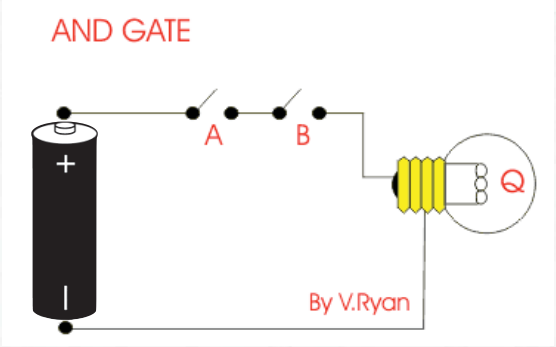
L'ordinateur à coeur ouvert

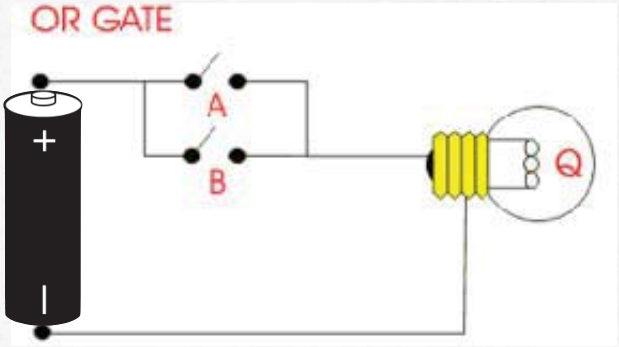
Andres Perez-Uribe
HEIG-VD / HES-SO

Des interrupteurs pour faire des fonctions logiques

A='0'

A='1'





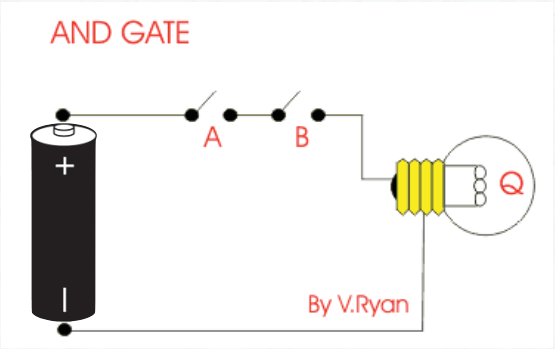
Ampoule allumée si A='1' ET B='1'

Ampoule allumée si A='1' OU B='1'

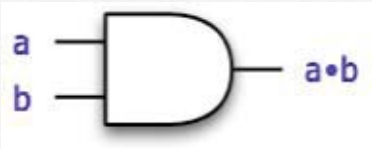
La fonction logique ET

A=’0’

A=’1’



a	b	a•b
0	0	0
0	1	0
1	0	0
1	1	1

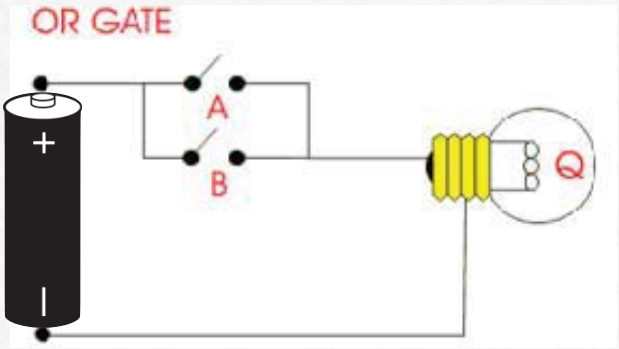


Ampoule allumée si A=’1’ ET B=’1’

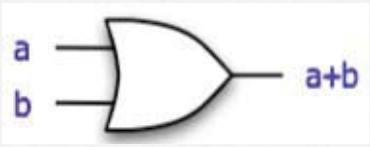
La fonction logique OU

A=’0’

A=’1’

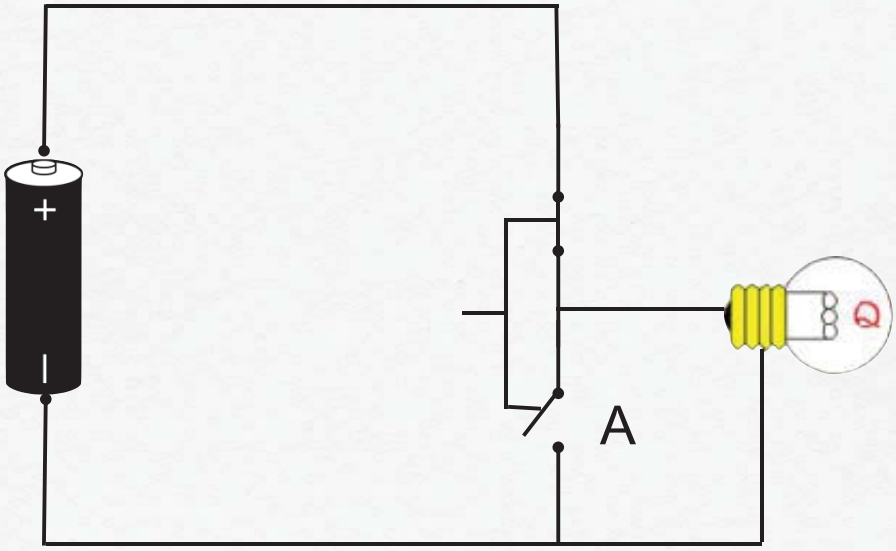


a	b	a+b
0	0	0
0	1	1
1	0	1
1	1	1

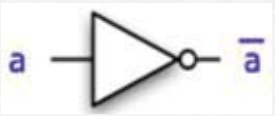


Ampoule allumée si A=’1’ OU B=’1’

La fonction logique NON

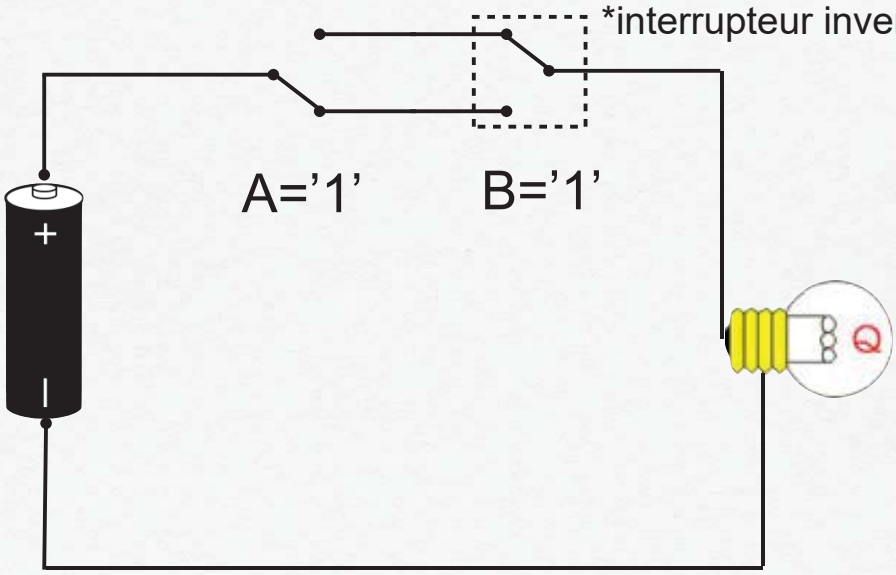


a	$\bar{a}$
0	1
1	0

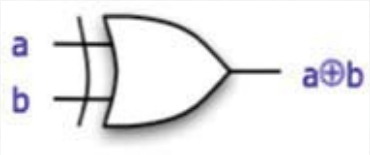


Ampoule allumée si A='0'

