Exercice A : Programmation assembleur (5 pts) Numéro :

```
#include "stdio.h"

char * expr = "a b + c *";  // (v1 v2 op1 v3 op2 )  est équivalent à ((v1 op1 v2) op3 v3)
int  rp = 0;                // rp : pointeur de lecture dans l'expression

int  stack[10];             // Pile de calcul
int  sp = 0;                // pointeur de pile : pointe sur la première case vide

int val[3] = {              // valeur des variables
    3,                      // valeur de a
    4,                      // valeur de b
    5,                      // valeur de c
};

void push(int v) {          // empilement d'une valeur dans la pile de calcul
    stack[sp] = v;
    sp = sp + 1;
}

int pop() {                 // dépilement d'une valeur de la pile de calcul
    sp = sp - 1;
    return stack[sp];
}

int isvar(char c) {         // test si l'argument est une variable
    return ((c >= 'a') && (c <= 'c'));
}

char read( char *expr) {   // lecture de l'expression : rend une variable ou un opérateur
    char c;
    do {
        c = expr[rp];
        rp = rp + 1;
    } while ((c != 0) && (c == ' '));
    return c;
}

int calc( char *e) {        // calcul de l'expression e (sans traitement d'erreur)
    int v;
    char x;
    do {
        x = read(e);
        if (isvar(x))
            v = val[x - 'a'];
        if (x == '+')
            v = pop() + pop();
        if (x == '*')
            v = pop() * pop();
        push(v);
    }
    while (x);
    return pop();
}

int main() {                // test
    int r;
    r = calc( expr);
    printf("%s = %d\n", expr, r);
    return 0;
}
```

A1) (1 pt) Écrivez les directives permettant l'allocation dans la section .data des 5 variables globales (expr, rp, stack, sp et val).

```
        .data
expr:   .asciiz "a b + c *"
rp:     .word   0
stack:  .space  40
sp:     .word   0
val:    .word   3, 4, 5
```

A2) (1 pt) Quelle taille occupe les variables expr et val ? Justifiez vos réponses.

expr : 10 octets = 9 pour les caractères et 1 pour le 0 de fin de chaine

val : 12 octets = 3 fois 4 octets

A3) (1 pt) Écrivez le code de la fonction push(), vous pouvez profitez que push() est une fonction terminale. Vous n'utiliserez que des registres temporaires.

```
push:   la      $9,  stack
        la      $10, sp
        lw      $11, ($10)
        sll     $12, $11,  2
        addu    $9,  $9,   $11
        sw      $4,  ($9)
        addiu   $11, $11, 1
        sw      $11, ($10)
        jr      $31
```

A4) (1 pt) La fonction « int `calc(char *e)`» utilise les registres $16 et $17 pour les variables v et x respectivement. **(a)** Donnez les valeurs de nr, nv et na et justifiez. **(b)** Ecrivez le prologue. Vous devez aussi sauver l'argument e dans la pile à la place réservée pour lui par la fonction « `int main()` ».

```
(a) # na = 1
    # nr = 3   $16, $17 et $31
    # nv = 2
(b)
calc:   la      $29, $29, -24   # int calc(int e)
        sw      $31, 20($29)
        sw      $17, 16($29)    # int v;
        sw      $16, 12($29)    # char x;
        sw      $4,  24($29)    # sauvegarde de l'argument de calc
```

A5) (1 pt) Vous allez écrire une partie du code de la fonction « int calc(char *e) » sachant que v et x utilisent respectivement les registres $16 et $17 et que le code de 'a' est 0x61, Vous pouvez utiliser les registres temporaires de vos choix.

```
x = read(e);
if (isvar(x))
    v = val[x - 'a'];
```

```
# x = read(e)
        lw      $4,  16($29)
        jal     read
        add     $17, $0, $2



#   if (isvar(x)) {
        add     $4,  $0, $17
        jal     isvar
        beq     $2,  $0, finsi




#       v = val[x - 'a']
        addiu   $8,  $17, -'a'
        sll     $8,  $8, 2
        la      $16, val
        add     $16, $16, $8
        lw      $16, ($16)



#     }
finsi :
```

