INF2170
Examen Final, Automne 1999
Ni documents ni calculatrices

1. (4 points) Quelle est l'utilité du complément à deux?

Le complément à deux permet une représentation simple des nombres entiers positifs et négatifs. De plus, il permet une implantation simple, au niveau matériel, d'un additionneur unique capable de traiter les nombres en complément à deux et en binaire pur.

2. (6 points) Soit `d0` contenant un nombre point-flottant en précision simple, que fait le code suivant?

```
MOVE.L   D0,D1
ADD.L    #$800000,D0
EOR.L    D0,D1
BMI      erreur
```

Ce code additionne 1 à l'exposant, et vérifie que la valeur originelle n'était pas infinie.

3. (9 points) Définir le plus précisément possible les termes suivants.

- pipe-line :**mécanisme au niveau matériel permettant un parallélisme d'exécution des instructions.**
- antémémoire : **mémoire se trouvant entre l'UCT et la mémoire centrale permettant de réduire la latence de réponse de la mémoire centrale.**
- interruption : **mécanisme permettant un arrêt coordonné du processus en exécution pour répondre à une requête externe ou un événement exceptionnel interne généré par le processeur lui-même. L'interruption est suivi par un appel, effectué par le matériel lui-même, à une sous-routine de service.**

4. (10 points) L'extrait de programme ci-dessous, tiré d'un programme de tri bulle, montre une boucle interne. Supposez que la touche + de votre clavier soit cassée et que vous ne puissiez plus taper de signe +; modifiez ce segment de programme pour qu'il fonctionne quand même sans signe plus.

```
precedent   EQU      -2
            move.w   D1,D2             ; boucler
boucle:     cmpm.w   (a0)+,(a1)+       ;   comparer a[j] et a[j+1]
            bge      test              ;   si a[j+1] < a[j]
            move.w   precedent(a0),d3  ;     ichanger
            move.w   (a0),precedent(a0)
            move.w   d3,(a0)
            moveq    #faux,d0          ;     enordre = faux
test:       dbra     D2,boucle         ; fin boucl
```

Solution. Il suffit de remplacer `cmpm.w` par trois instructions.

```
cmp.w  (a0),(a1)
lea    (2,a0),a0
lea    (2,a1),a1
```

Notez le fait essentiel que l'instruction `lea` ne modifie pas les bits du ccr. Ainsi, cette instruction peut être insérée après l'instruction `cmp` sans perturbé les bits testés par l'instruction `bge`.

5. (10 points) Écrivez un sous-programme Octal qui, étant donné une valeur entière sur 32 bits située dans le registre `d0`, produise la représentation octale de la valeur en caractères ASCII dans une zone de mémoire de 11 caractères dont l'adresse est contenue dans le registre `a0`.

6. (10 points) Le sous-programme Facture, présenté dans les notes de cours, reçoit en guise de paramètre une adresse dans le registre `a1`. Cette adresse est celle d'une zone mémoire de trois longs mots, chacun de ces longs mots contenant lui-même une adresse. L'adresse comprise dans le premier long mot est celle d'une zone de cinq mots mémoires comprenant chacun un code entier. L'adresse comprise dans le second long mot est celle d'un mot mémoire comprenant une valeur entière correspondant à une durée. L'adresse comprise dans le troisiéme long mot est celle d'un mot mémoire où l'entier résultat du traitement doit être rangé.

Écrivez les instructions du sous-programme Facture permettant de récupérer les cinq codes et la durée, ainsi que l'instruction permettant de ranger le résultat (que vous supposerez se trouver dans `d0`) à l'endroit prévu.

8. (12 points) Le code ci-dessous a été extrait du livre de Ford et Topp ; il effectue une opération sur une chaîne de caractères. Laquelle? Commentez le code.

```
      xdef          crypt1
crypt1:
      movem.l       d0-d2/a0,-(sp)   ;
crypt_0:                            ;
      move.b        (a0),d0           ;  Charge un octet de la chaîne
      beq.s         crypt_1           ;  Octet zéro => fin de chaîne.
      bchg          #7,d0             ;  inverse bit 7
      bchg          #0,d0             ;  inverse bit 0
      move.b        d0,d1
      move.b        d0,d2
      andi.b        #$db,d0           ;  Bits 2 et 5 à zéro.
      andi.b        #$04,d1           ;  Garde bit 2 seulement.
      lsl.b         #3,d1             ;  Aligne le bit 2 à la position 5.
      andi.b        #$20,d2           ;  Garde bit 5 seulement.
      lsr.b         #3,d2             ;  Aligne le bit 5 à la position 2.
      add.b         d2,d0             ;  Transfert bit 5
      add.b         d1,d0             ;  Transfert bit 2.
      move.b        d0,(a0)+          ;  Replace octet.
      bra.s         crypt_0           ;
 crypt_1:
      movem.l       (sp)+,d0-d2/a0  ;
      rts
      end
```

Tous les octets de la chaîne sont modifiés en appliquant les deux transformations suivantes. La première transformation inverse les bits 0 et 7 ; la seconde, interchange les bits 2 et 5. Il y a en quelque sorte un encryptage de la chaîne, car ces opérations sont réversibles.

9. (17 points) Un coefficient binômial est calculé de la façon suivante.

$$C(n,k) = C(n-1,k-1) + C(n-1,k); C(n,0) = 1; C(n,n) = 1$$

Le pseudocode pour résoudre ce problème est le suivant.

```
Binomial (n, k : Entier): Entier
  si k = 0     retourner 1
  sinon si k = n retourner 1
```

```
  sinon retourner Binomial (n-1, k-1) + Binomial (n-1, k)
Fin Binomial
```

Écrivez la fonction assembleur équivalente.

10. (14 points) Voici un appel de fonction :

```
SUBA.L   #2,sp
PEA      tableau
MOVE.W   D0,-(sp)
JSR      fouille
MOVE.W   (sp)+,D0
```

Écrivez le code de la fonction `fouille` qui cherche la valeur empilée avant l'appel à l'intérieur du `tableau`, dont l'adresse a, elle aussi, été empilée. Le tableau contient 200 mots indexés de 1 à 200. La fonction retourne le premier indice du tableau contenant la valeur recherchée. Si l'élément n'est pas dans le tableau alors la fonction retourne zéro.