CHAPITRE 4

Jeux d’instructions ARM

Plan du Chapitre

Introduction

Représentation binaire d'instructions ARM

- Traitement de données
- Branchements
- Transfert de données
- Multiplication

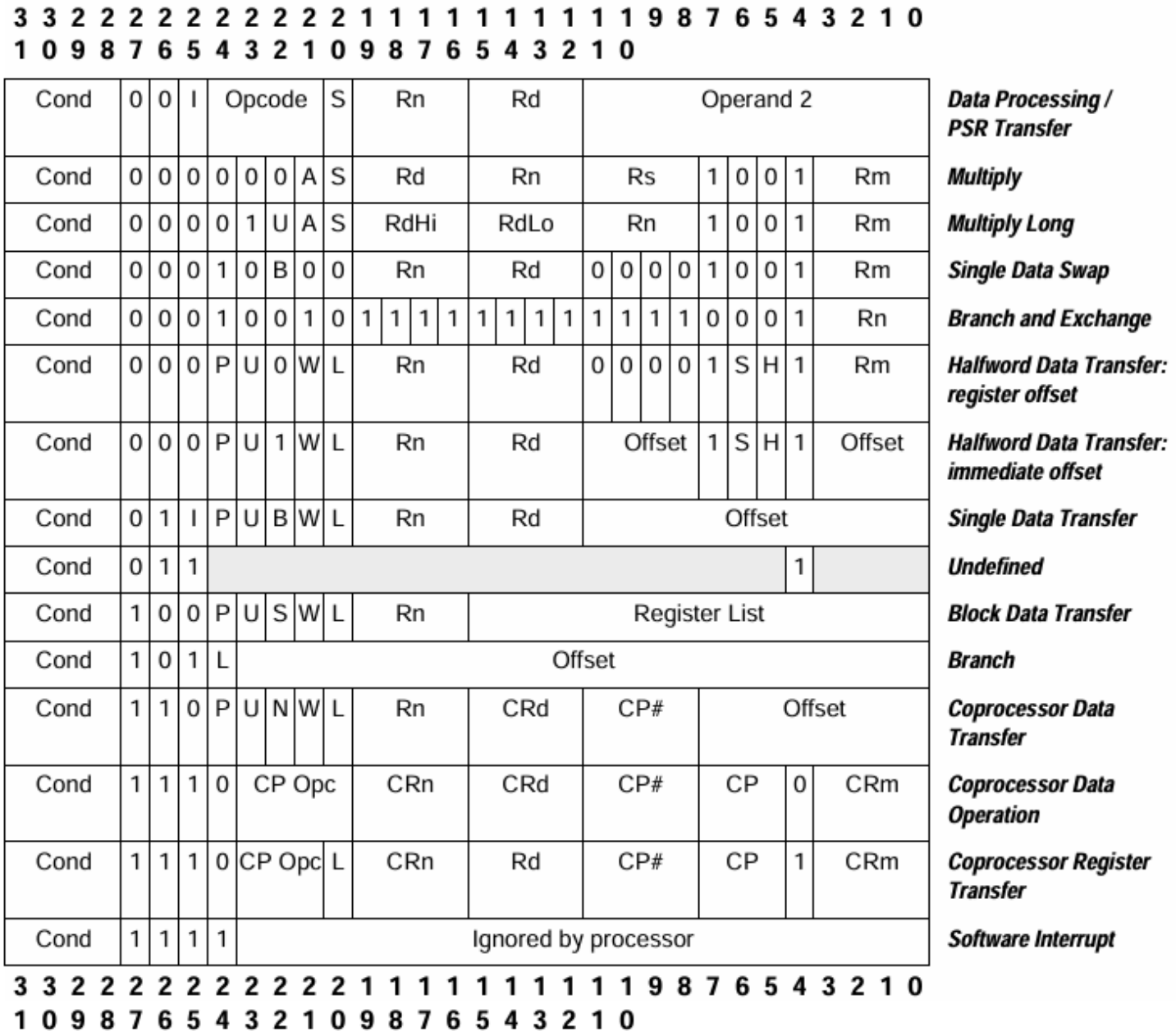
Introduction

Le jeu d'instructions ARM fait référence à l'ensemble des commandes que les processeurs ARM peuvent exécuter.
Comme la majorité des langages "machine", la syntaxe du langage assembleur de ARM dispose, un nombre des champs pour décrire une instruction sur **32 bits**:

Champ 1	Champ 2	Champ 3	Champ 4	Champ 5
Condition	Code opératoire	Résultat	Opérande1	Opérande2

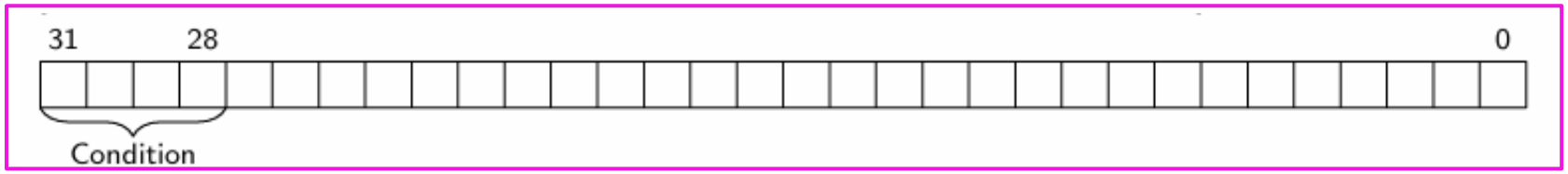
- **Condition** pour l'exécution conditionnelle d'une instruction. Elle est optionnelle à part pour le début d'un sous-programme,
- Le **code opératoire** pour définir l'opération à appliquer sur les opérandes sources,
- Le **résultat** est toujours avant les opérandes car certaines instructions n'ont qu'un seul opérande. Le résultat et les opérandes sont des indices (adresse) de registre contenant des données,
- Les registres **opérandes** pour LOAD et STORE sont des registres d'adresse (adressage indirect),
- L'**opérande2** peut être un registre ou une valeur immédiate.

Format binaire d'instructions ARM



Champ de condition Inst[31-28]

Toutes les instructions sont codées sur 32 bits dont les 4 bits de poids fort (Inst[31-28]) définissent les 16 conditions d'exécution possibles :



0000	**EQ** - $Z = 1$
0001	**NE** - $Z = 0$
0010	**HS**/**CS** - $C = 1$
0011	**LO**/**CC** - $C = 0$
0100	**MI** - $N = 1$
0101	**PL** - $N = 0$
0110	**VS** - $V = 1$
0111	**VC** - $V = 0$

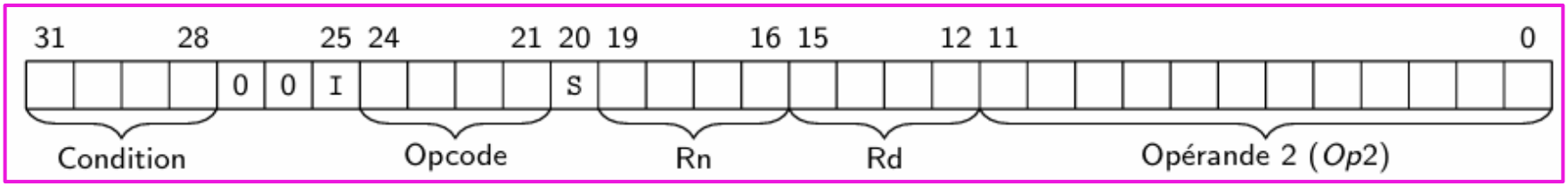
1000	**HI** - $C = 1$ et $Z = 0$
1001	**LS** - $C = 0$ ou $Z = 1$
1010	**GE** - supérieur ou égal
1011	**LT** - strictement inférieur
1100	**GT** - strictement supérieur
1101	**LE** - inférieur ou égal
1110	**AL** - toujours
1111	**NV** - réservé.

Le champ de condition de l'instruction détermine les circonstances dans lesquelles une instruction doit être exécutée. Si l'état des indicateurs N, Z, C et V remplit les conditions codées par le champ, l'instruction est exécutée, sinon elle est ignorée.

