Architecture des ordinateurs

Université de Tours

`Département informatique de Blois`

Examen 2019 - *Durée : 1h30*
(une feuille A4 manuscrite autorisée, calculatrice fournie)

*
* *

Question de cours (4 pts : $(2 + 2)$)

15 minutes

Les questions suivantes sont indépendantes.

1. Sur l'opérateur booléen $\downarrow$ Nor :

   (a) On rappelle que l'opérateur $x \downarrow y = \neg(x \vee y)$. Montrer que $\{\downarrow\}$ forme un système complet de connecteurs.

   Soient $x, y \in \{0, 1\}$. $\{\downarrow\}$ est un système complet de connecteurs s'il permet de représenter les opérateurs canoniques $\vee, \wedge$ et $\neg$.

   - $\neg x = \neg(x \vee x)$ – *Idempotence de* $\vee$
     $= x \downarrow x$
   - $x \vee y = \neg\neg(x \vee y)$ – *Principe de double négation*
     $= \neg(x \downarrow y)$
     $= (x \downarrow y) \downarrow (x \downarrow y)$ – *Application du* $\neg$ *selon l'opérateur* $\downarrow$
   - $x \wedge y = \neg(\neg x \vee \neg y)$ – *Loi de De Morgan*
     $= \neg x \downarrow \neg y$
     $= (x \downarrow x) \downarrow (y \downarrow y)$ – *Application du* $\neg$ *selon l'opérateur* $\downarrow$

   Dès lors $\{\downarrow\}$ est un système complet de connecteurs.

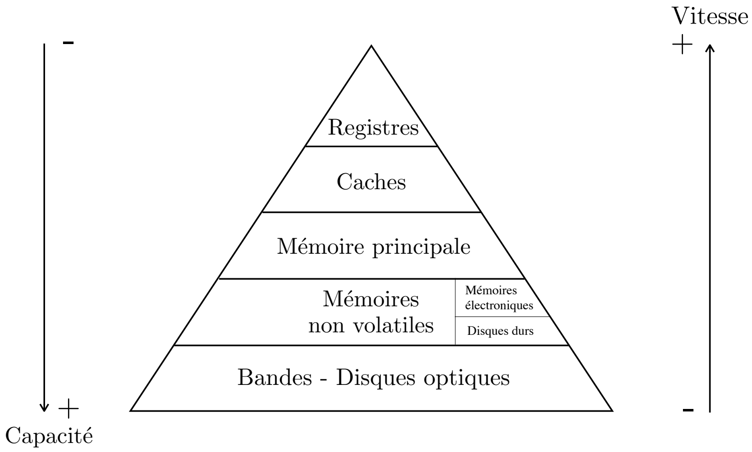
   $\square$

   (b) Montrer que $x \uparrow y = [(x \downarrow x) \downarrow (y \downarrow y)] \downarrow [(x \downarrow x) \downarrow (y \downarrow y)]$. Où $\uparrow$ désigne l'opérateur Nand tel que $x \uparrow y = \neg(x \wedge y)$.

   $x \uparrow y = \neg(x \wedge y)$
   $= \neg x \vee \neg y$ – *Loi de De Morgan*
   $= (x \downarrow x) \vee (y \downarrow y)$ – *Application du* $\neg$ *selon l'opérateur* $\downarrow$
   $= [(x \downarrow x) \downarrow (y \downarrow y)] \downarrow [(x \downarrow x) \downarrow (y \downarrow y)]$ – *Application du* $\vee$ *selon l'opérateur* $\downarrow$

   $\square$

2. Rappeler la pyramide de hiérarchie mémoire. Dans quelle catégorie de la pyramide se trouve l'accumulateur de l'UAL ? Expliquer en quelques lignes son utilité.



L'*accumulateur* est un registre. Il sert à stocker les calculs intermédiaires de l'unité arithmétique et logique afin d'éviter de les verser en mémoire principale puis de les recharger ; on évite alors un ralentissement et un engorgement des bus.

Problème 1 : Algorithme CORDIC (7 pts : $1 + (1.5 + 2) + (2 + 0.5)$)

40 minutes

L'algorithme CORDIC (COordinate Rotation Digital Computer) est très utilisé dans les micro-processeurs et les calculatrices pour le calcul des fonctions trigonométriques $\sin, \cos$ et $\tan$.

