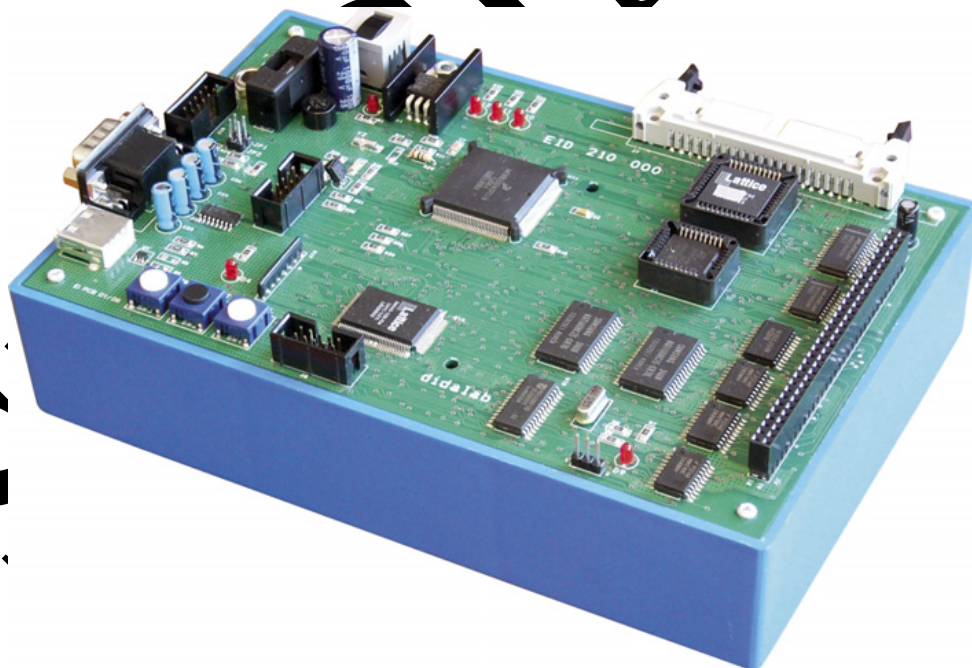


MANUEL de TRAVAUX PRATIQUES

Carte EID210 seule
Carte processeur à base du
microprocesseur 68332 (Cœur 68000)



SPECIMEN

SOMMAIRE

TP 0 : DECOUVERTE ET MISE EN ŒUVRE DU PACK LOGICIEL.....	5
<i>Avertissement.....</i>	<i>5</i>
0.1 ENONCE DU SUJET.....	5
0.2 MISE EN ŒUVRE.....	6
0.2.1 Installation matériel.....	6
0.2.2 Présentation du déroulement d'une phase complète de développement en assembleur.....	7
0.2.3 Démarrage du logiciel.....	8
0.2.4 Ouverture du fichier Assembleur « <i>tst_cpu.scr</i> »,.....	8
0.2.5 Visualisation du fichier « <i>tst_cpu.scr</i> ».....	9
0.2.6 Assemblage du fichier en ligne « <i>tst_cpu.scr</i> ».....	9
0.2.7 Configurer la vitesse de transmission.....	14
0.2.8 Configurer le moniteur.....	15
TP 1 : ECRITURE DANS UNE ZONE RAM.....	17
1.1 ENONCE DU SUJET.....	17
1.2 DETAIL DU CAHIER DES CHARGES.....	18
1.3 SOLUTION VARIANTE 1.....	19
1.3.1 Organigramme variante 1.....	19
1.3.2 Programme variante 1 en assembleur 68xxx.....	20
1.4 SOLUTION VARIANTE 2.....	21
1.4.1 Ordinogramme variante 2.....	21
1.4.2 Programme variante 2.....	22
TP 2 : COMMANDE DES DIODES SUR LE PORT "QS" DU MICRO-CONTROLEUR.....	23
2.1 ENONCE DU SUJET.....	23
2.2 SOLUTION.....	24
2.2.1 Analyse.....	24
2.2.2 Programme pour cahier des charges 2-1 en "Assembleur".....	25
2.2.3 Programme pour cahier des charges 2-1 en langage "C".....	26
2.2.4 Organigramme pour cahier des charges 2-2.....	27
2.2.5 Programme pour cahier des charges 2-2 en "Assembleur".....	28
2.2.6 Programme pour cahier des charges 2-2 en langage "C".....	29
2.2.7 Organigramme pour cahier des charges 2-3.....	30
2.2.8 Programme pour cahier des charges 2-3 en "Assembleur".....	31
2.2.9 Programme pour cahier des charges 2-3 en "C".....	32
TP 3 : REALISATION D'UN MODE "ECHO" A PARTIR DU TERMINAL.....	33
3.1 ENONCE DES SUJETS.....	33
3.2 SOLUTION: ENONCE A/.....	34
3.2.1 Détection de l'appui sur le bouton poussoir "CTRL".....	34
3.2.2 Utilisation de l'interface de communication série.....	34
3.2.3 Organigramme : Enoncé a/.....	35
3.2.4 Programme relatif à l'énoncé a/ en assembleur 68xxx.....	36
3.2.5 Programme relatif à l'énoncé a/ en langage "C".....	37
3.3 SOLUTION: ENONCE B/.....	38
3.3.1 Organigramme : Enoncé b/.....	38
3.3.2 Programme relatif à l'énoncé b/ en assembleur 68xxx.....	39
3.3.3 Programme relatif à l'énoncé b/ en langage "C".....	40

TP 4 : DONNER LA VALEUR D'UN REGISTRE SPECIFIE PAR L'UTILISATEUR	41
4.1 ENONCE DU SUJET	41
4.2 SOLUTION.....	42
4.2.1 Organigramme général.....	42
4.2.2 Organigrammes des sous programmes (ou fonctions).....	44
4.2.3 Programme en assembleur 68xxx.....	45
 TP 5 : ECRITURE OU LECTURE A UNE ADRESSE SPECIFIEE	 48
5.1 ENONCE DU SUJET	48
5.2 SOLUTION.....	49
5.2.1 Programme en assembleur 68xxx.....	50
5.2.2 Programme en "C"	55
 ANNEXE	 58
ANNEXE 1 FICHIER DE DEFINITIONS POUR PROGRAMMES EN ASSEMBLEUR	58
ANNEXE 2 FICHIER DE DEFINITIONS INCLUS DANS PROGRAMMES EN "C"	60

SPECIMEN

TP 0 : DECOUVERTE ET MISE EN ŒUVRE DU PACK LOGICIEL

Avertissement

Nota : La fiche de TP décrite ci-après n'a aucune ambition pédagogique, elle a simplement pour but d'aider l'utilisateur à la prise en main de l'ensemble logiciel et matériel EID210 Pack d'étude du microcontrôleur 68332. Elle est composée d'étapes successives très détaillées de la mise en œuvre du matériel et logiciel lors de la première utilisation.

0.1 Enoncé du Sujet

Objectifs :	<p>Démarrage de la carte mère 16/32 bits EID 210 000 à microcontrôleur 68332 :</p> <p>Mise en route, chargement d'un fichier, assemblage et vérification du fonctionnement en mode pas à pas d'un programme bouclé</p>
Cahier des charges :	<p>Utiliser un programme de quelques instructions effectuant les opérations suivantes :</p> <ul style="list-style-type: none"> ➤ Initialiser à 0 les registres d0, d1 et d2 ➤ Charger la valeur 5 en d0, 6 en d1, ➤ Additionner d0 et d1 avec résultat (B en hexadécimal) dans d0, ➤ Charger le mot long 2222 dans le registre d2 ➤ Reboucler le programme.

Matériel nécessaire :

Micro ordinateur de type PC sous Windows 95 ou ultérieur,

Carte mère 16/32 bits à microcontrôleur 68332, Réf : EID 210 000

Câble de liaison USB, ou à défaut câble RS232, Réf : EGD 000 003

Alimentation AC/AC 8V, 1 A Réf : EGD000001,

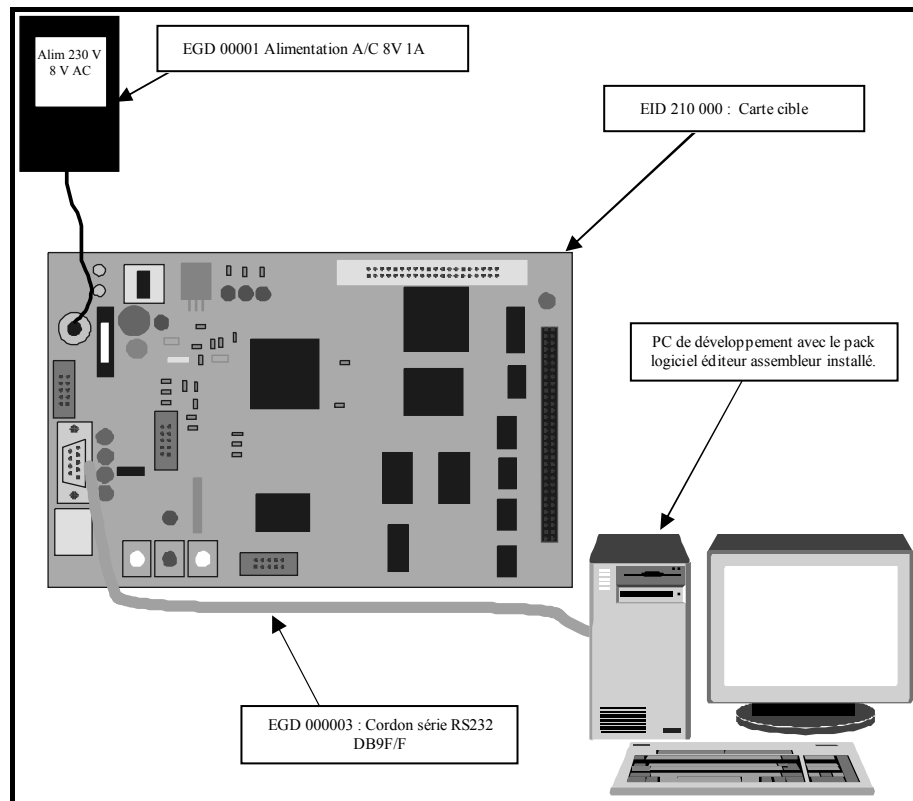
Fichier source assembleur fourni : «**tst_cpu.scr**»,

Durée : 2 heures

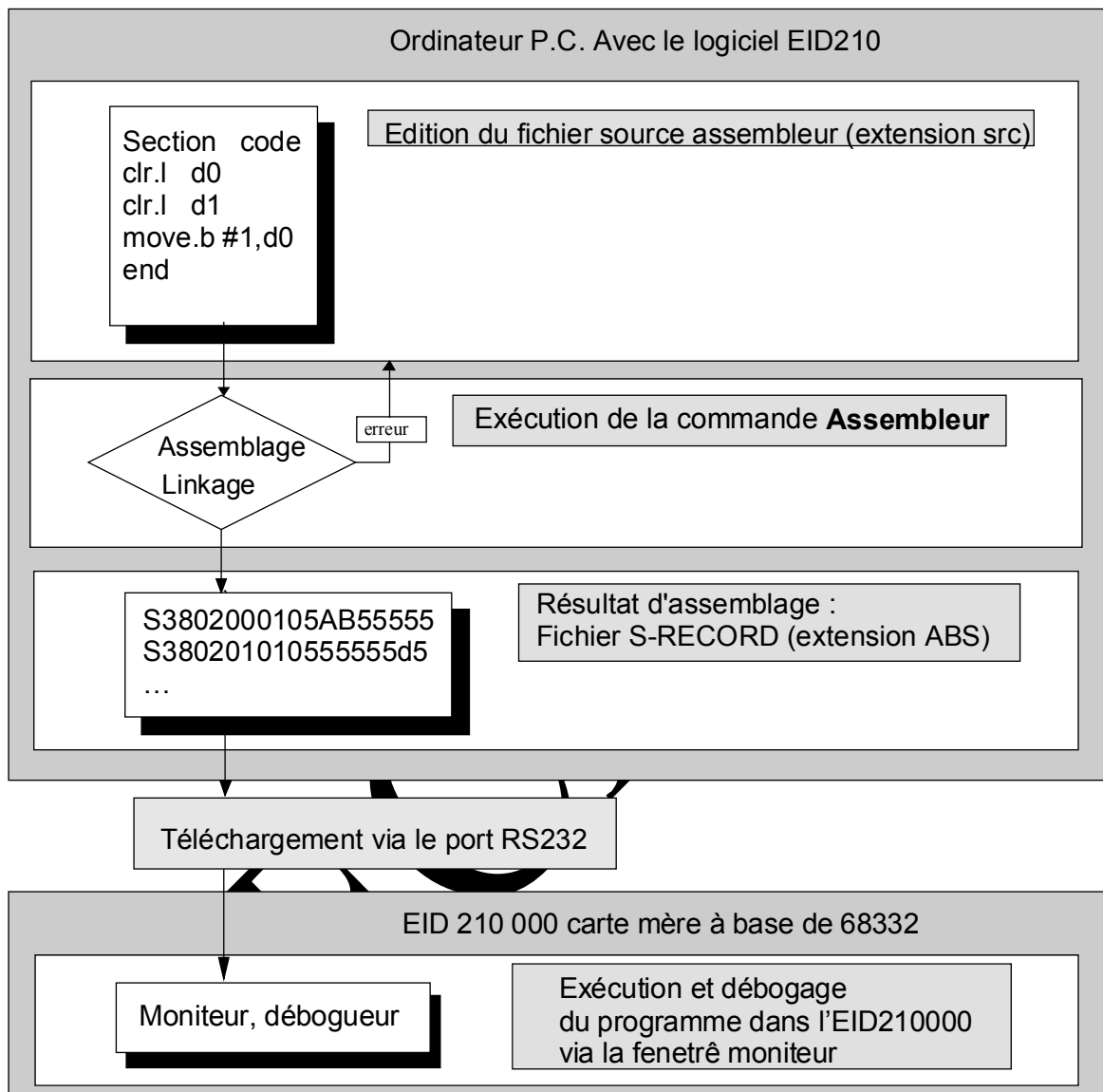
0.2 Mise en œuvre

0.2.1 Installation matériel

- ➔ Relier la carte EID 210 000 au PC de développement en assembleur (livré avec le matériel et préalablement installé conformément à la notice technique) par le câble USB ou par défaut par le câble série RS232
- ➔ Connecter le boîtier alimentation sur la carte EID 210 000, (7 à 12 V AC ou DC),
- ➔ Appuyer sur le bouton Marche Arrêt de la carte EID 210 000, la lampe témoin rouge doit s'allumer.



0.2.2 Présentation du déroulement d'une phase complète de développement en assembleur.



0.2.3 Démarrage du logiciel.

- ➔ Double cliquer sur l'icône « **Eid210** »

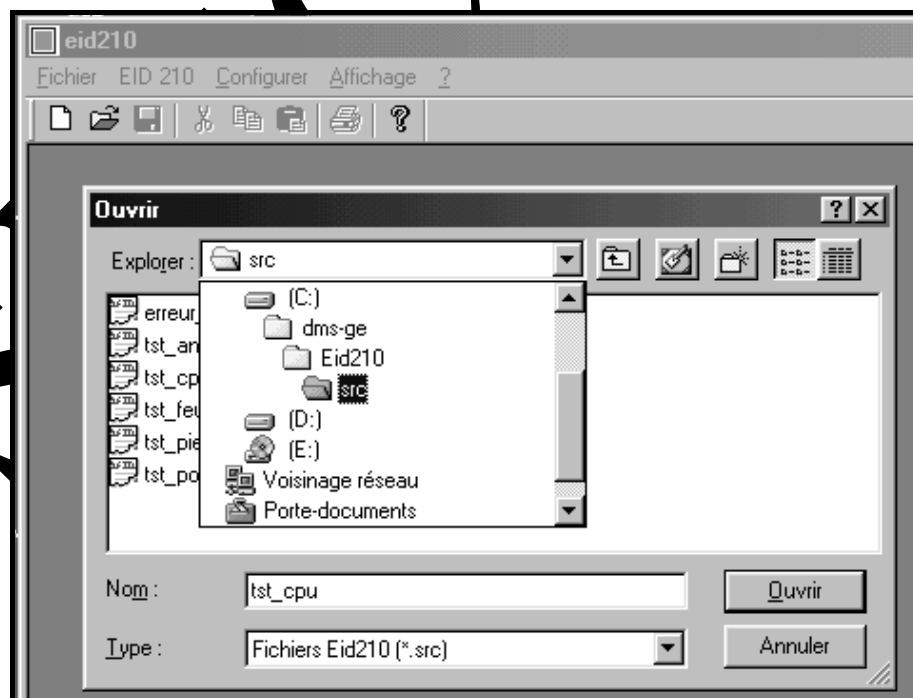


0.2.4 Ouverture du fichier Assembleur « *tst_cpu.src* »

- ➔ Cliquer sur « **Fichier** », puis sur « **Ouvrir** »

A l'aide de la fenêtre de l'Explorer, aller dans le repertoire : « *C:\dms-ge\Eid210\src* »

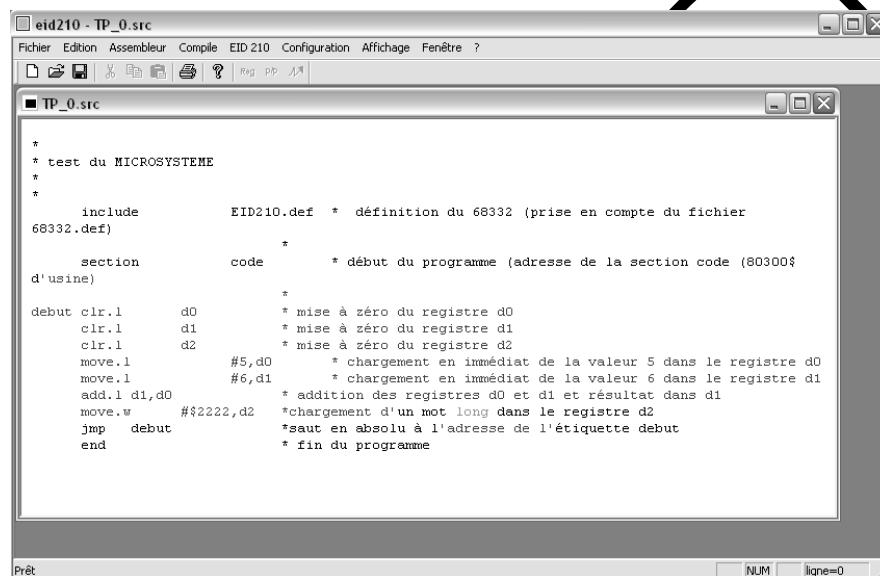
- ➔ Cliquer sur le fichier « *tst_cpu.src* », puis sur « **Ouvrir** »



0.2.5 Visualisation du fichier « TP_0.cr »

Après avoir cliquer sur ouvrir dans le chapitre précédent, le fichier apparaît comme ci-dessous.

