



1

EMU8086

- ✓ EMU8086 est **l'émulateur du microprocesseur 8086** (compatible Intel et AMD) et de **l'assembleur intégré**.
- ✓ L'émulateur exécute des programmes comme le vrai microprocesseur en mode pas-à-pas. Il montre les registres, la mémoire, la pile (stack), les variables et les drapeaux (flags). Toutes les valeurs de la mémoire peuvent être examinées et modifiées par un double-clic.
- ✓ Les instructions peuvent être exécutées en arrière (back) et en avant (forward).
- ✓ EMU8086 peut créer un petit (tiny) système d'exploitation (OS) et écrire son code binaire sur une disquette amorçable (bootable floppy disk).
- ✓ Le logiciel comprend plusieurs dispositifs/périphériques virtuels externes : robot, moteur pas-à-pas (stepper motor), affichage LED et intersection de feux de circulation (traffic lights intersection).

EMU8086 : Installation (1/2)

Exigences :

- ✓ Les **droits d'administration** pour Microsoft Windows XP/VISTA/7 ...
- ✓ Au moins **10 Mo d'espace disque** et une **résolution d'écran de 1024x768** ou plus.

Installation : Site officiel de son éditeur "Simulation Soft" : www.emu8086.com (hors service depuis 2018)

- Téléchargement :
 - Windows 7 & 8 : <https://emu8086-microprocessor-emulator.fr.softonic.com/>
 - Windows 10 & 11 : <https://softfamous.com/emu8086/>
 - Mac & Linux : il faut l'installer sur une machine virtuelle.
- Logiciel propriétaire avec une version d'essai (14 jours) disponible sur divers sites web.

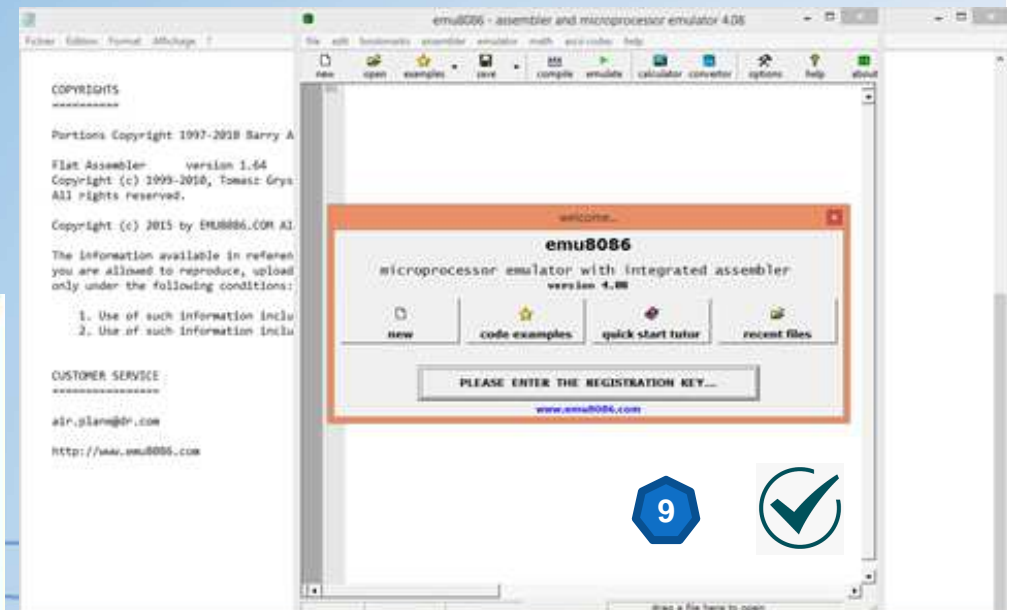
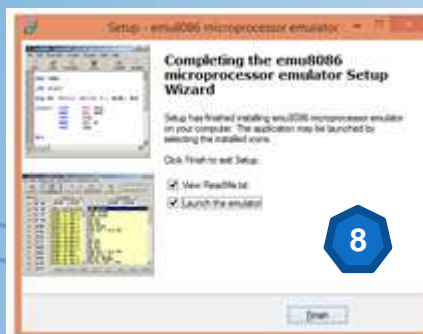
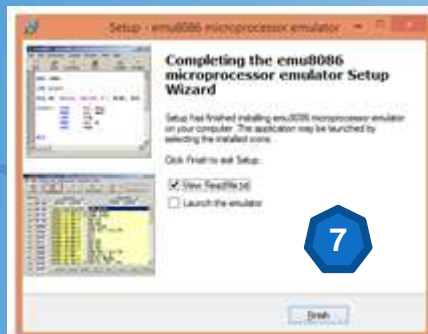
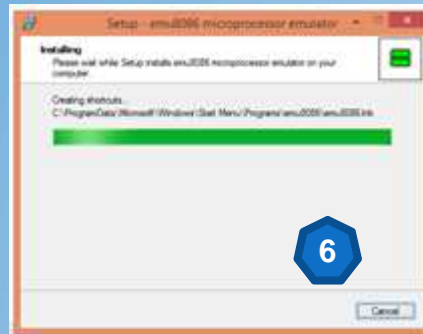
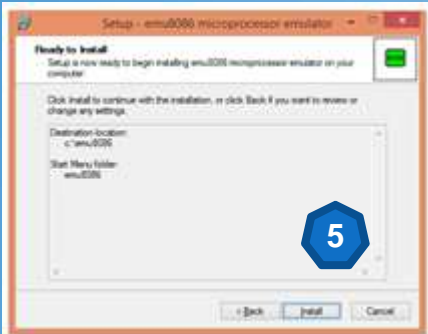
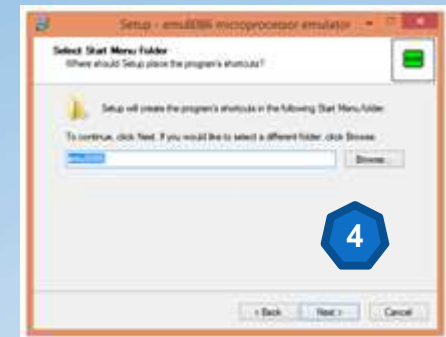
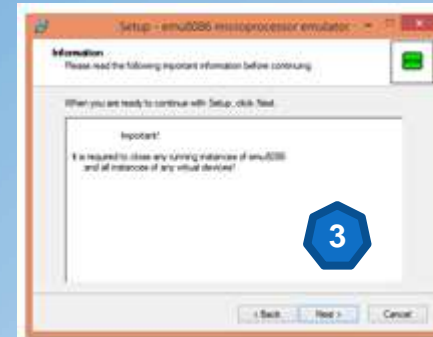
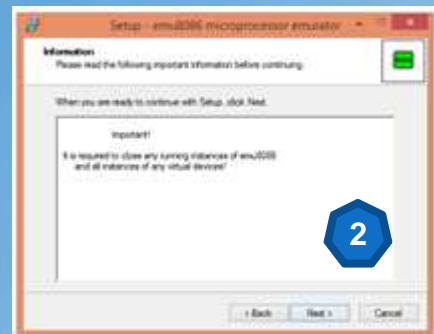
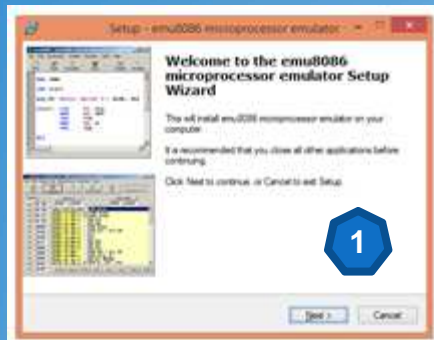
Activation :