Représentation binaire d'instructions

Traitement de données

Instruction[31-0]



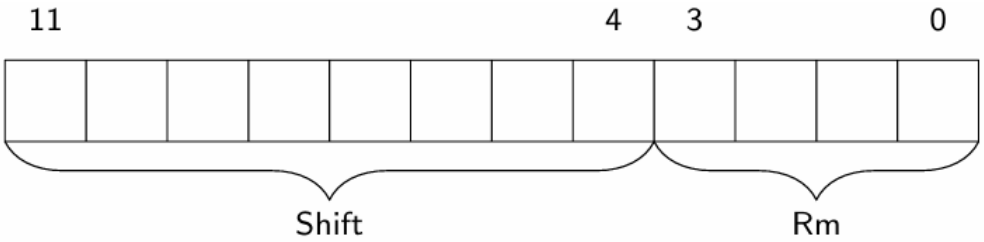
- **Condition** (4 bits): L'instruction n'est exécutée que si la condition sur les flags est satisfaite (Inst[31-28]),
- **I** (1 bit): Indique si l'opérande 2 est du type immédiat (Inst[25]),
- **Opcode** (4 bits): Définit l'opération à réaliser (Inst[24-21]),
- **S** (1 bit): Indique si les flags doivent être mis à jour (Inst[20]),
- **Rn** (4 bits): Registre correspondant au premier opérande(Inst[19-16]),
- **Rd** (4 bits): Registre correspondant à la destination (Inst[15-12]),
- **Opérande 2** (12 bits): Second opérande (Inst[11-0]).

Champ de Opcode Inst[24-21]

Opcode	Instruction	Niveau RTL
0000	AND	Rd <= Rn AND Op2
0001	XOR	Rd <= Rn XOR Op2
0010	SUB	Rd <= Rn - Op2
0011	RSB	Rd <= Op2 - Rn
0100	ADD	Rd <= Rn+Op2
0101	ADC	Rd <= Rn+Op2+C
0110	SBC	Rd<= Rn - Op2+C - 1
0111	RSC	Rd <= Op2 – Rn + C - 1
1000	TST	Positionne les FLags **de registre CPSR** pour Rn AND Op2
1001	TEQ	Positionne les FLags **de registre CPSR** pour Rn XOR Op2
1010	CMP	Positionne les Flags **de registre CPSR** pour Rn - Op2
1011	CMN	Positionne les Flags **de registre CPSR** pour Rn + Op2
1100	ORR	Rd<=Rn OR Op2
1101	MOV	Rd<=Op2
1110	BIC	Rd<=Rn AND NOT Op2
1111	MVN	Rd<=NOT Op2

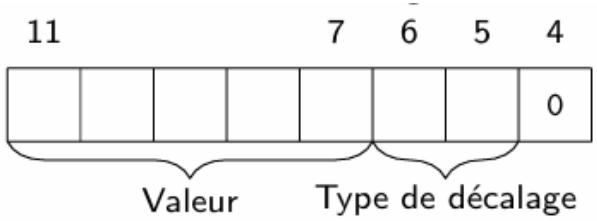
Champ Opérande 2 (registre): Inst[11-0]

Cas ou le champ **I** Inst[25] est égal à 0 => Second opérande est un registre:

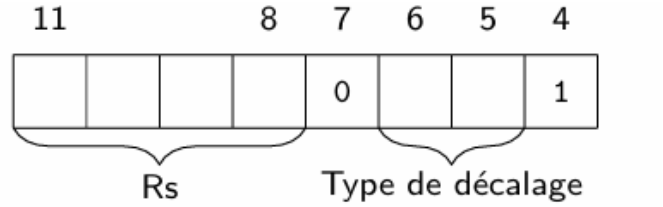


- **Rm** : Registre utilisé comme second opérande;
  - **Shift** : Décalage appliqué au registre Rm.

Cas ou la valeur de décalage '**Shift**' est une valeur immédiate:



Cas ou la valeur de décalage '**Shift**' est un registre:



Il existe 4 **types de décalage** :
00- Décalage à gauche logique,
01- Décalage à droite logique,
10- Décalage à droite arithmétique,
11- Rotation à droite.

