Architecture des ordinateurs

Licence Informatique - Université de Provence

Jean-Marc Talbot

jtalbot@cmi.univ-mrs.fr

Processeur : description - fonctionnement - microprogrammation

Processeur : description

Au coeur du processeur

On trouve au sein d'un processeur :

- des éléments de mémorisation : (banc de) registres - cache
- des éléments de calcul : unité arithmétique et logique (UAL-ALU) - unités de calcul flottant (FPU - Floating Point Unit)
- des éléments de commandes : unité de contrôle/commande

Unité de calcul

- Unité arithmétique et logique : ALU
  calculs sur les entiers - opérations booléennes
- Unité de calcul flottant : (FPU - Floating Point Unit)

  calculs sur les flottants : sqrt, sin, ...
- unité multimédia :

  calcul vectoriel (même instruction sur plusieurs donnée en parallèle)
  Intel MMX et SSE, AMD 3DNow !

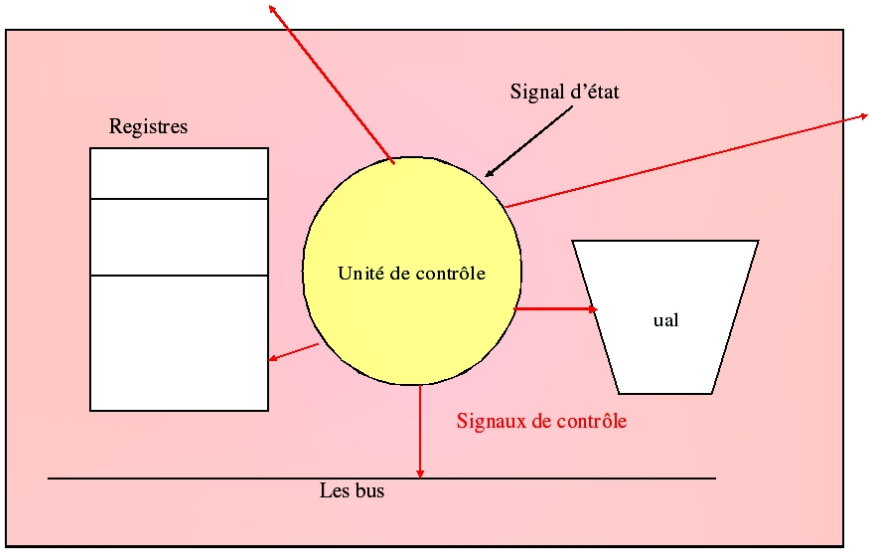
Plusieurs unités au sein d'un processeur :

3 ALU au sein du Pentium

Unité de contrôle/commande (I)

- unité qui coordonne le fonctionnement des autres éléments pour exécuter la séquence d'instructions constituant le programme.
- pour exécuter une instruction, deux cycles se succèdent
  - recherche de l'instruction à exécuter
  - exécution de l'instruction

Unité de contrôle/commande (II)



Unité de contrôle/commande (III)

Constitué :

- d'un ensemble de registres
  - registre d'instruction RI : permet de stocker l'instruction qui doit être exécutée
  - compteur programme PC : stocke l'adresse de la prochaine instruction à exécuter.
  - registre d'états (flag register) : permet de stocker des indicateurs sur l'état du système après l'exécution d'une instruction. par exemple,
    - C (pour carry) : vaudra 1 si une retenue est présente.
    - Z (pour Zero) : vaudra 1 si le résultat de la dernière opération réalisée est nul.
    - V (pour oVerflow) : vaudra 1 en cas de dépassement de capacité
    - N (pour Negative) : vaudra 1 si le résultat est négatif.

    Mis à jour par l'UAL
    - T (Trap flag) : mis à 1 le processeur fonctionne en mode pas à pas
    - IE (Interrupt Enable) : mis à 1 les interruptions sont prise en compte
    - ....

Unité de contrôle/commande (IV)

Constitué :

- d'un ensemble de registres
  - registre d'adresse : contient l'adresse de la donnée à lire ou à écrire en mémoire.
  - registres de données : contient temporairement la donnée lue ou à écrire en mémoire.
  - registre d'index XR (utilisé dans le mode d'adressage indexé) : l'adresse est obtenue en ajoutant son contenu à l'adresse contenue dans l'instruction ; peut être incrémenter/décrémenter automatiquement après son utilisation

    parcours efficace de tableaux
  - registre de base : contient l'adresse (le numéro de segment) à ajouter aux adresses (relatives) contenues dans les instructions.

Unité de contrôle/commande (V)

Constitué :

- **un horloge** qui permet la synchronisation des éléments et des évènements
- **un décodeur** qui détermine les opérations à exécuter en fonction du code de l'instruction.
- **un séquenceur** qui déclenche et coordonne les différentes opérations pour réaliser l'instruction

Cycle d'exécution d'une instruction

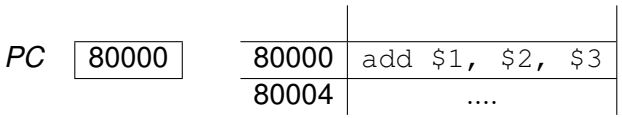
- Cycle de recherche :
  - On récupère dans *RI* l'instruction à exécuter (celle à l'adresse contenue dans *PC*)
  - On incrémente de compteur ordinal *PC*

Plus finement, utilisation des registres d'adresses et de données

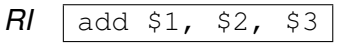
- Cycle d'exécution :
  - On décode l'instruction
  - Lire les adresses et les registres nécessaires à l'instruction
  - Déterminer que faire pour cette instruction
  - Le faire (ou le faire faire) (**utilisation d'une unité de calcul**)

Cycle d'exécution d'une instruction : exemple

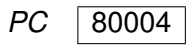
Cycle de recherche



1 On récupère l'instruction à exécuter
   - On met *PC* dans *RA* (le registre d'adresse)
   - On envoie un ordre de lecture à la mémoire
   - On place le contenue de *RD* (le registre de donnée) dans *RI*



2 On incrémente le compteur ordinal *PC*
   - Soit *PC* est muni d'un dispositif d'incrémentation
   - Soit on utilise l'ALU



