

## Devoir surveillé numéro 1

**Exercice 1** Un *encodeur avec priorité*  $3 \times 2$  est un circuit avec trois entrées  $a_1, a_2, a_3$  et deux sorties  $x_1, x_0$  qui codent en binaire le numéro de la *dernière* entrée non nulle. Par exemple si  $a_1 = a_2 = 1$  et  $a_3 = 0$ , le numéro de la dernière entrée non nulle est 2, et  $x_1x_0 = 10$ ; si les trois entrées sont nulles, les sorties aussi.

Ecrire des formules booléennes aussi simples que possible qui expriment les sorties en fonction des entrées, puis dessiner un circuit qui réalise cet encodeur. *Attention* : les formules doivent être justifiées sur la copie, et le dessin doit être soigneux.

**Exercice 2** Cet exercice propose l'analyse progressive du programme `ds1.yo` situé au verso de cette feuille. Pas de panique, inutile de commencer par lire *en détail* ce programme, et la plupart des questions sont indépendantes.

La fonction *map* est une fonction à deux arguments  $t$  et  $n$ , qui applique  $f$  à chacun des  $n$  éléments d'un tableau  $t$ ; autrement dit  $t_1, t_2, \dots, t_n$  sont remplacés par  $f(t_1), f(t_2), \dots, f(t_n)$ .

Pour que le programme et les questions figurent sur des feuilles séparées, les questions commencent page 3. *Attention* : les réponses qui n'apportent aucune information nouvelle par rapport à ce qui est déjà écrit noir sur blanc dans le programme ne rapportent aucun point; par exemple si la question est "à quoi sert l'instruction `pushl %eax`?", la réponse "cette instruction sert à empiler le registre `eax`" *n'est pas* une bonne réponse. . .

Programme ds1.yo

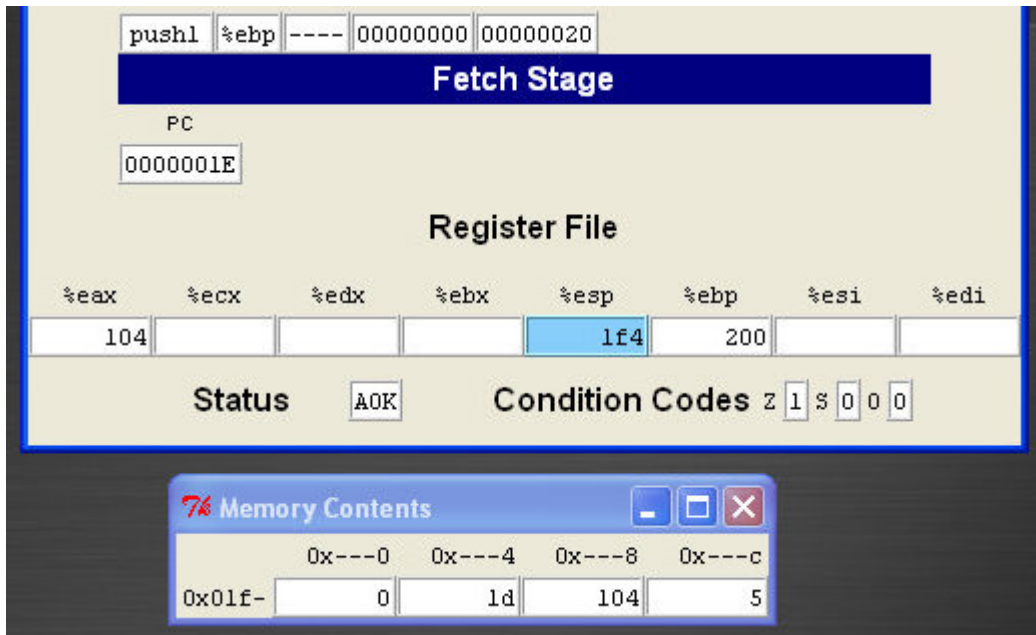
0x000:	308400020000		main:	irmovl	0x200,%esp
0x006:	2045			rrmovl	%esp,%ebp
0x008:	500800010000			mrmovl	p,%eax
0x00e:	a008			pushl	%eax
0x010:	308004010000			irmovl	u,%eax
0x016:	a008			pushl	%eax
0x018:	801e000000			call	map
0x01d:	10			halt	
0x01e:	a058		map:	pushl	%ebp
0x020:	2045			rrmovl	%esp,%ebp
0x022:	50150c000000			mrmovl	12(%ebp),%ecx
0x028:	6211			andl	%ecx,%ecx
0x02a:	716c000000			jle	L3
0x02f:	a038			pushl	%ebx
0x031:	503508000000			mrmovl	8(%ebp),%ebx
0x037:	502300000000		L1:	mrmovl	(%ebx),%edx
0x03d:	a018			pushl	%ecx
0x03f:	a028			pushl	%edx
0x041:	8071000000			call	f
0x046:	b028			popl	%edx
0x048:	b018			popl	%ecx
0x04a:	400300000000			rmmovl	%eax,(%ebx)
0x050:	308201000000			irmovl	1,%edx
0x056:	6121			subl	%edx,%ecx
0x058:	736a000000			je	L2
0x05d:	308204000000			irmovl	4,%edx
0x063:	6023			addl	%edx,%ebx
0x065:	7037000000			jmp	L1
0x06a:	b038		L2:	popl	%ebx
0x06c:	2045		L3:	rrmovl	%ebp,%esp
0x06e:	b058			popl	%ebp
0x070:	90			ret	
0x071:	500404000000		f:	mrmovl	4(%esp),%eax
0x077:	2001			rrmovl	%eax,%ecx
0x079:	6011			addl	%ecx,%ecx
0x07b:	6011			addl	%ecx,%ecx
0x07d:	6010			addl	%ecx,%eax
0x07f:	90			ret	
0x100:			.pos	0x100	
0x100:	05000000		p:	.long	5
0x104:	03000000		u:	.long	3
0x108:	04000000			.long	4
0x10c:	08000000			.long	8
0x110:	0e000000			.long	14
0x114:	2a000000			.long	42

