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

Examen UE INF241 : Introduction aux Architectures Logicielles et Matérielles

Première session 2011-2012, 23 mai 2012
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 (10 points)

On considère les procédures et fonctions suivantes :

```
1:  procedure calcul (adx : adresse, ady : adresse) {
2:      x,y: entiers naturels
3:      x := mem[adx]
4:      y := mem[ady]
5:      si x >= y alors {
6:          mem[adx] := x - y
7:      } sinon {
8:          mem[ady] := y - x
9:      }
10: }

20:  fonction pgcd (a : entier naturel, b: entier naturel) : entier naturel {
21:     u,v, resultat : entiers naturels
22:     si a != b alors {
23:         u := a
24:         v := b
25:         calcul(adresse de u, adresse de v)
26:         resultat := pgcd(u,v)
27:     } sinon {
28:         resultat := a
29:     }
30:     retourner resultat
31: }

40: procedure_principale () {
41:     x,y : entiers naturels
42:     x := 72
43:     y := 32
44:     afficher pgcd(x,y)
45: }
```

Questions :

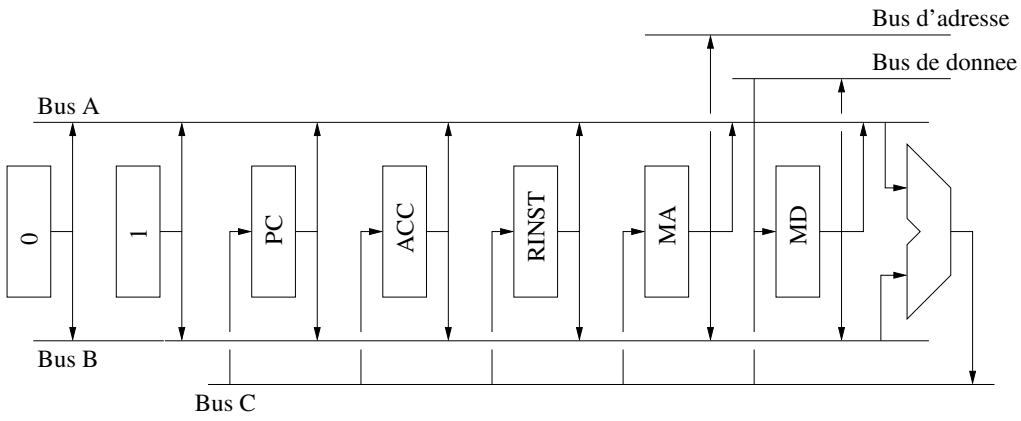
En considérant que le passage de paramètres et le retour des résultats se font exclusivement par la pile pour toutes les procédures et fonctions :

1. Donnez le code ARM de la procédure principale. **(1 point)**
2. Donnez l'état de la pile lorsque le premier appel à `pgcd` atteint la ligne 22, en précisant dans quelle fonction chaque donnée a été empilée. Indiquez également la position de `sp` et `fp`. **(2 points)**

3. Donnez le code ARM de l'appel de `calcul`, ligne 25. **(1 point)**
4. Donnez le code ARM de la fonction `calcul`. **(3 points)**
5. Donnez le code ARM de la fonction `pgcd`. **(3 points)**

2 Micro-programmation (10 points)

Dans cette partie, nous travaillons sur un processeur fictif dont la partie opérative est représentée sur la figure ci-dessous :



Ce processeur est proche du processeur jouet vu au cours 9. Il ne contient qu'**un seul registre de données directement visible par le programmeur** : `ACC` (pour accumulateur). Les instructions sont codées sur **1 ou plusieurs mots mémoire** : le premier mot représente le code de l'opération et les autres, s'ils existent, contiennent des adresses ou des constantes.

Nous pouvons constater que la partie opérative du processeur contient les registres `pc`, `acc`, `rinst`, `ma` (memory address), `md` (memory data) ainsi que les constantes 0 et 1. Les seules micro-actions permettant d'effectuer des transferts mémoire ou des opérations arithmétiques sont données ci-dessous :

md ← mem[ma]	lecture d'un mot mémoire.
mem[ma] ← md	écriture d'un mot mémoire.
reg ← reg1 op valeur	opérations arithmétiques : **op** peut être +,-,* ou /, la valeur peut être un second registre ou l'une des constantes.
reg ← valeur	copie : la valeur peut être un registre ou l'une des constantes.

Nous pouvons remarquer que tous les registres du processeur peuvent intervenir comme opérande ou résultat d'une opération arithmétique. Notre processeur est également capable d'effectuer des branchements vers des parties de son micro-programme identifiées par des étiquettes à l'aide des micro-actions suivantes :

jmp étiquette	branchement inconditionnel.	Effectue un branchement dans le micro-code
jmpc valeur, étiquette	branchement conditionnel.	Effectue un branchement si `rinst`=valeur

Les 3 types de transferts et les branchements constituent **le langage des micro-actions et des micro-conditions** de notre processeur.

Question 1 : Donnez le **microcode** ET **l'automate d'interprétation** (reset et *fetch* compris) des quatres instructions données ci-dessous dans le langage des micro-actions de notre processeur. Attention, comme précisé précédemment, chaque instruction est codée sur un mot mémoire, tout comme chacun de ses arguments. Ainsi, l'instruction `swap`, par exemple, sera codée sur 3 mots. **(7 points)**

`clr2` *adr*	Remise à zéro de la case mémoire située à l'adresse *adr*
`jmpSiZero` *adr*	Saut à l'adresse *adr* (absolue) si le contenu de l'accumulateur est nul
`add2` *adr*1, *val*, *adr*2	Calcule la somme des *val* éléments du tableau stockés à partir de l'adresse *adr*1 et stocke le résultat à l'adresse *adr*2
`swap` *adr*1, *adr*2	Echange les valeurs stockées aux adresses *adr*1 et *adr*2

Question 2 : On souhaite maintenant ajouter deux instructions pour gérer une pile en *full descending* (c'est la convention adoptée en cours !).

`empile`	Empile la valeur contenue dans le registre *acc*
`depile`	Met le sommet de pile dans le registre *acc*, puis supprime le sommet de pile

— Expliquez les changements nécessaires à opérer sur la structure du processeur pour pouvoir implanter ces deux instructions. **(1 point)**
— Donnez le **microcode** de ces deux nouvelles instructions de gestion de pile ET ajoutez les à l'**automate d'interprétation**. **(2 points)**

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	