

École Polytechnique de Montréal
Département de génie informatique et génie logiciel

INF1600

Architecture des micro-ordinateurs

TP5 : Assembleur en ligne et mémoire

Soumis par

Tamer Arar, 1828776

Éric Chao, 1827022

Section de labo #1

4 décembre 2016

Contenu	Points
Assembleur en ligne	/2
Mémoire cache	/4
Français écrit erroné	(-1)
Illisibilité du code (peu de commentaires, mauvaise structure...)	(-1)
Format de remise erroné (irrespect des noms de fichiers demandés, fichiers superflus, etc.)	(-1)
Total	/6

Exercice 1 - Assembleur en ligne

Voici notre routine *Decryption_fct()*, en assembleur en ligne, avec seulement 24 instructions.

```
asm volatile (  
    "movl %1, %0\n\t"           // [1,2,3,4,5,6,7,8]  
    "and $0xFF000000, %0\n\t"   // [0,2,3,0,0,0,0,0]  
    "shr $16, %0\n\t"           // [0,0,0,0,0,2,3,0]  
  
    "movl %1, %%ebx\n\t"        // [1,2,3,4,5,6,7,8]  
    "and $0xF000000F, %%ebx\n\t" // [1,0,0,0,0,0,0,8]  
    "rol $20, %%ebx\n\t"        // [0,0,8,1,0,0,0,0]  
  
    "xor %%ebx, %0\n\t"         // [0,0,8,1,0,2,3,0]  
  
    "movl %1, %%ebx\n\t"        // [1,2,3,4,5,6,7,8]  
    "and $0x0000F000, %%ebx\n\t" // [0,0,0,0,5,0,0,0]  
    "shl $16, %%ebx\n\t"        // [5,0,0,0,0,0,0,0]  
  
    "xor %%ebx, %0\n\t"         // [5,0,8,1,0,2,3,0]  
  
    "movl %1, %%ebx\n\t"        // [1,2,3,4,5,6,7,8]  
    "and $0x000F0000, %%ebx\n\t" // [0,0,0,4,0,0,0,0]  
    "shl $8, %%ebx\n\t"         // [0,4,0,0,0,0,0,0]  
  
    "xor %%ebx, %0\n\t"         // [5,4,8,1,0,2,3,0]  
  
    "movl %1, %%ebx\n\t"        // [1,2,3,4,5,6,7,8]  
    "and $0x00000F00, %%ebx\n\t" // [0,0,0,0,0,6,0,0]  
    "shl $4, %%ebx\n\t"        // [0,0,0,0,6,0,0,0]  
  
    "xor %%ebx, %0\n\t"         // [5,4,8,1,6,2,3,0]  
  
    "movl %1, %%ebx\n\t"        // [1,2,3,4,5,6,7,8]  
    "and $0x000000F0, %%ebx\n\t" // [0,0,0,0,0,0,7,0]  
    "shr $4, %%ebx\n\t"        // [0,0,0,0,0,0,0,7]  
  
    "xor %%ebx, %0\n\t"         // [5,4,8,1,6,2,3,7]  
  
    "or %2, %0\n\t"             // key | [5,4,8,1,6,2,3,7]  
  
    : "=a"(decrypted) // sorties (s'il y a lieu)  
    : "c"(crypted), "d"(key) // entrées  
    : "%ebx" // registres modifiés (s'il y a lieu)  
);
```

Exercice 2 - Mémoire Cache

1. Donnez le nombre de bits réservés pour le tag, l'ensemble et l'octet.

Mémoire principale de 1024Ko => 20 bits pour la mémoire principale

Placement direct :

Blocs de cache -> 16 octets -> $2^x = 16 \rightarrow x = 4$ (bits pour l'octet)

1 ensemble -> 1 bloc -> 16 octets. 16Ko/16o => 1K

$2^x = 1024 \rightarrow x = 10$ (bits pour l'ensemble)

$20 - 10 - 4 = 6$ (bits pour le tag)

<i>bits pour le tag</i>	<i>bits pour l'ensemble</i>	<i>bits pour l'octet</i>
6	10	4

Associative par ensemble de 2 blocs :

Blocs de cache -> 16 octets -> $2^x = 16 \rightarrow x = 4$ (bits pour l'octet)

