GIF-1001 Ordinateurs: Structure et Applications
Hiver 2015
Examen mi-session
24 février 2015
Durée: 110 minutes

Cet examen comporte 10 questions sur 13 pages (incluant celle-ci), comptabilisées sur un total de 100 points. L'examen compte pour 40% de la note totale pour la session. Assurez-vous d'avoir toutes les pages. Les règles suivantes s'appliquent:

- Vous avez droit à une feuille aide-mémoire 8.5×11 recto-verso, écrite à la main, ainsi qu'une calculatrice acceptée.
- Écrivez vos réponses dans le cahier bleu qui vous a été remis;
- L'annexe A contient une liste d'instructions ARM et de codes de conditions qui pourraient vous être utiles.

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

Question:	1	2	3	4	5	6	7	8	9	10	Total
Points:	15	10	10	10	5	10	10	10	10	10	100

La question 9 contient aussi 5 points bonus.

Bonne chance!

1. Registres spéciaux en assembleur ARM: `SP`, `PC` et `RL`.
   (a) (5 points) Qu'est-ce qu'indique le pointeur de pile (`SP`)? Quelles sont les deux instructions qui l'utilisent? Comment sa valeur est-elle modifiée lorsqu'on empile une donnée sur la pile?

   Solution: Le pointeur de pile pointe à l'élément sur le dessus de la pile. Les instructions `PUSH` et `POP` l'utilisent. Sa valeur est décrémentée de 4 octets lorsqu'on empile une donnée (`SP` = `SP` - 4).

   (b) (5 points) Quelle est la taille du registre `PC` en ARM? Que contient-il? De combien doit-il être incrémenté à chaque instruction? Peut-on y accéder avec une instruction `MOV`?

   Solution: La taille de `PC` est 32 bits (comme tous les registres en ARM!). Il contient l'adresse de la prochaine instruction à exécuter + 8. Il doit être incrémenté de 4 à chaque instruction. On peut tout à fait y accéder avec une instruction `MOV`.

   (c) (5 points) À quoi sert le registre de liens (`RL`)? Donnez un exemple de son utilisation.

   Solution: Le registre de liens sert à sauvegarder l'adresse de retour après l'exécution d'une fonction. Exemple:

```
BL maFonction ; LR = adresse de la prochaine instruction (PC-4 en ARM)
              ; PC = adresse de maFonction
...

maFonction
...
BX LR ; PC = LR
```

2. Répondez aux questions suivantes sur l'ALU.
   (a) (2 points) Qu'est-ce qu'un ALU, et où est-il situé?

   Solution: L'ALU est un circuit qui effectue des calculs arithmétiques et logiques. Il est situé à l'intérieur du microprocesseur.

   (b) (3 points) Décrivez trois drapeaux de l'ALU.

   Solution:

   - `N`: indique si le résultat est négatif;
   - `C`: indique si l'opération a généré une retenue (carry);
   - `V`: indique si l'opération a généré un débordement (overflow);
   - `Z`: indique si le résultat est égal à 0.

   (c) (5 points) Expliquez, à l'aide d'un exemple en assembleur ARM, à quoi servent les drapeaux de l'ALU dans un microprocesseur.

Solution: Il y a plusieurs possibilités ici, mais nous voulons voir l'emploi d'une instruction qui modifie les drapeaux de l'ALU (e.g. `SUBS`, `CMP`, etc.), et une autre instruction conditionnelle. Par exemple:

```
CMP R1, R2    ; comparons R1 et R2
BNE pasEgal   ; branche à une adresse si R1 n'est pas égal à R2.
```

3. Le schéma de la figure 1 représente l'architecture interne d'un microprocesseur simple, comme nous l'avons vu dans le cours. Utilisez-le pour répondre aux questions suivantes.

