Premier examen – Corrigé

Directives générales

- L'examen se fait individuellement. Tout plagiat sera rapporté à la direction du département et sévèrement puni.
- Vous avez droit aux manuels et aux notes de cours, ainsi qu'à vos notes personnelles.
- Vous devez identifier chacun de vos cahiers réponses avec les renseignements suivants : nom, prénom, numéro matricule, numéro du cahier et nombre total de cahiers.
- Vous devez rendre votre copie avant 11 h 20.
- Un maximum de 3 points peuvent être retranchés à cause de la qualité du français (0,1 point par faute).

Section 1 – Questions à choix de réponses

Question 1.1 – Historique de l'architecture IA32

Pour les deux sous questions suivantes, utilisez les choix de réponses suivants :

a. 286
b. 386
c. 4004
d. 80186
e. 8080
f. 8086
g. 8088
h. Pentium 4
i. Pentium
j. Pentium II
k. Pentium III
l. Pentium MMX
m. Pentium Pro
n. Xeon

1) Dans quel processeur de la famille IA32 le mode « protégé » a-t-il fait son apparition ?
2) Dans quel processeur de la famille IA32 la « pagination » a-t-elle fait son apparition ?

/ 1
(2 minutes)

Réponse 1.1

1) 286 (a)
2) 386 (b)

- Chaque bonne réponse donne 0,5 point.

Question 1.2 – Modes d'adressages

Pour chacune des sous questions suivantes, utilisez les choix de réponses suivants :

a. Immédiate
b. Implicite
c. Registre
d. Adressage direct
e. Adressage indirect
f. Adressage indirect avec déplacement
g. Adressage indirect avec index
h. Adressage indirect avec index et déplacement

Pour chacune des lignes de la routine assembleur, identifiez les modes d'adressage utilisés :

Numéro de ligne	
1	`strlen: mov esi, [esp+4]`
2	`mov eax, 0`
3	`@@: cmp byte ptr [esi+eax*1], 0`
4	`je @F`
5	`inc eax`
6	`jmp @B`
7	`@@: ret`

Figure 1.1.1 : strlen

/ 2
(5 minutes)

Réponse 1.2

1) Registre, Indirect avec déplacement
2) Registre, Immédiate
3) Indirect avec index, Immédiate
4) Immédiate, Implicite (le registre de drapeaux)
5) Registre
6) Immédiate
7) Implicite (le pointeur de pile et la pile)

- Pour chaque élément incorrect 0,2 point est retranché.

Section 2 – Questions à développement

Question 2.1 – Interface avec les langages de haut niveau

Le prototype suivant a été défini dans un programme C.

```
extern unsigned int __stdcall Sub32( unsigned int aA,  unsigned
int aB );
```

Figure 2.1.1 : Prototype de Sub32

Vous devez écrire cette fonction en assembleur. Elle doit retourner aA – aB. N'oubliez pas qu'elle doit être utilisée à partir d'un module C, donc d'un module autre que celui où votre fonction est définie.

Notes

- Le segment de code courant travaille sur des opérandes de 32 bits.
- Le segment de pile courant utilise des adresses de 32 bits.

/ 3
(15 minutes)

Réponse 2.1

```
1     public Sub32
2  Sub32:
3     mov eax, [esp+4]
4     sub eax, [esp+8]
5     ret 8
```

Figure 2.1.2 : Exemple de réponse

- La présence de la directive `public` donne 0,5 point (ligne 1 de l'exemple de réponse).
- La définition de l'étiquette donne 0,5 point (ligne 2 de l'exemple de réponse).
- La lecture dans le bon ordre des arguments `aA` et `aB` et la fonctionnalité de la soustraction donnent 1 point (ligne 3 et 4 de l'exemple de réponse).
- Le dépilement des arguments lors du retour donne 1 point (ligne 5 de l'exemple de réponse).

Question 2.2 – Type de données et organisation en mémoire

Voici le contenu d'une région de mémoire.

