Numéro d'anonymat (donné sur votre étiquette)

Examen 2023 – 2024

Architecture des ordinateurs 1 – LU3IN029

Durée : 2h00

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 5 exercices indépendants.

Exercice 1 : Circuit Booléen - Distance de Hamming – 8 points

La distance de Hamming DH de deux mots de n bits est un nombre qui est égal au nombre de bits qui diffèrent entre les 2 mots. En d'autres termes, il s'agit du nombre de bits à 1 du résultat du xor des 2 mots. Par exemple sur 8 bits, DH(0xF0, 0xA3) = 4.

La table de vérité suivante définit les valeurs des 2 sorties s_1, s_0, qui encodent l'entier définissant la distance de Hamming des 2 mots (sur deux bits) a_1a_0 et b_1b_0.

Question 1.1 : 2 points

Complétez les tables de vérité des sorties s_1 et s_0.

a_1	a_0	b_1	b_0	s_1	s_0
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	1	0
0	1	0	0		
0	1	0	1		
0	1	1	0		
0	1	1	1		
1	0	0	0		
1	0	0	1		
1	0	1	0		
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		

Solution:

a_1	a_0	b_1	b_0	s_1	s_0
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	0	1
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	1	0	0
0	1	1	0	1	0
0	1	1	1	0	1
1	0	0	0	0	1
1	0	0	1	1	0
1	0	1	0	0	0
1	0	1	1	0	1
1	1	0	0	1	0
1	1	0	1	0	1
1	1	1	0	0	1
1	1	1	1	0	0

Question 1.2 : 2 points

Donner les formes normales disjonctives de s_0 et s_1.

Solution:

$s_0 = \overline{a_1}.\overline{a_0}.\overline{b_1}.b_0 + \overline{a_1}.\overline{a_0}.b_1.\overline{b_0} + \overline{a_1}.a_0.\overline{b_1}.\overline{b_0} + \overline{a_1}.a_0.b_1.b_0 + a_1.\overline{a_0}.\overline{b_1}.\overline{b_0} + a_1.\overline{a_0}.b_1.b_0 + a_1.a_0.\overline{b_1}.b_0 + a_1.a_0.b_1.\overline{b_0}$

$s_1 = \overline{a_1}.a_0.b_1.\overline{b_0} + a_1.\overline{a_0}.\overline{b_1}.b_0 + a_1.a_0.\overline{b_1}.\overline{b_0} + \overline{a_1}.\overline{a_0}.b_1.b_0$

Question 1.3 : 2 points

Montrez que $s_0 = (a_0 \oplus b_0) \oplus (a_1 \oplus b_1)$. Vous procéderez par factorisation de la forme normale disjonctive de s_0 et ferez apparaitre des formes XOR ($\oplus$) et de XNOR ($\overline{\oplus}$).

Solution:

$s_0 = \overline{a_1}.\overline{a_0}.\overline{b_1}.b_0 + \overline{a_1}.\overline{a_0}.b_1.\overline{b_0} + \overline{a_1}.a_0.\overline{b_1}.\overline{b_0} + \overline{a_1}.a_0.b_1.b_0 + a_1.\overline{a_0}.\overline{b_1}.\overline{b_0} + a_1.\overline{a_0}.b_1.b_0 + a_1.a_0.\overline{b_1}.b_0 + a_1.a_0.b_1.\overline{b_0}$

$s_0 = \overline{a_0}.\overline{b_0}.(\overline{a_1}.b_1 + a_1.\overline{b_1}) + \overline{a_0}.b_0.(\overline{a_1}.\overline{b_1} + a_1.b_1) + a_0.\overline{b_0}.(\overline{a_1}.\overline{b_1} + a_1.b_1) + a_0.b_0.(\overline{a_1}.b_1 + a_1.\overline{b_1})$

