HE2B ESI
Haute École de Bruxelles-Brabant
École Supérieure d'Informatique
Bachelor en Informatique

BEJ - HAL - MBA
24 mai 2018

Microprocesseurs et Systèmes d'exploitation - *Q2*

Examen de première session

Nom:	Prénom:	Groupe:

Directives

- L'examen est composé de `deux` parties.
- Répondez sur des feuilles séparées ; un papier ministre pour chaque partie de l'examen.
- Il est conseillé de lire l'entièreté de l'énoncé d'une question avant d'y répondre.
- Indiquez clairement sur CHAQUE FEUILLE que vous utilisez, votre nom, prénom, votre groupe, et l'intitulé de l'examen.
- N'oubliez pas d'entourer l'acronyme de votre professeur de Microprocesseurs.
- Vous rendrez toutes les feuilles utilisées.
- Vous devez utiliser un stylo à bille noir ou bleu ou un porte-plume pour vos réponses. (Pas de crayon, ni de rouge.)
- Vous pouvez répondre aux questions dans l'ordre de votre choix. Veuillez à bien identifier la question en début de réponse.
- Vous ne pouvez vous aider d'aucune note de cours pour répondre à cet examen.
- L'utilisation de la calculatrice est strictement interdite.
- Vous disposez de 1h pour chacune des parties de l'examen, soit une durée totale de `2h00`.

PARTIE 1 : Systèmes d'exploitation (20 pts)

Q. 1 *(6 pts)*
Soit un système de fichiers `EXT`. Soit l'appel système `read` qui transfère `n` bytes en RAM depuis un fichier.

a) À quoi correspond le premier paramètre de l'appel système read et comment obtient-on sa valeur ?

b) Quelles informations (métadonnées du système et du fichier) sont nécessaires à l'appel système `read` pour localiser les bytes à transférer depuis le disque ?

c) Où se trouvent ces informations (métadonnées) ?

Q. 2 *(8 pts)*
Soit un système de fichiers `ext` utilisant des blocs de `1Kib`. Ce dernier contient deux répertoires `/premier` et `/premier/deuxieme` ainsi qu'un fichier `/premier/deuxieme/monfichier` de taille 4000 bytes.

a) Dessinez de manière détaillée le contenu de ce système de fichiers après l'éxécution des commandes suivantes :
```
cd /premier
ln -s deuxieme/monfichier Lien
```

b) Quelle taille a le fichier `Lien` ?

c) Quels changements surviennent dans votre représentation si on exécute maintenant la commande :
`rm monfichier`

Q. 3 *(2 pts)*
Dans un système de fichiers en allocation contiguë avec des secteurs de 2048 bytes. Localizez le byte 8500 d'un fichier qui démarre au secteur 10.

Q. 4 *(4 pts)*

a) Décrivez la technique de détection-reprise dans le cadre de la problématique des interblocages.

b) Comment détecter ?

c) Comment reprendre ?

d) Quel comportement a le système d'exploitation Linux vis à vis de cette problématique.

e) Justifiez le choix de linux.

PARTIE 2 : Microprocesseurs (20 pts)

Q. 5 *(10 pts)* `Interruptions.`
Supposez que l'on travaille sur un système à microprocesseurs 64 bits de la famille `x86`.

a) Qu'est-ce qu'une interruption ?

b) Comment est-elle signalée, identifiée avant d'être prise en charge par le microprocesseur ?

c) Donnez et expliquez les différentes actions à réaliser par le processeur avant le traitement d'une interruption. Quelles sont les différentes actions à réaliser par le processeur lorsqu'il termine le traitement d'une interruption ?

d) Quel est le rôle de la table des interruptions ? Quelle est sa localisation en mémoire ainsi que sa structure en mode réel et en mode protégé ?

e) Exécuter une interruption revient à exécuter son code. Où se trouve le code de l'interruption `n°i` en mode réel et en mode protégé ?

