ITI 1521. Introduction à l'informatique II

Concepts essentiels de **l'architecture des ordinateurs**

by

Marcel Turcotte

Version du 5 janvier 2020

Préambule

Préambule

Aperçu

Aperçu

Concepts essentiels de l'architecture des ordinateurs

Nous passons en revue les concepts essentiels de l'architecture des ordinateurs : modèle de von von Neumann, mémoire, et compilation. Nous simulons l'exécution d'un programme en langage machine à l'aide d'un modèle didactique d'un microprocesseur.

Objectif général :

- Cette semaine, vous serez en mesure de décrire en vos propres mots l'exécution du programme machine.

Vidéo d'introduction :

- `https://www.youtube.com/watch?v=WjhGiQD3kD0`

Préambule

Objectifs d'apprentissage

Objectifs d'apprentissage

- **Expliquer** dans vos propres mots les concepts suivants : mémoire, compilation, et variable.
- **Simuler** l'exécution d'un simple programme machine.

Lectures :

- -

Préambule

Plan du module

Plan

1 Préambule

2 Introduction

Introduction

Prérequis

Vous devez maîtriser ces concepts :

- **Types de données prédéfinis** et les **tableaux**
- **Structures de contrôles** :
  telles que `if`, `for`, `while`... ;
- **Abstraction procédural** :
  c'.-à.-d. comment décomposer un problème en sous-problèmes.

Pourquoi différents langages de programmation ?

Introduction

Architecture des ordinateurs

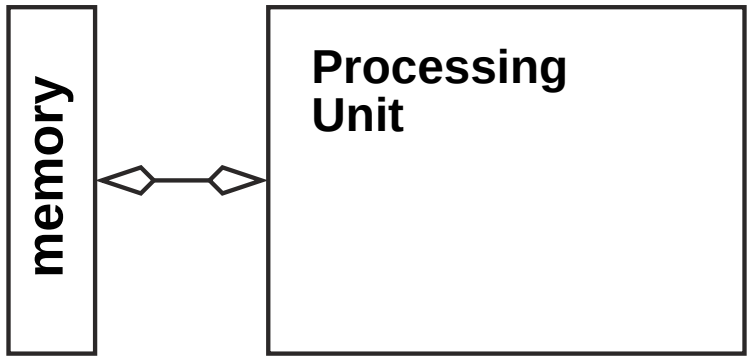
Modèle de von Neumann

L'**architecture** des ordinateurs modernes suit le modèle proposé par (John) **von Neumann** (1945).

mémoire : contient les **instructions** et les **données**

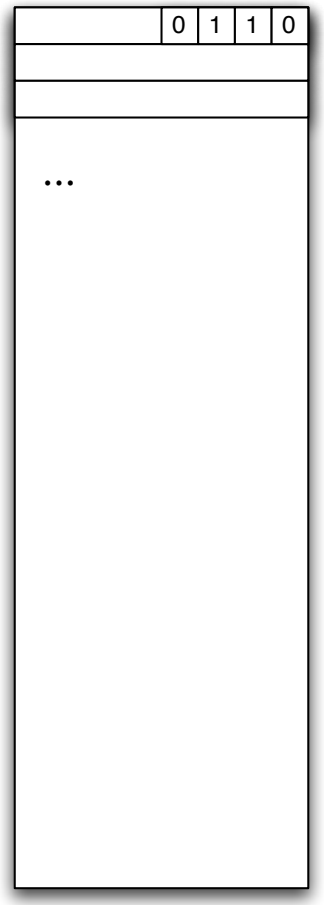
uct : l'unité de traitement effectue les **opérations arithmétiques** et **logiques**

uc : l'unité de contrôle **interprète les instructions**



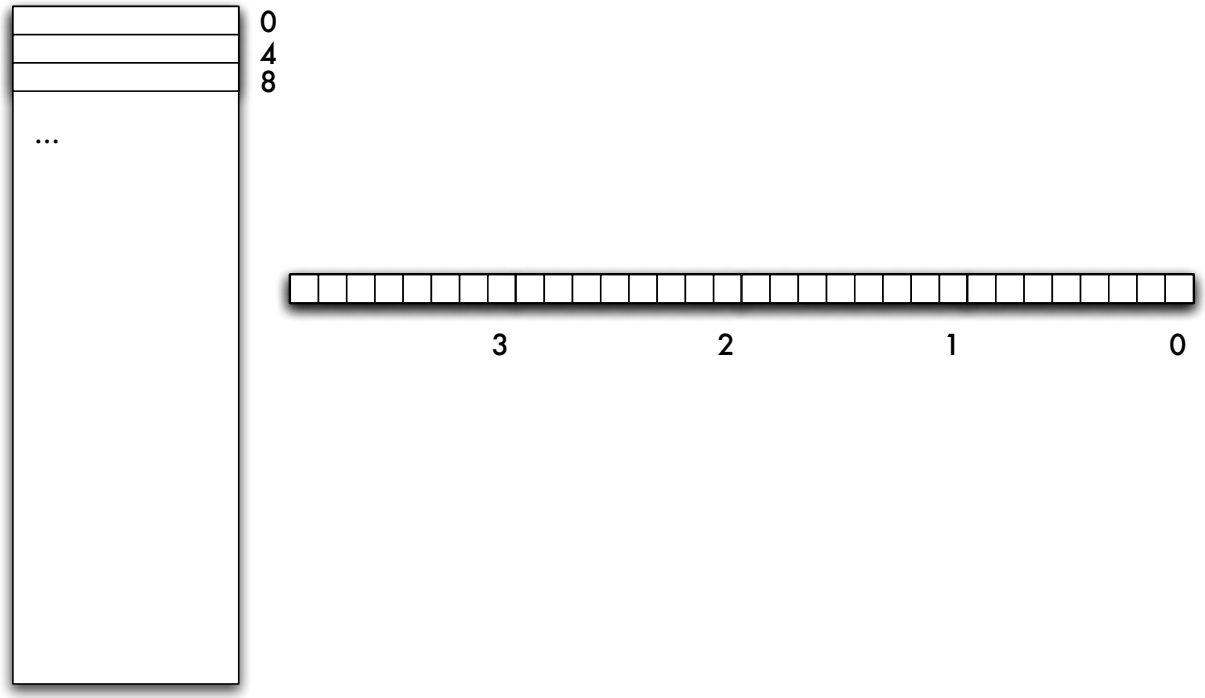
Modèle de la mémoire

- Peut-être vue comme un **immense tableau**,
  où chaque cellule du tableau peut contenir zéro ou un
  (*binary digit* — bits) ;



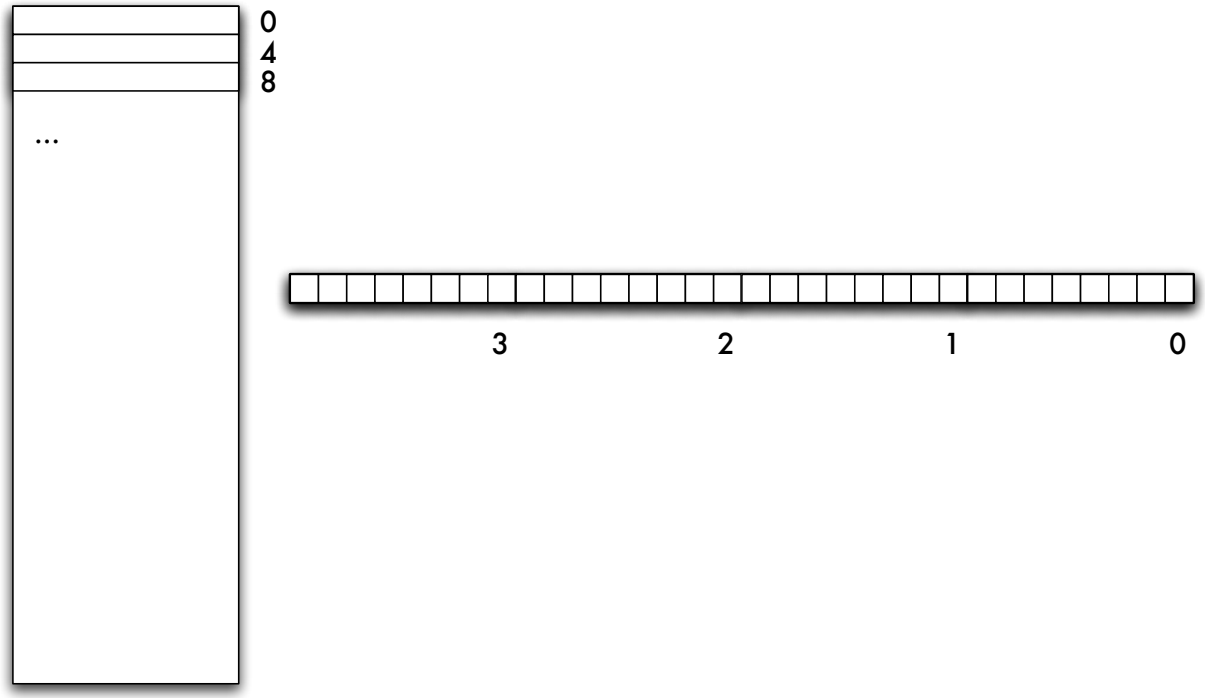
Modèle de la mémoire

- Chaque **octet** (groupement de 8 bits) possède une **adresse unique** (distincte)
- Les octets sont regroupés en **mots**
- **Certains types de données nécessitent plus d'un octet**



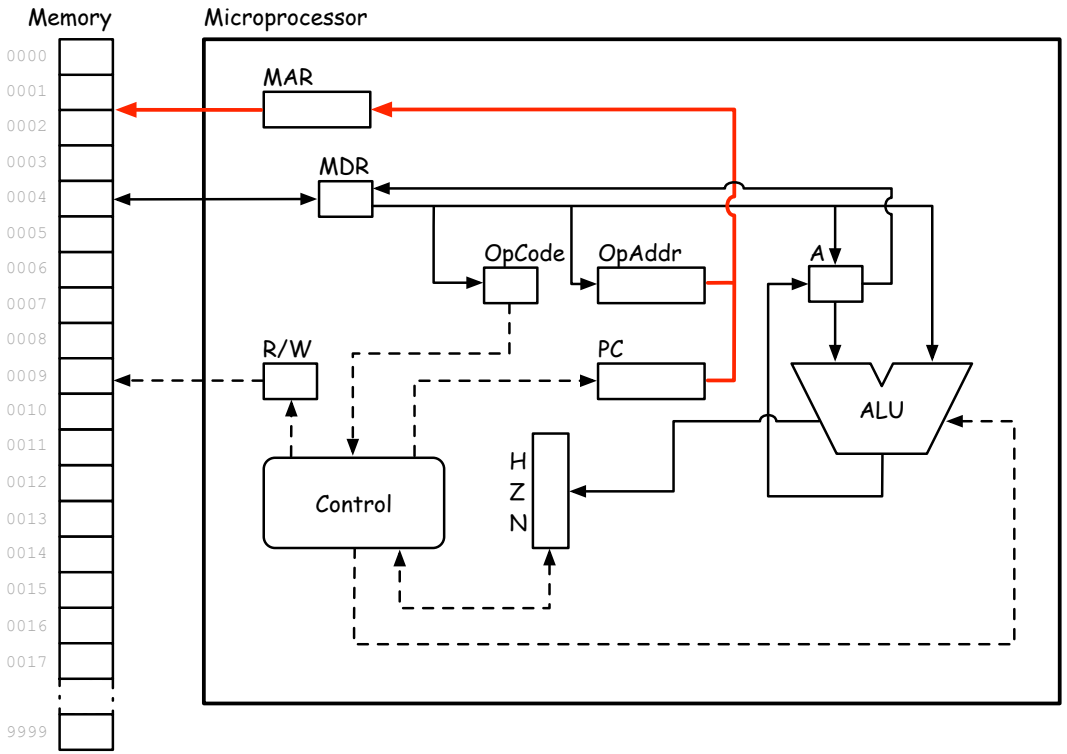
Modèle de la mémoire

- Ce type de mémoire est dit à accès direct (*Random Access Memory*)
- **Le temps d'accès aux cellules de la mémoire est uniforme et constant,**
  De l'ordre de 5 à 70 nano secondes (nano $= 10^{-9}$)



Architecture des ordinateurs

Modèle **simplifié** d'un microprocesseur (TC1101) et de son langage d'assemblage.



