

Systemes à Microprocesseurs

Cycle Ingénieur Troisième Année

Sébastien Bilavarn



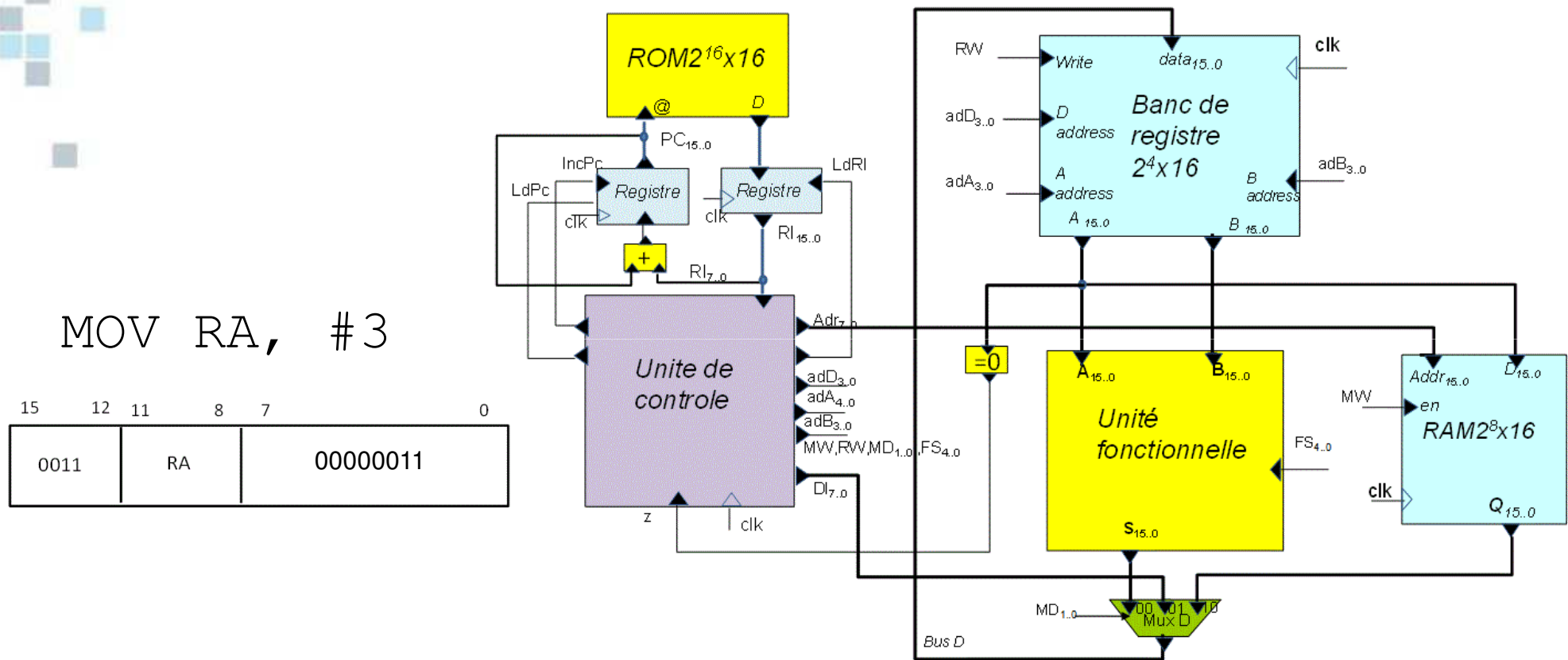
Plan

- 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

ARM Instruction Set Architecture

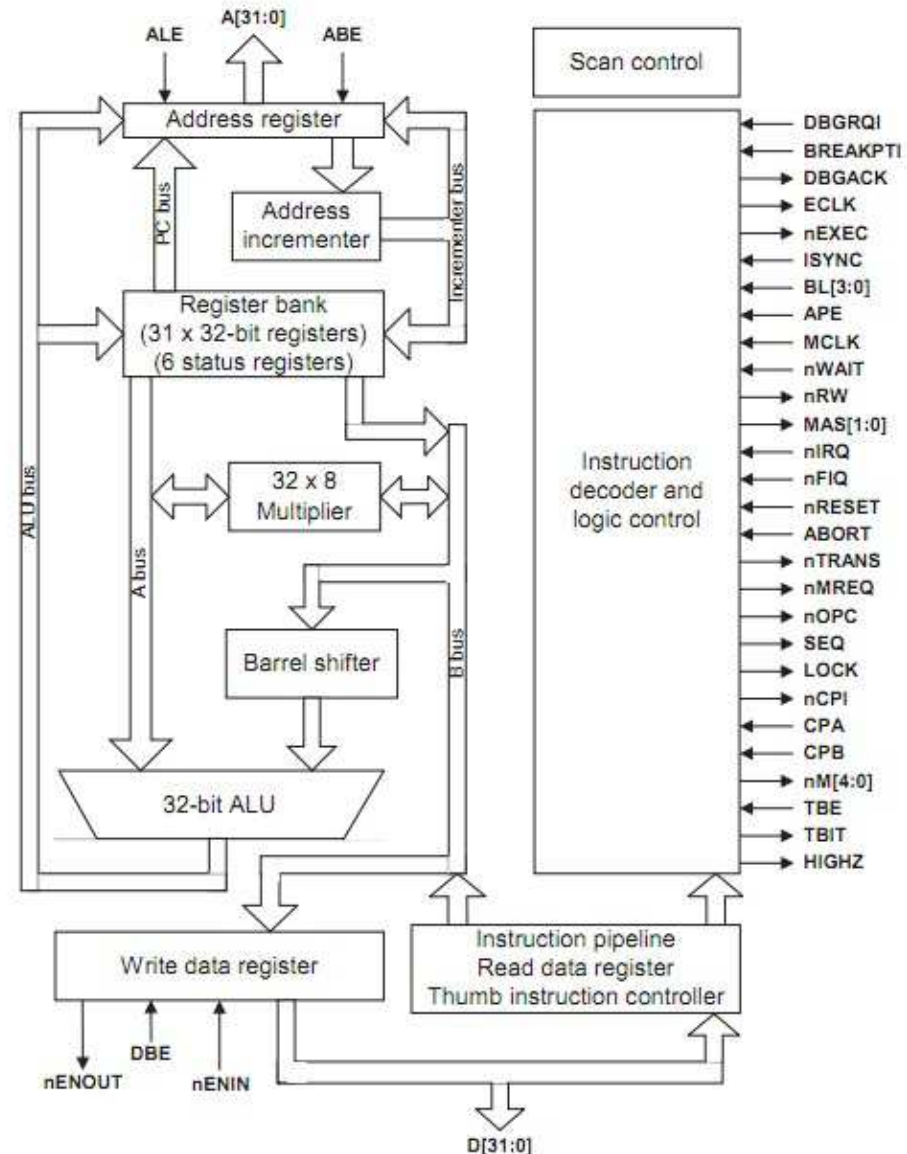
- Caractéristiques du processeur
 - Unité de traitement
 - Jeu d'instruction
- Structure d'un programme assembleur

