Université Blaise Pascal
CLERMONT-FERRAND

Architecture des systèmes à processeurs – IUT GEII (ISI-II2) -2

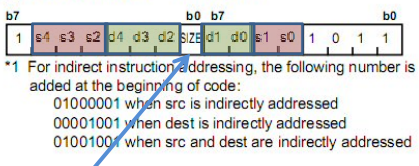
Christophe BLANC
www.christophe-blanc.info
IUT de Montluçon
Département Génie Electrique et Informatique Industrielle

Architecture des systèmes à processeurs

MODÈLE DE PROGRAMMATION D'UN PROCESSEUR

Codage d'une instruction M32C87 Renesas
MOV.size:G Source, Destination

(7) MOV.size:G src, dest



*1 For indirect instruction addressing, the following number is added at the beginning of code:
01000001 when src is indirectly addressed
00001001 when dest is indirectly addressed
01001001 when src and dest are indirectly addressed

C 9 8 B

1 1 0 0 1 0 0 1 1 0 0 0 1 0 1 1

.size	SIZE
.B	0
.W	1

src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0	src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

MOV.W:G R2, R0 ; Transférer contenu de R2 dans R0

Exécution d'un programme élémentaire (notation symbolique)

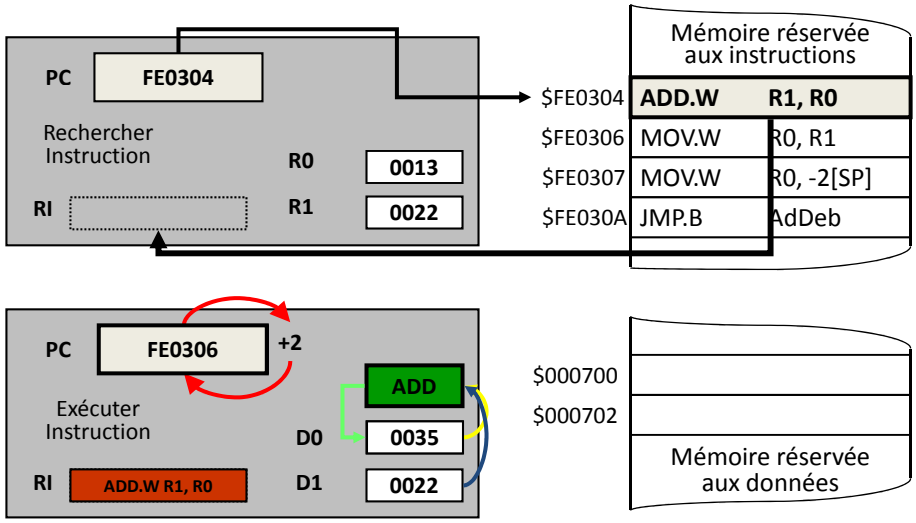
Adresse	Programme Source Assembleur			
	Étiquette	Opération	Opérande	Commentaire
$FE0304	AdDeb	ADD.W	R1, R0	; R0 <- R0 + R1
$FE0306		MOV.W	R0, R1	; R1 <- R0
$FE0307		MOV.W	R0, -2[SP]	; -2(SP) <- D0
$FE030A		JMP	AdDeb	; PC <- AdDeb

