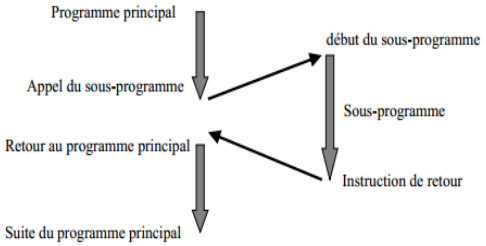
Chapitre 3- Les sous programmes et les interruptions

Utilisation de sous programmes

Lorsqu'une tache d'une application se répète dans un programme, il est conseillé de la considérer comme une entité complète dépendante du programme global d'où le nom de sous programme appelé aussi subroutine qui à la même signification que la notion de fonction en langage C.
Lorsqu'un programme a une certaine quantité de codes, il est utile de l'écrire en le découpant en différentes tâches. On fait appel aux sous programmes pour deux raisons principales:
1-Amélioration de la lisibilité du programme pour faciliter sa compréhension (programmation structurée)
2-Pour l'exécution répétitive d'un même groupe d'instructions.
Le programme en son entier sera composé d'une partie principale et d'une partie secondaire le sous programme. Le passage du programme principale au sous programme se fait par un appel (instruction de branchement) et le passage du sous programme au programme principal se fait par un retour.

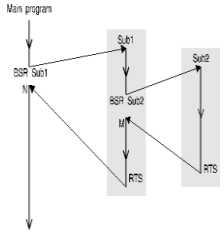


Le principe fondamental de cette démarche est de restituer le **contexte** qui était présent avant l'exécution du sous programme

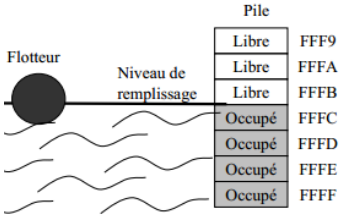
Un sous programme est un ensemble d'instructions qui sera exécuté en dehors du programme principal à un moment donné, il est caractérisé par son espace d'adressage en mémoire formant un bloc compact repéré par une adresse de début et ne peut pas être exécuté tout seul.
le microprocesseur traite l'exécution du sous programme selon les étapes suivantes:
1)Quitter le programme principal et sauvegarder le point de retour
2)Appel du sous programme
3) Exécuter le sous programme
4)Revenir au programme principal
5)continuer le programme principal à l'endroit ou il s'était arrêté
L'appel à un sous programme Se fait par les instructions JSR avec une adresse absolue ou BSR avec une adresse relative, alors que le retour se fait par l'instruction RTS
Comment revenir au bon endroit dans le programme principal ?
-Avant de quitter le programme principal avec par exemple l'instruction BSR ou JSR, le microprocesseur sauvegarde l'adresse de retour au programme principal (i.e. l'adresse de l'instruction du programme principal qui n'a pas été exécutée et qui suit immédiatement l'instruction de l'appel au sous programme) dans la pile.
-Avant de revenir du sous-programme avec l'instruction RTS, le microprocesseur restaure le registre PC avec l'adresse du programme principal qui a été sauvegardée dans la pile lors de l'appel au sous programme

La pile du 68000 (stack)

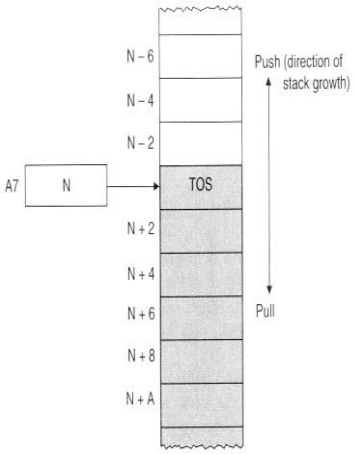
C'est une zone mémoire particulière caractérisée par une gestion LIFO (la dernière donnée empilée dans la pile c'est la première qui sera dépilée) par l'intermédiaire d'un Pointeur de Pile qui est une adresse qui indique le niveau de la pile. La raison d'utiliser LIFO est du au fait que le MC68000 autorise la récursion (jusqu'à 8 niveaux d'imbrication de sous programmes: un sous programme qui appelle un autre sous programme et ainsi de suite)



