Architecture des Ordinateurs
Chapitre III
(Processeur :Architecture externe)

Mokrani Hocine

dr.mokrani@gmail.com

Chapitre III (Processeur)

- Généralités sur les processeurs.
  - Caractéristique d'un processeur.
  - Processeur et programmation (RISC VS CISC).
  - Modes d'adressage.
  - Assembleur.
- Cas d'étude: Processeur MIPS R3000.
  - Assembleur MIPS.
  - Architecture externe.
  - Jeu d'instruction.
  - Exemple d'utilisation des instructions.
- Conclusion.

Généralité sur les processeurs

Caractéristiques d’un processeur

Quels sont les caractéristiques les plus importantes?

- La fréquence d’horloge.
- Nombre de cœurs.
- Le jeu d'instruction.
- Le schéma d'exécution. (Le parallélisme)
- Nombre et taille de registre de mémoire cache.
- …

Pas de réponse définie

=> Evaluation réel, exécuter plusieurs programmes (benchmarks)

Temps CPU

- $TEMPS_{CPU}$ = NI X CPI X Period
  - NI : Nombre d'instruction.
  - CPI: Cycle d'horloge par instruction.
  - Period: fréquence de l'horloge.

- Deux manière pour augmenter les performances:
  - Augmenter la fréquence d'horloge.
  - Réduire le temps de cycle par instruction.

- Deux buts opposés !!

Processeur et Programmation

D'un point de vue de la programmation, le processeur offre:

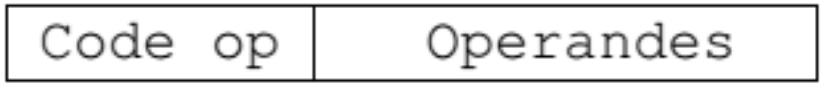
- Un certain jeu d'instructions qu'il sait exécuter.

- Un certain nombre de registres :
  - Utilisable/modifiable directement par le programme :
    registres de travail - pointeur de segment
    Registres vu par le jeu d'instructions
  - Modifiable indirectement par le programme :
    compteur ordinal -pointeur de pile - registre d'instruction - registre d'états
    registres manipulés implicitement par le jeu d'instructions

- Un certain nombre de manière d'accéder à la mémoire : modes d'adressage

Langage machine

- Le langage machine est le langage directement interprétable par le processeur.
- Le langage est défini par un ensemble d'instructions que le processeur exécute directement.
- Chaque instruction correspond à un nombre binaire (codé selon le cas sur un mot de 8 bits, de 16 bits, ... :
  - une partie codant l'opération à exécuter appelé **codeop** ou code opération.
  - une partie pour les **opérandes**.



- Un programme en langage machine est une suite de mots (Instructions) codant les opération et les opérandes.
- Chaque processeur possède son propre code machine.

Jeu d'instructions

- Le jeu d'instructions est l'ensemble des opérations élémentaires qu'un processeur peut accomplir.
- Le type de jeu d'instructions d'un processeur détermine son architecture.
- Deux types d'architectures:
  - RISC (Reduced Instruction Set Computer).
  - CISC (Complex Instruction Set Computer).

Jeu d'instructions

CISC VS RISC

CISC	RISC
AMD/Intel, Motorola...	PowerPC, MIPS, SPARC, Alpha, ARM
Complex Instruction Set Computer	*Reduced Instruction Set Computer*
Instructions complexes (plusieurs cycles d'horloge)	Instructions simples (un cycle)
Instructions de taille variable	Instructions de taille fixe (1 mot)
Instructions séquentielles	Pipeline

⊕ Programmation de plus haut niveau.
⊕ Compilation moins complexe
⊕ Programmation plus compacte. moins d'occupation en Mémoire et à l'exécution.

⊖Complexité du processeur.
⊖Fréquence d'horloge réduite.
⊖ Instructions lentes

⊕ Mode d'adressage réduit.
⊕ Pipeline plus simple et plus efficace. Traitement efficace.
⊕Fréquence d'horloge réduite.
⊕Instruction rapides.

⊖Beaucoup de registre
⊖Programme plus volumineux
⊖Compilation plus complexe.

Jeu d'instructions (Exemple)

- **Exemple :**

```
int chiche, a, b;
chiche = a + b;
```

- **CISC**

```
add [chiche], [a], [b]
```

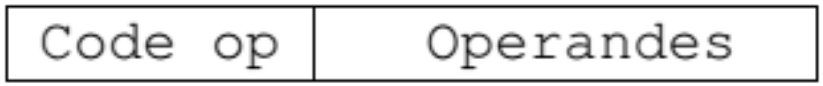
- **RISC**

```
lw R3, [a]
lw R4, [b]
add R5, R3, R4
sw R5, [chiche]
```

Modes d’adressage

- Les instructions du langage machine manipulent des données. Selon où ces données se trouvent, on parle de différents **modes d’adressage**.

- Comment interpréter les **Opérandes** pour trouver les données de l’instruction **Code op**.



