UNIVERSITE Joseph FOURIER, Grenoble
Licence de Sciences et technologie - Parcours MIN, INF, IAG

Contrôle continu UE INF241 : Introduction aux Architectures Logicielles et Matérielles

17 Mars 2014
Durée 1 h 30
Documents, calculettes, téléphones portables non autorisés
Le barême est donné à titre indicatif

1 Numération (4 points)

Pour cet exercice, **toutes les réponses devront être justifiées.**

(a) Donnez le code en binaire sur 8 bits des entiers relatifs suivants (codage en complément à deux) : $(-47)_{10}$ et $(+112)_{10}$. **(1 point)**

(b) Donnez le code en hexadecimal de $(56)_{10}$ et $(1011\ 1111\ 1010\ 0101)_2$. **(1 point)**

(c) Pour chacune des additions binaires suivantes : $(+112)_{10} + (-47)_{10}$ et $(+112)_{10} + (+47)_{10}$,

1. donnez le résultat apparent sur 8 bits (codage en complément à deux), **(0,5 point)**
2. donnez les valeurs des indicateurs Z, N, C et V. **(0,5 point)**

(d) Pour chacune des soustractions binaires suivantes : $(+112)_{10} - (+47)_{10}$ et $(-47)_{10} - (+112)_{10}$,

1. donnez le résultat apparent sur 8 bits (codage en complément à deux), **(0,5 point)**
2. donnez les valeurs des indicateurs Z, N, C et V. **(0,5 point)**

2 Processeur à accumulateur (6 points)

On travaille avec un processeur relié à de la mémoire. Les **adresses mémoires** sont des mots de **16 bits**. La mémoire est organisée en **octets**.

Le processeur a un registre accumulateur `Acc` de taille **8 bits**.

Il y a un registre d'état qui comporte un bit : `Z`. La comparaison de l'accumulateur avec une valeur immédiate positionne le bit du mot d'état.

Le compteur de programme (`pc`) **repère l'instruction qui suit celle qui est en cours d'exécution.**

Les opérations possibles sur l'accumulateur sont :

— `load#` charger l'accumulateur avec une valeur immédiate
— `load` charger l'accumulateur avec un mot de la mémoire
— `add#` une valeur immédiate à l'accumulateur
— `lsl` décaler le contenu de l'accumulateur d'1 position binaire à gauche
— `store` ranger le contenu de l'accumulateur dans la mémoire

Il y a deux types d'instructions de rupture de séquence :

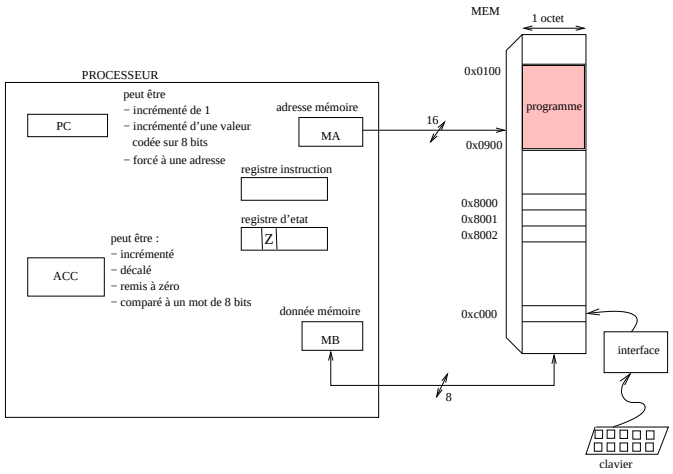

FIGURE 1 – Processeur et mémoire

code instruction	instruction	signification	codage
1	`load adr`	`Acc <-- Mem[adr]`	3 octets
2	`load# vi`	`Acc <-- vi`	2 octets
3	`store adr`	`Mem[adr] <-- Acc`	3 octets
4	`add# vi`	`Acc <-- Acc + vi`	2 octets
5	`lsl`	`Acc <-- Acc * 2`	1 octet
6	`lsr`	`Acc <-- Acc / 2`	1 octet
7	`cmp# vi`	`Acc - vi`	2 octets
8	`beq depl`	si Z=1 `pc <-- pc + depl`	2 octets
9	`bne depl`	si Z=0 `pc <-- pc + depl`	2 octets
A	`jmp adr`	`pc <-- adr`	3 octets