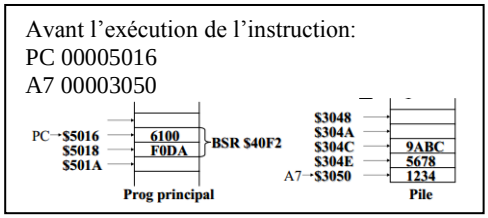
L'accès à la pile est effectué par l'intermédiaire du registre pointeur de pile (A7) appelé aussi (SP: stack pointer).
Celui-ci doit contenir absolument une valeur d'adresse PAIRE qui indique l'emplacement de la pile c'est un indicateur de niveau Ce registre A7 contient l'adresse du dernier élément de la pile : on dit qu'il pointe le sommet de la pile. Dans le cas du MC68000 la case mémoire pointée par A7 est une case remplie et la pile croit vers les

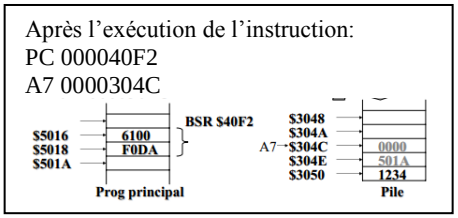


petites adresses car l'empilement des données dans la pile (remplissage: push) se fait par décrémentation du SP et le depilement (vidage:pop)se fait par incrémentation du SP .
Le contenu du registre A7 est décrémenté par 2 ou 4 lorsqu'une donnée de type mot (16bits) ou long mot (32 bits) est empilée ; il est incrémenté de même lorsqu'une donnée est dépilée.
On peut empiler:
-Des adresses de retour
-Des paramètres passés à la procédure (valeurs, adresses).
-Le contenu des registres dans la procédure appelée afin de pouvoir tous les restituer avec leur contenu d'origine avant des sortir du sous-programme (contexte du programme).

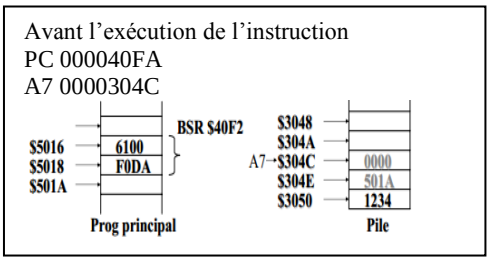


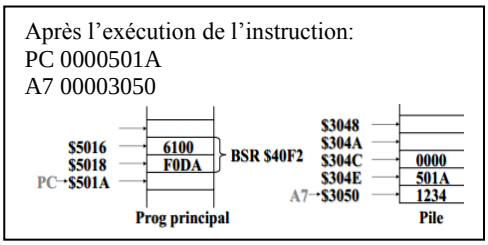
Exemple d'utilisation de la pile
BSR $40F2





RTS





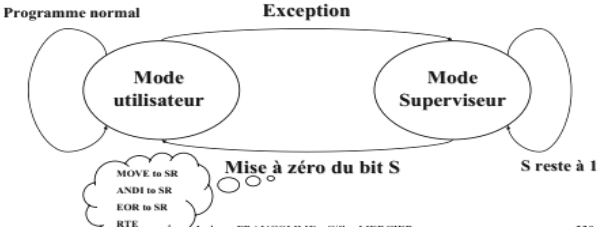
Pour Le MC68000 il n ya pas d'instruction explicite push et pop la manipulation de la pile se fait avec l'instruction move. Les opérations effectuées par BSR ou JSR et RTS ou RTR, se résument dans ce qui suit

