

Architecture des ordinateurs (X31I050)

Frédéric Goualard

Laboratoire d'Informatique de Nantes-Atlantique, UMR CNRS 6241
Bureau 112, bât. 11
Frederic.Goualard@univ-nantes.fr

Instructions & chemins de données

Définition d'un jeu d'instructions :

- ▶ Instructions disponibles ?
 - ▶ Jeu complet
 - ▶ Implémentation efficace
 - ▶ (Facile d'utilisation)
- ▶ Type de jeu d'instructions ?
 - ▶ Registres, mémoire, pile
- ▶ Représentation en mémoire ?

$A \leftarrow 3 + 5$?

Machine à pile	Machine à accumulateur	Machine à registres
PUSH X $top(pile) \leftarrow X$	LOAD X $acc \leftarrow X$	LOAD R_i, X $R_i \leftarrow X$
POP X $X \leftarrow top(pile)$	STORE X $X \leftarrow acc$	STORE R_i, X $X \leftarrow R_i$
ADD $POP\ t_2 ; POP\ t_1$ $PUSH\ t_1 + t_2$	ADD X $acc \leftarrow acc + X$	ADD R_i, R_j, R_k $R_i \leftarrow R_j + R_k$
push 3 push 5 add pop A	load 3 add 5 store A	load R1, 3 load R2, 5 add R3, R1, R2 store R3, A

Registres (`add r0, r1, r2`).

- 😊 Simplicité, nombre de cycles
- 😞 Augmente le nombre d'instructions d'un programme

Registre/mémoire (`add r0, MEM`).

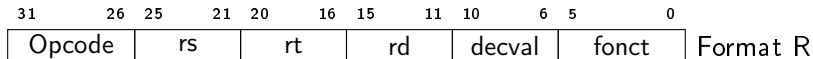
- 😊 Bonne densité du code, pas besoin de chargement des données
- 😞 Destruction d'un opérande, nombre de cycles variable

Mémoire/mémoire (`add MEM0, MEM1, MEM2`).

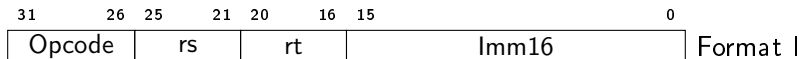
- 😊 Code très compact (pas de load/store)
- 😞 Problème d'accès mémoire

- ▶ *Endianness*
- ▶ Contraintes d'alignement
- ▶ Modes d'adressage :

Modes	Exemple (non MIPS)	Sens
Registre	add r0, r1, r2	$r_0 \leftarrow r_1 + r_2$
Immédiat	add r0, 4	$r_0 \leftarrow r_0 + 4$
Direct	add r0, (0x1001)	$r_0 \leftarrow r_0 + \text{MEM}[0x1001]$
Indirect	add r0, (r1)	$r_0 \leftarrow r_0 + \text{MEM}[r_1]$
Basé	add r0, 10(r1)	$r_0 \leftarrow r_0 + \text{MEM}[r_1 + 10]$
Indexé	add r0, (r1+r2)	$r_0 \leftarrow r_0 + \text{MEM}[r_1 + r_2]$



add, sub, ...

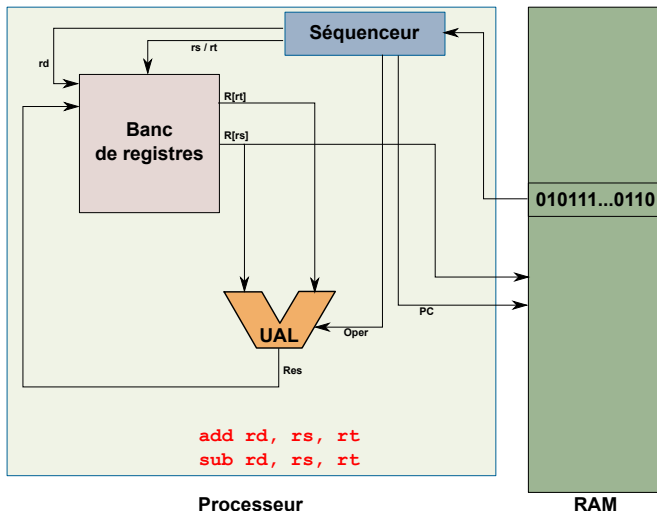


lw, sw, addi, beq, ...



