L’assembleur du MSP430

Définition

Un langage d'assemblage (ou langage assembleur ou simplement assembleur par abus de langage, abrégé ASM) est un **langage de bas niveau** proche du **langage machine** qui peut être **directement interprété par un processeur** tout en restant lisible par un utilisateur.

Il consiste à représenter les **combinaisons de bits employées en langage binaire** par des **symboles appelés mnémoniques** (du grec mnêmonikos, relatif à la mémoire), c'est-à-dire faciles à retenir

Par exemple, l'unité de contrôle d'un processeur particulier reconnaît l'instruction en langage machine suivante :

⇨ en hexadécimal : 4035 0055 ou en binaire : 01000000 00110101 00000000 01010101

En langage assembleur, cette instruction sera traduite par un équivalent plus facile à comprendre pour le programmeur

⇨ MOV.W #85H,R5

Ce qui signifie "mettre la valeur décimale 85 (0x55 en hexadécimal) dans le registre R5.

Les outils de développement

Routines particulières écrites en langage assembleur

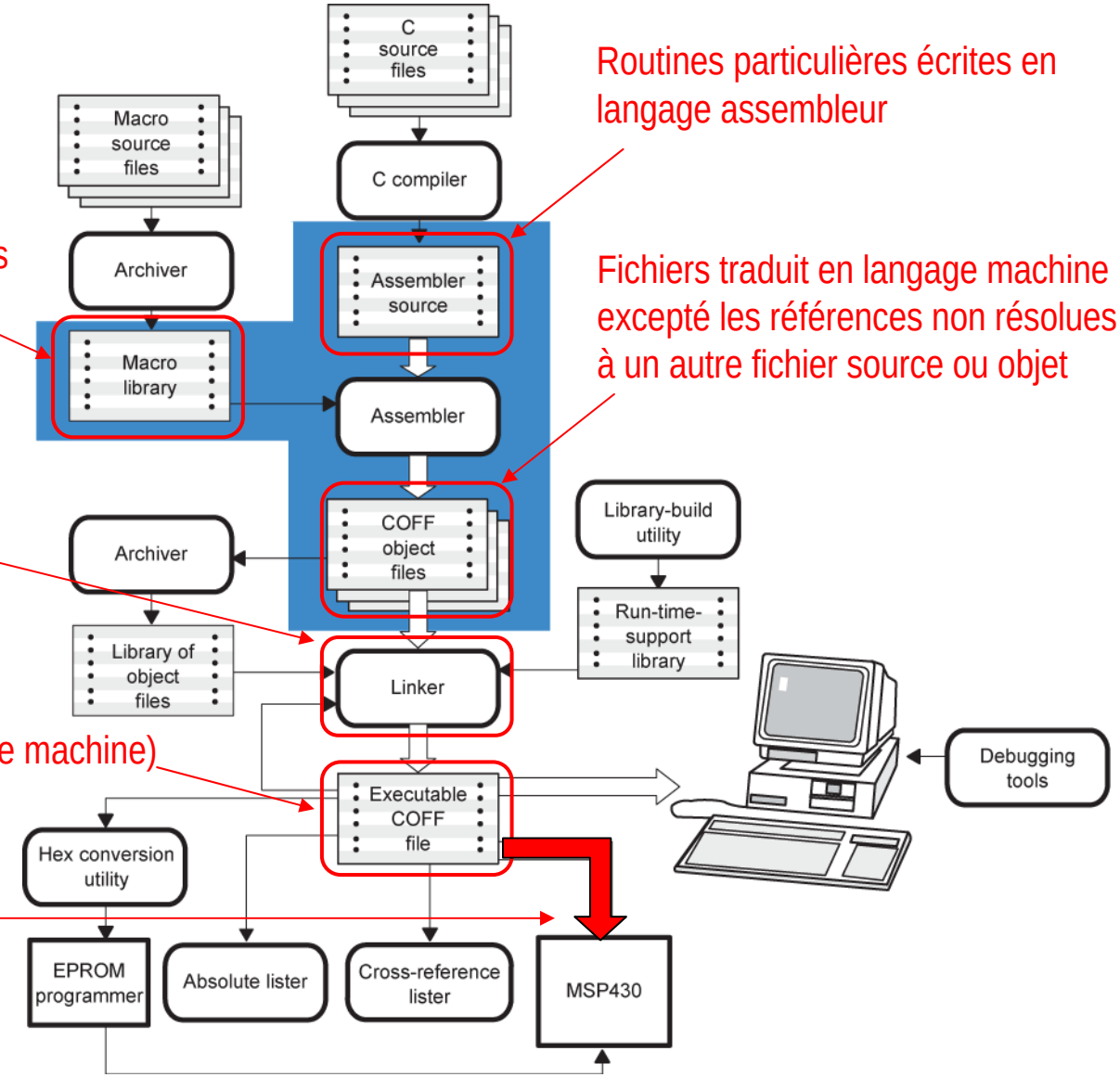
Routines ou macro instructions génériques

Fichiers traduit en langage machine excepté les références non résolues à un autre fichier source ou objet

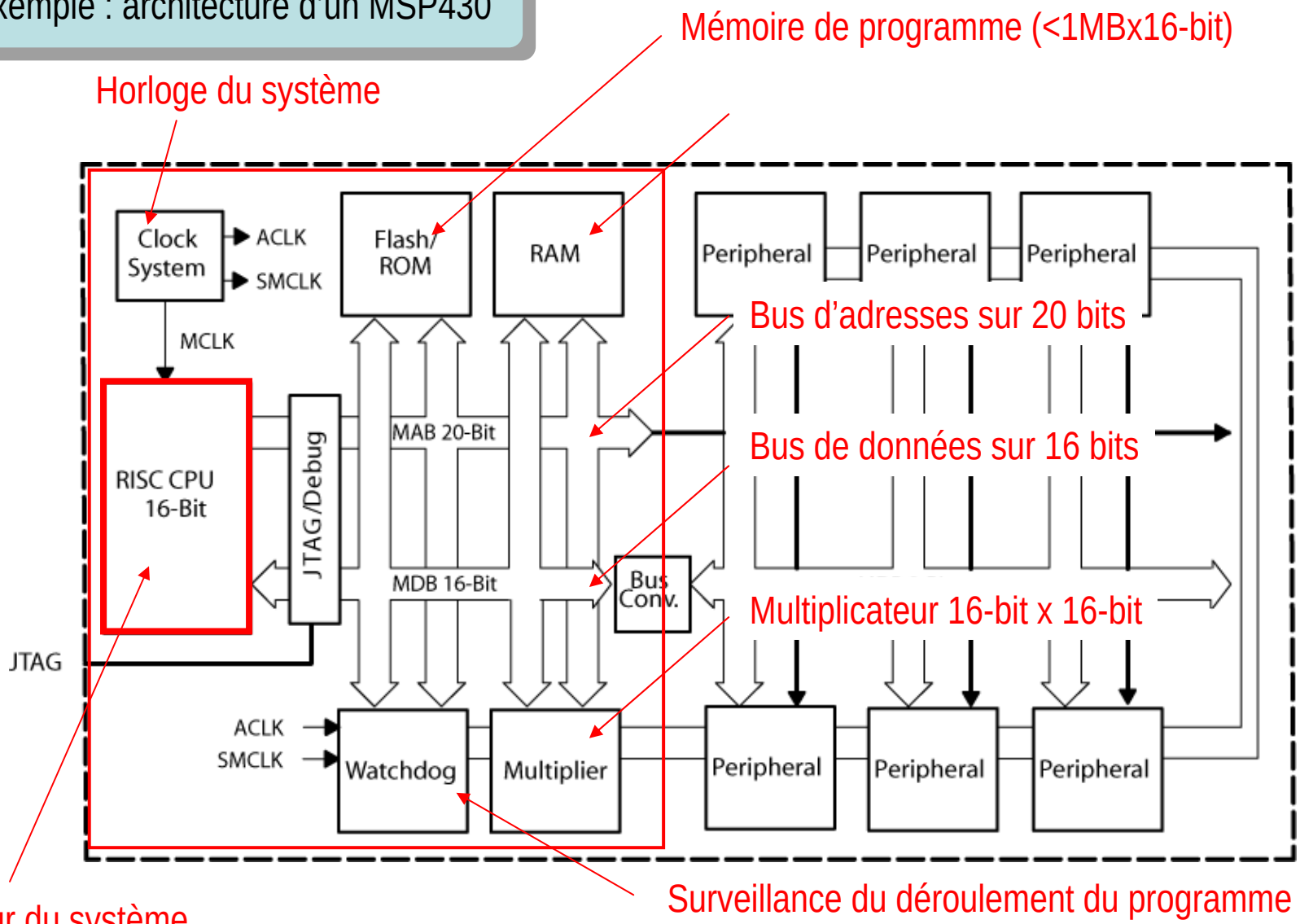
Éditeur de lien permettant de résoudre les références à d'autres fichiers sources ou objets

Fichier contenant le code exécutable (code machine)

Lien avec la cible pour le dépannage (debuggage) de l'application



Exemple : architecture d'un MSP430



Unité de calcul et registres spéciaux (SFR)

Les registres ont une largeur de 20 bits.

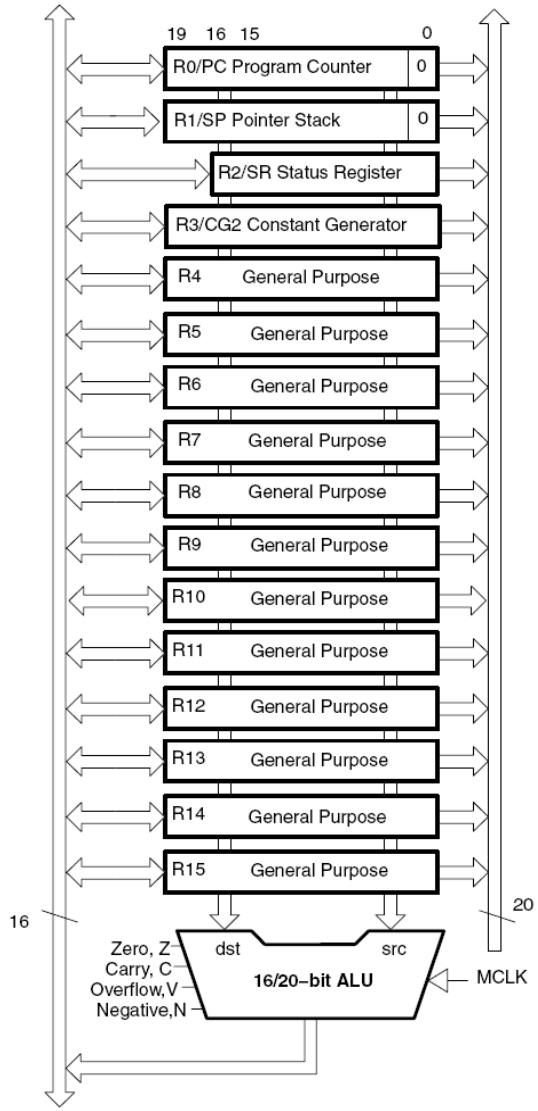
- le registre R0 est dédié au compteur de programme (PC),
- le registre R1 contient le pointeur de pile (SP),
- le registre R2 contient entre autre le registre d'état (SR),
- les registres R2 et R3 sont des générateurs de constantes.

L'unité de calcul à une largeur de 16 (20) bits, elle permet d'effectuer :

- des calculs arithmétiques (additions, soustraction, comparaison),
- des opérations logiques (AND, OR, XOR).

Des indicateurs permettent de caractériser les résultats

- résultat nul,
- résultat négatif,
- dépassement de capacité,
- retenue pour les calculs multiprécision.



Caractéristiques de l'unité centrale de traitement (CPU)

Les dispositifs de l'unité centrale de traitement incluent :

- une architecture RISC,
- une architecture orthogonale, chaque instruction est utilisable avec chaque mode d'adressage,
- un accès direct aux registres (SFR special function register) comprenant

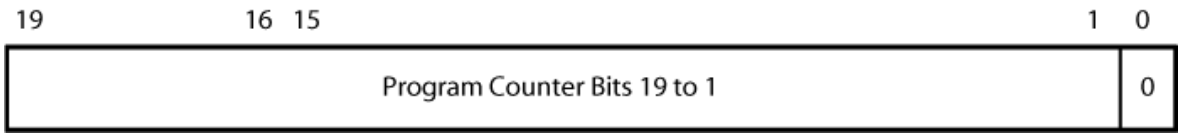
  - le compteur de programme (PC : program pointer),
  - le registre d'état (SR : state register),
  - le pointeur de pile (SP stack pointer),

- des opérations sur les registres SFR en un cycle d'horloge,
- un grand nombre de registres afin de diminuer la phase de recherche en mémoire,
- un bus d'adresses de 20 bits pour un accès direct sur toute la plage de mémoire,
- un bus de données de 16 (20) bits permettant la manipulation directe des données,
- un générateur de constantes fournissant six valeurs immédiates les plus utilisées,
- le transfert direct de mémoire-à-mémoire sans utilisation de registres intermédiaires,
- un format d'instruction sur un byte (8 bits), un mot (16 bits) ou un mot étendu (20 bits).