Zone mémoire réservée aux instructions

Mnémoniques à la place de données binaires

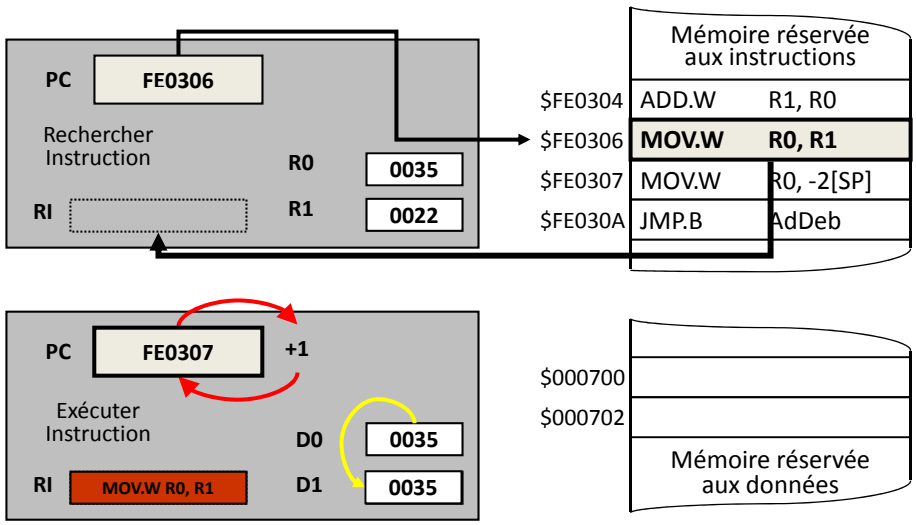
Exécution du programme

Instruction : ADD.W R1, R0



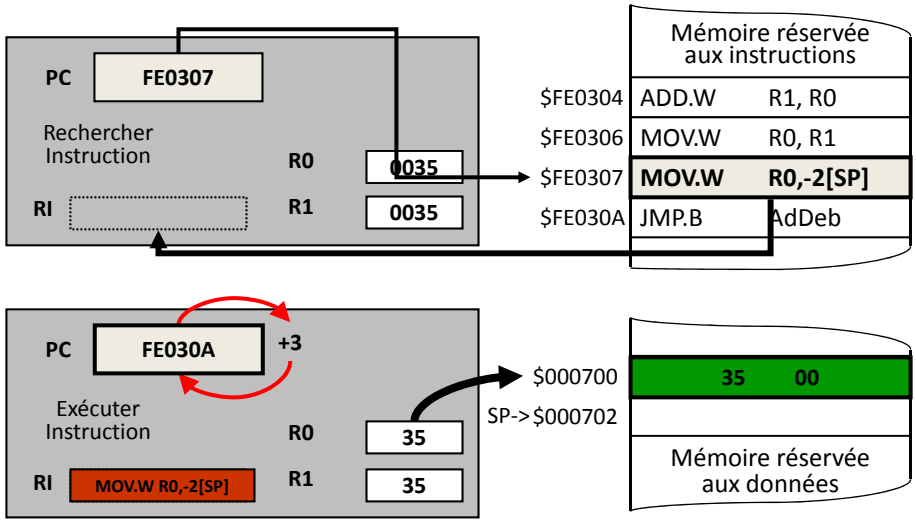
Exécution du programme

Instruction : MOV.W R0, R1



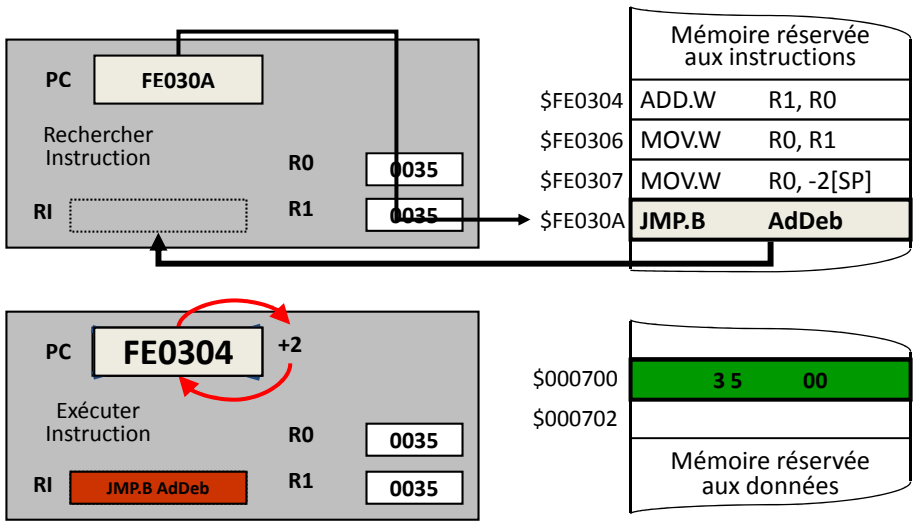
Exécution du programme

Instruction : MOV.W R0, -2[SP]



Exécution du programme

Instruction : JMP.B AdDeb

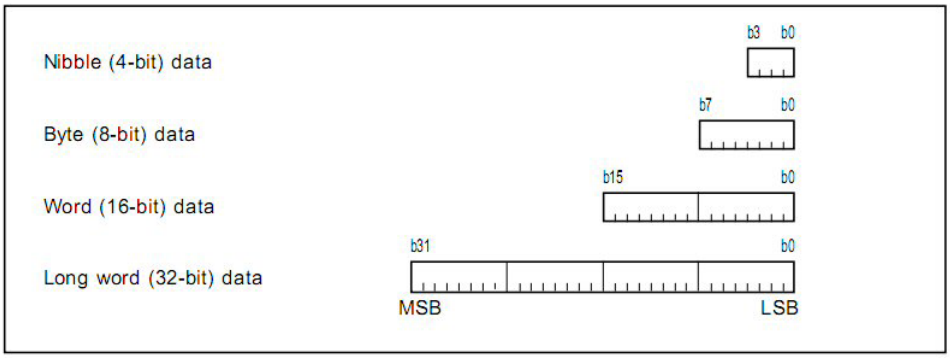


Types de Données

- Le **µ P** traite des données binaires **non typées**
  - C 'est le programmeur qui définit le type des données manipulées par :
    - le **choix** des instructions,
    - la **logique** de l 'algorithme.
- Les instructions traitent les données suivantes :
  - **Entiers signés** sur 8, 16, ou 32 bits
  - **Entiers non signés** sur 8, 16, ou 32 bits
  - **String**
  - Nombres exprimés en **virgule flottante** (réels)