Cycle d'exécution d'une instruction : exemple (II)

Cycle de recherche

3. Décodage de l'instruction (**Décodeur**)
   - identification d'une addition entre deux registres avec placement du résultat dans un registre
4. Préparation des données (**Séquenceur**)
   - On place les contenus des registres $2 et $3 dans les deux registres d'entrée de l'ALU
5. Déterminer ce qu'il faut faire (**Séquenceur**)
   - Envoi du signal de l'opération d'addition à l'ALU
6. Le faire (**Séquenceur**)
   - L'ALU ajoute les deux opérandes et place le résultat dans son registre de sortie
   - le contenu du registre de sortie de l'ALU est transféré dans le registre $1

Horloge

- définit le cycle de base : cycle machine
- utilisée pour synchroniser chaque étape des cycles de recherche et d'exécution

L'exécution du cycle de recherche ou d'exécution **prend un certain nombre** de cycle de base (dépendant de l'instruction)

Cycle CPU = temps d'exécution minimal d'une instruction (recherche + exécution)

Séquenceur (I)

Séquenceur = machine de Mealy

- recevant des informations du décodeur et des signaux d'états (entrées)
- produisant des signaux de commandes contrôlant les différentes unités

Réalisation :

- séquenceur câblé
- séquenceur micro-programmé

Séquenceur (II)

- Séquenceur câblé :
  - circuit séquentiel (synchrone) réalisé avec des portes logiques
  - Un sous-circuit pour chaque instruction, sous-circuit activé selon le code envoyé par le décodeur.
- Séquenceur micro-programmé :
  - Une ROM contient des micro-programmes composés de micro-instructions
  - Le séquenceur sait exécuter les séquences de micro-instructions

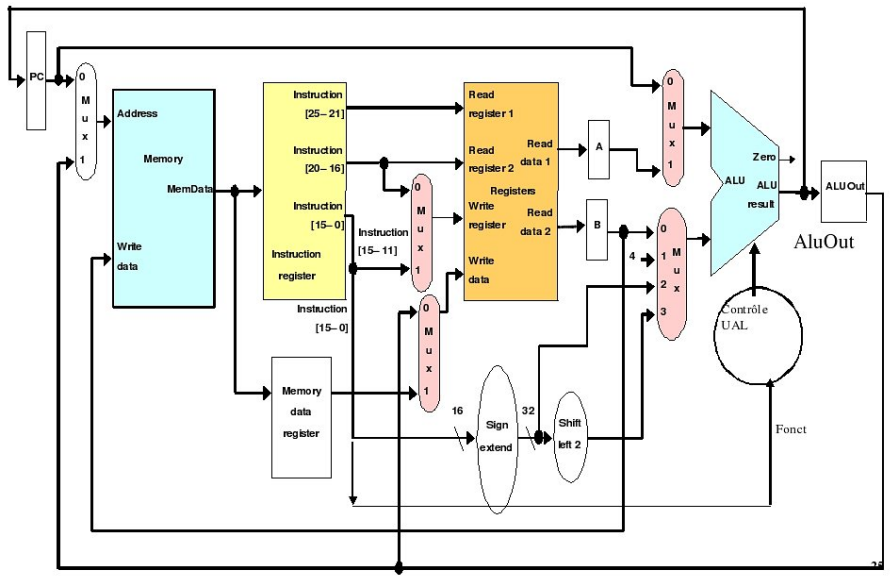
Chemin de données (I)

Un **chemin de données** est défini par

- l'ensemble des composants requis pour l'exécution d'une instruction : PC, UAL, banc de registres, mémoire, ...
- Liens entre ces composants : flux de données, signaux de lecture/écriture, multiplexage des unités partagées, ...

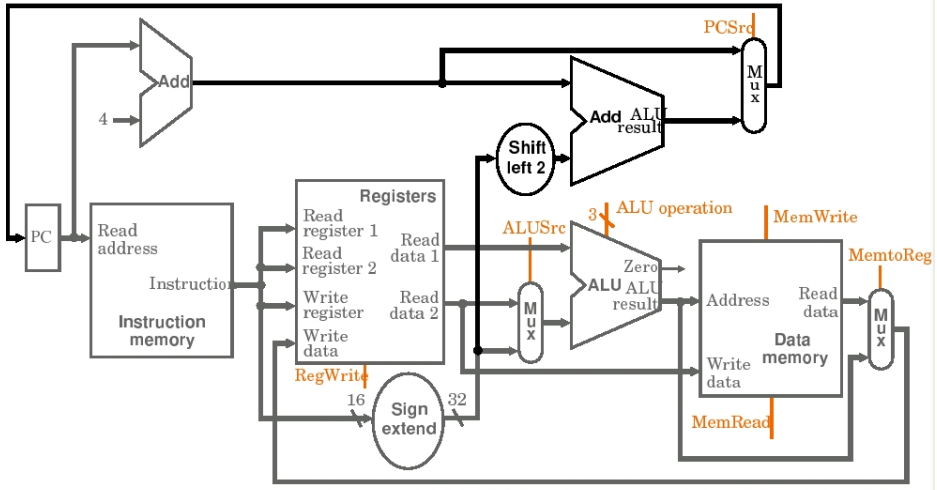
Selon les instructions, les composants requis et les liens existants entre eux varient

Chemin de données (II)

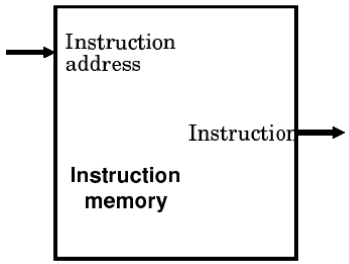


Processeur : fonctionnement - microprogrammation

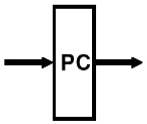
Une architecture d'un processeur MIPS



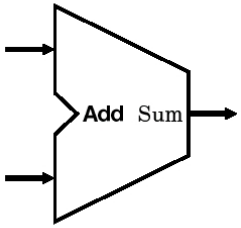
Composants du processeur MIPS (I)