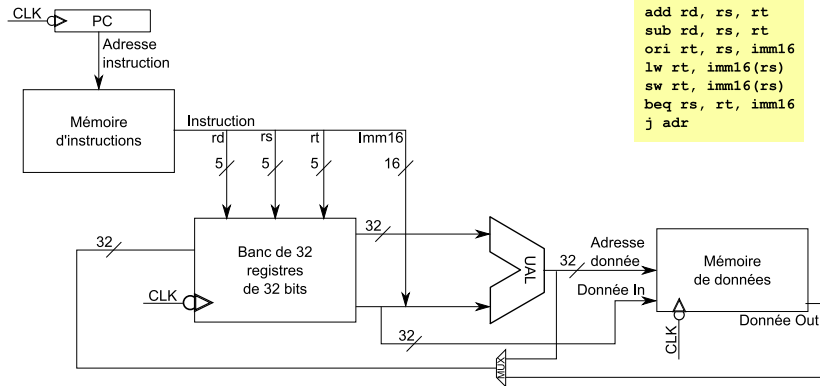
jal, j, ...

Vision simplifiée du chemin de données

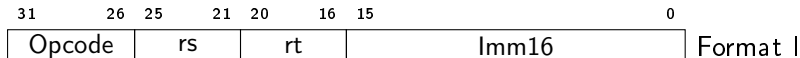


- ▶ Architecture du jeu d'instruction
 - ▢▶ Langage Transfert de Registres (*RTL*)
- ▶ RTL
 - ▢▶ Composants du chemin de données
 - ▢▶ Interconnection des composants
- ▶ Composants du chemin de données
 - ▢▶ Signaux de contrôles
- ▶ Signaux de contrôles
 - ▢▶ Logique de contrôle

- ▶ Addition et soustraction
 - ▶ `add rd, rs, rt`
 - ▶ `sub rd, rs, rt`
- ▶ « Ou » immédiat
 - ▶ `ori rt, rs, imm16`
- ▶ Chargement et rangement
 - ▶ `lw rt, imm16(rs)`
 - ▶ `sw rt, imm16(rs)`
- ▶ Branchement conditionnel
 - ▶ `beq rs, rt, imm16`
- ▶ Saut inconditionnel
 - ▶ `j adr`

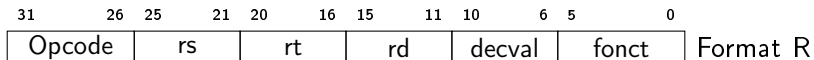


```
add rd, rs, rt
sub rd, rs, rt
ori rt, rs, imm16
lw rt, imm16(rs)
sw rt, imm16(rs)
beq rs, rt, imm16
j adr
```



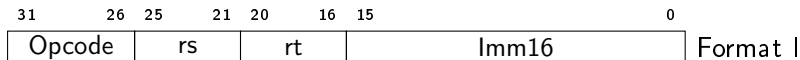
`lw rt, imm16(rs)`

1. $\text{MEM}[\text{PC}]$ *Extraire l'instruction de la mémoire*
2. $\text{PC} \leftarrow \text{PC} + 4$ *Calcul adresse prochaine instruction*
3. $\text{Adr} \leftarrow \text{R}[\text{rs}] + \text{SignExt}(\text{Imm16})$ *Calcul de l'adresse mémoire*
4. $\text{R}[\text{rt}] \leftarrow \text{MEM}[\text{Adr}]$ *Chargement de la donnée*



add rd, rs, rt

1. MEM[PC] *Extraire l'instruction de la mémoire*
2. $PC \leftarrow PC + 4$ *Calcul adresse prochaine instruction*
3. $R[rd] \leftarrow R[rs] + R[rt]$ *Addition*