Soit un angle $\theta \in [0, \pi/2[$, on considère un vecteur de coordonnées $v_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$. Le principe de l'algorithme CORDIC est d'appliquer une rotation de ce vecteur à chaque itération i jusqu'à converger vers l'angle θ désiré. À l'itération $n \to +\infty$, on obtient $x_n = \cos(\theta)$ et $y_n = \sin(\theta)$.
L'équation de récurrence du vecteur est donnée par la relation :

$$v_{i+1} = \begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} = \frac{1}{\sqrt{1 + 2^{-2i}}} \times \begin{pmatrix} x_i - \sigma_i 2^{-i} y_i \\ x_i \sigma_i 2^{-i} + y_i \end{pmatrix}$$

où :

- $x_0 = 1$
- $y_0 = 0$
- $z_0 = \theta$
- $z_{i+1} = z_i - \sigma_i \varepsilon_i$
- $\varepsilon_i = \arctan\left(2^{-i}\right)$
- $\sigma_i = \mathrm{sgn}(z_i)$ où $\mathrm{sgn}(x)$ désigne le signe de x.

Ces formules supposent de pouvoir calculer plusieurs éléments complexes, notamment les valeurs de $\arctan(x)$ ou de $\frac{1}{\sqrt{1+x}}$. C'est ce qui va nous occuper dans la partie suivante.

Approximation des valeurs $\arctan(x)$ et $\frac{1}{\sqrt{1+x}}$

1. Justifier pourquoi pour des petites valeurs de x, on peut dire que $\arctan(x) \approx x$. Pourquoi cette proprité est intéressante ici ? Pouvez-vous donner une approximation polynomiale plus précise ?

Pour $x \to 0$, on a $\arctan(x) \approx x$. En effet, le développement limité en 0 de $\arctan(x)$ est tel que $DL_0 : \arctan(x) = x - \frac{x^3}{3} + \frac{x^5}{5} + o(x)$.

> Le premier terme donne la tangeante de $\arctan(x)$ en 0, soit ici x. Le développement limité complet fournit une approximation polynomiale de la fonction au point 0.
>
> Cette propriété est intéressante ici car on cherche à calculer $\arctan(2^{-i})$, pour tout $i \geq 1$, soit pour un antécédent qui tend très vite vers la valeur 0. On a donc une approximation simple (en 0) d'une fonction compliquée à calculer.

2. Sur le calcul de 2^{-i}.

   (a) Quel opérateur d'arithmétique binaire, combiné à une division, permet de calculer facilement la valeur flottante 2^{-i} ?

   > L'opérateur $x \ll y = x \times 2^y$. Dès lors, on a : $2^{-i} = \frac{1}{2^i} = \frac{1}{(1 \ll i)}$

   (b) Calculer la valeur IEEE 754 de 2^{-i} pour tout $i \geq 1$.

   > Soit $X = 2^{-i}$.
   > De fait $E(X) = 0$ et $F(X) = 2^{-i}$.
   > Dès lors, on a $X = 0, \underbrace{00...0}_{i-1 \text{ fois}}1$. Il vient que :
   >
   > - $s = 0$
   > - $e = -i$
   > - $m = 0$

3. On souhaite donner une approximation de la valeur de $\frac{1}{\sqrt{1+a}}$, pour tout $a \in \mathbb{R}^+$, à l'aide de l'algorithme de Newton.

   (a) On se propose d'utiliser la fonction $f(x) = \frac{1}{x^2} - 1 - a$ pour l'algorithme de Newton appliqué au calcul de $\frac{1}{\sqrt{1+a}}$. Justifier ce choix.

   > On pose $\alpha = \frac{1}{\sqrt{1+a}}$.
   > f est bien de classe C^1. On doit vérifier également que $f(\alpha) = 0$:
   >
   > $$\begin{aligned} f(\alpha) &= \frac{1}{\alpha^2} - 1 - a \\ &= \frac{1}{\left(\frac{1}{\sqrt{1+a}}\right)^2} - 1 - a \\ &= \left(\sqrt{1+a}\right)^2 - 1 - a \\ &= 0 \end{aligned}$$
   >
   > Dès lors, on peut bien utiliser f pour calculer $\frac{1}{\sqrt{1+a}}$ à l'aide de l'algorithme de Newton.