Exemple Microprocesseur 16-bit



Architecture ARM7 TDMI

- Processeur 32-bit
- UAL 32-bit
- File de registres
- Registre à décalage
- Multiplieur 32x8





Instruction Set Architecture

- Architecture
 - Structure
- Instructions
 - Le fonctionnement est lié à l'architecture
- Le jeu d'instruction et son utilisation sont étroitement liés à l'architecture.
 - Certaines choses sont possibles et d'autre pas.
 - La connaissance de l'architecture et du fonctionnement sont nécessaires pour écrire du code et pour son optimisation



Caractéristiques générales

- Architecture load-store
 - Les instructions ne traitent que des données en registre et placent les résultats en registre. Les seules opérations accédant à la mémoire sont celles qui copient une valeur mémoire vers un registre (load) et celles qui copient une valeur registre vers la mémoire (store).
- Format de codage fixe des instructions
 - Toutes les instructions sont codées sur 32 bits.
- Format 3 adresses des instructions de traitement
 - Deux registres opérandes et un registre résultat, qui peuvent être spécifiés indépendamment.
- Exécution conditionnelle
 - Chaque instruction peut s'exécuter conditionnellement
- Instructions spécifiques de transfert
 - Instructions performantes de transfert multiples mémoire – registre
- UAL + shift
 - Possibilité d'effectuer une opération Arithmétique ou Logique et un décalage en une instruction (1 cycle), la ou elles sont réalisées par des instructions séparées sur la plupart des autres processeurs

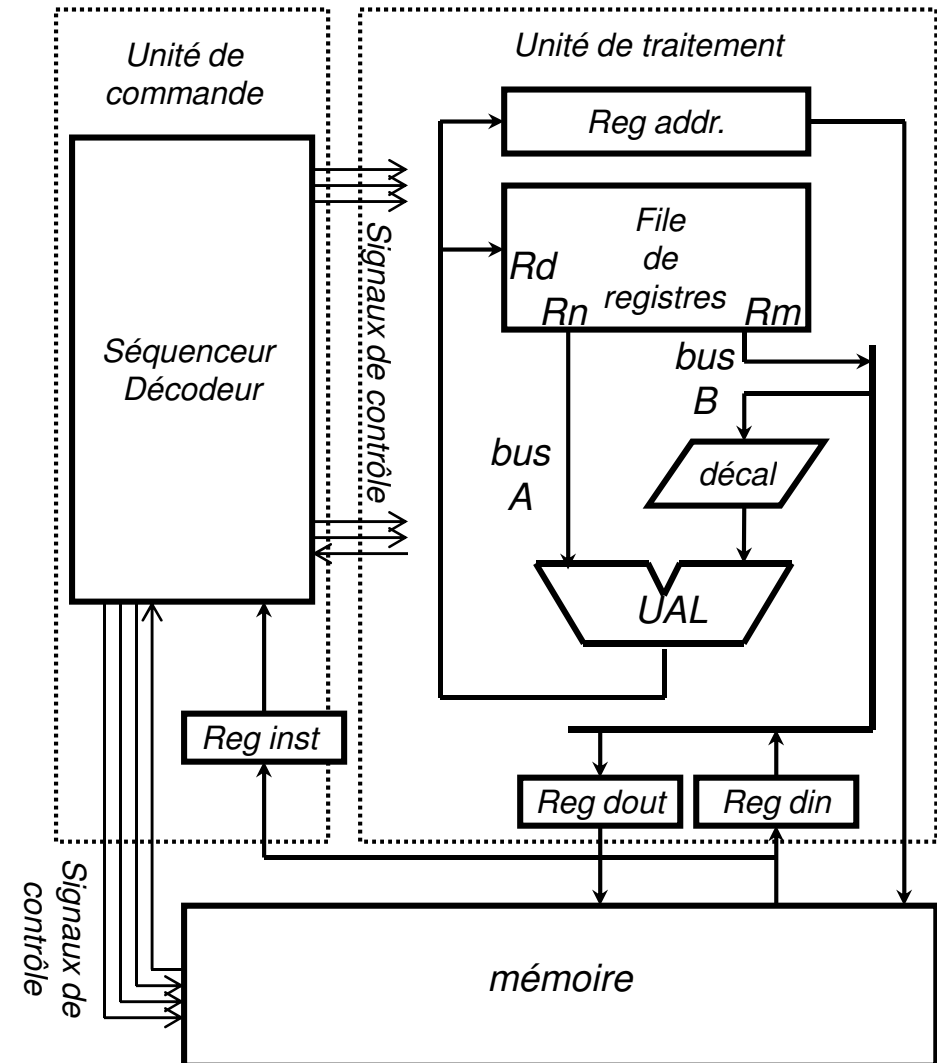
ARM Instruction Set Architecture

- Caractéristiques du processeur
 - **Unité de traitement**
 - Jeu d'instruction
- Structure d'un programme assembleur

Unité de traitement

Unités fonctionnelles de l'UT:

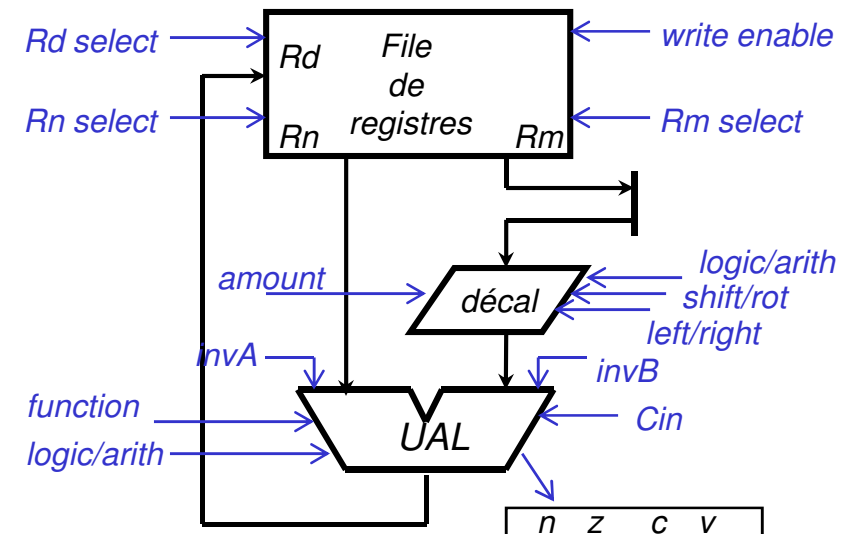
- File de registres
 - 16 registres pour permettre une manipulation souple des données et stockage des résultats de l'UAL
- Unité Arithmétique et Logique
 - 2 opérandes : entrée A donnée provenant d'un registre, entrée B donnée reliée au décaleur
 - Résultat de l'UAL: renvoyé dans un registre
- Registre à décalage
 - Opérations de décalage (1 décalage à gauche = $\times 2$, 1 décalage à droite = $/2$)
 - Opérations de rotation (décalage + ré-injection du bit perdu)
 - Associé à l'entrée B de l'UAL pour réaliser une instruction UAL + shift en 1 cycle