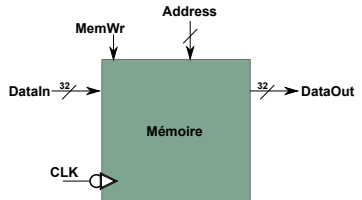
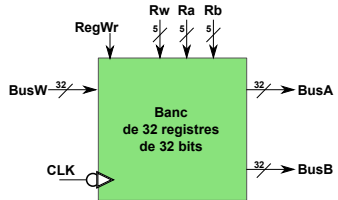
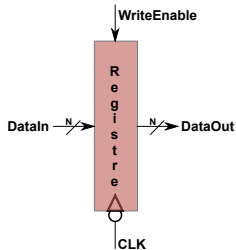
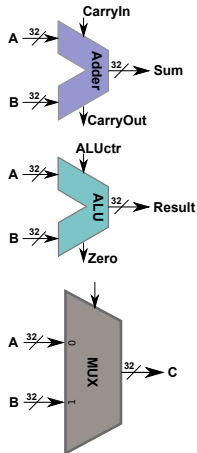
ori, rt, rs, imm16

1. MEM[PC] *Extraire l'instruction de la mémoire*
2. $PC \leftarrow PC + 4$ *Calcul adresse prochaine instruction*
3. $R[rt] \leftarrow R[rs] \vee \text{ZeroExt}(\text{Imm16})$ *Ou bit-à-bit*

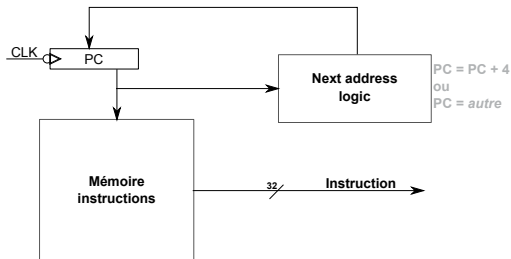


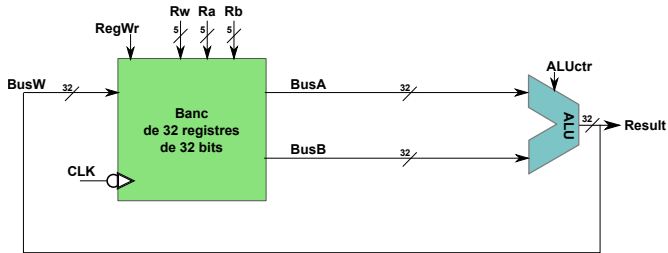
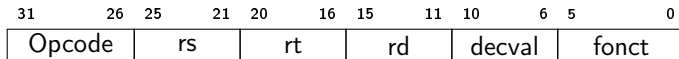
j adr

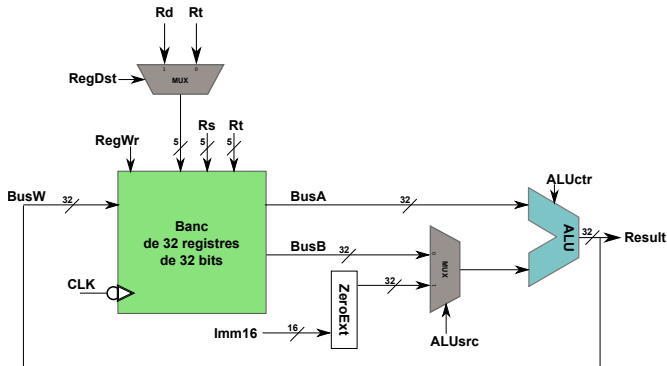
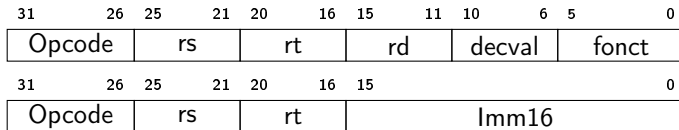
1. MEM[PC] *Extraire l'instruction de la mémoire*
2. $PC\langle 31:2 \rangle \leftarrow (PC\langle 31:28 \rangle \ll 26) \mid Adresse\langle 25:0 \rangle$ *Calcul adresse saut*