Registres spécifiques à l'unité centrale de traitement

Compteur de programme (PC)

Le compteur de programme de 20 bits (PC/R0) indique la prochaine instruction à exécuter. Chaque instruction emploie un nombre pair de bytes (deux, quatre, six ou huit), le PC est incrémenté en conséquence. Les instructions d'accès dans l'espace mémoire (jusqu'à 1MB) sont effectuées sur des mots (20 bits), par conséquent PC est aligné sur des adresses paires



Registres spécifiques à l'unité centrale de traitement

Compteur de programme (PC)

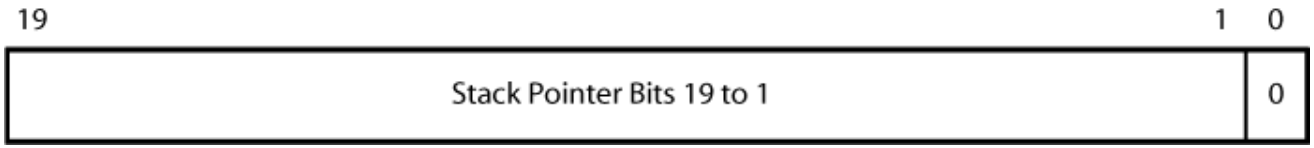
Le PC peut être adressé par toutes les instructions et tous les modes d'adressage.
Quelques exemples :

```
MOV.W   #LABEL,PC   ; Branchement à l'adresse LABEL (64KB inférieur)
MOVA    #LABEL,PC   ; Branchement à l'adresse LABEL (1MB)
MOV.W   LABEL,PC    ; Branchement à l'adresse contenue dans LABEL (64KB inférieur)
MOV     @R4,PC      ; Branchement  indirect à l'adresse continue R4 (64KB inférieur)
ADDA    #4,PC       ; Saut de deux mots (4 positions mémoire sur 1MB)
```

Registres spécifiques à l'unité centrale de traitement

Pointeur de pile (SP)

Le pointeur de pile de 20 bits (SP/R1) est employé par l'unité de traitement pour stocker les adresses de retour des appels de sous-routines et des routines interruptions.
Il emploie une pré-décrémentation et une post-incrémentation.
Le pointeur de pile peut être employé pour toutes les instructions et modes d'adressage.
Le pointeur de pile est initialisé dans la RAM par l'utilisateur, et il est aligné sur des adresses paires

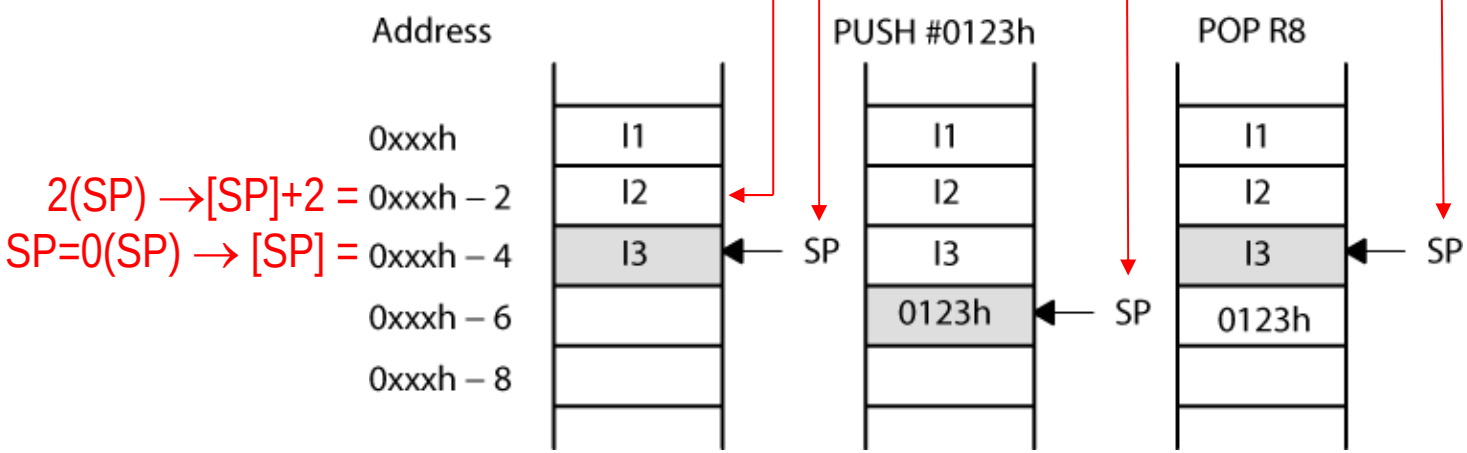


Registres spécifiques à l'unité centrale de traitement

Pointeur de pile (SP)

Quelques exemples d'utilisation du pointeur de pile :

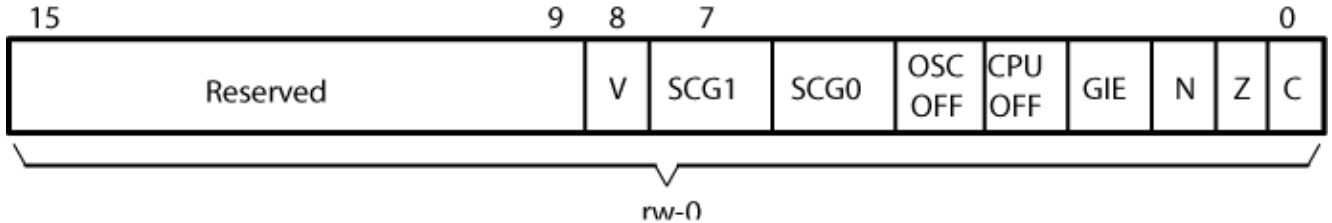
```
MOV   2(SP),R6   ; [SP]+2 -> R6
MOV   R7,0(SP)   ; R7 -> [SP]+0
PUSH  #0123h     ; 0123h -> [SP]
POP   R8         ; [SP] -> R8 (= 0123h)
```



Registres spécifiques à l'unité centrale de traitement

Registre d'état (SR)

Le registre d'état (SR/R2), utilisé comme registre de source ou de destination, ne peut être adressé sur toute sa longueur (16 bits)



V : Indicateur de dépassement.
Ce bit est mis à 1 lors d'un dépassement sur d'une opération arithmétique entre nombres considérés comme signés

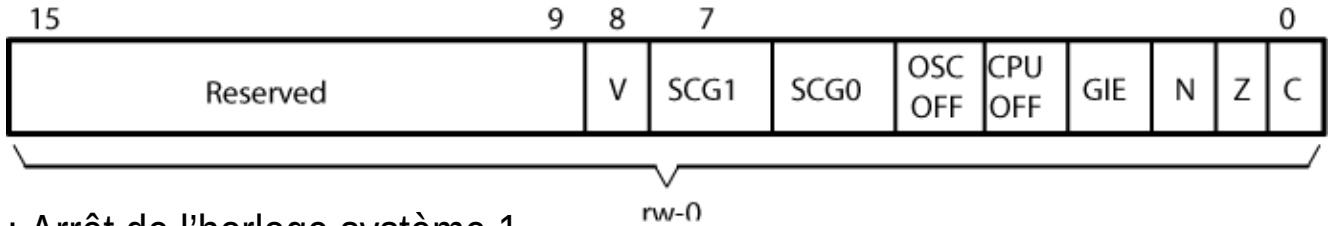
N : Indicateur de nombre négatif.
Ce bit est mis à 1 lorsque le résultat d'une opération arithmétique est négatif (bit de signe égal à 1).

Z : Indicateur de nombre nul.
Ce bit est mis à 1 lorsque le résultat d'une opération arithmétique est nul

C : Indicateur de report.
Ce bit est mis à 1 lorsque le résultat d'une opération arithmétique provoque un report

Registres spécifiques à l'unité centrale de traitement

Registre d'état (SR)



SCG1 : Arrêt de l'horloge système 1.
Ce bit, lorsqu'il est mis à 1 provoque l'arrêt DC0 si DCOCLK n'est pas utilisé comme Master Clock (MCLK)

SCG0 : Arrêt de l'horloge système 0.
Lorsque ce bit est mis à 1, FLL et la boucle de contrôle sont arrêtés

OSCOFF : Arrêt de l'oscillateur interne.
Ce bit, lorsqu'il est mis a 1 provoque l'arrêt de l'oscillateur a quartz LFXT1 si ce dernier n'est pas utilisé comme Master Clock (MCLK) ou Slave Clock (SCLK)

CPUOFF : Arrêt de l'unité de traitement (CPU).
Ce bit est à 1 pour l'arrêt de la CPU.

GIE : Activation (1) / désactivation (0) générale des interruptions.

Registres spécifiques à l'unité centrale de traitement

Registres CG1 et CG2

Six constantes spécifiques sont produites avec les registres générateur de constantes R2(CG1) et R3(CG2), sans exiger un mot de 16 bits supplémentaire pour coder l'instruction

REGISTRES	AS	CONSTANTES	REMARQUES
R2	00	- - - -	Mode registre
R2	01	(0)	Mode d'adressage absolu
R2	10	00004h	+4
R2	11	00008h	+8
R3	00	00000h	0
R3	01	00001h	+1
R3	10	00002h	+2
R3	11	0FFh, 0FFFFh, 0FFFFFh	-1

Registres spécifiques à l'unité centrale de traitement

Registres CG1 et CG2

Les avantages des registres générateur de constantes sont :

- aucunes instructions spéciales requises,
- aucun mot de code additionnel pour les six constantes,
- aucun accès mémoire requis pour rechercher la constante

L'assembleur utilise automatiquement les registres générateurs de constantes si une des six constantes prédéfinie est employée comme opérande immédiat de source.
Les registres R2 et R3, utilisés comme générateurs de constantes, ne peuvent pas être adressés explicitement ; ils agissent en tant que registres source.

Registres spécifiques à l'unité centrale de traitement

Registres CG1 et CG2

L'ensemble RISC du MSP430 comprend seulement 27 instructions.
Cependant, les registres générateur de constantes permettent au MSP430 d'avoir 24 instructions additionnelles émulées.

Par exemple, l'instruction à une seule d'opérande :

CLR dst

est émulé par l'instruction double-opérande de même longueur :

MOV R3,dst

où #0 est remplacé par l'assembleur R3 est utilisé avec As=00.

exemple CLR R5 → MOV #0,R5

INC dst

est remplacé par :

ADD R3,dst

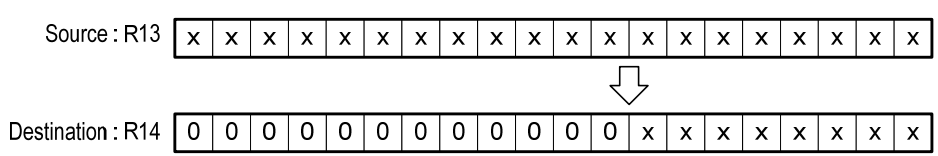
exemple INC R5 → ADD #1,R5

Registres spécifiques à l'unité centrale de traitement

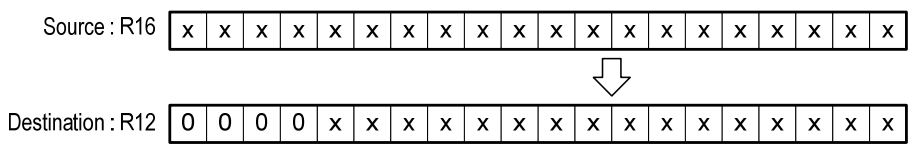
Registres à usage général R4 à R15

Les douze registres R4 à R15 de l'unité de traitement, peuvent être utilisés sur des largeurs de 8 bits (suffixe .B), 16 bits (suffixe .W), ou 20 bits (suffixe .A).
L'écriture d'un byte force la mise à 0 des bits [19…8].
L'écriture d'un mot force les bits [19…16] à 0.
La seule exception est l'instruction SXT qui réalise l'extension du signe sur l'ensemble.

