Université d'Alger 1

Faculté des Sciences
Département Informatique

Examen L2-ADO

24-Fevrier-2022

durée : 60 minutes

Nom : Prénom : Groupe : Matricule :		/ 20

Exercice 1 : (6 points) (0 point pour chaque réponse fausse ou non justifiée)

Vrai ou Faux : L'outil Assembleur produit un fichier exécutable

1 pt — Faux ! L'outil Assembleur crée des fichiers objets. C'est l'éditeur de lien qui crée des fichiers exécutables.

Vrai ou Faux : L'outil Assembleur traduit du texte écrit dans un langage évolué comme le C/C++ en instructions assembleurs

1 pt — Faux ! C'est le compilateur qui effectue cette tâche.

Convertir en décimal la valeur 0x89, en supposant une représentation non-signée sur un octet.

1 pt — $0x89 = 8 \times 16 + 9 = 137_{10}$

Convertir en décimal la valeur 0x89, en supposant une représentation en complément à deux sur un octet.

1 pt — $0x89 = 1000\ 1001_2$. bit de signe=1 donc conversion comp. à 2 : $0111\ 0111_2 = 7 \times 16 + 7 = 119$. Ainsi, $0x89 = -119_{10}$

Soit la fonction en langage C ci-dessous

```
uint32_t foo() {
 char str[12] = "Hello World";
 uint32_t  val = ((uint32_t*) str)[2];
 return val;
}
```

Si cette fonction est exécutée sur un système little-endian, quelle sera la valeur retournée par la fonction `foo()` (donnez votre réponse en hexadécimal). Notez que le type `uint32_t` fait référence à un entier 32 bits non signé.

2 pts — ((uint32_t*) str) converti le pointeur char* str en un pointeur sur des entiers non signés. ((uint32_t*) str)[2] = le troisième entier est affecté à val. Les bits de notre entier sont les codes ascii des caractères 'rld\0' (72 6C 64 00) lus selon le schéma little-endian. Donc, la valeur retournée est 0x00646C72.

Exercice 2 : (14 points) (0 point pour chaque réponse fausse ou non justifiée)

En plus du stockage de certains registres lors d'appels de fonctions, la pile est utilisée également pour stocker des variables locales. Complétez l'implémentation de la fonction **`verifyPassword`** (en verso). Cette fonction devrait vérifier si le mot de passe saisi est valide ou non.

La fonction **`verifyPassword`** est définie comme suit :

- *Entrées :* Pas d'arguments ; toutefois, la fonction reçoit une chaîne de caractères depuis la console.
- *Sorties :* $v0 renvoie 1 si la chaîne saisie est exactement "motsecret", et 0 dans le cas contraire.

Pour votre implémentation, supposez les étiquettes suivantes déjà définies :

- **Password :** Pointeur vers la chaîne de caractères " motsecret " stockée en mémoire
- **Get20chars :** Une fonction définie comme suit :
  - *Entrées :* $a0 est un pointeur vers un buffer.
  - *Effet :* lit des caractères depuis la console et remplit le buffer pointé par $a0 avec les données saisies. La chaîne est achevée avec le caractère NUL. Vous pouvez supposer dans votre code que la chaîne saisie a une longueur maximale de 19 caractères, sans compter le caractère final NUL.
  - *Sorties :* Pas de valeurs retournées.

```
verifyPassword:
   addi $sp, $sp, -24                    # réserver un espace de 20-octets pour le buffer
   sw _____$ra_,_____ 20($sp)            # sauvegarde de l'adresse de retour dans la pile
   move $a0, $sp_____                    # a0 = adresse du buffer_____________________
   jal Get20chars
   _________la_________ $t0, Password    # t0 = adresse de la chaine "motsecret"_______
   _________move_______ $t1, $sp         # t1 = adresse du buffer (chaine saisie)______
Loop:
   _________lb_________ $t2, 0($t0)      # t2 = *t0 (lire un caractère)________________
   _________lb_________ $t3, 0($t1)      # t3 = *t1 (lire un caractère)________________
   bne $t2, $t3, Fail
   beq $t2, $0, Pass____                 # si t2 == '\0' alors chaine identique________
   addi $t0, $t0, 1_____                 # t0++ (incrémenter le pointeur sur la chaine)__
   addi $t1, $t1, 1_____                 # t1++ (incrémenter le pointeur sur le buffer)__
   j Loop_______________                 # boucler la boucle___________________________
 Pass:
   addi $v0, $0, 1______                 # v0 = 1. Chaines comparées identiques________
   j End________________                 # sauter l'instruction suivante_______________
 Fail:
   move $v0, $0_________                 # v0 = 0. Chaines comparées diffères__________
 End:
   lw   $ra, 20($sp)____                 # restaurer l'adresse de retour depuis la pile__
   addi $sp, $sp, 24____                 # libérer l'espace alloué sur la pile_________
   jr $ra_______________                 # retour à la fonction appelante______________
```

0,5 pt — 1 pt — 0,5 pt — 0,5 pt — 0,5 pt — 1 pt — 0,5 pt — 1 pt — 0,5 pt — 1 pt — 1 pt — 1 pt — 1 pt

Supposons que toute pseudo-instruction éventuellement utilisée dans le code source ci-dessus sera convertie en une seule instruction réelle équivalente. Donnez le code machine associé à l'instruction « `bne $t2, $t3, Fail` ».

1 pt

Type I : op = 0x5, rs = 11 , rt = 10, imm = 6 instructions => 0x156a0006

Donnez le code machine associé à l'instruction « `addi $sp, $sp, -24` ».

1 pt

Type I : op = 0x8, rs = rt = 29, imm = -24 = 0xffe8 => 0x023bdffe8

Si l'adresse de l'instruction « `jal Get20chars` » est `0x00401000`, et si la table des symboles associe l'adresse `0x10400f00` à l'étiquette « `Get20chars` », donnez le code machine associé à cette instruction.

2 pt

les bits 31..28 du $pc (l'adresse de « `jal Get20chars` ») et de l'étiquette sont différents. Donc, l'instruction ne peut pas être représentée par un seul code machine. L'Assembleur convertira cette instruction en :

```
lui  $at, 0x1040       # type I: op=0xf, rs=0, rt=1, imm=0x1040      => 0x3C011040
ori  $at, $at, 0x0f00  # type I: op=0xd, rs=1, rt=1, imm=0x0f00      => 0x34210f00
jalr $at               # type R: op=0x0, rs=1, rt=rd=sh=0, func=0x9  => 0x00200009
```