Modes d'adressage

- **Adressage direct:** **Opérandes** est l'adresse où se trouve la donnée en mémoire.
- **Adressage relatif :** **Opérandes** contient un déplacement relatif par rapport à une adresse qui se trouve dans un registre précis (par exemple, le compteur ordinal PC).
- **Adressage indirect :** **Opérandes** contient le numéro d'un registre dont le contenu est l'adresse où se trouve la donnée en mémoire.
- **Adressage (indirect) indexé:** **Opérandes** contient le numéro d'un registre contenant une adresse **a**. La donnée est en mémoire à l'adresse **a + i** , où **i** est le contenu d'un autre registre dans **Opérandes** ou d'un registre spécifique, appelé registre d'index.
- **Adressage immédiat :** **Opérandes** est la valeur utilisée par l'instruction.

Cycle d'exécution d'une instruction

1. **Récupérer (en mémoire) l'instruction à exécuter :** RI ← Mémoire[PC]
   L'instruction à exécuter est présente en mémoire à l'adresse contenue dans le compteur de programme PC et est placé dans le registre d'instruction RI.

2. **Le compteur de programme est incrémenté :** PC ← PC + 4
   Par défaut, la prochaine instruction à exécuter est la suivante en mémoire (sauf si l'instruction est un saut).

3. **L'instruction est décodée :** On identifie les opérations qui vont devoir être réalisées pour exécuter l'instruction.

4. **L'instruction est exécutée :** Elle peut modifier les registres (opérations arithmétiques - lecture en mémoire), modifier la mémoire (écriture), modifier le registre PC (instructions de saut).

Assembleur

- Le **langage assembleur** ou (**assembleur**) est le langage de programmation.

  C'est une version lisible par un humain du langage machine, obtenu en remplaçant les valeurs entières du langage machine par des mnémoniques (instruction du langage assembleur).

- Pour la même machine, il peut exister différents **langages assembleur** : variation sur la syntaxe, mais un **langage machine**.

Cas d'étude
Processeur MIPS R3000

Processeur MIPS
(Un peu d'histoire)

- **1981:** John Hennessy (Stanford) invente un processeur **RISC** sans inter blocage entre les **pipelines**.
- **1984:** invente sa boite : MIPS Computer Systems.
- **1985:** il sort le premier processeur MIPS : le R2000.
- **1988:** il sort le R3000 Silicon Graphics.
- **1991 :** R4000 64 bits.
- **1992 :** Silicon Graphics (SGI) rachète MIPS, et changent le nom : MIPS Technologies.
- **1994 :** R8000, processeur super scalaire.
- **1997 :** 48 millions de MIPS vendus (plus que le motorola 68000).
- **1999 :** création de 2 licences différentes :
  - 32-bits MIPS (MIPS32)
  - 64-bits MIPS (MIPS64)

  Ces licences ont été achetées entre autres par Sony, Nec, Toshiba, Philips.

Assembleur MIPS

- Assembleur du processeur MIPS R3000
  (processeur de type RISC)

- processeur MIPS :
  NEC, SGI, console (Sony PSP, PS2), AdslBox (FreeBox, NeufBox)

- Assembleur proche des autres assembleurs RISC.

Architecture externe

- L'architecture externe est l'interface entre le processeur et le programmeur.
- Une architecture externe est composée de :
  - Registres visibles.
  - d'instructions.
  - Adressage.
  - Système d'interruptions / exceptions.

Architecture externe (MIPS R3000)

Processeur 32 bits constitué de:

- 32 registres de 32 bits
- une mémoire vive adressable de 2^{32} octets.
- Un compteur de programmes PC (Program Counter) sur 32 bits.
- Un registre d'instruction RI sur 32 bits.

Caractéristiques d'exécution:

- Le programme est stocké en mémoire.
- L'adresse de l'instruction en cours d'exécution est stockée dans le registre PC.
- L'instruction en cours d'exécution est stockée dans le registre RI.
- Une instruction est codée sur 32 bits.
- Stockage des données: Le mot de poids fort est stocké en dernier : **little-endian**
- Le processeur peut lire des instructions (32 bits), des mots (32 bits), des demi-mots (16 bits) et des octets (8 bits). Il peut écrire des mots, des demi-mots et des octets.
- Bus d'adresses de 32 bits et Bus de données de 8 bits

Registres MIPS

Les 32 registres du processeur MIPS sont :

Nom	Numéro	Description
$zero	0	constante 0
$at	1	réservé à l'assembleur
$v0,$v1	2-3	résultats d'évaluation
$a0,..,$a3	4-7	arguments de procédure
$t0,..,$t7	8-15	valeurs temporaires
$s0,..,$s7	16-23	sauvegardes
$t8,$t9	24-25	temporaires
$k0,$k1	26-27	réservé pour les interruptions
$gp	28	pointeur global
$sp	29	pointeur de pile
$fp	30	pointeur de bloc
$ra	31	adresse de retour

Structure des instructions

- Format binaire
  - Toutes les instruction son en 32 bits.
  - Les bits [26-31] sont réservés pour le **op-code.**
  - Les bits [0-25] dépend de la format d'instructions (**Opérande**).
- Les instructions sont groupées en des catégories:
  - Instructions arithmétique et logique.
  - Instructions d'accès mémoires.
  - Instructions de branchement conditionnel.
- Sur le processeur **Mips** trois formats sont possibles:
  - Instructions de type immédiat (Format I)
  - Instructions de type saut (Format J)
  - Instructions de type registre (Format R)
- Les 6 bits du **op-code** détermine le format de l'instruction.