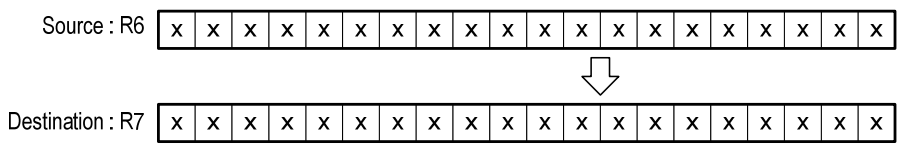
MOV.B R13,R14



MOV.W R16,R12



MOV.A R6,R7



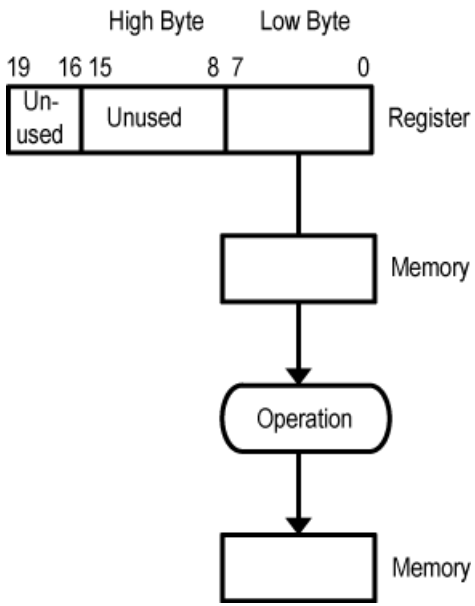
Registres spécifiques à l'unité centrale de traitement

Registres à usage général R4 à R15

Manipulation de bytes (.B) :

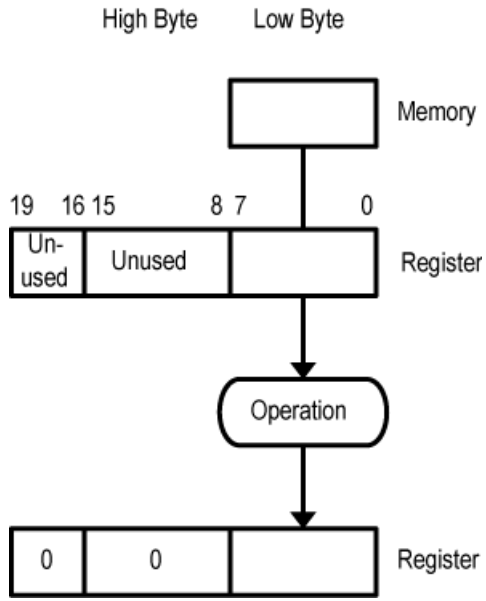
Registre (src) → Mémoire (dest)

ADD.B R8, MEM



Mémoire (src) → Registre (dest)

ADD.B MEM, R8



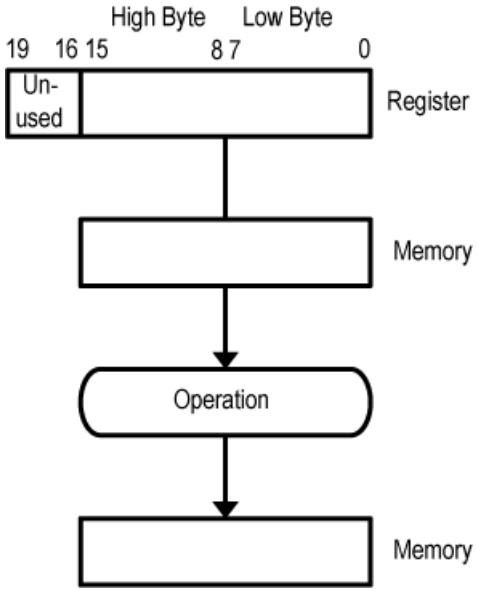
Registres spécifiques à l'unité centrale de traitement

Registres à usage général R4 à R15

Manipulation de mot (.W) :

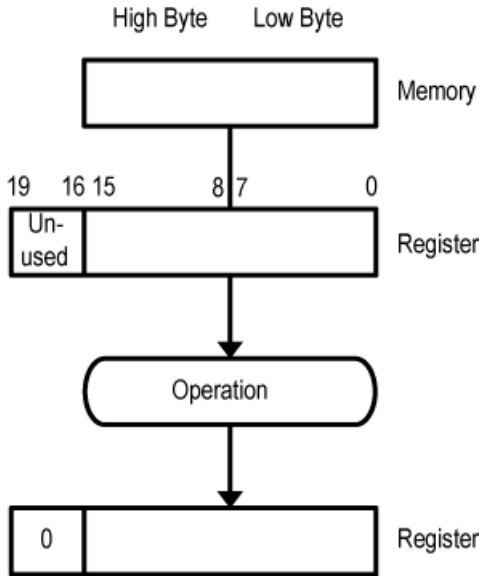
Registre (src) → Mémoire (dest)

ADD.W R8, MEM



Mémoire (src) → Registre (dest)

ADD.W MEM, R8



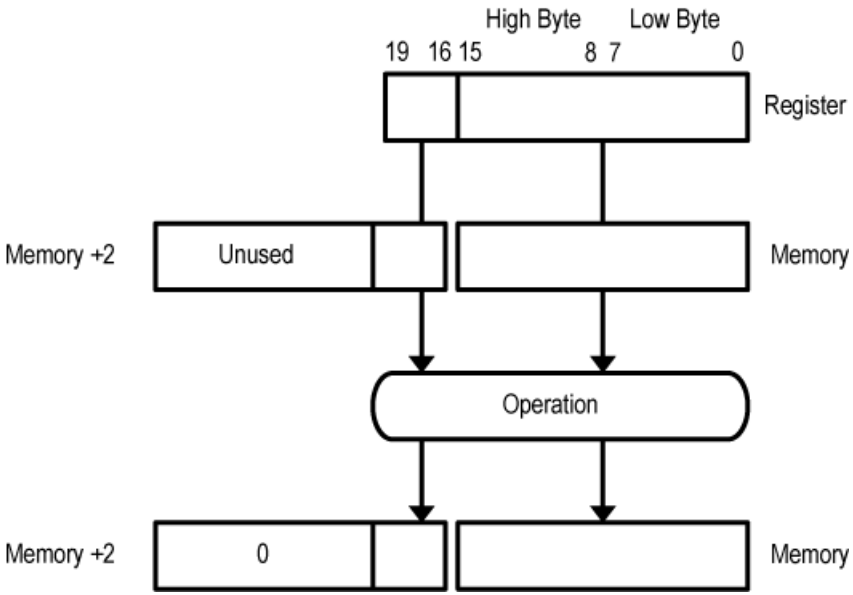
Registres spécifiques à l'unité centrale de traitement

Registres à usage général R4 à R15

Manipulation de mot d'adresse (.A) :

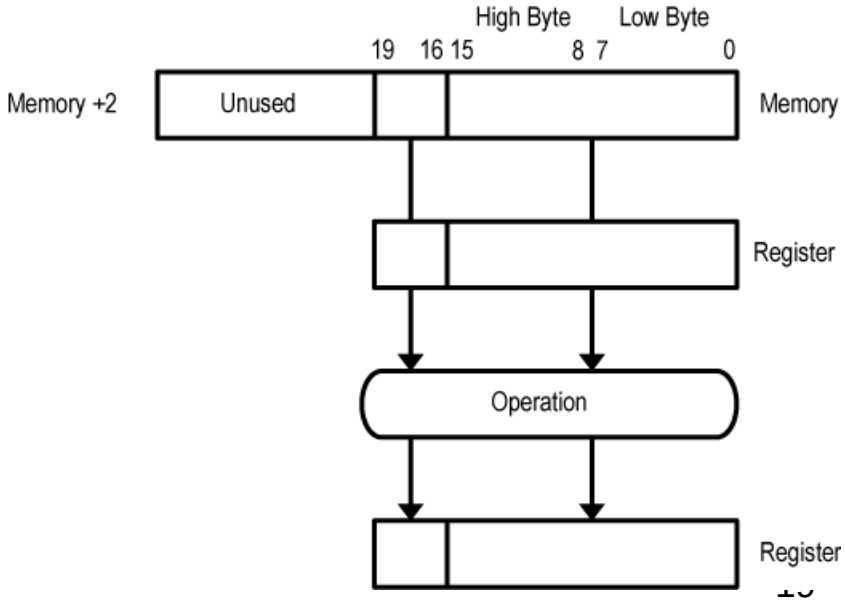
Registre (src) → Mémoire (dest)

ADD.A R8, MEM



Mémoire (src) → Registre (dest)

ADD.A MEM, R8



Modes d’adressage

Sept modes d'adressage pour les opérandes de source et quatre modes d'adressage pour les opérandes de destination permettent d’adresser l'espace d'adressage complet de la mémoire, sans exceptions

Ces modes d’adressage sont valables aussi bien sur de chargement (MOV) que sur des opérations arithmétiques (ADD, SUB, …)

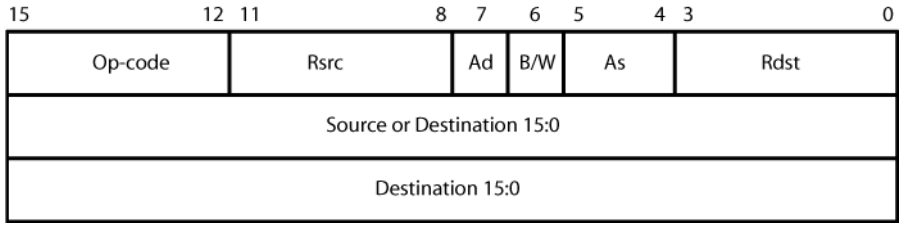
Modes d'adressage

		As/Ad	Mode d'adressage	Syntaxe	Description
Valable pour les opérandes de source	Valable pour les opérandes de destination	00/0	Mode registre	Rn	Opération entre registres.
		01/1	Mode indexé	x(Rn)	[Rn+x] pointe l'opérande. Les instructions utilisant ce mode d'adressage contiennent au moins deux mots. Le second contient x
		01/1	Mode symbolique *Adresse(R0) ⇒ [PC]+Adresse*	Adresse	[PC+x] pointe l'opérande. Il s'agit d'un mode indexé x(PC). Les instructions utilisant ce mode d'adressage contiennent au moins deux mots. Le second contient x
		01/1	Mode absolu *Adresse(R2) ⇒ #0+Adresse*	&Adresse	Le ou les deux mots de 16 bits suivant l'instruction contiennent les valeurs des adresses. Le mode utilisé est le mode indexé x(SR)
		10/-	Mode indirect	@Rn	[Rn] pointe l'opérande
		11/-	Mode indirect avec auto incrémentation	@Rn+	[Rn] pointe l'opérande puis Rn+1→Rn Il y a donc post-incrémentation
		11/-	Mode immédiat *@R0+ ⇒ PC+1 → PC*	#N	Valeur immédiate Les instructions utilisant ce mode d'adressage contiennent au moins deux mots. Le second contient #N Il s'agit d'un mode avec auto-incrémentation PC+1→PC puis [PC] pointe #N

Modes d'adressage en mode registre

As	*Registre*	*Syntaxe*	*Description*
00	Rn	Rn	Mode registre (l'opérande est contenue dans le registre n)
01	Rn	x(Rn)	Mode indexé (l'opérande est dans la mémoire à l'adresse [Rn]+x)
10	Rn	@Rn	Mode indirect (l'opérande est dans la mémoire à l'adresse [Rn])
11	Rn	@Rn+	Mode indexé avec auto incrémentation
Mode d'adressage utilisant le compteur de programme (registre PC)			
01	0(PC)	Label	Mode symbolique x(PC), l'opérande se trouve à l'adresse [PC]+x
11	0(PC)	#x	Mode immédiat @PC+, l'opérande se trouve à l'adresse suivant l'instruction
Mode d'adressage utilisant les registres générateur de constantes R2(SP) et R3(SR)			
01	R2 (SP)	&Label	Mode d'adressage absolu, l'opérande se trouve à l'adresse Label
10	R2 (SP)	#4	Constante, l'opérande est égale à 4
11	R2 (SP)	#8	Constante, l'opérande est égale à 8
00	R3 (SR)	#0	Constante, l'opérande est égale à 0
01	R3 (SR)	#1	Constante, l'opérande est égale à 1
10	R3 (SR)	#2	Constante, l'opérande est égale à 2
11	R3 (SR)	#-1	Constante, l'opérande est égale à -1

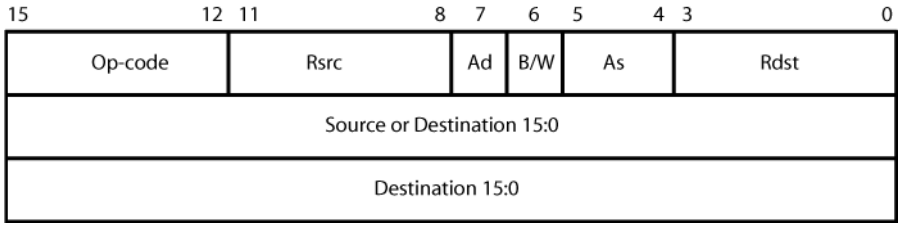
Instruction à deux opérandes (Format I)