$s_0 = \overline{a_0}.\overline{b_0}.(a_1 \oplus b_1) + \overline{a_0}.b_0.(\overline{a_1 \oplus b_1}) + a_0.\overline{b_0}.(\overline{a_1 \oplus b_1}) + a_0.b_0.(a_1 \oplus b_1)$

$s_0 = (\overline{a_0}.\overline{b_0} + a_0.b_0).(a_1 \oplus b_1) + (\overline{a_0}.b_0 + a_0.\overline{b_0}).(\overline{a_1 \oplus b_1})$

$s_0 = (\overline{a_0 \oplus b_0}).(a_1 \oplus b_1) + (a_0 \oplus b_0).(\overline{a_1 \oplus b_1})$

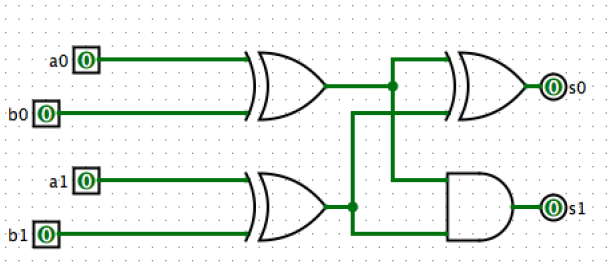
$s_0 = (a_0 \oplus b_0) \oplus (a_1 \oplus b_1)$

Pas demandé, mais par un même raisonnement :
$s_1 = (a_0 \oplus b_0).(a_1 \oplus b_1)$

Question 1.4 : 2 points

On suppose de plus que $s_1 = (a_0 \oplus b_0).(a_1 \oplus b_1)$. Dessinez le schéma correspondant au circuit sous forme de portes logiques AND, OR, XOR, XNOR à deux entrées.

Solution:



Exercice 2 : Instructions assembleur et petit programme – 6 points

Question 2.1 : 3 points

Compléter en hexadécimal (sur 32 bits) les valeurs contenues dans les registres après l'exécution des instructions suivantes :

```
.text

    lui   $8,   0x8000      # $8 =   _______________

    sra   $9,   $8, 16      # $9 =   _______________
```

```
srl  $10, $9, 1        # $10 =  _______________

sll  $11, $10, 12      # $11 =  _______________

ori  $12, $11, 0xBABA  # $12 =  _______________

andi $13, $12, 0xF0F0  # $13 =  _______________
```

Solution:

```
.data

    lui  $8,  0x8000      # $8 =  0x80000000
    sra  $9,  $8, 16      # $9 =  0xFFFF8000
    srl  $10, $9, 1       # $10 = 0x7FFFC000
    sll  $11, $10, 12     # $11 = 0xFC000000
    ori  $12, $11, 0xBABA # $12 = 0xFC00BABA
    andi $13, $12, 0xF0F0 # $13 = 0x0000B0B0
```

Question 2.2 : 3 points

Soit `0xo3o2o1o0` le contenu du registre `$8`. Écrire un programme assembleur MIPS, utilisant UNIQUEMENT des instructions arithmétiques et logiques, qui met dans `$8` la valeur `0xo0o2o1o3`, c'est-à-dire qui permute l'octet de poids faible avec l'octet de poids fort dans le mot. Par exemple, si initialement `$8 = 0xAB1278FF`, en sortie `$8` vaudra `0xFF1278AB`. On pourra utiliser les registres `$9` à `$15` pour les calculs intermédiaires.