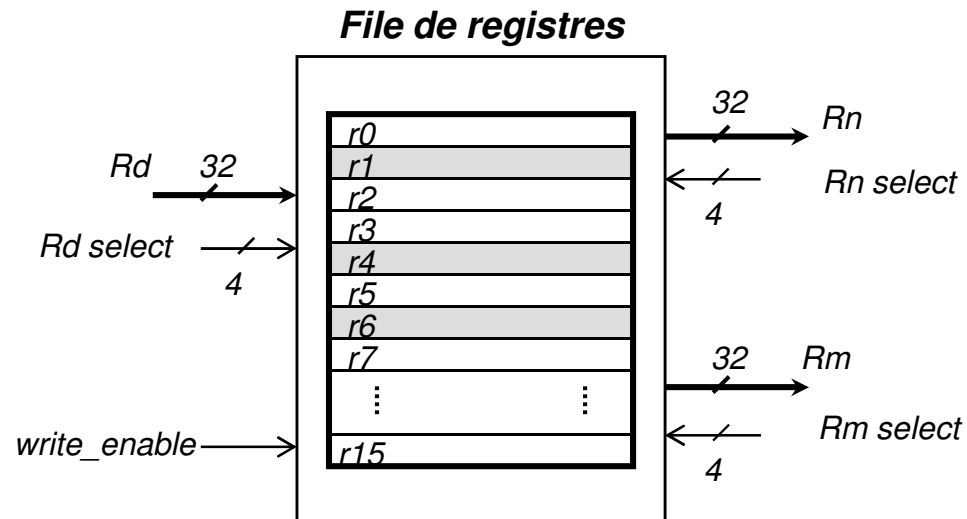
Organisation de l'unité de traitement

■ Signaux de commande de l'unité de traitement:

- File de registres
 - Rd, Rn, Rm select : sélection du registre cible dans la file des 16 registres
 - write enable: activation de la file de registres
- UAL
 - logic/arith+function: mode logique ou arithmétique, dans chaque mode choix de la fonction
 - InvA, invB: inversion des opérandes A et B (not)
 - Cin: injection du bit retenue C
 - Indicateurs: indiquent l'état de l'UAL après une opération (ex: retenue C)
- Décaleur
 - shift/rot: opération de décalage ou rotation
 - logic/arith: opération logique ou arithmétique
 - Amount: nombre de bits de décalage/rotation



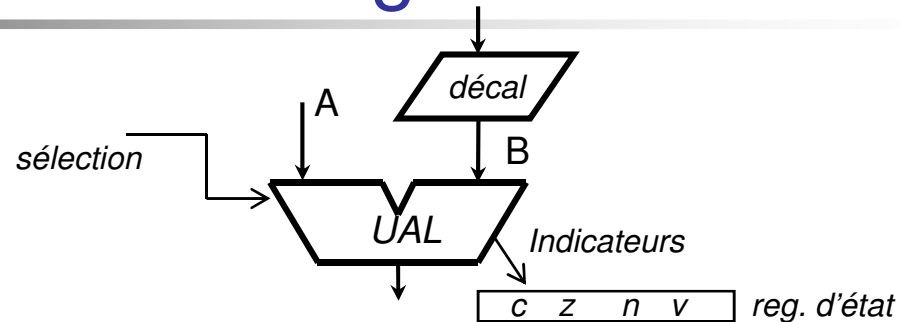
File de registres



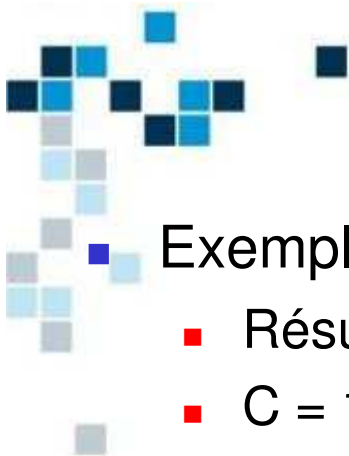
- 16 registres utilisateurs $r0, \dots, r15$
- Notation pour l'appel aux instructions:
 - INST Rd, Rn, Rm
 - Format 3 adresses (2 opérandes + résultat)
 - Rn registre opérande 1, relié au port A (32 bits)
 - Rm registre opérande 2, relié au port B (32 bits) par l'intermédiaire du décaleur → possibilité de rotation/décalage sur entrée B
 - Rd registre de destination (32 bits)
 - Select 4 bits pour le choix du registre parmi les 16 disponibles



UAL et registre à décalage



- Effectue des opérations arithmétiques et logiques
 - (ADD, SUB, AND, OR, ...)
- Associée au registre à décalage en entrée B
- Le registre d'état (SR) fournit des indications sur les résultats d'opération:
 - C: Carry
 - bit indicateur de dépassement pour une opération arithmétique (ou de décalage)
 - Z: Zero
 - bit indicateur de résultat nul de l'UAL
 - N: Négatif
 - bit indicateur de résultat négatif de l'UAL
 - V: Débordement (oVerflow)
 - bit indicateur de dépassement de capacité du résultat de l'UAL (modification du bit de signe)



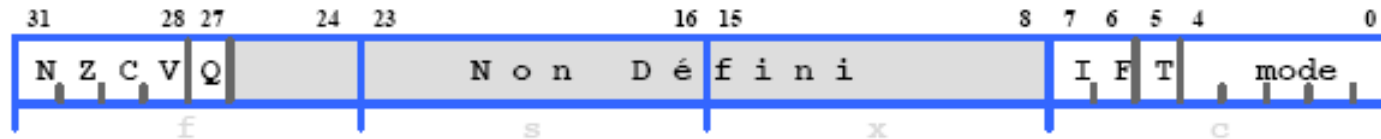
UAL et registre à décalage

- Exemple sur 4 bits : $1\ 0\ 1\ 0 + 1\ 0\ 0\ 1 = (1)\ 0\ 0\ 1\ 1$
 - Résultat de l'UAL: $0\ 0\ 1\ 1$
 - $C = 1$
 - $Z = 0$, le résultat est différent de $0\ 0\ 0\ 0$
 - $N = 0$, $0\ 0\ 1\ 1$ est un nombre positif car le bit de signe (4^{ème} bit) = 0
 - $V = 1$, car $1\ 0\ 1\ 0$ et $1\ 0\ 0\ 1$ sont des nombre négatifs et le résultat est positif

- les bits C, V, N sont examinés ou ignorés en fonction de l'interprétation des nombres
 - Si les nombres sont en représentation non-signée, C est utile, V et N inutiles
 - Si les nombre sont en représentation signée, C est inutile, V et N sont utiles
 - Ces indicateurs sont positionnés par l'UAL, le programmeur les utilise ou non en fonction de ses besoins (par exemple pour effectuer une addition sur 8 bits à partir de l'addition 4 bits dans le cas de l'exemple)

