

Architecture des Ordinateurs

Introduction

Licence Informatique - USTL

David Simplot
simplot@fil.univ-lille1.fr



À propos du cours

- Site web du cours :
 - <http://www.lifl.fr/~simplot/ens/archi>
- Les TD commencent la semaine du 15/10
- Les TP commencent la semaine du 22/10
- Évaluation :
 - Trois DS en Travaux Dirigés
 - Un projet en Travaux Pratiques (en Assembleur)
 - contrôle individuel sur machine
 - Un examen en janvier



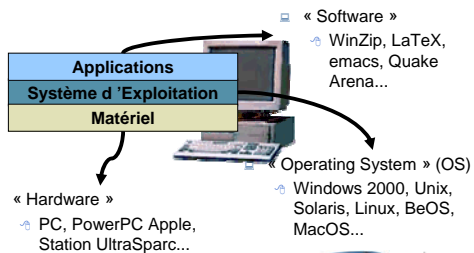
D. SIMPLOT - Architecture des Ordinateurs



2

Objectifs (1/2)

- Comment fonctionne un ordinateur ?

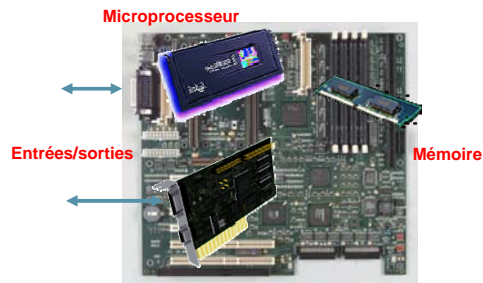


D. SIMPLOT - Architecture des Ordinateurs



3

Objectif (2/2)



D. SIMPLOT - Architecture des Ordinateurs



4

Plan du cours



- ▣ Introduction
 - ↪ Objectifs, Plan, Historique
- ▣ Partie I : Concepts de Base
 - ↪ Qu'est-ce qu'un ordinateur ?
- ▣ Partie II : Le microprocesseur
- ▣ Partie III : Liens avec le système d'exploitation
- ▣ Partie IV : Gestion de la mémoire et E/S

D. SIMPLOT - Architecture des Ordinateurs



5

Historique (1/7)

- ▣ Préhistoire
 - ↪ -500 Apparition des premiers outils pour calculer
 - bouliers, abaque
 - ↪ 1632 invention de la règle à calcul
 - ↪ 1642 Pascal invente la « Pascaline » 
 - ↪ 1833 Machine de Babbage
 - ↪ 1854 Boole publie un ouvrage sur la logique
 - ↪ 1904 invention du tube à vide
 - ↪ 1937 Alan Turing publie des articles sur les fonctions calculables
 - ↪ 1943 Création du ASCC Mark I (Harvard - IBM)
 - Automatic Sequence-Controlled Calculator
 - ↪ 1945 naissance du bug !  **Bogue !**

D. SIMPLOT - Architecture des Ordinateurs



6

Historique (2/7)

Les premiers ordinateurs

- ↻ 1946 Création de l'ENIAC
 - Electronic Numerical Integrator and Computer
 - architecture Von Neuman
- ↻ 1947 invention du transistor
- ↻ 1956 premier ordinateur à transistors le : TRADIC (Bell)
- ↻ 1958 premier circuit intégré créé par Texas Instrument
- ↻ 1960 premier jeu sur ordinateur : SpaceWar!
- ↻ 1964 langage de programmation BASIC
- ↻ 1968 invention de la souris (Stanford)
- ↻ 1969 Systèmes d'exploitation
 - MULTICS puis UNIX (Bell)



D. SIMPLOT - Architecture des Ordinateurs



7

Historique (3/7)

L'informatique dans un garage

- ↻ 1971 ARPANET (ancêtre de l'internet)
- ↻ 1971 Intel commercialise le premier microprocesseur
 - le 4004 (4 bits, 108 KHz, 2300 transistors en 10 microns)...
- ↻ 1972 Intel sort le 8008 (8 bits, 200 KHz, 3500 transistors)
- ↻ 1972 Bill Gates et Paul Allen fondent Traf-of-Data
- ↻ 1973 Gary Kildall écrit le système d'exploitation CP/M
- ↻ 1973 Invention du C pour le développement d'UNIX
- ↻ 1974 le français François Moreno invente la carte à puce
- ↻ 1974 Motorola commercialise son 1er processeur
 - le 6800 (8 bits)
- ↻ 1974 Intel sort le 8080 (8 bits,)



D. SIMPLOT - Architecture des Ordinateurs



8

Historique (4/7)

L'informatique dans un garage (suite)

- ↻ 1975 Traf-of-Data devient Micro-Soft
- ↻ 1976 Steve Jobs et Steve Wozniak commercialisent l'Apple Computer (à base de MOS Tech. 6502)
- ↻ 1976 Zilog sort le Z80
 - 8bits, 2.5MHz



Micro-informatique

- ↻ 1978 Intel lance son 8086
 - (16bits, 4.7 MHz, 29000 transistors à 3 microns)
- ↻ 1979 Taito sort le jeu Space Invaders...
- ↻ 1979 Motorola commercialise le 68000
 - 16/32 bits, 68000 transistors
- ↻ 1980 Sinclair sort le ZX80 à base de Z80...



D. SIMPLOT - Architecture des Ordinateurs



9

Historique (5/7)

Micro-Informatique (suite)

- 1980 IBM sous-traite le système d'exploitation de sa future machine (à base de 8086) à Microsoft...
 - QDOS → 86-DOS → MS-DOS
- 1982 Intel commercialise le 80286
 - 16 bits, 6 MHz, 134000 transistors
- 1982 Microsoft édite une version MS-DOS pour compatibles !
- Sony et Phillips inventent le CD-ROM
- 1984 Apple sort le Macintosh avec une interface graphique conviviale...
- ...

D. SIMPLOT - Architecture des Ordinateurs



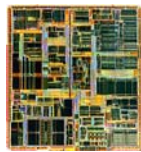
10

Historique (6/7)

1971, le 4004
4 bits, 108 KHz
2300 transistors (10 microns)



30 ans



<http://histoire.info.free.fr>
<http://www.intel.com>

2001, le pentium IV
64 bits - 1,4 GHz,
42 millions de transistors (0,18 microns)

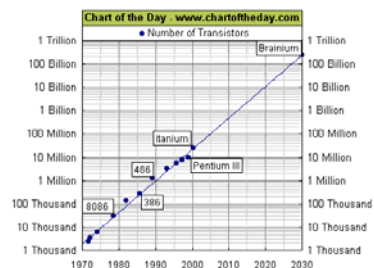
D. SIMPLOT - Architecture des Ordinateurs



11

Historique (7/7)

Loi de Moore



D. SIMPLOT - Architecture des Ordinateurs



12

Architecture des Ordinateurs

Partie I : Concepts de Base

1. Qu'est-ce qu'un ordinateur ?

David Simplot
simplot@fil.univ-lille1.fr



Au sommaire...

- ▣ **Modèle de Von Neuman / architecture réelle**
 - ▣ Représentation de l'information
 - ▣ Algèbre de Boole et fonctions booléennes
 - ▣ Conclusion

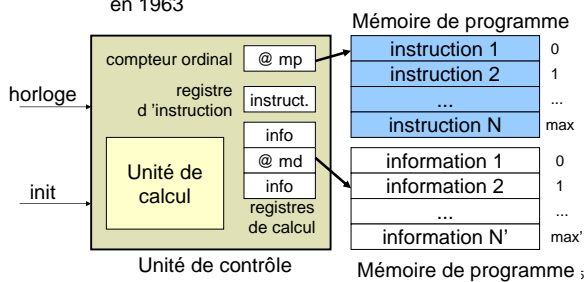
D. SIMPLOT - Architecture des Ordinateurs



14

Modèle de Von Neuman

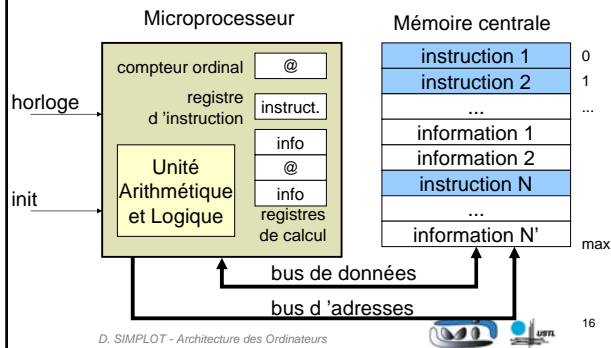
- ▣ inventé par le mathématicien hongrois Von Neuman en 1963



D. SIMPLOT - Architecture des Ordinateurs



Architecture réelle



D. SIMPLOT - Architecture des Ordinateurs

Instruction ou information ?

- ❑ Qu'est-ce qui fait la différence entre une instruction ou une information ?
- ❑ Qu'est-ce qu'une information ?
- ❑ Dans un ordinateur, il n'y a que des 0 ou des 1
 - courant → 1
 - pas de courant → 0
 - on parle de **bit** pour **binary digit**
- ❑ En mémoire, on a des mots binaires d'une taille fixée par le microprocesseur (ex. 8 bits, 16 bits,...)