User: ISHAAN,glaitm

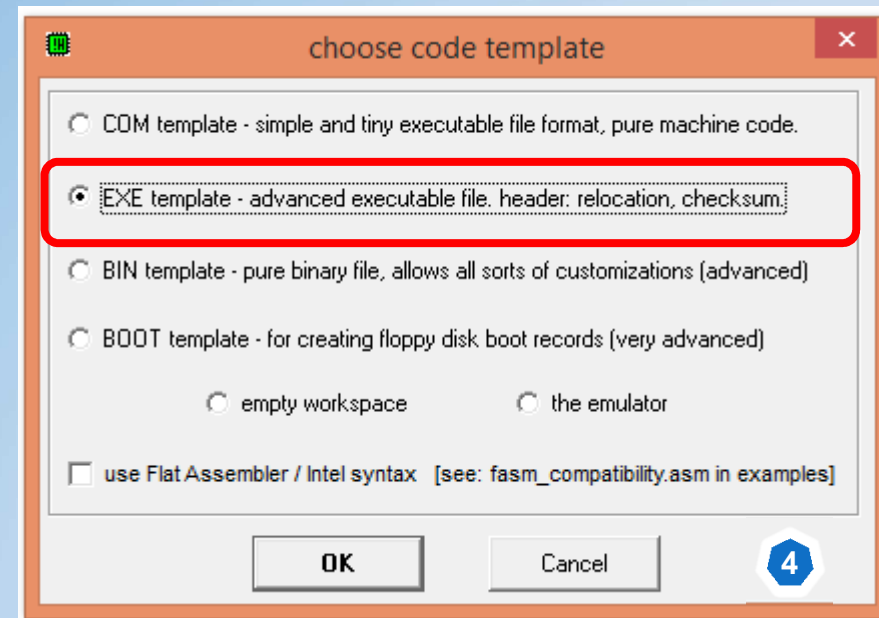
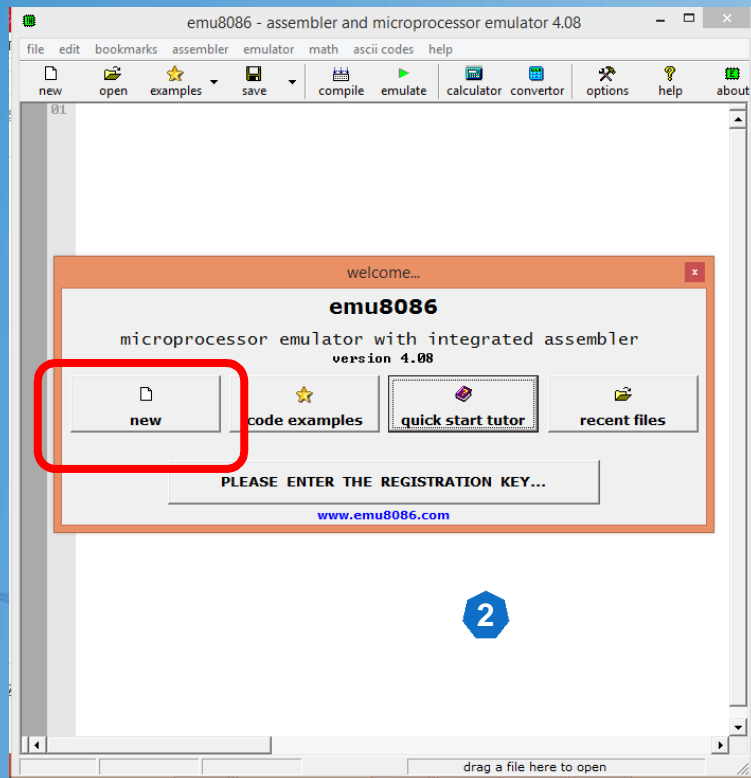
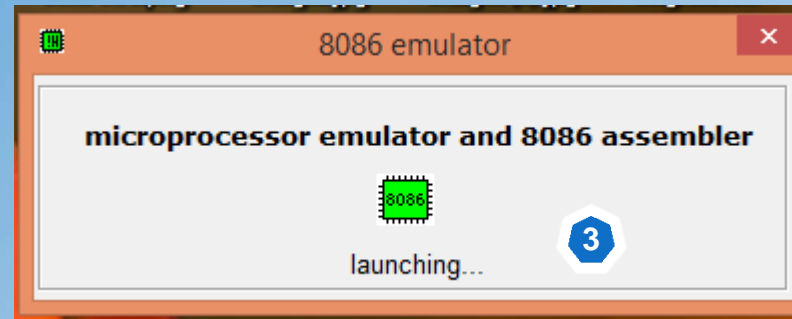
Key:27R3VDEFYFX4N0VC3FRTQZX



EMU8086 : Installation (2/2)

