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

Examen UE INF241 : Introduction aux Architectures Logicielles et Matérielles

25 mai 2011
Durée 2 h
Documents, calculettes, téléphones portables non autorisés
Le barême est donné à titre indicatif
En annexe, un résumé des instructions du processeur ARM

1 Codage des structures de contrôle (8 points)

On considère la fonction suivante :

```
1: unsigned int mult(unsigned int a, b) {
2:    unsigned int n;
3:    n = b;
4:    if (n == 0)
5:       return  0 ;
6:    else {
7:       n = n-1;
8:       return ( a + mult(a, n) );
9:    }
9: }
```

Question 1.1 : *Les paramètres et le résultat de la fonction* `mult` *sont passés par la pile. La variable locale* `n` *est stockée dans la pile. Dessinez la structure de la pile lors d'une exécution de la fonction* `mult`*. Donnez un code* `ARM` *pour la fonction* `mult` *en vous aidant du patron suivant :*

```
mult:
@ sauvegarde de l'adresse de retour dans la pile
@ sauvegarde du frame pointer dans la pile
@ mise en place du nouveau frame pointer
@ reservation de place pour la variable locale
@ empiler les variables temporaires
@ corps de la fonction
@ mise en place de la valeur de retour
@ depiler les variables temporaires
@ liberer place allouee a la variable locale
@ restaurer l'ancien frame pointer
@ recuperer l'adresse de retour
@ retour
```

On suppose maintenant l'existence d'une procédure `dec` et d'une fonction `add` :
- `dec` décrémente de 1 la valeur de la variable dont l'adresse lui est donnée en paramètre.
- `plus` retourne le résultat de l'addition de ses deux paramètres donnés.

Question 1.2 : *Le paramètre de la procédure* `dec`*, les paramètres et le résultat de la fonction* `plus` *sont passés par la pile. Donnez les parties de code* `ARM` *à modifier pour remplacer la ligne 7 de l'algorithme donné pour la fonction* `mult` *par* `dec (adresse de n); (* en langage C : dec(&n) *)` *et la ligne 8 par*
`return ( plus(a, mult(a,n)) )`

2 Un processeur spécialisé pour JAVA (12 points)

La machine virtuelle JAVA (JVM : Java Virtual Machine) est un processeur "imaginaire" conçu en même temps que le langage de programmation JAVA.

Le langage machine de la JVM peut, selon les cas :

- être traduit vers le langage machine d'un processeur déjà existant (ARM, Pentium, par exemple)
- être interprété par un programme. Ce programme, l'interprète JAVA, lit une instruction de la JVM, l'analyse, met à jour les structures de données correspondantes (variables, tableaux, ...) puis passe à l'instruction suivante. Cet interprète, écrit au départ en langage évolué, est compilé et exécuté sur un ordinateur quelconque.
- être interpété par un circuit spécial organisé comme un processeur (PicoJava par exemple).

Nous nous intéressons dans ce sujet à la troisième solution. **Nous étudions ainsi un processeur spécialisé**.

Les instructions sont sur 1, 2, 3, 4 ou 5 octets. Le premier octet est le code opération. Nous n'étudions dans ce problème que quelques instructions.

Notre ordinateur est particulier : il a deux mémoires :

- une mémoire appelée `MMD` contenant des mots de 32 bits pour les données.
- une mémoire appelée `MMI` contenant des octets pour les instructions.

Dans les deux cas on utilise des adresses d'octets.

Le processeur a entre autres deux registres connus du programmeur :

- `PC` : le compteur de programme contient l'adresse dans `MMI` de début de l'instruction en cours d'exécution.
- `OPTOP` : le pointeur de pile contient une adresse dans `MMD`. Le pointeur de pile repère la dernière "case pleine" (dernière valeur empilée) et la pile progresse vers les adresses croissantes (à la différence des machines étudiées en TD).

Par ailleurs, le processeur a des registres R1, R2, ... pour tout stockage de résultats intermédiaires. Les registres peuvent contenir des valeurs sur 8, 16 ou 32 bits, on ne se préoccupe pas ici des questions d'extension sur les poids forts.

Pour lire et écrire en mémoire `MMD`, il y a deux registres particuliers `ADR` et `DON`. On dispose des micro-actions `DON <-- MMD[ADR]` et `MMD[ADR]<--DON`.

Les lectures dans la mémoire `MMI` se font toujours avec l'adresse dans `PC`, les résultats pouvant être rangés dans le registre instruction `RINST` ou dans tout autre registre.

En plus des lecture/écriture dans la mémoire `MMD` et lecture dans la mémoire `MMI`, les micro-actions possibles sont les suivantes :

- `regA` <– 0 : mise à zéro d'un registre
- `regA` <– `regB` : transfert d'un registre vers un autre
- `regA` <– `regA` – : décrémentation du contenu d'un registre
- `regA` <– `regA` ++ : incrémentation du contenu d'un registre
- `regA` <– `regA` + `regB` : addition des contenus de deux registres
- `regA` <– `regA` << k : décalage de k bits vers la gauche
- flag <– COMP (COND, `regA`, `regB`) : donne la valeur booléenne de la comparaison de deux registres selon une condition COND.

`regA` et `regB` peuvent être tout registre du processeur.

Aucune autre micro-action que celles listées n'est possible.

Nous nous intéressons à trois instructions :

- L'instruction `IADD` (Integer ADDition) est codée sur un octet. Son effet est de dépiler les deux entiers en sommet de pile et d'empiler leur somme.
- L'instruction `IALOAD` (Integer Array LOAD) est codée sur un octet. Son effet est de dépiler les deux entiers en sommet de pile. La valeur contenue dans le sommet de pile est considérée comme un indice dans un tableau d'entiers, soit `I` ; la valeur contenue dans le sous-sommet est considérée comme l'adresse (dans `MMD`) de début d'un tableau `T` d'entiers. Après avoir dépilé les deux entiers, l'effet de l'instruction `IALOAD` est d'empiler la valeur `T[I]`.
- L'instruction `IF_ICMP_COND` (`COND` peut être `eq, ne, lt, le, gt, ge`) est codée sur trois octets : le code opération suivi de deux octets `Oc1` et `Oc2`. Son effet est de dépiler les deux entiers en sommet de pile, puis ces deux entiers sont comparés ; si la comparaison donne un résultat conforme à la condition `COND`, l'entier signé (`Oc1` * 256 + `Oc2`) est ajouté à l'adresse de début de l'instruction de saut. Si la comparaison donne une condition fausse, le branchement n'a pas lieu et le processeur traite l'instruction

suivant l'instruction en cours d'exécution.

Questions :

2.1 Donnez la séquence de micro-actions nécessaires pour l'acquisition et l'exécution de l'instruction `IADD`*.*

2.2 Donnez la séquence de micro-actions nécessaires pour l'acquisition et l'exécution de l'instruction `IALOAD`*.*

2.3 Donnez la séquence de micro-actions nécessaires pour l'acquisition et l'exécution de l'instruction `IF_ICMP_COND`*.*

2.4 Fusionnez les trois séquences obtenues précédemment en un seul graphe de contrôle.

2.5 Supposons que le processeur ait une fréquence de 1Giga Hz ; calculez le temps d'exécution de l'instruction`IADD`*. Donnez le détail de vos calculs.*

3 ANNEXE : 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`
`Bcc`	Branchement		cc = condition Cf. table ci-dessous
`BL`	Branchement à un sous-programme		adresse de retour 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 `cc` pour l'instruction de branchement `Bcc`.

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	