Solution:

```
.data

    x : .word 0xAB1278FF

.text

   lui $8, 0x1001
   lw $8, 0($8)

   ori $9, $0, 0xFF # masque pour sélectionner un octet
   and $10, $8, $9  # $10 : octet0
   sll $9, $9, 24
   and $11, $8, $9  # $11 : octet3

   lui $9, 0x00FF
   ori $9, $9, 0xFF00
   and $8, $8, $9   # $8 : octet2 et octet3 préservés, octet0 et octet3 mis à 0

   srl $11, $11, 24 # place l'octet3 en poids faible
   sll $10, $10, 24 # place l'octet0 en poids fort
   or $9, $10, $11  # $9 : octet0 en poids fort et octet3 en poids faible, 00 milieu
   or $8, $8, $9
```

Exercice 3 : Implantation de données et instructions en mémoire – 6 points

On considère l'implantation des instructions suivantes :

Adresse (de mot)	Vue par mot
0x00400000	0x3C081001
0x00400004	0x8D0A0000
0x00400008	0x8D0B0004
0x0040000C	0x014B6021

Question 3.1 : 3 points

Décodez les deux mots binaires implantés aux adresses `0x0040004` et `0x0040000C` : détaillez le contenu de chaque champs et sa signification.

Solution:

à l'adresse `0x00400004`, instruction `0x8D0A0000` :

- soit en binaire : `0b1000 1101 0000 1010 0000 0000 0000 0000`
- CODOP : `0b100011` : mnémonique : LW, donc format I : l'instruction est de la forme `lw Rt,Imm(Rs)`
- Rs : `0b01000` → `$8`
- Rt = `0b01010` → `$10`
- Imm16 : `0b 0000 0000 0000 0000` → offset = 0

Il s'agit de l'instruction `lw $10, 0($8)`.

à l'adresse `0x0040000C`, instruction `0x014B6021` :

- soit en binaire : `0b0000 0001 0100 1011 0110 0000 0010 0001`
- CODOP : `0b000000` : code : SPECIAL → FUNC : `0b100001`, donc mnémonique ADDU : l'instruction est de la forme `addu Rd, Rs, Rt`
- Rs : `0b01010` → `$10`
- Rt = `0b01011` → `$11`
- Rd : `0b01100` → `$12`

Il s'agit de l'instruction `addu $12, $10, $11`.

Question 3.2 : 3 points

Complétez l'implantation des données décrite dans le tableau ci-dessous, qui a été créée par les directives assembleurs suivantes :

```
.data
    a : .half 2, 10
    b : .asciiz "msg"
    c : .byte 0x0A, 0x0B, 0x0C
    d : .word 10, 20, 30
    e : .byte 'o','k'
```

	Vue par octet				Vue par mot
Adresse (de mot)	+ 0	+ 1	+ 2	+3	
0x10010000					
0x10010004					
0x10010008					
0x1001000c					
0x10010010					
0x10010014					
0x10010018					
0x1001001c					

Solution:

	Vue par octet				Vue par mot
Adresse (de mot)	+ 0	+ 1	+ 2	+3	
0x10010000	0x02	0x00	0x0A	0x00	0x000A0002
0x10010004	0x6D	0x73	0x67	0x00	0x0067736D
0x10010008	0x0A	0x0B	0x0C	0x00	0x000C0B0A
0x1001000c	0x0A	0x00	0x00	0x00	0x0000000A
0x10010010	0x14	0x00	0x00	0x00	0x00000014
0x10010014	0x1E	0x00	0x00	0x00	0x0000001E
0x10010018	0x6F	0x6B	0x00	0x00	0x00006B6F
0x1001001c					

Exercice 4 : Programmation assembleur – 16 points

On considère le programme C suivant. Il demande la saisie d'une chaine de caractères au clavier, puis compte le nombre de voyelles de cette chaine de caractères, puis affiche ce nombre. Tous les caractères ont un code ASCII sur un octet.

```
char str[20];
char set[] = "aeiouy";
int cpt = 0;

int est_voyelle( char c, char set[]){
   int j = 0;

   while( set[j] != 0x00) {
      if ( c == set[j] ) {
         return 1;
      }
      j = j+1;
   }
   return 0;
}

void main(){
   int i = 0, res;

   scanf("%s",str);         //appel système de saisie de chaine de caractères
   while ( str[i]!=0x00 ) {
      res = est_voyelle(str[i],set);
      if ( res ) {
         cpt = cpt + 1;
      }
      i = i+1;
   }
   printf("%d",cpt);

   exit();
}
```

Question 4.1 : 2 points

Complétez le schema de la pile ci-dessous, vue après le prologue de la fonction `est_voyelle`.