- Il comprend :
- Une première zone de texte, identifiée par «* »=> commentaires,
 - La fonction « **include** » qui définit les registres du microcontrôleur 68332,
 - L'adresse de début du programme, « **section code** », définit par défaut à l'adresse \$803 000 en hexadécimale, (voir ANNEXE 2)
 - Le programme en assembleur , avec une zone « **étiquette** » placée à gauche de la fenêtre, une zone « **Instruction** », un zone « **opérande** », puis enfin une zone « **commentaire** » identifiée à nouveau par « * ».
 - Détail des instructions dans le dossier « **RESSOURCE** » du document.



```

*
* test du MICROSYSTEME
*
*
include      EID210.def * définition du 68332 (prise en compte du fichier
68332.def)
*
section      code      * début du programme (adresse de la section code ($80300$
d'usine)
*
debut clr.l   d0      * mise à zéro du registre d0
clr.l   d1      * mise à zéro du registre d1
clr.l   d2      * mise à zéro du registre d2
move.l   #5,d0    * chargement en immédiat de la valeur 5 dans le registre d0
move.l   #6,d1    * chargement en immédiat de la valeur 6 dans le registre d1
add.l   d1,d0    * addition des registres d0 et d1 et résultat dans d1
move.w   #$2222,d2 *chargement d'un mot long dans le registre d2
jmp      debut   *saut en absolu à l'adresse de l'étiquette debut
end            * fin du programme
  
```

0.2.6 Assemblage du fichier en ligne « tst_cpu.scr »

→ Cliquer sur « **Assembleur** »

L'ordinateur assemble le programme puis affiche le résultat d'assemblage, dans le cas présent : Fichier « tst_cpu.scr »

Nombre d'erreur(s) = « 0 »

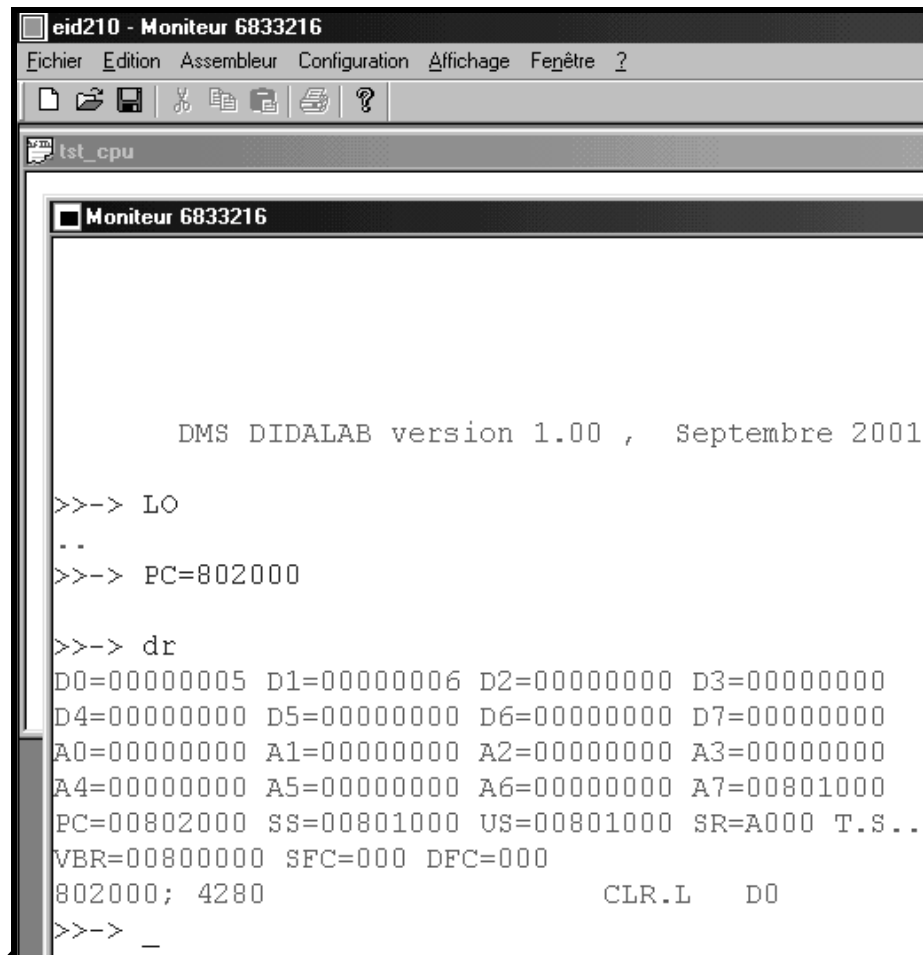
Nombre de warning(s) = « 0 »,

En cas de réponse de l'ordinateur « Le EID210 ne répond pas, se référer à l'annexe 1.

→ Cliquer sur « OK »

L'ordinateur télécharge le programme dans la carte cible EID 210 000, puis passe en mode moniteur.

→ Saisir « DR », puis retour chariot pour visualiser l'état des registres, accumulateurs de la CPU



```

eid210 - Moniteur 6833216
Fichier Edition Assembleur Configuration Affichage Fenêtre ?

tst_cpu

Moniteur 6833216

DMS DIDALAB version 1.00 , Septembre 2001

>>-> LO
..
>>-> PC=802000

>>-> dr
D0=00000005 D1=00000006 D2=00000000 D3=00000000
D4=00000000 D5=00000000 D6=00000000 D7=00000000
A0=00000000 A1=00000000 A2=00000000 A3=00000000
A4=00000000 A5=00000000 A6=00000000 A7=00801000
PC=00802000 SS=00801000 US=00801000 SR=A000 T.S..
VBR=00800000 SFC=000 DFC=000
802000; 4280 CLR.L D0
>>-> _
  
```

Nous pouvons observer les registres et accumulateurs du C.P.U., et principalement le Compteur ordinal pointé à l'adresse \$802000, la première instruction, code opératoire en hexadécimal « 4280 ». Avec la fonction de désassemblage nous pouvons lire « CLR.L D0 », initialisation à zéro de D0.

- ➔ Pour exécuter le programme en mode Pas à Pas saisir « SS », (Single Step), puis retour chariot.
- ➔ Pour exécuter un pas supplémentaire, renouveler le retour chariot.

```

Moniteur 6833216
>>-> =
D0=00000005 D1=00000006 D2=00000000 D3=00000000
D4=00000000 D5=00000000 D6=00000000 D7=00000000
A0=00000000 A1=00000000 A2=00000000 A3=00000000
A4=00000000 A5=00000000 A6=00000000 A7=00801000
PC=0080200A SS=00801000 US=00801000 SR=A000 T.S..0
VBR=00800000 SFC=000 DFC=000
80200A; D081                                ADD.L    D1,D0
>>-> =
D0=0000000B D1=00000006 D2=00000000 D3=00000000
D4=00000000 D5=00000000 D6=00000000 D7=00000000
A0=00000000 A1=00000000 A2=00000000 A3=00000000
A4=00000000 A5=00000000 A6=00000000 A7=00801000
PC=0080200C SS=00801000 US=00801000 SR=A000 T.S..0
VBR=00800000 SFC=000 DFC=000
80200C; 343C 2222                            MOVE.W    #$2222,D2
>>-> =
D0=0000000B D1=00000006 D2=00002222 D3=00000000
D4=00000000 D5=00000000 D6=00000000 D7=00000000
A0=00000000 A1=00000000 A2=00000000 A3=00000000
A4=00000000 A5=00000000 A6=00000000 A7=00801000
PC=00802010 SS=00801000 US=00801000 SR=A000 T.S..0
VBR=00800000 SFC=000 DFC=000
802010; 4EF9 00802000                        JMP        $802000
>>-> =

```

Nous pouvons constater et vérifier le déroulement du programme, conformément au source assembleur,

- D'abord, initialisation de D0, D1, D2,
- Chargement de 5 dans D0, 6 dans D1,
- Addition des 2 registres avec résultat dans D0,
- Chargement d'un mot long dans D2
- etc..

→ Pour visualiser le fichier listing,

- Cliquer sur « **Fichier** »
- Cliquer sur « **Ouvrir** »
- Dans la fenêtre type choisir « **Tous les fichiers(*.*)** »
- Cliquer sur le fichier « **tst_cpu.lis** »
- Cliquer sur « **Ouvrir** ».

```

eid210 - tst_cpu.lis
Fichier Edition Assembleur Configuration Affichage Fenêtre 2
tst_cpu.lis
t_cpu.lis
802000      1
802000      2 *
802000      3 * TEST DU MICROSYSTEME
802000      4 *
802000      5 *
802000      6 * FICHIER DE DÉFINITION DU 68332
802000      71 LIST00802000
802000      73 SECTION CODE
802000      74
802000 4280 | 75 DEBUT CLR.L D0 * MISE À ZÉRO DU REGISTRE D0
802002 4281 76 CLR.L D1 * MISE À ZÉRO DU REGISTRE D1
802004 4282 77 CLR.L D2 * MISE À ZÉRO DU REGISTRE D2
802006 7005 78 MOVE.L #5,D0 * CHARGEMENT EN IMMÉDIAT DE 5 DANS LE REGISTRE D0
802008 7206 79 MOVE.L #6,D1 * CHARGEMENT EN IMMÉDIAT DE 6 DANS LE REGISTRE D1
80200A D081 80 ADD.L D1,D0 * ADDITION DES REGISTRES D0 ET D1 ET RÉSULTAT DANS D1
80200C 343C 2222 81 MOVE.W #2222,D2 * CHARGEMENT D'UN MOT LONG DANS LE REGISTRE D2
802010 4EF9 00802000 82 JMP DEBUT * SAUT EN ABSOLU À L'ADRESSE DE L'ÉTIQUETTE DEBUT
802016      83 END * FIN DU PROGRAMME
warnings generated
No errors detected
  
```

Nous pouvons constater le listing de résultat d'assemblage contenant les adresses mémoires, les codes opératoires, opérandes et commentaires.

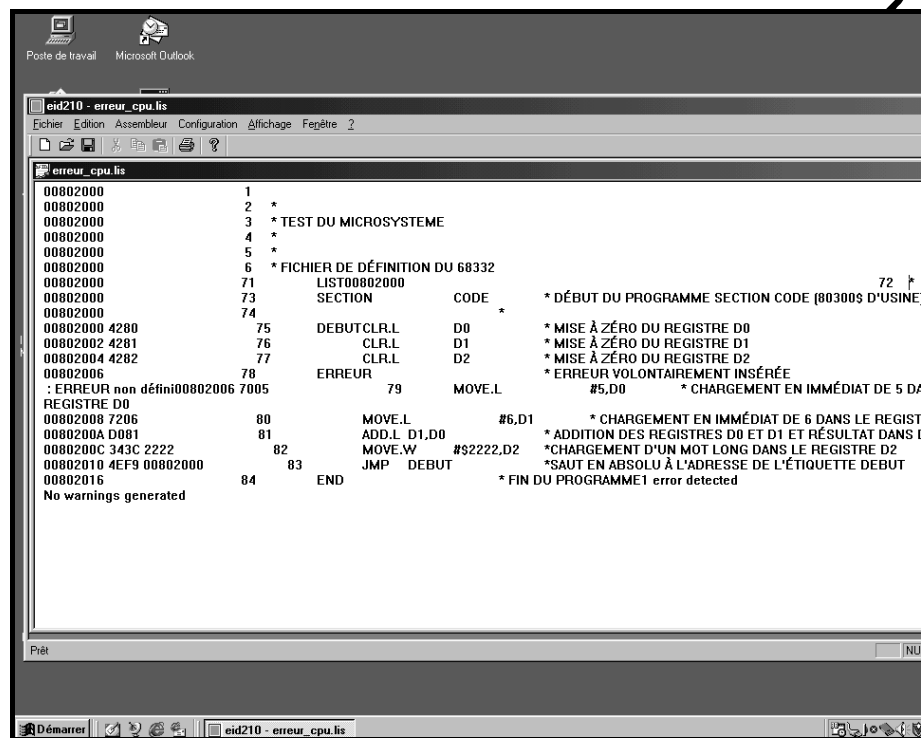
→ Cas d'un fichier comportant une erreur :

Reprendre les paragraphes 0.5 à 0.7, mais en utilisant le fichier source assembleur « **erreur.src** » à la place du fichier « **tst_cpu.src** ».

Durant la phase d'assemblage, l'assembleur signalera une erreur et refusera de commuter en mode moniteur.

➔ Pour visualiser l'anomalie ,

- Cliquer sur : « **Fichier** »,
- puis « **Ouvrir** »,
- puis « **tous types de fichiers (*.*)**,
- puis fichier « **erreur.lis** »,
- puis enfin « **Ouvrir** ».



```

eid210 - erreur_cpu.lis
Fichier  Edition  Assembleur  Configuration  Affichage  Fenêtre  2

erreur_cpu.lis
00802000      1
00802000      2 *
00802000      3 * TEST DU MICROSYSTEME
00802000      4 *
00802000      5
00802000      6 * FICHIER DE DÉFINITION DU 68332
00802000     71 LIST00802000
00802000     73 SECTION CODE * DÉBUT DU PROGRAMME SECTION CODE (80300$ D'USINE
00802000     74
00802000 4280 75 DEBUTCLR.L D0 * MISE À ZÉRO DU REGISTRE D0
00802002 4281 76 CLR.L D1 * MISE À ZÉRO DU REGISTRE D1
00802004 4282 77 CLR.L D2 * MISE À ZÉRO DU REGISTRE D2
00802006      78 ERREUR * ERREUR VOLONTAIREMENT INSÉRÉE
: ERREUR non défini00802006 7005 79 MOVE.L #5,D0 * CHARGEMENT EN IMMÉDIAT DE 5 DA
REGISTRE D0
00802008 7206 80 MOVE.L #6,D1 * CHARGEMENT EN IMMÉDIAT DE 6 DANS LE REGIST
0080200A D081 81 ADD.L D1,D0 * ADDITION DES REGISTRES D0 ET D1 ET RÉSULTAT DANS
0080200C 343C 2222 82 MOVE.W #S2222,D2 *CHARGEMENT D'UN MOT LONG DANS LE REGISTRE D2
00802010 4EF9 00802000 83 JMP DEBUT *SAUT EN ABSOLU À L'ADRESSE DE L'ÉTIQUETTE DEBUT
00802016      84 END * FIN DU PROGRAMME1 error detected

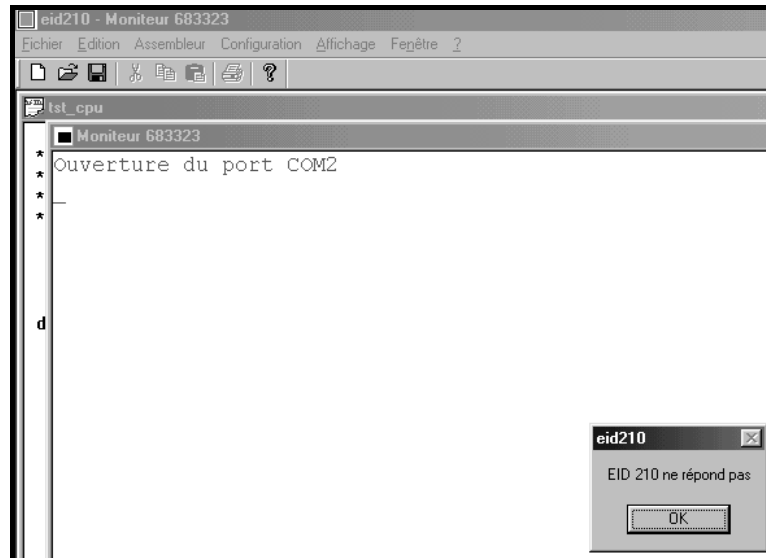
No warnings generated
  
```

Nous pouvons constater l'erreur signalée par l'assembleur entre les lignes de code :

- 00802006
- et
- 00802008, le mot « **erreur** » n'étant en aucun cas un code opératoire .

0.2.7 Configurer la vitesse de transmission

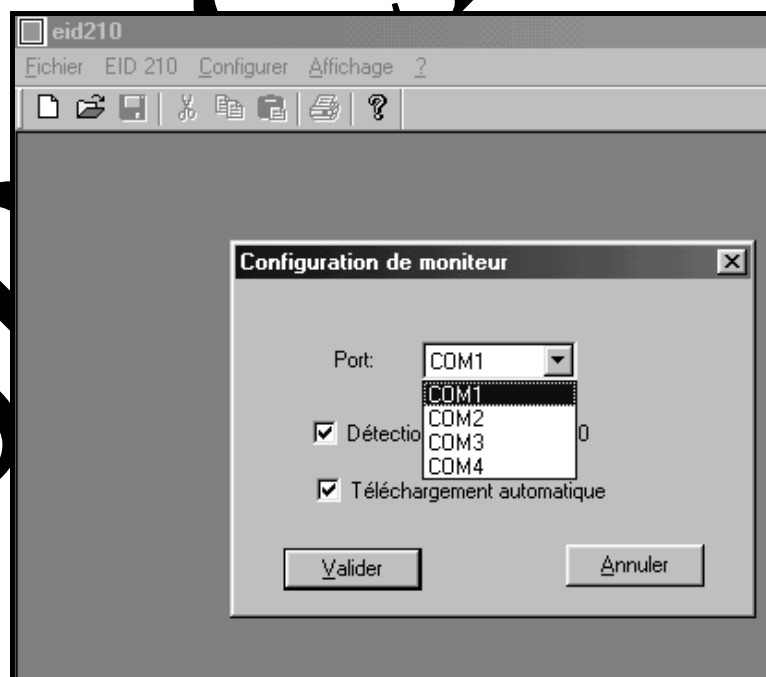
En cas de problème de communication entre la cible et la carte cible EID 100 000, comme indiquer ci-dessous,



Vérifier et paramétrer correctement la liaison série.

Lorsque tous les fichiers sont fermés,

Cliquer sur “**Configurer**”, puis “**Configuration de moniteur**”



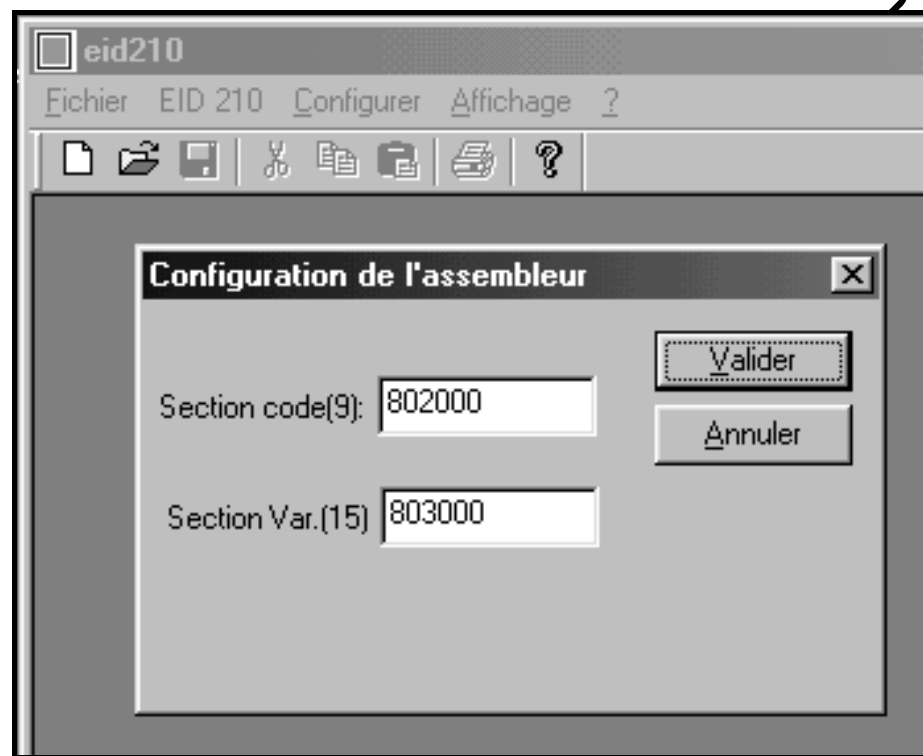
Activer la liaison série dans la fenêtre logiciel correspondante à la liaison matérielle utilisé, activer l’option “**Détection automatique**”, et “**Téléchargement automatique**”.

0.2.8 Configurer le moniteur

Configuration de l'assembleur:

Cliquer sur “**Configurer**”

Cliquer sur “**Assembleur**”.



- Section code (9) : Adresse de début du programme en mémoire vive de la carte cible par défaut (\$802 000).

- Section variable (15) : Adresse de début des variables utilisées dans le programme assembleur en mémoire vive de la carte cible, par défaut (\$803 000).