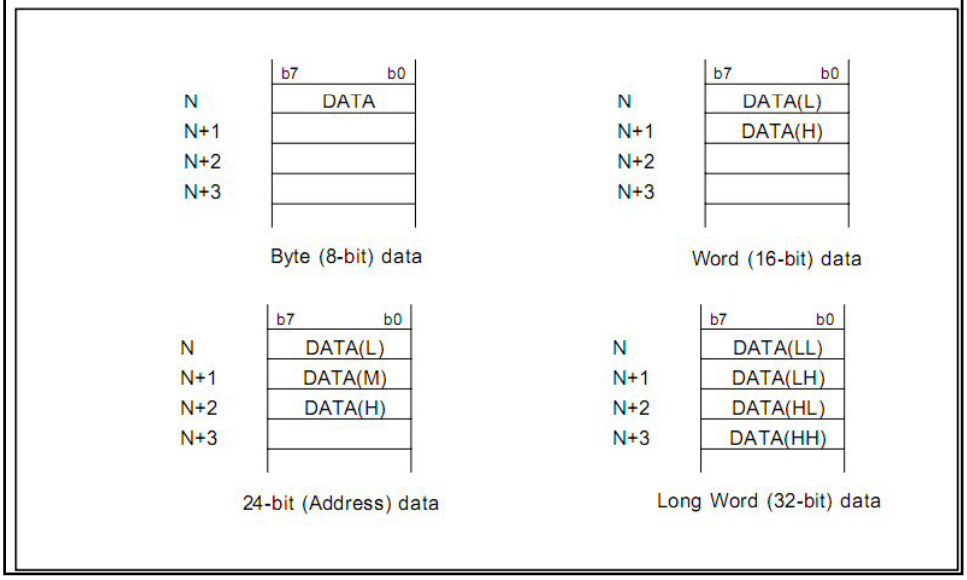
Data Arrangement

Data Arrangement in Register



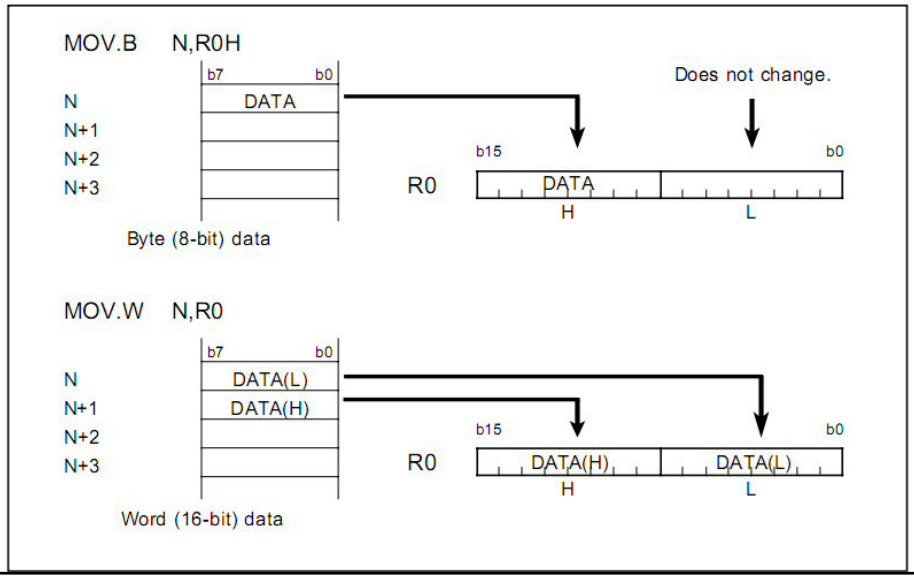
Data Arrangement

Data Arrangement in Memory



Data Arrangement

Data Arrangement in Memory



Étapes de développement d ’un projet industriel

- Cahier des charges à respecter **impérativement**
  - Document **contractuel** entre le concepteur et son client
- Dossier logiciel
  - Analyse **fonctionnelle**
  - Découpage en **modules**
  - **Communication** entre les modules
- Codage des programmes prenant en compte :
  - Les contraintes **techniques**
  - Les **spécifications** du client
  - Les compétences et les habitudes du **personnel**
  - Les **disponibilités** en outils de développement

État de l’art

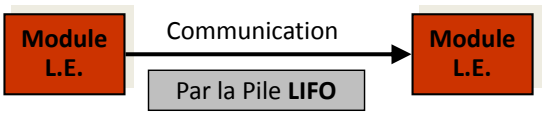
Avec pour objectif : minimiser les coûts

- **Séries faibles**
- Minimiser les coûts de développement
  - Codage en **langage évolué**
    - C, C++, Java, Visual Basic, …
    - Environ 95 % du code
- Parties critiques
  - Codage en assembleur
    - **Temps** de réponse **minimum**
      - **Taille** du code **minimum**
    - Accès aux **entrées sorties**

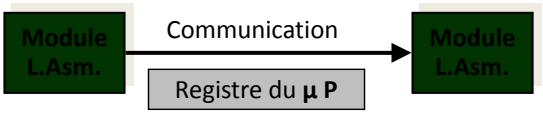
- **Grandes séries**
- Minimiser le coût
  - des composants
  - de fabrication
- Taille mémoire
  - Inférieure aux **seuils justifiés** par des aspects économiques
  - Codage **langage C** en général
    - Employé au maximum
    - Choix de structures efficaces
  - Codage en assembleur
    - Taille en **dessous** des seuils
    - Temps de réponse **critiques**

Programmation modulaire

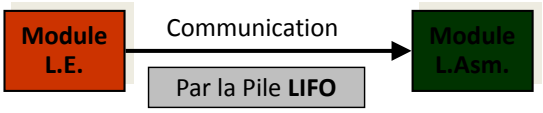
- Communication entre modules codés en langage évolué



✉ Communication entre modules codés en langage assembleur



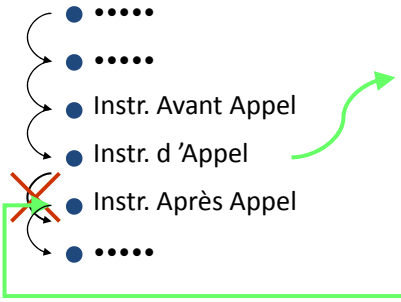
✉ Communication entre modules codés en :
- ◆ Langage assembleur
- ◆ Langage évolué



