UNIVERSITE GRENOBLE ALPES
Licence de Sciences et technologie - Parcours MIN, INF

Examen UE INF401 : Architectures des Ordinateurs

Mai 2019, durée 2 h 00
Document autorisé : 1 feuille A4 Recto/Verso de notes manuscrites.
Les calculettes 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)

Donner trois des registres du processeur ARM (parmi $R_{10} - R_{15}$) ayant une fonction particulière. Pour chacun, donner

— son nom abrégé et son nom usuel complet (en français ou en anglais) ;
— sa fonction (une phrase) et
— un exemple d'utilisation (une à deux instructions) avec une explication (une phrase).

2 Programmation en langage d'assemblage ARM (10 points)

Le but de cet exercice est de calculer le maximum des éléments d'un tableau d'entiers naturels puis d'afficher cette valeur en binaire. Pour cela, on considère la zone `.data` suivante :

```
.data
R:    .asciz "Max en binaire :"
@             1234567890123456   @indication pour compter les caractères
      .balign 4
T:    .word 12
      .word 11
      .word 14
      .word 44
      .word 99
      .word 452
      .word 1
      .word 4
      .word 42
      .word 241
```

Questions.

(a) Rappelez comment est représentée une chaine de caractères en mémoire et donnez la taille en octets de la chaine de caractères `R`. **(0.5 point)**

(b) Quel est le rôle de la directive `.balign 4` ? Pourquoi s'en sert-on ici ? **(0.5 point)**

(c) Quelle est la taille en octets du tableau `T` ? **(0.5 point)**

(d) En supposant que la zone `.data` est stockée à partir de l'adresse `0xC000`, quelle sera l'adresse (en hexadécimale) du début du tableau `T` ? **(0.5 point)**

2.1 Écrire une procédure

```
1: Procédure affBinaire(a: entier naturel)
2:      si a != 0 alors
3:           affBinaire(a/2)
4:           si a est pair alors
5:                EcrCar('0')
6:           sinon
7:                EcrCar('1')
8:           fin si
9:      fin si
```

(e) En appliquant **la méthode systématique vue en cours**, traduisez en `ARM` le corps de la procédure `affBinaire` ci-dessus en tenant compte des indications suivantes :
- `a` est un paramètre passé par la pile.
- `EcrCar` est une procédure du fichier `es.s` utilisé en TP (un rappel des fonctions de `es.s` est donné en annexe 0).
- Pour les variables temporaires vous utiliserez les registres `r0`, `r1` et `r2`, qui devront être sauvegardés en pile avant utilisation, puis restaurés suivant la convention du cours.

(4 points)

(f) En supposant un appel d'`affBinaire(42)` à partir du programme principal (`main`), dessinez l'état de la pile juste avant le premier appel récursif effectif (`bl`) en ligne 3 de cette procédure. **(1 point)**

2.2 Manipulation d'un tableau et appel d'une procédure

Nous allons maintenant compléter la zone `.text` suivante :

```
      .text
      .global main
main:
      @ partie à compléter
      b exit
ptrT:    .word T
ptrR:    .word R
```

(g) Complétez la zone `.text` ci-dessus avec le code `ARM` de l'algorithme ci-dessous en tenant compte des indications suivantes :
- Pour les variables `max` et `i`, vous utiliserez les registres `r2` et `r3`.
- `T` est défini dans la zone `.data` proposée au début de la section 2.

(2 points)

```
20: Programme principal
21:      max := T[0]
22:      pour i de 1 à 9 faire
23:           si max < T[i] alors
24:                max := T[i]
25:           fin si
26:     fin pour
```

(h) Traduisez en `ARM` les deux appels de fonctions ci-dessous en tenant compte des indications suivantes :
- On suppose que `max` est stocké dans le registre `r2`.
- `EcrChaine` est une procédure du fichier `es.s` utilisé en TP (un rappel des fonctions de `es.s` est donné en annexe 0).
- Le paramètre de la procédure `affBinaire` doit être stocké en pile, d'après la convention du cours.

