

GIF-1001 Ordinateurs: Structure et Applications
Hiver 2018
Examen mi-session
27 février 2018
Durée : 170 minutes
Professeur : Jean-François Lalonde

Cet examen comporte 7 questions sur 18 pages (incluant celle-ci), comptabilisées sur un total de 100 points. L'examen compte pour 40% de la note totale pour la session.

- Vous avez droit à une feuille aide-mémoire 8.5×11 recto-verso, écrite à la main, ainsi qu'une calculatrice acceptée;
- Assurez-vous que l'étiquette sur le cahier bleu corresponde bien à vous;
- Assurez-vous d'avoir toutes les pages;
- **Certaines questions apparaissent au verso** : regardez des deux côtés!
- Écrivez vos réponses dans le cahier bleu qui vous a été remis;
- SVP sortez vos cartes étudiantes et placez-la visiblement sur votre table de travail;
- L'examen contient quatre (4) annexes :
 - l'annexe A contient un rappel sur les unités en binaire et les logarithmes;
 - l'annexe B contient une liste d'instructions ARM ainsi que des codes de conditions;
 - l'annexe C contient la table ASCII;
 - l'annexe D contient le jeu d'instructions du simulateur du TP1.

La table ci-dessous indique la distribution des points pour chaque question.

Question:	1	2	3	4	5	6	7	Total
Points:	15	20	10	10	15	20	10	100

Bonne chance!

1. (15 points) Répondez aux questions suivantes sur la représentation des données dans un ordinateur.

(a) (1 point) Combien de bits sont nécessaires pour stocker le nombre entier de millilitres dans un litre ?

Solution: Il y a 1000 millilitres dans 1 litre, donc il faut 10 bits (1024 valeurs possibles) pour les représenter.

(b) (4 points) Calculez le résultat de $10 + 7$ en complément-2 :

i. Sur 5 bits. Écrivez votre réponse en binaire et en décimal. Indiquez s'il y a débordement.

Solution: $0b01010 + 0b00111 = 0b10001$ (-15). Il y a débordement.
(0.5 pour réponse binaire, 0.5 pour réponse décimal, 1 pour débordement)

ii. Sur 6 bits. Écrivez votre réponse en binaire et en décimal. Indiquez s'il y a débordement.

Solution: $0b001010 + 0b000111 = 0b010001$ (17). Il n'y a pas de débordement.
(0.5 pour réponse binaire, 0.5 pour réponse décimal, 1 pour débordement)

(c) (1 point) Comment fait-on pour détecter un débordement lors d'une addition en complément-2 ?

Solution: Quand les deux opérands ont le même bit de signe et que le signe change en sortie.

(d) (2 points) Une retenue est générée lorsque le résultat d'une addition est valide, mais qu'il nécessite un bit supplémentaire.

i. Donnez un exemple de retenue générée lors d'une addition sur 5 bits en arithmétique *non-signée*.

Solution: Par exemple : $20+20 = 0b10100 + 0b10100 = 0b(1)01000$ (40).

ii. Donnez un exemple de retenue générée lors d'une addition sur 5 bits en arithmétique *signée*.

Solution: Par exemple : $-2+2 = 0b11110 + 0b00010 = 0b(1)00000$ (0).

(e) (2 points) Comment le microprocesseur fait-il pour savoir que $0x4F5341$ signifie «OSA» en ASCII plutôt que 5198657 en décimal ?

Solution: Il ne peut pas le savoir !

(f) (4 points) La norme IEEE754 encode des nombres rationnels sur 32 bits de la façon suivante :

$$(\text{signe})1, \text{ mantisse} \times 2^{(\text{exposant}-127)} .$$



FIGURE 1 – Convention IEEE-754 sur 32 bits.

et les bits sont stockés selon la figure 1 :

- i. Quelle est la représentation de -13.5 en IEEE754 sur 32 bits ? Écrivez votre résultat en hexadécimal.

Solution: 0xC1580000

- ii. Quelle est la représentation décimale de 0x40F80000, encodé en IEEE754 sur 32 bits ?

Solution: 7.75

- (g) (1 point) Est-ce que la multiplication des deux nombres 0xC34D2000 et 0xD36EC000 représentés en IEEE754 donnera un résultat plus grand ou plus petit que 0 ?

Solution: Plus grand que 0, car ces deux nombres sont négatifs (bit de signe à 1 dans les deux cas).

