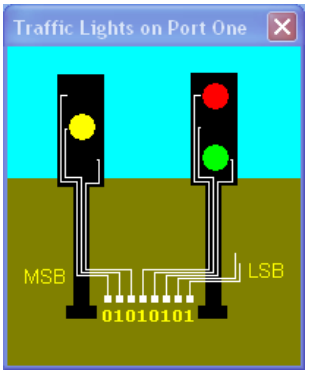
TD2 Assembleur : boucles et ports

P. Carreno - P. Portejoie

Samuel Le Berre
Groupe A2

Exercice 2-1

Observez le programme ci-dessous qui simule succinctement le fonctionnement de feux tricolores.




```
; Rôle : simule succinctement le fonctionnement de feux tricolores
; ================================================================
;
          CLO
Boucle:
                    ; Eteint toutes les lampes des feux tricolores
          MOV AL,0  ; en mettant le code 00000000 dans AL
          OUT 01    ; puis en envoyant AL au port correspondant

          MOV AL,FC ; De la même façon allume toutes les lampes
          OUT 01    ; des feux tricolores (code FC)
          JMP Boucle ; Recommence la séquence précédente
          END
```

- Donnez-en la traduction en langage d'assemblage du simulateur sms32 ; vous vous référerez à l'annexe « Calcul des sauts » pour résoudre le branchement inconditionnel *JMP Boucle (*Aidez-vous de la table des références construite par l'assembleur

- Complétez le programme de façon à ce qu'il s'arrête si l'utilisateur appuie sur la touche *Entrée* en fin de séquence (vous devrez faire en sorte qu'il soit sollicité).
  NB : *Entrée* = *Enter* = *RC* (Retour Chariot) = *CR* (Carriage Return)

- Refaites-en l'assemblage en portant votre attention sur le calcul d'offset ; pensez à construire, au fur et à mesure, la table des références et à l'utiliser pour les résoudre. Comme vu précédemment vérifiez vos résultats.

Exercice 2-2

- Ecrivez un programme qui calcule dans AL, par pas de 1, les nombres entre une borne inférieure et une borne supérieure (à définir à l'assemblage par 2 constantes stockées dans le 31ème et le 32ème mot de la mémoire ; attention, réfléchissez bien à leur adresse).

  NB : la borne supérieure peut être égalée, mais en aucun cas dépassée.

  Exemple : *2 et 5 ==> 2 3 4 5* doivent apparaître chronologiquement dans AL

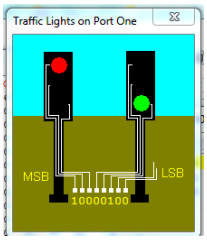
- Vérifiez sa conformité en simulant son exécution (faites-le tourner « à la main »).

- Donnez-en la traduction en langage d'assemblage du simulateur sms32.

TP2 Assembleur : boucles et ports

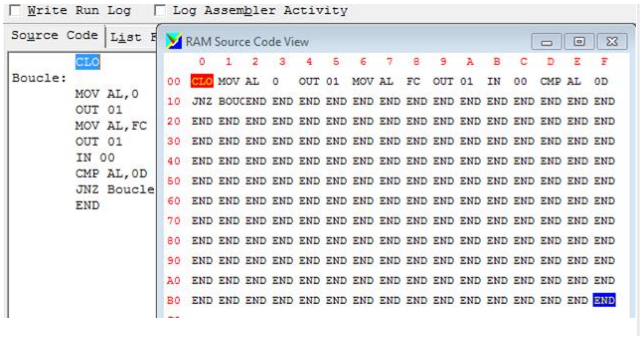
P. Carreno - P. Portejoie

Exercice 2-1

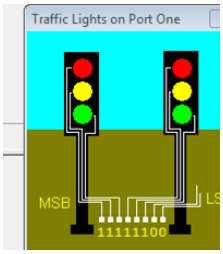


- Reprenez le deuxième programme de l'exercice 2-1 du TD et faites-le fonctionner. Vérifiez la conformité des assemblages faits manuellement en TD.