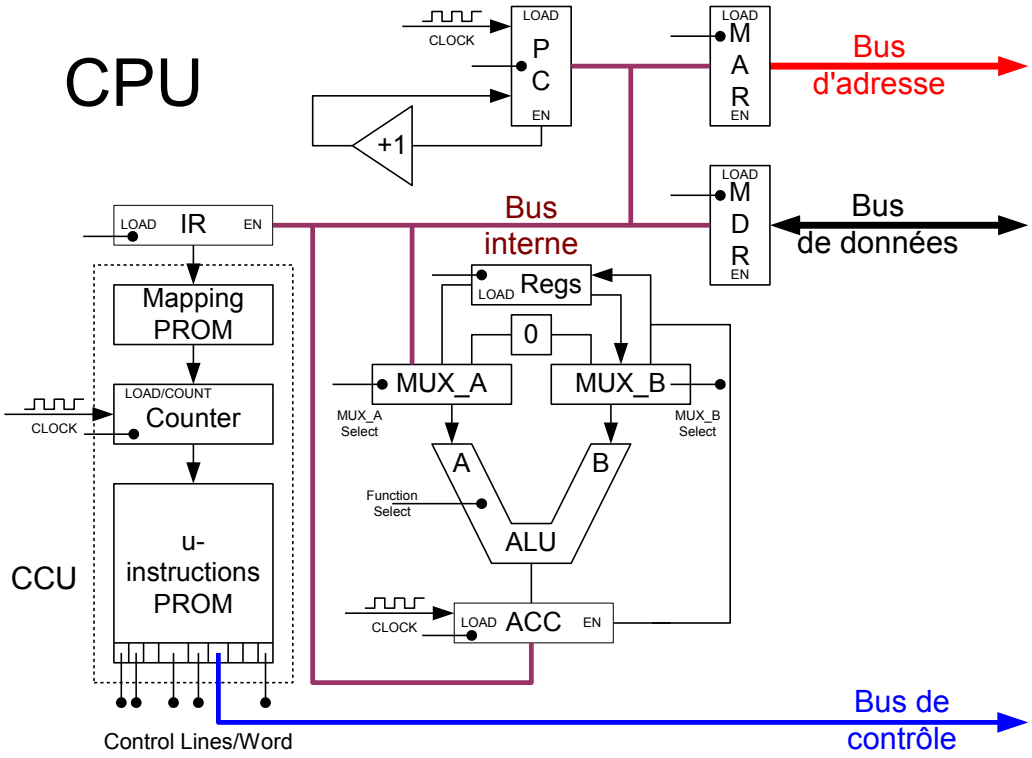

Figure 1: Architecture interne d'un microprocesseur simple pour la question 3.

(a) (3 points) Nommez et décrivez brièvement les trois étapes d'exécution d'une instruction dans un microprocesseur.

Solution:

1. "Fetch": lire la prochaine instruction à exécuter;
2. "Decode": décoder l'instruction (déterminer les micro-instructions à effectuer);
3. "Execute": exécuter ces étapes pour accomplir la tâche correspondant à l'instruction.

(b) (7 points) En utilisant le schéma de la figure 1, donnez les micro-instructions qui correspondent à l'instruction `STR R2 [R3]`. Vous pouvez utiliser la notation "`PC` $\rightarrow$ `IR`" pour représenter un déplacement du contenu du registre `PC` vers le registre `IR`, par exemple.

Solution: Tout d'abord, `PC` est placé dans `MAR`, et le bus de contrôle est activé en lecture. Ensuite, on transfert le contenu de `MDR` vers `IR`.

Ensuite, il y a deux principales étapes (l'ordre n'est pas important, tant que le signal de contrôle en écriture soit activé avec la deuxième étape):

1. (R2 vers MDR): On place la valeur de R2 dans l'accumulateur. Comme on passe par l'ALU, on additionne 0. Ensuite, on place la valeur de l'accumulateur sur le bus de données (MDR).
2. (R3 vers MAR): On place la valeur de R3 dans l'accumulateur. Comme on passe par l'ALU, on additionne 0. Ensuite, on place la valeur de l'accumulateur sur le bus d'adresse (MAR). Le bus de contrôle est activé en écriture.

Finalement, le PC doit être incrémenté pour lire la prochaine instruction.

4. Répondez aux questions suivantes sur la façon dont les données sont stockées dans un ordinateur.

   (a) (3 points) En complément-2, quel est l'intervalle de valeurs pouvant être représentées sur 4 bits? Sur 8 bits? Indiquez la valeur minimale et maximale pour chaque cas. Dérivez ensuite une formule générale en fonction du nombre de bits N.

   Solution:
   Sur 4 bits: -8 à +7, sur 8 bits: -128 à +127.
   Donc, de manière générale, on peut représenter les valeurs allant de -2^{N-1} jusqu'à $2^{N-1} - 1$.

   (b) (4 points) Indiquez quel(s) drapeau(x) de l'ALU seront activés lors des deux opérations suivantes en complément-2 sur 4 bits: $6 + 2$ et $-6 - 2$.