Questions.

1. Dans le programme `ds1.yo` le code de la fonction  $f$  commence à l'adresse `0x071` ; que calcule-t-elle, autrement dit que vaut  $f(x)$  ?

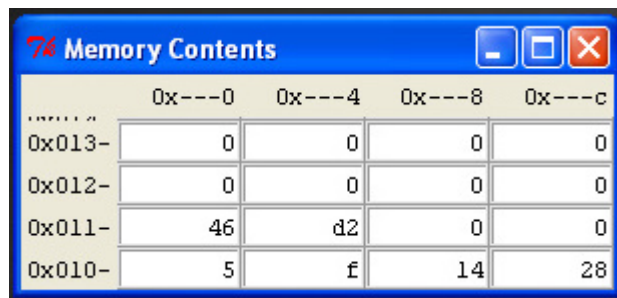
Les questions suivantes portent sur la fonction `main` (instructions d'adresses `0x000` à `0x01d`) ; pour répondre à ces questions il est inutile (et même fortement déconseillé) de lire le code de `map`.

2. Expliquer pourquoi l'instruction d'adresse `0x008` est une instruction `mrmovl` alors que celle d'adresse `0x010` est une instruction `irmovl`. A quoi servent les deux instructions `pushl` dans la fonction `main` ?
3. Voici le contenu de la pile et des registres *juste après* exécution de l'instruction `call map` par le simulateur Y86 (qui est donc prêt à exécuter la première instruction de la fonction `map`) :



Expliquer en détail les valeurs de chaque mot de la mémoire et de chaque registre (non nul).

4. Lorsque le programme s'arrête (instruction `halt`), quel est le contenu de la mémoire à partir de l'étiquette `u` (adresses `0x104` et suivantes) ? La réponse doit comporter pour chaque mot son adresse et sa valeur décimale.
5. Voici un cliché (partiel) de la mémoire une fois le programme exécuté :



Comparer ce cliché avec la réponse à la question précédente, en *détaillant* pour chaque mot (non nul) comment convertir en décimal sa représentation hexadécimale.

Les questions suivantes portent sur la fonction `map`.

6. Quel est le rôle de l'instruction d'adresse `0x022` : `mrmovl 12(%ebp),%ecx` ? Que contient le registre `%ecx` pendant l'exécution de la fonction `map` ? Expliquer le rôle des instructions d'adresses

0x050 à 0x058 :

```

irmovl 1,%edx
subl   %edx,%ecx
je     L2
    
```

7. Quel est le rôle de l'instruction d'adresse 0x031 : `mrmovl 8(%ebp), %ebx`? Que contient le registre `%ebx` pendant l'exécution de la fonction `map`? Quand est-il incrémenté, de combien, et pourquoi?
8. Pourquoi le registre `%ebx` est-il sauvegardé avant d'être utilisé? Quand est-il restauré? Si on supprime les deux instructions correspondantes, ce programme fonctionne-t-il correctement (il ne suffit pas de répondre par *oui* ou *non*, il faut justifier la réponse)?
9. Il reste à analyser le coeur de la fonction `map` :

```

L1:      mrmovl   (%ebx), %edx
        pushl   %ecx
        pushl   %edx
        call    f
        popl    %edx
        popl    %ecx
        rmmovl  %eax, (%ebx)

```

- (a) Expliquer le rôle de la première et de la dernière instruction, et pourquoi on utilise le registre `%eax` dans la dernière instruction.
- (b) La fonction `f` possède un seul argument, or on empile et dépile deux registres : expliquer ce mystère, et ce qui se passerait si on se contentait d'empiler et de dépiler l'argument de `f`.

Et pour ceux qui s'ennuieraient une belle image — et une dernière question...

10. Expliquer à quelle étape de l'exécution du programme a été pris le cliché ci-dessous, ainsi que le contenu de la pile et des registres à cet instant.

