

La pile et ses utilisations dans un système à microprocesseur

Module Info 2

Sebastien.Kramm@univ-rouen.fr

IUT GEII Rouen

2013-2014



1/23

Sommaire

Généralités

La pile du 9s12

Caractéristiques

Utilisation pour les sous-programmes

Utilisation pour les interruptions

Sauvegarde temporaire de valeurs



2/23

Notion de pile ("stack" en anglais)

Définition

La pile est une **zone de mémoire vive** dédiée à la sauvegarde de valeurs par le CPU. Elle est utilisée par le processeur de façon **automatique** pour :

- ▶ mémoriser l'adresse de retour lors d'un appel à un sous-programme,
- ▶ mémoriser l'adresse de retour et l'état des différents registres lors d'une interruption.

- ▶ Un programme en assembleur peut également l'utiliser pour sauvegarder de façon temporaire certains registres, sans avoir à déclarer d'emplacement mémoire particulier.



3/23

Caractéristiques

- ▶ Cette zone de mémoire est gérée comme une LIFO (*Last In - First Out*), via un **pointeur de pile** (*Stack pointer*).
- ▶ Celui-ci est un registre **dédié à cet usage**, et qui pointe sur la dernière adresse utilisée¹.
- ▶ Le concepteur du système doit :
 1. Prévoir une zone de mémoire vive pour la pile.
Sa taille dépendra de l'application envisagée (nombre de sous-programme et de routines d'interruptions susceptibles d'être imbriqués).
 2. Initialiser le pointeur de pile dans le bloc d'initialisation du programme.
Exemple : `lds #03fff;` (Freescale 9s12)

1. ou la première adresse disponible, selon les architectures.



4/23

Autres utilisations

- ▶ La pile est utilisée aussi par le code généré par un compilateur C :
 1. Les variables locales à une fonction sont créées sur la pile.
 2. Les arguments transmis à une fonction sont stockés sur la pile, avant l'appel.
 3. La valeur de retour d'une fonction est placée sur la pile, avant l'exécution du `rtos`.
- ▶ Ces points sont à prendre en considération dans le dimensionnement de la pile !
En embarqué sur architecture 16 bits :
 - ▶ `int` \Rightarrow 2 octets.
 - ▶ tableau de 100 `int` \Rightarrow 200 octets.

Sommaire

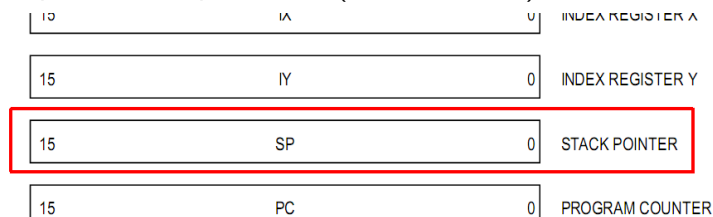
Généralités

La pile du 9s12

- Caractéristiques
- Utilisation pour les sous-programmes
- Utilisation pour les interruptions
- Sauvegarde temporaire de valeurs

Caractéristiques de la pile du 9s12 - 1

- ▶ Registre "pointeur de pile" : SP (*Stack Pointer*), sur 16 bits.



- ▶ La pile peut être placée n'importe où dans le plan mémoire (à condition que de la RAM s'y trouve !)
- ▶ Le pointeur de pile "avance en descendant" : pointe sur la dernière adresse utilisée, et à chaque "utilisation" (appel de sous-programme par exemple), sa valeur **diminue**.