La fonction logique OU-exclusif



a	b	a⊕b
0	0	0
0	1	1
1	0	1
1	1	0

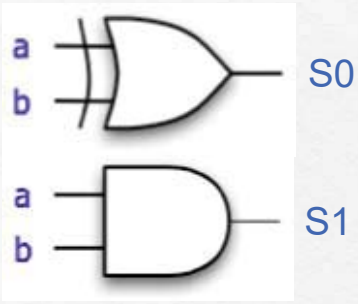


Un additionneur de deux bits

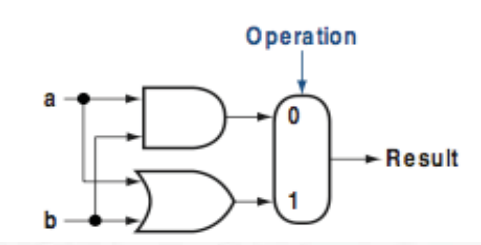
0 + 0 = 0 0
0 + 1 = 0 1
1 + 0 = 0 1
1 + 1 = 1 0

a	b	a•b
0	0	0
0	1	0
1	0	0
1	1	1

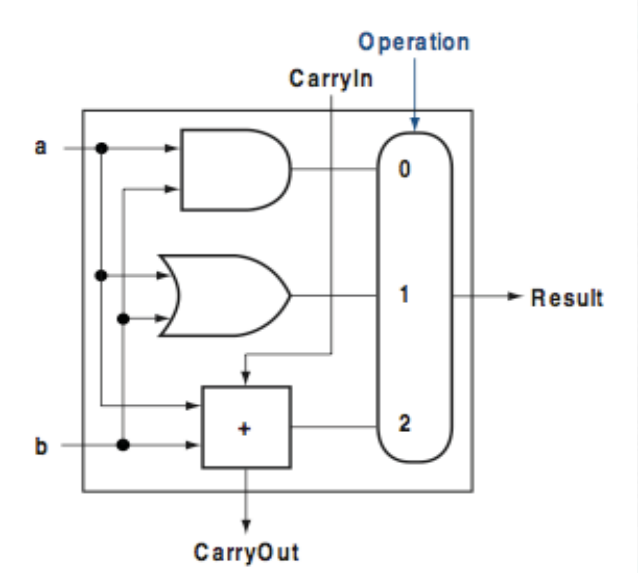
a	b	a⊕b
0	0	0
0	1	1
1	0	1
1	1	0



Vers une ALU “simple” 32-bit (1)

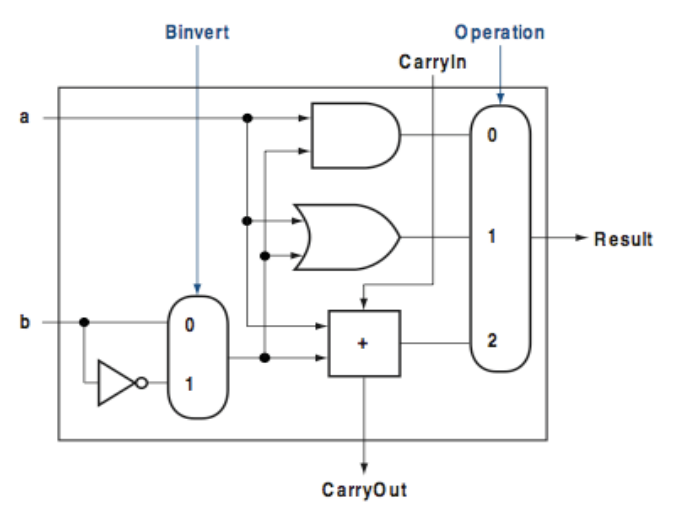


1-bit ALU: AND et OR

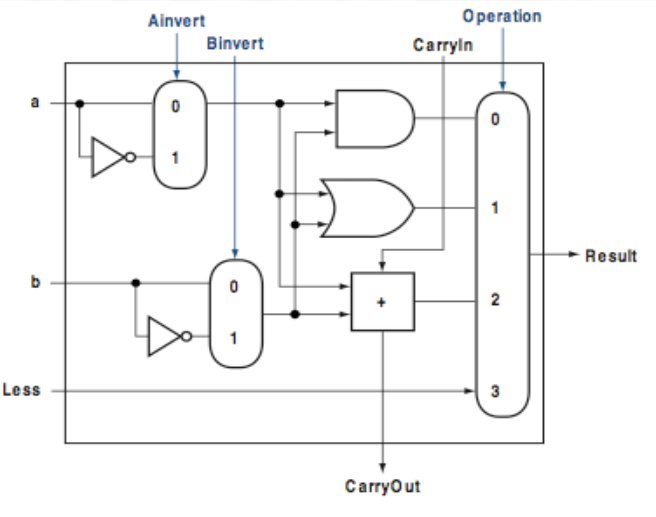


1-bit ALU: AND, OR et ADD

Vers une ALU “simple” 32-bit (2)

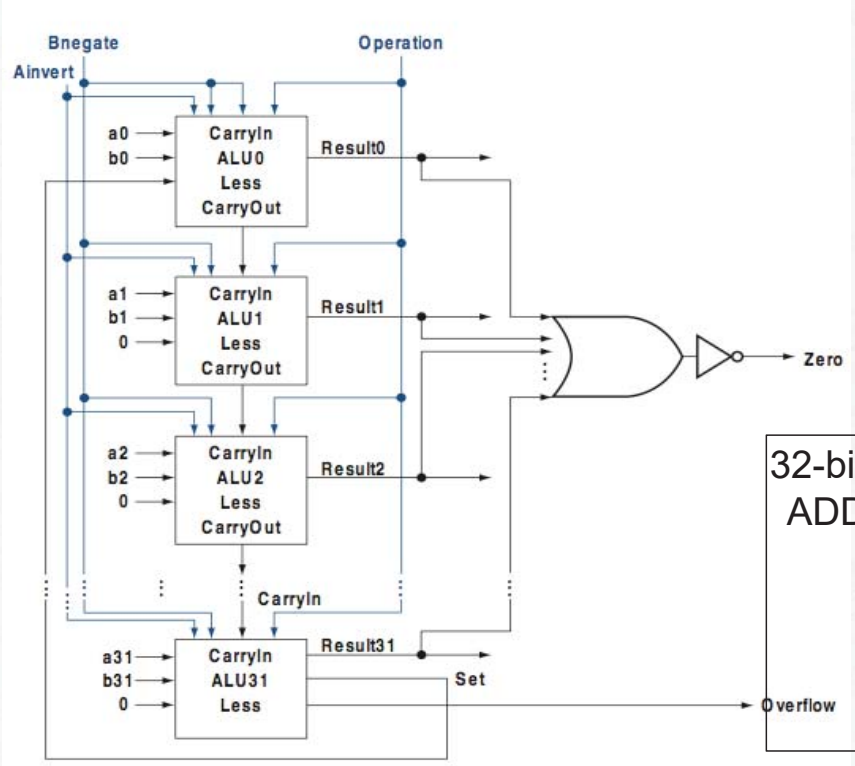


1-bit ALU: AND, OR, ADD et SUB



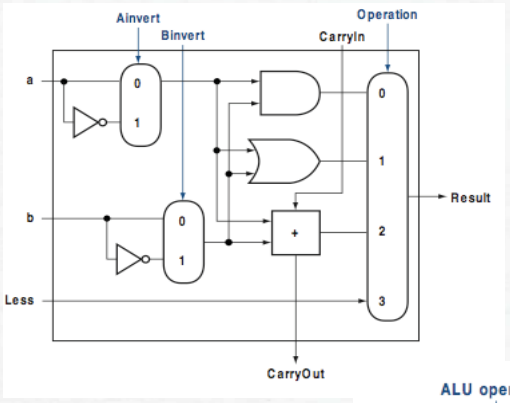
1-bit ALU: AND, OR, NOR, NAND, ADD, SUB

Vers une ALU “simple” 32-bit (3)