La **valeur** maximale pour un décalage est 31 pour un registre de 32 bits,

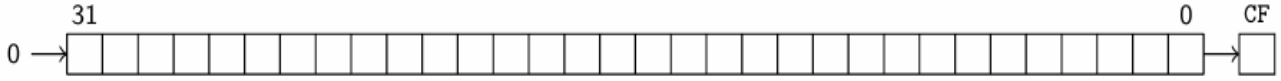
- La valeur est codée sous la forme d'un immédiat sur 5 bits ou une valeur d'un registre Rs.

Décalage 'shift'

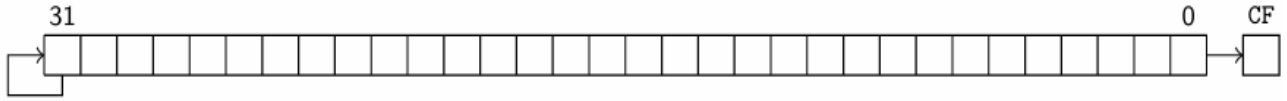
Une instruction peut spécifier 5 modes de décalage de son opérande 2 :

LSL: Logical shift left

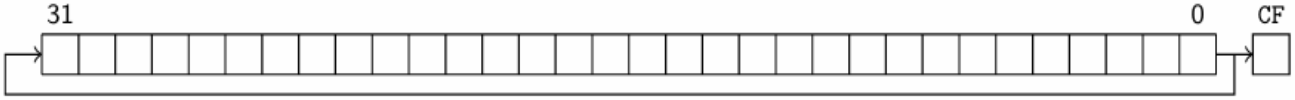
LSR: Logical shift right



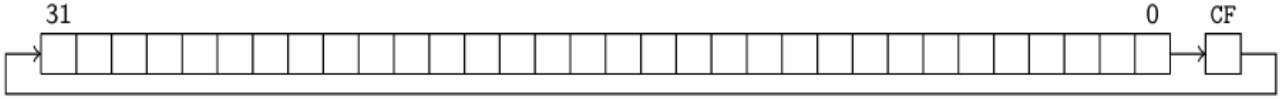
ASR: Arithmetic shift right



ROR: Rotate right



RRX :Rotate right with extend



RRX étendue à 1 bit, codé comme un **ROR** avec un bit supplémentaire qui est bit de report **C** (carry flag) dans le registre de statut.

Exemple : Décalage 'shift'

```
MOV R0, #1        ; Charger la valeur 1 dans le registre R0
LSL R0, R0, #3    ; Décaler les bits de R0 de 3 positions vers la gauche
```

Dans cet exemple, le contenu de R0 (initialement 1= 0001) est décalé de trois positions vers la gauche, ce qui donne la valeur 8=1000.

```
MOV R1, #-64      ; Charger la valeur 16 dans le registre R0
ASR R1, R1, #3    ; Décaler les bits de R1 de 3 positions vers la droite de manière arithmétique
```

Dans cet exemple, la valeur de R1 est initialement -64 (en binaire : 1111 1111 1100 0000 pour un entier signé de 8 bits). Après le décalage arithmétique vers la droite de 3 positions, la valeur devient -8 (en binaire : 1111 1111 1111 1000).

Le décalage arithmétique préserve le signe en reproduisant le bit de signe (le bit le plus à gauche) lors du décalage.

```
MOV R2, #128      ; Charger la valeur 128 dans le registre R2
ROR R2, R2, #3    ; Faire une rotation des bits de R2 de 3 positions vers la droite
```

Dans cet exemple, la valeur de R2 est initialement 128 (en binaire : 10000000). Après la rotation des bits vers la droite de 3 positions, la valeur devient 16 (en binaire : 00010000).

