Ing1 – Examen de rattrapage d'Architecture des Ordinateurs

Durée : 2H – Aucun document autorisé – Calculatrice Interdite

Exercice 1 – 1,5 pts (0.5 chacun)

On s'intéresse à la taille de la mémoire qu'un ordinateur peut gérer suivant le type d'adresses manipulées. Nous étudions dans cet exercice une mémoire adressée par mot de 64 bits. Les réponses sont à donner en octets (utilisez l'unité « multiple » adéquate) et devront être justifiées.

Préambule : 1 mots de 64 bits = 8 octets = 2^3 octets

1.a) Quelle est la taille théorique maximale de la mémoire pour des adresses de 16 bits.

2^{16} mots = 2^{19} octets = 512 Ko

1.b) Quelle est la taille théorique maximale de la mémoire pour des adresses de 32 bits.

2^{32} mots = 2^{35} octets = 32 Go

1.c) Quelle est la taille théorique maximale de la mémoire pour des adresses de 64 bits.

2^{64} mots = 2^{67} octets = 128 Eo

NB : rappel des unités de taille de données : o, Ko, Mo, Go, To, Po, Eo, Zo, Yo

Exercice 2 – 4,5 pts

On considère les entiers relatifs codés en complément à 2 sur 5 bits.

2.a) Quel est l'intervalle des nombres codables dans ce système ? [-2^4 ;2^4-1] soit [-16 ;+15] => 0.5 pt

2.b) Donner le code binaire des entiers relatifs suivants : -6, -10, +2, +14

- 6 : 11010 -10 : 10110 +2 : 00010 +14 : 01110 => 0.25 pt chacun

3.c) Calculer les 3 additions -6 + (-10) , -10 + 14 et 2 + 14 en binaire complément à 2 (donner le détail du calcul). Vous préciserez, en le justifiant, si le résultat obtenu est correct ou s'il y a dépassement de capacité.

1 point par calcul (0.5 pour poser le calcul et 0.5 pour interprétation + justification)

Rappel :

- **2 dernières retenues identiques => résultat correct**
- **addition de 2 nombres de signes opposés => correct**

-6+(-10) :

```
11110
 11010
 10110
_____
 10000
```

Résultat sur 5 bits : 10000 qui représente -16. Résultat correct (2 dernières retenues à 1)

(-10) + 14 :

```
11110
 10110
 01110
_____
 00100
```

Résultat sur 5 bits : 00100 qui représente 4. Résultat correct (2 dernières retenues à 1 ou addition de 2 nbs de signes opposés)

2 + 14 :

01110
 00010
 01110

_____
 10000

Résultat sur 5 bits : 10000 correspondant à -16. Résultat faux (2 dernières retenues différentes)

Exercice 3 – 5 pts

On s'intéresse à la traduction en assembleur du programme Java suivant qui calcule les 14 premiers termes de la suite de Fibonacci (pour info le dernier terme calculé vaut 233 en décimal) :

```
L1    int[] v = new int[14] ; // données sur 14 x 8 bits à l'adresse 0x100
L2
L3    v[0] = 0 ;
L4    v[1] = 1 ;
L5    for (int i = 0 ; i < 12 ; i++) {
L6          v[i+2] = v[i] + v[i+1];
L7    }
```

Donner la traduction des instructions (lignes 3 à 7) en assembleur. Si vous utilisez des instructions de sauts, remplacez les déplacements par des labels pour éviter de les calculer.

Remarque : Chaque entier est codé sur 2 mots ; les poids faibles sont rangés avant les poids forts. Par exemple, si v1[0] contient l'entier 0xA0, on a en mémoire

Adresse	Donnée
0x100	0x0
0x101	0xA

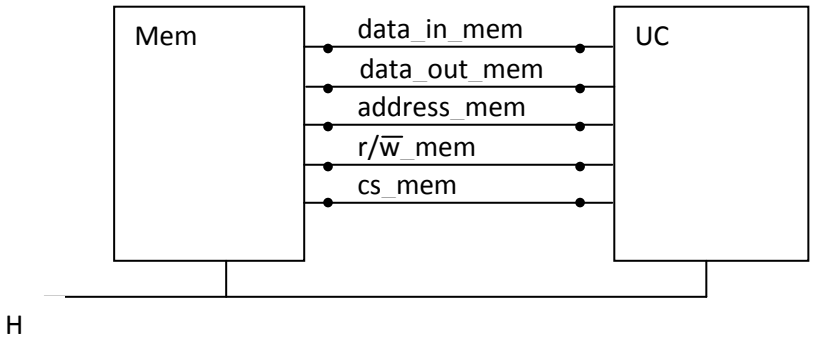
Une fiche récapitulative du langage assembleur utilisé est donnée en annexe.