Exercice B : Mémoires Cache (5 pts) **Numéro :**

On considère un cache de données de premier niveau write-through à correspondance directe, d'une capacité totale de 8Kio. La ligne de cache a une largeur de 32 octets (8 mots de 32 bits). Les adresses sont sur 32 bits.
Le but de l'exercice est d'analyser le remplissage de ce cache de données L1, puis d'estimer le nombre de cycles nécessaires à l'exécution d'un programme. On suppose que le cache de données est initialement vide. On considère la fonction suivante :

```
int32_t X[512][8];
 ...
void f(int32_t X[256][8]) {
     register int32_t i, j;
     for (j = 7; j >= 0; j -= 1) {
         for (i = 0; i < 256; i += 1) {
             X[i][j] = X[i][j] / 2;
         }
     }
 }
```

Dans tout l'exercice, on suppose que le tableau X est à l'adresse 0x10010000, et qu'il est passé à la fonction f.

Rappels : le mot clé "register" est une directive passée au compilateur pour qu'il place la variable **i** dans un registre plutôt que sur la pile ; la variable **i** reste donc en registre durant toute la durée d'exécution de la fonction et ni sa lecture ni son écriture ne provoquent d'accès au cache de données. Un int32_t est codé sur 4 octets.
Un tableau à deux dimensions TAB[L][C] est organisé en mémoire de la manière suivante : T[0][0], T[0][1], T[0][2]… T[0][C-1], T[1][0], T[1][1], T[1][2], etc.

B1 (0.5 pt). Donner le nombre de bits des champs offset, index et étiquette d'une adresse.

Offset : 5 bits
Index : 8 bits (256 lignes = 8192 / 32)
Etiquette : 19 bits

B2 (1 pt). Quels sont les éléments de X qui peuvent occuper les mots du cache suivants : ligne d'index 0, mot 0 ; et ligne d'index 19, mot 3 ?

Adresse du tableau X alignée sur la taille du cache, et taille de X de 16Kio, i .e. 2 fois la taille du cache.

Case d'index 0 mot 0: X[0][0], X[256][0]

Case d'index 19, mot 3 : X[19][3], X[275][3]

B3 (0.5 pt). Donner, dans l'ordre, les 3 premiers éléments de X lus par la fonction f.

X[0][7], X[1][7], X[2][7]

B4 (1 pt). Représenter dans le schéma ci-dessous la ou les case(s) valide(s) du cache après **3** itérations (1 case contient 1 ligne), en précisant les indexes (seules 3 cases sont représentées), et en mettant dans les parties « Mot x » les éléments de X présents (p. ex : X[12][3]).

Index	Adresse du Mot 0	Mot 7	Mot 6	Mot 5	Mot 4	Mot 3	Mot 2	Mot 1	Mot 0
0	0x10010000	X[0][7]	X[0][6]	X[0][5]	X[0][4]	X[0][3]	X[0][2]	X[0][1]	X[0][0]
1	0x10010020	X[1][7]	X[1][6]	X[1][5]	X[1][4]	X[1][3]	X[1][2]	X[1][1]	X[1][0]
2	0x10010040	X[2][7]	X[2][6]	X[2][5]	X[2][4]	X[2][3]	X[2][2]	X[2][1]	X[2][0]

B5 (1 pt). Calculer, en le justifiant, le nombre de miss de données rencontrées lors de l'exécution de cette fonction.

On charge la moitié des éléments du tableau sans conflit : à la fin des 2 boucles for, le tableau X occupe la totalité du cache, mais aucune ligne n'a été en conflit avec une autre. Il y a donc un miss par ligne, soit 256 miss de données, et ce malgré le fait que le tableau ait été parcouru dans le « mauvais » sens.

B6 (0.5 pt). Donner le taux de miss sur le cache de données pour cette fonction.

Il y a en tout 256 * 8 = 2048 lectures.

Le taux de miss est donc de 256 / 2048 = 12.5%

B7 (0.5 pt). La compilation de ce code a produit une boucle de 9 instructions pour la boucle interne (qui met donc 9 cycles à s'exécuter en présence d'un système mémoire parfait), et 3 en plus pour la boucle externe. On néglige l'effet des miss sur le cache d'instructions. Si un miss de données coute 17 cycles, calculer le nombre de cycles total, puis calculer le nombre moyen de cycles par élément.

