ASSEMBLEUR 80X86

A PROPOS
Ce document est un «résumé», car il ne donne que des explications succinctes des thèmes abordés. C'est donc un «pense bête» plutôt qu'un document de formation. En conséquence, si vous êtes novice, vous risquez de ne pas comprendre les concepts qui seraient nouveaux pour vous. Les personnes expérimentées, quant à elles, y trouveront un pense bête clair et concis.

NOUS		
l'auteur Laurent CONSTANTIN «Champ de la Sevrette» 47120 Saint Pierre sur Dropt France	En contrepartie, si vous y trouvez des erreurs, si vous avez des remarques ou si vous décelez des oublis, vous vous engagez à les lui indiquer.	

AVERTISSEMENT
En aucun cas Laurent CONSTANTIN ne pourra être tenu pour responsable des préjudices, de quelque nature que ce soit, pouvant résulter de l'utilisation du contenu de ce document.

1INTRODUCTION:

2PRÉSENTATION:

Sans objet.

3HISTORIQUE:

Sans objet.

4PRÉCISIONS SUR CE DOCUMENT:

L'assembleur 80x86 est un domaine très vaste.
Il y a les instructions spécifiques au 8086, au 80286, 80386, 80486, Pentium, les instructions MMX, ...
Pour ne pas vous embrouiller entre ces différentes versions, je vais faire ce qui suit:

je présente ce qui marche sur un 386

Donc, comme conséquences:

- je ne vais plus parler de numéro de processeur: je ne vous embrouillerai pas
- les programmes que vous ferez à l'aide de cette documentation ne seront pas optimisés pour un 80486 ou 586
- les programmes que vous ferez à l'aide de cette documentation ne marcheront peut être pas sur 8086 ou 286

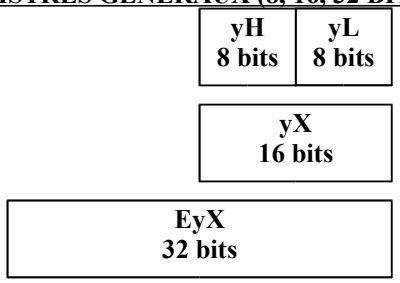
Dernière restriction: à l'intérieur du domaine du 80386, je ne parlerai que de ce qui est le plus important pour faire de petits programmes.

5POURQUOI APPRENDRE L'ASSEMBLEUR ?

"I believe that all programmers should learn the assembly language for the machines on which they work. ... Learning assembly intimately familiarizes the programmer with the hardware on which s/he is working. The more you know about your hardware environment, the better off you are." (source: ASM Magazine)

6LES REGISTRES:

7REGISTRES GÉNÉRAUX (8, 16, 32 BITS)



On a: yX = yH yL et EyX = {16bits} yX

où y = A (Accumulator) utilisé pour des calculs
B (Base) utilisé pour y mettre des adresses
C (Count) utilisé pour les compteurs
D (Data) utilisé pour les données

En fait, on peut faire ce que l'on veut. Par exemple, on peut utiliser AX comme compteur.

8REGISTRES DE SEGMENT (16 BITS)

CS	segment de code	pointe sur l'adresse du début du segment de code (programme)
SS	segment de pile	pointe sur l'adresse du début du segment de la pile
DS	segment de données	pointe sur l'adresse du début du segment des données
ES	segment extra	pointe sur l'adresse actuelle du segment extra
FS et GS		deux registres pour l'utilisateur

9REGISTRES DE DÉPLACEMENT (16, 32 BITS)

(E)BP	pointeur de base	pointe sur l'adresse des paramètres passés à la pile (16 et 32 bits)
(E)SP	pointeur de pile	pointe sur l'adresse actuelle de la pile (16 et 32 bits)
(E)SI	indice de source	pointe dans le segment des données (16 et 32 bits)
(E)DI	indice de destination	pointe dans le segment extra (16 et 32 bits)

10LE POINTEUR D'INSTRUCTION (16, 32 BITS)

(E)IP	pointeur d'instruction	pointe sur la prochaine instruction à exécuter (16 et 32 bits)

11LE REGISTRE DE DRAPEAUX (32 BITS)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
														vm	RF

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	NT	IO 1	IO 2	OF	DI	IN	T	SF	ZF		AC		PF		CF

CF	Carry Flag	retenue
PF	Parity Flag	parité (1 = impair ; 0 = pair)
AC	Auxilary Carry	
ZF	Zero Flag	zéro
SF	Sign Flag	signe (1 = négatif)
T	Trace	pour le débuggage
IN	Interruption	
DI	Direction	pour la direction des instructions sur les chaînes de caractères
OF	OverFlow	dépassement
IO1 IO2	niveau de priorité des I/O	
NT	Nested Task	contrôle IRET
RF	Resume Flag	utilisé avec les registres de débuggage
VM	Virtual Mode	mode virtuel

12LES AUTRES REGISTRES:

- 4 Control Register pour le contrôle de la mémoire paginée, le cache, ...
- 8 registres de débuggage où les débuggueurs vont mettre les points d'arrêts

13LA MÉMOIRE:

Je ne parle que du mode réél qui permet d'adresser 1M de mémoire.

La mémoire d'un PC est segmentée (ce qui permet de clairement séparer nos données, variables, ...).
L'adresse d'un octet en mémoire est définie sur 20 bits par un segment (16 bits) et un offset (16 bits).
Par exemple, avec le segment 29CA et l'offset 5D2A, on a l'adresse 29CA:5D2A qui correspond à:

segment:	0010	1001	1100	1010	- - - -
offset:	- - - -	0101	1101	0010	1010
adresse réelle sur 20 bits	0010	1111	1001	1100	1010

Cette adresse réelle est obtenue par addition (8086) ou par référence à une table (>8086)

Il existe donc 2^{12} manières de représenter une même adresse. Pour normaliser, une adresse a un offset compris entre 0 et F.
Donc 29CA0+5D2A == 2F9CA == 2F9C:000A (adresse normalisée)

14LES MODES D'ADRESSAGE DE LA MÉMOIRE:

Le cas le plus complet (Scaled Based Indexed Plus Displacement) est:

déplacement[base][index*n] <==> [déplacement+base+index*n]

avec:
- n = 1, 2, 4 ou 8
- déplacement = une constante
- base = EyX, yX, (E)BP, (E)SP, (E)SI, (E)DI
- index = EyX, yX, (E)BP, (E)SP, (E)SI, (E)DI

Les modes qui en découlent sont:

déplacement[base][index]	Based Indexed Plus Displacement
déplacement[index*n]	Scaled Indexed
déplacement[index]	Indexed
[base][index*n]	Scaled Based Indexed
[base][index]	Based Indexed
[index*n]	Scaled
[base]	Register Indirect
[déplacement]	Displacement Only

15LES INSTRUCTIONS DE BASE:

16L'AFFECTATION:

MOV	a, b	a := b	
XCHG	a, b	temp := b b := a a := temp	
PUSH	a	pile := a	
POP	a	a := pile	
PUSHA		pile := AX, BX, CX, DX, SP, BP, SI, DI	PUSHAD en 32 bits
POPA		AX, BX, CX, DX, SP, BP, SI, DI := pile	POPAD en 32 bits
PUSHF		pile := registre de drapeaux	PUSHFD en 32 bits
POPF		registre de drapeaux := pile	POPFD en 32 bits

17SAUT:

CMP	a, b	effectue a - b		
JMP	a	saut à l'adresse a		
JO	a	saut à l'adresse a si OF = 1		
JC	a	saut à l'adresse a si CF = 1		
JZ	a	saut à l'adresse a si ZF = 1		
JS	a	saut à l'adresse a si SF = 1		
JP	a	saut à l'adresse a si PF = 1		
JNO	a	saut à l'adresse a si OF = 0		
JNC	a	saut à l'adresse a si CF = 0		
JNZ	a	saut à l'adresse a si ZF = 0		
JNS	a	saut à l'adresse a si SF = 0		
JNP	a	saut à l'adresse a si PF = 0		
JE	a	saut à l'adresse a si a = b	(idem à JZ)	
JNE	a	saut à l'adresse a si a /= b	(idem à JNZ)	
JG	a	saut à l'adresse a si a > b	(idem à JNLE)	(en non signé: JA, JNBE)
JGE	a	saut à l'adresse a si a >= b	(idem à JNL)	(en non signé: JAE, JNB)
JL	a	saut à l'adresse a si a < b	(idem à JNGE)	(en non signé: JB, JNAE)
JLE	a	saut à l'adresse a si a <= b	(idem à JNG)	(en non signé: JBE, JNA)
JCXZ	a	saut à l'adresse a si CX=0		
JECXZ	a	saut à l'adresse a si ECX=0		

