1 Exercices : représentation de l'information

1.1 Tailles de codage[1]

1. Quel est l'intervalle des entiers positifs codables sur 16 bits en RBNS ?
2. On désire représenter en RBNS les entiers positifs compris entre 0 et 999. Quelle taille minimale, en bits, doivent avoir ces mots ? Donnez la formule de calcul !
3. … et si l'on avait des composants mémoires ternaires (3 états :0, 1, 2) ?
4. … et si l'on avait des composants mémoires décimaux (10 états :0..9)?

Quel est la taille minimale en chiffre b-aires, pour représenter l'intervalle [0, k] ?

1.2 L'octet et ses multiples

1. Sachant qu'une page de courrier contient en moyenne 2000 caractères et que le disque D contient 400 courriers de 3 pages chacun. Estimez la capacité minimale du disque en Mo.
2. Combien y a-t-il de Ko dans 2 Mo et dans 3 Go ?
3. Une transmission sur une ligne série à une vitesse de 1 Kb/s (1000 bits/seconde) permet de transférer combien d'octets par minute ? Donnez également la réponse en Ko par heure.

1.3 Conversions et additions RBNS

1. Remplissez le tableau suivant

Binaire (RBNS)	Octal	Hexadécimal	Décimal
1111 1100			
	77		
		12H	
			233
0010 1101			
	12		
		0EAH	
			272
		0 CAFE H	

2. En RBNS 8 bits, exprimez la formule de décodage d'un octet $b_7..b_0$ en l'entier positif décimal x qu'il représente : $x=f(b_i)$.
3. En RBNS 8 bits, exprimez une formule de codage d'un entier positif décimal x en un bit de l'octet $b_7..b_0$ qui le représente : $b_i=g(x)$.
4. Remplissez le tableau des additions binaires suivant (additions et résultats).

Binaire (RBNS)	Octal	Hexadécimal	Décimal
0010 0111+1110 0101			
	77+21		
		0C1H+4FH	
			138+117
0010 1111+0011 0101			

5. Ecrire l'algorithme de traduction d'un nombre en une chaîne de caractères le représentant en base 2. On dispose d'une fonction char toChar(int) transformant un entier en caractère : toChar(0)= «0».

---

[1] Donnez les résultats numériques à 10^{-2} près

1.4 Conversions et additions des entiers relatifs

1. Quel est l'intervalle des entiers codables sur 10 bits en C1 et C2 ?
2. Quel est l'intervalle des entiers représentables en excédent à 128 ? Quelle différence avec le C2 sur 8 bits ?
3. Remplissez le tableau suivant en codant sur un octet

x_{10}	x_2[2]	$(-x)_{VAS}$[3]	$(-x)_{C1}$	$(-x)_{E128}$[4]	$(-x)_{C2}$	$((-x)_{C2})_{16}$
0						
3						
127						
128						

4. Sur un octet en C2 et en excédent 128, effectuer les additions binaires suivantes en indiquant, le résultat décimal représenté, sa cohérence et le positionnement des indicateurs Carry et Overflow.
0+0; 1+1; -1+-1; 127+127; -128+-128; 1+-1; 3+-4; 127+-126; 2+-127

5. Quelles remarques tirez-vous des additions précédentes en C2 et en E128 ? Quelle(s) opération(s) d'ajustement vous semblent nécessaires pour la cohérence des additions binaires en représentation E128 ?

1.5 Rappels sur les entiers

1. Quels sont les intervalles d'entiers représentables en RBNS et en C2 sur un octet ?
2. Quel est l'indicateur de débordement en RBNS et en C2 ?
3. Soit la représentation hexadécimale $h_{n-1}..h_0$ d'un entier positif, exprimez la formule de décodage de $h_{n-1}..h_0$ en l'entier positif décimal x qu'il représente : $x=f(h_{n-1}..h_0)$. Exprimez la formule de codage d'un entier positif décimal x en un chiffre hexadécimal h_i de $h_{n-1}..h_0$ qui le représente : $h_i=g(x)$. On pose : $0 \leq h_i \leq 15$
4. Démontrez l'impossibilité d'un débordement (OF=1) après une addition d'un positif et d'un négatif en C2 sur un octet. (par l'absurde)

