L'ASSEMBLEUR

SUR

MICROCONTRÔLEUR

SIEMENS C167

LE MICROCONTRÔLEUR C167

Architecture interne

Le C167 est constitué des 5 éléments essentiels suivants :

- Une mémoire RAM contenant en particulier les registres (registres généraux, registres du processeur et registres de l'unité d'échange)
- Une unité d'échange incluant les contrôleurs de périphériques suivants :
  - convertisseur analogique/numérique
  - générateur d'impulsions
  - timers/compteurs
  - communication série synchrone et asynchrone
  - lignes d'entrées/sorties logiques
  - communication de type réseau
- Un contrôleur d'interruptions avec priorité et prise en compte rapide
- Un contrôleur de bus externe programmable

La mémoire est organisée en

segments de 64K octets pour le code. Le segment actif est celui désigné par le registre **CSP.**

pages de 16K pour les données. Quatre segments sont accessibles simultanément, ils sont désignés par les registres **DPP0**, **DPP1**, **DPP2** et **DPP3**.

La mémoire est organisée en mots de 16 bits. Toutefois certaines zones sont accessibles bit par bit.

Les registres

Le 167 possède 16 registres généraux (GPR) de 16 bits appelés **R0, R1 ... R15**. Les 8 premiers sont divisés en deux registres de 8 bits appelés **RL0, RL1 ... RL7** pour l'octet de faible poids et **RH0, RH1 ... RH7** pour l'octet de fort poids. Ces registres sont placés en mémoire dans uns zone pointée par le registre **CP**, ils peuvent donc être déplacés.

Il possède aussi 212 registres de fonctions spéciales (SFR) qui incluent les registres du processeur (registre d'état, pointeur de pile etc.) et les registres de l'unité d'échange. Ces registres sont placés dans une zone fixe de la mémoire .

Les opérandes

Le 167 traite les opérandes suivants :

- Entiers naturels sur 8 et 16 bits
- Entiers relatifs sur 8 et 16 bits en complément à 2
- Bits

JEU D'INSTRUCTIONS DU 167

Notation : pour simplifier les descriptions ci-dessous on choisira de désigner par :

RG	un registre général (R0 .. R15 , RL0 ... RL7 , RH0 .. RH7)
RG3	un registre général limité à R0, R1, R2 ou R3
SFR	un registre de fonction spéciale
#val	une valeur immédiate
mem	un opérande en mémoire

Déplacement de données

MOV dest,source *opération* : dest ← source. Sur 8 ou 16 bits. Le 167 connaît en fait 2 instructions :

MOVW	dest,source	Sur 16 bits
MOVB	dest,source	Sur 8 bits

En cas d'ambiguïté, il faudra utiliser ces codes directement. Par exemple, mov P8,[R0] sera interprété comme une opération sur 16 bits alors que P8 n'est que sur 8 bits. Pour éviter ce problème on pourra écrire explicitement : movb P8,[R0]

Les opérandes doivent être les suivants :

destination / source	RG	SFR	mem	[RG]	[RG+]	[RG+#depl]	[-RG]
RG	X	X	X	X		X	X
SFR	X	X	X	X			
mem	X	X		X			
#val	X	X					
[RG]	X	X	X	X	X		
[RG+]	X			X			
[RG+#depl]	X						

MOVBS dest,source *opération* : dest ← source.
dest est sur 16 bits et source sur 8, le transfert se fait avec extension du signe

MOVBZ dest,source *opération* : dest ← source.
dest est sur 16 bits et source sur 8, le transfert se fait sans extension du signe (complété par des 0)

Pour ces deux dernières instructions les opérandes doivent être les suivants :

destination / source	RG	SFR	mem
RG	X	X	X
SFR	X	X	X
mem	X	X	

Arithmétique

ADD op1 , op2 *opération* : op1 ← op1 + op2. L'addition est sur 8 ou 16 bits. Le 167 connaît en fait deux instructions :

ADDW	op1 , op2	L'addition est sur 16 bits.
ADDB	op1 , op2	L'addition est sur 8 bits.