D. SIMPLOT - Architecture des Ordinateurs



17

Système de numération (1/7) Exemple Base 4

- ❑ En base 10 (décimale), on utilise les chiffres :
 - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- ❑ En base 4, on utilise les chiffres :
 - 0, 1, 2, 3

D. SIMPLOT - Architecture des Ordinateurs



18

Système de numération (2/7)

Exemple Base 4

base 4	base 10	base 4	base 10
0	0	12	6
1	1	13	7
2	2	20	8
3	3		
10	4		
11	5		
		213	39

$3 + 1 \times 4 + 2 \times 4 \times 4 = 3 + 4 + 32 = 39$

D. SIMPLOT - Architecture des Ordinateurs



19

Système de numération (3/7)

Base B

□ Système par position:

↪ $d_{n-1}d_{n-2}\dots d_1d_0, d_1\dots d_m$

• ex : base 4 : 301,23 (n=3, m=4)

↪ $(N)_B = \underbrace{d_{n-1}B^{n-1} + d_{n-2}B^{n-2} + \dots + d_1B^1 + d_0B^0}_{\text{Partie entière}} + \underbrace{d_{-1}B^{-1} + \dots + d_{-m}B^{-m}}_{\text{Partie fractionnaire}}$

• ex : $(301,23)_4 = 3 \cdot 4^2 + 0 \cdot 4^1 + 1 \cdot 4^0 + 2 \cdot 4^{-1} + 3 \cdot 4^{-2}$
 $3 \cdot 16 + 0 \cdot 4 + 1 \cdot 1 + 2 \cdot 0,25 + 3 \cdot 0,0625 = 49,3125$

↪ B = base

↪ d_i = valeur du $i+1$ ème chiffre à la gauche de virgule

↪ d_j = valeur du j ème chiffre à la droite de la virgule

↪ n = nombre de chiffres entiers dans N

↪ m = nombre de chiffres fractionnaires dans N

D. SIMPLOT - Architecture des Ordinateurs



20

Système de numération (4/7)

Systèmes les plus utiles

□ B = 10 (Décimal)

• chiffres : (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

□ B = 2 (Binaire)

• chiffres : (0, 1)

□ B = 8 (Octal)

• chiffres : (0, 1, 2, 3, 4, 5, 6, 7)

□ B = 16 (Hexadécimal)

• chiffres : (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)

D. SIMPLOT - Architecture des Ordinateurs



21

Système de numération (5/7)

Conversions

- ❑ Comment passer d'un système de numération à l'autre (changement de base)?
- ❑ Algorithme 1: Définition
 - favorable pour les conversions **vers** le système décimal
 - $(N_1)_A \rightarrow (N_2)_{10}$
- ❑ Algorithme 2: Divisions et multiplications successives
 - favorable pour les conversions **à partir** du système décimal
 - $(N_1)_{10} \rightarrow (N_2)_B$
 - $(34,625)_{10} \rightarrow (?)_2$

D. SIMPLOT - Architecture des Ordinateurs



22

Système de numération (6/7)

Exemple de conversion

$(34,625)_{10} \rightarrow (100010,101)_2$

Partie entière:		Partie fractionnaire:
$34 \div 2 = 17$	r 0	$0,625 \times 2 = 1,25$
$17 \div 2 = 8$	r 1	$0,25 \times 2 = 0,5$
$8 \div 2 = 4$	r 0	$0,5 \times 2 = 1$
$4 \div 2 = 2$	r 0	
$2 \div 2 = 1$	r 0	
$1 \div 2 = 0$	r 1	

D. SIMPLOT - Architecture des Ordinateurs



23

Système de numération (7/7)

Binaire ↔ Hexadécimal

- ❑ Binaire → Hexadécimal
 - Il faut former des groupes de 4 bits en commençant au point.
 - Chaque groupe de 4 bits représente directement un chiffre hexadécimal.
- ❑ Hexadécimal → Binaire
 - Il faut convertir chaque chiffre hexadécimal à son équivalent binaire (4 bits).
- ❑ ex : 0000 ↔ 0 0101 ↔ 5 1010 ↔ A

D. SIMPLOT - Architecture des Ordinateurs



24

Nombres binaires (1/8) non-signés à virgule fixe

- ▣ C'est la base 2 avec un nombre de chiffres avant la virgule fixé et un nombre de chiffres après la virgule fixé
 - ↪ $d_{n-1}d_{n-2}d_{n-3} \dots d_2d_1d_0d_{-1} \dots d_{-m}$ avec n et m fixé
 - ↪ on « oublie » la virgule puisque l'on sait sa position...
 - ↪ cas particulier : m=0, ce sont les entiers naturels !
- ▣ $N = (00011010110)_{2(7,4)} = (?)_{10}$
 - ↪ Mot de 11 chiffres
 - ↪ Représentation en virgule fixe, avec 4 chiffres pour la partie fractionnaire
- ▣ Rappel : en base 2, un chiffre = un bit (**B**inary **d**igit)

D. SIMPLOT - Architecture des Ordinateurs



25

Nombres binaires (2/8) Exemple mots de 3 bits

	n=3 m=0	n=2 m=1	n=1 m=2	n=0 m=3
000	0	0	0	0
001	1	0,5	0,25	0,125
010	2	1	0,5	0,25
011	3	1,5	0,75	0,375
100	4	2	1	0,5
101	5	2,5	1,25	0,625
110	6	3	1,5	0,75
111	7	3,5	1,75	0,875

D. SIMPLOT - Architecture des Ordinateurs



26

Nombres binaires (3/8) Comparaison

- ▣ Comment savoir si $a < b$ où a et b sont des nombres binaires ?
- ▣ On compare bit à bit en commençant par la gauche.
- ▣ Le premier qui a un 0 alors que l'autre est à 1 est le plus petit...
 - ↪ 01001 est plus petit que 01011 (9 < 11)

000
001
010
011
100
101
110
111

27

D. SIMPLOT - Architecture des Ordinateurs



Nombres binaires (4/8)

Addition

- Pour n et m fixés (par exemple n=4 et m=0)

	1		
+	0 1 1 0	6	
	0 1 0 1	5	
=	1 0 1 1	11	

n=2 et m=2
 1,5
 1,25
 2,75

- Quels que soient n et m, c'est toujours la même technique...
- Attention au débordement : avec n=4 et m=0, on ne peut pas faire 6+11 (0110+1011=1 0001).

D. SIMPLOT - Architecture des Ordinateurs



28

Nombres binaires (5/8)

Nombres signés

- 3 façons de représenter +/- N avec n bits:

- ↪ Module et signe (noté M&S)
 - on utilise le bit le plus à gauche pour représenter le signe
 - ex : (n=4, m=0) 0011 ↔ 3 1011 ↔ -3
- ↪ Complément à 1 (noté Cà1)
 - pour un nombre négatif, on prend la représentation de la partie entière et on inverse tous les bits
 - ex : (n=4, m=0) 0100 ↔ 4 1011 ↔ -4
- ↪ Complément à 2 (noté Cà2)
 - idem, mais avant d'inverser, on soustrait 1
 - ex : (n=4, m=0) 0110 ↔ 6 1010 ↔ -6

D. SIMPLOT - Architecture des Ordinateurs



29

Nombres binaires (6/8)

Exemples de nombres signés

n=3, m=0

	M&S	Cà1	Cà2
011	3	3	3
010	2	2	2
001	1	1	1
000	0	0	0
111	-3	-0	-1
110	-2	-1	-2
101	-1	-2	-3
100	-0	-3	-4

D. SIMPLOT - Architecture des Ordinateurs



30

Nombres binaires (7/8)

Addition binaire (par complément)

- Le bit signe est traité comme tous les autres bits (on les additionne!)
- La soustraction est un cas particulier de l'addition; les nombres négatifs sont traités comme des nombres à additionner.
- Addition par Cà1 (Retenue? +1)
 - $0110 + 1110 = 1\ 0100 \rightarrow +1 \rightarrow 0101$ $6 + (-1) = 5$
 - $0001 + 1101 = 1110 \rightarrow 1110$ $1 + (-2) = -1$
- Addition par Cà2 (Directe)
 - $0110 + 1111 = 1\ 0101$ $6 + (-1) = 5$
 - $0001 + 1110 = 1111$ $1 + (-2) = -1$

D. SIMPLOT - Architecture des Ordinateurs



31

Nombres binaires (8/8)

Dépassement de capacité

- Un dépassement de capacité survient lorsque les opérandes ont le même signe et le résultat a un signe différent de celui des opérandes.
- Ex. $7+3$, mots de 4 bits, Cà2

$$\begin{array}{r}
 +7 \quad 0\ 111 \\
 +3 \quad 0\ 011 \\
 \hline
 (-5) \quad 1\ 010
 \end{array}$$

- Nom anglais : **overflow**

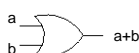
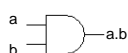
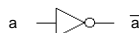
D. SIMPLOT - Architecture des Ordinateurs



32

Algèbre de Boole

- Pour les preuves, voir sur le site
- Pour pouvoir manipuler des 0 et des 1, on n'a besoin que de trois opérations :
 - Fonction **négation** (complémentation) « **NON** » (« NOT »)
 - noté avec une barre
 - $0 = 1$ et $1 = 0$
 - Fonction **conjonction** « **ET** » (« AND »)
 - noté « . »
 - $0.0 = 0.1 = 1.0 = 0$ $1.1 = 1$
 - Fonction **disjonction** « **OU** » (« OR »)
 - noté « + »
 - $0+0 = 0$ $0+1 = 1+0 = 1+1 = 1$



D. SIMPLOT - Architecture des Ordinateurs

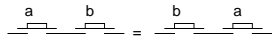


33

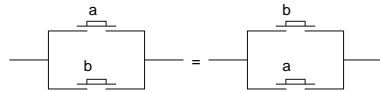
Axiomes de base (1/4)

Commutativité :

$a \cdot b = b \cdot a$



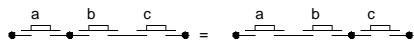
$a + b = b + a$



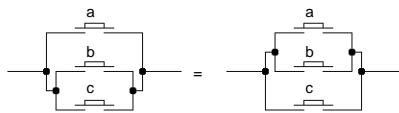
Axiomes de base (2/4)

