Ing1 – Examen d'Architecture des Ordinateurs

Durée : 2H – Aucun document autorisé – Calculatrice Interdite

Exercice 1 – 2 pts

On s'intéresse à la RAM que l'on peut installer sur un ordinateur personnel suivant le type d'adresses manipulées (les mots sont adressés par octets). Justifiez vos réponses.

1.a) Un ordinateur manipulant des adresses 32 bits peut théoriquement gérer une RAM de :

- 2 Go : oui/non ?
- 4 Go : oui/non ?
- 8 Go : oui/non

1.b) Un ordinateur manipulant des adresses 64 bits peut théoriquement gérer une RAM de :

- 2 Go : oui/non ?
- 4 Go : oui/non ?
- 8 Go : oui/non ?

1.c) On prend le cas des adresses 64 bits ; quelle peut-être la taille théorique maximale de sa mémoire virtuelle ?

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

Exercice 2 – 4 pts

On considère les entiers relatifs codés en complément à 2 sur 6 bits.

2.a) Quel est l'intervalle des nombres codables dans ce système ?

2.b) Donner le code binaire des entiers relatifs suivants : -11, +14

3.c) Calculer l'addition -11 + 14 en binaire complément à 2 (donner le détail du calcul). Vous préciserez si le résultat obtenu est correct ou s'il y a dépassement de capacité.

Exercice 3 – 5 pts

On s'intéresse à la traduction en assembleur du programme Java suivant :

```
int[] v1 = new int[8] ;    // données sur 8 x 8 bits à l'adresse 0x100
int[] v2 = new int[8] ;    // données sur 8 x 8 bits à l'adresse 0x110
int[] v3 = new int[8] ;    // données sur 8 x 8 bits à l'adresse 0x120
…
for (int i = 0 ; i < 8 ; i++) {
     v3[i] = v1[i] + v2[i];
}
```

Donner la traduction de la boucle `for` en assembleur. Si vous utilisez des instructions de sauts, remplacez les déplacements par des labels pour éviter de les calculer.

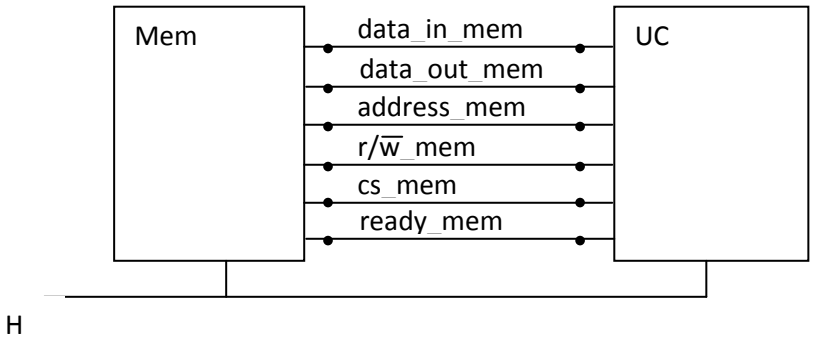
Remarque : Chaque entier est codé sur 2 mots ; les poids faibles sont rangés avant les poids forts. Par exemple, si v1[0] contient l'entier 0xA0, on a en mémoire

Adresse	Donnée
0x100	0x0
0x101	0xA

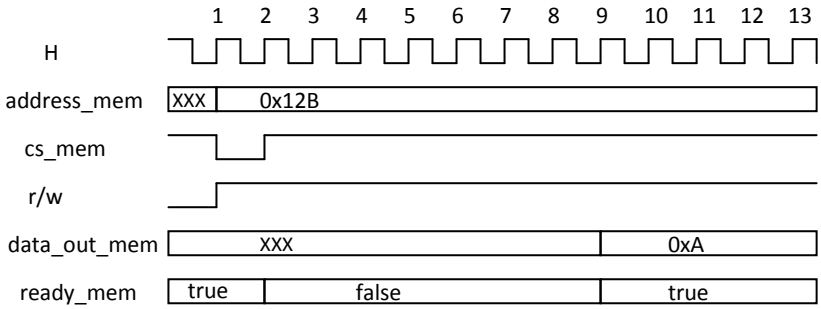
Une fiche récapitulative du langage assembleur utilisé est donnée en annexe.

Exercice 4 – 3 pts

Soit le dialogue entre la mémoire avec cache et l'unité de commande suivant :



L'unité de commande lit la donnée 0xA à l'adresse 0x12B. Le chronogramme de ce dialogue est donné ci-dessous :



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

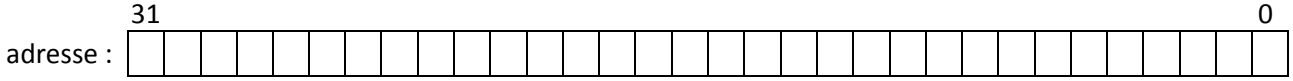
- U1 : Unité de Commande : ordre de lecture
- U2 : Unité de Commande : lecture du résultat (top minimal)
- M1 : Mémoire : prise en compte de l'ordre de lecture
- M2 : Mémoire : sortie du résultat

Exercice 5 – 6 pts

On adjoint à une mémoire principale adressée sur 32 bits une mémoire cache associative par ensemble. Les deux mémoires manipulent des blocs de 16 mots. La mémoire cache possède 32 ensembles de 8 blocs.

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

- le numéro d'un mot dans un bloc (offset)
- le numéro du bloc dans la mémoire pricipale
- le numéro de l'ensemble qui va accueillir le bloc de cette donnée dans le cache
- l'information additionnelle (TAG) que vous devez ajouter à un bloc de donnée pour retrouver la provenance de celui-ci.



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 vide. Montrez l'évolution de l'ensemble représenté par le tableau suivant, en fonction des accès (lecture ou écriture) aux données des blocs suivants : B1, B2, B3, B1, B4, B5, B4, B6, B7, B8, B9, B2, B1, 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).

t0			t1 : accès B1			t2 : accès B2			t3 : accèsB3			t4 : accès B1			t5 : accès B4			t6 : accès B5			t7 : accès B4		
Blocs	V	LRU	Blocs	V	LRU	Blocs	V	LRU	Blocs	V	LRU	Blocs	V	LRU	Blocs	V	LRU	Blocs	V	LRU	Blocs	V	LRU
U	F	U	B1	T	0																		
U	F	U	-	-	-																		
U	F	U	-	-	-																		
U	F	U	-	-	-																		
U	F	U	-	-	-																		
U	F	U	-	-	-																		
U	F	U	-	-	-																		
U	F	U	-	-	-																		
Défauts :			D																				

t8 : accès B6			t9 : accès B7			t10 : accès B8			t11 : accèsB9			t12 : accès B2			t13 : accès B1			t14 : accès B10		
Blocs	V	LRU	Blocs	V	LRU	Blocs	V	LRU	Blocs	V	LRU	Blocs	V	LRU	Blocs	V	LRU	Blocs	V	LRU

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 :

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]