Mémoire d'instructions

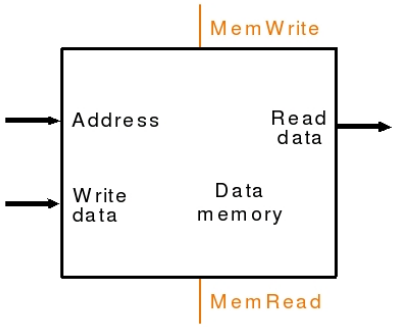


Compteur programme

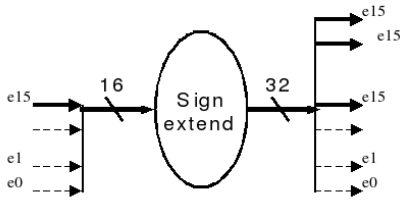


Additionneur

Composants du processeur MIPS (II)

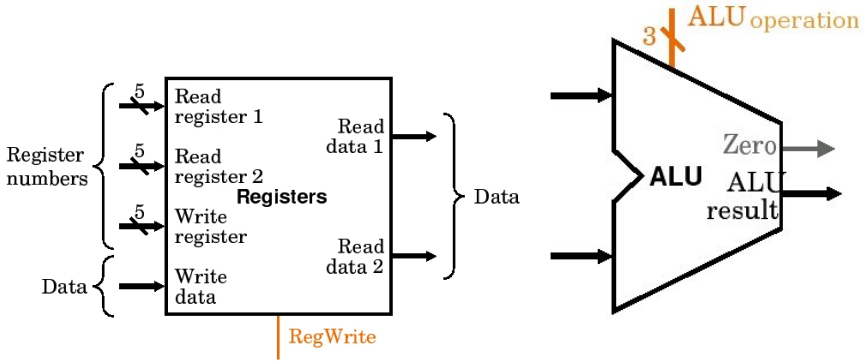


Mémoire de données



Extension signée

Composants du processeur MIPS (III)



Registres

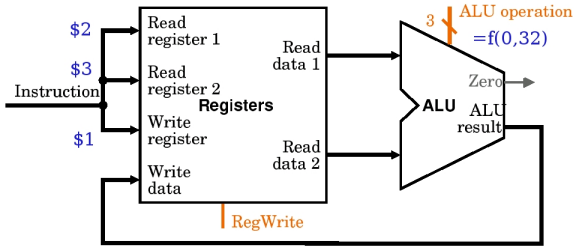
ALU

Format des instructions : rappel

Format	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
Format R	Code op	rs	rt	rd	sa	funct
Format I	Code op	rs	rt	adresse sur 16 bits		
Format J	Code op	adresse sur 26 bits				

Exécution de : `add $1,$2, $3`

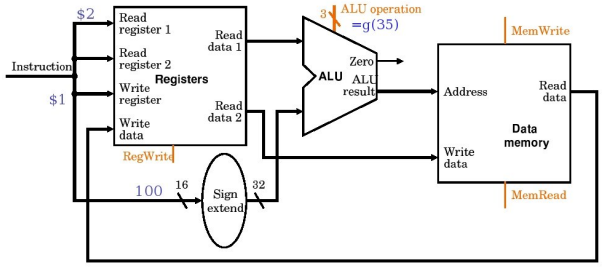
Codeop	rs	rt	rd	sa	funct
0	2	3	1	0	32



- le signal RegWrite contrôle l'écriture dans le banc de registres
- ALUoperation décrit le type de calcul réalisé
- le signal Zero est émis si le calcul vaut 0

Exécution de : `lw $1, 100($2)`

Code op	rs	rt	adresse sur 16 bits
35	2	1	100

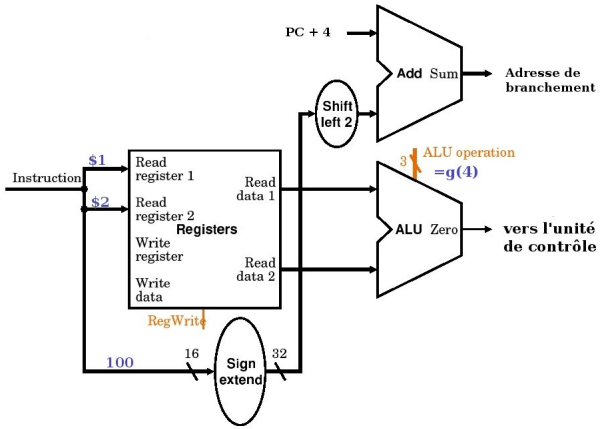


Le signal MemRead est activité.

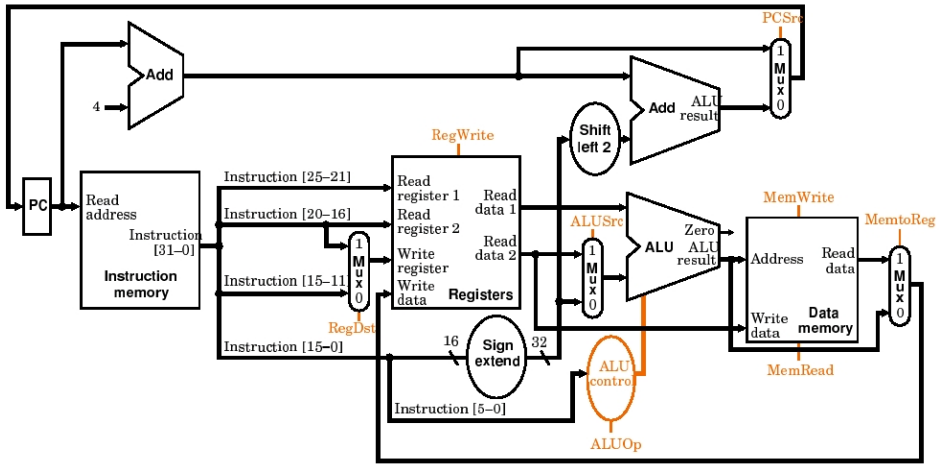
- "adresse 16 bits" est un déplacement relatif signé
- les signaux MemWrite et MemRead contrôlent respectivement l'écriture et la lecture dans la mémoire