Associativité

$a \cdot (b \cdot c) = (a \cdot b) \cdot c$



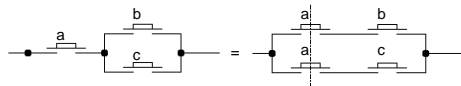
$a + (b + c) = (a + b) + c$



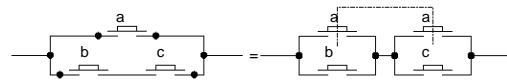
Axiomes de base (3/4)

Distributivité :

$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$



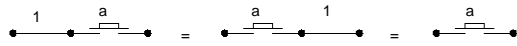
$a + (b \cdot c) = (a + b) \cdot (a + c)$



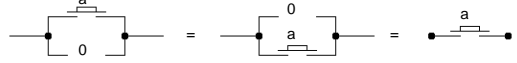
Axiomes de base (4/4)

Éléments neutres

☞ $1.a = a.1 = a$



☞ $0 + a = a + 0 = a$



Complément

☞ $a.\bar{a} = 0$

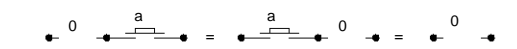
$a + \bar{a} = 1$



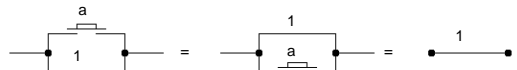
Propriétés (1/2)

Élément absorbant :

☞ $a.0 = 0.a = 0$



☞ $a + 1 = 1 + a = 1$



Absorption :

☞ $a.(a + b) = a$

$a + (a.b) = a$



Propriétés (2/2)

Idempotence :

☞ $a.a = a$

$a + a = a$

Involution :

☞ $\overline{\overline{a}} = a$

Théorème de De Morgan :

☞ $\overline{a.b} = \bar{a} + \bar{b}$

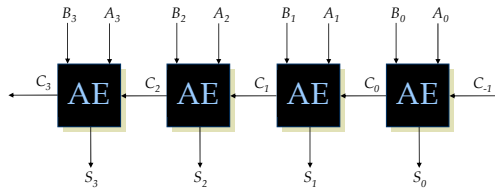
$\overline{a + b} = \bar{a} . \bar{b}$

Exercice : montrer ces propriétés à partir des axiomes...



Additionneur

- ▣ Additionneur parallèle de 4 bits
- ▣ AE = Additionneur Élémentaire



D. SIMPLOT - Architecture des Ordinateurs



40

Additionneur élémentaire (1/2)

- ▣ La table de vérité d'un additionneur élémentaire est

Ai	Bi	Ci-1	Ci	Si
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

- ▣ le nombre de 1 (en entrée) est impair \Leftrightarrow $S_i=1$

$\Rightarrow S_i = A_i \oplus B_i \oplus C_{i-1}$

- ▣ le nombre de 1 est supérieur (strictement) à 1 $\Leftrightarrow C_i=1$

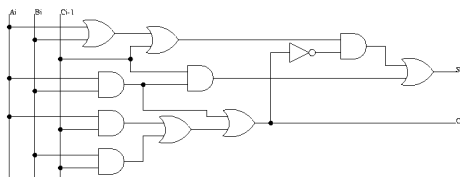
$\Rightarrow C_i = A_i B_i + A_i C_{i-1} + B_i C_{i-1}$

D. SIMPLOT - Architecture des Ordinateurs



41

Additionneur élémentaire (2/2)



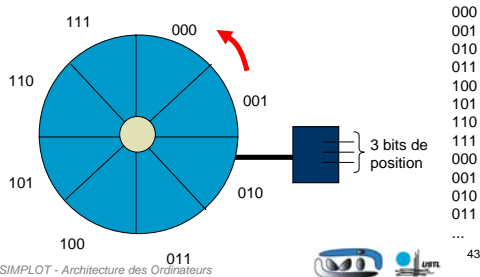
D. SIMPLOT - Architecture des Ordinateurs



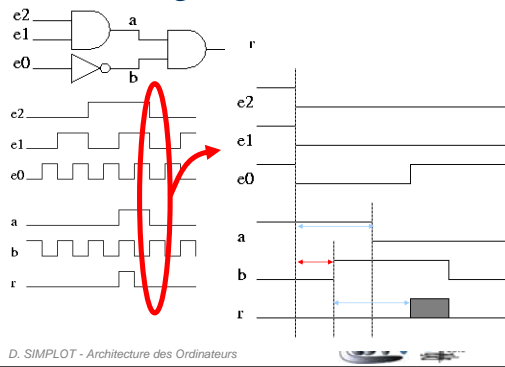
42

Autre codages des entiers (1/3)

- Considérons un disque dur à 8 secteurs :



Autre codages des entiers (2/3)



Autre codages des entiers (3/3)

- Code de Gray ou code réfléchi :
 - Pour passer d'une valeur à la suivante, on ne change qu'un bit...

0	000	0	000
1	001	1	001
2	011	2	011
3	010	3	010
4	110	4	110
5	111	5	111
6	101	6	101
7	100	7	100

En conclusion

- En TD :
 - arithmétique et logique binaire
- Les TP ne commencent pas cette semaine !







Architecture des Ordinateurs

Partie I : Concepts de Base

2. Composants élémentaires

David Simplot
simplot@fil.univ-lille1.fr



Objectifs

- Connaître les circuits élémentaires :
 - ↪ Fonctions arithmétiques : additionneurs, comparateurs
 - ↪ Fonctions combinatoires : codeurs, décodeurs, multiplexeurs, démultiplexeurs



D. SIMPLOT - Architecture des Ordinateurs



50

Au sommaire...

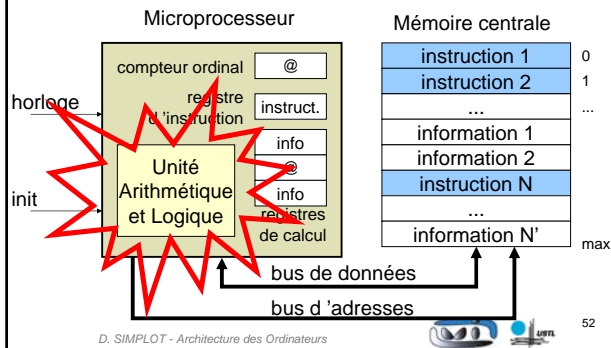
- **Circuits arithmétiques**
 - ↪ **additionneurs**, comparateurs
- Circuits combinatoires
 - ↪ codeurs, décodeurs, sélecteurs, multiplexeurs, démultiplexeurs
- Bascules, registres & mémoires
- Conclusion

D. SIMPLOT - Architecture des Ordinateurs



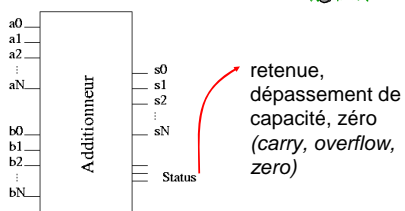
51

Additionneur (1/7)



Additionneurs (2/7)

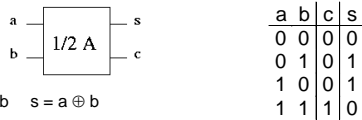
But :



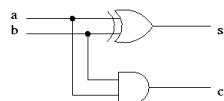
Additionneurs (3/7)

Demi-additionneur

Demi-additionneur :

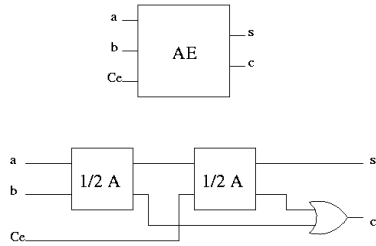


a	b	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Additionneurs (4/7)

Additionneur élémentaire



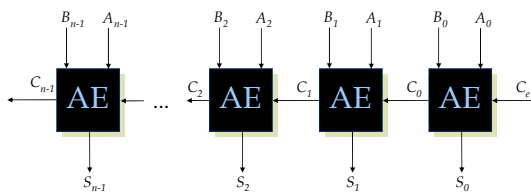
D. SIMPLOT - Architecture des Ordinateurs



55

Additionneurs (5/7)

▣ Additionneur n bits :



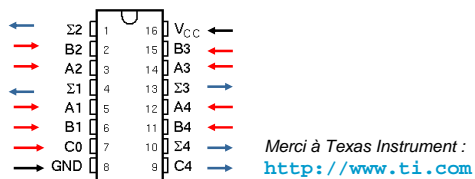
D. SIMPLOT - Architecture des Ordinateurs



56

Additionneurs (6/7)

▣ Il existe des circuits intégrés « additionneurs » 1 bit, 2 bits, 4 bits, etc... Le circuit 54/283 est un additionneur 4 bits :



D. SIMPLOT - Architecture des Ordinateurs



57

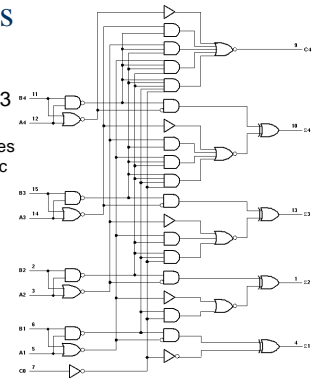
Additionneurs (7/7)

■ Circuit du 54 / 75283

↳ ce n'est pas une association d'AE, c'est un additionneur avec anticipation de retenue...



