— Si vous détectez une erreur de sujet, merci de l'indiquer dans votre copie.
— Pour être valables, vos réponses doivent être justifiées.
— Utilisez les noms des variables et des fonctions proposées dans l'énoncé.
— Le barème indicatif est susceptible d'évoluer.

Exercice 1 : Représentation des nombres [6 points]

Question 1 : [2,5 points] Donnez les valeurs **hexadécimales 32** bits affichées par les expressions notées (1) à (8). Expliquez le rôle de la fonction `foo` (pas d'analyse ligne par ligne).

Listing 1 – Evaluation entiers

```
unsigned foo(unsigned n){
  unsigned m = n;
  if (n==0) return 0;
  while ((n & 1) == 0){
    n = n >> 1;
  }
  n =  n >> 1;
  if (n==0)
    return m;
  else
    return 0;
}
```

```
void testEntier(){
  int N1 = 0xFEDC4321;
  int N2 = 0xCDEF1234;
  int N3 = -99;
  printf("(1)%x\n",N1 & N2);
  printf("(2)%x\n",N1 ^ N2);
  printf("(3)%x\n",(N1 | N2)>>1);
  printf("(4)%x\n",(unsigned)(N1 | N2)
     >>1);
  printf("(5)%x\n",N3);
  printf("(6)%x\n",foo(2));
  printf("(7)%x\n",foo(3));
  printf("(8)%x\n",foo(16));
}
```

Question 2 : [2,5 points] Donnez la valeur **hexadécimale 32** bits affichée par (1) et la valeur **réelle** affichée par (2). Expliquez le rôle de la fonction `foo2` (pas d'analyse ligne par ligne).

Listing 2 – Evaluation réels

```
float foo2(float f){
  unsigned bit = *(unsigned *)(&f);
  bit = bit & 0xFF800000;
  return *(float *)(&bit);
}
```

```
void testFloat(){
  float f=99.75,f2;
  f2  = foo2(f);
  printf("(1)%x\n",*(unsigned*)&f);
  printf("(2)%f\n",f2);
}
```

Question 3 : [1 point] Modifiez le code de la fonction `foo` afin d'avoir un paramètre entier signé 32 bits et de retourner un entier 32 bits signé afin d'obtenir le même fonctionnement que la fonction `foo` présentée sur le Listing 1.

```
int foo(int n);
```

Exercice 2 : Assembleur `Mips` *[3 points]*

Complétez le de la traduction de la fonction `foo` du Listing 1 en assembleur et de son appel. Le code de l'appel de la fonction affichera en hexadécimal la valeur retournée par la fonction si le paramètre réel est la valeur mémoire `n`. Ce résultat sera stocké dans la variable mémoire `res`.

Listing 3 – foo en assembleur

```
.data
  n : .word 17
  res : .word 0
.text
  lui $t0,_______ #(1)
  lw _______,0($t0) #(2)
  jal foo
  sw _______,_______($t0) #(3)
  add $a0,$0,_______ #(4)
  addi $v0,$0,34
  syscall
  li $v0,_______ #(5)
  syscall
foo:
  bne $a0,$0,fooNotZero
  add $v0,$0,_______ #(6)
  jr _______ #(7)
fooNotZero:
  sw $t0,0($sp)
  sw $t1,-4($sp)
  addi $sp,$sp,_______ #(8)
  add $t0,$0,_______ #(9)
fooBoucle:
  andi $t1,$a0,_______ #(10)
  bne _______,$0,fooFinBoucle
    #(11)
  srl $a0,_______,_______ #(12)
  beq $0,$0,_____________ #(13)
fooFinBoucle:
  _______ $a0,$a0,1 #(14)
  _______ $a0,$0,fooNotZero2
    #(15)
  add $v0,$0,_______ #(16)
  _______ $0,$0,_____________
    #(17)
fooNotZero2:
  addi $v0,$0,_______ #(18)
fooFin:
  addi $sp,$sp,_______ #(19)
  lw $t0,0($sp)
  _______ $t1,_______($sp)  #(20)
  jr $ra
```

Listing 4 – Programme de test du cache et du pipeline

```
.data
  # 16 caractères avec fin de chaîne
  src : .asciiz "Hello_World!!!!"
  des : .asciiz "xxxxxxxxxxxxxxx"
.text
  lui $a0,0x1001
  ori $a1,$a0,0x10 # inst1
  jal test # inst2
  addi $v0,$0,10
  syscall
test:
  lb $t0,0($a0)
  sb $t0,0($a1)
  addi $a0,$a0,1
  addi $a1,$a1,1
  bne $t0,$0,test #inst3
  jr $ra
```

Exercice 3 : Hiérarchie mémoire [3 points]

Soit un cache pédagogique de 256 octets, organisé par blocs de 4 octets (correspondance directe) puis de 8 octets (correspondance associative par ensemble de 4). Nous souhaitons évaluer les performances de la fonction `test` présentée sur le Listing 4.

Question 1 : [0,5 point] Que fait le programme présenté sur le Listing 4 (pas d'analyse ligne par ligne) ? Combien d'itérations sont effectuées par la fonction `test` ?

Question 2 : [1,5 points] Donnez et expliquez, en détail, la taille en bit de l'étiquette, de l'index et du déplacement dans les deux cas.

Question 3 : [1 point] Donnez et expliquez, en détail, les taux de succès lors de l'accès au cache par la fonction `test` dans les deux cas .

