ARCHITECTURE des ORDINATEURS : EXERCICES

Exercices sur la partie langage d'assemblage

Exercice 1 : Codage des données et organisation en mémoire

Sachant que les entiers et les flottants sont codés sur 32 bits, étudier l'implantation en mémoire de la structure suivante à partir de l'adresse, dont les contenus en Hexadécimal sont spécifiés, pour le cas gros-boutiste et petit-boutiste :
Struct {
Int a ; /* vaut 11121314
Double b ; /* vaut 2122232425262728
Char ***c ;/** vaut 31323334
Char d[7] ; /* vaut 'A','B','C','D','E','F','G'
Short e ; /* vaut 5152
Int f : /* vaut 61626364
} data ;

Exercice 2 : Modes d'adressage

Etant donnés les valeurs des mots mémoire ci-dessous et une machine à une adresse disposant d'un registre spécialisé dans le transfert de mots mémoire, quelles sont les valeurs chargées dans ce registre après l'exécution des instructions suivantes ?
Le mot 20 contient la valeur 40
Le mot 30 contient la valeur 50
Le mot 40 contient la valeur 60
Le mot 50 contient la valeur 70

a)LOAD IMMEDIAT 20 b)LOAD INDIRECT 20 c)LOAD DIRECT 30 d)LOAD DIRECT 20
e)LOAD IMMEDIAT 30 f)LOAD INDIRECT 30.

Exercice 3 : Codage expansif

Spécifiez si possible un code opération expansif qui permette de coder toutes les instructions dans les deux configuration suivantes :
1) Instructions sur 36 bits :
7 instructions avec deux champs adresse de 15 bits et un champ registre de 3 bits
500 instructions avec un champ adresse de 15 bits et un champ registre de 3 bits
50 instructions sans champ adresse ni registre
2) Instructions sur 12 bits :
4 instructions avec 3 champs registres sur 3 bits
255 instructions avec un champ registres
16 instructions sans registre.

Exercice 4 : comparaisons de machines types

Considérons les trois machines suivantes :
M0 est une machine à pile : elle peut ranger un mot mémoire dans une pile (PUSH X), ranger le sommet de la pile en mémoire et dépiler (POP X) , et les opérations arithmétiques consistent à enlever deux mots de la pile faire l'opération et mettre le résultat dans la pile (ADD et SUB ici)

M1 est une machine à accumulateur : elle peut transférer un mot mémoire dans l'accumulateur (LOAD X) et réciproquement (STORE X). Elle peut additionner un mot mémoire avec le contenu de l'accumulateur et ranger le résultat dans l'accumulateur (ADD X) et de même retirer un mot mémoire de l'accu (SUB X).

M2 est une machine à chargement-rangement, avec plusieurs registres généraux. Les accés de et vers la mémoire se font uniquement via des registres (LOAD Ri,X ou STORE Ri, X). Les opérations arithmétiques se font

uniquement de registre à registre (ADD Ri,Rj,Rk ou SUB Ri,Rj,Rk avec Ri destination, et dans le second cas différence Rj-Rk.
On considère que les variables A, B,C ,D sont en mémoire. Ecrire et comparer en taille de code et nombre d'accès mémoire le code pour les trois machines correspondant à la suite d'instructions :
A=B+C ;
B=A+C ;
D=A-B ;
On supposera que le compilateur (vous) tente d'optimiser le nombre d'accès mémoire.

Exercice 5 : Compilation de structures de contrôle.

On utilise dans cet exercice les jeux d'instructions MEXS, et IJVM
MEXS possède une mémoire gros-boutiste organisée en octets, 32 registres de 32 bits avec R0 câblé à 0.

Question 1 : Observer le jeu d'instructions MEXS. Quel est le nombre maximal de bits d'un immédiat dans une instruction arithmétique ? Comment peut-on réaliser le chargement de 0100 0000 (en Hexadécimal) dans le registre R1 ?
Décrire d'une manière générale une suite d'instructions permettant de charger un immédiat sur 32 bits dans un registre.

