Numéro d'anonymat (donné sur votre étiquette)

Examen 2021 – 2022

Architecture des ordinateurs 1 – LU3IN029

Durée : 1h30

Documents autorisés : Aucun document ni machine électronique n'est autorisé à l'exception du mémento MIPS.

Répondre directement sur le sujet. Ne pas désagrafer les feuilles.

Le barème indiqué pour chaque question n'est donné qu'à titre indicatif tout comme le barème total qui sera ramené à une note sur 20 points. Le poids relatif des exercices et des questions par contre ne changera pas.

L'examen est composé de 3 exercices indépendants.

Exercice 1 : Addition et soustraction avec saturation – 9 points

Question 1.1 : 3 points

Rappelez comment, étant donné un mot binaire `A` de `n` bits représentant un entier `p` codé en complément à deux, on peut calculer, le mot binaire de `n` bits représentant de l'opposé de cet entier c'est-à-dire `-p`.

Solution:

-A = not(A) + 1

On considère les mots binaires `m1` = `0b11000111` et `m2` = `0b01001100`. Quels entiers relatifs codés en complément à deux représentent ces deux mots ? Justifiez votre réponse (pas de point sans justification).

Solution:

`m1` = `0b11000111` = -128 + 64 + 7 = -57
`m2` = `0b01001100` = 64 + 8 + 4 = 76

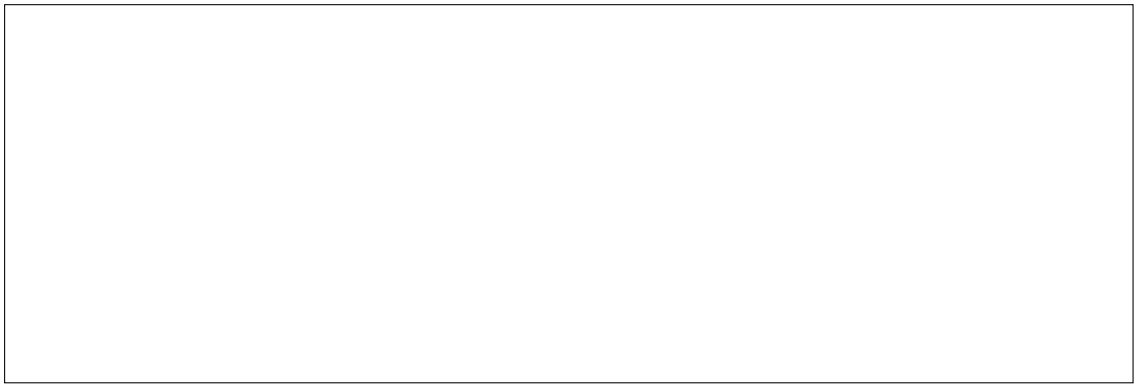
Réalisez la soustraction `m1 - m2`. Cette opération génère-t-elle un dépassement de capacité ? Justifiez votre réponse (pas de point sans justification).

Solution:

```
    0b11000111          0b11000111
-   0b01001100    +     0b10110100
   ------------   =   ------------
                       0b01111011
```

Le résultat est positif alors que les deux entiers additionnées sont négatifs. Il y a un dépassement de capacité. Cela se voit aussi car le résultat vaut -133 qui n'est pas représentable en complément à deux sur 8 bits.

Réalisez la soustraction `m2 - m1`. Cette opération génère-t-elle un dépassement de capacité ? Justifiez votre réponse (pas de point sans justification).



Solution:

```
     0b01001100          0b01001100
 -   0b11000111    +     0b00111001
     ------------    =   ------------
                         0b10000101
```

Le résultat est négatif alors que les deux entiers additionnées sont positifs. Il y a un dépassement de capacité. Cela se voit aussi car le résultat vaut 133 qui n'est pas représentable en complément à deux sur 8 bits.