18AFFECTATION CONDITIONNELLE:

CMP	a, b	effectue a - b		
SETO	a	(a := 1 si OF = 1)	ou	(a := 0 sinon)
SETC	a	(a := 1 si CF = 1)	ou	(a := 0 sinon)
SETZ	a	(a := 1 si ZF = 1)	ou	(a := 0 sinon)
SETS	a	(a := 1 si SF = 1)	ou	(a := 0 sinon)
SETP	a	(a := 1 si PF = 1)	ou	(a := 0 sinon)
SETNO	a	(a := 1 si OF = 0)	ou	(a := 0 sinon)
SETNC	a	(a := 1 si CF = 0)	ou	(a := 0 sinon)
SETNZ	a	(a := 1 si ZF = 0)	ou	(a := 0 sinon)
SETNS	a	(a := 1 si SF = 0)	ou	(a := 0 sinon)
SETNP	a	(a := 1 si PF = 0)	ou	(a := 0 sinon)
SETE	a	(a := 1 si a = b)	ou	(a := 0 sinon)
SETNE	a	(a := 1 si a /= b)	ou	(a := 0 sinon)
SETG	a	(a := 1 si a > b)	ou	(a := 0 sinon)
SETGE	a	(a := 1 si a >= b)	ou	(a := 0 sinon)
SETL	a	(a := 1 si a < b)	ou	(a := 0 sinon)
SETLE	a	(a := 1 si a <= b)	ou	(a := 0 sinon)

Remarque: il existe aussi SETA, SETAE, SETB, SETBE, SETNA, SETNAE, SETNB, SETNBE, SETNE, SETNG, SETNGE, SETNL, SETNLE, SETPE, SETPO

19LES ENTRÉES SORTIES:

OUT	port, a	port := b	(port entre 0 et FFFF)
IN	a, port	a := port	(port entre 0 et FFFF)

20TRAITER DES ADRESSES:

LEA	a, b	a := adresse de b	
LDS	a, b	DS := segment de b	a := offset de b
LES	a, b	ES := segment de b	a := offset de b
LFS	a, b	FS := segment de b	a := offset de b
LGS	a, b	GS := segment de b	a := offset de b
LSS	a, b	SS := segment de b	a := offset de b

21CALCUL ARITHMÉTIQUE:

ADD	a, b	a := a + b sans retenue			
ADC	a, b	a := a + b avec retenue			
SUB	a, b	a := a - b sans retenue			
SBB	a, b	a := a - b avec retenue			
INC	a	a := a + 1			
DEC	a	a := a - 1			
NEG	a	a := -a			
MUL	a	AX := AL* a			si a est un octet
		DX:AX := AX * a			si a est un mot
IMUL	a	idem à MUL, mais en signé			
DIV	a	AL := AX / a	AH := reste		si a est un octet
		AX := DX:AX / a	DX := reste		si a est un mot
IDIV	a	idem à DIV, mais en signé			

22OPÉRATIONS LOGIQUES:

AND	a, b	a := a et b
OR	a, b	a := a ou b
XOR	a, b	a := a xor b
NOT	a	a := complément(a)

23TRAITER DES BITS:

BSF	a, b	a := position du premier "1" de b en partant de droite (ZF := 1 si pas trouvé)	
BRF	a, b	a := position du premier "1" de b en partant de gauche (ZF := 0 si trouvé)	
BT	a, b	CF := a[b]	
BTC	a, b	CF := a[b]	a[b] := complément(a[b])
BTR	a, b	CF := a[b]	a[b] := 0
BTS	a, b	CF := a[b]	a[b] := 1

24LES FONCTIONS:

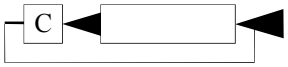
CALL	a	appel de la fonction située à l'adresse a
RET		retour de la fonction

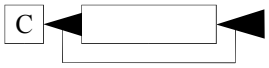
25LES INTERRUPTIONS:

INT	a	appel de l 'interruption (DOS ou BIOS) a
IRET		retour de l 'interruption

26LES ROTATIONS:

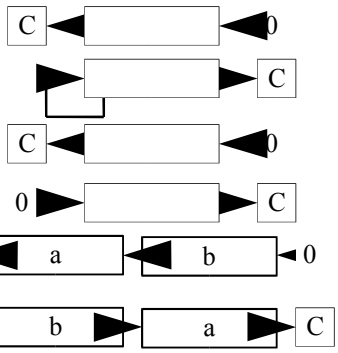
RCL	a, b	rotation à gauche de b bits de a en passant par CF
RCR	a, b	rotation à droite de b bits de a en passant par CF
ROL	a, b	rotation à gauche de b bits de a sans passer par CF
ROR	a, b	rotation à droite de b bits de a sans passer par CF





27LES DÉPLACEMENTS:

SAL	a, b	déplacement à gauche de b bits de a
SAR	a, b	déplacement à droite de b bits de a
SHL	a, b	déplacement à gauche de b bits de a
SHR	a, b	déplacement à droite de b bits de a
SHLD	a, b, d	déplacement à gauche de d bits (affecte a mais ne modifie pas b)
SHRD	a, b, d	déplacement à droite de d bits (affecte a mais ne modifie pas b)



28LES DRAPEAUX:

STC	CF := 1
STD	DF := 1
STI	IF := 1
CLC	CF := 0
CMC	CF := complément(CF)
CLI	IF := 0

29DIVERS:

NOP	pas d'opération

30ATTENTION...

On ne peut pas utiliser tous les types d'opérandes n'importe où. Seuls sont autorisés:

	2 opérandes	1 opérande
avec tous les opérateurs	R, R R, I R, M M, R M, I	R M
pour MOV, on a aussi	R, S M, S S, R S, M	
pour PUSH et POP, on a		S

S un registre de segment (CS, DS, ES, SS)
R un registre ordinaire (EyX, yX, yH, yL, SI, DI, BP, SP)
I un immédiat (78h, 54, ...)
M un des modes d'adressage de la mémoire

31LES INSTRUCTIONS PLUS ÉVOLUÉES:

32LES BOUCLES:

```
                MOV     CX, 100                 ; boucle de 100
eti:
                ici, passe 100 fois
                LOOP    eti             ; décrémente CX et boucle si CX /= 0
```

Remarque: *LOOPZ=LOOPE=branchement tant que CX/= et ZF=1*
LOOPNZ= branchement tant que CX/= et ZF=0

33INSTRUCTIONS SUR LES CHAINES DE CARACTERE:

nom	explication	exemple
STD	set direction_flag	STD
CLD	clear direction_flag	CLD
REP xx	répète CX fois l'instruction xx (décrémente CX à chaque passage)	MOV CX, 7 REP MOVSB ; exécute movsb 7 fois
REPZ xx	répète CX fois l'instruction xx et tant que le flag vaut zéro	MOV CX, 7 REPZ CMPSB ; répète 7 fois maximum

copie de string:
 	
MOVSB
MOVSW
MOVSD
	ES:[DI] := DS:[SI]
	if direction_flag = 0 then
	SI := SI + size; (size = B, W ou D)
	DI := DI + size;
	else
	SI := SI - size;
	DI := DI - size;
	endif;
		String1 DB "Coucou$"
		String2 DB 7 DUP(0)
		(penser à affecter ES et DS. Exemple:
		PUSH CS ; POP DS ; POP ES)
		MOV SI, OFFSET String1
		MOV DI, OFFSET String2
		CLD
		MOV CX, 7 ; longueur à copier
		REP MOVSB ; String2 := String1

compare strings
 	
CMPSB
CMPSW
CMPSD
	CMP DS:[SI], ES:[DI]
	if direction_flag = 0 then
	SI := SI + size; (size = B, W ou D)
	DI := DI + size;
	else
	SI := SI - size;
	DI := DI - size;
	endif;
		... (2 DB, 1 PUSH, 2 POP, 2 OFFSET)
		CLD
		MOV CX, 7 ; longueur à comparer
		REPZ CMPSB
		JNZ pas_egal
		...
		pas_egal:
		...

