TP1 de Système d'Exploitation

Objectif

Ce TP permet de faire mieux comprendre l'architecture et le fonctionnement interne d'un ordinateur, et la mise en pratique de quelques notions relatives au codage des données. Le langage assembleur nous servira de support pour comprendre le fonctionnement de la machine. Dans ce TP, nous nous bornerons à l'étude des registres et les instructions d'un processeur simple de la famille x86 (Intel 80x86 16 bits).

1. Eléments de support pour la pratique

a) **Usage de l'outil debug de MSDOS pour réaliser les TPs**

DOSBox permet de simuler DOS sur des Systèmes d'exploitation.

- Télécharger et installer DOSBox sur Wibdows, Linux ou MAC OS
- Chercher le fichier exécutable (.exe) de debug pour Windows
- Monter un lecteur (A,B,D, E, F etc..) sur un répertoire de votre ordinateur. Exemple mount D /Users/yfaye/Desktop
- Copier le (.exe) dans le repertoire /Users/yfaye/Desktop
- Lancer DOSBox et faire D: (comme si on se déplacer d'un lecteur sous DOS)
- Taper ensuite debug

Pour lancer **debug**, ouvrir une fenêtre DOS, puis taper `debug` (commande). Une fois *debug* ouvert avec " –", appuyer sur '?' suivit de la touche 'Enter' pour afficher la liste de commandes proposées.
Pour quittez *debug* appuyer sur 'q' + 'Enter'.

b) Les principales commandes de l'outil debug que sont

- R register: donne le contenu des principaux registres du processeur,
- D Dump ou Display: visualise d'une zone mémoire,
- E Enter: permet d'initialiser le contenu d'adresses mémoire,
- A Assemble: permet d'écrire des instruction en assembleur qui seront stockés en hexadécimal dans la mémoire.
- U Unassemble: permet de retrouver une instruction assembleur à partie d'une instruction en code machine (hexadécimal),
- G Go: permet d'exécuter un programme à partir d'une adresse donnée.
- T Trace: permet d'exécuter les instruction pas à pas.
- ? help: l'aide

c) Les registres du processeur CISC intel 8086

Un registre est un élément de mémoire interne du microprocesseur. Le micro-processeur CISC Intel 8086 (16bits) de la famille x86, qui est devenue l'architecture de processeur la plus répandue dans le monde des ordinateurs personnels, stations de travail et serveurs informatiques dispose d'un certain nombre de registres qui sont:

- 9 registres généraux

4 registres de travail

- l'accumulateur (**AX**), utilisé pour les opérations arithmétiques et le stockage de la valeur de retour des appels systèmes.
- BX: base (adresse mémoire)
- CX: compteur
- DX: données (entrées/sorties)
- 4 **registres d'index ou d'offset**
  - DI: indexe destination
  - SI: indexe source
  - BP: pointeur de base de la pile

- SP: pointeur sur le sommet de la pile
- IP: pointeur d'instruction, pointeur de pile (SP),

- **4 registres de segment mémoires**
  - registre de segment de code (CS),
  - registre de segment de données (DS),
  - registre de segment de pile (SS),
  - registre extra segment (ES),
- **1 registre d'état** (**FLAGS**).

Tous ces registres sont de 16 bits

NB: les registres **X (AX, BX, CX, DX)** peuvent être divisés en deux registres de 8 bits. **Le registre H** partie haute **(octet de gauche, poids fort)** et le registre **L** partie basse **(octet de droite, de poids faible)**. Par exemple, AX = **AH et AL**

d) Quelques mnémoniques du langage d'assemblage

Le tableau suivant présente quelques instructions de 80x86. Dans la première colonne on a les instructions en langage symbolique (langage d'assemblage), le code de l'instruction est donné en langage machine (hexadecimal qui peut être converti en binaire) dans la deuxième colonne. La 3ème colonne précise le nombre d'octets nécessaires pour coder l'instruction complète (opérande inclus). On note ***valeur*** une valeur sur 16 bits, et *adr* une adresse sur 16 bits également.

