M2P CCI : examen Langage Machine, Novembre 2014

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

Table des matières

1 Introduction et conventions (5mn, pas de question) 1

2 Questions sur le programme de conversion (1h20mn) 1
2.1 char_to_valchiffre : paramètres valeur et résultat (20mn) 1
2.2 string_to_tabval : while et arithmétique sur les pointeurs (40mn) . . . 2
2.3 Horner10 : boucle for et tableau (20mn) 2

3 Questions diverses (35mn) 2
3.1 Scanf("%s",...) est dangereux ! : chaînes, pile (20mn) 2
3.2 Mémoire, big/little endian (15mn) 4

4 Annexe : le programme de conversion 5

1 Introduction et conventions (5mn, pas de question)

Eviter l'instruction de multiplication mul dans les traductions.

Sauf précision contraire, la convention d'appel utilisée est celle de gcc, à savoir (pour le jeu d'instructions ARM v4) :
- Passage des quatre premiers paramètres dans les registres r0 à r3
- Résultat des fonctions dans r0 à la place du premier argument
- Adresse de retour passée dans le registre lr

Les principales questions portent sur la traduction du programme de conversion de chaîne en entier (équivalent de sscanf(chaine,"%d",...), présenté en annexe. On suppose que les paramètres c (char_to_valchiffre) et string (string_to_tabval) ne contiennent que des caractères ASCII (codes<128).

Le sujet est assez long. Si vous manquez de temps, sacrifiez de préférence une des deux questions de "divers".

2 Questions sur le programme de conversion (1h20mn)

2.1 char_to_valchiffre : paramètres valeur et résultat (20mn)

Traduire en langage d'assemblage la fonction char_to_valchiffre. La gestion de la pile sera simpifiée : pas de fp et les opérations sur la pile seront décrites par un simples commentaires empiler ou dépiler accompagnés de la liste des registres à copier. La constante NULL correspond à l'adresse 0.

2.2 string_to_tabval : while et arithmétique sur les pointeurs (40mn)

Traduire en langage d'assemblage (lignes commentées trad) :

1. la boucle while
2. l'affectation res= pshort - tabval ;

2.3 Horner10 : boucle for et tableau (20mn)

Traduire en langage d'assemblage la boucle for (commentée "trad") de la fonction horner10.

3 Questions diverses (35mn)

3.1 Scanf("%s",...) est dangereux ! : chaînes, pile (20mn)

Rappel : la langage C représente une chaîne de caractères sous la forme d'un tableau de char et un octet à 0 sert de marque de fin de chaîne.

Le programme suivant est censé écrire dans un fichier ma_trace une chaîne de caractères lue au clavier.

```
#include <stdio.h>
#include <string.h>

#define MAX 10  // longueur max de la chaine de l'utilisateur
#define L 200   // Longueur max du nom de fichier de trace

FILE *f;

char chaine_u[MAX+1]="";
char nom_trace [L+1]="ma_trace";

void lire (void)
{
  printf ("Entrer une chaîne de (au maximum)  %d caracteres\n",MAX);
  scanf ("%s",chaine_u);
}

// Demander a l'utilisateur de saisir une chaine
// d'au maximum L caracteres puis l'écrire dans
// un fichier de nom ma_trace.

int main (void)
{
```

```
  lire ();
  f = fopen (nom_trace,"w");
  if (f != NULL)
    fprintf (f,"%s\n",chaine_u);
  return 0;
}
```

Traduire en langage d'assemblage les déclarations de chaine_u et nom_trace et **expliquer** le +1 dans la dimension de chaine_u.

Voici deux exécutions de ce programme. La commande ls | wc -l permet de connaître le nombre de fichiers du répertoire courant.

```
turing> ls
turing> simgcc -Wall ../overflow.c -o overflow
turing> ls | wc -l
1
turing> armrun overflow
Entrer une chaîne de (au maximum)  10 caracteres
cas_normal
turing> ls
ma_trace  overflow
turing> ls | wc -l
2
turing> cat ma_trace
cas_normal
turing> armrun overflow
Entrer une chaîne de (au maximum)  10 caracteres
Ne_jamais!!_faire_confiance_a_l_utilisateur_qui_saisit_une_chaine
turing> cat ma_trace
cas_normal
turing> ls | wc -l
3
```

La deuxième exécution a créé un troisième fichier sans mettre à jour le contenu du fichier ma_trace :

1. **Expliquer** pourquoi
2. **Donner** le nom du nouveau fichier créé ainsi que son contenu

La question suivante est plus difficile (omettre si vous manquez de temps).

Considérons le scénario suivant :

1. main appelle la fonction f
2. f appelle scanf pour remplir un tableau chaîne qui est une variable locale de f

En fournissant une chaîne de caractères adaptée au code de f, un utilisateur peut théoriquement faire exécuter n'importe quelle séquence de code du programme au lieu de la suite du corps de main après l'appel de f.

En se basant sur le principe de gestion des fonctions avec la pile, **expliquer** le principe de réalisation de ce "tour de magie".

3.2 Mémoire, big/little endian (15mn)

Un entier codé sur 32 bits stocké en mémoire à une adresse 4X occupe quatre octets d'adresses consécutives : 4X, 4X+1, 4X+2 et 4X+3. Il existe deux conventions possibles ou "endianness" [1] :

1. Big endian : le poids des octets décroit avec les adresses (l'octet d'adresse 4X contient les bits de poids forts de l'entier).
2. Litte endian : le poids des octets croit avec les adresses (l'octet d'adresse 4X contient les bits de poids faibles de l'entier).