Ax := DS:SI
 	
LODSB
LODSW
LODSD
	EAX/AX/AL := DS:[SI]
	if direction_flag = 0 then
	SI := SI + size; (size = B, W ou D)
	else
	SI := SI - size;
	endif;
		... (2 DB, 1 PUSH, 2 POP, 2 OFFSET)
		CLD
		MOV CX, 7 ; longueur du string
		MOV BH, 0 ; page vidéo 0
		MOV AH, 0Eh ; fonction écrire
		autre_carac:
		LODSB ; AL := carac suivant
		INT 10h
		LOOP autre_carac

ES:DI := Ax
 	
STOSB
STOSW
STOSD
	ES:[DI] := EAX/AX/AL
	if direction_flag = 0 then
	DI := DI + size; (size = B, W ou D)
	else
	DI := DI - size;
	endif;	

cherche Ax dans string ES:DI
SCASB
SCASW
SCASD
	CMP EAX/AX/AL, ES:[DI]
	if direction_flag = 0 then
	  DI := DI + size; (size = B, W ou D)
	else
	  DI := DI - size;
	endif;
		... (1 DB, 1 PUSH, 1 POP, 1 OFFSET)
		CLD
		MOV CX, 7 ; longueur du string
		MOV AL, 'H' ; on cherche H
		REPNE SCASB
		JNZ pas_trouve

lecture d'une valeur dans le port spécifié dans DX
 	
INS
	ES:[DI] := port(DX)
	if direction_flag = 0 then
	  DI := DI + size; (size = B, W ou D)
	else
	  DI := DI - size;
	endif;	

écriture dans le port spécifié dans DX
 	
OUTS
	port(DX) := DS:[SI]
	if direction_flag = 0 then
	  SI := SI + size; (size = B, W ou D)
	else
	  SI := SI - size;
	endif;	

34LES CONVERSIONS:

```
CBW       ; AX := AL "étendu" (converti en Byte en Word)
CWD       ; DX:AX := AX "étendu" (converti Word en Double)
CWDE      ; EAX := AX "étendu" (converti Word en DoubleEtendu)
CDQ       ; EDX:EAX := EAX "étendu" (converti Double en Quadruple)
MOVSX     a, b   ; a := b étendu (a doit être 2 fois plus grand que b)
MOVZX     a, b   ; a := b étendu (a doit être 2 fois plus grand que b) en non signé
```

35DEUX AUTRES TYPES:

BCD: 0..9 codé sur 4 bits (16)
ASCII: 0..9 codé sur 8 bits (256)
On peut faire des calculs puis convertir les résultats dans ces formats:

	addition	soustraction	multiplication	division
BCD:	ADD a, b DAA (convertit a)	SUB a, b DAS (convertit a)		
ASCII:	ADD a, b AAA (convertit a)	SUB a, b AAS (convertit a)	MUL a, b AAM (convertit a)	AAD (convertit a) DIV a, b

36CE QUE JE NE DÉTAILLERAI PAS:

Spécifique au mode protégé:

ARPL	ajuste le niveau du privilège
CLTS	mets à 0 l'indicateur de changement de contexte Task-Switch-Flag
IBTS	insère une chaîne de bits
LAR	charge le droit d'accès
LGDT	charge le registre de la table des descripteurs globaux
LIDT	charge le registre de la table des descripteurs d'interruption
LMSW	charge le mot d'état de la machine
LSL	charge une limite de segment
LTR	charge le registre de tâche
SGDT	sauvegarde le registre de la table des descripteurs globaux
SIDT	sauvegarde le registre de la table des interruptions
SLDT	sauvegarde le registre de la table des descripteurs locaux
SMSW	sauvegarde le mot d'état de la machine
STR	sauvegarde le registre de tâche
VERR	teste l'autorisation de lecture d'un segment
VERW	teste l'autorisation d'écriture dans un segment
XBTS	prends une chaîne de bits

Divers:

BOUND	teste un nombre par rapport à des limites
BSWAP	conversion d'un registre en un format INTEL
CMPXCHG	compare l'accumulateur avec AL, AX, ou EAX
ENTER	construit un cadre de pile pour une procédure de haut niveau
ESC	accès pour le coprocesseur
HLT	arrêt du processeur en attente d'un évènement externe
INTO	active l'interruption 4 si l'indicateur OF est armé
INBD	efface le contenu de la mémoire cache du 486
INVLPG	exploitation du 486 en mode virtuel. multiprocesseur 8088
IRETD	retour d'interruption depuis un segment de 32bits
LAHF	charge en AH la partie basse du registre des indicateurs
LEAVE	libère le cadre de pile, installé par ENTER
LLDT	charge le registre de la table des descripteurs locaux
LOCK	verrouille le bus
SAHF	copie AH dans la partie basse du registre des indicateurs
STIOSB	transfère octet par octet le contenu de AL en ES:DI
STIOSW	transfère mot par mot le contenu de AL en ES:DI
STOSD	transfère double mot par double mot le contenu de EAX en ES:DI
WAIT	attends que la ligne BUSY ne soit plus active
XBINVD	efface le contenu de la mémoire cache du 486
XLAT	charge en AL l'octet de la table DS:BX+AL

37LES INTERRUPTIONS:

38PRINCIPE D'UTILISATION:

- on affecte les registres généraux:
  AX détermine la fonction choisie
  BX, CX, DX déterminent les paramètres
- on appelle l'interruption (INT xx) qui réalise la fonction
- cette interruption affecte les registres généraux
- on lit le contenu des registres généraux pour connaître le résultat de l'interruption.

Les interruptions principales sont:

10 (interruption BIOS) entrées/sorties vidéo
16 (interruption BIOS) entrées/sorties clavier
17 (interruption BIOS) entrées/sorties imprimante
21 (interruption DOS) fonctions DOS

39COMMENT SE DÉROULE UNE INTERRUPTION:

Quand le CPU détecte un signal d'interruption:

- il arrête l'activité courante
- il fait PUSHF PUSH CS PUSH IP
- il regarde à l'adresse 0000:(0000 + 4*numéro_int_demandé): c'est la table d'interruption
- il y trouve le vecteur d'interruption du numéro_int_demandé
- le vecteur d'interruption contient l'adresse de la fonction que l'on va exécuter
- quand il a fini d'exécuter cette fonction, il fait POP IP POP CS POPF
- il reprend l'activité courante

Il y a différents niveaux de priorité pour les interruptions: 0 à 255. Plus le numéro est faible, plus sa priorité est importante: donc une int_6 (opérande invalide) peut être interrompue par une int_0 (division par 0)

40LES INTERRUPTIONS LES PLUS IMPORTANTES:

Remarque: la première colonne des tableaux contient la valeurde AH.

INT 05H:

réaliser une copie d'écran sur LPT1

INT 10H:

00	*changer le mode vidéo* AL mode voulu	AL code du mode obtenu
01	*définir l'aspect du curseur* CH ligne de départ du curseur CL ligne de fin du curseur	
02	*positionner le curseur* BH page écran DH ligne DL colonne	
03	*savoir la position et l'aspect du curseur* BH page écran	DH ligne écran DL colonne écran CH ligne début du curseur CL ligne fin du curseur
05	*choisir la page écran courante* AL page écran	
06 07	*défiler la zone écran vers le haut* *défiler la zone écran vers le bas* AL nombre de ligne de défilement CH ligne du coin supérieur gauche CL colonne du coin supérieur gauche DH ligne du coin supérieur droit DL colonne du coin supérieur droit BH couleur des lignes vides	
08	*lire le caractère et la couleur (po. courante)* BH page écran	AL code ASCII du caractère AH couleur
09	*écrire à la position du curseur* BH page écran CX nombre de répétitions AL code ASCII BL couleur	
0A	*écrire sans changer la couleur* BH page écran CX nombre de répétitions AL code ASCII	
0B	*choisir la couleur du fond et du cadre* BH 00 BL couleur	
0F	*déterminer le mode vidéo*	AL mode vidéo AH nombre de caractères par ligne BH numéro de la page écran en cours
10	*changer la couleur du cadre* AL 01 BH couleur	

INT 15H:

84	*connaître l'état des boutons du joystick*	
	DX 00	FC adaptateur présent si 0
		AL état
		bit 7: bouton 1 du joystick 1
		bit 6: bouton 2 du joystick 1
		bit 5: bouton 1 du joystick 2
		bit 4: bouton 2 du joystick 2
84	connaître la position du joystick	
	DX 01	FC adaptateur présent si 0
		AX position X du joystick 1
		BX position Y du joystick 1
		CX position X du joystick 2
		DX position Y du joystick 2

INT 17H:

00	*impression*	
	AL code ASCII du caractère	AH état de l'imprimante
	DX imprimante	bit 0: time out
		bit 3: erreur de transmission
		bit 4: online
		bit 5: plus de papier
		bit 6: accusé de reception
		bit 7: imprimante occupée
01	*initialiser l'imprimante*	
	DX imprimante	résultat idem à ceux de la fonction 00
02	*tester l'état de l'imprimante*	
	DX imprimante	résultat idem à ceux de la fonction 00

INT 21H:

0E	*sélection du lecteur courant*	
	DL numéro du lecteur (A=0 ; B=1, ...)	AL nombre LASTDRIVE dans config.sys
19	*connaître le numéro du lecteur courant*	
		AL numéro du lecteur
1C	*connaître le type d'un lecteur*	
	DL numéro du lecteur	AL nombre de secteurs par cluster
		DX nombre de clusters
		DS:BX pointe sur l'octet d'identification
		F9 disquette 1M2 ou 720k
		F8 disque dur
		F0 disquette 1M44
30	*connaître le numéro de la version de DOS*	
		AL numéro de version principal
		AH sous version
43	*déterminer l'attribut d'un fichier*	
	AL 00	CF résultat valide si vaut 0
	DS adresse de segment de nom de fichier	CX attribut
	DX adresse d'offset du nom de fichier	
43	*fixer l'attribut d'un fichier*	
	AL 01	CF changé si vaut 0
	DS adresse de segment de nom de fichier	
	DX adresse d'offset du nom de fichier	
	CX attribut	

41INT 21H:

Généralités:

Le nom ASCIIZ est le nom du fichier suivi de '0' (et non '$'). Ex: DB "C:\test.txt",0
Pour de petits programmes: segment:offset <==> adresse (car on n'utilise que l'offset)
Un "handle" de 0 correspond au clavier et un "handle" de 1 correspond à l'écran.
On va utiliser les fonctions de l'interruption DOS 21.