1.6 Représentation entière DCB

1. Quel est l'intervalle d'entier représentable en DCB sur un octet ?
2. Indiquez les valeurs DCB et hexadécimales des octets suivants :
0000 0001; 1001 0011; 0101 1011; 1100 1000; 0111 0101; 0100 1001.
3. Effectuez les additions binaires suivantes sur des octets DCB et indiquez les résultats décimaux correspondants : 33+66; 12+34; 56+27; 19+19
4. Lors d'additions binaires d'octets en DCB, qu'est-ce qui indique l'incohérence du résultat (56+27 ou 19+19), à ne pas confondre avec le débordement (90+90) ? Une fois l'incohérence détectée, il faut déclencher l'opération d'ajustement du résultat. Quelle doit être cette opération d'ajustement ?

1.7 Représentation des nombres décimaux à virgule (DCB)

Soit le codage DCB à virgule suivant :

---

[2] la base 2 correspond à C1, C2, RBNS, VAS qui sont identiques pour les positifs.
[3] Valeur Absolue Signée
[4] Excédent à 128

Une suite d'octets permet de coder un nombre décimal signé comme suit :
• 1 octet d'en-tête donne (en RBNS) le nombre total n de quartets utilisés pour représenter les chiffres décimaux.
• 1 octet spécifie la position m (en RBNS) de la virgule (à gauche du $m^{ième}$ chiffre en partant de la droite).
• 1 quartet indique le signe du nombre (0H:+; 0FH:-).
• n quartets représentant les chiffres en DCB.

a) Quelle est la taille mémoire minimale et maximale utilisée pour représenter des nombres ? Donnez un exemple de code en hexadécimal pour les deux extrêmes.
b) Représenter en hexadécimal les codes des nombres suivants : -32,5; 3,14159; 1000; 255. Indiquez également la taille du code pour chaque nombre.
c) Avantage et inconvénient de ce type de codage ?

1.8 Représentation des nombres décimaux à virgule (Virgule flottante)

a) Ecrire la représentation en virgule flottante normalisée en base 10 des nombres suivants : 45; 67,89; 10^{-12}; 435,123; 0,0045
b) Ecrire la représentation en virgule flottante normalisée en base 2 avec 10 bits de mantisse (11 avec le bit caché) des nombres suivants : 8; 3,5; 33,625; 0,13 (utiliser la méthode des multiplications successives par 2 de la partie décimale : voir cours). Donner la valeur décimale exacte de la valeur binaire représentant 0,13.

1.9 Virgule flottante en machine

Soit le codage **IEEE-754** simple précision sur 32 bits (float) suivant :
- signe : 1 bit (0 : +, 1 : -)
- exposant : 8 bits en excédent 127 [-127, 128]
- mantisse : 23 bits en RBNS ; normalisé sans représentation du 1 de gauche ! La mantisse est **arrondie** !

a) Exprimez en binaire et en hexa les codes correspondants aux nombres suivants : 3,5; -3,5; 65,25; -333,13
b) Quels sont les deux plus grands (en valeur absolue) nombres positif et négatif ?
c) Quels sont les deux plus petits (en valeur absolue) nombres non nuls positif et négatif ?
d) Quel est le plus petit et le plus grand pas entre deux nombres exprimables ?
e) Vers quelle limite (grands nombres ou petits nombres) peut-on exprimer le plus possible de chiffres décimaux <u>significatifs</u> ?
f) Que faudrait-il faire pour augmenter ce nombre de chiffres décimaux significatifs ? Donner un exemple.
g) Quel taille minimale de mantisse permet d'obtenir 20 chiffres décimaux significatifs ?
h) Le type double code sa mantisse sur 52 bits. Combien de chiffres décimaux significatifs cela représente-t-il ?
i) Effectuer les additions suivantes en base 10 en utilisant une mantisse signée normalisée de 4 chiffres et un exposant signé de 2 chiffres :
1+-0,999; 1+-0,9992; 9,9+0,8 10^{-4}; -1+(1+0,0008); (-1+1)+0,0008; 10+0,001
Qu'en concluez-vous ?