Adresse	Octet
0x80120000	
0x80120001	
0x80120002	0x00
0x80120003	0x01
0x80120004	0x00
0x80120005	0x00
0x80120006	0x40
0x80120007	0x43
0x80120008	0x01
0x80120009	0x00
0x8012000a	

Figure 2.2.1 : Bloc de mémoire

1) Donnez, en nombre décimal, la valeur du mot (word) non signé se trouvant à l'adresse `0x80120002` ?
2) Donnez, en nombre décimal, la valeur du mot double (double word) non signé se trouvant à l'adresse `0x80120006` ?
3) Donnez, en nombre décimal, la valeur virgule flottante de précision simple (float) se trouvant à l'adresse `0x80120004` ?

/ 2
(10 minutes)

Réponse 2.2

1) `0x0100` soit 256.
2) `0x00014340` soit 85752.
3)

`0x43400000`
`0100 0011 0100 0000 0000 0000 0000 0000`
`01000011010000000000000000000000`
`0 10000110 10000000000000000000000`

Le bit de signe est 0. La valeur est donc positive.
L'exposant est 10000110_2 soit 134_{10}. Il faudra donc utiliser la puissance de 2 de 7_{10} (134_{10}-127_{10}).
La partie fractionnaire est $1.10000000000000000000000_2$, soit 1.5_{10}.
La valeur est donc $1.5 * 2^7$ soit 192.0.

- La valeur du mot donne 0,5 point.
- La valeur du mot double donne 0,5 point.
- La valeur virgule flottante donne 1 point.

Question 2.3 – Variables locales et bloc de pile (« Stack frame »)

Voici une fonction assembleur :

```
Fonction:
      enter 8, 0
      mov  eax, [ebp+8]  ; eax <- aA
      mov  [ebp-4], eax  ; lA <- eax
      mov  ebx, [ebp+12] ; ebx <- aB
      mov  [ebp-8], ebx  ; lB <- eax
      leave
      ret
```

Figure 2.3.1 : Fonction

Le prototype C de cette fonction est :

```
extern void __cdecl Fonction( int aA, int aB );
```

Figure 2.3.2 : Prototype de Fonction

Vous devez dessiner la pile telle qu'elle est juste avant d'exécuter l'instruction `leave`. Dans ce dessin vous devez indiquer clairement l'emplacement des arguments `aA` et `aB`, l'emplacement des variables locales `lA` et `lB`, l'emplacement de l'adresse de retour. Vous devez aussi indiquer où pointent les registres `esp` et `ebp`.

Notes

- Le segment de code courant travaille avec des opérandes de 32 bits.
- Le segment de pile courant utilise des adresses de 32 bits.

/ 3
(15 minutes)

Réponse 2.3

Adresse haute	…		
	`aB`	`[ebp+12]`	`[esp+20]`
	`aA`	`[ebp+8]`	`[esp+16]`
	Adresse de retour	`[ebp+4]`	`[esp+12]`
	ebp précédent	**`[ebp]`**	`[esp+8]`
	`lA`	`[ebp-4]`	`[esp+4]`
Adresse basses	**`lB`**	`[ebp-8]`	**`[esp]`**

Figure 2.3.3 : Représentation de la pile

- L'ordre des arguments donne 0,5 point.
- L'ordre des variables locales donne 0,5 point.
- La position de l'adresse de retour immédiatement sous `aA`, donne 0,5 point.
- La présence de l'espace pour le `ebp` précédent donne 0,5 point, même si elle n'est pas identifiée.
- L'indication que `esp` pointe sur `lB` donne 0,5 point.
- L'indication que `ebp` pointe sur l'espace réservé pour le `ebp` précédent donne 0,5 point.

Question 2.4 – Branchements conditionnels

Voici un code un peu tordu. Avant le début de l'exécution, les drapeaux « Carry », « Zero », « Sign » et « Overflow » sont tous à zéro.

```
      mov eax, 0a9876543h
      add eax, 065432100h
      mov eax, 1
      dec eax
      jne @F
      nop ; Point A
@@:
      jbe @F
      nop ; Point B
@@:
      jge @F
      nop ; Point C
@@:
      jc  @F
      nop ; Point D
```

Figure 2.4.1 : Comparaisons et branchements

1) Quel est l'état des drapeaux « Carry », « Zero », « Sign », « Overflow » après l'exécution de l'instruction `add` ?
2) Quel est l'état de ces mêmes drapeaux après l'exécution de l'instruction `dec` ?
3) Pour chacune des instructions `nop`, dites si elle est exécutée ou non.