Exécution de : `beq $1,$2, 100`

Code op	rs	rt	adresse sur 16 bits
4	2	1	100



Contrôle de l'ALU (I)



Contrôle de l'ALU (II)

Signaux de contrôle (ALUoperation)	Calcul réalisé
000	and
001	or
010	add
110	sub
111	slt

ALUoperation est calculé en fonction

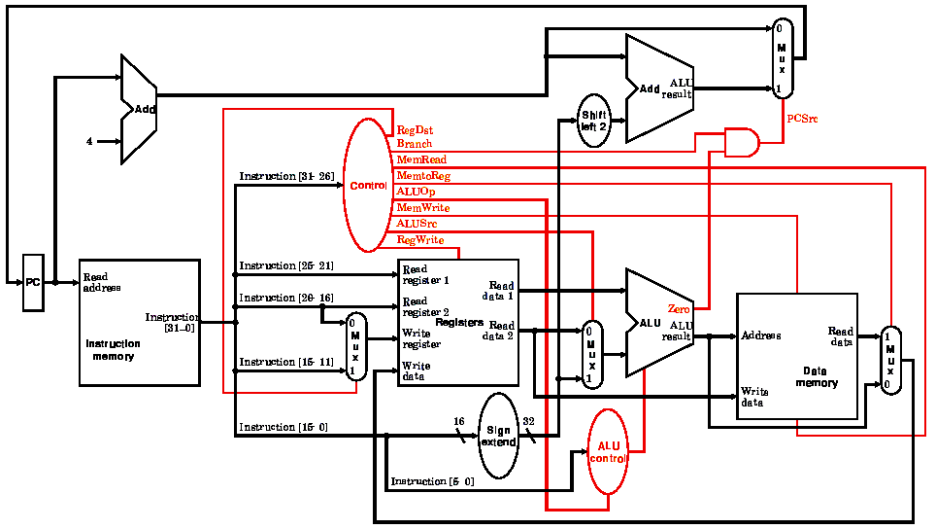
- du champ `funct`, les 6 bits de poids faible de l'instruction exécutée
- du signal ALUop sur 2 bits

Le signal ALUop est calculé en fonction du `Codeop`, les 6 bits de poids fort de l'instruction exécutée

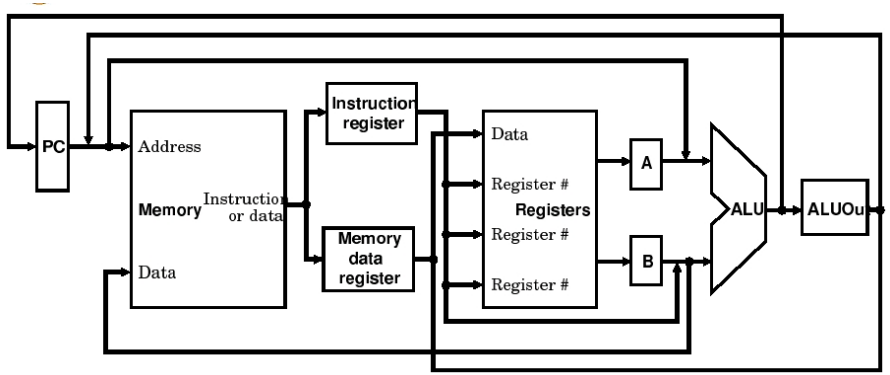
Contrôle de l'ALU (III)

`Codeop`	ALUop	`funct`	ALUoperation
`lw`	00		010
`sw`	00		010
`beq`	01		110
`add`	10	100000	010
`sub`	10	100010	110
`and`	10	100100	000
`or`	10	100101	001
`slt`	10	101010	111

L'unité de contrôle



L'architecture MIPS multi-cycle

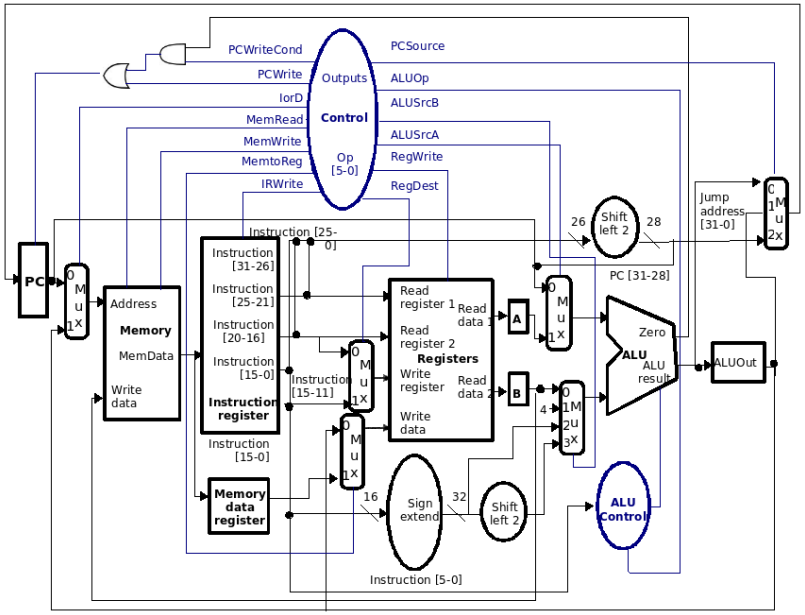


- registre d'instruction RI, registre de données RD
- une seule ALU avec des registres d'entrées A,B et un registre de sortie ALUout

L'architecture MIPS multi-cycle : cycle d'exécution

Etape	Type R	Référence mémoire	Branchements
Extraction	RI ← Mem[PC] et PC ← PC+4		
Décodage	A ← Reg[RI[25-21]] ; B ← Reg[RI[20-16]] ; ALUout ← PC + RI[15-0]*4		
Exécution	ALUout ← A op B	ALUout ← A+RI[15-0] Mem_data ← Mem[ALUout] | Mem[ALUout] ← B	si (A==B) alors PC ← ALUout
Ecriture	Reg[RI[15-11]] ← ALUout	| Reg(RI[15-0]) ← Mem_data	