- https://www.youtube.com/watch?v=S1V2Tg3oa7I
- https://www.youtube.com/watch?v=SY5-DJ5LAts

Mnémoniques, opCodes, description

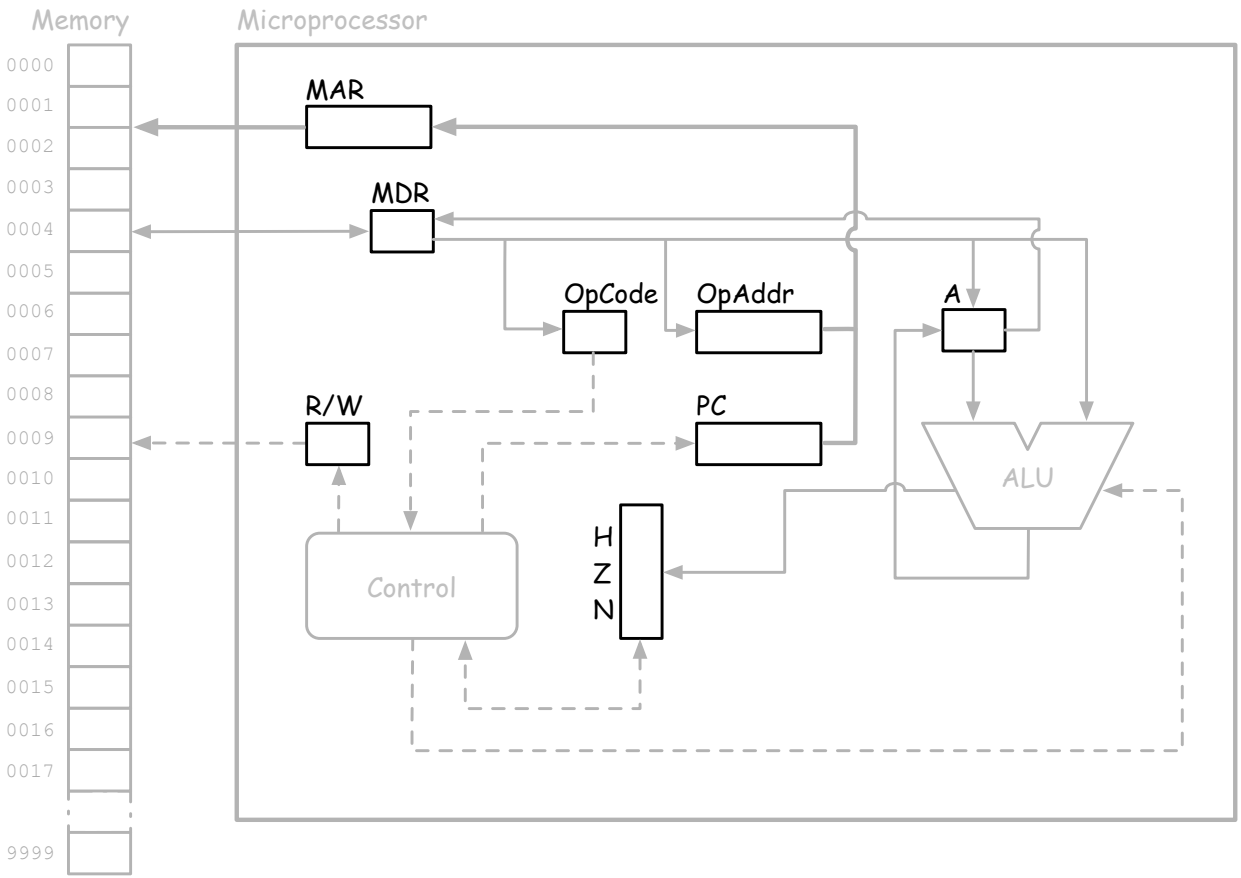
LDA	91	load x
STA	39	store x
CLA	08	clear (a=0, z=vrai, n=faux)
INC	10	incrémente accumulateur (modifie z et n)
ADD	99	ajoute x à l'accumulateur (modifie z et n)
SUB	61	retranche x de l'accumulateur (modifie z et n)
JMP	15	branchement inconditionnel vers x
JZ	17	branchement sur x si z==vrai
JN	19	branchement sur x si n==vrai
DSP	01	affiche la valeur se trouvant à l'adresse x
HLT	64	fin

Instructions du TC1101

- Ce microprocesseur supporte **11 instructions**.
  - Sur le tableau précédent vous trouverez à gauche le **nom de l'instruction**, au centre le **code machine**, et à droite la **description** de l'instruction.
- Les instructions dont le code est **pair** n'ont pas de paramètre, alors que les instructions dont le code est **impair** en on un.
- On utilise plutôt le terme **opérande** pour nommer le paramètre d'une instruction.
- L'opérérande est une **addresse mémoire**.
- `ttps://www.youtube.com/watch?v=V1o29Gl3_BU`

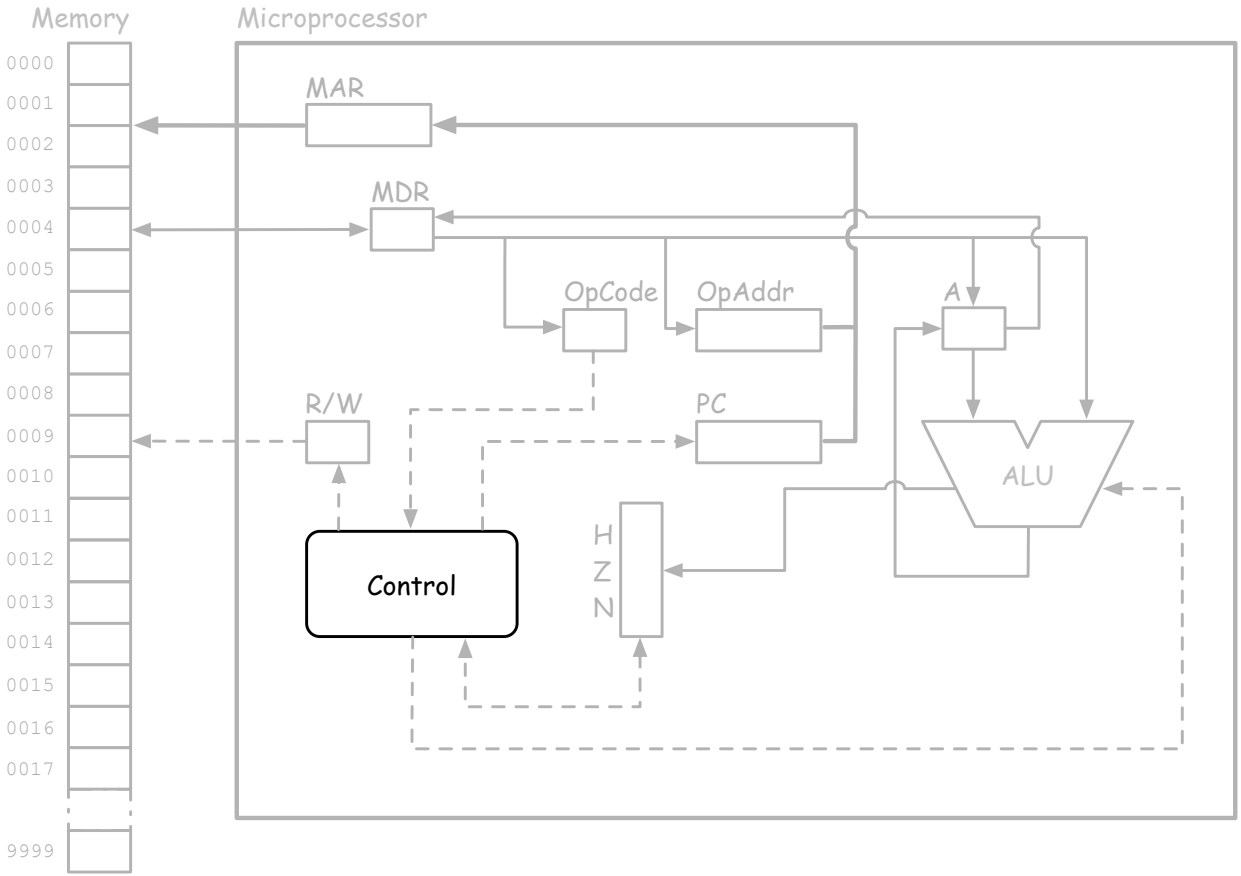
Registres

Les **registres** sont des unités de **mémoire spécialisées**.



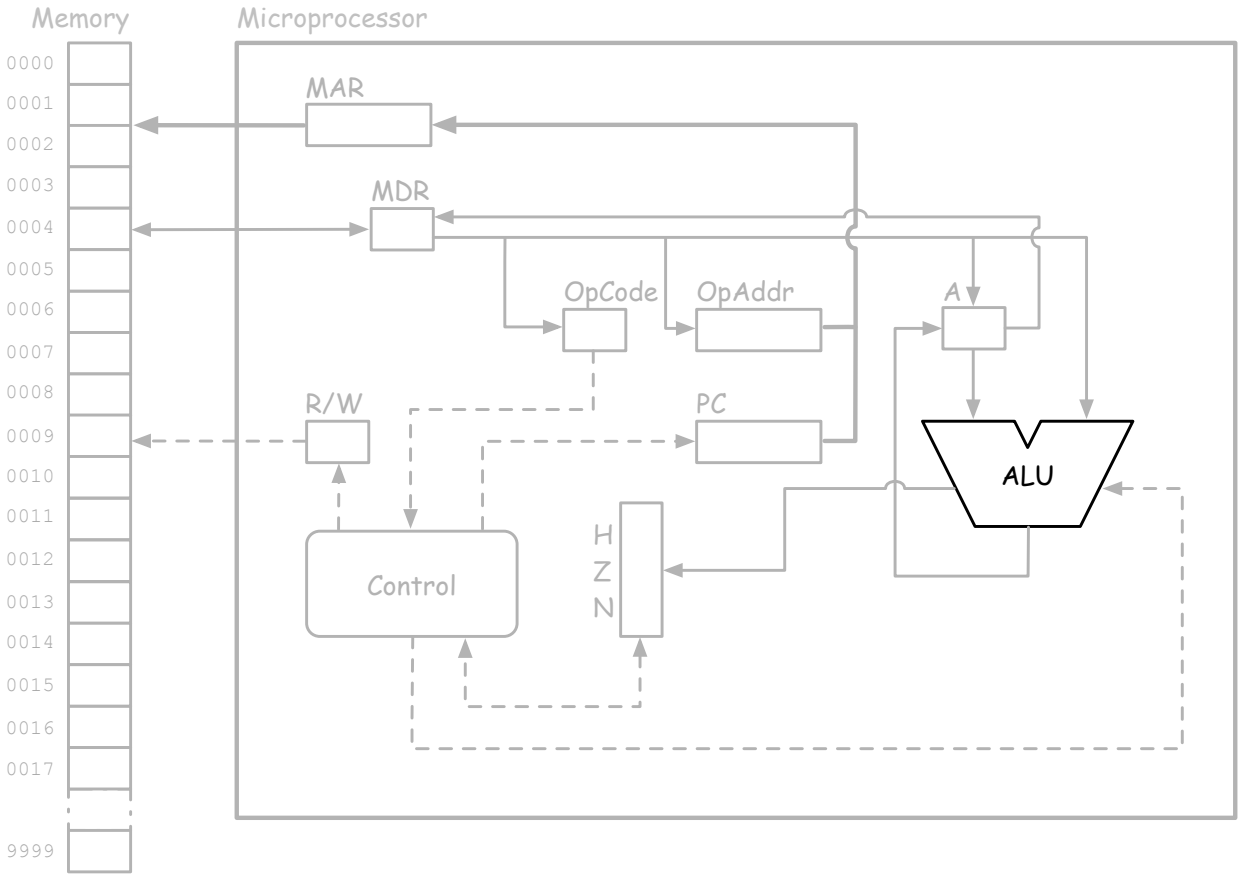
Unité de contrôle

L'**unité de contrôle** orchestre l'exécution des instructions.