BSR <label> =
[A7]← [A7] - 4
M([A7])]← [PC]
[PC]←[PC] + d8

On décrémente par 4 avant d'empiler l'adresse d'une longueur de 4 octets
Pour JSR on utilise une adresse absolue au lieu d'un déplacement d8 à8bits. RTR : Dépilement de PC pour le retour au programme principal et restitution de registre code condition

RTS =
[PC]←[M([A7])]
[A7]←[A7] + 4

Dans le MC68000 Il existe deux pointeurs de pile : Le registre A7 et le registre A7'.
Le premier est utilisé en mode Utilisateur et s'appelle USP. Le deuxième est utilisé en mode Superviseur et s'appelle SSP. Pour passer de «l'un à l'autre », il faut modifier le bit S (bit 13) du registre de code condition CCR.Bit S = 0 => Mode Utilisateur , Bit S = 1 => mode Superviseur
Mode Superviseur : Aucune restriction sur le jeu d'instructions, ni sur les zones adressées.
Mode Utilisateur : Toutes les instructions ne sont pas autorisées (Instructions privilégiées interdites).
Certaines zones ne peuvent pas être adressées :Octet de poids fort du registre CCR,L'accès au registre pointeur de pile SSP.
Le système d'exploitation dispose de l'accès à toute la machine (mode superviseur). Donne une abstraction qui simplifie les programmes utilisateur. Les exceptions provoquent le passage en mode superviseur



les instructions speciales orientées pile

MOVEM: Utilisation pour la sauvegarde du contenu des registres par une procédure appelante : (avec mot ou long mot)

MOVEM.L<Liste de registres>,<AE>

MOVEM.L<AE>, <Liste de registres>

Exemples: MOVEM.L A0-A4/D0-D7 , -(A7)

Les registres A0,A1,A2,A3,A4 et D0 à D7 sont sauvegardés dans la pile.

Pour leur restitution, il faudra écrire : MOVEM.L (A7)+ , A0-A4/D0-D7

Passage de paramètres

Dans un langage de haut niveau, procédure SIN(X) est appelée pour effectuer le calcul correspondant. Cette procédure utilise une donnée en entrée X et retournera une valeur en sortie. On parle de passage de paramètres qui s'effectue donc entre le sous programme appelé et le programme appelant

Passage de paramètre par valeur dans un registre

Un moyen utilisé consiste à placer la valeur dans un des registres de données (ou d'adresse).

```
MOVE.B #$20,D0
BSR SIN_X
…
SIN_X …
RTS
```

Cette Méthode est pratique lorsque le nombre de données à transférer est petit

L'inconvénient majeur du passage des paramètres par registres est le nombre de registres restant disponibles pour le programmeur. Pour le 68000, on peut au maximum transférer 15 Long mots (D0-D7, A0-A6).

Le passage de paramètres par la pile n'a pas cette limitation

On Rassemble tous les paramètres dans une table en mémoire pour diminuer le nombre de registres utilisés

Exemple: Code partiel du programme:

```
DebProg EQU $D000
Chaine EQU $1000
Test EQU $1100
Para_Bloc EQU $2000
```

Code partiel du programme:

```
ORG Chaine
DC.B 'electronique usto'
Fin_Chaine EQU *
ORG TEST
DC.B «MEM»
Fin_Test EQU *
ORG Para_Bloc
DS.L 4
```

Code partiel du programme:

```
ORG DebProg
…
MOVE.L #Para_Bloc , A0
MOVE.L #Chaine , (A0)+
MOVE.L #Fin_Chaine , (A0)+
MOVE.L #Test , (A0)+
MOVE.L #Fin_Test , (A0)
```

Code partiel du programme:

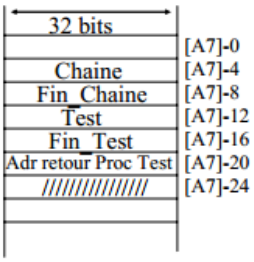
```
LEA Para_Bloc , A0
BSR Proc_Test
…
Proc_Test MOVEM.L A3-A6,-(A7)
MOVE.L (A0)+ , A3
```

```
MOVE.L (A0)+ , A4
MOVE.L (A0)+ , A5
MOVE.L (A0) , A6
; Corps du sous programme
…
MOVEM.L (A7)+ , A3-A6
RTS ;  Fin Proc_Test
```