Tous est conforme, s'assemble et s'exécute correctement

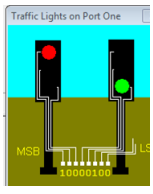


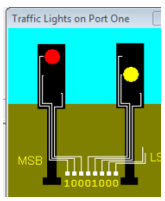

```
; LIST FILE : SUCCESS : No errors found.
;
; This shows the machine code that was generated and
; the addresses at which the codes were stored.
;
        CLO             ; [00]  FE
BOUCLE:                 ;
        MOV     AL,0    ; [01]  D0 00 00
        OUT     01      ; [05]  F1 01
        MOV     AL,FC   ; [06]  D0 00 FC
        OUT     01      ; [0A]  F1 01
        IN      00      ; [0C]  F0 00
        CMP     AL,0D   ; [0D]  DB 00 0D
        JNZ     BOUCLE  ; [10]  C2 F1
        END             ; [12]  00
;
; SUCCESS : No errors found
```

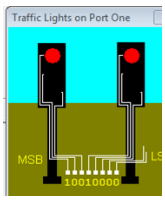


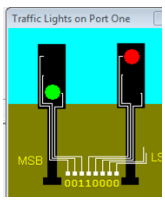
- Complétez cette version de façon à ce que le programme simule le fonctionnement d'un carrefour à feux tricolores français. On donne pour cela l'information suivante concernant le paramétrage des feux (vous commencerez par compléter ce tableau) : **On remplace FC par l'Hexadécimal correspondant a l'état souhaité.**

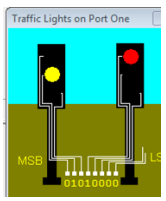
Rouge	Orange	Vert	Rouge	Orange	Vert	Inutilisé	Inutilisé	Hexadecimal	Feux	Temps
1	0	0	0	0	1	0	0	84	R-V	↓
1	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**88**	R-O	
1	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**90**	R-R	
0	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**30**	V-R	
0	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**50**	O-R	
1	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**90**	R-R	