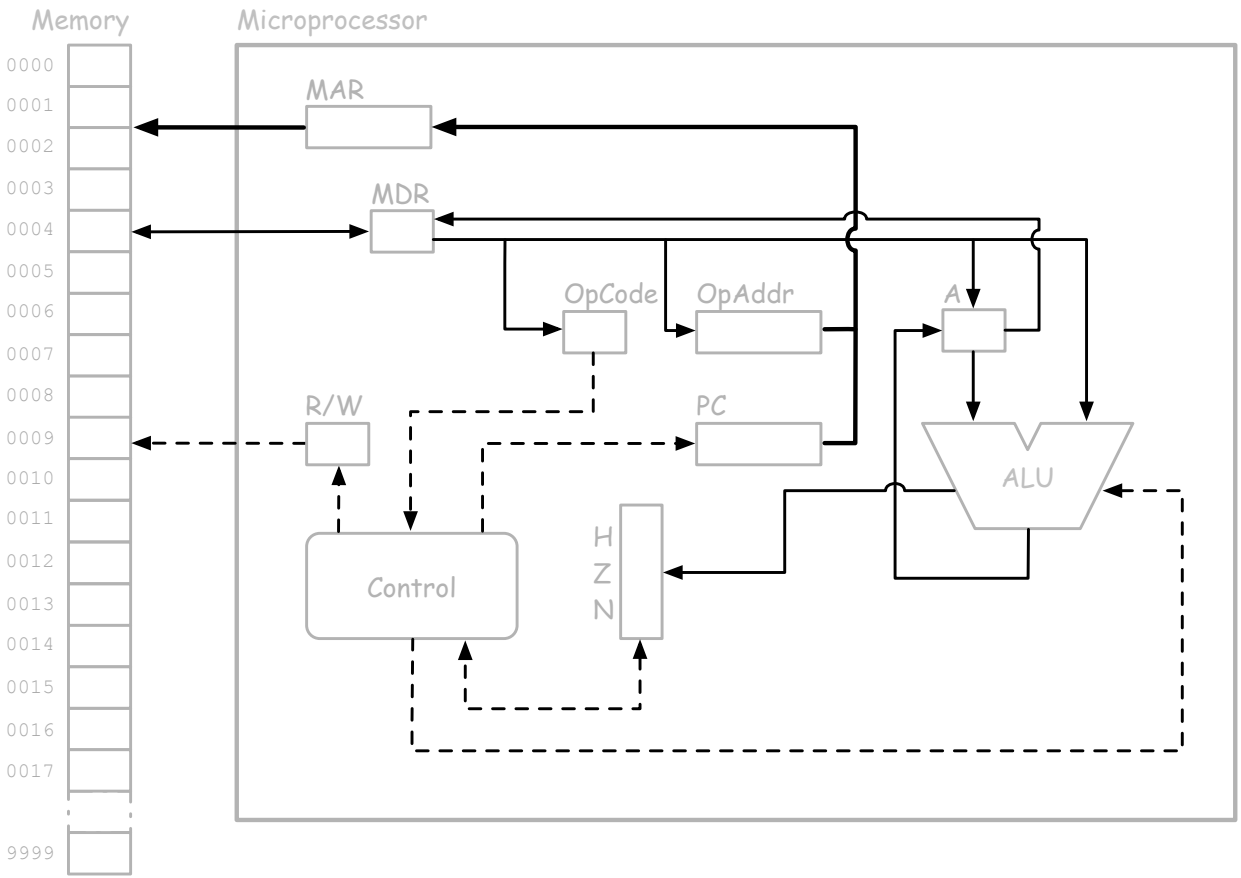
Unité arithmétique et logique

L'unité **arithmétique** et **logique** effectue les calculs.

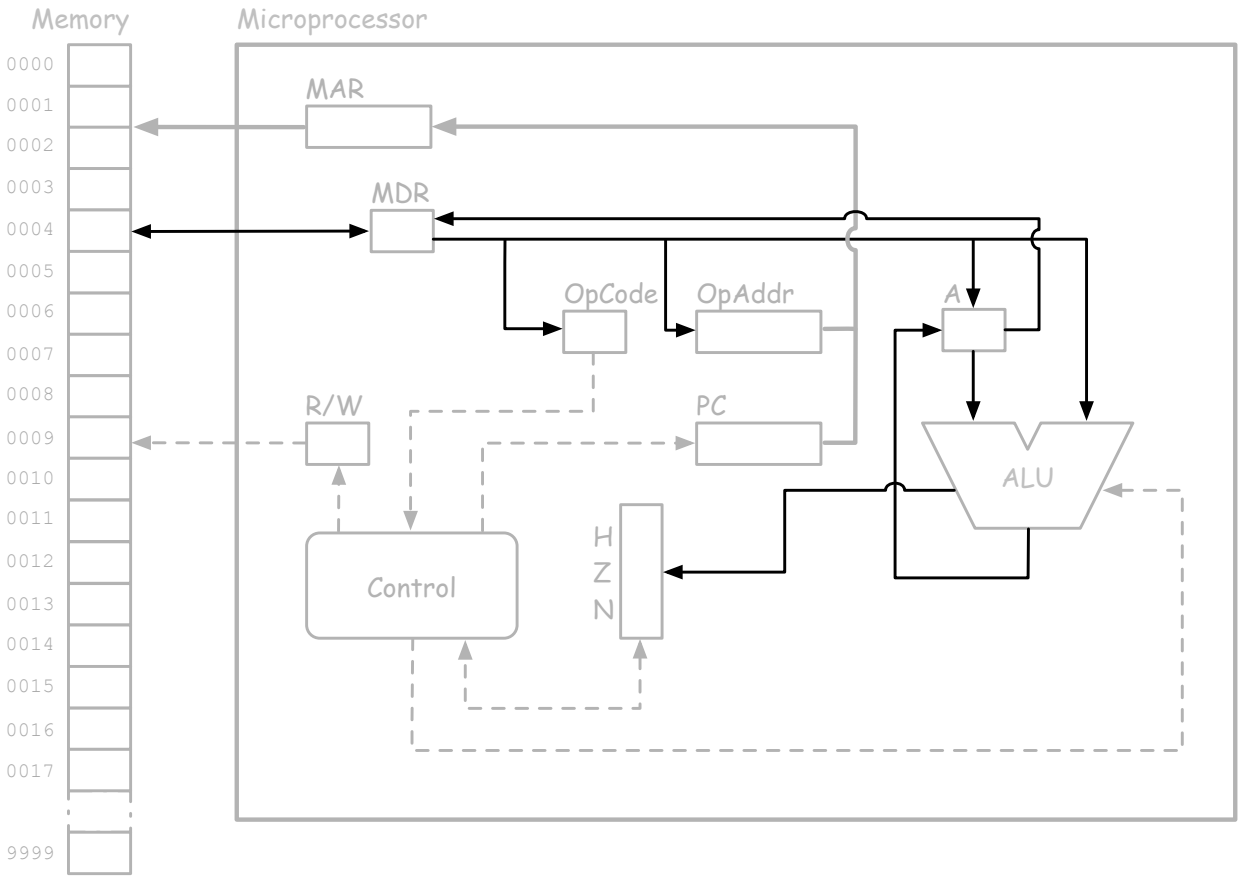


Bus

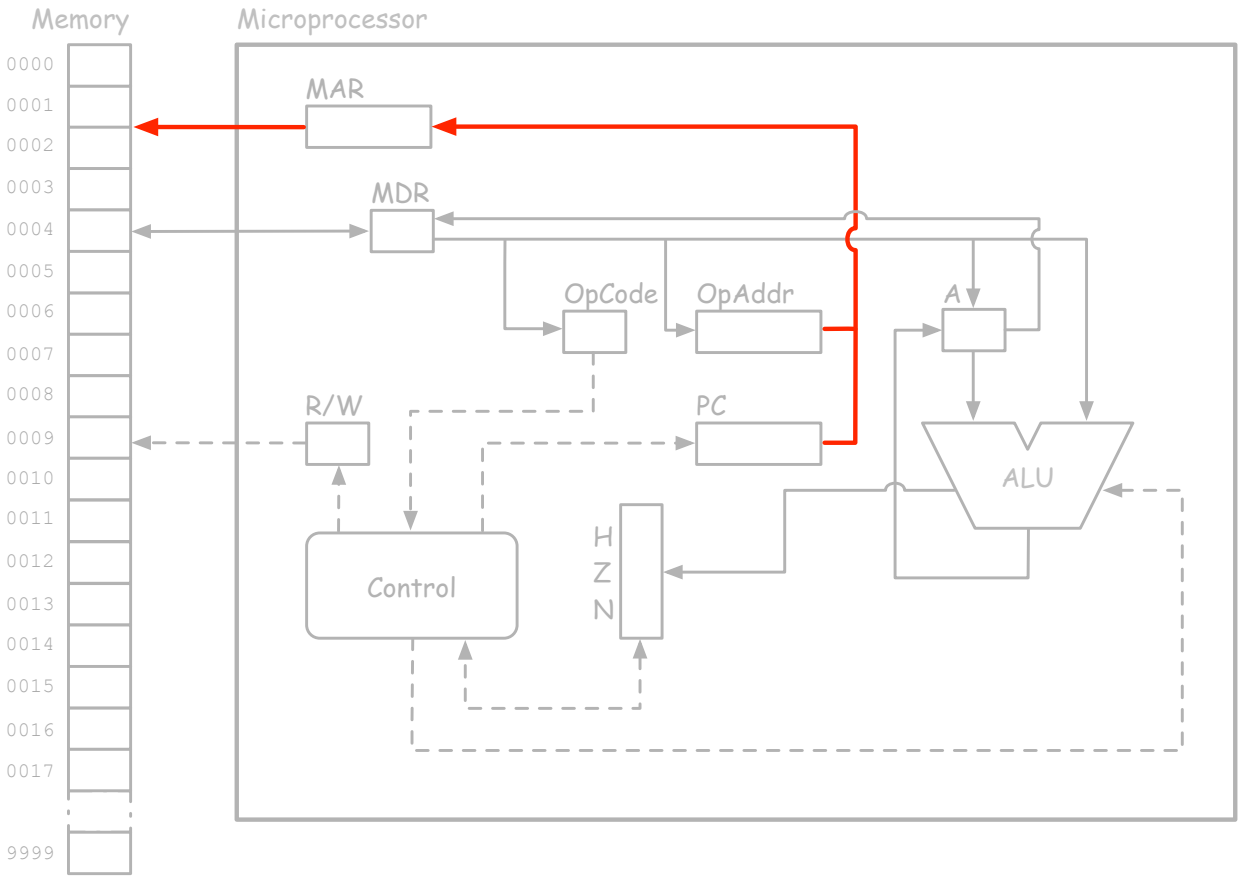
Les informations sont transférés d'une unité à l'autre sur des bus.