   (b) Donner l'itération de Newton x_{n+1} pour la fonction f proposée question 3. (a). Appliquer l'algorithme pour $n = 1$ avec $x_0 = \frac{1}{2}$.

   > L'itération de Newton est donnée par la formule :
   >
   > $$\begin{cases} x_0 \in \mathbb{R} \\ x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \end{cases}$$
   >
   > On a $f'(x) = -\frac{2}{x^3}$, il vient que $x_{n+1} = x_n + \frac{1}{2}\left(x_n - x_n^3(1-a)\right)$
   >
   > ***Application :***
   >
   > $x_0 = \frac{1}{2}$

$$x_1 = \frac{1}{2} + \frac{1}{2}\left(\frac{1}{2} - \frac{1}{2^3}(1-a)\right) = \frac{1}{16}(11+a)$$

(c) On applique l'algorithme de Newton au calcul de $\frac{1}{\sqrt{1+2}}$. Au rang $n = 4$ la valeur retournée est de 0.5773502692.

Comparer cette valeur avec celle donnée par la calculatrice. Sans le démontrer, la rapidité de convergence de l'algorithme vous semble d'ordre logarithmique, linéaire ou quadratique ?

La valeur au rang $n = 4$ est la même que celle donnée par la calculatrice.

On peut conclure que l'on a 10 chiffres significatifs correctes à la 4ème itération de l'algorithme. Sans peine, on peut émettre l'hypothèse que la convergence de l'algorithme est quadratique.

Codage de la valeur $\cos(1)$ **en IEEE 754**

On souhaite coder la valeur $\cos(1)$ sur un Half precision selon la norme IEEE 754.
Pour rappel, un Half est représenté par :

- 1 bit de signe
- 5 bits d'exposant E
- 10 bits de mantisse M
- Le biais associé ε vaut 15.

On applique CORDIC pour $\theta = 1$ et $n = 25$. L'algorithme retourne $\cos(1) \approx \frac{5403}{10000} = 0.54030$.

1. Coder la valeur $\frac{5403}{10000}$ sous le format Half precision de la norme IEEE 754. *On gardera la forme fractionnaire pour l'application de l'algorithme du calcul de la partie flottante.*

Soit $X = \frac{5403}{10000}$.
De fait $E(X) = 0$ et $F(X) = \frac{5403}{10000}$, on applique l'algorithme de calcul de la partie flottante.

$5403/10000$		
$\frac{5403}{10000} \times 2$	$\frac{5403}{5000}$	1
$\frac{403}{5000} \times 2$	$\frac{403}{2500}$	0
$\frac{403}{2500} \times 2$	$\frac{403}{1250}$	0
$\frac{403}{1250} \times 2$	$\frac{403}{625}$	0
$\frac{403}{625} \times 2$	$\frac{806}{625}$	1
$\frac{181}{625} \times 2$	$\frac{362}{625}$	0
$\frac{362}{625} \times 2$	$\frac{724}{625}$	1
$\frac{99}{625} \times 2$	$\frac{198}{625}$	0
$\frac{198}{625} \times 2$	$\frac{296}{625}$	0
$\frac{296}{625} \times 2$	$\frac{592}{625}$	0
$\frac{592}{625} \times 2$	$\frac{1184}{625}$	1
$\frac{559}{625} \times 2$	$\frac{1118}{625}$	1
$\vdots$	$\vdots$	$\vdots$

On a donc $X = 0,100010100011....$ Il vient que :

- $s = 0$
- $e = -1$. $E = e + \varepsilon = -1 + 15 = 14 = \langle 01110 \rangle_2$
- $m = M = 0001010001$

Soit :

0	01110	00010 10001

2. La valeur $\frac{5403}{10000}$ peut-elle être représentée dans un half sans faire d'erreur ? Justifier.

Non, car lors du calcul de la partie flottante avec l'algorithme de la question précédente, on ne tombe à 0. L'itération doit se poursuivre mais on n'a plus de place dans la mantisse de notre half.

Dès lors, $\frac{5403}{10000}$ ne peut pas être représenté exactement sous un half.

3. **Bonus (1pt)** : Comparer la valeur donnée par l'algorithme avec celle de la calculatrice. Que pouvez-vous dire sur la rapidité de convergence de l'algorithme CORDIC ?

