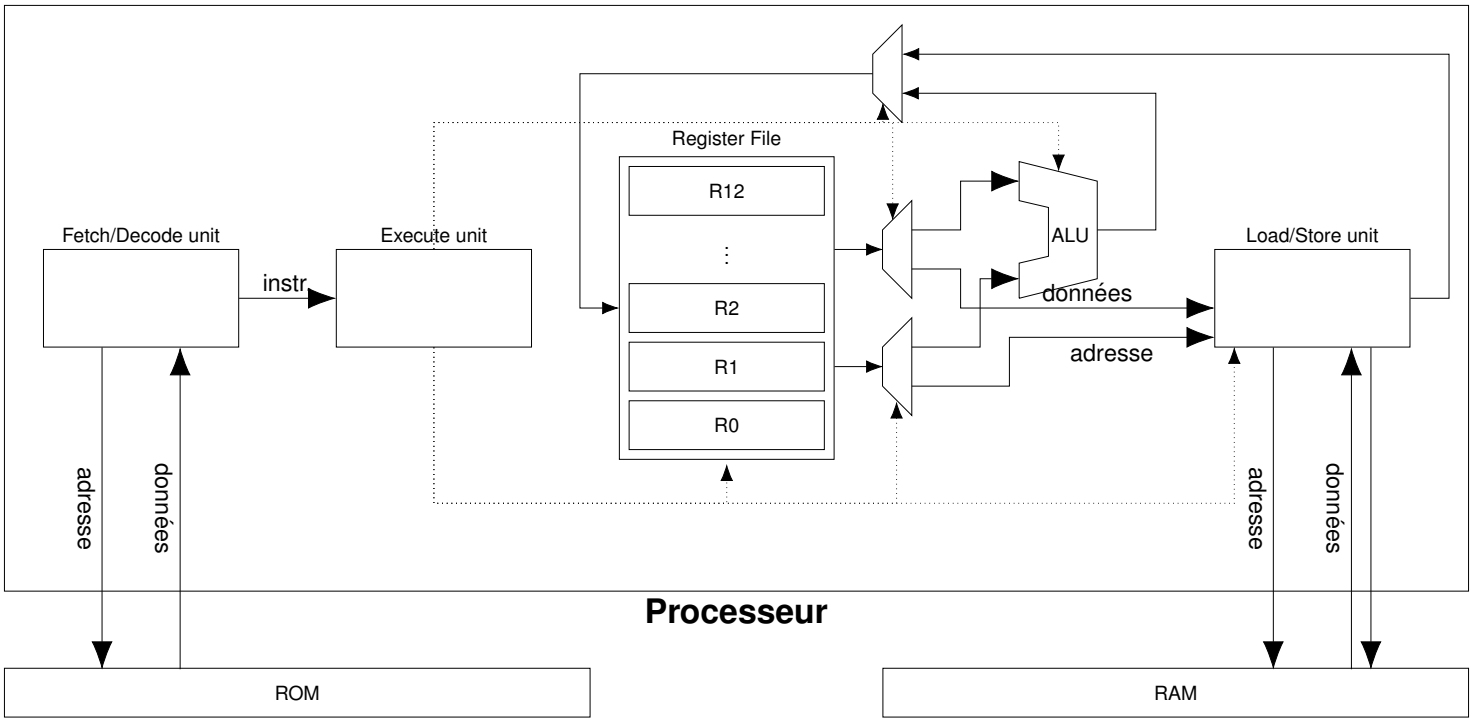
Architecture d'un processeur

Vincent Migliore

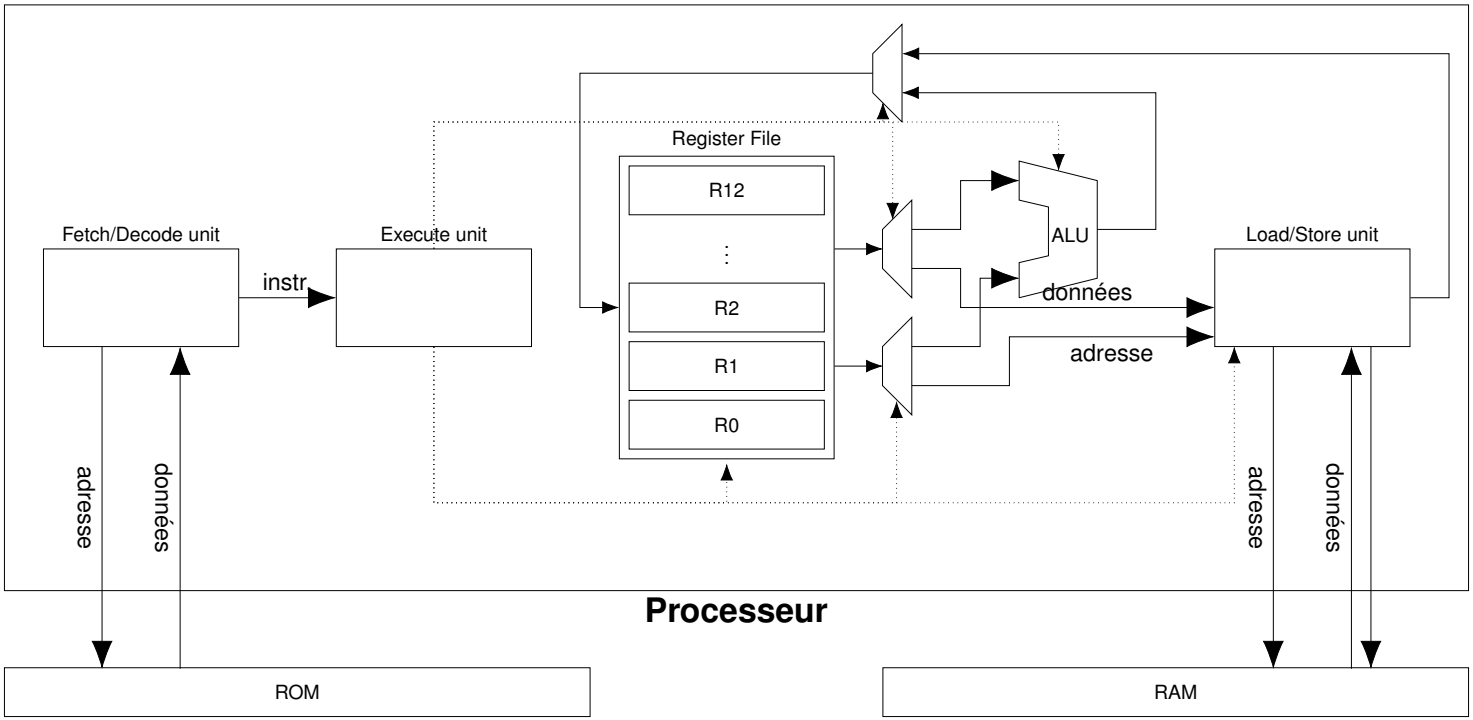
vincent.migliore@insa-toulouse.fr

Fonctionnement d'un processeur : Chemin de contrôle et de données



Chemin de contrôle / chemin de données

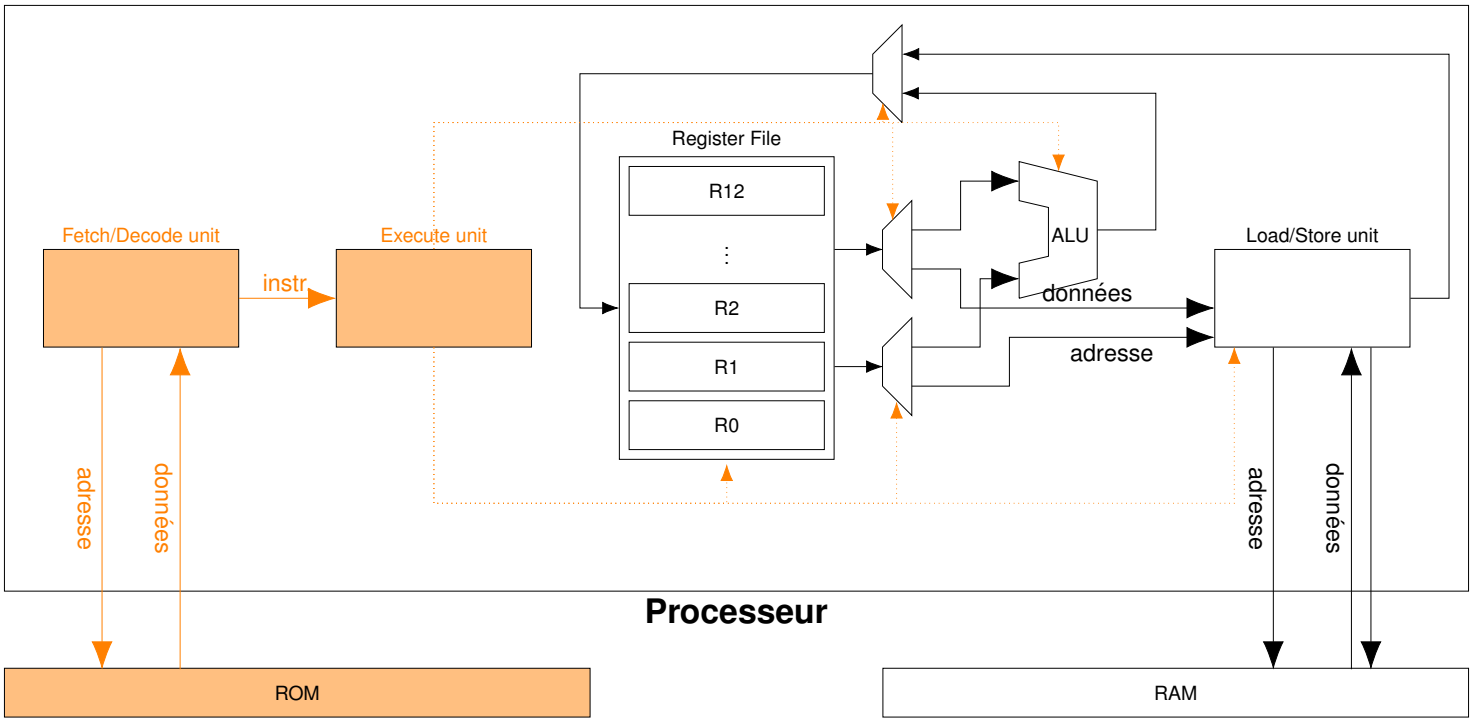
Un processeur est composé d'un ensemble d'unités internes permettant soit le traitement du code (composé d'instructions), soit le traitement des données (composé de variables). Le chemin interne pour le traitement du code se nomme le chemin contrôle, et le chemin interne pour le traitement des données se nomme le chemin de donnée. Le code est en principe stocké dans la ROM, et les données dans la RAM.