1.10 Codes alphanumériques

1. Quelle sont les tailles des codes ASCII, EBCDIC, ISO8859-1, Unicode ?
2. Calculez la représentation ASCII 7 bits + 1 bit de parité paire des chiffres de 0 à 9 et des lettres de A à F. Exprimez les résultats en hexadécimal.

3. Quelle opération doit-on utiliser sur chaque octet d'une chaine ASCII en parité paire pour la transformer en parité impaire ?

1.11 Code de Hamming

1. Combien de bits de contrôle sont-ils nécessaires pour 1, 4, 11 ou 26 bits d'information ?
2. Indiquez la formule de calcul de k le nombre de bits de contrôle par rapport à n le nombre de bits d'info.
3. Représenter en hexa. la chaîne des bits correspondant à « hexa » et « bit » en code de Hamming à parité paire en partant de l'ASCII 7 bits.

1.12 Code de Huffman

Soit la distribution probabiliste suivante pour les dix caractères représentant les chiffres décimaux :

Caractère	0	1	2	3	4	5	6	7	8	9
Probabilité	0,05	0,1	0,11	0,11	0,15	0,06	0,08	0,2	0,07	0,07

1. Etablir manuellement un code de Huffman pour cette distribution en expliquant votre démarche.
2. Donner la longueur moyenne de codage d'un chiffre du message.
3. Décrire les structures de données et l'algorithme utilisé en 1. pour construire l'arbre et calculer le code de Huffman associé.

2 La couche physique

2.1 Structure des ordinateurs

1. Un micro-processeur 32 bits à 2 GHz sans pipeline effectue une addition 32+32 bits (registre + case MC avec résultat en mémoire centrale) en 3 cycles d'horloge. Donnez la durée de cette instruction en seconde(s) et en nanosecondes. Où l'addition s'est-elle effectuée et pourquoi 3 cycles sont-ils nécessaires ?
2. Comment appelle-t-on le registre contenant les indicateurs d'état et de contrôle du programme ? Listez 2 ou 3 de ces indicateurs.
3. Si l'on observe l'UC alors qu'elle exécute l'instruction NOP (opération ne faisant rien) située à l'adresse MC 100H, quelle est la valeur du compteur ordinal ? Serait-ce la même valeur si l'on exécutait l'instruction ADD R1, 0FFFFH ?
4. Quel est l'intérêt primordial du disque magnétique par rapport à la bande magnétique en tant que support mémoire ?
5. Soit un disque dur 16 faces, 100 cylindres, 20 secteurs d'1/2 Ko par piste, tournant à 3600 tours/mn, ayant une vitesse de translation de 2 m/s et une distance de 10 cm entre la 1° et la dernière piste. Quel est le temps d'accès moyen à un secteur ? Quel est le temps de transfert d'un secteur ? Ce temps de transfert est-il moyen, variable, fixe ? Pourquoi a-t-on intérêt à positionner les 10 Ko d'un fichier sur le même cylindre ? Quel sera le temps de transfert d'un tel fichier ?

2.2 Rappels de logique booléenne

1. Rappelez les lois de Morgan et de distributivité des opérateurs logiques and et or. Donnez une formule de xor à l'aide de and, or et not.
2. Quelle est la table de vérité de M, la fonction booléenne majoritaire de 3 variables a, b, c ? M(a,b,c)=1 lorsqu'au moins 2 variables sont à 1.

2.3 La couche physique

1. Décrire les circuits and, or et xor binaires à l'aide des portes logiques de base (not, nand, nor). Donnez le nombre de transistors utilisés pour chaque circuit.
2. Est-il possible de décrire les circuits and, or et xor binaires à l'aide des seules portes logiques de base nand ? Si oui, dessinez les schémas correspondants et donnez le nombre de transistors utilisés pour chaque circuit.
3. Réaliser un schéma décrivant M par la **méthode des tableaux** et donner le nombre de transistors utilisés (réfléchissez).