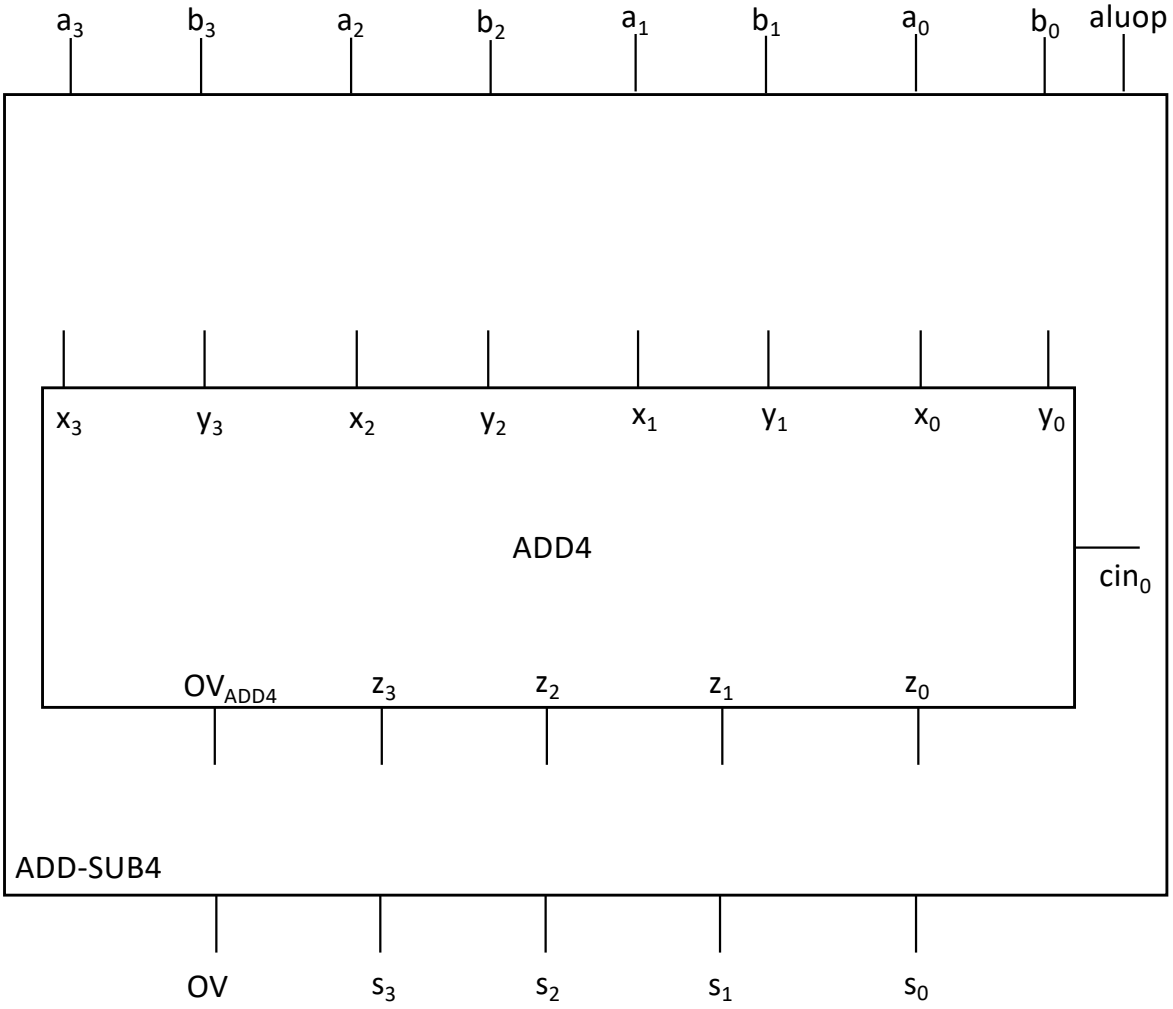
Question 1.2 : 2 points

On considère un additionneur 4 bits, que l'on appelle `ADD4`. Il a deux entrées $\texttt{X}=x_3x_2x_1x_0$ et $\texttt{Y}=y_3y_2y_1y_0$ qui sont des mots binaires de 4 bits, et $\texttt{cin}_0$ la retenue entrante du premier rang. Sa sortie est composée du mot binaire de 4 bits $\texttt{Z}=z_3z_2z_1z_0$ et du drapeau $\texttt{OV}_{\texttt{ADD4}}$ permettant de détecter un dépassement de capacité sur entiers relatifs. Ce drapeau est calculé par l'expression $\texttt{cout}_3 \oplus \texttt{cout}_2$ avec $\texttt{cout}_i$ la retenue du rang sortante du rang i.

On souhaite réaliser un additionneur-soustracteur 4 bits nommé ADD-SUB4 qui a deux entrées $\texttt{A}=a_3a_2a_1a_0$ et $\texttt{B}=b_3b_2b_1b_0$ qui sont des mots binaires de 4 bits et une commande `aluop` qui indique si on souhaite faire une addition ou une soustraction. Sa sortie est composée du drapeau `OV` permettant de détecter un dépassement de capacité et du mot binaire de 4 bits $\texttt{S}=s_3s_2s_1s_0$ qui vaut :

- `A` + `B` si `aluop` = 0
- `A` - `B` si `aluop` = 1

Complétez la figure ci-dessous avec une réalisation possible de ADD-SUB4 à partir de ADD4. Vous ne pouvez utiliser que des opérateurs de type AND, OR, NOT, XOR ou des multiplexeurs pour cette réalisation. Vous utiliserez les représentations schématiques usuelles de ces opérateurs.



Solution:

Vu en TP...