   Solution:
   $6 + 2 = 0110_b + 0010_b = 1000_b$: "overflow" est activé.
   $-6 - 2 = 1010_b + 1110_b = 11000_b$: "carry" est activé.

   (c) (3 points) Un nombre entier non-signé sur 16 bits peut aussi être stocké en ASCII en représentant chaque chiffre par sa valeur ASCII correspondante. Donnez un exemple où stocker le nombre en ASCII requiert 1) *moins* de mémoire; 2) *autant* de mémoire; et 3) *plus* de mémoire que de le représenter en binaire. Vous devez donner un exemple pour chacune de ces situations.

   Solution: ASCII utilise 8 bits pour stocker chaque caractère. Donc, si le nombre ne possède qu'un chiffre (0 à 9), nous n'avons besoin que de 8 bits pour le stocker en ASCII, et 16 en binaire. Pour les nombres entre 10 et 99, c'est équivalent (16 bits dans les deux cas). Pour tous les nombres > 99, ASCII aurait besoin d'au-moins 24 bits, tandis que la représentation binaire nécessite 16 bits.

5. (a) (1 point) Si un décodeur d'adresses utilise deux lignes du bus d'adresses pour le décodage, combien de composantes peuvent être contrôlées au maximum?

   Solution: 4.

   (b) (2 points) Un bus sert à connecter plusieurs composantes ensemble. Comment faire pour s'assurer qu'une seule composante est active à la fois?

   Solution: Le décodeur d'adresse sélectionne la bonne composante à activer, ce qui est fait via son entrée "enable". Lorsque le signal "enable" est activé, la composante se connecte au bus. Autrement, elle est en haute impédance sur le bus, c'est-à-dire qu'elle apparaît comme déconnectée du circuit.

   (c) On veut accéder à une donnée située en mémoire RAM et la placer dans un registre du microprocesseur.

      i. (1 point) Quelle devraient être les valeurs sur le bus de contrôle, d'adresses, et de données?

Solution: Contrôle = lecture, adresses = adresse voulue en RAM.

ii. (1 point) Laquelle de ces trois instructions devrait-on utiliser: `LDR`, `STR`, ou `MOV`?

Solution: `LDR`.

6. (10 points) La table suivante montre différentes adresses de la mémoire d'un ordinateur ARM. À partir de cette table, et en assumant que la valeur initiale de `PC` soit `0x0`, indiquez quelle sera la valeur des registres `R0`, `R1`, `R2`, `R3` et `PC` *après* l'exécution de l'instruction `B #-0x9C`.

   Note: l'annexe A pourrait vous aider à répondre à cette question.

Adresse	Instructions	Binaire
0x00000000	B #0x78	0xEA00001E
0x00000004	MUL R0, R7, R5	0xFFFFFF75
0x00000008	ANDEQ R0, R0, R2	0x000000E2
⋮	⋮	⋮
0x00000080	MOV R0, #4	0xE3A00004
0x00000084	LDR R1, [R0], #4	0xE5901000
0x00000088	LDR R2, [R0]	0xE5902004
0x0000008C	SUB R0, PC, R0	0xE04F0000
0x00000090	LSL R3, R2, #2	0xE0813002
0x00000094	B #-0x9C	0xEAFFFFFE

Solution: Après l'exécution, les registres auront les valeurs suivantes:

```
R0 = 0x0000008C
R1 = 0xFFFFFF75
R2 = 0x000000E2
R3 = 0x00000388
PC = 0x00000008
```

7. Les questions suivantes font référence à l'architecture ARM.

   (a) (2 points) Décrivez le rôle du programme assembleur, comme celui que l'on retrouve dans IAR.

   Solution: Il "traduit" le code assembleur (format texte, compréhensible par nous) en code binaire (compréhensible par le microprocesseur).

   (b) (2 points) Pourquoi ne peut-on pas référer à l'adresse d'une constante directement dans une instruction `MOV`?

   Solution: Car les adresses et les instructions sont sur 32 bits. On ne peut donc pas écrire une adresse à même une instruction: il n'y a pas assez de place!

   (c) (6 points) On vous donne le code assembleur suivant:

Adresse	Instructions
	`main`
`0x00000080`	`LDR R0, =maVariable`
`0x00000084`	`LDR R1, [R0]`
`0x00000088`	`MOV R1, #1`
`0x0000008C`	`ADD R0, R0, R1`
`0x00000090`	`B main`

   L'assembleur remplacera l'instruction à l'adresse `0x80` car le microprocesseur ne comprend pas ce qu'est l'étiquette `maVariable`. En supposant que l'assembleur utilise l'adresse `0x94`, par quelle instruction l'assembleur remplacera-t-il l'instruction originale?

