
Jeux d'instructions

Daniel Etiemble
de@lri.fr

Les jeux d'instructions

- Ensemble des instructions d'un processeur
- Format d'instructions
 - Lié au modèle (n,m)
 - Longueur fixe ou longueur variable
 - Accès aux données
- Un jeu d'instructions RISC simple
 - MIPS32
- Les grandes caractéristiques des jeux d'instructions
 - Branchements
 - Modes d'adressage
 - Appel de fonctions

Jeux d'instructions

- Des objectifs différents selon les classes d'applications
 - Vitesse maximale (PC, serveurs)
 - Taille de code minimale (embarqué)
 - Consommation
 - essentiel pour embarqué
 - important pour tous
- Taille des instructions
 - Fixe
 - Variable
- Modèles d'exécution

Les objectifs

- Performance
 - Pipeline efficace
 - Instructions de longueur fixe
 - Décodage simple
 - Modes d'adressage simples
 - Utiliser le parallélisme d'instructions
- Taille du code
 - Minimiser la taille des instructions
 - Instructions de longueur variable (ou fixe)
 - Accès aux données efficace
 - Modes d'adressage complexes et efficaces pour applications visées
- Compatibilité binaire avec les générations précédentes
 - Fondamental pour processeurs « généralistes » : Exemple IA-32 (x86)
 - Moins important pour processeurs plus spécialisés (embarqué, traitement du signal)

Modèles d'exécution

- Modèles d'exécution (n,m)
 - n : nombre d'opérandes par instruction
 - m : nombre d'opérandes mémoire par instruction
- Les modes principaux
 - RISC : (3,0)
 - Instructions de longueur fixe
 - Load et Store : seules instructions mémoire
 - IA-32 : (2,1)
 - Instructions de longueur variable

Modèle d'exécution RISC

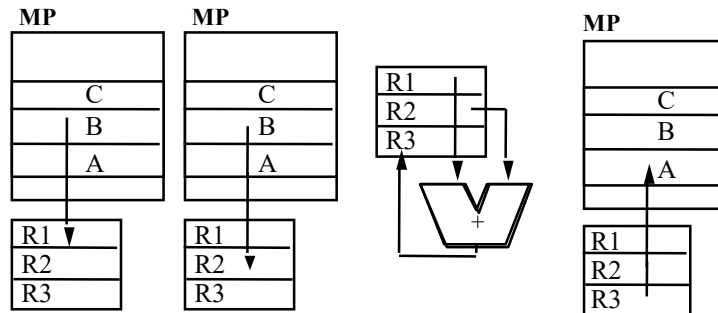
(n,m)

n : nombre d'opérandes par instruction
m : nombre d'opérandes mémoire par instruction

Ex : $A = B + C$

LOAD-STORE (3,0)

Load	R1	@B
Load	R2	@C
Add	R3	R2, R1
Store	R3	@A



Instructions de longueur fixe

Seules les instructions Load et Store accèdent à la mémoire

Registres: organisation RISC

- 32 registres généraux (entiers) R0 à R31
- 32 registres flottants
- Instructions UAL et mémoire
 - Registre – registre
 - $R_d \leftarrow R_{s1} \text{ op } R_{s2}$
 - Registre – immédiat
 - $R_d \leftarrow R_{s1} \text{ op } \text{immédiat}$
 - $R_d \leftrightarrow \text{Mémoire } (R_{s1} + \text{dépl.})$

REGISTRES MIPS 32

Registre	Nom	
0	zero	
1	\$at	assembleur
2-3	\$v0 - \$v1	Résultats
4-7	\$a0 - \$a3	Arguments fonction
8-15	\$t0 - \$t7	temporaires
16-23	\$s0 - \$s7	sauvegarde
24-25	\$t8 - \$t9	temporaire
26-27	\$k0 - \$k1	trappes
28	\$gp	Pointeur global
29	\$sp	Pointeur pile
30	\$fp	Pointeur trame
31	\$ra	Adresse retour

Modèle (2,1)

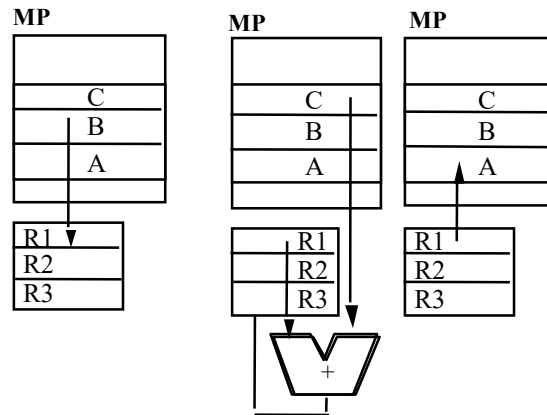
(n,m)

n : nombre d'opérandes par instruction
m : nombre d'opérandes mémoire par instruction