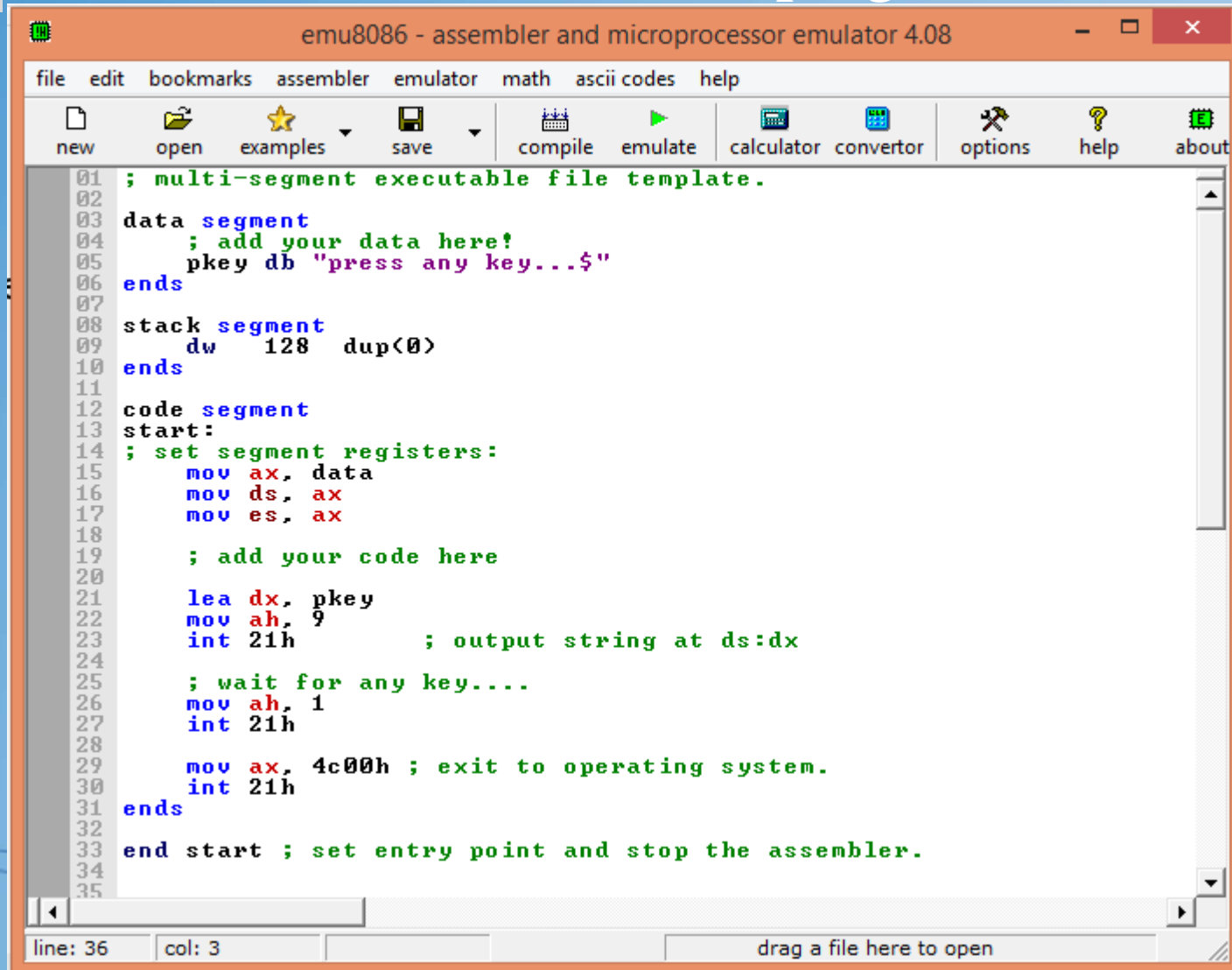


EMU8086 : Lancement du 1^{er} programme (1/6)



Remarque : Flat Assembleur (FASM) est un programme assembleur pour les architectures IA-32 et x86-64. Il est écrit en langage assembleur et existe pour les systèmes DOS, DexOS, GNU/Linux, Windows, et Menuet.

EMU8086 : Lancement du 1^{er} programme (2/6)



```

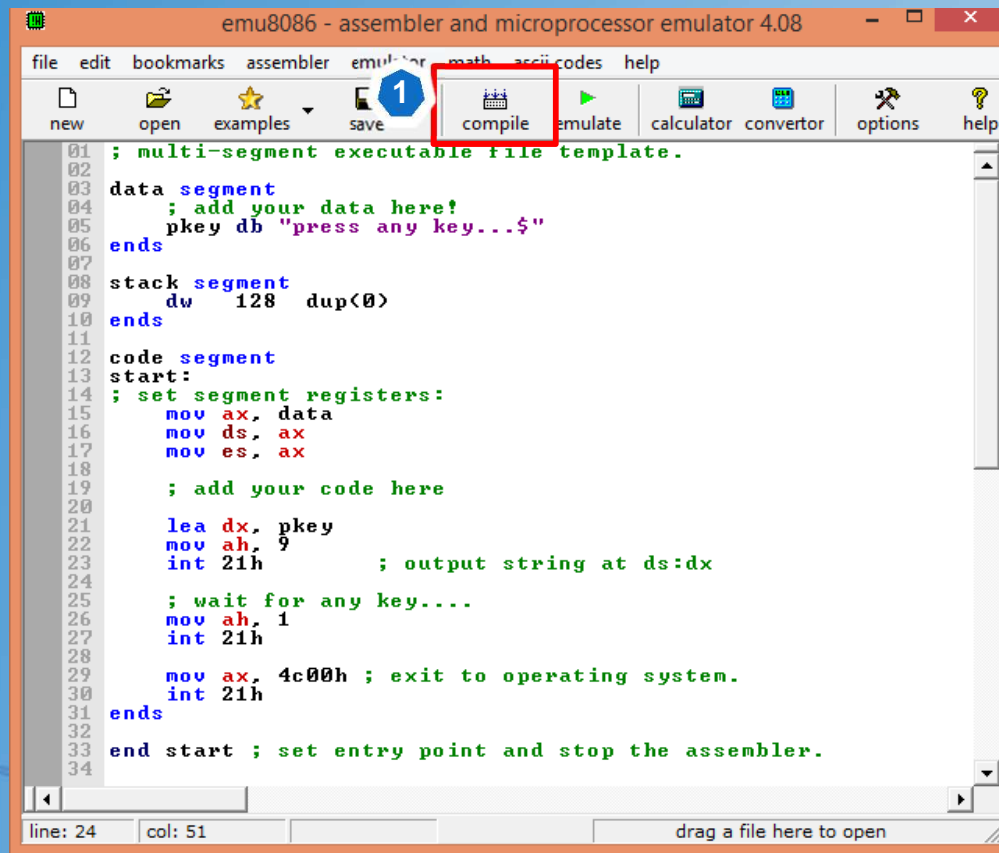
01 ; multi-segment executable file template.
02
03 data segment
04     ; add your data here!
05     pkey db "press any key...$"
06 ends
07
08 stack segment
09     dw 128 dup(0)
10 ends
11
12 code segment
13 start:
14     ; set segment registers:
15     mov ax, data
16     mov ds, ax
17     mov es, ax
18
19     ; add your code here
20
21     lea dx, pkey
22     mov ah, 9
23     int 21h          ; output string at ds:dx
24
25     ; wait for any key....
26     mov ah, 1
27     int 21h
28
29     mov ax, 4c00h ; exit to operating system.
30     int 21h
31 ends
32
33 end start ; set entry point and stop the assembler.
34
35

```

line: 36 col: 3 drag a file here to open

EMU8086 : Lancement du 1^{er} programme (3/6)

