TD1

Architecture et exécution d'un programme

Exercice 1 – Ordinateur simplifié

1. Un ordinateur simplifié dispose d'un compteur ordinal (`IP`), d'un registre d'instruction (`INST`) et d'un registre général (`R0`) et d'une mémoire RAM de 3 octets. Indiquer sur le schéma ci-dessous où se trouvent ces registres, la mémoire RAM et les bus d'adresse, de commande et de données.

2. On charge en mémoire le programme suivant :
   `R0` $\leftarrow 5$
   `R0` $\leftarrow$ `R0` $+7$
   On suppose que chaque instruction est codée sur 1 octet.
   Compléter le schéma ci-dessus en initialisant `IP` à 0.
3. Reprendre le schéma précédent en complétant ci-dessous la valeur des bus et les registres dans le schéma ci-dessous après la *lecture* de la première instruction :

4. Reprendre le schéma en complétant et en modifiant les valeurs après l'*exécution* de la première instruction :

5. Reprendre le schéma en complétant et en modifiant les valeurs après la *lecture* de la deuxième instruction.
6. Reprendre le schéma en complétant et en modifiant les valeurs après l'*exécution* de la deuxième instruction.

Exercice 2 – Une première addition en assembleur

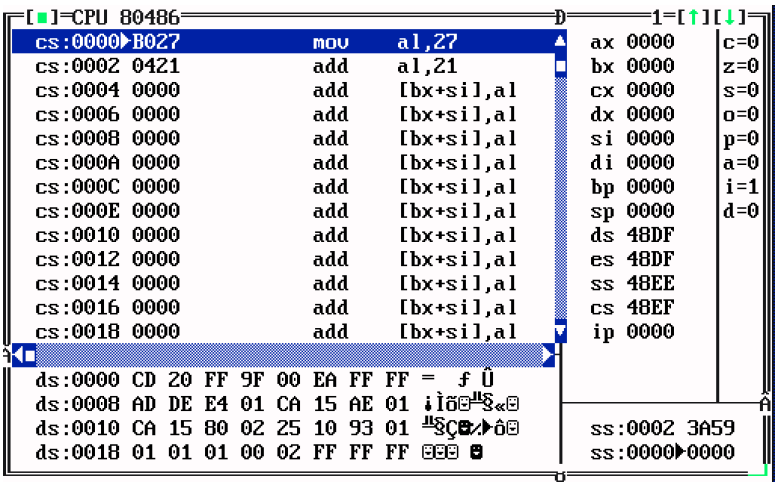
On a écrit le programme suivant en assembleur pour effectuer l'addition de $27h$ et $21h$:

```
1  assume cs:code
2
3  code SEGMENT
4  debut:
5    mov AL, 27h
6    add AL, 21h
7  code ENDS
8    END debut
```

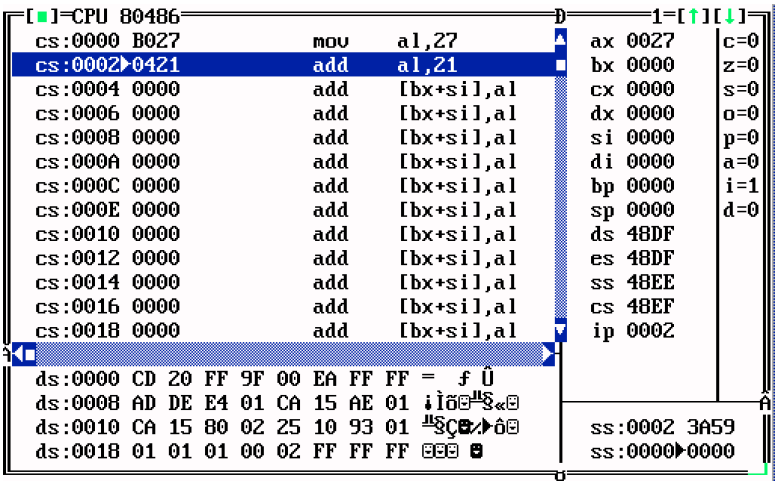
Après avoir assemblé et lié ce programme, l'exécution pas à pas dans Turbo Debugger a donné le résultat ci-contre.

1. Quel est le résultat de l'addition ?
2. Quel(s) registre(s) sont utilisés pour effectuer cette addition ?
3. Dans un tableau, représenter la zone mémoire du code avec les valeurs hexadécimales stockant le programme en langage machine.
4. À quel niveau de l'architecture 8086 trouve-t-on la zone mémoire du code ?
5. Comment l'unité centrale de traitement communique-t-elle avec la zone mémoire du code ?
6. Quels sont les *codop* produits par l'assemblage ? À quelles instructions assembleur correspondent-ils ? Préciser la ou les opérandes.