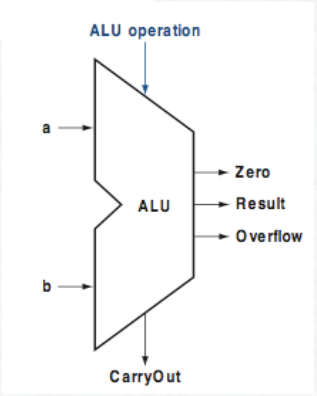


32-bit ALU: AND, OR, NAND, NOR, ADD avec calcul d’overflow, SUB, détecteur de zéro et SLT
SLT (set on less than)
si a < b alors sortie = 1
sinon sortie = 0

Une ALU simple 32-bit



1-bit ALU

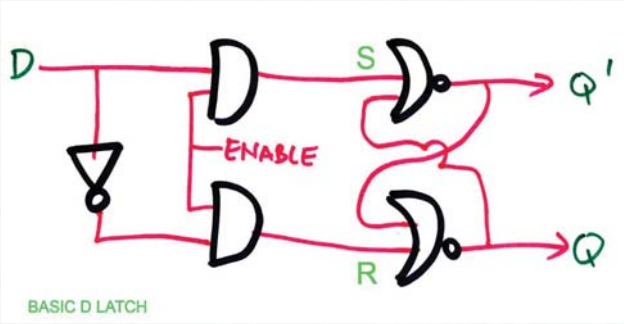


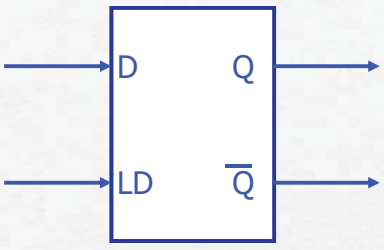
32-bit ALU

ALU op	Fonction
00000	AND
00001	OR
00010	ADD
01110	SUB
01111	SLT
11000	NOR
11001	NAND

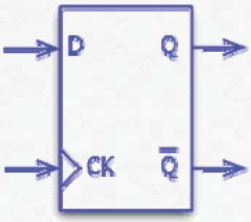
Ainv Binv Cin Op(mux)

Composants mémoire

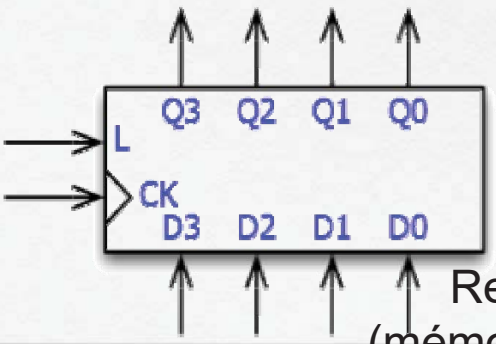




D-latch



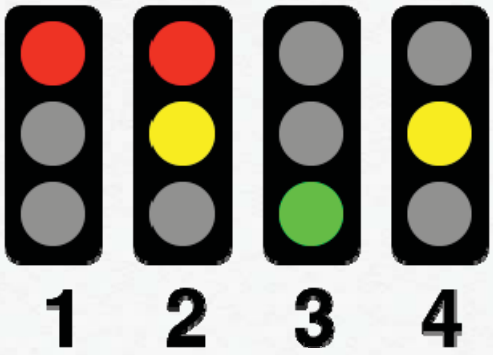
Flip-flop
(mémoire 1bit)

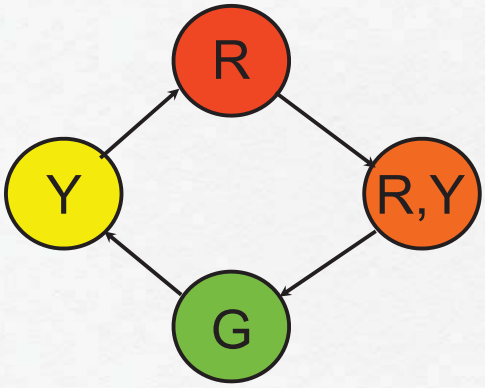


Registre
(mémoire n-bits)

