POLYTECH
NICE-SOPHIA

Systèmes à Microprocesseurs

Cycle Ingénieur Troisième Année

Sébastien Bilavarn

Plan

- Ch1 – Représentation de l'information
- Ch2 – ARM Instruction Set Architecture
- Ch3 – Accès aux données
- Ch4 – Programmation structurée
- Ch5 – Cycle d'exécution
- Ch6 – Codage binaire
- Ch7 – Microcontrôleur ARM Cortex-M

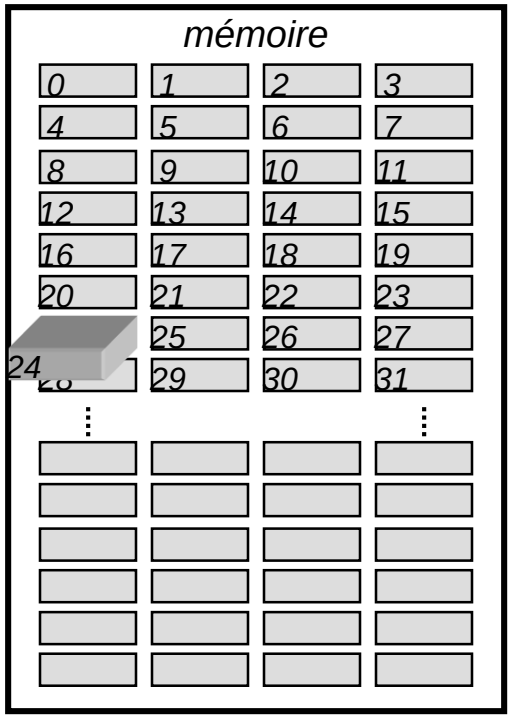
POLYTECH
NICE-SOPHIA

Accès aux données

- Organisation mémoire
- Instructions d'accès mémoire
- Modes d'adressage
- Instructions de transferts multiples
- Interface bus

Organisation interne de la mémoire

- Sur le processeur ARM:
  - Une cellule mémoire contient 1 octet (8 bits)
  - Une cellule mémoire possède une adresse
    - Problème: les échanges entre la mémoire et les registres se font sur des données de 32 bits, les types de données sont en format 8, 16, 32 ou 64 bits => il faut certaines règles pour un accès cohérent aux données



Organisation des données en mémoire

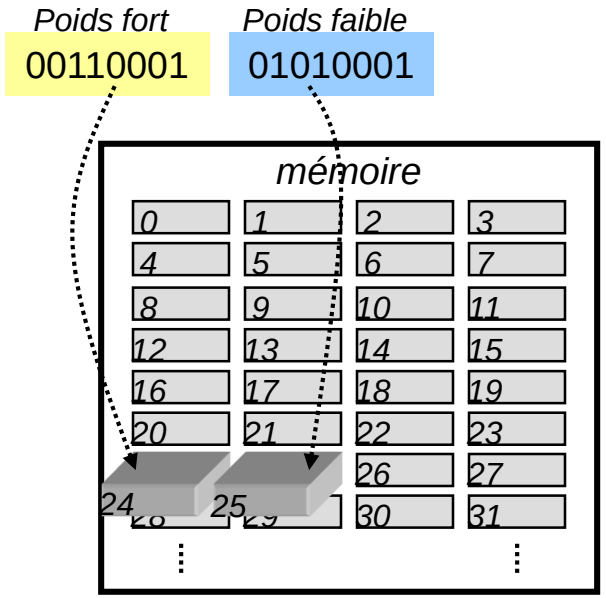
- Types scalaires
- Entiers 8 bits

  ou caractères ASCII → 1 cellule

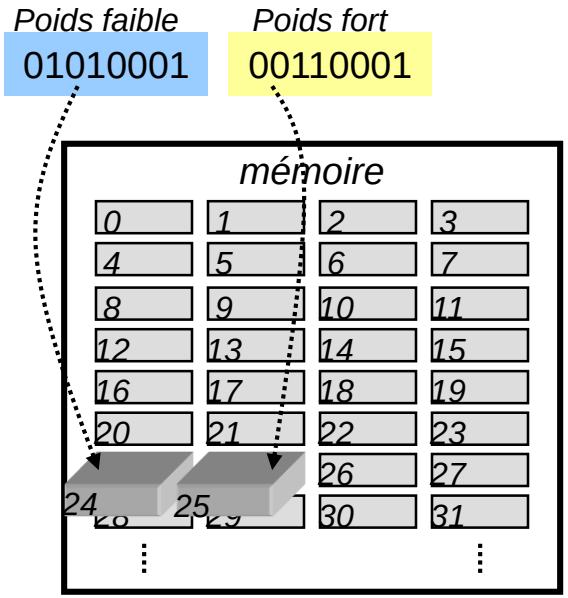
  - Entiers 16 bits → 2 cellules consécutives
  - Entiers ou flottant 32 bits → 4 cellules consécutives
  - Entiers ou flottants 64 bits → 8 cellules consécutives

Organisation des données en mémoire

- Convention pour les données de taille > 8 bits
  - Ordre de stockage
    - « big-endian » : octet de poids fort à l'adresse la plus faible
    - « little-endian » : octet de poids faible à l'adresse la plus faible
    - Exemple: stockage de l'entier $(12625)_{dec} = (0011000101010001)_{bin}$ sur 16 bits à l'adresse 24



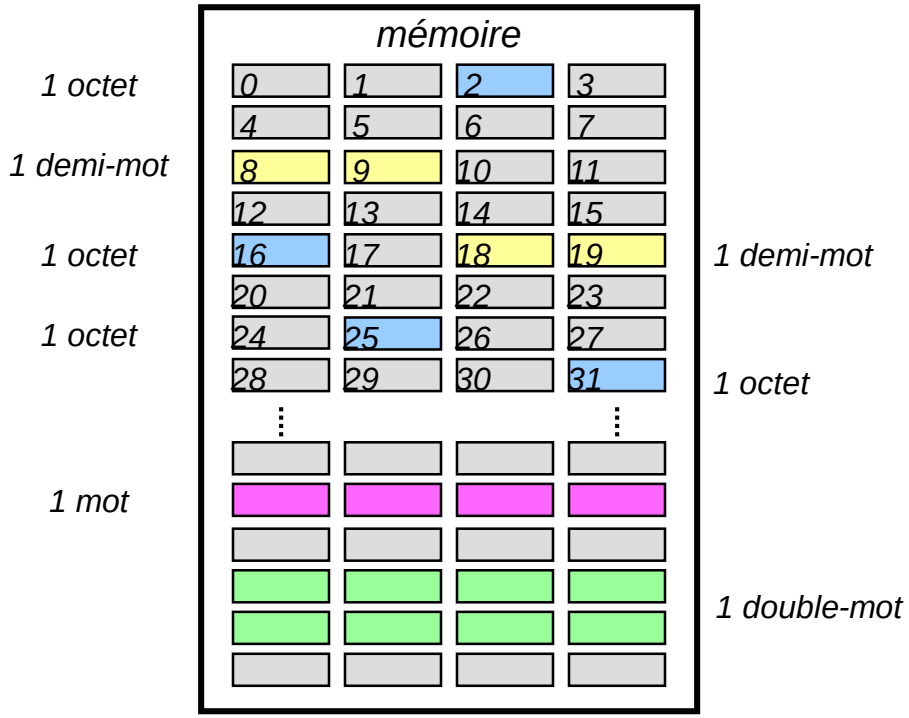
big-endian



little-endian

Organisation des données en mémoire

- Convention pour les données de taille > 8 bits
  - Alignement
    - Demi-mots (16 bits) à des adresses paires
    - Mots et double-mots (32 et 64 bits) à des adresses multiples de 4



Contrairement aux mots de 8 bits, on ne peut pas stocker en mémoire un mot de taille > 8 bits à n'importe quelle adresse (convention).

Les demi mots occupent 2 adresses consécutives suivant l'adresse de départ.