Exercice 4 : Pipeline [3 points]

Question 1 : [0,5 point] Indiquez les dépendances lecture après écriture de la fonction `test` qui peuvent être problématique.

Question 2 : [0,5 point] Expliquez les différences entre branchement au plus tôt et branchement retardé d'un cycle.

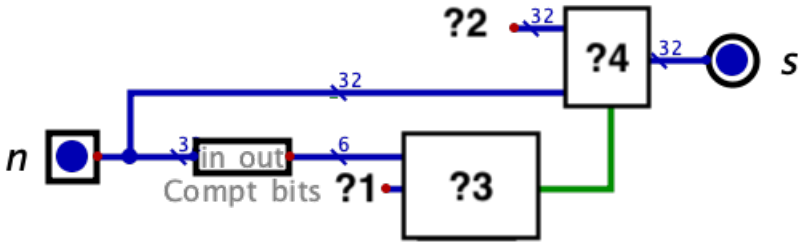
Question 3 : [1 point] Donnez le diagramme temporel pour un pipeline avec envoi et branchement au plus tôt ainsi que le nombre de cycle pour une itération de la fonction `test` si le branchement `#inst3` est pris.

Question 4 : [1 point] Réorganisez le code de la fonction `test` pour ne pas avoir de cycle d'attente avec l'envoi et le branchement retardé d'un cycle sans tenir compte de l'instruction `jr$ra`

Exercice 5 : Architecture [2,5 points]

Question 1 : [1,5 points] Expliquez les valeurs hexadécimales représentant les instructions `#inst1`, `#inst2` et `#inst3` présentées sur le Listing 4.

Question 2 : [1 point] Nous souhaitons implémenter le circuit permettant de reproduire le fonctionnement de la fonction `foo` présentée sur le Listing 1. Donnez les valeurs ou les noms des composants notés `?1` à `?4` . Le composant noté `Compt Bit` permet d'obtenir le nombre de bit à 1 de l'entrée `n`.



Exercice 6 : Exceptions et entrées/sorties [1,5 points]

Question 1 : [0,5 point] Donnez les valeurs contenues dans les registres `$t2` et `$t3` à la fin de l'exécution du programme présenté sur le Listing 5. Expliquez les éventuels problèmes d'exécution.

Listing 5 – Soustraction

```
lui $t0,0x8000
addi $t1,$0,1
subu $t2,$t0,$t1
sub $t3,$t0,$t1
```

Question 2 : [1 point] Donnez le programme permettant de lire un caractère sur la console externe et de placé le résultat dans le registre `$t1`, puis de l'afficher sur la console externe.

Exercice 7 : Rotation [1 point]

Question 1 : [0,5 point] Donnez le code C des fonctions `rotation????` qui permettent de ne pas perdre le bit sortant lors des décalages. Ce bit sortant remplacera le bit mis à 0 dans les opérations de décalage logique à gauche ou à droite de 1 rang.

```
unsigned rotationGauche(unsigned n)
unsigned rotationDroite(unsigned n)
rotationDroite(0x8000001) => 0xC0000000
rotationGauche(0x8000001) => 0x3
```

Question 2 : [0,5 point] Donnez le code assembleur des deux pseudo-instructions permettant de d'effectuer les rotations

```
rol $t2,$t1     // rotation gauche
ror $t2,$t1     // rotation droite
```

Élément de correction (pas de détail, les exercices étant semblables à ceux du du cours)

- Exo1 Q1 : 0xCCCC0220, 0x33335115, 0xFFFFA99A, 0x7FFFA99A, 0xFFFFFF9D, 2,0,10, foo retourne n si c'est une puissance de 2, 0 sinon
- Exo Q2 : 0x42C78000,64, foo2 retourne le plus grand puissance de 2 < f
- Exo1 Q3 : remplacer n = (n > > 1) par n = (n > > 1) & 0x7FFFFFFF
-
- Exo2 : 0x1001, $a0, $v0,4, 10, $ra, -8, $a0, 1, $t1, $a0,1, fooBoucle,srl, beq fooFin, 0, 8, lw -4
-
- Exo3 Q1 : copie chaine, 16 itérations
- Exo3 Q2 : direct 24, 6, 2 assoc ensemble 26, 3, 3
- Exo3 Q3 : direct 3/4 assoc ensemble 7/8
-
- Exo4 Q1 : lb et sb avec $t0
- Exo4 Q2 : cours
- Exo4 Q3 : 6 cycles par itérations
- Ex4 Q4 : déplacer addi $a1, ... après bne
-
- Exo5 Q1 : inst1 0x34850010, inst2 0x0c100005, inst3 0x1500FFFB
- Exo5 Q2 : ?1=1, ?2=0, ?3=comp, ?4=MX
-
- Exo6 Q1 : $t2 = 0x7FFFFFFF $t3 erreur dépassement arithmétique
- Exo6 Q2 : voir cours
-
- Exo7 Q1 : rol (n»31) | (n « 1)
- Exo7 Q1 : ror (n » 1) | (n « 31)
- Exo7 Q2 : rol srl $at,$t1,31 puis sll $t2,$t1,1 puis or $t2,$t2,$at
- Exo7 Q2 : ror sll $at,$t1,31 puis srl $t2,$t1,1 puis or $t2,$t2,$at