TEXAS INSTRUMENTS

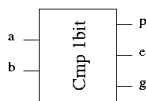


D. SIMPLOT - Architecture des Ordinateurs

58

Comparateurs (1/4)

■ Comparateur 1 bit :



a	b	p	e	g
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

↳ p = « a plus petit que b » = $\bar{a}.b$

↳ e = « a égal b » = $a \otimes b$ (le XNOR/coïncidence/équivalence)

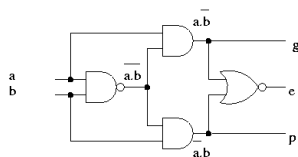
↳ g = « a plus grand que b » = $a.b$

D. SIMPLOT - Architecture des Ordinateurs

59

Comparateurs (2/4)

■ Circuit du comparateur 1 bit :



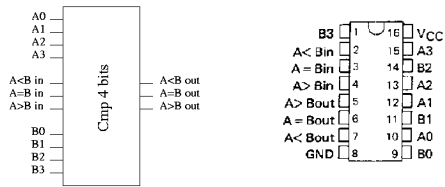
■ On peut utiliser les comparateurs 1 bit pour faire des comparateurs 2 bits. Puis on peut utiliser les comparateurs 2 bits pour faire des comparateurs 4 bits. Etc...

D. SIMPLOT - Architecture des Ordinateurs

60

Comparateurs (3/4)

- On peut également utiliser des circuits comparateurs dits « associatifs ». Par exemple, le SN54 / 7485 est un comparateur 4 bits associatif :



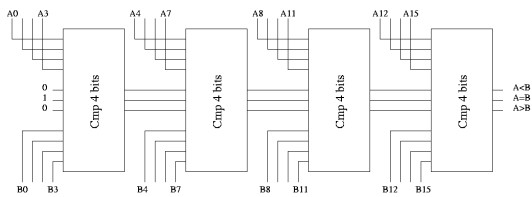
D. SIMPLOT - Architecture des Ordinateurs



61

Comparateurs (4/4)

- Application : construction d'un comparateur 16 bits :



D. SIMPLOT - Architecture des Ordinateurs

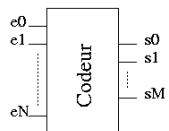


62

Codeurs (1/2)

- Code une entrée à N entrée où il n'y a qu'un seul 1 en binaire.

- $M = \log_2(N+1) - 1$
- « codeur N+1 vers M+1 »



D. SIMPLOT - Architecture des Ordinateurs

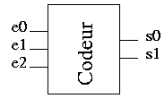


63

Codeurs (2/2)

- Codeur 3 vers 2 : $N=2, M=1$

e2	e1	e0	s0	s1
0	0	0	*	*
0	0	1	0	0
0	1	0	0	1
0	1	1	*	*
1	0	0	1	0
1	0	1	*	*
1	1	0	*	*
1	1	1	*	*



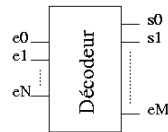
D. SIMPLOT - Architecture des Ordinateurs



64

Décodeurs (1/4)

- Décode un mot binaire
 - $2^{N-1} \leq M < 2^N$
 - $s_J = 1$ ssi $(e_N \dots e_1 e_0)_2 = J$
 - « décodeur $N+1$ vers $M+1$ »



- Exemple :

- $N=1, M=2$
- « décodeur 2 vers 3 »

e1	e0	s0	s1	s2
0	0	1	0	0
0	1	0	1	0
1	0	0	0	1
1	1	0	0	0

D. SIMPLOT - Architecture des Ordinateurs

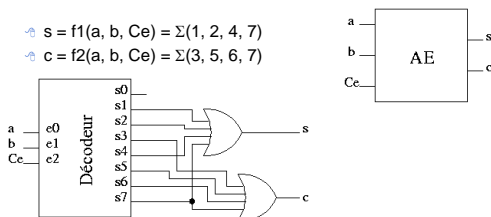


65

Décodeurs (2/4)

- Toute fonction logique $f(x_0, x_1, \dots, x_N)$ peut être réalisée avec un décodeur.
- Exemple l'additionneur élémentaire :

- $s = f_1(a, b, Ce) = \Sigma(1, 2, 4, 7)$
- $c = f_2(a, b, Ce) = \Sigma(3, 5, 6, 7)$



D. SIMPLOT - Architecture des Ordinateurs

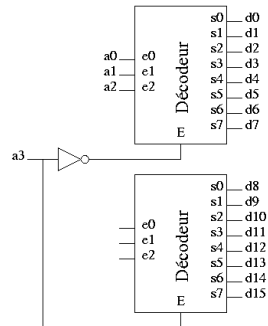


66

Décodeurs (3/4)

- On ajoute une entrée « Enabled »
 - si $E = 0$ alors toutes les sorties sont à 0
 - permet de construire de plus « gros » décodeurs...

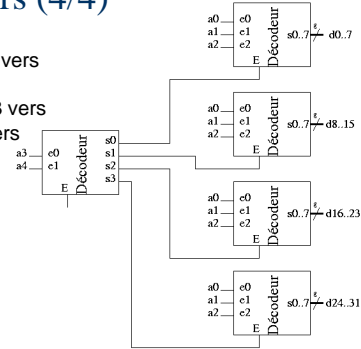
Décodeur 4 vers 16 à partir de décodeur 3 vers 8



D. SIMPLOT - Architecture des Ordinateurs

Décodeurs (4/4)

- Décodeur 5 vers 32 avec des décodeurs 3 vers 8 (et un 2 vers 4)...

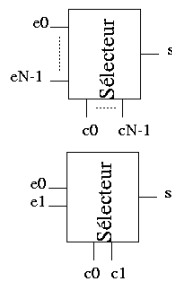


D. SIMPLOT - Architecture des Ordinateurs

Sélecteurs

- Un sélecteur prend N entrées ($e0..N-1$) et N commandes ($c0..N-1$) et copie l'entrée J si CJ est la seule à être à 1.
 - $S = 1$ ssi $EJ = 1$ et CJ est la seule commande à 1
- Exemple à 2 entrées :

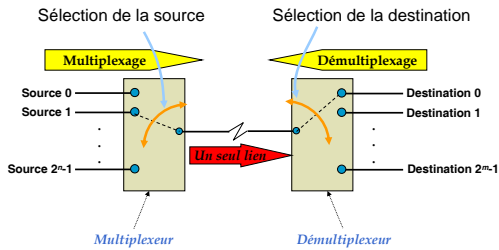
$c0$	$c1$	s
0	0	*
0	1	$e1$
1	0	$e0$
1	1	*



D. SIMPLOT - Architecture des Ordinateurs

Multiplexeurs et Démultiplexeurs

Utilisation :



D. SIMPLOT - Architecture des Ordinateurs

70

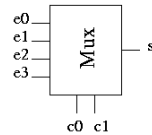
Multiplexeurs (1/2)

Le multiplexeur (**MUX**) sélectionne, à l'aide de N entrées de commande ($c0..n-1$), **une** des 2^N entrées d'information ($e0..2^N-1$) et la dirige à la sortie.

→ La sortie est égale à 1 ssi l'entrée ($c0c1...cN-1$)₂ est vraie.

Exemple: Multiplexeur 4 à 1.

c0	c1	s
0	0	e0
0	1	e1
1	0	e2
1	1	e3



On utilise un *décodeur* et un *sélecteur*...

D. SIMPLOT - Architecture des Ordinateurs

71

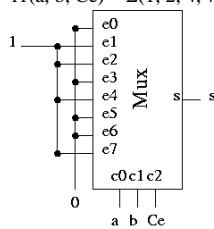
Multiplexeurs (2/2)

Synthèse de fonctions logiques avec un multiplexeur

Exemple de l'AE... (encore !)

a	b	Ce	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$s = f1(a, b, Ce) = \Sigma(1, 2, 4, 7)$$



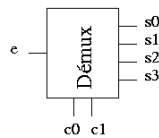
D. SIMPLOT - Architecture des Ordinateurs

72

Démultiplexeurs

- Opérateur inverse du multiplexeur. Il distribue le bit e reçu en entrée unique vers l'une des 2^N sorties possibles D (N : nombre d'entrées de sélection S).
- Exemple multiplexeur 4 sorties :

c0	c1	s0	s1	s2	s3
0	0	e	0	0	0
0	1	0	e	0	0
1	0	0	0	e	0
1	1	0	0	0	e



- C'est un décodeur avec e comme « enabled ».

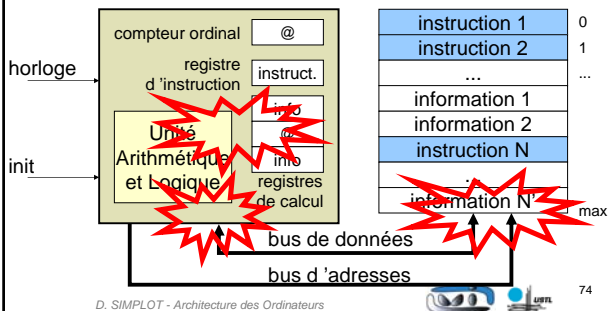
D. SIMPLOT - Architecture des Ordinateurs

73

Multiplexeurs et Démultiplexeurs

Microprocesseur

Mémoire centrale

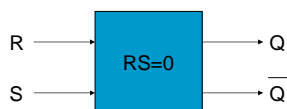


D. SIMPLOT - Architecture des Ordinateurs

74

Bascules, registres et mémoires (1/3)

- Bascule (bistable/flip-flop)
 - = moyen de mémoriser de l'information
- Différents type de bascules
 - RS=0, RS=0, RST, D, JK, JKT, RSME, JKME...
- Ex. Bascule RS=0



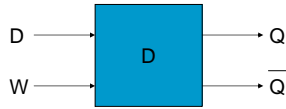
R	S	Q(t+1)
0	0	Q(t)
0	1	1
1	0	0

D. SIMPLOT - Architecture des Ordinateurs

75

Bascules, registres et mémoires (2/3)

Ex. Bascule D



→ La donnée stockée copie D si W=1 sinon, l'état de la bascule est inchangée

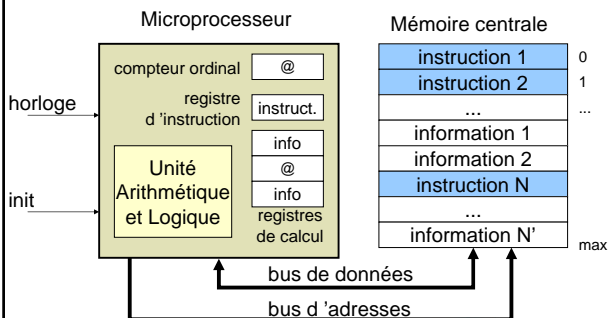


Bascules, registres et mémoires (3/3)

- ▣ Mémoires persistantes
 - ROM, EEPROM, Flash
 - Read-Only Memory
- ▣ Mémoires volatiles
 - RAM (Random Acces Memory)
 - Deux types statique ou dynamique
 - SRAM et DRAM



En conclusion



Architecture des Ordinateurs

Partie II : Microprocesseur

1. Mémoire et Entrées/Sorties

David Simplot
simplot@fil.univ-lille1.fr



Objectifs

- ☐ Comprendre les mécanismes de fonctionnement du microprocesseur
 - comment fonctionne la mémoire,
 - comment fonctionnent les entrées/sorties



D. SIMPLOT - Architecture des Ordinateurs



80

Au sommaire...