ADDC op1 , op2 *opération* : op1 ← op1 + op2 + retenue de l'opération précédente. L'addition est sur 8 ou 16 bits. Le 167 connaît en fait deux instructions :

ADDCW	op1 , op2	L'addition est sur 16 bits.
ADDCB	op1 , op2	L'addition est sur 8 bits.

SUB op1 , op2 *opération* : op1 ← op1 - op2. La soustraction est sur 8 ou 16 bits. Le 167 connaît en fait deux instructions :

SUBW op1 , op2 La soustraction est sur 16 bits.
SUBB op1 , op2 La soustraction est sur 8 bits.

SUBC op1 , op2 *opération* : op1 ← op1 - op2 - retenue de l'opération précédente. La soustraction est sur 8 ou 16 bits. Le 167 connaît en fait deux instructions :

SUBCW op1 , op2 La soustraction est sur 16 bits.
SUBCB op1 , op2 La soustraction est sur 8 bits.

Les opérandes doivent être les suivants :

Remarque : Seuls les registres R0 à R3 peuvent être utilisé pour l'adressage indirect ([RG] et [RG+])

op2 \ op1	RG	SFR	mem
RG	X	X	X
SFR	X	X	X
mem	X	X	
#val	X	X	
[RG3]	X		
[RG3+]	X		

MUL op1,op2 *opération* : MDH || MDL ← op1 * op2
Les opérandes sont considérés comme des entiers relatifs.

MULU op1,op2 *opération* : MDH || MDL ← op1 * op2
Les opérandes sont considérés comme des entiers naturels.

DIV oper *opération* : MDL ← quotient de MDL / oper
MDH ← reste de MDL / oper
L'opérande est sur 16 bits et la division porte sur des entiers relatifs.

DIVU oper *opération* : MDL ← quotient de MDL / oper
MDH ← reste de MDL / oper
L'opérande est sur 16 bits et la division porte sur des entiers naturels.

DIVL oper *opération* : MDL ← quotient de MDH || MDL / oper
MDH ← reste de MDH || MDL / oper
L'opérande est sur 16 bits et la division porte sur des entiers relatifs.

DIVLU oper *opération* : MDL ← quotient de MDH || MDL L / oper
MDH ← reste de MDH || MDL / oper
L'opérande est sur 16 bits et la division porte sur des entiers naturels.

Pour ces 6 instructions op1 , op2 et oper doivent être RG

NEG RG *opération* : op1 ← -op1. Le changement de signe est fait sur 8 ou 16 bits. Le 167 connaît en fait deux instructions :

NEGW RG Le changement de signe est fait sur 16 bits.
NEGB RG Le changement de signe est fait sur 8 bits.

Logique

CPL RG *opération* : RG ← complément de RG. Le complément est fait sur 8 ou 16 bits. Le 167 connaît en fait 2 instructions :

CPLW RG Le complément de signe est fait sur 16 bits.
CPLB RG Le complément de signe est fait sur 8 bits.

OR op1 , op2 *opération* : op1 ← op1 OU op2. Sur 8 ou 16 bits. Le 167 connaît en fait 2 instructions :

ORW op1 , op2 Sur 16 bits
ORB op1 , op2 Sur 8 bits

AND op1 , op2 *opération* : op1 ← op1 ET op2. Sur 8 ou 16 bits. Le 167 connaît en fait deux instructions :

ANDW op1 , op2 Sur 16 bits
ANDB op1 , op2 Sur 8 bits

XOR op1 , op2 *opération* : op1 ← op1 OU EXCLUSIF op2. Sur 8 ou 16 bits. Le 167 connaît en fait deux instructions :

XOR	op1 , op2	Sur 16 bits
XORB	op1 , op2	Sur 8 bits

Les opérandes possibles sont les mêmes que pour les opérations d'addition / soustraction (voir au dessus)

Décalages

SHR op1 , op2 *opération* : op1 est décalé op2 fois à droite (décalage logique). op1 doit être RG (sur 16 bits). op2 doit être RG ou #val.

