

Examen UE INF401 : Architectures des Ordinateurs

Mai 2021, durée 2 h 00

Document autorisé : 1 feuille A4 Recto/Verso de notes personnelles manuscrites.

Les caleuses et téléphones portables sont interdits.

La plupart des questions sont indépendantes, si vous avez des difficultés avec une question, passez à la suivante.
Le barème est donné à titre indicatif.

1 Question de cours ARM (3 points)

- (a) Rappelez la structure générale finale du schéma de traduction systématique en code ARM d'une fonction avec utilisation de la pile tel qu'elle a été vue en cours.
- (b) Dessinez l'état de la pile au début de l'exécution du corps d'une fonction avec un résultat entier, après exécution du prologue, dans le cas où la fonction comporte 2 arguments entiers (a et b), nécessite 1 variable locale entière (l) et utilise 3 registres temporaires (R1, R2 et R4). Tous les entiers, relatifs, sur 32 bits.

Éléments de correction.

- a)
 - 1) empiler l'adresse de retour (lr)
 - 2) empiler la valeur fp de l'appelant
 - 3) placer fp pour repérer les variables de l'appelée
 - 4) allouer la place pour les variables locales
 - 5) empiler les variables temporaires (registres) utilisées
- 6) **corps de la fonction**
- 7) si fonction, le résultat est rangé en **fp+8**
- 8) dépiler les variables temporaires (registres) utilisées
- 9) libérer la place allouée aux variables locales
- 10) dépiler fp
- 11) dépiler l'adresse de retour (lr)
- 12) retour à l'appelant : BX lr

b)

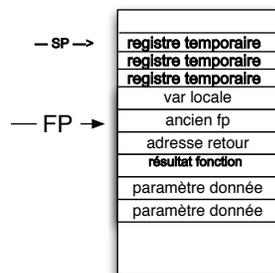


FIGURE 1 – Pile à l'exécution

2 Programmation de la fonction d'Ackermann en langage d'assemblage (ARM)

voir caseine

3 Automate, microprogrammation et processeur (7 points)

Dans cette partie, nous enrichissons le processeur fictif vu lors du cours et dont la partie opérative est représentée dans la figure ci-dessous :

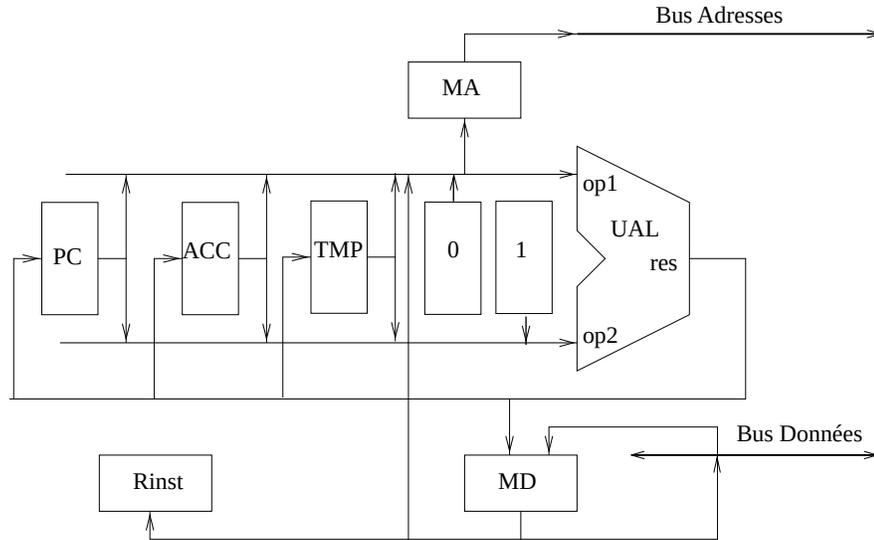


FIGURE 2 – Partie opérative du processeur

Structure de la partie opérative : micro-actions et micro-conditions. Dans la partie opérative on a les registres suivants : PC, ACC, Rinst, MA (memory address), MD (memory data) et TMP. Les transferts possibles sont les suivants :

$MD \leftarrow Mem[MA]$	lecture d'un mot mémoire.	C'est la seule possibilité en lecture!
$Mem[MA] \leftarrow MD$	écriture d'un mot mémoire	C'est la seule possibilité en écriture!
$Rinst \leftarrow MD$	affectation	Affectation spécifique à Rinst
$PC \leftarrow PC + 1$	incréméntation	Incréméntation spécifique à PC
$reg_0 \leftarrow 0$	mise à zéro	reg_0 est PC, ACC, ou TMP
$reg_0 \leftarrow reg_1$	affectation	reg_0 est PC, ACC, TMP, MA, ou MD reg_1 est PC, ACC, TMP, ou MD
$reg_0 \leftarrow reg_1 \ll$	décalage à gauche d'un bit	reg_0 est PC, ACC, TMP, ou MD reg_1 est PC, ACC, TMP, ou MD
$reg_0 \leftarrow reg_1 \text{ op } reg_2$	opération	reg_0 est PC, ACC, TMP, ou MD reg_1 est 0, PC, ACC, TMP, ou MD reg_2 est 1, PC, ACC, TMP, ou MD op : + ou -
$reg_0 \leftarrow (reg_1 \ll) \text{ op } reg_2$	opération avec décalage	reg_0 est PC, ACC, TMP, ou MD reg_1 est PC, ACC, TMP, ou MD reg_2 est 1, PC, ACC, TMP, ou MD op : + ou -

Seul le registre Rinst permet de faire des tests : $Rinst = \text{entier}$ (c'est donc la seule micro-condition).