3D	*ouvrir un fichier*	
	AL 0 read only; 1 write ; 2 read/write	si CF = 0 OK et AX := handle
	DS:DX segment:offset du nom ASCIIZ	si CF = 1 erreur
3E	*fermer un fichier*	
	BX handle du fichier à fermer	si CF = 0, le fichier est fermé
		si CF =1 erreur
3F	*lire dans un fichier*	
40	écrire dans un fichier	
	BX handle du fichier	si CF = 0 fonction réalisée et
	CX nombre d'octets à lire / écrire	AX := nombre d'octets lus / écrits
	DS:DX segment:offset d'un buffer	si CF = 1 erreur
3C	*créér un fichier (si existe, l'écrase)*	
	CL attribut	si CF = 0 OK et AX := handle
	DS:DX segment:offset du nom ASCIIZ	si CF = 1 erreur
41	*effacer un fichier*	
	DS:DX segment:offset du nom ASCIIZ	si CF = 0, le fichier est effacé
		si CF =1 erreur

INT 33H:

00	*réinitialisation du gestionnaire de souris*	AX=0 si pas de gestionnaire ; =FFFF si OK
01	*afficher le curseur de la souris*	
02	*masquer le curseur de la souris*	
03	*connaître la position et l'état de la souris*	BX état des boutons bit 0 bouton gauche bit 1 bouton droit bit 2 bouton central CX position X DX position Y
04	*positionner a souris* CX position X DX position Y	
05 06	*nombre d'appui sur le bouton* *nombre de relâchement du bouton* BX bouton dont on veut l'information bit 0 bouton gauche bit 1 bouton droit bit 2 bouton central	AX état des boutons bit 0 bouton gauche bit 1 bouton droit bit 2 bouton central BX nombre d'*appuis/relâchement* CX position X au dernier appel DX position Y au dernier appel
07 08	*fixer les coordonnées X mini/maximales* *fixer les coordonnées Y mini/maximales* CX position X/Y minimale DX position X/Y maximale	
1A	*fixer la sensibilité de la souris* BX nb de mickeys représentant 8 pts hori. CX nb de mickeys représentant 8 pts verti. DX seuil pour le doublement de la vitesse	
1B	*connaître la sensibilité de la souris*	BX nb de mickeys représentant 8 pts hori. CX nb de mickeys représentant 8 pts verti. DX seuil pour le doublement de la vitesse
1C	*fixer la fréquence d'interruption* BX fréquence bit 0 pas d'interruption bit 1 30 interruptions par seconde bit 2 50 interruptions par seconde bit 3 100 interruptions par seconde bit 4 200 interruptions par seconde	

Allez lire le paragraphe "Portions de programme" qui contient des exemples d'application de certaines de ces fonctions ainsi que d'autres interruptions.

42DEBUG:

Debug est un programme de permettant de réaliser facilement de petits programmes en assembleur.

43LES PARAMÈTRES DES COMMANDES:

nom	*exemple*	*explication*
adresse	CS:08	segment et offset
	100	offset dans le segment courant
arguments	/A:O	les paramètres que l'on passe au fichier que l'on teste
liste	00 AB D9	octets séparés par un espace
	"Bonjour"	texte ASCII
nombre	5	nombre en hexadécimal
fichier	TEST.COM	nom de fichier
intervalle	20 32	intervalle de 20 à 32
	103L24	intervalle de 103 à 127 (123+24)
	0L0	intervalle de 0 à 64000 (ceci est un cas particulier)

44LES COMMANDES:

Les paramètres entre {} sont facultatifs.

Ecrire:

A {adresse}	(Assembler) écrire de l'assembleur à partir de adresse (écrire une ligne blanche pour terminer)
E adresse {liste}	(Enter) mettre liste en mémoire, à partir de adresse (si on met un espace, on ne change pas; avec '-', on revient en arrière)
F intervalle liste	(Fill) remplir l'intervalle par liste (si liste vaut FE45, on va mettre: FE45FE...)

Lire:

D {intervalle}	(Dump) voir le contenu de l'intervalle (sous forme de texte et hexadécimale)
D {adresse}	(Dump) voir le contenu à partir de adresse (sous forme de texte et hexadécimale)
U {intervalle}	(Unassemble) voir le contenu de l'intervalle (en assembleur)
U {adresse}	(Unassemble) voir le contenu à partir de adresse (en assembleur)

Travailler sur des blocs:

C intervalle adresse	(Compare) comparer le contenu de l'intervalle et le contenu de ce qui commence à adresse
M intervalle adresse	(Move) copier l'intervalle vers la mémoire à partir de adresse
S intervalle liste	(Shearch) recherche liste dans l'intervalle

Travailler avec les ports:

I nombre	(Input) lire le port 'nombre' (série = 03F8, 02F8 ; parallèle = 0378)
O nombre 5F	(Output) écrire la valeur 5F sur le port nombre

Lecture et écriture dans un fichier:

N {fichier} {arguments}	(Name) définir le fichier avec lequel on travaille (et ses paramètres)
L {adresse}	(Load) charger le fichier (définit par le dernier N) à adresse
L	recharger le fichier courant
W {adresse}	(Write) sauver le fichier à partir de adresse (taille à sauver est dans CX)

Débugguage:

G {=adresse} {adresses}	(Go) exécuter à partir de adresse (les points d'arrêt sont en 'adresses')
T {=adresse} {nombre}	(Trace) avancer de 'nombre' pas
P {=adresse} {nombre}	(Proceed) avancer de 'nombre' pas (ne rentre pas dans les calls ou int)
R {registre}	(Register) voir le contenu du registre (doit être en 16 bits: yX)

Divers:

?	aide
H nombre1 nombre2	(Hexa) donne la somme et la différence de nombre1 et nombre2
Q	(Quit) quitter

45EMPLOI DE DEBUG POUR FAIRE UN DUMP DE LA MÉMOIRE:

Si le fichier ENTREE.DAT contient:

```
u 4C32 4C56
q
```

La commande:

```
DEBUG fich.com < ENTREE.DAT > RESULTAT.TXT
```

Va mettre le contenu (sous forme hexadécimale et texte) de l'intervalle 4C32 4C56 dans le fichier resultat.txt.

46UTILISATION DE DEBUG POUR COMPILER DE L'ASSEMBLEUR:

On créé d'abord un fichier (ex: fichier.asm) contenant le code assembleur suivant:

```
A               100             pour commencer à l'adresse 100h
...                             coeur du programme
MOV   AH, 4C                    dans la majorité des programmes, on retourne au DOS...
INT   21                        ... ces deux lignes permettent de retourner normalement au DOS
                                on laisse une ligne blanche
N               BELL.COM        nom du fichier dans lequel on va sauver les lignes en gras
R               CX              pour affecter CX avec...
i                               ... i où i est la taille du programme (l'adresse de la ligne blanche - 100h)
W                               pour sauvegarder dans bell.com la longueur contenue dans CX
Q                               pour sortir de DEBUG
```

Ensuite, on appelle DEBUG ainsi:

```
DEBUG <   fichier.asm
```

Et, DEBUG créé le fichier BELL.COM.

Remarque: seulement les lignes en gras sont de l'assembleur, le reste sont des directives pour le programme débug (passées grâce à la redirection <).

47RÉALISER UN FORMATAGE BAS NIVEAU AVEC DEBUG:

Si notre PC n'a pas un disque IDE et si on n'a pas de manière plus élégante de le faire, on peut formater le disque à bas niveau avec:

```
G C800:0
G C800:5
G C800:CCC
```

48MASM:

Masm est le compilateur de Microsoft.