Ex : $A := B + C$

REGISTRE-MEMOIRE (2,1)

Load	R1	@B
Add	R1	@C
Store	R1	@A



CISC compatible avec la technologie MOS des années 75-80

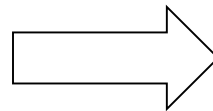
Caractéristiques IA-32

- Instructions de longueur variable

Op code	Reg. et M	Déplacement	Immédiat	
1 or 2	1 or 2	0, 1, 2 or 4	0, 1, 2 or 4	octets

- Inst dest, source

REG	REG
REG	MEM
REG	IMM
MEM	REG
MEM	IMM



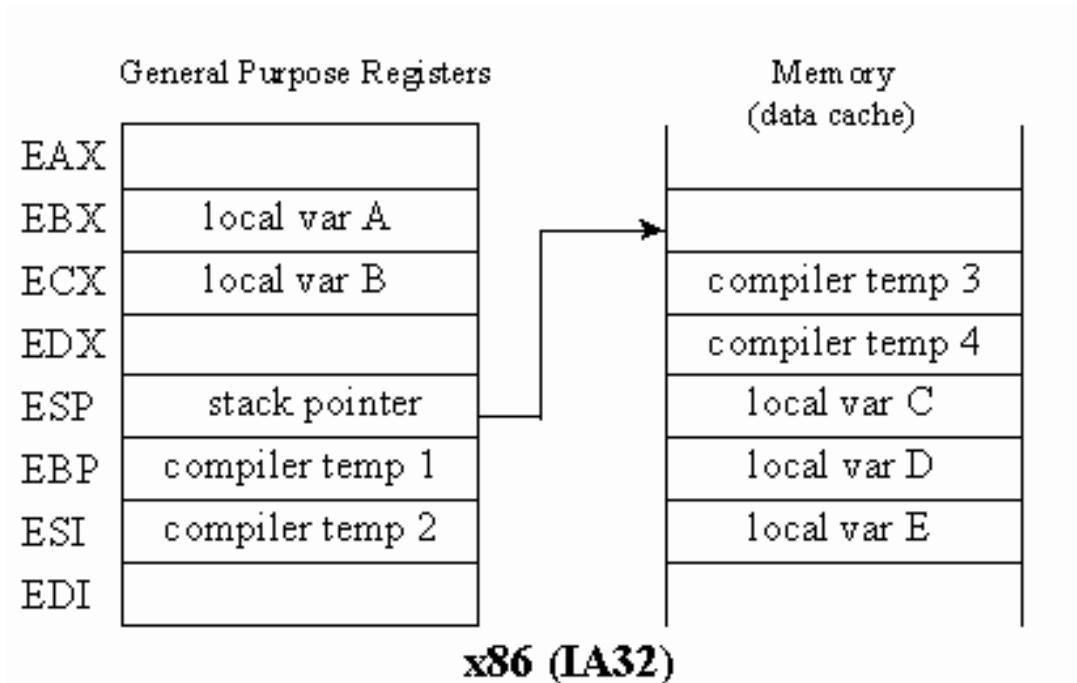
Lecture mémoire,
Exécution,
Ecriture mémoire

- Instructions complexes
 - Rep
- Modes d'adressage complexes

Adresse mémoire = Rb + RI x f + déplacement

Registres : organisation IA-32

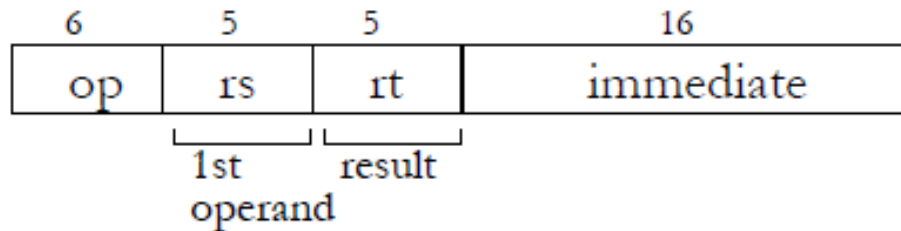
- Organisation non homogène
 - 8 registres «généraux» avec rôle spécifique
 - Registres flottants fonctionnant en pile (x87)
 - Registres «SIMD» (MMX, SSE-SSE2-SSE3-SSE4-AVX)