L'architecture MIPS multi-cycle : unité de contrôle



L'architecture MIPS multi-cycle : signaux 1 bit

Signal	Effet pour S=0	Effet pour S=1
RegDest	le registre de destination pour l'écriture est RI[20 :16]	le registre de destination pour l'écriture est RI[15 :11]
RegWrite		la donnée en écriture est stockée dans le registre à écrire
ALUSrcA	la 1ere opérande de l'ALU est PC	la 1ere opérande de l'ALU est A
MemRead		une donnée est lue en mémoire à l'adresse spécifiée
MemWrite		une donnée à écrire est écrite en mémoire à l'adresse spécifiée
MemtoReg	la donnée à écrire dans le registre provient du registre donnée mémoire	la donnée à écrire dans le registre provient de ALUout
IorD	l'adresse pour la mémoire est fournie par PC	l'adresse pour la mémoire provient de ALUout
IRWrite		La sortie de la mémoire est écrite dans RI
PCWrite		une valeur est écrite dans PC
PCWriteCond		PC est modifié si le valeur Zero de l'ALU vaut 1

L'architecture MIPS multi-cycle : signaux 2 bits

- ALUSrcB : la seconde entrée de l'ALU est
  - 00 : la valeur du registre B
  - 01 : la valeur 4
  - 10 : l'extension signée 16 bits de la valeur immédiate dans RI
  - 11 : l'extension signée 16 bits de la valeur immédiate dans RI décalée de 2
- PCSource : le PC sera écrit avec
  - 00 : la sortie de l'ALU (PC+4)
  - 01 : la valeur de ALUout
  - 10 : l'adresse de saut décalée de 2 et ajoutée à PC+4[31 :28]

Conception d'une unité de contrôle

L'unité de contrôle est un système synchrone

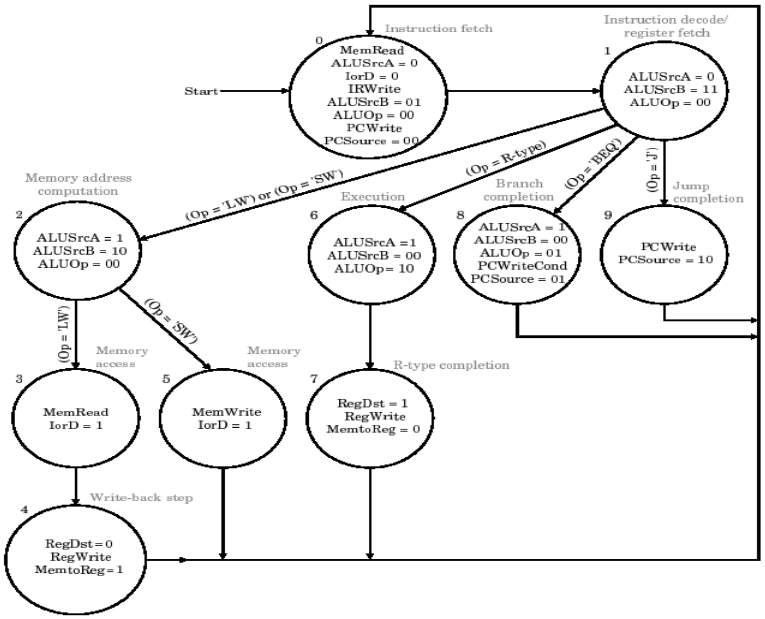
Le décodeur identifie l'instruction.

Le séquenceur envoie les signaux de contrôle orchestrant les différents éléments du chemin de données pour réaliser l'instruction.

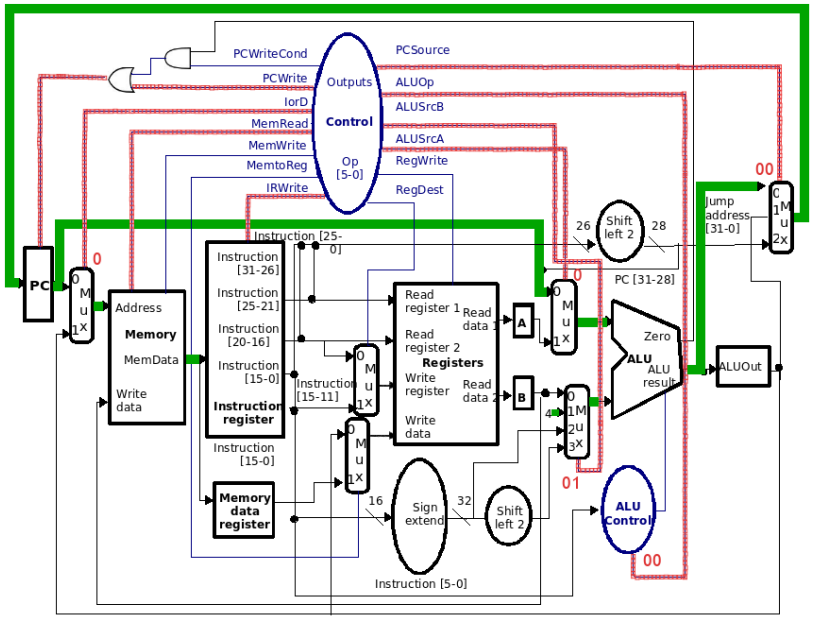
Séquenceur = machine de Moore

- entrées : informations du décodeur + informations des éléments du chemin de données
- sorties : signaux de contrôle