Exemple : stockage de 0x12345678 à l'adresse 0x10000				
Adresses des octets	0x10000	0x10001	0x10002	0x10003
Contenus Big endian	0x12	0x34	0x56	0x78
Contenus Little endian	0x78	0x56	0x34	0x12

La primitive de communication de base d'un réseau informatique est le transfert d'un tableau d'octets entre deux machines ("d'endianness" identiques ou pas).

```
void sw16 (unsigned short *val, unsigned short *res)
{
  register unsigned char *pv;
  register unsigned char *pr;
  // l'adresse val est copiée sans modification dans pv
  // le forceur (unsigned char *) permet d'ignorer la différence de type
  // entre pv et val
  pv=(unsigned char *) val;
  pr=(unsigned char *) res;
  *(pr+1)= *pv;
  *pr    = *(pv+1);
}
```

Expliquer l'utilité de la procédure sw16 pour copier un tableau d'entiers courts entre deux machines.

Ecrire en langage d'assemblage le corps d'une procédure similaire sw32 pour entiers 32 bits (vous pouvez omettre le prologue et l'épilogue de sauvegarde/restauration

1. Quelquefois francisées en "gros boutisme ou petit boutisme".

des resgistres modifiés).

Ecrire en langage d'assemblage une version de sw16 qui utilise des opérations de décalages au lieu de pointeurs.

4 Annexe : le programme de conversion

Le squelette de programme ci-dessous utilise la fonction sscanf pour convertir en une valeur entière une chaîne représentant un nombre (passée en argument de la ligne de commande : accessible en tant que argv[1]).

```
void main (int argc, char *argv[])
{
   // verification de argc omise
   sscanf (argv[1],"%d",&valentier);
   printf ("Conversion : %s donne %d\n",argv[1],valentier);
}
```

Le programme suivant réalise l'équivalent en deux étapes :

1. passer d'une chaîne de caractères chiffres à un tableau contenant les valeurs entières correspondant aux chiffres.
2. calculer la valeur de l'entier en utilisant le schéma de calcul de Horner

```
void main (int argc, char *argv[])
{
   register unsigned int n;
   unsigned short *lesvals;
   unsigned int valentier;

   // verification de argc omise
   n = strlen(argv[1]);  // strlen : string length
   // allocation dynamique de memoire pour le tableau lesvals
   lesvals = malloc (n*sizeof(unsigned short));
   n = string_to_tabval (argv[1],lesvals);
   valentier = horner10 (n,lesvals);
   printf ("valentier = %d\n",valentier);
}
```

La fonction char_to_valchiffre permet de passer d'un caractère à l'entier qu'il représente :

```
#include <stdio.h>
```

```
// Retourne un booléen "c est un caractère chiffre décimal"
// Le 2eme paramètre permet de récupérer l'entier qu'il représente

int char_to_valchiffre (char c, unsigned short *intval)
{
   if (c < '0') return 0;
   if (c > '9') return 0;
   if (intval != NULL) { *intval= c-'0';}
   return 1;
}
```

La fonction string_to_tabval l'appelle pour convertir toute une chaîne en tableau d'entiers :

```
#include "string_to_tabval.h"
#include "char_to_int.h"

// Parcourt la chaine de caracteres chiffres decimaux string
// et remplit le tableau tabval avec les valeurs entieres
// que les caracteres representent.  Exemple : "1324" -> {1,3,2,4}
//
// La conversion s'arrete
// + en fin de chaine
// + sur le 1er caractere qui n'est pas un chiffre decimal
//
// const indique que la fonction ne modifie pas la chaine string
// la fonction retourne le nombre de chiffres convertis

unsigned int string_to_tabval (const char *string,
                               unsigned short *tabval)
{
   register int ok;                     // a stocker dans r4
   register unsigned short *pshort;     // a stocker dans r5
   register unsigned int res;           // a stocker dans r6

   ok=1;
   pshort=tabval;
   while (ok>=1) { // ne pas transformer en while (ok!=0)  // trad
     ok = char_to_valchiffre (*string, pshort);            // trad
     if (ok) { /* Rappel : ==0 : faux, != 0 : vrai         // trad
       string++;                                           // trad
       pshort++;                                           // trad
     }                                                     // trad
   }                                                       // trad
   res= pshort - tabval;                                   // trad
```

```
   return res;
}
```

La dernière fonction calcule la valeur selon le schéma de Horner :

```
#include "horner10.h"

// Convertit un tableau de nb_chfffres entiers chiffres en un entier
// selon le schéma de Horner
// X = ((...((xn-1*10+xn-2)*10+xn-3)*10 ... +x2)*10 +x1)*10+x0
//
// Les chiffres sont stockés par ordre croissant de poids :
//    x0=chiffres[0] , x1=chiffres[1], ... xn-2=chiffres[n-2]

unsigned int horner10 (unsigned int nb_chiffres,
                       unsigned short *chiffres)
{
   register unsigned int res;      // a stocker dans r6
   register unsigned int i;        // a stocker dans r7

   res = 0;
   for (i=0;i<nb_chiffres; i++) {   // trad
     // Note : *10 = *(8+2)
     res = res * 10 + chiffres[i];  // trad
     }                               // trad
   return res;
}
```