Le débat RISC-CISC pour les PC

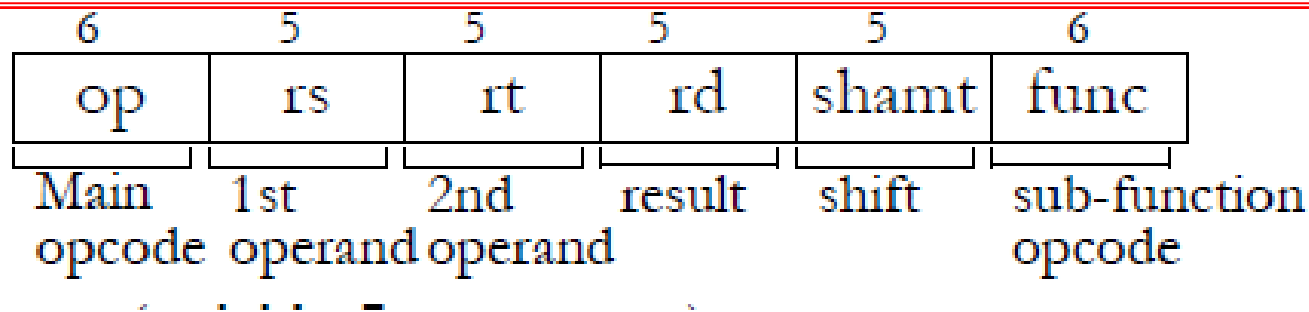
- Définition
 - RISC : modèle (3,0)
 - CISC : tous les autres
- RISC et pipeline
 - Les jeux d'instructions RISC facilitent la réalisation de pipelines performants
- « Solution » Intel et AMD pour IA-32
 - Convertir les instructions CISC en instructions RISC lors du décodage des instructions (conversion dynamique)
 - On conserve la compatibilité binaire
 - On a l'efficacité des pipelines « RISC »

MIPS : les instructions de format I



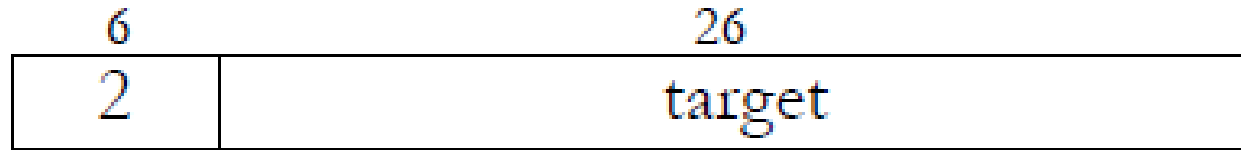
- Instructions mémoire
 - Adresse mémoire = $[rs] + \text{immédiate}$
- Instructions arithmétiques/logiques/comparaison avec immédiat
 - $[rt] \leftarrow [rs] \text{ opération (imm16 étendu sur 32 bits)}$
 - Extension signée ou extension avec zéros
- Instructions de branchement conditionnel
 - Adresse de branchement = $[CP] + 4 + 4 * \text{imm16 étendu (signé) sur 32 bits}$

MIPS : les instructions de format R

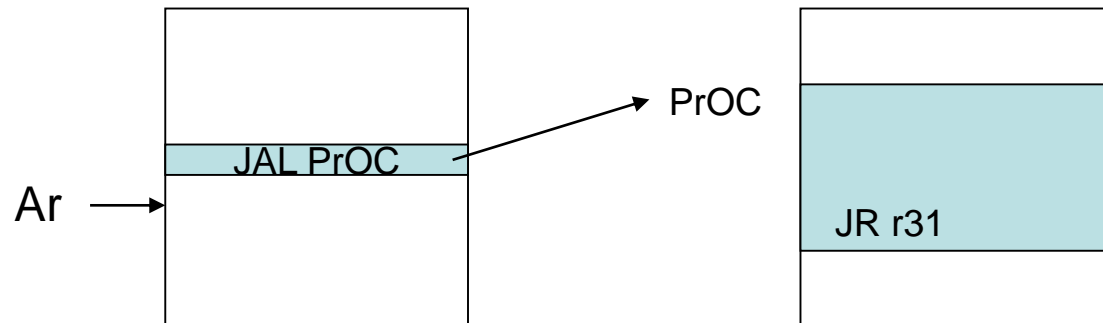


- Instructions arithmétiques/logiques/comparaison
 - $[rd] \leftarrow [rs] \text{ opération } [rt]$
- Instructions de décalage
 - $[rd] \leftarrow \text{décalage } [rs]$
 - Nombre de décalages :
 - 5 bits poids faible de $[rt]$
 - 5 bits contenus dans shamt

MIPS : les instructions de type J



- Instruction J
 - $CP \leftarrow CP_{31-28} | \text{target} | 00$
- Instruction JAL
 - range l'adresse de retour dans r31
 - $CP \leftarrow CP_{31-28} | \text{target} | 00$



