

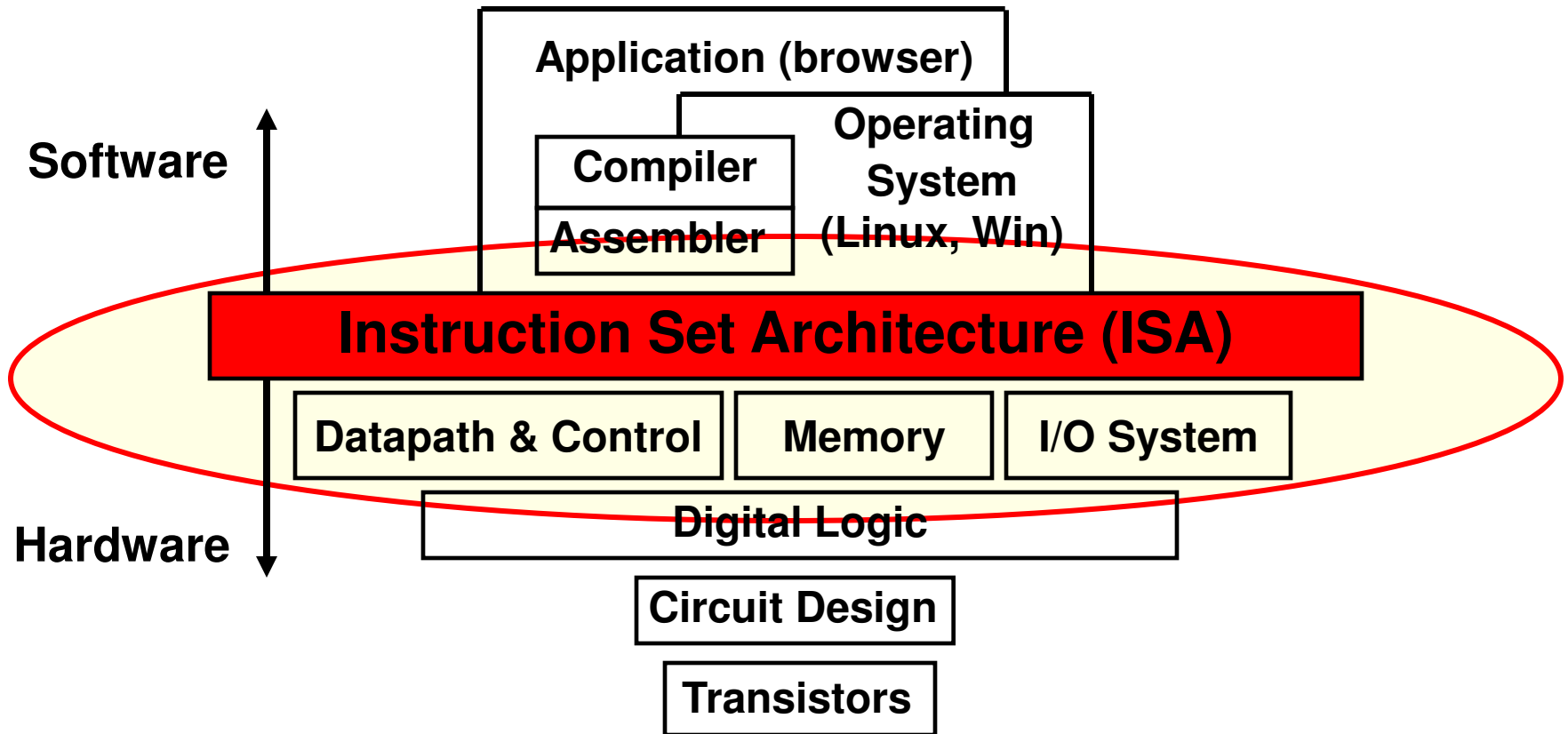
Architecture externe du MIPS R3000

Architecture du jeu  
d'instructions

Format de l'instruction MIPS

# Architecture du jeu d'Instruction

---



# Aperçu sur les niveaux du jeu d'instructions

---

- *ISA: Comment la machine apparaît pour un programme en langage machine (compilateur) ?*
  - L'organisation de la mémoire
  - Les instructions
    - Formats
    - Types (arithmetic, logical, data transfer, and flow control)
    - Modes (kernel and user)
  - Les opérandes
    - Registres
    - Le Type de données
    - L'adressage mémoire

# Computing Languages

Langage de Haut Niveau

```
temp  = v[k];  
v[k]   = v[k+1];  
v[k+1] = temp;
```

```
TEMP = V(K)  
V(K)  = V(K+1)  
V(K+1) = TEMP
```