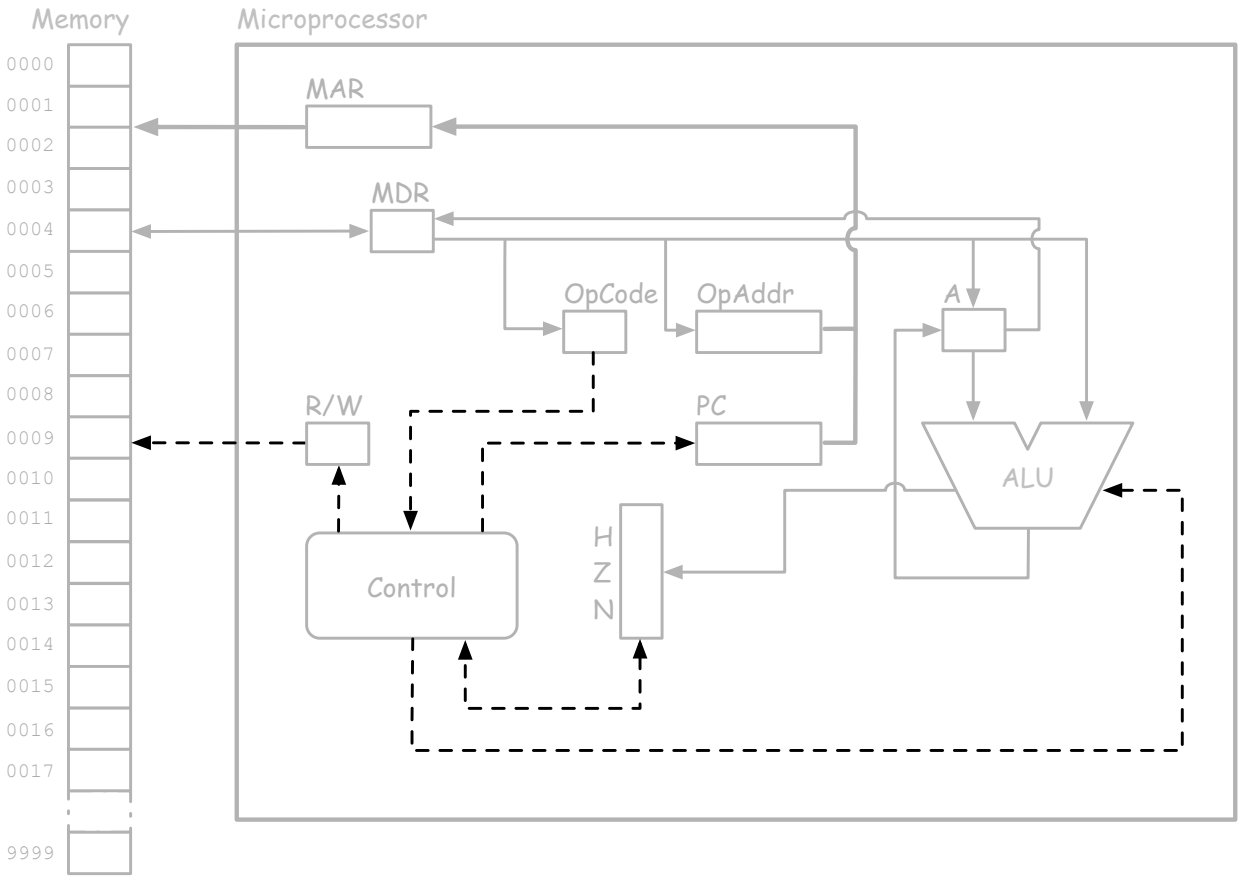
Bus de données



Bus d'adresses



Bus de contrôle

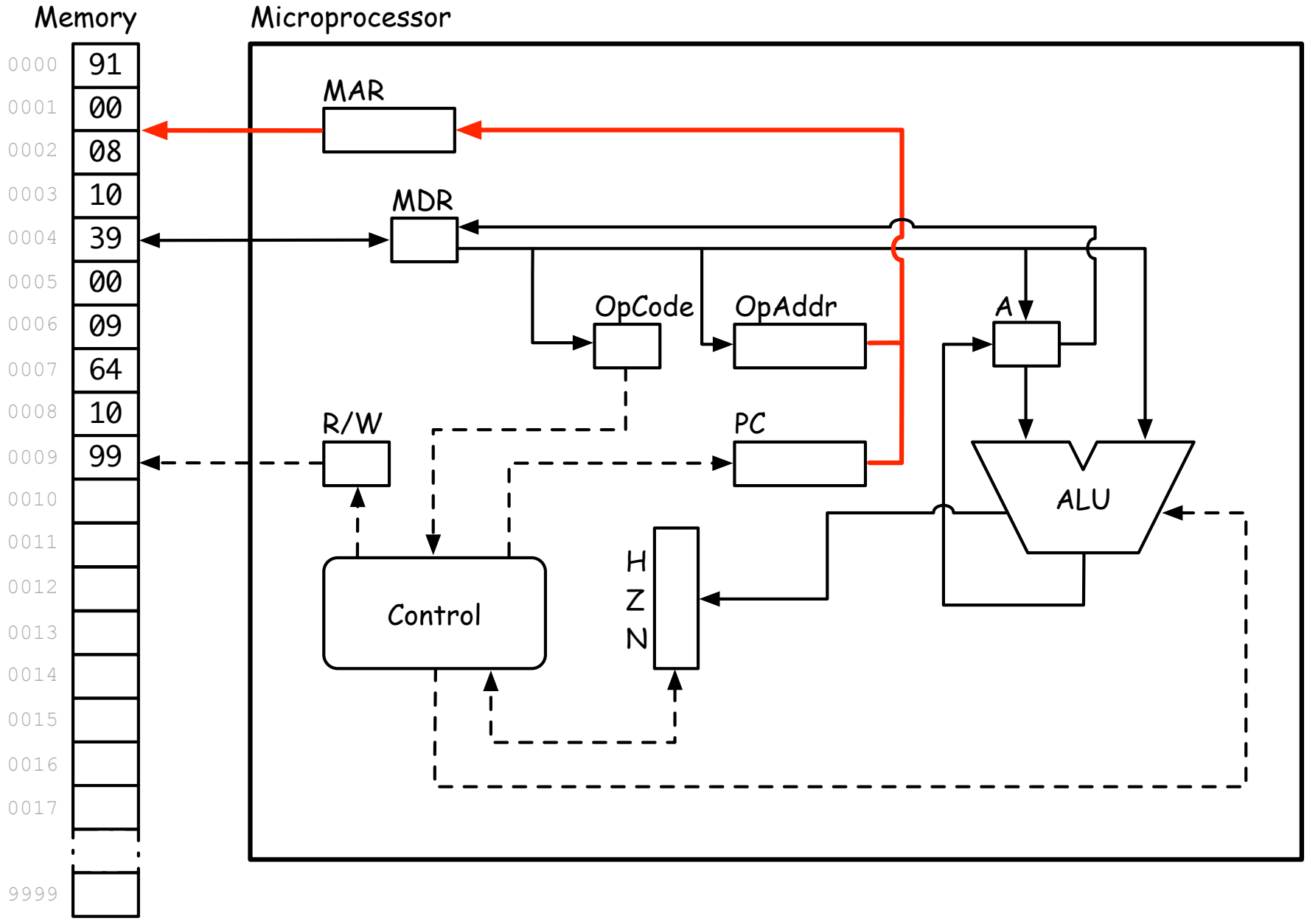


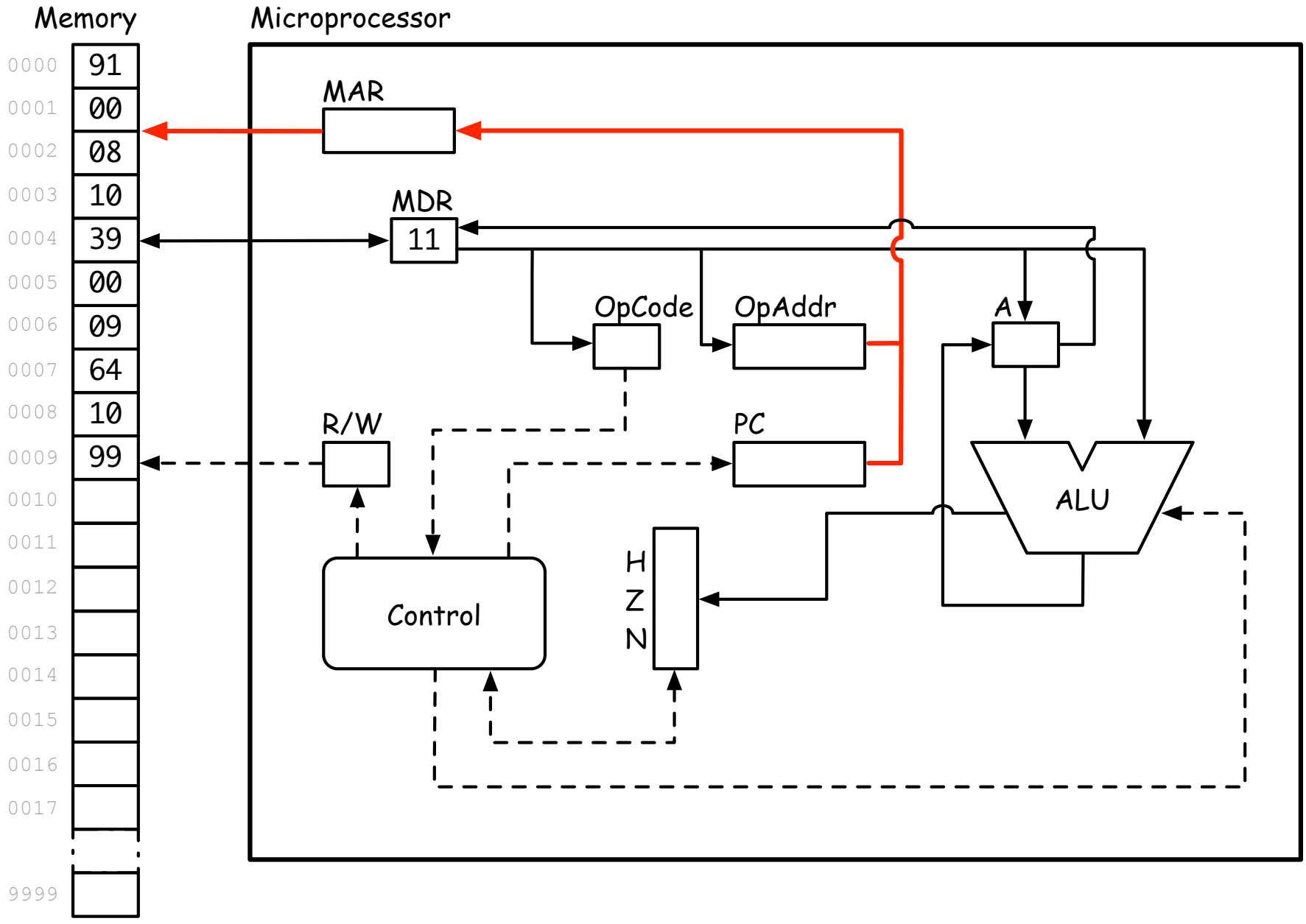
Transfert vers la mémoire

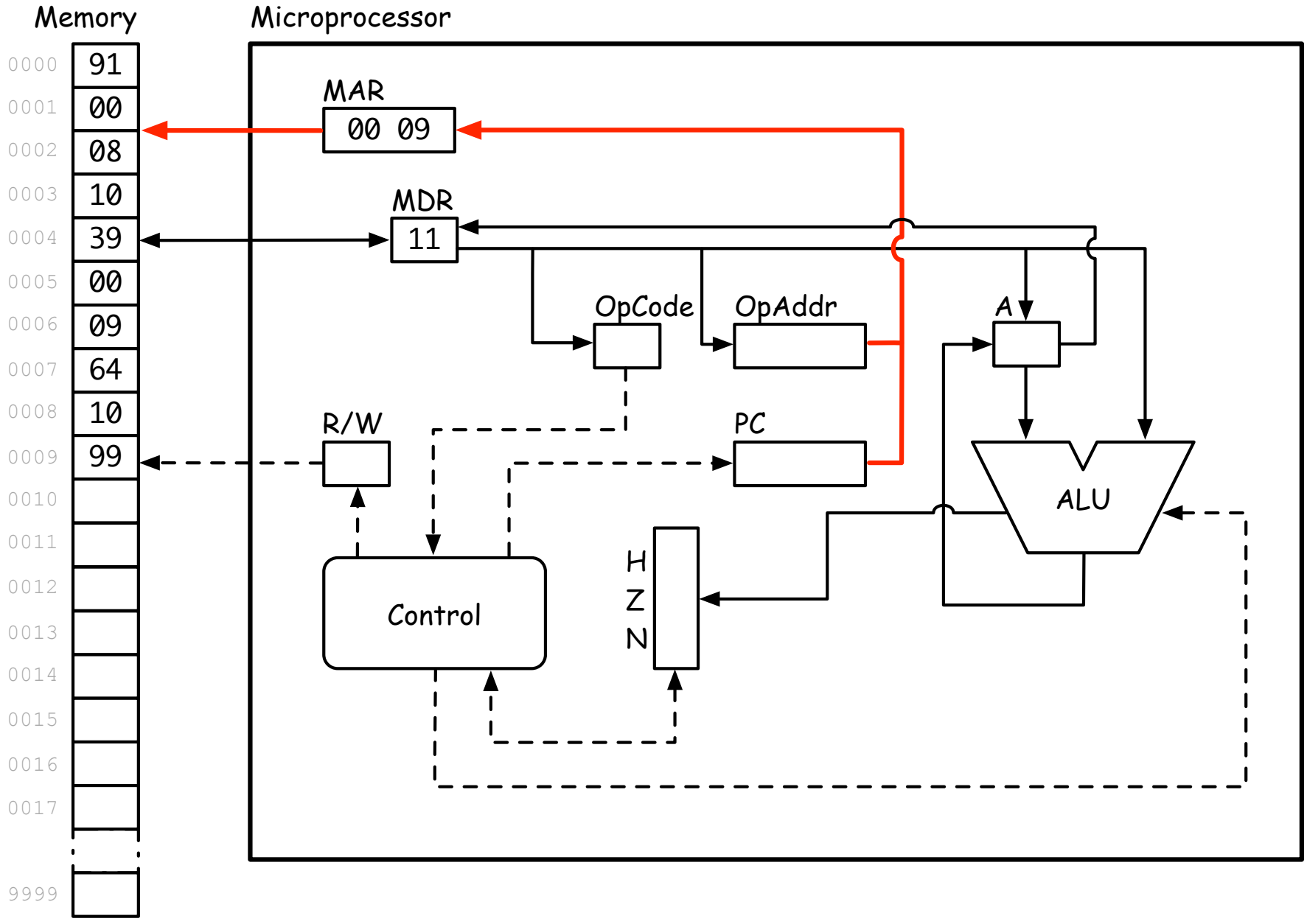
Afin de transférer une **valeur** v du microprocesseur vers **l'adresse** x de la mémoire :