Adresse (de mot)	contenu (mot)
0xFFFFFFFC	<fond de pile>
0xFFFFFFF8	
0xFFFFFFF4	
0xFFFFFFF0	
0xFFFFFFEC	
0xFFFFFFE8	
0xFFFFFFE4	
0xFFFFFFE0	

Solution:

Adresse (de mot)	contenu (mot)
0xFFFFFFFC	<fond de pile>
0xFFFFFFF8	main :res
0xFFFFFFF4	main :i
0xFFFFFFF0	main :arg1
0xFFFFFFEC	main :arg0
0xFFFFFFE8	voy : $31
0xFFFFFFE4	voy :j
0xFFFFFFE0	

et `$29 = 0xFFFFFFE4`.

Question 4.2 : 6 points

Donner le code assembleur correspondant à la fonction `est_voyelle`. Toute allocation en pile doit être assortie d'un commentaire expliquant la taille allouée. Vous devez respecter les conventions d'allocation en pile et d'appel de fonction. Vous pouvez optimiser votre code, en précisant le rôle des registres que vous utilise, et vous devez commenter votre code.

Solution:

```
est_voyelle :    # retourne 1 si le caractère passé dans $4 est dans l'ensemble implanté à
partir de l'adresse $5
    addiu $29, $29, -8     # $31 + 1 var loc j, initialisée à 0
    sw $31, 4($29)
    sw $0, 0($29)

    sw $5, 12($29)      # svg $5, modifié dans la boucle (et stable dans le main)

boucle2 :
    lb $10, 0($5)
    beq $10, $0, sortie_ko # tous les éléments parcourus sans avoir trouvé str[i]

    # cmp str[i] et set[j]
    beq $4, $10, sortie_ok

    # itération suivante
```

```
    addiu $5, $5, 1
    j boucle2

sortie_ko:
    xor $2, $2, $2
    j epilogue

sortie_ok:
    ori $2, $0, 1

epilogue:
    lw $5,12($29)
    lw $31, 4($29)
    addiu $29, $29, 8
    jr $31
```

Question 4.3 : 8 points

Donner le code assembleur correspondant aux déclarations globales et au programme principal. Toute allocation en pile doit être assortie d'un commentaire expliquant la taille allouée. Vous devez respecter les conventions d'allocation en pile et d'appel de fonction. Vous pouvez optimiser votre code, en précisant le rôle des registres que vous utilisez.

Solution:

```
.data
    str   : .space 20
    set : .asciiz  "aeiouy"
    .align 5
    cpt   : .word 0

.text
main:
    # pile : 1 var locale i, initialisée à 0 et res, + 2 arg
    addiu $29, $29, -16
    sw $0, 12($29)

    # saisie chaÃ®ne, terminée par les caractères 0x0A puis 0x00
    ori $2, $0, 8
    lui $4, 0x1001
    ori $5, $0, 20
    syscall

    lui $18, 0x1001  # &str[0]
    lw $19, 4($29)   # i
    lui $5, 0x1001
    addiu $5, $5, 20 # &set
    lw $15, 32($18)  # cpt

    or $16, $0, $5   # svg de $5 ds registre persistant

boucle:
    # tq str[i] != 0x00
    add $20, $19, $18   # & str[i]
    lb $4, 0($20) # str[i]
    beq $4, $0, apres_boucle

    #appel res = est_voyelle(str[i], set_accent)
    or $5, $0, $16   # restauration de $5 (si modifié dans it précédente de est_voyelle)
    jal est_voyelle  # appel est_accentue

    # if res then ... else ...
    beq $2, $0, fin_boucle # saut si res = 0 (caractère courant est non accentué)

then:
    # cpt = cpt+1
    addiu $15, $15, 1

fin_boucle:
    # i <- i+1 et retour début de boucle
    addiu $19, $19, 1
    j boucle

apres_boucle :
    # affiche cpt
    or $4, $15, $0
    ori $2, $0, 1
    syscall

    # pile et terminaison
    addiu $29, $29, +16
    ori $2, $0, 10
    syscall
```