Chemin de contrôle / chemin de données

Pour bien comprendre la différence entre chemin de contrôle et de données, prenons l'exemple d'une fonction *f* ayant pour argument des variables *a*, *b* et *c*.

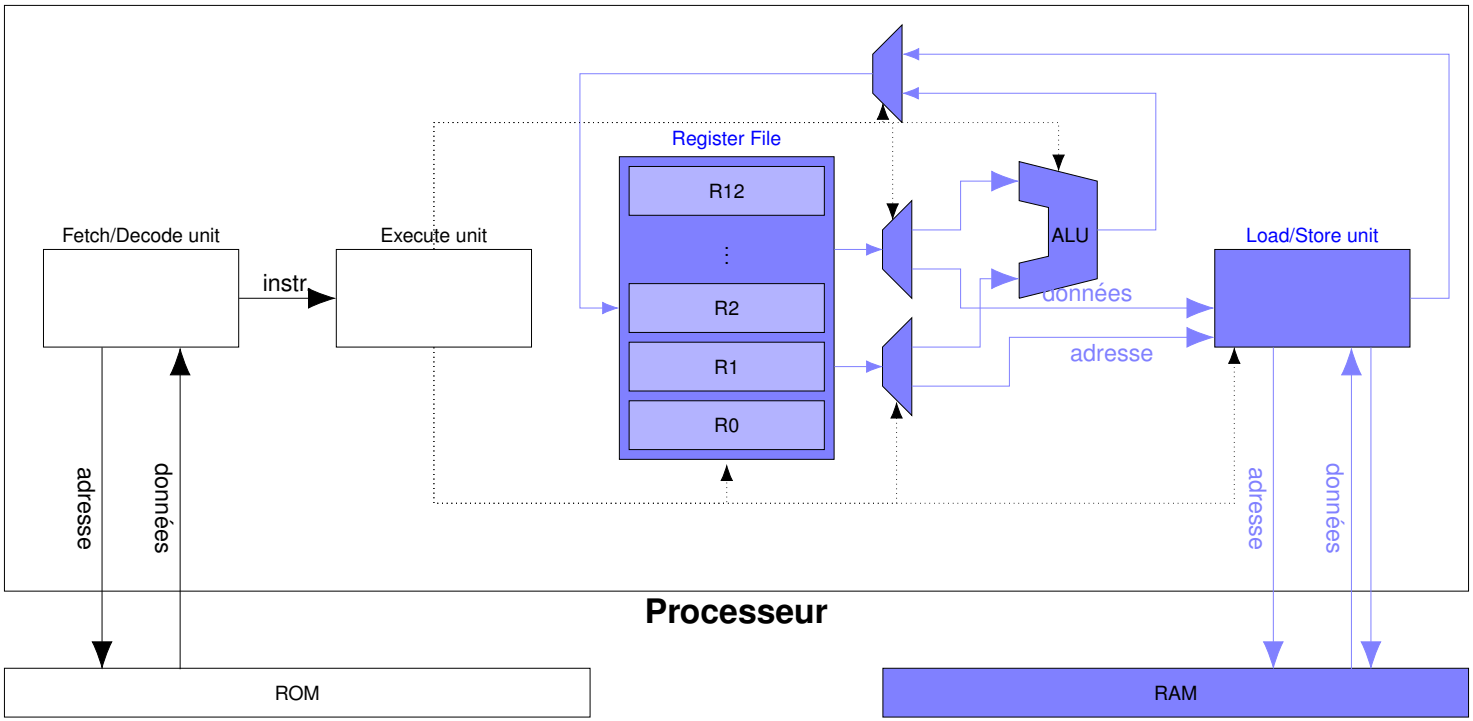
- La description de la fonction *f* est représentée sous forme d'une séquence d'instructions exécutable par le processeur, elle est associée au chemin de contrôle et stockée dans la ROM ;
- Le traitement des arguments, les variables intermédiaires manipulées par la fonction *f* ainsi que la valeur de retour sont des données, elles sont donc associées au chemin de données et stockées dans la RAM.



Chemin de contrôle

Les unités Fetch, Decode, Execute font partie du chemin de contrôle Elles ont pour rôle :

- Fetch : Lire l'instruction suivante depuis la mémoire programme (ROM) ;
- Decode : Décoder l'instruction afin de déterminer l'action (opcode) à exécuter, les opérandes et la destination ;
- Execute : exécute l'instruction en 1 ou plusieurs cycles.



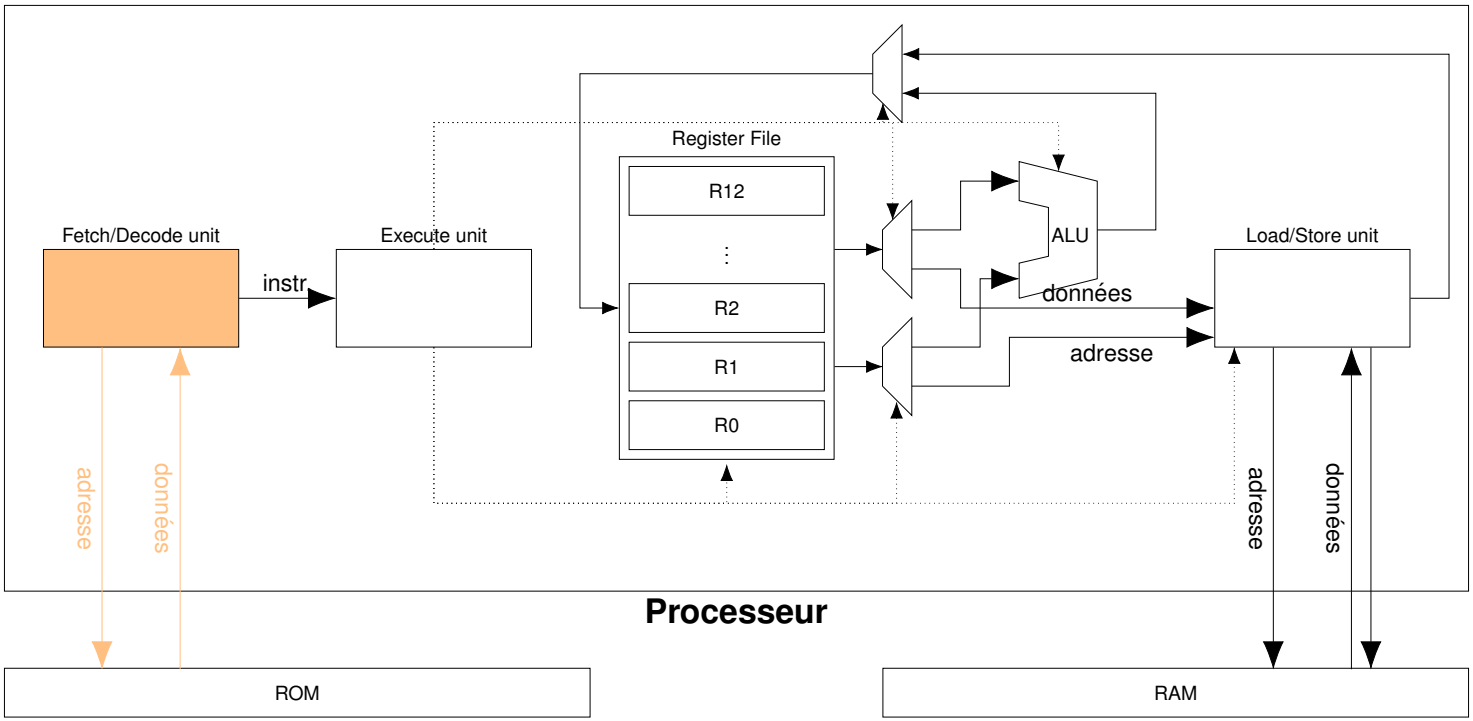
Chemin de données

Les unités Register File, ALU et Load/Store font partie du chemin de données. Elles ont pour rôle :

- Register File : Appelé banc de registres en français, c'est la mémoire interne du processeur permettant de stocker localement les variables ;
- ALU : Appelé unité arithmétique et logique (UAL) en français, son rôle est d'exécuter toutes les opérations arithmétiques et logiques. En général les opérandes sont au nombre de deux et sont des registres, mais certains processeurs peuvent aussi utiliser des adresses mémoire comme opérande (Intel) ;
- Load/Store : Unité permettant la lecture et écriture de la mémoire des données.