SPECIMEN

TP 1 : ECRITURE DANS UNE ZONE RAM

1.1 Enoncé du sujet

Objectifs :	Apprendre à manipuler une instruction de branchement conditionnel dans un programme en assembleur.
Cahier des charges :	<p>Ecrire un programme en assembleur qui va écrire l'alphabet de « A » à « Z » dans la zone mémoire commençant à l'adresse \$804000.</p> <ul style="list-style-type: none"> ➤ Sauvegarder les registres utilisés dans la pile r0, d0 ➤ Charger l'adresse de début \$804000 dans le registre r0, ➤ Initialiser le registre d0 avec la première lettre « A » ➤ Placer le caractère courant à l'adresse courante ➤ Préparer le caractère suivant, par incrémentation de d0, ➤ Tester si la lettre courante est « Z », ➤ Rebouclage si lettre courante # « Z », ➤ Restaurer le contexte des registres r0 et d0, ➤ Reboucler le programme, ➤ Fin de programme. <p>➤ A titre de variante, faire le même programme en utilisant la fonction dbf comme condition de rebouclage.</p>

Matériel nécessaire :

Micro ordinateur de type PC sous Windows 95 ou ultérieur,

Carte mère 16/32 bits microcontrôleur 68332, Réf : EID 210 000

Câble de liaison USB, ou par défaut câble RS232, Réf : EGD 000 003

Alimentation AC/AC 3V, 1 A Réf : EGD000001,

Durée : 2 heures

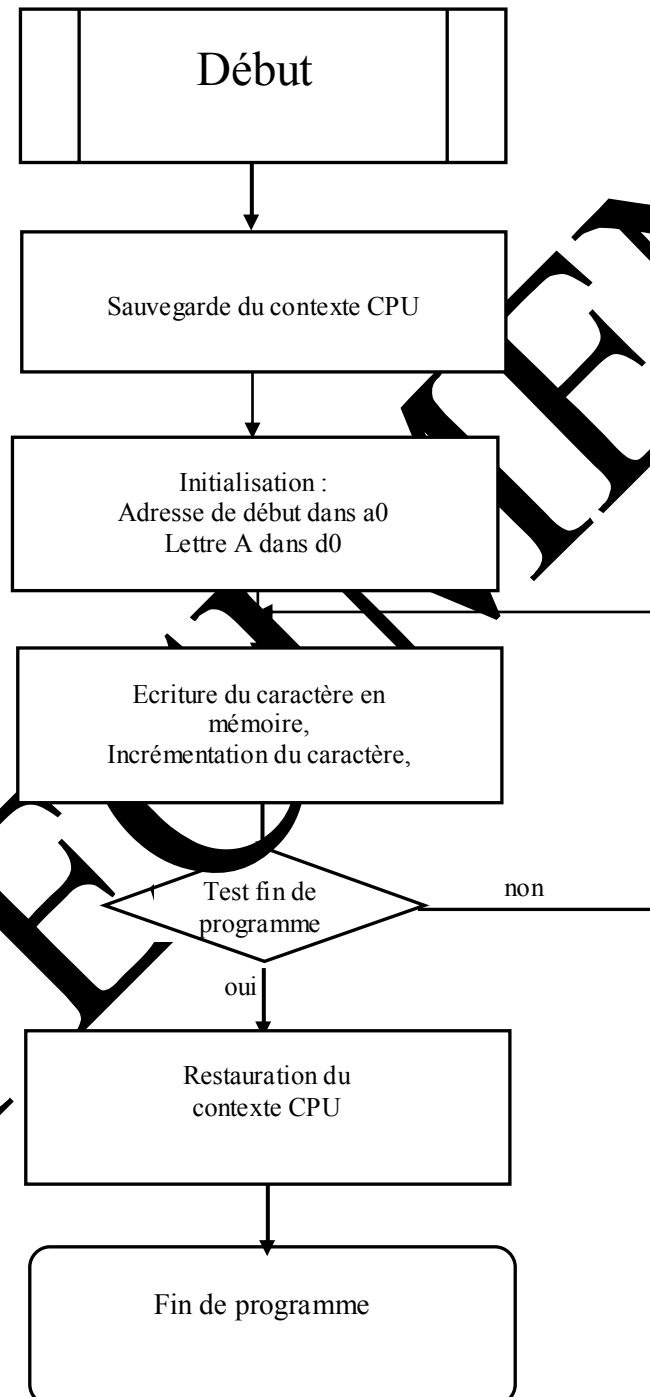
1.2 Détail du cahier des charges

- Sauvegarder les registres utilisés dans la pile : a0, d0
 - Charger l'adresse de début \$804000 dans le registre a0,
 - Initialiser le registre d0 avec la première lettre « A »,
 - Placer le caractère courant à l'adresse courante,
 - Préparer le caractère suivant, par incrémentation de d0,
 - Tester si la lettre courante est « Z »,
 - Rebouclage si lettre courante # « Z »,
 - Restaurer le contexte des registres a0 et d0,
 - Reboucler le programme,
 - Fin de programme.
-
- A titre de variante refaire le même programme en utilisant la fonction dbf comme condition de rebouclage.

SPECIMEN

1.3 Solution variante 1

1.3.1 Organigramme variante 1



1.3.2 Programme variante 1 en assembleur 68xxx

```

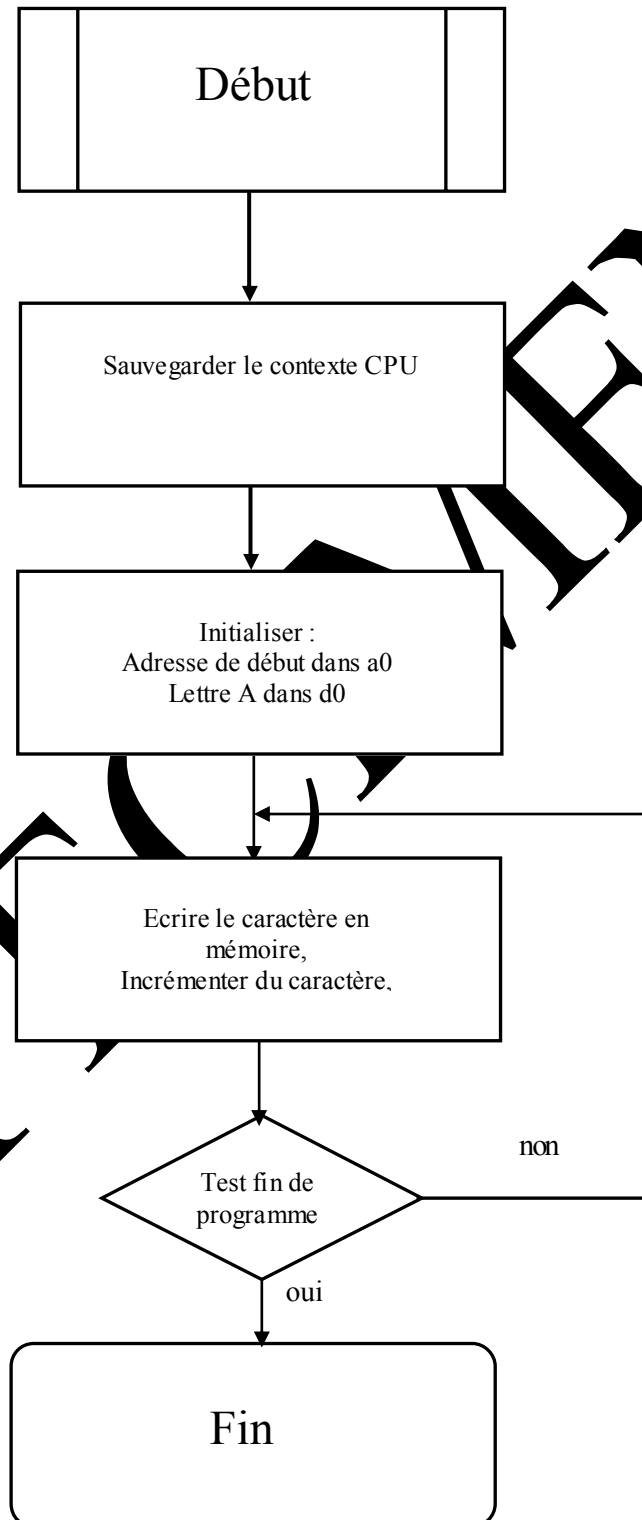
*****
*
*      TP SUR CARTE EID210 SEULE
*
*****
* Titre : Remplissage mémoire avec les lettres croissantes de l'alphabet.
* Langage: Assembleur croisé 68000 : Système: Pack EID 100 DMS DIDALAB
*****
*
*
*
*
*
*
*      include      EID210.def      * Définitions propre aux éléments de la carte processeur
*      section      code      $803000      * Début du programme section code ($803000)
*
*****
*
*      Initialisations,
*      les registres utilisés sont sauves dans la pile
*
*****
*
*      move.m.l      a0/d0,-(sp)      * sauvegarde des registres dans la pile
*      move.a.l      #$804000,a0      * Adresse de début d'écriture
*      move.b      #'A',d0      * Premier caractère (A) majuscule ASCII
*
*****
*
*      Remarque: pour mettre en minuscules il suffit d'initialiser d0 avec la valeur "a"
*      et de tester pour terminer la valeur "z"
*
*****
*
*      Début du programme principal
*
*****
boucle_1
    move.b      d0,(a0)+      * sauvegarde le caractère courant en mémoire
    addq.b      #1,d0      * Incrémenter d0 avant (incrémementation)
    cmp.b      #'z',d0      * test dernier caractère (comparaison avec "z")
    bls.s      boucle_1      * fin oui ?, si non retourner à boucle_1
    move.m.l      (sp)+,a0      * si non, restaure le contexte
    jmp      MONITEUR      * fin de programme et retour sous le controle du moniteur
end
* fin de programme

```

1.4 Solution variante 2

1.4.1 Ordinogramme variante 2

2



1.4.2 Programme variante 2

```

*****
*                               *
*      TP SUR CARTE EID210 SEULE      *
*                               *
*****
* Titre   : remplissage mémoire avec les lettres de l'alphabet.      *
* Langage: Assembleur croisé 68000   : Système: Pack EID 100 DMS DIDALAB *
*****

* une autre solution est possible en utilisant l'instruction de
* primitive de boucle dbcc 3 paramètres:
*
*      - 1 registre de données utilisé comme compteur,
*      - 1 (dé)branchement conditionnel
*      - 1 étiquette
*
*
*
*      include    EID210.def          * Définitions sur les éléments de carte
*
*      section    code    $803000    * Début du programme section code ($803000)
*
*
*****
*
*      Initialisations,
*      les registres utilisés sont sauves dans la pile
*
*****
*
*      move.m.l   a0/d0,-(sp)          * Sauvegarde des registres dans la pile
*      movea.l    #$804000,a0         * Adresse de début d'écriture
*      move.l     #25,d1              * Compteur avec nb lettres moins 1 !!
*      move.b     #'a',d0             * Première lettre de l'alphabet minuscule
*
*****
*
*      boucle du programme principal
*
*****
boucle_2:
    move.b       d0,(a0)              * Ecriture mémoire, + incrémentation de a0
    dbf.s        d1,boucle_2         * dbf, test d1=0, cond. tjrs fausse, sortie
    MONITEUR     d1                  * Lorsque le contenu de d1 est égal a -1
    * saut sous contrôle du moniteur
*
end                                  * fin de programme

```

TP 2 : COMMANDE DES DIODES SUR LE PORT "QS" DU MICRO-CONTROLEUR

2.1 Enoncé du sujet

<p>Objectifs:</p>	<p>Etre capable de commander les 3 diodes (repérées D10,D11et D12) connectées sur le port "QS" du microcontrôleur 68332.</p> <p>Etre capable de détecter l'appui sur la touche repérée "CTRL".</p> <p>Etre capable de mettre en œuvre le "Timer" interne au micro-contrôleur en mode d'interruption afin de réaliser une mesure de temps</p>
<p>Cahier des charges :</p>	<p>Sujet 2-1: Ecrire un programme en assembleur qui réalisera le cycle avec les trois leds connectées sur le port QS du microcontrôleur. Le passage d'un état à un autre se fait à chaque appui sur la touche "CTRL".</p> <p>En fait, on souhaite réaliser le cycle suivant:</p> <ul style="list-style-type: none"> ➤ Allumer la led repérée D10 (les 2 autres restent éteintes) ➤ Allumer la led repérée D11 (les 2 autres restent éteintes) ➤ Allumer la led repérée D12 (les 2 autres restent éteintes) ➤ Boucler <p>Sujet 2-2: On doit réaliser le même cycle que précédemment mais en automatique (sans avoir à appuyer sur la touche "CTRL"). Le passage d'un état à un autre se faisant au bout d'un laps de temps d'une seconde environ, réalisé par une temporisation de type "logiciel"</p> <p>Sujet 2-3: Idem cahier des charges précédent mais en utilisant le "Timer" interne au microcontrôleur en mode d'interruption.</p>

Matériel nécessaire :

Micro ordinateur de type PC sous Windows 95 ou ultérieur,
 Carte mère 16/32 bits à microcontrôleur 68332, Réf : EID 210 000
 Câble de liaison USB, ou à défaut câble RS232, Réf : EGD 000 003
 Alimentation AC/AC 8V, 1 A Réf : EGD000001,

Durée : 4 heures

2.2 Solution

2.2.1 Analyse

"Pilotage" des diodes électroluminescentes

Ces trois diodes D10, D11 et D12 sont connectées sur le port "QS" du microcontrôleur

- D10 sur la liaison PQS4
- D11 sur la liaison PQS5
- D12 sur la liaison PQS6

(Document ressource: Schémas structurels de la carte, "sheet 5 et 6")

Il faut tout d'abord configurer ces trois bits du port QS en sortie:

- ➔ écrire des 1 aux positions correspondantes du registre de contrôle du port QS
bit n°4 à 1; bit n°5 à 1; bit n°6 à 1
repères : 7654 3210
- ➔ le registre étant sur 16 bits cela donne 0000 0000 0111 0000 -> en Hexadécimal \$0070
- ➔ l'adresse de ce registre est définie dans le fichier de définition avec le label PQSCTR

Pour allumer une led il faut écrire un 0 dans le registre de données du port QS

- ➔ l'adresse de ce registre est définie dans le fichier de définition avec le label PORTQS
- ➔ pour allumer la led D10 uniquement, il faudra écrire 0000 0000 0110 0000

Détection de l'appui sur la touche "CTRL":

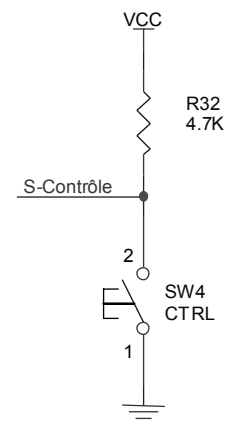
D'après le schéma ci-contre, l'appui sur le bouton poussoir "CTRL" entraîne un état logique '0' sur le signal "S-Contrôle".

L'état de ce signal "S-Contrôle" est accessible dans le registre d'état à l'emplacement de rang 8:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
----	----	----	----	----	----	---	---	---	---	---	---	---	---	---	---

Remarque:

- Le registre d'état est accessible grâce au label "REG_ETAT" dont l'adresse est définie dans le fichier d'inclure "EID210.def".
- Pour connaître l'état du bouton poussoir il suffit donc de lire le registre d'état et de faire un ET logique avec un masque de valeur: %0000 0001 0000 0000 = \$0100
- Si le résultat du masque donne \$0000, c'est qu'il y a appui sur la touche, par contre si le résultat donne \$0100, c'est que la touche est relâchée.



Réalisation d'une temporisation de type logiciel:

On réalise une temporisation logicielle en initialisant une variable à une certaine valeur et en décrémentant celle-ci jusqu'à ce qu'elle devienne nulle. La durée d'exécution de cette boucle de décrémentement constitue le laps de temps souhaitée. Dans le programme donné ci-après la variable est contenue dans le registre d0.

Réalisation d'une temporisation à l'aide du "Timer" interne au micro contrôleur:

Pour obtenir une interruption périodique toutes les 1 mS il faut initialiser les deux registres dont les labels ont été définis dans le fichier EID210.def:

"PICR" (Periodic Interrupt Control Register) à \$0760

"PITR" (Periodic Interrupt Timer Register) à \$0008.

Par ailleurs, il faudra initialiser la table des vecteurs et prévoir un programme d'interruption.

2.2.2 Programme pour cahier des charges 2-1 en "Assembleur"

```

*****
*                                     *
*          TP SUR CARTE EID210 SEULE          *
*                                     *
*****
*   Tester les 3 leds sur le port QS et l'entrée control "CTRL"   *
*                                     *
*   Cahier des charges:                                     *
*****
*   A chaque appui sur le bouton CTRL c'est une autre LED       *
*   qui s'allume soit le cycle D10 -> D11 -> D12 -> D10 ... etc   *
*   NOM du FICHIER: T_PQS.SRC                                     *
*****

* Inclusion du fichier définissant les différents labels
      include      EID210.def
      section      code
*
*****
*   INITIALISER   *
*****
* Configurer en sorties les 3 bits du port QS où sont connectées les diodes
debut      move.w      #$0070,PQSCTR      * 3 sorties sur LED
*****
*   BOUCLE PRINCIPALE   *
*****
* Allumer (par un 0) la LED de référence D10 connectée sur le bit 4 du port QS
DebBP      move.w      #$10,d0
AFF        move.w      d0,d1
           not.w        d1                * Complémenter pour allumer la LED
           and.w        #$0070,d1        * Ne valider que les 3 sorties leds
           move.w        d1,PORTQS        * Charger sur le port QS

* Détecter l'appui sur la touche "CTRL"

* Attendre tant que la touche "CTRL" est appuyée
ATT1       move.w      REG_ETAT,d2
           and.w        #$0100,d2
           beq          ATT1              Boucler si touche "CTRL" appuyée

* Attendre tant que la touche "CTRL" est libérée
ATT2       move.w      REG_ETAT,d2
           and.w        #$0100,d2
           bne          ATT2              * Boucler si touche "CTRL" libérée

* On a détecté l'appui sur la touche "CTRL"
* Passer à la LED suivante
           lsl          #1,d0
           bts          #7,d0            * Tester si on a dépassé
           beq          AFF              * Si on n'a pas dépassé, on affiche
           bra          DebBP            * Si on a dépassé on réinitialise

*
*   FIN de la boucle principale et du programme
*****
end      * Fin du fichier