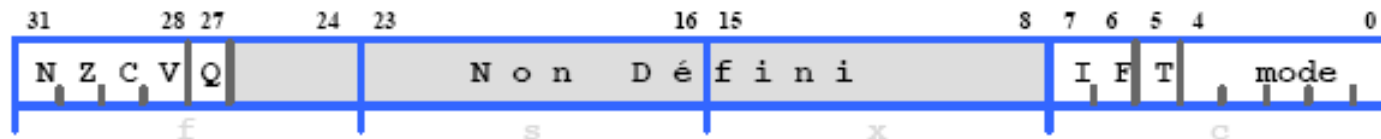
Le registre d'état

- CPSR: Current Program Status Register
- Contient les indicateurs Z (zero), N (negative), C (carry), V (overflow)
- Donne des informations sur le résultat d'une opération arithmétique ou d'une comparaison
- Permet à une instruction de s'exécuter ou non en fonction de ce résultat (exécution conditionnelle)
- Permet de conserver la retenue pour effectuer des opérations sur plus de 32-bit



Le registre d'état

- Indicateurs conditionnels
 - Z résultat nul
 - N résultat négatif
 - C retenue
 - V Débordement
- Q débordement
 - Indique un type de débordement particulier (arithmétique saturée)
- Zone non définie
- Validation des interruptions
 - I=1 dévalide IRQ
 - F=1 dévalide FIQ
- Mode thumb
 - T=0 mode ARM (32 bits)
 - T=1 mode thumb (16 bits)
- Indicateur de mode
 - Indiquent le mode actif: système, IRQ, FIQ, utilisateur...





UAL et registre à décalage

- Mnémonique pour décalage (*shift*)
 - LSL $\#n$: Logical Shift Left ($0 \leq n \leq 31$)
 - LSR $\#n$: Logical Shift Right ($1 \leq n \leq 32$)
 - ASR $\#n$: Arithmetic Shift Right ($1 \leq n \leq 32$)
 - ROR $\#n$: Rotate Right ($1 \leq n \leq 31$)
 - RRX: Rotate Right Extended (ROR étendu de 1 bit avec le bit de retenue C du registre d'état)
- Décalage ($>>$, $<<$): déplacement du mot vers la droite ou vers la gauche. Les positions libérées sont remplies avec des zéros (logique) ou avec le bit de signe (arithmétique).
 - Ex1: 0111, LSR #1 \rightarrow 0011, LSL #1 \rightarrow 1110
 - Ex2: 1011, ASR #1 \rightarrow 1101, ASL #1 \rightarrow 0110 pas d'extension de signe possible, identique à LSL #1
- Rotation: décalage circulaire, les bits perdus sont réinjectés
 - Ex: 0111, ROR #1 \rightarrow 1011
 - Avec retenue: rotation effectuée sur un bit de plus (C), ex (0) 0111 RRX \rightarrow (1) 0011



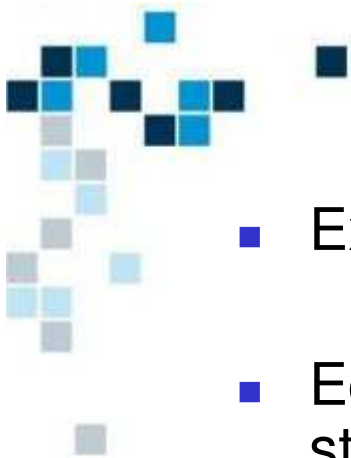
UAL et registre à décalage

■ Ex: l'instruction d'addition

- `ADD Rd, Rn, #literal`
- `ADD Rd, Rn, Rm`
- `ADD Rd, Rn, Rm, shift`

■ Exemples

- `ADD r3, r2, #1` @ $r3 \leftarrow r2 + 1$
 - Opérande 2 (Rm) = constante
- `ADD r3, r2, r5` @ $r3 \leftarrow r2 + r5$
- `ADD r3, r2, r5, LSL #2` @ $r3 \leftarrow r2 + (r5 \ll 2)$
 - opérande 2 (Rm) = opérande décalé.
 - Multiplie par 4 la valeur de r5, ajoute la valeur de r2 et stocke le résultat dans r3



UAL et registre à décalage

- Exercice
- Ecrire une séquence qui multiplie par 10 la valeur de r5 et stocke le résultat dans r6
 - $10r5 = 8r5 + 2r5$
 - 1) multiplier par 8 = 3 décalages à gauche
 - `MOV r1, r5, LSL #3` @ $r1 \leftarrow (r5 \ll 3)$
 - 2) multiplier par 2 = 1 décalage à gauche
 - `MOV r2, r5, LSL #1` @ $r2 \leftarrow (r5 \ll 1)$
 - 2) ajouter:
 - `ADD r6, r1, r2` @ $r6 \leftarrow r1 + r2$
 - Plus efficace:
 - `MOV r1, r5, LSL #3` @ $r1 \leftarrow (r5 \ll 3)$
 - `ADD r6, r1, r5, LSL #1` @ $r6 \leftarrow r1 + (r5 \ll 1)$

ARM Instruction Set Architecture

- Caractéristiques du processeur
 - Unité de traitement
 - Jeu d'instruction
- Structure d'un programme assembleur



Instructions de traitement

■ Opérations arithmétiques

Instruction	Description	Commentaire
ADD Rd, Rn, <i>operand2</i>	Add	$Rd := Rn + operand2$
ADC Rd, Rn, <i>operand2</i>	Add with carry	$Rd := Rn + operand2 + C$
SUB Rd, Rn, <i>operand2</i>	Subtract	$Rd := Rn - operand2$
SBC Rd, Rn, <i>operand2</i>	Subtract with carry	$Rd := Rn - operand2 + C - 1$
RSB Rd, Rn, <i>operand2</i>	Reverse subtract	$Rd := operand2 - Rn$
RSC Rd, Rn, <i>operand2</i>	Rev. sub. With carry	$Rd := operand2 - Rn + C - 1$

- Réalisent des opérations arithmétiques binaires sur 2 opérandes 32 bits
- Addition, soustraction, soustraction inverse (l'ordre des opérandes est inversé)
- C: valeur du bit de retenue dans le registre d'état CPSR



Instructions de traitement

■ Opérations logiques

Réalisent des opérations logiques sur chaque paire de bits des 2 opérandes: ET, OU, OU EX et BIT CLEAR

Instruction	Description	Commentaire
AND Rd, Rn, <i>operand2</i>	And	Rd := Rn and <i>operand2</i>
ORR Rd, Rn, <i>operand2</i>	Or	Rd := Rn or <i>operand2</i>
EOR Rd, Rn, <i>operand2</i>	Exclusive or	Rd := Rn xor <i>operand2</i>
BIC Rd, Rn, <i>operand2</i>	Bit clear	Rd := Rn and not <i>operand2</i>

■ Mouvement de contenu de registres

Instruction	Description	Commentaire
MOV Rd, <i>operand2</i>	Move	Rd := <i>operand2</i>
MVN Rd, <i>operand2</i>	Move not	Rd := not <i>operand2</i>

Instructions de traitement

- Pour chaque instruction précédente

- Rd = registre de destination

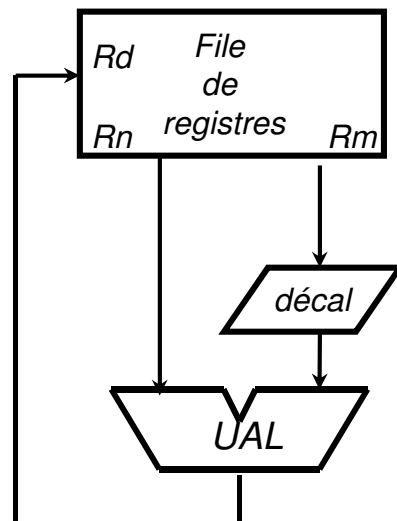
- Rn = 1er opérande

- *operand2* =

- *#literal*

- Rm

- Rm, *shift*



Le premier opérande Rn est toujours un registre.