Question 2 : Ecrire la séquence d'instructions assembleur correspondant à :
Int i,j
i=max(i,j)
(supposées indexées 1 et 2 respectivement dans les variables pour l'IJVM, rangées aux adresses @i et @j pour MEXS)
Question 3 : Soit la boucle :
Pour i=1, 256
S=s+2*i
Ecrire un programme assembleur correspondant pour MEXS et IJVM

Question 4 : on considère la boucle :
Pour i=0,999
S=s+X(i) où X est un vecteur d'entiers. Pourquoi ne peut- on écrire un programme assembleur en IJVM correspondant à cette boucle ? Ecrire le programme assembleur MEXS correspondant, sachant que le vecteur X est implanté à partir de l' adresse 10000000 et la variable s est placée à l'adresse 100 et la variable i à l'adresse 104
Question 5 : Ecrire les programmes assembleurs MEXS pour les boucles suivantes , sachant que les vecteurs X et Y sont implantés à partir des adresses 10000000 et 20000000 hexadécimal, s et i comme dans la question précédente .

Pour i=0,999	pour i=1, 999
S=s+X(i)+Y(i)	Y(i-1)=X(i)+X(i-1)

Exercice 6 : Multiplication

Question 1: Proposer une procédure IJVM permettant de réaliser une multiplication des deux mots au sommet de la pile.
Question 2: En supposant que le langage IJVM comporte en outre une instruction permettant de décaler d'un bit sur la gauche le sommet de la pile (SL) et d'un bit sur la droite avec recopie du bit le plus à gauche (SR), comment les utiliser pour réaliser cette même multiplication?

Exercice 7 : Factorielle

Proposer un programme IJVM, pouvant comporter plusieurs procédures, permettant de calculer de manière récursive la factorielle d'un entier. On simulera l'état de la pile pour un appel à factorielle 3.

10.1 Jeu d'instructions de MEXS

Mnémonique	Syntaxe	Description
ADD	ADD Rd, Ra, Rb	Rd <-Ra + Rb
ADDcc	ADDcc Rd, Ra, Rb	Rd <-Ra + Rb RCC affecté
ADDX	ADDX Rd, Ra, Rb	Rd <-Ra + Rb + CF
ADDXcc	ADDXcc Rd, Ra, Rb	Rd <-Ra + Rb + CF RCC affecté
SUB	SUB Rd, Ra, Rb	Rd <-Ra - Rb
SUBcc, SUBX, SUBXcc	-	-
AND	AND Rd, Ra, Rb	Rd <-Ra AND Rb
ANCcc	ANDcc DD Rd, Ra, Rb	Rd <-Ra AND Rb RCC affecté
OR	OR Rd, Ra, Rb	Rd <-Ra OR Rb
ORcc	ORcc Rd, Ra, Rb	Rd <-Ra OR Rb RCC affecté
XOR	XOR Rd, Ra, Rb	Rd <-Ra XOR Rb
XORcc	XORcc Rd, Ra, Rb	Rd <-Ra XOR Rb RCC affecté
LDB	LDB Rd, (Ra + Rb)	$Rd_{0:7}$ <- Mem8 (Ra + Rb) $Rd_{8:31}$ <- 0
LDSB	LDSB Rd, (Ra + Rb)	$Rd_{0:7}$ <- Mem8 (Ra + Rb) $Rd_{8:31}$ <- ES($Rd_{0:7}$)
LDH	LDH Rd, (Ra + Rb)	$Rd_{0:15}$ <- Mem16 (Ra + Rb) $Rd_{16:31}$ <- 0
LDSH	LDSH Rd, (Ra + Rb)	$Rd_{0:15}$ <- Mem16 (Ra + Rb) $Rd_{16:31}$ <- ES($Rd_{0:15}$)
LD	LD Rd, (Ra + Rb)	Rd <- Mem32 (Ra + Rb)
STB	STB (Ra + Rb), Rd	$Rd_{0:7}$ -> Mem8 (Ra + Rb)
STH	STH (Ra + Rb), Rd	$Rd_{0:15}$ -> Mem16 (Ra + Rb)
ST	ST (Ra + Rb), Rd	Rd -> Mem32 (Ra + Rb)
ADDI (1)	ADDI Rd, Ra, imm_{16}	Rd <- Ra + ES (imm_{16})
SLL	SLL Rd, imm_5, Ra	Rd <- Ra décalé à gauche de imm_5
SRL	SRL Rd, imm_5, Ra	Rd <- Ra décalé à droite de imm_5 avec remplissage à 0
SRA	SRA Rd, imm_5, Ra	Rd <- Ra décalé à droite de imm_5 avec extension de signe
Bxx(voir tableau suivant)	Bxx imm_{20} ou Bxx etiq	si condxx, PC <- PC + ES(imm_{20}) $\times$ 4
JMPL	JMPL Rd, Ra, Rb	Rd <- PC PC <- Ra + Rb
JMPLI	JMPLI Rd, Ra, imm_{16}	Rd <- PC PC <- Ra + ES (imm_{16}) $\times$ 4
SETHI	SETHI Rd, imm_{22}	$Rd_{10:31}$ <- imm_{22} $Rd_{0:9}$ <- 0