Exemple (Opération Arithmétique)

Code C	Assembleur
`A = B + C`	`add $s0, $s1, $s2`

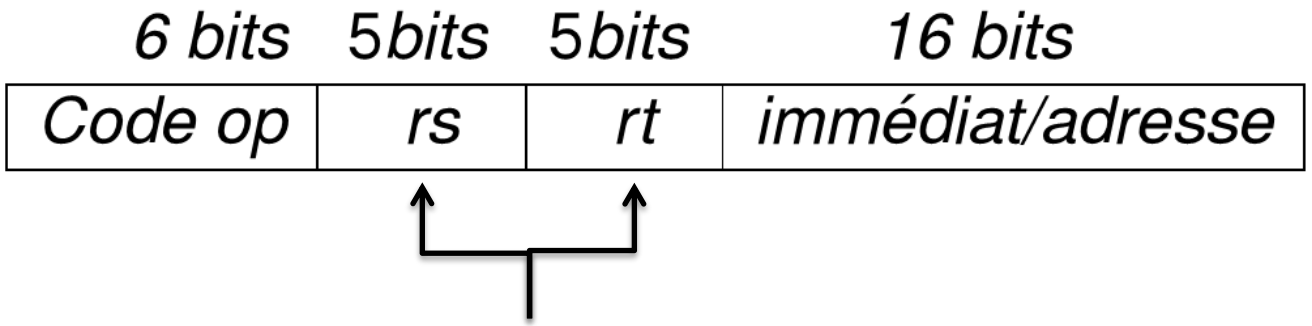
- Toutes les opérandes se trouvent dans des registres.
- Le choix des registres est déterminé par le compilateur :
  ici, **A → $s0, B → $s1, C →$s2**
- Le résultat est placé dans $S0, la première opérande.

Code C	Assembleur

`A = B + C + D`
`E = F - A`
	`add $t0, $s1, $s2`
	`add $s0, $t0, $s3`
	`sub $s4, $s5, $s0`

- Ici, **A → $s0, B → $s1, C → $s2, D→ $s3, E→ $s4, F → $s5**.
- Toutes les opérations arithmétiques ont trois opérandes
- C'est nécessaire car une instruction doit se coder sur un nombre borné de bits.

Format d'instructions I (I-Format)



On peut identifier 32 registres pour chaque 5 bits

- **rs :** registre source.
- **rt :** registre cible / condition de branchement.
- **immédiat/adresse :** opérande immédiate ou déplacement d'adresse.

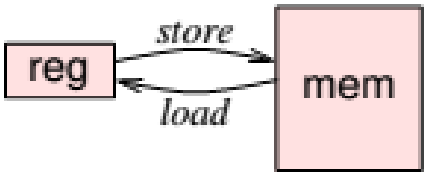
Format d'instructions I (I-Format)

Numeric code	Operation code	Description
Memory accesses		
35	`lw`	load word
43	`sw`	store word
Conditional branching		
4	`beq`	branch if equal
5	`bne`	branch if not equal
Arithmetic and logic with immediate operand		
8	`addi`	immediate addition
13	`ori`	immediate or

lw $2, 10($3)
Copie dans le registre **$2** la valeur située dans la mémoire principale à l'adresse **m** obtenue en ajoutant **10** au nombre stocké dans la registre **$3**.

beq $1, $2, 100
if $1=$2
PC → PC+4+4*100
else PC→ PC+4

addi $1, $2, 10
A = B +10;
Ici, A → $1, B → $2



Format d'instructions I (I-Format)

Exercice:

- Ecrire un exemple d'utilisation de l'instruction **sw.**

  sw $2, 10($s3)

- Ecrire un exemple d'utilisation de l'instruction **bne.**

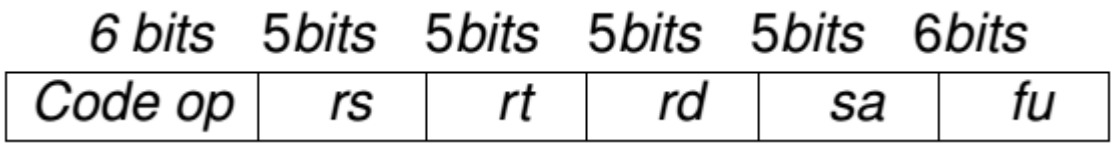
  bne $1, $2, 100

- Ecrire un exemple d'utilisation de l'instruction **ori.**

  ori $1, $2, 8

  Réalise un "ou logique" entre les la valeur binaire de 8 (étendus à 32 bits en mettant ceux de poids fort à 0) et le contenu de **$2,** le résultat étant placé dans **$1.**

