CC-TD-M – printemps 2010/2009 – LIF6 Architecture matérielle et logicielle

Durée 1h30 – Aucun document autorisé – Calculatrices, ordinateurs, téléphones portables interdits.

Pensez à bien numérotez vos réponses – On suppose que les nombres sont par défaut écrits en décimal.

I Analyse d'un programme en C

On considére le programme C ci-dessous.

```
1  - #include<stdio.h>
2  - int main(void) {
3  -   float x;
4  -   int i;
5  -
6  -   x=1.0;
7  -   i=0;
8  -   while(x != 0.0) {
9  -     x=x/2;
10 -     i=i-1;
11 -     /* valeur de x en fonction de i ? */
12 -   }
13 -   printf("i=%d\n", i);
14 -   return(0);
15 - }
```

En C, le type `float` représente les flottants simple précision de la norme IEEE-754 (modulo l'utilisation de bonnes options à la compilation, ce qui est la cas ici). On rappelle le tableau suivant issu du cours :

format	mantisse		exposant				taille
	précision	$p-1$	k	biais	$e_{\min}$	$e_{\max}$	
simple	$p=24$	23	8	127	-126	127	32 bits

Q.I.1) - La boucle `while` du programme ci-dessus se termine forcément : expliquez pourquoi.

L'ensemble des flottants représentable est un ensemble fini. En divisant de manière répétée un flottant positif par $2 > 1$, on obtient forcément 0 en un nombre fini de divisions.

Q.I.2) - Exprimez, sous la forme d'une puissance de 2, `x` en fonction de `i` à la ligne 11 du programme proposé.

$x = 2^i$

Q.I.3) - Déterminez μ, le plus petit flottant strictement positif sous-normal en simple précision. Exprimez votre résultat sous la forme d'une puissance de 2.

$\mu = 2^{-23} \times 2^{-126} = 2^{-149}$.

Q.I.4) - Lorsque l'on divise μ par 2 en arithmétique flottante simple précision, quel est le résultat calculé ? Vous pouvez vous aider d'un petit schéma pour justifier votre réponse.

On obtient 0.

Q.I.5) - Quelle est la valeur de `x` à la ligne 11 à l'avant-dernière itération de la boucle ? (Remarque : la valeur de `x` à la ligne 11 à la dernière itération est forcément 0). En déduire la valeur de `i` en sortie de boucle, *i.e.*, la valeur qui sera affichée.

D'après la question précédente, c'est $\mu = 2^{-149}$ la valeur de `x` à l'avant-dernière itération. Donc $i = -150$ en sortie de boucle.

II Langage machine

Un processeur 8 bits est doté d'un espace mémoire de 56 Kio ; à chaque adresse en mémoire centrale correspond une case de 1 octet. Le processeur dispose en outre d'un registre de travail sur 8 bits nommé `ACC` (pour « accumulateur »), permettant le stockage temporaire du résultat des opérations en arithmétique entière. Ses instructions sont de longueur fixe. Parmis les instructions figurent les suivantes (`@` désigne une adresse en mémoire centrale) :

Instruction	Codage (héxa.)	Fonction réalisée
`load @`	*A*0`@`	Charge le contenu de la case mémoire d'adresse `@` dans `ACC`
`store @`	*A*1`@`	Stocke le contenu du registre `ACC` dans la case mémoire d'adresse `@`
`add @`	*B*0`@`	Ajoute à `ACC` le contenu de la case mémoire d'adresse `@`, et place le résultat dans `ACC`. Les nombres sont manipulés comme des entiers naturels
`mul @`	*B*2`@`	Multiplie `ACC` par le contenu de la case mémoire d'adresse `@`, et place le résultat dans `ACC`.

Q.II.1) - Sur l'ordinateur considéré, combien d'octets sont nécessaires pour représenter chaque adresse de la mémoire centrale ? Justifiez votre réponse.

La capacité de la mémoire est de 56 Kio : 2^5 Kio $<$ 56 Kio $<$ 2^6 Kio. On a 2^6 Kio $= 2^6 \times 2^{10}$ o $= 2^{16}$ o : à raison d'un octet par adresse, cela donne 2^{16} adresses, donc 16 bits par adresses, ou 2 octets. On supposera donc par la suite que les adresses sont codées sur 2 octets.