Notes

- Le segment de code courant travaille avec des opérandes de 32 bits.
- Le segment de pile courant utilise des adresses de 32 bits.

/ 3
(15 minutes)

Réponse 2.4

1) Carry = 1, Zero = 0, Sign = 0 et Overflow = 0
2) Carry = 1, Zero = 1, Sign = 0 et Overflow = 0
3) Point A = oui, Point B = non, Point C = non, Point D = non

- Chaque drapeau exact donne 0,25 point.
- Chaque point d'exécution correctement prédit donne 0,25 point.

Question 2.5 – Programmation virgule flottante

Soit les deux variables virgule flottante de double précision `sX` et `sY` et les constantes `A`, `B` et `D`. Écrivez le code évaluant l'expression suivante.

```
sY = ( ( sX + A ) * ( sX - B ) ) / D
```

Considérez que vous ne connaissez pas l'état de l'unité virgule flottante au début de l'exécution de votre code. Cependant, vous n'avez pas à sauvegarder l'état de l'unité de virgule flottante.

```
.const
A dq 1.5
B dq 8.1
D dq 2.0

.data
sX dq 1.0
sY dq 0.0
```

Figure 2.5.1 : Déclaration des variables et constantes

Notes

- Le segment de code courant travaille avec des opérandes de 32 bits.
- Le segment de pile courant utilise des adresses de 32 bits.

/ 3
(15 minutes)

Réponse 2.5

```
finit

fld   sX
fadd  A
fld   sX
fsub  B
fmulp st(1), st(0)
fdiv  D
fstp  sY
```

Figure 2.5.2 : Exemple de réponse

- Un point est accordé pour la présence de l'instruction finit.
- Deux points sont accordés pour la gestion de la pile de registre virgule flottante et la logique de l'approche.

Question 2.6 - Programmation MMX

Vous devez écrire une fonction assembleur en utilisant les instructions MMX. Cette fonction doit trouver la valeur maximum dans un tableau d'octets non signés. Le prototype C de la fonction est le suivant :

```
extern unsigned char __cdecl  Max( unsigned char *aPtr );
```

Figure 2.6.1 : Prototype de Max

Le tableau a une longueur de 128 éléments. La fonction retourne la valeur maximale trouvée.

/ 5
(25 minutes)

Réponse 2.6

```
Max:

    mov esi, [esp+4]
    mov ecx, 32

    pxor MM0, MM0

@@: pmaxub MM0, [esi]

    add esi, 8
    loop @B

    movq      MM1, MM0
    punpcklbw MM1, MM1
    punpckhbw MM0, MM0
    pmaxub    MM0, MM1

    movq      MM1, MM0
    punpcklbw MM1, MM1
    punpckhbw MM0, MM0
    pmaxub    MM0, MM1

    movq      MM1, MM0
    punpcklbw MM1, MM1
    punpckhbw MM0, MM0
    pmaxub    MM0, MM1

    movd eax, MM0
    and  eax, 0ffh

    emms

    ret
```

Figure 2.6.2 : Exemple de réponse

- Un point est accordé pour le respect de la convention d'appel (argument, valeur de retour et retour sans nettoyer la pile).
- Un point est accordé pour la structure de contrôle en boucle permettant de traiter le tableau au complet.
- Un point est accordé pour les instructions MMX effectuant le maximum 8 éléments à la fois.
- Un point est accordé pour les instructions MMX (ou non) trouvant le maximum parmi les huit maximums partiels contenus dans un registre MMX.
- Un point est accordé pour la présence de l'instruction emms.