Q. 6 *(4 pts)* `Démarrage de l'ordinateur.`
Supposez que l'on travaille sur un système à microprocesseurs 64 bits de la famille `x86`.

a) Qu'est-ce que le BIOS ? Quelle est sa localisation en mémoire ?

b) Quelles sont les différentes actions effectuées par le BIOS dès son démarrage ?

Q. 7 *(6 pts)* `Codage des instructions.`
Supposez que l'on travaille sur un système à microprocesseurs 64 bits de la famille `x86`.
En vous aidant de la documentation fournie en annexe et en expliquant toute votre démarche, traduisez en langage machine (binaire et hexadécimal) l'instruction assembleur suivante :

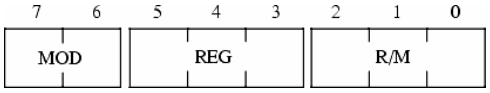
```
add r9, [rsi + 1025]
```

Documentation

Les codes binaires des différents registres

AL, AX, EAX, RAX, R8L, R8W, R8D, R8	000	AH, SP, ESP, RSP, R12L, R12W, R12D, R12	100
CL, CX, ECX, RCX, R9L, R9W, R9D, R9	001	CH, BP, EBP, RBP, R13L, R13W, R13D, R13	101
DL, DX, EDX, RDX, R10L, R10W, R10D, R10	010	DH, SI, ESI, RSI, R14L, R14W, R14D, R14	110
BL, BX, EBX, RBX, R11L, R11W, R11D, R11	011	BH, DI, EDI, RDI, R15L, R15W, R15D, R15	111

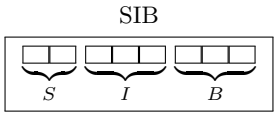
La structure du byte ModR/M



Adressage	Exemple	ModR/M
registre	RAX	1 1 _ _ _ _ _ _
indirect	[RAX]	0 0 _ _ _ _ _ _
indirect + court	[RAX+10]	0 1 _ _ _ _ _ _
indirect + long	[RAX+800]	1 0 _ _ _ _ _ _
direct	[adresse]	0 0 _ _ _ 1 0 1
indirect indexé	[RAX+4*RBX]	0 0 _ _ _ 1 0 0
indexé + court	[RAX+4*RBX+10]	0 1 _ _ _ 1 0 0
indexé + long	[RAX+4*RBX+800]	1 0 _ _ _ 1 0 0

La structure du byte SIB

- Utilisé en complément à ModR/M
- Pour les modes d'adressages indexés
  $(B + S \times I)$



S : facteur multiplicatif (scale)
$2^i \Rightarrow 00 = 1\times, 01 = 2\times, 10 = 4\times, 11 = 8\times$