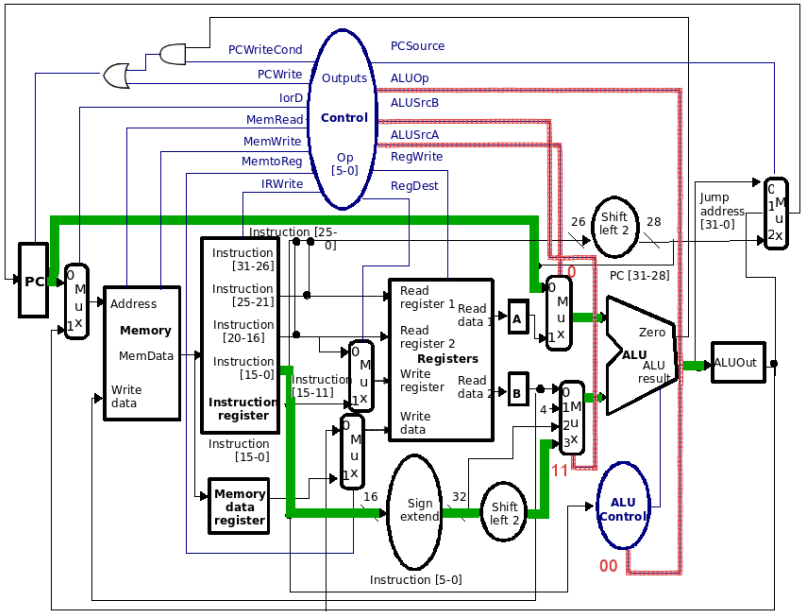
Modélisation d'un séquenceur



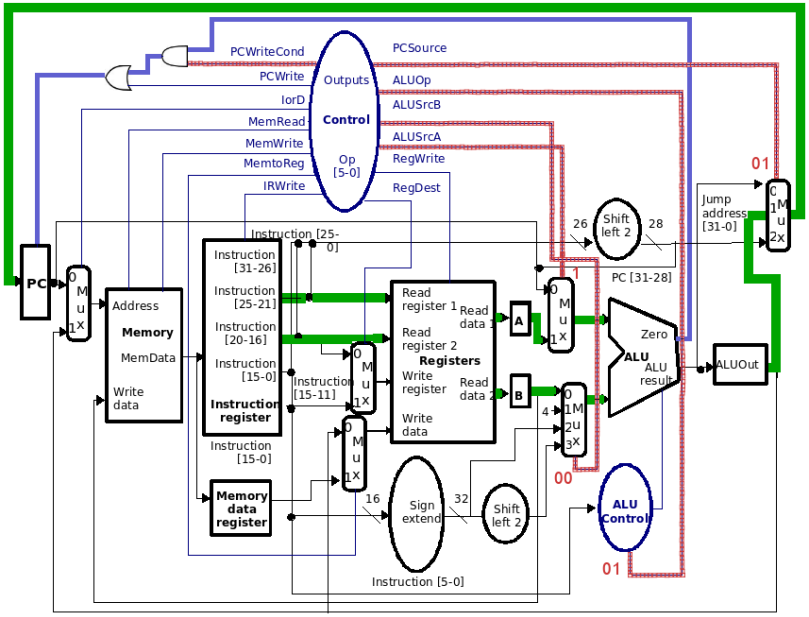
Chemin de données : état 0



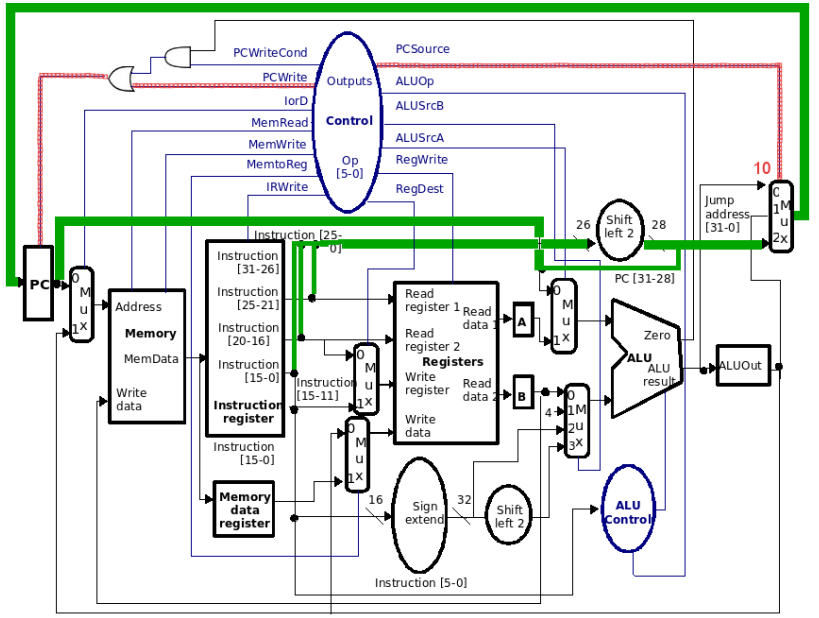
Chemin de données : état 1



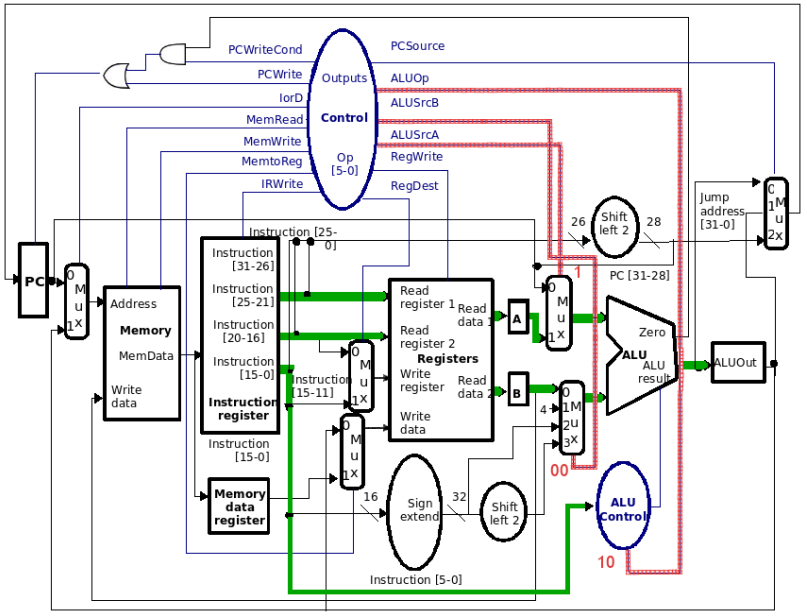
Chemin de données : état 8 (branch. cond. : `beq`)