Question 2.7 – Extension SIMD 2

L'extension SIMD 2 ajoute des instructions permettant de mieux contrôler la gestion de la mémoire cache du Pentium 4. Pourquoi est-il parfois avantageux de lire ou d'écrire des données en mémoire, sans que celles-ci soient placées en mémoire cache ?

/ 2
(10 minutes)

Réponse 2.7

Plusieurs algorithmes traitent un grand nombre de données placées linéairement en mémoire, sans utiliser plusieurs fois les mêmes données. Ces données sont dites « non temporelles ». Dans ce cas, le fait de placer ces données en mémoire cache n'apporte aucun avantage. De plus, les placer en mémoire cache oblige le processeur à retirer d'autres données de la cache, pour libérer de l'espace. Il peut donc en résulter une baisse de performance, si les données expulsées de la cache sont à nouveau nécessaires rapidement.

- Un point accordé pour le concept de donnée non temporelle.
- Un point accordé pour le désavantage de sortir des données temporelles de la mémoire cache.

Question 2.8 – Segmentation

La figure suivante donne le contenu de la mémoire, ainsi que la valeur du « Global Descriptor Table Register » (`GDTR`) et du « Local Descriptor Table Register » (`LDTR`)

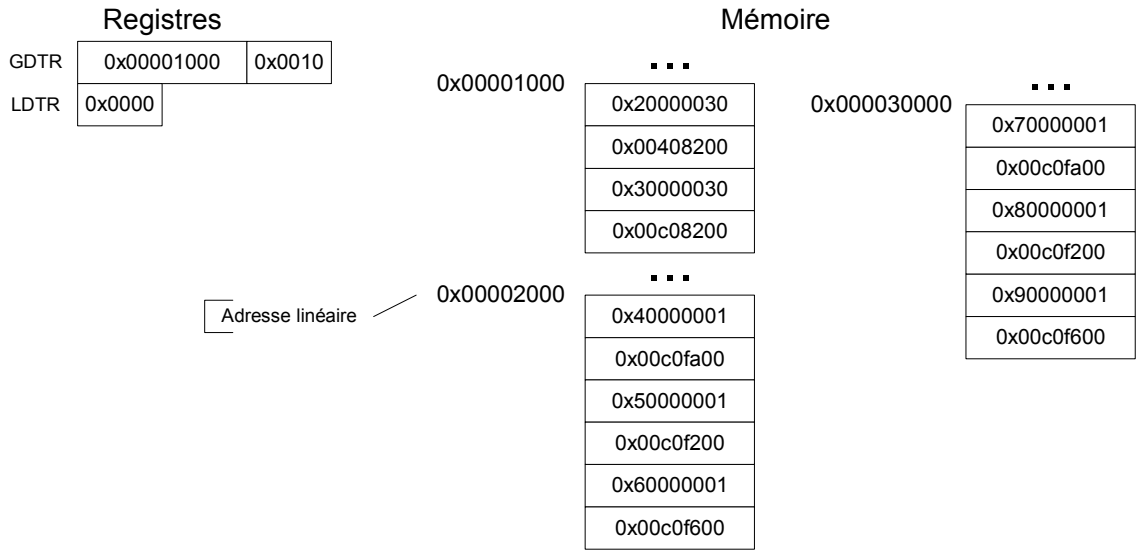

Figure 2.8.1 : Tables de segments

Utilisez les informations de la figure pour convertir l'adresse logique suivante en adresse linéaire correspondante.

Sélecteur de segment	Déplacement
`0x000f`	`0x00000100`

Figure 2.8.2 : Adresse logique

Dites aussi si cette espace mémoire peut être lue, écrite et/ou exécutée.

/ 2
(10 minutes)

Réponse 2.8

Étape 1 – Décoder le sélecteur de segment

`0x0017 = 0000 0000 0000 1111`$_2$
Index = `0000 0000 0000 1`$_2 = 2_{10}$
Indicateur de table = 1_2 = Locale
Niveau de privilège = $11_2 = 3_{10}$ = Utilisateur

Étape 2 – Déterminer l'adresse du descripteur de la table locale

LDTR = 0
Index = 0
Indicateur de table = 1 = Globale
Niveau de privilège = 0 = Système

Le premier descripteur de la table globale est donc le descripteur de la table locale. Le registre `GDTR` donne l'adresse de départ de la table des descripteurs globaux. L'adresse du premier descripteur global est donc `0x00001000`.