C/Java Compiler

Fortran Compiler

Langage d'assemblage

```
lw $t0, 0($2)  
lw $t1, 4($2)  
sw $t1, 0($2)  
sw $t0, 4($2)
```

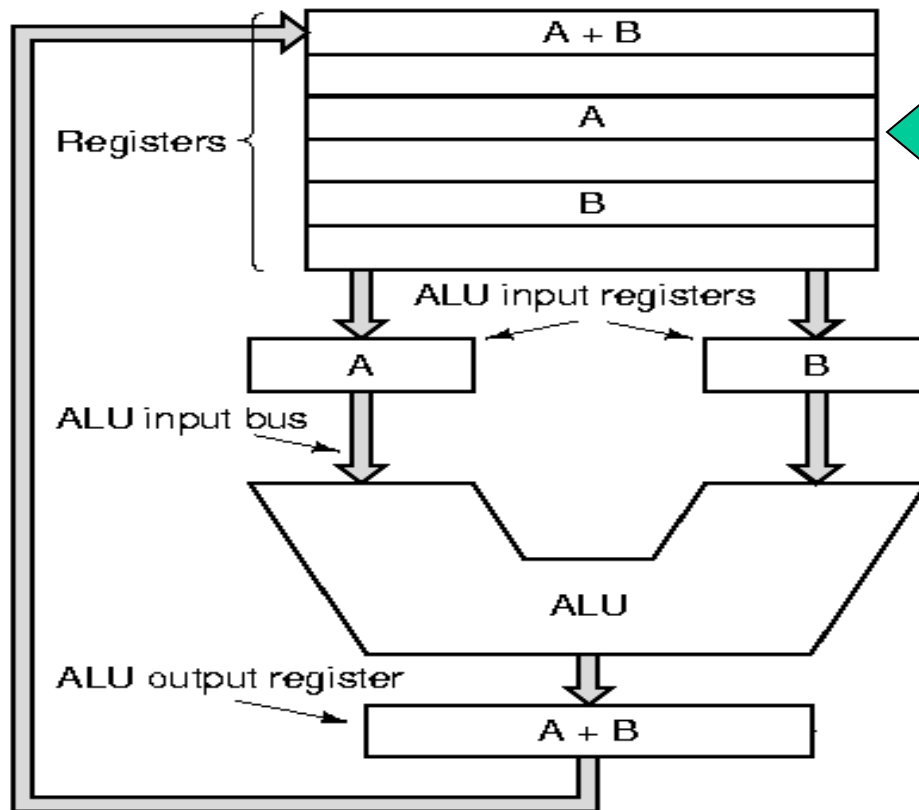
MIPS Assembler

Langage Machine

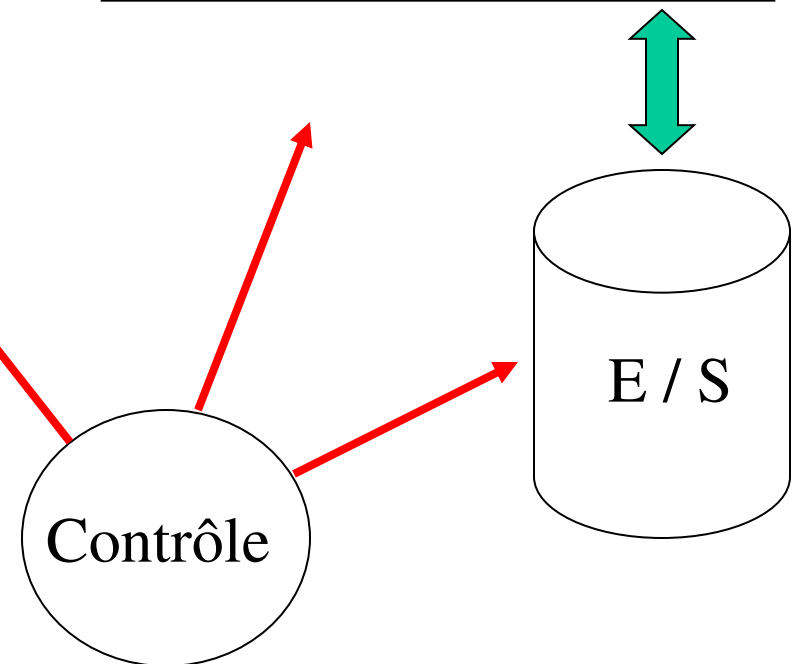
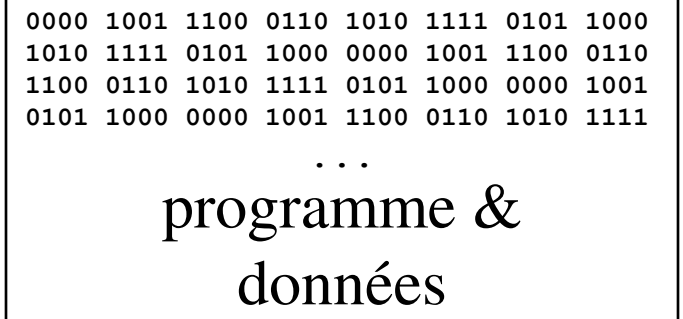
```
0000 1001 1100 0110 1010 1111 0101 1000  
1010 1111 0101 1000 0000 1001 1100 0110  
1100 0110 1010 1111 0101 1000 0000 1001  
0101 1000 0000 1001 1100 0110 1010 1111
```

