L2 MMIA parcours MIAGE

Interrogation Corrigée d'architecture

Durée 1h30. Documents de cours et TD autorisés.

Exercice 1

Question 1: Commentez cycle par cycle ce qui se passe lors de l'exécution de la suite d'octets (située en mémoire d'instructions:

ILOAD

1

Iload1	H=LV	Copie de la base LV dans H. Après cette instruction MBR contient la valeur 1.
Iload2	MAR=MBR+H; rd	Ajout de 1 à la base pour trouver l'adresse de la variable, chargée dans MAR, lancement lecture
Iload3	MAR=SP=SP+1;	Incrémentation du pointeur de pile, positionnement du registre d'adresse. A la fin de ce cycle, la valeur de la variable 1 arrive dans MDR
Iload4	PC=PC+1; fetch; wr	Lancement de l'écriture (au nouveau sommet de pile), et de la recherche de la prochaine instruction après incrémentation compteur ordinal
Iload5	TOS=MDR; goto Main1	Stockage dans TOS de la valeur du sommet de pile, retour à la boucle principale.

Question 2: Même question avec la suite d'octets :

WIDE

ILOAD

1

0

Rq : la micro-instruction wide_iload 1 est située à 256 mots d'écart de la micro-instruction iload1.

Wide 1	PC=PC+1 ; fetch ; goto (MBR OR 0x1OO)	PC avance pour pointer sur l'octet ayant la valeur1. MBR contient le code ILOAD en fin de cycle. La prochaine micro-instruction est donc celle qui se trouve à 256 cases de la valeur de MBR ; donc ici wide_iload 1.
Wide_iload 1	PC=PC+1; fetch	PC pointe sur l'octet de valeur 0. En attendant son arrivée, MBR vaut 1.

Wide_iload2	H=MBR<<8	Les deux octets de poids faible de H valent 10 suite au décalage de l'octet 1. MBR en fin de cycle vaut 0
Wide_iload3	H=MBR or H	Reconstitution du numéro de la variable écrit sur deux octets dans H. Ici 10.
Wide_iload4	MAR=LV+H;rd; goto iload 3	MAR pointe sur l'adresse de la variable dont l'instruction a décrit le numéro sur 2 octets. On est ramené à la situation de l'instruction iload après que le calcul de cet adresse ait été fait, et la lecture de la variable soit à l'instruction iload2.
Iload3	MAR=SP=SP+1;	Incrémentation du pointeur de pile, positionnement du registre d'adresse. A la fin de ce cycle, la valeur de la variable 10 (en octets) arrive dans MDR
Iload4	PC=PC+1; fetch; wr	Lancement de l'écriture (au nouveau sommet de pile), et de la recherche de la prochaine instruction après incrémentation compteur ordinal
Iload5	TOS=MDR; goto Main1	Stockage dans TOS de la valeur du sommet de pile, retour à la boucle principale.

Exercice 2

On considère la fonction C suivante:

```
int sommen(int n)
int i, s=0;
for (i=1,i<=n,i++)
s=s+i
return(s)
```

Question 1 : Ecrire une fonction qui implémente la fonction sommen en langage IJVM. On supposera que lors de l'appel à cette fonction, le paramètre n est stocké à l'adresse LV+1, la variable s à l'adresse LV+2 et la variable i à l'adresse LV+3. Juste avant l'instruction de retour, la pile contiendra la valeur à retourner.

	CODE	NB d'octets	Nb de micro-instructions + main1 suivant	Appel avec paramètre de valeur 4
	BIPUSH 0	2	4	22 cycles avant la boucle
	ISTORE 2	2	7	
	BIPUSH 1	2	4	
	ISTORE 3	2	7	
Boucle	ILOAD 3	2	6	4 itérations *46 cycles= 184 cycles
	DUP	1	3	
	ILOAD 1	2	6	
	IFCMPEQ fin	3	7	
	ILOAD 2	2	6	
	IADD	1	4	
	IINC 3 1	3	7	
	GOTO boucle	3	7	
Fin	ILOAD 2	2	6	15 cycles après la boucle
	IRETURN	1	9	
	TOTAL	**=26**		**221 cycles**

(on pouvait supposer que le GOTO microprogrammé permet de gérer les offsets négatifs, ou utiliser l'instruction GOTOR vue en cours, car sinon, tel qu'il est écrit, le micro-programme ne permet pas de gérer les retours arrière)

Question 2: A l'aide du microprogramme, calculer le nombre de cycles d'horloge nécessaires pour réaliser un appel à sommen(4). Indiquez également le nombre d'octets nécessaires pour ranger les

instructions de sommen.