49UTILISATION DE MASM:

compiler: masm fich.asm linker: link /Ffich.exe fich1.obj fich2.obj

50STRUCTURE D'UN PROGRAMME .COM:

```
COMMENT |
ici des commentaires sur plusieurs lignes
        |
                                ; commentaire

        .XLIST                  ; début de la zone n'apparaissant pas dans le .LST
        NAME        nom
        TITLE       titre
        SUBTTL      sous-titre
        PAGE  66,132            ; pour imprimer à partir de MASM, 66 lignes/132 colonnes
        INCLUDE     fichier     ; inclut le fichier
        .LIST                   ; fin de la zone n'apparaissant pas dans le .LST

        ici les constantes (byte ou word, mais pas double)
masection   SEGMENT     PUBLIC
        ASSUME      CS:masection, DS:masection
        ORG         100h
pgm:
        JMP     debut
ici, les variables(elles sont définies par leur adresse)
debut:
        ici, le coeur du programme
masection   ENDS
        END     pgm
```

51LES CONSTANTES:

```
CST0    EQU     91              ; CST0 := 91 en décimal
CST1    EQU     0ABCDh          ; CST1 := ABCD en hexadécimal (0 obligatoire pour faire la
                                  différence entre un chiffre et une variable commençant par
a)
CST2    EQU     01001110B       ; CST2 := 01001110 en binaire
CST3    EQU     'a'             ; CST3 := code ASCII de 'a'
CST4    EQU     OFFSET av0      ; CST4 := OFFSET de la variable v0
CST5    EQU     (CST2 * 2) + 1  ; CST5 := résultat d'un calcul contenant des constantes
```

52LES VARIABLES:

```
av0     DB      91              ; [av0] := 91 en décimal (Byte)
av1     DW      0ABCDh          ; [av1] := ABCD en hexadécimal (Word)
av2     DB      01001110B       ; [av2] := 01001110 en binaire (Byte)
av3     DB      'a'             ; [av3] := code ASCII de 'a'
av4     DW      OFFSET av0      ; [av4] := OFFSET de la variable "v0"
av5     DB      "Coucou",'$'    ; [av5] := C   [av5+1] := o ...
av6     DD      ?               ; [av6] est réservée mais non initialisée (double)
av7     DB      6 DUP (23h)     ; [av7] := 23   [av7+1] := 23 ... (6 fois)
atab    DB      "AA", 0 , "BB", 0
        DB      "CC", 0, 0      ; tableau de strings
CST8    EQU     $ - OFFSET atab ; CST8 := longueur du tableau
```

53LES OPÉRANDES:

```
On peut utiliser:
BYTE PTR DS:[0487H]          ; [DS:0487]      (sert quand ambiguïté: MOV [avar],3 car
WORD PTR DS:[0487H]          ; [DS:0487]      ... on ne sait pas si 3 est Byte ou Word)
OFFSET      avar                      ; OFFSET(DS:avar) <==> avar
OFFSET      aetiquette                ; OFFSET(CS:aetiquette) <==> aetiquette
SEG     avar                  ; SEGMENT(DS:avar) <==> DS
```

54LE COEUR DU PROGRAMME:

```
debut:
                XOR     AX,AX
                MOV     DS,AX
                MOV     AH,12h
                INT     10h             ; penser au h de Hexadécimal
                TEST    AX, CX          ; effectue AX AND CX pour la comparaison
                JNE     pas_egal                ; saut a l'étiquette
pas_egal:
                CALL    proce           ; appel de la procédure
                ...                     ; ne pas oublier le MOV AX,4C00h; INT 21h

proce   PROC    NEAR            ; une procédure
                ...
                CALL    autre_p
                RET
proce   ENDP

autre_p PROC    NEAR            ; une autre procédure
                ...
                JMP     pas_egal                ; pas dans les règles de l'art, mais ça marche
autre_p ENDP
```

55LE ASSUME:

```
        ASSUME      CS:masection, DS:masection      ; dit que CS et DS sont utilisables

        XOR     AX,AX                               ; AX := 0
        MOV     DS,AX                               ; DS := 0
        ASSUME      DS:NOTHING                      ; n'utilise pas DS

        PUSH    CS      ; pile := CS
        POP     DS      ; DS := pile
        ASSUME      DS:masection                    ; peut de nouveau utiliser DS
```

56LES MACROS:

Comparaison avec les procédures:

Une procedure met, puis récupère, l'adresse de retour dans la pile.
Une macro est spécifique à MASM et (comme le define en C) est équivalente à une portion de programme que l'on vient placer à chaque appel (comme en C, mettre des parenthèses autour des paramètres dans le cas des ambiguïtés: (para)*2 (sinon si para vaut 2+4, on va faire: 2+4*2= 10 /= (2+4)*2) = 12).
Une macro est plus rapide à exécuter car on n'a pas de push et de pop dans la pile.
Une macro prend plus de place car MASM fait la copie de la portion de programme autant de fois que la macro est appelée.

Déclaration:

```
nomm    MACRO           para1, para2    ;; paramètres (deux ; sinon le commentaire est aussi
                                        ;; copié dans la programme)
        LOCAL var1, var2, eti           ;; variables et étiquettes locales
```

```
            ADD     BX, para2
            MOV     AX, para1
            INT     10h
            IFDIF   <para1>, <0>            ;; si para1 /= 0
                    var1 = 01h              ;; on affecte var1
            ELSE
                    var2 = 01h
            ENDIF
            IFNB    <para2>                         ;; si on a passé le paramètre para2
                    ...
            ELSE
              IFIDN         <para1>,<0>     ;; si para1 vaut 0
                    MOV AX, 0600h + (   - (para1) AND 0FFh    )
              ENDIF
            ENDIF
            IFDEF   var1                    ;; si var1 a été affectée
                    ...
            ENDIF
    eti:    ...
            jc      eti
            ENDM
```

Utilisation:

```
            nomm    10, 67                  ; avec 2 paramètres
            nomm    8Dh                     ; avec 1 paramètre
            nomm                            ; sans paramètre
```

Les tests:

Remarque: les délimiteurs de chaines de caractères sont < >

```
            IFNB    <para2>                         si on a passé le paramètre para2    (nomm 56, 3Eh)
            IFDEF   para2                   si on a passé le paramètre para2    (nomm 56, 3Eh)
            IFB     <para2>                         si para2 n'est pas passé (if blank)  (nomm 56)
            IF      para1 LE 0              si para1 <= 0       (existe aussi: EQ, ...)
            IFDIF   <para1>, <var1>         si para1 /= var1
            IFIDN   <para1>,<0>             si para1 == 0
            IFDIFI  <para1>, <var1>         si para1 /= var1    (ignore la différence Maj/minuscule)
            IFIDNI  <para1>,<var2>          si para1 == var2    (ignore la différence Maj/minuscule)
            IFDEF   var1                    si var1 a été affectée
            ...                             idem avec les variables
```

Petit exemple:

```
    ecritcar MACRO          ch1,ch2,ch3,ch4,ch5,ch6,ch7,ch8,ch9,ch10
                    mov     ah,02h
                    IRP     char,<ch1,ch2,ch3,ch4,ch5,ch6,ch7,ch8,ch9,ch10>
                      IFB   <char>
                            EXITM
                      ENDIF
                      mov   dl,char
                      int   21h
                    ENDM            ; termine le IRP qui est une macro de répétition
                    ENDM            ; termine ecritcar
```

Utilisation:

```
            ecritcar  'a', 'e', 'd'         ; écrit "aed"
```

57 PORTIONS DE PROGRAMMES:

58L'ÉCRAN:

Ecrire une chaîne:

```
    @100:   JMP             10B             pour sauter la zone de données
    @102:   DB              'coucou',A,D,'$'  on va écrire coucou et sauter une ligne (A=010 D=013)
```

```
@10B:  MOV   DX, 102          met l'adresse de la chaîne de caractères dans DX
       MOV   AH, 9            sélection de la fonction 9
       INT   21               appel de la fonction 9 de l'interruption DOS 21
```

Effacer l'ecran:

```
MOV   AX, 0600h
MOV   BH, 23h                 couleur
XOR   CX, CX
MOV   DX, 184Fh
INT   10h                     fait un CLS
```

Ecrire en utilisant la mémoire vidéo:

couleur = { 0=noir ; 1=bleu ; 2=vert ; 3=cyan ; 4=rouge ; 5=magenta ; 6=marron ; 7=blanc }
couleur_fond = couleur + 8 (si on veut en clignotant)
couleur_texte = couleur + 8 (si on veut en surbrillant)
*l'attribut_couleur = couleur_fond * 16 + couleur_texte*

```
       MOV   AX, 0B800h       ; segment du buffer vidéo
       MOV   ES, AX           ; ES := B800
       MOV   DI, 80           ; va en haut et milieu  (2*40)
       MOV   AH, 02h          ; écrit avec l'attribut couleur vert
       MOV   AL, 'I'          ; écrit la lettre I
       MOV   ES:[DI], AX      ; écrit
```