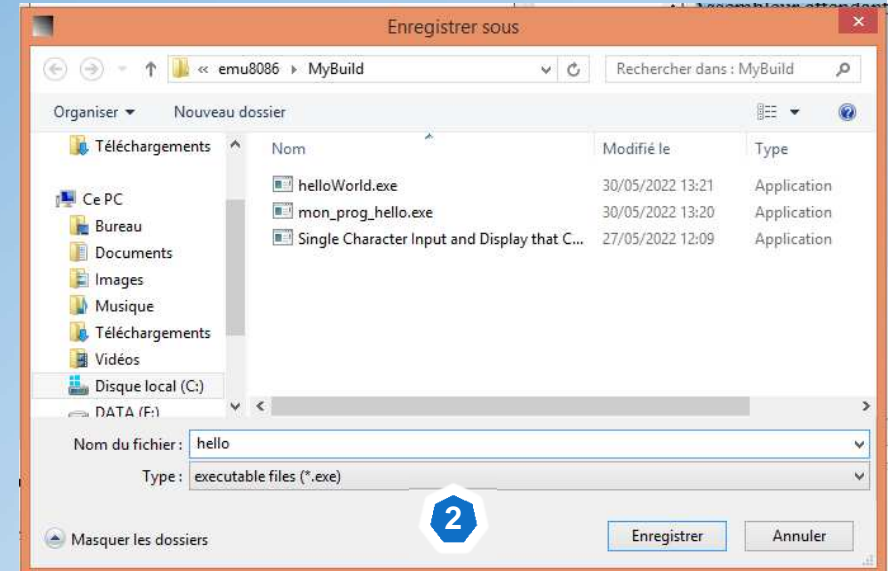
Après avoir créer le fichier via **New -> .exe** (étapes précédentes) vous aurez cette fenêtre à gauche ouvrant le code source d'un programme en Assembleur. Cliquez dans une 1^{ère} fois sur **Compile**, une boîte de dialogue (fenêtre à droite) s'affiche pour choisir un nom du fichier exécutable qui sera créer après assemblage et édition de liens.



```

01 ; multi-segment executable file template.
02
03 data segment
04 ; add your data here!
05 pkey db "press any key...$"
06 ends
07
08 stack segment
09 dw 128 dup(0)
10 ends
11
12 code segment
13 start:
14 ; set segment registers:
15 mov ax, data
16 mov ds, ax
17 mov es, ax
18
19 ; add your code here
20
21 lea dx, pkey
22 mov ah, 9
23 int 21h ; output string at ds:dx
24
25 ; wait for any key....
26 mov ah, 1
27 int 21h
28
29 mov ax, 4c00h ; exit to operating system.
30 int 21h
31 ends
32 end start ; set entry point and stop the assembler.
33
34

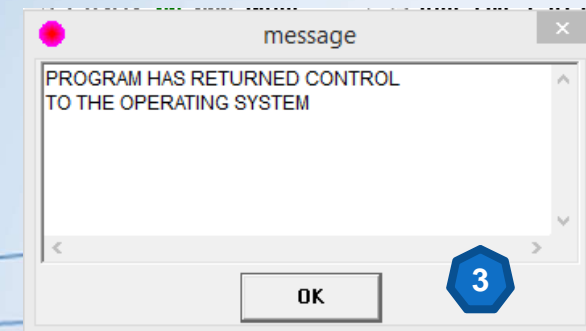
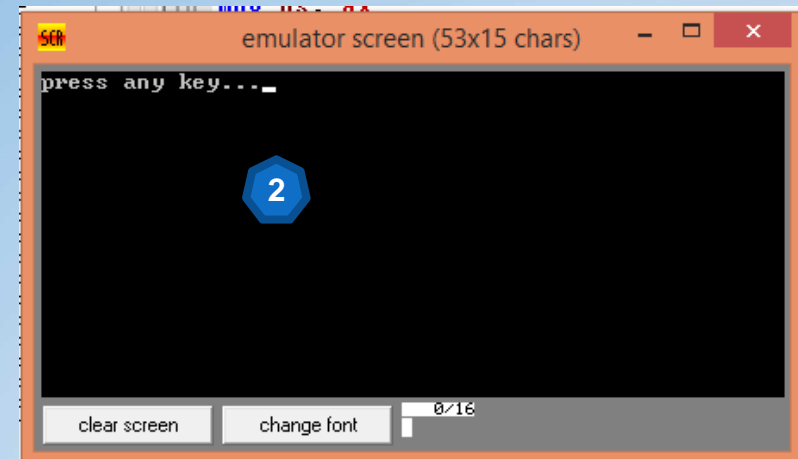
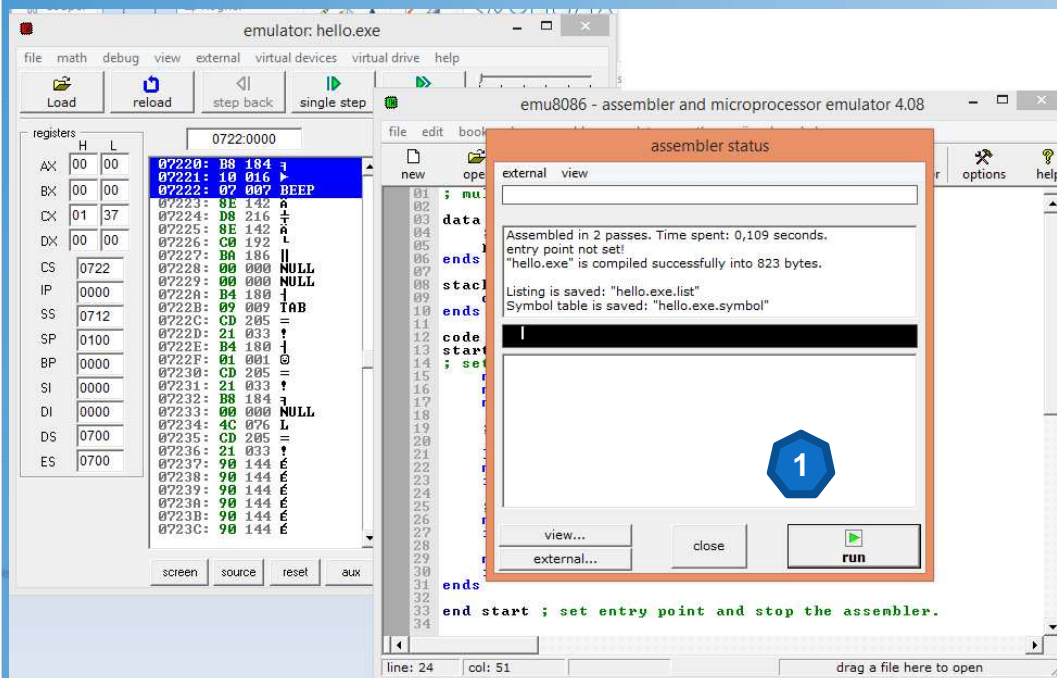
```



Renommer le **hello** ou **Hello**. Ensuite, cliquer sur enregistrer et le fichier sera créé dans un dossier qui s'appelle **MyBuild** existant dans la hiérarchie d'installation d'emu8086, en général il sera dans « C:\emu8086\MyBuild ».

EMU8086 : Lancement du 1^{er} programme (4/6)

Cliquer ensuite sur Run de la fenêtre qui s'ouvre pour exécuter le fichier .exe, après vous aurez la console DOS avec un message « press any key... », appuyer sur une touche du clavier et cliquer ensuite sur OK de la boîte dialogue qui s'affiche. Fermer la console et passer à l'étape suivante (prochaine diapositive).



EMU8086 : Lancement du 1^{er} programme (5/6)

Maintenant, fermer la fenêtre DOS (écran émulateur) et cliquer sur run dans la fenêtre à gauche, une boîte de dialogue s'affiche, cliquer sur Oui puis cliquer une autre fois sur run et commenter le nouveau affichage.

The screenshot shows the EMU8086 emulator interface. The main window displays assembly code with memory addresses and hex values. A dialog box 'emu8086' is open, asking 'program terminated. reload: C:\emu8086\MyBuild\hello.exe?' with 'Oui' and 'Non' buttons. The 'original source code' window on the right shows the assembly code being executed. Numbered callouts 1 through 4 highlight specific actions: 1 points to the 'run' button, 2 points to the 'Oui' button in the dialog, 3 points to the 'original source code' window, and 4 points to the 'run' button again.