Type Instr.	Mnémonique	Code Opération	Octets	EFFET
Déplacement	MOV AX, valeur	B8	3	AX ← valeur
	MOV AH, valeur	B4	2	AH← valeur
	MOV AL, valeur	B0	2	AL← valeur
	MOV AX, [adr]	A1	3	AX←contenu de l'adresse adr
	MOV [adr], AX	A3	3	Range AX à l'adresse adr
Arithmétique	ADD AX, valeur	05	3	AX←AX+valeur
	ADD AX, [adr]	03 06	4	AX←AX+contenu de adr
	SUB AX, valeur	2D	3	AX←AX-valeur
	SUB AX, [adr]	2B 06	4	AX←AX-contenu de adr
Décalage	SHR AX, 1	D1 E8	2	Décalage AX à droite
	SHL AX, 1	D1 E0	2	Décalage AX à gauche
Incrémenation	INC AX	40	1	AX←AX+1
	DEC AX	48	1	AX←AX-1
Comparaison	CMP AX, valeur	3D	3	Compare AX et valeur
	CMP AX, [adr]	3B 06	4	Compare AX et contenu de adr
Branchement	JMP adr	EB	2	Saut inconditionnel
	JE adr	74	2	Saut si =
	JNE adr	75	2	Saut si ≠
	JG adr	7F	2	Saut si >
	JLE adr	7E	2	Saut si <=
	JA adr			Saut si CF =0
	JB adr			Saut si CF=1
Logique	OR AX, FF00			AX←AX OU FF00
	OR AX, BX			AX← AX OU BX
	AND AX, FF00			AX←AX AND FF00
	XOR AX, FFFF			AX←AX XOR FFFF

Par exemple, les instructions en assembleur sur le registre AX permettant d'additionner 3 avec 2 sont:

```
mov  AX,2
add  AX,3
```

Le code correspondant en langage machine est:

	hexadécimal	binaire
`mov AX,2`	`B80200`	`101110000000001000000000`
`add AX,3`	`050300`	`000001010000001100000000`

2. Exercices Pratiques

a) Visualisation du contenu de la mémoire.

La command `d` permet de visualiser le contenu d'un segment. Tapez:

- `d;` pour visualiser le contenu d'un segment.
  Noter que les adresses sont de la forme [**segment:déplacement**] comme par exemple 0CAA:0100 correspond au segment **0CAA**, au déplacement **0100** dans le segment.''
- `d 0210` , pour visualiser la mémoire à partir 0210
- `e 0210`, pour initialiser les adresses mémoires à partir de 0210. Saisir à partir de 0210 jusqu'à l'adresse 021F les valeurs hexadécimales suivantes: 4C 33 49 4E 46 4F 7E 55 41 53 5A 20 20 20 20 26
- `xxxx:0210 XX.4C XX.33 XX.49 XX.4E XX.46 XX.4F XX.7E XX.55 XX.41 XX.53 XX.5A XX.20 XX.20 XX.20 XX.20 XX.26` (où les XX représentent les valeurs existantes), il faut mettre un espace après chaque saisie de 2 chiffres hexadécimaux puis valider après avoir terminé.
- `d 210` , pour visualiser à nouveau le contenu de la mémoire.

Vous pouvez encore chargez en utilisant une suite de commandes move (`MOV`)

Pour se faire, utilisez l'instruction `MOV` pour charger successivement les valeurs dans la partie basse de l'accumulateur (AL).

Vous passerez en mode Assemblage avec la commande `A`: une adresse d'insertion d'une instruction apparaisse.

- `A` ; puis noter l'adresse mémoire `xxxx` et saisir l'instruction suivante :
- `MOV AL, 4C`
- `D xxxx ;` visualise la partie mémoire qui contient l'instruction que vous venez de saisir
- `t=xxxx` ; qui provoquera l'exécution de cette instruction et affichera le contenu des registres. Vous pourrez constater que le contenu de **AL** est bien devenu **4C**.

Le registre IP (Instruction Pointer) contient toujours l'adresse de l'instruction à exécuter, la commande `T` tout court exécute l'instruction logée dans cette adresse.

- `R ;` Pour visualiser le contenu de IP
- `T ;` exécute l'instruction pointée par IP
- `T xxxx ;` commence l'exécution à partir de l'instruction à l'adresse `xxxx` jusqu'à rencontrer un point d'arret.

En procédant d'une façon analogue, transférer le contenu de **AL** à l'adresse mémoire **0100** hexadécimal.

- `MOV [0100], AL;` transfert le contenu de AL à l'adresse 0100.
- `t;` exécute l'instruction
- `d 0100 ;` pour voir si le contenu de AL y est.
- `a`
- `mov al, 33`
- `mov [0101], al`
- `t`
- `.....`
- `.....`
- `mov al, 73`

Affichez le contenu de cette série d'adresses avec la commande `d`.

Que constatez vous sur la colonne qui est tout à fait à droite ?

De façon similaire, utiliser la table ASCII pour afficher votre nom sur la colonne de droite.