2. (20 points) Répondez aux questions suivantes portant sur le microprocesseur du simulateur du travail pratique 1. Dans ce système, toutes les instructions du microprocesseur sont encodées sur 16 bits et se décomposent comme suit :

- Bits 15 à 12 : Opcode de l'instruction
- Bits 11 à 8 : Registre utilisé comme premier paramètre.
- Bits 7 à 0 : Registre ou constante utilisés comme deuxième paramètre

Comme à l'habitude, le bit 0 est le moins significatif, et 15 le plus significatif. Le nombre identifiant le registre PC est 0xF (15), et le jeu d'instruction est décrit en annexe D.

- (a) (1 point) En arithmétique non-signée, quelle est la plus grande constante pouvant être utilisée dans ce jeu d'instructions ? Écrivez votre réponse en décimal et expliquez pourquoi.

Solution: 8 bits sont réservés à la constante, donc la plus grande constante est 0xFF, soit 255.
(0.5 point pour 255 (ou 0xFF), 0.5 point pour le 8 bits)

- (b) (2 points) Est-ce que le microprocesseur peut distinguer les instructions `MOV R0, #0x02` et `MOV R0, R2` étant donné que, dans les deux cas, les paramètres sont encodés de la même façon ? Si oui, comment fait-il ? Sinon, pourquoi ?

Solution: Oui, c'est grâce à l'opcode qui indique quelle «version» de l'instruction MOV est utilisée.
(1 point pour oui, 1 point pour l'explication)

- (c) (5 points) Chacune des descriptions suivantes peut être exécutée en *une seule instruction*, indiquez laquelle. *Important* : pour chaque instruction, écrivez votre résultat en texte (ex :

MOV R0, #0x0) et en binaire (ex : 0x4000).

- i. Place la valeur 13 dans R2.

Solution: MOV R2, #0xD 0x420D
(0.5 point pour le texte, 0.5 point pour le binaire).

- ii. Lit la valeur en mémoire à l'adresse indiquée par R2 et stocke le résultat dans R1 ;

Solution: LDR R1, [R2] 0x8102
(0.5 point pour le texte, 0.5 point pour le binaire).

- iii. Si la valeur du registre R3 est égale à 0, place 0x4 dans PC ;

Solution: JZE R3, #0x4 0xF304
(0.5 point pour le texte, 0.5 point pour le binaire).

- iv. Effectue l'addition du contenu de R2 et de R3, et place le résultat dans R2.

Solution: ADD R2, R3 0x1203
(0.5 point pour le texte, 0.5 point pour le binaire).

- v. Effectue un branchement à l'adresse 0x7.

Solution: MOV PC, #0x7 0x4F07
(0.5 point pour le texte, 0.5 point pour le binaire).

- (d) (5 points) Traduisez le programme suivant en binaire, et écrivez votre réponse en hexadécimal. Les numéros de ligne sont indiqués à gauche.

```

1 LDR R3, [R1]
2 SUB R1, R0
3 JZE R0, #0x1
4 STR R2, [R3]
5 MOV PC, #0x9A

```

Solution:

0x8301
0x2100
0xF001
0x9203
0x4F9A

(1 point par instruction.)

(Suite au verso de la page)

- (e) Soit le programme suivant. Pour chaque ligne, on indique l'adresse (qui commence à 0x0), suivie de l'instruction en format binaire. Les numéros de ligne sont indiqués à gauche.

1	0x0	0x4040
2	0x1	0x8000
3	0x2	0x4141
4	0x3	0x8101
5	0x4	0x4200
6	0x5	0xF109
7	0x6	0x1200
8	0x7	0x6101
9	0x8	0x4F05
10	0x9	0x4142
11	0xA	0x9201

- i. (3 points) Écrivez le programme assembleur correspondant au code binaire ci-haut.

Solution:

```
MOV R0, #0x40
LDR R0, [R0]
MOV R1, #0x41
LDR R1, [R1]
MOV R2, #0x00
JZE R1, #0x09
ADD R2, R0
SUB R1, #0x01
MOV PC, #0x05
MOV R1, #0x42
STR R2, [R1]
```

(-0.5 point par erreur, minimum de 0)

- ii. (4 points) Décrivez, *en une seule phrase*, ce que ce programme fait. Indiquez clairement les adresses employées pour les données en entrée et en sortie. *Indice* : pour déterminer ce que fait ce programme, placez de faibles valeurs fictives (e.g. entre 1 et 5) aux adresses mémoire 0x40 et 0x41, exécutez ce programme pas à pas, et observez l'évolution du contenu des registres au fil du temps. *Important* : vous devez décrire le comportement global du programme ; toute réponse décrivant les instructions une par une recevra la note de 0.