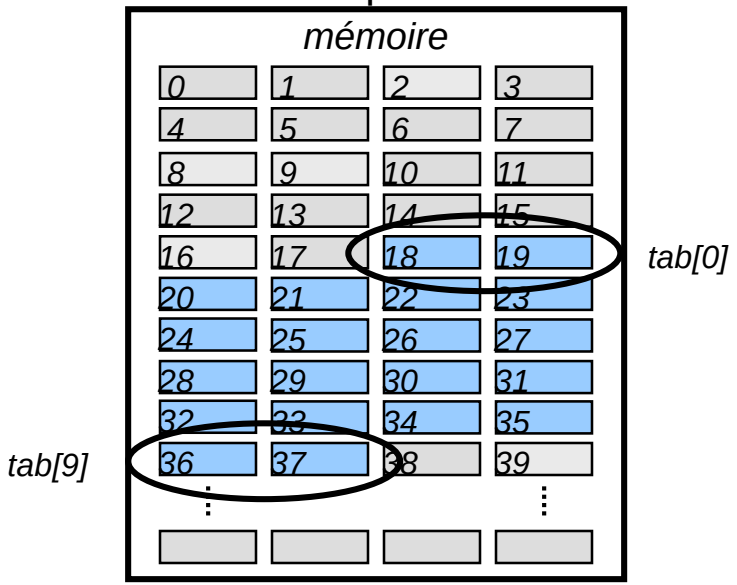
Les mots doivent occuper un groupe de 4 adresses consécutives suivant l'adresse de départ.

Les double mots doivent occuper un groupe de 8 adresses consécutives suivant l'adresse de départ.

Organisation des données en mémoire

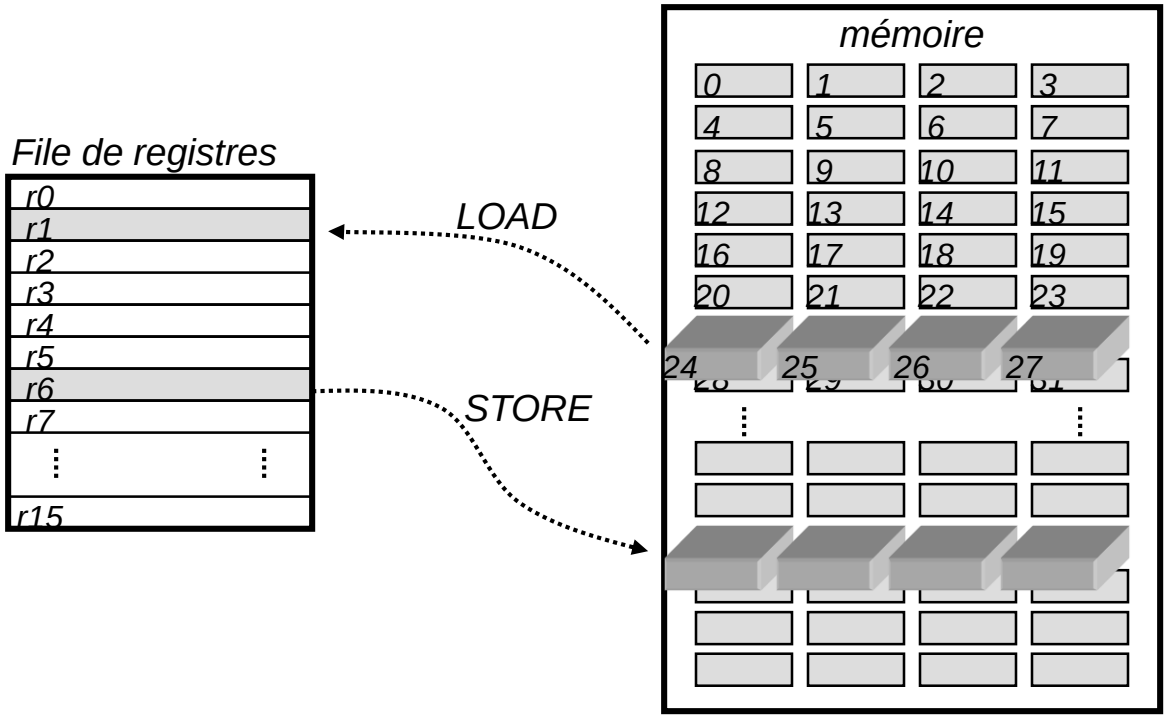
- Types structurés
  - tableaux
    - Typiquement, une succession de données de même format rangées à des adresses consécutives
    - Le rangement des éléments en mémoire doit respecter les contraintes d'alignement: 1er élément à une adresse paire, les éléments sont donc placés à des adresses paire consécutives
    - Exemple un tableau de 10 entiers représentés sur 16 bits

short int tab[10]



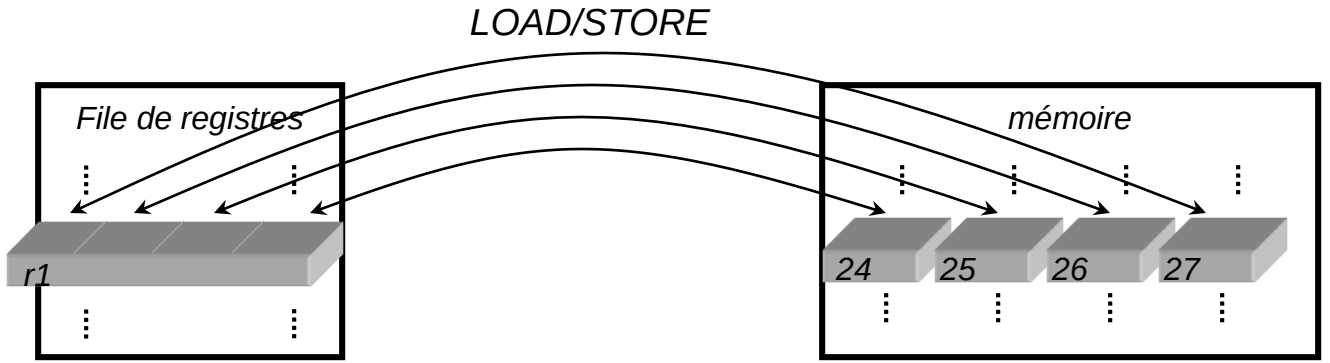
Organisation de la mémoire

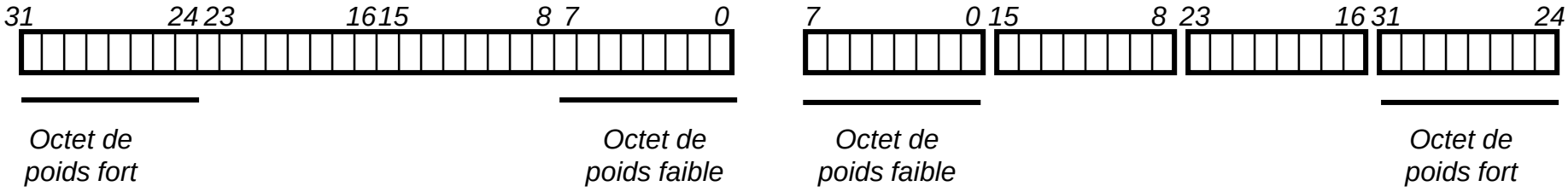
- Accès mémoire sur les processeurs ARM
  - Accès 8, 16 ou 32-bit
  - Exemple d'accès 32-bit



Organisation de la mémoire

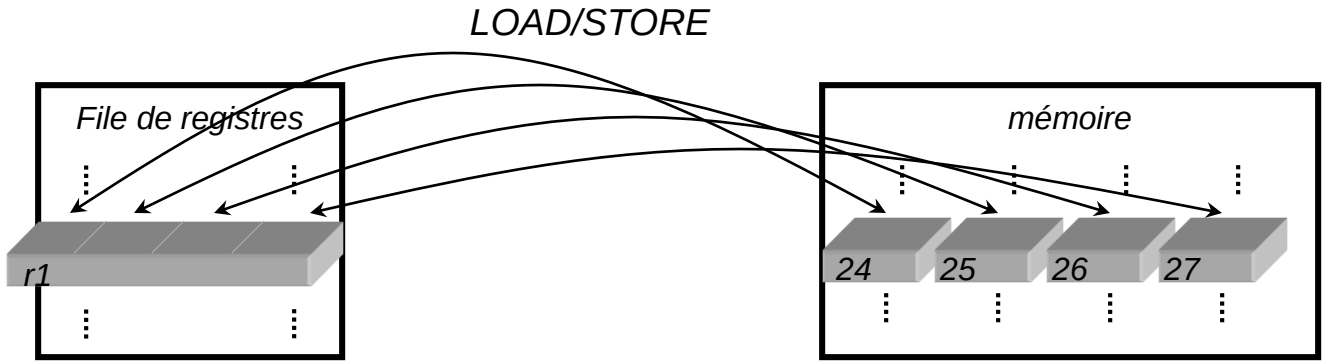
- Accès mémoire sur les processeurs ARM
  - Organisation « little endian » par défaut
  - Octet de poids fort à l'adresse forte