(1 point)

```
27:      EcrChaine("Max en binaire :")
28:      affBinaire(max)
```

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 :

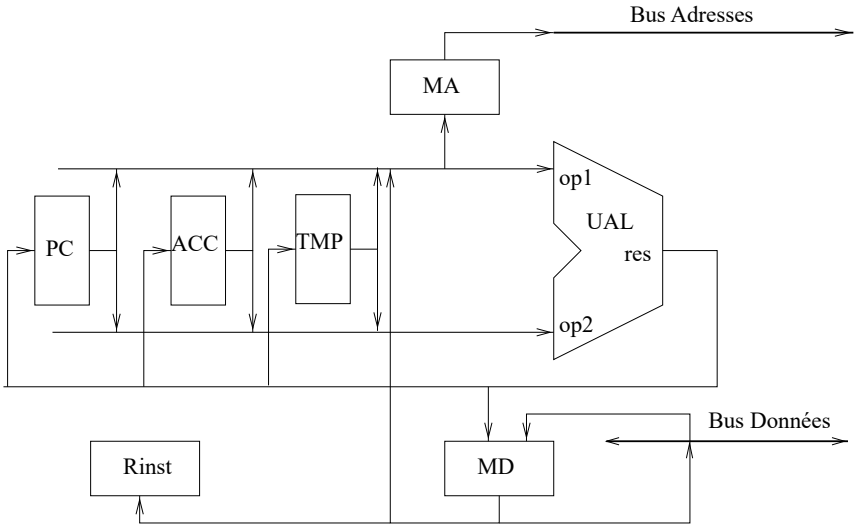

FIGURE 1 – Partie opérative du processeur

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**	incrémentation	Incrémentation 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**$_1$ <<	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**$_1$ **op reg**$_2$	opération	reg$_0$ est `PC`, `ACC`, `TMP`, ou `MD` reg$_1$ est `PC`, `ACC`, `TMP`, ou `MD` reg$_2$ est `PC`, `ACC`, `TMP`, ou `MD` op : + ou −
reg$_0$ $\leftarrow$ **(reg**$_1$ << **) 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 `PC`, `ACC`, `TMP`, ou `MD` op : + ou −

Seul le registre `Rinst` permet de faire des tests : `Rinst` = 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 :