Registers:

Register	H	L
AX	01	24
BX	00	00
CX	01	37
DX	00	00
CS	F400	
IP	0200	
SS	0712	
SP	00FA	
BP	0000	
SI	0000	
DI	0000	
DS	0710	
ES	0710	

Assembly Code (original source code):

```

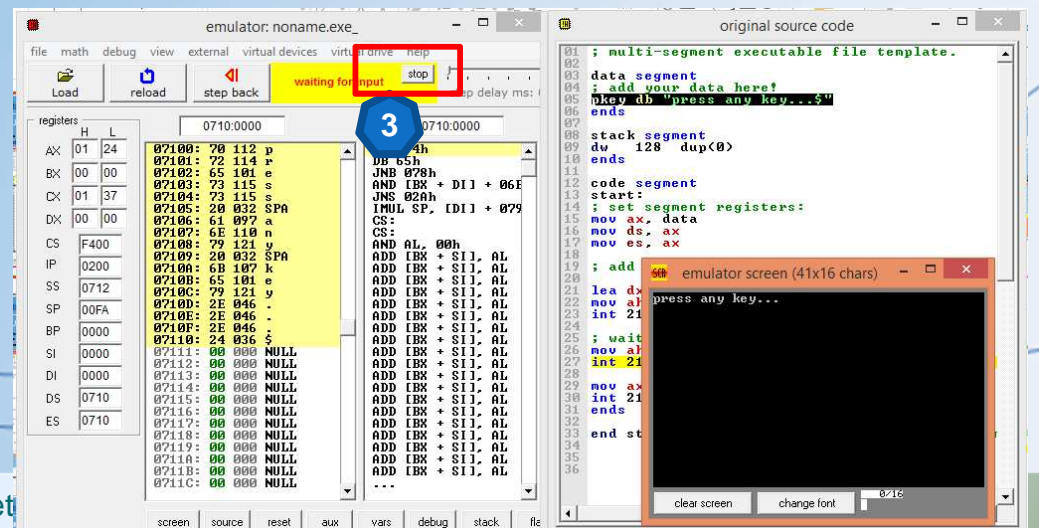
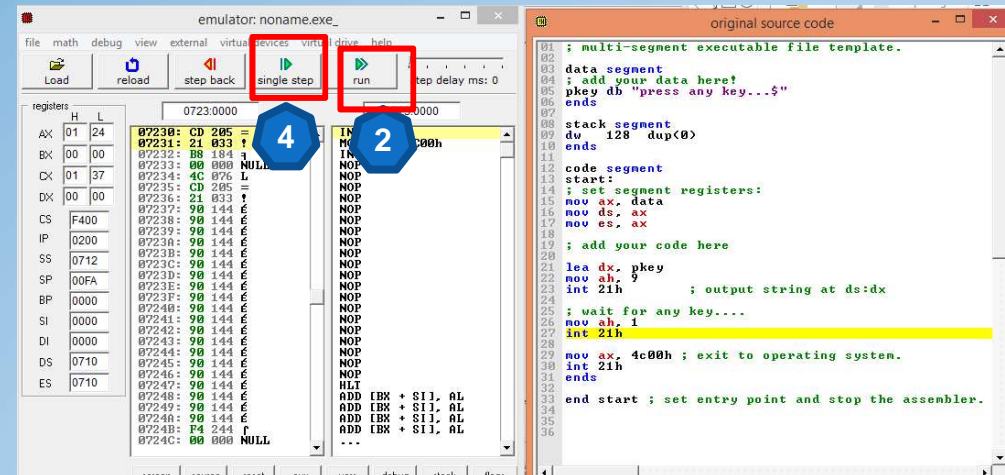
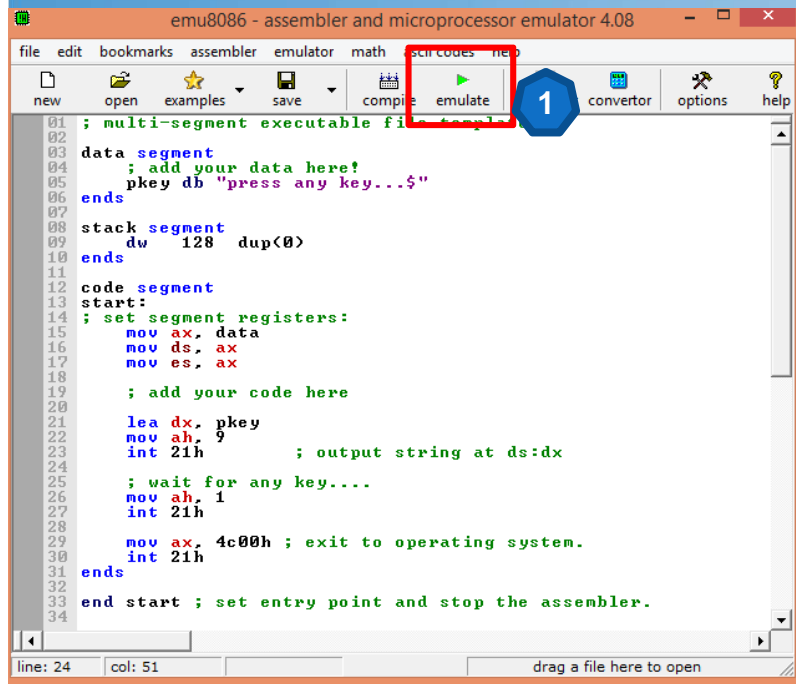
01 ; multi-segment executable file template.
02
03 data segment
04 ; add your data here!
05 pkey db "press any key...$"
06 ends
07
08 stack segment
09 dw 128 dup(0)
10 ends
11
12 code segment
13 start:
14 ; set segment registers:
15 mov ax, data
16 mov ds, ax
17 mov es, ax
18
19 ; add your code here
20
21 lea dx, pkey
22 mov ah, 9
23 int 21h ; output string at ds:dx
24
25 ; wait for any key....
26 mov ah, 1
27 int 21h
28
29 mov ax, 4c00h ; exit to operating system.
30 int 21h
31 ends
32
33 end start ; set entry point and stop the assembler.
34
35
36

```

EMU8086 : Lancement du 1^{er} programme (6/6)

Maintenant, on va essayer le bouton **Emulate**, cliquer dans la fenêtre à gauche sur **Emulate**, ensuite cliquer sur **run** dans la prochaine fenêtre qui s'ouvre à droite. Visualiser le résultat et interpréter la. Ensuite cliquer sur **Stop** et retourner à la 1^{ère} fenêtre, cliquer sur **single step** pour visualiser le résultat de chaque instruction étape par étape. Faites la même chose jusqu'à la dernière instruction.

Interpréter le résultat obtenu à chaque single step
Analyser les différents registres, la mémoire, l'ALU, la pile et interpréter le registre d'état (FLAGS) ?