* : le bit d'état est affecté
\- : le bit d'état n'est pas affecté
0 : le bit d'état est mis à0
1 : le bit d'état est mis à 1

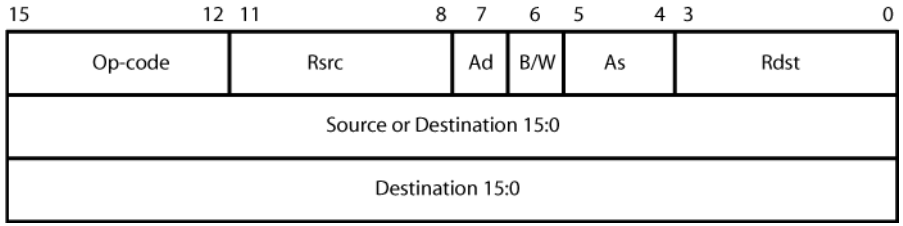
Mnémonique	S-reg, D-reg	Opération	Bits d'état V	N	Z	C
MOV(.B)	src, dst	src → dst	-	-	-	-
ADD(.B)	src, dst	src + dst → dst	*	*	*	*
ADDC(.B)	src, dst	src + dst +C → dst	*	*	*	*
SUB(.B)	src, dst	src - dst → dst	*	*	*	*
SUBC(.B)	src, dst	src - dst +C → dst	*	*	*	*
CMP(.B)	src, dst	src - dst	*	*	*	*
DADD(.B)	src, dst	src + dst +C → dst (décimal)	*	*	*	*
BIT(.B)	src, dst	src .and. dst	0	*	*	$\overline{Z}$
BITC(.B)	src, dst	.not. src .and. dst → dst	-	-	-	-
BITS(.B)	src, dst	src .or. dst → dst	-	-	-	-
XOR(.B)	src, dst	src .xor. dst → dst	*	*	*	$\overline{Z}$
AND(.B)	src, dst	src .and. dst → dst	0	*	*	$\overline{Z}$

Instruction à deux opérandes (Format I)



Opcode				*src*	*Ad*	*B/W*	*As*	*dst*	*Instructions à deux opérandes*
0	1	0	0	src	Ad	B/W	As	dst	MOV : Charge source dans la destination
0	1	0	1	src	Ad	B/W	As	dst	ADD : Addition de source à destination
0	1	1	0	src	Ad	B/W	As	dst	ADDC : Addition de source et report à destination
0	1	1	1	src	Ad	B/W	As	dst	SUBC : Soustraction de source et report à destination
1	0	0	0	src	Ad	B/W	As	dst	SUB : Soustraction de source à destination
1	0	0	1	src	Ad	B/W	As	dst	CMP : Compare source à destination (pas de modif. de dst)
1	0	1	0	src	Ad	B/W	As	dst	DADD : Addition BCD de source à destination
1	0	1	1	src	Ad	B/W	As	dst	BIT : Test bit(s) particulier(s) de destination
1	1	0	0	src	Ad	B/W	As	dst	BIC : Force à 0 bit(s) particulier(s) de destination
1	1	0	1	src	Ad	B/W	As	dst	BIS : Force à 1 bit(s) particulier(s) de destination
1	1	1	0	src	Ad	B/W	As	dst	XOR : OU exclusif de source avec destination
1	1	1	1	src	Ad	B/W	As	dst	AND : ET de source avec destination

Instruction à deux opérandes (Format I)



Exemple :

MOV.W R10,R8

4A08h → Op-code = 4, Rsrc = 10, Ad=0, B/W=0, As=0, Rdst=8

(Mode registre, Mode registre)

MOV.B @R10,5(R8)

4AE8h
0005h → Op-code = 4, Rsrc = 10, Ad=1, B/W=1, As=2, Rdst=8
Source or Destination=5

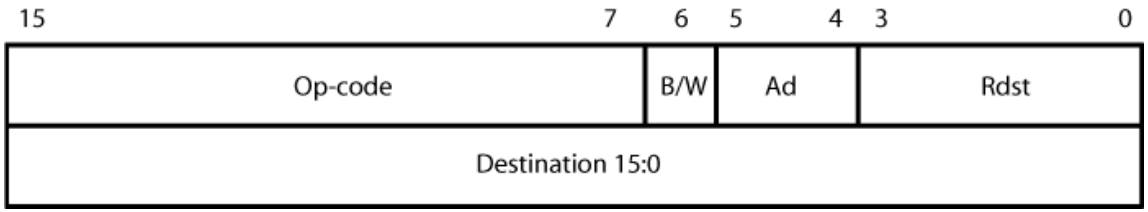
(Mode registre indexé, Mode indirect)

ADD.W #3253,&3110

50B2h
0CB5h
0C26h → Op-code = 5, Rsrc = 0, Ad=1, B/W=0, As=3, Rdst=2
Source or Destination=3253
Destination=3110

(Mode absolu, Mode immédiat)

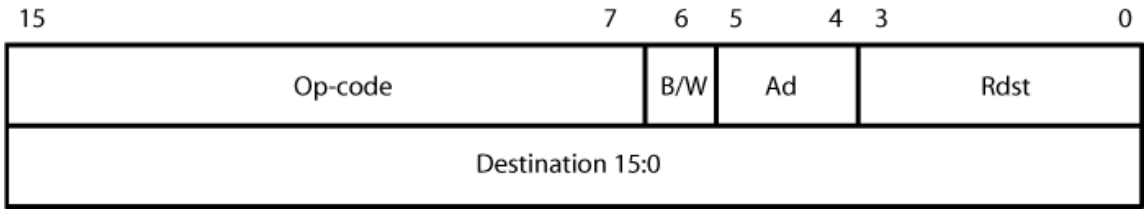
Instruction à une opérande (Format II)



* : le bit d'état est affecté
\- : le bit d'état n'est pas affecté
0 : le bit d'état est mis à0
1 : le bit d'état est mis à 1

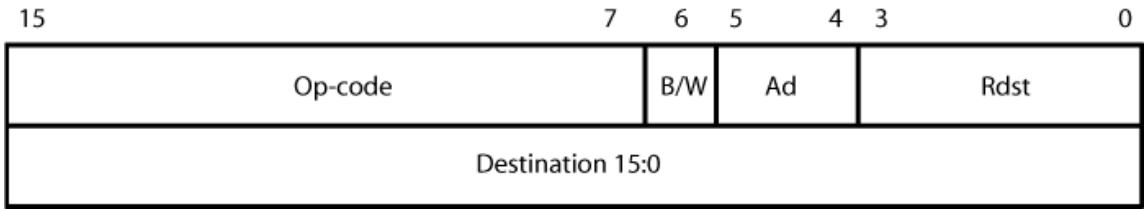
Mnémonique	S-reg, D-reg	Opération	Bits d'état			
			V	N	Z	C
RCC(.B)	dst	C → MSB → … → LSB → C	*	*	*	*
RRA(.B)	dst	MSB → MSB → … → LSB → C	0	*	*	*
PUSH(.B)	src	SP - 2 → SP, src → @SP	-	-	-	-
SWPB(.B)	dst	Croisement de bytes	-	-	-	-
CALL	dst	SP - 2 → SP, PC + 2 → @SP dst - PC	-	-	-	-
RETI		TOS → SR, SP + 2 → SP	*	*	*	*
SXT	dst	Bit7 → Bit8 … Bit15	0	*	*	*

Instruction à une opérande (Format II)



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Instructions
0	0	0	1	0	0	Opcode			B/W	As		registre				Instruction arithmétique à une seule opérande
0	0	0	1	0	0	0	0	0	B/W	As		registre				RRC : Rotation à droite au travers du carry
0	0	0	1	0	0	0	0	1	0	As		registre				SWPB : Croissement de byte (swap)
0	0	0	1	0	0	0	1	0	B/W	As		registre				RRA : Rotation arithmétique à droite
0	0	0	1	0	0	0	1	1	0	As		registre				SXT : Extension du signe d'un byte à un word
0	0	0	1	0	0	1	0	0	B/W	As		registre				PUSH : Sauve une valeur dans la pile
0	0	0	1	0	0	1	0	1	0	As		registre				CALL : Appel de sous programme
0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	RETI : Retour d'une routine d'interruption

Instruction à une opérande (Format II)



Exemple :

CALL #routine

Mode immédiat

12B0h
31A2h
→
Op-code = 25h, B/W = 0, Ad=3, Rdst=0
Destination routine = 31A2h

RRC R8

Mode registre

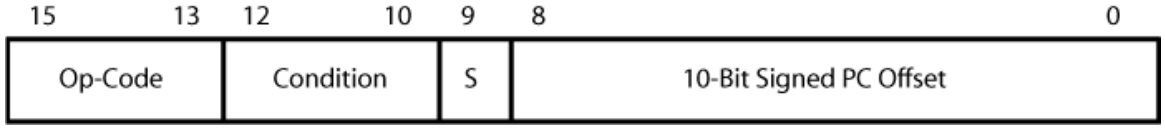
1008h → Op-code = 20h, B/W = 0, Ad=0, Rdst=8

PUSH.W @R5

Mode indirect

1225h → Op-code = 25h, B/W = 0, Ad=2, Rdst=5

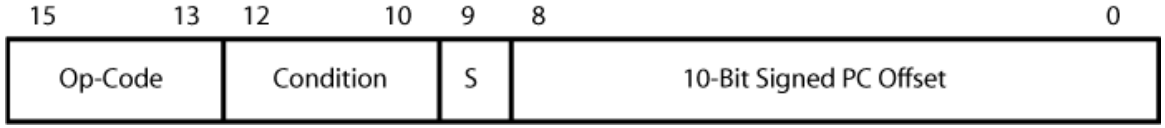
Instruction de saut (Format III)



L'offset signé de 10 bits de l'instruction de saut est multiplié par deux (décalage de 1 bit vers la gauche et le bit de signe est étendu pour avoir un mot de 20 bits. Ceci permet des sauts relatifs dans la plage de -511 à +512 mots autour du compteur de programme (PC) sur les 20 bits de la largeur des adresses sans affecter les bits d'état

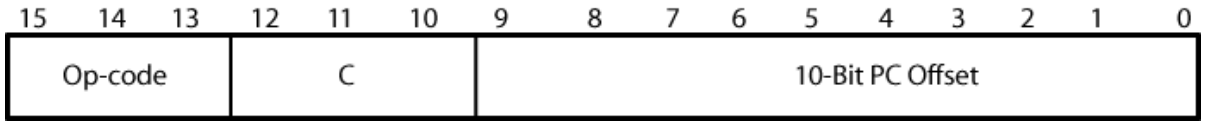
Mnémonique	S-reg, D-reg	Opération
JEQ/JZ	Label	Saut à Label si Z=1
JNE/JNZ	Label	Saut à Label si Z=0
JC	Label	Saut à Label si C=1
JNC	Label	Saut à Label si C=0
JN	Label	Saut à Label si N=1
JGE	Label	Saut à Label si (N .xor. V)=0
JL	Label	Saut à Label si (N .xor. V)=1
JMP	Label	Saut inconditionnel à Label

Instruction de saut (Format III)



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	*Instructions*
0	0	1	condition			offset (10 bits signé)										Saut conditionnel ; PC=PC +2·offset
0	0	1	0	0	0	offset (10 bits signé)										JNE / JNZ : saut si Z = 0
0	0	1	0	0	1	offset (10 bits signé)										JEQ / JZ : saut si Z = 1
0	0	1	0	1	0	offset (10 bits signé)										JNC : saut si C = 0
0	0	1	0	1	1	offset (10 bits signé)										JC / JHS : saut si C = 1
0	0	1	1	0	0	offset (10 bits signé)										JN : saut si N=1
0	0	1	1	0	1	offset (10 bits signé)										JGE : saut si (N .xor. V) =0
0	0	1	1	1	0	offset (10 bits signé)										JL : saut si (N .xor. V) =1
0	0	1	1	1	1	offset (10 bits signé)										JMP : saut sans condition

Instruction de saut (Format III)



Exemple :

314Eh : JMP LABEL 3C2Dh → Op-code = 1, C = 7, S=0, PC Offset=2D

31A8h 314Eh+2*2D=31A8h

314Eh : JZ LABEL 242Dh → Op-code = 1, C = 1, S=0, PC Offset=2D

314Eh : JNC LABEL 282Dh → Op-code = 1, C = 2, S=0, PC Offset=2D

Instructions émulées

Mnémonique	Opération		Bits d'état V	N	Z	C
ADC(.B) dst	Addition du report avec l'opérande	ADDC(.B) #0,dst	*	*	*	*
BR dst	Branchement indirect	MOV dst,PC	-	-	-	-
CLR(.B) dst	Mise à zéro de l'opérande	MOV(.B) #0,dst	-	-	-	-
CLRC	Mise à zéro du bit d'état C	BIC #1,SR	-	-	-	0
CLRN	Mise à zéro du bit d'état N	BIC #4,SR	-	0	-	-
CLRZ	Mise à zéro du bit d'état Z	BIC #2,SR	-	-	0	-
DADC(.B) dst	Addition du report pour une addition BCD	DADD(.B) #0,dst	*	*	*	*
DEC(.B) dst	Décrémentation par 1 de l'opérande	SUB(.B) #1,dst	*	*	*	*
DECD(.B) dst	Décrémentation par 2 de l'opérande	SUB(.B) #2,dst	*	*	*	*
DINT	Désactivation des interruptions	BIC #8,SR	-	-	-	-
EINT	Activation des interruptions	BIS #8,SR	-	-	-	-
INC(.B) dst	Incrémentation par 1 de l'opérande	ADD(.B) #1,dst	*	*	*	*