Mécanismes d'appel et de retour de sous programme

- Pour tous les langages de programmation
  - – Assembleurs, évolués (C, Basic, .), objets (C++, Java, .)
  - – Un programme : suite d 'appels de sous programmes

✉ Description macroscopique



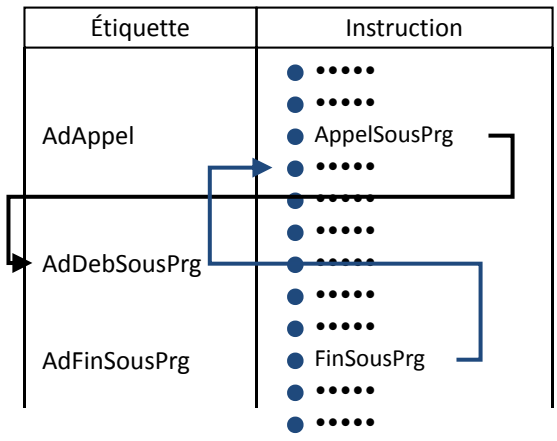
✉ Description détaillée
- ◆ Problèmes à résoudre
  - ➔ Se brancher à l 'adresse de début du sous programme
  - ➔ Ménager l 'avenir en mémorisant l 'adresse de retour
  - ➔ Permettre l 'imbrication de sous programmes

Pile LIFO (Last In, First Out)

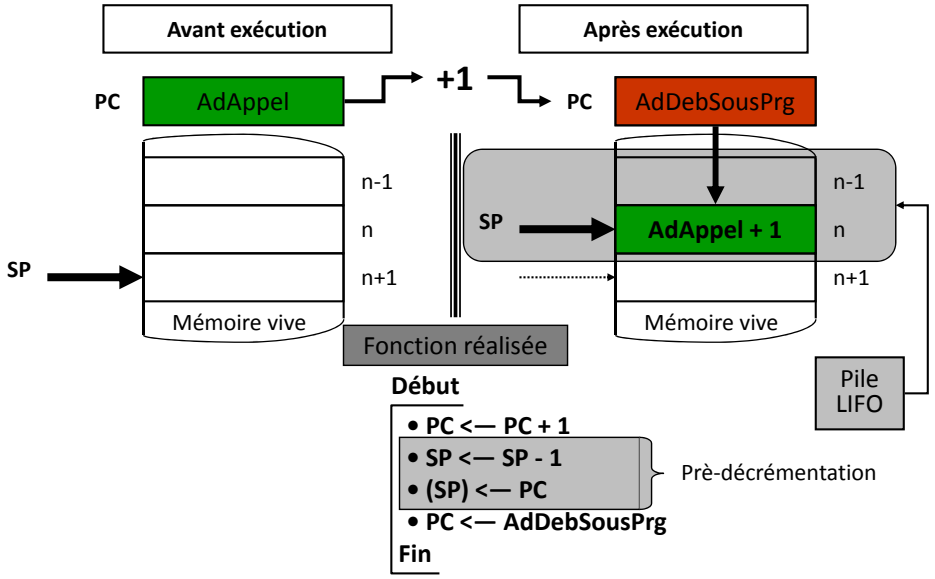
- Description
  - Zone mémoire dont le contenu est géré par :
    - Un pointeur de pile **(stack pointer) - registre d'adresse spécifique**
    - Deux primitives :
      - **Empiler (Objet)**
      - **Dépiler (Objet)**
- Hypothèses théoriques
  - Instruction codée sur une seule ligne mémoire
  - Largeur de la mémoire **(nombre de bits en sortie) identique à la taille du compteur ordinal (PC)**
  - Notation symbolique