SHL op1 , op2 *opération* : op1 est décalé op2 fois à gauche (décalage logique). op1 doit être RG (sur 16 bits). op2 doit être RG ou #val.

ASHR op1 , op2 *opération* : op1 est décalé op2 fois à droite (décalage arithmétique). op1 doit être RG (sur 16 bits). op2 doit être RG ou #val.

ROR op1 , op2 *opération* : op1 est décalé op2 fois à droite (décalage cyclique). op1 op1 doit être RG (sur 16 bits). op2 doit être RG ou #val.

ROL op1 , op2 *opération* : op1 est décalé op2 fois à gauche (décalage cyclique). op1 doit être RG (sur 16 bits). op2 doit être RG ou #val.

Comparaisons

CMP op1 , op2 *opération* : compare op1 à op2. Les 2 opérandes sont sur 8 ou 16 bits. Le 167 connaît en fait deux instructions :

CMP	op1 , op2	Les 2 opérandes sont sur 16 bits.
CMPB	op1 , op2	Les 2 opérandes sont sur 8 bits.

Les opérandes doivent être les suivants :

op2 \ op1	RG	SFR
RG	X	
SFR	X	X
mem	X	X
#val	X	X
[RG3]	X	
[RG3+]	X	

CMPI1 RG , op2 *opération* : compare RG à op2 puis incrémente RG. Les 2 opérandes sont sur 16 bits.

CMPI2 RG , op2 *opération* : compare RG à op2 puis ajoute 2 à RG. Les 2 opérandes sont sur 16 bits.

CMPD1 RG , op2 *opération* : compare RG à op2 puis décrémente RG. Les 2 opérandes sont sur 16 bits.

CMPD2 RG , op2 *opération* : compare RG à op2 puis enlève 2 à RG. Les 2 opérandes sont sur 16 bits.

Pour les 4 instructions précédentes op2 peut être RG ou SFR ou mem ou #val.

Opérations sur les bits

BSET dest.n° *opération* : mise à 1 du bit de dest désigné par n°

BCLR dest.n° *opération* : mise à 0 du bit de dest désigné par n°

BMOV dest.nd , source.ns *opération* : bit de dest désigné par nd ← bit de source désigné par ns

BMOVN dest.nd , source.ns *opération* : bit de dest désigné par nd ← complément du bit de source désigné par ns

BAND dest.nd , source.ns *opération* : bit de dest désigné par nd ← bit de dest désigné par nd ET bit de source désigné par ns

BOR dest.nd , source.ns *opération* : bit de dest désigné par nd ← bit de dest désigné par nd OU bit de source désigné par ns

BXOR dest.nd , source.ns *opération* : bit de dest désigné par nd ← bit de dest désigné par nd OU EXCLUSIF bit de source désigné par ns

BCMP dest.nd , source.ns *opération* : comparaison du bit de dest désigné par nd avec celui de source désigné par ns

Les indicateurs positionnés sont :
- Z si les 2 bits sont à 0
- V si l'un des 2 au moins est à 1
- C si les 2 sont à 1
- N s'ils sont différents

On peut donc utiliser les branchements selon la signification suivante :

les 2 bits sont à 0	cc_Z
les 2 bits sont à 1	cc_C
l'un au moins est à 1	cc_V
l'un au moins est à 0	cc_NC
les 2 bits sont égaux	cc-NN
les 2 bits sont différents	cc-N

BFLDH dest , #masque , #val *opération* : octet de gauche de dest ← #val uniquement dans les bits désignés par des 1 dans masque. Les bits de val en vis à vis de 0 dans masque doivent être à 0.

BFLDL dest , #masque , #val *opération* : octet de droite de dest ← #val uniquement dans les bits désignés par des 1 dans masque. Les bits de val en vis à vis de 0 dans masque doivent être à 0.