Question 3: Ecrire une implémentation des instructions de cette fonction dans le langage du jeu d'instruction dit « jeu projet ». On notera @n,@s et @i les adresses (constantes) des variables correspondantes.

	LDIR R1 @n
	SETI R0 0
	SETI R4 1
	SDIR R4 @s
	SDIR R3 @i
	LDIR R2 @s
Boucle	COMP R3 R1
	BPOS fin
	IADD R2 R2 R3
	SDIR R2 @s
	IADD R3 R3 R4
	JUMP boucle
fin	

Question 4: On considère maintenant l'implémentation récursive de la même fonction C:

```
int sommenrec(int n)
{if (n==0) return(0);
return(n+sommenrec(n-1));}
```

Ecrire une implémentation de cette fonction en IJVM (avec appel récursif). On supposera que la fonction est numérotée 5 dans la zone des constantes pointée par CPP.

En observant la séquence de micro-instructions associées à Invoquevirtual et à Ireturn, on constate que pour que la procédure récursive fonctionne, il est nécessaire qu'avant chaque appel récursif sur la pile se trouve non seulement la valeur du paramètre (n-1) mais également au dessous un mot (ici de valeur 0) qui sert ultérieurement à stocker le pointeur sur la case contenant l'adresse de retour.

Avec cette configuration, il est nécessaire d'avoir en début de zone d'instructions deux octets représentant le nombre de paramètres +1 (ici 2).

	Zone d'instructions	Nombre d'octets	Nb cycles (avec main1) par instructions	Nb cycles lors de l'appel à sommenrec(4)
	Nombre 2 sur 2 octets (nb paramètres +1)	2		
	Nombre 0 sur 2 octets (taille de la zone de données de sommenrec)	2		
	ILOAD 1	2	6	Ces instructions sont faites 5 fois: 18*4+20 (la dernière fois n=0) =92 cycles
	BIPUSH 0	2	4	
	IFEQ nzero	3	10 (si branchement) ou 8 sinon	
	ILOAD 1	2	6	Ces instructions sont faites 4 fois : 47*4 cycles= 188 cycles
	BIPUSH 1	2	4	
	ISUB	1	4	
	INVOQUEVIRTUAL 5	3	23	
	ILOAD 1	2	6	
	IADD	1	4	
nzero	IRETURN	1	9	Instructions faite 5 fois= 40 cycles
		TOTAL 23 octets en zone instructions		TOTAL 320 cycles

Etat de la pile juste avant l'invoquevirtual :

adresse	**valeur**
SP->	n-1
	0
	LV ancien
a	@PC retour
	n
LV->	a

Etat de la pile juste après invoquevirtual :

adresse	**valeur**
SP->	LV ancien (Base1)
b	@PC retour
	n-1
LV->	b
	LV ancien
a	@PC retour
	n
Base1	a

Etat de la pile juste avant ireturn :

adresse	**valeur**
SP->	**1+2+…+n-1**
	LV ancien (Base1)
b	@PC retour
	n-1
LV->	b
	LV ancien
a	@PC retour
	n
Base1	a

Etat de la pile juste après ireturn :

adresse	**valeur**
SP->	1+2+…n-1
	LV ancien
a	@PC retour
	n
LV->	a

Question 5: Décrire les états successifs de la mémoire lors de l'exécution de l'appel à sommenrec (4).

Le premier appel est celui avec paramètre 4 qui est déjà en cours. On présente dans les tableaux suivants les états de la pile d'exécution.

Avant le 2e invoquevirtual

adresse	valeur
SP->	3
	0
	LV ancien
a	@PC retour
	4
LV->	a

Avant le 3e invoquevirtual

adresse	valeur
SP->	**2**
	0
	LV ancien (Base1)
b1	@PC retour
	3
LV->	b1
	LV ancien
a	@PC retour
	4
Base1	a

Avant le 4e

adresse	valeur
SP->	**1**
	0
	LV ancien (Base2)
b2	@PC retour
	2
LV->	**b2**
	LV ancien (Base1)
b1	@PC retour
	3
Base2	b1
	LV ancien
a	@PC retour
	4
Base1	a

Avant le 5e et dernier

adresse	valeur
SP->	**0**
	0
	LV ancien (Base3)
b3	@PC retour
	1
LV->	**b3**
	LV ancien (Base2)
b2	@PC retour
	2
Base 3	**b2**
	LV ancien (Base1)
b1	@PC retour
	3
Base 2	b1
	LV ancien
a	@PC retour
	4
Base1	a