MIPS : les instructions mémoire

- Chargement (load)
 - $[Rt] \leftarrow \text{Mémoire}([Rs] + \text{simm16})$
 - Chargement mot de 32 bits
 - Chargement mot de 16 bits avec extension de signe ou extension zéros
 - Chargement octet avec extension de signe ou extension zéros
- rangement (store)
 - $\text{Mémoire}([Rs] + \text{simm16}) \leftarrow [Rt]$
 - rangement mot de 32 bits
 - rangement mot de 16 bits
 - rangement octet

lw	lw Rt, imm16(Rs)	$Rt \leftarrow \text{Mem32}(Rs + \text{simm16})$
lb	ldb Rt, imm16(Rs)	$Rt \leftarrow \text{ES} \mid \text{Mem8}(Rs + \text{simm16})$
lbu	ldbu Rt, imm16(Rs)	$Rt \leftarrow \text{Zero} \mid \text{Mem8}(Rs + \text{simm16})$
lh	ldh Rt, imm16(Rs)	$Rt \leftarrow \text{ES} \mid \text{Mem16}(Rs + \text{simm16})$
lhu	ldhu Rt, imm16(Rs)	$Rt \leftarrow \text{Zero} \mid \text{Mem16}(Rs + \text{simm16})$
sw	stw Rs, imm16(Rt)	$\text{Mem32}(Rs + \text{simm16}) \leftarrow Rt$
sb	stb Rs, imm16(Rt)	$\text{Mem8}(Rs + \text{simm16}) \leftarrow Rt$
sh	sth Rs, imm16(Rt)	$\text{Mem16}(Rs + \text{simm16}) \leftarrow Rt$

MIPS : les instructions arithmétiques et transfert

add	add Rd,Rs,rB	$Rd \leftarrow Rs + Rt$ (trappe sur débordement)
addi	addi Rt,Rs,imm16	$Rt \leftarrow Rs + \text{imm16}$ (trappe sur débordement)
addu	addu Rd,Rs,rB	$Rd \leftarrow Rs + Rt$
addiu	addiu Rt,Rs,imm16	$Rt \leftarrow Rs + \text{imm16}$
sub	sub Rd,Rs,Rt	$Rd \leftarrow Rs - Rt$
mul	mul Rd,Rs,Rt	$Rd \leftarrow Rs * Rt$ (32 bits poids faible du résultat)

Pseudo-instructions

move	move Rd,Rs	$Rd \leftarrow Rs$ (add Rd,Rs,r0)
------	------------	-----------------------------------

MIPS : les instructions logiques

and	and Rd,Rs,Rt	$Rd \leftarrow Rs \text{ et } Rt$ (ET logique bit à bit)
andi	andi Rt,Rs,imm16	$Rt \leftarrow Rs \text{ et } zimm16$
or	or Rd,Rs,Rt	$Rd \leftarrow Rs \text{ ou } Rt$ (OU logique bit à bit)
ori	ori Rt,Rs,imm16	$Rt \leftarrow Rs \text{ ou } zimm16$
xor	xor Rd,Rs,Rt	$Rd \leftarrow Rs \text{ xor } Rt$ (OU exclusif bit à bit)
xori	xori Rt,Rs,imm16	$Rt \leftarrow Rs \text{ xor } zimm16$
nor	nor Rd,Rs,Rt	$Rd \leftarrow Rs \text{ nor } Rt$ (NOOr bit à bit)
andhi	andhi Rt,Rs,imm16	$Rt \leftarrow Rs \text{ et } imm16 0000000000000000$
lui	lui Rt, imm16	$Rt \leftarrow imm16 0000000000000000$

Chargement d'une adresse dans un registre

la Rt, ETIQ

lui Rt, %hi ETIQ // 16 bits poids fort

ori Rt, Rt,%lo ETIQ // 16 bits poids faible

MIPS : les instructions de comparaison

- SLT Rd,Rs,Rt (signé)
 - $Rd \leftarrow 1$ si $(Rs < Rt)$ **vrai** et $\leftarrow 0$ si $(Rs \geq Rt)$ **faux**
- SLTI Rd,Rs,ES-imm16 (signé)
 - $Rd \leftarrow 1$ si $(Rs < \text{simm32})$ **vrai** et $\leftarrow 0$ si $(Rs \geq \text{simm32})$ **faux**
- SLTU Rd,Rs,Rt (non signé)
 - $Rd \leftarrow 1$ si $(Rs < Rt)$ **vrai** et $\leftarrow 0$ si $(Rs \geq Rt)$ **faux**
- SLTIU Rd,Rs, EZ-imm16 (non signé)
 - $Rd \leftarrow 1$ si $(Rs < \text{zimm32})$ **vrai** et $\leftarrow 0$ si $(Rs \geq \text{simm32})$ **faux**

MIPS : les instructions de branchement conditionnel

- *Bcond* Rs,Rt,ETIQ
 - Si (*Rs cond* Rt) vrai, branchement à l'adresse $PC+4+SIMM16|00$
 - Si (*Rs cond* Rt) faux, séquence : adresse $PC+4$
 - Les conditions
- *Bcond* Rs, ETIQ
 - Si (*Rs cond* 0) vrai, branchement à l'adresse $PC+4+SIMM16|00$
 - Si (*Rs cond* 0) faux, séquence : adresse $PC+4$
 - Les conditions
 - GTZ ($>$)
 - GEZ (\geq)
 - LTZ ($<$)
 - LEZ (\leq)