Pour ces instructions dest doit être adressable bit par bit c'est à dire qu'il doit s'agir
- d'un registre RG
- ou d'un registre SFR adressable par bit (tous ne le sont pas)
- ou d'une variable placé dans la zone adressable bit par bit de la mémoire

Branchements

JMP etiq *opération* : branchement inconditionnel à l'étiquette

JMP cc , etiq *opération* : branchement conditionnel à l'étiquette

JMP cc , [RG] *opération* : branchement conditionnel à l'adresse contenue dans RG

Le branchement sans condition s'exprime par **cc_UC**. Pour les différentes conditions de branchement exprimées par cc se référer à la table ci-dessous :

condition	comparaison d'entiers naturels	comparaison d'entiers relatifs
égal	**cc_EQ**	**cc_EQ**
différent	**cc_NE**	**cc_NE**
inférieur	**cc_ULT**	**cc_SLT**
inférieur ou égal	**cc_ULE**	**cc_SLE**
supérieur	**cc_UGT**	**cc_SGT**
supérieur ou égal	**cc_UGE**	**cc_SGE**
débordement	**cc_C**	**cc_V**
non débordement	**cc_NC**	**cc_NV**
positif		**cc_NN**
négatif		**cc_N**

Il existe une dernière condition exprimée par **cc_NET** qui est vrai lorsque les deux valeurs comparées sont différentes et sont aussi différentes de 80h si les opérandes comparés étaient sur 8 bits et de 8000h s'ils étaient sur 16 bits. Ce type de branchement peut être utilisé pour écrire des recherches dans des tables dont 80h ou 8000h serait le symbole de terminaison.

Le 167 connaît en fait 4 instructions :

JMPA	cc , etiq	branchement conditionnel à l'étiquette sans changer de segment de code.
JMPR	cc , etiq	branchement conditionnel à l'étiquette sans changer de segment de code. La différence avec la précédente est que l'étiquette désignée doit être près (-127 à 127 octets de distance)
JMPI	cc , [RG]	branchement conditionnel à l'adresse contenue dans RG sans changer de segment de code
JMPS	seg , etiq	branchement inconditionnel à l'étiquette avec changement de segment de code (l'opérande seg est 0 , 1 , 2 ou 3)

JB	oper.n° , etiq	*opération* : branchement à l'étiquette si le bit de oper désigné par n° est à 1
JNB	oper.n° , etiq	*opération* : branchement à l'étiquette si le bit de oper désigné par n° est à 0
JBC	oper.n° , etiq	*opération* : branchement à l'étiquette si le bit de oper désigné par n° est à 1 et mise à 0 de ce bit
JNBC	oper.n° , etiq	*opération* : branchement à l'étiquette si le bit de oper désigné par n° est à 0 et mise à 1 de ce bit

Pour les 4 instructions précédentes dest doit être adressable par bit c'est à dire qu'il doit s'agir
d'un registre RG
ou d'un registre SFR adressable par bit (tous ne le sont pas)
ou d'une variable placé dans la zone adressable par bit de la mémoire

Pile

PUSH	oper	*opération* : empile oper. L'opérande peut être RG ou SFR sur 16 bits.
POP	oper	*opération* : dépile oper. L'opérande peut être RG ou SFR sur 16 bits.
SCXT	op1 , op2	*opération* : empile op1 puis exécute op1← op2. op1 peut être RG ou SFR sur 16 bits tandis que op2 peut être #val ou RG ou SFR ou mem

Cette dernière instruction est utilisée pour changer de contexte (op1 est alors le registre CP qui sert de pointeur vers les registres généraux RG)

Sous programmes et interruptions

CALL	etiq	*opération* : appel inconditionnel de procédure
CALL	cc , etiq	*opération* : appel conditionnel de procédure
CALL	cc , [RG]	*opération* : appel conditionnel de la procédure à l'adresse contenue dans RG

Le 167 connaît en fait 4 instructions :

CALLA	cc , etiq	*opération* : appel conditionnel de procédure sans changer de segment de code.
CALLR	etiq	*opération* : appel inconditionnel de procédure sans changer de segment de code. La procédure désignée doit être près (-127 à 127 octets de distance)
CALLI	cc , [RG]	*opération* : appel conditionnel de la procédure à l'adresse contenue dans RG sans changer de segment de code.