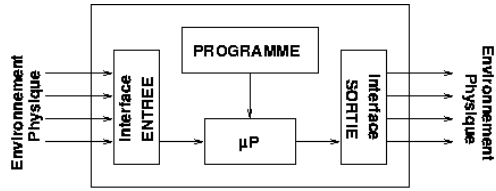
- ☐ **Machine à base de μP**
- ☐ Mémoire
- ☐ Entrées/Sorties

D. SIMPLOT - Architecture des Ordinateurs



81

Machine à μP (1/2)

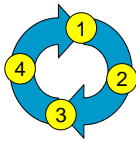


D. SIMPLOT - Architecture des Ordinateurs



82

Machine à μP (2/2)



- ▣ 1 – Lire l'instruction
- ▣ 2 – Décoder l'instruction
- ▣ 3 – Exécuter l'instruction
- ▣ 4 – Préparer l'instruction suivante

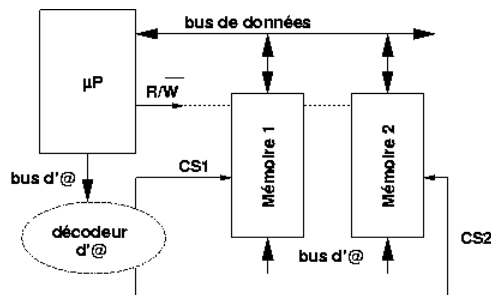
- ▣ Performance du μP dépend
 - ↪ Nombre de cycle d'horloge moyen pour faire 1 à 4
 - ↪ Vitesse de l'horloge
 - ↪ Compacité du code
- ▣ Deux approches :
 - ↪ CISC ou RISC

D. SIMPLOT - Architecture des Ordinateurs



83

Mémoire (1/7) Principe de fonctionnement



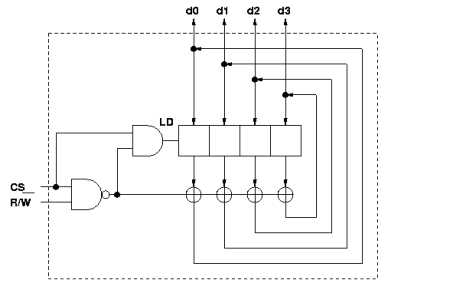
D. SIMPLOT - Architecture des Ordinateurs



84

Mémoire (2/7)

Pour une case mémoire



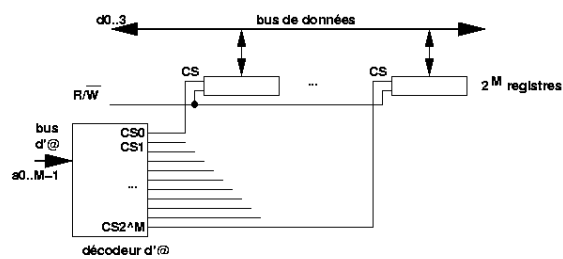
D. SIMPLOT - Architecture des Ordinateurs



85

Mémoire (3/7)

Cas mémoire 2 mots et 2^M mots



D. SIMPLOT - Architecture des Ordinateurs



86

Mémoire (4/7)

Assemblage de mémoire

□ Espace d'adressage du μP

- ↪ Déterminé par la taille du bus d'adresses
 - 10 bits $\rightarrow 2^{10}$ mots = 1024 mots = 1 **Kilomot**
 - 16 bits $\rightarrow 2^{16}$ mots = 65536 mots = 64 **Kilomot**
 - 20 bits $\rightarrow 2^{20}$ mots = 1024x1Kmot = 1 **Mégamot**
 - 30 bits $\rightarrow 2^{30}$ mots = 1024x1Mmot = 1 **Gigamot**
 - 32 bits $\rightarrow 2^{32}$ mots = 4x1Gmot = 4 **Gigamot**
- ↪ On découpe l'espace d'adressage en « plages mémoire ».

D. SIMPLOT - Architecture des Ordinateurs



87

Mémoire (5/7)

Assemblages de mémoire (suite)

- Ex. µP 8 bits (bus de données) et 23 bits d'adressage
- soit 8 Mo
- On peut découper en plages de 1 Mo

7FFFFF
700000
...
3FFFFF
300000
2FFFFF
200000
1FFFFF
100000
0FFFFF
000000

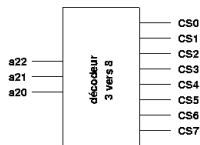
23 bits : 010 0010 1101 1001 0110 0001
a22 ... a0 88



Mémoire (6/7)

Assemblage de mémoire (suite)

- On utilise un décodeur 3 vers 8 avec a20..22

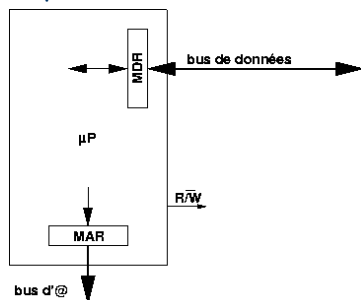


- Ex. 2 mémoires 1 Mo, 1 mémoire de 2 Mo, 1 mémoire de 512 Ko



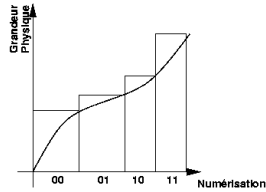
Mémoire (7/7)

Côté µP

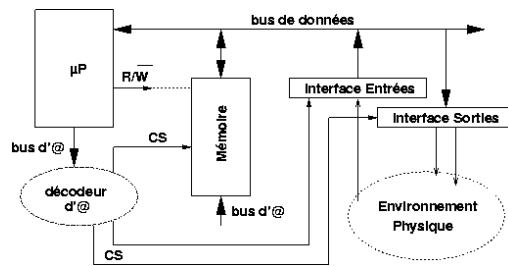


Entrées/Sorties (1/2)

- ▣ Sont accédées via l'adresse mémoire :
 - ↳ Certaines adresses sont réservées (à la conception de l'ordinateur ou de manière dynamique) aux entrées/sorties
 - ↳ Digital-Digital
 - Série ou parallèle
 - Entrées ou sorties
 - ↳ Digital-analogue
 - Sorties
 - ↳ Analogue-digital
 - entrées



Entrées/Sorties (2/2)



D. SIMPLOT - Architecture des Ordinateurs

93

Architecture des Ordinateurs

Partie II : Microprocesseur

2. Instructions machines

David Simplot
simplot@fil.univ-lille1.fr



Objectifs

- ▣ Voir les instructions élémentaires du microprocesseur
- ▣ Comment on les réalise à l'intérieur du μ P...

D. SIMPLOT - Architecture des Ordinateurs



95

Au sommaire...

- ▣ **Instructions**
- ▣ Sous-routines
- ▣ INT/DMA
- ▣ Microprogrammation

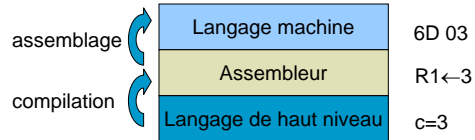
D. SIMPLOT - Architecture des Ordinateurs



96

Instructions (1/14)

- Programmes = suite d'instructions



- Trois types d'instructions

- De rangement
- De calcul
- De branchement

D. SIMPLOT - Architecture des Ordinateurs



97

Instructions (2/14)

Exemples d'instructions « machine »

R1←4	MOV R1,4 LD R1,4	Mettre la valeur 4 dans le registre R1
R1←R2	MOV R1,R2 LD R1,R2	Mettre la valeur de R2 dans le registre R1
R1←MEM(1515)	MOV R1,[1515]	Copie la valeur stockée en mémoire à l'@ 1515 dans R1
R1←R1+2	ADD R1,2	Additionne 2 à la valeur de R1 et range dans R1
R1←R1+1	INC R1	Incrémente de 1 la valeur de R1
R2←R2-R4	SUB R2,R4	Soustrait à R2 la valeur de R4 et range dans R2

D. SIMPLOT - Architecture des Ordinateurs

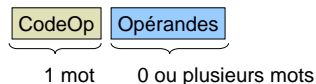


98

Instructions (3/14)

Codage des instructions

- Structure :



- La plupart du temps, certaines opérandes sont implicitement données dans le code opération. Par exemple :

- MOV R1, 4 2 mots = 89 04
- ADD R2, 3 2 mots = B2 03

D. SIMPLOT - Architecture des Ordinateurs



99

Instructions (4/14)

Codage des instructions (suite)