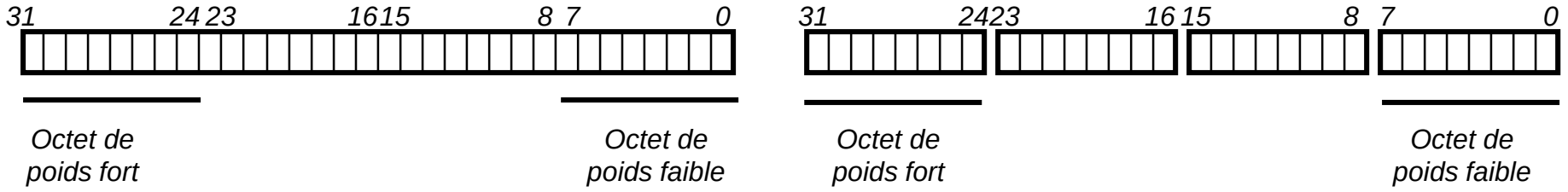




Organisation de la mémoire

- Accès mémoire sur les processeurs ARM
  - Configuration possible en « big endian »
  - Octet de poids fort à l'adresse faible





POLYTECH
NICE-SOPHIA

Accès aux données

- Organisation mémoire
- Instructions d'accès mémoire
- Modes d'adressage
- Instructions de transferts multiples
- Interface bus

Instructions d'accès mémoire

- Instructions sur les mots (32 bits)
  - LDR Rd, *effective address* Load Register
  - STR Rd, *effective address* Store Register
- Instructions sur les demi-mots (*Half-word 16 bits*)
  - LDRH Rd, *effective address* *(*Unsigned)
  - STRH Rd, *effective address*
  - LDRSH Rd, *effective address* *(*Signed)
  - STRSH Rd, *effective address*
- Instructions sur les octets (*Byte 8 bits*)
  - LDRB Rd, *effective address* *(*Unsigned)
  - STRB Rd, *effective address*
  - LDRSB Rd, *effective address* *(*Signed)
  - STRSB Rd, *effective address*

Instructions d'accès mémoire

Instruction	Rd[31…24]	Rd[23…16]	Rd[15…8]	Rd[7…0]
LDR	Mem[*e.a.*]$_{word}$			
LDRH	0000…0000		Mem[*e.a.*]$_{half-word}$	
LDRSH	Extension bit de signe		Mem[*e.a.*]$_{half-word}$	
LDRB	0000…0000			Mem[*e.a.*]$_{byte}$
LDRSB	Extension bit de signe			Mem[*e.a.*]$_{byte}$

- *e.a.* effective address

Supposons que r2 contienne la valeur hexadécimale 0x1000

LDR r1, [r2] renvoie 4 octets lus aux adresses 0x1000, 0x1001, 0x1002, 0x1003

LDRH r1, [r2] renvoie 2 octets lus aux adresses 0x1000, 0x1001, extension des 2 octets manquant par 0

LDRSH r1, [r2] renvoie 2 octets lus aux adresses 0x1000, 0x1001 , extension des 2 octets manquant par bit de signe

LDRB r1, [r2] renvoie l'octet lu à l'adresse 0x1000, extension des 3 octets manquant par 0

LDRSB r1, [r2] renvoie l'octet lu à l'adresse 0x1000, extension des 3 octets manquant par bit de signe

POLYTECH
NICE-SOPHIA

Accès aux données

- Organisation mémoire
- Instructions d'accès mémoire
- Modes d'adressage
- Instructions de transferts multiples
- Interface bus

Modes d'adressage

- Un mode d'adressage spécifie comment accéder aux données.
- La terminologie varie d'une architecture à une autre.
- Modes d'adressage ARM:
  - Adressage immédiat
    - L'adresse est spécifiée directement dans l'instruction.
    - `LDR r0, #1000 ; r0 ← mem[1000]`
  - Adressage indirect par registre
    - Un registre spécifie une adresse mémoire où se trouve la donnée.
    - `LDR r0, [r1] ; r0 ← mem[r1]`
  - Adressage indexé
    - 2 registres sont utilisés pour calculer une adresse mémoire: le registre de base et le registre d'index (offset)
    - Par exemple, un tableau: le premier élément est l'adresse de base (*base address*), les autres éléments sont adressés en ajoutant un *index* à l'adresse de base (offset).
    - `LDR r0, [r1, #4] ; r0 ← mem[r1+4]`
    - `LDR r0, [r1, r2] ; r0 ← mem[r1+r2]`

Modes d'adressage

- Un mode d'adressage spécifie comment accéder aux données.
- La terminologie varie d'une architecture à une autre.
- Modes d'adressage ARM:
  - Adressage immédiat
    - L'adresse est spécifiée directement dans l'instruction.
    - `LDR r0, #1000` `; r0 ← mem[1000]` INTERDIT EN ASM ARM
  - Adressage indirect par registre
    - Un registre spécifie une adresse mémoire où se trouve la donnée.
    - `LDR r0, [r1]` `; r0 ← mem[r1]`
  - Adressage indexé
    - 2 registres sont utilisés pour calculer une adresse mémoire: le registre de base et le registre d'index (offset)
    - Par exemple, un tableau: le premier élément est l'adresse de base (*base address*), les autres éléments sont adressés en ajoutant un *index* à l'adresse de base (offset).
    - `LDR r0, [r1, #4]` `; r0 ← mem[r1+4]`
    - `LDR r0, [r1, r2]` `; r0 ← mem[r1+r2]`

Modes d'adressage

- Modes d'adressage indirects sur processeur ARM

Notation	Nom	Adresse effective
[Rn]	Adressage indirect	Valeur de Rn
[Rn, *offset*]	Adressage indirect pré-indexé	Valeur de Rn + *offset*
[Rn, *offset*]!	Adressage indirect pré-indexé automatique	Valeur de Rn + *offset*
[Rn], *offset*	Adressage indirect post-indexé (auto)	Valeur de Rn

- *Rn* = registre de base
- *offset* =
  - *#literal*
  - ± Rm
  - ± Rm, *shift*

Modes d'adressage

- Les modes pré-indexé automatique et post-indexé modifient le registre Rn
  - On effectue deux opérations : accès mémoire ET modification du registre de base Rn.

- Mode pré-indexé automatique
  - On fait d'abord Rn ← Rn + *offset*
  - Puis on accède à mem[Rn]
- Mode post-indexé
  - On accède d'abord à mem[Rn]
  - Puis on fait Rn ← Rn + *offset*

Modes d'adressage

- Exemples
  - `LDR r3,[r1]` — $r3 \leftarrow mem[r1]_{word}$
    - Adressage indirect
  - `LDR r3,[r1,#2]` — $r3 \leftarrow mem[r1+2]_{word}$
    - Adressage pré-indexé
  - `LDR r3,[r1,+r4]` — $r3 \leftarrow mem[r1+r4]_{word}$
    - Adressage pré-indexé
  - `LDR r3,[r1,+r4]!` — $r1 \leftarrow r1 + r4;$ $r3 \leftarrow mem[r1]_{word}$
    - Adressage pré-indexé automatique
  - `LDR r3,[r1],+r4` — $r3 \leftarrow mem[r1]_{word}$ $r1 \leftarrow r1 + r4;$
    - Adressage post-indexé automatique
  - `LDR r3,[r1,-r4, LSL #2]` — $r3 \leftarrow mem[r1-(r4<<2)]_{word}$
    - Adressage pré-indexé

Modes d'adressage

- Exemples
  - `STR r3,[r1]` — $mem[r1]_{word} \leftarrow r3$
    - Adressage indirect
  - `STR r3,[r1,#2]` — $mem[r1+2]_{word} \leftarrow r3$
    - Adressage pré-indexé
  - `STR r3,[r1,+r4]` — $mem[r1+r4]_{word} \leftarrow 3$
    - Adressage pré-indexé
  - `STR r3,[r1,+r4]!` — $r1 \leftarrow r1 + r4;$ $mem[r1]_{word} \leftarrow r3$
    - Adressage pré-indexé automatique
  - `STR r3,[r1],+r4` — $mem[r1]_{word} \leftarrow r3$ $r1 \leftarrow r1 + r4;$
    - Adressage post-indexé automatique
  - `STR r3,[r1,-r4, LSL #2]` — $mem[r1-(r4<<2)]_{word} \leftarrow r3$
    - Adressage pré-indexé