Méthode des tableaux :

a) écrire la table de vérité de la fonction n-aire

b) construire dans le schéma une ligne pour chaque variable et une ligne pour son complément.

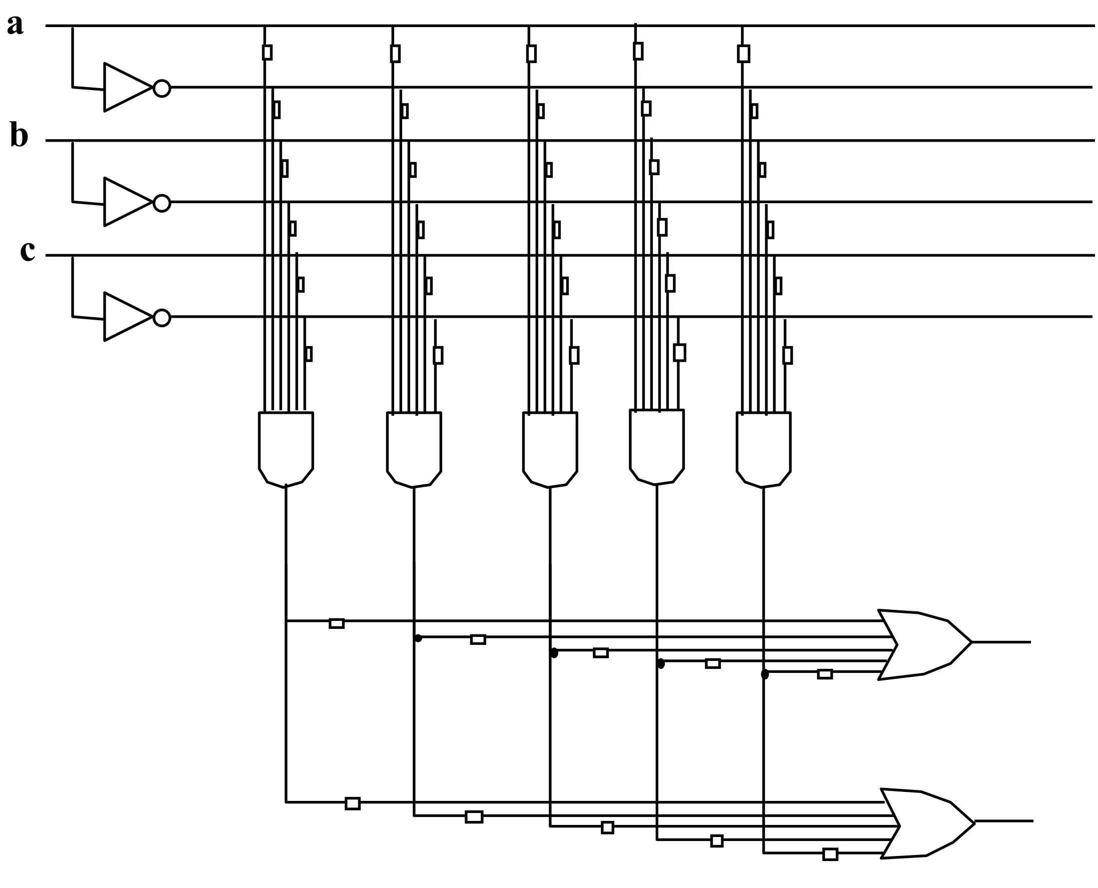
c) Pour chaque résultat=1 de la fonction
  - construire une porte and n-aire
  - y faire entrer les n paramètres du résultat (variables ou leur complément)

d) Relier les sorties de tous ces and vers un or dont la sortie produira le résultat escompté.

4. Décrire la fonction M, majorité de trois variables, vue au II à l'aide de not, de nand et de nor. Donnez le nombre de transistors utilisés pour ce circuit.
5. Trouver un schéma décrivant M et n'utilisant que 9 transistors !

