CC-TD-M – lundi 15/11/2010 – LIF6 Architecture matérielle et logicielle

Mioara Joldes – Nicolas Louvet

Aucun document autorisé. Les calculatrices, ordinateurs, téléphones et autres sont interdits.
Durée 1h30. Le barème est susceptible de modifications.

I 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. Parmi 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.I.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.I.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.I.3) - Combien d'octets occupe chacune des instructions décrites ? Justifiez 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.I.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_H	0132_H	0133_H	0134_H	0135_H	0136_H
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.I.5) - Donnez le codage en mémoire centrale des quatres premières instructions du programme de la question précédente, en supposant que la première instruction débute à l'adresse $(0105)_H$. Indiquez le contenu

des cases mémoire en hexadécimal. Comme 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.I.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_H	0132_H	0133_H	0134_H	0135_H	0136_H
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

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 exceptionnelles 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 :

$$\begin{aligned} (0, 2)_{10} &= (0, 0011001100110\underline{0011})_2 \times 2^0 \\ &= (1, 1001100\ 11\underline{0011})_2 \times 2^{-3} \\ &\approx (1, 1001101)_2 \times 2^{-3}, \end{aligned}$$

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 Imparité

On considère la fonction $f : \mathbb{B}^3 \to \mathbb{B}$ définie de la manière suivante :
- $f(x_2, x_1, x_0) = 1$ si le mot $x_2x_1x_0$ comprend un nombre impair de bits à 1,
- $f(x_2, x_1, x_0) = 0$ si le mot $x_2x_1x_0$ comprend un nombre pair de bits à 1.

La fonction f est souvent appelée fonction *imparité*.

Q.III.1) - Dressez la table de vérité de $f(x_2, x_1, x_0)$ en fonction de x_2, x_1, x_0.

Q.III.2) - Donnez une expression booléenne sous forme normale disjonctive pour f.

Q.III.3) - Exprimez f uniquement à l'aide de l'opérateur XOR (noté $\oplus$).

Q.III.4) - Donnez un logigramme pour f en utilisant uniquement des portes XOR.

IV Mémoire cache

On considère le programme C suivant :

```
int main(void) {
  int A[16], B[16], C[16];
  int i;
  for(i=0; i<16; i++) C[i] = A[i] + B[i];
  ...
}
```

On suppose que l'on compile et que l'on exécute ce programme sur un ordinateur qui ne dispose que d'un seul niveau de cache de données direct : ce cache comporte 4 entrées de 32 octets (on ne se préoccupe pas du chargement des instructions, ni de la variable `i` dans cet exercice). On note les lignes de cache $\ell_0, \ell_1, \ell_2, \ldots$, et les entrées du cache e_0, e_1, e_2, e_3. Les entiers de type `int` sont d'une taille de 32 bits. Les tableaux sont stockés de manière contigüe en mémoire, dans l'ordre de leur déclaration.

Q.IV.1) - Combien d'entiers de type `int` peut contenir une ligne de cache ?

Un `int` occupe 4 o. Une ligne de cache de 32 o peut donc contenir 8 int.

Q.IV.2) - Combien de lignes de caches consécutives occupe chacun des tableaux `A`, `B` et `C` ?

Chaque ligne de cache compte 8 int, donc un tableau de 16 int occupe 2 lignes de cache consécutives.

Q.IV.3) - En supposant que `A[0]` est aligné sur le début de la ligne de cache ℓ_0, indiquez sur un schéma à quelles lignes de cache appartiennent les élément de `A`, `B` et `C`. Indiquez également sur ce même schéma dans quelles entrées du cache seront rangés ces élements lorsqu'ils seront accédés.

- `A[0....7]` appartiennent à ℓ_0, stockés dans e_0
- `A[8...15]` appartiennent à ℓ_1, stockés dans e_1
- `B[0....7]` appartiennent à ℓ_2, stockés dans e_2
- `B[8...15]` appartiennent à ℓ_3, stockés dans e_3
- `C[0....7]` appartiennent à ℓ_4, stockés dans e_0
- `C[8...15]` appartiennent à ℓ_5, stockés dans e_1

Q.IV.4) - Dans un tableau de la forme indiquée ci-dessous, indiquez quelle ligne de cache contient chaque entrée du cache à la fin de chaque itération de la boucle `for(i=0; i<16; i++) C[i] = A[i] + B[i]` ; (Vous placerez un « ? » quand on ne peut pas savoir). Indiquez également le nombre de *cache miss* qui ont eu lieu au cours de l'itération. Quel est le nombre total de *cache miss* ?

`i`	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
e_0																
e_1																
e_2																
e_3																
cache miss																

`i`	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
e_0	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14
e_1	?	?	?	?	?	?	?	?	15	15	15	15	15	15	15	15
e_2	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
e_3	?	?	?	?	?	?	?	?	13	13	13	13	13	13	13	13
cache miss	3	2	2	2	2	2	2	2	3	2	2	2	2	2	2	2

Soit un total de 20 *cache miss*.