Dans le cas de procédure récursive, ce type de passage de paramètres ne convient plus car le nombre de valeurs générées par la procédure peut être important.
Exemple : calcul de factoriel n (n!)
Mécanisme du passage de paramètres par la pile : PEA <ea>: Push Effective Address calcule l'adresse effective et l'empile dans la pile pointée par A7

Le transfert des paramètres dans la pile s'effectue à l'aide de l'instruction PEA ;

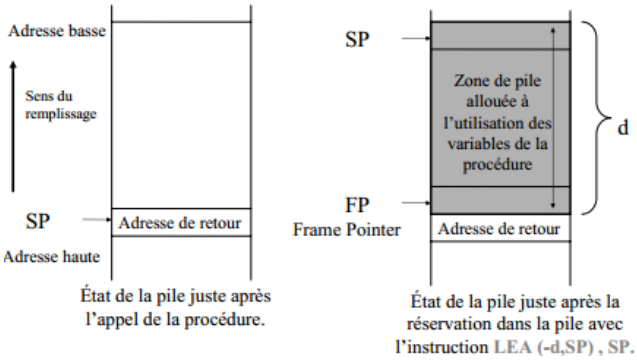
```
…
PEA Chaine
PEA Fin_Chaine
PEA Test
PEA Fin_Test
BSR Proc_Test
LEA (16, A7) , A7
…
Proc_Test  LEA (4,A7) , A0
MOVEM.L (A0)+ , A3-A6
```



En plus du passage de paramètres par la pile, il est souvent nécessaire d'avoir à manipuler des variables temporaires dans une procédure ; Il faudra donc allouer une zone mémoire pour le stockage de ces variables

Exemple : Pour réserver 100 mots dans la pile

```
;Initialisation du registre A6 comme pointeur de cadre (frame Pointer)
LEA (-4 , A7) , A6
;Création de la zone de pile allouée aux variables de la procédure
LEA (-200 , A7) , A7
;Corps de la subroutine
;Libération de la zone mémoire allouée aux variables du sous programme
LEA (200 , A7) , A7
RTS
```



Allocation dynamique de mémoire

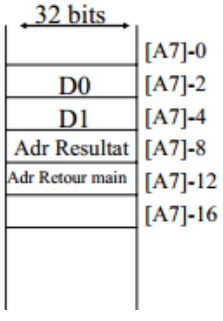
Il existe deux instructions qui effectuent automatiquement cette tache :
LINK :Construit la zone mémoire allouée
ULNK : libère l'espace mémoire alloué
Réservation d'un espace en pile de 64 octets :

```
Proc LINK A1 , #-64
…
ULNK A1; de la mémoire libération
RTS
```

LINK En langage RTL :
- Mettre [SP] ← [SP] - 4
- [M([SP])] ← [A1]
- [A1] ← [SP]
- [SP] ← [SP] – 64

ULINK En langage RTL :
- [SP] ← [A1]
- [A1] ← [M([SP])]
- [SP] ← [SP] + 4

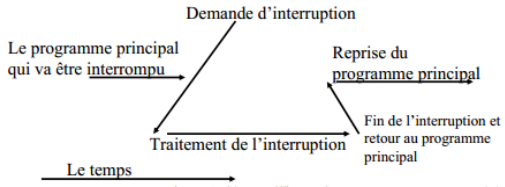
```
Main MOVE.W D0 , -(SP)
MOVE.W D1 , -(SP)
PEA Resultat
```