# Exécution du programme

Chemin de données  
(Datapath)



Mémoire



**Fetch – Decode – Execute Cycle**

# Langage MIPS

- Arithmétique
  - **add, sub, mult, div**
- Logique
  - **and, or, sll** (Décalage Logique à gauche), **srl** (Décalage Logique à droite)
- Transfert de données
  - **lw** (chargement d'un mot), **sw** (rangement d'un mot)
  - **lui** (Chargement immédiat d'un entier non signé)
- Branchement
  - *Conditionnel*: **beq, bne, slt**
  - *Inconditionnel*: **j** (saut vers l'étiquette), **jr** (saut vers l'adresse rangée dans le registre ), **jal** (retour après un appel d'une procédure)

**Codes Opérations MIPS  
de base que nous allons  
utiliser dans les  
traitements de données**

# Langage MIPS

---

## Branchements & Sauts

Instruction	Signification
beq Rs, Rt, label	If (Rs == Rt)      PC $\leftarrow$ label
bne Rs, Rt, label	If (Rs != Rt)      PC $\leftarrow$ label
bltz Rs, label	If (Rs < 0 )      PC $\leftarrow$ label
blez Rs, label	If (Rs $\leq$ 0 )      PC $\leftarrow$ label
bgtz Rs, label	If (Rs > 0 )      PC $\leftarrow$ label
bgez Rs, label	If (Rs $\geq$ 0 )      PC $\leftarrow$ label
j    jlabel	PC $\leftarrow$ jlabel
jr   Rs	PC $\leftarrow$ Rs
jal   jlabel	\$ra $\leftarrow$ PC + 4, PC $\leftarrow$ jlabel
jalr   Rs	\$ra $\leftarrow$ PC + 4, PC $\leftarrow$ Rs

# Instructions MIPS

- Principe d'une simple instruction MIPS
  - **Remarque 1: La simplicité d'une instruction favorise la régularité**

<b>add</b> <b>a,   b,   c</b>  3 opérandes (registers)	<b># a = b + c;</b>  commentaire
---	---

```
# a = b + c + d + e;
```

add a, b, c  
add a, a, d  
add a, a, e

*Format:* <opcode> <sortie> <entrée 1> <entrée 2>

- Ces instructions sont des représentations symboliques de ce que MIPS comprend réellement



## Exemple

# Compilation d'une affectation C en MIPS

$$f = (g + h) - (i + j);$$

```
add    t0, g, h    # variable temporaire t0 contenant
                  g+h
```

```
add    t1, i, j    # variable temporaire t1 contenant i+j
```

sub **f**, t0, t1 # f reçoit le résultat final

Dans MIPS, ces notations représentent  
*les noms des registres*

# Les Opérandes

---

- Les Opérandes ne peuvent pas être une variable quelconque (comme en C)
  - Registres
    - Nombre Limité (32 registres généraux MIPS de 32-bit (4 octets)) sauf :
      - ❑ Le registre zéros qui vaut toujours 0, même après une écriture.
      - ❑ Le registre ra, utilisé implicitement par certaines instructions pour sauver l'adresse de retour avant un saut.
- **Remarque 2: Un registre est Plus petit et plus rapide**

# Les Opérandes


---

- Désignation: *numéros* ou *noms*

\$8 - \$15 => \$t0 - \$t7 (variables temporaires)

\$16 - \$22 => \$s0 - \$s8 (correspondent aux variables en C)

