M2P CCI : Corrigé Langage Machine, Novembre 2008

Représenter octet par octet et en hexadécimal le **contenu** de la **section data**.

Contenu de la mémoire pour une machine little endian				
Adresses octets (hexa)	10000	10001	10002	10003
Contenus octets (hexa) LE	61	62	63	64
	←			chaî
Adresses octets (hexa)	10004	10005	10006	10007
Contenus octets (hexa) LE	65	66	67	68
	ne			→
Adresses octets (hexa)	10008	10009	1000a	1000b
Contenus octets (hexa) LE	50	30	70	20
	← x →		← y →	
Adresses octets (hexa)	1000c	1000d	1000e	1000f
Contenus octets (hexa) LE	71	00	00	00
	← z →			

Contenu de la mémoire pour une machine big endian				
Adresses octets (hexa)	10000	10001	10002	10003
Contenus octets (hexa) BE	61	62	63	64
	←			chaî
Adresses octets (hexa)	10004	10005	10006	10007
Contenus octets (hexa) BE	65	66	67	68
	ne			→
Adresses octets (hexa)	10008	10009	1000a	1000b
Contenus octets (hexa) BE	30	50	20	70
	← x →		← y →	
Adresses octets (hexa)	1000c	1000d	1000e	1000f
Contenus octets (hexa) BE	00	00	00	71
	← z →			

Expliquer pourquoi strchr **ne retourne pas NULL** lors de la recherche des caractères **zéro**, **p majuscule** et **espace**. **Quelles adresses** seraient retournées par strchr pour ces caractères sur une machine de type **big endian** ?

Dans la réservation de place, le marqueur de fin de chaîne (octet à 0) a été oublié. Pour strchr, la chaîne se termine donc au premier octet nul après le caractère h, soit le deuxième (little endian) ou le premier (big endian) octet de z. Les octets non nuls de x et y sont donc considérés comme des codes ASCII de caractères faisant partie de la chaîne : 'P' (0x50), '0' (0x30), 'p' (0x70), espace (0x20) et 'q' (0x71).

Strchr recherche les caractères dans la chaîne "abcdefgP0p q" (little endian) ou "bcdefg0P p" (big endian) et trouve donc les caractères '0','P' et ' '.

Pour une machine big endian, les adresses trouvées seraient 0x10009 (P), 0x10008 (0) et 0x1000a (espace).

Contenu de la mémoire pour une machine little endian				
Adresses octets (hexa)	10000	10001	10002	10003
Contenus octets (hexa) LE	61	62	63	64
	←			→
				ch
Adresses octets (hexa)	10004	10005	10006	10007
Contenus octets (hexa) LE	65	66	67	68
	aî			
Adresses octets (hexa)	10008	10009	1000a	1000b
Contenus octets (hexa) LE	00	00	50	30
	→	↔	←	→
	ne	*align*	*x*	
Adresses octets (hexa)	1000c	1000d	1000e	1000f
Contenus octets (hexa) LE	70	20	00	00
	←	→	↔	→
	y		*align*	
Adresses octets (hexa)	10010	10011	10012	10013
Contenus octets (hexa) LE	71	00	00	00
	←			→
	z			

Corriger cette traduction en langage d'assemblage de ces déclarations. **A quelle adresse** (après correction) est stockée la variable z ?

Il suffit de rajouter la marque de fin de chaîne manquante, soit un octet à 0, sans oublier de maintenir l'alignement correct pour les variables qui suivent (adresses paires pour x et y, et multiple de 4 pour z). On rajouter une directive .byte ou utiliser la directive asciz. La variable z est alors stockée à l'adresse 0x10010.

```
chaine:     .byte 'a'
              ...
            .byte 'h'
            .byte 0
            .balign 2
x:          .half  0x3050
y:          .half  0x2070
            .balign 4
z:          .word 0x71
```

```
chaine:     .asciz "abcdefgh"


            .balign 2
x:          .half  0x3050
y:          .half  0x2070
            .balign 4
z:          .word 0x71
```

