Examen Final

- Durée 2 heures. Documents non autorisés. Calculatrices, téléphones et ordinateurs portables interdits. Une annexe comportant certains rappels est fournie en fin de sujet.
- Il ne sera répondu à aucune question concernant la compréhension des énoncés, celle-ci fait partie intégrante du DS. *Les brouillons ne seront pas acceptés*.

1.1 Questions (6 pts)

Répondre à chacune des questions suivantes. La clarté des réponses sera prise en compte dans la notation.

a) Expliquez, en une dizaine de lignes maximum, ce qu'est le jeu d'instruction d'un microprocesseur.

b) Expliquez, en une dizaine de lignes maximum, ce qu'est le code binaire d'une instruction. Sur combien de bits est normalement codée une instruction ARM?

c) On considère la séquence d'instructions ARM suivante :

```
MOV   R5, R3, LSL #1
ADD   R5, R5, R3, LSL#3
MOV   R7, R5, LSL#1
ADD   R5, R5, R3, LSL#3
```

En considérant un pipeline à trois étages (*fetch*, *decode*, *execute*), combien de cycles sont nécessaires pour exécuter ces quatre instructions à la suite ? Justifiez votre réponse à l'aide d'un schéma et d'explications, en une dizaine de lignes maximum. (2 pts).

d) On considère la séquence d'instructions ARM suivante :

```
        MOV         R1, #56
        MOV         R2, #49
        BL          max
wait:   B           wait
max:    CMP         R1, R2
        MOVLT       R0, R2
        MOVGE       R0, R1
        MOV         PC, LR
```

En supposant que la première instruction est à l'adresse `0x8000`, quelle est la valeur du registre `LR` après exécution de l'instruction `BL max`? (2 pts).

1.2 Conversion majuscule / minuscule (5 pts)

Soit la chaîne de caractères suivante, supposée être déclarée en mémoire:

```
MSG:  .byte 104, 101, 108, 108, 111
```

a) Quel est le texte exact de la chaîne de caractère `MSG` ?

b) Ecrire un programme assembleur ARM qui remplace cette chaîne de caractères par son équivalent en majuscules. Expliquez votre programme en maximum 10 lignes.

On rappelle qu'en code ASCII, les lettres '`a`' à '`z`' sont représentées par les codes `97` à `122` et les lettres '`A`' à '`Z`' sont représentées par les codes `65` à `90`. La conversion minuscule vers majuscule revient donc à une soustraction de `32`.

1.3 Nombre d'éléments pairs d'un tableau (10 pts)

On souhaite écrire un programme qui compte le nombre d'éléments pairs d'un tableau.
On supposera que les tableaux manipulés sont de type `.word` et qu'ils contiennent dix éléments.

1) Ecrire un sous-programme `PARITE` qui prend en paramètre un nombre dans le registre `R1` et renvoie `0` dans le registre `R0` si le nombre est pair, `1` si le nombre est impair. Expliquez le fonctionnement de votre programme. (4 pts)
2) Ecrire un sous-programme `NB_PAIRS` qui prend en paramètre l'adresse d'un tableau par la pile et renvoie le nombre d'éléments pairs dans le registre `R0`. Ce sous-programme devra utiliser le sous-programme `NB_PAIRS`. (4 pts)
3) Ecrire un programme principal qui permette de tester le sous-programme `NB_PAIRS` sur le tableau suivant: `7, 8, 9, 6, 5, 4, 3, 4, 5, 6`. (2 pts)

ANNEXE

Instructions arithmétiques et logiques (ARM)

Instruction	***Opération***	***Description***
`ADD Rd, Rn, <op2>`	`Rd ← Rn + <op2>`	`"ADD" : addition`
`ADC Rd, Rn, <op2>`	`Rd ← Rn + <op2> + (C)`	`"ADD/Carry": addition+retenue`
`SUB Rd, Rn, <op2>`	`Rd ← Rn - <op2>`	`"SUBtract" : soustraction`
`RSB Rd, Rn, <op2>`	`Rd ← <op2> - Rn`	`"Reverse SuBtract" : soustraction renversée`
`MOV Rd, <op2>`	`Rd ← <op2>`	`«MOVe» : rangement d'une valeur dans un registre`
`MVN Rd, <op2>`	`Rd ← non <op2>`	`«MoVe Not» : négation logique bit à bit`
`AND Rd, Rn, <op2>`	`Rd ← Rn et <op2>`	`«AND» : et logique bit à bit`
`ORR Rd, Rn, <op2>`	`Rd ← Rn ou <op2>`	`«OR Register» : ou logique bit à bit`
`EOR Rd, Rn, <op2>`	`Rd ← Rn oux <op2>`	`«Exclusive OR» : ou exclusif bit à bit`
`BIC Rd, Rn, <op2>`	`Rd ← Rn et non <op2>`	`«BIt Clear» : masquage inverse`

`<op2>` représente le deuxième opérande qui peut être:

- `#literal` : une valeur immédiate
- `Rm` : le nom d'un registre qui contient la valeur de l'opérande.
- `Rm, shift` : un registre associé à une opération de décalage.

Mnémoniques pour décalage (ARM)

- `LSL #n` : Logical Shift Left ($0 \leq n \leq 31$)
- `LSR #n` : Logical Shift Right ($1 \leq n \leq 32$)
- `ASR #n` : Arithmetic Shift Right ($1 \leq n \leq 32$)
- `ROR #n` : Rotate Right ($1 \leq n \leq 31$)
- `RRX`: Rotate Right Extended (ROR étendu de 1 bit avec le bit de retenue C du registre d'état)

Suffixes de condition usuels (et indicateurs d'états associés)

- `EQ` Equal/equals zero (`Z=1`)
- `NE` Not Equal (`Z=0`)
- `GE` Signed greater than or equal (`N=V`)
- `LT` Signed less than (`N≠V`)
- `GT` Signed greater than (`Z=0` et `N=V`)
- `LE` Signed less than or equal (`Z=1` ou `N≠V`)

Instructions d'accès à la pile (en mode Full Descending)

- `STR Rd, [SP, #-4]!` pour empiler un registre
- `STMFD SP!, {reglist}` pour empiler une liste de registres
- `LDR Rd, [SP], #4` pour dépiler un registre
- `LDMFD SP!, {reglist}` pour dépiler une liste de registres