Examen final
Corrigé et Barème de correction
INF2600 — Programmation de systèmes sur microprocesseur

Département de génie informatique
École Polytechnique de Montréal

Hiver 2003

Question 1 (3 points)

a. (2 points)

```
        .text
        .global miroir

miroir:
        mov $8,%ecx
        xor %al,%al
        mov 4(%esp),%dl
1:      shl $1,%dl
        rcr $1,%al
        loop 1b
        ret
```

b. (1 point) En utilisant une table de conversion. La table de 256 valeurs prend 256 octets.

Question 2 (3 points)

```
        .text
        .global __8char_intci
        .global get_i__8char_int
        .global add_i__8char_inti
__8char_intci:
        mov 4(%esp), %ecx
        mov 8(%esp), %eax
        mov %al, (%ecx)
        mov 12(%esp), %eax
        mov %eax, 4(%ecx)
        mov %ecx, %eax
        ret
get_i__8char_int:
        mov 4(%esp), %eax
        mov 4(%eax), %eax
        ret
add_i__8char_inti:
        movl 4(%esp),%edx
        movl 4(%edx),%eax
        addl 8(%esp),%eax
        movl %eax,4(%edx)
        ret
```

Question 3 (6 points)

```
int dix = 10;
double arrondir = 0.0000005;

void float2text(float val, char *buf)
{
  int exposant; // contiendra l'exposant
  int temp1, temp2; // variables temporaire pour votre code
  int tempf; // contiendra la valeur sans le signe ni l'exposant
  tempf = val;
  // votre code pour mettre le '+' ou le '-'  (se fait en 5 instructions)
   asm( "mov $'+',%%al\n\t"
        "test %1,%1\n\t"
        "jns 1f\n\t"
        "mov $'-',%%al\n\t"
        "1: stosb\n\t"
        : "=D" (buf)
        : "r" (val), "0" (buf)
        : "eax");
  // votre code pour enlever le signe, enlever le 1,
  // mettre l'exposant à zéro et
  // arrondir la dernière décimale qui sera affichée.
  // Si  val = +-1,f*2^exp , ce code met  tempf = 0,f + 0,0000005
  // (aide: mettre le bon exposant et le bon signe,
  //  soustraire 1 puis ajouter 'arrondir')
   asm( "andl $0x007FFFFF, %0\n\t"
        "addl $0x3F800000, %0\n\t"
        "flds %0n\t"
        "fld1\n\t"
        "fsubrp\n\t"
        "faddl arrondir\n\t"
        "fstps %0\n\t"
        : "=m" (tempf)
        : "0" (tempf)
        : );
  *(buf++) = '1';  // place le caractère '1' où pointe 'buf'
                   // et incrémente le pointeur après
  *(buf++) = ',';
  // votre code pour que l'arrondi à l'entier soit une troncation
  // (se fait en 3 instructions)
```

```
    asm( "fstcw %0\n\t"
         "orw $0x0C00, %0\n\t"
         "fldcw %0\n\t"
         :
         : "m" (temp1)
         : );
  // votre code pour placer les six décimales
  // boucler six fois sur:
  //     multiplier tempf par 10
  //     le prochain caractère est floor(tempf)+'0'
  //     (où 'floor' est la partie entière)
  //     enlever la partie entière de tempf (soustraire)
  // s'assurer qu'il ne reste rien sur la pile du coprocesseur
  // (n'utilisez pas 'finit')
    asm( "mov $6, %%ecx\n\t"
         "1: fimull dix\n\t"
         "fist %1\n\t"
         "fisub %1\n\t"
         "mov %1, %%al\n\t"
         "add $'0', %%al\n\t"
         "fstosb\n\t"
         "loop 1b\n\t"
         "fstp %%st\n\t"
         : "=D" (buf)
         : "m" (temp1), "t" (tempf), "0" (buf)
         : "eax", "ecx" );
  *(buf++) = '*';
  *(buf++) = '2';
  *(buf++) = '^';
  // votre code pour trouver l'exposant (se fait en 3 instructions)
  // Si  val = +-1,f*2^exp , ce code met  exposant = exp
    asm( "shr $23,%0\n\t"
         "and $0xFF, %0\n\t"
         "sub $127, %0\n\t"
         : "=r" (exposant)
         : "0" (val)
         : );
  itoa(exposant, buf, 10);
  // ceci met l'exposant en base dix où pointe 'buf'
}
```