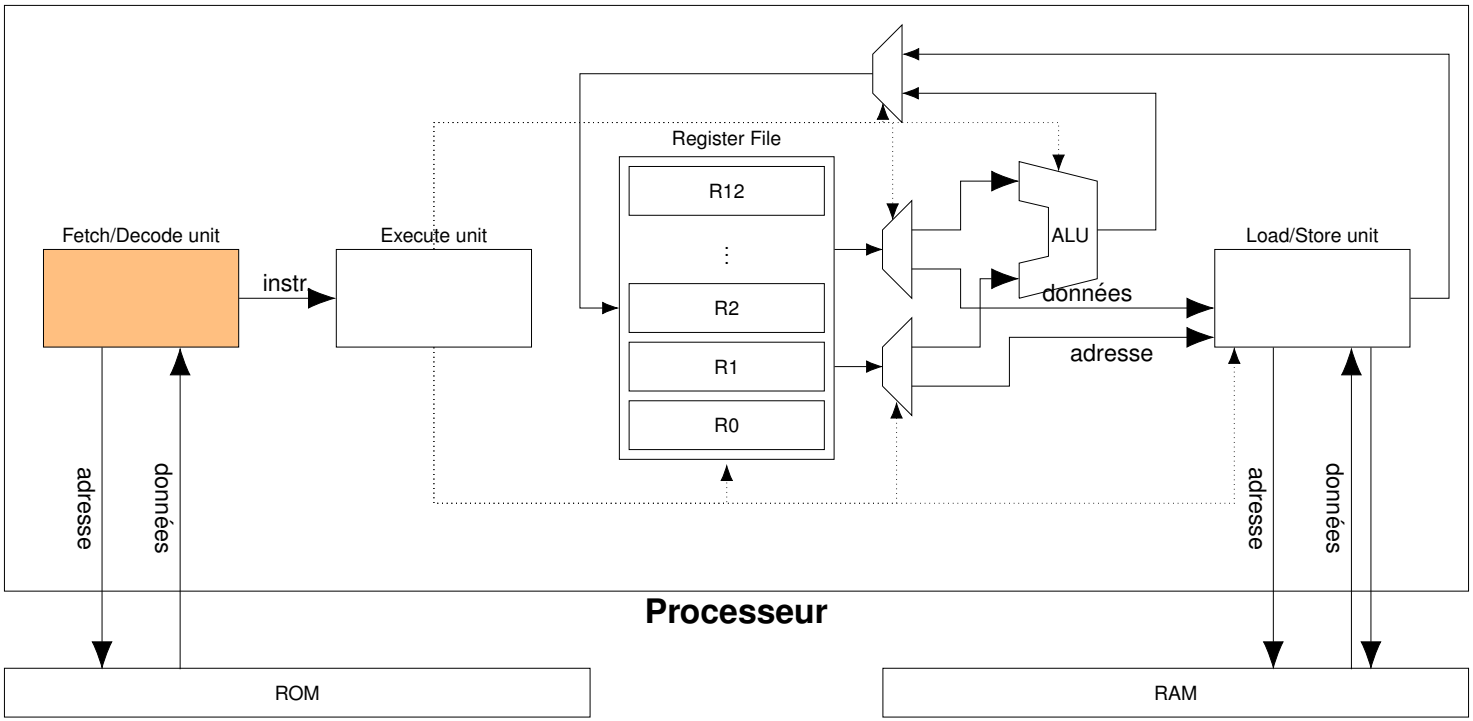
Le pipeline d'un processeur

Récapitulatif pour exécuter une instruction



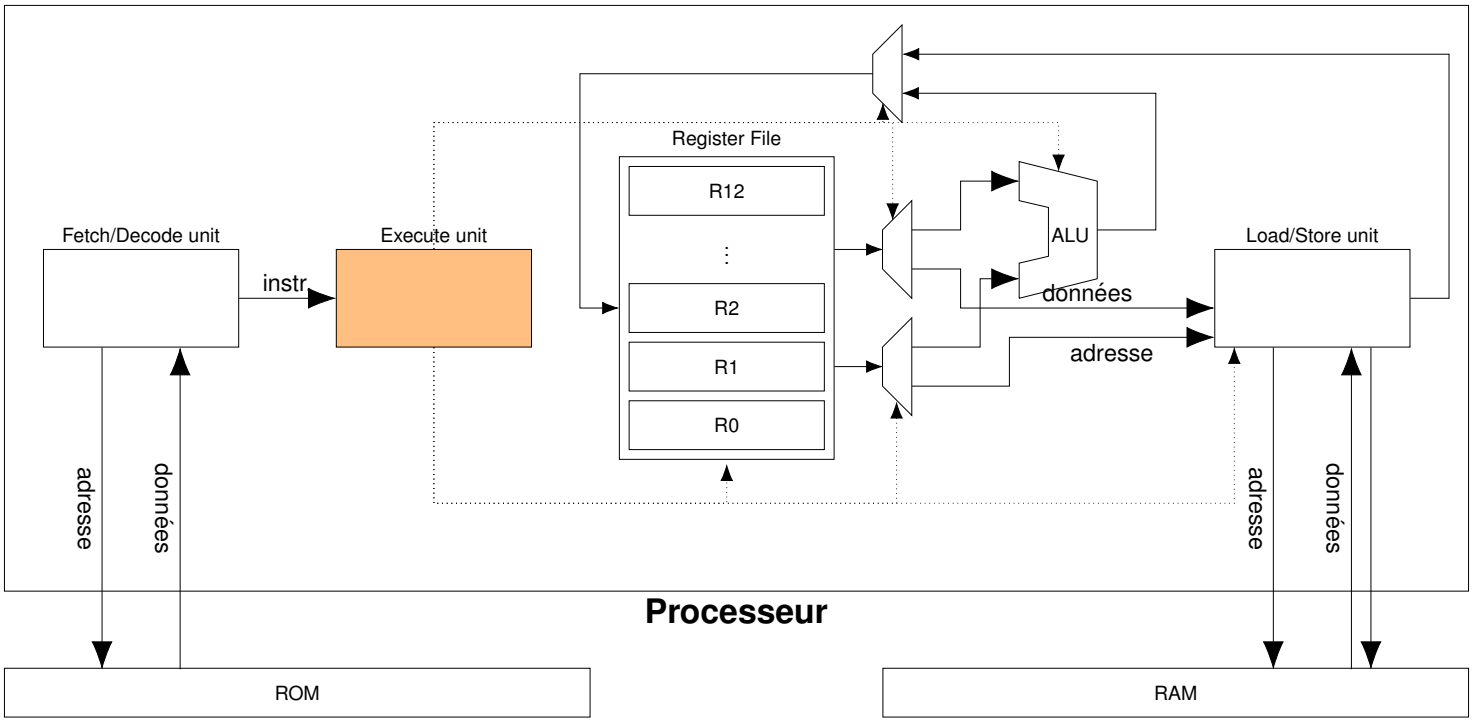
Etape 1 : Fetch

L'unité Fetch lit l'instruction suivante en mémoire programme.