Solution: Il calcule le produit des valeurs stockées en mémoire aux adresses 0x40 et 0x41 (inclusivement), et stocke le résultat à l'adresse 0x42.

(2 points pour le produit, 1 point pour les adresses en entrée, 1 point pour l'adresse en sortie)

3. (10 points) Un système de type «memory-mapped I/O» possède les caractéristiques suivantes :
- un bus d'adresse de 24 bits, avec les 3 bits les plus significatifs (MSB) utilisés pour le décodeur d'adresse ;
 - un bus de données de 24 bits ;
 - deux mémoires RAM où chaque octet possède une adresse différente ;
 - trois autres périphériques sont branchés sur les bus ;
 - il stocke les données en mémoire avec la convention «little endian» ;
 - si on nomme les bits les plus significatifs (MSB) du bus d'adresse b_{23} , b_{22} et b_{21} , le décodeur sélectionne les périphériques de la façon suivante :

b_{23}	b_{22}	b_{21}	Périphérique activé
0	0	0	RAM 1
0	0	1	RAM 2
0	1	0	Périphérique 1
0	1	1	Périphérique 2
1	X	X	Périphérique 3

Ici, «X» indique soit 0 ou 1, sans importance.

- (a) (2 points) Quelle est la taille maximale de la mémoire RAM 1 ? Écrivez votre réponse en mega-octets (Mo), et écrivez votre démarche.

Solution: $24 - 3 = 21$ bits sont utilisés pour générer les adresses, et chaque octet possède une adresse différente. On a donc 2^{21} octets, soit $2^1 = 2\text{Mo}$.
(1 point pour le bon nombre de bits, et 1 point pour la bonne réponse)

- (b) On emploie une instruction pour stocker la valeur 0xABCDEF à l'adresse 0x203000.
- i. (1 point) À quel périphérique cette adresse correspond-elle ?

Solution: À la RAM 2.

- ii. (2 points) Indiquez les adresses du périphérique correspondant qui sont modifiées par cette instruction, ainsi que la valeur placée à chacune de ces adresses.

Solution: $0x3000 = 0xEF$, $0x3001 = 0xCD$, $0x3002 = 0xAB$
(bonnes adresses = 1 point, bonnes valeurs = 1 point)
(Si les adresses sont spécifiées ainsi : $0x203000 = 0xEF$, $0x203001 = 0xCD$, $0x203002 = 0xAB$, 1 point seulement)

- (c) (3 points) La carte de la mémoire («memory map») d'un système indique les premières et dernières adresses du microprocesseur correspondant à chaque périphérique branché sur les bus. Quelle est la carte de la mémoire de ce système ?

Solution:	$0x000000$ à $0x1FFFFFF$	RAM 1	(0.5 point par périphérique, 1 point total)
	$0x200000$ à $0x3FFFFFF$	RAM 2	
	$0x400000$ à $0x5FFFFFF$	Périphérique 1	
	$0x600000$ à $0x7FFFFFF$	Périphérique 2	
	$0x800000$ à $0xFFFFFFF$	Périphérique 3	

point pour périphérique 3. Si l'octet le plus significatif est ok, 2 points au total)

- (d) (2 points) Du point de vue du microprocesseur, quelle est la différence entre l'écriture d'une donnée en RAM 2 par rapport à l'envoi d'une donnée au périphérique 2 ?

Solution: Il n'y a que l'adresse qui change.

4. (10 points) Pour vous pratiquer, vous écrivez le code ARM suivant :

```

1 SECTION INTVEC
2
3 B main
4
5 SECTION CODE
6
7 mavar ASSIGN32 0x01ABCDEF
8
9 main
10 LDR R0, mavar
11 LDR R0, =mavar
12 MOV R3, #1
13
14 SECTION DATA

```

Vous démarrez ensuite le simulateur, et vous observez que le contenu de la mémoire est :

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0x00000000	1F	00	00	EA	--	--	--	--	--	--	--	--	--	--	--	--
...																
0x00000080	EF	CD	AB	01	0C	00	1F	E5	00	00	9F	E5	01	30	A0	E3
0x00000090	80	00	00	00	--	--	--	--	--	--	--	--	--	--	--	--

Répondez aux questions suivantes concernant la relation entre votre code et le contenu en mémoire.