59LE SYSTÈME:

Voir les paramètres passés au programme:

```
MOV   BX, 80              car les paramètres commencent à l'adresse 81 du PSP
INC   BX                  on incrémente BX (cette ligne est à l'adresse 103)
MOV   DL, BYTE[BX]        on met dans DL l'octet présent à l'adresse BX
MOV   AH, 2               sélection de la fonction 2
INT   21                  appel de la fonction 2 de l'interruption DOS 21: affichage à l'écran
CMP   DL, D               on compare DL à Dh donc à 013 en ASCII, code de entrée
JNE   103                 si on n'est pas en fin de ligne, on continue à incrémenter BX
```

Retourner un code d'erreur à DOS:

```
MOV   AL, D4              retourne l'erreur D4h = errorlevel 212
MOV   AH, 4C
INT   21                  on retourne sous DOS
```

Rebooter l'ordinateur à chaud: ne marche pas dans une fenêtre DOS

```
MOV   BX, 40
MOV   DS, BX          DS reçoit 40
MOV   BX, 1234
MOV   [72], BX        0040:0072 := 1234 (on n'effectue pas de test de mémoire; 0 = on l'effectue)
JMP   FFFF:0000       on exécute un boot
```

Rebooter l'ordinateur à froid: ne marche pas dans une fenêtre DOS

```
JMP   FFFF:0000       on exécute un boot
```

Planter l'ordinateur:

```
JMP   100             on fait une boucle infinie          (code 0235  0254)
```

60LE BUZZER:

Emettre un bip:

```
MOV   DL, 7        le code ASCII du carac est 7 (donc, un bip sonore)
MOV   AH, 2        sélection de la fonction 2
INT   21           appel de la fonction 2 de l'interruption DOS 21
```

Emettre une fréquence:

On utilise le Programmable Interrupt Timer 8253 qui a une fréquence d'horloge de 1193181 Hertz. 1,19Mhz/FFFF=18,2 et 1193181=12:34DD.

```
MOV   BX, 0202     BX reçoit la fréquence en Hz
MOV   AX, 34DD     prépare pour la division
MOV   DX, 0012     prépare pour la division
DIV   BX           AX reçoit DX:AX / BX
MOV   BX, AX       BX reçoit la fréquence compréhensible par l'ordinateur
IN    AL, 61       on désactive la vérification de parité de la RAM (gagne en vitesse)...
OR    AL, 03       ..., on active le buzzer et...
OUT   61, AL       ...on associe le compteur 2 au buzzer
MOV   AL, B6       on sélectionne le compteur 2, en binaire, pour générer un signal carré...
OUT   43, AL       ...et on va envoyer le LSB puis le MSB (tout ça est contenu dans B6 !)
MOV   AL, BL       on envoie ...
OUT   42, AL       ...le LSB de BX au compteur 2 (port 42)...
MOV   AL, BH       ...puis ...
OUT   42, AL       ...le MSB de BX au compteur 2 (port 42)
MOV   AH, 0        on attend...
INT   16           ...l'appui sur une touche
IN    AL ,61       on...
AND   AL, FC       ...déactive (FC <==> /03)...
OUT   61, AL       ...le buzzer, réactive le contrôle de parité,...
```

61LE CLAVIER:

Vider le buffer:

```
MOV   AX, 0C00     sélection de la fonction C, sous fonction 0
INT   21           appel de la fonction C-0 de l'interruption DOS 21
```

Attendre l'appui sur une touche:

```
MOV   AH, 0        sélection de la fonction 0
INT   16           appel de l'interruption BIOS 16: dans AH, retourne le code de pression de
la
                   touche et dans AL, retourne le code ASCII correspondant

MOV   AH, 8        sélection de la fonction 8: pas d'affichage à l'écran (fonction 1: affichage)
INT   21           appel de la fonction 8 de l'interruption 21: dans AL, renvoie le code ASCII
CMP   AL, 55       on attend l'appui sur le u (code ascii de 55)
JNE   100          tant que l'on n'a pas pressé, on boucle
```

Voir si une touche a été appuyée:

```
MOV   AH, B        sélection de la fonction B
INT   21           appel de la fonction B de l'interruption DOS 21 (AL renvoie 0 ou FF)

MOV   AH, 6        sélection de la fonction 6
MOV   DL, FF       obligatoire pour faire fonctionner cette fonction
INT   21           appel de la fonction 6: ZF = 1 et AL = 0 si pas de caractère sinon:
                   ZF = 0 et AL caractère lu
```

62L'HORLOGE:

Connaître la date,...:

```
MOV   AH, 2A      sélection de la fonction 2A
INT   21          appel de la fonction 2A de l'interruption DOS 21 (AH renvoie le jour de la
                  semaine, DL la date, DH le mois et CX l'année)
```

Connaître l'heure,...:

```
MOV   AH, 2C      sélection de la fonction 2C
INT   21          appel de la fonction 2C de l'interruption DOS 21 (CH reçoit l'heure, CL les
                  minutes, DH les secondes et DL les centièmes)
```

Attendre 10 secondes:

```
@100: JMP   103
@102: DB    0                 on réserve un octet@103: MOV   AH, 2C          sélection de
la fonction 2C@105:     INT   21              appel de la fonction 2C de l'int21 (DH reçoit les
secondes)
@107: ADD   DH, A             on ajoute 10 aux secondes
@10A:CMP    DH, 3C            on compare le résultat à 60
@10D:JL     112               s'il est inférieur, on saute
@10F: SUB   DH, 3C            sinon, on enlève 60
@112: MOV   [102], DH         on met ce résultat à l'adresse 102
@116: INT   21                appel de la fonction 2C de l'int21 (DH reçoit les secondes)
@118: CMP   DH, [102]         on compare l'heure courante à la valeur de l'adresse 102
@11C: JNE   116               si on n'a pas atteind les 10 secondes, on continue
```

Attendre BX/18,2 secondes:

```
      MOV   AX, 0
      INT   1A          dans CX:XD: la valeur du compteur incrémenté chaque 1/18,2 sec
      ADD   BX, DX      BX reçoit la valeur à atteindre
@I:   INT   1A
      CMP   BX, DX
      JNE   @I
```

63LES DISQUES:

Voir si un lecteur est prêt:

```
@100: MOV   AX, 2524          on redirige l'interruption 24 (erreurs critiques)...
@103: MOV   DX, 113                   ... à l'adresse 113@106:   INT     21@108:
MOV   AH, 1C          on va lire les infos du lecteur...@10A:MOV  DL, 1          ... 1 (cad A:)
(0 = défaut ; 2 = B: ; ...)@10C:    INT   21@10E:       MOV   AX,4C00       si fin
normale@111:    INT   21@113:       MOV   AX,4C01       si on a eu une erreur critique
(dans notre cas: on ... @116:       INT   21            ... n'a pas pu lire sur le disque)
```

Savoir sur quel disque on a booté:

```
MOV   AX, 3305    sélection de la fonction 33, sous fonction 5
INT   21          appel de la fonction 33-05 de l'interruption 21 (DL renvoie 1= a:, 3 = c:)
```

Connaître le type du disque par défaut:

```
MOV   AH, 1B      sélection de la fonction 1B
INT   21          appel de la fonction 1B de l'interruption DOS21
MOV   AL, [BX]    AL reçoit F0=1M44 F8=disque_dur F9=720kou1M2 FC=180k FD=360k
```