I : registre d'index

B : registre de base

La structure du préfixe REX

0	1	0	0	W	R	X	B

Quelques opcodes

ADD — Add

Opcode	Instruction	Op/En	64-bit Mode	Compat/Leg Mode	Description
04 *ib*	ADD AL, *imm8*	I	Valid	Valid	Add *imm8* to AL.
05 *iw*	ADD AX, *imm16*	I	Valid	Valid	Add *imm16* to AX.
05 *id*	ADD EAX, *imm32*	I	Valid	Valid	Add *imm32* to EAX.
REX.W + 05 *id*	ADD RAX, *imm32*	I	Valid	N.E.	Add *imm32 sign-extended to 64-bits* to RAX.
80 /0 *ib*	ADD *r/m8, imm8*	MI	Valid	Valid	Add *imm8* to *r/m8*.
REX + 80 /0 *ib*	ADD *r/m8*, imm8*	MI	Valid	N.E.	Add *sign*-extended *imm8* to *r/m8*.
81 /0 *iw*	ADD *r/m16, imm16*	MI	Valid	Valid	Add *imm16* to *r/m16*.
81 /0 *id*	ADD *r/m32, imm32*	MI	Valid	Valid	Add *imm32* to *r/m32*.
REX.W + 81 /0 *id*	ADD *r/m64, imm32*	MI	Valid	N.E.	Add *imm32 sign-extended to 64-bits* to *r/m64*.
83 /0 *ib*	ADD *r/m16, imm8*	MI	Valid	Valid	Add *sign*-extended *imm8* to *r/m16*.
83 /0 *ib*	ADD *r/m32, imm8*	MI	Valid	Valid	Add *sign*-extended *imm8* to *r/m32*.
REX.W + 83 /0 *ib*	ADD *r/m64, imm8*	MI	Valid	N.E.	Add *sign*-extended *imm8* to *r/m64*.
00 /*r*	ADD *r/m8, r8*	MR	Valid	Valid	Add *r8* to *r/m8*.
REX + 00 /*r*	ADD *r/m8*, r8**	MR	Valid	N.E.	Add *r8* to *r/m8*.
01 /*r*	ADD *r/m16, r16*	MR	Valid	Valid	Add *r16* to *r/m16*.
01 /*r*	ADD *r/m32, r32*	MR	Valid	Valid	Add r32 to *r/m32*.
REX.W + 01 /*r*	ADD *r/m64, r64*	MR	Valid	N.E.	Add r64 to *r/m64*.
02 /*r*	ADD *r8, r/m8*	RM	Valid	Valid	Add *r/m8* to *r8*.
REX + 02 /*r*	ADD *r8*, r/m8**	RM	Valid	N.E.	Add *r/m8* to *r8*.
03 /*r*	ADD *r16, r/m16*	RM	Valid	Valid	Add *r/m16* to *r16*.
03 /*r*	ADD *r32, r/m32*	RM	Valid	Valid	Add *r/m32* to *r32*.
REX.W + 03 /*r*	ADD *r64, r/m64*	RM	Valid	N.E.	Add *r/m64* to *r64*.

*In 64-bit mode, r/m8 can not be encoded to access the following byte registers if a REX prefix is used: AH, BH, CH, DH.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
RM	ModRM:reg (r, w)	ModRM:r/m (r)	NA	NA
MR	ModRM:r/m (r, w)	ModRM:reg (r)	NA	NA
MI	ModRM:r/m (r, w)	imm8/16/32	NA	NA
I	AL/AX/EAX/RAX	imm8/16/32	NA	NA

