DESS CCI : Corrigé Langage Machine, Décembre 2005

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

Les deux premières des quatre questions de la deuxième partie peuvent être traitées dans n'importe quel ordre.

La première partie est une petite question indépendante simple sur les constructeurs algorithmiques qui ne devrait pas prendre plus d'un quart du temps.

1 Constructions algorithmiques

Traduire en langage d'assemblage les déclarations de variables et les instructions C de l'extrait de programme C suivant :

```
/* Declarations */
short int t[7] = {2, 4, 5, 7, -1};
int i = 2;
int j = 3;

/* ... */

/* instructions */
if ((t[i] >= 2) && (t[j] < 4))
  {
  j = j + 1;
  }
i = i * 8;
```

```
          .data
t:        .hword     2
          .hword     4
          .hword     5
          .hword     7
          .hword     -1
          .skip     4

i:        .word      2
j:        .word      3


@          if (t[i] < 2)  goto finsi
@          if (t[i] >= 4) goto finsi
@          j = j + 1
```

```
@finsi: i = i << 3

@ r0 : t
@ r1 : valeur de i
@ r2 : valeur de t[i]
@ r3 : valeur de j
@ r4 : temporaire adresse

          .text
          ldr     r0, =t
          ldr     r3, =i
          ldr     r1, [r3]

          ldrsh     r2, [r0,r1]
          cmp     r2, #2
          blt     finsi

          cmp     r2, #4
          bge     finsi

          ldr     r3, =j
          ldr     r3, [r3]
          add     r3, r3, #1
          str     r3, [r3]

finsi:    mov     r1, r1, LSL #3
          str     r1, [r3]
```

2 Pointeurs et procédures

2.1 Présentation du programme C

```
unsigned short int x5py (unsigned short int x, unsigned short int y)
{
return (...);     /* Retrouvez l'expression C */
                  /* utilisée dans ce return  */
}


unsigned short int xplusy (unsigned short int x, unsigned short int y)
{
return (x+y);
}

void test_x5py()
```

```
{
...                /* On ne vous donne pas le corps de test_x5py */
                   /* Retrouvez-le à partir de test_x5py.s       */

}

void main  ()
{
test_x5py();
appeler ();
printf ("%d <-- %d %d \n", c, a, b);
}
```

Dans le fichier ptr.c, l'une des fonctions est appelée directement, et l'autre via un pointeur de fonction (type fonc *).

```
/*************************************************************/
/* Extrait du fichier ptr.c */
/*************************************************************/

include "x5py.h"

extern unsigned short int d;     /* précise à l'avance le type de d */
                                 /* pour la déclaration de ps       */

unsigned short int x;
unsigned short int *ps = &d;     /* pointeur initialisé */
unsigned short int d=4;          /* vraie déclaration de d */
unsigned short int e=5;
unsigned short int f;
fonc *pointe_fonc = &xplusy;   /* pointeur pour le stockage d'une adresse */
                               /* initialisé à l'étiquette d'une fonction */

void appel_via_pointeur ()
{
d ++;
f = (*pointe_fonc) (d, 9*e);    /* A traduire */
                                /* appelle la fonction pointée */
e--;
}

void appel ()
{
appel_via_pointeur();            /* --> exécutera xplusy */
pointe_fonc = &x5py;
appel_via_pointeur();            /* --> exécutera x5py */
```

```
}
```

2.2 Pointeurs

On rappelle que traduction en langage d'assemblage de la déclaration d'un pointeur est indépendante du type d'objet pointé. L'unique spécificité des pointeurs de fonctions est que l'étiquette utilisée comme valeur initiale est définie dans text au lieu de data ou bss.

Question a : Traduire en langage d'assemblage les déclarations de a, ps, b, c et pointe_fonc.

```
@ Variante avec data et bss                Variante moins correcte
@                                          en mettant tout dans data
@
      .data                      .data
a:    .short    4  1000      a: .short 4
b:    .short    5  1002      b: .short 5
                             c: .skip  2   1004

                             x: .skip  2   1006
      .balign      4
ps:   .word     d  1004                    1008
d:    .short    4  1008                    100A
e:    .short    5  100A                    100C

      .bss
c:    .skip     2  2000
x:    .skip     2  2002
f:    .skip     2  2004      f: .skip  2   100D
      .balign   4  2006                    1010
pointe_fonc: .word xplusy 2008             1010
```

Question b : Donner l'adresse (à la laquelle sera stocké le premier des octets) de chacune des variables (a, b, ... pointe_fonc). (a,ps,b,c,pointe_fonc) = (1000, 1004, 1002, 2000, 2008).