- L'utilisation des noms des registres rendent vos codes plus lisibles

f	=	(g	+	h)	-	(i	+	j);		add	\$t0,	\$s1,	\$s2
↑		↑		↑		↑		↑		add	\$t1,	\$s3,	\$s4
\$s0		\$s1		\$s2		\$s3		\$s4		sub	\$s0,	\$t0,	\$t1

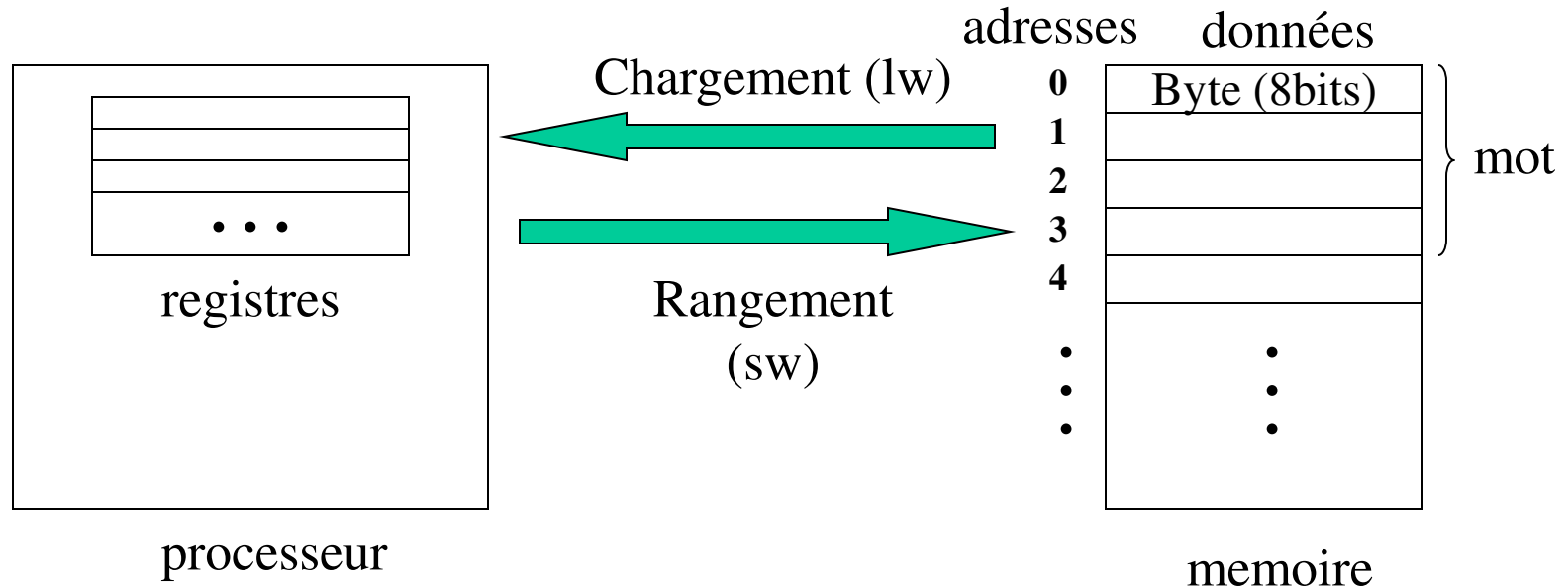
# Conventions sur les Registres

---

Nom	NumérosRegistre	Usage	Preserved on call
\$zero	0	Valeur constante 0	n.a.
\$at	1	Réservé par l'assembleur	n.a.
\$v0-\$v1	2-3	Retour de valeurs	no
\$a0-\$a3	4-7	Arguments (procédures/fonctions)	yes
\$t0-\$t7	8-15	Temporaires	no
\$s0-\$s7	16-23	Temporaires sauvegardés	yes
\$t8-\$t9	24-25	Temporaries	no
\$k0-\$k1	26-27	Réservés par l'OS	n.a.
\$gp	28	Pointeur global	yes
\$sp	29	Pointeur deepile	yes
\$fp	30	Frame pointer	yes
\$ra	31	Return address	yes

# Memoire

- Contient des instructions and des données du programme
  - Les Instructions sont recherchées (fetch) *automatiquement par l'unité de contrôle*
  - Les Données sont transférées explicitement entre la mémoire et le processeur (registres)
- Instructions de transfert de données