POLYTECH
NICE-SOPHIA

Accès aux données

- Organisation mémoire
- Instructions d'accès mémoire
- Modes d'adressage
- Instructions de transferts multiples
- Interface bus

Instructions de transfert

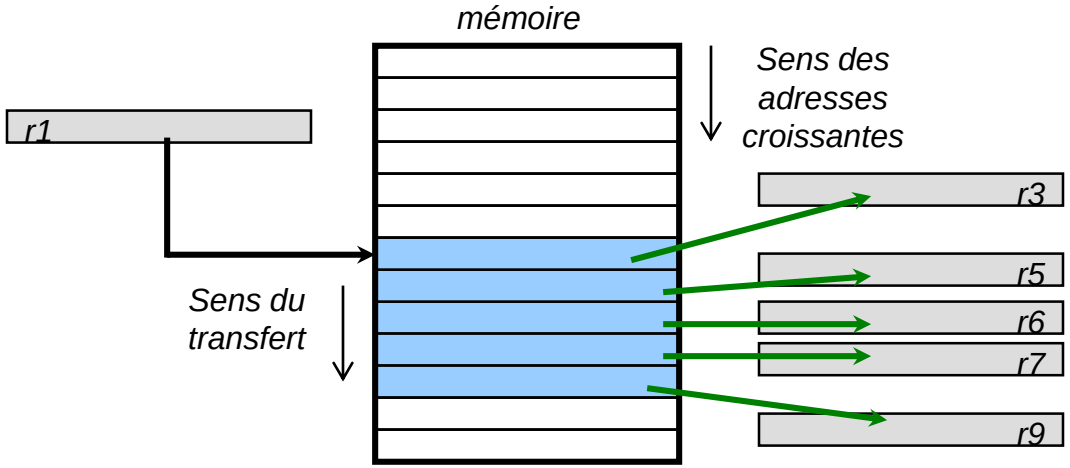
- Instructions de transferts multiples
  - LDM*mode* Rn, *reglist* — Load Multiple
  - LDM*mode* Rn!, *reglist*
  - STM*mode* Rn, *reglist* — Store Multiple
  - STM*mode* Rn!, *reglist*
  - *Reglist*: liste de registres
  - Rn: registre de base
  - *mode*
    - IA increment after — IB increment before
    - DA decrement after — DB decrement before

Instructions de transfert

- Instruction de transferts multiples (LDM*IA* Rn, *reglist*)
  - Exemple
    - LDMIA r1, {r3, r5-r7, r9}

Équivaut à la séquence

- LDR r3, [r1]
- LDR r5, [r1, #4]
- LDR r6, [r1, #8]
- LDR r7, [r1, #12]
- LDR r9, [r1, #16]



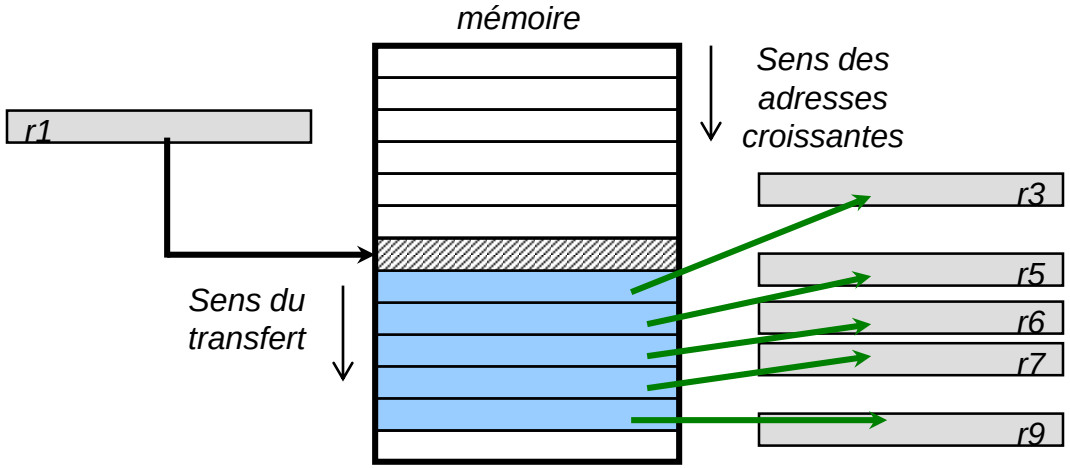
IA: on INcrémente r1 APRES (accès à la donnée)

Instructions de transfert

- Instruction de transferts multiples (LDM*IB* Rn, *reglist*)
  - Exemple
    - LDMIB r1, {r3, r5-r7, r9}

Équivaut à la séquence

- LDR r3, [r1, #4]
- LDR r5, [r1, #8]
- LDR r6, [r1, #12]
- LDR r7, [r1, #16]
- LDR r9, [r1, #20]



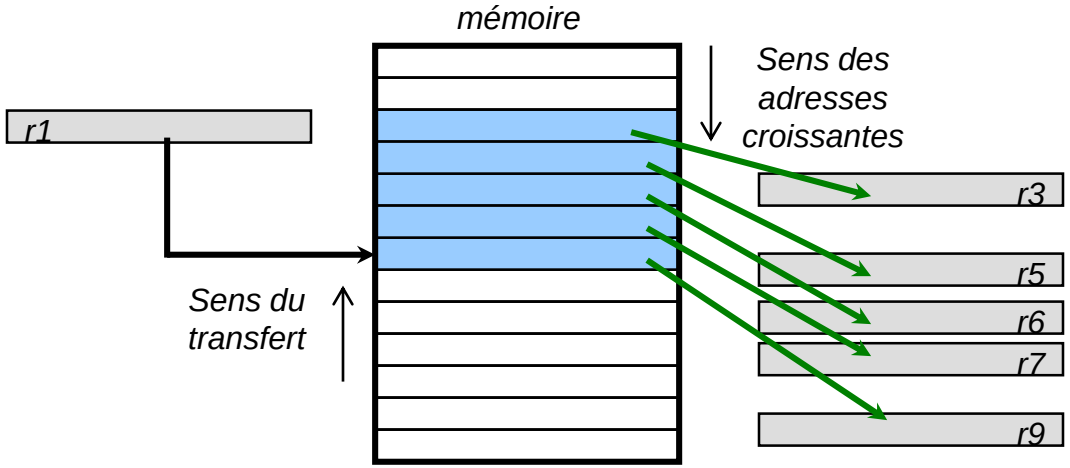
IB: on INcrémente r1 AVANT (accès à la donnée)

Instructions de transfert

- Instruction de transferts multiples (LDM*DA* Rn, *reglist*)
  - Exemple
    - LDMDA r1, {r3, r5-r7, r9}

Équivaut à la séquence

- LDR r9, [r1]
- LDR r7, [r1, #-4]
- LDR r6, [r1, #-8]
- LDR r5, [r1, #-12]
- LDR r3, [r1, #-16]



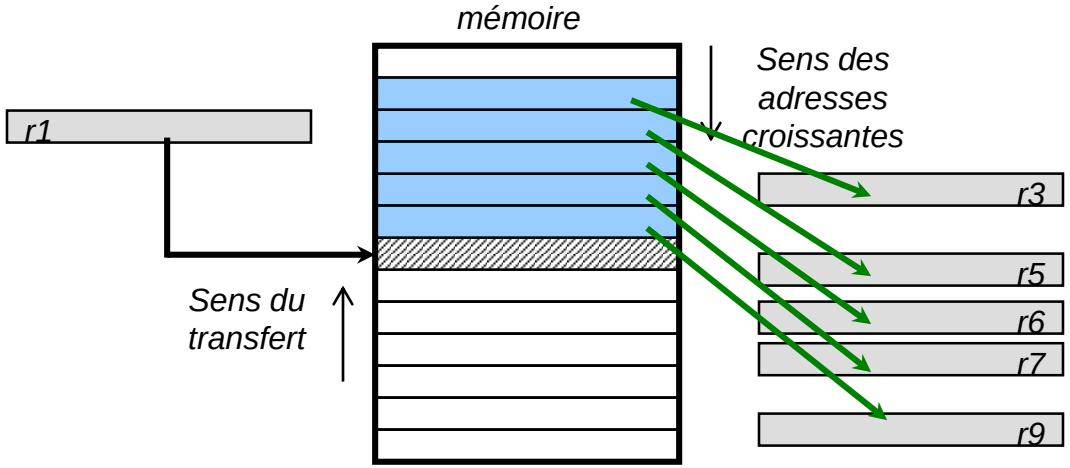
DA: on DEcrémente r1 APRES (accès à la donnée)