Mécanisme d 'appel/Retour d 'un sous programme

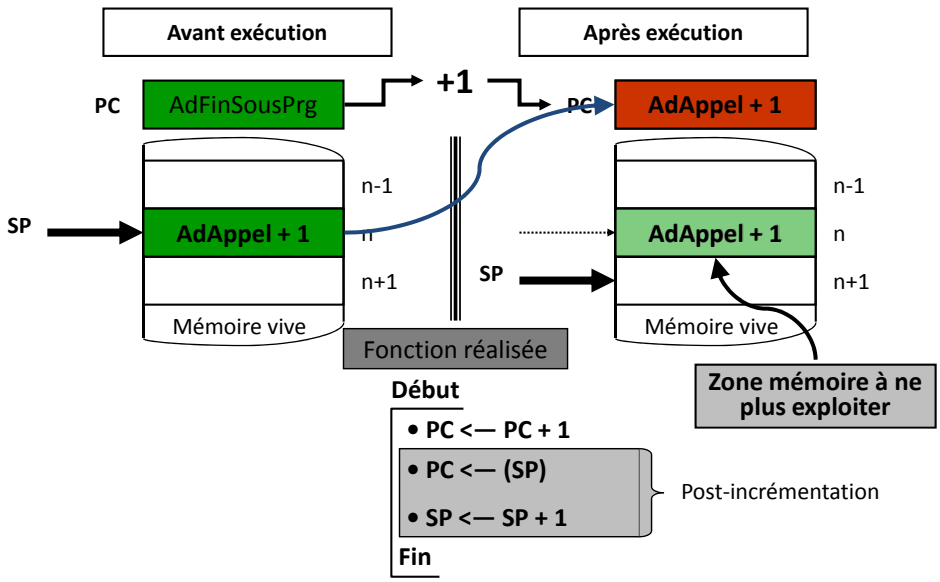
- Algorithme



Instruction d'appel de sous programme

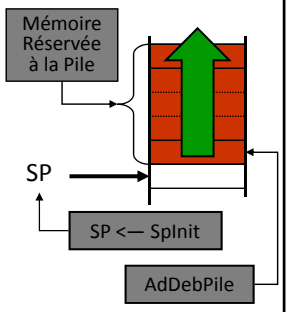


Instruction de retour de sous programme



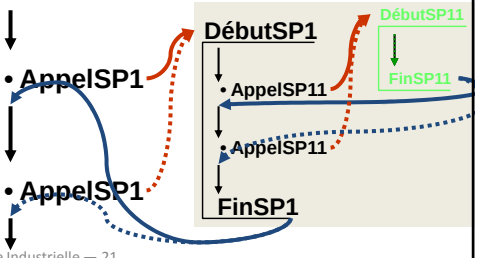
Commentaires supplémentaires

- Développement **complet** d'une application
  - Définir une zone de mémoire vive **réservée à la pile**
  - **Initialiser** le pointeur de pile en conséquence



- Développement d'une application à partir d 'un noyau déjà existant
  - Zone mémoire vive réservée pour la pile **déjà définie**
  - En conséquence, **ne pas initialiser** le pointeur de pile

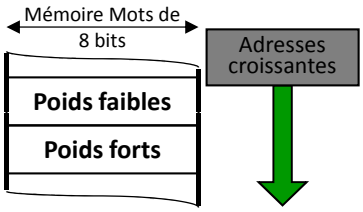
- **Imbrication** de sous programmes
  - La structure de la pile LIFO permet cette imbrication



- Les **principes** doivent être adaptés aux **caractéristiques** de **chaque microprocesseur**

Application Processeur M32C87

- Bus de donnée : **16 bits**
- Bus d 'adresse : **24 bits**
  - Convention Renesas
    - **Poids faibles en premier**
      - Adresse paire
    - **Poids forts en second**
      - Adresse impaire



Programme Source Assembleur		
Étiquette	Opération	Opérande
DEBUT	JSR	AdDebSsPrg
	•••••••	••••••
AdDebSsPrg	NOP	
	RTS	

Codage des instructions (programme objet)

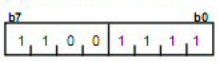
JSR AdDebSsPrg — Jump to Subroutine

Code instruction

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

Hexadécimal			
		C	F
0	2	0	0

(1) JSR.W label



dsp16 = address indicated by label - (start address of instruction +1)