MIPS : instructions de décalage et rotation

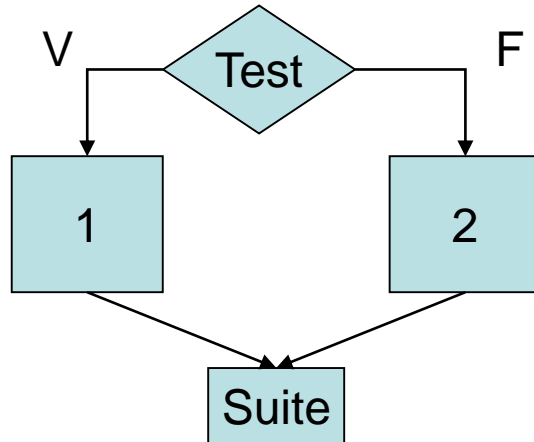
- Décalages logiques à droite et à gauche
 - insertion de zéros
- Décalage arithmétique à droite
 - insertion du bit de signe
- Rotations
 - Les bits sortants à droite sont réintroduits à gauche

srlv	srl Rd, Rs, Rt	$Rd \leftarrow Rs \gg (\text{nb spécifié dans Rt})$ - Logique
srl	srl Rd, Rs, imm5	$Rd \leftarrow Rs \gg (\text{imm5})$ - Logique
sra	sra Rd, Rs, Rt	$Rd \leftarrow Rs \gg (\text{nb spécifié dans Rt})$ - Arithmétique
sra	sra Rd, Rs, imm5	$Rd \leftarrow Rs \gg (\text{imm5})$ - Arithmétique
sllv	sll Rd, Rs, Rt	$Rd \leftarrow Rs \ll (\text{nb spécifié dans Rt})$
sll	sll Rd, Rs, imm5	$Rd \leftarrow Rs \ll (\text{imm5})$ - Logique

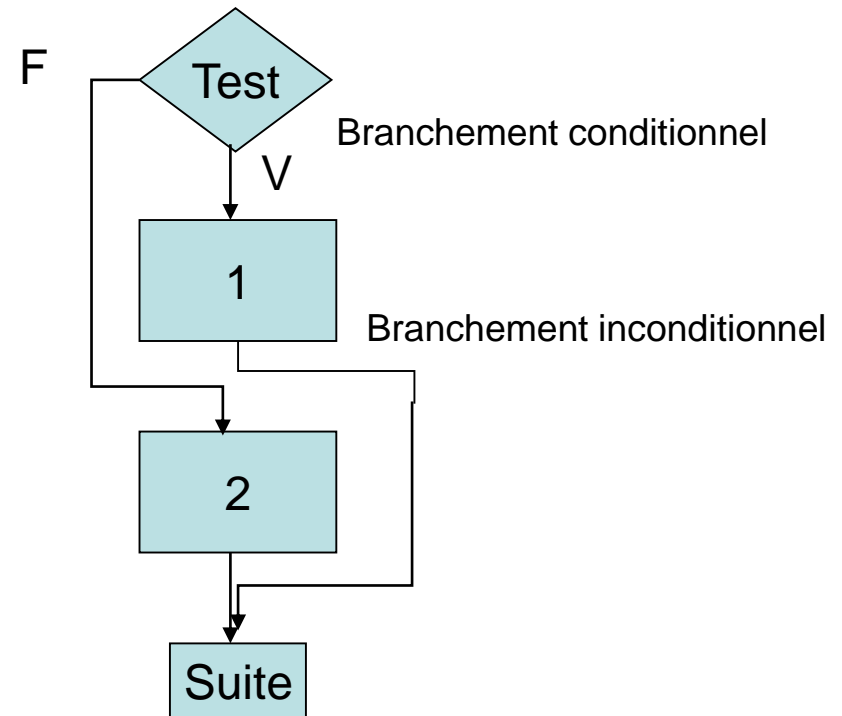
rotrv	rotrv Rd, Rs, Rt	$Rd \leftarrow \text{rotation droite de Rs (nb spécifié dans Rt)}$
rotr	rotr Rd, Rs, imm5	$Rd \leftarrow \text{rotation droite de Rs (imm5)}$

Jeux d'instructions : branchements conditionnels

Si condition alors « suite d'instructions 1 » sinon « suite d'instructions2 »



Instructions de test
Instructions de branchement conditionnel



Branchements conditionnels

- Test et branchement en 1 seule instruction
 - Bcond rs,rt,ETIQ
 - EQ, NE pour MIPS
- Résultat du test dans un registre et Branchement conditionnel selon condition dans un registre
 - MIPS
 - SLT (1 ou 0 dans un registre) et Bcond Rs1, ETIQ
 - cond : toutes les conditions sauf EQ et NE. Comparaison Rs et 0
- Résultat du test dans un registre code condition et branchement conditionnel selon code condition
 - ARM
 - CMP : résultats de la comparaison dans le registre code condition
 - Bcond ETIQ : branchement selon le registre code condition