BSR Calcul
LEA (8 ,SP) , SP
…
FIN BRA FI

Les exceptions

Le 68000 (comme tout microprocesseur) est doté d'un mécanisme qui permet à l'apparition d'un événement fortuit, d'une situation d'exception, d'abandonner le traitement en cours et de transférer l'exécution à un autre traitement associé à cet événement. Dans la plupart des cas, le premier traitement doit être poursuivi, ce qui implique une sauvegarde du contexte lors du transfert.
L'événement fortuit peut avoir deux origines:
–Interne au système: il traduit une anomalie d'exécution ou un cas très particulier de fonctionnement du logiciel (division par 0).
–Externe: une condition matérielle particulière doit être prise en compte (par exemple, un périphérique cherche à «attirer l'attention» du microprocesseur on parle d'interruption).
Les exceptions sont un lien entre les trois composantes essentielles d'un système micro-programmé:
-Le matériel
-Le logiciel
-Le système d'exploitation
Le mécanisme d'une exception est semblable à celui d'un appel à un sous-programme



Plusieurs événements exceptionnels pouvant se présenter simultanément, leur prise en compte doit être hiérarchisée. Ceci couvre l'ordre dans lequel s'effectue le lancement des traitements d'exception, mais aussi la possibilité de suspendre l'exécution de l'un d'eux au profit d'un autre considéré de priorité plus élevé

Vectorisation des exceptions

Un microprocesseur peut distinguer un nombre donné d'exceptions de natures différentes. Il fait correspondre à chacune d'elles un traitement particulier en exploitant une table d'adresses de lancement de ces traitements.

La localisation de cette table est fixe, implicitement connue, mais son contenu est programmable et va être adapté à l'application développée. Ce procédé est nommé: vectorisation des exceptions
Chaque vecteur contient l'adresse de la première instruction à exécuter suite à l'exception ; cette adresse est le point d'entrée de la fonction de traitement de l'exception.
-Toutes les exceptions sont traitées en mode Superviseur

Déroulement d'une exception

Suite à la reconnaissance d'une exception une copie interne du registre d'état du processeur est prise et le processeur est placé en mode superviseur sans aucun traçage (S=1, T1=0, T0=0) ;
pour les interruptions et le RESET le masque d'interruption est mis à jour (priorité de l'interruption ou priorité 7 pour un RESET).
Pour les interruptions seulement un numéro de vecteur d'exception est obtenu à l'aide d'un cycle de lecture spécial sur le bus
-Une copie avant exception du registre d'état (SR), le compteur ordinal (PC) ainsi que des informations complémentaires dépendantes de l'exception sont poussées (empilées) dans la pile superviseur (sauf pour un RESET)
-Le vecteur d'exception est utilisé pour aller chercher la nouvelle valeur du compteur ordinal et les instructions de traitement de l'exception sont exécutées à partir de cette adresse (pour un RESET le pointeur de pile superviseur est également chargé)

Types d'exceptions

-Les exceptions peuvent être classées en fonction de leur nature, de leur possibilité de retour à l'instruction suivant l'exception et du format de l'empilement associé.

En général les exceptions causées par les interruptions, les instructions TRAP (TRAP #n, TRAPcc, TRAPV, CHK, CHK2...), les violations de privilège, les divisions par zéro et les instructions illégales offrent la possibilité de retour à l'instruction suivante

Retour d'Exception

L'instruction RTE (privilégiée) permet de retourner à l'instruction suivant celle où s'est produite une exception. Cette instruction examine le mot contenant le format de l'empilement pour restituer les informations poussées dans la pile.

Exemple des exceptions programmées : Instruction TRAP

TRAP #numéro

Le numéro, de valeur 0 à15, fait partie du code de l'instruction. Il sélectionne un traitement parmi un ensemble de 16, associé à cette instruction. La fonction habituelle de TRAP est de permettre à des programmes utilisateur d'effectuer des appels systèmes

Les exceptions d'origine externe

Les Interruptions

Une interruption on est un événement asynchrone (imprévisible: peut se produire a n'importe quel moment lors de l'exécution d'un programme par le microprocesseur) qui permet d' interrompre de déroulement normal d'un programme pour aller exécuter une tâche particulière.