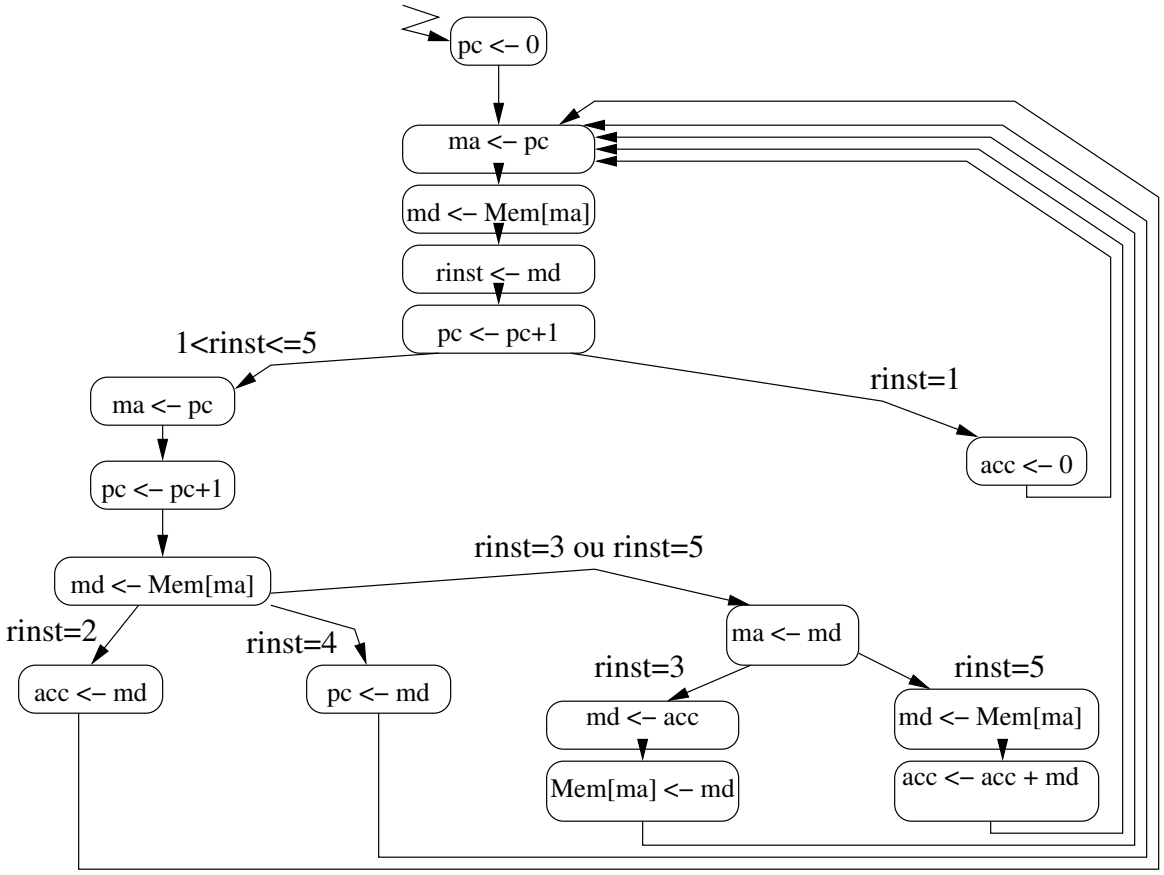

FIGURE 2 – Graphe de contrôle

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

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

— 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 2.

Enrichissement du langage. Nous souhaitons enrichir le langage de notre processeur en ajoutant une instruction de multiplication réduite (pour *3, *5), avec deux modes d'adressage (adressage implicite pour `ACC` et mode d'adressage direct).

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
`3ac`	9	multiplication de `ACC` par 3	1
`5ac`	10	multiplication de `ACC` par 5	1
`m3@ ad`	11	multiplication du mot à l'adresse mémoire `ad` par 3	2
`m5@ ad`	12	multiplication du mot à l'adresse mémoire `ad` par 5	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	2
1	1
2	3
3	15
4	5
5	15
6	5
7	15
8	5
9	14
10	4
11	2
12	0
13	0
14	1
15	0

Questions.

(a) Expliquez l'interprétation d'une instruction `st` suivant le graphe de contrôle de la figure 2, vous pouvez utiliser un exemple. **(1 point)**.

(b) Proposez un programme assembleur ayant une image mémoire identique à celle donnée pour l'état initial de la mémoire **(1 point)**.

(c) Simulez, en suivant le graphe de contrôle de la figure 2, 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, 10 lignes en tout) **(1 point)**.

(d) Simulez, au niveau assembleur, l'exécution de treize instructions du programme en donnant en particulier les valeurs de `ACC`. Vous pourrez utiliser un tableau de simulation adapté à la situation. **(1 point)**.

(e) Donnez les modifications à apporter à l'automate fourni par le graphe de contrôle de la figure 2 afin d'interpréter les instructions supplémentaires :
   — `3ac` et `5ac` **(1,5 points)**
   — `m3@ ad` et `m5@ ad` **(1,5 points)**

   Indication : vous pourrez utiliser le registre `TMP` et les micro-actions de décalage.

Tableau de simulation (pour la question c, à recopier sur votre copie)

	Micro-Action (exécutée)	PC	Rinst	ACC	MA	MD	Mem[14]	Mem[15]
0		?	?	?	?	?	1	0
1	pc ← 0	0						
2								
3								
4								
5								
6								
7								
8								
9								
10								

4 ANNEXE 0 : fonctions d'entrée/sortie

Nous rappelons les principales fonctions d'entrée/sortie du fichier `es.s`.