Instructions de transfert

- Instruction de transferts multiples (LDM*DB* Rn, *reglist*)
  - Exemple
    - LDMDB r1, {r3, r5-r7, r9}

Équivaut à la séquence

- LDR r9, [r1, #-4]
- LDR r7, [r1, #-8]
- LDR r6, [r1, #-12]
- LDR r5, [r1, #-16]
- LDR r3, [r1, #-20]



DB: on DEcrémente r1 AVANT (accès à la donnée)

Instructions de transfert

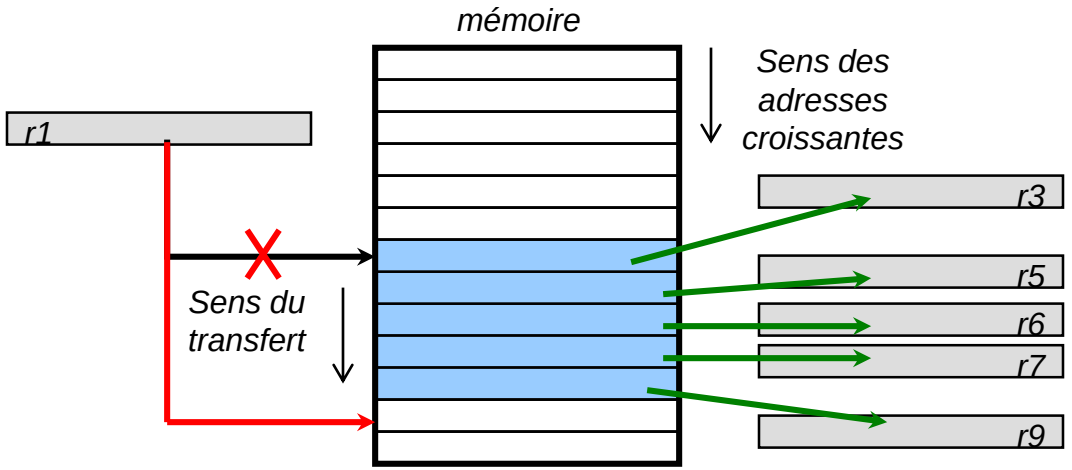
- Instruction de transferts multiples (LDM*IA* Rn!, *reglist*)
  - Exemple
    - LDMIA r1!, {r3, r5-r7, r9}

Équivaut à la séquence

- LDR r3, [r1] , #4
- LDR r5, [r1] , #4
- LDR r6, [r1] , #4
- LDR r7, [r1] , #4
- LDR r9, [r1] , #4

! = Write back

On INcrémente r1 APRES (accès à la donnée)

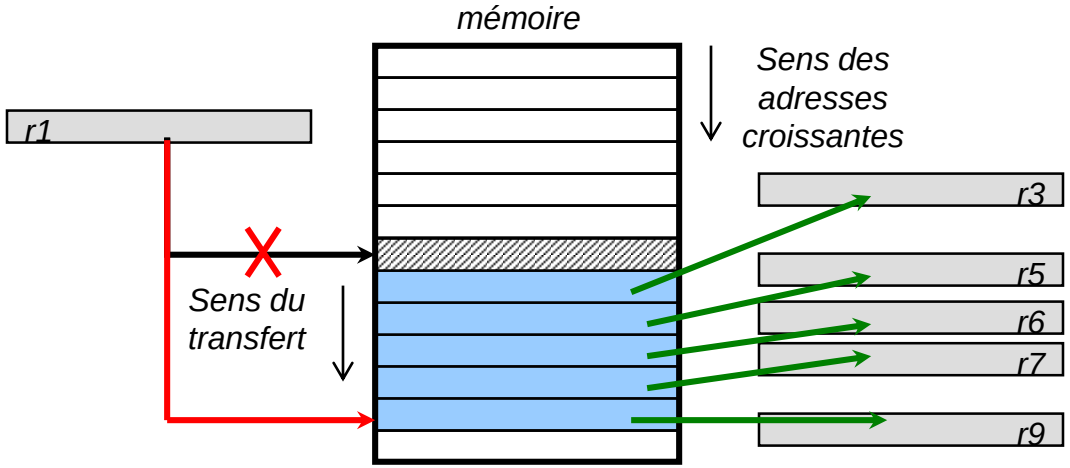


Instructions de transfert

- Instruction de transferts multiples (LDM*IB* Rn!, *reglist*)
  - Exemple
    - LDMIB r1!, {r3, r5-r7, r9}

Équivaut à la séquence

- LDR r3, [r1, #4]!
- LDR r5, [r1, #4]!
- LDR r6, [r1, #4]!
- LDR r7, [r1, #4]!
- LDR r9, [r1, #4]!



On INcrémente r1 AVANT (accès à la donnée)

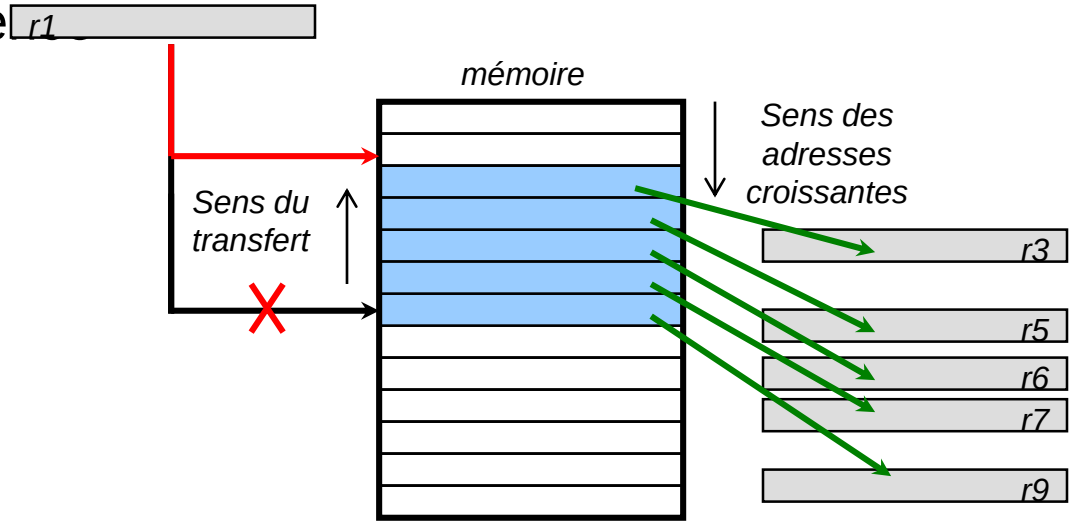
Instructions de transfert

- Instruction de transferts multiples (LDM*DA* Rn!, *reglist*)
  - Exemple
    - LDMDA r1!, {r3, r5-r7, r9}

Équivaut à la séque

- LDR r9, [r1], #-4
- LDR r7, [r1], #-4
- LDR r6, [r1], #-4
- LDR r5, [r1], #-4
- LDR r3, [r1], #-4

On DEcrémente r1 APRES (accès à la donnée)