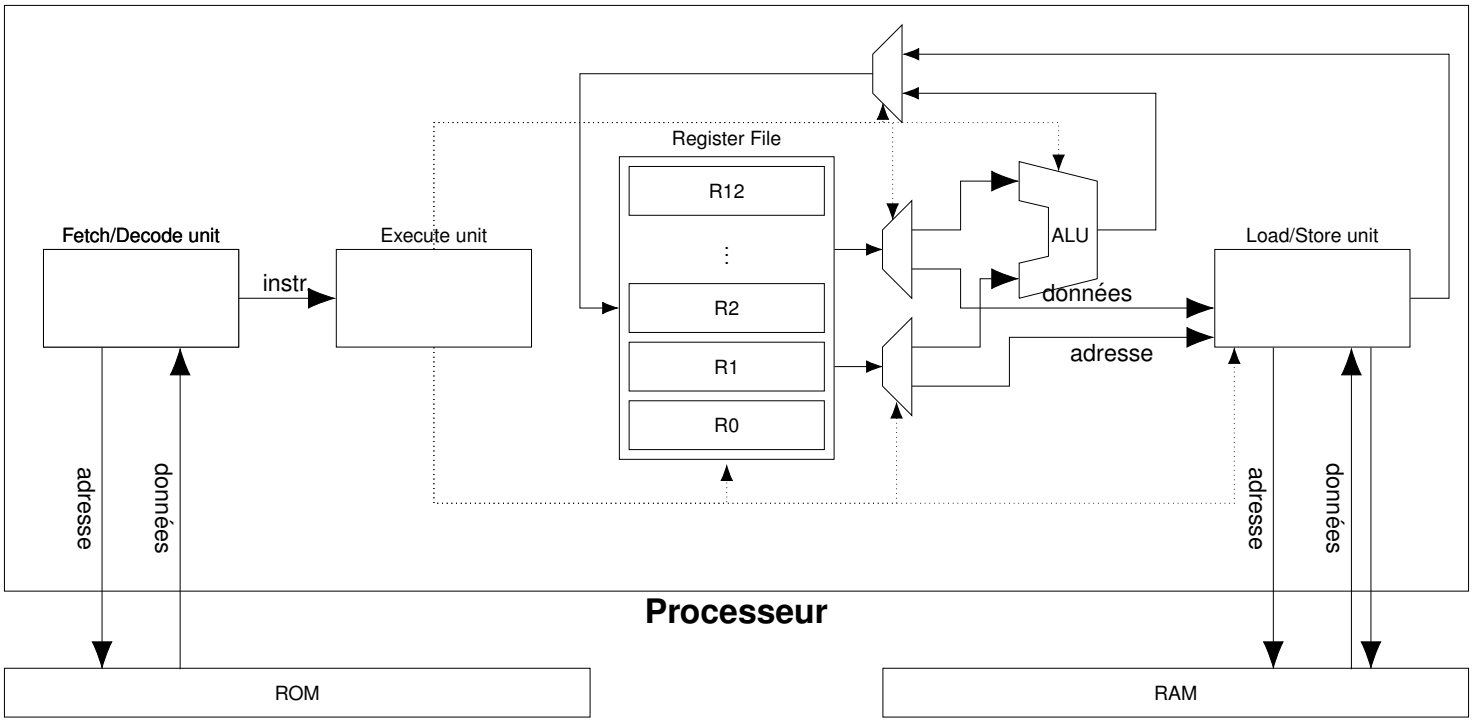
Etape 2 : Decode

L'unité Decode extrait de l'instruction l'opération à exécuter, les opérandes et la destination.



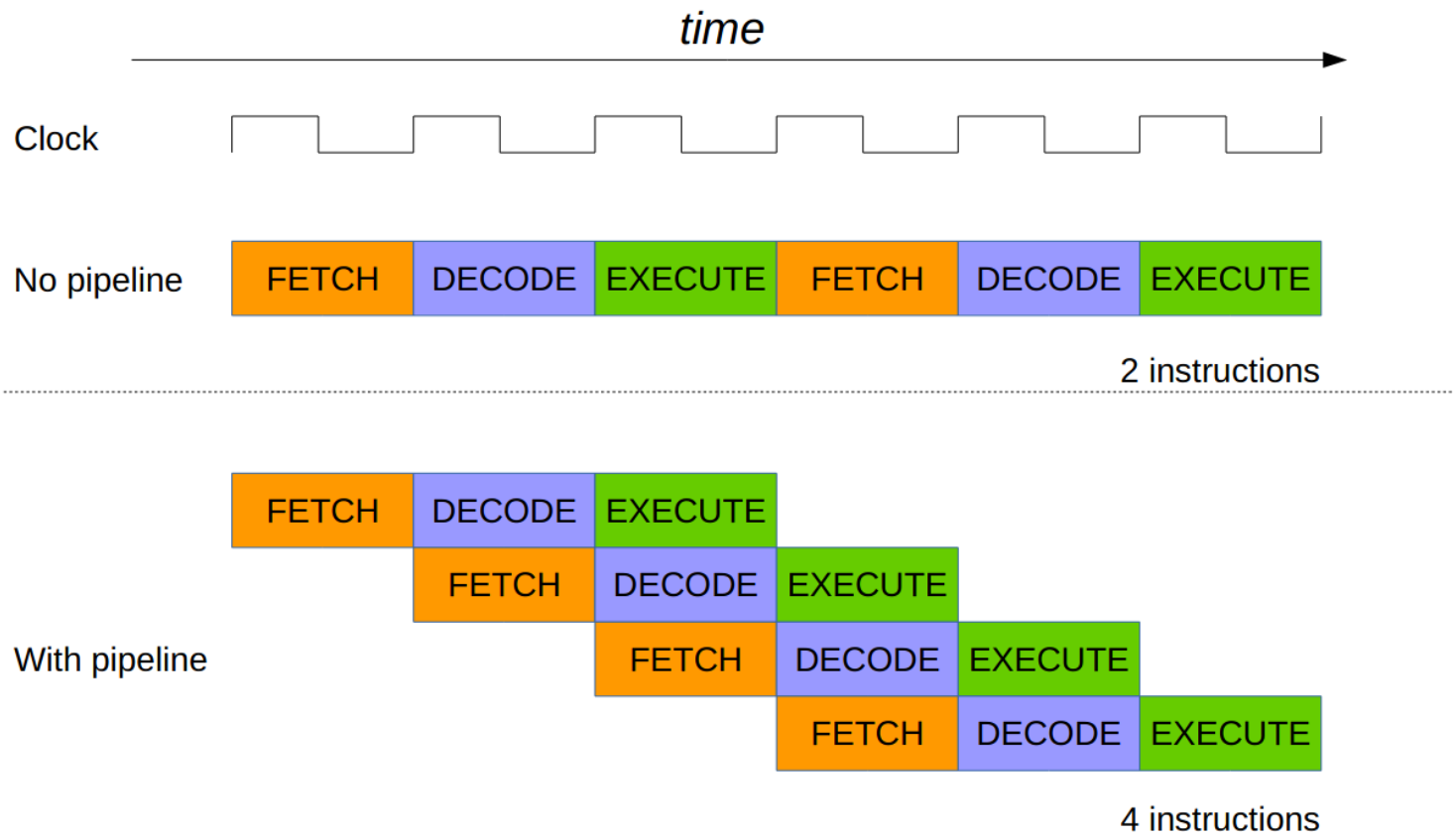
Etape 3 : Execute

L'unité Execute exécute l'instruction sur 1 ou plusieurs cycles.



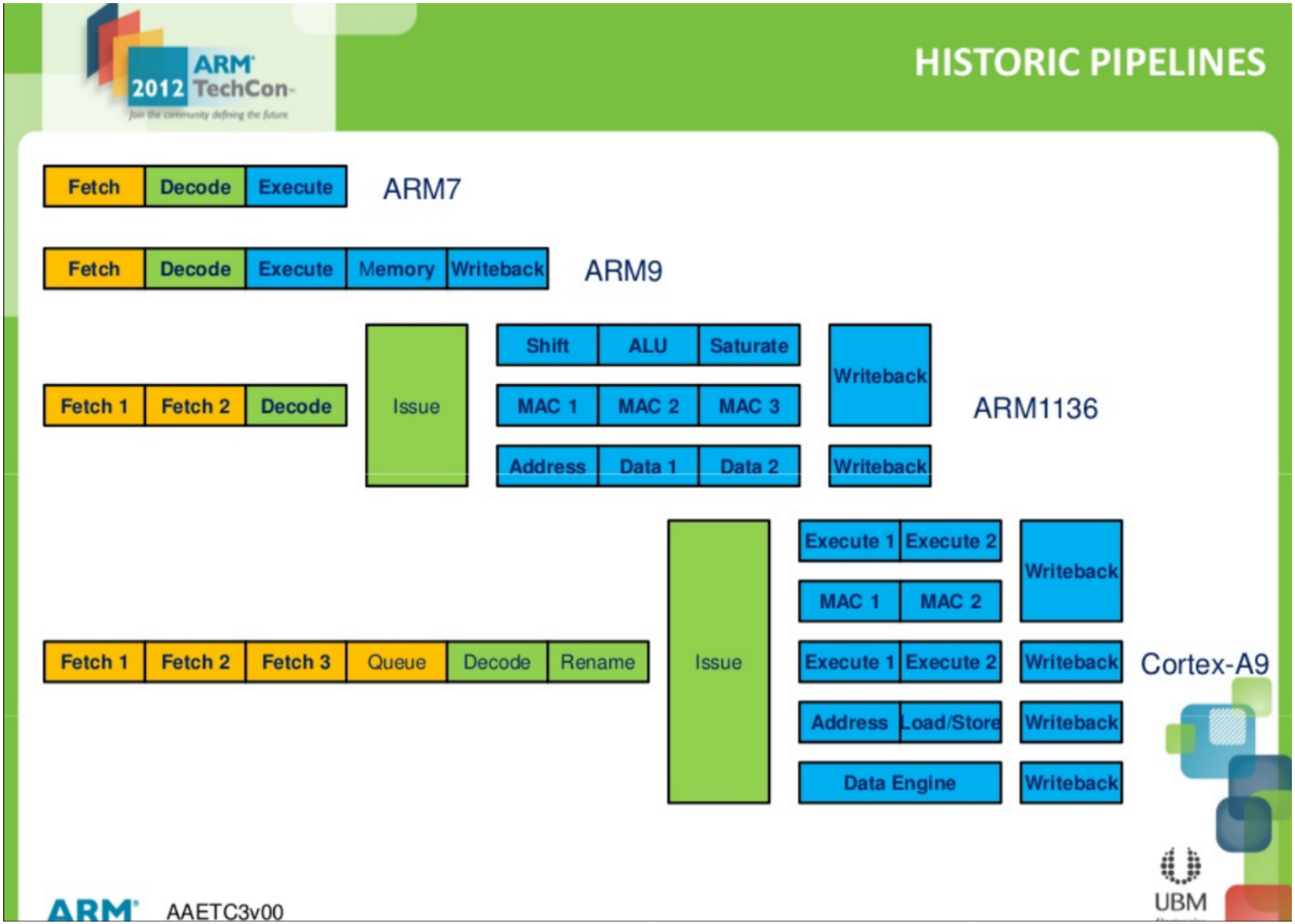
Définition du pipeline

Les opérations de Fetch, Decode et Execute nécéssitant des composants différents, elles sont parallelisables : Les opérations sont pipelinables.



Intéret du pipeline

Dans cet exemple, l'intérêt du pipeline est immédiat : il permet d'exécuter bien plus d'instructions pour un même nombre de cycle.


2012 ARM TechCon
Join the community defining the future
HISTORIC PIPELINES
Fetch
Decode
Execute
ARM7
Fetch
Decode
Execute
Memory
Writeback
ARM9
Fetch 1
Fetch 2
Decode
Issue
Shift
ALU
Saturate
MAC 1
MAC 2
MAC 3
Address
Data 1
Data 2
Writeback
Writeback
Writeback
ARM1136
Fetch 1
Fetch 2
Fetch 3
Queue
Decode
Rename
Issue
Execute 1
Execute 2
MAC 1
MAC 2
Execute 1
Execute 2
Address
Load/Store
Data Engine
Writeback
Writeback
Writeback
Writeback
Cortex-A9
ARM
AAETC3v00
UBM

La pile

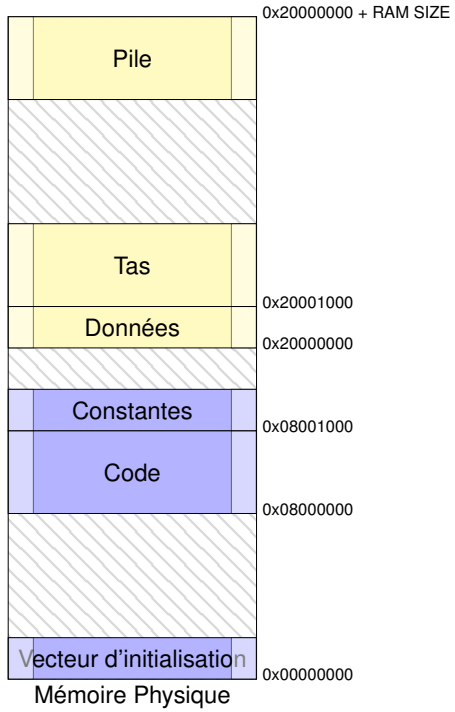
Le besoin

Le banc de registres du processeur ayant une taille limitée, il peut rapidement être saturé pendant l'exécution d'un programme. Il faut donc mettre en place une méthode simple pour le processeur pour libérer de la place.

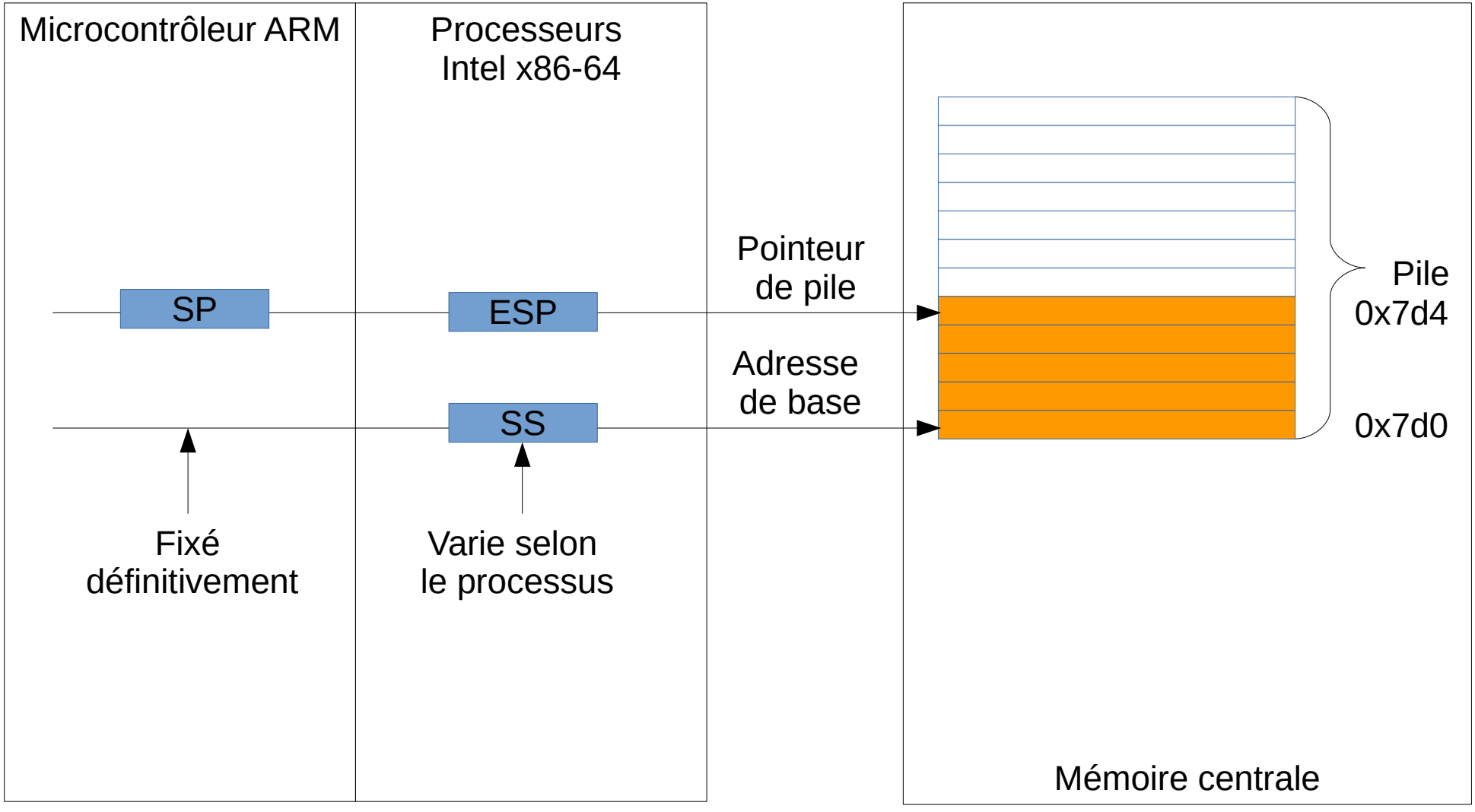
Définition

La pile est un espace mémoire de la RAM permettant au processeur de protéger temporairement la valeur de certains registres pour libérer de la place.

- Elle fonctionne en LIFO : Last In, First Out.
- Elle est en principe descendante : l'ajout d'une valeur à la pile la fait croître vers le bas en termes d'adresse (à l'inverse du tas).

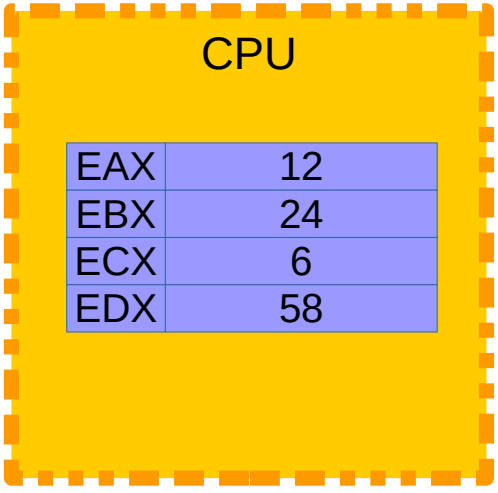


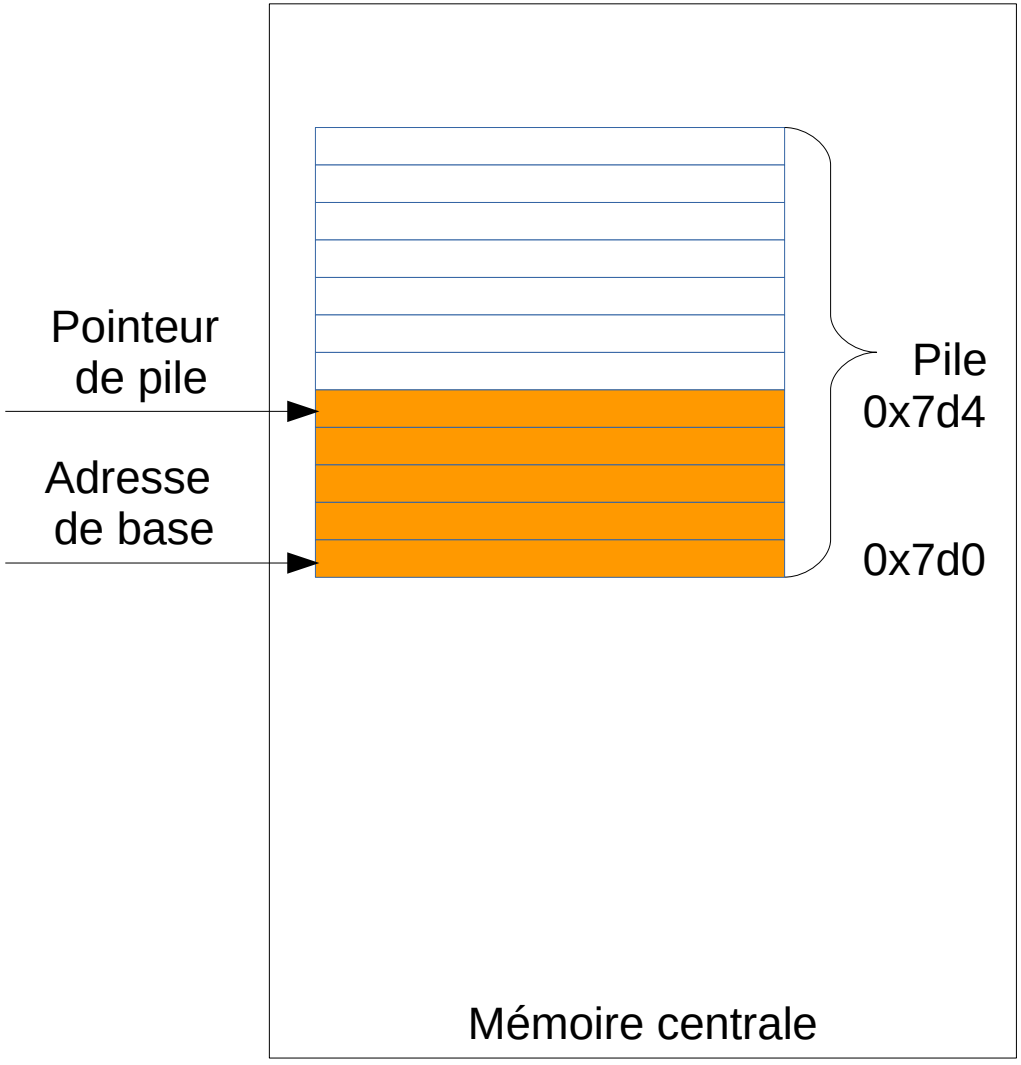
La pile


Microcontrôleur ARM
Processeurs
Intel x86-64
SP
ESP
SS
Pointeur
de pile
Adresse
de base
Fixé
définitivement
Varie selon
le processus
Pile
0x7d4
0x7d0
Mémoire centrale

La pile

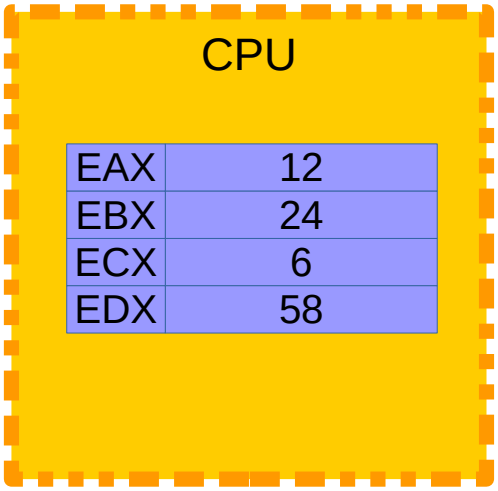
- Exemple :
  - On suppose que les registres du processeur sont tous occupés par une valeur.
  - Hors on a besoin de faire une opération entre la valeur du registre EAX et une donnée en mémoire (disons 34).

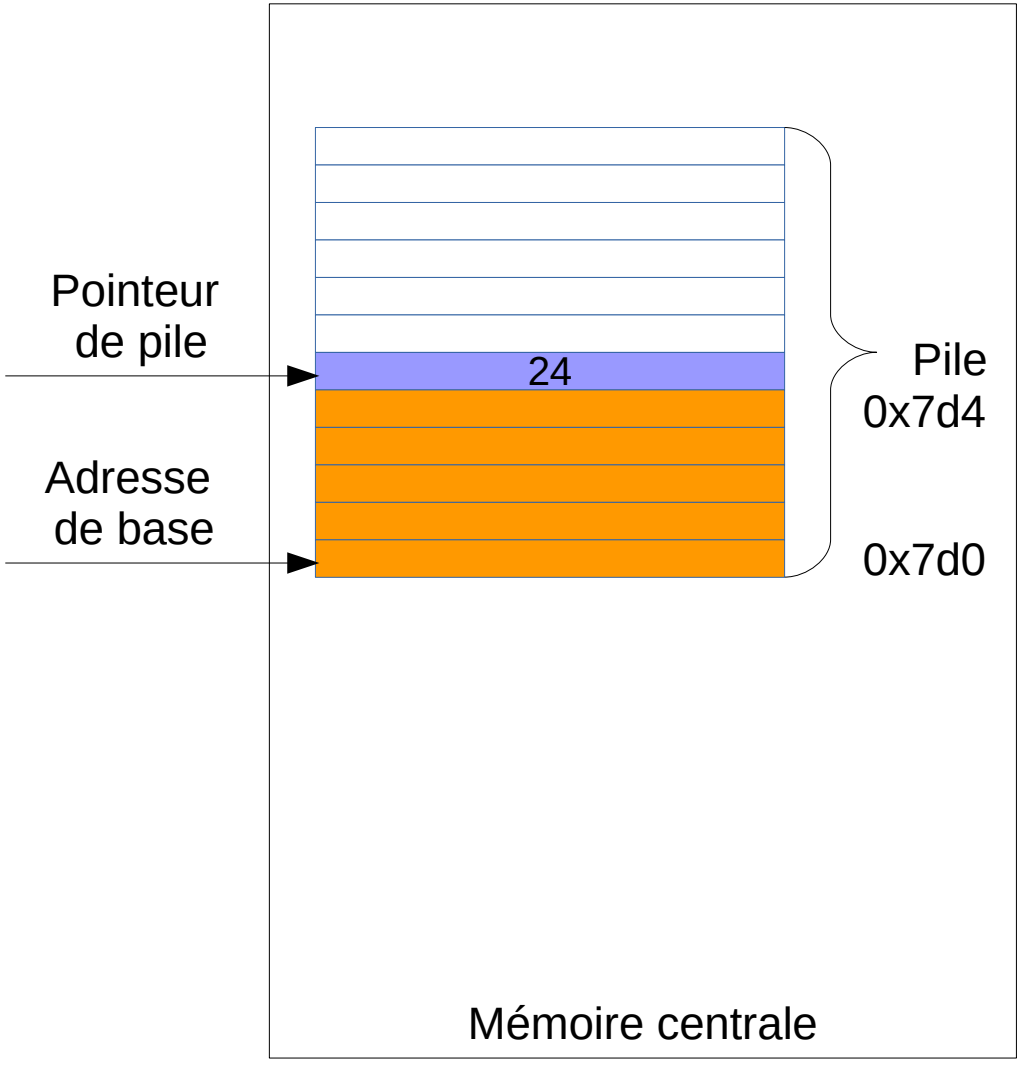




La pile

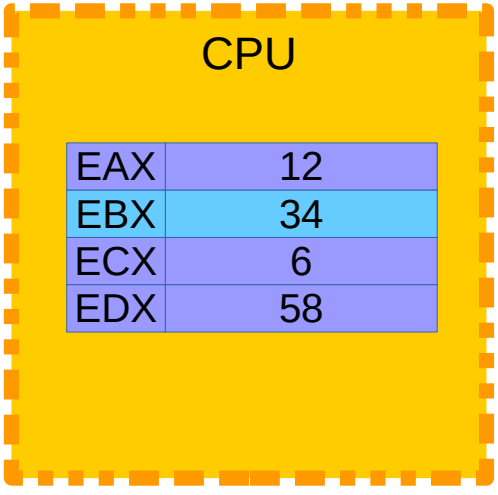
- On empile EBX car nous n'en n'avons pas besoin dans l'immédiat.
- EBX est donc sauvegardé en mémoire centrale.
- PUSH {EBX}

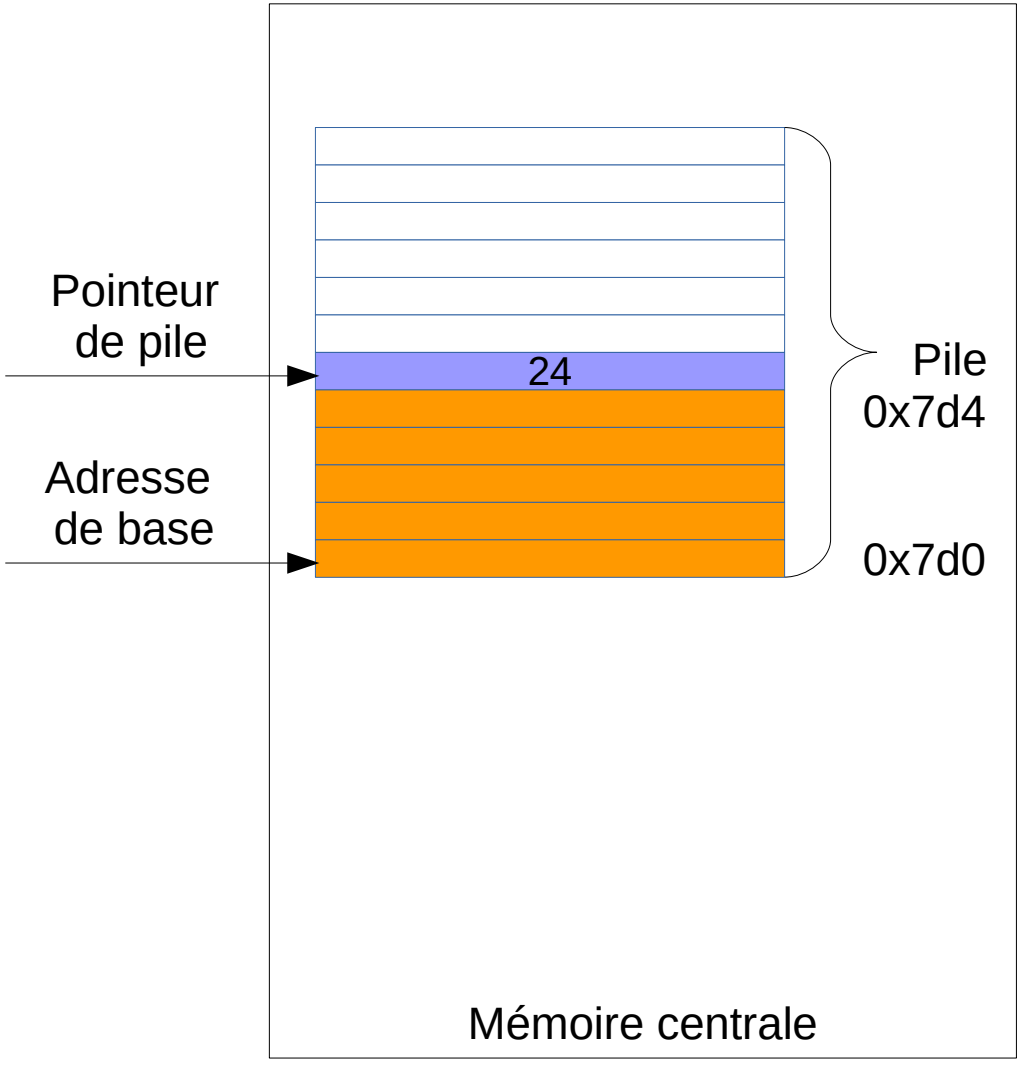




La pile

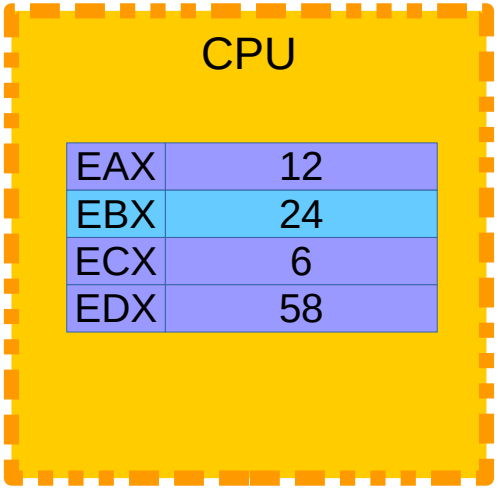
- On copie notre valeur 34 dans le registre EBX.
- On fait l'opération souhaitée avec le registre EAX.

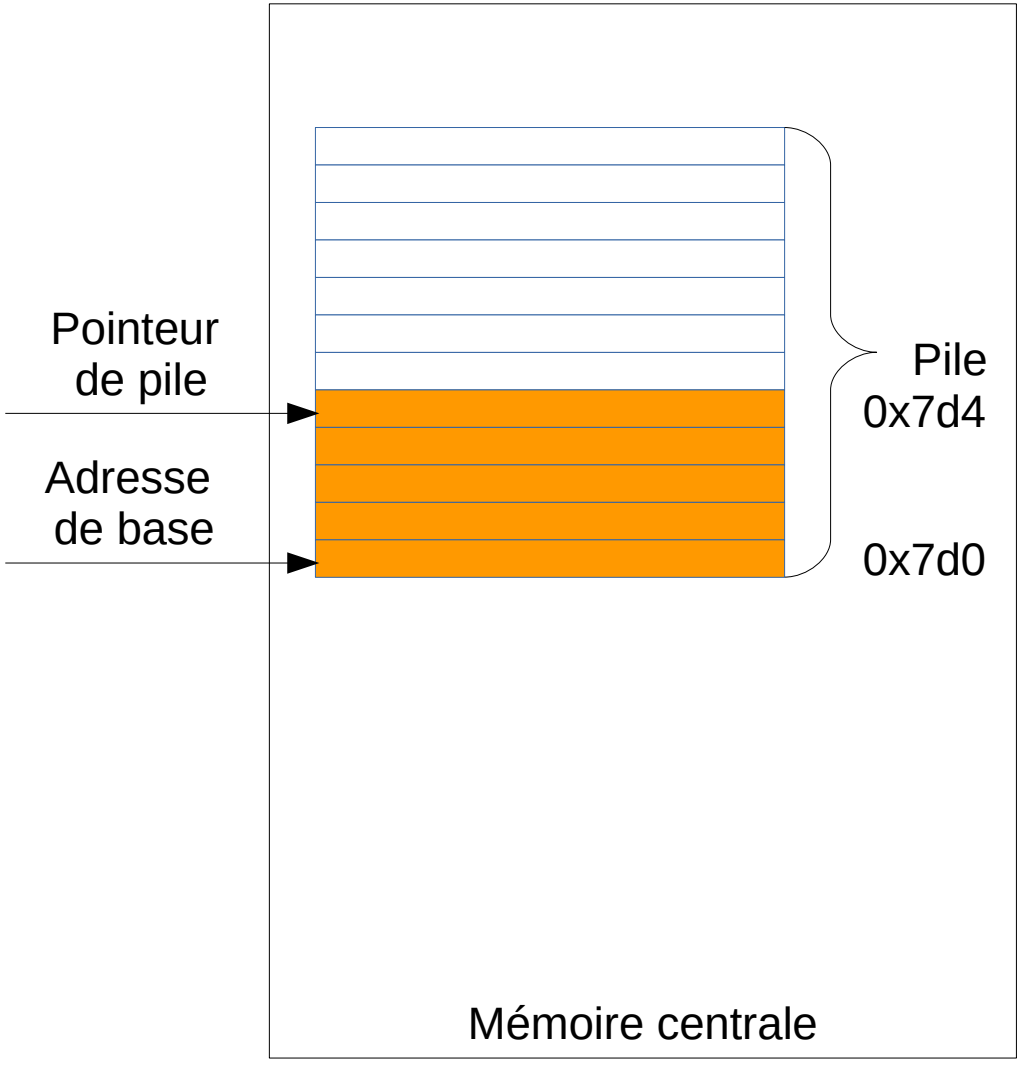




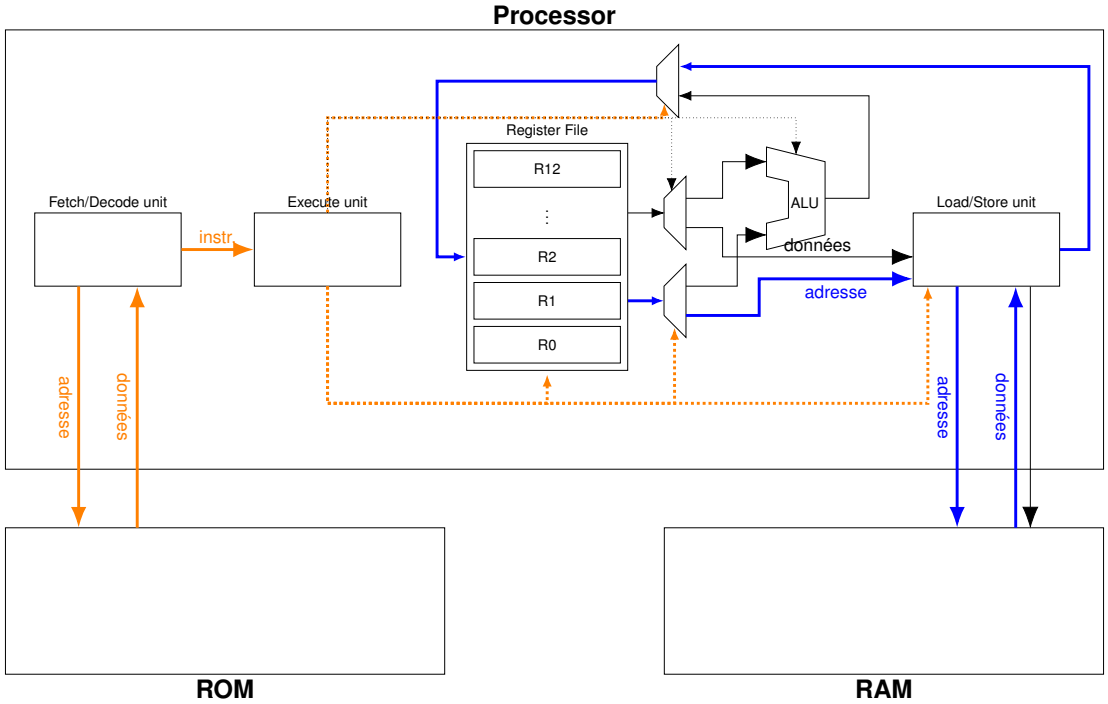
La pile

- Cette valeur 34, nous n'en avons plus besoin, nous pouvons dépiler dans le registre EBX.
- POP {EBX}





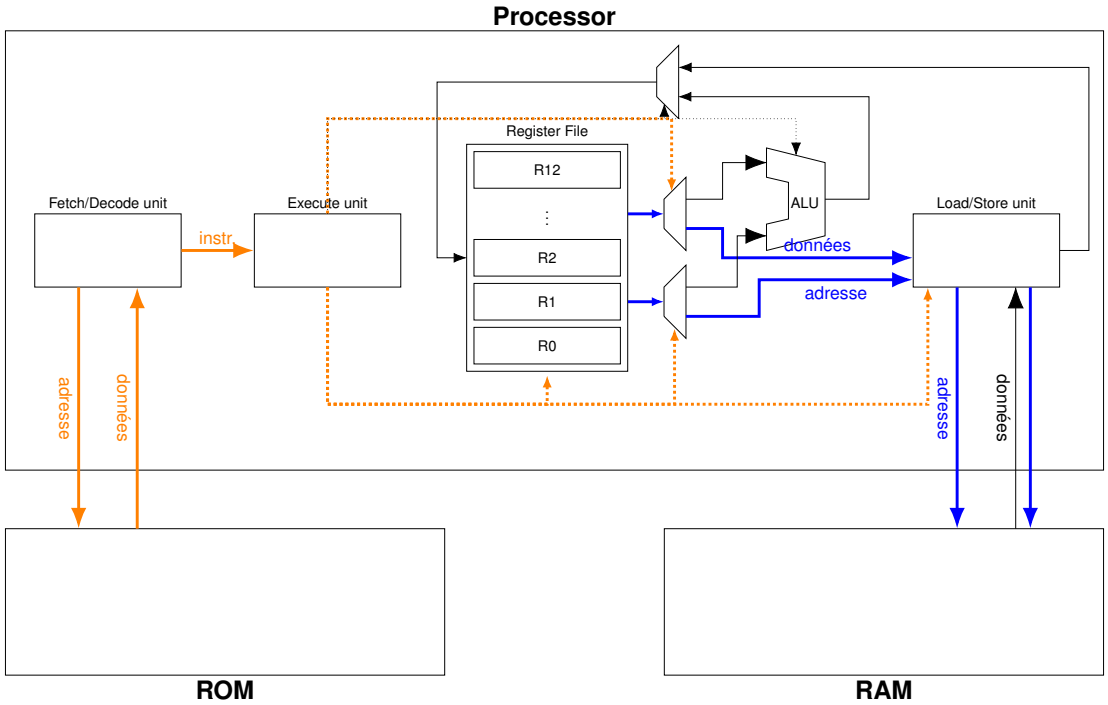
Instructions classiques



Opération de lecture : LDR (load register)

- Format : LDR Rt,[Rn,imm5]
- Copie dans le registre Rt la valeur à l'adresse Rn + imm5.
- Format binaire :

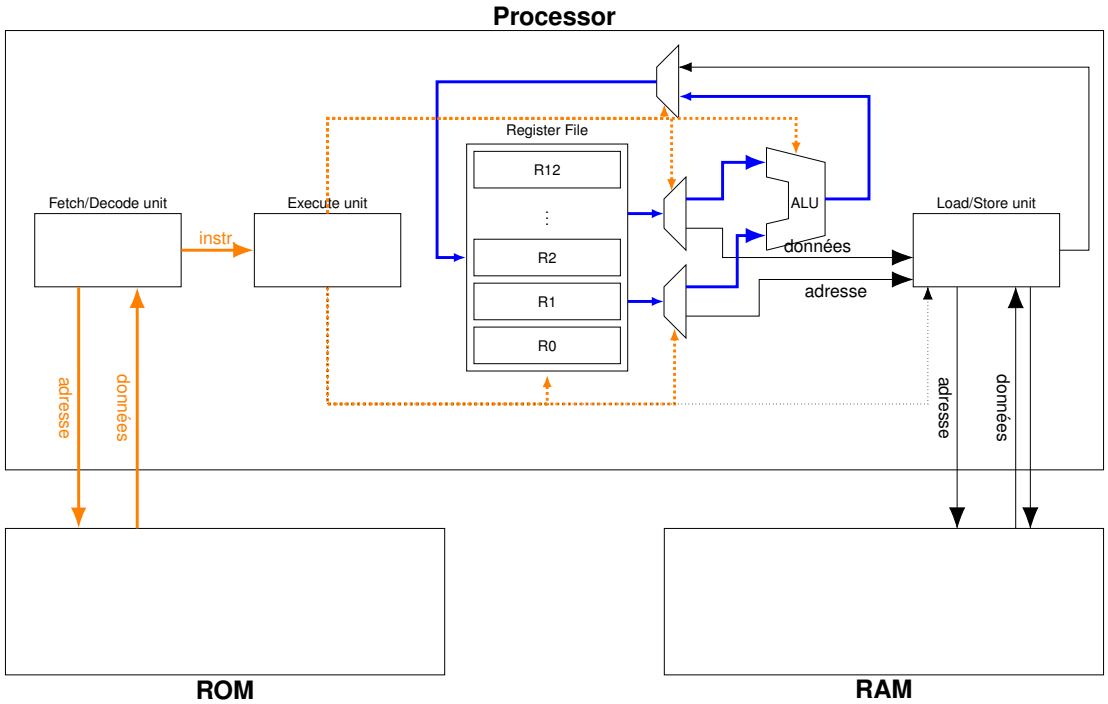
15	14	13	12	11	10 9 8 7 6	5 4 3	2 1 0
0	1	1	0	1	imm5	Rn	Rt



Opération d'écriture : STR (store register)

- Format : STR Rt,[Rn,imm5]
- Copie la valeur du registre Rt à l'adresse Rn + imm5.
- Format binaire :

15	14	13	12	11	10 9 8 7 6	5 4 3	2 1 0
0	1	1	0	0	imm5	Rn	Rt



Opération d'addition : ADD

- Format : ADD Rd, Rn, Rm
- Additionne la valeur de Rn et Rm et copie le résultat dans Rd..
- Format binaire :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	0	Rm			Rn			Rd		

Branchement conditionnel

Branchement conditionnel : Exemple du Cortex-M

```
1: int distance(int a, int b) {
2:     int res = 0;
3:     if(a < b) {
4:         res = b - a;
5:     }else{
6:         res = a - b;
7:     }
8:     return res;
9: }
```

```
1: CMP R0,R1
2: BGT else
3: SUB R0,R1,R0
4: B   return
   else:
5: SUB R0,R0,R1
   return:
6: BX LR
```

La fonction étudiée est la fonction distance : elle prend comme arguments deux entiers et renvoie la valeur absolue de la différence. Un branchement est utilisé ici pour calculer la valeur absolue.

Etude de la traduction de la fonction en instructions (à droite)

Dans l'appel de fonction du Cortex-M, les arguments sont passés à la fonction par registre (de R0 à R3). Ici *a* sera stockée dans R0 et *b* dans R1. La valeur de retour est stockée dans R0. L'opération CMP permet de comparer deux entiers, et est utilisée le plus souvent avant une opération de branchement conditionnel. BGT (Branch Greater Than) est l'opération de branchement, et permet de sauter au label *else* (ligne 5) si R0 $>$ R1. B (Branch) est un saut inconditionnel, et permet de sauter au label *return* (ligne 7) pour ne pas exécuter les instructions associées au else. En conclusion, si a $>$ b, on calcule R1 - R0 (i.e *b* - *a*) et on stocke le résultat dans R0, sinon R0 - R1 (i.e. *a* - *b*). BX LR est l'instruction de retour de fonction.

Original :

```
1: CMP R0,R1
2: BGT else
3: SUB R0,R1,R0
4: B   return
   else:
5: SUB R0,R0,R1
   return:
6: BX LR
```

Variante :

```
1: CMP R0,R1
2: BGT else
3: SUB R0,R1,R0
4: BX LR
   else:
5: SUB R0,R0,R1
6: BX LR
```

Variante 1 : retour prématuré

Pour éviter de faire un branchement à la fin du *if* pour ne pas exécuter la branche *else*, il est tout à fait possible de positionner un retour de fonction à la fin du bloc *if* (ligne 4 de la variante).

Original :

```
1: CMP R0,R1
2: BGT else
3: SUB R0,R1,R0
4: B   return
else:
5: SUB R0,R0,R1
return:
6: BX LR
```

Variante :

```
1: SUB R0,R1,R0
2: CMP R0,R1
3: BLT return
4: SUB R0,R0,R1
   return:
5: BX LR
```

Variante 2 : opérations anticipées

Une deuxième possibilité est d'anticiper en calculant la branch if dans tous les cas, puis si $a > b$, écraser le résultat calculé avec la bonne valeur. Ici, BLT (Branch Less Than) est la version de BGT mais pour le cas à R0 $<$ R1. Cette solution permet d'avoir un code plus compacte (5 instructions).