Exemple : calcul de la valeur absolue

Mettre dans R1 la valeur absolue du contenu de R2

: MIPS

```
ADD R1,R2,R0
SLT R3,R0,R2
BGTZ R3,Suite
SUB R1,R0,R2
```

Suite :

ARM

```
MOV R1,R2
CMP R2,#0
BGT Suite
RSB R1,R2,#0 // R1 ← 0-R2
```

Suite :

ARM (*instructions conditionnelles*)

```
MOV R1,R2
CMP R2,0
RSBLT R1,R2,#0
```

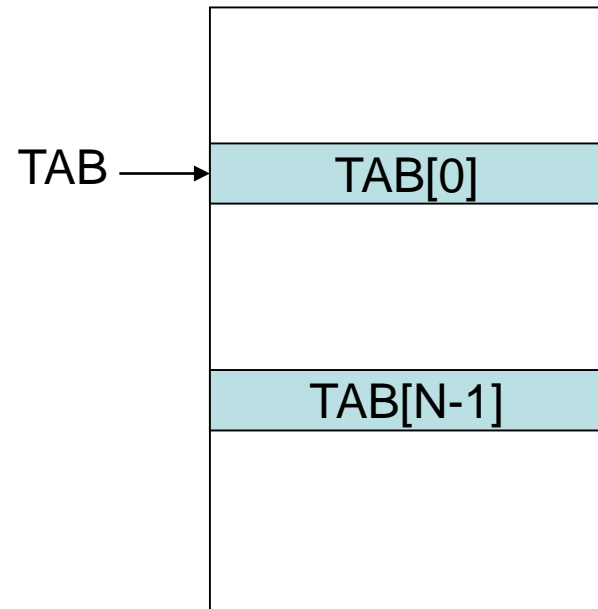
Accès à un tableau

- Exemple : somme des éléments d'un tableau de N entiers à l'adresse TAB

```
S=0
For (i=0;i<N;i++)
    S+=TAB[i];
```

Version MIPS

```
LA r2, TAB           // adresse du tableau dans r2
MOVE r1, r0          // r1=0
MOV r3, N            // N dans r3
SLL r3,r3,2          // 4*N dans r3
ADD r3,r3,r2         // Adresse du mot après TAB[N-1]
Deb : LW r4,0(r2)     // chargement de Tab[i]
ADD r1,r1,r4         // accumulation
ADDI r2,r2,4         // adresse de l'élément suivant
BNEr2,r3,Deb        // boucle tant que r2<r3
// Résultat dans r1
```



2 instructions pour accéder à chaque élément du tableau

Modes d'adressage plus complexes

- Exemple ARM

Instructions mémoire

Instruction	Signification	Action
LDR	Chargement mot	$Rd \leftarrow Mem_{32}(AE)$
LDRB	Chargement octet	$Rd \leftarrow Mem_8(AE)$
STR	Rangement mot	$Mem_{32}(AE) \leftarrow Rd$
STRB	Rangement octet	$Mem_8(AE) \leftarrow Rd$

Modes d'adressage

Mode	Assembleur	Action
Déplacement 12 bits, Pré-indexé	$[Rn, \#deplacement]$	Adresse = $Rn + \text{déplacement}$
Déplacement 12 bits, Pré-indexé avec mise à jour	$[Rn, \#deplacement] !$	Adresse = $Rn + \text{déplacement}$ $Rn \leftarrow \text{Adresse}$
Déplacement 12 bits, Post-indexé	$[Rn], \#deplacement$	Adresse = Rn $Rn \leftarrow Rn + \text{déplacement}$
Déplacement dans Rm Préindexé	$[Rn, \pm Rm, \text{décalage}]$	Adresse = $Rn \pm [Rm] \text{ décalé}$
Déplacement dans Rm Préindexé avec mise à jour	$[Rn, \pm Rm, \text{décalage}] !$	Adresse = $Rn \pm [Rm] \text{ décalé}$ $Rn \leftarrow \text{Adresse}$
Déplacement dans Rm Postindexé	$[Rn], \pm Rm, \text{décalage}$	Adresse = Rn $Rn \leftarrow Rn \pm [Rm] \text{ décalé}$