Question 4 (5 points)

```
        .text
        .global produit
produit:
        push %ebp
        mov %esp, %ebp
        pusha
        mov 8(%ebp), %edx  # pointeur vers résultat.m[i][j] (commence avec i=j=0)
        mov 12(%ebp), %esi # pointeur vers a.m[i][0] (commence avec i=0)
        mov 16(%ebp), %edi # pointeur vers b.m[0][j] (commence avec j=0)
        mov $4, %eax    # nombre de lignes
1:      mov $4, %ecx    # nombre de colonnes
2:      fldz            # part avec la valeur zéro
        mov $3*8, %ebx  # k*8, pour l'indice (boucle sur k de 3 à 0)
3:      fldl (%esi, %ebx)       # charge l'élément a.m[i][k]
        fmull (%edi, %ebx, 4) # multiplie par l'élément a.m[k][j]
        faddp           # additionne à la valeur
        sub $8, %ebx    # passe au prochain k
        jnb 3b          # boucle tant que k n'est pas plus petit que 0
        fstpl (%edx)    # place la valeur dans résultat.m[i][j]
        add $8, %edx    # passe à la prochaine adresse pour le résultat.m[i][j+1]
        add $8, %edi    # passe à la prochaine colonne (b.m[0][j+1])
        loop 2b         # boucle pour chaque colonne
        sub $4*8, %edi  # revient au début de la ligne (b.m[0][0])
        add $4*8, %esi  # passe à la prochaine ligne (a.m[i+1][0])
        dec %eax
        jnz 1b          # boucle pour chaque ligne
        popa
        pop %ebp
        ret
```

Question 5 (3 points + 1 point bonus)

a. (2 points)

```
        .text
        .global setPixel
setPixel:
        mov 4(%esp), %ecx       # image
        mov 12(%esp), %eax      # y
        mull (%ecx)             # * longueur
        addl 8(%esp), %eax      # + x
        mov 8(%ecx), %ecx       # data
        mov 16(%esp), %edx      # c
        mov %edx, (%ecx, %eax, 4) # place la couleur au bon endroit
        ret
```

b. (1 point) Seul le nom de la fonction va changer, selon le *name mangling* puisque le pointeur `this` est passé comme premier (la fonction reçoit donc exactement les mêmes arguments).

c. (1 point bonus) Les déplacements dans la structure `enteteimage` seront changées, puisque le compilateur ajoutera un pointeur vers la table de méthodes virtuelle au début de la structure.

Question 1 (3 points)

Miroir d'un nombre.

Soit un nombre sur huit bits ABCDEFGH (où chaque lettre est un bit, 0 ou 1), vous voulez trouver son miroir, le nombre HGFEDCBA. Ceci peut se faire avec le code C suivant :

```
unsigned char miroir(unsigned char val)
{
  int i;
  unsigned char res;
  res = 0;
  for(i=0; i<8; ++i) {
    res >>= 1;
    if(val & 0x80) res |= 0x80;
    val <<= 1;
  }
}
```

Petite révision de C :

- `X & Y` est le 'et' bit à bit de `X` et `Y`
- `X | Y` est le 'ou' bit à bit de `X` et `Y`
- `X << Y` décale `X` de `Y` bits vers la gauche
- `X >> Y` décale `X` de `Y` bits vers la droite
- `if(x)` la condition `x` est vrai si et seulement si elle est différente de zéro

a. Réécrivez cette routine en assembleur optimisé, sans augmenter la taille mémoire requise. (Se fait en 7 instructions, le compilateur en génère 12 avec le `-fomit-frame-pointer`)

```
        .text
        .global miroir
miroir:
```

b. Comment optimiser plus, si on enlève la contrainte sur la taille ? Qu'est-ce qui prend de la place, et combien ?