Question 1.3 : 2 points

Donnez une expression du drapeau `OV` de ADD-SUB4 qui ne dépend que des bits de signe de ses entrées et sortie `A`, `B` et `S` ainsi que de `aluop`.

Solution:

$\texttt{OV} = \overline{\texttt{aluop}}.(\texttt{a}_3.\texttt{b}_3.\overline{\texttt{s}_3} + \overline{\texttt{a}_3}.\overline{\texttt{b}_3}.\texttt{s}_3) + \texttt{aluop}.(\texttt{a}_3.\overline{\texttt{b}_3}.\overline{\texttt{s}_3} + \overline{\texttt{a}_3}.\texttt{b}_3.\texttt{s}_3)$

Question 1.4 : 2 points

On souhaite modifier notre additionneur-soustracteur pour qu'il renvoie, en cas de dépasssement de capacité lors de l'addition avec ADD4, la valeur la plus proche du résultat correcte à savoir le plus petit entier lorsque la valeur attendue est négative et le plus grand entier lorsque la valeur attendue est positive.

Donnez l'expression des bits de sortie $\texttt{s}_3$, $\texttt{s}_2$, $\texttt{s}_1$, $\texttt{s}_0$ de ADD-SUB4 en fonction de (certains) bits des entrées de ADD-SUB4, de (certains) bits de $\texttt{Z}$ et $\texttt{OV}_{\texttt{ADD4}}$ sorties de ADD4 pour réaliser cet additionneur-soustracteur à saturation.

Expliquez avec une phrase rédigée votre réponse.

Solution:

$\texttt{s}_3 = \texttt{OV}.\texttt{a}_3 + \overline{\texttt{OV}}.\texttt{z}_3$
$\texttt{s}_i = \texttt{OV}.\overline{\texttt{a}_3} + \overline{\texttt{OV}}.\texttt{z}_i$ pour i < 3

Exercice 2 : Addition récursive de grands entiers – 21 points

Certains algorithmes manipulent des entiers naturels de grande taille, par exemple de 1024 bits, et réalisent des opérations arithmétiques dessus. On a besoin de fonctions réalisant ces opérations qui ne sont pas réalisables directement par le processeur. Par exemple une fonction qui additionne deux grands entiers naturels et détermine s'il y a un dépassement de capacité.

Dans cet exercice, on considère des entiers naturels de 2^k bits avec $k > 5$ et on les représente par "morceau" avec un tableau d'entiers naturel de 32 bits (type entier non signé en C) de longueur `len` = 2^{k-5}. Par exemple, un entier naturel de 128 bits (soit 2^7 bits) sera représenté par un tableau de 2^2 entiers naturels de 32 bits.

Ainsi, l'entier naturel $\texttt{ABABABABCDCDCDCDEFEFEFEF87878787}_{16}$ sera representé par un tableau comportant dans l'ordre les 4 mots suivant : `0x87878787`, `0xEFEFEFEF`, `0xCDCDCDCD`, `0xABABABAB`. Les 32 bits de poids faible de l'entiers sont rangés dans la case d'indice le plus petit (0), et les 32 bits de poids fort sont rangés dans la case d'indice le plus grand (3) : le rangement est "petit boutien".

Nous allons implémenter une fonction d'addition de deux grands entiers naturels représentés par des tableaux d'entiers naturels de 32 bits.

Le code C correspondant à cette addition réalisée de manière récursive est donné ci-après. La fonction prend en paramètre deux tableaux `n1` et `n2`, le tableau représentant le grand entier résultat `sum` ainsi qu'un indice `i`. La fonction calcule récursivement l'addition des entiers naturels de 32 bits d'indice compris entre `i` et 0 dans les tableaux `n1` et `n2`, et range le résultat dans le tableau `sum`. La retenue correspondant à l'addition des entiers d'indice `i` est renvoyée.

```
unsigned char add_big_uint_rec(unsigned int n1[],  unsigned int n2[],
                               unsigned int sum[], unsigned int i)
{
  unsigned int cout_i;
  unsigned int cin_i = 0;

  if (i > 0){
    cin_i = add_big_uint_rec(n1, n2, sum, i - 1);
  }

  sum[i] = n1[i] + n2[i] + cin_i;
  if (sum[i] < n1[i])     // détection débordement
    cout_i = 1;
  else
    cout_i = 0;
  return cout_i;
}
```

Question 2.1 : 13 points

Donnez le code assembleur correspondant à la fonction `add_big_uint_rec`. Vous agrémenterez votre code de commentaires faisant le lien avec le code source. Vous devez **justifier** dans vos commentaires la taille du contexte alloué pour la fonction.

Le code de la fonction peut être **optimisé** en particulier les variables locales peuvent être optimisées en registre.