MOV R1, 4

- 89 04
- 1000 1001 0000 0100
- 10001 001 0000 0100
- MOV R1 4

ADD R2, 3

- B2 03
- 1011 0010 0000 0011
- 10110 010 0000 0011
- ADD R2 3

D. SIMPLOT - Architecture des Ordinateurs



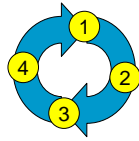
100

Instructions (5/14)

Exemple d'exécution

Mémoire :

- 1515 = 89
- 1516 = 04
- 1517 = B2
- 1518 = 03



- 1-Lire l'instruction
- 2-Décoder
- 3-Exécuter
- 4-Préparer l'instruction suivante

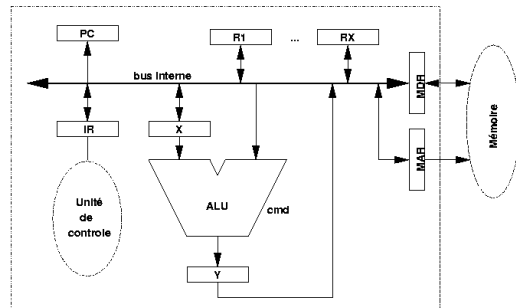
D. SIMPLOT - Architecture des Ordinateurs



101

Instructions (6/14)

Exemple d'exécution (suite)



D. SIMPLOT - Architecture des Ordinateurs



102

Instructions (7/14)

Exemple d'exécution (suite)

- ☐ Phase 1 – Lire l'instruction
 - ↪ PC vaut 1515
 - ↪ Mettre PC dans MAR et donner l'ordre de lecture
 - PCout – LDMAR
 - Read – WaitMemory
 - ↪ Placer la valeur de MDR dans IR pour que l'instruction soit décodée
 - MDRout – LDIR

 - ↪ Pb. Lors de la phase 3, on va avoir besoin de récupérer les paramètres qui sont à PC+1
 - → on anticipe

D. SIMPLOT - Architecture des Ordinateurs



103

Instructions (8/14)

Exemple d'exécution (suite)

- ☐ Phase 1 (suite)
 - ↪ On profite du fait que la lecture en mémoire est lente pour incrémenter PC
 - ↪ Dès que l'on est en phase 3, PC pointe vers le premier argument
 - ↪ Nb. S'il n'y a pas d'arguments, PC pointe vers l'instruction suivante.
 - ↪ « code » réel de la phase 1 :
 - PCout – LDMAR – LDX
 - Read – INCX – LDY
 - Yout – LDPC – WaitMemory
 - MDRout - LDIR

D. SIMPLOT - Architecture des Ordinateurs



104

Instructions (9/14)

Exemple d'exécution (suite)

- ☐ Phase 2 – décodage de l'instruction
 - ↪ Pris en charge par l'UC (unité de contrôle)
- ☐ Phase 3 – exécution
 - ↪ Deux sous-phases :
 - 3.1 Récupérer les arguments éventuellement
 - 3.2 Exécution
 - ↪ 3.1 lecture argument + incrémentation PC
 - PCout – LDMAR – LDX
 - Read – INCX – LDY
 - Yout – LDPC - WaitMemory

D. SIMPLOT - Architecture des Ordinateurs



105

Instructions (10/14)

Exemple d'exécution (suite)

- Phase 3 (suite)
 - 3.1 lecture argument + incrémentation PC
 - PCout – LDMAR – LDX
 - Read – INCX – LDY
 - Yout – LDPC - WaitMemory
 - 3.2 exécuter l'instruction (ici MOV R1,4)
 - R1out – LDX
 - MDRout – ADD – LDY
 - Yout – LDR1
- Phase 4 (préparer l'instruction suivante)
 - Rien pour l'instant ☺

D. SIMPLOT - Architecture des Ordinateurs



106

D. SIMPLOT - Architecture des Ordinateurs



107

D. SIMPLOT - Architecture des Ordinateurs



108

Architecture des Ordinateurs

Partie II : Microprocesseur

2. Instructions machines (suite)

David Simplot
simplot@fil.univ-lille1.fr

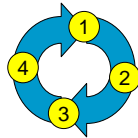


Objectifs

- ▣ Voir les instructions élémentaires du microprocesseur
- ▣ Comment on les réalise à l'intérieur du µP...

- ▣ Quatre phases du µP

- ▣ Trois types d'instructions
 - ↳ De rangement
 - ↳ De calcul
 - ↳ De branchement



D. SIMPLOT - Architecture des Ordinateurs



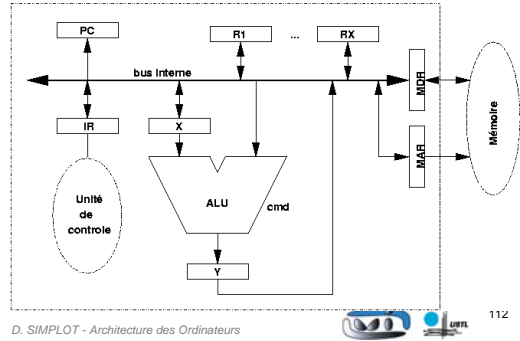
Au sommaire...

- ▣ **Instructions (suite)**
- ▣ Sous-routines
- ▣ INT/DMA
- ▣ Microprogrammation

D. SIMPLOT - Architecture des Ordinateurs



Instructions (6/14) Exemple d'exécution (suite)



Instructions (7/14) Exemple d'exécution (suite)

- ☐ Phase 1 – Lire l'instruction
 - ↪ PC vaut 1515
 - ↪ Mettre PC dans MAR et donner l'ordre de lecture
 - PCout – LDMAR
 - Read – WaitMemory
 - ↪ Placer la valeur de MDR dans IR pour que l'instruction soit décodée
 - MDRout – LDIR
 - ↪ Pb. Lors de la phase 3, on va avoir besoin de récupérer les paramètres qui sont à PC+1
 - → on anticipe

D. SIMPLOT - Architecture des Ordinateurs  113

Instructions (8/14) Exemple d'exécution (suite)

- ☐ Phase 1 (suite)
 - ↪ On profite du fait que la lecture en mémoire est lente pour incrémenter PC
 - ↪ Dès que l'on est en phase 3, PC pointe vers le premier argument
 - ↪ Nb. S'il n'y a pas d'arguments, PC pointe vers l'instruction suivante.
 - ↪ « code » réel de la phase 1 :
 - PCout – LDMAR – LDY
 - Read – INCX – LDY
 - Yout – LDPC – WaitMemory
 - MDRout - LDIR

D. SIMPLOT - Architecture des Ordinateurs  114

Instructions (9/14)

Exemple d'exécution (suite)

- ☐ Phase 2 – décodage de l'instruction
 - ↪ Pris en charge par l'UC (unité de contrôle)
- ☐ Phase 3 – exécution
 - ↪ Deux sous-phases :
 - 3.1 Récupérer les arguments éventuellement
 - 3.2 Exécution
 - ↪ 3.1 lecture argument + incrémentation PC
 - PCout – LDMAR – LDX
 - Read – INCX – LDY
 - Yout – LDPC - WaitMemory

D. SIMPLOT - Architecture des Ordinateurs



115

Instructions (10/14)

Exemple d'exécution (suite)

- ☐ Phase 3 (suite)
 - ↪ 3.1 lecture argument + incrémentation PC
 - PCout – LDMAR – LDX
 - Read – INCX – LDY
 - Yout – LDPC - WaitMemory
 - ↪ 3.2 exécuter l'instruction (ici MOV R1,4)
 - R1out – LDX
 - MDRout – ADD – LDY
 - Yout – LDR1
- ☐ Phase 4 (préparer l'instruction suivante)
 - ↪ Rien pour l'instant ☺

D. SIMPLOT - Architecture des Ordinateurs



116

Instructions (11/14)

Instructions de branchement

- ☐ Trois types de branchement
 - ↪ Branchements simples
 - ↪ Branchements conditionnels
 - ↪ Branchements sous-routines
- ☐ Deux types de références
 - ↪ Absolu ou relatif
 - ↪ En relatif, on ne donne pas une adresse mais un déplacement...
- ☐ Branchements simples :
 - ↪ JMP 1789 ou JP 1789
 - ↪ C'est équivalent à un MOV dans PC

D. SIMPLOT - Architecture des Ordinateurs

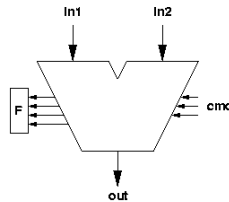


117

Instructions (12/14)

Instructions de branchement (suite)

- ▣ Branchement conditionnels
 - ↪ Sur quoi porte la condition ?
 - ↪ Lorsque l'on fait une opération arithmétique et logique, l'ALU positionne un registre de flags !



Instructions (13/14)

Instructions de branchement (suite)

- ▣ Chaque bit du registre F est un « flag »
 - ↪ Ex :
 - Z le résultat est zéro
 - N le résultat est négatif
 - V l'opération a engendré un dépassement de capacité
 - Etc.
- ▣ La condition de saut porte sur les flags
 - ↪ JZ 1789
 - Saute à l'adresse 1789 si Z est positionné



Exemple 1

- ▣ En C :


```
if ( cpt == 10 )
    cpt = 0;
else
    cpt++;
```
- ▣ La variable cpt est soit en mémoire soit dans un registre (ici on suppose que c'est dans R3).


```
1515: CMP R3, 0A
1517: JNZ 1525 ; partie else
1520: MOV R3, 0
1522: JMP 1526 ; suite du programme
1525: INC R3
1526: ...
```

Exemple 2

- En C :


```
for ( i=0 ; i<15 ; i++)
  a[i] = a[i] + i;
```
- i est dans le registre R1, a est un tableau de bytes dont l'adresse de début est dans R2


```
1515: MOV R1, 00      ; initialisation boucle
1517: MOV R3, R2
1518: CMP R1, 0F      ; début boucle
1520: JGE 1530
1523: MOV R4, [R3]
1524: ADD R4, R1      ; a[i] = a[i] + i
1525: INC R1
1526: INC R3
1527: JMP 1518
1530: ...
```

D. SIMPLOT - Architecture des Ordinateurs



121

Instructions (14/14) Modes d'adressage

- Valeur immédiate
 - ↪ MOV R1, 3E
- Valeur d'un registre
 - ↪ MOV R1, R2
- Adressage direct
 - ↪ MOV R1,[1515]
- Adressage indirect
 - ↪ MOV R1,[R2]
- Adressage indexé
 - ↪ MOV R1,[1515+R2]

D. SIMPLOT - Architecture des Ordinateurs



122

Sous-routines (1/7)

- Instruction CALL ou JSR
 - ↪ C'est une instruction de branchement
- ```

...
1492: CALL 1789
1495: ...
...
1515: CALL 1789
1518: ...
...
1789: ...
...
1815: RET
```
- ↪ Pb. Comment savoir si l'on revient en 1495 ou 1518 ?
  - ↪ Quels sont les registres que l'on peut utiliser dans la sous-routines ?