- (a) (2 points) De l'adresse 0x80 à 0x83 inclusivement, pourquoi la variable `mavar` apparaît-elle dans l'ordre inverse (0xEFCDAB01) plutôt que dans l'ordre que vous aviez spécifié (0x01ABCDEF) ?

Solution: Parce que nous sommes en «little endian», ce qui stocke l'octet le moins significatif dans l'adresse mémoire la plus basse.

- (b) (1 point) Quelle est la valeur binaire de l'instruction `LDR R0, mavar` sur 32 bits ? Écrivez votre réponse en hexadécimal.

Solution: 0xE51F000C

- (c) (1 point) Quelle est la valeur binaire de l'instruction `MOV R3, #1` sur 32 bits ? Écrivez votre réponse en hexadécimal.

Solution: 0xE3A03001

- (d) (1 point) Sachant que les bits b_{24} à b_{21} d'une instruction ARM correspond à son «opcode», quel est l'opcode de l'instruction MOV? Ici, b_0 est le bit le moins significatif et b_{31} le plus significatif.

Solution: 0b1101 ou 0xD

- (e) (1 point) Sachant que le bit b_{20} d'une instruction MOV doit être mis à 1 si l'on désire mettre à jour les drapeaux du CPSR, quel est le code binaire de l'instruction MOVS R3, #1? Ici, b_0 est le bit le moins significatif et b_{31} le plus significatif. Écrivez votre réponse en hexadécimal.

Solution: 0xE3B03001

- (f) (2 points) À l'adresse 0x90, on remarque la présence de la valeur 0x80 (sur 32 bits). Pourtant, vous n'aviez pas placé cette valeur explicitement dans votre code. À quoi correspond cette valeur? Pourquoi a-t-elle été placée à cet endroit par l'assembleur?

Solution: C'est l'adresse de la variable `mavar`. Elle a été placée à cet endroit afin que l'instruction `LDR R0, =mavar` puisse y accéder via une instruction relative à PC.
(1 point pour l'adresse, 1 point pour le remplacement par instruction relative à PC)

- (g) (1 point) Lorsque vous exécutez le code et que vous vous arrêtez à l'instruction `LDR R0, mavar`, vous remarquez que l'instruction courante correspond à `LDR R0, [R15, #-0xC]`. Pourquoi l'instruction a-t-elle été remplacée?

Solution: Parce que le microprocesseur ne «comprend» pas les variables, il ne comprend que du binaire. Donc, l'assembleur remplace ce genre d'instructions par des instructions relatives à PC (R15).

- (h) (1 point) Dans la question précédente, pourquoi l'assembleur a-t-il choisi #-0xC dans `LDR R0, [R15, #-0xC]`?

Solution: Le déplacement #-0xC représente la différence entre l'adresse de PC (0x8C) et l'adresse de la variable (0x80).

5. (15 points) Répondez aux questions portant sur le code assembleur ARM suivant (les numéros de ligne sont indiqués à gauche) :

```

1 SECTION INTVEC
2
3 B main
4
5 SECTION CODE
6
7 main
8
```

```
9 LDR SP, =maPile
10 ADD SP, SP, #4
11
12 LDR R0, =source
13 LDR R1, =destination
14 PUSH {R1}
15 POP {R5}
16 ADD R1, R1, #12
17
18 BL fonctionMystere
19 B main
20
21 fonctionMystere
22
23 debut
24 CMP R0, R5
25 BEQ fin
26
27 LDR R2, [R0], #4
28 STR R2, [R1], #-4
29 B debut
30
31 fin
32 BX LR
33
34 SECTION DATA
35
36 source      ASSIGN32 0x1, 0x2, 0x3, 0x4
37 destination ALLOC32 4
38 maPile      ALLOC32 1
```

- (a) (1 point) Pourquoi doit-on rajouter 4 à SP à la ligne 10 ?

Solution: Car le pointeur de pile (SP) débute à la fin de la pile. Il est décrémenté par un PUSH, et incrémenté par un POP.

- (b) (1 point) Quelle instruction pourrait-on utiliser pour remplacer le PUSH et le POP aux lignes 14 et 15 ?