— `bl EcrHexa32` affiche le contenu de `r1` en hexadécimal.
— `bl EcrZdecimal32` affiche le contenu de `r1` en décimal sous la forme d'un entier relatif de 32 bits.
— `bl EcrZdecimal16` affiche le contenu de `r1` en décimal sous la forme d'un entier relatif de 16 bits.
— `bl EcrZdecimal8` affiche le contenu de `r1` en décimal sous la forme d'un entier relatif de 8 bits.
— `bl EcrNdecimal32` affiche le contenu de `r1` en décimal sous la forme d'un entier naturel de 32 bits.
— `bl EcrNdecimal16` affiche le contenu de `r1` en décimal sous la forme d'un entier naturel de 16 bits.
— `bl EcrNdecimal8` affiche le contenu de `r1` en décimal sous la forme d'un entier naturel de 8 bits.
— `bl EcrChaine` affiche la chaîne de caractères dont l'adresse est dans `r1`.
— `bl EcrCar` affiche le caractère dont le code ASCII est dans `r1`.
— `bl Lire32` récupère au clavier un entier 32 bits et le stocke à l'adresse contenue dans `r1`.
— `bl Lire16` récupère au clavier un entier 16 bits et le stocke à l'adresse contenue dans `r1`.
— `bl Lire8` récupère au clavier un entier 8 bits et le stocke à l'adresse contenue dans `r1`.
— `bl LireCar` récupère au clavier un caractère et stocke son code ASCII à l'adresse contenue dans `r1`.

5 ANNEXE I : instructions du processeur ARM

Nom	Explication du nom	Opération	Remarque
`AND`	AND	et bit à bit	
`EOR`	Exclusive OR	ou exclusif bit à bit	
`SUB`	SUBstract	soustraction	
`RSB`	Reverse SuBstract	soustraction inversée	
`ADD`	ADDition	addition	
`ADC`	ADdition with Carry	addition avec retenue	
`SBC`	SuBstract with Carry	soustraction avec emprunt	
`RSC`	Reverse Substract with Carry	soustraction inversée avec emprunt	
`TST`	TeST	et bit à bit	pas `rd`
`TEQ`	Test EQuivalence	ou exclusif bit à bit	pas `rd`
`CMP`	CoMPare	soustraction	pas `rd`
`CMN`	CoMpare Not	addition	pas `rd`
`ORR`	OR	ou bit à bit	
`MOV`	MOVe	copie	pas `rn`
`BIC`	BIt Clear	et not bit à bit	
`MVN`	MoVe Not	not (complément à 1)	pas `rn`
`MUL`	MULtiplication	multiplication tronquée	pas `r15`
`B**`	Branch	rupture de séquence conditionnelle	** = condition, cf. ci-dessous
`BL`	Branch and Link	appel sous-programme	adresse de retour dans r14
`LDR`	Load Register	lecture mémoire	
`STR`	Store Register'	écriture mémoire	

L'opérande source d'une instruction `MOV` peut être une valeur immédiate notée `#5` ou un registre noté `Ri`, `i` désignant le numéro du registre. Il peut aussi être le contenu d'un registre sur lequel on applique un décalage de `k` bits ; on note `Ri, DEC #k`, avec `DEC` $\in$ {`LSL`, `LSR`, `ASR`, `ROR`}.

6 ANNEXE II : codes conditions du processeur ARM

La table suivante donne les codes de conditions arithmétiques `**` pour l'instruction de rupture de séquence `B**`.

mnémonique	signification	condition testée
EQ	égal	Z
NE	non égal	$\overline{Z}$
CS/HS	$\geq$ dans N	C
CC/LO	$<$ dans N	$\overline{C}$
MI	moins	N
PL	plus	$\overline{N}$
VS	débordement	V
VC	pas de débordement	$\overline{V}$
HI	$>$ dans N	$C \wedge \overline{Z}$
LS	$\leq$ dans N	$\overline{C} \vee Z$
GE	$\geq$ dans Z	$(N \wedge V) \vee (\overline{N} \wedge \overline{V})$
LT	$<$ dans Z	$(N \wedge \overline{V}) \vee (\overline{N} \wedge V)$
GT	$>$ dans Z	$\overline{Z} \wedge ((N \wedge V) \vee (\overline{N} \wedge \overline{V}))$
LE	$\leq$ dans Z	$Z \vee (N \wedge \overline{V}) \vee (\overline{N} \wedge V)$
AL	toujours	