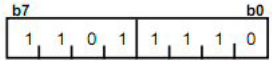
NOP — No Operation

Code instruction

7	6	5	4	3	2	1	0
1	1	0	1	1	1	1	0

Hexadécimal	
D	E

(1) NOP



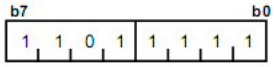
RTS — Return from Subroutine

Code instruction

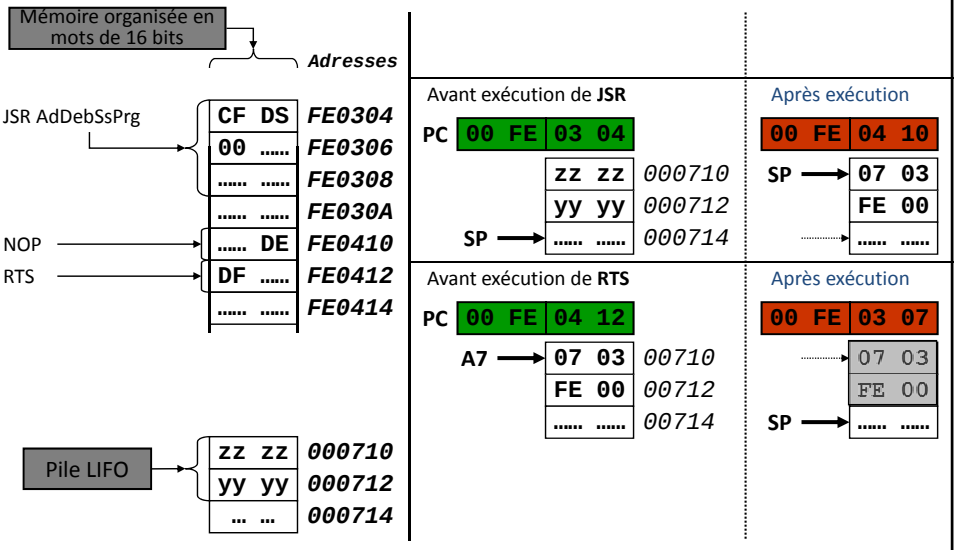
7	6	5	4	3	2	1	0
0	1	1	1	0	1	0	1

Hexadécimal	
D	F

(1) RTS



Etat de la pile pendant l'exécution du programme

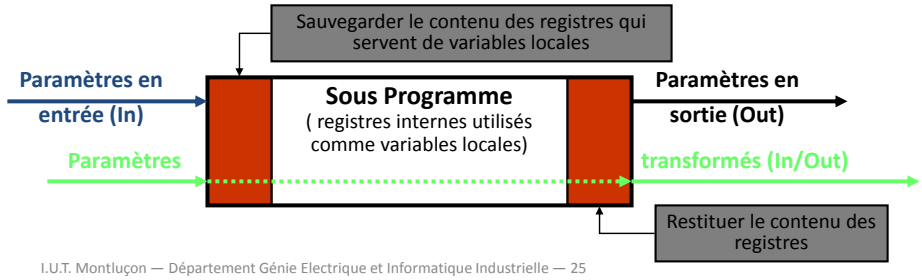


Empilement et dépilement d'objets

- Objectifs
  - Décomposer une application en **modules**
    - **Sous programme** en langage assembleur équivalent à une **procédure en langage évolué**
    - Passage des paramètres par **registres internes**
  - Variables locales au module
    - Utilisation des **registres internes** (plus rapides)
    - **Ne doivent pas interférer** avec le programme appelant

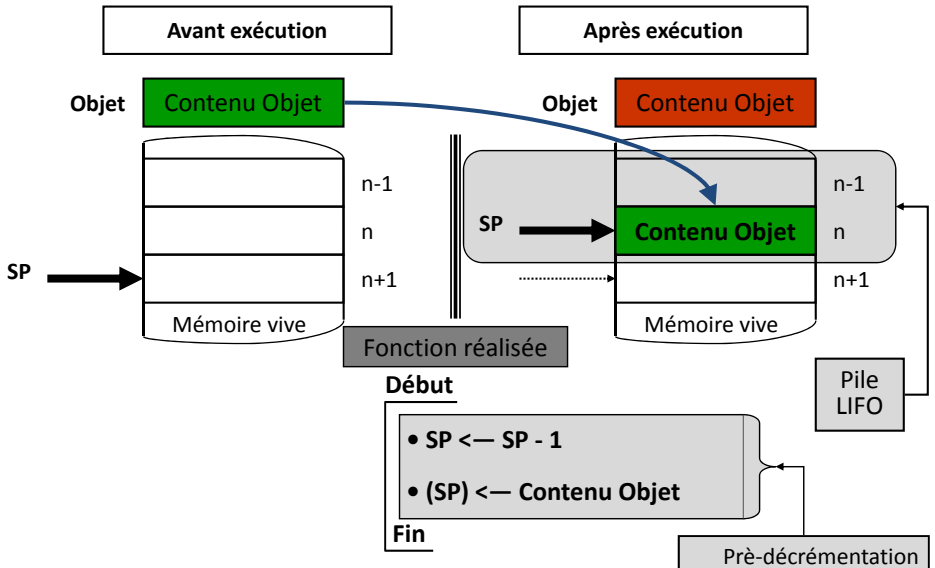
✉ Solution

- **Sauvegarder le contenu des registres** utilisés (variables locales) au début du sous programme
- **Restituer le contenu des registres** avant l'instruction de retour de sous programme