Solution: MOV R5, R1

- (c) (2 points) Sachant que la section DATA débute à l'adresse 0x1000, quelle est la valeur de R1 après l'exécution de l'instruction ADD R1, R1, #12 à la ligne 16 ?

Solution: 0x101C

- (d) (2 points) Que fait l'instruction STR R2, [R1], #-4 à la ligne 28 ?

Solution: Tout d'abord, elle copie le contenu de R2 en mémoire à l'adresse contenue dans R1. Ensuite, elle décrémente R1 de 4.
(0.5 point pour destination, 0.5 point pour adresse, 1 point pour décrément)

- (e) (1 point) Comment l'instruction `BEQ fin` (ligne 25) fait-elle pour savoir si la condition `EQ` est satisfaite? Quelle autre instruction affecte cette condition?

Solution: Elle regarde le statut des drapeaux du microprocesseur. Dans le cas d'un `EQ`, il faut que le drapeau «Z» soit égal à 1. L'instruction affectant cette condition est le `CMP R0, R5`.

(0.5 point pour les drapeaux, 0.5 point pour avoir mentionné `CMP`.)

(Suite au verso de la page)

- (f) (2 points) La fonction `fonctionMystere` modifie le registre `R2`. Quelles sont les deux instructions qui, rajoutées à cette fonction, feraient en sorte que `fonctionMystere` ne modifierait aucun registre autre que `R0` et `R1`? Indiquez clairement l'endroit où chacune de ces deux instructions devraient être rajoutées.

Solution: `PUSH R2` devrait être rajoutée après `fonctionMystere`, et `POP R2` juste avant le `BX LR`. Il est aussi possible de le faire avant et après les `LDR/STR`, mais cette méthode est moins efficace.

(1 point pour le `PUSH`, 1 point pour le `POP`)

- (g) Comme vous pouvez le constater, la fonction `fonctionMystere` parcourt les éléments du tableau `source`.

- i. (1 point) Comment la fonction fait-elle pour savoir quand arrêter de boucler?

Solution: Lorsque l'adresse dans le tableau `source` correspond à la première adresse du tableau `destination`.

- ii. (3 points) Quels registres sont utilisés pour passer des arguments à la fonction? À quoi ces arguments correspondent-ils?

Solution: `R0` contient l'adresse du premier élément du tableau `source`, `R1` contient l'adresse du tableau `destination` et `R5` contient l'adresse après le dernier élément de `source`

(0.5 point pour chaque registre, 0.5 point pour chaque bonne explication)

- iii. (2 points) Décrivez *en une seule phrase* ce que fait `fonctionMystere`. *Important* : vous devez décrire le comportement global du programme; toute réponse décrivant les instructions une par une recevra la note de 0.

Solution: La fonction copie le contenu du tableau `source` dans le tableau `destination`, dans l'ordre inverse.

(1 point pour la copie, 1 point pour l'ordre inverse)

6. (20 points) Répondez aux questions suivantes, portant sur l'assembleur ARM.

- (a) (1 point) Expliquez la différence entre «big endian» et «little endian».

Solution: En little endian, on stocke la valeur de l'octet le moins significatif dans l'adresse la plus basse. En big endian, on le stocke dans l'adresse la plus élevée.

- (b) (2 points) Pourquoi doit-on choisir entre «big» et «little endian» en ARM, mais pas dans l'ordinateur du TP1 ?

Solution: En ARM, chaque octet possède une adresse mais les mots sont de 4 octets. Il faut donc choisir dans quel ordre stocker les octets. Dans le cas du TP1, chaque mot (de 2 octets) possède une seule adresse.
(1 point pour l'ARM, 1 point pour le TP1)

- (c) (2 points) Pourquoi doit-on incrémenter PC de 4 à chaque instruction ?

Solution: Car chaque instruction fait 4 octets et que chaque octet possède une adresse.
(1 point pour taille de l'instruction, 1 point pour chaque octet...)

- (d) (4 points) Écrivez un court programme en assembleur ARM qui effectue les trois étapes suivantes :

1. Multiplie la valeur contenue dans R2 par 2, *sans utiliser l'instruction MUL*, et stocke le résultat dans R2 ;
2. Si le résultat de la multiplication est plus grand que 4, brancher à l'adresse 0x80 ;
3. Si le résultat de la multiplication est plus petit ou égal à 4, brancher à l'adresse 0x100.