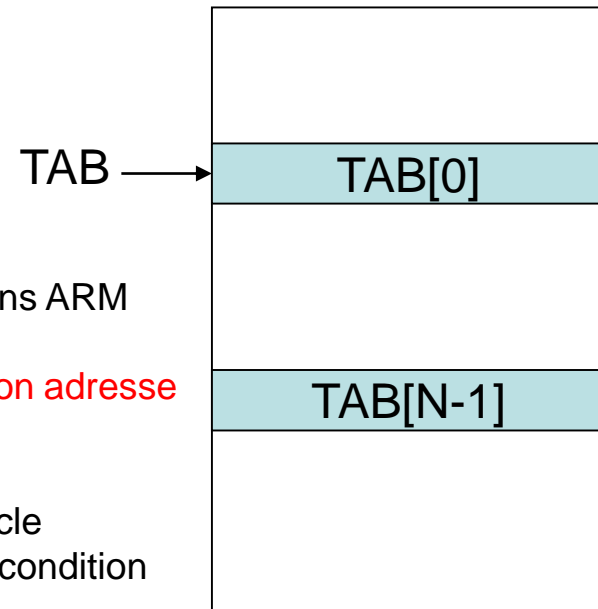
Accès à un tableau avec adressage post-indexé

- Exemple : somme des éléments d'un tableau de N entiers à l'adresse TAB

```
S=0
For (i=0;i<N;i++)
    S+=TAB[i];
```

Version ARM

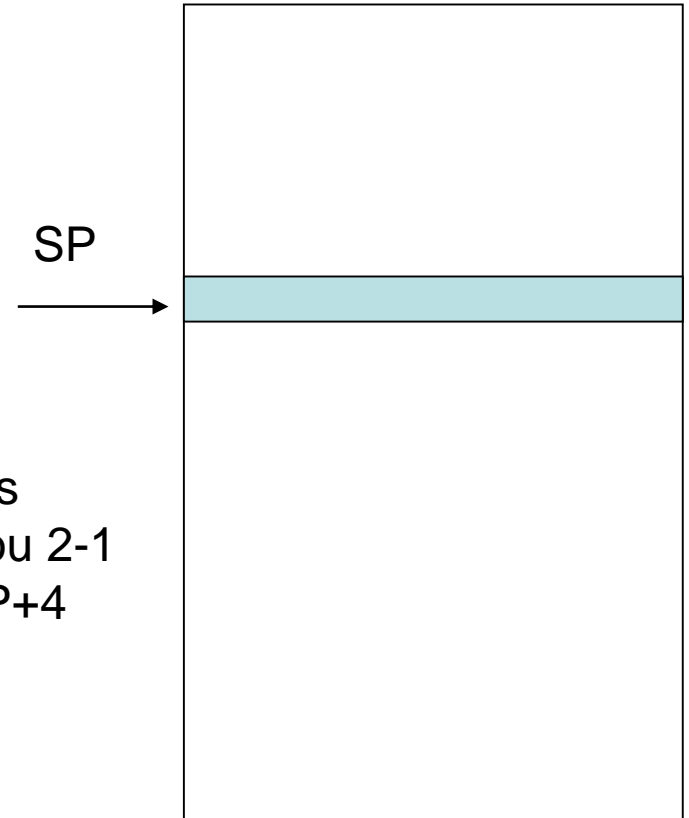
```
ADR r2, TAB      // adresse du tableau dans r2
MOV r0, 0         // r0=0. NB : r0 est différent de 0 dans ARM
MOV r1, N         // N dans r1
Deb : LDR r3,[r2],#4 // chargement de Tab[i] et préparation adresse
                  // suivante
ADD r0,r0,r3      // accumulation
SUBS r1,r1,#1     // décrémentation compteur de boucle
                  // et positionnement du registre code condition
BGT Deb          // boucle tant que r1>0
                  // Résultat dans r0
```



1 seule instruction pour accéder à chaque élément du tableau

La pile mémoire

- Mécanisme d'accès mémoire sans adressage explicite
 - L'adresse mémoire est contenu dans un registre « pointeur de pile » ou « stack pointer » (SP)
- Instructions spécifiques d'accès
 - Empilement (Push)
 - Dépilement (Pop)



Push Ri

- 1) $SP \leftarrow SP - 4$
- 2) $MEM(SP) \leftarrow Ri$

4 possibilités
-Ordre 1-2 ou 2-1
-SP-4 ou SP+4

POP Ri

- $Ri \leftarrow MEM(SP)$
 $SP \leftarrow SP + 4$

Les instructions « pile »

- Pointeur de pile
 - RISC : SP est un registre général
 - r29 dans MIPS, r13 dans ARM
 - IA-32 : SP est un registre spécifique
- Instructions Push et Pop
 - RISC : implantées avec instructions ADDI, Load et Store
 - IA-32 : Push et Pop
- Empilement et dépilement de plusieurs registres
 - ARM : les instructions STM(store multiple) et LDM (load multiple) avec le registre r13 permettent en une instruction un empilement et un dépilement de plusieurs registres
 - IA-32 : PUSHA et POPA permettent d'empiler et dépiler tous les registres généraux