D. SIMPLOT - Architecture des Ordinateurs



123

---

---

---

---

---

---

---

---

## Sous-routines (2/7)

### Adresse de retour

- On utilise une pile

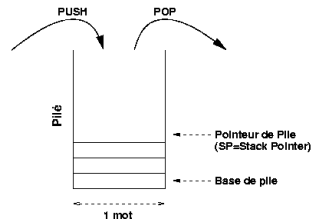
→ = LIFO

→ Deux instructions :

- push
- pop

→ On utilise un registre pointeur de pile SP

- push → on décrémente SP
- pop → on incrémente SP



D. SIMPLOT - Architecture des Ordinateurs



124

---

---

---

---

---

---

---

---

## Sous-routines (3/7)

### Adresse de retour (suite)

- Après la lecture de l'instruction CALL, PC contient l'adresse de retour

- Lors du CALL :

→ On « pousse » PC

→ On met l'adresse de la sous-routine dans PC

- Lors du RET

→ On « pope » l'adresse de retour et on la met dans PC



**La sous-routine doit rendre la pile « propre », i.e. autant de push que de pop...**

D. SIMPLOT - Architecture des Ordinateurs



125

---

---

---

---

---

---

---

---

## Sous-routine (4/7)

### Registres utilisables

- Pour les registres...

→ La sous-routine sauvegarde dans la pile les valeurs des registres qu'elle va utiliser

```
1789: push R1
 push R2
 push R5
 mov R1, 5
 ...
 pop R5
 pop R2
 pop R1
1815: ret
```

D. SIMPLOT - Architecture des Ordinateurs



126

---

---

---

---

---

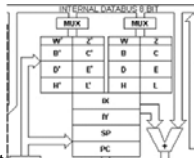
---

---

---

## Sous-routine (5/7) Registres utilisables (suite)

- Autre méthode :
  - Le  $\mu P$  dispose de plusieurs « banques » de registres



Z80  
de Zilog

- Pb. de cette méthode :
  - Les paramètres de la routine ne peuvent être passés via les registres
  - Nombre d'appels imbriqués limité par le nombre de banques

D. SIMPLOT - Architecture des Ordinateurs



127

---

---

---

---

---

---

---

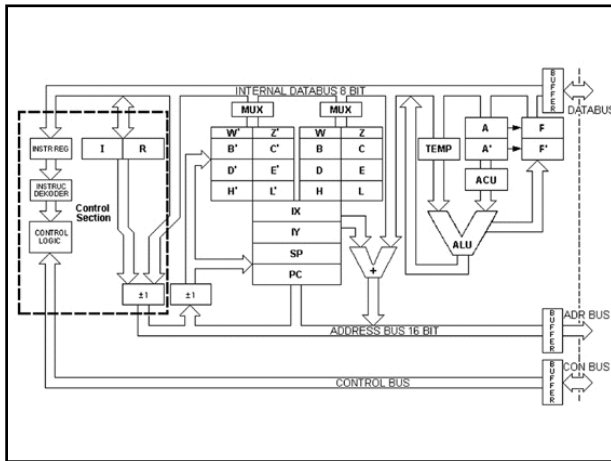
---

---

---

---

---




---

---

---

---

---

---

---

---

---

---

---

---

## Sous-routines (6/7) Passage de paramètres

- Deux possibilités
  - Registres
    - Sauf si banques de registres
  - Pile

```

push R3 ; argument 1
push R7 ; argument 2
1515: call 1789
pop R7
pop R3

1789: push R1
 push R2
 push R3
 ...
 ret

• Argument 2 en SP+6 - Argument 1 en SP+7

```

Obligatoire pour rendre la pile « propre »

D. SIMPLOT - Architecture des Ordinateurs



129

---

---

---

---

---

---

---

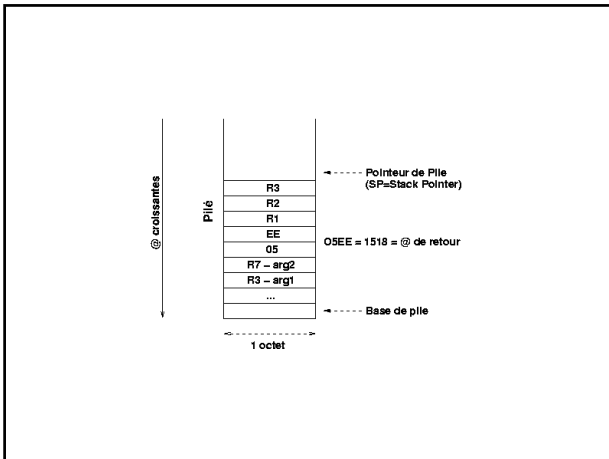
---

---

---

---

---




---

---

---

---

---

---


---

---

## Sous-routines (7/7)

### Passage de paramètres (suite)

- ❑ Inconvénients des deux techniques
  - Par registres :
    - Nombre de paramètres limité
  - Par pile :
    - Coûteux pour de petites fonctions
    - → technique de compilation *in-lining*

D. SIMPLOT - Architecture des Ordinateurs  131

---

---

---

---

---

---

---

---

D. SIMPLOT - Architecture des Ordinateurs  132

---

---

---

---

---

---

---

---

# Architecture des Ordinateurs

## Partie II : Microprocesseur

### 3. Interruptions et DMA

David Simplot  
simplot@fil.univ-lille1.fr



Licence Maîtrise d'Informatique de Lille

---

---

---

---

---

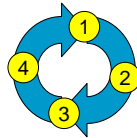
---

---

---

## Objectifs

- ▣ 1. Mémoire et entrées/sorties
  - ▣ 2. Instructions machines
  - ▣ **3. Interruptions et DMA**
  - ▣ 4. Microprogrammation
- 
- ▣ Gestion des interruptions ?
  - ▣ Quatre phases du µP
    - 1 = Lire
    - 2 = Décoder
    - 3 = Exécuter
      - Args + exécution
    - 4 = Préparer l'instruction suivante
      - ? → interruptions



D. SIMPLOT - Architecture des Ordinateurs



134

---

---

---

---

---

---

---

---

## Au sommaire...

- ▣ **Introduction**
- ▣ Interruptions matérielles
- ▣ Interruptions logicielles
- ▣ Direct Access Memory
- ▣ Exemple d'implémentation

D. SIMPLOT - Architecture des Ordinateurs



135

---

---

---

---

---

---

---

---

## Introduction (1/2)



- ▣ INT / DMA ...
- ▣ Késako ?
  - ↳ INT = Interruptions
  - ↳ DMA = Direct Memory Access
    - Accès direct en mémoire
- ▣ Mécanisme d'interruption
  - ↳ Déroulements
    - Lors de l'exécution des instructions
  - ↳ Interruptions matérielles
    - Gestion des périphériques
  - ↳ Interruptions logicielles
    - Appels système

D. SIMPLOT - Architecture des Ordinateurs



136

---

---

---

---

---

---

---

---

---

---

## Introduction (2/2)

- ▣ Déroulements :
  - ↳ Débordement numérique
  - ↳ Division par zéro
  - ↳ Non-reconnaissance d'une instruction
  - ↳ Accès à la mémoire illégale
  - ↳ ...
- ▣ Interruptions matérielles
  - ↳ Demande de données d'un périphérique
  - ↳ Alerte provenant du matériel (batterie faible)
- ▣ Interruptions logicielles
  - ↳ Appels de fonctions du système d'exploitation
  - ↳ Fonctionne sur le même principe que les deux précédentes, mais sont générées par une instruction spécifique : INT

D. SIMPLOT - Architecture des Ordinateurs



137

---

---

---

---

---

---

---

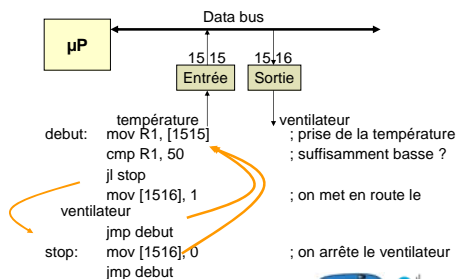
---

---

---

## Interruptions matérielles (1/7)

- ▣ Ex. Programmation de régulation de la température



D. SIMPLOT - Architecture des Ordinateurs



138

---

---

---

---

---

---

---

---

---

---

## Interruptions matérielles (2/7)

- On introduit un pb supplémentaire :

- Prise en compte du niveau de la batterie
- Première méthode → scrutation active (comme pour la t°)
 

```

debut: mov R1, [1515] ; prise de la température
 cmp R1, 50 ; suffisamment basse ?
 jl stop
 mov [1516], 1 ; on met en route le
 ventilateur
 jmp suite
stop: mov [1516], 0 ; on arrête le ventilateur
suite: mov R1, [1414] ; niveau de la batterie
 cmp R1, 5 ; moins de 5%
 jg debut
 call AURG ; arrêt d'urgence
 jmp debut