Notes et exercices sur la partie micro-architecture

Langage de description des micro-instructions:

Les actions réalisées simultanément dans un cycle sont séparées par des points-virgule.

Signaux indiqués (rd,wr, fetch)

Instruction suivante: si c'est celle écrite juste en dessous on n'indique rien sinon goto etiquette.sauts conditionnels (par rapport aux bits N et Z): If N goto x else goto y

Transferts de registre à registre par signe d'affectation: exemple MAR=SP SP transféré dans le registre MAR.

Opérations arithmétiques (commande de l'ALU): exemple PC=PC+1; PC est mis sur le bus B, commande de l'incrémentation de l'ALU et PC en registre de sortie sur le bus C.

Décalage: exemple H=MBRU<<8 : commande de l'ALU pour transfert et décalage d'un octet vers la gauche.

Affectations et opérations en chaîne: exemple MDR=TOS=MDR OR H décrit le fait que MDR est mis sur le bus B, un ou logique est réalisé dans l'ALU en autorisant l'entrée de H, et sur le bus C TOS et MDR sont indiqués pour récupérer le résultat.

Format des instructions IJVM, placement des micro-instructions dans la mémoire de commande.

Les différents formats d'instructions ISA:

1 octet:
DUP, IADD,IAND, IOR,IRETURN, ISUB,NOP, POP, SWAP

2 octets:

(1 octet code opération, 1 octet constante ou numéro de variable)
BIPUSH, ILOAD,ISTORE

3 octets:

(1 octet code opération, 2 octets pour un opérande)
GOTO, IFEQ, IFLT,IF_ICMPEQ, INVOKEVIRTUAL, LDC-W

(1 octet code opération, 2 opérandes d'un octet)
IINC

4 octets:

(1 octet préfixe WIDE, 1 octet code opération, 2 octets opérande)
WIDE ILOAD, WIDE ISTORE

Placement des micro-instructions

Chaque instruction IJVM, y compris le préfixe WIDE, est une adresse dans la mémoire de commande, correspondant à celle de la première micro-instruction associée.

Les autres micro-instructions sont placées dans les trous restant dans la mémoire de commande(pas nécessairement consécutivement).
La contrainte sur le placement est la réalisation des sauts éventuels de séquence (avec le champ JAM). Lorsque l'instruction suivante est conditionnée par un bit de sortie, il faut que les deux branches se trouvent à 256 mots l'un de l'autre