Architecture et Système (15 points)

QCM : une seule réponse juste par question, cocher lisiblement.
Pour chaque question :

Vous aurez **0** point en cas d'absence de réponse.
+ 0,5 points par réponse juste,
- 0,25 points par réponse fausse

Ne répondez pas au hasard, vous risquez de perdre des points
Si la note est négative pour cet exercice, elle est ramenée à 0.

Architecture matérielle

1. Qu'est-ce qu'un SoC (System on Chip) ?

- ☐ Un processeur spécialisé pour les opérations graphiques.
- ☑ Un circuit contenant un système entier, le processeur, la mémoire et les contrôleurs de périphériques.
- ☐ Un type de mémoire utilisée dans les ordinateurs modernes.
- ☐ Un logiciel utilisé pour le développement de systèmes embarqués.

2. Quelle est la différence entre la mémoire RAM et ROM ?

- ☑ La RAM est volatile et la ROM est non-volatile.
- ☐ La RAM est utilisée pour le stockage à long terme et la ROM pour le stockage temporaire.
- ☐ Il n'y a pas de différence significative.
- ☐ La ROM est plus rapide que la RAM.

3. Quel composant permet de faire des entrées-sorties dans almo1 (le SoC vu en cours) ?

- ☑ Le composant TTY.
- ☐ Le composant TIMER.
- ☐ Le composant ICU.
- ☐ Le composant BUS.

4. Quel composant peut initier une requête de lecture et d'écriture sur le bus dans almo1 ?

- ☑ Uniquement le MIPS.
- ☐ Les mémoires RAM et ROM.
- ☐ Les contrôleurs d'entrées/sorties.
- ☐ L'ICU qui gère déjà les requêtes d'interruption.

5. Quel est le rôle de l'ICU (Interrupt Controller Unit) dans almo1 ?

- ☐ Il sert à préparer les interruptions pour les applications.
- ☑ Il combine les IRQ de tous les contrôleurs de périphérique.
- ☐ Il stocke les données utilisateur.
- ☐ Il contrôle les interruptions provoquées par le MIPS.

6. Qu'est-ce que le registre TTY_STATUS indique dans le contrôleur TTY ?

- ☐ Le nombre de caractères restants dans la mémoire.
- ☑ Si un caractère est en attente de lecture.
- ☐ Les interruptions du terminal TTY.
- ☐ La couleur de l'écran du terminal TTY.

Chaîne de compilation

7. Dans quel fichier se trouve la description de l'espace d'adressage du SoC ?

- ☐ Dans le fichier hcpua.S.
- ☑ Dans le fichier kernel.ld.
- ☐ Dans le fichier Makefile.
- ☐ Dans le fichier kinit.c.

8. Comment imposer le placement d'adresse d'une fonction ou d'une variable en mémoire lors de la production d'un programme exécutable ?

- ☐ En utilisant le compilateur avec des options spéciales.
- ☐ En modifiant le code source de la fonction ou de la variable.
- ☑ En utilisant l'éditeur de lien avec un fichier ldscript.
- ☐ En reprogrammant le système d'exploitation.

9. Pourquoi utilise-t-on le mot clé volatile en C pour les registres de périphériques ?

- ☑ Pour indiquer que la valeur du registre peut changer à tout moment.
- ☐ Pour augmenter la vitesse de traitement du registre.
- ☐ Pour rendre le registre invisible.
- ☐ C'est juste une indication pour le debug, mais ce n'est pas obligatoire.