Le langage d'assemblage. Nous rappelons que ce processeur comporte un seul registre de données directement visible par le programmeur : ACC (pour accumulateur). La taille du codage d'une adresse et d'une donnée est un mot de 4 bits. La mémoire comporte donc 16 mots adressables.

Les instructions sont décrites ci-dessous. On donne pour chacune une syntaxe de langage d'assemblage, le code machine, la sémantiques et la taille du codage :

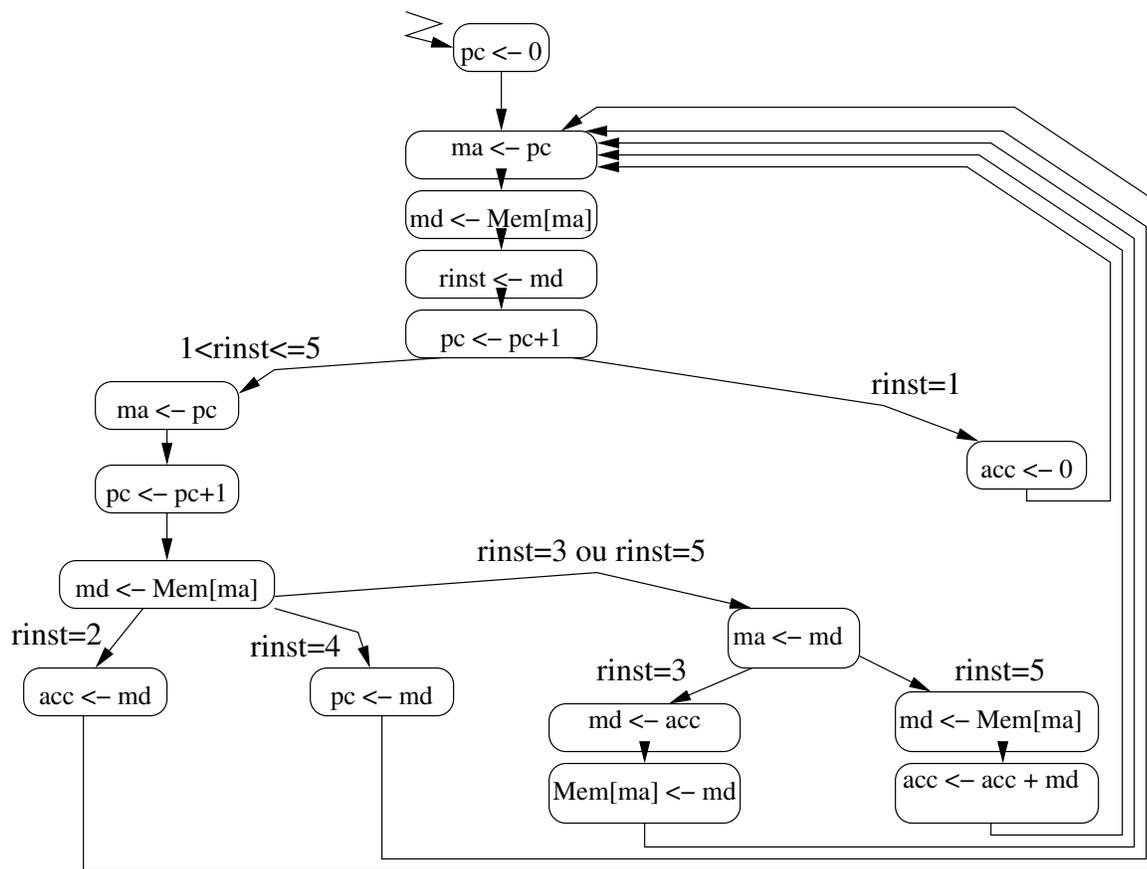


FIGURE 3 – Graphe de contrôle

instruction	code	signification	mots
clr	1	mise à zéro du registre ACC	1
ld# vi	2	chargement de la valeur immédiate <i>vi</i> dans ACC	2
st ad	3	rangement en mémoire à l'adresse <i>ad</i> du contenu de ACC	2
jmp ad	4	saut incondtionnel à l'adresse <i>ad</i>	2
add ad	5	mise à jour de ACC avec la somme de ACC et de la valeur à l'adresse <i>ad</i>	2

Les instructions sont codées sur **1 ou 2 mots de 4 bits chacun** :

- le premier mot représente le code de l'opération (clr, ld, st, jmp, add) ;
- le deuxième mot, s'il existe, contient une adresse (*ad*) ou bien une constante (*vi*).