Caractéristiques de la pile du 9s12 - 2

- ▶ **Aucun** mécanisme de protection au niveau processeur : la pile n'est protégée :
 - ▶ ni contre des écritures "sauvages" par le programme utilisateur,
 - ▶ ni contre des débordements sur d'autres zones mémoires. \Rightarrow En assembleur, c'est au programmeur de prendre toutes les mesures de sécurité nécessaires.
- ▶ L'utilisation d'un langage évolué (C) dispense de la gestion "bas niveau" de la pile.
(Initialisation et gestion automatique)

Utilisation lors de l'appel d'un sous-programme

- ▶ L'exécution de bsr ou jsr provoque :
 - ▶ la sauvegarde dans la pile la valeur de l'adresse de retour,
 - ▶ l'incrémement (+2) de SP,
 - ▶ la recopie dans PC de l'adresse spécifiée.
- ▶ 2 octets de pile utilisés.
- ▶ Exemple : soit le programme suivant :

```

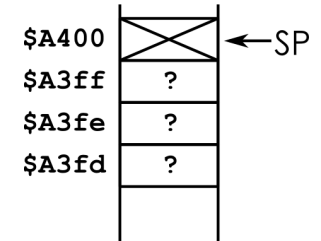
1  b000 8e a3 ff      lds #a400
2  b003 4f          DEBUT  clra
3  b004 bd b0 0a     jsr  MON-SP
4  b007 5f          clrb
5  b008 20 fe      FIN    bra  FIN
6
7  b00a 01          MON_SP nop  ; rien du tout ...
8  b00b 39          rts
    
```

1 - Initialisation du registre SP

```

1  b000 8e a3 ff      lds #a400
    
```

- ▶ initialise SP sur le sommet de la zone réservée à la pile : \$A400. (Cette adresse n'est jamais utilisée.)
- ▶ Les valeurs dans la pile sont indéterminées.



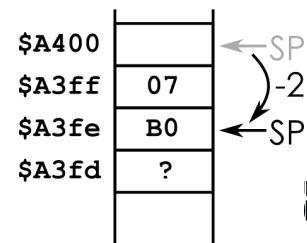
2 - Appel du sous-programme

```

1  b004 bd b0 0a     jsr  MON_SP
2  b007 5f          clrb
3  ...
4  b00a 01          MON_SP nop
    
```

- ▶ Lors de l'exécution de la ligne "jsr MON_SP", le CPU :
 1. calcule l'adresse de la prochaine instruction ("clrb"), soit \$B007,
 2. sauvegarde cette valeur dans la pile (2 octets),
 3. décrémente le pointeur de pile de 2,
 4. recopie dans PC la valeur \$B00A
⇒ l'exécution se poursuit à cette adresse (étiquette MON_SP)

- ▶ Le pointeur de pile contient alors la valeur \$A3FE.



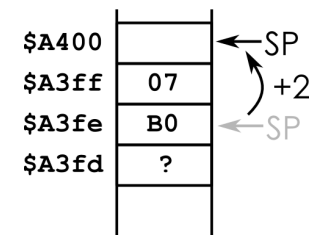
3 - Retour au programme "appelant"

```

1  b00b 39          rts
    
```

- ▶ Lors de l'exécution de l'instruction "rts", le CPU va :
 1. lire dans la pile (via SP) l'adresse de retour,
 2. placer cette valeur (\$B007) dans PC (et donc, poursuite de l'exécution à cette adresse),
 3. incrémenter de 2 le registre SP.

⇒ On est revenu au point de départ.