10. Quel est le rôle de l'éditeur de liens (linker) dans la chaîne de compilation ?

- ☐ Convertir le code source en code objet.
- ☐ Exécuter le programme compilé.
- ☑ Fusionner différents fichiers objets et bibliothèques en un fichier exécutable.
- ☐ Vérifier la syntaxe du code source.

11. Qu'est-ce qu'un fichier objet dans la chaîne de compilation ?

- ☐ Un fichier contenant le code source du programme.
- ☐ Un fichier exécutable prêt à être lancé.
- ☑ Un fichier contenant du code binaire pour le processeur généré par le compilateur.
- ☐ Un script utilisé pour automatiser la compilation.

12. Qu'est-ce que la 'compilation croisée' ?

- ☑ Compiler un programme sur une plateforme pour qu'il s'exécute sur une autre.
- ☐ Compiler un programme en utilisant plusieurs langages de programmation.
- ☐ Compiler un programme pour qu'il fonctionne sur toutes les plateformes sans modification.
- ☐ Une technique pour réduire le temps de compilation en utilisant plusieurs processeurs.

Système d'exploitation

13. Quel est le rôle principal du système d'exploitation dans un ordinateur ?

- ☑ Gérer les ressources matérielles et logicielles et fournir des API pour les applications.
- ☐ Stocker des données de manière permanente.
- ☐ Exécuter des applications de traitement de texte et de tableur.
- ☐ Connecter différents ordinateurs en réseau.

14. Comment une application utilisateur utilise-t-elle les services du kernel ?

- ☐ Par des interruptions matérielles.
- ☑ Grâce aux appels syscall.
- ☐ En accédant directement à la mémoire du kernel.
- ☐ Via des appels de fonctions directes.

15. Comment le kernel se protège-t-il des applications utilisateur dans un système MIPS ?

- ☐ En utilisant des mécanismes de chiffrement.
- ☐ Par le contrôle d'accès basé sur les rôles.
- ☑ Grâce au mode d'exécution du MIPS.
- ☐ En isolant l'application dans un environnement virtuel.

16. Quel est le rôle principal de kentry dans un système d'exploitation ?

- ☐ Gérer la communication entre les applications utilisateur.
- ☑ Servir de point d'entrée principal pour le noyau quelles que soient les causes.
- ☐ Allouer de la mémoire pour les processus.
- ☐ Exécuter des commandes utilisateur.

17. Comment le vecteur de syscall est-il implémenté ?

- ☐ Comme un ensemble de fonctions dans le noyau.
- ☑ Comme un tableau de pointeurs de fonctions.
- ☐ Comme une liste chaînée de commandes.
- ☐ Comme un registre spécial dans le CPU.

18. Quel est le rôle de la fonction _start dans le démarrage d'une application utilisateur ?

- ☐ Elle initialise le système d'exploitation.
- ☑ Elle agit comme le premier point d'entrée du programme utilisateur.
- ☐ Elle configure les périphériques externes.
- ☐ Elle crée un environnement d'exécution pour le kernel.

Gestionnaire d'interruptions

19. Qu'arrive-t-il à l'exécution du programme utilisateur lorsqu'une interruption se produit et que le contrôle est transféré à kentry ?

- ☐ Le programme utilisateur est terminé.
- ☑ L'exécution du programme utilisateur est temporairement suspendue.
- ☐ Le programme utilisateur continue à s'exécuter en parallèle.
- ☐ Le programme utilisateur redémarre.

20. Comment une IRQ (Interrupt ReQuest) est-elle initiée ?

- ☐ Par un utilisateur via une interface graphique.
- ☐ Par un programme lorsqu'il rencontre une erreur.
- ☑ Par un contrôleur de périphérique lorsqu'un événement se produit.
- ☐ Automatiquement à intervalles réguliers par le système d'exploitation.

21. Que se passe-t-il lorsqu'une interruption est traitée ?

- ☐ Le système d'exploitation arrête toutes les autres opérations.
- ☐ L'application en cours est fermée immédiatement.
- ☑ Une routine d'interruption (ISR) est exécutée pour gérer l'événement.
- ☐ Le processeur augmente sa vitesse pour traiter l'interruption.

22. Comment le noyau du système d'exploitation identifie-t-il la routine d'interruption appropriée à appeler lorsqu'une IRQ se produit ?

- ☐ Il sélectionne la routine suivante dans le vecteur d'interruption.
- ☑ Il utilise une table indexée par le numéro d'IRQ contenant les adresses des ISR.
- ☐ Il demande à l'utilisateur de choisir la routine appropriée.
- ☐ Il exécute toujours la même routine quelle que soit l'IRQ.

23. Que signifie masquer une IRQ ?

- ☐ Supprimer l'IRQ
- ☐ Ignorer l'IRQ.
- ☑ Bloquer le signal entre sa source et sa destination.
- ☐ Envoyer l'IRQ à un autre périphérique.

24. Qui peut demander le masquage d'une IRQ ?

- ☐ N'importe quelle application.
- ☑ Seulement le noyau du système d'exploitation.
- ☐ Le contrôleur de périphérique lui-même.
- ☐ L'utilisateur via une interface graphique.

Mode d'exécution et registres spéciaux

25. Quelle instruction spécifique est utilisée pour sortir de kentry et retourner à l'exécution normale après le traitement d'une interruption ?

- ☐ jr c0_epc
- ☐ syscall
- ☑ eret
- ☐ return

26. Quelle est la syntaxe des instructions mtc0 et mfc0 dans le MIPS et peut-on manipuler les registres du coprocesseur 0 avec d'autres instructions ?

- ☐ mtc0 $C0, $GPR et mfc0 $C0, $GPR; oui, on peut utiliser d'autres instructions.
- ☑ mtc0 $GPR, $C0 et mfc0 $GPR, $C0; non, on ne peut pas utiliser d'autres instructions.
- ☐ mtc0 $C0, $GPR et mfc0 $GPR, $C0; non, on ne peut pas utiliser d'autres instructions.
- ☐ mtc0 $GPR, $C0 et mfc0 $C0, $GPR; oui, on peut utiliser d'autres instructions.

27. Que contient le champ XCODE du registre c0_cause ?

- ☐ Il contient le code des instructions en binaire avant exécution.
- ☑ Il contient un code correspondant à la cause d'appel du noyau.
- ☐ Il contient le code de la coordonnée X du curseur de terminal.
- ☐ Ce champ n'existe pas.

28. Quelle est la principale différence entre les modes kernel et user sur un processeur MIPS ?

- ☑ Le mode kernel permet l'accès à tout l'espace d'adressage, le mode user n'accède qu'à une partie.
- ☐ Le mode kernel exécute uniquement le code de boot, le mode user exécute toutes les applications.
- ☐ Le mode kernel est utilisé pour les opérations graphiques, le mode user gère les lignes de commande.
- ☐ Il n'y a pas de différence significative entre les deux modes.

29. Une adresse mappée en mémoire est toujours accessible quel que soit le mode d'exécution du MIPS ?

- ☐ Oui, elle est toujours accessible en lecture et en écriture.
- ☑ Non, elle n'est toujours accessible que si le MIPS est en mode kernel.
- ☐ Oui, mais seulement en mode utilisateur.
- ☐ Non, elle n'est jamais accessible, peu importe le mode.

30. Quel est le rôle des registres c0_sr, c0_cause et c0_epc dans le coprocesseur 0 du MIPS ?

- ☐ Ils contrôlent les opérations arithmétiques du processeur.
- ☐ Ils sont utilisés pour la gestion de l'espace d'adressage.
- ☑ Ils sont impliqués dans la gestion des modes d'exécution et des exceptions.
- ☐ Ils stockent les données de calcul temporaires pour les applications utilisateur.