MOV — Move

Opcode	Instruction	Op/En	64-Bit Mode	Compat/Leg Mode	Description
88 /*r*	MOV *r/m8,r8*	MR	Valid	Valid	Move *r8* to *r/m8*.
REX + 88 /*r*	MOV *r/m8***,r8****	MR	Valid	N.E.	Move *r8* to *r/m8*.
89 /*r*	MOV *r/m16,r16*	MR	Valid	Valid	Move *r16* to *r/m16*.
89 /*r*	MOV *r/m32,r32*	MR	Valid	Valid	Move *r32* to *r/m32*.
REX.W + 89 /*r*	MOV *r/m64,r64*	MR	Valid	N.E.	Move *r64* to *r/m64*.
8A /*r*	MOV *r8,r/m8*	RM	Valid	Valid	Move *r/m8* to *r8*.
REX + 8A /*r*	MOV *r8***,r/m8****	RM	Valid	N.E.	Move *r/m8* to *r8*.
8B /*r*	MOV *r16,r/m16*	RM	Valid	Valid	Move *r/m16* to *r16*.
8B /*r*	MOV *r32,r/m32*	RM	Valid	Valid	Move *r/m32* to *r32*.
REX.W + 8B /*r*	MOV *r64,r/m64*	RM	Valid	N.E.	Move *r/m64* to *r64*.
8C /*r*	MOV *r/m16,Sreg***	MR	Valid	Valid	Move segment register to *r/m16*.
REX.W + 8C /*r*	MOV *r16/r32/m16, Sreg***	MR	Valid	Valid	Move zero extended 16-bit segment register to *r16/r32/r64/m16*.
REX.W + 8C /*r*	MOV *r64/m16, Sreg***	MR	Valid	Valid	Move zero extended 16-bit segment register to *r64/m16*.
8E /*r*	MOV *Sreg,r/m16***	RM	Valid	Valid	Move *r/m16* to segment register.
REX.W + 8E /*r*	MOV *Sreg,r/m64***	RM	Valid	Valid	Move *lower 16 bits of r/m64* to segment register.
A0	MOV AL,*moffs8**	FD	Valid	Valid	Move byte at (*seg:offset*) to AL.
REX.W + A0	MOV AL,*moffs8**	FD	Valid	N.E.	Move byte at (*offset*) to AL.
A1	MOV AX,*moffs16**	FD	Valid	Valid	Move word at (*seg:offset*) to AX.
A1	MOV EAX,*moffs32**	FD	Valid	Valid	Move doubleword at (*seg:offset*) to EAX.
REX.W + A1	MOV RAX,*moffs64**	FD	Valid	N.E.	Move quadword at (*offset*) to RAX.
A2	MOV *moffs8*,AL	TD	Valid	Valid	Move AL to (*seg:offset*).
REX.W + A2	MOV *moffs8****,AL	TD	Valid	N.E.	Move AL to (*offset*).
A3	MOV *moffs16**,AX	TD	Valid	Valid	Move AX to (*seg:offset*).
A3	MOV *moffs32**,EAX	TD	Valid	Valid	Move EAX to (*seg:offset*).
REX.W + A3	MOV *moffs64**,RAX	TD	Valid	N.E.	Move RAX to (*offset*).
B0+ *rb ib*	MOV *r8, imm8*	OI	Valid	Valid	Move *imm8* to *r8*.
REX + B0+ *rb ib*	MOV *r8***, imm8*	OI	Valid	N.E.	Move *imm8* to *r8*.
B8+ *rw iw*	MOV *r16, imm16*	OI	Valid	Valid	Move *imm16* to *r16*.
B8+ *rd id*	MOV *r32, imm32*	OI	Valid	Valid	Move *imm32* to *r32*.
REX.W + B8+ *rd io*	MOV *r64, imm64*	OI	Valid	N.E.	Move *imm64* to *r64*.
C6 /0 *ib*	MOV *r/m8, imm8*	MI	Valid	Valid	Move *imm8* to *r/m8*.
REX + C6 /0 *ib*	MOV *r/m8***, imm8*	MI	Valid	N.E.	Move *imm8* to *r/m8*.
C7 /0 *iw*	MOV *r/m16, imm16*	MI	Valid	Valid	Move *imm16* to *r/m16*.
C7 /0 *id*	MOV *r/m32, imm32*	MI	Valid	Valid	Move *imm32* to *r/m32*.
REX.W + C7 /0 *id*	MOV *r/m64, imm32*	MI	Valid	N.E.	Move *imm32 sign extended to 64-bits* to *r/m64*.

* The *moffs8*, *moffs16*, *moffs32* and *moffs64* operands specify a simple offset relative to the segment base, where 8, 16, 32 and 64 refer to the size of the data. The address-size attribute of the instruction determines the size of the offset, either 16, 32 or 64 bits.

** In 32-bit mode, the assembler may insert the 16-bit operand-size prefix with this instruction (see the following "Description" section for further information).

***In 64-bit mode, *r/m8* can not be encoded to access the following byte registers if a REX prefix is used: AH, BH, CH, DH.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
MR	ModRM:r/m (w)	ModRM:reg (r)	NA	NA
RM	ModRM:reg (w)	ModRM:r/m (r)	NA	NA
FD	AL/AX/EAX/RAX	Moffs	NA	NA
TD	Moffs (w)	AL/AX/EAX/RAX	NA	NA
OI	opcode + rd (w)	imm8/16/32/64	NA	NA
MI	ModRM:r/m (w)	imm8/16/32/64	NA	NA