Les 3 instructions précédentes empilent le compteur ordinal (IP)

CALLS	seg , etiq	*opération* : appel inconditionnel de procédure avec changement de segment de code (l'opérande seg est 0 , 1 , 2 ou 3)

Cette instruction empile le compteur ordinal (IP) et le pointeur de segment de code (CSP)

PCALL	oper , etiq	*opération* : appel inconditionnel de procédure précédé d'un empilement de oper. oper doit être RG ou SFR sur 16 bits
RET		*opération* : retour de sous-programme.

Le 167 connaît en fait 3 instructions :

RET		*opération* : retour de sous-programme sans changer de segment de code.
RETS		*opération* : retour de sous-programme avec changement de segment de code (voir CALLS).
RETI		*opération* : retour de procédure d'interruption
RETP	oper	*opération* : retour de sous-programme avec dépilement de oper qui

doit être RG ou SFR sur 16 bits (vois PCALL).

TRAP #n° *opération* : appel de l'interruption de numéro n°
Les indicateurs (PSW), le registre de segment de code (CSP) et le compteur ordinal (IP) sont empilés puis la procédure d'interruption désignée par n° (0 à 127) est lancée.

Instructions de contrôle

EXTR #num *opération* : instruction permettant l'accès aux registres SFR en zone étendue (adresses F000 à F1FF), num désigne le nombre d'instructions suivantes ayant accès à cette zone (1 à 4)

ATOMIC #num *opération* : instruction permettant d'indiquer que les num instructions suivantes ne peuvent pas être interrompues (1 à 4)

Les instructions suivantes sont protégées :

SRST *opération* : réinitialisation (reset) du 167

IDLE *opération* : le processeur se met en repos (seule l'unité d'échange fonctionne). Il ne sortira de cet état que par une interruption.

PRWDN *opération* : arrêt du 167, seul un reset physique peut le relancer

SRVWDT *opération* : réamorce l'horloge 'chien de garde'

DISWDT *opération* : invalide l'horloge 'chien de garde'

EINIT *opération* : termine les initialisations. Les instructions protégées ne sont plus utilisables.

NUMEROS DES INTERRUPTIONS DE PERIPHERIQUES

Cause	Registre associé	Numéro
Timer 2 (GPT1)	T2IC	34
Timer 3 (GPT1)	T3IC	35
Timer 4 (GPT1)	T4IC	36
Timer 5 (GPT2)	T5IC	37
Timer 6 (GPT2)	T6IC	38
Fin de conversion A/N	ADCIC	40
Perte d'information sur le convertisseur A/N	ADEIC	41
Générateur MLI (PWM)	MWMIC	63
Interruption externe ligne 8 de P2	EXICON	24
Interruption externe ligne 9 de P2	EXICON	25
Interruption externe ligne 10 de P2	EXICON	26
Interruption externe ligne 11 de P2	EXICON	27
Interruption externe ligne 12 de P2	EXICON	28
Interruption externe ligne 13 de P2	EXICON	29
Interruption externe ligne 14 de P2	EXICON	30
Interruption externe ligne 15 de P2	EXICON	31

L'ASSEMBLEUR DU 167

La syntaxe d'une ligne d'assembleur est :

[etiquette :] COP [operande1 [, operande2]] [; commentaire]
ou ; commentaire

Désignation des opérandes dans les instructions autres que de branchement :

Registre : On met le nom du registre qui contient l'opérande
exemple : MOV R0,R1

Valeur immédiate : On met le symbole # suivi de la valeur de l'opérande selon la forme :
décimal #123 #-4
hexadécimal #1ah #0ffa3h (le 0 à gauche est nécessaire si la valeur commence par une lettre)
binaire #11001b
ASCII #'a' #'Bonjour'
exemples : MOV R0,#ff03h
MOV RL0,#'A'

Direct : On met le nom de la variable
exemple : MOV R0,var1

Indirect : 4 types sont possibles :

- par registre : On met le nom du registre entre crochets.
[RG] désigne l'objet pointé par le registre RG
exemple : MOV R0,[R1]