Le second opérande peut être une constante littérale (valeur dite immédiate, précédée par #)

Ex1: ADD r3, r3, #1 @ bien que le format 3 adresses permet de spécifier 3 registres différents, ils ne sont pas obligatoirement distincts

Le second opérande peut être le contenu d'un registre

Le second opérande peut être le contenu d'un registre sujet à une opération de décalage (pas le cas du premier opérande). Les deux opérations (décalage + instruction) sont effectuées en une seule instruction (1 cycle d'horloge)

Il n'y a pas d'instruction de décalage séparée en assembleur ARM. Elles sont toujours combinées à une autre instruction (ADD, MOV etc)

Par défaut, le registre d'état n'est pas mis à jour.



Instructions de traitement

■ Opérations de comparaison

Instruction	Description	Commentaire
CMP Rn, <i>operand2</i>	Compare	$CPSR \leftarrow Rn - operand2$
CMN Rn, <i>operand2</i>	Compare negative	$CPSR \leftarrow Rn + operand2$
TST Rn, <i>operand2</i>	Test	$CPSR \leftarrow Rn \text{ and } operand2$
TEQ Rn, <i>operand2</i>	Test equal	$CPSR \leftarrow Rn \text{ xor } operand2$

Les opérations de comparaison ne produisent pas de résultat, elles prennent deux opérandes en paramètre et positionnent les indicateurs NZCV en fonction de l'opération sélectionnée (CMP, CMN, TST, TEQ)



Instructions modifiant CPSR

- Instructions de comparaison
- Syntaxe

- **CMP Rn, operand2** *Compare*
 - Effectue la soustraction du premier et deuxième opérande et met à jour le registre d'état (le résultat n'est pas conservé)
 - Positionne Z, N, C, V en fonction de l'opération $r1 - r2$
- **CMN Rn, operand2** *Compare negated*
 - Effectue l'addition du premier et du deuxième opérande et met à jour le registre d'état (le résultat n'est pas conservé)
 - Positionne Z, N, C, V en fonction de l'opération $r1 + r2$
- **Operand2 =**
 - *#literal*
 - Rm
 - Rm, *shift*
- **Exemple**
 - MOV r4, #12 @ r4 <- 12
 - MOV r5, #15 @ r5 <- 15
 - CMP r4, r5 @ compare 12 et 15
 - Résultat: N = 1, Z = 0, C = 0, V = 0



Instructions modifiant CPSR

- Instructions de test

- Syntaxe

- TST Rn, *operand2*

Test

- Affecte les bits N, Z et C après une opération ET logique bit à bit entre les 2 opérandes.
 - Positionne Z, N, C, V en fonction de l'opération Rn AND *operand2*
 - Peut servir à tester si plusieurs bit d'un registre sont à 0.

- TEQ Rn, *operand2*

Test Equivalence

- Affecte les bits N, Z et C après une opération OU exclusif bit à bit entre les 2 opérandes.
 - Positionne Z, N, C, V en fonction de l'opération Rn XOR *operand2*
 - Peut servir à déterminer si les bits de deux valeurs sont les mêmes.

- Operand2 =

- *#literal*
 - Rm
 - Rm, *shift*

Exemple: TST r2, #3 (Bit test)

3 = 0x0011, le « et » renvoie les 2 bits de droite étendus à gauche par des 0, si Z=1 les 2 bits de poids faible sont à 0



Instructions modifiant CPSR

- Autres instructions pouvant affecter le registre d'état CPSR:
 - Toutes les instructions qui réalisent des opérations arithmétiques et logiques
 - En assembleur ARM, il suffit d'ajouter au mot-clé de l'instruction le suffixe " S "
 - Exemples:
 - ADD r1, r2, r3 @ n'affecte pas le registre d'état
 - ADDS r1, r2, r3 @ affecte N, Z, C, V en fonction du résultat
 - utilité des bits N, Z, C, V et de cette forme (ADDS) pour le calcul arithmétique.
 - Utilité la plus importante du registre d'état: exécuter ou non une ou plusieurs instructions en fonction de l'évaluation d'une condition.



Instructions conditionnelles

- Sur ARM, toutes les instructions peuvent s'exécuter de façon conditionnelle: une instruction sera exécutée ou non en fonction de l'état des bits N, Z, C, V
 - En assembleur ARM, il suffit d'ajouter au mot-clé de l'instruction un suffixe qui représente sa condition d'exécution

Extension Mnémonique	Interprétation	Etat indicateur
EQ	Equal/equals zero	Z = 1
NE	Not Equal	Z = 0
CS/HS	Carry set/unsigned higher or same	C = 1
CC	Carry clear/unsigned lower	C = 0
MI	Minus/negative	N = 1
PL	Plus/positive or zero	N = 0



Instructions conditionnelles

- Exécution conditionnelle (suite)

Extension Mnémonique	Interprétation	Etat indicateur
VS	Overflow	$V = 1$
VC	No overflow	$V = 0$
HI	Unsigned higher	$C = 1$ and $Z = 0$
LS	Unsigned lower or same	$C = 0$ or $Z = 1$
GE	Signed greater than or equal	$N = V$
LT	Signed less than	$N \neq V$
GT	Signed greater than	$Z = 0$ and $N = V$
LE	Signed less than or equal	$Z = 1$ or $N \neq V$



Instructions conditionnelles

- Exemple d'exécution conditionnelle en fonction du résultat d'une comparaison

```
CMP r4, r5           @ comparer r4 et r5
SUBGT r4, r4, r5      @ si >, alors r4 ← r4 - r5
SUBLE r5, r5, r4      @ si ≤, alors r5 ← r5 - r4
```

- Exemple d'exécution conditionnelle en fonction du résultat d'un calcul

```
SUBS r4, r4, #10      @ r4 ← r4 - 10
MOVPL r5, #1          @ si r4>0, r5 ← +1
MOVMI r5, #-1         @ si r4<0, r5 ← -1
MOVEQ r5, #0          @ si r4=0, r5 ← 0
```