Question 2 (3 points)

Méthodes dans une classe. Vous voulez compter les occurrences de certains caractères dans un texte. Pour ça, vous faites une classe comme suit (qui n'est pas de la bonne programmation objet, mais ce n'est pas le but ici ...) :

```
class char_int {
public:
  char c;
  int i;

  char_int(char _c, int _i);
  int get_i();
  void add_i(int x);
};
char_int::char_int(char _c, int _i) { c = _c; i = _i; }
int char_int::get_i() { return i; }
void char_int::add_i(int x) { i += x; }
```

Programmez en assembleur les trois méthodes de la classe (équivalentes à celles présentement écrites en C++).

```
.text
.global __8char_intci
.global get_i__8char_int
.global add_i__8char_inti
```

Question 3 (6 points)

Conversion d'un nombre à virgule flottante en une représentation textuelle. Vous voulez une sous-routine pour imprimer un `float` à l'écran dans le format suivant :

`±1,ffffff*2^±e`

Soit :

- un caractère '+' ou un caractère '-' pour dire si le nombre est positif ou négatif.
- un '1' et une virgule (vous n'avez pas à vérifier si le nombre est dénormalisé ou infini).
- six décimales (en base dix).
- une étoile, un deux et un exposant (le caractère ^ est ici utilisé pour dénoter l'exposant malgré qu'il veut dire autre chose en C).
- l'exposant en base dix (précédé d'un '-' s'il est négatif).

Le `float` est en format IEEE 754 sur 32 bits, pour certaines parties de code vous devez utiliser les différents champs de ce format plutôt que d'utiliser le coprocesseur.

Le programme C suivant est utilisé :

```
int dix = 10;
double arrondir = 0.0000005;

void float2text(float val, char *buf)
{
  int exposant; // contiendra l'exposant
  int temp1, temp2; // variables temporaire pour votre code
  int tempf; // contiendra la valeur sans le signe ni l'exposant
  tempf = val;
  // votre code pour mettre le '+' ou le '-'  (se fait en 5 instructions)

   asm( "_____________________________________________\n\t"

        "_____________________________________________\n\t"

        "_____________________________________________\n\t"

        "_____________________________________________\n\t"

        "_____________________________________________\n\t"

        "_____________________________________________\n\t"

        "_____________________________________________\n\t"

        :_____________________________________________
```

```
      :_____________________________________________

      :_____________________________________________ );
// votre code pour enlever le signe, enlever le 1,
// mettre l'exposant à zéro et
// arrondir la dernière décimale qui sera affichée.
// Si  val = +-1,f*2^exp , ce code met  tempf = 0,f + 0,0000005
// (aide: mettre le bon exposant et le bon signe,
//  soustraire 1 puis ajouter 'arrondir')

 asm( "_____________________________________________\n\t"

      "_____________________________________________\n\t"

      "_____________________________________________\n\t"

      "_____________________________________________\n\t"

      "_____________________________________________\n\t"

      "_____________________________________________\n\t"

      "_____________________________________________\n\t"

      "_____________________________________________\n\t"

      "_____________________________________________\n\t"
      : "=m" (tempf)
      : "0" (tempf)
      : );
*(buf++) = '1';  // place le caractère '1' où pointe 'buf'
                 // et incrémente le pointeur après
*(buf++) = ',';
// votre code pour que l'arrondi à l'entier soit une troncation
// (se fait en 3 instructions)

 asm( "_____________________________________________\n\t"

      "_____________________________________________\n\t"
```

```
      "_____________________________________________\n\t"

      "_____________________________________________\n\t"

      "_____________________________________________\n\t"

      :_____________________________________________

      :_____________________________________________

      :_____________________________________________ );
// votre code pour placer les six décimales
// boucler six fois sur:
//     multiplier tempf par 10
//     le prochain caractère est floor(tempf)+'0'
//     (où 'floor' est la partie entière)
//     enlever la partie entière de tempf (soustraire)
// s'assurer qu'il ne reste rien sur la pile du coprocesseur
// (n'utilisez pas 'finit')

 asm( "_____________________________________________\n\t"

      "_____________________________________________\n\t"

      "_____________________________________________\n\t"

      "_____________________________________________\n\t"

      "_____________________________________________\n\t"

      "_____________________________________________\n\t"

      "_____________________________________________\n\t"

      "_____________________________________________\n\t"

      "_____________________________________________\n\t"

      "_____________________________________________\n\t"
```

```
        "_____\n\t"

        :_____

        :_____

        :_____ );
  *(buf++) = '*';
  *(buf++) = '2';
  *(buf++) = '^';
  // votre code pour trouver l'exposant (se fait en 3 instructions)
  // Si  val = +-1,f*2^exp , ce code met  exposant = exp

   asm( "_____\n\t"

        "_____\n\t"

        "_____\n\t"

        "_____\n\t"

        "_____\n\t"

        :_____

        :_____

        :_____ );
  itoa(exposant, buf, 10);
  // ceci met l'exposant en base dix où pointe 'buf'
}
```