La calculatrice redonne la valeur $\cos(1) = 0.5403023059$.

CORDIC donne donc 5 chiffres exactes à la 25 itérations. La convergence semble assez lente d'ordre linéaire, voire logarithmique.

Problème 2 : Assembleur (5 pts : $2 + 3$)

20 minutes

On souhaite ici programmer l'algorithme de la *conjecture de Syracuse* en Assembleur MIPS.
Soit $N \in \mathbb{N}^*$, la suite de Syracuse $(\mathcal{S}_n)_{n\in\mathbb{N}}$ est définie telle que :

$$\begin{cases} \mathcal{S}_0 = N & \\ \mathcal{S}_{n+1} = \frac{\mathcal{S}_n}{2} & \text{Si } \mathcal{S}_n \text{ est pair} \\ \mathcal{S}_{n+1} = 3 \times \mathcal{S}_n + 1 & \text{Si } \mathcal{S}_n \text{ est impair} \end{cases}$$

La conjecture établie que $\forall N \in \mathbb{N}^*, \exists k \in \mathbb{N} | \mathcal{S}_k = 1$, c'est-à-dire que pour tout nombre N de départ, la suite converge vers la valeur 1 à partir d'un certain rang k.
Le but de cet exercice est de déterminer le plus petit rang k où la suite $\mathcal{S}_k = 1$.

1. Écrire la forme linéaire (c'est-à-dire le pseudo-code assembleur) de algorithme de calcul du plus petit rang k tel que $\mathcal{S}_k = 1$ pour un nombre N de départ.

$\mathcal{S}_n \leftarrow N$

$k \leftarrow 0$

`while :`

  $\mathcal{S}_n = 1$? go to `fin`

  $k \leftarrow k + 1$

  $(\mathcal{S}_n \,\&\, 1) = 0$? go to `pair`

  $\mathcal{S}_n \leftarrow 3 \times \mathcal{S}_n + 1$

  go to `while`

`pair :`

  $\mathcal{S}_n \leftarrow \mathcal{S}_n / 2$

  go to `while`

`fin :`

  print k

2. Compléter le programme en assembleur MIPS syracuse.asm donné en annexe qui calcule le plus petit rang k tel que $\mathcal{S}_k = 1$.

   On donne en annexe les mnémoniques communs utilisés en MIPS. On suppose que N est contenu dans le registre `$t0` et qu'on dispose de la routine d'impression `print_int` et `print_string`.

   Ex : Si $N = 5$, le programme affichera "$k = 5$".

   Voir annexe.

Problème 3 : Clé de Parité (4 pts : $1.5 + 1 + 1.5$)

15 minutes

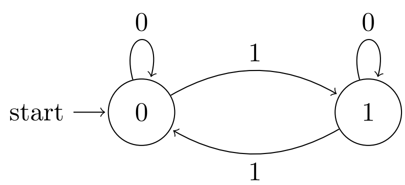
On souhaite représenter le circuit séquentiel permettant de calculer la clé de parité d'une séquence binaire $\langle x_n...x_2x_1\rangle$. Pour se faire, on représente chaque bit x_i par une variable d'entrée e qui vaut 1 si x_i vaut 1 et 0 sinon.
On note $c^{(i)}$ la clé de parité résultante pour la sous-séquence $\langle x_{i-1}...x_1\rangle$.
La clé de parité $c^{(i+1)}$ est alors donnée par la table de vérité ci-dessous :

$c^{(i)}$	e	$c^{(i+1)}$
0	0	0
0	1	1
1	0	1
1	1	0

1. Dessiner l'automate de Moore associée à la table de vérité de $c^{(i+1)}$.

   Il y'a deux valeurs possibles pour la variable $c^{(i)}$ donc deux états dans l'automate : 0 et 1.

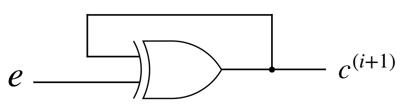
   Les transitions sont ensuite données par la valeur de e. On obtient alors :



2. Donner l'équation booléene de la variable $c^{(i+1)}$.

   Tout simplement : $c^{(i+1)} = c^{(i)} \oplus e$.