   Solution: L'assembleur mettra l'adresse de maVariable à l'adresse `0x94`, et remplacera l'instruction `LDR R0, =maVariable` par `LDR R0, [PC, #12]`.

8. Répondez aux questions suivantes portant sur les interruptions.

   (a) (2 points) Quelle est la différence entre le traitement d'une interruption et un appel de fonction?

   Solution: Une interruption peut survenir n'importe quand.

   (b) (2 points) Qu'est-ce qu'une exception? Donnez un exemple.

   Solution: Une exception survient lorsque l'exécution d'un programme donne un résultat imprévu ou inattendu. Exemples possibles: division par 0, accès mémoire interdit, faute matérielle, instruction invalide, etc.

   (c) (2 points) Qu'est-ce que la table des vecteurs d'interruption? Quel programme la gère, habituellement?

Solution: La table des vecteurs d'interruption stocke l'adresse des routines de traitement des interruption, et ce, pour chaque interruption. Elle est habituellement gérée par le système d'exploitation.

(d) (4 points) Décrivez brièvement ce qui se passe lorsqu'une interruption est soulevée.

Solution: Les étapes suivantes surviennent:

1. Le contrôleur d'interruption détermine si l'interruption doit être traitée en fonction de sa priorité;
2. Si l'interruption doit être traitée, le microprocesseur obtient l'adresse de l'ISR dans la table des vecteurs d'interruption;
3. Les registres, drapeaux de l'ALU, et l'adresse de retour sont sauvegardés sur la pile;
4. On place l'adresse de l'ISR dans PC, et cette dernière est exécutée pour traiter l'interruption;
5. Lorsque l'ISR est terminée, on replace l'environnement à sa place et le programme continue son exécution normalement (à moins de faute importante, auquel cas le programme ne reprend jamais).

9. (10 points) Une multiplication peut être représentée par une application successive d'additions. Par exemple, $3 \times 5 = 5 + 5 + 5 = 3 + 3 + 3 + 3 + 3 = 15$. Pour cette question, vous aurez à écrire une fonction en assembleur ARM qui implémente une multiplication grâce à une série d'additions.

   Détails:

   1. Ne traitez que le cas où les deux valeurs sont toujours strictement positives (> 0).
   2. Votre fonction doit prendre ses arguments dans les registres `R0` et `R1`, et doit retourner le résultat dans `R0`.
   3. Votre fonction *ne doit pas* utiliser l'instruction `MUL`.
   4. L'appel de votre fonction *ne doit pas* modifier les registres autres que `R0`.
   5. L'annexe A contient une liste d'instructions et de codes de conditions pouvant vous être utiles.

```
main
  MOV R0, #4
  MOV R1, #8
  BL maMultiplication  ; appel de votre fonction, qui devrait calculer 4 x 8
  MOV R3, R0           ; récupère valeur de retour, qui devrait contenir 32
  ...
maMultiplication
  ; écrivez le code de votre fonction.
```

   Indices:

   1. Additionnez la valeur contenue dans `R0`. Combien de fois faut-il l'additionner? Il faut le faire "`R1`-1" fois. Par exemple, plus haut nous avons écrit $3 \times 5 = 5 + 5 + 5$. Cela est donc deux additions: $(5 + 5) + 5$.
   2. Testez votre fonction avec des valeurs simples (ex: 1×2, 2×3) pour vous assurer de son bon fonctionnement.

   Question bonus:

   (a) (5 points (bonus)) Modifiez votre code pour traiter correctement le cas où `R0` ou `R1` auraient une valeur de 0.

Solution: Le code suivant est un exemple (incluant le bonus):

```
maMultiplication
    ; Nous allons additioner la valeur dans R0, R1 fois
    PUSH {R1,R3}       ; Nous allons avoir besoin d'un registre supplémentaire

    CMP R1, #0      ; Cas spécial: R1 = 0
    BEQ zero

    MOV R3, R0      ; sauvegarder la valeur
```

```
debut
    CMP R1, #1      ; si R1 est égal à 1, nous avons terminé!
    BEQ fin
    ; R1 est plus grand que 1. Il faut additionner!
    ADD R0, R0, R3  ; R0 = R0 + R3
    SUB R1, R1, #1  ; R1 = R1 - 1
    B debut         ; et on recommence!
zero
    MOV R0, #0      ; R1 = 0, donc le résultat R0 = 0
fin
    POP {R1,R3}        ; Restaurons R3
    BX LR
```

10. Indiquez si chacun des énoncés suivants est vrai ou faux.

(a) (1 point) Selon la loi de Moore, si le nombre de transistors par pouce carré d'un microprocesseur typique était K en 2008, il était environ $8K$ en 2014.