Systèmes séquentiels

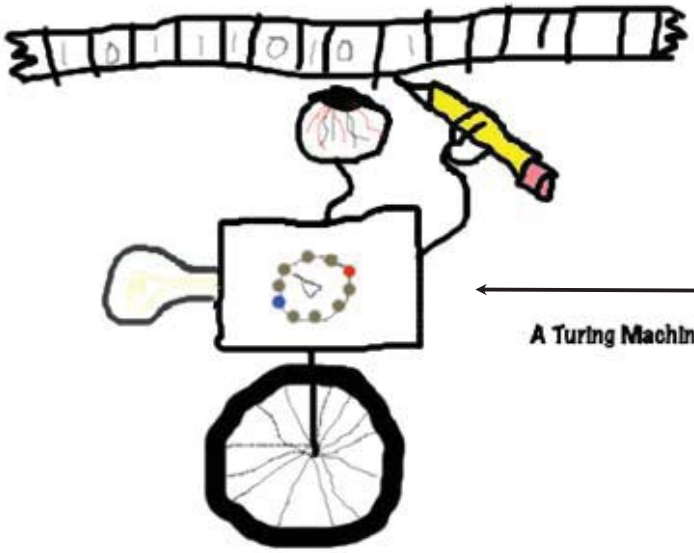
Flip-flops + fonctions logiques





Machine d'états finis

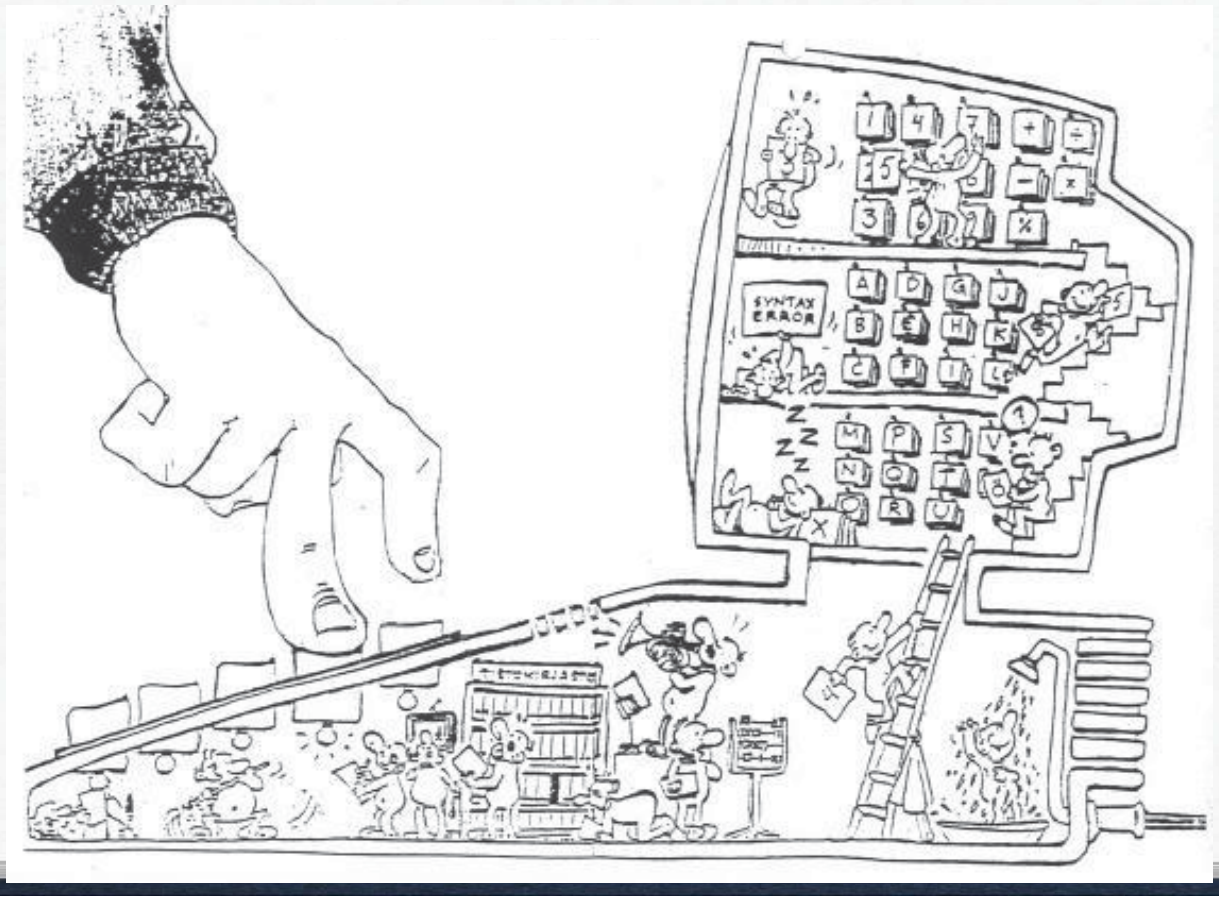
Machine de Turing



mémoire

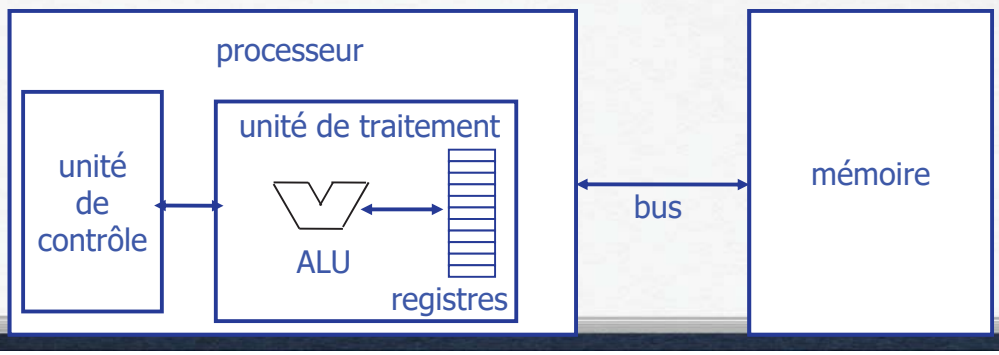
système
séquentiel

Peter Boothe <peter@cs.uoregon.edu>



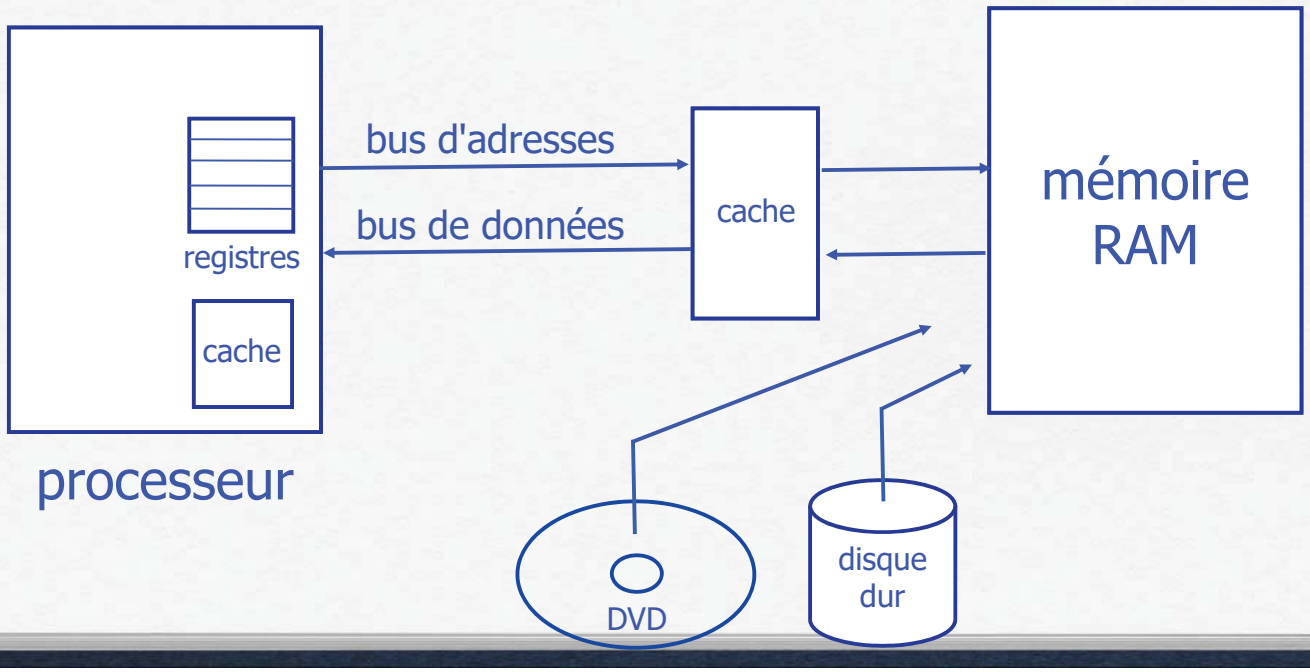
Architecture de von Neumann

- Le processeur est composé de deux parties: l'unité de traitement, qui contient l'ensemble d'opérateurs arithmétiques et logiques, groupés autour d'une ou plusieurs ALUs (Arithmetic and Logic Unit); et l'unité de contrôle qui coordonne les différentes activités du processeur
- Les données à traiter et les instructions sont stockées dans la même mémoire