Instructions émulées

Mnémonique	Opération		Bits d'état V	N	Z	C
INC(.B) dst	Incrémentation par 1 de l'opérande	ADD(.B) #1,dst	*	*	*	*
INCD(.B) dst	Incrémentation par 2 de l'opérande	ADD(.B) #2,dst	*	*	*	*
INV(.B) dst	Inversion bit à bit de l'opérande	XOR(.B) #-1,dst	*	*	*	*
NOP	Pas d'opération	MOV R3,R3	-	-	-	-
POP dst	Extraction de l'opérande de la pile	MOV @SP+,dst	-	-	-	-
RET	Retour d'une sous routine	MOV @SP+,PC	-	-	-	-
RLA(.B) dst	Décalage arithmétique à gauche	ADD(.B) dst,dst	*	*	*	*
RLC(.B) dst	Décalage logique à gauche au travers de C	ADDC(.B) dst,dst	*	*	*	*
SBC(.B) dst	Soustraction du report avec l'opérande	SUBC(.B) #0,dst	*	*	*	*
SETC	Mise à un du bit d'état C	BIS #1,SR	-	-	-	1
SETN	Mise à un du bit d'état N	BIS #4SR	-	1	-	-
SETZ	Mise à un du bit d'état Z	BIS #2,SR	-	-	1	-
TST(.B) dst	Comparaison de l'opérande avec zéro	CMP(.B) #0,dst	0	*	*	1

Exemple d'instruction émulée

Instruction de format I (deux opérandes)

CLR dst → MOV R3,dst ← *exemple* → CLR R5 → MOV #0,R5

Opcode				*src*				*Ad*	*B/W*	*As*		*dst*			
0	1	0	0	0	0	1	1	0	0	0	0	0	1	0	1

→ code machine 4305h

4 — 3 — 5

Mode immédiat (utilisation des registres de constante)

16 bits

Mode registre

INC dst → ADD 0(R3),dst ← *exemple* → INC R5 → ADD #1,R5

Opcode				*src*				*Ad*	*B/W*	*As*		*dst*			
0	1	0	1	0	0	1	1	0	0	0	1	0	1	0	1

→ code machine 5315h

5 — 3 — 5

Mode immédiat (utilisation des registres de constante)

16 bits

Mode registre

Temps d'exécution et longueur des instructions (Format I)

Mode d'adressage src	Mode d'adressage dst	Nb de cycles	Longueur de l'instruction	Exemple	
Rn	Rm	1	1	MOV	R5,R8
	PC	1	1	BR	R9
	X(Rm)	4	2	ADD	R5,4(R6)
	MEM	4	2	XOR	R8,MEM
	&MEM	4	2	MOV	R5,&MEM
@Rn	Rm	2	1	AND	@R4,R5
	PC	2	1	BR	@R8
	X(Rm)	5	2	XOR	@R5,8(R6)
	MEM	5	2	MOV	@R5,MEM
	&MEM	5	2	XOR	@R5,&MEM
@Rn+	Rm	2	1	ADD	@R5+,R6
	PC	3	1	BR	@R9+
	X(Rm)	5	2	XOR	@R5,8(R6)
	MEM	5	2	MOV	@R9+,MEM
	&MEM	5	2	MOV	@R9,&MEM
x(Rn)	Rm	3	2	MOV	2(R5),R7
	PC	3	2	BR	2(R6)
	X(Rm)	6	3	MOV	4(R7),MEM
	MEM	6	3	ADD	4(R4),6(R9)
	&MEM	6	3	MOV	2(R4),&MEM

Temps d'exécution et longueur des instructions (Format I)

Mode d'adressage src	Mode d'adressage dst	Nb de cycles	Longueur de l'instruction	Exemple	
#N	Rm	2	2	MOV	#20,R9
	PC	3	2	BR	#2AEh
	X(Rm)	5	3	MOV	#0300h,0(SP)
	MEM	5	3	ADD	#33,MEM
	&MEM	5	3	ADD	#33,&MEM
MEM	Rm	3	2	AND	MEM,R6
	PC	3	2	BR	MEM
	X(Rm)	6	3	CMP	MEM,MEM1
	MEM	6	3	MOV	MEM,0(SP)
	&MEM	6	3	MOV	MEM,&MEM1
&MEM	Rm	3	2	MOV	&MEM,R8
	PC	3	2	BRA	&MEM
	X(Rm)	6	3	MOV	&MEM,MEM1
	MEM	6	3	MOV	&MEM,0(SP)
	&MEM	6	3	MOV	&MEM,&MEM1

Temps d'exécution et longueur des instructions (Format II)

Mode d'adressage	Nb de cycles			Longueur de l'instruction	Exemple
	RRA, RRC, SWPB, SXT	PUSH	CALL		
Rn	1	3	3	1	SWPB R5
@Rn	3	3	4	1	RRC @R9
@Rn+	3	3	4	1	SWPB @R10+
#N	n.a.	3	4	2	CALL #LABEL
x(Rn)	4	4	4	2	CALL 2(R7)
MEM	4	4	4	2	PUSH MEM
&MEM	4	4	4	2	STX &MEM

Temps d'exécution et longueur des instructions (Format III)

Toutes les instructions de saut exigent une instruction dont la longueur est de un mot et dont le temps d'exécution sera de deux cycles d'horloge MCLK

Temps d'exécution et longueur des instructions interruption, reset et appel de sous routine

Action	Temps d'exécution Nb de cycles MCLK	Longueur des instructions (mot de 16 bits)
Retour d'interruption RETI	3	1
Retour de sous-routine RET	3	1
Demande d'interruption	5	-
Reset du watch dog	4	-
Reset (RST/NMI)	4	-

Instructions en mode étendu

Pour les microcontrôleurs ayant plus de 64kB de mémoire, au maximum 1MB, il existe des instructions étendues.
Le but de ce cours n'est pas d'entrer dans ce degré de détails

Pour plus d'information voir : MSP430x4xx Family User's Guide (SLAU056G.pdf)

Directives assembleur

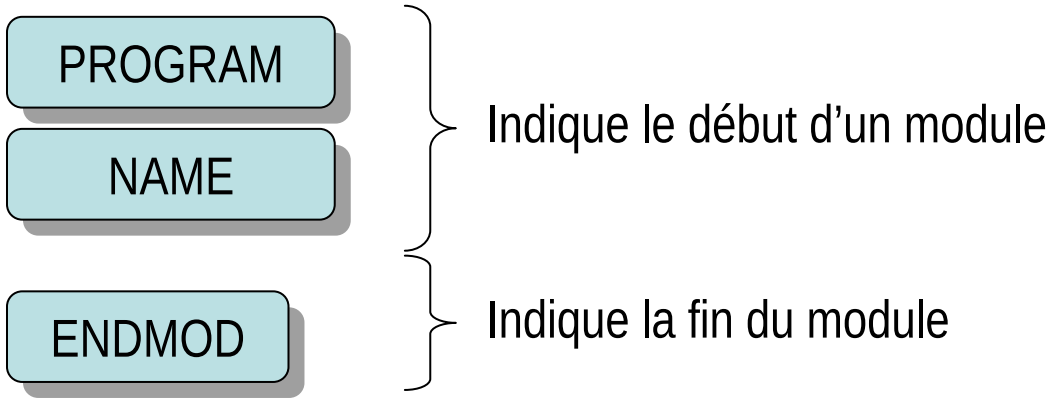
Mémoire à disposition

La mémoire associée à un microcontrôleur est de plusieurs technologies

ROM	Mémoire permanente non modifiable par l'utilisateur
FLASH, EEPROM	Mémoire permanente modifiable par l'utilisateur
RAM	Mémoire volatile modifiable par l'utilisateur

Décomposition du programme en module

Un programme peut être décomposé en plusieurs modules pour en faciliter la compréhension et les modifications

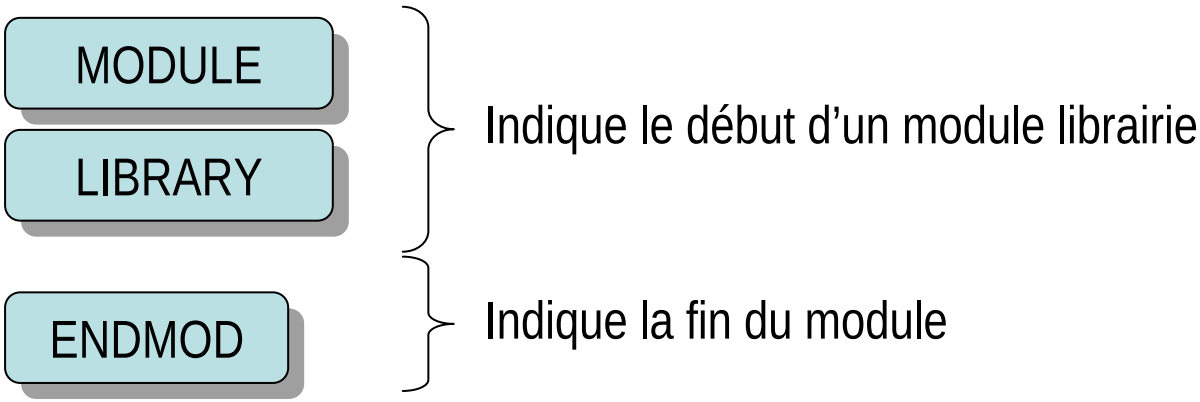


Un module PROGRAM ou NAME est automatiquement lié aux autres modules par l'éditeur de lien, même s'il n'a pas de lien avec les autres modules

Au début d'un nouveau module tous les symboles d'utilisateur sont supprimés, exceptés ceux créés près DEFINE, #define, ou MACRO, le compteur local de programme (PLC) est mis à zéro et le segment est mis en absolu par défaut (ASEG)

Décomposition en programme et en module

Les modules de programme peut être utilisé comme librairie. Dans ce cas, c'est la directive suivante qui doit être utilisée :



Un module MODULE ou LIBRARY n'est lié aux autres modules que s'il existe une référence de type PUBLIC et EXTERN entre eux. Les modules MODULE ou LIBRARY permettent donc de gagner de la place ne mémoire

Au début d'un nouveau module tous les symboles d'utilisateur sont supprimés, exceptés ceux créés par DEFINE, #define, ou MACRO, le compteur local de programme (PLC) sont mis à zéro et le segment est mis en absolu par défaut (ASEG)

Décomposition en programme et en module

La fin d'un fichier contenant un ou plusieurs modules de programme doit se terminer par la directive suivante :

END } Indique la fin du module

Tout ce qui suit une directive END est ignoré par l'assembleur

Décomposition en programme et en module

Fichier main.s43

```
#include "msp430xG46x.h“

        PROGRAM MAIN
        EXTERN ADC12_ISR
;-------------------------------------------------------------------------------
        RSEG    CSTACK                  ; Define stack segment
;-------------------------------------------------------------------------------
        RSEG    CSTART                  ; Assemble to Flash memory
;-------------------------------------------------------------------------------
RESET       MOV.W   #SFE(CSTACK),SP     ; Initialize stackpointer
            …
            …


;-------------------------------------------------------------------------------
        COMMON  INTVEC                  ; Interrupt Vectors
;-------------------------------------------------------------------------------
        ORG     ADC12_VECTOR            ; ADC12 Vector
        DW      ADC12_ISR
        ORG     RESET_VECTOR            ; POR, ext. Reset
        DW      RESET
        END
```

Fichier ADC_interrupt.s43

```
#include "msp430xG46x.h"

        LIBRARY  INTERRUPTION
        PUBLIC ADC12_ISR

A0result    EQU     01100h              ; Channel A0 results
A1result    EQU     01110h              ; Channel A1 results

;-------------------------------------------------------------------------------
        RSEG ISR_CODE                   ; Define ISR_CODE segment
;-------------------------------------------------------------------------------
ADC12_ISR ; Interrupt Service Routine for ADC12
;-------------------------------------------------------------------------------
        MOV.W   &ADC12MEM0,A0result(R5) ; Move results to RAM
        MOV.W   &ADC12MEM1,A1result(R5) ; Move results to RAM
        …
        …

        RTI
        END
```