Pts clés :

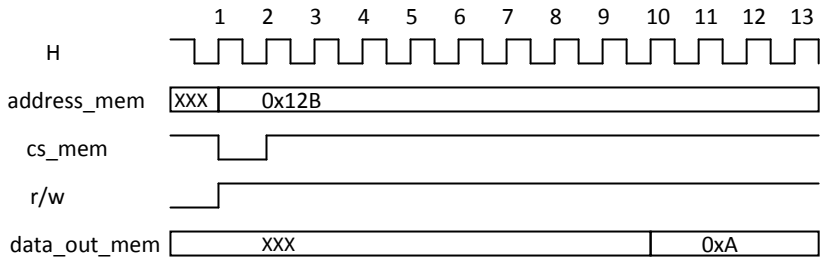
- initialisation : 1 pt. 4 mots à écrire
  - @0x100 : 0000
  - @0x101 : 0000 (v[0] = 0)
  - @0x102 : 0001
  - @0x103 : 0000 (v[1] = 1)
- Boucle 12 fois : 1 pt. Le plus simple est d'initialiser R7 à 12 et de faire un DJNZ debutBoucle à chaque fin de tour de boucle
- Addition des poids faibles : 1,5 pts. ADD : (0x104+i) <- (0x102+i) + (0x100+i)
- Addition des poids forts : 1,5 pts. ADC : (0x105+i) <- (0x103+i) + (0x101+i)
- Vérifier le calcul des @, la bonne utilisation du jeu d'instructions fourni et le fait qu'il ne perde pas la retenue si jamais ils utilisent une opération arith (sauf INC) entre ADD et ADC.
-

Exercice 4 – 3 pts

Soit le dialogue entre une mémoire et une unité de commande suivant :



La mémoire est à temps de réponse constant (pas de signal ready). On souhaite mesurer ce temps de réponse grâce au chronogramme suivant afin de régler au mieux l'unité de commande. Dans cet exemple, l'unité de commande lit la donnée 0xA à l'adresse 0x12B.



Précisez à quel top d'horloge (front montant) interviennent les traitements suivants :

- U1 : Unité de Commande : ordre de lecture **TOP 1**
- U2 : Unité de Commande : lecture du résultat (top minimal) **TOP11**
- M1 : Mémoire : prise en compte de l'ordre de lecture **TOP 2**
- M2 : Mémoire : sortie du résultat **TOP 10**

1 pt (non divisible) pr U1 + M1
1 pt (non divisible) pr M2 + U2

En déduire le nombre de cycles d'attente minimum de l'unité de commande pour un ordre de lecture.

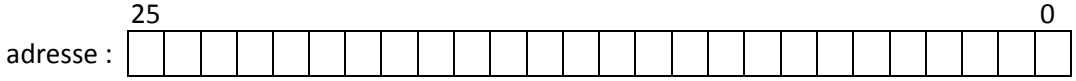
⇨ 9 cycles d'attentes : 1 pt

Exercice 5 – 6 pts

On adjoint à une mémoire principale adressée sur 26 bits une mémoire cache associative par ensemble. Les deux mémoires manipulent des blocs de 8 mots. La mémoire cache possède 64 ensembles de 16 blocs.

5.a) A partir de l'adresse d'une donnée en mémoire principale, expliquer à l'aide du schéma suivant comment trouver : 2 pts (0.5 chacun)

- le numéro d'un mot dans un bloc (offset) : 0..2
- le numéro du bloc dans la mémoire principale : 3..25
- le numéro de l'ensemble qui va accueillir le bloc de cette donnée dans le cache : 3..8 (si n° bloc juste)
- l'information additionnelle (TAG) que vous devez ajouter à un bloc de donnée pour retrouver la provenance de celui-ci : 9..25 (si n° bloc juste)



5.b) Chaque ensemble est géré en LRU (Least Recently Used). Chaque bloc est estampillé par un bit de validité (V) et une date (LRU) sur 3 bits (de 0 à 7), la valeur 0 désignant le plus récemment utilisé. On part d'un cache rempli à l'instant t0. Montrez l'évolution de l'ensemble représenté par le tableau suivant, en fonction des accès (lecture ou écriture) aux données des blocs suivants : B10, B5, B7, B8, B7, B2, B12, B3, B5, B6, B10 (en considérant que tous ces blocs sont destinés à cet ensemble).

Mettez en évidence les défauts de cache. On pourra utiliser les caractères suivants : 'U' (Undefined), '-' (contenu inchangé), 'D' (défaut de cache), 'F' (False) et 'T' (True).