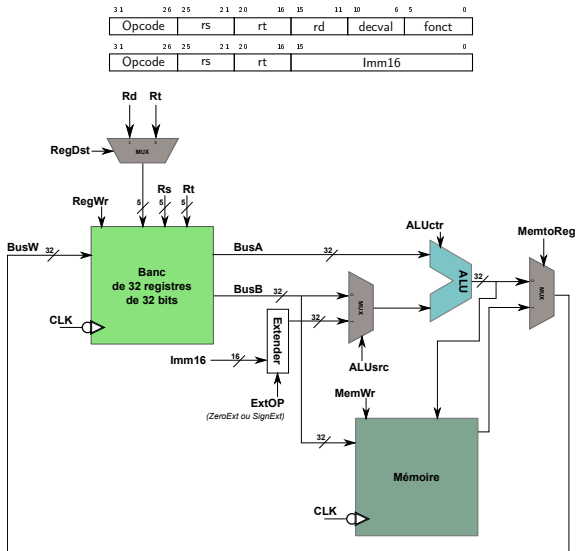


Extraction de l'instruction courante et mise à jour de PC









- ▶ PC : adresse sur 32 bits de l'instruction suivante
- ▶ Chaque instruction est sur 4 octets
 - ⇒ 2 bits de poids faible toujours nuls
 - ⇒ PC stocké *sur 30 bits*

Modification de PC :

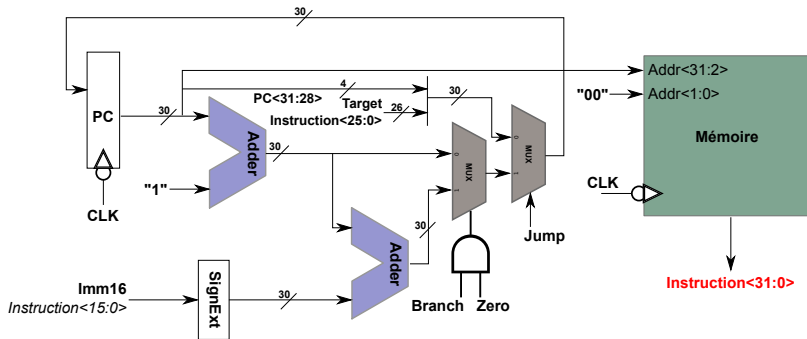
Séquentiel. $PC\langle 29:0 \rangle \leftarrow PC\langle 29:0 \rangle + 1$

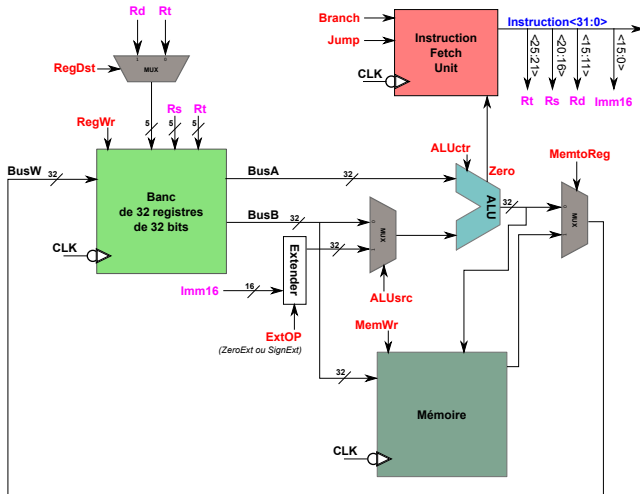
Saut relatif. $PC\langle 29:0 \rangle \leftarrow PC\langle 29:0 \rangle + 1 + \text{SignExt}(\text{Imm16})$