Solution:

```
add_big_uint_rec:
   addiu $29, $29, -28  # nr = 0 + na = 4 + $31 + nv = 2 = 7 mots
   sw    $31, 24($29)

   # $4 = n1, $5 = n2, $6 = sum, $7 = i
```

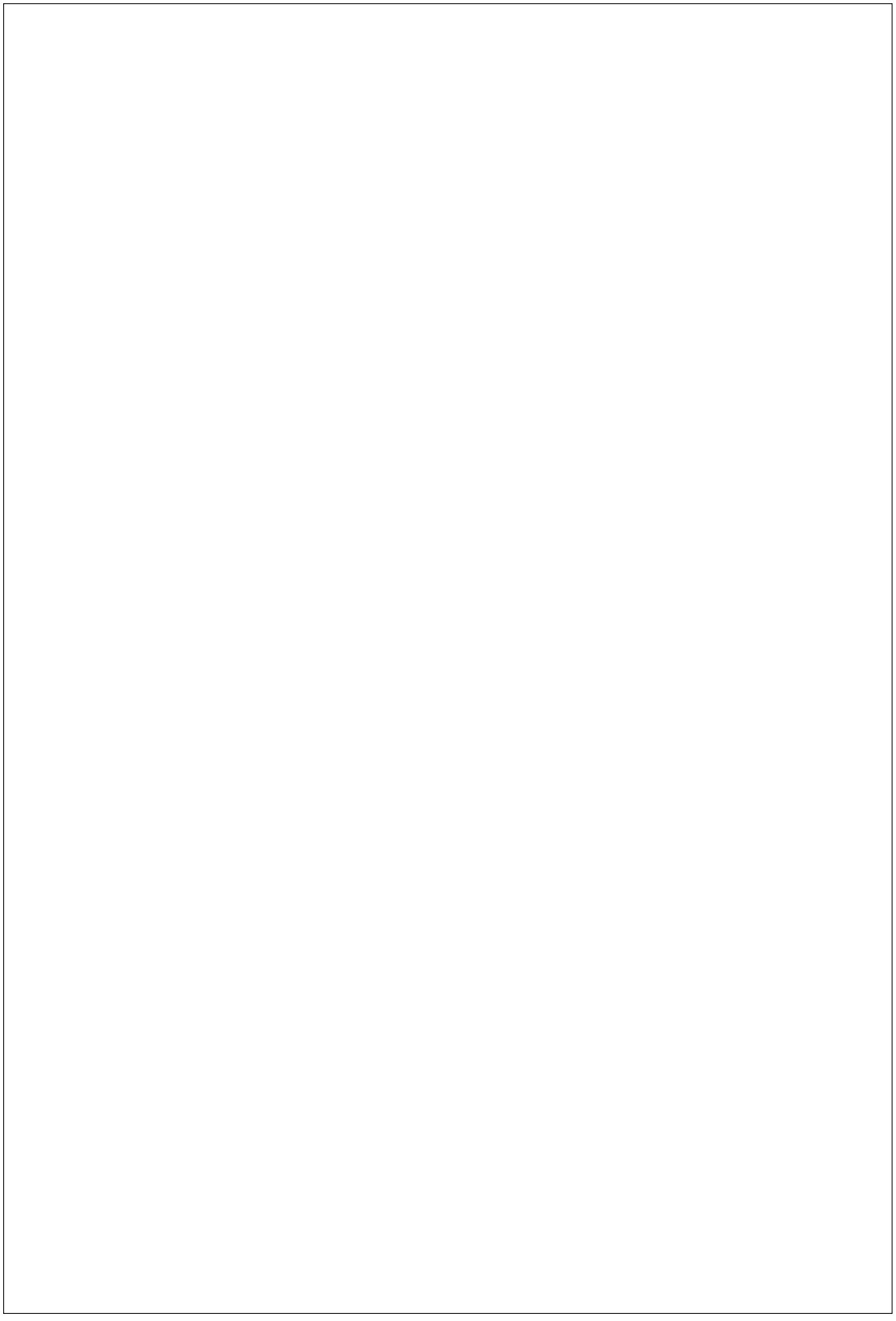
```
   blez  $7, cas_non_rec  # si  i <= 0 saut cas non rec

   # appel rec
   sw    $4, 28($29)        # sauvegarde n1
   sw    $5, 32($29)        # sauvegarde n2
   sw    $6, 36($29)        # sauvegarde sum
   sw    $7, 40($29)        # sauvegarde i

   addiu $7, $7, -1         # 4eme param = i - 1
   jal   add_big_uint_rec
   #$2 contient cin_i
   ori   $8, $2, 0          # cin_i
   lw    $4, 28($29)
   lw    $5, 32($29)
   lw    $6, 36($29)
   lw    $7, 40($29)
   j suite
cas_non_rec:
   xor   $8, $8, $8         # cin_i = 0
suite:
   sll   $7, $7, 2          # i * 4
   addu  $4, $4, $7
   lw    $10, 0($4)         # n1[i]
   addu  $5, $5, $7
   lw    $11, 0($5)         # n2[i]

   addu  $12, $10, $11
   addu  $12, $12, $8       # n2[i] + n1[i] + cin_i
   addu  $6, $6, $7
   sw    $12, 0($6)         # sum[pos] =

   sltu  $13, $12, $10      # $13 = 1 si sum[i] < n1[i]
   beq   $13, $0, no_ov
   ori   $2, $0, 1          # cout = 1
   j epilogue
no_ov:
   ori   $2, $0, 0          # cout = 0
epilogue:
   lw    $31, 24($29)
   addiu $29, $29, 28
   jr    $31
```