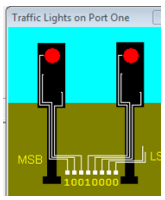








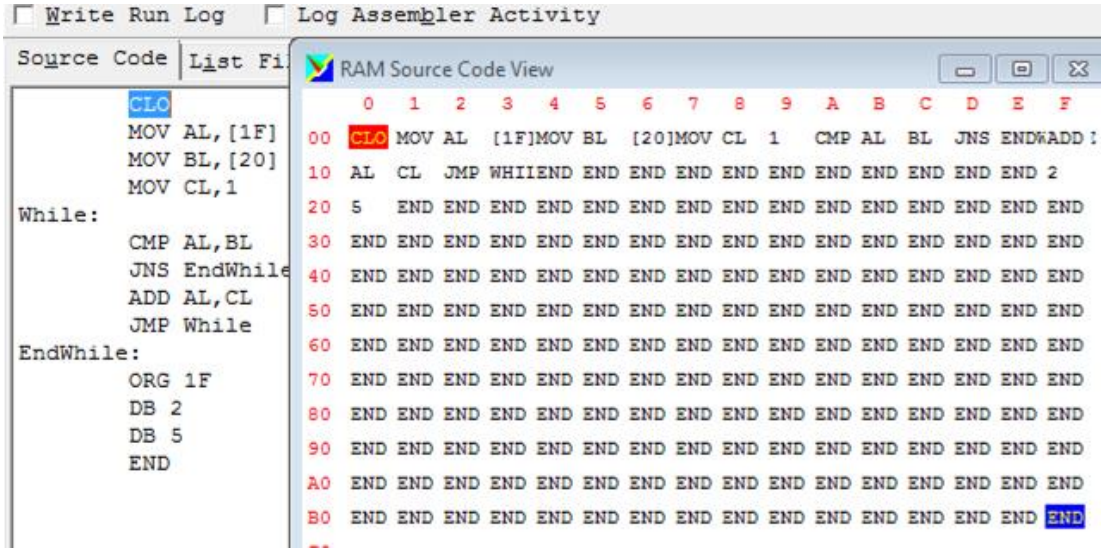


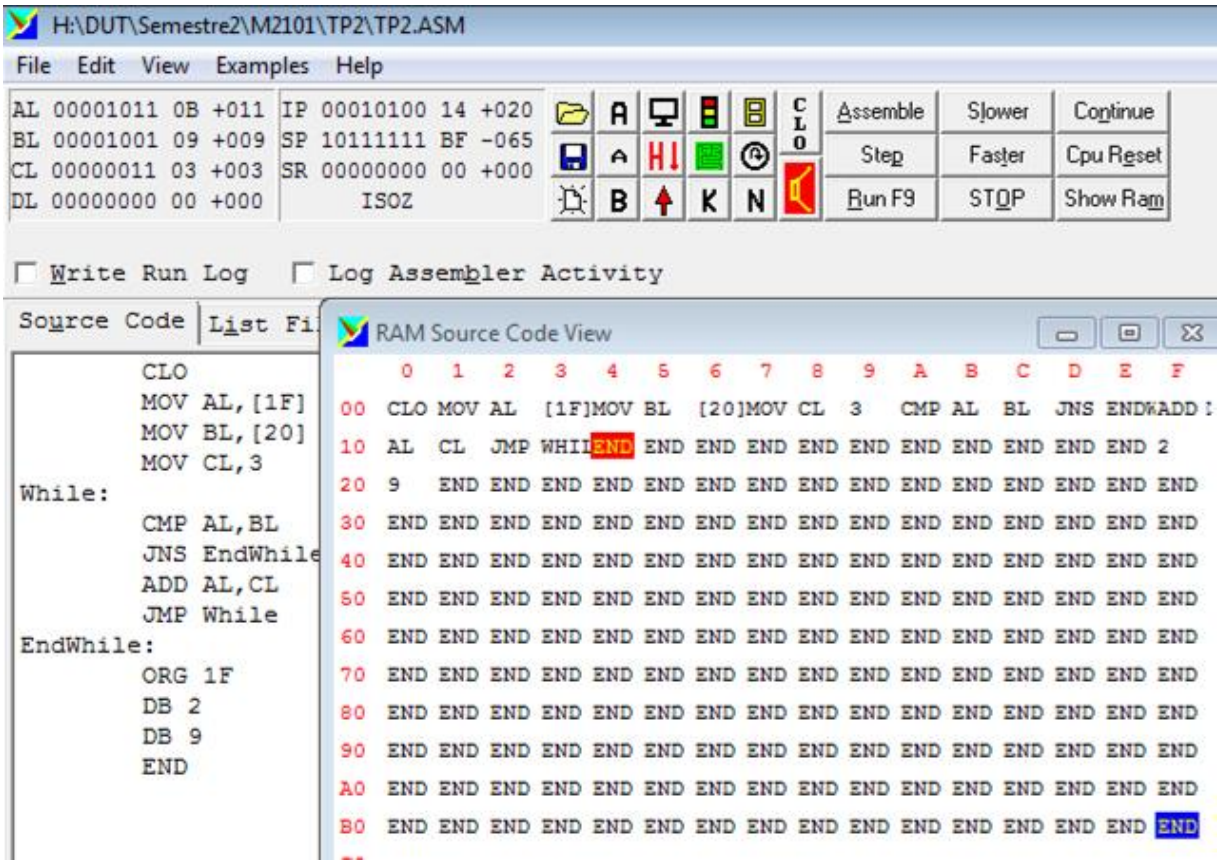


Exercice 2-2

- Reprenez le programme de l'exercice 2-2 du TD et faites-le fonctionner. Vérifiez la conformité des assemblages faits manuellement en TD. **Tout est normal et l'assemblage est bien fait**

- Modifiez le programme afin de pouvoir en fixer le pas **par initialisation d'une constante en mémoire, à l'assemblage (*DB*)**. Testez-le avec *2 à 5, pas de 1 (==> observation de 2 3 4 5 dans AL)*, puis avec *2 à 9, pas de 3 (==> observation de 2 5 8 dans AL)*.