Étape 3 – Interpréter le descripteur de la table locale

La valeur du descripteur est `0x20000030 : 0x00408200`
Base = `0x00002000`
Limite = `0x00030` = 48 octets (car G = 0)
A = 0
Type = 1 = Lecture et écriture
S = 0 = Segment système
DPL = 0 = Système
P = 1 = Présent en mémoire
D = 1 = Segment > 64 Ko
G = 0 = Limite exprimée en octets
Une longueur de 48 octets indique que la table de descripteur local contient six descripteurs.

Étape 4 – Déterminer l'adresse du descripteur de segment

Le descripteur à considérer est celui qui porte l'index 1 dans la table de descripteurs locaux. La table locale débute à l'adresse `0x00002000`. Chacun des descripteurs compte 8 octets. Le descripteur portant l'index 1 se trouve donc à l'adresse `0x00002008`.

Étape 5 – Interpréter le descripteur de segment

La valeur du descripteur est `0x50000001 : 0x00c0f200`
Base = `0x00005000`
Limite = `0x00001` = 1 page de 4 Ko (car G = 1)
A = 0
Type = 1 = Lecture et écriture
S = 1 = Segment utilisateur
DPL = 3 = Utilisateur
P = 1 = Présent en mémoire
D = 1 = Segment > 64 Ko
G = 1 = Limite exprimée en page de 4 Ko.

Nous savons donc déjà que le programme utilisateur a le droit de lire et d'écrire cette case mémoire, mais qu'il n'a pas le droit de l'exécuter.

Étape 6 – Calculer l'adresse linéaire

Le segment débute à l'adresse `0x00005000`
Le déplacement est de `0x00000100`
L'adresse linéaire est donc `0x00005100`

- 0,5 point est accordé pour le décodage du descripteur de segment.

- 0,5 point est accordé pour l'utilisation du LDTR et la recherche dans la table globale.
- 0,5 point est accordé pour le calcul de l'adresse de base de la table locale.
- 0,5 point est accordé pour le calcul final de l'adresse linéaire.