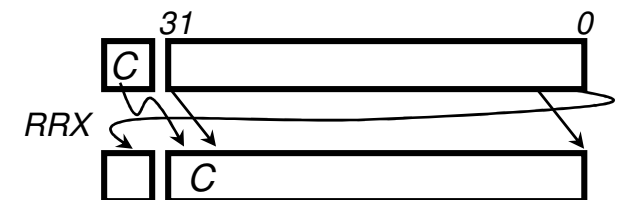
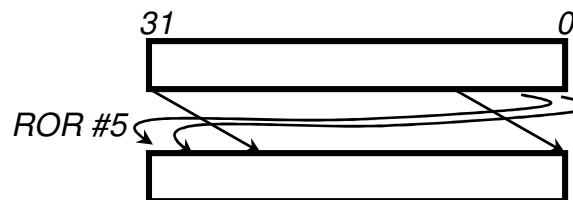
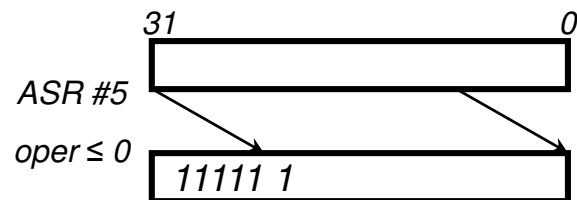
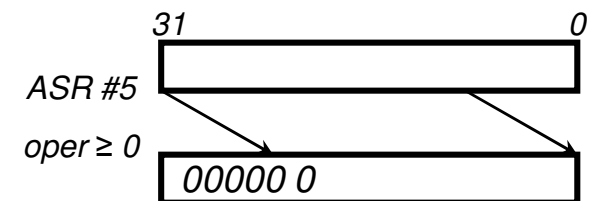
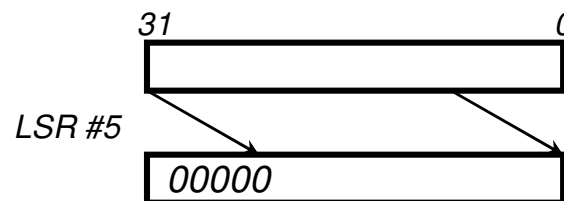
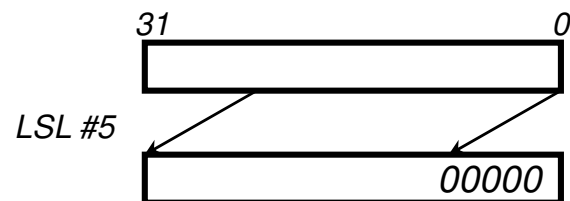
- Exemple de branchement conditionnel

```
ADDS r6, r4, r5       @ r6 ← r4 + r5
BLVS ErrDébordement   @ Branch and Link (BL) si VS
                      @ si débordement (VS),
                      @ appel du sous-programme
                      @ ErrDébordement
```

Instructions de traitement

■ Rotations/décalages

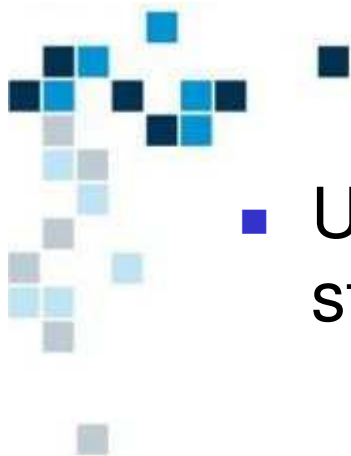
Instruction	Description	Commentaire
LSL # <i>n</i>	Logical shift left	$0 \leq n \leq 31$
LSR # <i>n</i>	Logical shift right	$1 \leq n \leq 32$
ASL # <i>n</i>	Arithmetic shift left	Même fonction que LSL
ASR # <i>n</i>	Arithmetic shift right	$1 \leq n \leq 32$
ROR # <i>n</i>	Rotate right	$1 \leq n \leq 31$
RRX	Rotate right exented	Rotation 1 bit vers la droite utilisant le bit C de CPSR





Instructions de multiplication

- La multiplication de deux valeurs 32 bits produit un résultat sur 64 bits.
 - MUL: range les 32 bits de poids faible dans le registre de destination.
 - `MUL r4, r0, r1` @ $r4 \leftarrow r0 * r1$
 - SMUL, UMUL:
 - `SMUL r4, r5, r0, r1` @ $[r5, r4] \leftarrow r0 * r1$
- La multiplication de nombres signés est différente de la multiplication de nombres non signés.
 - MUL/SMUL: nombres signés
 - UMUL: nombres non signés



Instructions de branchement

- Une exécution normale utilise des instructions stockées à des adresses mémoire consécutives.
- Une instruction de branchement provoque un branchement (saut) vers une adresse mémoire différente
 - soit de manière permanente (branch B)
 - soit de manière temporaire (branch and link BL).

Une instruction de branchement détermine quelle est la prochaine instruction à exécuter. Le mécanisme d'exécution des instructions repose sur le registre PC (Program Counter) qui contient l'adresse de l'instruction suivante à exécuter. Dans une exécution normale il est incrémenté de 4 à chaque instruction.

2 types de branchements: branchement simple (et permanent):le processeur poursuit l'exécution à une adresse différente de l'adresse consécutive. Branchement temporaire, avec un retour vers l'instruction à l'adresse consécutive. Ce type de branchement est utilisé pour l'appel de sous programmes où l'exécution doit être reprise à l'endroit où elle a été interrompue à la fin de l'exécution du sous programme. Dans ce cas, il est nécessaire de sauvegarder de l'adresse de retour (dans le registre LR, Link Register) pour retrouver la séquence originale d'instructions.



Branchements simples

- Branch (B)

```
B          LABEL @ branchement non conditionnel...  
MOV        R1, R2  
LABEL ...           @ ... jusqu'à ici
```

- Quand le processeur rencontre une instruction de branchement, il procède directement à l'instruction suivant l'étiquette `LABEL` au lieu d'exécuter l'instruction située directement après la branche `B` (`MOV R1, R2`).



Branchements simples (conditionnels)

■ Exemple: C

```
tmp = 0;
```

```
For (i=0; i<5; i++)
```

```
    tmp += i;
```

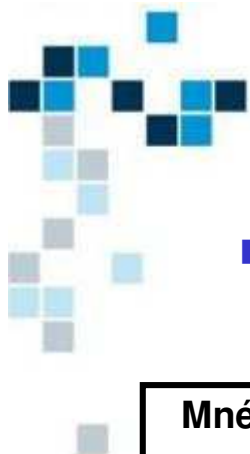
```
...
```

Parfois, il est nécessaire que le processeur prenne la décision de prendre ou de ne pas prendre une branche. On utilise alors un branchement conditionnel.

Ex: pour réaliser une boucle, un branchement vers le début de la boucle est nécessaire, mais ce branchement ne s'exécute qu'un certain nombre de fois.

■ Assembleur

	MOV r4, #0	@ tmp=0
	MOV r5, #0	@ i=0
LOOP	ADD r4, r4, r5	@ tmp+=i
	ADD r5, r5, #1	@ i++
	CMP r5, #5	
	BLT LOOP	@ i<5 : réitérer



Branchements simples (conditionnels)

■ Conditions de branchements

Mnémonique	Interprétation	commentaire
B	Unconditional	Always take this branch
BAL	Always	Always take this branch
BEQ	Equal	Comparison equal or zero result
BNE	Not equal	Comparison not equal or non-zero result
BPL	Plus	Result positive or zero
BMI	Minus	Result minus or negative
BCC	Carry clear	Arithmetic operation did not give carry-out
BLO	Lower	Unsigned comparison gave lower
BCS	Carry set	Arithmetic operation gave carry-out

Il y a plusieurs formes de branchements conditionnels (fonction de l'évaluation d'une condition), toutes les formes possibles sont énumérées dans les 2 tableaux suivants, avec leur interprétation.