Exemple : Décalage 'shift'

```
MOV R0, #5
MOV R1, #2
LSL R1, R1, #1    ;Décaler les bits de R1 de 1 position vers la gauche, donc R1 devient 4 (en binaire : 00000100)
ADD R2, R0, R1 ;Additionner R0 et R1, et stocker le résultat dans R2(R2= 5+4 = 9)
```

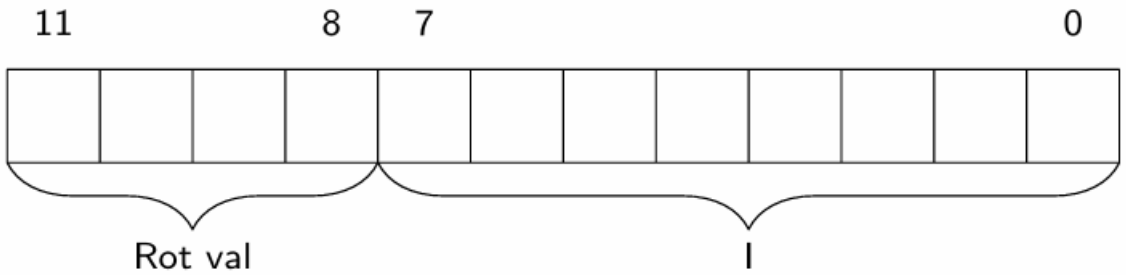
Ou bien

```
ADD R2, R0, R1, LSL #1    ;Décaler les bits de R1 de 1 position vers la gauche, donc R1 devient 4 (en binaire : 00000100) et Additionner R0 et R1, et stocker le résultat dans R2 (R2 = 5 + 4 = 9)
```

```
MOV R3, #12   ; (12 en binaire = 00001100)
MOV R4, #3    ; (3 en binaire = 00000011)
AND R5, R3, R4, LSR #1    ; Décaler les bits de R4 de 1 position vers la droite, donc R4 devient 1 (en binaire : 00000001) et Effectuer l'opération AND entre R3 et R4 bit à bit, et stocker le résultat dans R5 (R5 = 00001100 AND 00000001 = 00000000, donc R5 = 0)
```

Champ Opérande 2 (immédiat): Inst[11-0]

Cas ou le champ **I** Inst[25] est égal à 1:



- **I** : Valeur utilisée comme second opérande,
- **Rot val** : Champ de rotation, spécifiant combien de bits la valeur immédiate doit être tournée à droite.

Exemple :

- **ORR r0, r1, #0x1, ROR #2**

Cond		I	OpCode	S	Rn	Rd	Operand2
1110	00	1	1100	0	0001	0000	0010 0000 0001

- **OpCode** (1100) : est le code pour ORR.
- **Operand2 (0010 0000 0001)** : Valeur immédiate 0x1 (0000 0001) avec une rotation de 2 bits à droite (0010).

Exemples : codage binaire d'instructions

- **ADDEQ r0, r1, r2**

Cond		I	OpCode	S	Rn	Rd	Operand2
0000	00	0	0100	0	0001	0000	0000 0000 0010

- **Cond** (0000) Indique que l'instruction est exécutée si la condition EQ (égal) est vraie.
- **OpCode** (0100) est le code pour ADD.
- **S** (0) indique que les drapeaux de condition ne sont pas mis à jour.
- **Rn** (0001) représente le registre source r1.
- **Rd** (0000) représente le registre destination r0.
- **Operand2** (0000 0000 0010) représente le registre r2.

- **MOV r0, #1**

Cond		I	OpCode	S	Rn	Rd	Operand2
1110	00	1	1101	0	0000	0000	0000 0000 0001

- **Cond** (1110) : indique que l'instruction est toujours exécutée.
- **OpCode** (1101) est le code pour MOV.
- **Operand2** (0000 0000 0001) représente la valeur immédiate 1.

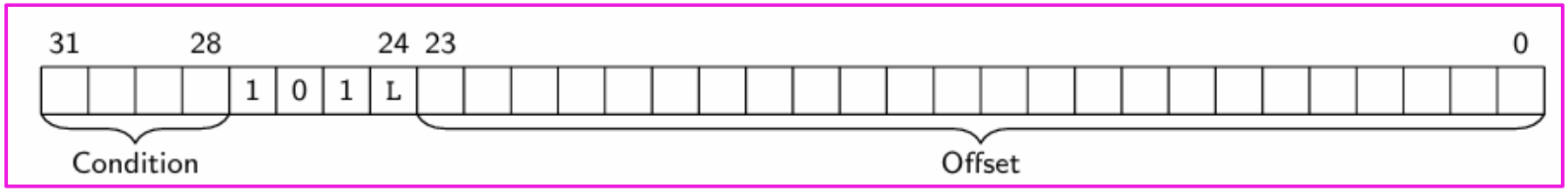
- **AND r0, r1, r2, LSL #2**

Cond		I	OpCode	S	Rn	Rd	Operand2
1110	00	1	0000	0	0001	0000	0001 0000 0010

- **OpCode** (0000) : est le code pour AND.