- par registre avec déplacement : On met le nom du registre et la valeur du déplacement entre crochets.
[RG+#depl] désigne l'objet pointé par le registre RG augmenté de depl
exemple : MOV R0,[R1+#4]

- par registre avec post incrémentation : On met le nom du registre suivi du symbole +, le tout entre crochets.
[RG+] désigne l'objet pointé par le registre RG, RG sera incrémenté d'une valeur de 1 si l'instruction porte sur un octet et de 2 si elle porte sur 16 bits
exemples : MOV R0,[R1+] R1 est augmenté de 2 après le MOV
MOV RH0,[R1+] R1 est augmenté de 1 après le MOV

- par registre avec pré décrémentation : On met le nom du registre précédé du symbole -, le tout entre crochets.
[-RG] désigne l'objet pointé par le registre RG, RG sera décrémenté d'une valeur de 1 si l'instruction porte sur un octet et de 2 si elle porte sur 16 bits
exemples : MOV R0,[-R1] R1 est diminué de 2 avant le MOV
MOV RH0,[-R1] R1 est diminué de 1 avant le MOV

Bit : On met le nom du registre ou de la variable qui contient l'opérande suivi d'un point et du numéro de bit. Le registre ou la variable doivent être adressables bit par bit.
exemples : BSET R0.4
BCMP var.0,R4.5

Désignation des opérandes dans les instructions de branchement :

Direct : On met l'étiquette qui désigne l'instruction à laquelle on doit aller.

exemple : JMP cc_BGE,debut

Registre : On met, entre crochets, le nom du registre qui contient l'adresse de l'instruction à laquelle on doit aller.

exemple : JMP cc_UC,[R0]

La segmentation

On définit un segment ou une page par 2 directives. L'une au début et l'autre à la fin :

segment de code :

```
nomc    SECTION CODE WORD       au début et
nomc    ENDS                    à la fin
```

page de données :

```
nomd    SECTION DATA            au début et
nomd    ENDS                    à la fin
```

page de données adressables bit par bit:

```
nomb    SECTION DATA BITADDRESSABLE    au début et
nomb    ENDS                           à la fin
```

Il faut indiquer au compilateur l'initialisation de DPP3 sur la page de données permettant l'accès aux registres par la directive :

ASSUME DPP3:SYSTEM

Par contre, les autres registres de page doivent être initialisés par le programme grâce à la séquence :

mov DPPi,#pag nomd

placée en tout début de programme et le compilateur en est informé par la directive :

ASSUME DPPi:nomd

Remarque : seul DPP0 est utilisable pour une page adressable bit par bit.

Pointeurs sur des variables

Une variable est désignée par son adresse à l'intérieur d'une page elle même désignée par un registre de page DPP0 à DPP3.

Pour initialiser un registre de page (DPPi) pour désigner une page on fera :

MOV DPPi,#**pag** nomd

où nomd désigne le nom de la page

Pour initialiser un registre général (Ri) comme pointeur sur une variable (vard) on fera :

MOV Ri,#vard

Constantes et variables

Constantes : Elles sont définies par :

nom EQU valeur

Variables initialisées : Elles sont déclarées et initialisées par :

nom taille liste de valeurs

Elle ne peuvent pas être placées dans des sections adressables bit par bit.

taille est l'un des mots suivants :

DB qui désigne 1 octet
DW qui désigne un mot

La liste de valeurs est constituée d'une suite d'éléments séparés par des virgules.
exemple : var DB 12,0ffh, 'ABC' var est sur 5 octets et contient :

12 dans le premier octet
0ffh dans le deuxième
65 (code ASCII de A) dans le troisième
66 (code ASCII de B) dans le quatrième
67 (code ASCII de C) dans le cinquième

Variables non initialisées : Elles sont déclarées par :

nom taille nombre

taille est l'un des mots suivants :

DBIT qui désigne 1 bit
DS qui désigne 1 octet
DSW qui désigne un mot

nombre indique combien d'unité de taille sont réservées

exemples :

var1 DBIT 12 var1 est sur 12 bits.
Elle doit être placée dans une section de données adressable bit par bit
var2 DSW 4 var2 occupe 4 mots. Elle peut être placée dans n'importe quelle section de données adressable bit par bit ou non