— branchement conditionnel : `bne` ou `beq`
branchement relatif, opérande = déplacement sur **8 bits**
— branchement inconditionnel : `jmp`
branchement absolu, opérande = adresse sur **16 bits**

Le code opération est toujours sur **1 octet** (le premier), l'opérande quand il existe est sur **1 octet** (`vi`, `depl`) ou **2 octets** (`adr`, le premier représente les poids forts de l'adresse).

Considérons le code suivant, où N est une valeur immédiate entière et positive :

```
        load# N
        store 0xC000
        load  0xC000
        store 0x8000
        load# 0
        store 0x9000
DEBUT:  load  0x8000
        cmp#  1
```

```
       bne   SUITE
       jmp   FIN
SUITE: lsr
       store 0x8000
       load  0x9000
       add#  1
       store 0x9000
       jmp   DEBUT
FIN:
```

— Quelle taille maximum de mémoire ce processeur peut-il gérer **(1 point)** ?
— Que fait le code précédent **(2 points)** ?
— En supposant que l'adresse de chargement est 0xA000, donnez l'image en mémoire (adresse et code binaire) de chaque instruction du code précédent **(3 points)**.

3 Programmation en ARM (11 points)

Nous rappelons que les caractères d'une chaine de caractères sont stockés dans un tableau d'octets terminé par 0.

1. Rappelez le nom de la zone où sont déclarées les variables non-initialisées **(0,5 point)**.
2. Déclarez une variable non-initialisée `buffer` de taille 50 octets **(0,5 point)**.
3. Quelle est la taille maximale d'une chaîne de caractères pouvant être stockée dans la variable `buffer` **(0,5 point)** ?
4. Expliquez dans le détail le code suivant **(1,5 point)** :

```
main:
        ldr r0,ptrBuffer
        mov r2,#0
boucle: add r1,r0,r2
        bl LireCar
        ldrb r3,[r1]
        cmp r3,#'\n'
        beq finboucle
        add r2,r2,#1
        bal boucle
finboucle:
        add r1,r0,r2
        mov r3,#0
        strb r3,[r1]

        bal exit
ptrBuffer: .word buffer
```

5. Rappelez comment tester si un caractère est une lettre minuscule **(0,5 point)**.
6. Rappelez comment convertir une lettre minuscule en majuscule **(0,5 point)**.
7. Donnez le code ARM qui teste si un caractère stocké dans $r0$ est une minuscule et le cas échéant la convertit en majuscule **(1 point)**.

8. Traduire en ARM le code suivant, i sera implanté dans le registre $r1$ **(3 points)**.

```
i:=0
tant que buffer[i] != 0 faire
     si buffer[i] est une minuscule alors
        convertir buffer[i] en majuscule
     fin si
     i:=i+1
fin tant que
```

9. Traduire en ARM le code suivant, i sera implanté dans le registre $r1$ et taille dans le registre $r2$ **(3 points)**.

```
i:=0
tant que i < taille/2 faire
     c=buffer[i]
     buffer[i]:=buffer[taille-1-i]
     buffer[taille-1-i]:=c
     i:=i+1
fin tant que
```

4 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`
`Bxx`	Branchement		xx = condition
			Cf. table ci-dessous
`BL`	Branchement à un		adresse de retour
	sous-programme		dans r14=LR
`LDR`	“load”		
`STR`	“store”		

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`}.

La table suivante donne les codes de conditions arithmétiques `xx` pour l'instruction de branchement `Bxx`.

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	

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

Les fonctions d'affichages :

— `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.

Les fonctions de saisie clavier :

— `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`.