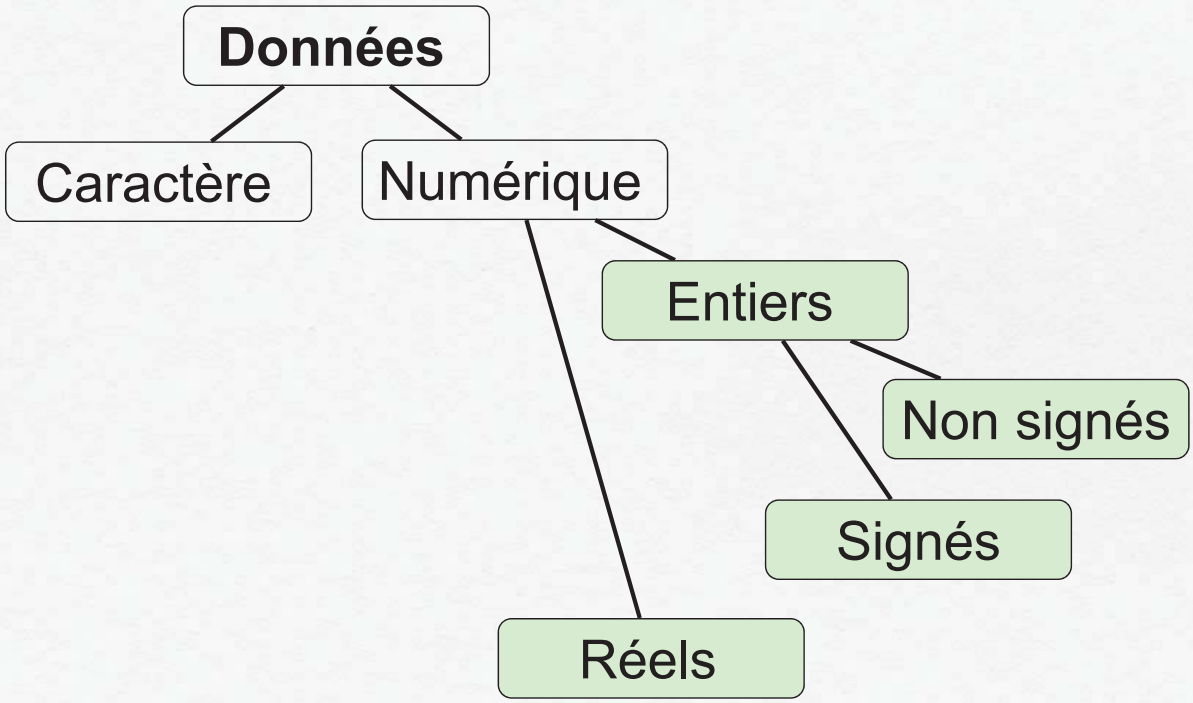


- Pour lire une donnée ou une instruction, le processeur doit envoyer une adresse à la mémoire par le biais du bus d'adresses
- La mémoire répond en envoyant l'information demandée par le biais du bus de données

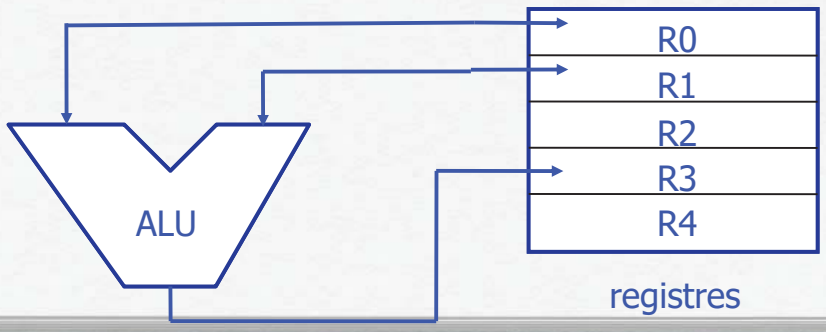
Hiérarchie des mémoires



Représentation des données



- Le traitement des données est effectué à l'intérieur du processeur par l'ALU, qui peut faire des opérations arithmétiques et logiques de base entre deux opérandes (addition, soustraction, et, ou, décalage, etc)
- Les deux opérandes de l'ALU et son résultat sont stockés dans les registres internes du processeur



- L'accès au registres est plus rapide qu'à la mémoire. Mais les registres sont peu nombreux (par exemple, 32 dans le cas de l'architecture MIPS)
- Comme les données se trouvent initialement dans la mémoire, il faut une instruction pour les ramener dans les registres: c'est l'instruction **LOAD**
- L'instruction **LOAD** utilise deux paramètres: l'adresse de la mémoire que l'on veut lire et le numéro du registre
- Exemple: si la donnée **toto** se trouve à la position 20 de la mémoire et nous voulons la stocker dans le registre 4 du processeur, il faut exécuter l'instruction

```
LOAD R4, 20
```

- Exemple: si nous voulons additionner les variables **toto** et **titi**, qui se trouvent dans les positions (adresses) 20 et 21 de la mémoire, il faut d'abord les ramener dans les registres internes du processeur
- Les instructions:

```
LOAD R4, 20
LOAD R5, 21
```

ramènent **toto** et **titi** dans les registres 4 et 5 du processeur, respectivement

- Une fois dans les registres internes, le processeur peut donner l'ordre à l'ALU d'effectuer l'addition: c'est l'instruction **ADD**
- L'instruction **ADD** utilise trois paramètres: les deux registres source et le registre destination, où le résultat de l'addition sera sauvegardé
- Par exemple, l'instruction

```
ADD R0, R4, R5
```

réalise l'addition de **R4** avec **R5** et stocke le résultat dans **R0**

- Si l'opération voulue était

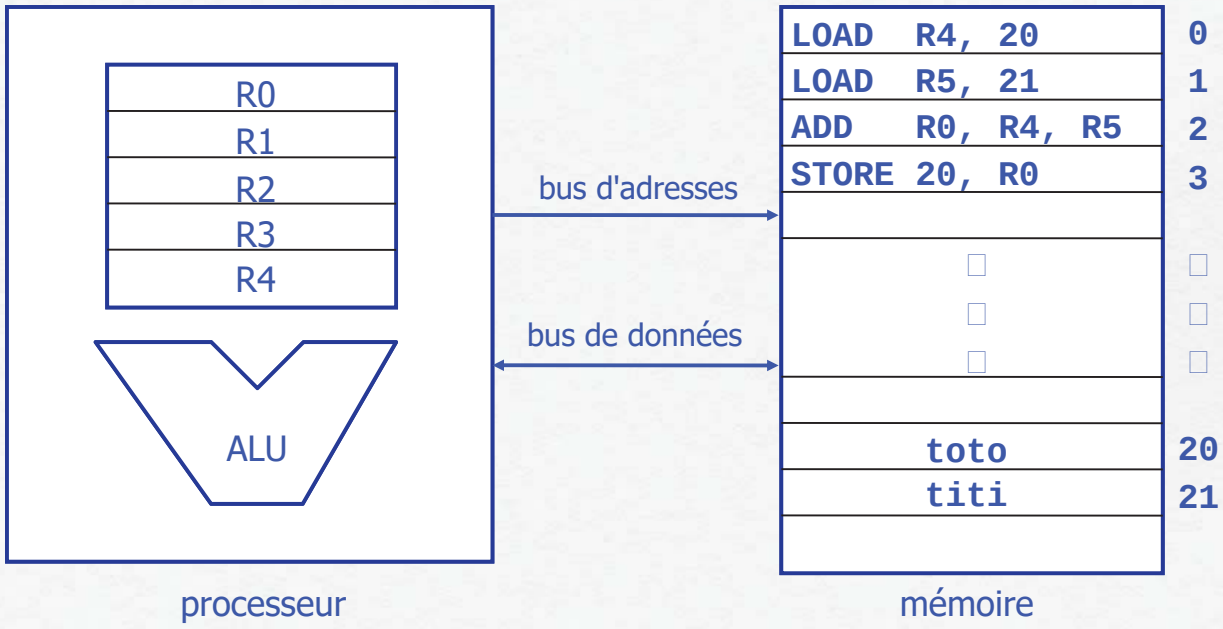
```
toto = toto + titi
```

il faut écrire le contenu du registre **R0** (le résultat de l'addition) dans la position 20 de la mémoire