EMU8086 : une vue d'ensemble des différents composants

The image displays the EMU8086 emulator interface with several components highlighted by red boxes:

- emulator: hello.com_**: The main emulator window showing registers (AX, BX, CX, DX, CS, IP, SS, SP, BP, SI, DI, DS, ES), memory addresses (0711:000D, 0713:0000), and assembly instructions (INT 010h, INC SI, JMP 0103h, MOV AH, 00h, INT 016h, RET, NOP).
- ALU - arithmetic & logic unit**: A window showing the ALU register values (FEDCBA9876543210) and the result of the current operation (0000000000000000).
- stack**: A window showing the stack memory layout (0700:FFFE to 0700:FFC6) and the current stack pointer (0700:FFC6).
- flags**: A window showing the status flags (CF, ZF, SF, OF, PF, AF, IF, DF) and their current values.
- stop on condition**: A dialog box for setting a breakpoint based on an operand, condition, and expression.
- variables**: A window for defining and editing variables (e.g., MSG, 48h).
- original source code**: A window showing the assembly code being executed, including comments and instructions like `name "hello", org 100h, jmp start, msg db 'Hello, world!', start:, mov si, 0`.
- Random Access Memory**: A window showing the memory layout (F400:01C0 to F400:0220) and the current memory contents.

EMU8086 : Table ASCII

En décimal

ascii codes																			
000:	null	032:	spa	064:	@	096:	`	128:	Ç	160:	Á	192:	Ł	224:	α				
001:	␣	033:	!	065:	A	097:	a	129:	ü	161:	í	193:	ł	225:	β				
002:	␣	034:	"	066:	B	098:	b	130:	É	162:	ó	194:	ŧ	226:	Γ				
003:	␣	035:	#	067:	C	099:	c	131:	é	163:	ú	195:	ı	227:	Π				
004:	␣	036:	\$	068:	D	100:	d	132:	â	164:	ñ	196:	ı	228:	Σ				
005:	␣	037:	%	069:	E	101:	e	133:	ä	165:	ñ	197:	ı	229:	σ				
006:	␣	038:	&	070:	F	102:	f	134:	å	166:	œ	198:	ı	230:	μ				
007:	beep	039:	'	071:	G	103:	g	135:	æ	167:	ı	199:	ı	231:	τ				
008:	back	040:	<	072:	H	104:	h	136:	ç	168:	ı	200:	ı	232:	θ				
009:	tab	041:	>	073:	I	105:	i	137:	ë	169:	ı	201:	ı	233:	Ω				
010:	newl	042:	*	074:	J	106:	j	138:	è	170:	ı	202:	ı	234:	δ				
011:	␣	043:	+	075:	K	107:	k	139:	ı	171:	ı	203:	ı	235:	δ				
012:	␣	044:	,	076:	L	108:	l	140:	ı	172:	ı	204:	ı	236:	ε				
013:	cret	045:	-	077:	M	109:	m	141:	ı	173:	ı	205:	ı	237:	ø				
014:	␣	046:	.	078:	N	110:	n	142:	ı	174:	ı	206:	ı	238:	ε				
015:	␣	047:	/	079:	O	111:	o	143:	ı	175:	ı	207:	ı	239:	ε				
016:	␣	048:	0	080:	P	112:	p	144:	ı	176:	ı	208:	ı	240:	ε				
017:	␣	049:	1	081:	Q	113:	q	145:	ı	177:	ı	209:	ı	241:	ı				
018:	␣	050:	2	082:	R	114:	r	146:	ı	178:	ı	210:	ı	242:	ı				
019:	␣	051:	3	083:	S	115:	s	147:	ı	179:	ı	211:	ı	243:	ı				
020:	␣	052:	4	084:	T	116:	t	148:	ı	180:	ı	212:	ı	244:	ı				
021:	␣	053:	5	085:	U	117:	u	149:	ı	181:	ı	213:	ı	245:	ı				
022:	␣	054:	6	086:	V	118:	v	150:	ı	182:	ı	214:	ı	246:	ı				
023:	␣	055:	7	087:	W	119:	w	151:	ı	183:	ı	215:	ı	247:	ı				
024:	␣	056:	8	088:	X	120:	x	152:	ı	184:	ı	216:	ı	248:	ı				
025:	␣	057:	9	089:	Y	121:	y	153:	ı	185:	ı	217:	ı	249:	ı				
026:	␣	058:	:	090:	Z	122:	z	154:	ı	186:	ı	218:	ı	250:	ı				
027:	␣	059:	;	091:	[123:	<	155:	ı	187:	ı	219:	ı	251:	ı				
028:	␣	060:	<	092:	\	124:	ı	156:	ı	188:	ı	220:	ı	252:	ı				
029:	␣	061:	=	093:]	125:	>	157:	ı	189:	ı	221:	ı	253:	ı				
030:	␣	062:	>	094:	^	126:	~	158:	ı	190:	ı	222:	ı	254:	ı				
031:	␣	063:	?	095:	_	127:	Δ	159:	f	191:	ı	223:	ı	255:	res				

En hexadécimal

ascii codes																			
00:	null	20:	spa	40:	@	60:	`	80:	Ç	A0:	Á	C0:	Ł	E0:	α				
01:	␣	21:	!	41:	A	61:	a	81:	ü	A1:	í	C1:	ł	E1:	β				
02:	␣	22:	"	42:	B	62:	b	82:	É	A2:	ó	C2:	ŧ	E2:	Γ				
03:	␣	23:	#	43:	C	63:	c	83:	é	A3:	ú	C3:	ı	E3:	Π				
04:	␣	24:	\$	44:	D	64:	d	84:	â	A4:	ñ	C4:	ı	E4:	Σ				
05:	␣	25:	%	45:	E	65:	e	85:	ä	A5:	ñ	C5:	ı	E5:	σ				
06:	␣	26:	&	46:	F	66:	f	86:	å	A6:	œ	C6:	ı	E6:	μ				
07:	beep	27:	'	47:	G	67:	g	87:	æ	A7:	ı	C7:	ı	E7:	τ				
08:	back	28:	<	48:	H	68:	h	88:	ç	A8:	ı	C8:	ı	E8:	θ				
09:	tab	29:	>	49:	I	69:	i	89:	ë	A9:	ı	C9:	ı	E9:	Ω				
0A:	newl	2A:	*	4A:	J	6A:	j	8A:	è	AA:	ı	CA:	ı	EA:	δ				
0B:	␣	2B:	+	4B:	K	6B:	k	8B:	ı	AB:	ı	CB:	ı	EB:	δ				
0C:	␣	2C:	,	4C:	L	6C:	l	8C:	ı	AC:	ı	CC:	ı	EC:	ε				
0D:	cret	2D:	-	4D:	M	6D:	m	8D:	ı	AD:	ı	CD:	ı	ED:	ø				
0E:	␣	2E:	.	4E:	N	6E:	n	8E:	ı	AE:	ı	CE:	ı	EE:	ε				
0F:	␣	2F:	/	4F:	O	6F:	o	8F:	ı	AF:	ı	CF:	ı	EF:	ε				
10:	␣	30:	0	50:	P	70:	p	90:	ı	B0:	ı	D0:	ı	F0:	ε				
11:	␣	31:	1	51:	Q	71:	q	91:	ı	B1:	ı	D1:	ı	F1:	ı				
12:	␣	32:	2	52:	R	72:	r	92:	ı	B2:	ı	D2:	ı	F2:	ı				
13:	␣	33:	3	53:	S	73:	s	93:	ı	B3:	ı	D3:	ı	F3:	ı				
14:	␣	34:	4	54:	T	74:	t	94:	ı	B4:	ı	D4:	ı	F4:	ı				
15:	␣	35:	5	55:	U	75:	u	95:	ı	B5:	ı	D5:	ı	F5:	ı				
16:	␣	36:	6	56:	V	76:	v	96:	ı	B6:	ı	D6:	ı	F6:	ı				
17:	␣	37:	7	57:	W	77:	w	97:	ı	B7:	ı	D7:	ı	F7:	ı				
18:	␣	38:	8	58:	X	78:	x	98:	ı	B8:	ı	D8:	ı	F8:	ı				
19:	␣	39:	9	59:	Y	79:	y	99:	ı	B9:	ı	D9:	ı	F9:	ı				
1A:	␣	3A:	:	5A:	Z	7A:	z	9A:	ı	BA:	ı	DA:	ı	FA:	ı				
1B:	␣	3B:	;	5B:	[7B:	<	9B:	ı	BB:	ı	DB:	ı	FB:	ı				
1C:	␣	3C:	<	5C:	\	7C:	ı	9C:	ı	BC:	ı	DC:	ı	FC:	ı				
1D:	␣	3D:	=	5D:]	7D:	>	9D:	ı	BD:	ı	DD:	ı	FD:	ı				
1E:	␣	3E:	>	5E:	^	7E:	~	9E:	ı	BE:	ı	DE:	ı	FE:	ı				
1F:	␣	3F:	?	5F:	_	7F:	Δ	9F:	f	BF:	ı	DF:	ı	FF:	res				