```

2.2.3 Programme pour cahier des charges 2-1 en langage "C"

```

/*****
*
*      TP SUR CARTE EID210 SEULE
*
*****
*      Tester les 3 LEDs connectée sur le port "QS" et l'entrée "CTRL"
*
*
*      Cahier des charges:
*****
*      Le programme teste les 3 sorties utilisées DIO du port QS
*      A chaque appui sur le "switch" CTRL il y a permutation de l'état des leds
*      "D12 D11 D10"
*
*      Nom du fichier source en "c" :   T_portQS.C
*****
*****/

/* Liste des fichiers à inclure */
#include "Cpu_reg.h"
#include "EID210_reg.h"
#include "EID210_reg.h"
#include "Structures_donnees.h"
#include <stdio.h>

/* Déclaration des variables*/

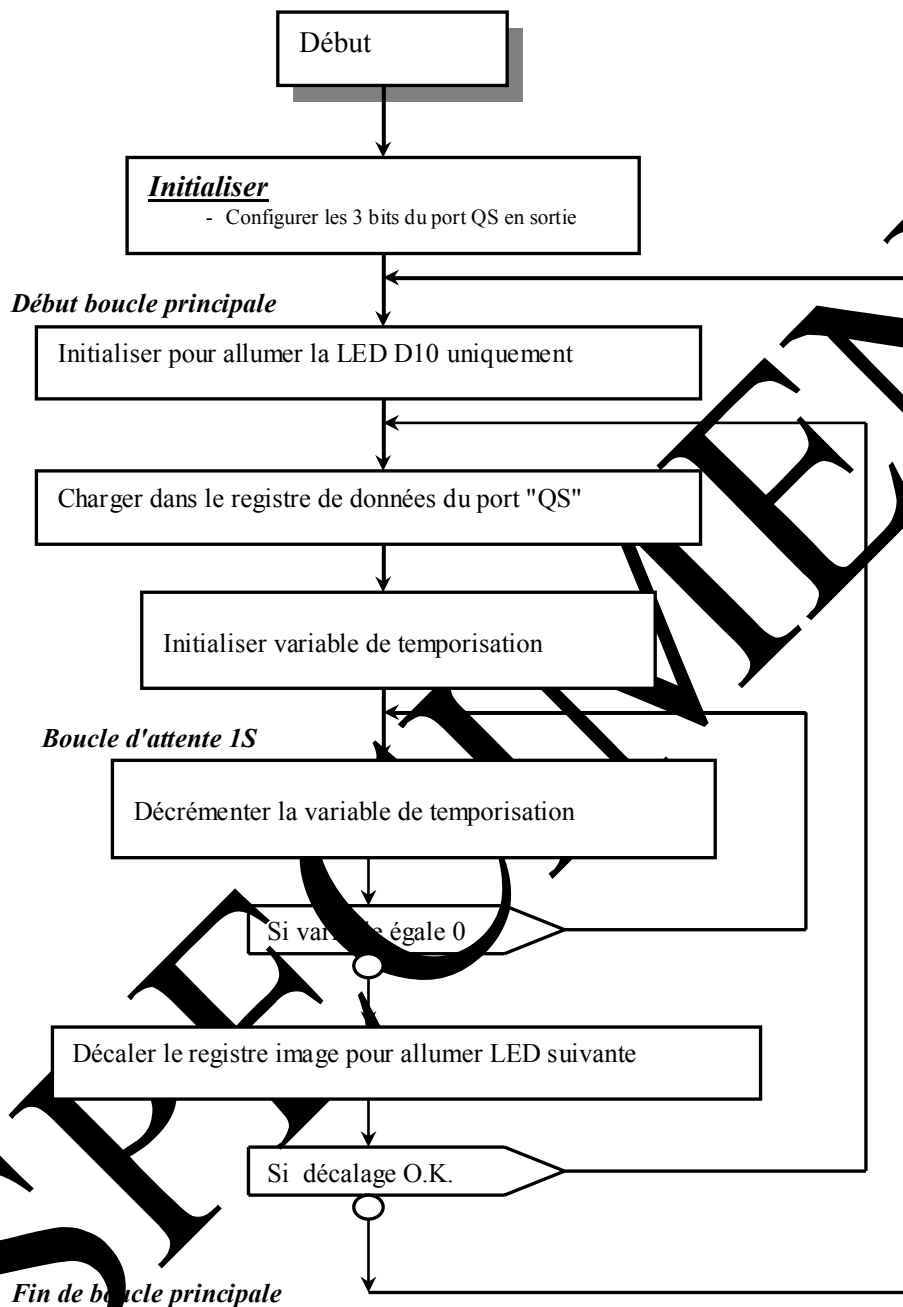
/*=====*/
/* Fonction principale */
/*=====*/
main()
{
/* Déclarations */
*****/
BYTE e_portqs;          /* Variable sur 8 bits donnant l'état du port QS donc des LEDs */
e_portqs=0x10;
/* Initialisations */
*****/
e_portqs=0x10;
PQSCTR=0x0070;          /* Les 3 sorties sur lesquelles sont connectées les leds*/
                          /* seront les sorties PQS[0..3) */
                          /* écriture immédiate */
PORTQS=e_portqs;
/* On écrit un message sur le moniteur */
printf("Test du port QS\n");
printf("Appuyer sur la touche CTRL pour faire une permutation\n");

/* Boucle principale */
*****
do
{
    printf("portqs=0x%2x\n",e_portqs);          /* Visualiser l'état du port QS sur le moniteur */
    e_portqs<<=1;                                /* Préparation etats futurs */
    if (e_portqs==0x80) e_portqs=0x10;
    while(S_Contrôle==1,
          PORTQS=e_portqs;
          while(S_Contrôle==0);
    }while(1);
    /* Attente appui sur touche CTRL */
    /* Chargement nouvelle valeur sur LEDS */
    /* Attente relachement de la touche CTRL */
    /* Fin de la boucle principale */
}

} /* Fin de la fonction principale */

```

2.2.4 Organigramme pour cahier des charges 2-2



2.2.5 Programme pour cahier des charges 2-2 en "Assembleur"

```

*****
*                                     *
*          TP SUR CARTE EID210 SEULE          *
*                                     *
*****
* Faire un chenillard avec les 3 leds sur le port QS          *
*                                     *
* Cahier des charges:                                         *
*****
* Les leds s'allument suivant le cycle                       *
*      D10 -> D11 -> D12 -> D10 ... etc                       *
* Le temps pendant lequel une led s'allume est d'environ 1S *
* Ce temps sera généré par une boucle "logiciel"             *
* NOM du FICHIER: CHENI_1.SRC                                *
*****
* Inclusion du fichier définissant les différents labels
include      68332.def

section      code

*      INITIALISER
*****
* Configurer en sorties les 3 bits du port QS où sont connectées les diodes
debut      move.w    #$0070, PQSCTR    * 3 sorties sur LED

*      BOUCLE PRINCIPALE
*****
* Allumer (par un 0) la LED de référence D10 connectée sur le bit 0 du port QS
DebBP      move.w    #$10,d0
ALUM       move.w    d0,d1
           not.w      d1                * Complémenter pour allumer sur un 0
           and.w      #$0070,d1        * Ne valider que les 3 sorties leds
           move.w     d1,PORTQS        * Charger sur le port QS

* Boucle d'attente d'environ 1 seconde
           move.l     #$001FFFFFF,d2
ATT        sub.l     #1,d2
           bne        ATT

* Passer à la LED suivante
           lsl        #1,d0
           btst       #1,d0            * Tester si on a dépassé
           beq        ALUM             * Si on n'a pas dépassé, on affiche
           bra        DebBP            * Si on a dépassé on réinitialise

* FIN de boucle principale et du programme
*****
end

```

2.2.6 Programme pour cahier des charges 2-2 en langage "C"

```

/*****
*
*      TP SUR EID210 seul
*
*****
* Faire un chenillard avec les 3 leds sur le port QS
*
* Cahier des charges
*****
* Les leds s'allument suivant le cycle
*      D10 -> D11 -> D12 -> D10 ... etc
* Le temps pendant lequel une led s'allume est d'environ 1S
* Ce temps sera généré par une boucle "logiciel"
* NOM du FICHIER: CHENI_1.C
*****
*****/

/* Liste des fichiers à inclure */
#include "Cpu_reg.h"
#include "EID210_reg.h"
#include <stdio.h>

/*=====*/
/* Fonction principale */
/*=====*/
main()
{
/* Déclarations */
*****/
BYTE e_portqs;          /* Variable sur 8 bits donnant l'état du port QS dont les LEDs */
unsigned int i;         /* Pour temporisation "logiciel" */

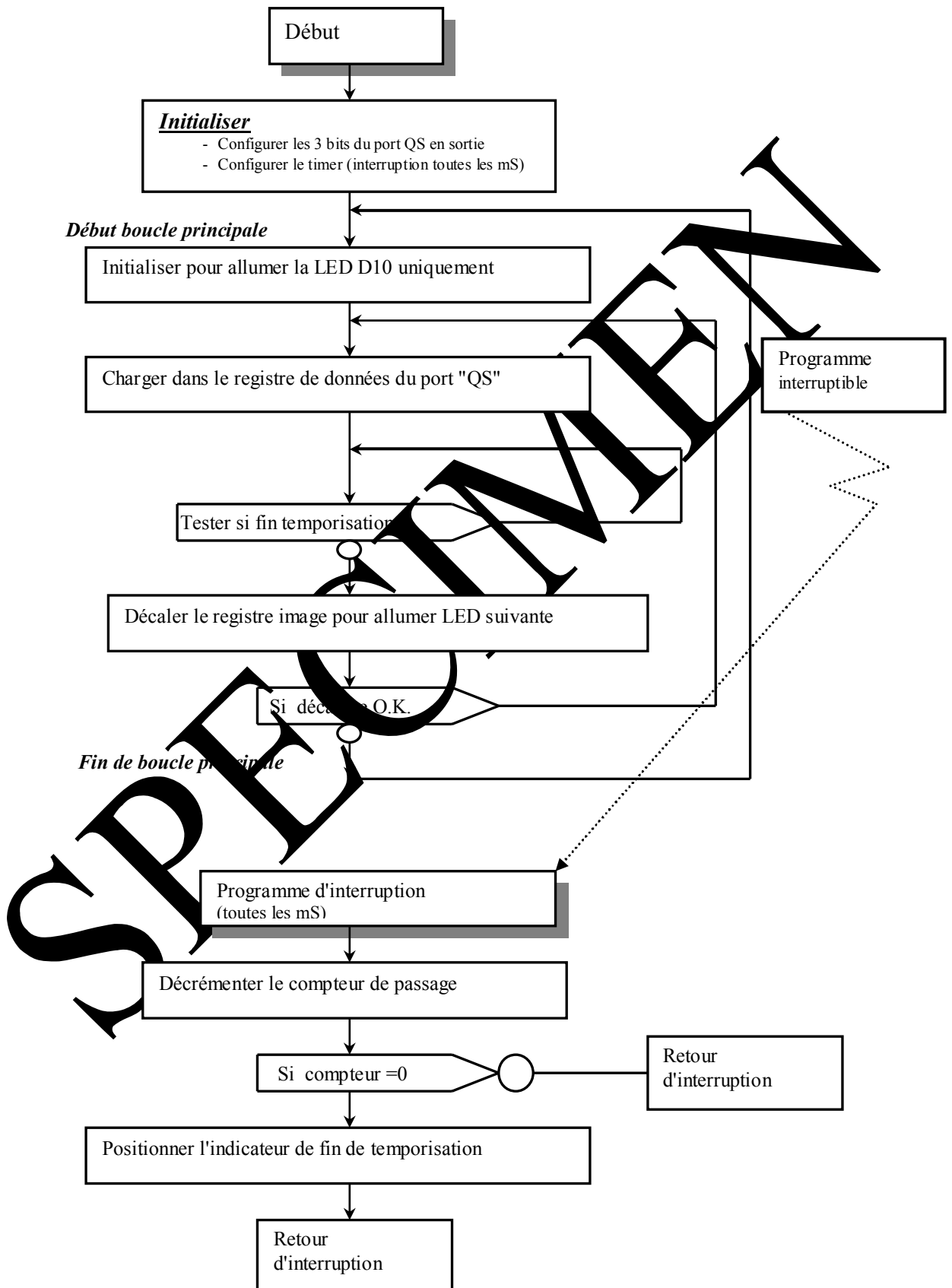
/* Initialisations */
*****/
e_portqs=0x10;
PQSCTR=0x0070;          /* Les 3 sorties sur lesquelles sont connectées les leds*/
                        /* seront des sorties PQS(0..3) */

/* On écrit un message sur le moniteur */
printf("Permutation des LEDs D12 D11 D10 à chaque second\n");
printf("    La temporisation est de type logiciel\n");
/* Boucle principale */
*****/
do
{
    /* Écriture de leur valeur sur le port QS */
    PORTC=e_portqs;
    /* Attendre environ 1S (temps logiciel) */
    for(i=0;i<=5000;i++);
    /* Préparation aux états futurs */
    e_portqs<<=1;
    if (e_portqs==0x80) e_portqs=0x10;
}while(1);                /* Fin de la boucle principale */

} /* Fin de la fonction principale */

```

2.2.7 Organigramme pour cahier des charges 2-3



2.2.8 Programme pour cahier des charges 2-3 en "Assembleur"

```

*****
*                               TP SUR CARTE EID210 SEULE                               *
*****
* Faire un chenillard avec les 3 leds sur le port QS                               *
*                               *                                                       *
* Cahier des charges:                                                         *
*****
* Les leds s'allument suivant le cycle                                         *
*   D10 -> D11 -> D12 -> D10 ... etc                                         *
* Le temps pendant lequel une led s'allume est d'environ 1S                 *
* Ce temps sera généré par la base de temps du 68332                         *
* NOM du FICHIER: CHENI_2.SRC                                                 *
*****
* Inclusion du fichier définissant les différents labels
include      EID210.def
* Déclaration des variables
*****
section      var
COMPTEUR     ds.l      1
INDICATEUR   ds.b      1
section      code
*****
* PROGRAMME PRINCIPALE
*****
* INITIALISER
*****
* Configurer en sorties les 3 bits du port QS où sont connectées les diodes
debut        move.w    #$0070,PQSCTR    * 3 sorties sur LED
* Configurer la base de temps
        move.l         #96,d0           * 96 est le n° du vecteur d'interruption
        asl.l          #2,d0
        add.l          #tab_vect,d0    * Initialiser la table des vecteurs
        move.l         d0,a0           * f_it_bt est l'adresse de la fonction d'interruption
        move.l         #it_bt,a1
        move.l         a1,(a0)
        move.l         #1000,COMPTEUR  * 1000*1mS =
        move.b         #$00,INDICATEUR * de fin de comptage
        move.w         #$0008,PICR     * 1 interruption toutes les 1 ms
        move.w         #$0760,PICR
* BOUCLE PRINCIPALE
*****
* Allumer (par un 0) la LED de référence D10 connectée sur le bit 4 du port QS
DebBP        move.w    #$10,ALUM
ALUM          move.w    d0,d1
        not.w          d1
        and.w          #0070,d1       * Complémenter pour allumer sur un 0
        move.w         d1,PORTQS      * Ne valider que les 3 sorties leds
        * Charger sur le port QS
* Boucle d'attente de fin de temporisation
ATT          move.b     INDICATEUR,D2
        cmp.b          #01,D2
        bne            ATT
        move.b         #$00,INDICATEUR
* Passer à la suivante
        lsl            #1,d0
        btst           #7,d0
        beq            ALUM
        bra            DebBP
        * Tester si on a dépassé
        * Si on n'a pas dépassé, on affiche
        * Si on a dépassé on réinitialise
* FIN de boucle principale, FIN du programme principal
*****
* FONCTION D'INTERRUPTION
* associée à la base de temps
*****
it_bt        sub.l      #$00000001,COMPTEUR
        cmp.l          #$00000000,COMPTEUR
        bne            it_ret
        move.b         #$01,INDICATEUR
        move.l         #1000,COMPTEUR
        * Retourner si pas égal à 0
        * C'est la fin tempo
        * Réinit tempo
        * Retour d'interruption
it_ret       rte
* Fin de la fonction d'interruption
*****
end
* Fin du fichier source

```

2.2.9 Programme pour cahier des charges 2-3 en "C"

```

/*****
*
* TP SUR EID210 seul
*
* Faire un chenillard avec les 3 leds sur le port QS
* Cahier des charges
*
* Les leds s'allument suivant le cycle
* D10 -> D11 -> D12 -> D10 ... etc
* Le temps pendant lequel une led s'allume est d'environ 1S
* Ce temps sera g n r  par le timer programmable
* g r  en interruption
* NOM du FICHIER: CHENI_2.C
*
*****/

/* Liste des fichiers   inclure */
#include "Cpu_reg.h"
#include "EID210_reg.h"
#include "Structures_donnees.h"
#include <stdio.h>
/* D claration des variables */
int CPTR_mS; /* d claration des variables et compteurs de millisecondes */
unsigned char Fin_tempo, Tempo_en_cours; /* d claration variables pour gestion temporisation */

/* FONCTION d'interruption pour temporisation (base de temps: toutes les 1ms)
=====
void irq_bt()
{
    if (Tempo_en_cours==1)
    {
        CPTR_mS++;
        if (CPTR_mS ==1000) Fin_tempo=1, Tempo_en_cours=0, CPTR_mS=0;
    }
} // Fin de la d finition de la fonction d'interruption pour la temporisation
/*****
/* Fonction principale */
/*****
main()
{
    /* D clarations */
    *****/
    BYTE e_portqs=0x10; /* Variable 8 bits donnant l' tat du port QS donc des LEDs */
    unsigned int i;
    /* Initialisations */
    *****/
    // Pour base de temps
    //*****
    CPTR_mS=0; /* Initialisation compteurs millisecondes
    SetVect(96,&irq_bt); /* mise en place de l'autovecteur
    PISR = 0x0008; /* Une interruption toutes les millisecondes
    PISR = 0x0700; /* 96 = 60H
    Tempo_en_cours=0; /* Temporisation inactive
    // Pour PORT QS sur lequel sont connect es les LEDs
    PQSCR=0x0070; /* 3 sorties sur lesquelles sont connect es les leds*/
    /* 3 sorties sur lesquelles sont connect es les leds*/
    /* seront des sorties PQS(6..3) */

    /* On  crit un message   l' cran */
    printf("Permutation des LEDs D12 D11 D10\n");

    /* Boucle principale */
    *****/
    do
    {
        //  criture Valeur sur le port QS
        PORTQS=e_portqs;
        // Attendre environ 1S (Tempo logiciel)
        Tempo_en_cours=1, Fin_tempo=0; // Initialisation pour tempo
        do{}while(Fin_tempo==0); // Attendre fin tempo
        // Pr paration aux  tats futurs
        e_portqs<<=1;
        if (e_portqs==0x80) e_portqs=0x10;
    }while(1); /* Fin de la boucle principale */

} // Fin de la fonction principale */

```


TP 3 : REALISATION D'UN MODE "ECHO" A PARTIR DU TERMINAL

3.1 Enoncé des sujets

Objectifs :	<p>Etre capable de configurer et d'utiliser la fonction de communication série RS232 (fonction interne au micro-contrôleur 68332), tout d'abord en mode "Emission" (liaison "simplex") puis en mode "Emission / Réception" (liaison "duplex").</p> <p>Etre capable de détecter une transition (changement d'état) sur une entrée logique.</p> <p>Etre capable de définir des constantes (message constant en caractères ASCII) et des variables.</p>
Cahier des charges :	<p>Enoncé a/ Envoyer un caractère prédéfini au terminal connecté à la liaison série RS232) à chaque fois que l'on appuie sur la bouton poussoir repéré "CTRL".</p> <p>Enoncé b/ Au lancement du programme, il y a envoi d'un message prédéfini (chaîne de caractère), ensuite, le programme réalise le mode "écho" : si on tape sur une touche du clavier de l'ordinateur, le caractère est renvoyé (il est affiché à l'écran).</p> <p><u>Remarque :</u> Dans ce cas la liaison sera de type "half duplex" car l'émission et la réception ne se font pas simultanément.</p>

Matériel nécessaire :

Micro ordinateur de type PC sous Windows 95 ou ultérieur,

Carte mère 16/32 bits à microcontrôleur 68332, Réf : EID 210 000

Câble de liaison USB, ou à défaut câble RS232, Réf : EGD 000 003

Alimentation AC/AC 8V, 1 A Réf : EGD000001,

Durée : 4 heures

3.2 Solution: Enoncé a/

3.2.1 Détection de l'appui sur le bouton poussoir "CTRL"

D'après le schéma ci-contre, l'appui sur le bouton poussoir "CTRL" entraîne un état logique '0' sur le signal "S-Contrôle".

L'état de ce signal "S-Contrôle" est accessible dans le registre d'état à l'emplacement de rang 8:

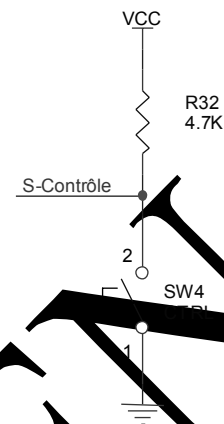
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
----	----	----	----	----	----	---	---	---	---	---	---	---	---	---	---

Remarque:

- Le registre d'état est accessible grâce au label "REG_ETAT" dont l'adresse est définie dans le fichier à inclure "EID210.def".

- Pour connaître l'état du bouton poussoir il suffit donc de lire le registre d'état et de faire un ET logique avec un masque de valeur: %0000 0001 0000 0000 = \$0100

Si le résultat du ET donne \$0000, c'est qu'il y a appui sur la touche, par contre, si le résultat donne \$0100, c'est que la touche est relâchée.