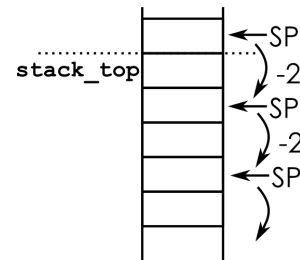
Interêt de l'utilisation d'une pile

- ▶ Ce mecanisme est **récuratif** : un sous-prog. peut appeler un sous-prog., qui lui-même peut appeler un sous-prog., qui peut appeler des sous-prog...
- ▶ Le processeur va toujours s'y retrouver dans les appels successifs (sauf en cas de **corruption** de la pile !)

```

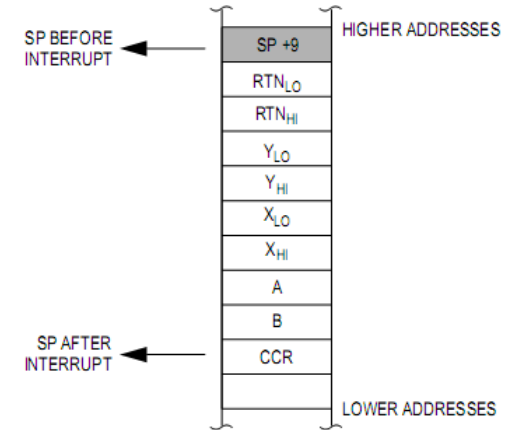
1  debut ...
2      bsr  sp1
3      ...
4
5  sp1  ...
6      bsr  sp2
7      rts
8
9  sp2  ...
10     bsr  sp3
11     rts
12
13  sp3  ...
14

```



Interruptions

- ▶ Principe identique pour les interruptions.
- ▶ Différence : **tous** les registres sont sauvegardés (sauf SP).
- ▶ 9 octets de pile utilisés.
- ▶ L'instruction rti réalise l'opération inverse (dépilement des valeurs dans les registres).

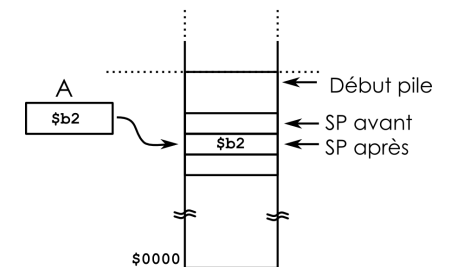


Sauvegarde temporaire de valeurs

- ▶ Le programmeur peut également utiliser la pile pour sauvegarder des registres de façon temporaire, en utilisant les instructions d'empilement et de dépilement.
- ▶ Ces opérations sont dénommées
 - ▶ **Push** (= pousser) empilement, sauvegarde.
 - ▶ **Pull** (= tirer) dépilement, récupération.
- ▶ Ceci est particulièrement utile si on manque de registres.
 - ▶ Plus souple que d'avoir à réserver une case mémoire.
 - ▶ Permet de créer des sous-programmes récuratifs.

Empilement / Dépilement

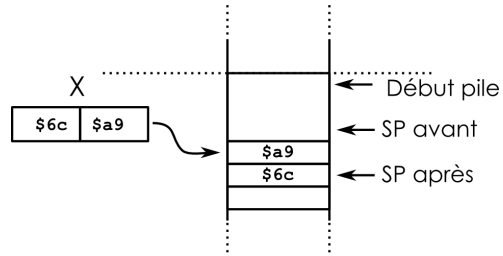
- ▶ psha : sauvegarde la valeur de A dans la pile, et décrémente SP de 1



- ▶ pula : charge A avec la valeur lue sur la pile, et incrémente SP. (opération inverse)

Empilement / Dépilement

- ▶ `pshx` : sauvegarde la valeur de X dans la pile, et décrémente SP de 2 (registre 16 bits)



- ▶ Pour les registres 16 bits, le poids faible est empilé d'abord, puis le poids fort.
- ▶ `pulx` : charge X avec la valeur lue sur la pile, et incrémente SP de 2. (opération inverse)

Exemple d'utilisation

```
1   ldaa 0,x ; lecture dans une table
2   psha      ; Push A
3   ldaa autre_chose
4   ...      ; ici, un calcul avec A
5   staa resultat
6   pula     ; Pull A: on récupère la valeur initiale
```

Attention au risque d'erreurs

- ▶ Si on empile plusieurs registres, il faudra les récupérer dans l'ordre inverse où on les a placés.

```
1   ...
2   psha
3   pshb
4   ...
5   pula
6   pulb
7   ...
```

⇒ Ici, inversion de A et de B !

- ▶ Ne pas insérer entre l'empilement et le dépilement un appel à un sous-programme !

```
1   ...
2   psha
3   bsr  MON_SP
4   ...
5   MON_SP pula
6   ...
```