0x---8	0x---c
0x007-	0	0	0	0
0x006-	0	0	0	0
0x005-	40000000	12	9058b000	1

6. Où se trouve le reste de la division de 22 par 7 ? Pourquoi ? Pouvez-vous expliquer, au moins sommairement, les autres valeurs visibles sur la dernière ligne de ce cliché ?

On souhaite modifier le programme `test` de la façon suivante : on range dans un tableau u une suite de *couples* d'entiers (a_i, b_i), avec $b_i > 0$, terminée par le couple $(0, 0)$, et le programme doit construire un tableau v constitué des couples (q_i, r_i) (quotient et reste de la division de a_i par b_i). Par exemple, avec le tableau u codé comme suit (colonne de gauche), le cliché de la colonne de droite représente le contenu de la mémoire à la fin de l'exécution du programme.

```
.pos 0x100
u:      .long   22
        .long   7

        .long   13
        .long   5

        .long   75
        .long   16

        .long   0
        .long   0
v:
```

Memory Contents

	0x---0	0x---4	0x---8	0x---c
0x01f-	3e	4b	10	134
0x01e-	0	0	0	200
0x01d-	0	0	0	0
0x01c-	0	0	0	0
0x01b-	0	0	0	0
0x01a-	0	0	0	0
0x019-	0	0	0	0
0x018-	0	0	0	0
0x017-	0	0	0	0
0x016-	0	0	0	0
0x015-	0	0	0	0
0x014-	0	0	0	0
0x013-	4	b	0	0
0x012-	3	1	2	3
0x011-	4b	10	0	0
0x010-	16	7	d	5

7. Ecrire cette nouvelle version du programme `test`. *Attention* : inutile de gribouiller des instructions illisibles et décousues, qui ne vous rapporteront aucun point ; commencez par écrire simplement la boucle qui permet de parcourir le tableau u, puis si vous avez le temps complétez progressivement le programme ; remplacez les portions de programme que vous n'avez pas le temps d'écrire par de courts commentaires décrivant le travail qui reste à faire.