Q.II.2) - Quelle est la taille en octets du compteur ordinal `CO` sur cet ordinateur ?

L'UCT doit pouvoir aller chercher ses instructions dans une mémoire dont les adresses sont codées sur 2 octets : on peut donc supposer que la taille de `CO` est de de 2 octets.

Q.II.3) - Combien d'octets occupent chacune des instructions décrites ci-dessus ? Justifiez brièvement votre réponse.

On détermine le nombre d'octets nécessaires pour le codage du code-op, d'après les codes en hexadécimal de l'énoncé. Comme chaque instruction prend comme opérande une adresse, on ajoute ensuite 2 octets. Toutes les instruction sont donc codées sur 3 octets.

Q.II.4) - Écrire en assembleur un morceau de programme qui évalue l'expression suivante, en respectant le parenthésage : $(a \times x + b) \times (c \times y + d)$, puis range le résultat à l'adresse $(0130)_H$. On suppose l'association suivante entre les cases mémoires et les variables utilisées dans l'expression précédente.

case mémoire d'adresse	(0131)	(0132)	(0133)	(0134)	(0135)	(0136)
variable	x	y	a	b	c	d

Vous ne devez pas utiliser de case mémoire supplémentaire.

`load` 0131_H
`mul` 0133_H
`add` 0134_H
`store` 0130_H
`load` 0132_H
`mul` 0135_H
`add` 0136_H
`mul` 0130_H
`store` 0130_H

Q.II.5) - Représentez en mémoire centrale le morceau de programme précédent, en supposant que le première instruction débute à l'adresse $(0105)_H$. Indiquez le contenu des cases mémoire en hexadécimal. Comme en cours et en TD, vous utiliserez la convention « gros-boutiste » pour le stockage des octets.

Instruction	Adresse	Contenu
`load` 0131_H	0105_H	$A0$
	0106_H	01
	0107_H	31
`mul` 0133_H	0108_H	$B2$
	$010A_H$	01
	$010B_H$	33
`add` 0134_H	$010C_H$	$B0$
	$010D_H$	01
	$010E_H$	34
`store` 0130_H	$010F_H$	$A1$
	0110_H	01
	0111_H	30
`load` 0132_H	0112_H	$A0$
	0113_H	01
	0114_H	32
`mul` 0135_H	0115_H	$B2$
	0116_H	01
	0117_H	35
`add` 0136_H	0118_H	$B0$
	$011A_H$	01
	$011B_H$	36
`mul` 0130_H	$011C_H$	$B2$
	$011D_H$	01
	$011E_H$	30
`store` 0130_H	$011F_H$	$A1$
	0120_H	01
	0121_H	30

Q.II.6) - Représentez, après chaque instruction du programme, le contenu de `CO`, de `ACC` et de la case d'adresse $(0132)_H$. On suppose les affectations initiales suivantes pour les différentes variables

case mémoire d'adresse	(0131)	(0132)	(0133)	(0134)	(0135)	(0136)
variable	x	y	a	b	c	d
valeurs	$(4)_{10}$	$(8)_{10}$	$(4)_{10}$	$(36)_{10}$	$(53)_{10}$	$(17)_{10}$

Indiquez le contenu de `CO` en hexadécimal, et ceux de ACC et de la case mémoire d'adresse $(0130)_H$ en décimal. Donnez votre résultat sous forme d'un tableau.

Il faut bien faire attention à effectuer toutes les opérations modulo $2^8 = 256$.

