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 Représentation en complément à 2

Q.I.1) - Comment sont représentés $(34)_{10}$ et $(-42)_{10}$ en complément à 2 sur 8 bits (complément à 2^8) ?

$(34)_{10} = (00100010)_2$ donc $(34)_{10} = (00100010)_{\overline{2}}$. D'autre part, $(42)_{10} = (00101010)_2$, donc $(-42)_{10} = (11011010)_2$.

Q.I.2) - Comment sont représentés $(34)_{10}$ et $(-42)_{10}$ en complément à 2 sur 12 bits (complément à 2^{12}) ?

$(34)_{10} = (000000100010)_2$ donc $(34)_{10} = (000000100010)_{\overline{2}}$. D'autre part, $(42)_{10} = (000000101010)_2$, donc $(-42)_{10} = (111111011010)_2$.

Q.I.3) - Proposez une régle permettant de passer de l'écriture d'un entier relatif en complément à 2 sur p bits à son écriture en complément à 2 sur $p + k$ bits (en conservant l'égalité des valeurs bien-entendu). Justifiez brièvement votre réponse.

Il suffit de répéter le bit de signe sur les k nouveaux bits de poids fort.

II 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 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 exeptionnelles ici).

Q.II.1) - Quel nombre est représenté par le code octal $c = 7045_8$, 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 $c = 7045_8$ représente $111\ 000\ 100\ 101_2 = 1\ 1100\ 0100101_2$. 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.II.2) - Comment est représenté le nombre $(0, 2)_{10}$ représentation positionnelle à virgule en binaire ? Détaillez un peu vos calculs.

On trouve après calculs que $(0, 2)_{10} = (0, \underline{0011})_2$.

Q.II.3) - Comment peut-on représenter le nombre $(0, 2)_{10}$ 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 :

$$\begin{aligned}(0,2)_{10} &= (0,0011001100110011)_2 \times 2^0 \\ &= (1,1001100\ 110011)_2 \times 2^{-3} \\ &\approx (1,1001101)_2 \times 2^{-3},\end{aligned}$$

après arrondi au plus proche. On a donc $N(e_c) = e(c) + 8 = -3 + 8 = 5 = (0101)_2$ et $m_c = 1001101_2$. Le codage c de $(0,2)_{10}$ au format flottant, en arrondi au plus proche, sera donc $c = 0\ 0101\ 1001101_2 = 0010\ 1100\ 1101_2 = 2CD_H$.

III Langage machine

Un processeur 8 bits est doté d'un espace mémoire de 64 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 codées sur 1 à 4 octets. 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*2`@`	Stocke le contenu du registre `ACC` dans la case mémoire d'adresse `@`
`add @`	0206`@`	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

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

La capacité de la mémoire est de 64 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.III.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.III.3) - Combien d'octets occupent chacune des instruction décrites ci-dessus ? Justifiez brièvement votre réponse, puis donnez vos résultats sous forme d'un tableau.

On détermine le nombre d'octet nécessaire 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.

Instruction	Codage (héxa.)	Taille
`load @`	*A*0`@`	3 o
`store @`	*A*2`@`	3 o
`add @`	0206`@`	4 o

Q.III.4) - Écrire en assembleur un morceau de programme qui ajoute le contenu des cases mémoires d'adresses $(0130)_H$ et $(0131)_H$, puis range le résultat à l'adresse $(0132)_H$.

`load` 0130_H
`add` 0131_H
`store` 0132_H

Q.III.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` 0130_H	0105_H	$A0$
	0106_H	01
	0107_H	30
`add` 0131_H	0108_H	02
	0109_H	06
	$010A_H$	01
	$010B_H$	31
`store` 0132_H	$010C_H$	$A2$
	$010D_H$	01
	$010E_H$	32

Q.III.6) - Représentez, après chaque instruction du programme, le contenu de `CO`, de `ACC` et de la case d'adresse $(0132)_H$. Supposez que le cases mémoires d'adresses $(0130)_H$ et $(0131)_H$ contiennent initialement les valeurs $(88)_H$ et $(05)_H$ respectivement. Indiquez les contenus des registres et de la case mémoire en hexadécimal, et donnez votre résultat sous forme d'un tableau.