Branchements simples (conditionnels)

■ Conditions de branchements (suite)

Mnémonique	Interprétation	commentaire
BHS	Higher or same	Unsigned comparisaon gave higher or same
BVC	Overflow clear	Signed integer operation; no overflow occurred
BVS	Overflow set	Signed integer operation; overflow occurred
BGT	Greater than	Signed integer comparison gave greater than
BGE	Greater or equal	Signed integer comparison gave greater or equal
BLT	Less than	Signed integer comparison gave less than
BLE	Less or equal	Signed integer comparison gave less than or equal
BHI	Higher	Unsigned comparison gave higher
BLS	Lower or same	Arithmetic comparison gave lower or same

Chaque condition correspond à une configuration des bits NZCV du registre d'état. Par exemple, les conditions EQ et NE correspondent respectivement à Z=1 et Z=0. Les conditions MI et PL correspondent à N=1 et N=0...



Branchements simples (conditionnels)

- Exécution conditionnelle
 - Branchement conditionnel

```
CMP r0, #5
BEQ BYPASS                @ if (r0!=5) {
ADD r1, r1, r0            @ r1:=r1+r0-r2
SUB r1, r1, r2            @ }
```

BYPASS ...

- Instructions conditionnelles

```
CMP r0, #5                @ if (r0!=5) {
ADDNE r1, r1, r0          @ r1:=r1+r0-r2
SUBNE r1, r1, r2          @ }
```

En assembleur ARM, l'exécution conditionnelle ne s'applique pas qu'aux branchements. Elle peut également s'appliquer aux instructions. Une branche utilisée pour désactiver l'exécution d'un petit nombre d'instructions (ADD, SUB dans l'exemple) peut être remplacée en donnant à ces instructions la condition opposée (Not Equal dans l'exemple). La deuxième solution est plus compacte et plus rapide.



Branchements temporaires

■ Branch and link (BL)

```
BL      SP1  
  
ADD     R3, R5, R6  
  
...  
  
SP1     ...  
  
MOV PC, LR                                @ return
```

■ Mécanisme d'appel de sous programme

Lors d'un branchement vers un sous programme, il faut pouvoir revenir à l'exécution de la séquence originale d'instructions (l'instruction suivant le branchement vers l'étiquette SP1) lorsque le sous programme a terminé sa tâche. Ceci est réalisé en sauvegardant l'adresse de l'instruction suivante, également appelée l'adresse de retour et qui est placée dans un registre spécialement prévu: le registre LR (Link Register R14).

Lorsque le sous programme a terminé, le retour à la séquence originale se fait en plaçant l'adresse de retour sauvegardée dans LR (R14) dans le compteur programme PC (R15): **MOV PC, LR** (ou **MOV R15, R14**)

ARM Instruction Set Architecture

- Caractéristiques du processeur
 - Unité de traitement
 - Jeu d'instruction
- Structure d'un programme assembleur

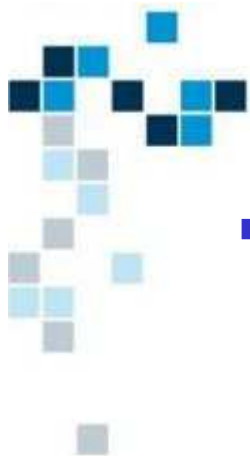


Structure d'un programme assembleur (GNU ARM)

```
.text                                @ directive pour l'assemblage
.align 4
.global start

RES:    .word    0                    @ zone de données
N       .word    5                    @
NUM1    .word    3, -17, 27, -12, 322 @
start:                                     @ zone de code

        LDR      r1, N                @
        ADR      r2, NUM1              @
        MOV      r0, #0                @
LOOP:   LDR      r3, [r2]              @
        ADD      r0, r0, r3            @
        ADD      r2, r2, #4            @
        SUBS     r1, r1, #1            @
        BGT      LOOP                 @
        STR      r0, RES               @ Fin d'assemblage
```



Structure d'un programme assembleur

- Un programme assembleur est constitué
 - De zones différentes pour le code et les données
 - Les instructions et les déclarations ne sont pas mélangées pour des raisons d'assemblage.
 - De directives assembleur
 - Pour préciser zone de code, zone de données.
 - Présence de variables ou de procédures externes.
 - De labels et commentaires
 - Un programmeur assigne un label (étiquette) ou un commentaire à une instruction à sa convenance personnelle.
 - Labels et commentaires n'ont pas d'effet sur le code source.
 - Les commentaires permettent de faciliter la lecture et la compréhension, de documenter le programme
 - Si un label est présent, l'assembleur définit le label comme équivalent à l'adresse de l'instruction dans le code assemblé (code objet). Un label peut ainsi être vu comme une adresse (adresse d'une instruction lors d'un branchement, adresse d'une donnée déclarée dans la zone de données).