Décomposition en programme et en module

Main.s43

```
PROGRAM main


END
```

Programme principal et
table des vecteurs d'interruption

lib1.s43
lib2.s43

```
LIBRARY funct1
ENDMOD
LIBRARY funct2
ENDMOD
…
END
```

Librairies de sous routine et macros

interA.s43
interB.s43

```
LIBRARY inter1
ENDMOD
LIBRARY inter2
ENDMOD
…
END
```

Routines d'interruption

Ne sera chargé en mémoire que les modules référencés

↓

Gain de place mémoire

↓

Mapping de la mémoire

Le choix de l'emplacement du programme et des données dépend de leur type

FLASH, EEPROM — Code du programme, table des vecteurs d'interruption, informations permanentes, constantes

RAM — Données susceptibles d'être modifiées en cours d'exécution

Segmentation de la mémoire

Définition de la plage mémoire

La plage complète de la mémoire est décomposée en segment principaux

CODE Mémoire permanente FLASH ou EEPROM

CONST Mémoire permanente FLASH ou EEPROM

DATA Mémoire volatile RAM

Segmentation de la mémoire exemple pour le MSP430FG4617

Type de segment

Chaque segment peut être décomposé en plusieurs types

CODE

- INFO : Information sur l'application
- CSTART : Code de démarrage de l'application
- CODE : Code du programme
- ISR_CODE : Code des sous routines d'interruption
- INTVEC : Table des vecteurs d'interruption
- RESET : Vecteur d'interruption RESET

Voir fichier XLINK de commande pour l'éditeur de lien : -f lnk430fg4617.xlc

Segmentation de la mémoire exemple pour le MSP430FG4617

Type de segment

Chaque segment peut être décomposé en plusieurs types

CONST

- DATA16_C : Constantes sur 16 bits ou chaîne de caractères
- DATA16_ID : Constantes sur 16 bits pour l'initialisation de constante de type DATA16_I
- DATA20_C : Constantes sur 20 bits ou chaîne de caractères
- DATA20_ID : Constantes sur 20 bits pour l'initialisation de constante de type DATA20_I

Voir fichier XLINK de commande pour l'éditeur de lien : -f lnk430fg4617.xlc

Segmentation de la mémoire exemple pour le MSP430FG4617

Type de segment

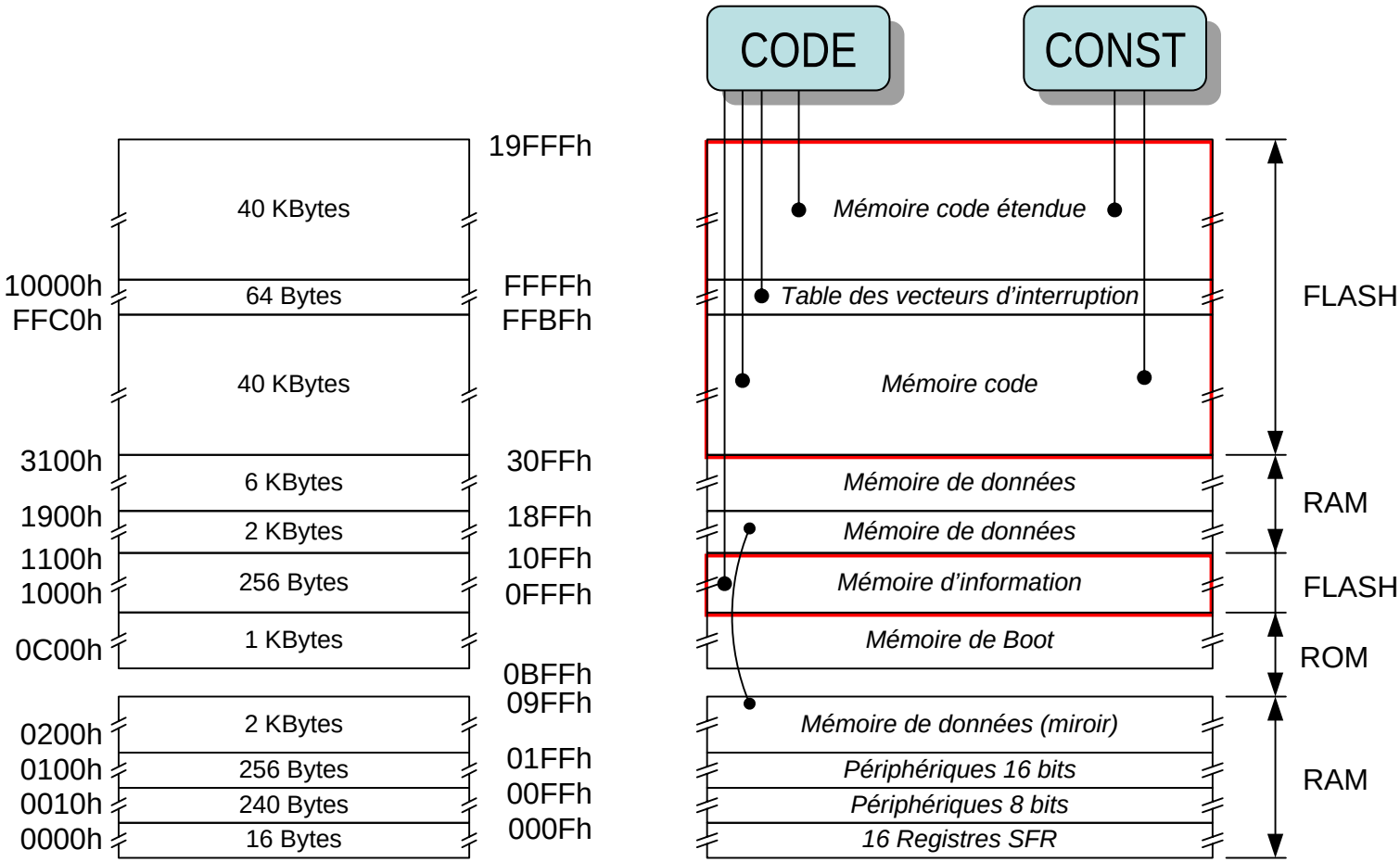
Chaque segment peut être décomposé en plusieurs types

DATA

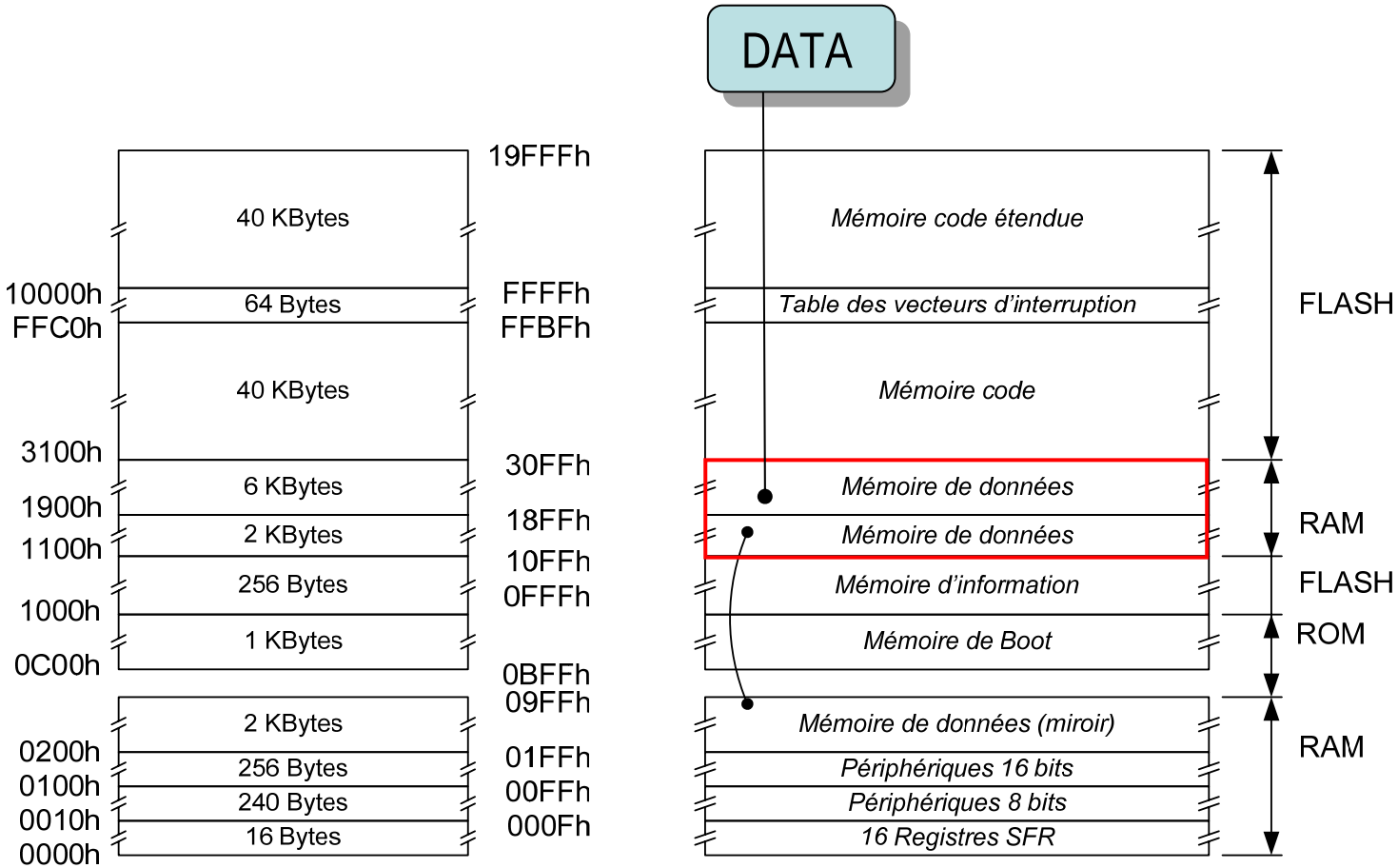
- DATA16_I : données 16 bits initialisée
- DATA16_Z : données 16 bits initialisée à zéro
- DATA16_N : données 16 bits non initialisée
- DATA20_I : données 20 bits initialisée
- DATA20_Z : données 20 bits initialisée à zéro
- DATA20_N : données 20 bits non initialisée
- CSTACK : segment de mémoire réservé à la pile

Voir fichier XLINK de commande pour l'éditeur de lien : -f lnk430fg4617.xlc

Segmentation de la mémoire exemple pour le MSP430FG4617



Segmentation de la mémoire exemple pour le MSP430FG4617



Segmentation de la mémoire exemple pour le MSP430FG4617

```
;-------------------------------------------------------------------------------
        RSEG    CSTACK          ; Define stack segment
;-------------------------------------------------------------------------------
        RSEG    CSTART          ; Assemble to Flash memory
;-------------------------------------------------------------------------------
RESET       MOV.W   #SFE(CSTACK),SP                 ; Initialize stackpointer
StopWDT     MOV.W   #WDTPW+WDTHOLD,&WDTCTL          ; Stop watchdog
            BIS.B   #0FH,&P6SEL                     ; Enable A/D inputs
…


;-------------------------------------------------------------------------------
        RSEG    ISR_CODE        ; Define ISR_CODE segment
;-------------------------------------------------------------------------------
ADC12_ISR   ; Interrupt Service Routine for ADC12
;-------------------------------------------------------------------------------
            MOV.W   &ADC12MEM0,A0result(R5)     ; Move results to RAM
…
```

Segment DATA : [1100h - 30FFh]

Segment CODE : [3100h - FFFFh]

Segment CODE : [3100h - FFFFh]

Directive de contrôle des types de segment : CODE, CONST, DATA

RSEG RSEG *type de segment* ; *CSTART, CSTACK, CODE, DATA16_I, …*

Cette directive définit un segment réadressable (relocatable segment)

→ *permet de placer plusieurs modules dans un même type de segment*

```
PROGRAM MAIN
        EXTERN  ADC12_ISR                    Pile en RAM
;-------------------------------------------------------------------------------
        RSEG    CSTACK                  ; Define stack segment
;-------------------------------------------------------------------------------
        RSEG    CSTART                  ; Assemble to Flash memory
;-------------------------------------------------------------------------------
RESET       MOV.W   #SFE(CSTACK),SP                 Code en FLASH
StopWDT     MOV.W #WDTPW+WDTHOLD,&WDTCTL
…
```

Fichier de lien : lnk430fg4617.xcl

```
…
-Z(DATA)CSTACK+_STACK_SIZE#
…
-Z(CODE)CSTART,ISR_CODE=3100-FFBF
…
```

Directive de contrôle des types de segment : CODE, CONST, DATA

RSEG

Proposition de l'utilisation de RSEG
et des types de segment CSTART, CODE, CODE_ISR, CSTACK