EMU8086 : programme somme deux nombres entiers (1/2)

; Programme somme deux nombres entiers

data segment; segment de donnees

x db 10d

y db 05h

s db ?

msg dw 'Somme = \$'

ends

stack segment ; segment de pile

dw 128 dup(0)

ends

code segment

start:

; initialise les registres du segment

mov ax, data ;

mov ds, ax

mov es, ax

; Traitement : somme

mov al, x ; place la valeur de x dans le registre AL

add al, y ; effectue la somme du contenu de AL (x) avec la valeur de y

mov s, al

; suite ...

lea dx, msg ; LEA: Load Effective Address

mov ah, 09h

int 21h ; affiche le message a ds:dx

mov dl, s

mov ah, 02h

int 21h ; affiche le resultat sous forme de caractere a ds:dl

; attend la frappe d'une touche au clavier

mov ah, 1

int 21h

mov ax, 4c00h ; exit vers le system d'exploitation

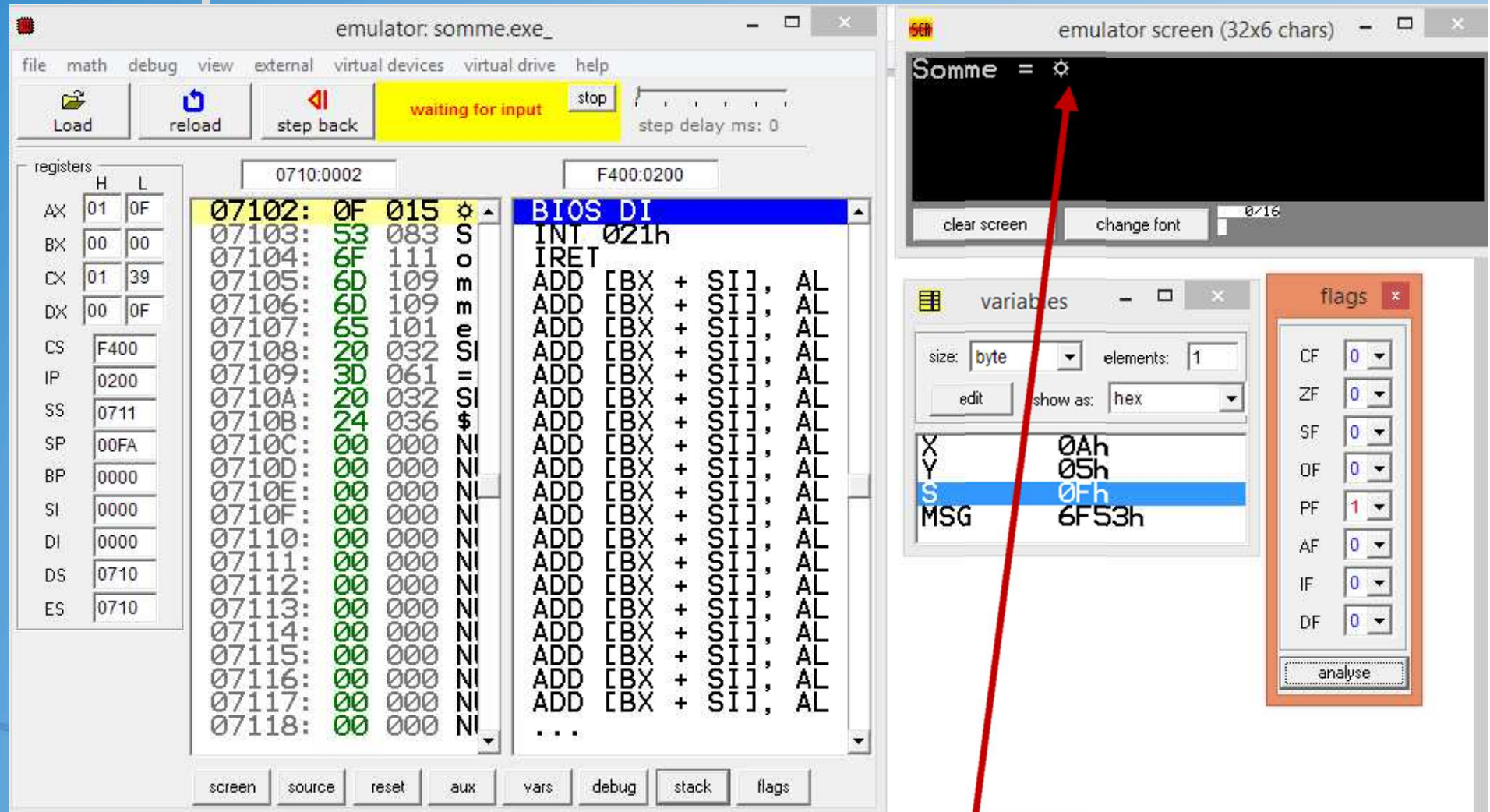
int 21h

ends

end start ;