NB: on peut inscrire plusieurs instructions, pour les exécuter, il faut spécifier l'adresse de début et de fin des instructions à exécuter par la commande `g` (pour pointer) `suivie` de la commande `t` (pour continuer l'exécution) comme par exemple:

- `g=100 102`, pour exécuter l'instruction inscrite de 100 jusqu'à l'adresse 102.
- t ; pour exécuter l'instruction à l'adresse 102.

Opérations avec les registres

Taper `r` pour visualiser le contenu des registres:

Addition du contenu de deux registre

- `a 100;` pour se positionner à l'adresse 100
- `mov ax,1;` mettre 1_{16} dans AX
- `mov bx,1;` mettre 1 dans BX
- `add ax,bx;` additionner AX avec BX et mettre le contenu dans AX. Le fait que le résultat soit automatiquement mis dans AX montre que AX est le registre Accumulateur (résultat).
- `int 3`; point d'arret
- mov, ax, 2
- mov bx, 3

Tapez `r`, les contenus des registres n'ont pas changé

- `u 100`; pour désassembler et observer le code machine engendré

- `g=100 110;` pour exécuter les instructions de l'adresse 100 à l'adresse 110. Sans le point d'arrêt (instruction `int 3`), l'exécution se poursuit.

Mettre des instructions à un emplacement mémoire

Exemple:

- `a 100;` mettre des instructions à partir de l'adresse 100.
- **xxxx:0100** `mov al, AA`
- **xxxx:0102** `mov ah, BB`
- **xxxx:0104**
- `g=100 102`
- `t`

Calcul du complément à 2 par l'ordinateur

Montrons que l'ordinateur utilise le complément à 2 pour représenter les nombres négatifs.
Afficher le contenu des registres

- `r`

Aller à l'adresse 100 puis charger la valeur 30

- `a 100`
- `mov al, 30 ;` le $30_{16} = 48_{10}$

Chargez le contenu de AL à l'adresse 200 hexadécimale.

- `mov [200], al`
- `d 200, pour visualiser`
- `mov al, 0`
- `sub al, [200]`, soustraire à al ce qu'il y'a à l'adresse 200, le résultat est stocké dans AL.
- `t`

Vérifier que le contenu de AL, est la représentation en complément à 2 du nombre $(0\text{-}30)_{16}$

L'opérateur OU exclusif (XOR)

- `xor ax, 11`

Vérifier à partir du contenu actuel de AX que l'opérateur XOR est réalisé entre 11 et le contenu d'avant de AX.

La pile (1)

La pile est un emplacement mémoire représenté par [SS:SP] géré par le processeur en LIFO grâce au instructions *push* et *pop*.

Exemple :

```
Y:\>debug
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=00FD  BP=0000  SI=0000  DI=0000
DS=0765  ES=0765  SS=0765  CS=0765  IP=0100   NV UP EI PL NZ NA PO NC
```

Le segment où se trouve la pile est SS=0765 et à l'adresse 00FD qu'on peut voir dans le schéma suivant :

```
-d 0765:00F0                          La pile contient cette adresses:00FD
0765:00F0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 A0
0765:0100  B8 BB BB BB AA AA 50 53-58 5B 00 00 00 00 00 00
0765:0110  00 00 8F E9 00 F0 03 74-B2 00 8E 00 34 00 54 07
0765:0120  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0765:0130  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
```

Donc le SP contient les adresses des valeurs qui seront empiler/dépiler mais pas les valeurs en tant que telles. Les deux instructions *push* et *pop* permettent respectivement de placer et de retirer une valeur à l'emplacement mémoire (la pile) contenu dans SP (registre de la pile qui contient l'adresse du dernier emplacement).

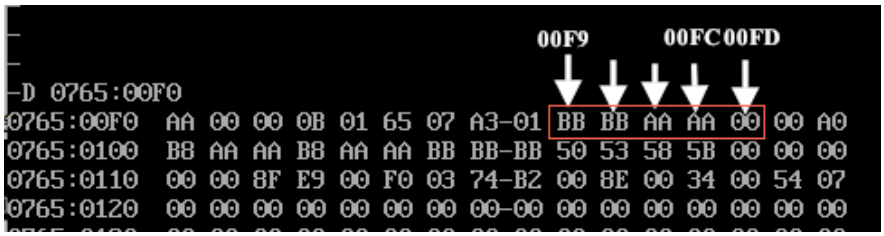
Exemple :

- `mov ax,aaaa` ; ax contient aaaa
- `mov bx,bbbb` ; bx contient bbbb
- `push ax` ; place aaaa dans la pile et décrémente SP de 2 (car on place 16 bits et puisque l'adressage est faite en octet, c'est 2 adresses)
- `push bx` ; place bbbb dans la pile et décrémente SP de 2