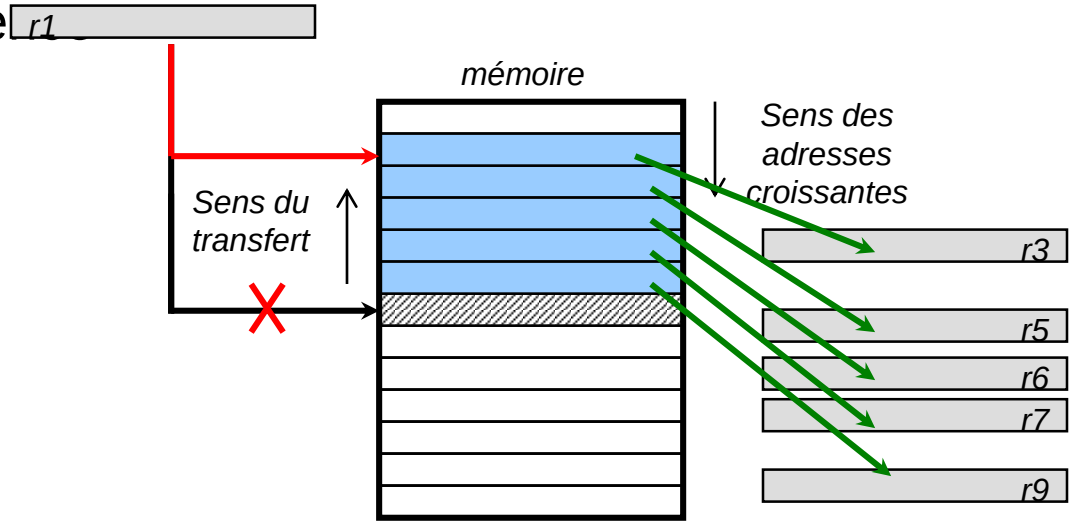
Instructions de transfert

- Instruction de transferts multiples (LDM*DB* Rn!, *reglist*)
  - Exemple
    - LDMDB r1!, {r3, r5-r7, r9}

Équivaut à la séque

- LDR r9, [r1, #-4]!
- LDR r7, [r1, #-4]!
- LDR r6, [r1, #-4]!
- LDR r5, [r1, #-4]!
- LDR r3, [r1, #-4]!

On DEcrémente r1 AVANT (accès à la donnée)



Exemple récapitulatif

```
                @ Directives
        .text
        .align 4
        .global start
                @ Section de données
EX_1:   .word   0xFFEEFDFC
EX_2:   .word   1, 2, 3, 4, 5, 6
EX_3:   .word   7, 8, 9, 10, 11, 12
                @ Section de code
                @ mot, demi mot, octet,
                @ signe/non signe
        ADR     R1, EX_1
        LDR     R3, [R1]
        LDRH    R3, [R1]
        LDRSH   R3, [R1]
        LDRB    R3, [R1]
        LDRSB   R3, [R1]

                @ Modes d'adressage
        ADR     R1, EX_2
        LDR     R3, [R1]  @ indirect par reg.
        MOV     R2, #4
        LDR     R3, [R1, R2] @ indexé
        ADR     R1, EX_3
        LDR     R3, [R1, R2]! @ pre-indexé auto
        ADR     R1, EX_3
        LDR     R3, [R1], R2 @ post indexé auto
                @ tranferts mulitiples
                @ SANS MAJ reg. de base
        ADR     R1, EX_3
        LDMIA   r1, {r3, r5-r7, r9}
        LDMIB   r1, {r3, r5-r7, r9}
        LDMDA   r1, {r3, r5-r7, r9}
        LDMDB   r1, {r3, r5-r7, r9}
                @ AVEC MAJ reg. de base
        LDMIA   r1!, {r3, r5-r7, r9}
        LDMIB   r1!, {r3, r5-r7, r9}
        LDMDA   r1!, {r3, r5-r7, r9}
        LDMDB   r1!, {r3, r5-r7, r9}
wait:   b       wait
```

POLYTECH
NICE-SOPHIA

Accès aux données

- Organisation mémoire
- Instructions d'accès mémoire
- Modes d'adressage
- Instructions de transferts multiples
- Interface bus

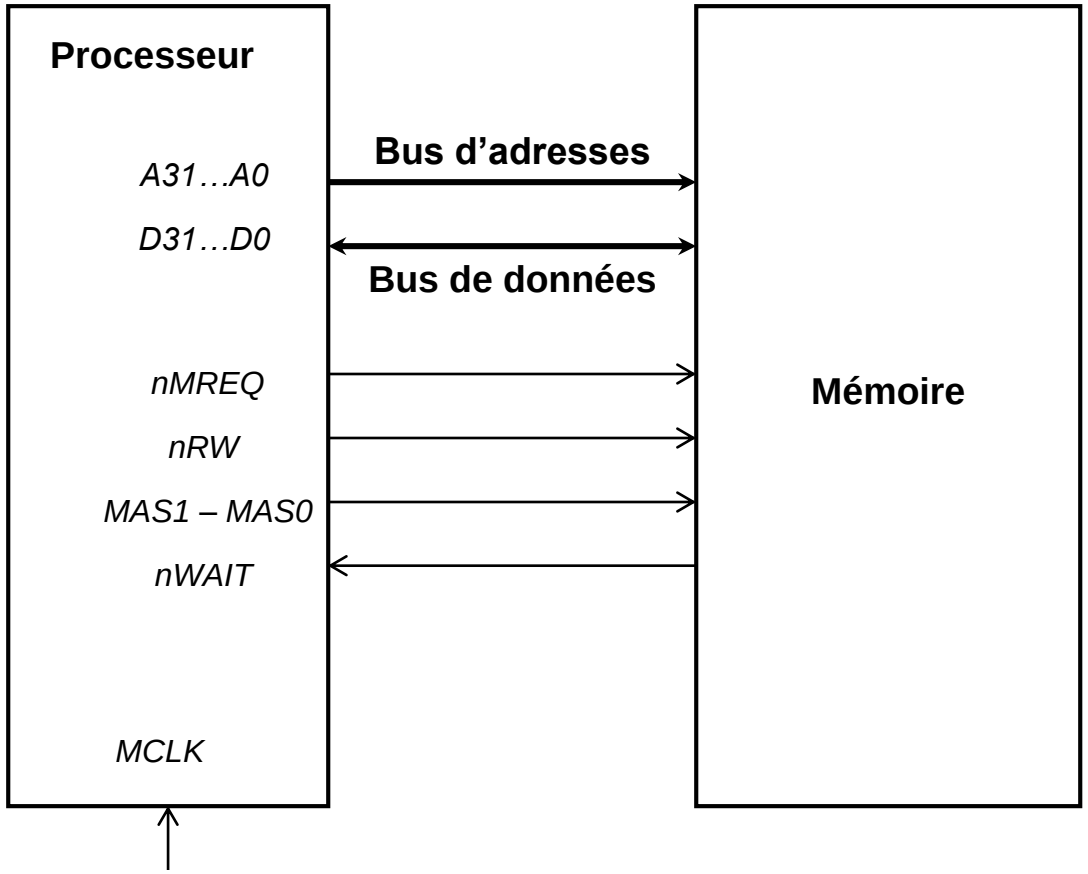
Interface bus

- Transport des données entre le processeur et la mémoire
  - Le transport des données se fait par le bus de données 32 bits
  - Bidirectionnel permettant l'accès en lecture / écriture
- Sélection de l'adresse des cellules mémoire accédées
  - Sélection des adresses par le bus d'adresse qui spécifie la localisation en mémoire des cellules accédées (32 bits)
- Positionnement des signaux de contrôle
  - Un accès mémoire nécessite l'utilisation de signaux de commande
    - Pour définir le type d'accès (lecture/écriture)
    - Pour définir la taille des données accédées (8, 16, 32 bits)
    - Etc.
- L'interface bus définit un protocole de synchronisation des échanges
  - C'est à dire la façon précise d'activer ces éléments lors de transferts de données processeur / mémoire

Interface bus

- Signaux de contrôle et bus:
  - Bus d'adresse
    - A31…A0: spécifie les emplacements d'octets (32 bits -> 4Go), une adresse correspond à 1 mot de 8 bits
  - Bus de données
    - D31…D0: transporte les données. 32 bits: 1 accès à une adresse peut renvoyer 1, 2, ou 4 octets consécutifs (d'où les règles d'alignement)
  - MAS: Memory Access Size
    - taille des données accédées (8, 16, 32 bits)
  - nMREQ: Memory Request
    - Demande d'accès à la mémoire
  - nRW: Read/Write
    - Demande d'accès en lecture ou écriture
  - nWAIT
    - Insertion d'états d'attente
  - MCLK: Memory Clock

    'n' désigne des signaux actifs à l'état bas (0)



Accès aux données

- Transport des données par le bus de données
  - Broches D31…D0

  - Correspondance entre les broches du bus de données et les cellules mémoires

Bus	Mode little- endian	Mode big- endian
D7…D0	Adresses de la forme 4n	Adresses de la forme 4n+3
D15…D8	Adresses de la forme 4n+1	Adresses de la forme 4n+2
D23…D16	Adresses de la forme 4n+2	Adresses de la forme 4n+1
D31…D24	Adresses de la forme 4n+3	Adresses de la forme 4n