2.4 Circuits combinatoires

1. Vous devez construire un schéma réalisant la fonction M Majorité de 3 variables à l'aide d'un multiplexeur 8 lignes d'entrée, 3 lignes de sélection, 1 ligne de sortie. Indiquez le câblage du Mux.
2. Indiquez une ou plusieurs utilisations des multiplexeurs.
3. Décrire le schéma de l'implantation mémoire centrale de 2 Kquartet adressable grâce à 8 boîtiers mémoire d'1Kb et des circuits combinatoires de votre choix. Dessinez notamment les bus d'adresses et de données.
4. Programmez le PLA suivant afin qu'il fournissent les deux fonctions logiques suivantes : $ab\overline{c} + \overline{b}\,\overline{a}$; $(a+b)(\overline{a}+c)$



3 La couche machine

3.1 Format des instructions

1. Quel est l'intérêt de placer le résultat des opérations arithmétiques et logiques dans l'un des opérandes ?

3.2 Adressages

1. En fonction des valeurs de la MC décrite en a), combien vaut l'accumulateur après chaque opération décrite en b) ?

a) adrs	20	30	40	50
valeur	40	50	30	70

b) LOAD IMMEDIAT 20
LOAD DIRECT 20
BX=50 ;LOAD INDIRECT BX
LOAD INDIRECT 20 (rare)
LOAD IMMEDIAT 40
BX=40 ;LOAD INDEX [BX-10]

2. Décrivez la suite des états de la pile d'octets et de son pointeur, suite à l'exécution des instructions suivantes : LOADIMM SP,100; PUSH 46; PUSH 33; POP R1; PUSH 21; PUSH R1;POP R2;POP R1; PUSH R2; PUSH R1, PUSH SP (opérandes en hexa. et push stocke avant de décrémenter (pas un 8086) !)

3.3 Appels procéduraux

1. Soit le programme et les procédures suivantes :

```
program a;      procedure b;     procedure c;      procedure d;
a1: b;            b1:c;             c1:d;              d1:write('Bonjour');
a2: c;            b2:d;             c2:finprocedure  d2:finprocedure
a3: d;          b3:finprocedure
a4: finprogram
```

Décrivez les états successifs de la pile et de son pointeur à l'aide des adresses a1,a2, …,b1,… lorsque le programme a est appelé avec un sommet de pile initial à 100H (push stocke avant de décrémenter (pas un 8086) !)

2. Soit la procédure suivante :

```
procedure aff(n:entier)
a1:si n≤0 alors
a2:write('Bonjour !') sinon
a3:début aff(n-1);
a4:write('Hello') fin
a5:finprocedure
```

Décrivez les états successifs de la pile et de n à l'aide des adresses ai lorsque la procédure aff est appelée avec n=3. Indiquez également les affichages.

3. Soit la procédure de Hanoï :

```
procedure hanoi(n:entier;D,A:1..3); {attention D?A}
var l:entier
si n>0 alors début
  l:=6-(D+A)
  hanoi(n-1,D,l);
```

```
a1:writeln(n,' : ', D,'-->',A);
  hanoi(n-1,l,A);
a2:fin
finprocedure
```

Décrivez les états successifs de la pile à l'aide des adresses ai lorsque la procédure hanoi est appelée avec n=3,D=1,A=2. Indiquez également les affichages.

3.4 Algorithmes Assembleur

1. Ecrire l'algo. de remise en cohérence d'une addition binaire sur des paramètres en excédent128.
2. Ecrire l'algorithme **optimisé** de multiplication par additions successives de deux entiers naturels codés sur un octet en Binaire Non Signé. Les paramètres sont passés par les registres AH et AL (8 bits) et le résultat est retourné en AX (16 bits). Utilisez un langage algorithmique.