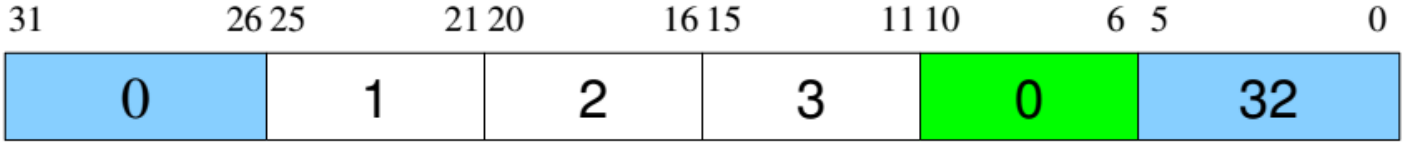
Format d'instructions R (R-Format)



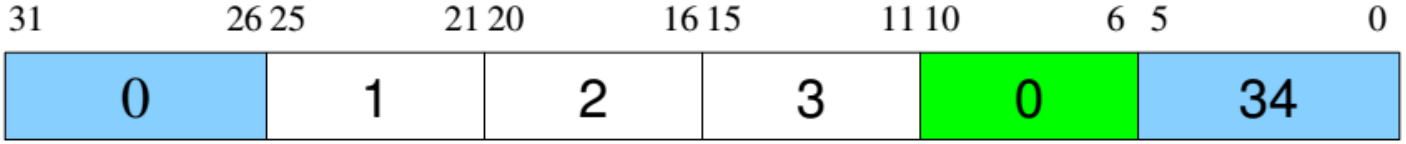
- **rs :** registre source 1,
- **rt :** registre source 2,
- **rd :** registre destination,
- **sa :** nombre de décalage à effectuer (shift amount) Un nombre immédiat.
- **fu :** identificateur de la fonction.

Format d'instructions R (R-Format)

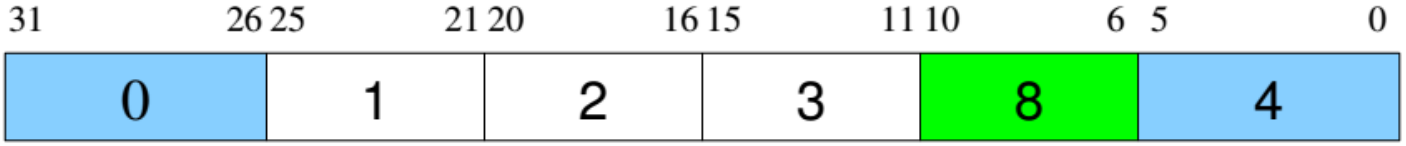
```
add $3, $1, $2 # $3 ← $1 + $2
```




```
or $3, $1, $2 # $3 ← $1 or $2
```

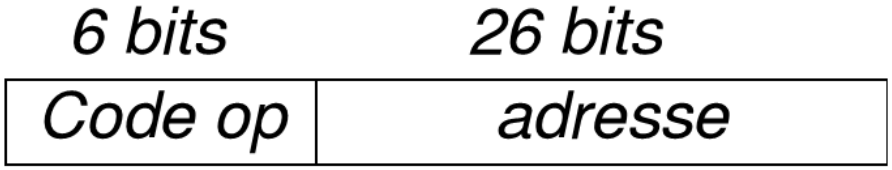



```
sll $3, $2, 8 # $3 ← shift_left($2, 8)
```



- **sll**: Shift left logical
  Décalage du nombre dans **$2** vers la gauche de **huit** positions et mettre le résultat dans **$3**.

Format d'instructions J
J-Format



Les adresses dans les instructions ne sont pas sur 32 bits !

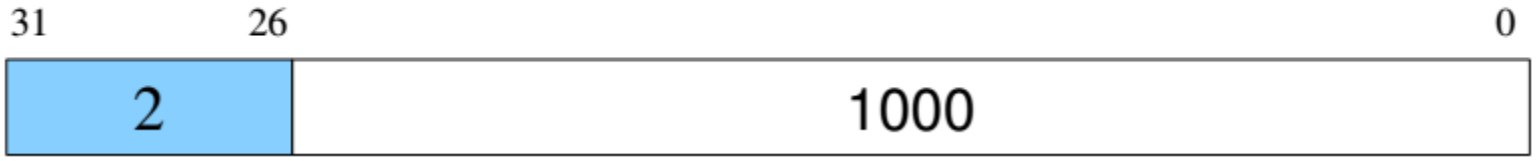
- Pour les instructions de type I-Format : **16** bits
  ➔ Adresse = (PC + 4) + signé(16 bits) * 4 **adressage relatif**