Fichier main.s43

```
            RSEG   CSTACK                  ; define stack segment
            RSEG   CSTART                  ; segment from lower CODE segment address
main:       mov.w  #SFE(CSTACK),SP         ; initialisation du stack
            …
```

Fichiers name_subroutine.s43

```
            RSEG   CODE                    ; subroutine
function:   …
```

Fichiers name_ISR.s43

```
            RSEG   ISR_CODE                ; interrupt service routine
Nom_ISR: …
```

Directive de contrôle des types de segment : CODE, CONST, DATA

ASEG

ASEG *adresse* ; *adresse de départ du segment, même effet que la directive ORG placée au début d'un segment*

Cette directive défini un segment absolu

→ *permet de définir une plage d'adresse pour un segment dans toute la plage mémoire*

```
NAME reset
        EXTERN main
;-------------------------------------------------------------------------
        ASEG 0xFFFE         ; Define absolute segment
;-------------------------------------------------------------------------
reset:  DC16 main           ; Instruction that executes on startup
        END
```

L'adresse 0FFFEh correspond à la position du vecteur d'interruption RESET

Adresse de démarrage du programme principal

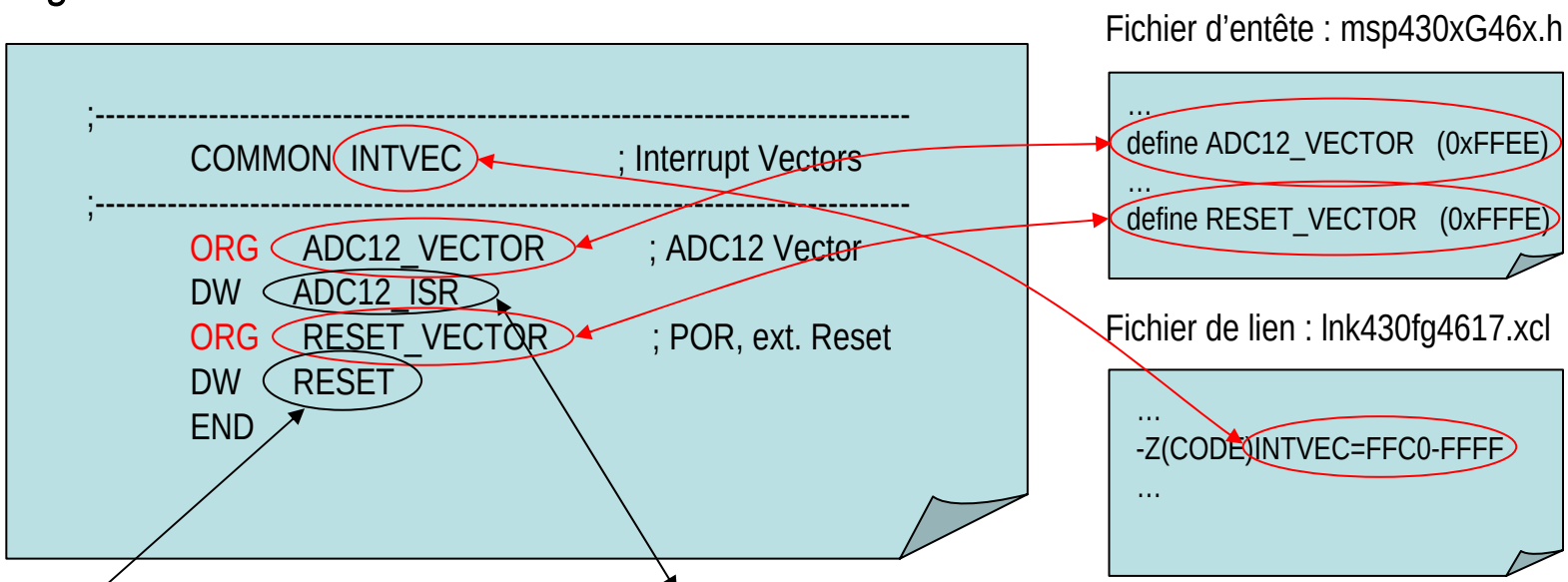
Directive de contrôle des types de segment : CODE, CONST, DATA

ORG

ORG expr ; *valeur à assigné au compteur local de programme (PLC)*

Cette directive force une valeur au compteur local de programme PLC

→ *permet de placer des instructions, ou des données à un endroit précis du segment en court*



Directive de contrôle des types de segment : CODE, CONST, DATA

ORG

ORG expr ; *valeur à assigné au compteur local de programme (PLC)*

Cas particulier : incrémentation du compteur local de programme

```
;-------------------------------------------------------------------------
        RSEG  CODE          ; début d'un segment de code
;-------------------------------------------------------------------------
        MOV.W    R7,R8      ; value moved from R7 to R8
        …
        ORG    $+10h        ; PLC est incrémenté de 16
        ADD.B    #024,0(R7)
        …
```

Directive de contrôle des types de segment : CODE, CONST, DATA

COMMON COMMON *type de segment* ; INTVEC, *DATA16_I, DATA16_Z, …, DATA16_C*

Cette directive permet à tous les segments de mêmes nom de se superposer

```
;-------------------------------------------------------------------------
        COMMON  INTVEC              ; Interrupt Vectors
;-------------------------------------------------------------------------
        ORG   ADC12_VECTOR          ; ADC12 Vector
        DW    ADC12_ISR
        ORG   RESET_VECTOR          ; POR, ext. Reset
        DW    RESET
```

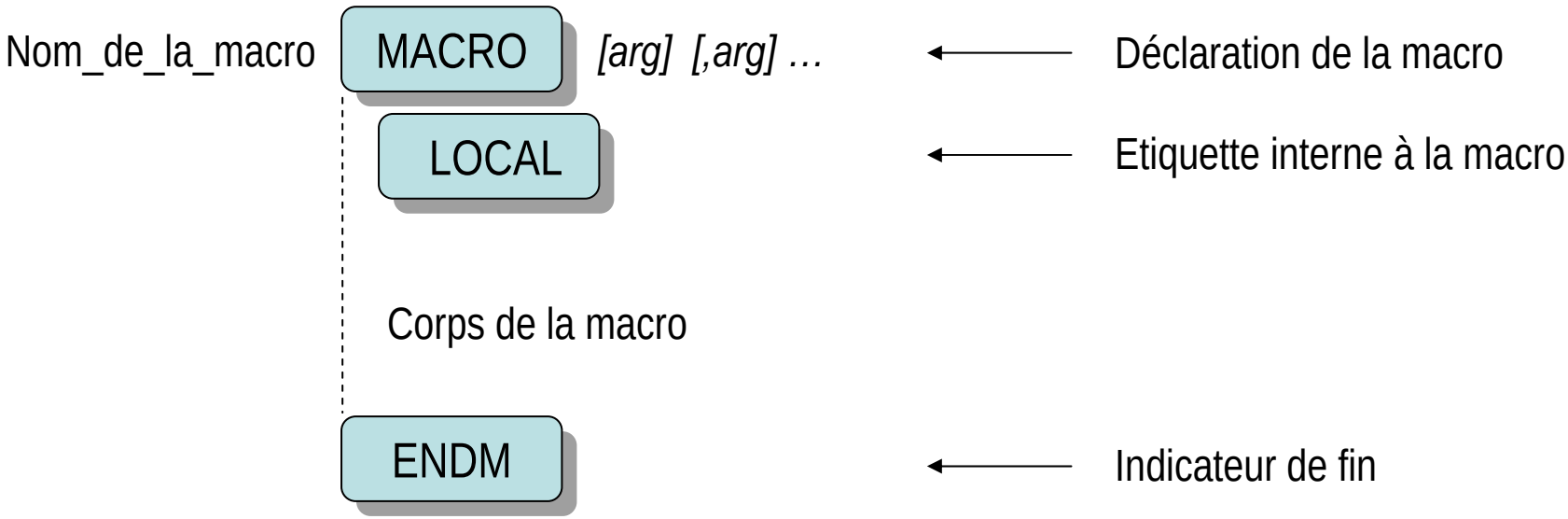
```
;-------------------------------------------------------------------------
        COMMON  INTVEC              ; Interrupt Vectors
;-------------------------------------------------------------------------
        ORG   PORT2_VECTOR          ; PORT2 Vector
        DW    PORT2_ISR
        ORG   DAC12_VECTOR          ; DAC16 Vector
        DW    DAC12_ISR
```

```
/************************************************************
* Interrupt Vectors
************************************************************/
#define DAC12_VECTOR            (0xFFDC) /* DAC 12 */
#define DMA_VECTOR              (0xFFDE) /* DMA */
#define BASICTIMER_VECTOR       (0xFFE0) /* Basic Timer / RTC */
#define PORT2_VECTOR            (0xFFE2) /* Port 2 */
#define USART1TX_VECTOR         (0xFFE4) /* USART 1 Transmit */
#define USART1RX_VECTOR         (0xFFE6) /* USART 1 Receive */
#define PORT1_VECTOR            (0xFFE8) /* Port 1 */
#define TIMERA1_VECTOR          (0xFFEA) /* Timer A CC1-2, TA */
#define TIMERA0_VECTOR          (0xFFEC) /* Timer A CC0 */
#define ADC12_VECTOR            (0xFFEE) /* ADC */
#define USCIAB0TX_VECTOR        (0xFFF0) /* USCI A0/B0 Transmit */
#define USCIAB0RX_VECTOR        (0xFFF2) /* USCI A0/B0 Receive */
#define WDT_VECTOR              (0xFFF4) /* Watchdog Timer */
#define COMPARATORA_VECTOR      (0xFFF6) /* Comparator A */
#define TIMERB1_VECTOR          (0xFFF8) /* Timer B CC1-2, TB */
#define TIMERB0_VECTOR          (0xFFFA) /* Timer B CC0 */
#define NMI_VECTOR              (0xFFFC) /* Non-maskable */
#define RESET_VECTOR            (0xFFFE) /* Reset [Highest Priority] */
```

Directive de définition des macros

Une macro est une instruction, définie par l'utilisateur, qui peut regrouper plusieurs instructions assembleur

Lors de l'assemblage, l'instruction utilisateur est replacée par le groupe d'instructions assembleur qui la compose



Nom_de_la_macro est le nom donné par l'utilisateur à la macro, *arg* correpond à un argument passé par paramètre

Directive de définition des macros

Définition d'une macro "Addition"

```
Addition    MACRO operande1, operande2, resultat
            LOCAL Addition1, Addition2
            MOV.W  operande1,resultat
            ADD.W  operande2,resultat
            JN Addition1
            BIT.W  #0100h,SR
            JZ    Addition2
            MOV.W  #8000h,resultat
            JMP    Addition2
Addition1:
            BIT.W  #0100h,SR
            JZ    Addition2
            MOV.W  #7FFFh,resultat
Addition2:
            ENDM
```

Utilisation de la macro "Addition"

```
            …
            Addition R5, R6, R7
            …
```

```
            MOV.W  R5,R7
            ADD.W  R6,R7
            JN Addition1
            BIT.W  #0100h,SR
            JZ    Addition2
            MOV.W  #8000h,R7
            JMP    Addition2
Addition1:
            BIT.W  #0100h,SR
            JZ    Addition2
            MOV.W  #7FFFh,R7
Addition2:
```

Directive de définition des macros

La macro doit être définie avant son utilisation

Par exemple dans un fichier d'entête : *MacroLib.h*

Fichier contenant un ou plusieurs module
par exemple : *main.s43*

```
Nom_macro   MACRO param1, param2, …
            LOCAL  Label1, Label2
            …
            …
            ENDM
```

```
#include "msp430.h"
#include “MacroLib.h”

    PROGRAM    main     ; module name
    …
    RSEG  CODE

init:       MOV    #SFE(CSTACK), SP
main:       NOP
            …
            Nom_macro R14,R12
            …
            …
            END
```