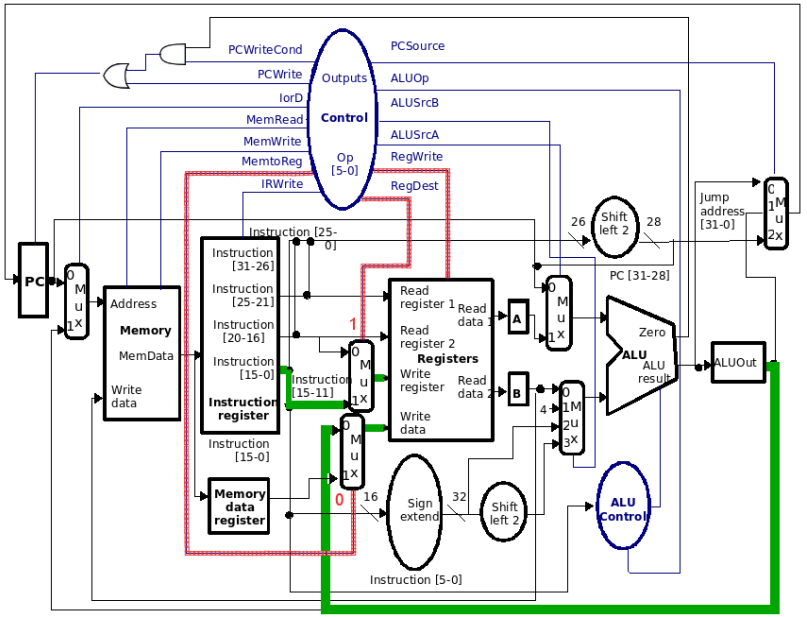
Chemin de données : état 9 (branch. incond. : `j`)



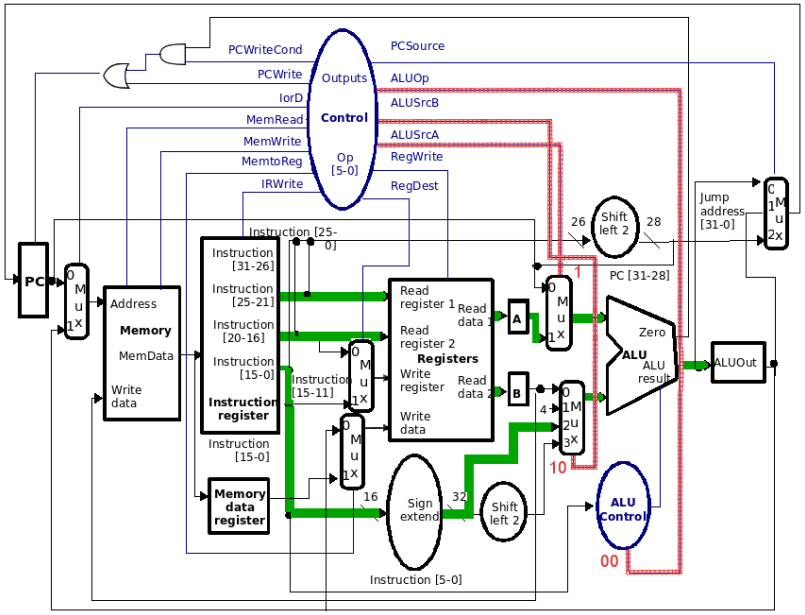
Chemin de données : état 6 (type R : partie I)



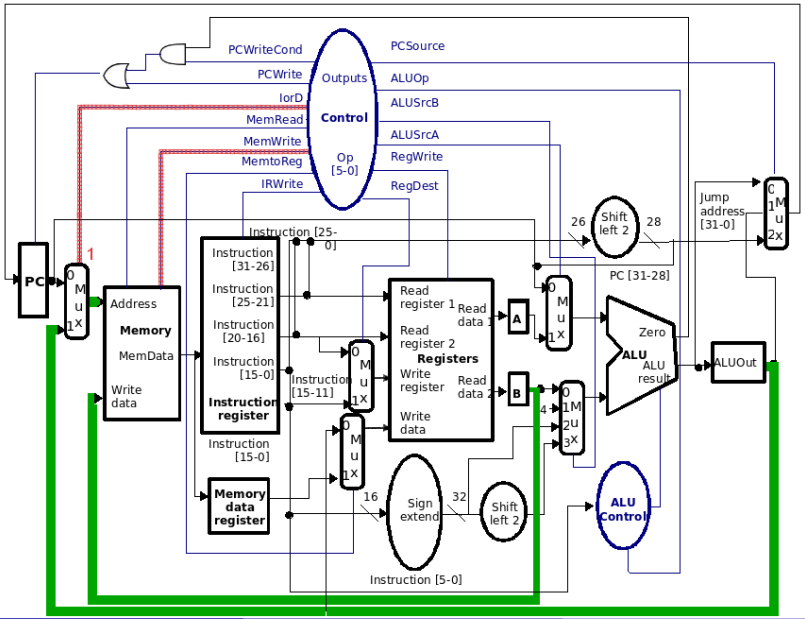
Chemin de données : état 7 (type R : partie II)



Chemin de données : état 2 (sw : partie I)



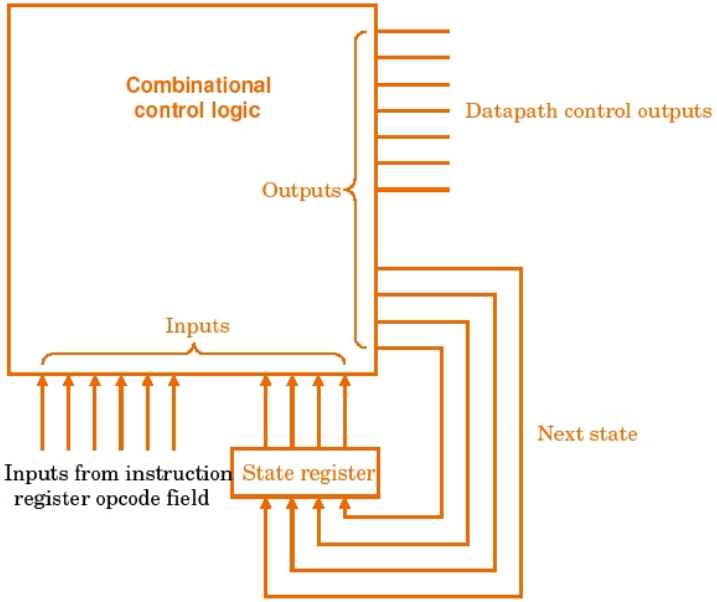
Chemin de données : état 5 (sw : partie II)



Réalisation d'un séquenceur

- solution cablée : l'unité de contrôle est une implantation cablée de l'automate
  - privilégiée pour les processeurs RISC
- solution microprogrammée : un microprogramme contenu dans une ROM réalise l'automate
  - privilégiée pour les processeurs CISC

Séquenceur : solution cablée



Séquenceur : solution microprogrammée

Concevoir la réalisation de l'automate comme un programme qui implante les instructions du langage machine.