- Pour les instructions de type J-Format : **26** bits
  ➔ On obtient l'adresse d'un mot mémoire (de **32** bits) en ajoutant devant les 26 bits les 4 bits de poids fort de PC (Il faut multiplier par 4 pour l'adresse d'un octet) **adressage direct restreint**

PC <- PC_{31-28} :: IR_{25-0} :: 00

Format d'instructions J
J-Format

```
j 1000 # PC ← ((PC + 4) & 0xf0000000) + 4 * 1000
```




```
jal 1000 # $31 ← PC + 4
# PC ← ((PC + 4) & 0xf0000000) + 4 * 1000
```

- « **`jal`** » est utilisable pour les appels de fonctions.
- Le « **`return`** » est implémenté par **`jr $ra`**

```
jr $ra # $ra = $31
```

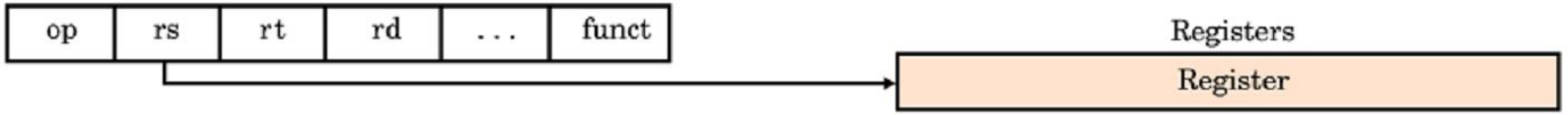
Mode d’adressage MIPS

Mode d'adressage Mips

1. Immediate addressing

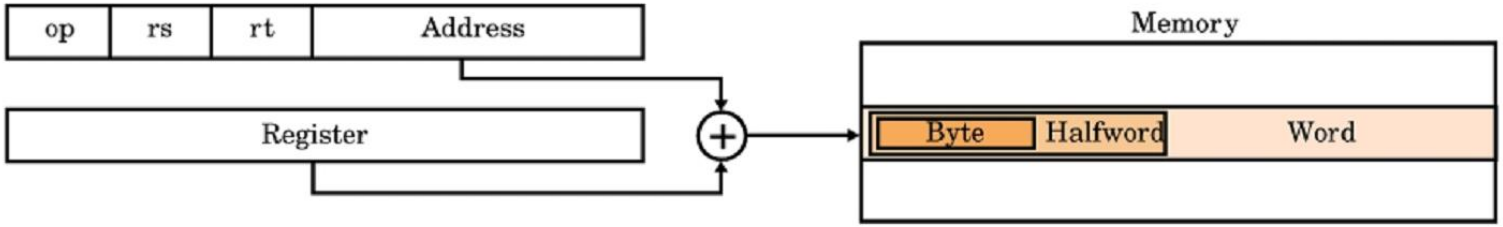
Addi $1, $2, 4

2. Register addressing



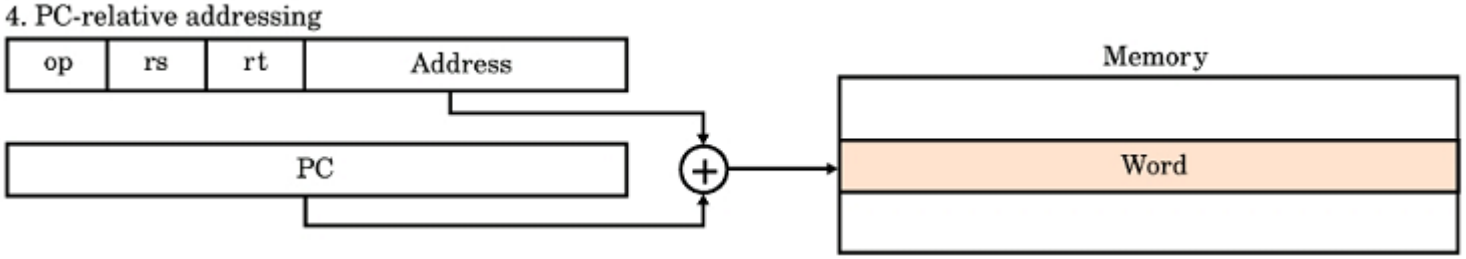
Add $1, $2, $3

3. Base addressing

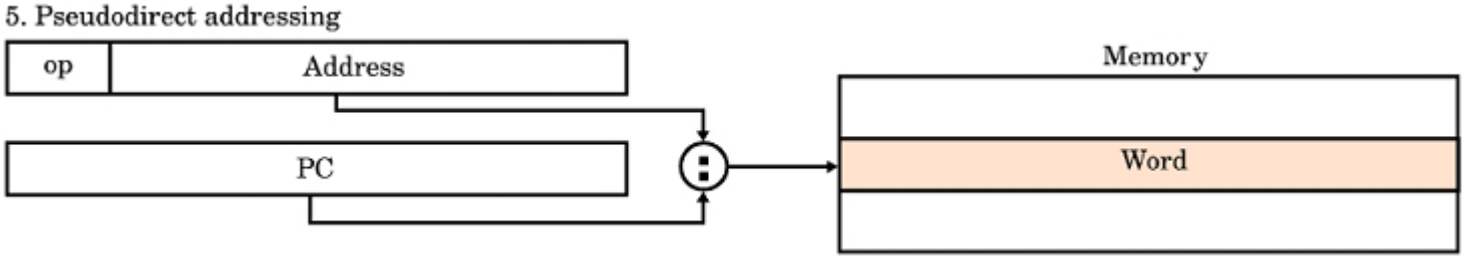


lw $1, 100 ($2)