# Modèle de la Memoire

---

- Memoire est adressable sur 1 byte (1 byte = 8 bits)
- Seulement load/store peut accéder à la memoire de données
- Unité de transfert : *mot* (4 bytes)
  - $M[0], M[4], M[4n], \dots, M[4,294,967,292]$
- Les mots doivent être **Alignés**
  - Les mots commencent aux adresses 0, 4, ...  $4n$
- Les adresses sont 32 bits de longueur
  - $2^{32}$  bytes ou  $2^{30}$  mots

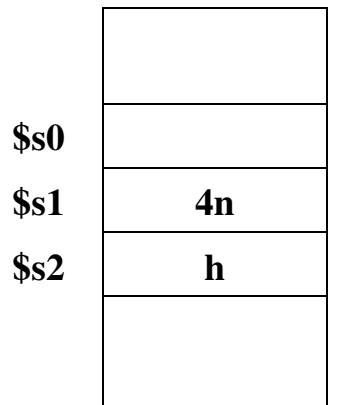
# Chargement / Rangement

$A[0] = h + A[2];$

lw \$t0, 8(\$s1)

add \$t0, \$s2, \$t0

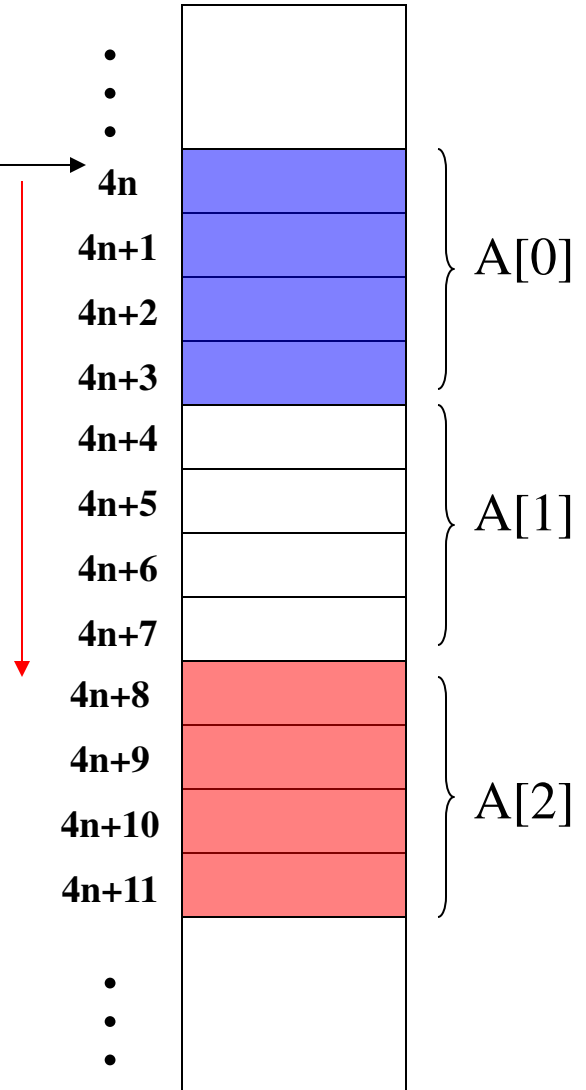
sw \$t0, 0(\$s1)



registres

Base d'adresse  
(\$s1)

Offset (8)



# Gros et Petit Boutiste

---

$$(3101)_{10} = \mathbf{12} * 16^2 + \mathbf{1} * 16^1 + \mathbf{13} * 16^0$$
$$= (00 \ 00 \ 0c \ 1d)_{16}$$

•	
•	
•	
4n	0 0
4n+1	0 0
4n+2	0 c
4n+3	1 d
•	
•	
•	

**Gros Boutiste**  
**(Big Endian)**

•	
•	
•	
4n	1 d
4n+1	0 c
4n+2	0 0
4n+3	0 0
•	
•	
•	

**Petit Boutiste**  
**(Little Endian)**



# Remarques générales

---

- ISA: Interface hardware / software
- Instructions MIPS
  - Arithmétique: `add $t0, $s0, $s1`
  - Transfert de données: `lw/sw`, for example: `sw $t1, 8($s1)`
- Les Operandes doivent être des registres
  - 32 32-bit registers
  - `$t0 - $t7` => `$8 - $15` (adresses)
  - `$s0 - $s7` => `$16 - $23` (adresses)
- Memoire: large, Tableau lineire d'une seule dimension des bytes  $M[2^{32}]$ 
  - L'adresse Memoire est un index dans le tableau des bytes
  - Les mots sont Alignés : `M[0]`, `M[4]`, `M[8]`, ..., `M[4,294,967,292]`
  - L'ordre des Bytes soit Gros/Petit boutiste