On a l'état des registres:

```
AX=AAAA  BX=BBBB  CX=0000  DX=0000  SP=00F9  BP=0000  SI=0000  DI=0000
DS=0765  ES=0765  SS=0765  CS=0765  IP=010B   NV UP EI PL NZ NA PO NC
0765:010B 58            POP     AX SP passe de 00FD à 00F9 car on a emplier AAAABBBB
```

Un dump de la pile montre que



- `pop ax` ; place bbbb dans ax, et incrémente SP de 2
- `pop bx` ; place aaaa dans bx, et incrémente SP de 2
- `int 3` ; fin du programme

Exécutons ce programme par une succession d'appuie sur la touche 't'. Constatez qu'a chaque *push*, SP est décrémenté selon la taille de la valeur du registre (SP contient FFEE, après *push,* SP est décrémenter: FFEC) par lequel nous avons empiler, et inversement à chaque *pop*, SP est incrémenté.

Les appels de procédures

Les appels de procédures sont effectués par l'instruction CALL qui est un *branchement inconditionnel*.

Exemple : CALL *AdresseDebutProcedure*

L'instruction RET marque la fin d'une procédure, elle ne prend pas d'argument. Après RET, le processeur passe à l'instruction qui suit immédiatement le CALL.

L'intruction CALL effectue les opérations suivantes :

- Empiler la valeur de IP qui pointait sur l'instruction qui suit le CALL
- Placer dans IP l'adresse de la première instruction de la procédure

L'intruction RET effectue les opérations suivantes :

- Dépiler une valeur et la ranger dans IP qui pointait sur l'instruction qui suit le CALL

- `a 300`
- `mov ax,2`
- `push cx`
- `add ax,2`
- `pop cx`
- `ret;` fin de saisie et retour `a l'invite de commandes de debug
- `a 100`
- `mov ax,11`
- `call 300`
- `add ax,2`

Exécuter la fonction en mode pas à pas (commande `t=100`) et suivre l'évolution des contenus des registres IP et SP au cours de l'exécution en tapant t.

[SS:SP] contient l'adresse dans laquelle le processeur doit revenir après un `ret` (point d'arrêt)**.**

Le registre FLAG

Les branchements conditionnels utilise les indicateurs, qui sont des bits du *registre d'état* (8 bits) du processeur.

Flag		1	0
Overflow	**OF**	**OV** (Overflow)	**NV** (No Overflow)
Direction	**DF**	**DN** (Decrement)	**UP** (Increment)
Interrupt	**IF**	**EI** (Enabled)	DI (Desabled)
Sign Flag	**SF**	**NG** (Negative)	**PL** (Positive)
Zero Flag	**ZF**	**ZR** (Zero)	**NZ** (No Zero)
Auxiliary Carry	**AF**	**AC** (Auxiliary Carry)	**NA** (No AC)
Parity	**PF**	**PE** (Even)	**PO** (Odd)
Carry Flag	**CF**	**CY** (Carry)	**NC** (No Carry)

- ZF=1 (positionné à ZR) lorsque le résultat de la dernière opération (addition soustraction) est zéro ou une comparaison si les 2 opérandes sont égaux, sinon ZF=0 (positionné à NZ).
  - Exemple : ZF initialement à **NZ**

    `Mov ax, 1234`

```
Mov bx, 1234
Cmp ax, bx ; ZF=ZR
```

- CF=1 (positionné à CY) s'il y a une retenue après l'addition ou la soustraction des bits de poids fort des opérandes, sinon CF=0 (positionné à NC).
  - Exemple : **FC** initialement à **NC**

```
Mov ax, FFFF
Mov bx, FFFF
Add ax, bx ; CF=CY
```

- SF=1 (positionné à NG) si le bit de poids fort d'une addition ou soustraction est 1, sinon SF=0 (positionné à PL).
  - Exemple : **SF** initialement à **NG**

```
Mov ax, 1111
Mov bx, 1111
Add ax, bx ; SF=PL
```

- OF=1 (positionné à OV) si le résultat d'une addition ou soustraction donne un nombre non codable (16bits) dans l'accumulateur quand par exemple l'addition de 2 nombre positifs donne un codage négatif, sinon SF=0 (positionné à NV).
  - Exemple : **OF** initialement à **NV**

```
Mov ax, 4000
Mov bx, 6000
Add ax, bx ; OF=OV car le bit de poids fort=1 alors que c'est un nombre positif
```

- AF=1 (positionné à AC) si le résultat d'une opération arithmétique addition ou soustraction génère un résultat provoquant une retenue sur le troisième bit de poids fort (en comptant à partir de 0), sinon SF=0 (positionné à NA).
  - Exemple : A**F** initialement à **NA**

```
Mov AX, 1000
Mov BX, 0008
Sub AX,BX ; AF positionné à AC
```