Accès aux données

- Sélection des cellules mémoire accédées
  - Broches A31…A0
  - Espace adressable : 2^{32} octets = 4 Go
  - Alignement des données en fonction de leur taille (8, 16 ou 32 bits)

Taille	A31…A0	Partie du bus concernée
Octet (8 bits)	Quelconque	D31…D24 ou D23…D16 ou D15…D8 ou D7…D0
Demi-mot (16 bits)	Adresses paires (A0=0)	D31…D16 ou D15…D0
Mot (32 bits)	Adresses multiples de 4 (A0 = A1 = 0)	D31…D0

Format des échanges

- Type d'accès nRW (Read/Write)
  - 0 → accès en lecture (load)
  - 1 accès en écriture (write)

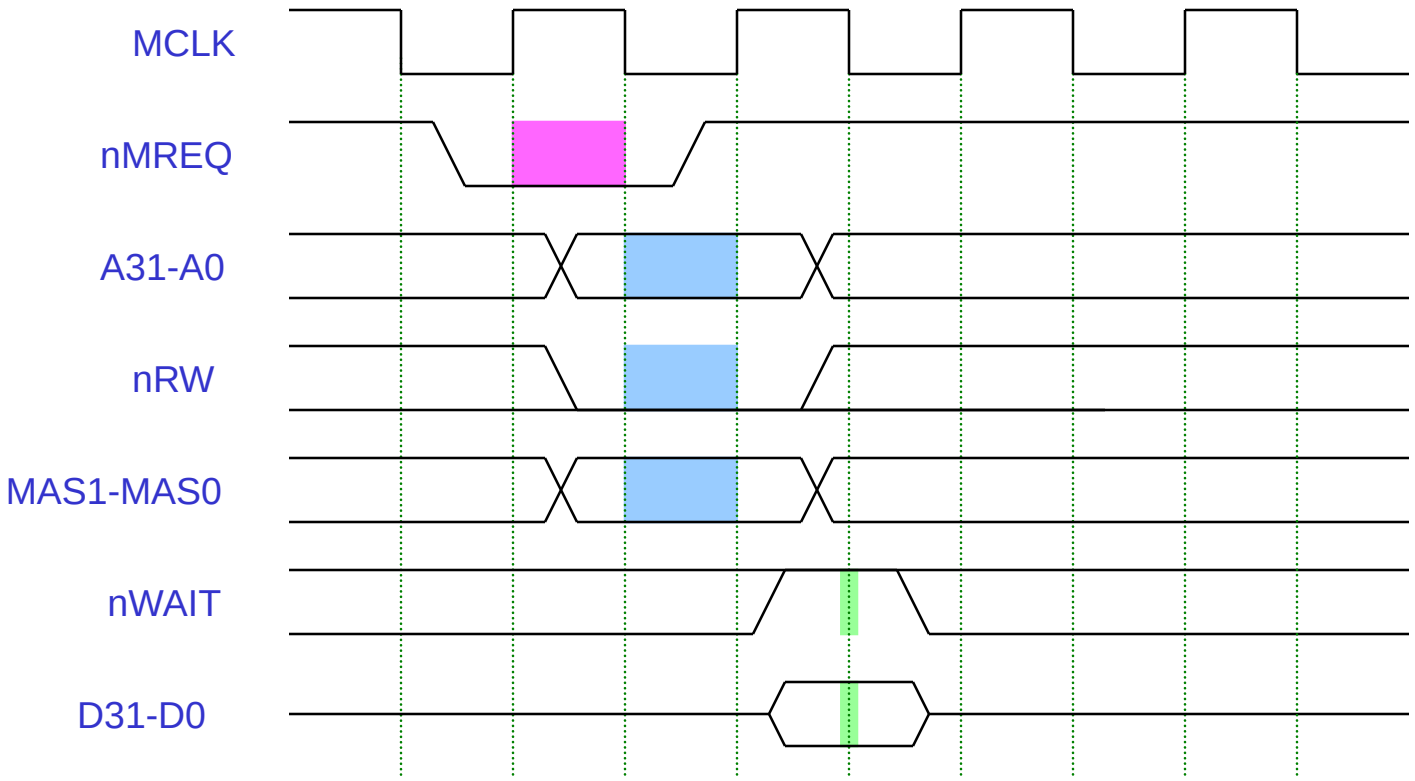
- Taille des données accédées MAS (Memory Access Size)
  - 00 → Octet (8 bits)
  - 01 → Demi-mot (16 bits)
  - 10 → Mot (32 bits)

Synchronisation des échanges

- Demande d'accès nMREQ (Memory Request)
  - 0 → le processeur demande à accéder à la mémoire
- Insertion d'états d'attente nWAIT
  - 0 → la mémoire demande au processeur d'attendre encore un cycle d'horloge
- Horloge MCLK (Memory Clock)
  - Horloge
  - Les données lues en mémoires sont échantillonnées sur les fronts descendants de MCLK si nWAIT est à 1