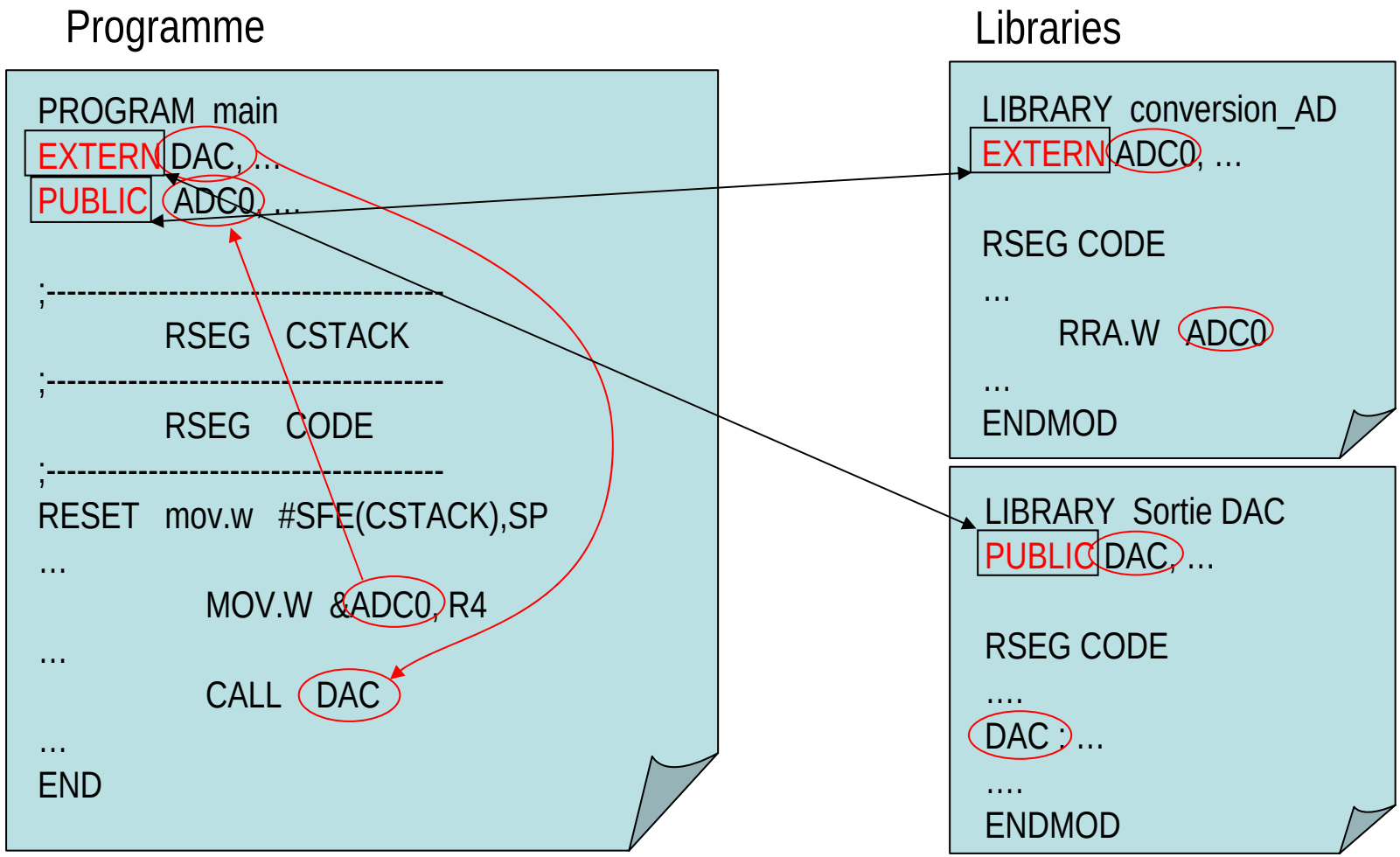
Directive de contrôle des symboles

Ces directives contrôlent comment les symboles sont partagés entre les modules

EXTERN	EXTERN symbole	Indique que la déclaration du symbole est externe au module
PUBLIC	PUBLIC symbole	Indique que la déclaration du symbole est accessible pour les autres modules

Directive de contrôle des symboles

Ces directives contrôlent comment les symboles sont partagés entre divers modules



Directives d'assignation d'une valeur à un symbole

=	Nom = valeur	Assigne une valeur à un symbole (portée locale). Extension à d'autres modules par PUBLIC et EXTERN.
EQU	Nom EQU valeur	
ALIAS	Nom ALIAS valeur	
DEFINE	Nom DEFINE valeur	Assigne une valeur à un symbole (portée au fichier) Extension à d'autres fichiers par PUBLIC et EXTERN
LIMIT	Déclaration d'une constante de 8 bits	
ASSIGN	Nom ASSIGN valeur	Assigne une valeur à un symbole (portée locale) Le même symbole peut être assigné plusieurs fois Pas d'extension possible à d'autres modules
VAR	Nom VAR valeur	

Directives d'assignation d'une nom à un registre spécifique

SFRB

[const] SFRB *Registre* = *adresse*

SFRB ADC12MCTL0 = (0x0080) : défini un nom (ADC12CTL0) à un **registre de contrôle de 8 bits** du convertisseur ADC12. Ce registre est **accessible en écriture et en lecture**

const SFRB ADC12MCTL0 = (0x0080) : même définition mais **accessible qu'en lecture**.

SFRW

[const] SFRW *Registre* = *adresse*

SFRW ADC12MEM0 = (0x0080) : défini un nom (ADC12MEM0) à un **registre de de données de 16 bits** du convertisseur ADC12. Ce registre est **accessible en écriture et en lecture**

const SFRW ADC12MEM0 = (0x0080) : même définition mais **accessible qu'en lecture**.

Directives d'assignation d'une nom à un registre spécifique

SFRTYPE

[const] SFRTYPE *Registre attribut* [,*attribut*] = *adresse*

READ	: en lecture
WRITE	: en écriture
BYTE	: contenu sur 8 bits
WORD	: contenu sur 16 bits

SFRTYPE P1DIR WRITE BYTE = (0x0022) : défini un nom (P1DIR) au **registre de contrôle de direction (8 bits)** du port P1.
Ce registre est **accessible en écriture seulement**

Directives d'allocation et de définition des données

Ces directives définissent des constantes

DC8	(DB)	Déclaration d'une constante de 8 bits
DC16	(DW)	Déclaration d'une constante de 16 bits
DC32	(DL)	Déclaration d'une constante de 32 bits
DC64		Déclaration d'une constante de 64 bits
DF32	(DF)	Déclaration d'une constante virgule flottante de 32 bits
DF64		Déclaration d'une constante virgule flottante de 64 bits
.double		Déclaration d'une constante virgule flottante de 64 bits (format TI)

Directives d'allocation et de définition des données

Ces directives permettent d'allouer de l'espaces mémoire

DS8	(DS)	Allocation mémoire pour des variables 8 bits
DS16	(DS 2)	Allocation mémoire pour des variables 16 bits
DS32	(DS 4)	Allocation mémoire pour des variables 32 bits
DS64	(DS 8)	Allocation mémoire pour des variables 64 bits
.float		Allocation de mémoire pour une variable de type 32 bits flottant (format TI)

Directives d'allocation et de définition des données

Les directives d'allocation et de définition des données sont corroborées à des segments mémoire particuliers

Taille	Type de segment mémoire	
	DATA16.C : CONST	DATA16.N : DATA
Entier 8 bits	DC8, DB	DS8, DS
Entier 16 bits	DC16, DW	DS16, DS 2
Entier 32 bits	DC32, DL	DS32, DS 4
Entier 64 bits	DC64	DS 64 DS 8
Flottant 32 bits	DC32, DF	DS32
Flottant 64 bits	DF64	DS64

Directives d'allocation et de définition des données

Exemple : déclaration de tableaux

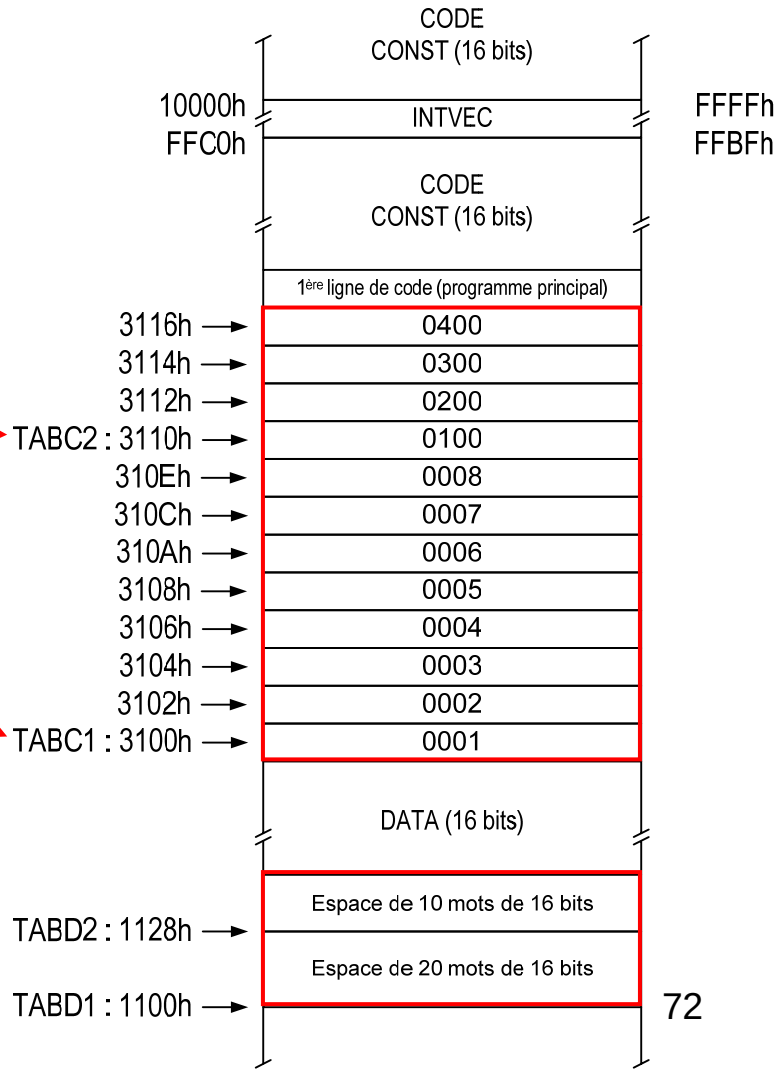
```
//---- Déclaration de tableaux de constantes ----//

        RSEG  DATA16_C
TABC1   DC16  0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8
TABC2   DC16  0x100, 0x200, 0x300, 0x400

//------- Déclaration des tableaux de variables -------//

        RSEG  DATA16_N
TABD1   DS16   20
TABD2   DS16   10
```

Allocation dans la mémoire



Directives d'allocation et de définition des données

Exemple : déclaration d'une « lookup table

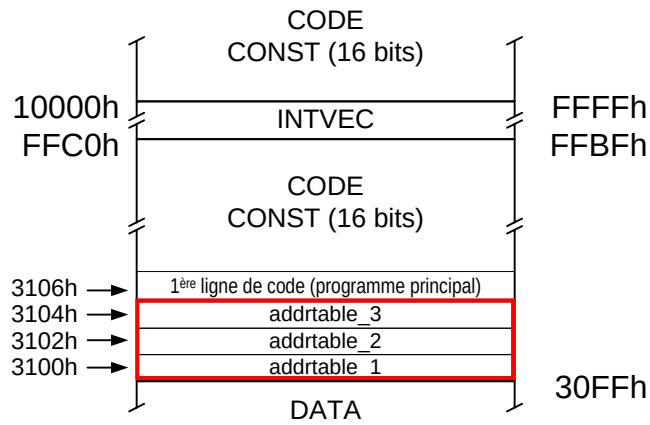
```
PROGRAM  table

RSEG     DATA16_C

table          DW  addrtable_1, addrtable_2, addrtable_3

RSEG           CODE
               …
addrtable_1:
               MOV.W       R5,R8
               RET
addrrable_2:
               ADD.W       @R7,0(R8)
               RET
addrtable_3:
               RRA         R6
               RET
END
```

Allocation dans la mémoire



Directives d'allocation et de définition des données

Définition de constantes

```
// Définition de chaînes de caractères
    message : DC8 'Please enter your name'

// Définition de chaînes de caractères avec caractère de fin de chaîne
    message : DC8 "Please enter your name"
```

Directives issue des directive standard du langage C

- #define : assigne une valeur ou un type à un nom
- #if : assemble les instructions qui suivent si la condition est vraie
- #else : assemble les instructions qui suivent si la condition est fausse
- #elif : introduit une nouvelle condition dans un bloc #if … #endif
- #ifdef : assemble les instructions si le symbole qui suit est défini
- #ifndef : assemble les instructions si le symbole qui suit n'est pas défini
- #endif : marque la fin d'un bloc #if, #ifdef ou #ifndef
- #undef : rend un symbole comme non défini

Directives issue des directive standard du langage C

#include : insère le fichier dont le nom est donné à la suite de la directive

#error : génére un erreur

#message : génére un message sur l'affichage standard

Directives issue des directives standard du langage C

Exemple dans fichier d'entête mps430xG46x.h

```
#ifdef __IAR_SYSTEMS_ASM__

        #define DEFC(name, address) sfrb name = address;
        #define DEFW(name, address) sfrw name = address;

#endif /* __IAR_SYSTEMS_ASM__*/

        :

#define ADC12MEM0_          (0x0140)
DEFW(   ADC12MEM0, ADC12MEM0_)

        :
```

Variable indiquant que l'assembleur est actif

Définition de macros

Fin du bloc

Equivalent à :
DEFW ADC12MEM0 = (0x0140)