- L'instruction **STORE** modifie une position de mémoire avec le contenu d'un registre
- Notre programme complet est maintenant:

```
LOAD  R4, 20
LOAD  R5, 21
ADD   R0, R4, R5
STORE 20, R0
```

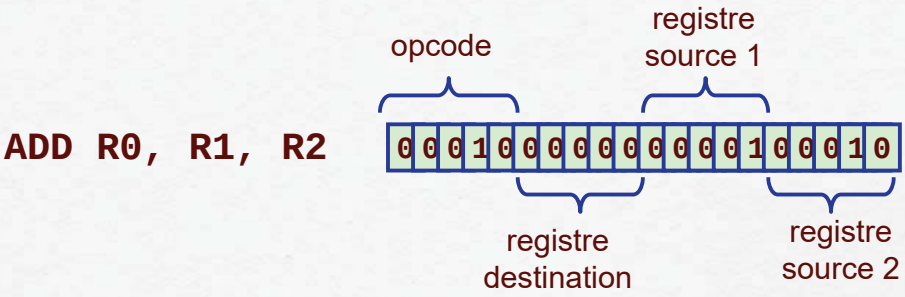
- Ce programme doit être stocké dans la mémoire de l'ordinateur



Jeu d'instructions

- Chaque processeur à un jeu d'instructions, qui comporte l'ensemble d'opérations que le processeur peut exécuter.
- Chaque instruction est identifiée avec un code, appellé opcode (e.g., 00010, 00011) et avec un mnémonique (e.g., ADD, SUB).
- Une suite d'instructions d'un processeur donne lieu à un programme en langage machine

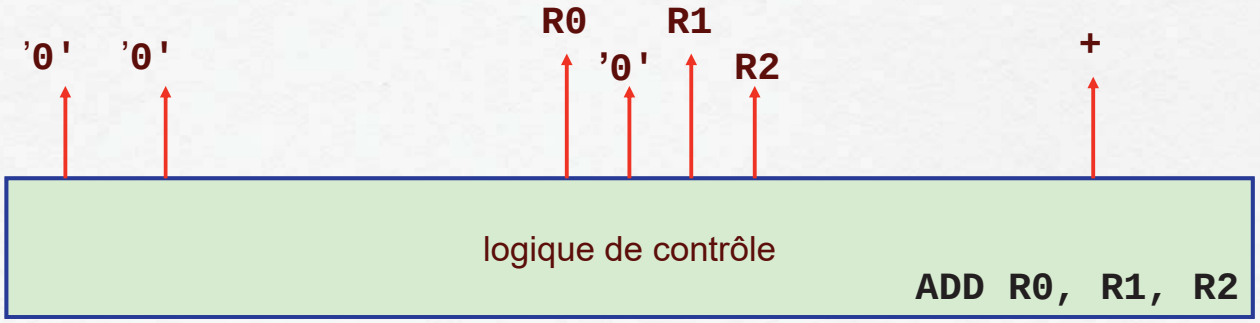
Langage machine



Une instruction en langage machine indique l'opération à réaliser ainsi que les registres (ou les adresses mémoire) à utiliser comme sources et comme destination.

Décodage d'instructions

Le décodeur d'instruction reconnaît l'opcode et génère les signaux de contrôle nécessaires pour exécuter l'opération