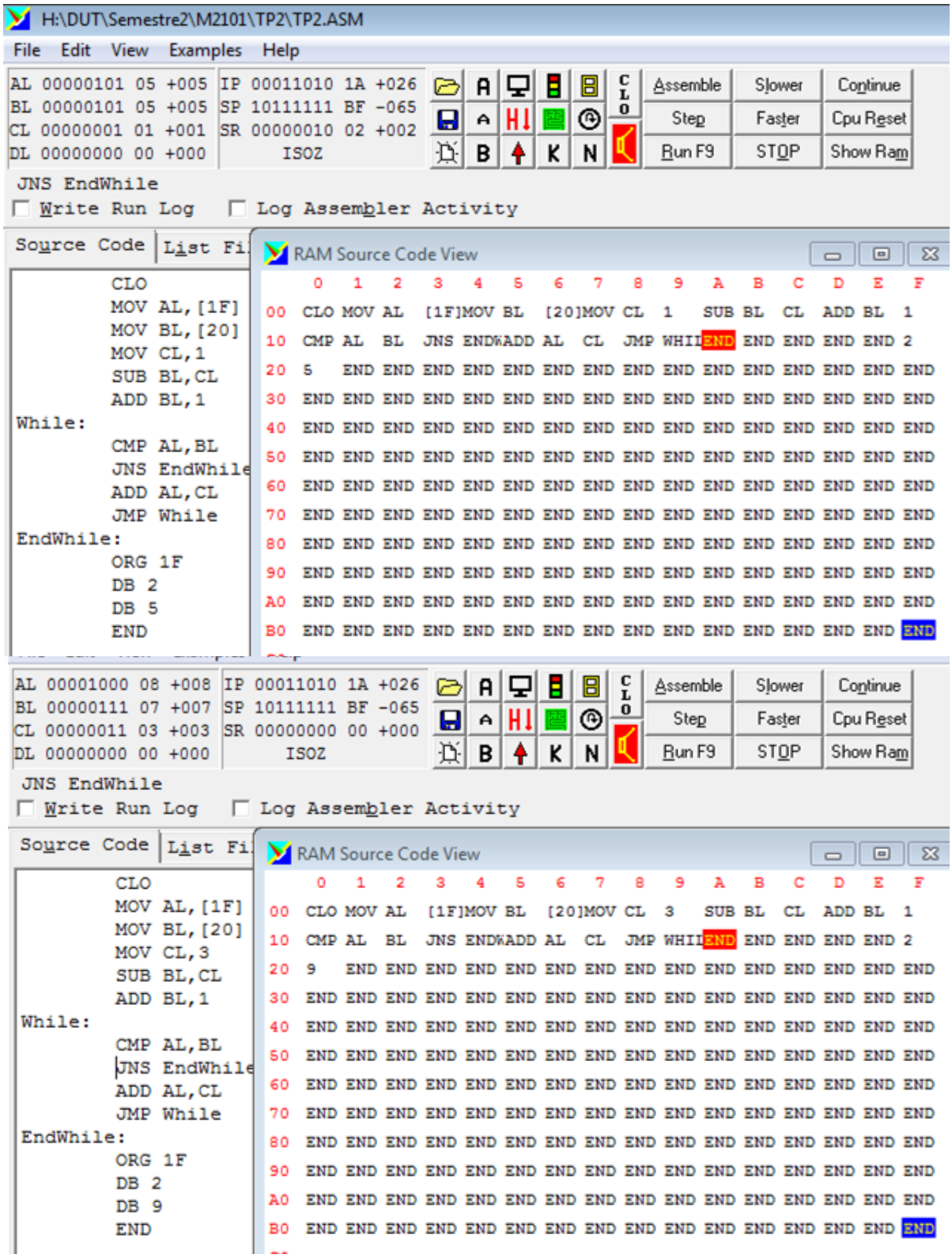




NB : à chaque phase de test observez bien le contenu de AL (c'est là que se construit le résultat) pour en vérifier la conformité avec l'énoncé (dépassement de la borne supérieure interdit). En cas de problème, passez à la question ci-après.

- Dans le cas du pas de 3, le problème de dépassement de la borne supérieure que vous avez dû observer est purement algorithmique : l'algorithme qui ne posait pas de problème par pas de 1 doit être modifié pour permettre un pas supérieur à 1. Pour ce faire il faut ajuster la borne supérieure avant la boucle par : **BL := BL - pas + 1** (hypothèse que BL reçoit RAM[1F] en début de programme).

  Corrigez le programme et vérifiez à nouveau son fonctionnement dans les 2 cas.