Instant	contenu de `CO`	contenu de `ACC`	contenu de la case d'adresse $(0130)_H$
après `load` 0131_H	0108_H	4	inchangé
après `mul` 0133_H	$010C_H$	16	inchangé
après `add` 0134_H	$010F_H$	52	inchangé
après `store` 0130_H	0112_H	52	52
après `load` 0132_H	0115_H	8	52
après `mul` 0135_H	0118_H	168	52
après `add` 0136_H	$011C_H$	185	52
après `mul` 0130_H	$011F_H$	77	52
après `store` 0130_H	0122_H	77	77

III Représentation à virgule flottante

On travaille sur une machine où les flottants binaires sont codés sur 12 bits. On considère le code :

$c =$

11	10	9	8	7	6	5	4	3	2	1	0
s	e_c				m_c						

Le nombre rationnel représenté est $f(c) = (-1)^s \times (1, m_c)_2 \times 2^{e(c)}$. Si $1 \leq N(e_c) \leq 14$, alors $e(c) = N(e_c) - 2^3$ (on ne se préoccupe pas du codage des sous-normaux ni des valeurs exceptionnelles ici).

Q.III.1) - Quel nombre est représenté par le code hexadécimal $c = 625_H$, si on le regarde comme le codage d'un flottant ? Donnez votre résultat sous la forme d'une représentation positionnelle à virgule en décimal.

Le code $c = 625_H$ représente $011\ 000\ 100\ 101_2 = 0\ 1100\ 0100101_2$. L'exposant du flottant considéré est donc $e(c) = N(e_c) - 8 = (1100)_2 - 8 = 12 - 8 = 4$. La mantisse du flottant est $(1, 0100101)_2$. On a donc

$$\begin{aligned} f(c) &= (1, 0100101)_2 \times 2^4 \\ &= (10100, 101)_2 \times 2^4. \end{aligned}$$

On trouve que $(10100)_2 = (20)_{10}$ et $(0, 101)_2 = 0.5 + 0.125 = (0, 625)_{10}$, donc $f(c) = (20, 625)_{10}$.

Q.III.2) - Comment s'écrit le nombre $(0, 35)_{10}$ en représentation positionnelle à virgule en binaire ? Détaillez un peu vos calculs.

On trouve après calculs que $(0, 35)_{10} = (0, 01\underline{0110})_2$.

Q.III.3) - Comment peut-on représenter le nombre $(0, 35)_{10}$ dans le format flottant binaire décrit ci-dessus ? Vous effectuerez un arrondi au plus proche, et exprimerez votre résultat sous la forme d'un code hexadécimal.

On travaille avec en tout 8 bits de précision :

$$\begin{aligned} (0, 35)_{10} &= (0, 0101100110\underline{0110})_2 \times 2^0 \\ &= (1, 0110011\ 0\underline{0110})_2 \times 2^{-2} \\ &\approx (1, 0110011)_2 \times 2^{-2}, \end{aligned}$$

après arrondi au plus proche. On a donc $N(e_c) = e(c) + 8 = -2 + 8 = 6 = (0110)_2$ et $c_m = 0110011_2$. Le codage c de $(0, 2)_{10}$ au format flottant, en arrondi au plus proche, sera donc $c = 0\ 0110\ 0110011_2 = 0011\ 0011\ 0011_2 = 333_H$.

Q.III.4) - On note f_1 l'approximation de $(0,35)_{10}$ sous la forme d'un flottant obtenue ci-dessus. On note f_2 le flottant dont le code en hexédécimal est 625_H. Posez l'opération $f_1 + f_2$ en binaire, avec arrondi final au plus proche (il faut : exprimer f_1 et f_2 sous formes normalisées ; aligner les virgules dans les écritures de f_1 et f_2 ; effectuer l'addition exactement ; arrondir à la précision du résultat ; exprimer le résultat arrondi sous forme normalisée).

IV Représentation en complément à 2

Q.IV.1) - Comment sont représentés $-(78)_{10}$ et $-(59)_{10}$ en complément à 2 sur 8 bits ?

Q.IV.2) - Posez l'opération $-(78)_{10} - (59)_{10}$ en utilisant le complément à 2 sur 8 bits. Que constatez-vous ?

Q.IV.3) - Comment sont représentés $-(78)_{10}$ et $-(59)_{10}$ en complément à 2 sur 12 bits ?

Q.IV.4) - Posez l'opération $-(78)_{10} - (59)_{10}$ en utilisant le complément à 2 sur 12 bits. Que constatez-vous ?