Question 2.2 : 4 points

On considère le programme principal donné ci-dessous avec les déclaractions de données globales.

```
unsigned int len = 4;
unsigned int tab1[] = {0xA9000000,   0x3, 0x7ffffffe, 0x80000001};
unsigned int tab2[] = {0x57000000, 0x127,        0x1, 0x7fffffff};
unsigned int res[4];

void main(){
  unsigned int overflow;

  overflow = add_big_uint_rec(tab1, tab2, res, len - 1);
  printf("%d", overflow);

  exit();
}
```

Donnez le code assembleur correspondant à ces déclarations et au `main`. Vous agrémenterez votre code de commentaires faisant le lien avec le code source. Vous devez **justifier** dans vos commentaires la taille du contexte alloué pour le programme principal

Le code du programme principal peut être **optimisé** en particulier les variables locales peuvent être optimisées en registre.

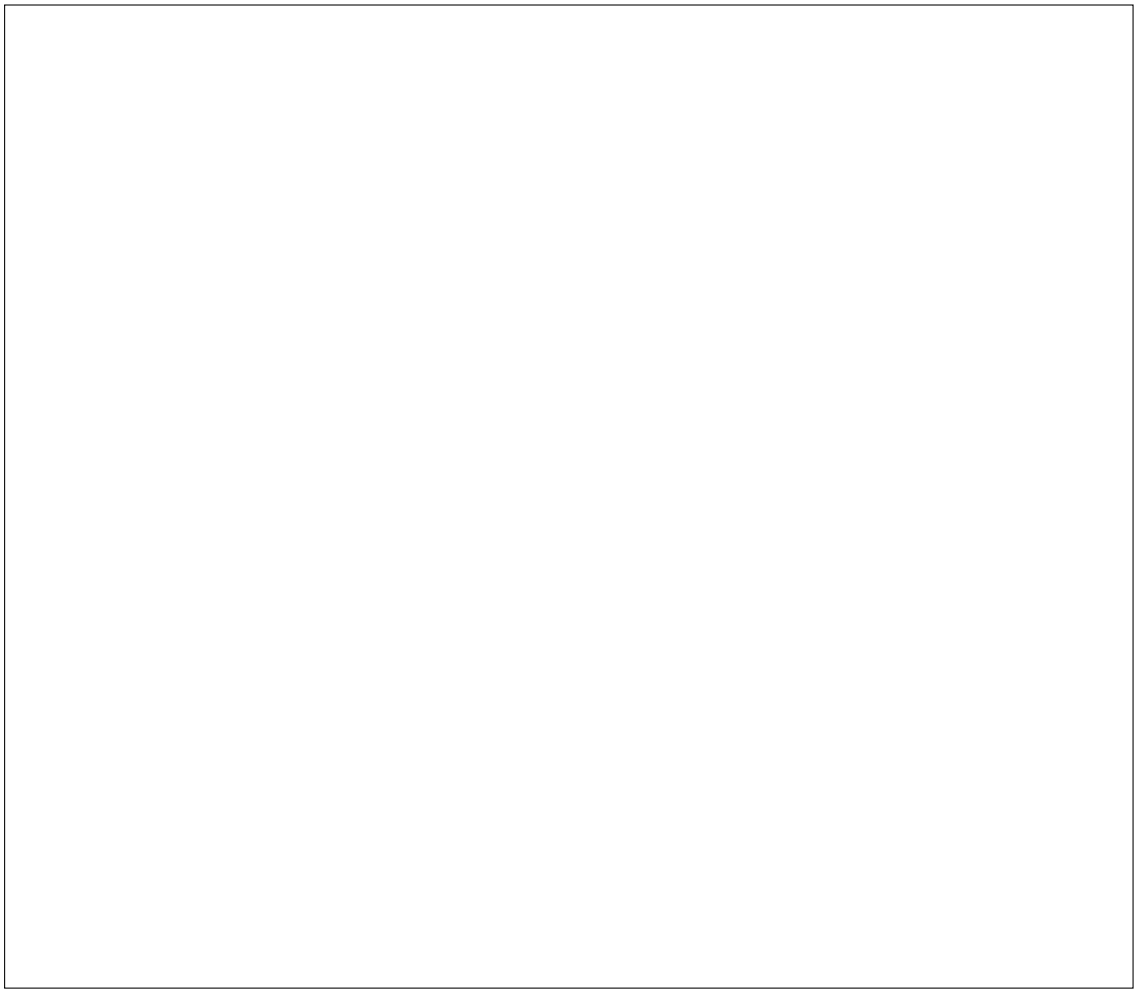
Solution:

```
.data
```

```
len:   .word 4
tab1:  .word 0xA9000000,   0x3, 0x7ffffffe, 0x80000001
tab2:  .word 0x57000000, 0x127,        0x1, 0x7fffffff
res:   .align 2
       .space 16
.text
    addiu $29, $29, -20    # na = 4 + nv = 1
    lui   $4, 0x1001
    ori   $4, $4, 4        # tab1
    ori   $5, $4, 16       # tab2
    addiu $6, $5, 16       # res
    lw    $7, -4($4)       # len
    addiu $7, $7, -1
    jal   add_big_uint_rec

    ori   $4, $2, 1
    ori   $2, $0, 1
    syscall

    addiu $29, $29, 20
    ori    $2, $0, 10
    syscall
```