Total 4 pts en comptant en négatif (minoré par 0 ;-))
-1 pt si erreur sur les défauts de cache signalés
-0.5 par erreur sur un accès (mauvais emplacement ou mauvais LRU)

t0			t1 : accès B10			t2 : accès B5			t3 : accèsB7			t4 : accès B8			t5 : accès B7			t6 : accès B2			t7 : accès B12		
Blocs	V	LRU	Blocs	V	LRU	Blocs	V	LRU	Blocs	V	LRU	Blocs	V	LRU	Blocs	V	LRU	Blocs	V	LRU	Blocs	V	LRU
B1	T	3	-	-	4	-		5	-	-	6	-		7	-	-	-	B2	T	0			1
B2	T	7	-	-	-	B5	T	0	-	-	1	-		2	-	-	-			3			4
B3	T	2	-	-	3	-		4	-	-	5	-		6	-	-	-			7	B12	T	0
B6	T	0	-	-	1	-		2	-	-	3	-		4	-	-	-			5			6
B9	T	1	-	-	2	-		3	-	-	4	-		5	-	-	-			6			7
B10	T	4	-	-	0	-		1	-	-	2	-		3	-	-	-			4			5
B7	T	6	-	-	-	-		7	-	-	0	-		1	-	-	0			1			2
B12	T	5	-	-	-	-		6	-	-	7	B8	T	0	-	-	1			2			3
Défauts :						D						D						D			D		

t8 : accès B3			t9 : accès B5			t10 : accès B6			t11 : accès B10		
Blocs	V	LRU	Blocs	V	LRU	Blocs	V	LRU	Blocs	V	LRU
-	-	2	-	-	3	-	-	4	-	-	5
-	-	5	-	-	0	-	-	1	-	-	2
-	-	1	-	-	2	-	-	3	-	-	4
-	-	7	-	-	-	-	-	0	-	-	1
B3	T	0	-	-	1	-	-	2	-	-	3
-	-	6	-	-	-	-	-	7	-	-	0
-	-	3	-	-	4	-	-	5	-	-	6
-	-	4	-	-	5	-	-	6	-	-	7
D											

Annexe : assembleur Machine de Von Neumann

Le langage d'assemblage est celui utilisé en TD avec une augmentation du nombre de registres.

Description générale :

- Machine 4 bits (taille mots mémoire et opérande ALU), adresses 12 bits
- 8 registres 4 bits R_0 à R_7
- 1 compteur ordinal PC 12 bits
- 1 registre de données MDR 4 bits
- 1 registre d'adresse MAR 12 bits

Instructions de déplacements de données :

LD R_0 ($R_0R_1R_2$) MAR <- $R_0R_1R_2$; Lecture ; R_0 <- MDR
Lecture du mot contenu à l'adresse $R_0R_1R_2$ en mémoire puis rangé dans R_0

LD R_0 R_j R_0 <- R_j, j dans [1-7]
Copie du registre R_j vers R_0

LD R_0 V R_0 <- V
Copie de la valeur constante V (4 bits) vers R_0

LD R_j R_0 R_j <- R_0, j dans [1-7]
Copie du registre R_0 vers R_j

LD ($R_0R_1R_2$) R_3 MAR <- $R_0R_1R_2$; MDR <- R_3 ; Ecriture
Ecriture du mot contenu dans R_3 dans la mémoire à l'adresse $R_0R_1R_2$

Instructions de sauts

JRC d si FLAG_C = 1, PC <- PC + d

JRNC d si FLAG_C = 0, PC <- PC + d

JRZ d si FLAG_Z = 1, PC <- PC + d

JRNZ d si FLAG_Z = 0, PC <- PC + d

JR d PC <- PC + d
Saut inconditionnel

DJNZ d R_7--; si R_7>0, PC <- PC + d
décrémente R_7 puis saut si R7 > 0

Remarque : les déplacements sont relatifs de -2^{11} à $2^{11}-1$ et ne modifient pas FLAG_C et FLAG_Z

Instructions arithmétiques (Flag_C à 1 si débordement ou calcul impossible et FLAG_Z à 1 si résultat nul)

ADD R_0 R_j R_0 <- R_0 + R_j ; j dans[0-7]

ADC R_0 R_j R_0 <- R_0 + R_j + FLAG_C ; j dans[0-7]

MINUS R_0 R_j R_0 <- R_0 - R_j ; j dans[0-7]

MULT R_0 R_j R_0 <- R_0 * R_j ; j dans[0-7]

DIV R_0 R_j R_0 <- R_0 / R_j ; j dans[0-7]

MOD R_0 R_j R_0 <- R_0 % R_j ; j dans[0-7]

INC R_j R_j++ ; j dans[0-7] ; cette instruction ne met pas à jour FLAG_C et FLAG_Z !