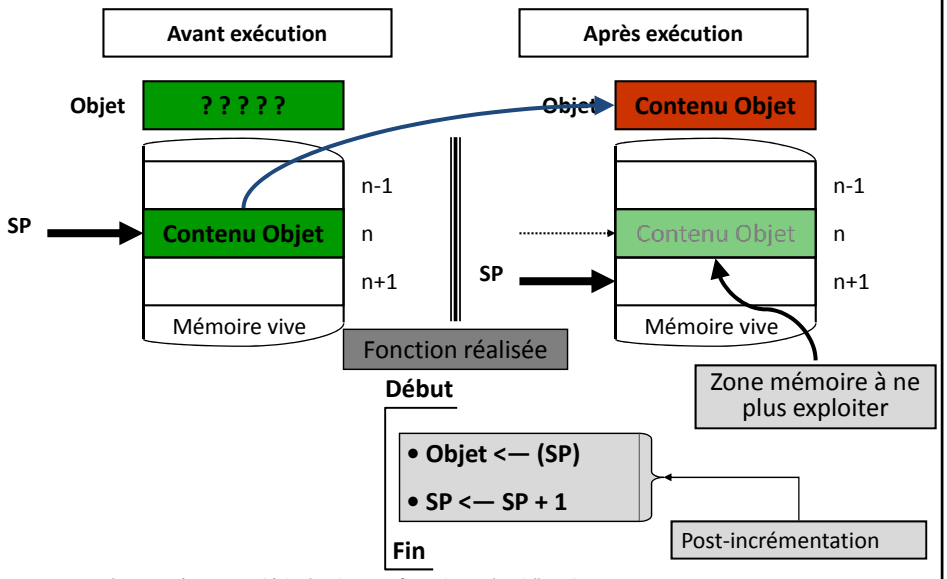
Empilement d'objets

Hypothèse : taille objet = taille mémoire



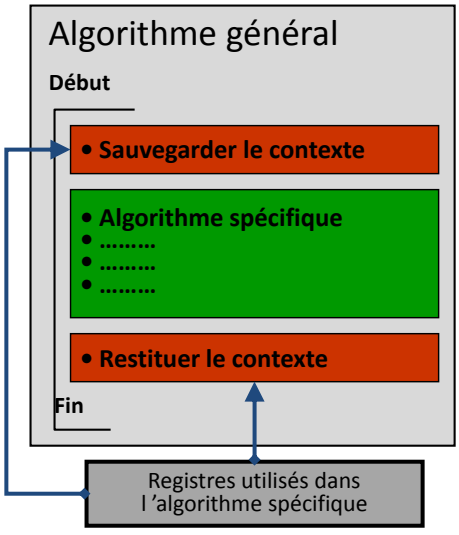
Dépilement d'objet

Même hypothèse

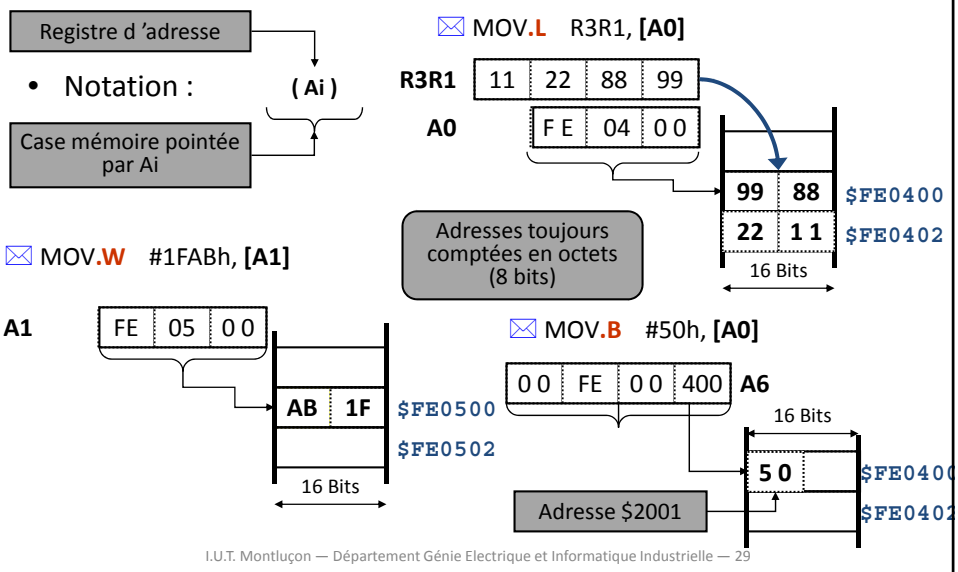


Application : réaliser un sous programme utilitaire

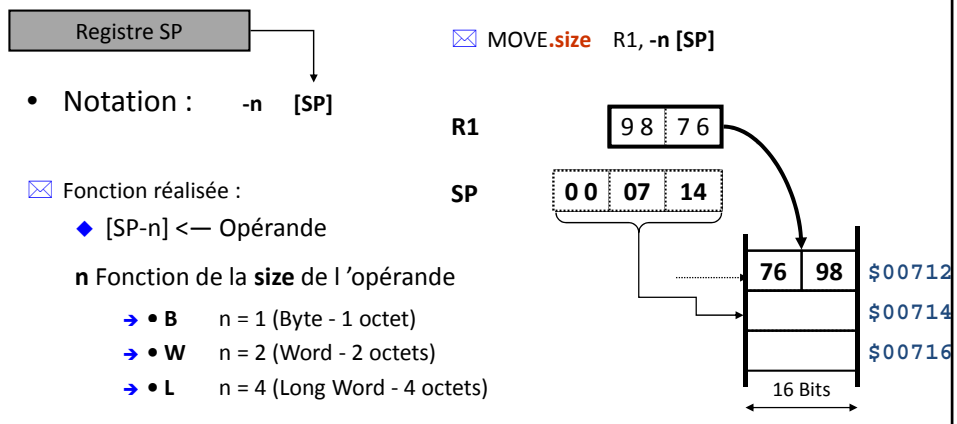
- Objectif
  - Mise en œuvre **facile**
- Méthode
  - Décrire **l'action réalisée**
  - Définir clairement **l'interface de communication**
  - **Sauvegarder et restituer le contexte** dans le sous programme utilitaire
- **Si** impératifs de **rapidité**
  - Ne pas sauvegarder **systématiquement** les registres utilisés
  - Noter les **registres détruits**



Mode d'adressage : adress register relative (registre d'adresse)



Mode d'adressage : adress register relative (Stack Pointer)



Fonction réalisée :

- [SP-n] <— Opérande

n Fonction de la **size** de l 'opérande

- • **B** n = 1 (Byte - 1 octet)
- • **W** n = 2 (Word - 2 octets)
- • **L** n = 4 (Long Word - 4 octets)

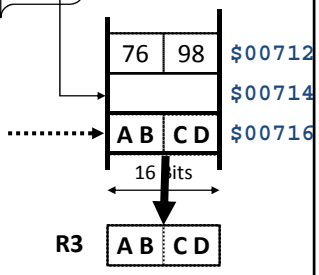
Mode d'adressage : adress register relative (Stack Pointer)