Le micro-programme de MIC1

Etiquette	Actions	Commentaire
Main1	PC=PC+1;fetch, goto (MBR)	MBR contient le code opération (ISA), fetch octet suivant ; branchement
Nop1	Goto Main1	
Iadd1	MAR=SP=SP-1; rd	Décrémentation du pointeur de sommet de pile, et tranfert dans MAR lancement de la lecture de ce mot en mémoire
Iadd2	H=TOS	TOS contient l'ancien mot au sommet de la pile, transféré dans H ici (l'autre mot est arrivé dans MBR)
Iadd3	MDR=TOS=MDR+H; wr, goto Main1	Additionne et range le résultat dans MDR et dans TOS, lance l'écriture à l'adresse contenue dans MAR (sommet de pile), retour à la boucle principale.
Isub1	MAR=SP=SP-1,rd	
Isub2	H=TOS	
Isub 3	MDR=TOX=MDR-H; wr: goto Main 1	
Iand1	MAR=SP=SP-1,rd	
Iand2	H=TOS	
Iand3	MDR=TOX=MDR AND H; wr: goto Main 1	
Ior1	MAR=SP=SP-1,rd	
Ior2	H=TOS	
Ior3	MDR=TOX=MDR OR H; wr: goto Main 1	
Dup1	MAR=SP=SP+1	
Dup2	MDR=TOS;wr;goto Main1	
Pop1	MAR=SP=SP-1; rd	
Pop2		Pourquoi une instruction vide?

Pop3	TOS=MDR; goto Main1	
Swap1	MAR=SP-1; rd	
Swap2	MAR=SP	
Swap3	H=MDR; wr	
Swap4	MDR=TOS	
Swap5	MAR=SP-1; wr	
Swap6	TOS=H; goto Main1	
Bipush1	SP=MAR=SP+1	
Bipush2	PC=PC+1;fetch	
Bipush3	MDR=TOS=MBR; wr; goto Main1	
Iload1	H=LV	MBR contient alors l'index de la variable. Copie de la base LV dans H
Iload2	MAR=MBRU+H; rd	Ajout de l'index (bits de poids faible) 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,
Iload4	PC=PC+1; fetch; wr	Lancement de l'écriture, et de la recherche de la prochaine instruction après incrémentation compteur ordinal
Iload5	TOS=MDR; goto Main1	MDR contient toujours le mot du sommet de la pile (rangé le cycle précédent). Stockage dans TOS, retour à la boucle principale.
Istore1	H=LV	
Istore2	MAR=MBRU+H	
Istore3	MDR=TOS; wr	
Istore4	SP=MAR=SP-1;rd	
Istore5	PC=PC+1;fetch	
Istore6	TOS=MDR; goto Main1	

Wide1	PC=PC+1;fetch; goto (MBR OR 0x100)	
Wide_ iload1	PC=PC+1;fetch	
Wide_ iload2	H=MBRU<<8	
Wide_ iload3	H=MBRUor H	
Wide_ iload4	MAR=LV+H;rd;goto iload3	
Wide_ istore1	PC=PC+1;fetch	
Wide_ istore2	H=MBRU<<8	
Wide_ istore3	H=MBRUor H	
Wide_ istore4	MAR=LV+H;rd;goto istore3	
Ldc_w1	PC=PC+1;fetch	
Ldc_w2	H=MBRU<<8	
Ldc_w3	H=MBRUor H	
Ldc_w4	MAR= H+CPP;rd;goto iload3	
Iinc1	H=LV	
Iinc2	MAR=MBRU+H;rd	
Iinc3	PC=PC+1;fetch	
Iinc4	H=MDR	
Iinc5	PC=PC+1;fetch	
Iinc6	MDR=MBR+H;wr;goto Main1	
Goto1	OPC=PC-1	
Goto2	PC=PC+1 ; fetch	

Goto3	H=MBR<<8	
Goto4	H=MBRU OR H	
Goto5	PC=OPC+H ; fetch	
Goto6	Goto Main1	
Iflt1	MAR=SP=SP-1 ; rd	
Iflt2	OPC=TOS	
Iflt3	TOS=MDR	
Iflt4	N=OPC ; if(N) goto T else goto F	
Ifeq1	MAR=SP=SP-1 ; rd	
Ifeq2	OPC=TOS	
Ifeq3	TOS=MDR	
Ifeq4	N=OPC ; if(Z) goto T else goto F	
Ificmpeq1	MAR=SP=SP-1 ; rd	
Ificmpeq2	MAR=SP=SP-1	
Ificmpeq3	H=MDR ;rd	
Ificmpeq4	OPC=TOS	
Ificmpeq5	TOS=MDR	
Ificmpeq6	Z=OPC-H ; if (Z) goto T else goto F	
T	OPC=OPC-1 ;fetch :goto goto2	
F	PC=PC+1	
F1	PC=PC+1 ; fetch	