1 ensemble -> 2 bloc -> 32 octets. 16Ko/32o => 512

$2^x = 512 \rightarrow x = 9$ (bits pour l'ensemble)

$20 - 9 - 4 = 7$ (bits pour le tag)

<i>bits pour le tag</i>	<i>bits pour l'ensemble</i>	<i>bits pour l'octet</i>
7	9	4

Associative par ensemble de 4 blocs :

Blocs de cache -> 16 octets -> $2^x = 16 \rightarrow x = 4$ (bits pour l'octet)

1 ensemble -> 1 bloc -> 64 octets. 16Ko/64o => 256

$2^x = 256 \rightarrow x = 8$ (bits pour l'ensemble)

$20 - 8 - 4 = 8$ (bits pour le tag)

<i>bits pour le tag</i>	<i>bits pour l'ensemble</i>	<i>bits pour l'octet</i>
8	8	4



2. Remplissez le tableau pour chaque accès mémoire.

Tableau I. Accès mémoire en tenant compte d'un algorithme de remplacement FIFO et des politiques *write-back* et *write-allocation*.

Accès	Direct				2 blocs				4 blocs			
	Tag	Ens.	Hit	w-b	Tag	Ens.	Hit	w-b	Tag	Ens.	Hit	w-b
WR 0xF7014	61	769			123	257			247	1		
WR 0x1BF02	6	1008			13	496			27	240		
RD 0xF7018	61	769	x		123	257	x		247	1	x	
RD 0x30CCA	12	204			24	204			48	204		
WR 0xFCCCCF	63	204		x	127	204			255	204		
WR 0xF701E	61	769	x		123	257	x	x	247	1	x	
RD 0x48CC1	18	204		x	36	204			72	204		
WR 0xE3019	56	769		x	113	257			227	1		
WR 0xFBFOF	62	1008		x	125	496			251	240		
WR 0xE3017	56	769	x		113	257	x	x	227	1	x	

3. Donnez l'état de la cache à la fin de ces accès.

Tableau II. État de la cache à la fin des accès précédents.

Direct		2 blocs			4 blocs				
Set	Tag0	Set	Tag0	Tag1	Set	Tag0	Tag1	Tag2	Tag3
204	18	204	36	127* 	1	247*	227*	Vide	Vide
769	56*	257	123*	113*	204	48	255*	72* 	Vide
1008	62*	496	13*	125*	240	27*	251*	Vide	Vide

4. En supposant qu'un succès d'accès à la cache prenne 10 ns et qu'un défaut ou un accès à la mémoire principale prenne 160 ns, donnez le temps d'accès effectif de la cache pour la totalité de cette série d'accès.

Le temps d'accès effectif est calculé par :

$$h_p * t_p + h_c * t_c$$

où h_p = le nombre de *hit* sur mémoire principale / le nombre total d'accès

h_c = le nombre de *hit* sur mémoire cache / le nombre total d'accès

t_p = temps d'accès à la mémoire principale



t_c = temps d'accès à la mémoire cache

$$((3/10) * (10\text{ns})) + ((7/10) * (160\text{ns})) = \mathbf{115 \text{ ns}}$$

Le temps d'accès effectif de la cache est de 115 ns.

5. Quel est l'avantage d'augmenter le nombre de blocs ? On en a profité dans cet exercice ?

L'avantage est que cela permet d'augmenter le nombre d'accès à la cache. Ça permet aussi d'augmenter la capacité de stockage en diminuant le nombre de remplacement permettant ainsi moins d'écriture ou de lecture à la mémoire principale. Le temps de traitement est alors diminué.

Nous avons profité de cet avantage, car dans le placement direct, il n'y a qu'un seul bloc, donc il a fallu remplacer des blocs dans la cache et écriture et lire plus souvent dans la mémoire principale (*write-back*). Dans le cas de la mémoire associative à deux blocs, lors du RD 0x48CC1, il a fallu remplacer le bloc ayant le tag 24 par le bloc ayant le tag 36, car il ne restait plus d'espace dans l'ensemble. Finalement, dans la cache associative à quatre blocs, il n'y eu aucun remplacement de blocs ce qui est optimal.