Solution: Par exemple :

```
LSL R2, R2, #1
CMP R2, #4
MOVGT PC, #0x80
MOVLE PC, #0x100
```

(1 point par instruction valide)

- (e) (6 points) Écrivez un court programme en assembleur ARM qui calcule la somme s du carré des éléments d'un vecteur v de longueur N :

$$s = \sum_{i=1}^N v_i^2,$$

où le vecteur v contient des éléments de 32 bits en mémoire commençant à l'adresse 0x1000, et N est dans R0. La somme s doit être stockée dans R3. Vous disposez d'une fonction `puissanceDeux` qui prend un nombre en entrée dans R1 et retourne le carré de ce nombre dans R1 également. *Vous n'avez pas à implémenter la fonction `puissanceDeux`, assumez qu'elle existe et que vous pouvez l'utiliser.*

En résumé, implémentez le pseudo-code suivant :

Solution: Par exemple :

```
(1 point pour la préparation)
MOV R3, #0
MOV R2, #0x1000
```

```

R3 ← 0 ;
R2 ← 0x1000 ;
while R0 > 0 do
  R1 ← Memoire[R2] ;
  Appel de la fonction puissanceDeux
  R3 ← R3 + R1 ;
  R2 ← R2 + 4 ;
  R0 ← R0 - 1 ;
end

```

```

boucle
(1 point pour condition de sortie de la boucle)
CMP R0, #0
BEQ fin

(1 point pour lecture en mémoire)
LDR R1, [R2], #4
(1 point pour appel de fonction)
BL puissanceDeux

(1 point pour incrémentation des registres R2, R3, R0)
ADD R3, R3, R1
SUB R0, R0, #1

(1 point pour boucle principale)
B boucle
fin

```

(Suite au verso de la page)

- (f) (5 points) Considérez le code suivant. Pour chaque ligne, on indique l'adresse (qui commence à 0x80), suivie de l'instruction en format binaire. Les numéros de ligne sont indiqués à gauche.

```

1 0x80 MOV R1, PC
2 0x84 MOV R0, #1
3 0x88 PUSH {R1}
4 0x8C SUB R5, PC, #8
5 0x90 MOV R0, #3
6 boucle
7 0x94 SUBS R0, R0, #1
8 0x98 ADDGT R5, R5, #12
9 0x9C BLT boucle
10 0xA0 BX R5
11 0xA4 MOV R0, #1
12 0xA8 POP {PC}

```

Indiquez l'ordre des 20 premières instructions exécutées par le microprocesseur en utilisant

leur *numéro de ligne* correspondant. Vous pouvez assumer qu'une pile a préalablement été préparée. Vous devez indiquer la ligne d'une instruction conditionnelle (par exemple, MOVEQ) même si sa condition (par exemple, EQ) n'est pas satisfaite.

Solution: 1-2-3-4-5-7-8-9-10-8-9-10-11-12-3-4-5-7-8-9... (recommence en boucle infinie)

(2 points pour les 9 premiers, 2 points pour les 5 suivants, 1 point pour la boucle finale, -1 si la ligne 6 est identifiée)

7. (10 points) Répondez aux questions suivantes par une réponse courte.

(a) (1 point) Nommez une différence entre le RISC et le CISC.

Solution: Plusieurs options possibles : instructions de taille fixe vs variables, instructions en CISC plus complexes, programmes en CISC de taille plus compacte, RISC utilise généralement plus de registres, RISC nécessite une architecture matérielle plus simple, donc généralement moins énergivore, etc.

(b) (1 point) Pourquoi est-il avantageux d'utiliser un pipeline à trois étages (comme en ARM) ?

Solution: Car les trois opérations (lecture, décodage, exécution) peuvent être effectuées simultanément plutôt que séquentiellement, cela est donc plus rapide !

(c) (1 point) À quoi sert un décodeur d'adresses ?

Solution: À déterminer quelle composante sera activée, en fonction d'un (ou plusieurs) bits du bus d'adresses.

(d) (1 point) Vrai ou faux ? Lors de l'exécution d'une instruction STR, le bus de contrôle est placé en lecture.

Solution: Faux

(e) (1 point) Quelles sont les deux étapes effectuées par un microprocesseur ARM lorsqu'il exécute l'instruction BL *etiquette* ?