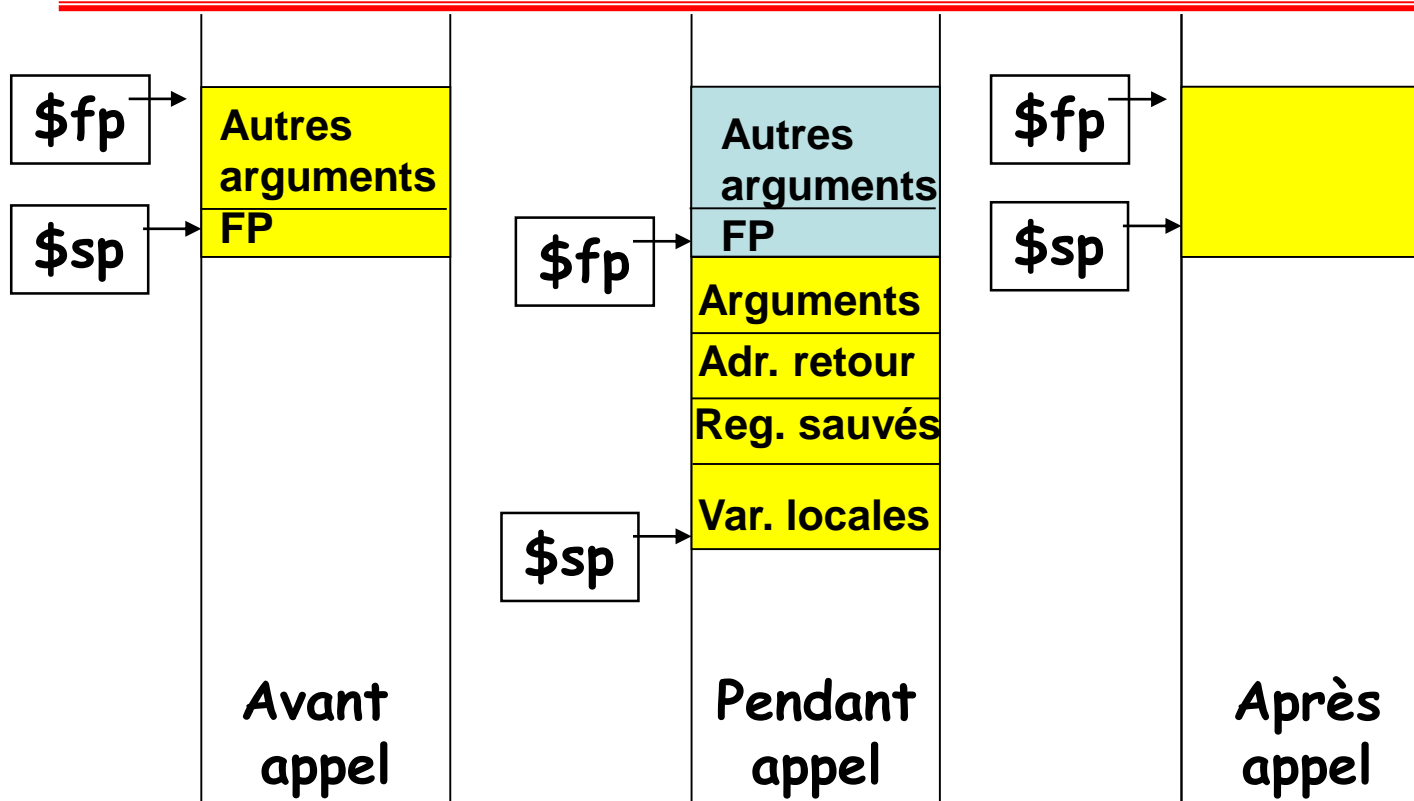
Les appels/retours de fonctions/procédures

- Appel de fonctions
 - Passage des paramètres
 - Par registre : nombre limité (ex : 4 pour MIPS)
 - Par pile : paramètres supplémentaires
 - Sauvegarde de l'adresse de retour
 - Dans un registre général
 - R31 (MIPS), R14 (ARM)
 - Dans un registre spécifique
 - LR (PowerPC)
 - Dans la pile
 - IA-32
- Retour
 - Récupération du résultat
 - Par registres
 - Par pile
 - Instruction chargeant l'adresse de retour dans PC
 - RET \equiv JR R31 (MIPS)
 - RET \equiv POP PC dans IA-32

Fonctions : la pile d'appel

- Pile d'appel = contexte d'exécution de la fonction
 - Arguments de la fonction appelée
 - Sauvegarde des registres
 - Adresse de retour (indispensable pour fonctions imbriquées)
 - Registres utilisés par la fonction (pour pouvoir les restaurer pour la fonction appelante)
 - Variables locales à la fonction appelée

Pile d'appel



FP (*Frame pointer* ou Pointeur de trame) pointe sur le premier mot d'une fonction

Adresse stable durant l'exécution de la fonction

SP (*Stack pointer* ou Pointeur de pile)

SP évolue au cours de l'exécution de la fonction

FP est initialisé par SP et SP est restauré à partir de FP