Nombre de cycles total = 256 * 8 * 9 + 8 * 3 + 256 * 17 = 22 808

Il y a 2048 éléments, soit une moyenne de 22 808 / 2048 = 11.14 cycles/élément

C1/ Le composant ICU est connecté aux lignes d'interruptions de la façon suivante. Le tty est connecté sur l'entrée irq_in[2] de l'ICU, le timer est connecté sur l'entrée irq_in[3] et le DMA est connecté sur l'entrée irq_in[7]. Comment faut-il programmer le masque de l'ICU pour démasquer uniquement ces trois lignes d'interruptions ?

- 0x00000111
- (1 << 8) | (1 << 4) | (1 << 3)
- => 0x0000008C
- 141

C2/ On suppose qu'une tâche qui s'exécute sur le processeur est interrompue par une interruption matérielle. Donnez la succession des événements qui se produisent dans le système (l'ordre est important).

- Acquittement interruption; Sauvegarde par le processeur du compteur ordinal dans EPC ; Saut du processeur à l'adresse int_handler ; Saut du processeur à l'adresse 0x80000180.
- Saut du processeur à l'adresse 0x80000180 ; Sauvegarde par le processeur du compteur ordinal dans EPC ; Saut du processeur à l'adresse int_handler ; Acquittement interruption.
- Saut du processeur à l'adresse int_handler ; Acquittement interruption ; Sauvegarde par le processeur du compteur ordinal dans EPC ; Saut du processeur à l'adresse 0x80000180.
- => Sauvegarde par le processeur du compteur ordinal dans EPC; Saut du processeur à adresse 0x80000180; Saut du processeur à l'adresse int_handler; Acquittement interruption.

C3/ On suppose que le processeur se branche à l'adresse 0x80000180. On constate que le register CR contient la valeur 0x00000018. Quelle est la cause de l'appel au GIET ?

- => C'est une exception de type "instruction bus error".
- C'est une exception de type "data bus error".
- C'est un appel système.
- C'est une interruption matérielle.

C4/ Lors d'une interruption, d'une exception ou d'un appel système, quels registres sont automatiquement modifiés de manière matérielle par le processeur lui-même ?

- Les registres $29 et $31.
- => Les registres EPC, CR et SR.
- Les registres $26 et $27.
- Les registres $1 à $15, EPC et CR.

C5/ Lorsqu'un programme utilisateur exécute un appel système, quel registre utilise le GIET pour déterminer le numéro de l'appel à effectuer ?

- $4
- => $2
- $31
- $29

C6/ Quelle affirmation concernant l'instruction mfc0 est-elle vraie ?

- Elle doit toujours être suivie d'une instruction "nop", afin de vider le contenu du pipeline.
- De la même façon que mflo, elle permet de récupérer le contenu du registre nommé c0.
- C'est l'instruction de masque booléen de tous les registres systèmes.

=> C'est une instruction habituellement privilégiée permettant de lire les registres systèmes.

C7/ Dans le code du GIET, à quoi sert la ligne de code "_isr_func_t _interrupt_vector[32] = { [0…31] = & _isr_default }", extraite du fichier irq_handler.c ?

- À initialiser les numéros des interruptions avec les valeurs 0 à 31.
- À initialiser la fonction _isr_default avec la bonne valeur.

=> À installer une ISR par défaut, utilisée lorsqu'une interruption non prévue se produit.

- À vérifier que les indices des interruptions sont bien compris entre 0 et 31.

C8/ Laquelle de ces actions ne nécessite pas que le processeur soit en mode superviseur ?

=> Exécuter l'instruction assembleur "syscall" .

- Exécuter l'instruction assembleur "eret" .
- Masque les interruptions matérielles.
- Accéder à l'adresse 0xA0A0A0A0.

C9/ Dans quel registre est stocké l'information qui code le mode courant du processeur (user/kernel) ?

- Dans le registre système CR uniquement.

=> Dans le registre système SR seulement.

- Dans les deux registres systèmes CR et SR.
- Dans aucun de ces registres puisqu'on passe en mode kernel par une interruption.

C10/ Comment un périphérique signale t-il au système qu'il a terminé le traitement qu'on lui a demandé ?

- En générant une exception.

=> En générant une interruption.

- En utilisant un appel système.
- Le logiciel applicatif doit toujours vérifier régulièrement si le traitement est terminé.