L'automate d'interprétation de ce langage est donné dans la figure 3.

Question.

- (e) Expliquez l'interprétation d'une instruction `jmp` suivant le graphe de contrôle de la figure 3, vous pouvez utiliser un exemple. (1 point).

Éléments de correction. L'interprétation de `jmp` commence par récupérer l'instruction : `ma <- pc`, `md <- Mem[ma]`, `rinst <- md`, `pc <- pc+1`, puis récupère le paramètre (adresse de saut) : `ma <- pc`, `pc <- pc+1`, `md <- Mem[ma]`, puis exécute le saut `pc <- mb` et reboucle sur la récupération de la prochaine instruction.

Enrichissement du langage. Nous souhaitons enrichir le langage de notre processeur en ajoutant deux instructions de lecture/écriture en mémoire permettant de gérer le parallélisme (test-and-set et fetch-and-add).

Sémantique opérationnelle des instructions à ajouter. Les instructions à ajouter, leur code, leur sémantiques et la taille de leur codage sont données dans la table ci-dessous :

instruction	code	signification	mots
<code>tns@ ad</code>	7	échange de la valeur à l'adresse <code>ad</code> et de la valeur de l'accumulateur	2
<code>fna@ ad</code>	8	mise à jour de ACC avec la valeur à l'adresse <code>ad</code> puis ajout de 1 à la valeur à l'adresse <code>ad</code>	2

État initial de la mémoire. On suppose que le programme suivant est stocké en mémoire, la zone `.text` commence à l'adresse `0` et la zone `.data` commence à l'adresse `14`.

Adresse	Valeur en mémoire
0	1
1	4
2	7
3	2
4	1
5	3
6	14
7	5
8	14
9	4
10	3
11	0
12	0
13	0
14	1
15	0

Questions.

- (f) Proposez un programme assembleur ayant une image mémoire identique à celle donnée pour l'état initial de la mémoire. Pour la syntaxe des zones, étiquettes, commentaires, pseudo-instruction, directives, etc. vous pouvez vous inspirer de ARM. **(1 point)**.

Éléments de correction.

Adresse	Valeur en mémoire	Version Asm
0	1	<code>clr</code>
1	4	<code>jmp et7</code>
2	7	
3	2	<code>et3 : ld# 1</code>
4	1	
5	3	<code>st et14</code>
6	14	
7	5	<code>et7 : add et14</code>
8	14	
9	4	<code>jmp et3</code>
10	3	
11	0	
12	0	
13	0	<code>et13 : .val 0</code>
14	1	<code>et14 : .val 1</code>
15	0	<code>et15 : .val 0</code>

- (g) Simulez, en suivant le graphe de contrôle de la figure 3, l'exécution au niveau des micro-actions du début du programme stocké en mémoire. Pour répondre, vous remplirez un tableau de simulation similaire à celui défini ci-après (avec une ligne par micro-action, 20 lignes en tout de 1 à 20, ligne 0 exclue, la ligne 1 est donnée.) **(1,5 point)**.

Éléments de correction.

	Micro-Action (exécutée)	PC	Rinst	ACC	MA	MD	Mem[14]	Commentaires
0		?	?	?	?	?	1	
1	pc ← 0	0						
2	ma ← pc				0			
3	md ← Mem[ma]					1		
4	rinst ← md		1					
5	pc ← pc +1	1						
6	acc ← 0			0				
7	ma ← pc				1			
8	md ← Mem[ma]					4		
9	rinst ← md		4					
10	pc ← pc +1	2						
11	ma ← pc				2			
12	pc ← pc +1	3						
13	md ← Mem[ma]					7		
14	pc ← md	7						
15	ma ← pc				7			
16	md ← Mem[ma]					5		
17	rinst ← md		5					
18	pc ← pc +1	8						
19	ma ← pc				8			
20	md ← Mem[ma]					14		

(h) Donnez les modifications à apporter à l'automate fourni par le graphe de contrôle de la figure 3 afin d'interpréter les instructions supplémentaires :

- **tns@ ad (1,5 point)**
- **fna@ ad (1,5 point)**

Indication : vous pouvez utiliser le registre TMP.

(i) Est-ce que vous pouvez factoriser une partie des états ajoutés, l'automate obtenu peut-il être optimisé pour le nombre d'états ? Si oui, montrez comment. **(0,5 point)**

Éléments de correction.

Pour **tns@ ad**, on ajoute une alternative à la fin de l'exécution de l'instruction **add ad**. À la place de la dernière action, on ajoute la suite d'actions suivante (alternative sur le code de **rinst=7**) : **tmp ← md, md ← acc, acc ← tmp, Mem[ma] ← md**.

Pour **fna@ ad**, on ajoute une seconde alternative à la fin de l'exécution de l'instruction **add ad**. À la place de la dernière action, on ajoute la suite d'actions suivante (alternative sur le code de **rinst=8**) : **acc ← md, md ← md + 1, Mem[ma] ← md**.

À faire : le graphe de l'automate pour que tout soit clair.

Remarque, les deux dernières actions peuvent être partagées.