4 Assembleur

4.1 Entrées/Sorties

idée : devant un problème à résoudre en assembleur, écrire un algorithme utilisant la mémoire centrale et les registres du 8086 comme données ainsi que les structures de contrôles :
si … alors … sinon …; pour c=K1 jusqu'à K2 faire … finfaire;
tantque … faire …finfaire; répéter … jusquà …;
cas où … fincas;

Puis réécrire cet algorithme dans une procédure assembleur à l'aide des règles de traduction données en TD. On précisera toujours en commentaire de début de procédure, l'objectif de la procédure, les paramètres d'appel et de résultat, enfin, les registres modifiés dans la procédure.
ex :

```
MAPROC          PROC NEAR
;----------------------------------------------------------
; cette procédure réalise …
; entrée : AX, BX : opérandes de l'opération …
; sortie : CX : …
; modifiés : aucun
;----------------------------------------------------------
…
MAPROC          ENDP
```

Pour les Entrées/Sorties, on supposera, par la suite, qu'il existe deux procédures prédéfinies LITUNCAR (resp. ECRUNCAR) qui permettent de lire sans écho (resp. d'écrire) le caractère ascii tapé au clavier (resp. situé en AL) et de l'affecter à AL (resp. de l'afficher sur l'écran). Ces deux procédures correspondent en fait à des services usuels du système d'exploitation (appels systèmes réalisés par INT). Pour la lecture, seul le registre AL est modifié (résultat) tandis que ECRUNCAR ne modifie aucun registre.

Par la suite, on écrira toujours l'algo. avant la procédure !
1. Ecrire une procédure ENTR1CAR qui permet de lire un caractère ascii au clavier et d'en faire son écho sur l'écran ! (trivial). Cette procédure retourne le car. lu dans AL.
2. Ecrire une procédure qui affiche "GO" puis qui lit un caractère avec écho, puis passe à la ligne suivante pour afficher 10 fois le caractère entré. Cette proc ne modifie aucun registre !
3. Même chose que précédemment sauf que l'on réitère (boucle ENTR1CAR puis 10 ECRUNCAR) tant qu'on n'a pas lu le caractère Z.
4. Même chose que précédemment (3.) sauf qu'un caractère minuscule est affiché 10 fois tandis qu'un caractère majuscule est affiché 20 fois et tout autre caractère 15 fois.

4.2 Ascii et entiers non signés

1. Ecrire une procédure ASCTOBIN qui prend AL comme paramètre d'entrée et de sortie, et qui transforme le caractère ascii AL en l'entier compris entre 0 et 15 (0..0FH) correspondant. Dans le cas où le paramètre d'entrée n'est pas dans {'0', '1', …'9','A', 'B', …'F'} on retournera le code d'erreur 1111 1111 dans AL.
2. Ecrire la procédure BINTOASC qui réalise l'opération inverse en traduisant en ASCII la représentation binaire située sur le quartet le moins significatif d'AL. Attention, il n'y a pas de code d'erreur de retour!

3. Ecrire une procédure qui prend AX comme paramètre d'entrée (2 ascii représentant un code hexa) et AL comme paramètre de sortie, et qui transforme les 2 caractères ascii AH, AL en l'entier compris entre 0 et 255 (0..0FFH) correspondant. Dans le cas où l'un des caractères d'entrée n'est pas dans {'0', '1', …'9','A', 'B', …'F'} on retournera le code d'erreur 1111 1111 dans AH sinon on retournera 0 dans AH.

4.3 Ascii et complément à deux

1. A l'aide des procédures vues ci-dessus, écrire une procédure ECRIC2AX ayant comme paramètre d'entrée AX qui représente un entier signé en complément à deux sur 16 bits. Cette procédure affichera la chaine ascii (correspondant à un entier décimal signé ou non) terminée par un retour à la ligne (ascii OAH).

Ex : AX=0FFFEH <écran> : -2↲
AX=0A9 <écran> : 169↲

4.4 Quelques questions

1. Quelle est la taille maximum du code objet d'une instruction du 8086 ? Donnez un exemple d'instruction assembleur codée en cette taille.
2. Quel est le nombre maximum de segments que le 8086 peut adresser sans modifier de registre ?
3. Si l'instruction TEST n'existait pas, quelle suite d'instructions pourrait la remplacer ? Et pour CMP ?
4. Quelle est la signification de l'indicateur AF ?

