M2P CCI : examen Langage Machine, Novembre 2008

Deux heures, tous documents et calculatrices autorisés. Ordinateurs (PC) interdits.

Table des matières

1 Introduction (10mn)

On suppose que la **convention d'appel** de procédure ou fonction passe les **quatre premiers paramètres** dans les registres **r0 à r3** et que le **résultat** d'une fonction est retourné **dans r0** à la place du premier paramètre.

La fonction **strchr** reçoit en paramètre une **chaîne** et un **caractère** à rechercher dans cette chaîne de caractères. Elle **retourne** un pointeur contenant l'**adresse** à laquelle se trouve ce caractère dans la chaîne, et NULL si le caractère n'est pas présent dans la chaîne. **Strchr** fait partie de la bibliothèque C standard.

```
/* L'attribut const indique que la chaîne s ne peut être modifiée */
/* et que toute instruction du type *p = ... est interdite        */
/* c aurait aussi pu etre declare de type char                    */

const char * mon_strchr (const char *s, int c)
  {
  register const char *p, *resultat;   /* dans r5 */
  register resultat = NULL;            /* dans r6 */
  p = s;
  while (*p != 0)
    {
    if (*p == c)
      {
      resultat = p; break;  /* trouve : break termine la boucle */
      }
    p++;
```

```
    }
  return resultat;
  }
```

On peut aussi écrire une fonction **nchr** qui retourne l'**indice** du caractère recherché dans la chaîne, et une valeur négative en cas d'absence.

2 Question : déclarations et section data (40mn)

Le programme C à traduire contient les déclarations suivantes :

```
char chaine [] = "abcdefgh";
short x = 0x3050;
short y = 0x8765;
long  z = 0x71;
```

Voici leur traduction en langage d'assemblage (.global omis) :

```
                .data
chaine:         .byte     'a'
                .byte     'b'
                .byte     'c'
                .byte     'd'
                .byte     'e'
                .byte     'f'
                .byte     'g'
                .byte     'h'
x:              .half     0x3050
y:              .half     0x2070
z:              .word      0x71
```

On suppose que la section **data** commence à l'adresse **0x10000** et que la machine est de type **little endian** (la définition de big et little endian est rappelée en annexe).

Question a : Représenter octet par octet et en hexadécimal le **contenu** de la **section data** à partir de l'adresse 0x10004. Vous pouvez utiliser l'annexe 2 comme feuille de réponse si vous le souhaitez.

La traduction en langage d'assemblage ci-dessus étant incorrecte, la fonction strchr retourne des résultats inattendus :

x=	'a'	'c'	'i'	'j'	'0'	'P'	' '	'_'
strchr(x) =	10000	10002	00000	00000	10009	10008	1000b	00000

Question b : 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** ?

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

3 Question : constructeurs algorithmiques (30mn)

Question d : Traduire en langage d'assemblage la boucle while de la fonction **mon_strchr**. Notez que le type des variables est char (et non unsigned char).

4 Question : procédures et pointeurs (40 mn)

Il existe plusieurs standards pour la représentation des caractères spécifiques aux différentes langues [1] (accents, cédille, ˜, ...).

Les formats iso-xx et UTF8.yy sur 8 bits sont spécialisées dans un groupe de langues[2], mais permettent de représenter les caractères avec le type C char. Le format générique unicode n'est pas restreint à un groupe de langues, mais code les caractères sur 16 bits : on utilise alors le type C wchar_t synonyme de short [3].

Les fonctions standard sur les chaînes de caractères sont disponibles dans les deux variantes de codage des caractères (dont strchr et wcsstr pour la recherche d'un caractère dans une chaîne).

Une réalisation possible de la fonction nchr est donnée ci-dessous pour les deux variantes de format.

```
/* Variante pour char */
const char *adr;
int nchr (char car,
          const char *chaine)
  {
  register int res;
  adr = strchr (chaine,car);
  res = adr - chaine;
  return res;
  }
```

```
/* variante pour wchar_t */
const wchar_t *wadr;
int wnchr (wchar_t car,
           const wchar_t *chaine)
  {
  register int res;          /* dans r8 */
  wadr = wcsstr (chaine,car);
  res = wadr - chaine;
  return res;
  }
```

Qeustion e : Comment traduirait-on en langage d'assemblage l'instruction suivante : **wadr++** ?

[1] basées sur l'alphabet, contrairement par exemple à l'écriture du chinois

[2] exemple : iso-latin1 et UTF8.FR pour le français

[3] On trouve dans les fichiers d'en-tête standard cette définition : typedef unsigned short wchar_t ;

Question f : Traduire en langage d'assemblage l'instruction **res = wadr - chaine**. Rappel : wadr est stocké en mémoire.

Question g : Traduire en langage d'assemblage l'appel de procédure **wadr = wcsstr (chaine,car)**.

Question h : 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.

5 Annexe 1 : définition de big et little endian

Soit un entier codé sur 32 bits et soit 4*X l'adresse à laquelle il est stocké en mémoire. L'entier occupe donc quatre octets, d'adresses 4*X, 4*X+1, 4*X+2 et 4*X+3, dont chacun contient deux chiffres hexadécimaux, soit huit bits.

Les paquets de 8 bits des entiers dont la taille dépasse l'octet peuvent être rangés en mémoire par ordre de poids croissant (little endian) ou décroissant (big endian). Les deux méthodes sont utilisées dans les machines actuelles.

A titre d'exemple, voici comment est stocké l'entier 0x12345678 à l'adresse 0x1000 :

Adresse de l'octet	Contenu big endian	Contenu little endian
0x1000	0x12	0x78
0x1001	0x34	0x56
0x1002	0x56	0x34
0x1003	0x78	0x12

6 Annexe 2 : contenu mémoire (tableau à remplir)

Voici un exemplaire de tableau vide que vous pouvez utiliser pour répondre à la question a.

Adresses octets (hexa)	10000	10001	10002	10003
Contenus octets (hexa)				
Adresses octets (hexa)	10004	10005	10006	10007
Contenus octets (hexa)				
Adresses octets (hexa)	10008	10009	1000a	1000b
Contenus octets (hexa)				
Adresses octets (hexa)	1000c	1000d	1000e	1000f
Contenus octets (hexa)				
Adresses octets (hexa)	10010	10011	10012	10013
Contenus octets (hexa)				

Le contenu de la section data après correction ne vous est pas demandé. Mais rien ne vous empêche de remplir le même tableau pour visualiser la correction et répondre à la question c.

Adresses octets (hexa)	10000	10001	10002	10003
Contenus octets (hexa)				
Adresses octets (hexa)	10004	10005	10006	10007
Contenus octets (hexa)				
Adresses octets (hexa)	10008	10009	1000a	1000b
Contenus octets (hexa)				
Adresses octets (hexa)	1000c	1000d	1000e	1000f
Contenus octets (hexa)				
Adresses octets (hexa)	10010	10011	10012	10013
Contenus octets (hexa)				