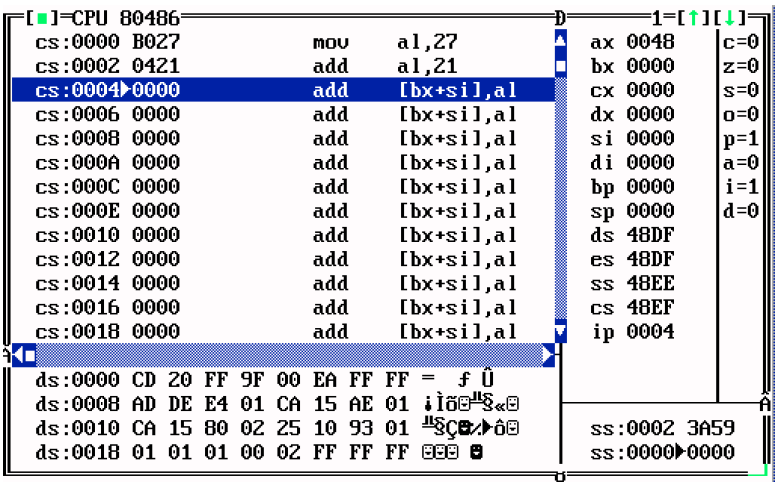
Démarrage :



Après l'exécution de la première instruction :



Après l'exécution de la deuxième instruction :



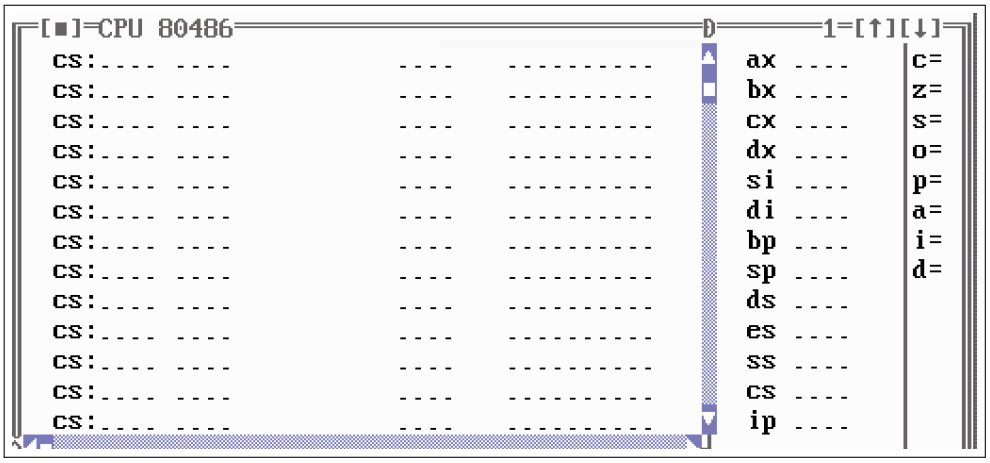
Exercice 3 – Une soustraction en assembleur

On a écrit le programme suivant en assembleur pour effectuer la soustraction de $27h$ et $21h$:

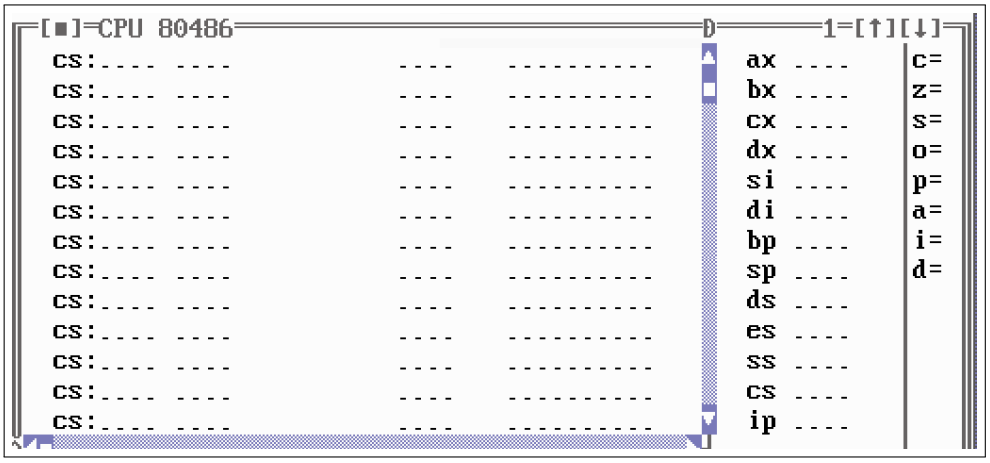
```
1  assume cs:code
2
3  code SEGMENT
4  debut:
5    mov AL, 27h
6    sub AL, 21h
7  code ENDS
8    END debut
```