Question c : Traduire en langage d'assemblage l'instruction **f = *ps ;**.

```
@ a mettre dans la section text qui devra inclure un .ltorg
@
      ldr  r0, =ps      @ adresse de ps : &ps
      ldr  r1, [r0]   @ contenu de ps : *&ps
        ldrh r2, [r1]   @ contenu de d pointé par ps
        ldr  r3, =f     @ adresse de f
      strh r2, [r3]   @ valeur de f
```

Question d : Supposons maintenant que ps pointe sur un élément d'un tableau de fonctions. Traduire en langage d'assemblage l'instruction
ps++ ;.

```
@ a mettre dans la section text qui devra inclure un .ltorg
      ldr      r0, =ps            @ adresse de ps (&ps)
      ldr      r1, [r0]        @ contenu de ps (*&ps)
        add      r1, r1, #4      @ contenu modifié
      str      r1, [r0]        @
```

3 Questions sur les procédures

3.1 Appel ordinaire

Le programmeur a traduit manuellement le fichier x5py.c en langage d'assemblage dans le fichier x5py.s.

```
/*************************************************************/
/* Extraits du fichier x5py.s */
/*************************************************************/

          .global test_x5py
test_x5py: @ prologue de la fonction et sauvegarde des registres sont omis

    @ debut du corps de test_x5py
          ldr     r0,=a                              @ instr 1
          ldrh    r0, [r0]                           @ instr 2
          add     r0,r0, #1

          ldr     r1,=b
          ldrh    r1, [r1]
          add     r1, r1, #2

          mov     lr, pc
          ldr     pc, =x5py

          ldr     r1, =c
          strh    r0, [r1]                           @ instr 10
          @ fin du corps de test_x5py

    @ epilogue de la fonction et restauration des registres sont omis
          mov     pc, lr

          .global x5py
```

```
x5py:   @ prologue de la fonction et sauvegarde des registres sont omis

          @ debut du corps de x5py
          add     r4, r0, r0, LSL #2   @ parametres x et y  dans r0 et r1
          add     r0, r4, r1           @ résultat dans r0
          @ fin du corps de test_x5py

          @ epilogue de la fonction et restauration des registres sont omis
          mov     pc, lr
          .ltorg
```

Question a : Retrouver comment s'écrit en C le corps de test_x5py.

```
c = x5py (a+1, b+2);
```

Question b : Existe-t-il une instruction ARM qui pourrait remplacer la séquence **mov lr, pc ; ldr pc,=x5py** ? b x5py (ou bal x5py)

Question c : Expliquer en quelques mots la différence entre un branchement relatif et un branchement absolu. relatif : PC + deplac absolu : PC <- nouvelle adresse

Question d : Quel calcul réalise l'instruction add r4,r0,r0, LSL #2 ?. Retrouver comment s'écrivait en C la fonction x5py. r4 = r0 * 5 return (5*x+y) ;

Question e : Quelle est la valeur de r0 après l'exécution des instructions 1, 2 et 10 du corps de test_x5py ? 1 adresse de a (1000), 2 contenu de a (4) et 10 valeur de a (5)

3.2 Transformation de x5py en procédure

Suite à un malentendu lors de la phase de spécification, x5py a été programmée comme une fonction à deux arguments retournant un entier court de type **unsigned short int x5py** (...), alors qu'elle devait être programmée comme une procédure de type **void x5py** (...).

Question : Réécrire en C x5py comme une procédure, ainsi que l'instruction d'appel de x5py dans test_x5py. Comment est modifié le corps de x5py dans x5py.s ?

```
void x5py (unsigned short int x, unsigned short int y,
           unsigned short int *resultat)
{
* resultat =x*5 + y;
}
x5py (a+1, b+2, &c);
```

```
        add   r4,r0,r0, LSL #2
        add   r4, r4, r1
        strh  r4, [r2]
```

3.3 Appel via un pointeur

Question : Traduire en langage d'assemblage l'instruction

f = (*pointe_fonc) (d, 9*e) ;

```
ldr      r0, =d
ldrh      r0, [r0]
ldr      r1, =e
ldrh    r1, [r1]
add      r1, r1, r1, LSL #3
ldr      r5, = ppinte_fonc
ldr      r5, [r5]
mov      lr, pc
mov      pc, r5
ldr      r5, =f
strh      r0, [r5]
```