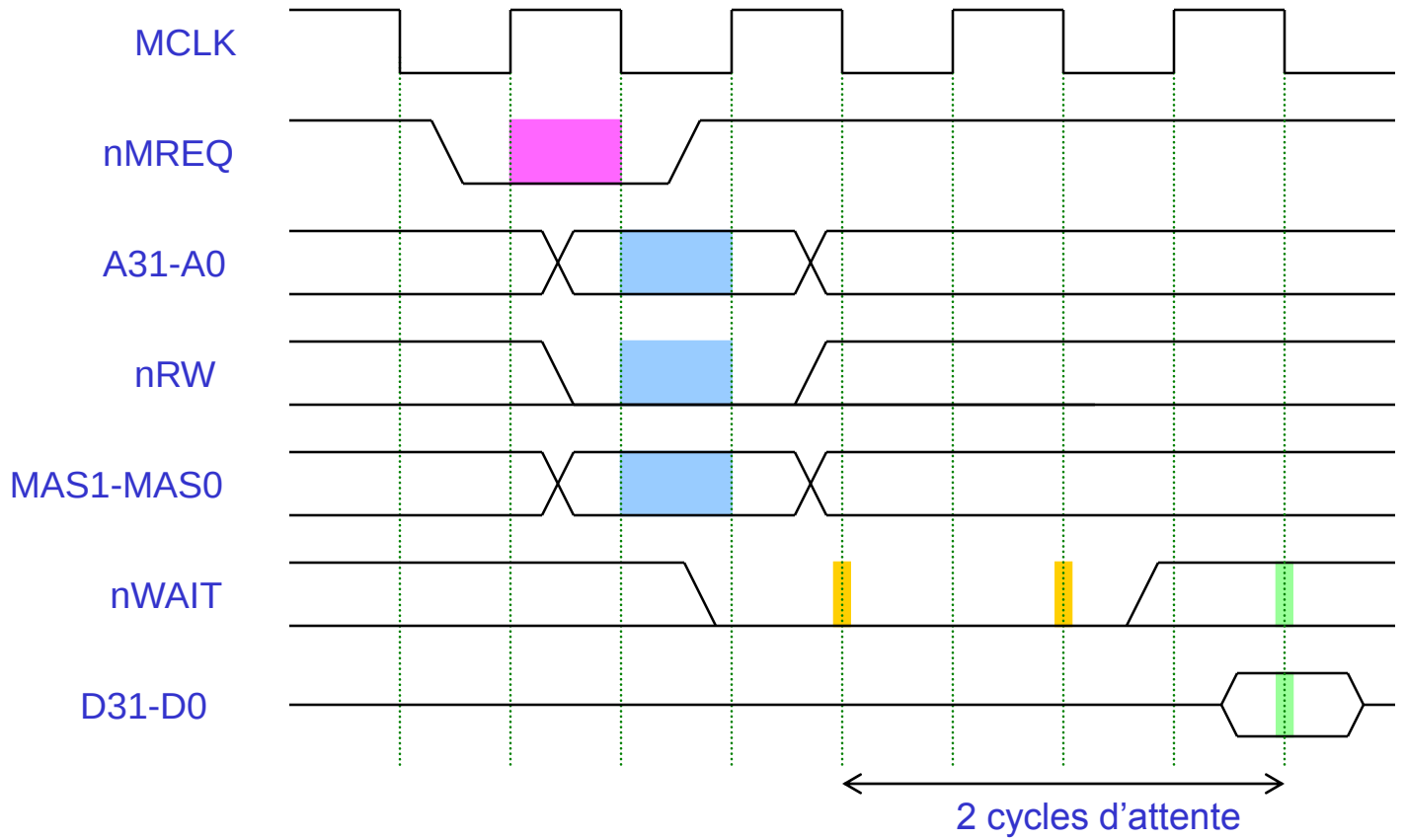
Accès à une donnée simple

- Accès en lecture sans cycle d'attente



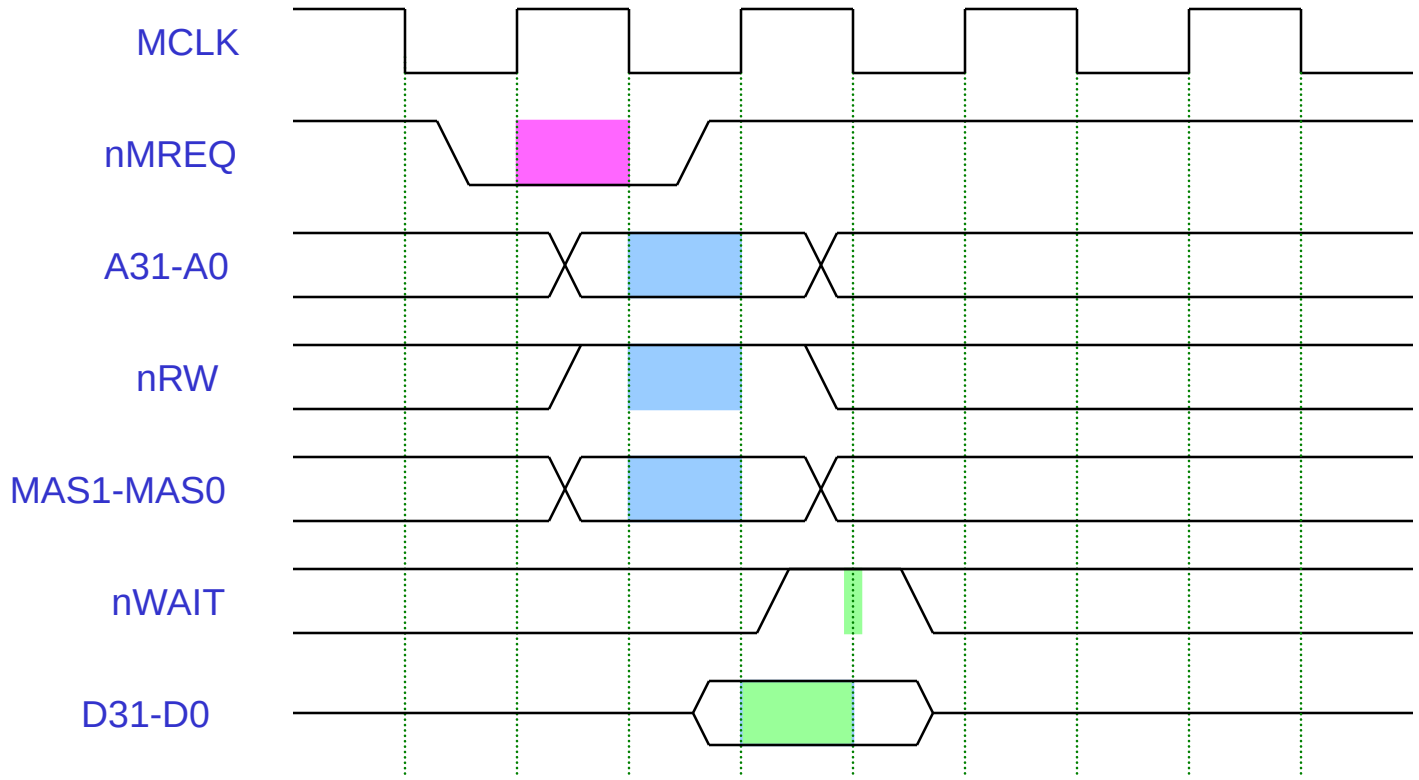
Accès à une donnée simple

- Accès en lecture avec cycles d'attente



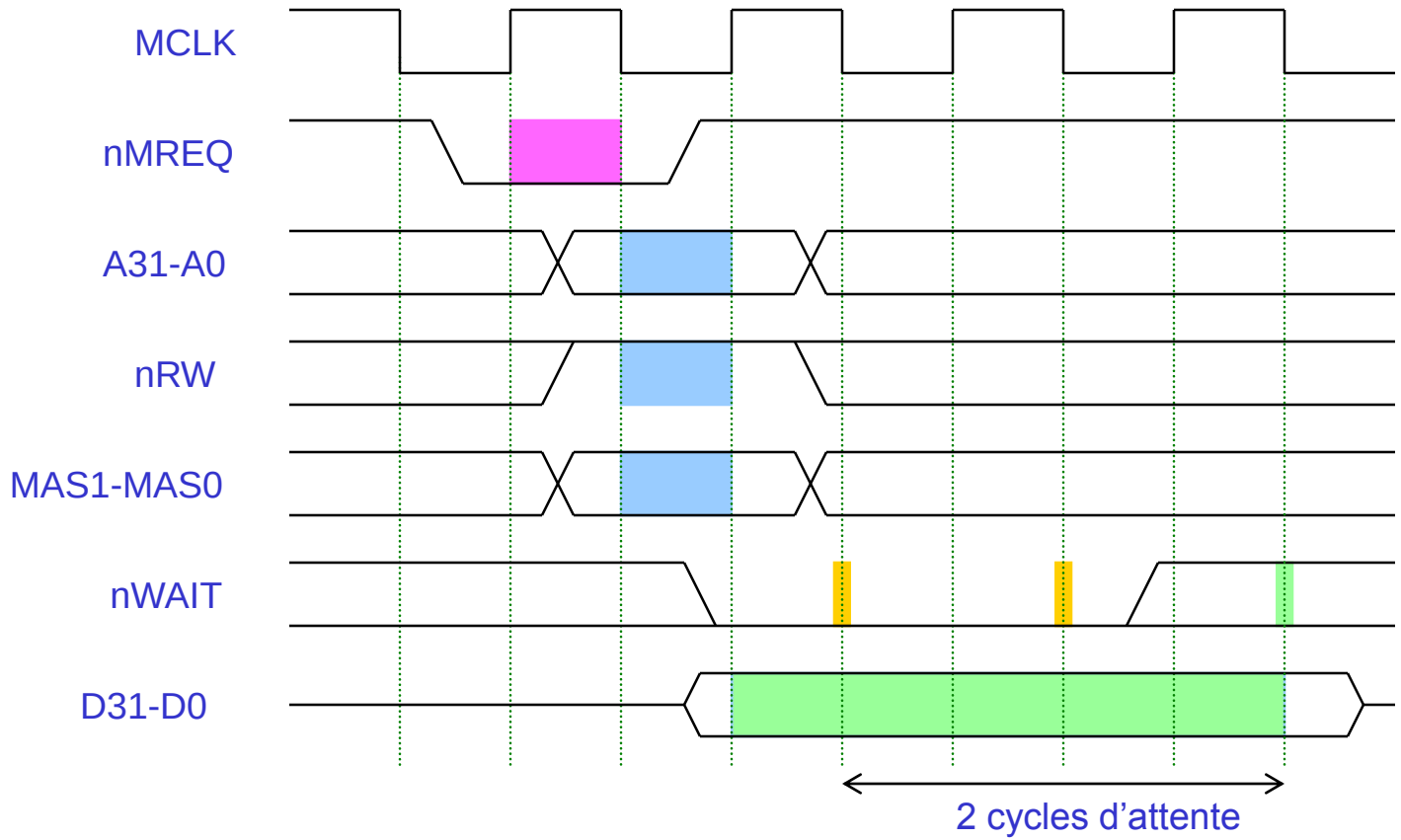
Accès à une donnée simple

- Accès en écriture sans cycle d'attente



Accès à une donnée simple

- Accès en écriture avec cycles d'attente



Accès successifs

- On peut démarrer un nouveau cycle d'accès sans attendre que le précédent soit terminé (exécution pipeline)

  → succession d'accès en lecture sans cycles d'attente

  → un accès par période d'horloge

Accès en lecture d'1 donnée en 2 cycles: lec ADDR + RW. Pendant l'étape RW de l'accès 1, on démarre lec ADDR de l'accès 2 (recouvrement) … N accès peuvent être réalisés en N+1 cycles au lieu de 2N cycles.