Saut absolu. $PC\langle 29:0 \rangle \leftarrow PC\langle 29:26 \rangle \mid \text{Adresse}\langle 25:0 \rangle$

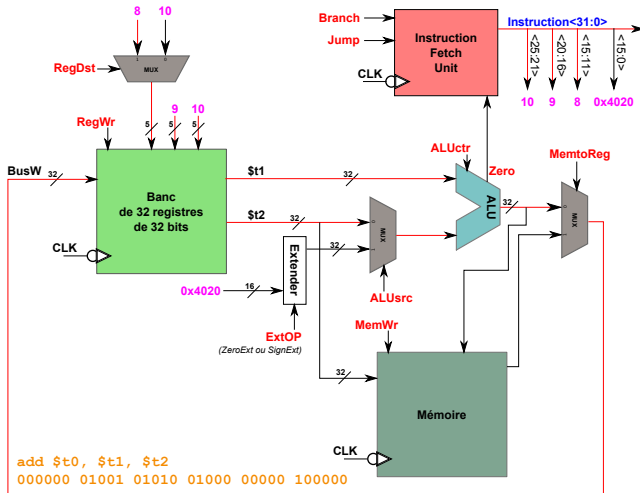
Mémoire adressée sur 32 bits : $PC\langle 29:0 \rangle \ll 2$

LS2M Instruction Fetch Unit





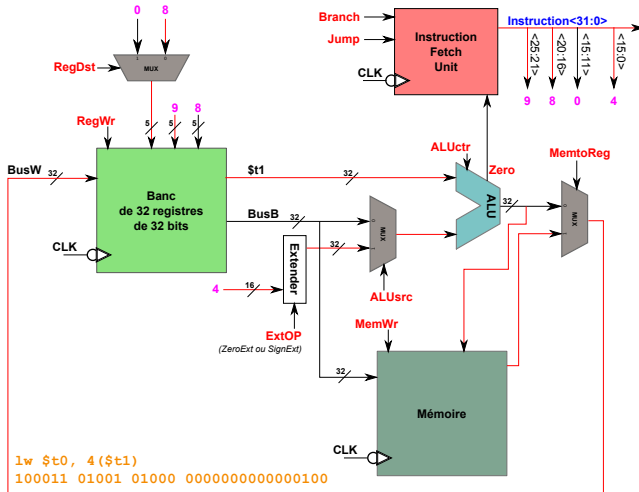
Détermination des signaux en fonction des instructions



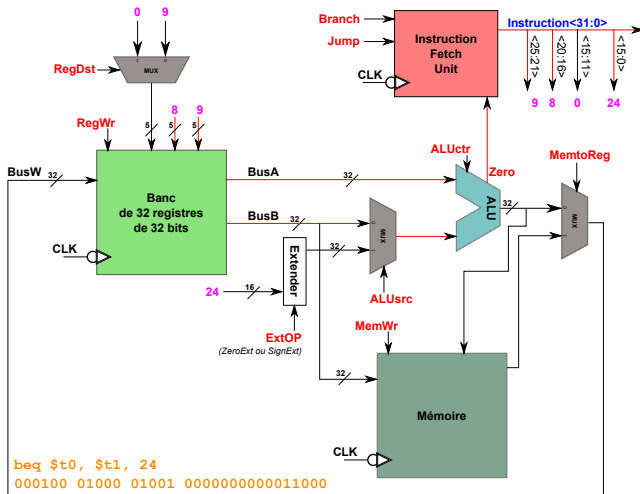
Détermination des signaux en fonction des instructions