Avant Ireturn :

adresse	valeur
SP->	**0**
	LV ancien (Base4)
b4	@PC retour
	0
LV->	**b4**
	LV ancien (Base3)
b3	@PC retour
	1
Base 4	**b3**
	LV ancien (Base2)
b2	@PC retour
	2
Base 3	**b2**
	LV ancien (Base1)
b1	@PC retour
	3
Base 2	b1
	LV ancien
a	@PC retour
	4
Base1	a

Après le premier Ireturn :

adresse	**valeur**
SP->	**0**
	LV ancien (Base3)
b3	@PC retour
	1
LV=Base 4	**b3**
	LV ancien (Base2)
b2	@PC retour
	2
Base 3	**b2**
	LV ancien (Base1)
b1	@PC retour
	3
Base 2	b1
	LV ancien
a	@PC retour
	4
Base1	a

Après le 2e Ireturn :

adresse	**valeur**
SP=Base 4	**1**
	LV ancien (Base2)
b2	@PC retour
	2
LV=**Base 3**	**b2**
	LV ancien (Base1)
b1	@PC retour
	3
Base 2	b1
	LV ancien
a	@PC retour
	4
Base1	a

Après le 3e Ireturn :

adresse	**valeur**
SP=**Base 3**	**3**
	LV ancien (Base1)
b1	@PC retour
	3
LV=Base 2	b1
	LV ancien
a	@PC retour
	4
Base1	a

Après le 4e Ireturn :

adresse	**valeur**
SP=Base 2	6
	LV ancien
a	@PC retour
	4
LV=Base1	a

Après le 5e Ireturn :

adresse	**valeur**
SP=Base 1	10

Question 6 : A l'aide du micro-programme calculer le nombre de cycles d'horloge nécessaires pour réaliser un appel à sommenrec(4). Indiquez également le nombre d'octets nécessaires pour ranger les instructions de sommenrec.

Cf tableau de la question 4

Question 7: Quelle est l'implémentation la plus efficace?

L'emploi de la récursivité a un surcoût au profit de la simplicité et de la taille du code très souvent. Ici, la différence en temps de calcul est tout de même substantielle.

Exercice 3

Que fait la suite de micro instructions suivantes?:

myst1	MAR=SP=SP+1 goto myst2
myst2	H=MDR=TOS if (N) goto myst3bis else goto myst3
myst3	wr, goto main1
myst3bis	TOS=MDR=-H;goto myst3

Cette suite de micro-instructions calcule la valeur absolue du mot au sommet de la pile et l'empile.