Registre pointeur de pile

- Notation : **n[SP]**

✉ MOVE.**W** 2**[SP]** , R3

SP 00 **07** **14**



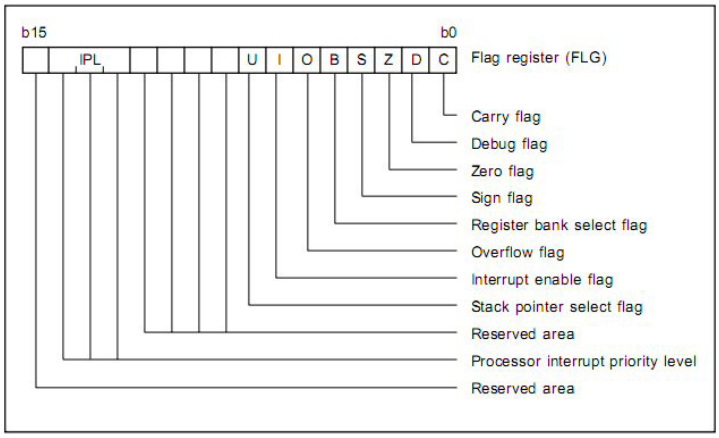
✉ Fonction réalisée :

- ◆ Opérande <— [SP+n]

n Fonction de la **taille** de l 'opérande

- ➔ • **B** n = 1 (Byte - 1 octet)
- ➔ • **W** n = 2 (Word - 2 octets)
- ➔ • **L** n = 4 (Long Word - 4 octets)

Registe d'état (FLG – Flag Register)



✉ Indicateurs **modifiés** après :

- ◆ Une instruction **arithmétique**
- ◆ Une instruction **MOV**

✉ **Conséquence :**

- ◆ Sauvegarder **FLG en premier** sur la pile : **PUSHC FLG**

Application
Sous Programme utilitaire

Cahier des charges

Sous programme

Aucun contenu de registre modifié

Remarque

L 'algorithme spécifique à la fonction réalisée utilise le registre R1

Sauvegarder le contexte

PUSHC FLG

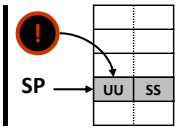
SR | SS | UU → CCR

SP →

Etat de la pile

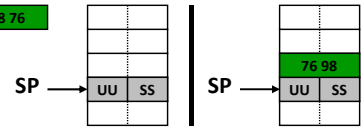
Avant | Après

Exécution Instruction



MOVE.W R1,-2(SP)

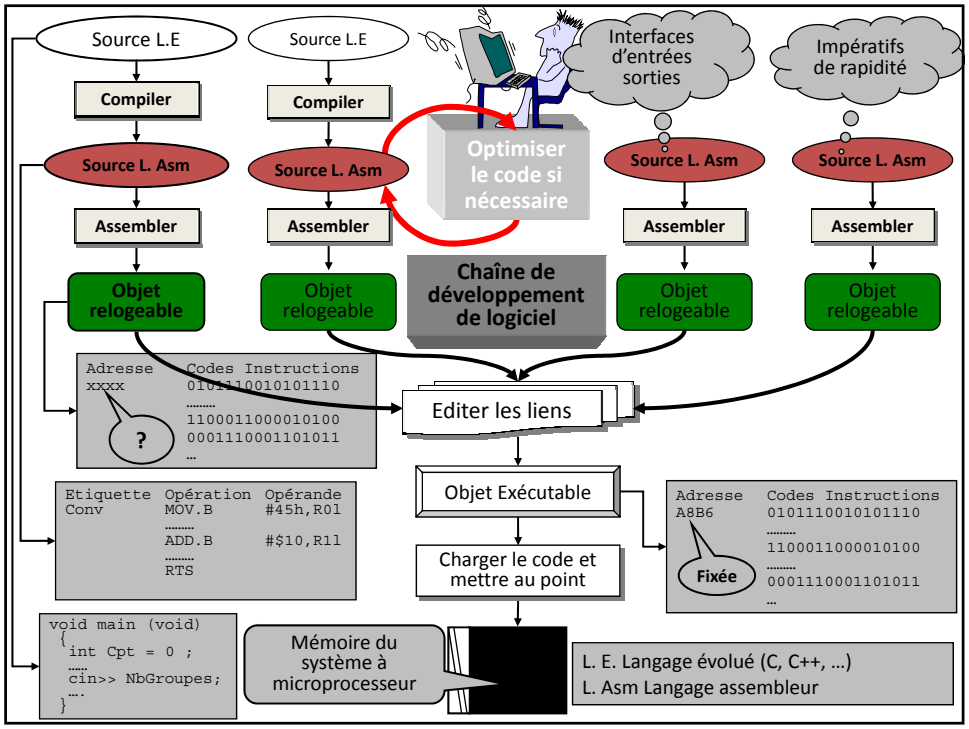
R1 98 76



Restituer le contexte

```
MOV.W -2[SP],R1
POPC FLG
```

I.U.T. Montluçon — Département Génie Electrique et Informatique Industrielle — 33



Fonction assurée par chaque programme de la chaîne de développement

- Un compilateur transforme un programme source écrit en langage évolué (ASCII) en source assembleur (ASCII) du microprocesseur cible
  - Optimisation possible du langage assembleur généré par le compilateur
- Un assembleur transforme le source assembleur en objet relogeable
  - Langage binaire dont les adresses peuvent être recalculées
- L 'éditeur de liens lie tous les modules relogeables (modules créés par le programmeur, bibliothèques) pour créer l 'objet exécutable
  - Langage binaire chargeable dans la mémoire du microprocesseur cible
  - Création de la table des symboles