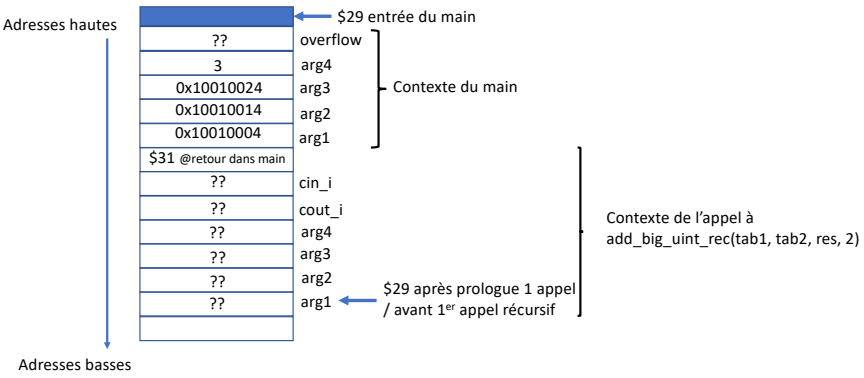


Question 2.3 : 4 points

Sur le schéma ci-dessous indiquez la position du pointeur de pile juste avant le premier appel récursif à la fonction fonction `add_big_uint_rec`.
Pour chaque emplacement (représentant un mot) alloué :

- indiquez à droite de l'emplacement à quoi il correspond,
- indiquez dans l'emplacement son contenu (valeur décimale ou hexadécimale) s'il est connu, s'il est non significatif ou inconnu indiquez ? ?.

Solution:


Adresses hautes
$29 entrée du main
??
overflow
3
arg4
0x10010024
arg3
0x10010014
arg2
0x10010004
arg1
Contexte du main
$31 @retour dans main
??
cin_i
??
cout_i
??
arg4
??
arg3
??
arg2
??
arg1
$29 après prologue 1 appel
/ avant 1er appel récursif
Contexte de l'appel à
add_big_uint_rec(tab1, tab2, res, 2)
Adresses basses

Exercice 3 : Architecture et programmation système – 15 points

Pour chaque question du QCM, vous avez 4 affirmations et vous devez dire, pour chacune, si elle est vraie ou fausse. Toutes les affirmations peuvent être vraies, ou fausses, ou un mélange de vraies et de fausses.

Vous devez cocher 4 cases par question. Pour chaque question, vous avez :
- 1 point si vous avez coché les 4 cases sans erreur.
- 0,5 point si vous avez 1 erreur ou coché seulement 3 cases.
- 0 point si vous avez commis 2 erreurs ou plus ou coché 2 cases ou moins.

Question 3.1 : 8 points

1. **Affirmations sur l'espace d'adressage du MIPS**
   (a) vrai [ ] ou faux [ ]
   Les registres des contrôleurs de périphériques ne sont pas dans l'espace d'adressage du MIPS.
   (b) vrai [ ] ou faux [ ]
   Les registres du processeur sont accessibles dans l'espace d'adressage du MIPS.
   (c) vrai [ ] ou faux [ ]
   L'espace d'adressage n'est accessible que si le MIPS est en mode user.
   (d) vrai [ ] ou faux [ ]
   L'espace d'adressage, c'est l'ensemble des adresses que le MIPS peut produire.