Représentation binaire d'instructions

Branchement



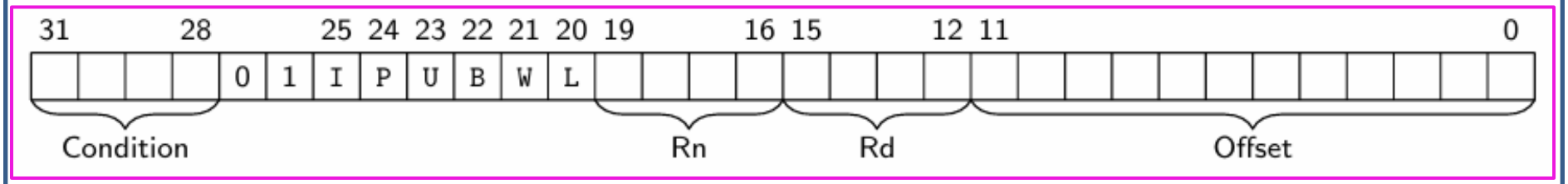
- **Condition** : condition d'exécution de l'instruction;
- **L** : Si Link = 1 R14 <=PC +4;
  Link= 0 branchement simple,
- **Offset** : PC <=PC +8+(Offset*4).

- Le retour se fait en rechargeant le registre LR dans le registre PC.
- Pour les fonctions non terminales, LR devra être sauvegardé.

Le LR (R14) doit lui-même être empilé lorsqu'il y a des appels sous-programmes imbriqués. L'instruction de dépilage à la fin du sous-programme permet de recharger le PC avec la valeur de LR.

Représentation binaire d'instructions

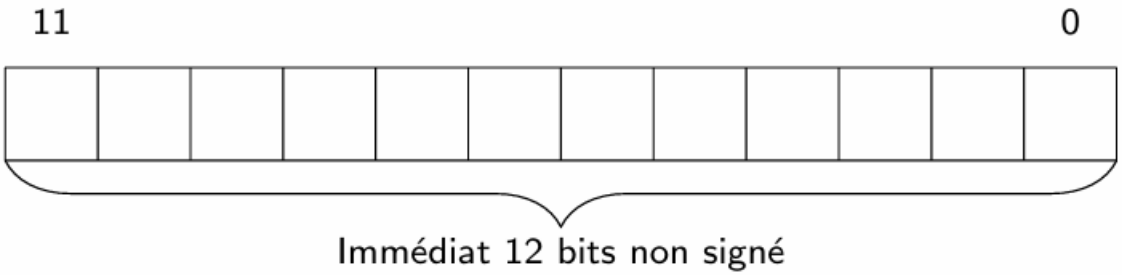
Accès mémoires simples



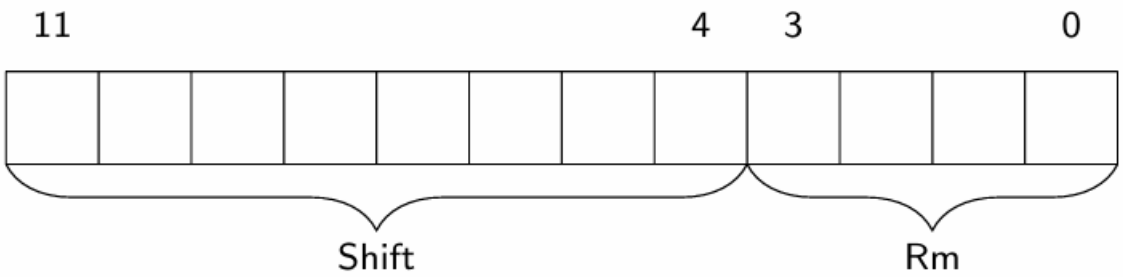
- **Condition** : L'instruction n'est exécutée que si la condition sur les flags est satisfaite;
  - **I** : L'Offset correspond à un immédiat si égal à 1;
  - **P** : Pre/Post indexation (Pré si égal à 1);
  - **U** : Up/Down ajout de l'Offset si égal à 1;
  - **B** : Byte/Word octet si égal à 1;
  - **W** : Write-back modification adresse de base si égale à 1;
  - **L** : Load/Store lecture mémoire si égal à 1;
- **Rn** : Registre de base (adresse);
- **Rd** : Registre source (écriture) ou destination;
- **Offset** : Immédiat ou registre combiné au registre de base pour constituer l'adresse.