4.5 Décalages et Rotations

1. Ecrire l'algorithme puis la procédure proche COMPTE ayant comme paramètre d'entrée BX et comme paramètre résultat CX dont la valeur comptabilise le nombre de bits à 1 de BX. Seul CX peut être modifié.

2. Ecrire l'algorithme puis la procédure en assembleur du 8086 qui renvoit dans AX la taille (en RBNS [0, 16]) de la plus longue suite de 1 **consécutifs** dans le registre paramètre d'entrée AX. Cette procédure ne doit modifier aucun autre registre!

exemples : AX en entrée	AX en sortie
383H ou 0000 0011 1000 0011	3H
0FF61H ou 1111 1111 0110 0001	8H

3. Ecrire l'algorithme puis la procédure en assembleur du 8086 qui renvoit dans AX la valeur 0FFFFH si la configuration binaire de AX, le registre paramètre d'entrée, est symétrique par rapport à son centre, 0H sinon. Cette procédure ne doit modifier aucun autre registre à part AX !

exemples : AX en entrée	AX en sortie
0000 0011 1100 0000	0FFFFH
1111 1001 1001 1111	0FFFFH
1111 1000 0000 1101	0H

4.6 Division

1. Ecrire l'algorithme (si…alors…sinon, tq…) de division entière par **soustractions successives** de deux entiers naturels codés sur 8 bits en RBNS. Utiliser les noms de registres du 8086 comme noms de variables sachant que les paramètres d'entrée de l'algorithme sont AH (dividende) et AL (diviseur), et que les paramètres de sortie sont BH (reste) et BL (quotient). Exemple : 35(AH) div 19(AL)=1(BL) et il reste 16(BH).

En cas de division par 0, retourner 0 dans AX et BX.

2. Ecrire la procédure correspondant à l'algorithme précédent en langage d'assemblage du 8086.

4.7 Mutiplication

1. Ecrire l'algorithme (si…alors…sinon, tq…) de multiplication par décalages et additions, de deux entiers naturels codés sur 8 bits en RBNS. Utiliser les noms de registres du 8086 comme noms de variables sachant que les paramètres d'entrée de l'algorithme sont AH et AL, et que le paramètre de sortie est AX. Rappel : 35*19=35*16+35*2+35*1

2. Ecrire la procédure correspondante en langage d'assemblage du 8086.

3. Ecrire la procédure de multiplication (par décalages et additions) de deux entiers **relatifs** en C2 sur 8 bits.

4.8 Segments

Un source assembleur doit, entre autre, posséder trois définitions de segments :

```
MONCODE SEGMENT
…                           ;1ère instruction a exécuter
MONCODE ENDS

MESDONNE SEGMENT
…                           ;définition des données (DB,DW,…)
MESDONNE ENDS

MAPILE SEGMENT
…                           ; réservation de place pour la pile d'exécution
MAPILE ENDS
```

Dans le segment de données, des directives d'assemblage (ou pseudo-instructions) permettent de réserver de la place et d'initialiser (ou non) des données (nombres, chaînes, tableaux, structures). On rappelle ces directives : DB (Define Byte), DW (Define Word), DD (Define Double) ainsi que les options DUP et ?. Exemples : VARCH DB 'exemple de chaîne\0'
TAB100 DW 100 DUP (0,1,2,3,4) TNONINIT DD 10 DUP (?)

Lors du chargement du programme en mémoire centrale, le registre DS est positionné sur le segment où sont installées les données de MESDONNE tandis que SS et CS pointent sur la pile et le code.

On supposera, par la suite, qu'il existe un appel système (procédure ECCHAINE) permettant d'afficher la chaîne de caractères pointée par DS,SI et qui se termine par un caractère NULL (\0 ascii 0).

1. Ecrire le segment de données définissant 3 chaînes : "Message1", "Message2", "Erreur". Ecrire le segment de code qui lit un caractère au clavier et qui affiche le message 1 (resp. 2) si le caractère lu est '1' (resp. '2'). Pour tout autre caractère on affichera le message d'erreur.