   Solution:
   (a) faux ; (b) faux ; (c) faux ; (d) vrai

2. **Affirmations sur l'architecture vue dans le module**
   (a) vrai [ ] ou faux [ ]
   C'est le noyau du système d'exploitation qui décide du nombre de terminaux TTY du SoC.
   (b) vrai [ ] ou faux [ ]
   Le code de démarrage du MIPS commence à partir de l'adresse 0x80000180.
   (c) vrai [ ] ou faux [ ]
   Les registres des contrôleurs des TTY sont dans la section des variables globales du noyau.
   (d) vrai [ ] ou faux [ ]
   Le noyau choisit au démarrage où sont les segments d'adresses utilisées par les mémoires.

   Solution:
   (a) faux ; (b) faux ; (c) faux ; (d) faux

3. **Affirmations sur les modes d'exécution du MIPS**
   (a) vrai [ ] ou faux [ ]
   L'instruction `stmd` permet de changer le mode d'exécution du MIPS (user et kernel).
   (b) vrai [ ] ou faux [ ]
   L'instruction `syscall` est une instruction utilisable seulement en mode kernel.
   (c) vrai [ ] ou faux [ ]
   C'est le registre `c0_status` qui permet de définir le mode d'exécution du MIPS.
   (d) vrai [ ] ou faux [ ]
   L'instruction `eret` permet de passer du mode kernel au mode user.

   Solution:
   (a) faux ; (b) faux ; (c) vrai ; (d) vrai

4. **Affirmations sur la chaine de compilation**
   (a) vrai [ ] ou faux [ ]
   L'édition de liens produit le fichier exécutable à partir des fichiers objets.
   (b) vrai [ ] ou faux [ ]
   L'outil `gcc` prend exclusivement en argument du code en langage C.
   (c) vrai [ ] ou faux [ ]
   `gcc` appelle le Makefile pour produire l'exécutable grâce à ses règles de dépendances.
   (d) vrai [ ] ou faux [ ]
   L'édition des liens permet à l'application user de connaitre les adresses des fonctions syscall.

   Solution:
   (a) vrai ; (b) faux ; (c) faux ; (d) faux

5. **Affirmations sur le fichier ldscript**
   (a) vrai [ ] ou faux [ ]
   Il y a 1 espace d'adressage, mais 2 fichiers ldscript, 1 pour le noyau et 1 pour l'application user
   (b) vrai [ ] ou faux [ ]
   Le fichier ldscript contient la description des régions de l'espace d'adressage occupées par la mémoire et la manière de les remplir avec les sections présentes dans les fichiers objet (.o).
   (c) vrai [ ] ou faux [ ]
   Les variables définies dans le fichier ldscript sont accessibles depuis le programme C.
   (d) vrai [ ] ou faux [ ]
   Le fichier ldscript est nécessaire pour produire un exécutable à partir d'un programme C

   Solution:
   (a) vrai ; (b) vrai ; (c) vrai ; (d) vrai

6. **Affirmations sur le système d'exploitation**
   (a) vrai [ ] ou faux [ ]
   Le noyau du système d'exploitation est écrit en assembleur.
   (b) vrai [ ] ou faux [ ]
   Le noyau du système d'exploitation a un seul point d'entrée pour l'utilisateur, quelle que soit la cause d'appel.
   (c) vrai [ ] ou faux [ ]
   Seul le noyau peut accéder aux périphériques car c'est lui qui connait les adresses d'implantations des périphériques dans l'espace d'adressage.
   (d) vrai [ ] ou faux [ ]
   Le noyau et la libc du système d'exploitation s'exécutent en mode kernel.

   Solution:
   (a) faux ; (b) vrai ; (c) faux ; (d) faux

7. **Affirmations sur le passage de mode**
   (a) vrai [ ] ou faux [ ]
   Il y a 5 adresses d'entrée du noyau : le boot, kentry et les syscalls, interruptions, exceptions.
   (b) vrai [ ] ou faux [ ]
   Lorsqu'une application user s'exécute, elle peut masquer les interruptions temporairement.
   (c) vrai [ ] ou faux [ ]
   Le noyau utilise le registre $2 pour connaitre la cause d'appel (syscall, interruption, exception)
   (d) vrai [ ] ou faux [ ]
   Les arguments des appels système sont donnés dans les registres $4 à $7.

Solution:
(a) faux ; (b) faux ; (c) faux ; (d) vrai

8. **Affirmations les IRQ**
   (a) vrai [ ] ou faux [ ]
   Le signal IRQ qui sort d'un contrôleur de périphérique contient le numéro de périphérique.
   (b) vrai [ ] ou faux [ ]
   Pour acquitter une IRQ, le noyau doit accéder aux registres du contrôleur de périphérique.
   (c) vrai [ ] ou faux [ ]
   Le vecteur d'interruption est indexé par les numéros d'IRQ.
   (d) vrai [ ] ou faux [ ]
   Les IRQ sont masquées à l'entrée dans le noyau.