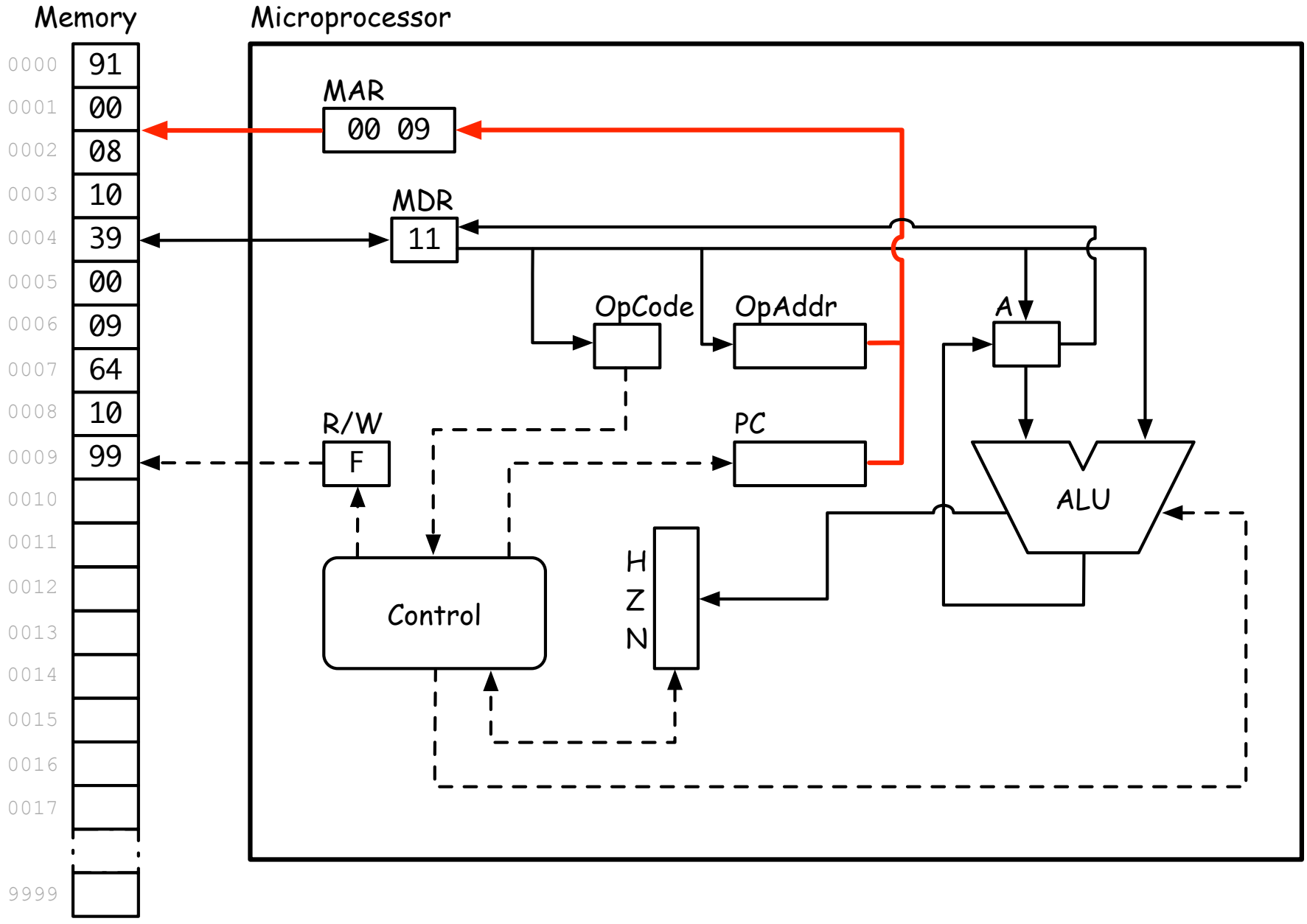
1. mettre v dans le **registre de donnée** (MDR),
2. mettre x dans le **registre d'adresse** (MAR),
3. mettre le **bit de statut RW à faux**,
4. **activer la ligne de contrôle** «access_memory».

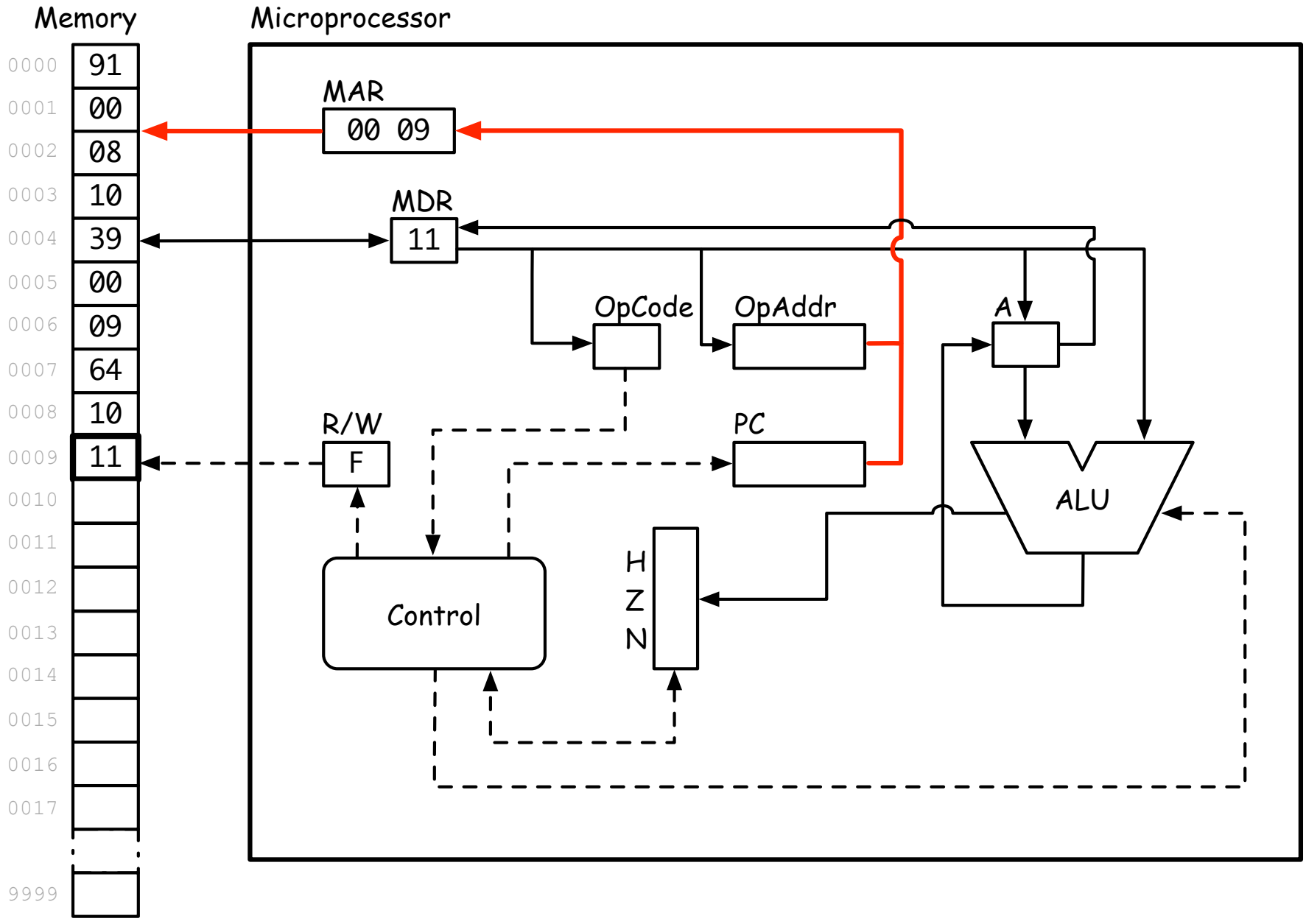
- `https://www.youtube.com/watch?v=h4EKwBZ251c`