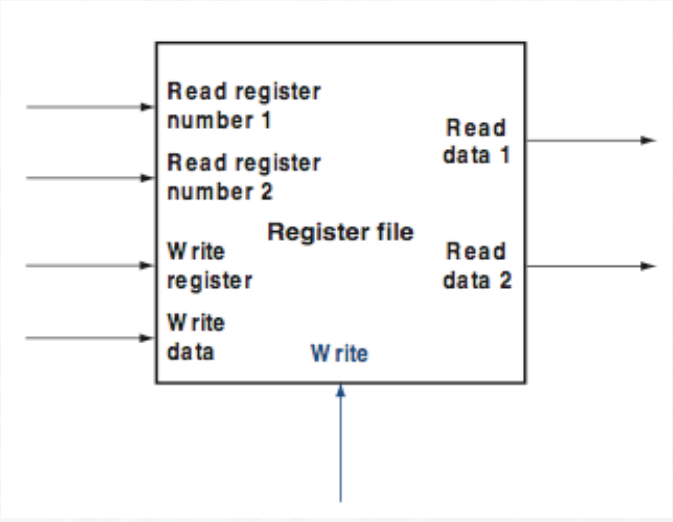


Les registres du processeur

ADD R0, R1, R2

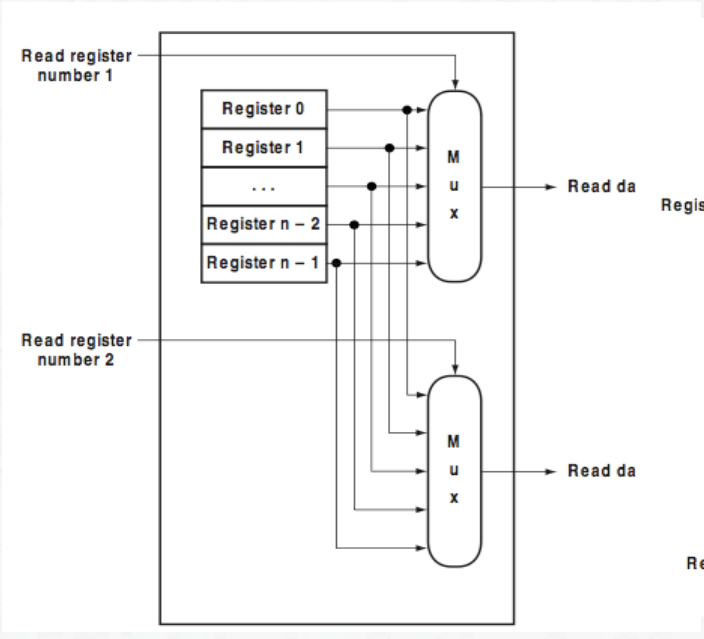
registre destination

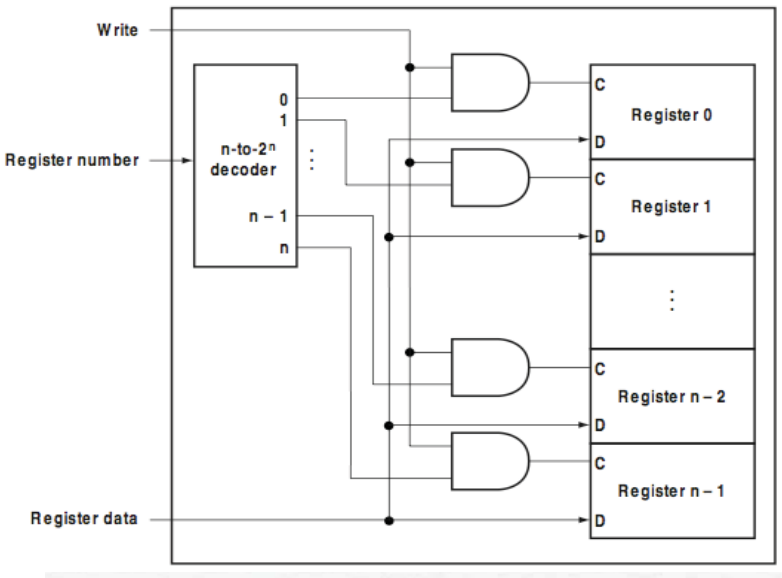
registres source



2-ports en lecture / 1-port en écriture

Implémentation des registres





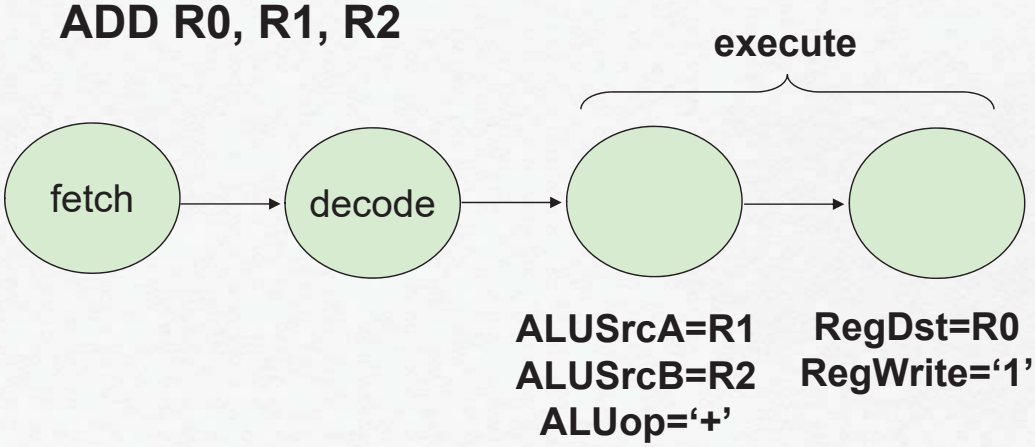
lecture **écriture**

L’unité de contrôle d’un processeur

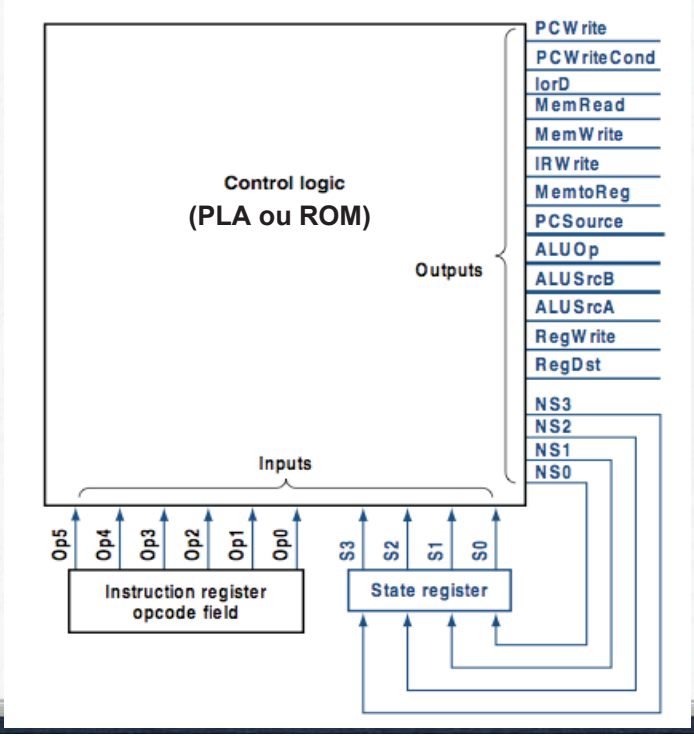
- L’unité de contrôle doit gérer l’exécution des opérations de base du processeur (c’est à dire du jeu d’instructions).
- Chaque instruction du processeur est exécutée en plusieurs phases, prenant plusieurs cycles d’horloge.
- Chaque instruction est d’abord cherchée dans la mémoire de l’ordinateur (fetch), après elle est décodée (decode) et enfin exécutée (execute).
- La phase execute, peut prendre plusieurs cycles d’horloge (e.g., la multiplication séquentielle).

Exemple d’exécution d’une instruction

ADD R0, R1, R2

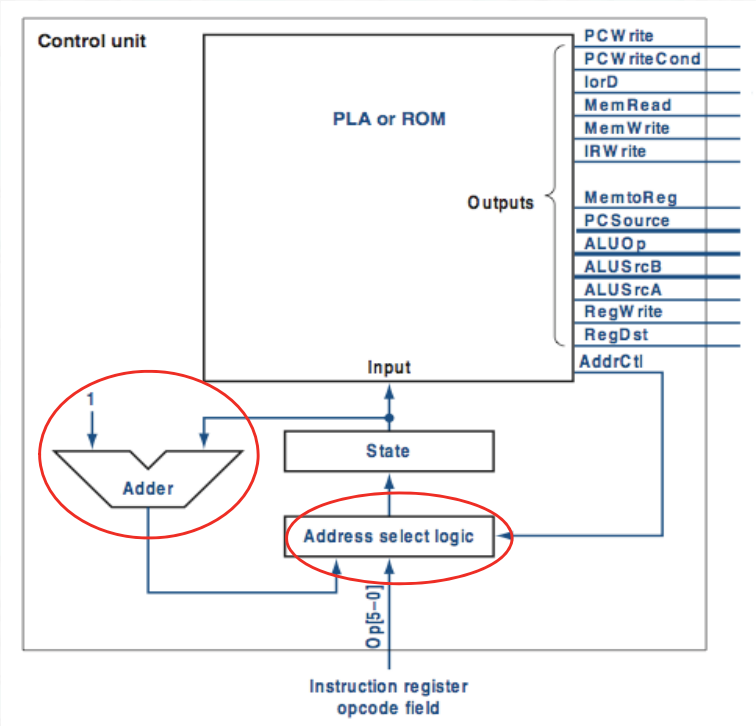


Unité de contrôle réalisée à l'aide d'une machine séquentielle



Unité de contrôle réalisée à l'aide d'un séquenceur

L'utilisation d'un compteur et d'un circuit incrémenteur simplifie la conception et modification de l'unité de contrôle.