64UN EXEMPLE COMPLET:

```
masection       SEGMENT PUBLIC
        ASSUME      CS:masection, DS:masection
        ORG     100H                    ; origine du programme
pgm:
        JMP     _texte                  ; écriture d'un texte
texte   DB      0Ah, 0Dh, 'Parametre:', '$'
_texte: MOV     DX, OFFSET texte
        MOV     AH, 9h
        INT     21h

        MOV     CL, CS:[0080h]          ; CL := longueur des parametres
        MOV     CH, 0
        ADD     CX, 81h                         ; CX := position de fin des paramètres
        MOV     SI, CX
        MOV     BYTE PTR DS:[SI], '$'   ; rajoute la fin de chaine
        MOV     DX, 81h                         ; la chaine commence en 81h
        MOV     AH, 9h
        INT     21h                     ; affiche les paramètres
        CALL    newline

        MOV     AL, 56h                 ; test de mes fonctions ...
        CALL    hexatex                 ; ... hexatex et
        CALL    newline
        MOV     AX, DS
        CALL    hexatxt                 ; ... hexatxt

        MOV     AX, 4C00h               ; ne pas oublier sinon on finit
        INT     21h                     ; dans les choux

;*****************************************************************************
; affiche le contenu de AL  ex: 43 (0100 0011)
hexatex PROC    NEAR    ;       AH      AL
        PUSH    AX
        PUSH    CX
        PUSH    DX
        MOV     AH, AL          ; 0100 0011     0100 0011
        SHR     AH, 1           ; 0010 0001
        SHR     AH, 1           ; 0001 0000
        SHR     AH, 1           ; 0000 1000
        SHR     AH, 1           ; 0000 0100
        AND     AL, 0FH         ;               0000 0011
        ADD     AL, '0'         ;
        CMP     AL, '9'         ;
        JBE     R_1             ;       si 0011 < 9: saut en R_1
        ADD     AL, 'A'-'0'-10  ;       ajoute 'A'
R_1:    ADD     AH, '0'         ;
        CMP     AH, '9'         ;
        JBE     R_2             ; si 0100 < 9 : saut en R_2
        ADD     AH, 'A'-'0'-10  ; ajoute 'A'
R_2:    MOV     CL, AL          ; sauve AL
        MOV     DL, AH
        MOV     AH, 2h
        INT     21h             ; affiche AH
        MOV     DL, CL
        MOV     AH, 2h
        INT     21h             ; affiche CL
        POP     DX
        POP     CX
        POP     AX
        RET
```

```
hexatex  ENDP

;*************************************************************************
; affiche le contenu de AX
hexatxt  PROC    NEAR
         PUSH    AX
         PUSH    CX
         MOV     CL, AL
         MOV     AL, AH
         CALL    hexatex
         MOV     AL, CL
         CALL    hexatex
         POP     CX
         POP     AX
         RET
hexatxt  ENDP

;*************************************************************************
; saute une ligne
newline  PROC    NEAR
         PUSH    AX
         PUSH    DX
         MOV     DL, 0Ah
         MOV     AH, 2h
         INT     21h
         MOV     DL, 0Dh
         MOV     AH, 2h
         INT     21h
         POP     DX
         POP     AX
         RET
newline  ENDP

masection        ENDS
         END     pgm
```

65UN TSR (TERMINATE AND STAY RESIDENT):

Comme son nom l'indique, un TSR est un programme qui reste résident en mémoire.

```
         JMP     installe
         ... ici le coeur du programme restant en mémoire (exemple: ancien23 et nouvo23)
installe:
         ... ici, on prépare (exemple: redirection de l'interruption 23)
         MOV     DX, OFFSET installe     ; DX := taille en octets du programme résident
         MOV     CL, 4
         SHR     DX, CL                  ; DX := le nombre de paragraphes du prog. résident
         INC     DX                      ; on prend une marge de sécurité
         MOV     AX, 3100h               ; fonction de TSR
         INT     21h                     ; on est résident
```

66REDIRIGER DES INTERRUPTIONS:

```
        JMP     installe

nouvo23:                        ; nouvelle inter 23 (on n'appelle pas l'int d'origine)
        ...
        IRET
        ancien23_off  DW  ?
        ancien23_seg  DW  ?

nouvo23bis:                     ; nouvelle inter 23 (on appelle l'int d'origine)
        ...
        PUSHF                   } on appelle l'interruption d'origine car 9A est ...
        var_bidon     DB  9Ah   } ... le code hexadécimal de CALL
        ancien23_off  DW  ?     } on fait donc un CALL ancien23_seg:ancien23_off
        ancien23_seg  DW  ?     }
        ...
        IRET

installe:
        MOV   AX, 3523h                 ; récupère le vecteur de INT 23 (control-C)
        INT   21h
        MOV   WORD PTR ancien23_off, BX ; sauvegarde l'offset du vecteur
        MOV   WORD PTR ancien23_seg, ES ; sauvegarde le segment du vecteur
        PUSH  CS
        POP   DS                        ; DS := CS = segment où est la nouvelle
fonction
        MOV   DX,OFFSET nouvo23         ; DX := offset de la nouvelle fonction
        MOV   AX,2523h                  ; change l'interruption
        INT   21h
```

67IMPRIMER:

```
PRINT   PROC  NEAR
        MOV   SI, debut_ou_imprimer
        MOV   CX, nombre_de_caractères_à_imprimer
        MOV   AH, 2
        XOR   DX, DX                        ; sélection de l'imprimante 0
        INT   17h                       ; renvoie l'état de l'imprimante dans AH
        TEST  AH, 10000000B             ; si busy
        JZ    PRINT_DONE
        TEST  AH, 00100000B             ; si out of paper
        JNZ   PRINT_DONE
PRINT_LOOP:
        LODSB                           ; AL := carac
        XOR   AH,AH                     ; AH := 0
        INT   17h                       ; imprime
        ROR   AH, 1                     ; vérifie le bit de time out
        JC    PRINT_DONE                ; s'il est mis, on arrête
        LOOP  PRINT_LOOP
PRINT_DONE:
        RET
PRINT   ENDP
```

68AFFICHER LE CONTENU D'UN FICHIER:

```
masection       SEGMENT PUBLIC
        ASSUME      CS:masection, DS:masection
        ORG     100H                    ; origine du programme
pgm:
        JMP     debut
segbuf  DW      0
longfic DW      0
erreur  DB      "Erreur en ouvrant `ANSI.TXT'$"
fichier DB      "ANSI.TXT", 0
debut:
        MOV     BX, SS          ; réserve suffisament de place dans la pile ...
        ADD     BX, 300h        ; ... pour ne pas écraser le programme courant
        MOV     [segbuf], BX
        CALL    affiche
        MOV     AX, 4C00h
        INT     21h

affiche PROC    NEAR
        MOV     AX, 3D00h                   ; fonction ouvre fichier
        MOV     DX, OFFSET fichier          ; nom du fichier
        INT     21h
        JC      err                         ; erreur d'ouverture
        MOV     BX, AX                      ; BX := handle du fichier
        MOV     DS, [segbuf]
        MOV     DX, 0                       ; le buffer est en [segbuf]:0000
        MOV     AH, 3Fh                         ; fonction lire
        MOV     CX, 0FFFFh                  ; on va essayer de tout lire
        INT     21h                         ; on lit et on met dans le buffer
        MOV     CS:[longfic], AX            ; nombre d'octets lus
        MOV     AH, 3EH                     ; fonction fermer fichier
        INT     21h                         ; on ferme le fichier
        CLD                                 ; direction du LODSB
        MOV     SI, 0
        MOV     CX, CS:[longfic]
boucle: MOV     AH, 2                       ; fonction écrire
        LODSB                               ; AL := saisiecarac dans buffer
        MOV     DL, AL
        INT     21h                         ; affiche un caractère
        DEC     CX
        JNE     boucle
        RET
err:
        MOV     AH, 9
        MOV     DX, OFFSET erreur
        INT     21h
        RET
affiche ENDP
masection       ENDS                        ; fin de la section
        END     pgm
```

69DIVERS:

Pour sauver une variable en mémoire:

```
@100    JMP     103
@102    DB      0               réserve 1 octet
                MOV     [103], DL       on sauve DL
```

70UN PEU DE VIDÉO:

71LES MODES VIDÉO:

nº	text gra.	text résol	pixel box	pixel résol	couleurs	pa-ges	@	système
00h	T	40x25	8x14		16 gris	8	B800	EGA
	T	40x25	8x16		16	8	B800	MCGA
	T	40x25	9x16		16	8	B800	VGA
01h	T	40x25	8x14		16	8	B800	EGA
	T	40x25	8x16		16	8	B800	MCGA
	T	40x25	9x16		16	8	B800	VGA
02h	T	80x25	8x14		16 gris	4	B800	EGA
	T	80x25	8x16		16	4	B800	MCGA
	T	80x25	9x16		16	4	B800	VGA
03h	T	80x25	8x14		16	4	B800	EGA
	T	80x25	8x16		16	4	B800	MCGA
	T	80x25	9x16		16	4	B800	VGA
04h	G	40x25	8x8	320x200		4	B800	CGA, EGA, MCGA, VGA
05h	G	40x25	8x8	320x200	4 gris		B800	CGA, EGA
	G	40x25	8x8	320x200	4		B800	MCGA, VGA
06h	G	80x25	8x8	640x200	2		B800	CGA, EGA, MCGA, VGA
07h	T	80x25	9x14		mono	var	B000	MDA, Hercules, EGA
	T	80x25	9x16		mono		B000	VGA
0Dh	G	40x25	8x8	320x200	16	8	A000	EGA, VGA
0Eh	G	80x25	8x8	640x200	16	4	A000	EGA, VGA
0Fh	G	80x25	8x14	640x350	mono	2	A000	EGA, VGA
10h	G	80x25	8x14	640x350	4	2	A000	64k EGA
				640x350	16		A000	256k EGA, VGA
11h	G	80x30	8x16	640x480	mono		A000	VGA, MCGA, ATI EGA, ATI VIP
12h	G	80x30	8x16	640x480	16/256k		A000	VGA, ATI VIP
	G	80x30	8x16	640x480	16/64		A000	ATI EGA Wonder
13h	G	40x25	8x8	320x200	256/256k		A000	VGA, MCGA, ATI VIP