Memory
Microprocessor
0000
91
0001
00
0002
08
0003
10
0004
39
0005
00
0006
09
0007
64
0008
10
0009
99
0010
0011
0012
0013
0014
0015
0016
0017
9999
MAR
MDR
OpCode
OpAddr
A
R/W
PC
ALU
Control
H
Z
N


Memory
Microprocessor
0000
91
0001
00
0002
08
0003
10
0004
39
0005
00
0006
09
0007
64
0008
10
0009
99
0010
0011
0012
0013
0014
0015
0016
0017
9999
MAR
MDR
11
OpCode
OpAddr
A
R/W
PC
ALU
Control
H
Z
N


Memory
Microprocessor
0000
91
0001
00
0002
08
0003
10
0004
39
0005
00
0006
09
0007
64
0008
10
0009
99
0010
0011
0012
0013
0014
0015
0016
0017
9999
MAR
00 09
MDR
11
OpCode
OpAddr
A
R/W
PC
ALU
Control
H
Z
N

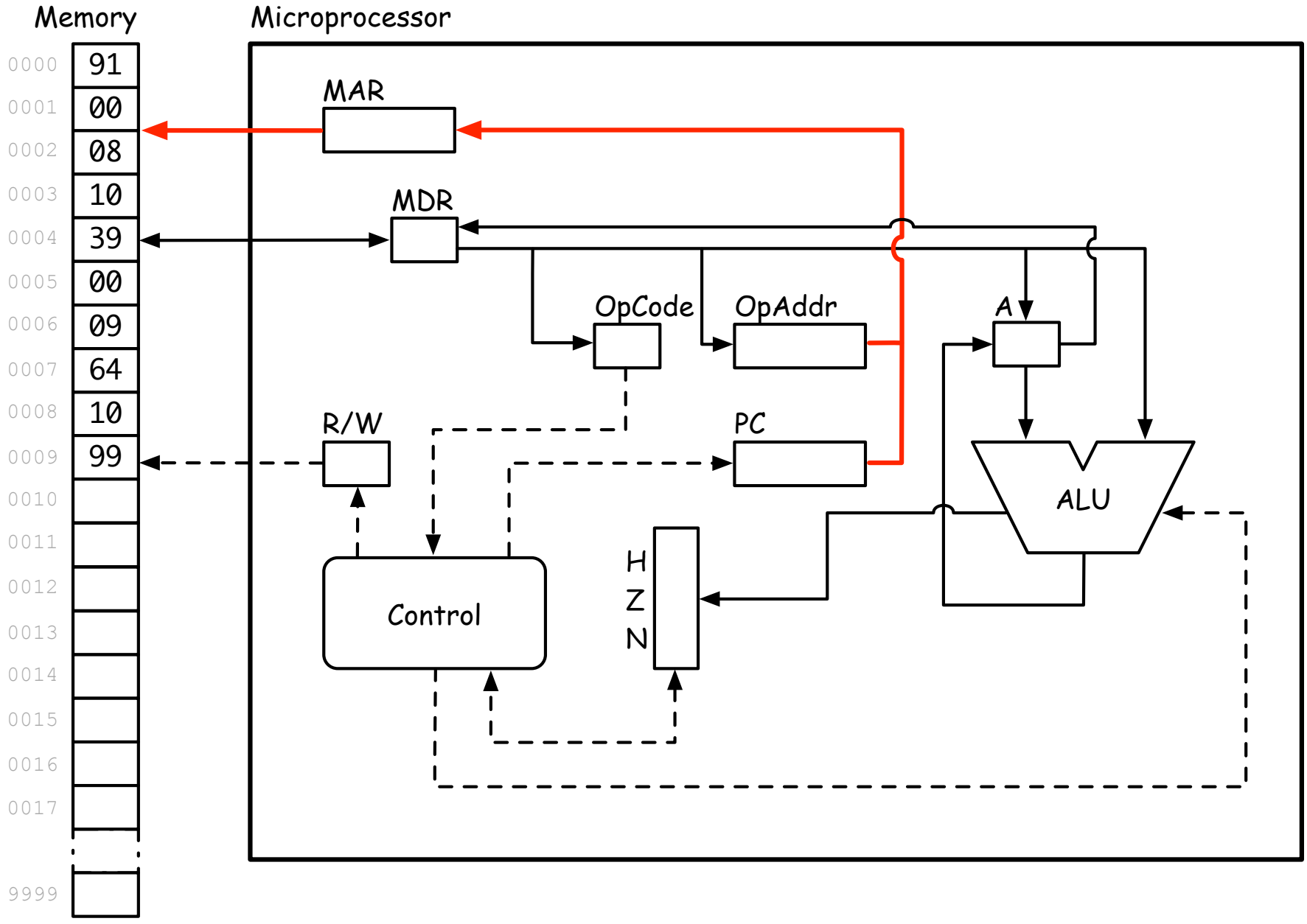

Memory
Microprocessor
0000
91
0001
00
0002
08
0003
10
0004
39
0005
00
0006
09
0007
64
0008
10
0009
99
0010
0011
0012
0013
0014
0015
0016
0017
9999
MAR
00 09
MDR
11
OpCode
OpAddr
A
R/W
F
PC
ALU
Control
H
Z
N

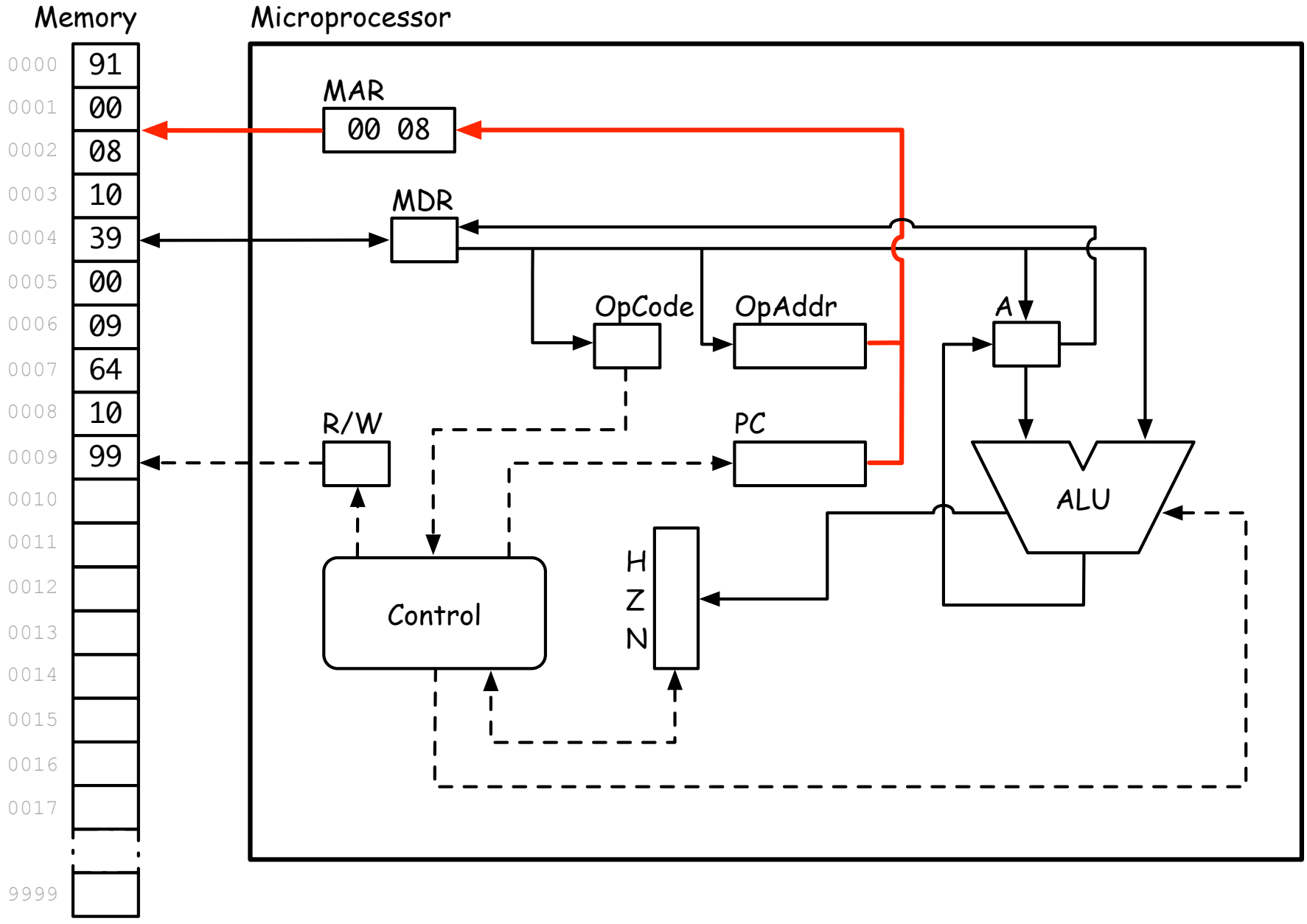

Memory
Microprocessor
0000
91
0001
00
0002
08
0003
10
0004
39
0005
00
0006
09
0007
64
0008
10
0009
11
0010
0011
0012
0013
0014
0015
0016
0017
9999
MAR
00 09
MDR
11
OpCode
OpAddr
A
R/W
F
PC
ALU
Control
H
Z
N