2. Ecrire un programme (segment de données et de code) qui stocke une chaîne initialisée en segment de données et dont le code affiche cette chaîne inversée.

4.9 Passage de paramètres

Vous devez écrire une procédure REPAFF qui affiche n fois une chaîne s. Les paramètres d'entrée n, s sont passés sur la pile par le programme appelant : n est d'abord empilé sur un mot en RBNS puis l'adresse complète (segment, déplacement) de la chaîne est empilée avant que l'appel à REPAFF soit fait. Au retour, vous devrez vider la pile et n'avoir modifié aucun registre.

5 Systèmes de gestion de fichiers

5.1 Fichiers structurés

Un système de fichiers a les caractéristiques suivantes :
1 disque dur 100 Mo utiles pour les fichiers décomposés en blocs de 1 Ko numérotés de 0 à …
Allocation (libre/occupé) par tableau de bits
1 fichier "Client" d'articles de taille fixe = 110 octets

1. Facteur de Blocage du fichier Client et % inutilisé
2. Type de données du tableau de bits en Pascal ou en C en supposant le type bit déjà défini.
3. Occupation disque en Ko du tableau de bits
4. Soit le fichier Client ayant 100 articles compactés : nombre de blocs occupés, volume du fichier et espace disque utilisé.

5.2 Accès Séquentiel

1. En Accès Séquentiel, quel est le nombre moyen d'E/S nécessaires pour accéder à un article quelconque du fichier Client ?

Soit les primitives systèmes suivantes :
booléen **ouvrir**(chaine nomfic, chaine mode); // initialise le pointeur d'article
// avec mode ∈ {"read","write","rw","append"}
booléen **fermer**(chaine nomfic); // retournent vrai si pas de problème
booléen **FinDeFichier**(chaine nomfic); // retourne vrai si on est en EOF
chaine **lire**(chaine nomfic); // retoune l'article courant sous un format chaine
booléen **ecrire**(chaine nomfic, chaine article); // surchage l'article courant

- et la fonction spécifique à un article du fichier Client:

int **extraireTotalCommande**(chaine article); // retourne la valeur totale
// des commandes d'un client (extrait le champ TOTCOM).

2. écrire algorithmiquement la fonction de calcul de la somme globale des commandes de tous les clients.
3. écrire l'algorithme de calcul de la moyenne des commandes clients
4. Le fichier Client étant trié en ordre croissant sur le champ entier NUMCLI, (la fonction int **extraireNumCli**(chaine article); étant donnée), écrire la procédure de recherche et d'affichage du total des commandes d'un client quelconque dont le numéro est saisi au clavier.
5. Un ajout de client s'effectue en fin de fichier (append) et le NUMCLI immédiatement supérieur au dernier article du fichier Client est affecté au nouveau Client. Ecrire l'algorithme d'ajout d'un article en supposant connue la fonction : chaine **construitArticle**(int ncli, int totalCommande); Quel inconvénient a cet algorithme d'ajout ? Avez-vous une idée d'amélioration sans utiliser d'accès direct ?
6. Destruction d'un article Client : avantages et inconvénients du compactage à chaque destruction, périodique, et du non-compactage.

5.3 Accès direct

1. En accès dispersé (haché) peut-on obtenir un fichier trié sur la clé ?
2. Ecrire une fonction de hachage réalisant le modulo MAXHACH sur la somme des codes ASCII d'une chaîne de caractères.

3. Un index linéaire (non hiérarchique) peut-il être non trié ?
4. Quel est l'intérêt d'un index creux primaire par rapport à un index dense secondaire ?

5.4 Allocation

1. Taille maxi d'un fichier Unix (Blocs de 1 Ko, Ad Phys sur 4 octets)
2. Inconvénients du chaînage déporté par rapport à l'adressage direct
3. Ecrire les algorithmes de First-Fit et de Best-Fit pour DemandeBlocs(n) sur un tableau de bits libre/occupé.