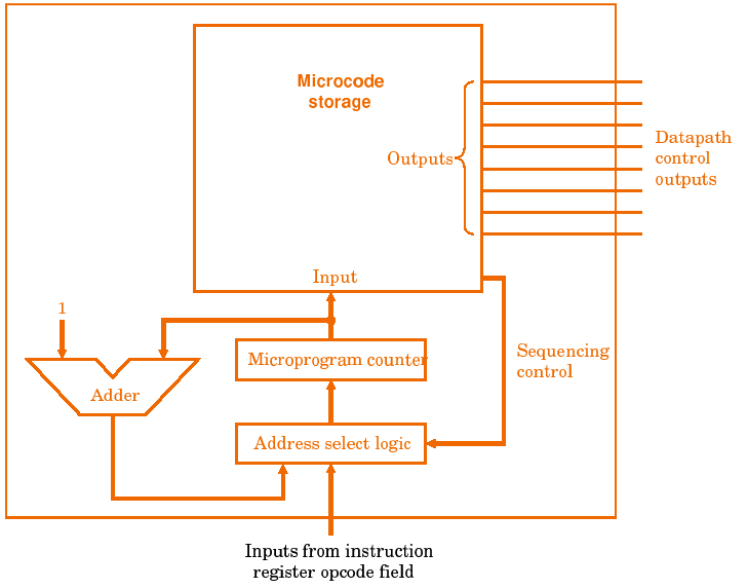
Un **microprogramme** pour une instruction du langage machine est une représentation du contrôle nécessaire à l'exécution de l'instruction et est constitué d'une suite de **microinstructions**.

Chaque microinstruction pilote un ensemble de signaux de contrôle du chemin de données.

2 types de microprogrammation :

- horizontale
- verticale

Contrôleur de microcode



Microprogrammation horizontale

Dans chaque microinstruction, il y a

- une partie "contrôle" : un bit est dédié à chacune des signaux de contrôle
- une partie "instruction suivante" : détermine la microinstruction suivante à exécuter parmi
  - la microinstruction suivante dans la mémoire
  - la première microinstruction de l'instruction suivante
  - un branchement selon des signaux d'états
- Peu de microinstructions nécessaires à la réalisation d'une instruction
- Les microinstructions peuvent très longues (une centaines de bits)

Microprogrammation horizontale MIPS

2 bits	1 bit	2 bits	3 bits	4 bits	4 bits	2 bits
Contrôle ALU	SRC1	SRC2	Contrôle registre	Mémoire	Contrôle écriture PC	suivante

Champs	Signaux
Contrôle ALU	ALUop
SRC1	ALUSrcA
SRC2	ALUSrcB
Contrôle registre	RegWrite,MemtoReg,RegDst
Mémoire	MemRead, MemWrite,IorD,ORWrite
Contrôle écriture PC	PCWrite,PCWriteCond,PCsource

Microprogrammation verticale

Les microinstructions sont de petite taille et se décomposent en 2 parties :

Codeop	données

Les données prennent un sens selon le Codeop

Jeu d'instructions sophistiqué

- émission de signaux (nécessitant un décodage)
- saut
- branchement conditionnel
- ...

Microprogrammation verticale MIPS

```
00 :  send        0000
01 :  send        0001
02 :  if j          10
03 :  if beq        12
04 :  if R-type     14
05 :  send        0010
06 :  if sw         17
07 :  send        0011
08 :  send        0100
09 :  goto          00
10 :  send        1001
11 :  goto          00
12 :  send        1000
13 :  goto          00
14 :  send        0110
15 :  send        0111
16 :  goto          00
17 :  send        0101
18 :  goto          00
```

En supposant un codage des signaux à émettre pour chacun des états : 10 codes différents $\Rightarrow$ 4 bits

Interruptions - exceptions (I)

- Les **exceptions** sont des évènements anormaux provenant (de l'impossibilité) de l'exécution de l'instruction en cours.
  - **ADEL**/**ADES** : adresse incorrecte respectivement en lecture et en écriture - adresse non alignée ou dans une zone d'accès interdite.
  - **OVF** : overflow - la dernière opération réalisée ne produit pas un résulat représentable sur 32 bits
  - **RI** : Codeop illégal - la mémoire à l'adresse PC ne contient pas une instruction.
  - ...
- Les **interruptions** sont des évènements déclenchés
  - par des périphériques : interruption **matérielle**
  - par le programme lui-même : interruption **logicielle**

Interruptions - exceptions (II)

En MIPS

- le registre CR `Cause Register` contient en cas d'interruption ou d'exception, la cause pour laquelle on fait appel au programme de traitement des interruptions/exceptions.
- le registre EPC `Exception Programm Counter` contient
  - l'adresse de retour (PC + 4) en cas d'interruption
  - l'adresse de l'instruction fautive en cas d'exception

Interruptions - exceptions (III)

- En cas d'exception, la procédure stockée à l'adresse `0x80000080` (le gestionnaire d'exception) est exécutée.

  Les exceptions sont **fatales** (à l'exécution du programme)

  CR et EPC sont uniquement utilisés pour identifier l'exception

- En cas d'interruptions, la procédure stockée à l'adresse `0x80000080` (le gestionnaire d'exception) est exécutée.

  CR est utilisé pour identifier l'exception

  EPC est utilisé comme adresse de retour après le traitement de l'exception.