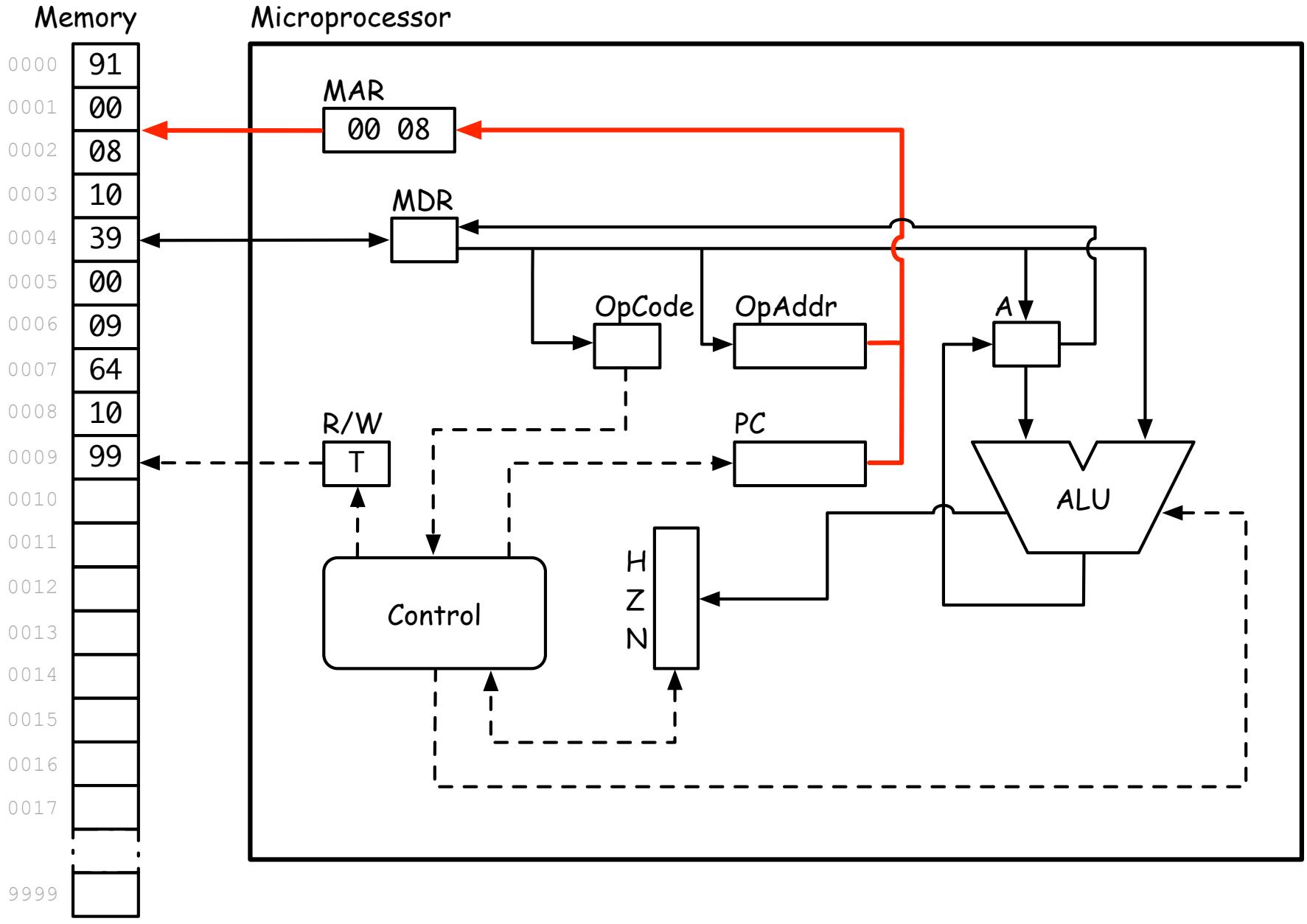
Transfert depuis la mémoire

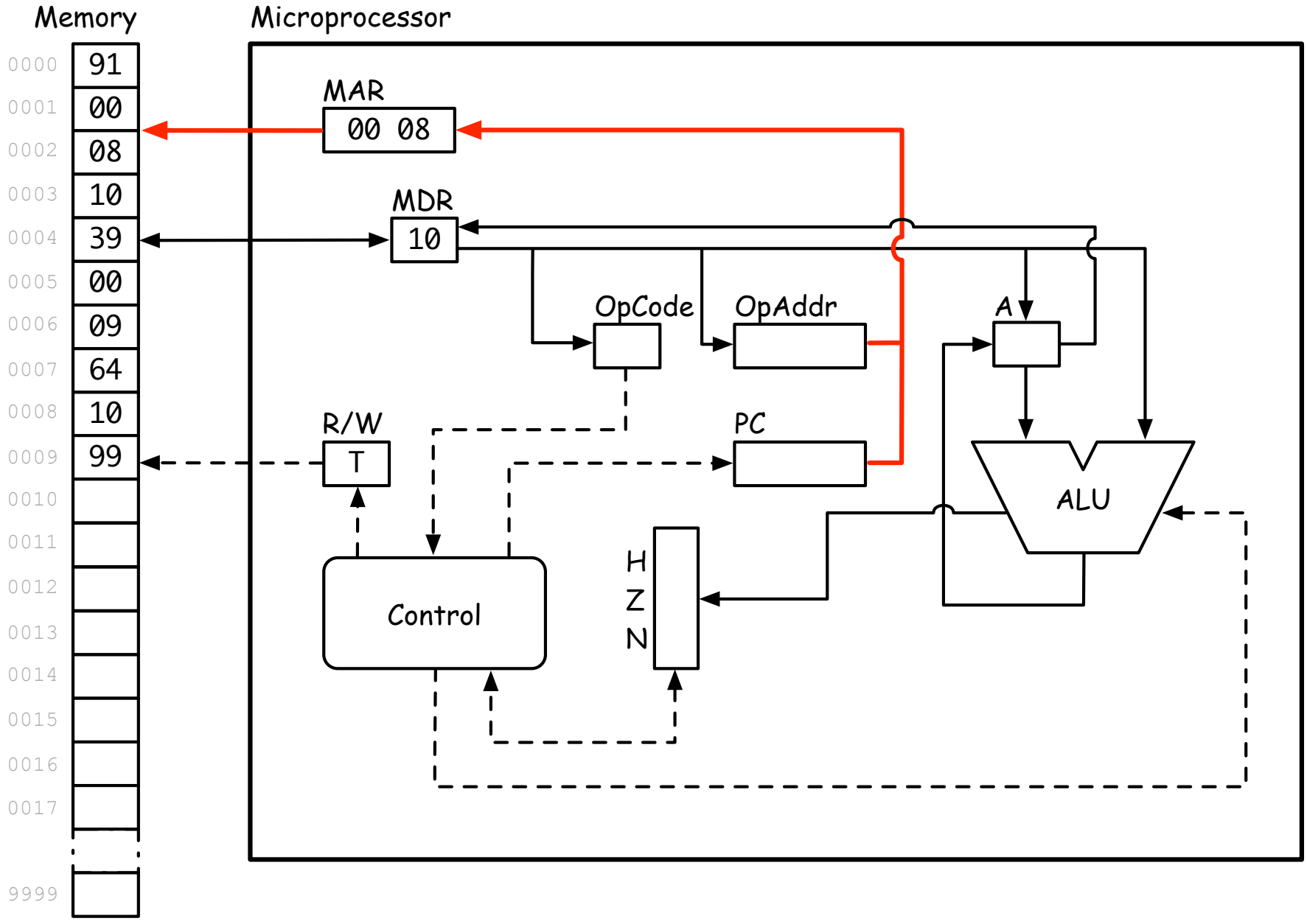
Afin de transférer une valeur **de l'adresse** x (en mémoire) **vers le microprocesseur** :

1. mettre la valeur x dans le **registre d'adressen** (MAR),
2. mettre le **bit de statut RW à vrai**,
3. **activer** la ligne de contrôle «access_memory»
4. le **registre de donnée** (MDR) contient maintenant une **copie** de la valeur se trouvant à l'adresse x de la mémoire.


Memory
Microprocessor
0000
91
0001
00
0002
08
0003
10
0004
39
0005
00
0006
09
0007
64
0008
10
0009
99
0010
0011
0012
0013
0014
0015
0016
0017
9999
MAR
MDR
OpCode
OpAddr
A
R/W
PC
ALU
Control
H
Z
N


Memory
Microprocessor
0000
91
0001
00
0002
08
0003
10
0004
39
0005
00
0006
09
0007
64
0008
10
0009
99
0010
0011
0012
0013
0014
0015
0016
0017
9999
MAR
00 08
MDR
OpCode
OpAddr
A
R/W
PC
ALU
Control
H
Z
N


Memory
Microprocessor
0000
91
0001
00
0002
08
0003
10
0004
39
0005
00
0006
09
0007
64
0008
10
0009
99
0010
0011
0012
0013
0014
0015
0016
0017
9999
MAR
00 08
MDR
OpCode
OpAddr
A
R/W
T
PC
ALU
Control
H
Z
N


Memory
Microprocessor
0000
91
0001
00
0002
08
0003
10
0004
39
0005
00
0006
09
0007
64
0008
10
0009
99
0010
0011
0012
0013
0014
0015
0016
0017
9999
MAR
00 08
MDR
10
OpCode
OpAddr
A
R/W
T
PC
ALU
Control
H
Z
N

Cycle d'instructiontransfert-décode-exécut

1. transfert :
   - 1.1 transfert de l'OPCODE,
   - 1.2 incrémente PC,
2. selon OPCODE alors transfert opérande :
   - 2.1 transfert premier octet,
   - 2.2 incrémente PC,
   - 2.3 transfert second octet,
   - 2.4 incrémente PC,
3. exécute.

$\Rightarrow$ *«Fetch-Decode-Execute Cycle»*.

Compilation

Les programmes, suites d'énoncés d'un langage de programmation de haut niveau, sont traduits (**compilés**), en langage de bas niveau (assembleur, code machine), directement interprétable par le matériel.
L'expression $y = x + 1$ est traduite en assembleur :

```
LDA X
INC
STA Y
HLT
```

qui est ensuite traduit en code machine :

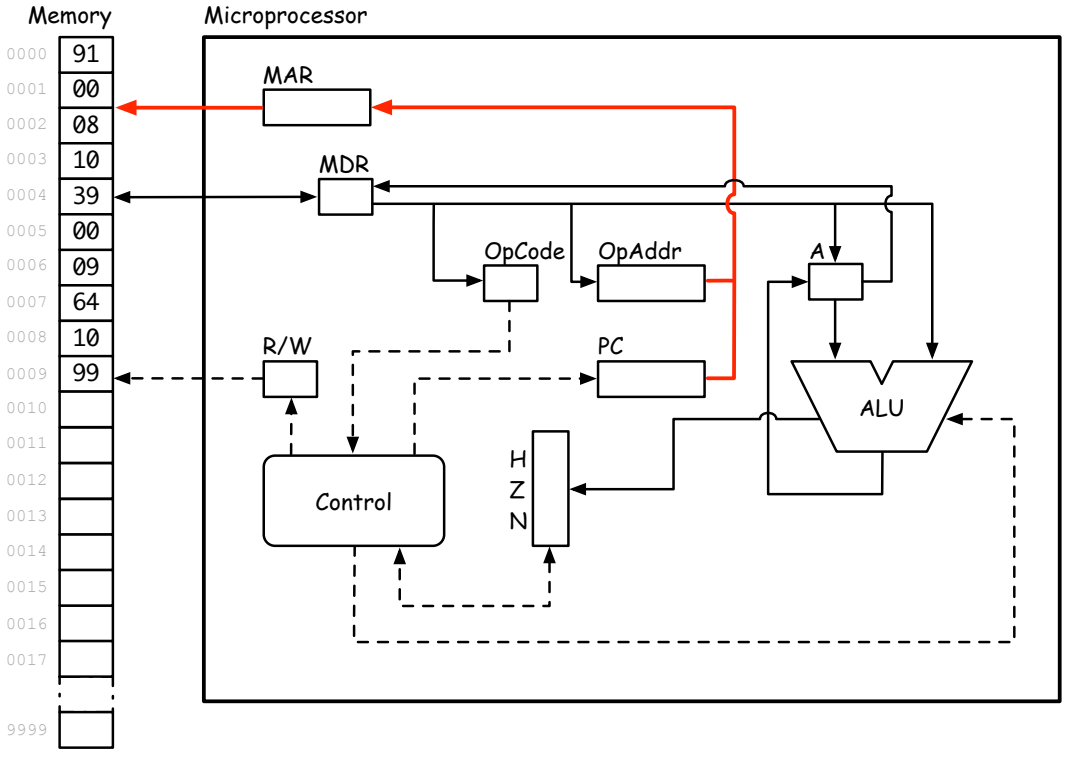
91 00 08	10	39 00 09	64	10	99

L'**expression** $y = x + 1$ est traduite en **assembleur** :

```
LDA X
INC
STA Y
HLT
```

qui est ensuite traduit en **code machine** :