Mode d'adressage Mips



bne $1, $2, 100



j 1000

Exemples d'utilisation des instructions (Arithmétique)

Code C	Assembleur

`A = B + C + D`
`E = F - A`
	`add $t0, $s1, $s2`
	`add $s0, $t0, $s3`
	`sub $s4, $s5, $s0`

- Ici, **A → $s0, B → $s1, C → $s2, D→ $s3, E→ $s4, F → $s5**.
- Toutes les opérations arithmétiques ont trois opérandes
- C'est nécessaire car une instruction doit se coder sur un nombre borné de bits.

Exemples d’utilisation des instructions

Instructions Arithmétique

- **Comment traduire A=B ?**

  Sachant que A → $1, B→ $2 et que le registre $0 vaut toujours 0 on peut écrire :

  add $1, $0, $2

- Pratiquement, il vaut mieux utiliser l'instruction move :

  A = B <-> move $1, $2

- move est une **pseudo-instruction** : sa traduction en langage machine est celle de add $1, $0, $2.

Instruction de chargement directe

- **li** r, r, VAL (ou li r, VAL)
  (load immediate) charge la valeur val (sur 32 bits) dans le registre r.

- **li** est une **pseudo-instruction**, sa traduction en assembleur (langage machine) est:
  - **lui** r, r, VAL 1
  - **ori** r, r, VAL 2

- **VAL1**, **VAL2** sont respectivement les **16 bits** de **poids fort** et de **poids faible** de **VAL**.

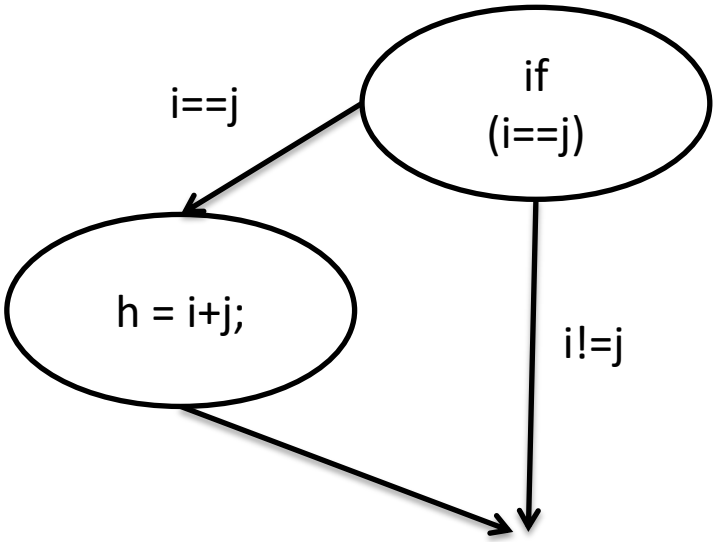
- **lui** r, r, VAL1

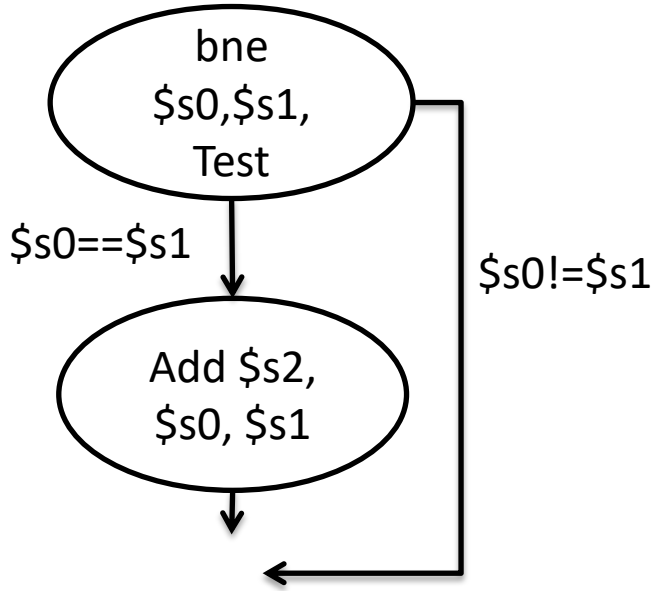
  (load upper immediate) **utiliser** les **16 bits** de **VAL1** pour **initialiser** les **16 bits** de **poids fort** du registre **r**, les **16 bits de poids faible** étant **mis à 0**.