Question 2.9 – Pagination

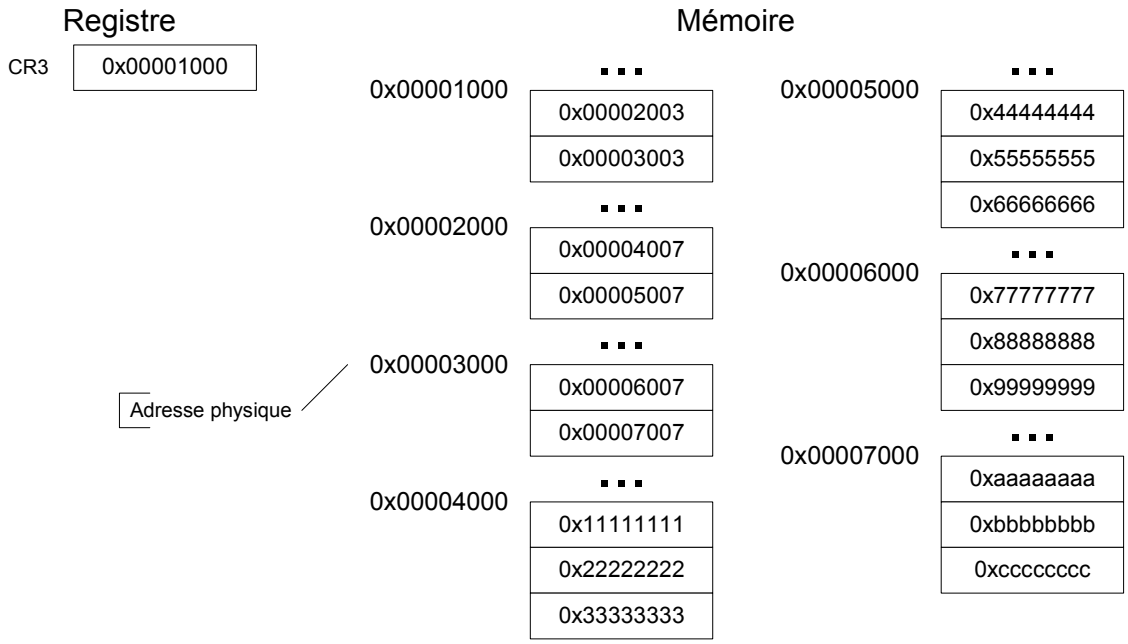

Figure 2.9.1 : Répertoires et tables de pages

En utilisant les infirmations de la figure, déterminez l'adresse physique correspondant à l'adresse linéaire 0x00001004. Donnez aussi la valeur du mot double lue à cette adresse. Est-ce qu'une lecture à cette adresse linéaire provoquera une faute de page ?

/ 2
(10 minutes)

Réponse 2.9

Étape 1 – Diviser l'adresse linéaire en ses trois parties

`0x00001004`
`0000 0000 0000 0000 0001 0000 0000 0100`
`0000000000 0000000001 000000001000`
Index de la table de page = `0000000000` = 0
Index dans la table de page = `0000000001` = 1
Déplacement dans la page = `000000000100` = 4

Étape 2 – Déterminer l'adresse de la table de page

Le registre CR3 nous indique que le répertoire des tables de pages débute à l'adresse physique `0x00001000`. Le premier entré du répertoire (index 0) contient `0x00002003`

P = 1 = Présente en mémoire
R/W = 1 = Lecture écriture permise
U/S = 0 = Page système
PWT = 0 = Cache activé pour les écritures
PCD = 0 = Cache activé pour les lectures
A = 0
PS = 0 = Page de 4 Ko
Adresse de la table de pages = `0x00002000`

Étape 3 – Déterminer la base de la page

La table de page débute à 0x00002000 et l'index dans la table de page est 1. L'adresse de base de la page se trouve donc à l'adresse physique `0x00002004`. Cette case de mémoire contient `0x00005007`.

P = 1 = Présente en mémoire
R/W = 1 = Écriture permise
U/S = 1 = Page utilisateur
PWT = 0 = Cache activé pour les écritures
PCD = 0 = Cache activé pour les lectures
A = 0
D = 0
PAT = 0 = Page de 4 Ko
Adresse de base de la page = `0x00005000`

Comme la page est présente en mémoire, la lecture de l'adresse ne provoquera pas de faute de page.

Étape 4 – Déterminer l'adresse physique

L'adresse de base de la page est `0x00005000` et le déplacement dans la page est de 4 donc l'adresse physique complète est `0x00005004`. Le mot double contenu à cette adresse est `0x55555555`.

- Un point est accordé pour le calcul de l'adresse.
- 0,5 point est accordé pour la valeur du mot double.
- 0,5 point est accordé pour l'indication qu'il n'y aura pas de faute de page.

Question 2.10 – Codage des instructions

Donnez la représentation hexadécimale de l'instruction suivante.

```
add eax, [esi+16]
```

Notes

- Le segment de code courant travaille avec des opérandes de 32 bits.
- Le segment de pile courant utilise des adresses de 32 bits.

/ 2
(10 minutes)

Réponse 2.10

Étape 1 – Déterminer le code d'opération

Le manuel nous donne `0x03`.

Étape 2 – Déterminer l'octet ModR/M

La table du chapitre 2 du volume 2 nous donne un mode 01 et un R/M de 110 pour ce mode d'adressage avec un déplacement encodé sur 8 bits. Elle nous donne aussi la valeur complète de `0x46`.

Étape 3 – Le déplacement

Le déplacement est de 16 octets. Il est encodé sur 8 bits. La valeur est donc `0x10`.

Étape 4 – Assemblage des trois parties

0x03 0x46 0x10