Définition d'une banque de registres

De par l'initialisation faite par START167 le programme dispose d'une banque de registres prédéfinie. Toutefois il peut être utile d'en définir d'autres pour pouvoir rapidement changer de contexte par exemple lors de la prise en compte d'interruptions. Cette définition se fera par la directive suivante :

nombanque REGBANK

On pourra ensuite initialiser CP pour utiliser cette nouvelle banque de registres par :

MOV CP,#nombanque si on ne veut pas le sauvegarder
et par SCXT CP,# nombanque si on veut le sauvegarder et le restituer par un POP CP

Procédures

Une procédure est définie par :

nom PROC

code de la procédure terminé par RET

nom ENDP

Programme principal

En raison de l'utilisation du code d'initialisation du système contenu dans le fichier START167.a66, le programme principal est une procédure impérativement appelée **main.**

Procédures d'interruption

Une procédure d'interruption est définie par :

nom PROC INTERRUPT nomit=n° USING mesregs

code de la procédure terminé par RET ou RETI

nom ENDP

nomit : nom de l'interruption. Il pourra être utilisé pour appeler cette interruption par une instruction TRAP sous la forme :

```
TRAP    nomit
```

n° : numéro de l'interruption (0 à 127) permettant de lui faire correspondre le vecteur d'interruption

mesregs : nom d'une banque de registres définie par la directive REGBANK (voir ci-dessus)
Il ne faudra pas oublier d'initialiser en début de procédure le registre CP pour utiliser ces nouveaux registres par les instructions :

```
SCXT    CP,#mesregs
NOP
```

Et d'en restituer l'ancienne valeur avant de terminer par les instructions :

```
POP     CP
RETI
```

Squelette d'un programme en assembleur

	code en assembleur	explication
	$MODINF (43)	type de processeur
	$INCLUDE (reg167.inc)	déclaration des registres
	ASSUME DPP3:SYSTEM	page contenant les registres
donnees	SECTION DATA	page de données
	définition des variables et constantes	sera rendue accessible par DPP2
donnees	ENDS	
bits	SECTION DATA BITADDRESSABLE	page de données adressable bit par bit
	définition des variables et constantes	accessible par DPP0
bits	ENDS	
prog	SECTION CODE WORD	segment de code
main	PROC NEAR	programme principal
	GLOBAL main	pour être connu de la procédure d'initialisation
	ASSUME DPP2:donnees	pour pouvoir accéder à nos données
	mov DPP2,#pag donnees	
	ASSUME DPP0:bits	pour pouvoir accéder à la page de données
	mov DPP0,#pag bits	adressable bit par bit
	..... code	retour à la procédure d'initialisation
	ret	fin du prog principal
main	ENDP	début d'une procédure
sp1	PROC	
	..... corps de la procédure 1	fin de la procédure
sp1	ENDP	
.....		
spn	PROC	etc.
	..... corps de la procédure n	
spn	ENDP	fin du segment de code
prog	ENDS	fin du fichier
	END	

Compilation séparée

Il est possible d'écrire des programmes en assembleur constitués de modules compilés séparément. Le lien entre ces modules se fait alors lors de l'édition de liens.

Les étiquettes (nom de variable , de constante ou de procédure) contenues dans un module et devant pouvoir être vues depuis un autre seront repérées dans le module où elles sont déclarées par la directive:

PUBLIC etq1 , etq2 , ...

Elles seront réintroduites dans chacun des modules qui les utiliseront par la directive :

EXTRN etq1 : type1 , etq2 : type2 , ...

Les types sont : BIT, BYTE , WORD , BITWORD pour les variables (bit, octet, mot de 16 bits ou mot adressable bit par bit). Ces déclarations devront apparaître dans une section de données approprié de façon à ce ces variables soient accessibles à l'aide du registre de page associé à cette section

DATA3, DATA4, DATA8 ou DATA16 pour les constantes définies par EQU selon leur taille en bit

NEAR ou FAR pour les procédures selon quelles sont ou pas dans le même segment