Environnement GNU ARM

- Etapes d'assemblage:
 - L'assemblage est l'opération qui permet de transformer le code source (écrit en *langage d'assemblage*) en code *objet* qui contient la traduction en binaire des instructions du programme.
 - Cette traduction peut être incomplète (cas de programmes composés de plusieurs fichiers). L'assembleur n'est alors pas capable de traduire l'ensemble du programme directement. Pour résoudre ce problème, on procède en deux étapes:
 - chaque fichier est compilé séparément et produit un fichier objet ainsi qu'un ensemble d'information (symboles publics réutilisés entre modules).
 - Les informations recueillies lors de la compilation séparée sont exploitées lors de la réunion des différents fichiers objets, au cours de l'opération d'*édition des liens*. pour produire le programme exécutable final qui sera placé en mémoire.



```
C:\Invite de commandes - arm-elf-insight appli.elf
C:\Documents and Settings\Bilavarn\Mes documents\developpement\gnuarm\DEMO>
C:\Documents and Settings\Bilavarn\Mes documents\developpement\gnuarm\DEMO>
C:\Documents and Settings\Bilavarn\Mes documents\developpement\gnuarm\DEMO>
C:\Documents and Settings\Bilavarn\Mes documents\developpement\gnuarm\DEMO>
C:\Documents and Settings\Bilavarn\Mes documents\developpement\gnuarm\DEMO>
C:\Documents and Settings\Bilavarn\Mes documents\developpement\gnuarm\DEMO>
C:\Documents and Settings\Bilavarn\Mes documents\developpement\gnuarm\DEMO>
C:\Documents and Settings\Bilavarn\Mes documents\developpement\gnuarm\DEMO>
C:\Documents and Settings\Bilavarn\Mes documents\developpement\gnuarm\DEMO>
C:\Documents and Settings\Bilavarn\Mes documents\developpement\gnuarm\DEMO>dir
Le volume dans le lecteur C n'a pas de nom.
Le numéro de série du volume est 98B0-D578

Répertoire de C:\Documents and Settings\Bilavarn\Mes documents\developpement\gnuarm\DEMO

03/10/2007 16:21 <REP>      .
03/10/2007 16:21 <REP>      ..
01/10/2007 10:44          831 demo.s
                1 fichier(s)          831 octets
                2 Rép(s) 32 753 180 672 octets libres

C:\Documents and Settings\Bilavarn\Mes documents\developpement\gnuarm\DEMO>arm-elf-as -mcpu=arm7tdmi -g -o demo.o demo.s

C:\Documents and Settings\Bilavarn\Mes documents\developpement\gnuarm\DEMO>dir
Le volume dans le lecteur C n'a pas de nom.
Le numéro de série du volume est 98B0-D578

Répertoire de C:\Documents and Settings\Bilavarn\Mes documents\developpement\gnuarm\DEMO

03/10/2007 16:21 <REP>      .
03/10/2007 16:21 <REP>      ..
03/10/2007 16:21          1 404 demo.o
01/10/2007 10:44          831 demo.s
                2 fichier(s)          2 235 octets
                2 Rép(s) 32 753 176 576 octets libres

C:\Documents and Settings\Bilavarn\Mes documents\developpement\gnuarm\DEMO>arm-elf-ld -Ttext=0x8000 -e start -o appli.elf demo.o

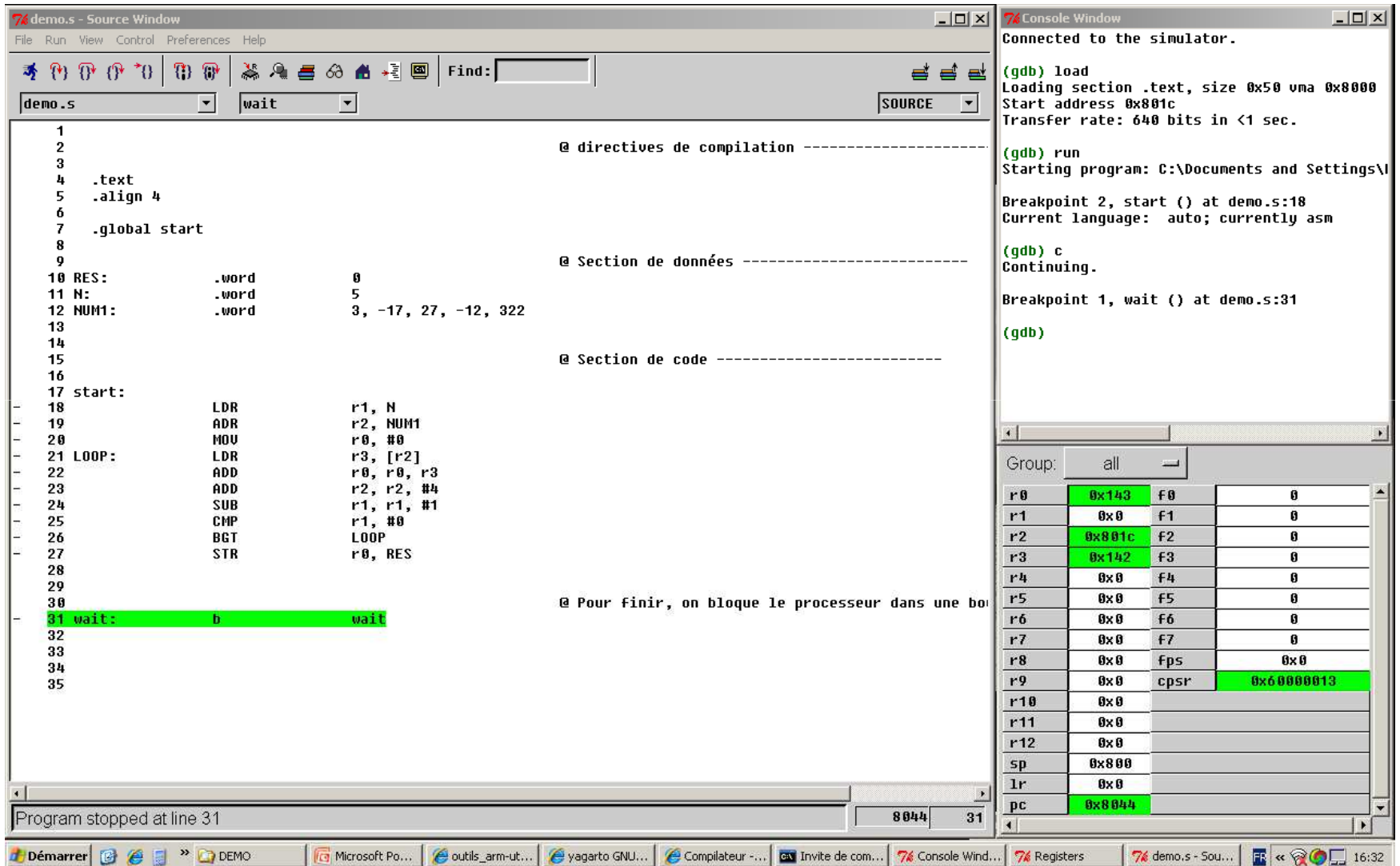
C:\Documents and Settings\Bilavarn\Mes documents\developpement\gnuarm\DEMO>dir
Le volume dans le lecteur C n'a pas de nom.
Le numéro de série du volume est 98B0-D578

Répertoire de C:\Documents and Settings\Bilavarn\Mes documents\developpement\gnuarm\DEMO

03/10/2007 16:23 <REP>      .
03/10/2007 16:23 <REP>      ..
03/10/2007 16:23          34 129 appli.elf
03/10/2007 16:21          1 404 demo.o
01/10/2007 10:44          831 demo.s
                3 fichier(s)          36 364 octets
                2 Rép(s) 32 753 139 712 octets libres

C:\Documents and Settings\Bilavarn\Mes documents\developpement\gnuarm\DEMO>arm-elf-insight appli.elf
```

DEMO



The screenshot displays the GDB debugger interface with two main windows: 'demo.s - Source Window' and '74 Console Window'.

demo.s - Source Window: Shows assembly code for a program. The code includes directives for text, align, and global, followed by data definitions (RES, N, NUM1) and a code section starting with 'start:'. The program logic involves loading values into registers, performing arithmetic operations (ADD, SUB), and a loop structure. Line 31, 'wait: b wait', is highlighted in green, indicating the current execution point. The status bar at the bottom indicates 'Program stopped at line 31'.

74 Console Window: Displays GDB commands and their outputs. The commands executed are 'load', 'run', 'c', and 'c'. The output shows the program loading successfully, starting at address 0x801c, and the current language is set to 'asm'. The breakpoint is set at line 31.

Registers Window: A table showing the state of various registers. The 'r0' register contains the value 0x143, and the 'pc' (program counter) register contains 0x8044. Other registers like r1, r2, r3, etc., are shown with their current values.

Register	Value	Register	Value
r0	0x143	F0	0
r1	0x0	F1	0
r2	0x801c	F2	0
r3	0x142	F3	0
r4	0x0	F4	0
r5	0x0	F5	0
r6	0x0	F6	0
r7	0x0	F7	0
r8	0x0	fps	0x0
r9	0x0	cpsr	0x60000013
r10	0x0		
r11	0x0		
r12	0x0		
sp	0x800		
lr	0x0		
pc	0x8044		