Champ offset: Inst[11-0]

Cas ou le champ I Inst[25] est égal à 1 :



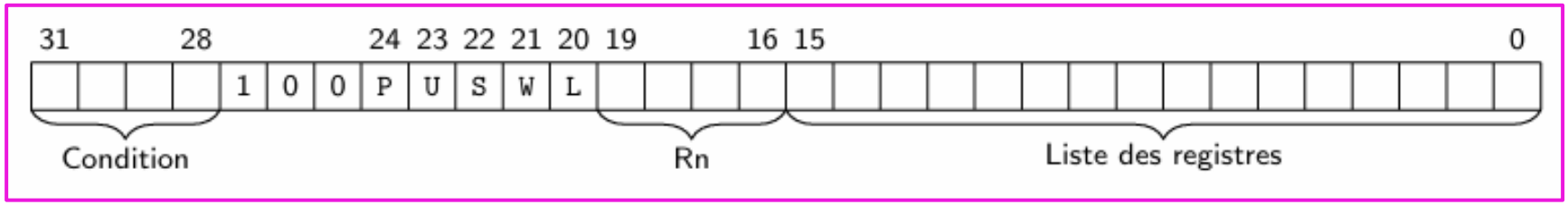
Cas ou le champ I Inst[25] est égal à 0 :



- **Même codage que pour les instructions de traitement de donnée (opérande 2).**

Représentation binaire d'instructions

Accès mémoires multiples



Condition : condition d'exécution de l'instruction;

P : Pré/Post indexation (Pré si égal à 1);

U : Up/Down ajout de l'Offset si égal à 1;

S : Voir spécification détaillée;

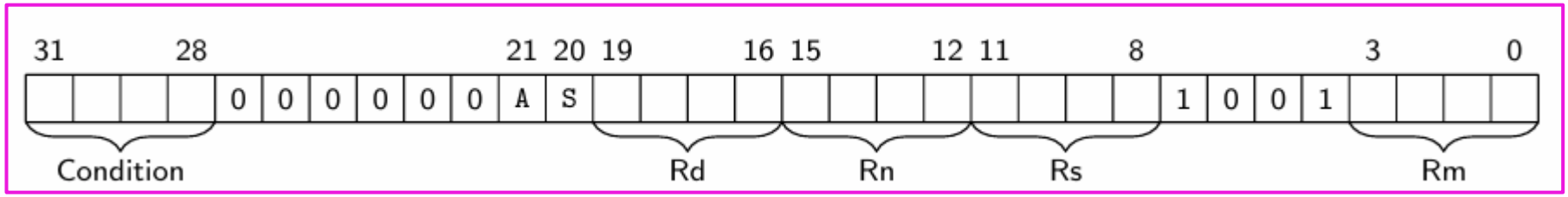
W : Write-back modification adresse de base si égale à 1;

L : Load/Store lecture mémoire si égal à 1;

Rn : Registre de base (adresse);

Liste : Liste des registres source/destination.

Représentation binaire d'instructions
Multiplication



- **Condition** : condition d'exécution de l'instruction;
  - A : Rd <=Rm x Rs+Rn si égal à 1
    sinon Rd <= Rm x Rs.
  - S : la mise à jour des flags, si égal à 1.
- **Rd** : Registre de destination;
- **Rn, Rs et Rm** : Registres source;