ordinateurs



Détermination des signaux en fonction des instructions



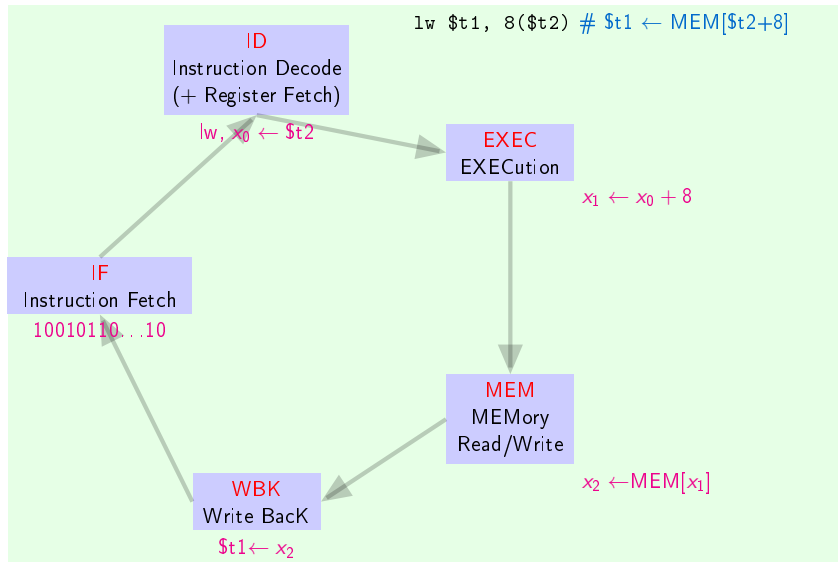
Détermination des signaux en fonction des instructions

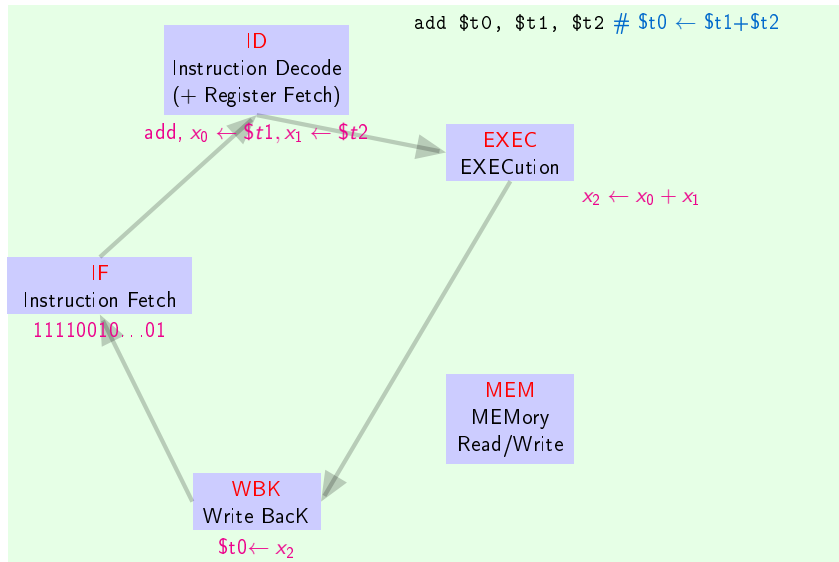
Processeur mono-cycle

- 😊 Simple à implémenter
- 😞 Temps de cycle dépend de l'instruction la plus longue
- 😞 Des unités fonctionnelles du CPU inutilisées à chaque cycle

Solution :

- ▶ Décomposer chaque instruction en étapes exécutées en un cycle : *processeur multi-cycle*





Processeur mono-cycle

- 😊 Simple à implémenter
- 😞 Temps de cycle dépend de l'instruction la plus longue
- 😞 Des unités fonctionnelles du CPU inutilisées à chaque cycle

Solution :

- ▶ Décomposer chaque instruction en étapes exécutées en un cycle : *processeur multi-cycle*
- ▶ Optimisation : optimiser l'utilisation des unités fonctionnelles : *pipeline*