91 00 08	10	39 00 09	64	10	99



- https://www.youtube.com/watch?v=K293jNEFnF4

Description du TC1101

PC : Compteur de programme (*Program Counter*), registre contenant l'adresse de la prochaine instruction à exécuter ;

OPCODE : registre d'instruction (parfois dénoté IR), contient l'OPCODE de l'instruction en cours ;

opAddr : L'opérande de l'instruction en cours d'exécution. L'opérande est toujours une adresse (pour le TC-1101 seulement). Certaines instructions nécessitent la valeur contenue à l'adresse indiquée ; cette valeur n'est pas transférée par le cycle de transfert de base, mais doit être transférée par l'opération elle-même lors de l'exécution (voir étape 3 du cycle de transfert ainsi que la description de chacune des instructions ci-bas) ;

Description du TC1101

MDR : registre donnée mémoire (*Memory Data Register*). Une donnée transférée (*read/loaded*) de la mémoire vers le processeur est toujours placée dans le registre de donnée. De même, une donnée transférée du processeur vers la mémoire (*stored*) doit être placée dans le registre de donnée ;

MAR : registre adresse mémoire (*Memory Address Register*). Ce registre contient l'adresse mémoire à partir de laquelle une donnée doit être lue ou vers laquelle elle doit être écrite ;

Description du TC1101

A : Accumulateur. Toutes les opérations arithmétiques utilisent ce registre comme opérande et aussi pour sauver leur résultat ;

H (bit) : le bit de statut «Halt». Ce bit est mis-à-jour par l'instruction «halt» (hlt). Si ce bit est vrai, le processeur s'arrête lorsque l'opération courante est terminée ;

N (bit) : le bit de statut «Negative». Les opérations arithmétiques affectent la valeur vrai à ce bit lorsqu'elles produisent un résultat négatif. Certaines opérations n'affectent pas la valeur de ce bit, ainsi sa valeur ne reflète pas toujours l'état courant de l'accumulateur ;

Description du TC1101

Z (bit) : le bit de statut «Zero». Les opérations arithmétiques affectent la valeur vrai à ce bit lorsqu'elles produisent le résultat zéro. Certaines opérations n'affectent pas la valeur de ce bit, ainsi sa valeur ne reflète pas toujours l'état courant de l'accumulateur ;

RW (bit) : le bit de statut «READ/WRITE». Si sa valeur est vrai, signifie qu'une donnée sera lue (*fetched*) de la mémoire et transférée dans le registre MDR de l'unité centrale. Si faux, signifie qu'une donnée sera transférée du registre MDR vers la mémoire.

Language assembleur

- Le langage assembleur est **peu expressif**.
- Chaque type de microprocesseur possède son propre langage assembleur. Les programmes **ne sont pas portables** d'un type d'ordinateur à un autre.

Language à haut niveau

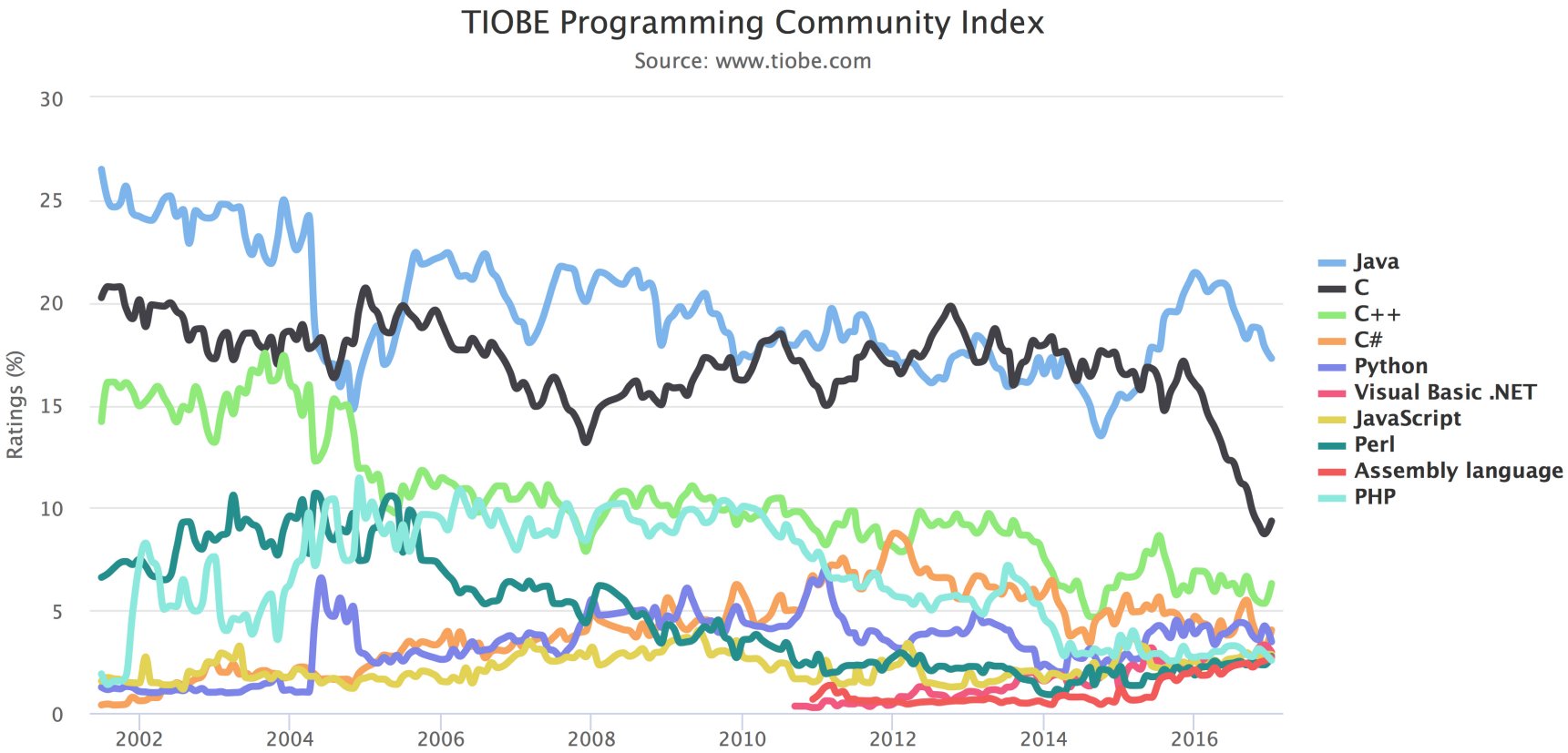
- Un langage à haut niveau est **expressif**.
- Généralement, un langage à haut niveau est aussi **portable**.

Paradigmes des langages de programmation

- **Impératif** ou **procédural**
- **Orienté objet**
- **Descriptif**
  - Fonctionnel
  - Logique
  - Par contraintes

Pourquoi Java ?

Pourquoi Java ?



Pourquoi Java ?

1	Java	17%
2	C	9%
3	C++	6%
4	C#	4%
5	Python	4%
6	Basic	3%
7	JavaScript	3%
8	Perl	3%
9	Assembly	3%
10	PHP	3%

TIOBE Programming Community Index

Pourquoi Java ?

1	JavaScript
2	Java
3	PHP
4	Python
5	C#
5	C++
5	Ruby
8	CSS
9	C
10	Objective-C

The **RedMonk** Programming Language Rankings: January 2016

Pourquoi Java ?

Java partage le **premier rang** avec C des langages les plus utilisés, pourtant, je ne connais pas d'applications Java. Où trouve-t-on des applications Java ?

- La partie **serveur** des applications et **services Web**
- **Applications mobiles** (téléphones portables)

2015 Top Trending Careers in Tech : Java Developer

- info.theladders.com/career-advice/top-market-trends-in-tech

Pourquoi Java ?

« According to a report from NetApplications, which has measured browser usage data since 2004, Oracle's Java Mobile Edition has surpassed Android as **the #2 mobile OS** on the internet at 26.80%, with iOS at 46.57% and Android at 13.44%. And the trend appears to be growing. Java ME powers hundreds of millions of low-end 'feature phones' for budget buyers. In 2011, **feature phones made up 60% of the install base in the U.S.** »

Slashdot
3 janvier 2012
http://bit.ly/xSk5pN

Pourquoi Java ?

- La programmation en **C** demande une grande discipline (gestion de la mémoire, manipulation de pointeurs. . .)
- Java est un **bon véhicule pour l'enseignement** (interface, héritage simple, types génériques. . .)
- Lorsqu'on maîtrise Java, l'apprentissage des langages orientés objet ou impératifs est **simple**

Pourquoi Java ?

Source : https://commons.wikimedia.org/wiki/File:Netflix_logo.svg

https://go.java/netflix.html

Prochain module

- **Types de données**

Division par soustractions successives

```
[1]   CLA
      STA Quot
[2]   LDA X
[3]   SUB Y
[4]   JN  [7]
[5]   STA Temp
      LDA Quot
      INC
      STA Quot
      LDA Temp
[6]   JMP [3]
[7]   ADD Y
[8]   STA Rem
[9]   DSP Quot
[10]  DSP Rem
[11]  HLT
X     BYTE 25
Y     BYTE 07
Quot  BYTE 00
Rem   BYTE 00
Temp  BYTE 00
```

Division : code machine

```
[1]   CLA                    08
      STA Quot               39      00 44
[2]   LDA X                  91      00 42
[3]   SUB Y                  61      00 43
[4]   JN  [7]                19      00 29
[5]   STA Temp               39      00 46
      LDA Quot               91      00 44
      INC                    10
      STA Quot               39      00 44
      LDA Temp               91      00 46
[6]   JMP [3]        ⟹       15      00 07
[7]   ADD Y                  99      00 43
[8]   STA Rem                39      00 45
[9]   DSP Quot               01      00 44
[10]  DSP Rem                01      00 45
[11]  HLT                    64
X     BYTE 25                25
Y     BYTE 07                07
Quot  BYTE 00                00
Rem   BYTE 00                00
Temp  BYTE 00                00
```

Division : code machine

```
08 39 00 44 91 00 42 61 00 43 19 00 29 39 00 46 91 00 44 10 39 00
44 91 00 46 15 00 07 99 00 43 39 00 45 01 00 44 01 00 45 64 25 07
00 00 00
```

References I


E. B. Koffman and Wolfgang P. A. T.
***Data Structures : Abstraction and Design Using Java*.**
John Wiley & Sons, 3e edition, 2016.

Marcel **Turcotte**

Marcel.Turcotte@uOttawa.ca

École de **science informatique** et de génie électrique (SIGE)

Université d'Ottawa