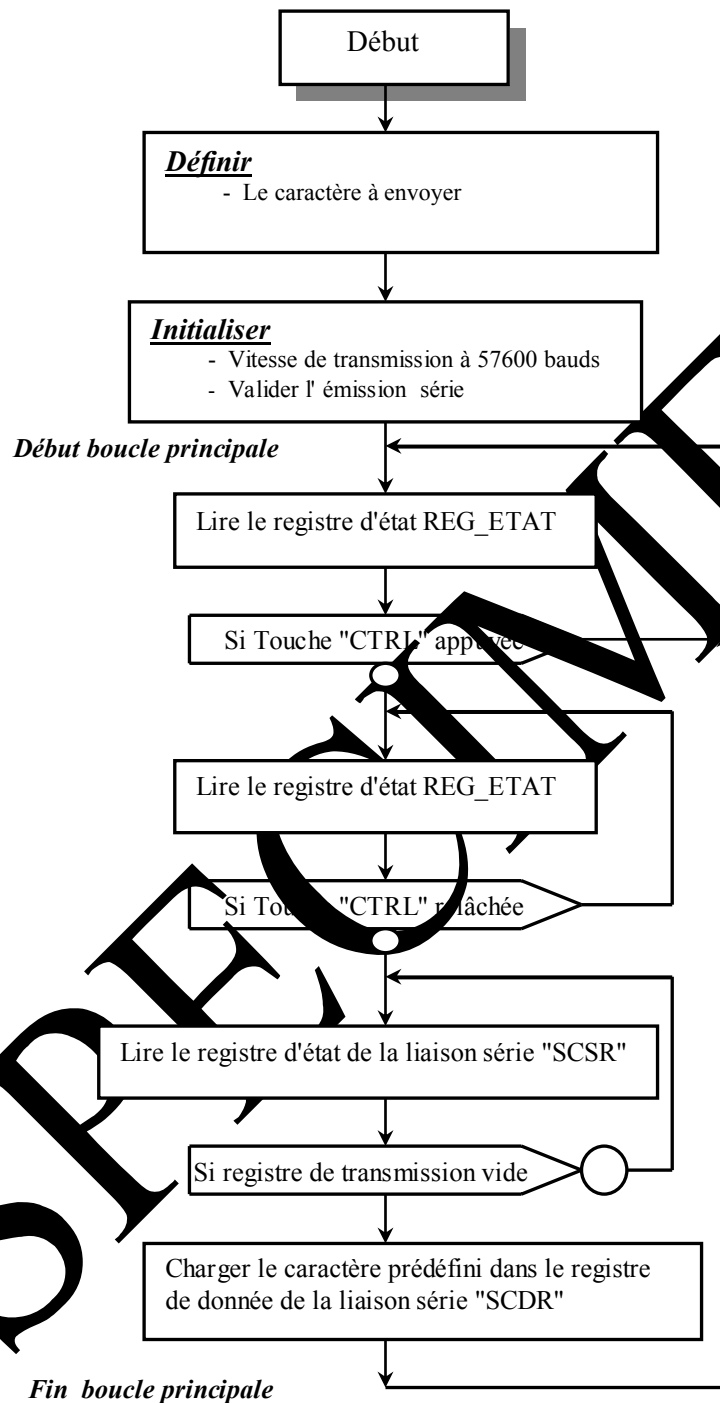
3.2.2 Utilisation de l'interface de communication série

L'utilisation de l'interface série s'effectue grâce à 4 registres de 16 bits dont les labels et les adresses ont été définis dans le fichier de définitions à inclure EID210.def

- ➔ Deux registres de contrôle (Serial Communication Control Register)
 - "SCCR0" pour définir la vitesse de communication, suivant la formule:

$$\text{Baud rate} = \frac{\text{Fréquence système}}{16 \times \text{data}}$$
 Avec "data" la valeur à charger dans le registre "SCCR0" et "Fréquence système", la fréquence de fonctionnement interne qui est un multiple de la fréquence du quartz connecté sur les entrées "XTAL" et "EXTAL" du micro-contrôleur. Pour satisfaire la vitesse de communication du moniteur (57600 Bauds) il faut initialiser ce registre à 2.
 - "SCCR1" pour définir le mode de fonctionnement:
 - bit de rang 2 : (RE Receive Enable) doit être positionné à 1 pour autoriser la réception,
 - bit de rang 3 : (TE Transmit Enable) doit être positionné à 1 pour autoriser la transmission.
 Il faut donc initialiser ce registre à %0000 0000 0000 1100 = \$000C.
- ➔ Un registre de donnée de label "SCDR" (Serial Communication Data Register)
 - Sous ce même label se cachent en fait deux registres, l'un servant à l'émission (accessible en écriture) et l'autre servant à la réception (accessible en lecture).
 - Pour envoyer un caractère via la liaison série, il suffit de charger son code ASCII dans le registre SCDR (à condition d'avoir vérifié préalablement qu'il soit vide) et pour recevoir un caractère il suffit de lire le code ASCII dans le registre SCDR (à condition d'avoir vérifié préalablement que celui-ci est plein).
- ➔ Un registre d'état de label "SCSR" (Serial Communication Status Register) dont:
 - * le bit de rang 8 ("TDRE" Transmit Data Register Empty) est à 1 lorsque le registre de donnée est vide, ce qui indique que l'on peut envoyer un caractère,
 - * le bit de rang 6 ("RDRF" Receive Data Register Full) est à 1 lorsque le registre de réception est plein, ce qui indique que l'on vient de recevoir un caractère et disponible en lisant le contenu du registre de donnée "SCDR".
 Des masques de labels "TDRE" et "RDRF" ont été définis dans le fichier EID210.def qui permettent de tester l'état de ces bits.

3.2.3 Organigramme : Enoncé a/



3.2.4 Programme relatif à l'énoncé a/ en assembleur 68xxx

```

*****
*                TP SUR CARTE EID210 SEULE                *
*****
*                ENVOYER UN CARACTERE SUR LE PORT          *
*                DE COMMUNICATION SERIE                    *
*  Cahier des charges:                                     *
*****
*  A chaque appui sur la touche repérée "CTRL" située sur la
*  carte EID210, il y a envoi d'un caractère qui doit apparaître
*  sur l'écran de l'ordinateur
*
*  NOM du FICHIER: T_SERIE1.SRC
*****
*  DEFINITION ET DECLARATIONS                             *
*****
*  Inclusion du fichier définissant les différents labels
*  include          EID210.def
*****
*  Déclaration du caractère à envoyer
*****
Char          EQU          $30          * On enverra le caractère 0 de ASCII (c'est $30)

section        code
*****
*  DEBUT DU PROGRAMME EXECUTABLE
*****
*  INITIALISER
*****
*  Les initialisations suivantes sont inhibées car le moniteur a déjà configuré le port série !
*  La vitesse de transmission
Debut          move.w      #9,SCCR0          * Pour avoir une vitesse de 57600 Bauds
* Valider emission (bit "RE")
               move.w      #S0004,SCCR1

*****
*  BOUCLE PRINCIPALE
*****
Deb_BP
* Détecter l'appui sur la touche "CTRL"
* Attendre tant que la touche "CTRL" est appuyée
ATT1          move.w      R_ESTAT,d0
               and.w      #100,d2
               beq         ATT1          * Boucler si touche "CTRL" appuyée

* Attendre tant que la touche "CTRL" est libérée
ATT2          move.w      R_ESTAT,d0
               and.w      #S0000,d2
               bne         ATT2          * Boucler si touche "CTRL" libérée

* On a Détecté l'appui sur la touche "CTRL"
* Attendre que la transmission est libre
ATT3          move.w      SCSR,d0
               and.w      #TDRE,d0
               beq         ATT3          * Acquérir le registre d'état de la liaison série
               * Bit indiquant si registre de transmission est vide
               * Transmit Data Register Empty
               * Boucler si transmission non prête
               move.w      #char,SCDR

               bra         Deb_BP          * Boucler
*****
*  FIN de boucle principale et du programme
*****
end          * Fin du fichier source assembleur
*****

```

3.2.5 Programme relatif à l'énoncé a/ en langage "C"

```

/*****
*
*      TP SUR EID210 seul
*
*****
*
*      ENVOYER UN CARACTERE SUR LE PORT
*      DE COMMUNICATION SERIE
*
*
*      Cahier des charges:
*****
*
*      A chaque appui sur la touche repérée "CTRL" située sur la
*      carte EID210, il y a envoi d'un caractère qui doit apparaître
*      sur l'écran de l'ordinateur
*
*
*      NOM du FICHIER: T_SERIE1.C
*****
*****/

/* Liste des fichiers à inclure */
#include "Cpu_reg.h"
#include "EID210_reg.h"
#include "Structures_donnees.h"
#include <stdio.h>

/*=====*/
/*      Fonction principale      */
/*=====*/
main()
{
/* Déclarations */
*****/
char caractere=0x30;      /* Ce sera le caractère '0' */

/* Initialisations */
SCCR0 = 9;                /* Pour la vitesse de transmission de 57600 bauds
SCCR1 = 0x000C; /* Pour valider l'émission et la réception

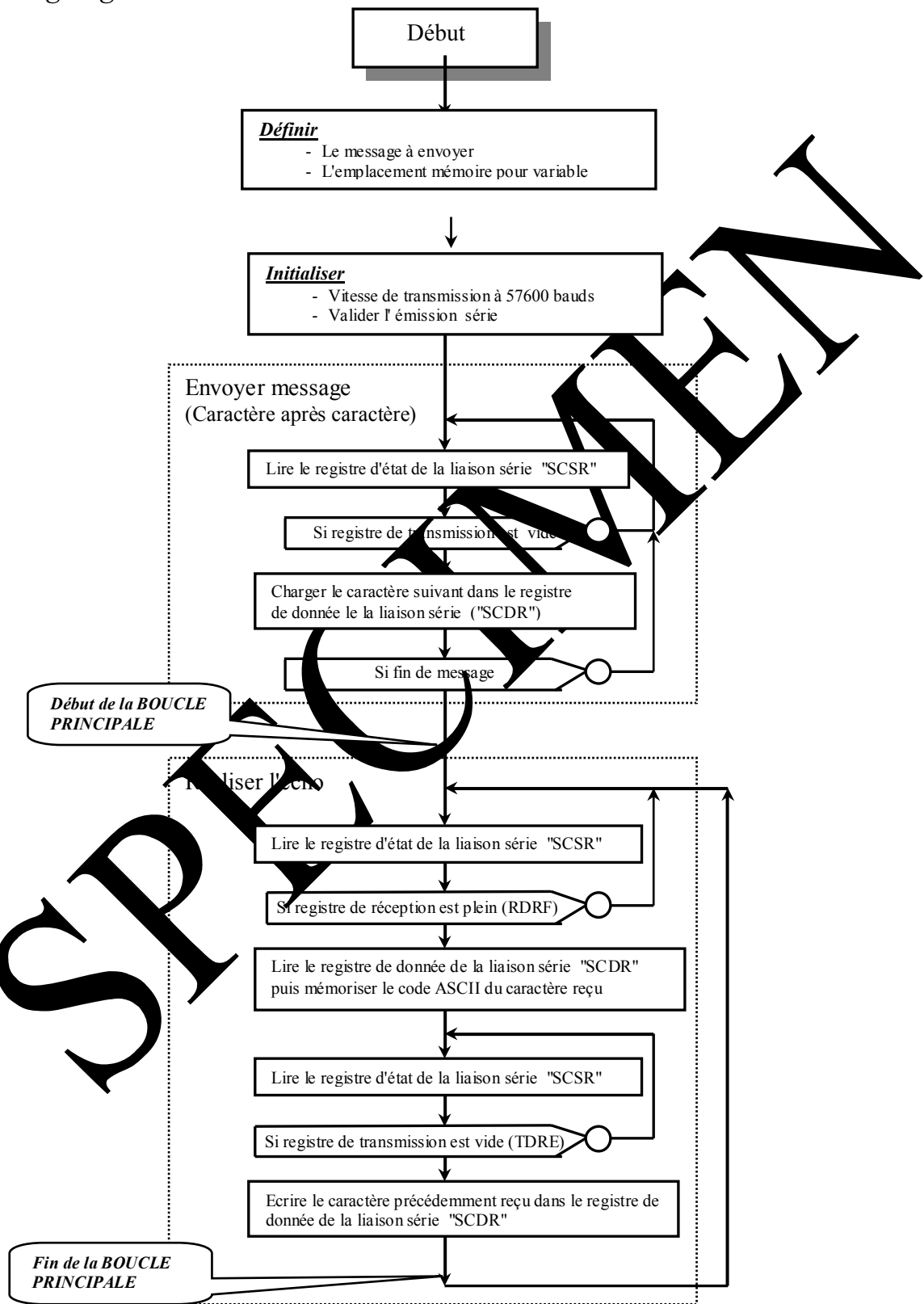
/* Boucle principale */
*****/
do
{
    while(S_Control==1); /* Attente appui sur touche CTRL */
    while(S_Control==0); /* Attente relachement de la touche CTRL */
    while(!(SCS==TX_EMPTY)); /* Attendre liaison série libre */
    SCDR=caractere; /* Envoyer le caractère sur la liaison série RS232 */
}while(1); /* Fin de la boucle principale */

} /* Fin de la fonction principale */

```

3.3 Solution: Enoncé b/

3.3.1 Organigramme : Enoncé b/



3.3.2 Programme relatif à l'énoncé b/ en assembleur 68xxx

```

*****
*                               TP SUR CARTE EID210 SEULE                               *
*****
*                               RENVoyer LE CARACTERE RECU SUR LE PORT                               *
*                               DE COMMUNICATION SERIE                               *
*   Cahier des charges:                                                           *
*****
*   - Au lancement du programme, il y a envoi d'un message                       *
*   - Ensuite, le programme réalise le mode "écho" :                             *
*   - si on tape un caractère sur le clavier de l'ordinateur, celui-ci nous revient, affiché à l'écran *
*   - NOM du FICHIER: T_SERIE2.SRC                                               *
*****
*   DEFINITION ET DECLARATIONS *
*****
* Inclusion du fichier définissant les différents labels
*   include      EID210.def
*****
*   Déclaration des variables *
*****
*   section      var
Message      dc.b      ' BONJOUR!  Tapez sur une touche, le caractère doit s'afficher '
Char         ds.w      1      * Pour mémoriser le caractère reçu
*   section      code
*****
*   DEBUT DU PROGRAMME EXECUTABLE *
*****
*   INITIALISER
*****
*   La vitesse de transmission
Debut        move.w    #9,SCCR0      * Pour avoir une vitesse de 57600
*   Valider émission et réception
                move.w    #$00C,SCCR1
                move.l    #0,d1      * Dans d1, le nombre de caractères envoyé
                move.l    #Message,A1      * Dans A1, l'adresse de début message
*   Envoi du message
ATT1          Attendez transmission libre
                move.w    SCSR,d0      * Acquérir le registre d'état de la liaison série
                and.w     #TDRE,d0      * Bit indiquant si registre de transmission est vide
                beq       ATT1          * Boucler si transmission non prête
                move.b     (A1),d0
                move.w     d0,SCDR
                add.l      #1,d1      * Passer au caractère suivant
                add.l      #1,A1      * Test si envoi message terminé
                cmp.l      #66,d1      * il y a 66 caractères dans le message
                bne       ATT1
ATT2          move.w     SCSR,d0      * Acquérir le registre d'état de la liaison série
                and.w     #TDRE,d0      * Bit indiquant si registre de transmission est vide
                beq       ATT2          * Boucler si transmission non prête
                move.w     #0D,SCDR      * 0D est le code ASCII du CR "Retour chariot"
ATT3          move.w     SCSR,d0
                and.w     #TDRE,d0
                beq       ATT3          * Boucler si transmission non prête
                move.w     #0A,SCDR      * 0A est le code ASCII du LF "Sauter ligne"
*   Boucle principale
*****
*   Attendez réception caractère
Deb_BP        move.w    SCSR,d0      * Acquérir le registre d'état de la liaison série
                and.w     #RDRE,d0      * Bit indiquant si registre de réception est plein
                beq       Deb_BP        * Recieve Data Register Full
                move.w     SCDR,char      * On récupère le caractère reçu
*   Attendez transmission prête
AT2           move.w     SCSR,d0      * Acquérir le registre d'état de la liaison série
                and.w     #TDRE,d0      * Bit indiquant si registre de transmission est vide
                beq       AT2           * Transmit Data Register Empty
                * Renvoyer le caractère reçu
                move.w     char,SCDR
                bra       Deb_BP        * Boucler
*   FIN de boucle principale et du programme
*****
end
* Fin du fichier source assembleur

```

3.3.3 Programme relatif à l'énoncé b/ en langage "C"

```

/*****
*      TP SUR EID210 seul
*****
*      RENVoyer LE CARACTERE RECU SUR LE PORT
*      DE COMMUNICATION SERIE
*
*      Cahier des charges:
*****
*      - Au lancement du programme, il y a envoi d'un message
*      - Ensuite, le programme réalise le mode "écho" :
*      si on tape un caractère sur le clavier de l'ordinateur,
*      celui-ci nous revient, affiché à l'écran
*
*      NOM du FICHIER: T_SERIE2.C
*****
*****/

/* Liste des fichiers à inclure */
#include "Cpu_reg.h"
#include "EID210_reg.h"
#include "Structures_donnees.h"
#include <stdio.h>

/*=====*/
/*      Fonction principale      */
/*=====*/
main()
{
/* Déclarations */
*****/
char caractere;

/* Initialisations */
SCCR0 = 9;           // Pour la vitesse de transmission de 9600 bauds
SCCR1 = 0x000C;      // Pour valider l'émission et la réception
printf("Realisation de la fonction ECHO : Entrez des caractères au clavier \n");

/* Boucle principale */
*****/
do
{while(!(SCSR&RDRF)); /* Attendre la réception d'un caractère */
  caractere=SCDR; /* Récupérer le caractère reçu */
  while(!(SCSR&TDRE)); /* Attendre liaison série libre */
  SCDR=caractere; /* Renvoyer le caractère sur la liaison série RS232 */
}while(1); /* Fin de la boucle principale */

} /* Fin de la fonction principale */

```


TP 4 : DONNER LA VALEUR D'UN REGISTRE SPECIFIE PAR L'UTILISATEUR

4.1 Enoncé du sujet

<p>Objectifs :</p>	<ul style="list-style-type: none"> - Etre capable de configurer et d'utiliser la fonction de communication série RS232 (fonction interne au micro-contrôleur 68332), en mode Émission-Réception" (liaison "duplex"). - Etre capable d'acquérir un caractère, de tester sa cohérence, puis d'exécuter une action prédéfinie (répondre par un message prédéfini) . - Etre capable de convertir un mot binaire sur 16 bits en 16 caractères ASCII. - Etre capable de structurer un programme en faisant appel, pour des actions répétitives à des sous programmes (assembleur) ou fonctions (langage 'C').
<p>Cahier des charges :</p>	<p>Initialiser les registres de donnée non utilisés dans le programme à des valeurs remarquables, et ce, sur 16 bits: D2 = \$2222 , D3 = \$3333 etc ...</p> <p>Au lancement du programme, il y a envoi d'un message prédéfini (chaîne de caractères): ' NUMERO DU REGISTRE ? de 2 a 7 '</p> <p>Lorsque l'utilisateur tape le numéro du registre dont il désire connaître la valeur, le programme contrôle le code qu'il reçoit, envoie le message : "NUMERO DE REGISTRE NON VALIDE" si erreur , sinon lit le registre spécifié puis envoie la réponse sous la forme: d = xxxxxxxxxxxxxxxx (avec des différents états binaires). On retourne ensuite au départ (demande de numéro de registre).</p>

Matériel nécessaire :