Instant	contenu de `CO`	contenu de `ACC`	contenu de la case d'adresse $(0132)_H$
après `load` 0130_H	0108_H	88_H	inchangé
après `add` 0131_H	$010C_H$	05_H	inchangé
après `store` 0132_H	$010F_H$	inchangé	$8D_H$

Q.III.7) - Supposez maintenant que les cases d'adresses $(0130)_H$ et $(0131)_H$ contiennent initialement les valeurs $(254)_{10}$ et $(28)_{10}$ respectivement. Après l'exécution du programme, quel sera le contenu de la case d'adresse $(0132)_H$? Donnez votre résultat en décimal, et justifiez brièvement.

$254 + 28 \mod 2^8 = 282 \mod 256 = 26$.

Q.III.8) - Écrire en assembleur un programme qui échange le contenu des cases mémoire d'adresses $(0130)_H$ et $(0131)_H$.

Il est nécessaire d'utilisez une case mémoire pour effectuer un stockage intermédiaire. Utilisons la case d'adresse $(0132)_H$.

`load` 0130_H
`store` 0132_H `load` 0131_H
`store` 0130_H `load` 0132_H
`store` 0131_H

IV Analyse d'un petit programme en C

On considére le programme C ci-dessous. On va surtout s'intéresser à la portion du programme située entre les lignes 7 et 13.

```
1  - #include <stdio.h>
2  - int main(void) {
3  -   unsigned char a, b, m, r;
4  -   /* On initialise a et b */
5  -   a = 53; b = 5;
6  -
7  -   r = 0;
8  -   m = a;
9  -   while(b != 0) {
10 -     if(b%2 == 1) r = r+m;
11 -     b = b/2;
12 -     m = m*2;
13 -   }
14 -
15 -   printf("r=%d\n", r);
16 -   return(0);
17 - }
```

En C, le type `unsigned char` code des entiers naturels binaires (non-signés) sur 8 bits. Dans ce programme, `b/2` et `b%2` désignent le quotient et le reste dans la division euclidienne de `b` par 2.

Q.IV.1) - Quels sont, à leur initialisation, les représentations en mémoire de `a` et `b` en binaire ?

Q.IV.2) - Connaissant l'écriture de `b` en binaire, comment déterminer si `b%2` prend pour valeur 0 ou 1 ?

Q.IV.3) - Reportez dans un tableau de la forme suivante les valeurs de `r`, `m` et `b` (en écriture binaire) au cours des itérations de la boucle `while`. Détaillez (sous votre tableau) les calculs en binaire qui ne peuvent se faire facilement de tête. Précisez également à quelle étape la boucle prend fin.

	r	b	m
à la première entrée dans la boucle	$(00000000)_2$		
à la fin de la 1ère itération			
à la fin de la 3ème itération			
⋮	⋮	⋮	

	r	b	m
à la première entrée dans la boucle	$(00000000)_2$	$(00000101)_2$	$(00110101)_2$
à la fin de la 1ère itération	$(00110101)_2$	$(00000010)_2$	$(01101010)_2$
à la fin de la 2ème itération	$(00110101)_2$	$(00000001)_2$	$(11010100)_2$
à la fin de la 3ème itération	$(00001001)_2$	$(00000000)_2$	$(10101000)_2$

La boucle `while` au bout de la 3ème itération, car `b`=0.

Q.IV.4) - Quelle valeur sera affichée par la ligne `printf("r=%d\n", r);` du programme ? Exprimez votre résultat en décimal. Comment interprétez ce résultat ? Justifiez votre réponse, par exemple en effectuant à la main un petit calcul en binaire et le même calcul en décimal.

La valeur affichée sera $(1001)_2 = (9)_{10}$. On a $(110101)_2 \times (101)_2 = (100001001)_2$, donc $(110101)_2 \times (101)_2 \mod 2^8 = (1001)_2 = (9)_{10}$. Le résultat calculé est $(53)_{10} \times (5)_{10} \mod 2^8 = (265)_{10} \mod (256)_{11} = (9)_{10}$, ce qui est le résultat du produit de `a` par `b` modulo 256.

Q.IV.5) - D'une manière plus générale, quel sera le résultat produit par l'exécution du programme, quelles que soient les entrées `a` et `b` ? Justifiez brièvement votre réponse.

L'algorithme présenter est l'algorithme classique pour effectuer le produit de deux entiers naturels écrits en binaire, mais toutes les opérations intermédiaires sont effectuées modulo 2^8. Le résultat obtenu est donc le produit de `a` par `b` modulo 256.