F2	Goto Main1	
Invoke1	PC=PC+1 ;fetch	
Invoke2	H=MBRU<<8	
Invoke3	H=MBRU OR H	
Invoke4	MAR=CPP+H ;rd	
Invoke5	OPC=PC+1	
Invoke6	PC=MDR ;fetch	
Invoke7	PC=PC+1 ; fetch	
Invoke8	H=MBRU<<8	
Invoke9	H=MBRU OR H	
Invoke10	PC=PC+1 ;fetch	
Invoke11	TOS=SP-H	
Invoke12	TOS=MAR=TOS+1	
Invoke13	PC=PC+1 ;fetch	
Invoke14	H=MBRU<<8	
Invoke15	H= MBRU OR H	
Invoke16	MDR=SP+H+1 ;wr	
Invoke17	MAR=SP=MDR	
Invoke18	MDR=OPC ;wr	
Invoke19	MAR=SP=SP+1	
Invoke20	MDR=LV ;wr	

Invoke21	PC=PC+1 ; fetch	
Invoke22	LV=TOS ; goto Main1	
Ireturn1	MAR=SP=LV ;rd	
Ireturn2		
Ireturn3	LV=MAR=MDR ; rd	
Ireturn4	MAR=LV+1	
Ireturn5	PC=MDR ; rd ; fetch	
Ireturn6	MAR=SP	
Ireturn7	LV=MDR	
Ireturn8	MDR=TOS ;wr ; goto Main1	

Exercices sur la partie micro-architecture

Exercice 1: Observer les commentaires des micro-instructions Iaddx et Iloadx. Compléter le tableau pour les autres micro-instructions.

Exercice 2: Proposer un placement en mémoire de commande pour l'ensemble des micro-instructions (compte-tenu des code opération fournis pour l'IJVM)

Exercice 3: Dans le format d'une micro-instruction, les registres du bus B ont codés sur 4 bits alors que le champs correspondant au bus C fait 9 bits. Pourquoi n'a-t-on pas utilisé un décodeur comme sur le bus B?

Exercice 4: Dérouler pas à pas l'exécution de la méthode factorielle écrite précédement en IJVM.

Exercice 5: proposer une implémentation dans le micro-programme pour de nouvelles instructions de la couche ISA:
1)Il s'agit d'une instruction ISHR permettant de décaler à droite une valeur. Elle utilise les deux valeurs du haut de la pile. A l'issue du traitement, une seule valeur, le résultat, est rangée dans la pile (à la place des deux opérandes). La seconde valeur du haut de la pile est l'opérande sur lequel porte le décalage. Il subit un décalage à droite d'un nombre de bits compris entre 0 et 31. Le nombre de bits est écrit dans les 5 bits de poids faible du mot qui est au sommet de la pile. Le bit de signe est dupliqué sur la droite d'un nombre de bits correspondant aux bits décalés. Le code opération de ISHR est 0x07A.
2) De même on considère une instruction ISHL permettant de décaler la deuxième valeur du haut de la pile d'un nombre de bits correspondant aux 5 bits de poids faible du sommet de la pile. Comme pour la précédente instruction, à la fin, les deux mots du sommet sont remplacés par le résultat de l'opération. Des zéros remplacent les bits décalés.
3) Ajouter ce qu'il faut au micro-programme pour implémenter l'instruction POPTWO, qui efface les deux mots du sommet de la pile.

Exercice 6: Proposer un mécanisme (instruction couche ISA et micro-instructions correspondantes) permettant de gérer des tableaux d'entiers dans les variables d'un programme IJVM.