Micro ordinateur de type PC sous Windows 95 ou ultérieur,

Carte mère 16/32 bits à microcontrôleur 68332, Réf : EID 210 000

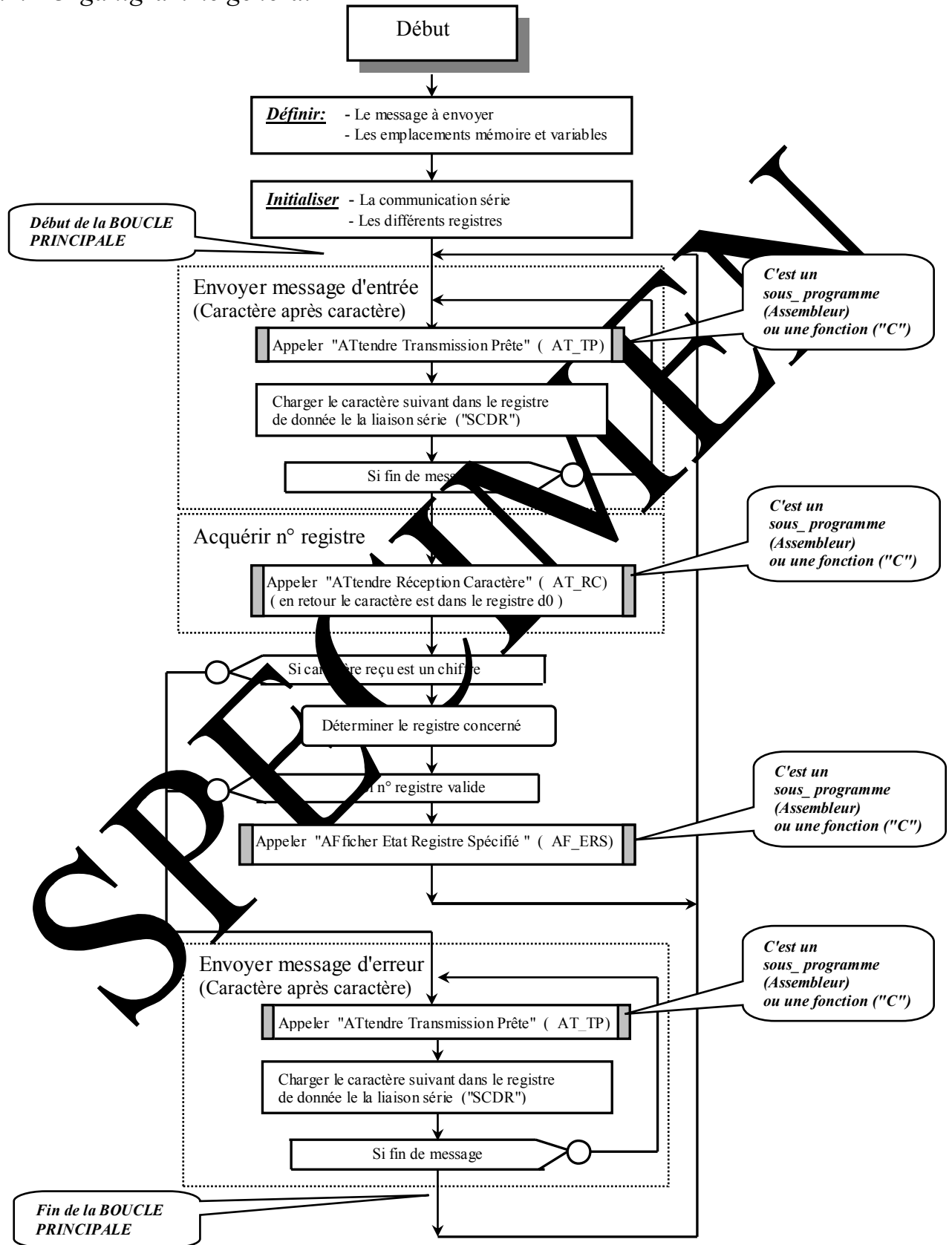
Câble de liaison USB, ou à défaut câble RS232, Réf : EGD 000 003

Alimentation AC/AC 8V, 1 A Réf : EGD000001,

Durée : 4 heures

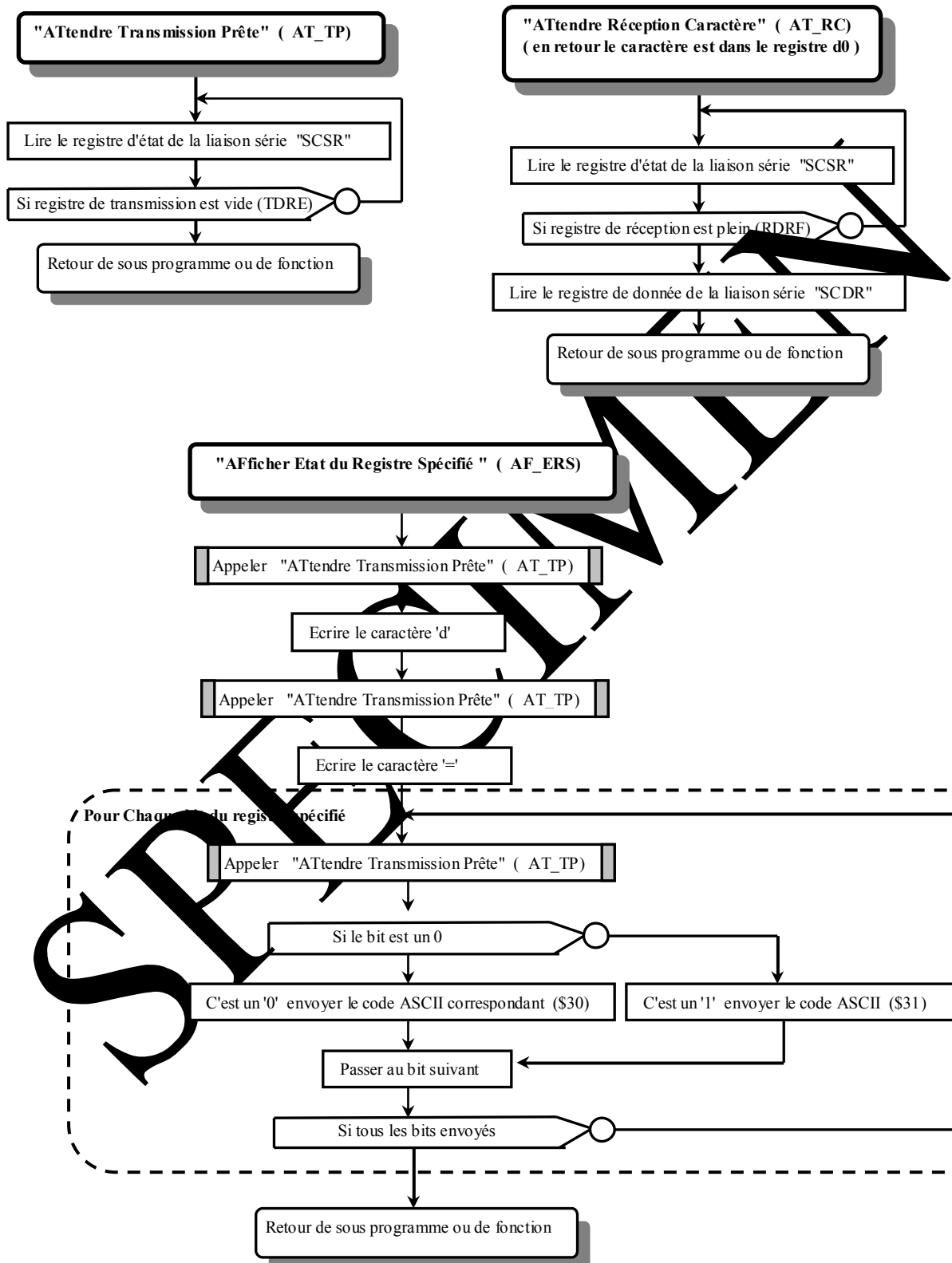
4.2 Solution

4.2.1 Organigramme général



SPECIMEN

4.2.2 Organigrammes des sous programmes (ou fonctions)



4.2.3 Programme en assembleur 68xxx

```

*****
*                                     *
*               TP SUR CARTE EID210 SEULE               *
*                                     *
*****
*               AFFICHAGE DU CONTENU D'UN REGISTRE       *
*                                     *
*   Cahier des charges:                                *
*****
*   - Au lancement du programme, il y a envoi d'un message
*   - On doit taper le n° du registre de donnée dont on désire
*     connaître la valeur (le n° doit être de 2 à 7 inclus)
*
*   NOM du FICHIER: T_SERIE3.SRC
*****
*****

*   DEFINITION ET DECLARATIONS                       *
*****
* Inclusion du fichier définissant les différents labels
*   include          EID210.def
*****
*   Déclaration des variables                         *
*****
section    var

Message    dc.b      'Numero du registre de donnée ? de 2 a 7 '
Mes_erreur dc.b      'Numero de registre non valide '
Char       ds.w       1                                * Pour mémoriser le premier caractère reçu
Num        ds.b       1                                * Pour mémoriser le numéro du bit du fichier

section    code
*****
*   DEBUT DU PROGRAMME EXECUTABLE                     *
*****
*   INITIALISER
*****
* Les initialisations suivantes sont inhibées car le moniteur a déjà configuré le port série !
* La vitesse de transmission
Debut      move.w     #9,SCCR0                          * Pour avoir une vitesse de 57600 Bauds
* Valider emission et reception
          move.w     #S00,C,SCCR1
* Initialisation des registres
          move.w     #S200,d2
          move.w     #S33,d3
          move.w     #S444,d4
          move.w     #S555,d5
          move.w     #S666,d6
          move.w     #S777,d7
*****
*   Boucle PRINCIPALE
*****
* Envoi du message d'attente
Deb_B      move.w     #S00,A1
          move.w     #Message,A1
          * Dans d1, le nombre de caractères envoyé
          * Dans A1, l'adresse du début message
* Passage à la ligne suivante et saut de ligne
          bsr        AT_TP
          move.w     #S0D,SCDR
          bsr        AT_TP
          move.w     #S0A,SCDR
          * Pour attendre si transmission prête
          * S0D est le code ASCII du CR "Retour chariot"
          * Pour attendre si transmission prête
          * S0A est le code ASCII du LF "Sauter ligne"
          * Pour attendre si transmission prête
Aff_suite_mes bsr        AT_TP
          move.b     (A1),d0
          move.w     d0,SCDR
          add.l      #1,d1
          add.l      #1,A1
          cmp.l      #43,d1
          bne        Aff_suite_mes
          * Passer au caractère suivant
          * Test si envoi message terminé
          * il y a 43 caractères dans le message
* Passage à la ligne suivante et saut de ligne
          bsr        AT_TP
          move.w     #S0D,SCDR
          bsr        AT_TP
          move.w     #S0A,SCDR
          * Pour attendre si transmission prête
          * S0D est le code ASCII du CR "Retour chariot"
          * Pour attendre si transmission prête
          * S0A est le code ASCII du LF "Sauter ligne"
* suite page suivante

```

* Suite du programme

* Réception du n° du registre dont on désire connaître la valeur

* Le caractère reçu doit être un chiffre entre 0 et 9 (Code ASCII entre \$30 et \$39)

```

    bsr      AT_RC          * pour ATtente Réception caractère
    move.w   SCDR,char      * On récupère le caractère reçu
    move.w   char,d0
    and.w    #$00FF,d0
    cmp.w    #$0030,d0      * Test si le caractère reçu est un chiffre
    blt      EM_erreur      * Les codes ASCII des chiffres sont supérieurs à $30
    cmp.w    #$0039,d0      * On attend un chiffre
    bgt      EM_erreur      * Les codes ASCII des chiffres sont inférieurs à $39

```

* Affichage de l'état du registre spécifié en binaire

```

    move.w   char,d0
    and.w    #$000F,d0
    cmp.w    #$0002,d0
    bne      test_si_d3     * Sortir si ce n'est pas d2
    * On affiche l'état de d2
    move     d2,d1
    bsr      AF_ERS         * Vers Affichage Etat Registre Spécifié
    bra      Deb_BP        * Retour au début de la boucle principale
test_si_d3  cmp.w    #$0003,d0
    bne      test_si_d4     * test si on demande l'état de d3
    * On affiche l'état de d3
    move     d3,d1
    bsr      AF_ERS         * Vers Affichage Etat Registre Spécifié
    bra      Deb_BP        * Retour au début de la boucle principale
test_si_d4  cmp.w    #$0004,d0
    bne      test_si_d5     * test si on demande l'état de d4
    * On affiche l'état de d4
    move     d4,d1
    bsr      AF_ERS         * Vers Affichage Etat Registre Spécifié
    bra      Deb_BP        * Retour au début de la boucle principale
test_si_d5  cmp.w    #$0005,d0
    bne      test_si_d6     * test si on demande l'état de d5
    * On affiche l'état de d5
    move     d5,d1
    bsr      AF_ERS         * Vers Affichage Etat Registre Spécifié
    bra      Deb_BP        * Retour au début de la boucle principale
test_si_d6  cmp.w    #$0006,d0
    bne      test_si_d7     * test si on demande l'état de d6
    * On affiche l'état de d6
    move     d6,d1
    bsr      AF_ERS         * Vers Affichage Etat Registre Spécifié
    bra      Deb_BP        * Retour au début de la boucle principale
test_si_d7  cmp.w    #$0007,d0
    bne      EM_erreur      * test si on demande l'état de d7
    * On affiche l'état de d7
    move     d7,d1
    bsr      AF_ERS         * Vers Affichage Etat Registre Spécifié
    bra      Deb_BP        * Retour au début de la boucle principale

```

* N° de registre non valide (entre 0 et 7 inclus) donc Envoi Message d'erreur

```

EM_erreur  move.w   #0,d1    * Dans d1, le nombre de caractères envoyé
    move.w   #Mes_erreur,A1 * Dans A1, l'adresse du début message

```

* Passage à la ligne suivante et saut de ligne

```

    bsr      AT_TP          * Pour attendre si transmission prête
    move.w   #$0D,SCDR      * $0D est le code ASCII du CR "Retour chariot"
    bsr      AT_TP          * Pour attendre si transmission prête
    move.w   #$0A,SCDR      * $0A est le code ASCII du LF "Sauter ligne"
EM_erreur1 bsr      AT_TP          * Pour attendre si transmission prête
    move.b   (A1),d0
    move.w   d0,SCDR
    add.l    #1,d1          * Passer au caractère suivant
    add.l    #1,A1          * Test si envoi message terminé
    cmp.l    #31,d1         * il y a 31 caractères dans le message
    bne     EM_erreur1

```

Boucler jmp Deb_BP

* FIN de boucle principale et du programme principal

```

*****
*      Sous programme d'attente si transmission prête
*****
AT_TP      move.w    SCSR,d0 * Acquérir le registre d'état de la liaison série
           and.w      #TDRE,d0 * Bit indiquant si registre de transmission est vide
           beq         AT_TP    * Transmit Data Register Empty
           rts          * Boucler si transmission non prête
           * Retour de sous programme

*****
*      Sous programme d'attente si réception caractère
*****
AT_RC      move.w    SCSR,d0 * Acquérir le registre d'état de la liaison série
           and.w      #RDRF,d0 * Bit indiquant si registre de réception est plein
           beq         AT_RC    * Recieve Data Register Full
           rts          * Boucler si rien n'est reçu
           * Retour de sous programme

*****
*      Sous programme d'affichage de l'état du registre spécifié
*****
AF_ERS     * Envoyer caractères 'd'
           bsr         AT_TP    * Pour attendre si transmission prête
           move.w      #$64,SCDR * $64 est le code ASCII de la lettre d
           * Envoyer le n° du registre
           bsr         AT_TP    * Pour attendre si transmission prête
           move.w      char,d0
           move.w      d0,SCDR   * Dans le registre SCSR, le code ASCII du n°
           * Envoyer caractères '='
           jsr         AT_TP    * Pour attendre si transmission prête
           move.w      #$3D,SCDR * $3D est le code ASCII du caractère =
           move.b      #16,num   * d0 contient le rang de bit affiché
           * On envoie les 15 états binaires à l'aide de l'un des bits MSB ... MSB
AF_ERS2 lsl.w      #1,d1
           bcc         AF_ERS0   * Sortir si bit = 0
           * C'est un 1 donc on envoie le code ASCII du chiffre 1
           jsr         AT_TP    * Pour attendre si transmission prête
           move.w      #$0031,SCDR * On envoie le code ASCII du chiffre 1
           bra         AF_ERS1
AF_ERS0     * C'est un 0 donc on envoie le code ASCII du chiffre 0
           jsr         AT_TP    * Pour attendre si transmission prête
           move.w      #$0030,SCDR * On envoie le code ASCII du chiffre 0
AF_ERS1 * On passe au bit suivant
           sub.b      #1,num
           bcc         AF_ERS2
           bra         AF_ERS1
           * Retour de sous programme

* Fin du sous-programme d'affichage de l'état du registre spécifié
*****

*****
* FIN DES SOUS-PROGRAMMES
*****

end
* Fin du fichier source assembleur

```

TP 5 : ECRITURE OU LECTURE A UNE ADRESSE SPECIFIEE

5.1 Enoncé du sujet

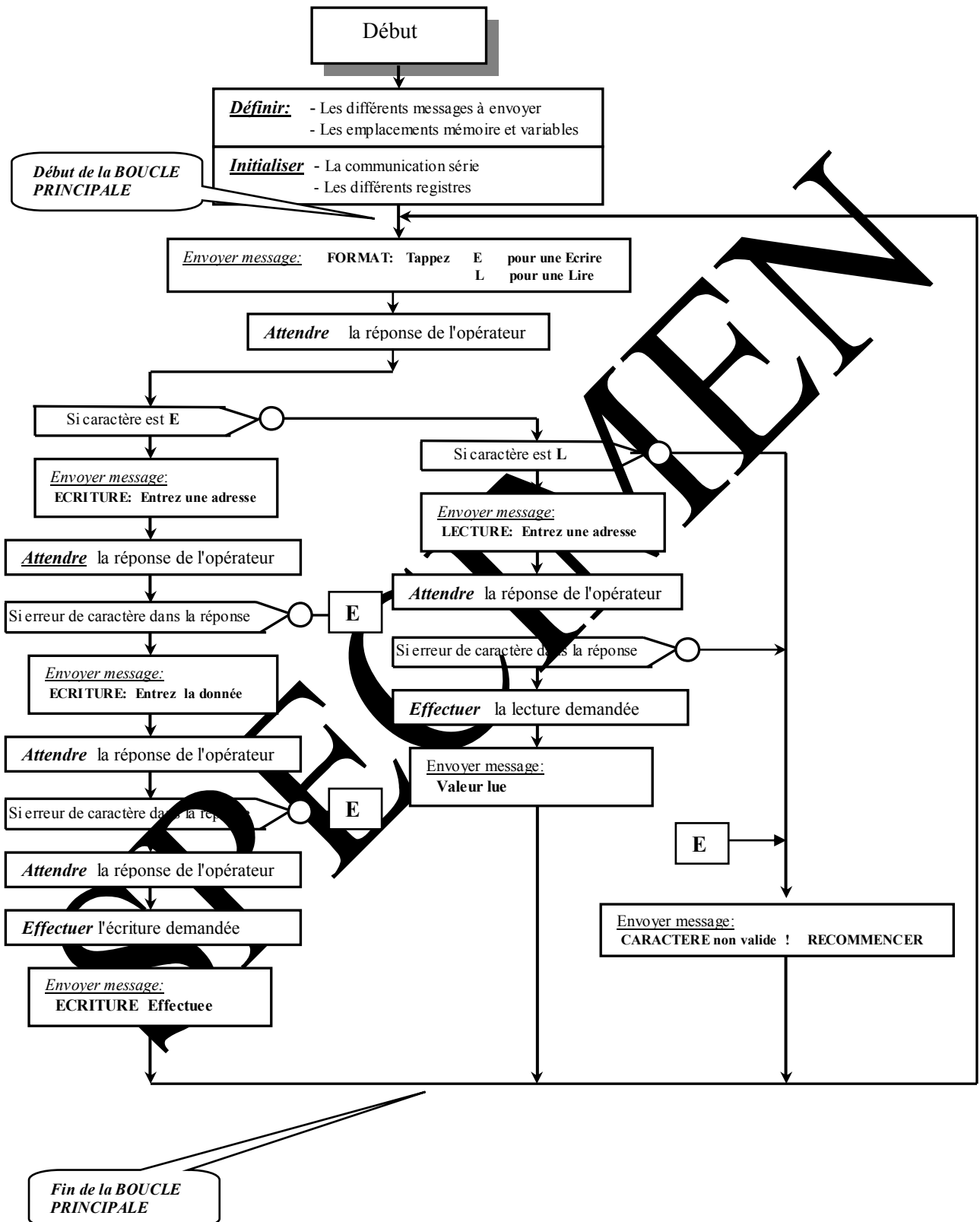
<p>Objectifs :</p>	<p>Etre capable de configurer et d'utiliser la fonction de communication série RS232 (fonction interne au micro-contrôleur 68332), en mode Émission-Réception" (liaison "duplex").</p> <p>Etre capable d'acquérir un message (chaîne de caractère) constituant une commande, de l'analyser pour détecter des erreurs éventuelles puis de l'exécuter et enfin d'y répondre.</p> <p>Etre capable de convertir des informations codées ASCII en Hexadécimal et inversement.</p>
<p>Cahier des charges :</p>	<p>Au lancement du programme, envoi d'un message prédéfini (chaîne de caractères informant de la syntaxe: "FORMAT: Tappez 'E' pour une Ecrire 'L' pour une Lire "</p> <p>Si la réponse est 'E', on demande l'adresse sur 6 digits puis la donnée sur 4 digits. Il y a contrôle des informations reçues (codes ASCII des codes HEXA), si une erreur est détectée, il y a envoi d'un message d'erreur "CARACTERE non valide ! RECOMMENCER"</p> <p>Si aucune erreur n'est détectée, l'écriture est effectuée et envoi d'un message "ECRITURE Effectuee"</p> <p>Si la réponse est 'L', on demande l'adresse sur 6 digits. Il y a contrôle des informations reçues (codes ASCII des codes HEXA). Si une erreur est détectée, il y a envoi d'un message d'erreur "CARACTERE non valide ! RECOMMENCER"</p> <p>Si aucune erreur n'est détectée, l'écriture est effectuée et envoi d'un message "Valeur lue a l' adresse specifiee: xxxx " où xxxx est le mot lu à l'adresse spécifiée, sur 16 bits codés en HEXA .</p>