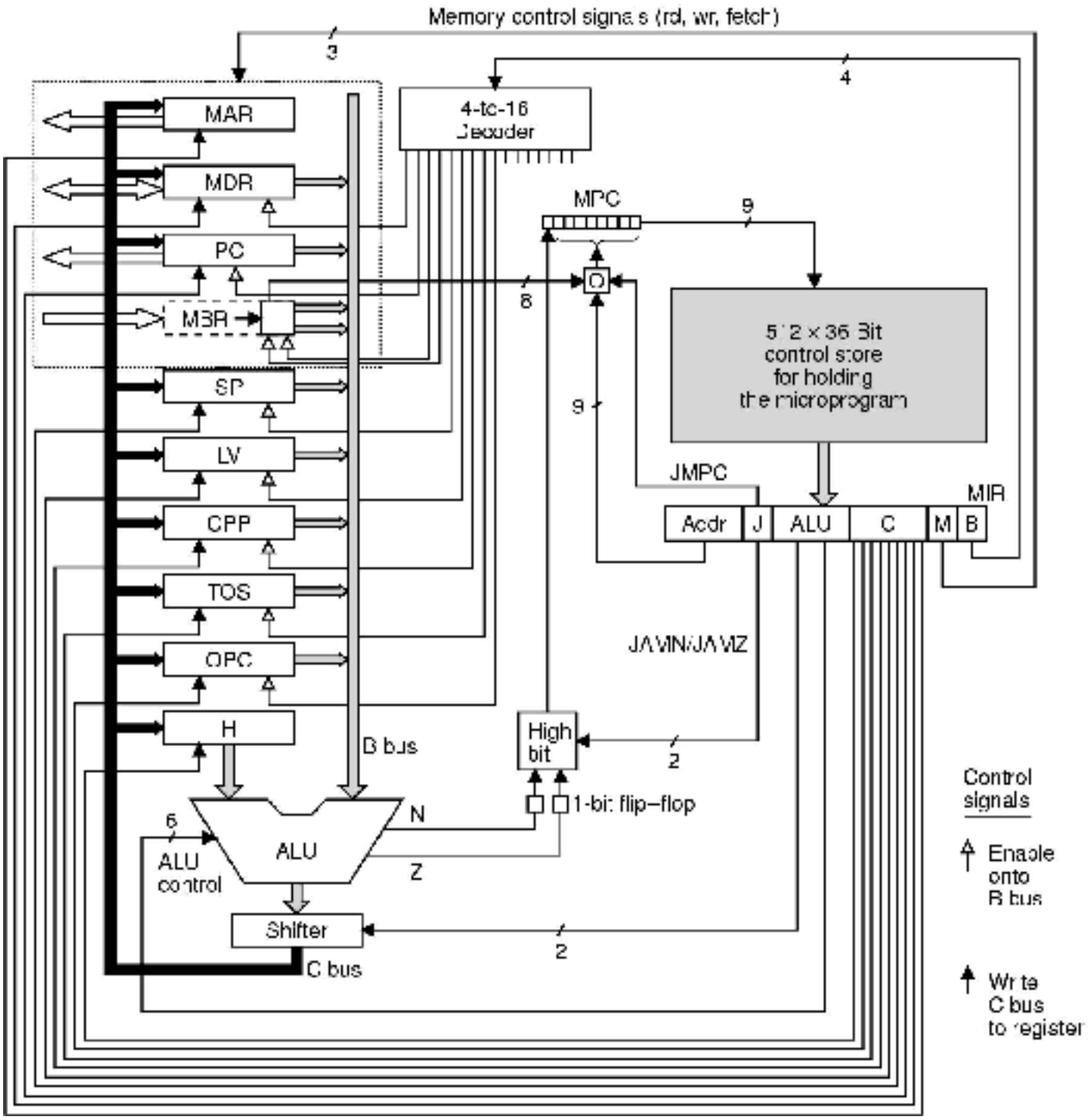
ANNEXE

Jeu IJVM

BIPUSH octet	push octet dans la pile	0x010
DUP	Duplique le mot du sommet et l'empile	0x59
GOTO offset	Branchement inconditionnel	0xA7
IADD	pop de deux mots, push de leur somme	0x60
IAND	pop de deux mots, push de leur ET	0x7E
IFEQ offset	pop un mot de la pile, branchement s'il vaut 0	0x99
IFLT offset	pop un mot de la pile branchement s'il est <0	0x9B
IF_ICMPEQ offset	pop deux mots, branchement si égaux	0x9F
IINC numvar const	additionne constante à variable locale	0x84
ILOAD numvar	push une variable locale dans la pile	0x15
INVOKEVIRTUAL dep	invoque une méthode, dep est un index/ bloc de consantes	0xB6
IOR	pop de deux mots dans la pile, push de leur OR	0x80
IRETURN	retour de méthode avec valeur entière	0xAC
ISTORE numvar	pop mot de la pile et range dans variable locale	0x36
ISUB	pop deux mots, push différence	0x64
LDC_W index	push constante de zone constante	0x13
NOP	rien	0x00
POP	efface sommet pile	0x57
SWAP	permute deux mots du sommet de pile	0x5F
WIDE	préfixe d'instruction: l'instruction a un paramètre numvar ou const sur 16 bits (ordinairement sur 8)	0xC4

Jeu Projet

syntaxe	**effet**	**commentaire**
LDIR Rd Imm	Rd←Mem[Imm]	Chargement d'un mot mémoire dans un registre par adressage direct
SDIR Rd Imm	Mem[Imm] ← Rd	Rangement d'un mot mémoire dans un registre par adressage direct
IADD Rd Ra Rb	Rd←Ra+Rb	Addition entière. Registre condition positionné
ISUB Rd Ra Rb	Rd←Ra-Rb	Soustraction entière. Registre condition positionné
COMP Ra Rb	Ra-Rb	Comparaison, RCC positionné
IMUL Rd Ra Rb	Rd←Ra*Rb	Multiplication de deux entiers, RCC positionné
Bccc dep	Si ccc PC←PC+ ES(dep)	Branchement conditionnel les ccc sont : EQU, NEG, POS, NEN, PON pour =0, <0, >0, <=0, >=0
JUMP dep	PC←PC+ ES(dep)	Branchement inconditionnel
NOOP		Ne fait rien
SETI Rd Imm	Rd←ES(Imm)	Affecte une valeur immédiate entière à un registre.


Memory control signals (rd, wr, fetch)
3
4
4-to-16 Decoder
MAR
MDR
PC
MBR
SP
LV
CPP
TOS
OPC
H
MPC
9
8
O
512 × 36 Bit control store for holding the microprogram
9
JMPC
MIR
Addr
J
ALU
C
M
B
JAVIN/JAVIZ
High bit
2
1-bit flip-flop
B bus
N
Z
6
ALU control
ALU
Shifter
2
C bus
Control signals
Enable onto B bus
Write C bus to register