La prise en compte de l'interruption dépend de l'état d'un masque d'interruption formé par les bits I2, I1, I0 du registre d'état. Les niveaux 1 à6 sont masquables

- Si le niveau de la demande est supérieur à la valeur du masque, le traitement de l'interruption débute.

-Sinon, le traitement est différé jusqu'à ce que le masque devienne inférieur au niveau (le programme en cours se poursuit normalement).

Le niveau 7 est non masquable

Détermination du numéro du vecteur d'exception :

Un vecteur est une position mémoire où le processeur recherche l'adresse du traitement de l'exception. Chaque vecteur est identifié par un numéro codé sur 8 bits (256 vecteurs peuvent être distingués). Le numéro est généré de façon interne et automatique par le 68000 pour toutes les exceptions, sauf les interruptions

– L'adresse absolue d'un vecteur est déterminée en multipliant par 4 son numéro.

– Ce calcul est effectué par le microprocesseur et n'est pas modifiable. L'adresse d'un vecteur donné est donc constante.

– Le contenu du vecteur doit être défini en fonction de l'implantation en mémoire des traitements d'exceptions. Il est programmé par l'utilisateur

La détermination du numéro associé à une interruption requiert un dialogue avec une logique externe

– Le 68000 exécute un cycle de reconnaissance d'interruption au cours duquel il place sur les lignes A0, A1, A2 du bus d'adresses le niveau de l'interruption reconnu

– Une logique externe choisit une vectorisation automatique ou une vectorisation utilisateur

Exemples de sous programmes

```
ORG $1000 ; début de programme
start:
 MOVEQ.L #15,D1  ; 15  D1
 MOVEQ.L #25,D2  ; 25   D2
 JSR GetMin  ; Appel à GetMin (15  D0
 MOVEQ.L #30,D1  ; 30  D1
 MOVEQ.L #16,D2  ; 16  D2
 JSR GetMin  ; Appel à GetMin (16  )  D0
 BRA  FIN
GetMin CMP.L  D1,D2  ; Compare D1 et D2.
     BLE d2min  ; Saut à d2min si D2 >= D1 (comparaison signée).
d1min  MOVE.L  D1,D0  ; D1 est inférieur à D2, on copie D1 dans D0.
     RTS  ; Sortie du sous-programme.
d2min  MOVE.L  D2,D0  ; D2 est inférieur ou égal à D1, on copie D2 dans D0.
     RTS  ; Sortie du sous-programme.
   SIMHALT          ; halt simulator
```

```
FIN:
   END     start     ; last line of source
```

Exemple2

Ecrire un sous programme qui récupère les deux mots déjà placés sur la pile, et qui les traite, et qui sauvegarde le résultat sur la pile. Le programme principal est le suivant

```
ORG $1000 ; début de programme
start:
 LEA   $FFFFF0,A7
 MOVE.W  #15,-(A7) ;  Mise sur la pile du premier paramètre
 MOVE.W  #-89,-(A7) ; Mise sur la pile du seconde paramètre
 BSR   CALCUL   ;Sous programme de calcul
 MOVE.W  (A7)+,D0  ;   Récupération du résultat
 BRA   FIN ;
CALCUL  MOVE.L  (A7)+,A5
       MOVE.W  (A7)+,D2
       MOVE.W  (A7)+,D3
       ADD.W       D2,D3
       MOVE.W  D3,-(A7)
       MOVE.L  A5,-(A7)
       RTS
   SIMHALT            ; halt simulator
FIN:
   END     start     ; last line of source
```

Exemple3

```
ORG $1000 ; début de programme
start:
  LEA $FF9000,A7  ; initialisation du pointeur de pile
  MOVE.L #$AABBCCDD,-(A7) ; empilement de données
  BSR RIEN ; appel au sous programme RIEN
  BRA FIN ;fin du programme principal
RIEN RTS ; sous programme vide
   SIMHALT            ; halt simulator
FIN:
   END     start     ; last line of source
```