Matériel nécessaire :

Micro ordinateur de type PC sous Windows 95 ou ultérieur,
 Carte mère 16/32 bits à microcontrôleur 68332, Réf : 210 000
 Câble de liaison USB, ou à défaut câble RS232, Réf : EGD 000 003
 Alimentation AC/AC 8V, 1 A Réf : EGD000001,

Durée : 4 heures

5.2 Solution



5.2.1 Programme en assembleur 68xxx

```

*****
*                                     TP SUR CARTE EID210 SEULE                                     *
*****
*      ECRITURE OU LECTURE A UNE ADRESSE MEMOIRE      *
*  Cahier des charges:                               *
*****
* - Au lancement du programme, il y a envoi d'un message
* - Format d'une demande d'écriture à une adresse spécifiée:
*   Eaaaaaa=??   où aaaaaa -> adresse en Hexadécimal
*               ?? -> la valeur à écrire en Hexa
* - Format d'une demande de lecture à une adresse spécifiée:
*   Laaaaaa   où aaaaaa -> adresse en Hexadécimal
* - La réponse est (aaaaaa)=??
*
* NOM du FICHIER: T_SERIE4.SRC
*****
*****
*      DEFINITION ET DECLARATIONS      *
*****
* Inclusion du fichier définissant les différents labels
include      EID210.def

*****
*      Déclaration des variables      *
*****
section      var
Mes_entree1      dc.b      'FORMAT: Tapez E pour Ecrire'
Mes_entree2      dc.b      'L pour une Lecture'
Mes_entree3      dc.b      'Attention, si écriture en ram, effacez les données "var" et "code"'
Mes_rep_Lec      dc.b      'LECTURE: Entrez l'adresse (sur 6 caractères HEXA)'
Mes_rep_Ecr1     dc.b      'ECRITURE: Entrez une adresse (sur 6 caractères HEXA)'
Mes_rep_Ecr2     dc.b      'ECRITURE: Entrez la donnée (sur 4 caractères HEXA)'
Mes_rep_Ecr3     dc.b      'ECRITURE Effectuee'
Mes_Val_Lue      dc.b      'Valeur lue à l'adresse spécifiée'
Mes_erreur       dc.b      'CARACTERE non valide ! RECOMMENCER'
Char             ds.w      1      * Pour mémoriser le premier caractère reçu
Num              ds.b      1      * Pour mémoriser le numéro du caractère reçu
Nombre           ds.b      1      * Pour nombre de caractères à afficher
AD_ASCII         ds.b      6      * Pour mémoriser l'adresse en ASCII
AD_HEX           ds.b      1      * Pour mémoriser l'adresse en HEXA
DATA_lue         ds.b      1
DATA_HEX         ds.w      1      * Pour donnée en hexadécimal
DATA_ASCII       ds.b      1      * Pour mémoriser la donnée en ASCII

section      code
*****
*      DEBUT DU PROGRAMME EXECUTABLE      *
*****
*      INITIALISER      *
*****
* Les initialisations suivantes sont réalisées car le moniteur a déjà configuré le port série !
* La vitesse de transmission
Debut      move.w      #9,SCCR0      * Pour avoir une vitesse de 57600 Bauds
* Valider emission et reception
move.w     #S002C,SCCR1

*****
*      BOUCLE PRINCIPALE      *
*****
Deb_BP      * Envoi du message d'entrée
move.l     #Mes_entree1,A1      * Dans le registre A1, l'adresse du message
move.b     #55,nombre          * Nombre de caractères à afficher
jsr        Env_Mes
move.l     #Mes_entree2,A1
move.b     #55,nombre          * Nombre de caractères à afficher
jsr        Env_Mes
move.l     #Mes_entree3,A1
move.b     #70,nombre          * Nombre de caractères à afficher
jsr        Env_Mes

* Suite page suivant

```

```

* SUITE du programme
* Réception de l'ordre
*****
* Le premier caractère reçu doit être E (Code ASCII $45 ) ou L (Code ASCII $4C )
Test_RC bsr          AT_RC          * pour ATtente Réception caractère
                move.w    SCDR,char  * On récupère le caractère reçu
                move.w    char,d0
                and.w      #$0045,d0
                cmp.w      #$0045,d0  * Test si le caractère reçu est E
                bne        Test_crL   * Aller tester si c'est L

* C'est un Ecriture à une adresse spécifiée d'une donnée spécifiée
*****
                move.l      #Mes_rep_Ecr1,A1
                move.b      #55,nombre  * Nombre de caractères à afficher
                jsr         Env_Mes     * Envoyer message

* Attente de l'adresse sur 6 caractères hexadécimaux
                jsr         ATT_AD      * Vers sous programme de réception de l'adresse
                cmp.w      #0,d0
                beq         Test_RC_E   * Réception adresse avec erreur
                jsr         AT_TP
                move.w      #$20,SCDR   * $20 est le code ASCII d'un "ESPACE"
                move.l      #AD_ASCII,A1
                move.b      #6,nombre  * Nombre de caractères à afficher
                jsr         Env_Mes     * Envoyer message (envoi adresse)
                move.b      #55,nombre  * Nombre de caractères à afficher
                move.l      #Mes_rep_Ecr2,A1
                jsr         Env_Mes     * Envoyer message

* Attente de la donnée sur 4 caractères hexadécimaux (un "word")
                jsr         ATT_DATA
                cmp.w      #0,d0
                beq         Test_RC_E   * Réception donnée avec erreur
                jsr         AT_TP
                move.w      #$20,SCDR   * $20 est le code ASCII d'un "ESPACE"
                move.l      #DATA_ASCII,A1
                move.b      #4,nombre  * Nombre de caractères à afficher
                jsr         Env_Mes     * Envoyer message

* Adresse et donnée sont correctes alors Effectuer l'écriture
                move.l      AD_HEX,d0
                lsl.l      #4,d0
                lsr.l      #4,d0        * Charger l'adresse en hexa
                move.l      d0,A0
                move.w      DATA_HEX,A,d0  * Charger la donnée
                move.w      d0,(A0)        * Effectuer l'écriture
                * Envoi "ECRITURE Effectuée"
                move.l      #Mes_rep_Ecr3,A1
                move.b      #55,nombre  * Nombre de caractères à afficher
                jsr         Env_Mes     * Envoyer message
                * faire deux sauts de ligne
                jsr         AT_TP
                move.w      #0A,SCDR     * $0A est le code ASCII du LF "Sauter ligne"
                jsr         AT_TP
                move.w      #0A,SCDR     * $0A est le code ASCII du LF "Sauter ligne"

* Fin ordre ECRITURE D'UNE A UNE ADRESSE SPECIFIEE
                bra         Lab_BP      * Boucler une fois l'écriture effectuée
*****
Test_crL move.w      d0
                and.w      #$004C,d0
                cmp.w      #$004C,d0  * Test si le caractère reçu est L
                bne        Test_RC_E   * Evoi message Erreur

* C'est une Lecture à une adresse spécifiée
*****
                move.l      #Mes_rep_Lec,A1
                move.b      #60,nombre  * Nombre de caractères à afficher
                jsr         Env_Mes

* Attente de l'adresse sur 6 caractères hexadécimaux
                jsr         ATT_AD
                cmp.w      #0,d0
                beq         Test_RC_E   * Réception adresse avec erreur
                jsr         AT_TP
                move.w      #$20,SCDR   * $20 est le code ASCII d'un "ESPACE"
                move.l      #AD_ASCII,A1
                move.b      #6,nombre  * Nombre de caractères à afficher
                jsr         Env_Mes     * Envoyer message (visu adresse)

```

```

* SUITE du programme
* Lecture à l'adresse spécifiée
    move.l    AD_HEX,A0
    lsl.l     #4,d0
    lsr.l     #4,d0
    move.l    d0,A0
    move.w    (A0),d0
    move.w    d0,DATA_lue
* Afficher message de résultat
    jsr       TRAD_ASCII
    move.b    #44,nombre
    jsr       AT_TP
    move.w    #S20,SCDR
    move.l    #Mes_Val_Lue,A1
    jsr       Env_Mes
    jsr       AT_TP
    move.w    #S20,SCDR
    move.b    #4,nombre
    move.l    #DATA_ASCII,A1
    jsr       Env_Mes
* Fin de lecture à une adresse spécifiée
    * faire deux sauts de ligne
    jsr       AT_TP
    move.w    #S0A,SCDR
    jsr       AT_TP
    move.w    #S0A,SCDR
    bra       Deb_BP
* Le caractère est faux, on recommence la réception de l'ordre
*****
Test_RC_E    jsr       AT_TP
              move.w    #S0A,SCDR
              move.l    #Mes_erreur,A1
              move.b    #60,nombre
              jsr       Env_Mes
              * faire deux sauts de ligne
              jsr       AT_TP
              move.w    #S0A,SCDR
Boucler      bra       Deb_BP
* FIN de boucle principale et du programme principal
*****
*****
* SOUS PROGRAMME d'attente si transmission
*****
AT_TP        move.w    SCDR,d0
              and.w     #RDRE,d0
              beq       AT_TP
* SOUS PROGRAMME d'attente si réception caractère
*****
AT_RC        move.w    SCDR,d0
              and.w     #RDRE,d0
              beq       AT_RC
* Fin du sous-programme d'affichage de l'état du registre spécifié
*****
* Suite page suivante

```

```

* SUITE TP du programme
*****
* SOUS PROGRAMME d'envoi d'un message avec saut de ligne et retour chariot
*****
Env_Mes move.b    #$0,d1          * Dans d1, le nombre de caractères envoyé
Aff_suite_mes    bsr          AT_TP      * Pour attendre si transmission prête
                move.b    (A1),d0
                move.w    d0,SCDR
                add.l     #1,d1          * Passer au caractère suivant
                add.l     #1,A1          * Test si envoi message terminé
                cmp.b     Nombre,d1     * il y a Nombre caractères dans le message
                bne       Aff_suite_mes
                * Passage à la ligne suivante et saut de ligne
                bsr          AT_TP      * Pour attendre si transmission prête
                move.w    #$0D,SCDR     * $0D est le code ASCII du CR "Retour chariot"
                bsr          AT_TP      * Pour attendre si transmission prête
                move.w    #$0A,SCDR     * $0A est le code ASCII du LF "Sauter ligne"
                rts          * Retour de sous programme

*      Fin du sous programme
*****
* SOUS PROGRAMME de réception de l'adresse sur 6 caractères en ASCII
* et constitution adresse en HEXA (3 octets)
*****
ATT_AD move.b     #0,Num          * Num caractère = 0
                move.l     #AD_ASCII,A1
                move.l     #0,AD_HEX
ATT_AD1 jsr        AT_RC          * pour Attente Réception caractère
                move.w     SCDR,char * On récupère le caractère reçu
                move.w     char,d0
                jsr        Test_CH * Aller tester si caractère Hexadécimal
                cmp.w     #0,d0      * Valeur retournée égale à 0 si erreur
                beq        ATT_AD_err * Retour avec erreur
                move.b     d0,(A1)    * mémoriser le caractère reçu
                move.b     #5,d3      * Reconstituer l'adresse en HEXA
                sub.b     num,d3
                and.l     #00000F,d3 * Dans d3 le nombre de décalages
                lsl.l     #2,d3      * à faire effectuer au caractère
                lsl.l     d3,d1      * faire les décalages
                or.l      d1,AD_HEX
                add.l     #1,A1      * Passer au caractère qui suivant
                add.b     #1,Num
                cmp.b     #6,Num     * Test si les 6 caractères d'adresse sont acquis
                bne       ATT_AD_err
                rts
ATT_AD_err rts          * Retour avec erreur (do=0)
*      Fin du sous programme
*****
* SOUS PROGRAMME de réception de donnée sur 4 caractères en ASCII
* et constitution de la donnée en HEXA (2 octets) -> ex: "WORD"
*****
ATT_DATA move.b     #0,Num          * Num caractère = 0
                move.l     #DATA_ASCII,A1
                move.l     #0,DATA_HEX
ATT_DATA1 jsr        AT_RC          * pour Attente Réception caractère
                move.w     SCDR,char * On récupère le caractère reçu
                move.w     char,d0
                jsr        Test_CH * Aller tester si caractère Hexadécimal
                cmp.w     #0,d0      * Valeur retournée égale à 0 si erreur
                beq        ATT_DATA_err * Retour avec erreur
                move.b     d0,(A1)    * mémoriser le caractère reçu HEXA en ASCII
                move.b     #3,d3      * Pour reconstituer la donnée en HEXA
                sub.b     num,d3
                and.w     #000F,d3
                lsl.w     #2,d3
                lsl.w     d3,d1      * faire les décalages
                or.w      d1,DATA_HEX
                add.l     #1,A1      * Passer au caractère qui suivant
                add.b     #1,Num
                cmp.b     #4,Num     * Test si les 4 caractères de donnée sont acquis
                bne       ATT_DATA1
                rts
ATT_DATA_err rts          * Retour avec erreur (do=0)
*      Fin du sous programme
* SUITE page suivante

```

* SUITE du programme

 * SOUS PROGRAMME de test si caractère Hexadécimal en ASCII et traduction en HEXA

```
Test_CH and.w      #$00FF,d0
                cmp.w      #$0030,d0 * Test si le caractère reçu est un chiffre
                blt         Test_CH_err * Les codes ASCII des HEXA sont supérieurs à $30
                cmp.w      #$0039,d0
                bgt         Test_CH1 * Les codes ASCII des chiffres sont inférieurs à $39
                move.w      d0,d1      * Dans d0 le code ASCII du caractère HEXA
                andi.w      #$000F,d1 * Dans d1 le code HEXA de 0 à 9
                rts
                * Retour si c'est OK

Test_CH1 cmp.w     #$0041,d0 * Test si le caractère reçu est une lettre HEXA
                blt         Test_CH_err * Les codes ASCII des lettres HEXA sont sup à $41
                cmp.w      #$0046,d0
                bgt         Test_CH_err * Les codes ASCII des lettres HEXA sont inf à $46
                move.w      d0,d1      * Dans d0 le code ASCII du caractère HEXA
                andi.w      #$000F,d1 * Dans d1 le code HEXA de 0 à 9
                add.w       #9,d1
                rts
                * Retour si c'est OK

Test_CH_err move.w  #0000,d0 * Retour avec 0 si erreur
                rts
```

* Fin du sous programme

 * Sous programme de traduction de la donnée de l'HEXA en ASCII

```
TRAD_ASCII move.b  #0,Num * Num caractère = 0
                move.l  #DATA_ASCII,A1
TRAD_ASCII1 move.w  DATA_lue,d0 * Valeur lue, c'est de l'Hexa
                move.b  #3,d3      * Pour isoler le quartet à convertir
                sub.b    num,d3     * Calculer le nombre de décimales
                and.w    #$000F,d3
                lsl.w    #2,d3
                lsr.w    d3,d0      * faire les décimales
                and.w    #$000F,d0 * Isoler quartet
                * Convertir en ASCII
                cmp.b    #9,d0
                bgt         TRAD_Lettre * Si plus grand que 9, c'est une lettre (A à F)
                * C'est un chiffre (0 à 9)
                or.b     #$30,d0 * Codes des chiffres sont de $30 à $39
                bra      TRAD_ASCII2
TRAD_Lettre add.b     #55,d0 * Code ASCII des lettres HEXA sont de $41 à $46 (65 à 70)
TRAD_ASCII2 move.l  d0,(A1)
                add.l    #1,A1
                move.b    d0,d1
                sub.b     #1,d1
                bne      #4,Num * Test si les 4 caractères de donnée sont traduits
                bra      TRAD_ASCII1 * Continuer si ce n'est pas fini
                rts
                * retourner si c'est fini
```

* Fin du sous programme

 * FIN DES SOUS PROGRAMMES

end * Fin du fichier source assembleur

5.2.2 Programme en "C"

```

/*****
* TP SUR EID210 seul
*****/
* ECRITURE OU LECTURE A UNE ADRESSE MEMOIRE
*
* Cahier des charges:
*****/
* - Au lancement du programme, il y a envoi d'un message
* - Format d'une demande d'écriture à une adresse spécifiée:
*   Eaaaaa=?? où aaaaaa -> adresse en Hexadécimal
*   ?? -> la valeur à écrire en Hexa
* - Format d'une demande de lecture à une adresse spécifiée:
*   Laaaaaa où aaaaaa -> adresse en Hexadécimal
* - La réponse est (aaaaaa) = ??
*
* NOM du FICHIER: T_SERIE4.C
*****/

/* Liste des fichiers à inclure */
#include "Cpu_reg.h"
#include "EID210_reg.h"
#include "Structures_donnees.h"
#include <stdio.h>

// Déclaration des prototypes des fonctions
char Acquerir_info(char); // Paramètre passé, le nombre de caractères Hexa à acquirir
// Paramètre renvoyé le nombre d'erreurs détectées lors de l'acquisition

// Déclaration des variables globales
unsigned int info_acquise;
char caractere;

/*=====*/
/* Fonction principale */
/*=====*/
main()
{
/* Déclarations locales à la fonction main */
*****/
char Mes_entree[]=" FORMAT: Tapez E pour une Ecrire"
"\n Tapez L pour une Lire ";
char Mes_rep_Lec[]=" LECTURE: Entrez une adresse (sur 6 caractères HEXA) ";
char Mes_rep_Ecr1[]=" ECRITURE: Entrez une adresse (sur 6 caractères HEXA) ";
char Mes_rep_Ecr2[]=" ECRITURE: Entrez la donnée (sur 4 caractères HEXA) ";

char Mes_erreur[]=" CARACTERE non valide ! IL FAUT COMMENCER !! ";
char erreur;
unsigned int AD_HEXa; // Pour mémoriser une adresse en hexadécimal
unsigned short DATA_HEXa,*pointeur; // Pour mémoriser une donnée en hexadécimal

/* Boucle principale */
*****/
do
{
printf("%s\n",Mes_entree);
caractere=getchar(); // Attendre qu'un caractère soit tapé puis le récupérer
if (caractere=='E' || caractere=='L')
{ // C'est pour écrire à une adresse spécifiée
printf("%s\n",Mes_rep_Ecr1);
erreur=Acquerir_info(6); // On attend les 6 caractères de l'adresse
if(erreur==0)
{ AD_HEXa=info_acquise;
printf("Vous voulez écrire à l'adresse: %x\n",AD_HEXa);
printf("%s\n",Mes_rep_Ecr2);
erreur=Acquerir_info(4); // On attend les 4 caractères de la donnée
if(erreur==0)
{ DATA_HEXa=(short)info_acquise;
printf("C'est OK On écrit la donnée : %4.4x \n",DATA_HEXa);
pointeur=(short *)AD_HEXa;
*pointeur=DATA_HEXa;
}
else printf("%s\n",Mes_erreur);
}
else printf("%s\n",Mes_erreur);
}
} while (1);
} // FIN d'écriture à une adresse spécifiée