Solution: Il place PC-4 dans LR, puis effectue un branchement à l'adresse indiquée par *etiquette*.
(0.5 point par élément)

(f) (1 point) Vrai ou faux ? Les trois principaux bus sont les bus d'adresse, de données et d'instructions.

Solution: Faux, il s'agit du bus de contrôle plutôt que d'instructions.

(g) (1 point) Quelles sont les trois étapes principales du cycle d'instructions ?

Solution: Lecture—décodage—exécution.

- (h) (1 point) Expliquez la différence entre le complément-2 (signée) et la représentation binaire non-signée.

Solution: Le bit le plus significatif vaut 2^{N-1} en notation non-signée, et -2^{N-1} en notation signée.

- (i) (1 point) Vrai ou faux? Dans l'architecture von Neumann, les données et les instructions sont dans des mémoires différentes.

Solution: Faux.

- (j) (1 point) Combien de bits peut-on représenter avec trois caractères en hexadécimal?

Solution: 12 bits

A Annexe : Unités et logarithmes

A.1 Unités

Petit rappel sur les unités :

$$\begin{aligned} 1\text{Ko} &= 2^{10}\text{o} &= & 1\,024 \text{ octets} \\ 1\text{Mo} &= 2^{20}\text{o} = 1\,024\text{Ko} &= & 1\,048\,576 \text{ octets} \\ 1\text{Go} &= 2^{30}\text{o} = 1\,024\text{Mo} &= & 1\,073\,741\,824 \text{ octets} \end{aligned}$$

A.2 Logarithme en base 2

Il est facile de calculer des logarithmes en base 2 à partir de logarithmes dans une autre base N (ex : 10). Pour ce faire, appliquez l'équation suivante :

$$\log_2 x = \frac{\log_N x}{\log_N 2}.$$

B Annexe : Instructions ARM et codes de conditions

Instruction	Description
ADD Rd, Rs, Op1	$Rd \leftarrow Rs + Op1$
AND Rd, Rs, Op1	$Rd \leftarrow Rs \text{ AND } Op1$
ASR Rd, Rs, #cte	$Rd \leftarrow Rs / 2^{cte}$
B etiquette	$PC \leftarrow \text{adresse}(\text{etiquette})$
BL etiquette	$LR \leftarrow PC - 4, PC \leftarrow \text{adresse}(\text{etiquette})$
BX Rs	$PC \leftarrow Rs$
CMP Rs, Op1	Change les drapeaux comme $Rs - Op1$
LDR Rd, etiquette	$Rd \leftarrow \text{valeur}(\text{etiquette})$
LDR Rd, =etiquette	$Rd \leftarrow \text{adresse}(\text{etiquette})$
LDR Rd, [Rb, Op1]	$Rd \leftarrow \text{Mem}[Rb + Op1]$
LDR Rd, [Rb], Op1	$Rd \leftarrow \text{Mem}[Rb], Rb \leftarrow Rb + Op1$
LDR Rd, [Rb, Op1]!	$Rb \leftarrow Rb + Op1, Rd \leftarrow \text{Mem}[Rb]$
LSL Rd, Rb, #cte	$Rd \leftarrow Rb \times 2^{cte}$
MUL Rd, Rn, Rs	$Rd \leftarrow Rn \times Rs$
MVN Rd, Op1	$Rd \leftarrow !Op1$ (inverse les bits)
POP {Liste Reg}	Charge les registres en ordre croissant à partir de la pile, $SP \leftarrow SP - 4 \times (\text{nombre de registres})$
PUSH {Liste Reg}	$SP \leftarrow SP + 4 \times (\text{nombre de registres})$, Met la liste de registres sur la pile dans l'ordre décroissant
STR Rs, etiquette	$\text{valeur}(\text{etiquette}) \leftarrow Rs$
STR Rs, [Rb, Op1]	$\text{Mem}[Rb + Op1] \leftarrow Rs$
STR Rs, [Rb], Op1	$\text{Mem}[Rb] \leftarrow Rs, Rb \leftarrow Rb + Op1$
STR Rs, [Rb, Op1]!	$Rb \leftarrow Rb + Op1, \text{Mem}[Rb] \leftarrow Rs$
SUB Rd, Rs, Op1	$Rd \leftarrow Rs - Op1$

TABLE 1 – Instructions ARM. Op1 dénote une opérande de type 1, soit une constante, un registre ou un registre décalé.