3. Dessiner le circuit séquentiel de $c^{(i+1)}$.



Annexes

Programme assembleur MIPS : Problème 2

```
##
# @author Clément Moreau
# File_name : syracuse.asm
# Description : Calcul du premier rang k tel que S_k = 1 où (S_n)
#               est la suite de Syracuse.
##
### Data section (Vous pouvez déclarer ici d'autres données et constantes) ###
.data
RANG   : .asciiz "k = "

### Text section ###
.text
.globl _main_
### Début du programme (à compléter) ###
_main_ :
    li $t0, N # S_0 = N (à définir)
```

```
      la $a0, RANG
      jal print_string
      li $a0, 0           # Compteur
      while :
          beq $t0, 1, fin
          addi $a0, $a0, 1
          andi $t1, $t0, 1 # Parité de S_n
          beq $t1, 0, pair
          mul $t0, $t0, 3
          addi $t0, $t0, 1
          j while
      pair :
          div $t0, $t0, 2
          j while
```

```
### Sorti du programme ###
fin :
    jal print_int       # Impression du rang k
    li $v0, 10
    syscall
## Routines d'impression ##
# print_int et print_string impriment le contenu du registre $a0
##
print_int :
    li $v0, 1
    syscall
    jr $ra
print_string :
```

```
li $v0, 4
syscall
jr $ra
```

Mnémoniques communs MIPS : Problème 2

Soient les registres $\$\mathtt{r}_{\mathtt{i}\in\{0,1,2\}}$, le registre `$CO` correspondant au compteur ordinal.

- `li $r0, n` effectue $\$\mathtt{r}_0 \leftarrow \mathtt{n}$.
- `lb $r0, n($r1)` effectue $\$\mathtt{r}_0 \leftarrow RAM[\$\mathtt{r}_1 + \mathtt{n}]$.
- `sb $r0, n($r1)` effectue $RAM[\$\mathtt{r}_1 + \mathtt{n}] \leftarrow \$\mathtt{r}_0$.
- `move $r0, $r1` effectue $\$\mathtt{r}_0 \leftarrow \$\mathtt{r}_1$.
- Opérations logiques et arithmétiques. Avec $\mathtt{Op} \in \{\mathtt{and}, \mathtt{or}, \mathtt{xor}, \mathtt{sll}, \mathtt{srl}, \mathtt{add}, \mathtt{sub}, \mathtt{mul}, \mathtt{div}\}$.

  `Op $r0, $r1, $r2` effectue $\$\mathtt{r}_0 \leftarrow \mathtt{Op}(\$\mathtt{r}_1, \$\mathtt{r}_2)$, et

  `Op $r0, $r1, n` effectue $\$\mathtt{r}_0 \leftarrow \mathtt{Op}(\$\mathtt{r}_1, \mathtt{n})$.
- `j label` effectue $\$\mathtt{CO} \leftarrow RAM[\mathtt{label}]$
- Branchement conditionnel. Avec $\mathtt{Op} \in \{\mathtt{beq}, \mathtt{bne}, \mathtt{blt}, \mathtt{ble}, \mathtt{bgt}, \mathtt{bge}\}$.
  - `beq $r0, $r1, label` effectue $\$\mathtt{CO} \leftarrow RAM[\mathtt{label}]$ si et seulement si $\$\mathtt{r}_0 = \$\mathtt{r}_1$, sinon, le programme se poursuit séquentiellement. (`bne` pour $\$\mathtt{r}_0 \neq \$\mathtt{r}_1$).
  - `blt $r0, $r1, label` effectue $\$\mathtt{CO} \leftarrow RAM[\mathtt{label}]$ si et seulement si $\$\mathtt{r}_0 < \$\mathtt{r}_1$, sinon, le programme se poursuit séquentiellement. (`ble` est utilisée pour la relation $\leq$).
  - `bgt $r0, $r1, label` effectue $\$\mathtt{CO} \leftarrow RAM[\mathtt{label}]$ si et seulement si $\$\mathtt{r}_0 > \$\mathtt{r}_1$, sinon, le programme se poursuit séquentiellement. (`bge` est utilisée pour la relation $\geq$).

  Ces opérations sont aussi valables pour la signature `Op $r0, n, label`. Dans ce cas, $\$\mathtt{r}_1$ est substitué par $\mathtt{n} \in \mathbb{Z}$. La sémantique de l'opération de branchement est conservée.