- ori r, r VAL2

  Réaliser un **"ou logique"** entre les **16 bits de VAL2** (**étendus à 32 bits** en mettant ceux de poids fort à 0) et le **contenu de r**, le **résultat** étant **placé dans r**.

Branchements conditionnels



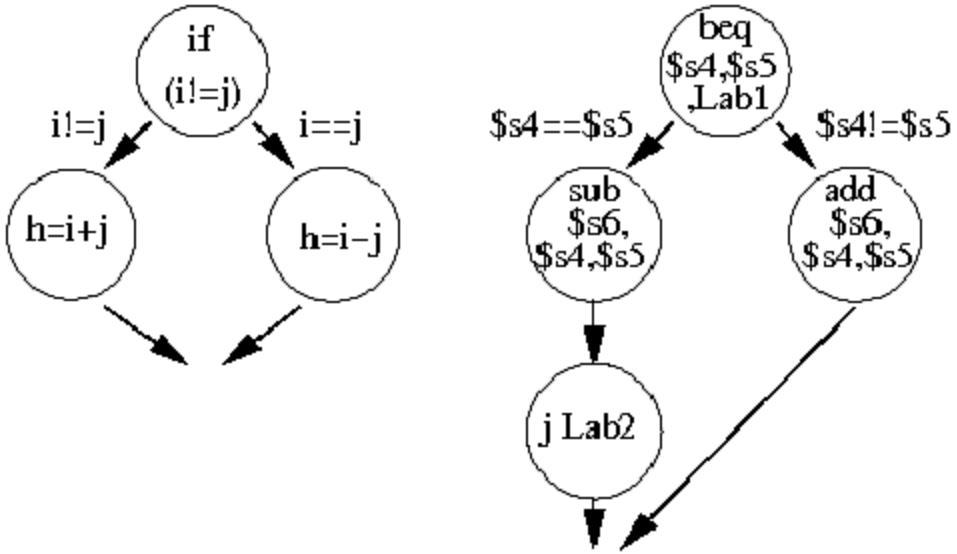


Code C	Assembleur

`if (i==j) h =i+j;`	`bne $s0, $s1, Test`
	`add $s2, $s0, $s1`
	`Test :`

Ici, i↦$s0, j↦$s1, h↦$s2.

Branchements inconditionnels

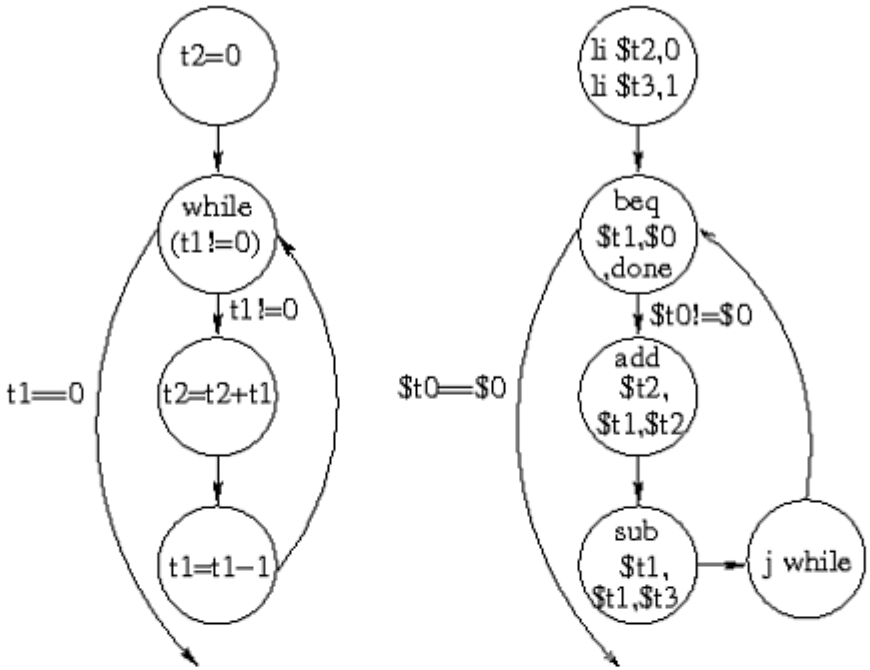


Code C	Assembleur

`if (i !=j) h =i+j`
`else h =i-j`
	`beq $s4, $s5, Lab1`
	`add $s6, $s4, $s5`
	`j Lab2`
	`Lab1 :sub $s6, $s4, $s5`
	`Lab2 :`

Ici, i → $s4, j → $s5, h→$s6.