Code	Condition	Code	Condition
CS	Retenue (carry)	CC	Pas de retenue
EQ	Égalité	NE	Inégalité
VS	Débordement	VC	Pas de débordement
GT	Plus grand	LT	Plus petit
GE	Plus grand ou égal	LE	Plus petit ou égal
PL	Positif	MI	Négatif

TABLE 2 – Codes de condition.

C Annexe : Table ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Char	Dec	Hx	Oct	Char
0	0	000	NUL	43	2B	053	+	86	56	126	V
1	1	001	SOH	44	2C	054	,	87	57	127	W
2	2	002	STX	45	2D	055	-	88	58	130	X
3	3	003	ETX	46	2E	056	.	89	59	131	Y
4	4	004	EOT	47	2F	057	/	90	5A	132	Z
5	5	005	ENQ	48	30	060	0	91	5B	133	[
6	6	006	ACK	49	31	061	1	92	5C	134	\
7	7	007	BEL	50	32	062	2	93	5D	135]
8	8	010	BS	51	33	063	3	94	5E	136	^
9	9	011	TAB	52	34	064	4	95	5F	137	_
10	A	012	LF	53	35	065	5	96	60	140	'
11	B	013	VT	54	36	066	6	97	61	141	a
12	C	014	FF	55	37	067	7	98	62	142	b
13	D	015	CR	56	38	070	8	99	63	143	c
14	E	016	SO	57	39	071	9	100	64	144	d
15	F	017	SI	58	3A	072	:	101	65	145	e
16	10	020	DLE	59	3B	073	;	102	66	146	f
17	11	021	DC1	60	3C	074	<	103	67	147	g
18	12	022	DC2	61	3D	075	=	104	68	150	h
19	13	023	DC3	62	3E	076	>	105	69	151	i
20	14	024	DC4	63	3F	077	?	106	6A	152	j
21	15	025	NAK	64	40	100	@	107	6B	153	k
22	16	026	SYN	65	41	101	A	108	6C	154	l
23	17	027	ETB	66	42	102	B	109	6D	155	m
24	18	030	CAN	67	43	103	C	110	6E	156	n
25	19	031	EM	68	44	104	D	111	6F	157	o
26	1A	032	SUB	69	45	105	E	112	70	160	p
27	1B	033	ESC	70	46	106	F	113	71	161	q
28	1C	034	FS	71	47	107	G	114	72	162	r
29	1D	035	GS	72	48	110	H	115	73	163	s
30	1E	036	RS	73	49	111	I	116	74	164	t
31	1F	037	US	74	4A	112	J	117	75	165	u
32	20	040	Space	75	4B	113	K	118	76	166	v
33	21	041	!	76	4C	114	L	119	77	167	w
34	22	042	"	77	4D	115	M	120	78	170	x
35	23	043	#	78	4E	116	N	121	79	171	y
36	24	044	\$	79	4F	117	O	122	7A	172	z
37	25	045	%	80	50	120	P	123	7B	173	{
38	26	046	&	81	51	121	Q	124	7C	174	
39	27	047	'	82	52	122	R	125	7D	175	}
40	28	050	(83	53	123	S	126	7E	176	~
41	29	051)	84	54	124	T	127	7F	177	DEL
42	2A	052	*	85	55	125	U				

D Annexe : Jeu d'instructions du microprocesseur du TP1

Mnémonique	Opcode	Description
MOV Rd, Rs	0000	$Rd \leftarrow Rs$
MOV Rd, Const	0100	$Rd \leftarrow \text{Const}$
ADD Rd, Rs	0001	$Rd \leftarrow Rd + Rs$
ADD Rd, Const	0101	$Rd \leftarrow Rd + \text{Const}$
SUB Rd, Rs	0010	$Rd \leftarrow Rd - Rs$
SUB Rd, Const	0110	$Rd \leftarrow Rd - \text{Const}$
LDR Rd, [Rs]	1000	$Rd \leftarrow \text{Mem}[Rs]$
STR Rd, [Rs]	1001	$\text{Mem}[Rs] \leftarrow Rd$
JZE Rc, Const	1111	si $Rc = 0$, $PC \leftarrow \text{Const}$
JZE Rc, Rs	1011	si $Rc = 0$, $PC \leftarrow Rs$

TABLE 3 – Jeu d'instructions du microprocesseur du TP1