Contenu de la mémoire pour une machine big endian				
Adresses octets (hexa)	10000	10001	10002	10003
Contenus octets (hexa) BE	61	62	63	64
	←			→
				ch
Adresses octets (hexa)	10004	10005	10006	10007
Contenus octets (hexa) BE	65	66	67	68
		aî		
Adresses octets (hexa)	10008	10009	1000a	1000b
Contenus octets (hexa) BE	00	00	30	50
	→	↔	←	→
	ne	*align*	*x*	
Adresses octets (hexa)	1000c	1000d	1000e	1000f
Contenus octets (hexa) BE	20	70	00	00
	←	→	←	→
	y		*align*	
Adresses octets (hexa)	10010	10011	10012	10013
Contenus octets (hexa) BE	00	00	00	71
	←			→
		z		

1 Question : constructeurs algorithmiques (35mn)

Traduire en langage d'assemblage la boucle while de la fonction **mon_strchr**.

```
/* code C expansé équivalent */
char tmp;
         goto test_w;
corps_w: if (tmp != c) goto finsi;
         resultat = p;
         goto suite;
fin_si:  p++;
test_w:  tmp = *p;
         if (tmp != 0) goto corps_w
suite:   return resultat;
```

D'après la convention d'appel spécifiée dans l'introduction, les paramètres s et c sont passés par les registres r0 et r1. De même, le résultat est supposé retourné dans r0.

```
         .text
         @ s : r0   c: r1   p : r5   resultat : r6   tmp : r8

         bal  test_w
corps_w: cmp r8,r1        @ if (tmp != c) goto finsi
         bne  fin_si
         mov  r6, r5      @ resultat = p
         bal  suite       @ goto suite
```

```
fin_si:   add  r5, r5, #1  @ p++
test_w:   ldrsb r8, [r5]   @ tmp = *p
          cmp  r8, #0      @ if (tmp != 0) goto corps_w
          bne  corps_w
suite:    mov  r0, r6      @ return resultat
```

2 Question : procédures et pointeurs (35 mn)

Traduire en langage d'assemblage l'instruction **wadr ++** ?

La variable wadr est stockée en mémoire et sizeof (wchar_t) = 2 puisque wchar_t est synonyme de short

```
        @ r9, r10 :temporaires
        .text
        ldr r10,= wadr
        ldr r9, [r10]    @ le pointeur contient une adresse : 32 bits
        add r9, r9, #2   @ + 1 * sizeof(wchar_t = 2)
        str r9, [r10]
```

Traduire en langage d'assemblage l'instruction **res = wadr - chaine**.

Réciproquement, pour que la différence entre deux pointeurs corresponde à une différence d'indices entre éléments de tableau, elle doit être implicitement divisée par la taille d'un élément.

```
        .bss
        .skip 2
adr:    .skip 4

        .text
        @  res : r8  chaine : r1  car : r0   tmp1 : r9 tmp2 : r10
        ldr  r10,=adr
        ldr  r9, [r10]      @ r9 = *&adr   (r9 = adr)
        sub  r8, r9, r1
        mov  r8, r8, LSR #1 @ LSR 1 correspond à /2  (/sizeof(wchar_t)
```

Traduire en langage d'assemblage l'appel de procédure **wadr = wcsstr (ch,car)**.

```
        .text
        @ on peut detruire le contenu de car, mais
        @ la valeur de chaine doit être sauvegardee pour l'instruction
        @ suivante
```

```
mv  r9, r1       @ sauver chaine
mv  r1, r0       @ c = car
mv  r0, r1       @ s = chaine
bl  wcsstr       @ resultat dans r0
ldr r10,= adr
str r0, [r10]    @ adr = wsstr(...)
mv  r1, r9       @ restaurer le contenu de ch pour la suite
```

Ecrire en C une procédure retournant à la fois l'adresse et l'indice d'un caractère dans une chaîne et **un exemple d'appel** de cette procédure.

Pour modifier une variable, on passe en paramètre un pointeur sur cette variable. Pour modifier un pointeur, on passe en paramètre un pointeur sur ce pointeur.

```
void  proc (const char *chaine, char c,
              char **adresse, int *position)
{
const char *adr;
adr = wcsstr (chaine,c);
*adresse = adr;
*position = adr - chaine;
}

void main (void)
{
const char *pt;
int indice;
proc ("bonjour", 'j', &pt,&indice);
}
```