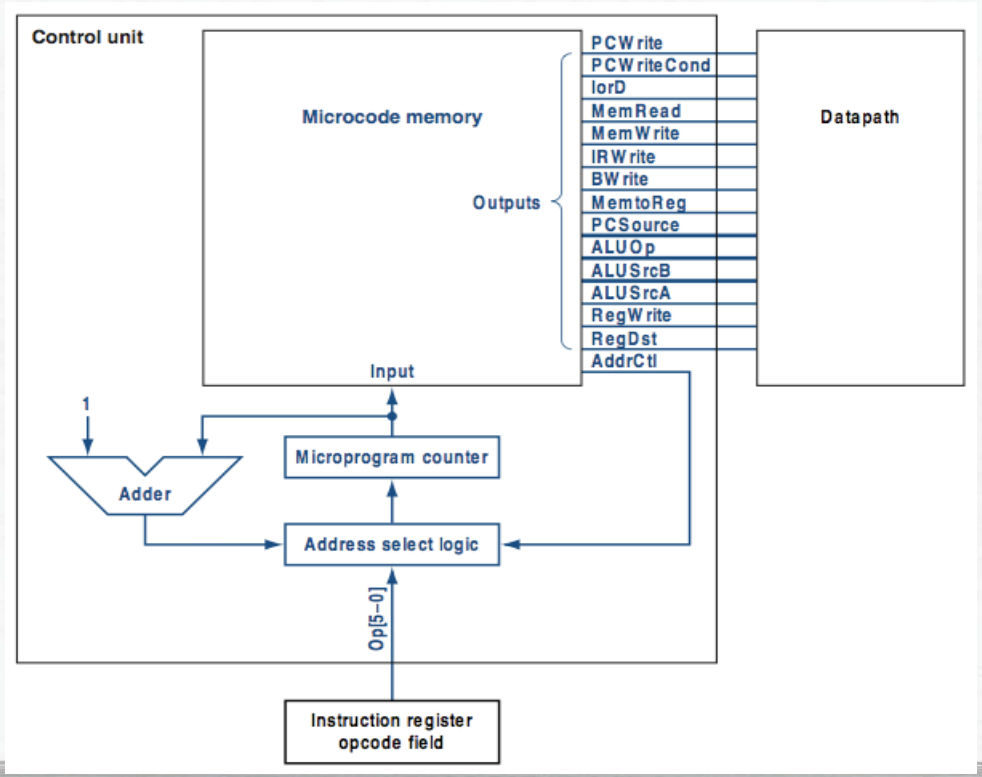
Unité de contrôle microprogrammée

- Les différents pas nécessaires pour exécuter une instruction peuvent être vus comme des microinstructions.

- Chaque microinstruction fournit les valeurs (bits) des signaux de contrôle nécessaires pour gérer les composants de l'unité de traitement du processeur.

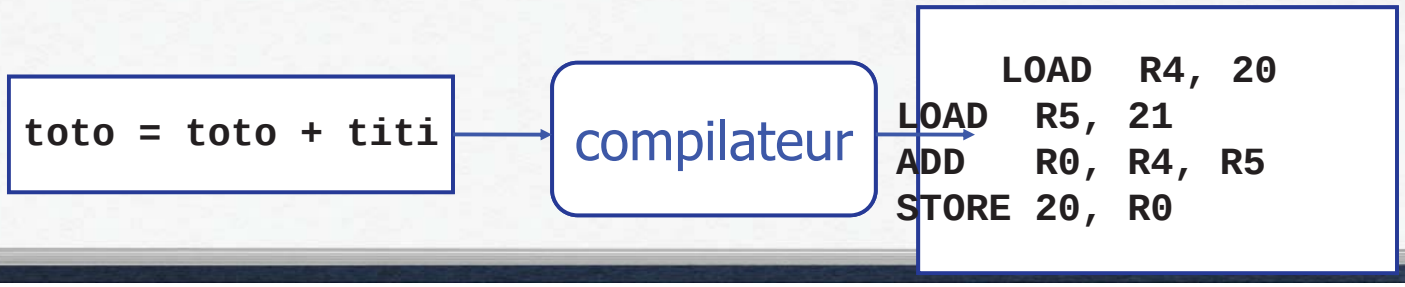
- L'unité de contrôle devient donc un microprogramme, une vision abstraite du hardware qui nous donne pas mal de flexibilité lors de la conception.

Unité de contrôle microprogrammée



Programmation en langage de haut niveau

- Les programmes sont normalement écrits utilisant des langages appelés de haut niveau (C, Java, C++, Ada, Basic, etc), où il est fait abstraction des détails internes du processeur
- Ensuite, le programme est traduit par un autre programme, le compilateur, dans le programme correspondant en langage machine

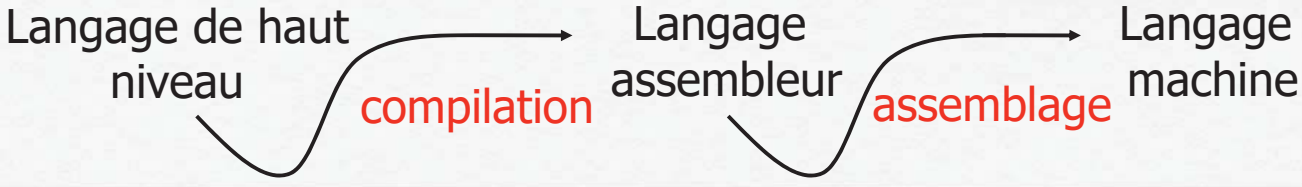


Compilation et assemblage

```
var
   a,b,c : Integer;
begin
   a:=5;
   b:=8;
   c := a + b;
end.
```

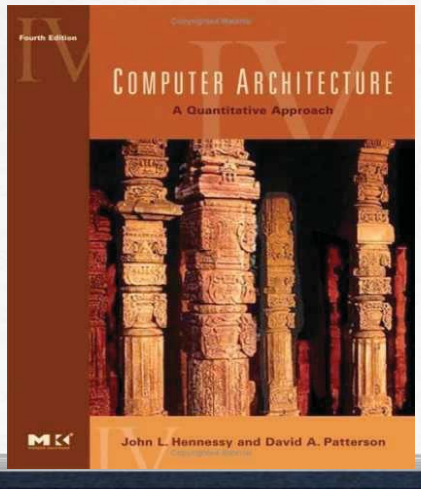
```
load  $4, 50($0)
load  $5, 54($0)
add   $2, $4, $5
store $2, 58($0)
```

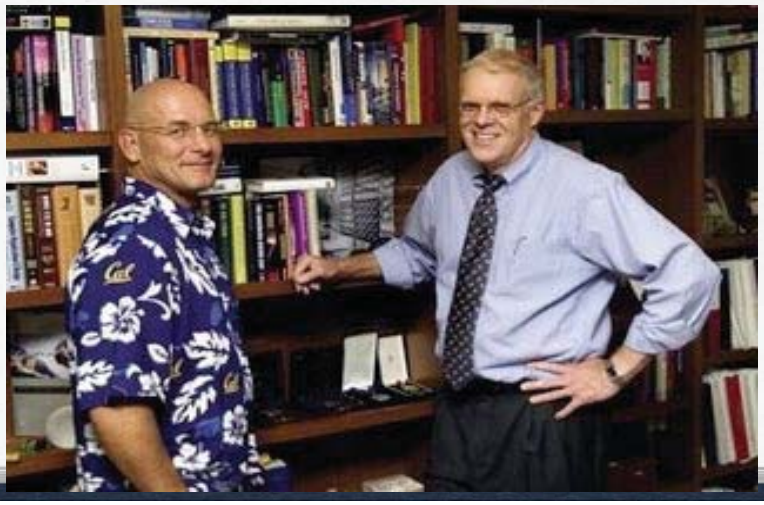
```
1010 1111 1011 1111
0100 0000 0011 0010
1010 1111 1011 1111
0101 0000 0011 0110
0010 0111 1011 1101
1111 1111 1110 0000
1010 0000 1011 1111
0010 0000 0011 1010
```



Références

- John Hennessy & David A. Patterson, “Computer Architecture”, Appendixes B et C
- “The pattern on the stone”, Daniel Hillis
- Cours de systèmes logiques d’Eduardo Sanchez





A ajouter

- Niveau de cache
- L1,L2,L3,L4