   Solution:
   (a) faux ; (b) vrai ; (c) vrai ; (d) vrai

Question 3.2 : 2 points

Lors de l'exécution de l'instruction `syscall`, le MIPS est dérouté vers le noyau du système d'exploitation et les registres `PC, c0_EPC, c0_SR` et `cO_CAUSE` sont modifiés en même temps (l'ordre n'a pas d'importance). Dites quelles sont ces modifications pour chaque registre en les expliquant.

Solution:

- PC $leftarrow$ 0x80000180
  Première adresse du noyau ou entrée du noyau ou encore adresse de `kentry`
- EPC $leftarrow$ PC ou EPC $leftarrow$ adresse du syscall
  EPC contient l'adresse de l'instruction syscall.
- cO_SR.EXL $leftarrow$ 1 ou juste EXL $leftarrow$ 1
  Le MIPS passe en mode d'exception
- c0_CAUSE.XCODE $leftarrow$ 8 ou juste XCODE $leftarrow$ 8
  Le champ XCODE indique la cause d'appel, ici 8 est le code de syscall

Question 3.3 : 5 points

L'écriture et la lecture dans les registres des contrôleurs de périphérique est normalement faite en langage C, mais pour cet examen, nous allons écrire la fonction `int tty_gets(int tty, char buf[])` en assembleur. L'adresse du contrôleur de TTY est `__tty_regs_maps=0xD0200000`, elle est définie dans le fichier '`kernel.ld`' et le label `__tty_regs_maps` est directement utilisable en assembleur. L'ordre des registres de contrôle est (par adresse croissante et pour chaque tty) : `write`, `status`, `read` et `unused`. Chacun de ces 4 registres fait 4 octets, mais seul leur octet de poids faible est utilisé. Les registres de chaque TTY se suivent, d'abord TTY0, puis TTY1, etc.

- `write` est le registre de sortie vers l'écran.
- `status` est le registre qui contient 0 lorsqu'aucune touche n'a été tapée au clavier,
- `read` est le registre qui contient le code ASCII de la touche tapée au clavier.
- `unused` n'est pas utilisé

La fonction `tty_gets(int tty, char buf[])` prend en argument le numéro du TTY et un pointeur sur un buffer de caractères, elle lit les caractères venant du clavier et les écrit dans le buffer et vers le TTY (loopback), jusqu'à rencontrer le caractère `\n`. À la fin, la fonction écrit le caractère de fin de chaine (`0`) dans le buffer et elle rend le nombre de caractères lus.

Complétez le code assembleur suivant (dans le texte directement) et justifiez succinctement chaque réponse (de Q1 à Q5) dans le cadre qui suit (une réponse juste non justifiée n'a pas tous les points).

```
 1:  tty_gets: # int tty_gets(int tty, char buf[])
 2:        ori     $2,     $0,     0
 3:        ori     $10,    $0,     '\n'
 4:        la      $8,     __tty_regs_maps
 5:        sll     $4,     $4,     ________________    --> Q1
 6:        addu    $8,     $8,     $4
 7:  tty_gets_loop:
 8:        lb      $9,     _____________($8)           --> Q2
 9:        beq     $9,     $0,     ________________    --> Q3
10:        lb      $9,     _____________($8)           --> Q4
11:        sb      $9,     0($8)
12:        sb      $9,     0($5)
13:        addiu   $5,     $5,     1
14:        addiu   $2,     $2,     ________________    --> Q5
15:        bne     $9,     $10,    tty_gets_loop
16:  tty_gets_fin:
17:        sb      $0,     0($5)
18:        jr      $31
```

Solution:

```
 1:  tty_gets: # int tty_gets(int tty, char buf[])
 2:        ori     $2,     $0,     0
 3:        ori     $10,    $0,     '\n'
 4:        la      $8,     __tty_regs_maps
 5:        sll     $4,     $4,     ____4___________    --> Q1
 6:        addu    $8,     $8,     $4
 7:  tty_gets_loop:
 8:        lb      $9,     ____4_______($8)           --> Q2
 9:        beq     $9,     $0,     tty_gets_loop      --> Q3
10:        lb      $9,     ____8_______($8)           --> Q4
11:        sb      $9,     0($8)
12:        sb      $9,     0($5)
13:        addiu   $5,     $5,     1
14:        addiu   $2,     $2,     1                  --> Q5
15:        bne     $9,     $10,    tty_gets_loop
16:  tty_gets_fin:
17:        sb      $0,     0($5)
18:        jr      $31
```

- Q1 : il faut multiplier par 16 pour chaque TTY, d'où le décalage à gauche de 4 bits
- Q2 : Il faut lire le status en boucle jusqu'à ce qu'il soit différent de 0
- Q3 : on boucle sur la lecture de status
- Q4 : on lit le registre read
- Q5 : on incrémente le nombre de caractères lus