Exercice 2-3

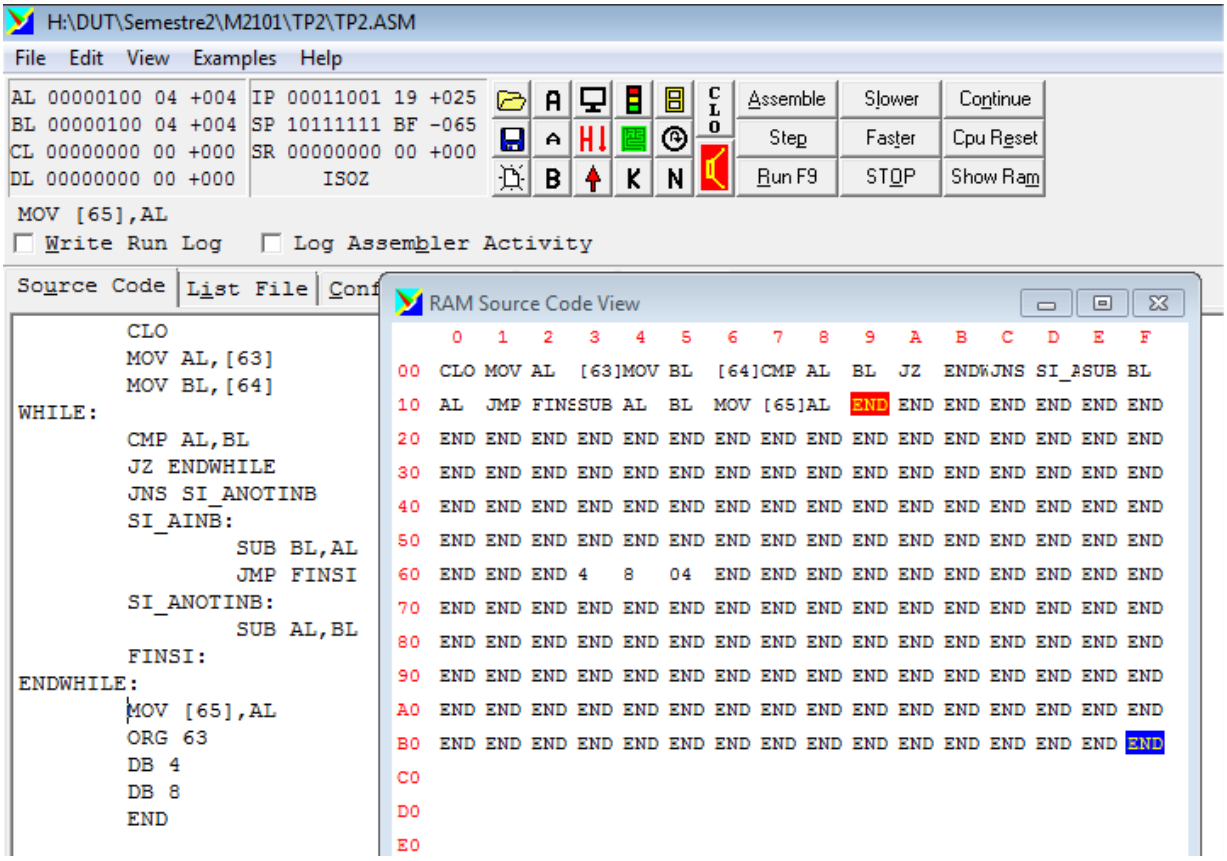
- Ecrivez un programme qui calcule le PGCD de 2 nombres initialement stockés en RAM aux adresses 100 et 101 et range le résultat à l'adresse 102. L'algorithme vous est donné ci-dessous ; aidez-vous également des exemples de traduction fournis en annexe.

Algorithme

```
TANT_QUE (A<>B)
        SI (A<B) ALORS
             B := B - A
        SINON
             A := A - B
        FINSI
FIN_TANT_QUE
PGCD := A                ; ou PGCD := B
```

- Faites-le fonctionner (jeu de tests au choix).

- De façon identique à ce que vous avez fait en TD, justifiez le calcul d'offset pour 2 instructions de branchement de votre choix (un cas de déplacement positif et un cas de déplacement négatif).



Calcul par table de référence :

Etat	Emplacement	Référence
ENDWHILE	16	0A
SI_ANOTINB	13	0C
FINSI	16	11

- 16-0A ⇔ 01 + (-0A) ⇔ 16 + F6 = 0001 0110 + 1111 0110 = 0000 1100 = 0C

0A = 0000 1010
-0A = 1111 0101 + 1 => 1111 0110 => F6

- 13-0C ⇔ 13 + (-0C) ⇔ 13 + F4 = 0001 0011 + 1111 0100 = 0000 0111 = 07

0C = 0000 1100
-0C = 1111 0011 +1 => 1111 0100 => F4

- 16-11 ⇔ 16 + (-11) ⇔ 16 + EF = 0001 0110 + 1110 1111 = 0000 0101 = 05

11 = 0001 0001
-11 = 1110 1110 +1 => 1110 1111 => EF

Calcul par les mots :

ENDWHILE => 11 mots ⇔ 0B +1 = 0000 1011 +1 = 0000 1100 => 0C

SI_ANOTINB => 6 mots ⇔ 06 +1 = 0000 0110 +1 = 0000 0111 => 07

FINSI => 4 mots ⇔ 04 + 1 = 0000 0100 +1 = 0000 0101 = 05