TP Assembleur avec EMU 8086

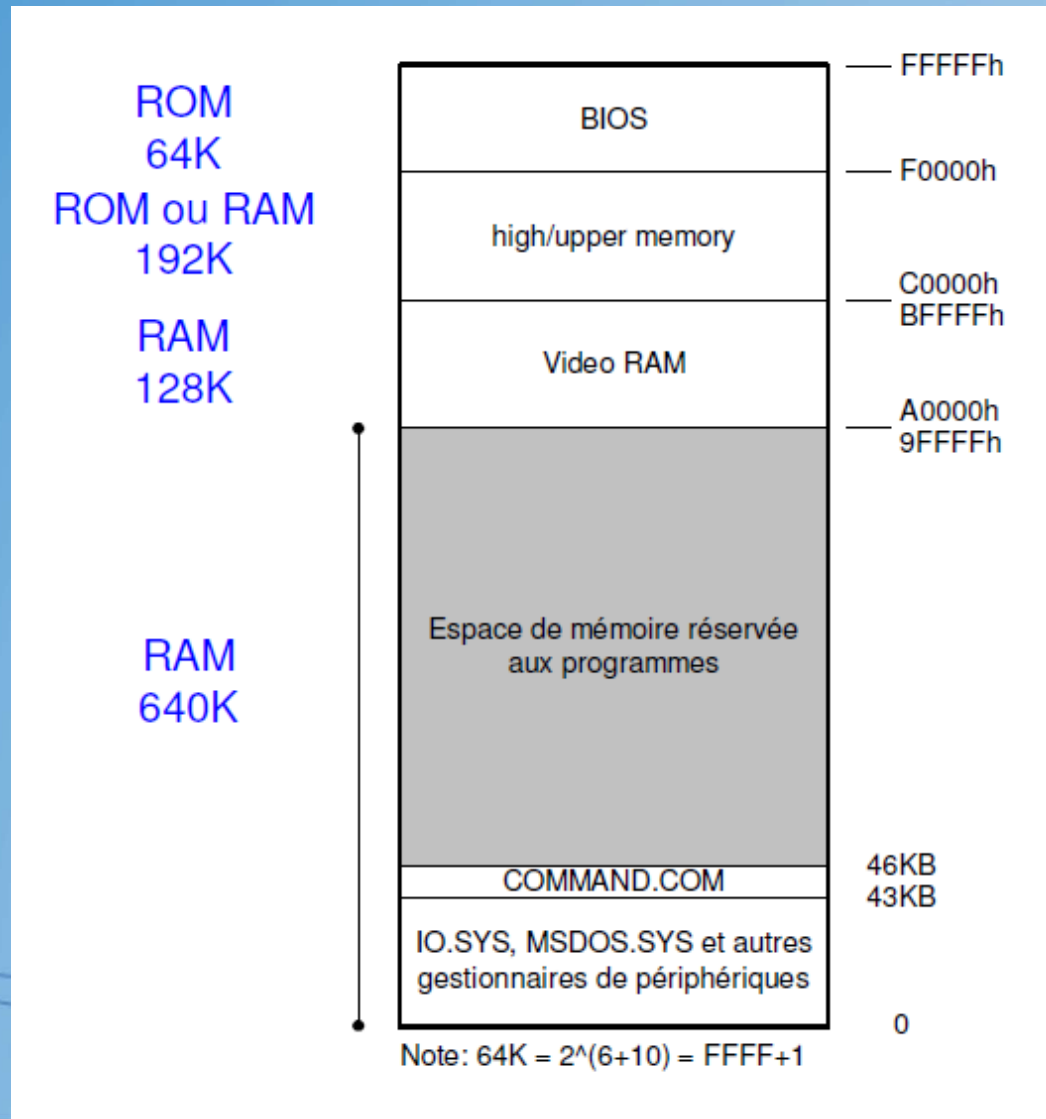
EMU8086 : programme somme deux nombres entiers (2/2)



Equivalent à 0Fh dans la table ASCII => résultat correct

TP Assembleur avec EMU 8086

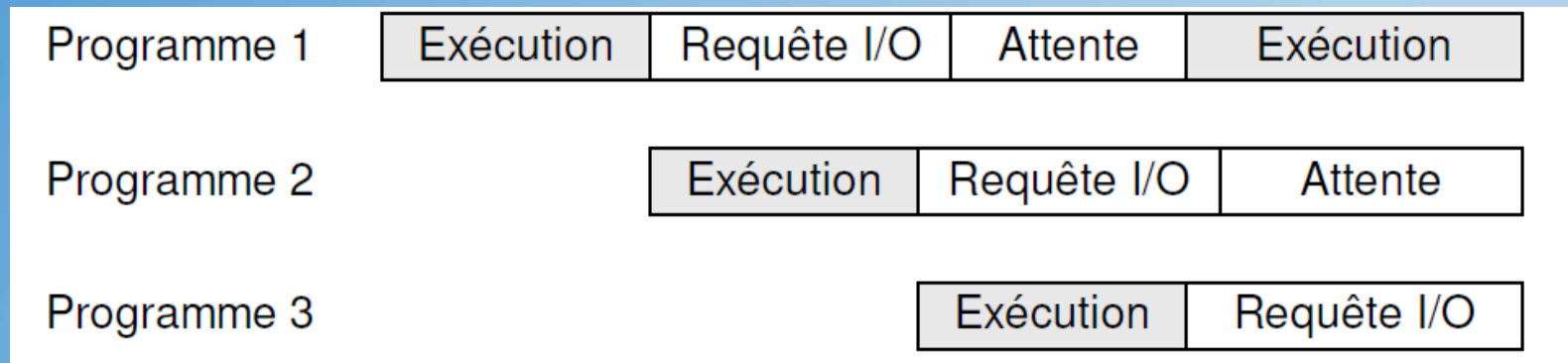
Organisation de mémoire typique de MSDOS (Ordinateurs multi-tâches)



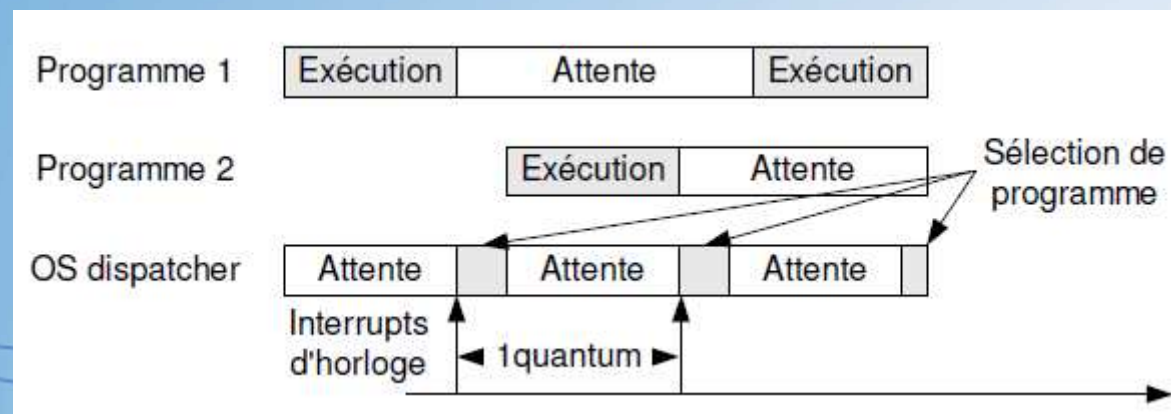
TP Assembleur avec EMU 8086

Organisation de mémoire typique de MSDOS (Ordinateurs multi-tâches)

=> **Partage du CPU lors d'attente après les I/O**



=> **Partage du CPU dans le temps**



Pipline (chaîne de traitement) : Ordinateurs multi-tâches

Pipeline est l'élément d'un processeur dans lequel l'exécution des instructions est découpée en plusieurs étapes. Le premier ordinateur à utiliser cette technique est l'IBM Stretch, conçu en 1961.

Avec un pipeline, le processeur **peut commencer à exécuter une nouvelle instruction sans attendre que la précédente soit terminée**. Chacune des étapes d'un pipeline est appelé **étape (stage)**. Le nombre *d'étages d'un pipeline est appelé sa profondeur*.