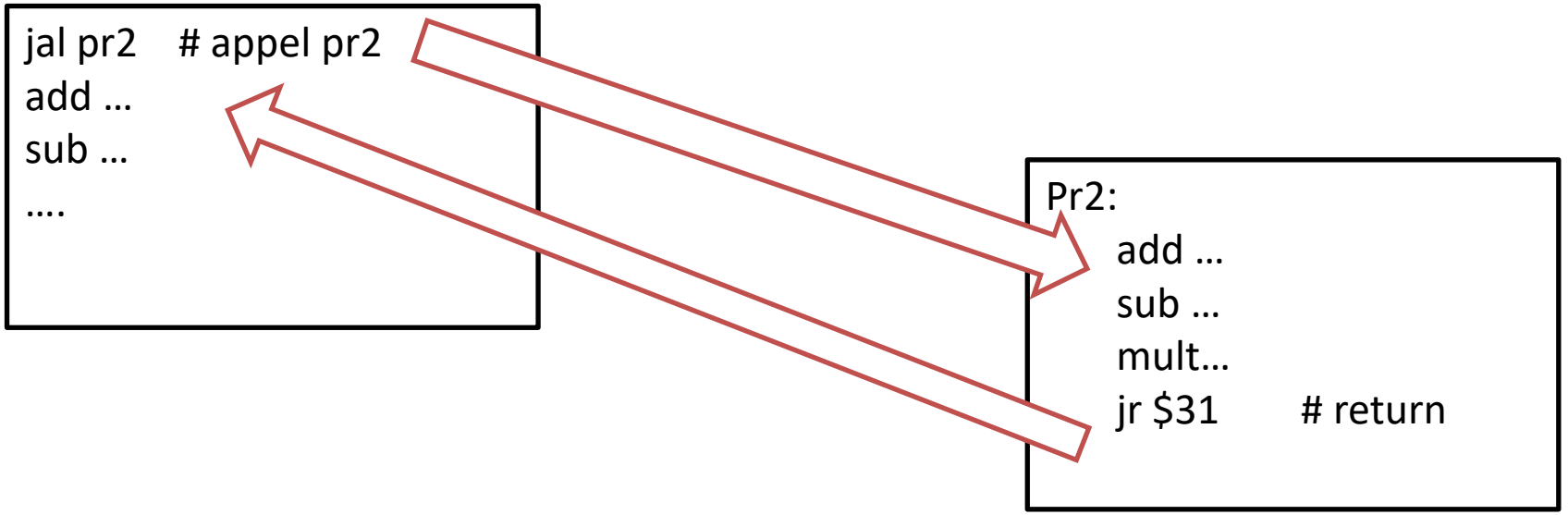
Branchement inconditionnel



Code C	Assembleur

t2=0
while (t1 != 0){
t2 = t2 + t1
t1=t1-1 }
	li $t2, 0
	li $t3, 1
	while :beq $t1, $0, done
	add $t2, $t1, $t2
	sub $t1, $t1, $t3
	j while
	done :

Appel de sous Programme



- L'instruction **jal pr2** permet d'exécuter le sous-programme de label **pr2**. L'adresse de retour est stocker dans le registre **$31**.
- Cependant,
  - Le sous-programme peut affecter les valeurs contenues dans les registres au moment de l'appel : pas de notion de variables locales et de portée/masquage de variables.
  - La sauvegarde de l'adresse de retour dans un registre ne permet pas l'enchaînement des appels à des sous-programmes, encore moins des sous-programmes récursifs.

Appel de sous programme

- Solution :
  - Sauvegarder la valeur des registres (en mémoire) de l'appelant et restaurer ces valeurs à l'issue de l'appel.
  - Sauvegarder l'adresse de retour du programme appelant en mémoire.

- On sauvegarde les (une partie des) registres en mémoire dans une pile.

- Les registres **$a0-$a3** sont ceux qui ne sont pas sauvegardées car ils contiennent lors de l'appel la valeur des paramètres effectifs, et au retour les valeurs retournés par le sous-programme.

Appel de sous programme (Utilisation d'une Pile)

- Une **pile** est une mémoire qui se manipule via deux opérations :
  - push : empiler un élément (le contenu d'un registre) au sommet de la pile.
  - pop : dépiler un élément (et le récupérer dans un registre).

- Ces deux instructions n'existent pas en assembleur **MIPS**, mais elles peuvent être "simulées".

  - En utilisant les instructions sw et lw.
  - En stockant l'adresse du sommet de pile dans le registre $sp (le pointeur de pile)

Appel de sous-programmes (politique de gestion de la pile)

- Deux politiques de sauvegarde des registres :
  - sauvegarde par l'appelant : le programme appelant sauvegarde tous les registres sur la pile (avant l'appel).
  - sauvegarde par l'appelé : le programme appelant suppose que tous les registres seront préservés par le sous-programme appelé.

- Remarque:

  un sous-programme doit rendre la pile intacte.

Conclusion

- Processeur MIPS de 32 bits.
- Jeu d'instructions de type RISC.
  - Plusieurs formats d'instructions: I-Format, R-Format, J-Format.
  - Pas de gestion réelle de la pile.
- Plusieurs modes d'adressage.
  Adressage immédiat, adressage par registre, Adressage basique, adressage relatif, adressage direct restreint.

- Comment les instructions sont exécutées sur le processeurs?
- Peut-on exécuté plusieurs instructions au même temps?

  => Etudier l'architecture interne du processeur Mips.