1. Sachant que le *codop* de l'instruction « `sub AL, ...` » est « `2C` », compléter ci-dessous l'exécution pas à pas dans Turbo Debugger de ce programme (après avoir été assemblé et lié).
2. Représentez la zone mémoire du code dans un tableau.

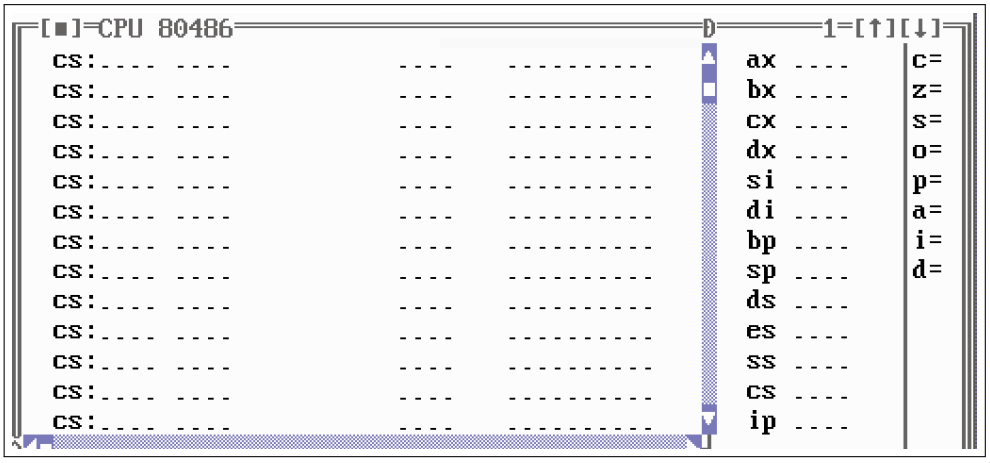
Démarrage :



Après l'exécution de la première instruction :



Après l'exécution de la deuxième instruction :



Exercice 4 – D'autres programmes en assembleur

De la même manière que dans l'exercice précédent, en vous aidant de l'annexe, exécutez pas à pas les programmes suivants et représentez la zone mémoire du code dans un tableau :

1. Une addition avec `AX`

```
1  assume cs:code
2
3  code SEGMENT
4  debut:
5    mov AX, 1409h
6    add AX, 263h
7  code ENDS
8    END debut
```

2. Une addition de nombres en écriture décimale

```
1  assume cs:code
2
3  code SEGMENT
4  debut:
5    mov AX, 18913
6    add AX, 15616
7  code ENDS
8    END debut
```

3. Une autre addition de nombres en écriture décimale

```
1  assume cs:code
2
3  code SEGMENT
4  debut:
5    mov AX, 142
6    add AX, 151
7  code ENDS
8    END debut
```

4. La même chose (?) avec `AL`

```
1  assume cs:code
2
3  code SEGMENT
4  debut:
5    mov AL, 142
6    add AL, 151
7  code ENDS
8    END debut
```

5. La même chose (?) avec `AL` et `AX`

```
1  assume cs:code
2
3  code SEGMENT
4  debut:
5    mov AX, 0
6    mov AL, 142
7    add AX, 151
8  code ENDS
9    END debut
```

6. La même chose (?)

```
1  assume cs:code
2
3  code SEGMENT
4  debut:
5    xor AX, AX
6    mov AL, 142
7    add AX, 151
8  code ENDS
9    END debut
```

Annexe - Quelques *codop*

symbole	codop	octets
`mov AH, valeur`	B4	2
`add AH, valeur`	80C4	3
`sub AH, valeur`	80EC	3
`mov AL, valeur`	B0	2
`add AL, valeur`	04	2
`sub AL, valeur`	2C	2
`mov AX, valeur`	B8	3
`add AX, valeur`	05	3
`sub AX, valeur`	2D	3
`xor AH, AH`	32E4	2
`xor AL, AL`	32C0	2
`xor AX, AX`	33C0	2