```

SUITE PAGE SUIVANTE

```

else if(caractere=='L' || caractere=='l')
{ // C'est pour lire à une adresse spécifiée
//-----
printf("%s\n", Mes_rep_Le c);
erreur=Acquerir_info(6); // On attend les 6 caractères de l'adresse
if(erreur==0)
{ printf("Vous voulez lire à l'adresse: %x\n", AD_HEX A);
AD_HEX A=info_acquise;
pointeur=(short *)AD_HEX A;
DATA_HEX A=*pointeur;
printf("La donnée lue à l'adresse spécifiée est : %4.4x\n", DATA_HEX A);
}
else printf("%s\n", Mes_erreur);
}
else printf("%s\n", Mes_erreur);
// FIN de lecture à une adresse spécifiée
}while(1); /* Fin de la boucle principale */
}

/* Fin de la fonction principale */
//=====

// FONCTION D'acquisition d'une information (adresse ou donnée en HEXA à partir du clavier)
//=====

char Acquerir_info(char nb) // Paramètre passé, le nombre de caractères Hexa à acquies
// Paramètre renvoyé le nombre d'erreurs détectées lors de l'acquisition
{
// Déclaration des variables locales
char Num, nb_erreur; // Pour mémoriser le numéro du caractère reçu
unsigned int temp;
// Lers instructions
nb_erreur=0;
info_acquise=0;
// Boucle pour acquies le nombre de caractères HEXA de l'adresse
for(Num=nb; Num>=1; Num--)
{ caractere=lnRs232(); // Attendre qu'un caractère soit tapé puis le récupérer
if((caractere>=0x30)&&(caractere<=0x39)) // Test si c'est un chiffre
{ temp=caractere&0x00000F; temp=temp<<((Num-1)*4);
info_acquise=info_acquise|temp; // Ranger dans l'adresse, à la bonne place
}
else if((caractere>=0x41)&&(caractere<=0x46)) // Test si c'est une lettre HEXA (entre A et F)
{ temp=caractere&0x00000F; temp=temp<<0x09; temp=temp<<((Num-1)*4);
info_acquise=info_acquise|temp; // Ranger dans l'adresse, à la bonne place
}
else // Le caractère tapé n'est pas correcte
{ nb_erreur++;
break; // Ce n'est pas un caractère HEXA, donc on sort
}; // Fin de la boucle de récupération de l'adresse
}
return nb_erreur;
} // FIN fonction d'acquisition d'info à partir du clavier
//=====

// FIN du fichier

```


SPECIMEN

ANNEXE

ANNEXE 1 Fichier de définitions pour programme en Assembleur

Nom du fichier EID210.def

```

no list
MONITEUR      EQU      $400          * retour au moniteur

```

* Pour le micro-contrôleur 68332

* Définition des adresses des registres du module QSM

```

PORTQS EQU      $FFFC14
PQSCTR EQU      $FFFC16
SCCR1   EQU      $FFFC0A
SCCR0   EQU      $FFFC08
SCSR    EQU      $FFFC0C
SCDR    EQU      $FFFC0E
TDRE    EQU      $0100
RDRF    EQU      $0040
RAF      EQU      $0020

```

* SIM

```

PITR      EQU      $FFFA24
PICR      EQU      $FFFA22

```

* définition des registres du module TPU

```

TRAMMCR EQU      $FFFB00
TRAMTST EQU      $FFFA04
TRAMBAR EQU      $FFFA20
TPUMCR EQU      $FFFE00
TCR      EQU      $FFFE02
CFSR0   EQU      $FFFE0C
CFSR1   EQU      $FFFE0E
CFSR2   EQU      $FFFE10
CFSR3   EQU      $FFFE12
HSQR0   EQU      $FFFE14
HSQR1   EQU      $FFFE16
HSRR0   EQU      $FFFE18
HSRR1   EQU      $FFFE1A
CPR0    EQU      $FFFE1C
CPR1    EQU      $FFFE1E
CTRL_TPU0 EQU      $FFFE00
CTRL_TPU1 EQU      $FFFE10
CTRL_TPU2 EQU      $FFFE20
CTRL_TPU3 EQU      $FFFE30
CTRL_TPU4 EQU      $FFFE40
CTRL_TPU5 EQU      $FFFE50
CTRL_TPU6 EQU      $FFFE60
CTRL_TPU7 EQU      $FFFE70
CTRL_TPU8 EQU      $FFFE80
CTRL_TPU9 EQU      $FFFE90
CTRL_TPU10 EQU      $FFFA00
CTRL_TPU11 EQU      $FFFE00
CTRL_TPU12 EQU      $FFFC00
CTRL_TPU13 EQU      $FFFD00
CTRL_TPU14 EQU      $FFFE00
CTRL_TPU15 EQU      $FFFF00

```

* Pour les fonctions périphériques de la carte EID210

* controle , Port_C, CNA, CAN, PC104

```

CTRL      EQU      $900000
REG_ETAT  EQU      CTRL
Port_C    EQU      CTRL+$100
DIR_Port_C EQU      Port_C+2
CNA       EQU      $B10000
SA0       EQU      CNA
CAN       EQU      $B20000
PC104     EQU      $B30000

```

* Table des vecteurs en RAM

```

Tab_vect EQU      $800000
list

```

SPECIMEN

ANNEXE 2 Fichiers de définitions inclus dans programmes en "C"

```
//=====
// STRUCTURES DE DONNEES UTILES A LA PROGRAMMATION EN C
//=====

// Nom du fichier: Structures_donnees.h
// Date de création: Aout 2001
// Date de dernière modification: Janv 2003
// Auteurs: T. HANS J.L. ROHOU
//=====

/* Redéfinition des types */
typedef unsigned char BYTE;
typedef unsigned short WORD;
typedef unsigned long ULONG;

/* Union pour accéder à un octet (BYTE) soit en direct, soit en individualisant les 8 bits
//=====
union byte_bits
{
    struct
    {
        unsigned char b7:1;
        unsigned char b6:1;
        unsigned char b5:1;
        unsigned char b4:1;
        unsigned char b3:1;
        unsigned char b2:1;
        unsigned char b1:1;
        unsigned char b0:1;
    } bit;
    BYTE valeur;
};

/* Union pour accéder à un mot de 16 bit soit en direct soit en individualisant les 16 bits
//=====
union word_bits
{
    struct
    {
        unsigned char b15:1;
        unsigned char b14:1;
        unsigned char b13:1;
        unsigned char b12:1;
        unsigned char b11:1;
        unsigned char b10:1;
        unsigned char b9:1;
        unsigned char b8:1;
        unsigned char b7:1;
        unsigned char b6:1;
        unsigned char b5:1;
        unsigned char b4:1;
        unsigned char b3:1;
        unsigned char b2:1;
        unsigned char b1:1;
        unsigned char b0:1;
    } bit;
    WORD valeur;
};

/* Union pour accéder à un mot de 16 bit soit en direct soit en individualisant les 16 bits par ensembles de 2 bits (Utile pour les sorties TPU)
//=====
union word_duos
{
    struct
    {
        unsigned char duo7:2;
        unsigned char duo6:2;
        unsigned char duo5:2;
        unsigned char duo4:2;
        unsigned char duo3:2;
        unsigned char duo2:2;
        unsigned char duo1:2;
        unsigned char duo0:2;
    } duo;
    WORD valeur;
};
```

// Suite page suivante

```
/* Structure pour accéder à un mot de 16 bits soit en direct soit en séparant sur 8 bits de poids forts (b15 à b8)
et gardant unis les 8 bits de poids faibles (O_lsb)
*****/
```

```
union word_bits_octet
{
    struct
    {
        unsigned char b15:1;
        unsigned char b14:1;
        unsigned char b13:1;
        unsigned char b12:1;
        unsigned char b11:1;
        unsigned char b10:1;
        unsigned char b9:1;
        unsigned char b8:1;
        unsigned char O_lsb:8;
    } bits_octet;
    WORD val_wbo;
};
```

```
/* Structure pour séparer l'octet de poids forts (O_msb) de l'octet de poids faibles (O_lsb)
d'un mot de 16 bits, avec l'octet de poids fort pouvant être séparé en 8 bits individuels
*****/
```

```
struct word_bytes
{
    union byte_bits O_msb;
    unsigned char O_lsb;
};
```

```
/* Union pour accéder à un mot de 16 bits (WORD) soit en direct soit en séparant sur 8 bits de poids faibles (b0 à b7)
et gardant unis les 8 bits de poids forts (O_msb)
*****/
```

```
union word_octet_bits
{
    struct
    {
        unsigned char O_msb:8;
        unsigned char b7:1;
        unsigned char b6:1;
        unsigned char b5:1;
        unsigned char b4:1;
        unsigned char b3:1;
        unsigned char b2:1;
        unsigned char b1:1;
        unsigned char b0:1;
    } octet_bits;
    WORD valeur;
};
```

```
// Fin de fichier
```

```
/*=====
* DEFINITION DES LABELS ET ADRESSES PERMETTANT L'ACCES AUX REGISTRES DU MICROPROCESSEUR 68332
*=====
*
* Nom du fichier : CPU_REG.H
* Date de création : début 2001
* Date de la dernière révision : Février 2002
* Auteurs: J. ROHON, T. HANS
*=====*/
```

```
#ifndef CPU_H
#define CPU_H
```

```
/*=====
```

```
*/
```

```
/* " System Integration Module "
```

```
/*=====
```

```
#define VBR *(WORD*) 0x000000 /* Vector Base Register Vecteur de base 5-6*/
```

```
/* registre de configuration du system integration module */
```

```
#define Simcr *(WORD*) 0xFFFA00 /* registre de controle de la configuration du SIM */
```

```
#define Syncr *(WORD*) 0xFFFA04 /* registre de controle de l'horloge æp */
```

```
#define Sypcr *(WORD*) 0xFFFA21 /* registre de controle du systeme de protection */
```

```
#define Picr *(WORD*) 0xFFFA22 /* registre de controle des interruptions */
```

```
#define Pitr *(WORD*) 0xFFFA24 /* registre de controle du timer d'interruption */
```

```
#define Rsr *(WORD*) 0xFFFA07 /* reset register */
```

// Suite page suivante

```

/* registre de configuration des chip selects */
#define Cspar0 *(WORD*) 0xFFFA44 /* reg. de contr. de la config du CSboot et des chip selects CS0 à CS5 */
#define Cspar1 *(WORD*) 0xFFFA46 /* registre de controle de la configuration des chips selects CS6 à CS10 */
#define Csbart *(WORD*) 0xFFFA48 /* registre de configuration de l'adresse de base du chip select BOOT */
#define Csort *(WORD*) 0xFFFA4A /* registre de controle des options de configuration du chip select BOOT */
#define Csbar0 *(WORD*) 0xFFFA4C /* registre de configuration de l'adresse de base du chip select CS0 */
#define Csor0 *(WORD*) 0xFFFA4E /* registre de controle des options de configuration du chip select CS0 */
#define Csbar1 *(WORD*) 0xFFFA50 /* registre de configuration de l'adresse de base du chip select CS1 */
#define Csor1 *(WORD*) 0xFFFA52 /* registre de controle des options de configuration du chip select CS1 */
#define Csbar2 *(WORD*) 0xFFFA54 /* registre de configuration de l'adresse de base du chip select CS2 */
#define Csor2 *(WORD*) 0xFFFA56 /* registre de controle des options de configuration du chip select CS2 */
#define Csbar3 *(WORD*) 0xFFFA58 /* registre de configuration de l'adresse de base du chip select CS3 */
#define Csor3 *(WORD*) 0xFFFA5A /* registre de controle des options de configuration du chip select CS3 */
#define Csbar4 *(WORD*) 0xFFFA5C /* registre de configuration de l'adresse de base du chip select CS4 */
#define Csor4 *(WORD*) 0xFFFA5E /* registre de controle des options de configuration du chip select CS4 */
#define Csbar5 *(WORD*) 0xFFFA60 /* registre de configuration de l'adresse de base du chip select CS5 */
#define Csor5 *(WORD*) 0xFFFA62 /* registre de controle des options de configuration du chip select CS5 */
#define Csbar6 *(WORD*) 0xFFFA64 /* registre de configuration de l'adresse de base du chip select CS6 */
#define Csor6 *(WORD*) 0xFFFA66 /* registre de controle des options de configuration du chip select CS6 */
#define Csbar7 *(WORD*) 0xFFFA68 /* registre de configuration de l'adresse de base du chip select CS7 */
#define Csor7 *(WORD*) 0xFFFA6A /* registre de controle des options de configuration du chip select CS7 */
#define Csbar8 *(WORD*) 0xFFFA6C /* registre de configuration de l'adresse de base du chip select CS8 */
#define Csor8 *(WORD*) 0xFFFA6E /* registre de controle des options de configuration du chip select CS8 */
#define Csbar9 *(WORD*) 0xFFFA70 /* registre de configuration de l'adresse de base du chip select CS9 */
#define Csor9 *(WORD*) 0xFFFA72 /* registre de controle des options de configuration du chip select CS9 */
#define Csbar10 *(WORD*) 0xFFFA74 /* registre de configuration de l'adresse de base du chip select CS10 */
#define Csor10 *(WORD*) 0xFFFA76 /* registre de controle des options de configuration du chip select CS10 */
/*****
* Pour le TPU
* " Time Processor Unit "
*****/
#define CFSR0 *(short*)(0xFFFE0C) // Channel Function Select Register
#define CFSR1 *(short*)(0xFFFE0E) // Permet de définir la fonction soumise pour chacun des
#define CFSR2 *(short*)(0xFFFE10) // 15 canaux (lignes d'entrée-sortie TPU0 à TPU15)
#define CFSR3 *(short*)(0xFFFE12) // 4 bits sont affectés à un même canal
#define HSQR0 *(short*)(0xFFFE14) // Host Sequence Register
#define HSQR1 *(short*)(0xFFFE16) // Permet d'effectuer un échantillonnage des canaux configurés en entrée
#define HSRR0 *(short*)(0xFFFE18) // Host Sequence Request Register
#define HSRR1 *(short*)(0xFFFE1A) // Permet d'accéder aux canaux d'entrée-sortie
#define CPR0 *(short*)(0xFFFE1C) // Channel Priority Register
#define CPR1 *(short*)(0xFFFE1E) // Permet de définir des niveaux de priorité
#define CTRL_TPU0 *(short*)(0xFFFF00) // Control
#define CTRL_TPU1 *(short*)(0xFFFF01) // Une zone de 8 mots de stat est réservée à chaque canal d'entrée sortie
#define CTRL_TPU2 *(short*)(0xFFFF02)
#define CTRL_TPU3 *(short*)(0xFFFF03)
#define CTRL_TPU4 *(short*)(0xFFFF04)
#define CTRL_TPU5 *(short*)(0xFFFF05)
#define CTRL_TPU6 *(short*)(0xFFFF06)
#define CTRL_TPU7 *(short*)(0xFFFF07)
#define CTRL_TPU8 *(short*)(0xFFFF08)
#define CTRL_TPU9 *(short*)(0xFFFF09)
#define CTRL_TPU10 *(short*)(0xFFFF0A)
#define CTRL_TPU11 *(short*)(0xFFFF0B)
#define CTRL_TPU12 *(short*)(0xFFFF0C)
#define CTRL_TPU13 *(short*)(0xFFFF0D)
#define CTRL_TPU14 *(short*)(0xFFFF0E)
#define CTRL_TPU15 *(short*)(0xFFFF0F)

// Pour le temporisateur programmable
#define PIT *(short*)(0xFFFA24) //
#define PCR *(short*)(0xFFFA22) //

/*****
* Pour le QSM
* "Queue d Serial Module"
*****/
#define QSMCR *(WORD*) 0xFFFC00
#define QILVR *(WORD*) 0xFFFC04
#define SCCR0 *(WORD*) 0xFFFC08
#define SCCR1 *(WORD*) 0xFFFC0A
#define SCSR *(WORD*) 0xFFFC0C
#define SCDR *(WORD*) 0xFFFC0E

#define PORTQS *(WORD*) 0xFFFC14
#define PQSCTR *(WORD*) 0xFFFC16 /* PQSPAR-DDRQS */

#define TDRE (WORD) 0x0100
#define RDRF (WORD) 0x0040
#define RAF (WORD) 0x0020

#endif

```

```
//=====
//      DECLARATIONS DES ADRESSES DES ELEMENTS DE LA CARTE EID210
//=====

//      Nom du fichier:      EID210_reg.h
//      Date de création:    Aout 2001
//      Date de dernière modification: Janv 2003
//      Auteurs:             T. HANS J.L. ROHOU
//=====

#ifndef _EID210_reg.h
#define _EID210_reg.h

/*      Version materielle et logicielle      */
/*=====*/
#define VERSION_HARD      0x00      /* Version et revision du hard */
#define REVISION_HARD     0x00
#define VERSION_SOFT      0x00      /* Version et revision du programme */
#define REVISION_SOFT     0x00

/* Adresses de bases des périphériques */
#define CTRL              0x900000      /* CPLD de contrôle */
#define REG_ETAT (*(union word_bits_octet*) (CTRL+0x00)) /* registre d'état (en lecture uniquement) */
#define REG_CTRL (*(WORD*) (CTRL+0x02))
#define A_Port_C (*(struct word_bytes*) (CTRL+0x100)) /* Accès au registre de donnée du Port C */
#define A_Dir_Port_C (*(struct word_bytes*) (CTRL+0x102)) /* Accès au registre de direction du port C */
#define USB              0xB00000      /* Ad. de base Port USB CS3 */
#define CNA               0xB10000      /* Ad. de base du CNA Num -> A CS4 */
#define SA0               (*(BYTE*) (CNA+0x00)) /* Sortie Analogique voie 0 (SA0) */
#define SA1               (*(BYTE*) (CNA+0x02)) /* Sortie Analogique voie 0 (SA0) */
#define SA2               (*(BYTE*) (CNA+0x04)) /* Sortie Analogique voie 0 (SA0) */
#define SA3               (*(BYTE*) (CNA+0x06)) /* Sortie Analogique voie 0 (SA0) */
#define CAN               0xB20000      /* Convertisseur A->N Num -> A CS6 */
#define BUS               0xB30000      /* Ad. de base du BUS Num -> A CS7 */

/* Pour accéder aux différentes informations du registre d'état */
/* Bits accessibles en lecture uniquement */
#define Etat_reset        REG_ETAT.bits_octet.b15 /* RESERVE, ne touchez pas si RESET! */
#define E_Irq_CAN         REG_ETAT.bits_octet.b14 /* Etat du Bit de fin de Conversion Ana -> Num */
#define E_Irq_USB         REG_ETAT.bits_octet.b13 /* Etat du Bit d'état de la ligne d'interruption USB */
#define E_Irq4_Bus        REG_ETAT.bits_octet.b12 /* Etat du Bit d'état ligne IRQ4 du BUS */
#define E_Irq3_Bus        REG_ETAT.bits_octet.b11 /* Etat du Bit d'état ligne IRQ3 du BUS */
#define E_Irq2_Bus        REG_ETAT.bits_octet.b10 /* Etat du Bit d'état ligne IRQ2 du BUS */
#define E_Irq1_Bus        REG_ETAT.bits_octet.b9 /* Etat du Bit d'état ligne IRQ1 du BUS */
#define S_Contrôle        REG_ETAT.bits_octet.b8 /* Etat du Etat du Switch de Contrôle */
#define N_VERSION         REG_ETAT.bits_octet.O_Isb /* N° de Version soft PLD sur 8 bits */

/* Pour autoriser (valider) les interruptions */
#define VALID_IRQs (*(union word_bits_octet*) (0x900000))
#define Valid_IrqCtrl     VALID_IRQs.octet_bits.b15
#define Valid_IrqCan       VALID_IRQs.octet_bits.b14
#define Valid_Irq1         VALID_IRQs.octet_bits.b13
#define Valid_Irq2         VALID_IRQs.octet_bits.b12
#define Valid_Irq3         VALID_IRQs.octet_bits.b11
#define Valid_Irq4         VALID_IRQs.octet_bits.b10
#define Valid_IrqUsb       VALID_IRQs.octet_bits.b9
#define Valid_unus         VALID_IRQs.octet_bits.b8

// Pour une gestion directe du port C
#define PortC             A_Port_C.O_msb.valeur
#define DirPortC          A_Dir_Port_C.O_msb.valeur
#define DirPortC_Dir      A_Dir_Port_C.O_msb.bit.b0
#define PortC_Dir         DirPortC_Dir
#define PC0               A_Port_C.O_msb.bit.b0
#define PC1               A_Port_C.O_msb.bit.b1
#define PC2               A_Port_C.O_msb.bit.b2
#define PC3               A_Port_C.O_msb.bit.b3
#define PC4               A_Port_C.O_msb.bit.b4
#define PC5               A_Port_C.O_msb.bit.b5
#define PC6               A_Port_C.O_msb.bit.b6
#define PC7               A_Port_C.O_msb.bit.b7

// Pour la gestion du convertisseur A->N
#define Fin_Conv_AN E_Irq_CAN

#endif
```