Solution: Vrai

(b) (1 point) Un ordinateur ayant plusieurs mémoires aura plusieurs bus distincts pour accéder à chaque mémoire individuellement.

Solution: Faux

(c) (1 point) Bien que la longueur d'une instruction peut varier d'un jeu d'instructions à un autre, toutes les instructions ont toujours la même taille à l'intérieur d'un jeu d'instructions.

Solution: Faux

(d) (1 point) Le jeu d'instructions ARM est de type CISC.

Solution: Faux

(e) (1 point) Pour diviser par deux en décalant des bits vers la droite, il faut que le bit le plus significatif du nombre décalé soit dupliqué.

Solution: Vrai

(f) (1 point) Une mémoire dynamique est une mémoire qui peut être lue seulement.

Solution: Faux

(g) (1 point) Une mémoire est décrite avec deux nombres: la taille des mots, et le nombre d'adresses.

Solution: Vrai

(h) (1 point) Dans une architecture "Memory-Mapped I/O" (comme en ARM), on utilise des instructions différentes selon qu'on veuille accéder aux mémoires ou aux périphériques.

Solution: Faux

(i) (1 point) Le décodeur d'adresse détermine quelle composante sera activée en fonction de la valeur placée sur le bus d'adresses.

Solution: Vrai

(j) (1 point) Le bus de données doit avoir la même taille que la largeur des mots stockés en mémoire.

Solution: Faux

A Annexe: Instructions ARM et codes de conditions

Dans la table suivante, `Op1` dénote une opérande de type 1, et `Op2` une opérande de type 2.

Mnémonique	Description
`ADD Rd, Rs, Op1`	`Rd = Rs + Op1`
`ADC Rd, Rs, Op1`	`Rd = Rs + Op1 + Carry`
`AND Rd, Rs, Op1`	`Rd = Rs AND Op1`
`ASR Rd, Rs, #imm`	`Rd = Rs / 2`$^{\texttt{imm}}$
`Bcc Offset`	`PC = PC + Offset`, si `cc` est rencontré
`BLcc Offset`	Comme `B`, `LR` = Adr. de l'instr. suivante
`CMP Rs, Op1`	Change les drapeaux comme `Rs-Op1`
`LDR Rd, [Rs, Op2]`	`Rd = Mem[Rs + Op2]`
`LDR Rd, [Rs], Op2`	`Rd = Mem[Rs], Rs = Rs + Op2`
`LDR Rd, [Rs, Op2]!`	`Rs = Rs + Op2, Rd = Mem[Rs]`
`LSL Rd, Rs, #imm`	`Rd = Rs x 2`$^{\texttt{imm}}$
`MUL Rd, Rs, Op1`	`Rd = Rs x Op1`
`MVN Rd, Op1`	`Rd = !Op1` (inverse les bits)
`POP {Reg List}`	Met la liste de registres sur la pile
`PUSH {Reg List}`	Met la liste de registres sur la pile
`SBC Rd, Rs, Op1`	`Rd = Rs - Op1 - C`
`STR Rd, [Rs, Op2]`	`Mem[Rs + Op2] = Rd`
`STR Rd, [Rs], Op2`	`Mem[Rs] = Rd, Rs = Rs + Op2`
`STR Rd, [Rs, Op2]!`	`Rs = Rs + Op2, Mem[Rs] = Rd`
`SUB Rd, Rs, Op1`	`Rd = Rs - Op1`

Mnémonique	Condition	Mnémonique	Condition
`CS`	Carry Set	`CC`	Carry Clear
`EQ`	Equal (Zero Set)	`NE`	Not Equal (Zero Clear)
`VS`	Overflow Set	`VC`	Overflow Clear
`GT`	Greater Than	`LT`	Less Than
`GE`	Greater Than or Equal	`LE`	Less Than or Equal
`PL`	Plus (Positive)	`MI`	Minus (Negative)
`HI`	Higher Than	`LO`	Lower Than
`HS`	Higher or Same	`LS`	Lower or Same