Question 4 (5 points)

Produit matriciel. Vous voulez faire une fonction qui calcule de produit matriciel de deux matrices 4x4 et qui retourne la matrice résultante. Soit les définitions C suivantes :

```
struct mat4x4{
  double m[4][4];
};
struct mat4x4 produit(struct mat4x4 *a, struct mat4x4 *b);
```

Après l'opération, chaque élément `m[i][j]` de la matrice résultante devrait être la somme sur les `k` des `(*a).m[i][k] * (*b).m[k][j]`.

Écrivez la fonction `produit` en assembleur optimisé.

Aides : Votre programme devrait donc être trois boucles imbriquées. Vous n'avez pas besoin de garder les pointeurs `a`, `b` et le résultat, vous pouvez garder des pointeurs vers `(*a).m[i][0]`, `(*b).m[0][j]` et `résultat.m[i][j]`. Si vous parcourez les éléments dans le bon ordre, trouver le pointeur vers le prochain `résultat.m[i][j]` ne demande qu'une addition. Gardez `k*sizeof(double)` comme indice, au lieu de `k`, pour simplifier les accès mémoire. Notez que les dimensions de la matrice sont une petite puissance de 2. On peut soustraire une constante à un pointeur pour revenir au début de la colonne (ou la ligne), s'il pointe vers la fin de la colonne (ou la ligne).

```
        .text
        .global produit
produit:
```

Question 5 (3 points + 1 point bonus)

Pixel. Vous avez une image en format TRGB, et nous la manipulons avec la structure suivante (comme dans le TP) :

```
struct enteteimage
{
  int longueur, hauteur;
  unsigned char *data;
};
```

Puis nous utilisons la structure suivante pour une couleur :

```
struct couleur {
  unsigned char b, g, r, t;
};
```

Nous voulons faire une fonction qui change la couleur d'un pixel de coordonnées `x,y` :

```
void setPixel(enteteimage *image, int x, int y, couleur c);
```

a. Écrivez cette fonction en assembleur (vous n'avez pas à vérifier que x,y se trouve à l'intérieur de l'image) :

```
        .text
        .global setPixel
setPixel:
```

b. Si nous voulions, en C++, mettre cette fonction comme une méthode :

```
class enteteimage
{
  int longueur, hauteur;
  unsigned char *data;
public:
  void setPixel(int x, int y, couleur c);
  // ... autres méthodes non virtuelles pour la classe enteteimage
};
```

Que doit-on changer dans le programme (a) ? Pourquoi ? (vous n'avez pas à faire les modifications, juste dire ce que ça change)

c. [bonus ; pas dit dans les notes de cours] Dans la classe en (b), si on ajoutait des méthodes virtuelles, est-ce que ça changerait le code de la fonction `setPixel` ? Quoi/Pourquoi ?