- PF=1 (positionné à PE) si dans les 8 bits de poids faible du résultat de la dernière opération, le nombre de bits à 1 est pair, sinon PF=0 (positionné à PO).
  - Exemple : **PF** initialement à **PE**

```
Mov ax, 4008
Mov bx, 6003
Add ax, bx ; PF=PO, car AX=A00B et dans 0B on a trois 1 (dans l'octet de poids faible)
```

- IF=1 (positionné à EI) si les interruptions sont autorisées, sinon IF=0 (positionné à DI).
  - L'instruction **`CLI`** (CLear Interrupt) positionne l'indicateur IF à 0 (interruptions ignorées),
  - L'instruction **`STI`** (SeT Interrupt) positionne l'indicateur IF à 1 (interruption autorisées).
  - NB : On distingue trois types d'interruptions: les interruptions appelées par le microprocesseur (ou interruptions internes), celles appelées par le logiciel et celles appelées par les périphériques (ou interruptions externes). On parle quelquefois d'exceptions pour les deux premiers types.
  - Les interruptions sont appelées par le microprocesseur en général en réponse à une erreur, par exemple l'interruption 00h est appelée lorsqu'on essaie de diviser par 0.
  - Les interruptions logicielles: sont appelées par le programme.
  - Les deux types d'interruptions que peuvent déclencher les périphériques grâce à deux broches sont :
    - La broche appelée NMI (pour NonMaskable Interrupt) déclenche une interruption qui n'est pas masquable, c'est-à-dire qu'on ne peut pas ne pas en tenir compte.
    - La broche appelée INTR (pour l'anglais INTeRrupt) permet de déclencher une interruption masquable en plaçant le numéro de l'interruption désirée sur le bus des données.
  - **La commande `CLI` n'a d'effet que sur les interruptions externes masquables**
  - La syntaxe de **l'appel d'une interruption** logicielle en langage symbolique est: `INT constant.` Cette instruction provoque le stockage dans la pile SP la mise à zero des indicateur TF et IF (16bits dans la pile) . et la et la sauvegarde des contenus de CS et IP (16bits+16bits), ainsi le pointeur de pile est décrémenté de 6.
  - Le retour d'une interruption est determine par l'instruction `IRET Interrupt RETurn.` Les contenus de CS, de IP et du registre des indicateurs sont rechargés depuis la pile. Le pointeur de pile est incrémenté de 6.
  - La commande P (Proceed) permet de ne pas voir les détail (Trace) de l'exécution de l'interruption

```
Mov ah, cx
Int 12 ; il se produit une interruption ; IF=DI, l'exécution avec la commande P permet de masquer les détails de l'interruption
```

Faire des traces (t) successifs ;

```
--
IRET ;  l'interruption est terminée ; IF=EI
CLI ;  IF=DI
CLI ;  IF=EI
```

- DF=1 (positionné à DN) si le transfert de données se fait en décrémentant les offsets, sinon en incrémentation des offsets DF=0 (positionné à UP).
  - L'instruction `CLD` (CLear Direction) positionne l'indicateur DF à 0 (UP : incrémentation des offsets),
  - L'instruction `STD` (SeT Direction) positionne l'indicateur DF à 1 (DOWN : décrémentation des offsets).
  - Exemple : D**F** initialement à **UP**

```
A 0100
0765:0100 STD ;  DF=DN
0765:0101 CLD ;  DF=UP
```

Le registre BP (Base Pointer)

Il permet de lire des valeurs sur la pile sans les dépiler ni modifier SP

- Exemple
  - `Mov BP,SP` ; BP pointe sur le sommet de la pile (adresse du sommet de la pile dans BP)
  - `Mov AX,[BP]` ; lit sans dépiler, met dans AX ci qu'il y a à l'adresse contenu dans BP .
  - `Mov AX,[BP+3]`

Les registres suivants ne seront pas étudiés dans ce TP

Les registres d'index permettent de mémoriser une adresse particulière (par exemple : début d'un tableau), ils sont aussi utilisés pour adresser la mémoire de manière différente: mode d'adressage indexé.

L'indexe SI : (source indexe) : pointeur sut la source

- Exemple :
  MOV AX, [SI+10000H] ; place le contenu de la mémoire d'adresse 1000+le contenu de SI, dans le registre AX.

L'indexe DI : (Destination indexe) : pointe sur la destination

Le registre ES (Extra segment) : ES : pointe vers les données du programme multi-segments (*extra segment*).

Vous avez à votre disposition la table ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL