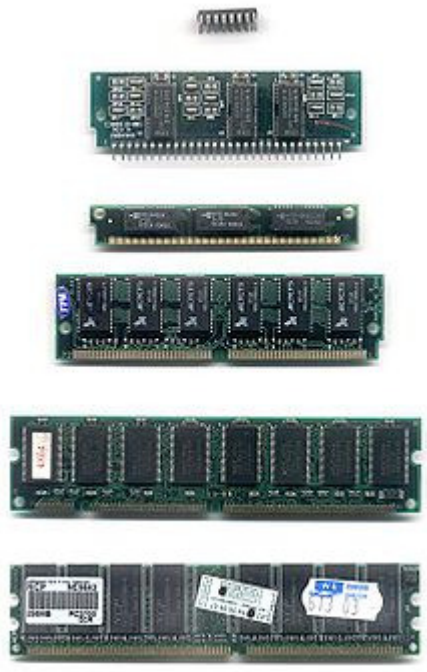
Organisation de la mémoire principale modes d’adressage et accès aux données

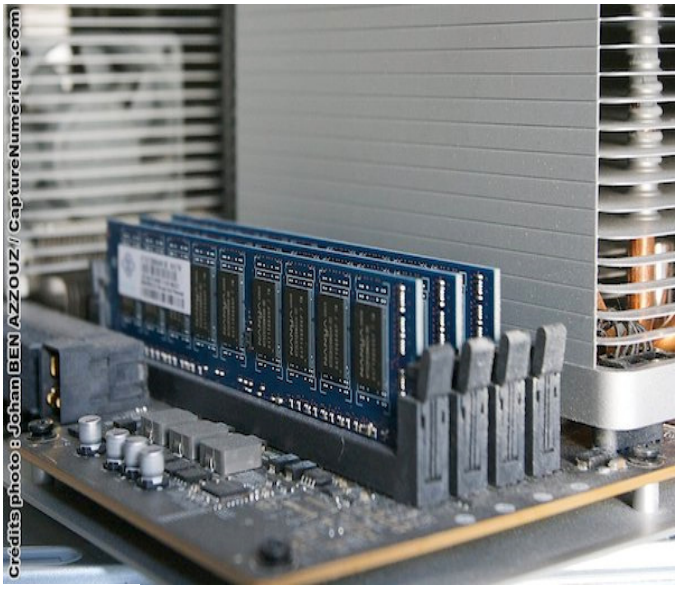
Organisation de la mémoire principale, modes d'adressage et accès aux données

Consigne :

1. **Identifier les notions/définitions importantes, être capable de les expliquer.**
2. **Refaire les exemples illustrant ces notions.**
3. **les appliquer en faisant les exercices notés exercice à faire.**

Contexte : Mémoire principale et Microprocesseur : rangement de données en mémoire principale. Comment les données sont rangées ? Comment elles sont identifiées (localisées → adresse d'une donnée en mémoire) ? Comment on peut y accéder pour les lire, les modifier (les traiter → calculs) et les ranger (écrire) ?





Composants mémoire (image extraite d'internet)

Objectifs :

- Calculer en hexadécimal l'adresse mémoire d'une donnée rangée en mémoire sous la forme : un octet, un mot (2 octets), un mot long (4 octets)
- Expliquer le rangement en mémoire d'une structure de données sous la forme d'un vecteur, d'une matrice : tableau 1D, 2D
- Ecrire des séquences d'instructions pour accéder à des données : instruction de chargement (LOAD), instruction de rangement (STORE) en utilisant les 3 principaux modes d'adressage du microprocesseur ARM – Cortex M3 : indirect, indirect avec déplacement, indirect avec index
- Décrire le résultat obtenu après l'exécution de ces séquences d'accès à la mémoire.

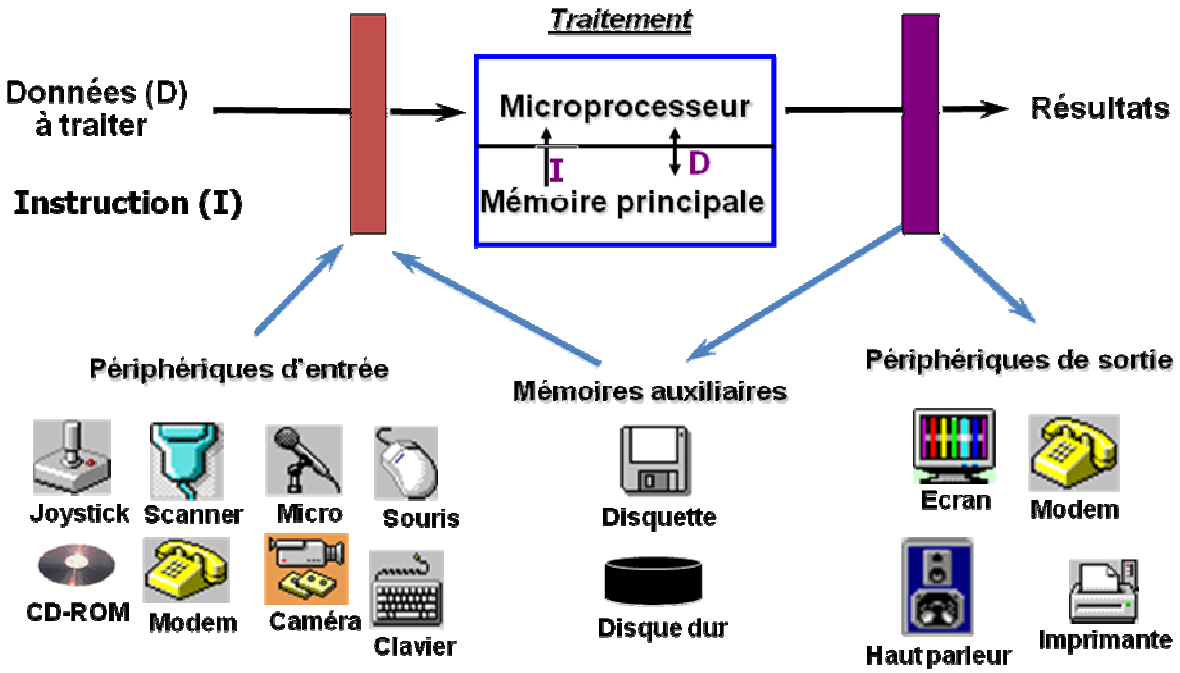


Schéma extrait d'internet

1. Architecture d'un ordinateur

On peut décrire l'architecture d'un ordinateur, c'est-à-dire son organisation, en se basant sur le modèle de Von Neumann. *John Von NEUMANN (1903 – 1957), mathématicien américain d'origine hongroise, a donné son nom à l'architecture utilisée dans la plupart des ordinateurs*.

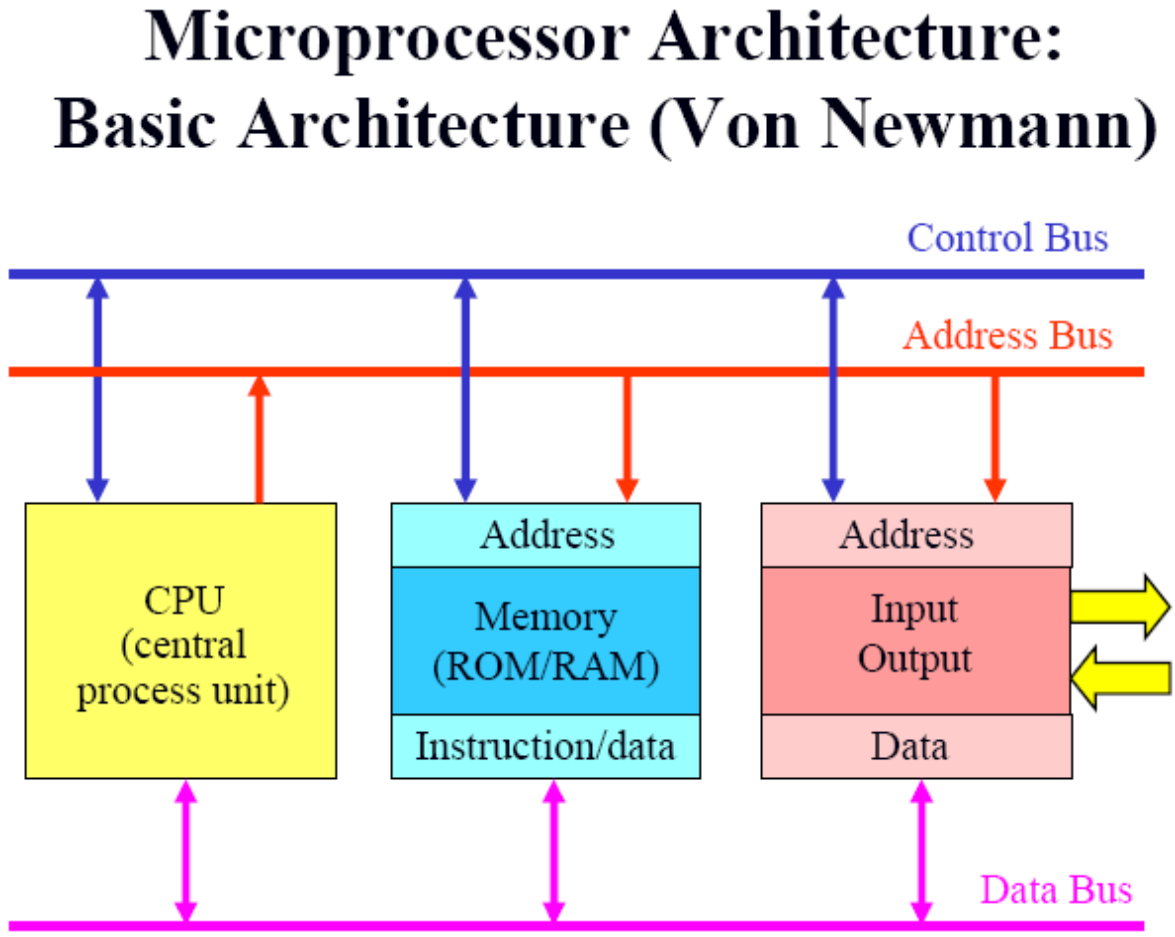


Schéma extrait d'internet

L'architecture d'un ordinateur basée sur ce modèle comprend :

- le **processeur** (CPU : Central Processing Unit)
- la **mémoire principale** : Read Only Memory (ROM), Random Access Memory (RAM)
- et les **entrées/sorties** : Input – Output

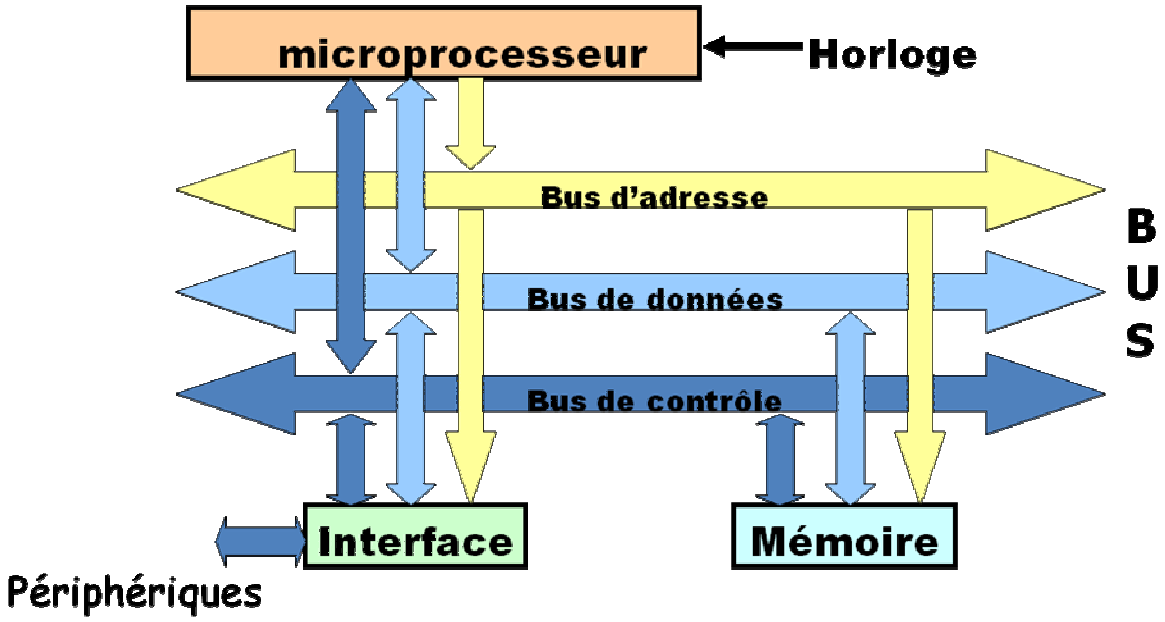


Schéma extrait d'internet

La communication microprocesseur - mémoire est assurée via 3 bus :

- **le bus de données** : bus **bidirectionnel** pour l'échange d'informations (instructions, Données) entre le microprocesseur ←→ et la mémoire,
- **le bus d'adresse** : bus **unidirectionnel** pour adresser la mémoire, il indique l'adresse (le « Numéro ») de la case mémoire à laquelle le microprocesseur veut accéder (accès soit en lecture soit en écriture),
- **le bus de contrôle** : il comprend différents signaux de contrôle dont le signal de lecture/écriture appelé R/$\overline{W}$ pour Read/Write. Si R/$\overline{W}$ = 0 le microprocesseur demande une écriture, sinon une lecture.

La taille en bits de ces bus est de :

- m bits pour le bus d'adresse,
- n bits pour le bus de données
- Et α bits pour le bus de contrôle.

M, n et α dépendent du type de microprocesseur utilisé.

L'espace mémoire adressable est fixé par m (la taille en bits du bus d'adresse – le bus d'adresse comprend m fils = m signaux, ces signaux sont notés : a_{m-1}, a_{m-2}, ...a_0. Le bit de poids faible de l'adresse est a_0) **et il est égal à 2^m éléments, les adresses sont comprises dans l'intervalle [0, …, 2^m – 1].** De l'élément d'adresse 0, jusqu'à l'élément d'adresse 2^m – 1.

Exemple :

Si m = 16 bits, le bus d'adresse comprend 16 lignes (16 fils) notées : a_0 à a_{15}, a_0 est le bit de poids faible.

La taille en octets de l'espace adressable est 2^{16} octets soit 64 kilo octets, c'est-à-dire 2^6 x 2^{10} (1 kilo = 1024 = 2^{10}).

Adresse sur 16 bits en Hexadécimal	**Cases mémoire**
0x0000	1er élément
...	.....
0x1000	….
…..	….
0xFFFF	Dernier élément

Remarque importante : une adresse identifie l'emplacement en mémoire d'un élément codé sur 8 bits (soit un octet).

A ce stade vous êtes capable de :

- Expliquer l'organisation de l'architecture de base d'un système à base de microprocesseur
- Expliquer le rôle et le fonctionnement de principe des éléments suivants : microprocesseur, mémoire principale
- Expliquer les rôles : bus de données, bus d'adresse, signal R/$\overline{W}$
- Trouver la taille de l'espace adressable par un microprocesseur à partir de la taille en bits de son bus d'adresse
- Calculer les adresses en hexadécimal de cet espace : adresse de début et adresse de fin

2. Structures de données de type tableau et implantation en mémoire

Les données traitées par le microprocesseur sont stockées temporairement en mémoire principale.

Dans ce qui suit, on s'intéresse uniquement au type élémentaire tableau.

2.1. Tableau à une dimension (1D)

Un tableau à une dimension est une liste contiguë d'éléments de même type.

Chaque élément est localisé par son indice i ou son adresse.

Exemple : on peut illustrer un tableau T de taille n de la façon suivante :

				T(i)					
0	1			*i*					*n-1*

Ce tableau est implémenté sous la forme de n emplacements consécutifs en mémoire principale.

La zone mémoire attribuée à un tableau est définie par :

- L'adresse de début d'implantation du tableau
- La taille des éléments stockés : un élément peut être de taille 8/16/32/64… bits
- Le nombre n des éléments stockés

Exemple :

Une chaîne de caractères «bonjour» peut être implantée sous la forme d'un tableau de caractères. Chaque élément du tableau représente un caractère codé en ASCII et prend un emplacement en mémoire, c'est-à-dire est stocké sous la forme d'un octet. Si ce tableau commence à l'adresse 0x1000 (adresse en hexadécimal), la taille de ce tableau est n = 7 (la chaîne contient 7 caractères).

Adresse sur 16 bits codée en Hexadécimal	**Cases mémoire**
0x0000	
…..	
….	

0x1000	1e élément de T = Code ASCII de b = **0x62**
0x1001	2e élément de T = Code ASCII de o = **0x6F**
0x1002	3e élément de T = Code ASCII de n = **0x6E**
0x1003	4e élément de T = Code ASCII de j = **0x6A**
0x1004	5e élément de T = Code ASCII de o = **0x6F**
0x1005	6e élément de T = Code ASCII de u = **0x75**
0x1006	Dernier élément de T = code ASCII de r = **0x72**
……	….

2.2 Tableau à deux dimensions (2D)

Un tableau 2D est implanté de façon contiguë en mémoire : ligne par ligne ou colonne par colonne. Soit un tableau de taille m x n, m étant le nombre de lignes et n le nombre de colonnes.

E_{11}	**--**	**--**	**--**	**--**	**E_{1n}**
E_{m1}					**E_{mn}**

L'implantation ligne/ligne de ce tableau est représentée ci-dessous :

E_{11}	--	--	E_{1n}	E_{21}	--	E_{2n}	--	--	--	--	--	E_{m1}	--	--	E_{mn}
ligne 1				ligne 2								ligne m			

A ce stade vous êtes capable de :

- Expliquer comment sont rangés en mémoire les éléments d'un tableau 1D (vecteur) ou 2D (matrice)
- Identifier/donner l'adresse en hexadécimal de début et l'adresse de fin de ce tableau
- Identifier/donner l'adresse en hexadécimal d'un élément de ce tableau

3. Microprocesseur ARM Cortex – M3 : registres internes et organisation de la mémoire

Le microprocesseur Cortex – M3 est construit autour d'une **architecture de type « load – store »** : chargement/rangement en référence aux instructions de lecture (**L**oa**D**) et d'écriture (**ST**ore).

Il utilise des données de 32 bits et dispose d'un bus d'adresse de 32 bits.

Les données en mémoire principale sont d'abord chargées dans les **registres internes** du microprocesseur, lequel effectue les opérations sur ces registres et, par la suite, range le résultat (rangement temporaire dans des registres) en mémoire principale.

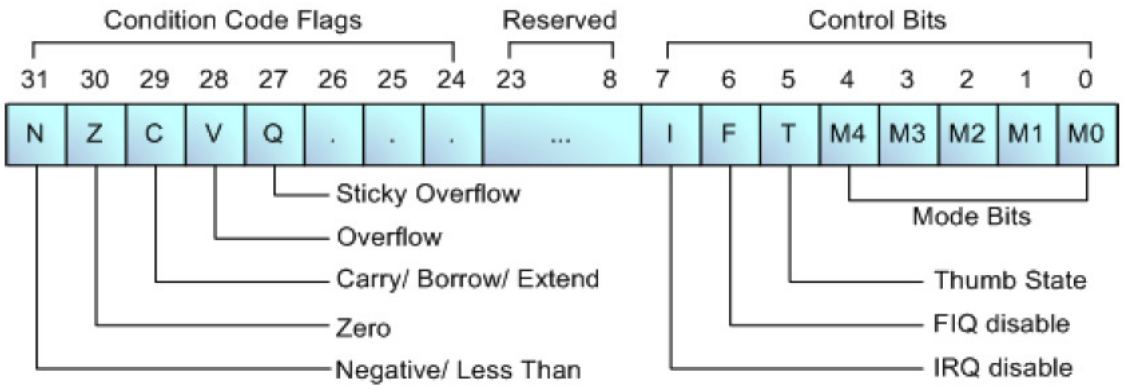
3.1. Les **registres** internes

Le microprocesseur cortex – M3 dispose de 17 registres (R0 à R15, et PSR).
La taille de chaque registre est de 32 bits.

- **13 registres à usage général : R0 à R12**. R0 à R7 sont dits « low registers » et R8 à R12 « high registers ».

- **Un compteur ordinal ou Program counter : R15** qui contient ***l'adresse de la mémoire de la prochaine instruction à exécuter***. Le bit 0 de ce registre est toujours à 0 (adresse paire). Comme les instructions sont codées sur 16 bits (cas : Thumb) ou 32 bits (Thumb2), elles occupent au moins 2 adresses consécutives.

- **Un registre d'état (Program Status regsiter) : *x*PSR** avec ***x*** pouvant être **A**=Application, **I**=Interrupt ou **E**=Execution. **Ce registre contient les indicateurs** (voir cours1)



Important : Les bits 28 à 31 de ce registre contiennent les indicateurs/drapeaux (Condition Code Flags) :

N	bit 31 – bit de poids fort de APSR	Recopie le bit de poids fort du résultat, si N = 1 alors le résultat <0
Z	bit 30 de APSR	Si Z = 1 alors le résultat = 0
C	bit 29 de APSR	Si C = 1 il y a un débordement de la représentation non signée : entiers naturels – bit appelé retenue (Carry)
V	bit 28 de APSR	Si V = 1 il y a un débordement de la représentation signée : entiers – bit appelé oVerflow

A ces registres, on ajoute 2 registres spéciaux : **R13 (registre pointeur de pile), R14 (Link register).**

Exemple : Opération d'addition et positionnement des différents indicateurs.

0x FFFF FFFF	0x 8000 0000	0x 7FFF FFFF
+ 1	+ 0x 8000 0000	+ 1
= 0x 0000 0000	= 0x 0000 0000	= 0x 8000 0000
C = 1 et Z = 1	**C = Z = V = 1**	**N = V = 1**

Interpréter les résultats obtenus dans l'exemple ci-dessus :

.....

.....

.....

.....

.....

Dans les cas où V = 1, montrer comment on peut obtenir un résultat correct (V=0) :

.....

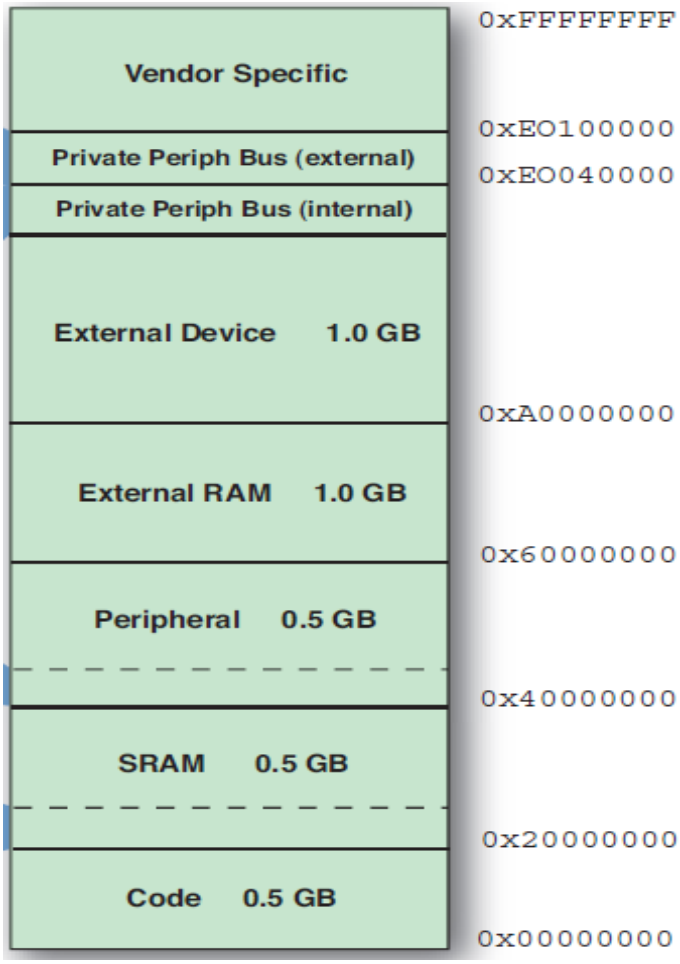
.....

.....

3.2. **Organisation mémoire** – et microprocesseur **ARM Cortex-M3**

L'espace mémoire du Cortex-M3 est de 4Go, soit 2^{32} adresses consécutives (bus d'adresse de 32 bits). Une adresse mémoire correspond à un octet (**toujours**).

Tous les accès à la mémoire se font via des instructions **LoaD** (instruction **LD**) ou **STore** (instruction **ST**).



Contient le code (le code du programme à exécuter par le microprocesseur

L'espace mémoire SRAM compris entre l'adresse 0x2000 0000 et 0x4000 0000 est accessible pour les données

A ce stade vous êtes capable de :

- Enumérer les principaux registres internes du microprocesseur ARM Cortex – M3
- Donner la taille en bits de chaque registre, ainsi que son rôle
- Donner la taille en octets de l'espace adressable par le microprocesseur ARM Cortex – M3
- Expliquer le rôle respectif de la mémoire de code et de la mémoire SRAM
- Donner l'adresse de début et de fin de la mémoire de code et de la mémoire SRAM

3.3. **Instructions** de base du microprocesseur **ARM Cortex – M3**

Soit la syntaxe suivante :

Code opération	**opérande 1, opérande 2, {opérande 3}**

Code opération = ce champ correspond à l'opération à effectuer

Opérande 1 = destination, contient le résultat de l'opération
Opérande 2 = donnée
Opérande 3 : peut être optionnel

L'opérande 1 est la destination, les opérandes 2 et 3 sont les opérandes sources.

Exemple :

Cas d'un opérande immédiat : MOV R6, #0x55

MOV : est le code opératoire donné sous la forme d'un mnémonique : MOV : tranférer

R6 : opérande 1 = la destination = contient le résultat de l'opération

#0x55 est une constante (valeur **imm**édiate) : représente l'opérande 2.

L'exécution de cette instruction modifiera le contenu du registre R6 :
on a R6 ← 0x0000 0055. Les indicateurs N, Z, C et V resteront inchangés (ils garderont leurs états précédents)

Dans le cas de l'instruction : MOV R6, #0x55, l**e suffixe S** indique le positionnement des indicateurs (flags), dans cette exemple l'instruction MOVS ne positionne aucun indicateur à 1 donc on aura : N = Z = C = V = 0.

Remarque : l'opérande immédiat 0x55 fait partie du code de l'instruction. Comme le code d'une instruction peur être soit codé sur 16 ou 32 bits, la valeur de l'opérande immédiat (constante) est limitée par la taille du code de l'instruction.

3.4. Instructions d'accès à la mémoire : **Chargement – Mémoire (instruction LoaD)** : chargement d'un registre du microprocesseur par le contenu d'une case mémoire.

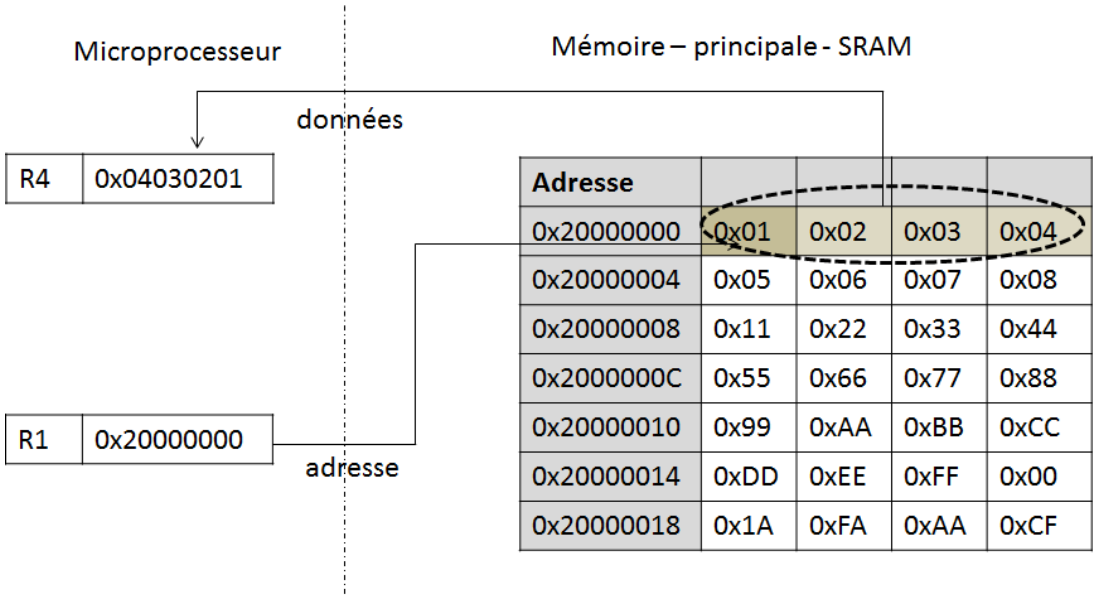
LDR R_k , [R_n] : R_k ← contenu de la mémoire dont l'adresse est donné par le registre R_n est chargé dans le registre R_k.

Illustration : On suppose que R1 = 0x2000 0000 (c'est le registre noté R_n), que la mémoire contient les données suivantes :

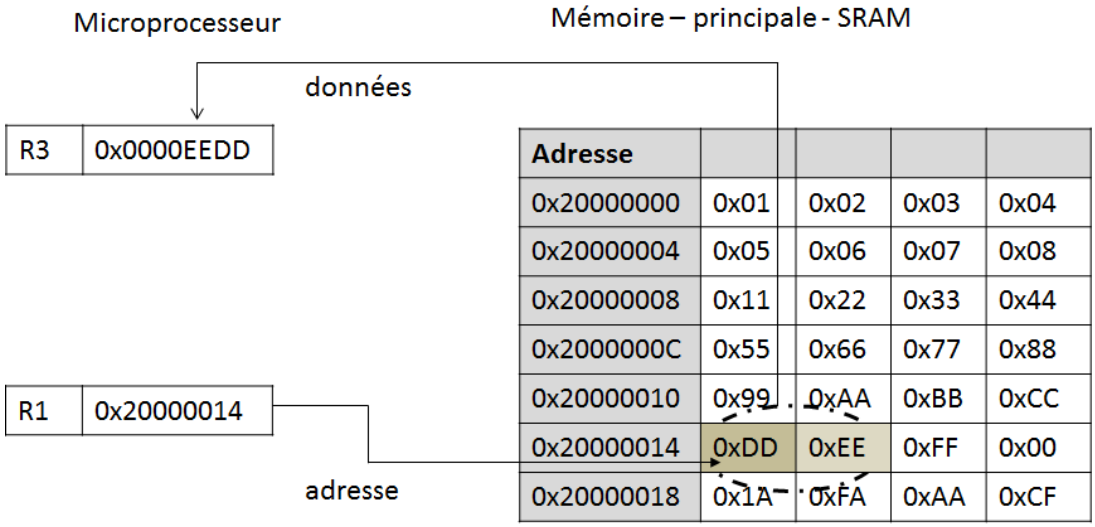
adresse				
0x2000 0000	0x11	0x22	0x33	0x44
0x2000 0004	0x55	0x66	0x77	0x55

L'exécution de l'instruction LDRR4,[R1] changera (transfert) d'un mot long (4 octets) de la mémoire dans le registre R4 (registre destination, note R_k). Le contenu de R4 = 0x44 33 22 11

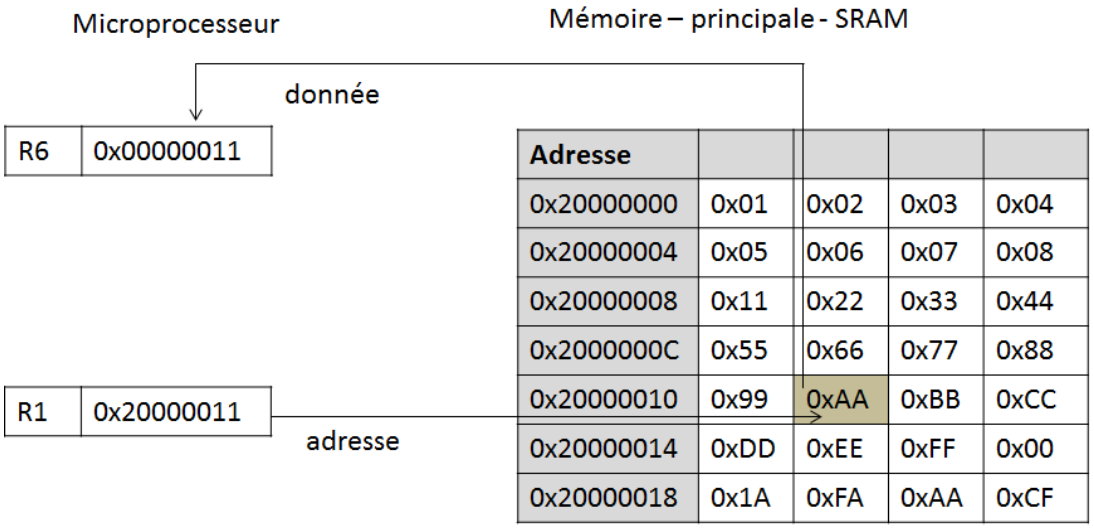
R4	Octet poids fort			Octet poids faible
	0x44	0x33	0x22	0x11



LDR R4,[R1]



LDRH R3,[R1]

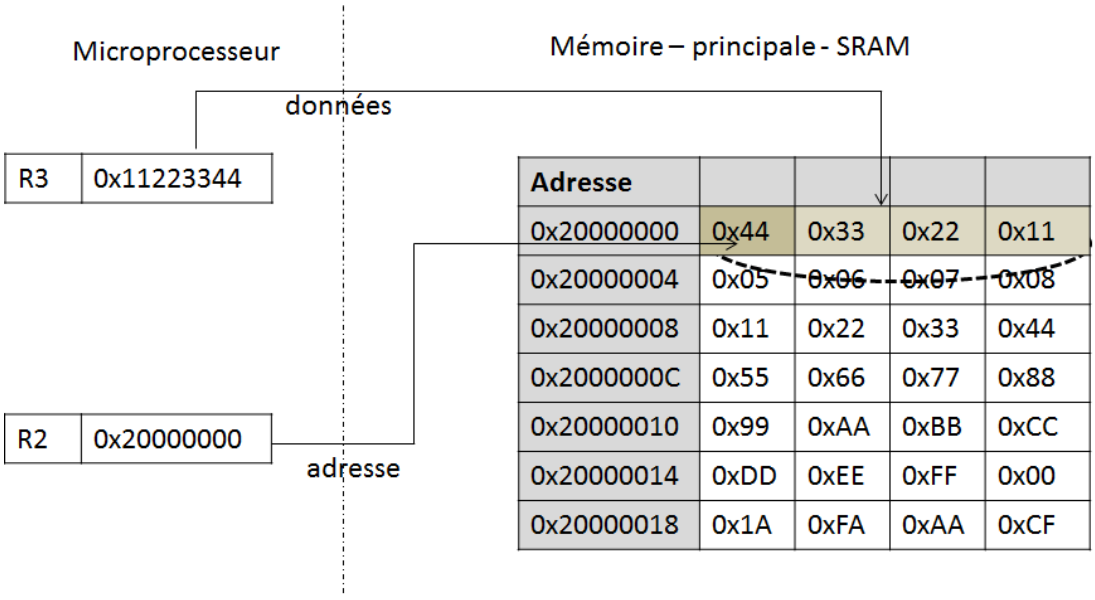


LDRB R6,[R1]

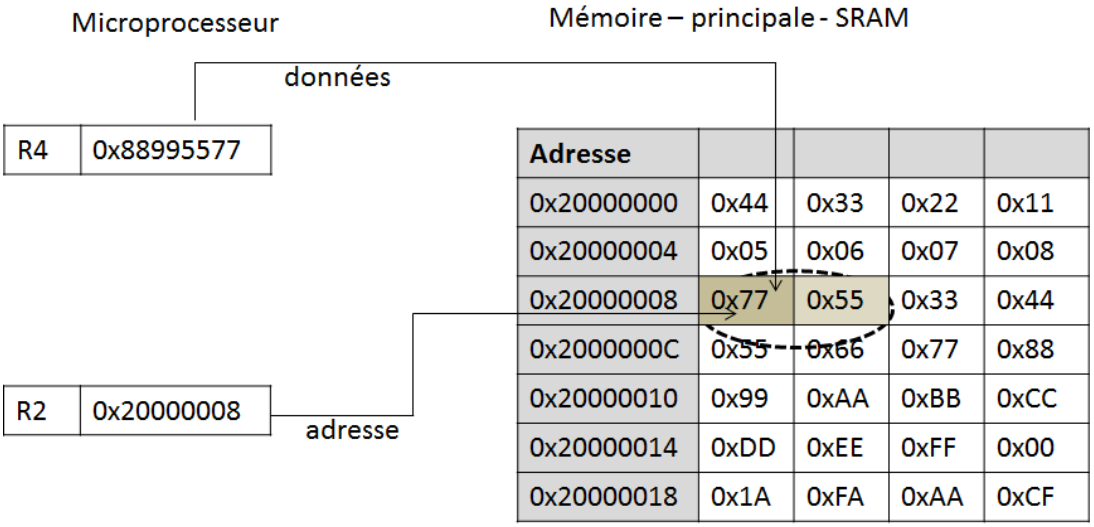
3.5. Instructions d'accès à la mémoire : **Rangement en mémoire (instruction STore)** : chargement d'un registre du microprocesseur par le contenu d'une case mémoire.

Syntaxe de l'instruction de rangement (Store**) : rangement d'un registre du microprocesseur dans une case mémoire**

STR **R_n , [adr_mem]** **:** le contenu du registre R_n → est rangé dans la mémoire à partir de l'adresse définie par [adr_mem].

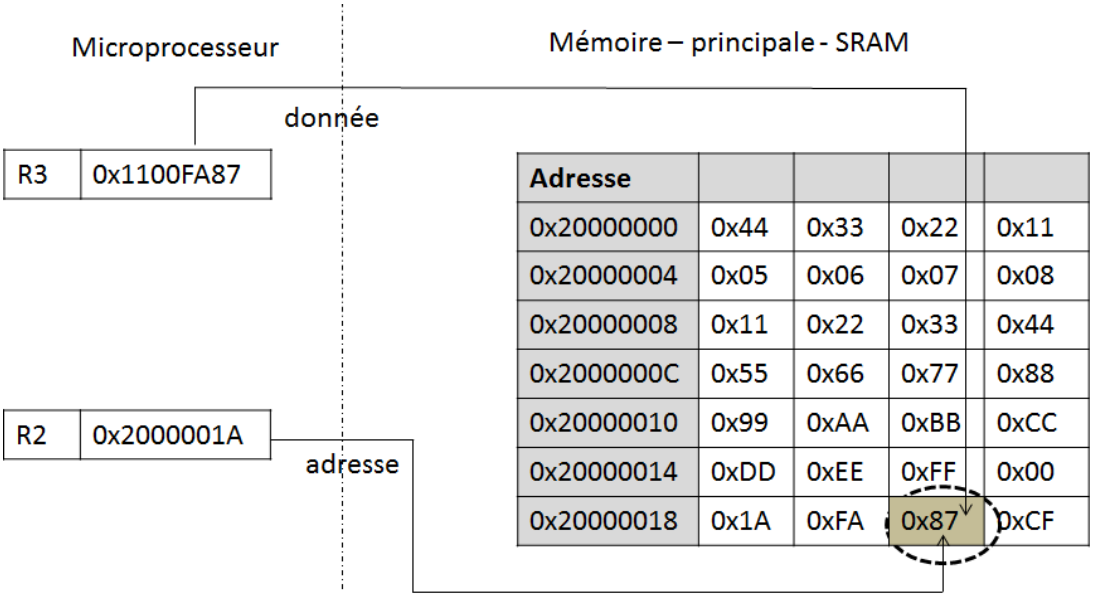


STR R3,[R2]



Adresse				
0x20000000	0x44	0x33	0x22	0x11
0x20000004	0x05	0x06	0x07	0x08
0x20000008	0x77	0x55	0x33	0x44
0x2000000C	0x55	0x66	0x77	0x88
0x20000010	0x99	0xAA	0xBB	0xCC
0x20000014	0xDD	0xEE	0xFF	0x00
0x20000018	0x1A	0xFA	0xAA	0xCF

STRH R4,[R2]



Adresse				
0x20000000	0x44	0x33	0x22	0x11
0x20000004	0x05	0x06	0x07	0x08
0x20000008	0x11	0x22	0x33	0x44
0x2000000C	0x55	0x66	0x77	0x88
0x20000010	0x99	0xAA	0xBB	0xCC
0x20000014	0xDD	0xEE	0xFF	0x00
0x20000018	0x1A	0xFA	0x87	0xCF

STRB R5,[R2]

A ce stade vous êtes capable de :

- Expliquer le rangement des données en mémoire se fait selon la norme little endian et illustrer ces explications par des exemples
- Ecrire des séquences d'instructions d'accès à la mémoire avec les instructions LDR, STR dans le cas de données codées sur : 1 octet, 2 octets, 4 octets
- Illustrer l'exécution de ces séquences par des schémas

3.6. modes d'adressage :

Le tableau ci-dessous donne **les 3 principaux modes d'adressage**. Ainsi, l'adresse de la mémoire, notée adr_mem est déterminée par :

- Le contenu d'un registre. Ce **mode est appelé mode d'adressage indirect**, il est noté [R_n].
- Le contenu d'un registre ± un constante. La valeur de la constante est appelée *offset*. C'est le **mode indirect avec déplacement**, noté [R_n, #$\pm offset$] .
- La somme de deux registres. C'est le **mode indirect avec index**, noté [R_n, R_m]. R_n contient l'adresse de base et R_m la valeur de l'index.

[adr_mem]	Mode d'adressage	Calcul de l'adresse de la mémoire
[R_n]	indirect	adr_mem = R_n
[R_n, #$\pm offset$]	Indirect avec déplacement	adr_mem = $R_n \pm offset$
[R_n, R_m]	Indirect avec index	adr_mem = R_n + R_m

A ce stade vous êtes capable de :

- Expliquer le rôle d'un mode d'adressage
- Expliquer les 3 modes d'adressage : indirect, indirect avec déplacement, indirect avec index
- Illustrer par des exemples l'utilisation de ces 3 modes d'adressage aussi bien dans le cas d'un chargement d'une donnée de la mémoire vers un registre interne que dans le cas d'un rangement d'une donnée contenu dans un registre dans la mémoire