72PASSER EN MODE VGA

```
MOV     AX, 0013h       ; 0003h pour retourner en mode texte
INT     10h
```

73ECRIRE UN PIXEL:

```
MOV     AH, 0Ch
MOV     AL, [adrcouleur]
MOV     CX, [adrx]
MOV     DX, [adry]
MOV     BX, 1
INT     10h                     ; équivalent à Mem [$A000:X+(Y*320)] := couleur;
On peut faire plus rapide:
MOV     AX, 0A000h
MOV     ES, AX                  ; ES := A000
MOV     BX, [adrx]              ; BX := x
MOV     DX, [adry]              ; DX := y
MOV     DI, BX                  ; DI := x
MOV     BX, DX                  ; BX := y
SHL     DX, 8                   ; DX := 256y
SHL     BX, 6                   ; BX := 64y
ADD     DX, BX                  ; DX := 256y + 64y = 320y
ADD     DI, DX                  ; DI := x + 320y
MOV     AL, [adrcouleur]        ; AL := couleur
STOSB                           ; ES:DI := couleur
Ou même encore plus rapide:
MOV     AX, 0A000h              } on n'a besoin de le faire ...
MOV     ES, AX  ; ES := A000    } ... que pour le premier point
MOV     BX, [adrx]              ; BX := x    (donc BH = x/256)
MOV     CX, [adry]              ; CX := y    (donc CL = y)
ADD     BH, CL                  ; BX/256 := x/256 + y     (donc BX = x + 256y)
SHL     CX, 6                   ; CX := 64 * CX = 64y
ADD     BX, CX                  ; BX := BX + 64y = x + 256y + 64y = x + 320y
MOV     AL, [adrcouleur]                ; AL := couleur
MOV     ES:[BX], AL             ; A000:[x + 320y] := couleur
```

74ATTENDRE LE RETOUR DU FAISCEAU D'ÉLECTRONS DE L'ÉCRAN (RETOUR PAGE):

```
Le faire avant de changer la palette sinon, l'image sautille.
                MOV     DX, 3DAh
eti1:
                IN      AL, DX
                AND     AL, 08h
                JNZ     eti1
eti2:
                IN      AL, DX
                AND     AL, 08h
                JZ      eti2
```

75CONNAÎTRE LA PALETTE:

La palette contient 256 couleurs. En modifiant une couleur dans la palette (ex: coul_numéro_45), on modifie la couleur que l'on verra à l'écran lorsqu'il y a 45 dans la mémoire vidéo.

```
MOV     DX, 03C7h
MOV     AL, num_couleur
OUT     DX, AL              ; Port[$03C7] := num_couleur
ADD     DX, 2               ; DX := $03C9
IN      AL, DX
MOV     [adrr], AL          ; r := Port[$03C9]
IN      AL, DX
MOV     [adrg], AL          ; g := Port[$03C9]
IN      AL, DX
MOV     [adrb], AL          ; b := Port[$03C9]
```

76AFFECTER LA PALETTE:

```
Port[$3c8] := num_couleur;
Port[$3c9] := r;
Port[$3c9] := g;
Port[$3c9] := b;
```

77EFFACER L'ÉCRAN:

```
pour I = 0 à 64000 faire
                [$A000:I] := couleur_sur _un_octet
fin_du_pour
```

Ce qui est équivalent à:

```
PUSH    ES
MOV     CX, 32000
MOV     ES, [A000]
XOR     DI, DI
MOV     AL, [acouleur]
MOV     AH, AL
REP     STOSW
POP     ES
```

78AFFICHER UN POINT:

```
MOV     AX, 13          passe en mode graphique
INT     10
MOV     AL, 4           couleur du point
MOV     CX, 100                 colonne 100
MOV     DX, 200                 ligne 200
MOV     AH, 0C          sélection de la fonction C
INT     10              appel de la fonction C de l'interruption BIOS 10
XOR     AX, AX                  pour attendre une touche pressée
INT     16              attend la touche
MOV     AX, 3           repasse en mode texte
INT     10
```

79L'ASSEMBLEUR ET LE C:

80EXEMPLE:

```
unsigned var = 0x1245;
asm {
        push    AX
        mov     AH, 0xBC
        mov     AX, var
        mov     var, AH
        pop     AX
}
```

81UTILISER LES INTERRUPTIONS DOS:

```
ax = bdos(0x19, dx, al);                    inter 21 , fonction 19,...
```

Exemple pour écrire un message:

```
static void mesg(char * s)
{ while (*s) bdos (2, *s++, 0); }
```

82VOIR LA VALEUR PRÉSENTE À UNE ADRESSE:

```
var = peek (segment, offset);
```

83REDIRIGER UNE ERREUR CRITIQUE:

```
#pragma warn -par                           ignore le warning de compilation car paramètres non utilisé
void far fonc (unsigned deverr, unsigned errval, unsigned far * devhdr)
{ _hardresume(_HARDERR_FAIL); }             on reprend
#pragma warn +par
...
harderr (fonc);                             ce que l'on met dans le main pour rediriger vers fonc
```

84ACCÉDER À LA TABLE DES VECTEURS D'INTERRUPTION:

```
unsigned get_int_seg(int Intnum)
{ unsigned *IntVec=(unsigned *)0x0;
  return(IntVec[Intnum*2+1]);
}
```

85SPÉCIFIQUE À WATCOM:

Definir une fonction assembleur dans le code CPP:

```
extern dword converti_de_wordAdword( word size );
#pragma aux  converti_de_wordAdword =    \
        "xor eax"                        \
        "mov eax, ebx"                   \
        parm   [BX];
```

BX est le parametre (donc BX := size)
EAX est la valeur de retour (donc dword := EAX)

```
void scrollactivepgup(char,char,char,char,char,char);
#pragma aux scrollactivepgup = \
        "mov AH,6"   \
        "int 10h"    \
        parm [ch] [cl] [dh] [dl] [al] [bh] \
        modify [ah];
```

Importer une fonction depuis un fichier .ASM:

Dans le fichier assembleur:

```
.code
public _lseg
_lseg dw 0
public _getlowmem
_getlowmem PROC
   PUSH EBX
   MOV  AX, _lseg
   ...
   POP EBX
   RET
_getlowmem ENDP
END
```

Dans le fichier C:

```
#ifdef  __cplusplus
extern "C" {
#endif
   extern word lseg;
   word getlowmem(word size);
#ifdef  __cplusplus
}
#endif
#pragma aux getlowmem "_*" parm [EBX] caller;     // caller <==> rien n'est mis dans la pile
                                                  // on ne passe que par BX
#pragma aux get_lseg "_*" ;
```

86DIVERS:

87PLUS RAPIDE:

AND	AX, AX	(21 C0)	est plus rapide que	CMP AX, 0	(3D 00 00)
XOR	AX, AX	(31 C0)	est plus rapide que	MOV AX, 0	(B8 00 00)

88COMMENT CONNAITRE SI LE CPU EST 8086, 80286, ...

Hooker l'interruption 06h (opérandes inconnus) et essayer:

SHL	DX, 5	(C1 E2 05)	80186 et plus
SMSW	DX	(0F 01 E2)	80286 et plus
MOV	EDX, CR0	(0F 20 C2)	80386 et plus
XADD	DX, DX	(0F C1 D2)	80486 et plus

89PAS D'EFFET DE BORD:

```
PUSHF
PUSH   AX ... DX
CALL   procedure
POP    DX ... AX
POPF
```

90CONCLUSION:

91COUVERTURE DU SUJET:

A mon avis, ce résumé couvre 30% du sujet.

92THÈMES NON ABORDÉS:

Sans objet.

93OÙ APPROFONDIR SES CONNAISSANCES:

Sur internet... par exemple:
http://webster.ucr.edu/Page_asm/ArtofAssembly/ArtofAsm.html
La liste des interruptions est à: ftp://ftp.simtel.net/pub/simtelnet/msdos/info/interxxx.zip
où à: ftp://garbo.uwasa.fi/pc/programming/interxxx.zip