```

D. SIMPLOT - Architecture des Ordinateurs



139

---

---

---

---

---

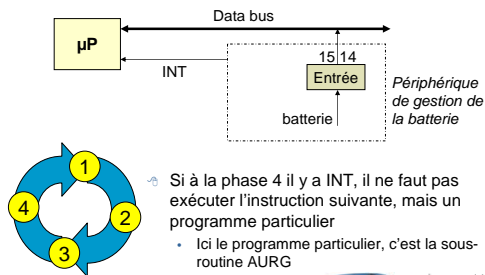
---

---

---

## Interruptions matérielles (3/7)

- Deuxième méthode :



D. SIMPLOT - Architecture des Ordinateurs



140

---

---

---

---

---

---

---

---

## Interruptions matérielles (4/7)

```

debut: mov R1, [1515]
 cmp R1, 50
 jl stop
 mov [1516], 1
 jmp debut
stop: mov [1516], 0
 jmp debut
...
...
AURG: ...
...
ret

```

INT

Pb. du registre F lors du déclenchement d'une interruption

D. SIMPLOT - Architecture des Ordinateurs



141

---

---

---

---

---

---

---

---

## Interruptions matérielles (5/7)

- ▣ Suivant le  $\mu P$ , il peut y avoir plusieurs lignes d'interruptions
  - ↳ INT1, INT2, ... (généralement 2 ou 3)
- ▣ Il y existe des interruptions non-masquable
  - ↳ NMI = Non-Maskable Interruption



- ▣ Les autres interruptions sont donc masquables
  - ↳ Instructions DI (disable int.) et EI (enable int.)
  - ↳ Utile pour les sections critiques et initialisations

D. SIMPLOT - Architecture des Ordinateurs



142

---

---

---

---

---

---

---

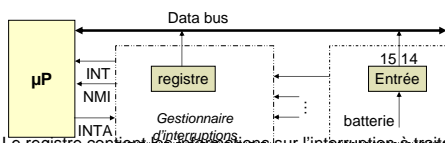
---

---

---

## Interruptions matérielles (6/7)

- ▣ Comment gérer tous les périphériques avec seulement 2 ou 3 signaux d'interruptions ?



- ▣ Le registre contient les informations sur l'interruption à traiter
  - ↳ La première instruction du programme de traitement des interruptions va donc être de lire ce registre sur le bus de données.
  - ↳ Pour éviter les conflits sur le bus de données, cette information n'est mise sur le bus que sur envoi du signal INTA

D. SIMPLOT - Architecture des Ordinateurs



143

---

---

---

---

---

---

---

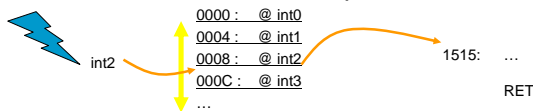
---

---

---

## Interruptions matérielles (7/7)

- ▣ L'information donnée par le gestionnaire d'interruption fait référence à un **vecteur d'interruptions**



- ▣ Au choix :
  - ↳ Implémentation soft :
    - Un seul branchement d'INT lors de la phase 4 qui lui va consulter le vecteur d'interruption
  - ↳ Implémentation hard :
    - Lors de la phase 4, on consulte le registre d'interruption et on fait le bon branchement...

D. SIMPLOT - Architecture des Ordinateurs



144

---

---

---

---

---

---

---

---

---

---



## Interruptions logicielles

- ▣ Identique à une interruption matérielle, mais elle est déclenchée par une instruction :
  - INT X
  - où X est le numéro de l'interruption que l'on veut déclencher...
  - On se réfère toujours au vecteur d'interruption (qui peut être changé dynamiquement).
- ▣ Ex. avec l'architecture intel :
  - INT 10h : appels BIOS
    - Basic Input/Output System
  - INT 21h : appels fonctions « système » (ici DOS)

D. SIMPLOT - Architecture des Ordinateurs



145

---

---

---

---

---

---

---

---

---

---

## Direct memory access (1/2)

- ▣ Avec les interruptions, on a transformé les attentes actives (e.g. par scrutation) en attente passive :
  - Le processeur peut faire une autre tâche en attendant le périphérique
- ▣ Mais pour le transfert d'informations, c'est le  $\mu P$  qui envoie sert de passerelle entre le périphérique et la mémoire...
- ▣ Solution : DMA
  - Accès Direct à la Mémoire
  - Direct Memory Access

D. SIMPLOT - Architecture des Ordinateurs



146

---

---

---

---

---

---

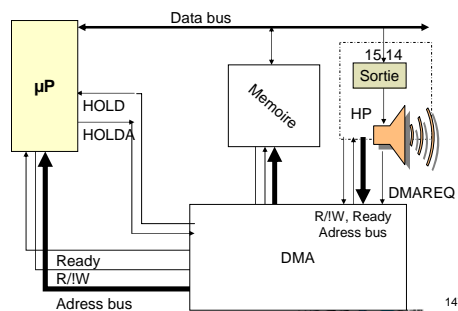
---

---

---

---

## Direct memory access (2/2)



D. SIMPLOT - Architecture des Ordinateurs



147

---

---

---

---

---

---

---

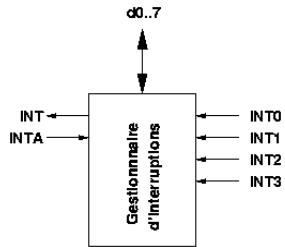
---

---

---

## Exemple d'implémentation (1/7)

- Gestionnaire d'interruptions
  - ↳ 4 périphériques



D. SIMPLOT - Architecture des Ordinateurs



148

---

---

---

---

---

---

---

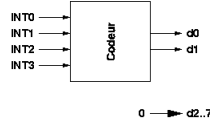
---

---

---

## Exemple d'implémentation (2/7)

- Signal INT
  - ↳  $INT = INT0 + INT1 + INT2 + INT3$
- Sur réception d'un signal INTA
  - ↳ Positionner d0..7 le numéro de l'interruption levée
  - ↳ Utilisation d'un codeur



| INT3..0 | d1..0 |
|---------|-------|
| 0000    | ??    |
| 0001    | 00    |
| 0010    | 01    |
| 0011    | ??    |
| 0100    | 10    |
| 0101    | ??    |
| 0110    | ??    |
| 0111    | ??    |
| 1000    | 11    |
| 1001    | ??    |
| 1010    | ??    |
| 1011    | ??    |
| 1100    | ??    |
| 1101    | ??    |
| 1110    | ??    |
| 1111    | ??    |

D. SIMPLOT - Architecture des Ordinateurs



149

---

---

---

---

---

---

---

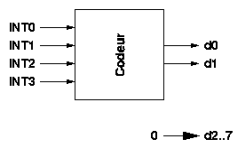
---

---

---

## Exemple d'implémentation (3/7)

- On instaure une priorité



| INT3..0 | d1..0 |
|---------|-------|
| 0000    | 00    |
| 0001    | 00    |
| 0010    | 01    |
| 0011    | 00    |
| 0100    | 10    |
| 0101    | 00    |
| 0110    | 01    |
| 0111    | 00    |
| 1000    | 11    |
| 1001    | 00    |
| 1010    | 01    |
| 1011    | 00    |
| 1100    | 10    |
| 1101    | 00    |
| 1110    | 01    |
| 1111    | 00    |

D. SIMPLOT - Architecture des Ordinateurs



150

---

---

---

---

---

---

---

---

---

---

## Exemple d'implémentation (4/7)

- ☐ Comment calculer d0 et d1
  - ↪ Utilisation d'un programme ad'hoc
    - Voir sur le site du cours
    - Fonction à 4 variables logiques
    - Pour d0 :
      - 5 points vrais (le reste à faux) : 2, 6, 8, 10 et 14
      - $d0 = (INT3 \cdot INT2 + INT1) \cdot INTO$
    - Pour d1 :
      - 3 points vrais (le reste à faux) : 4, 8 et 12
      - $d1 = (INT3 + INT2) \cdot INT1 \cdot INTO$




---

---

---

---

---

---

---

---

---

---

```

simplot@brunehaut--/ens/a.../old/C/opt] ./qmc
Nombre de variables de votre fonction logique :
4
Vos variables seront étiquetées de a3 à a0.
Nombre de lignes de la table de vérité : 16

Vous désirez faire la saisie par :
1. points vrais - le reste à faux,
2. points faux - le reste à vrai,
3. points vrais et points faux - le reste
 indifférent,
4. points vrais et points indifférents - le reste à
 faux,
5. points faux et points indifférents - le reste à
 vrai.
Votre choix : 1

Nombre de points vrais : 3
Nombre de points faux : 13
Nombre de points indéfinis : 0
Entrez en décimal (a0=pds faible) les points
vrais
4
8
12
Mode d'optimisation :
...

```

Etape 1 : trouver les implicants premiers

...

Liste des implicants premiers :

```
*100
1*00
```

Etape 2 : recouvrement optimal

liste des termes à valider

```
{ 0} 4 0100
{ 1} 8 1000
{ 2} 12 1100
```

```
[0] *100 ----X---|---X 3
[1] 1*00 ----|---X---X 3
```

suppression des lignes inutiles... aucune
règle 1 : IP obligatoires... [0] [1]
terminé !

f = a2!a1!a0 + a3!a1!a0

simplot@brunehaut--/ens/a.../old/C/opt]

Etape 1 : trouver les implicants premiers

...

Liste des implicants premiers :

```
*100
1*00
```

Etape 2 : recouvrement optimal

liste des termes à valider

```
{ 0} 4 0100
{ 1} 8 1000
{ 2} 12 1100
```

```
[0] *100 ----X---|---X 3
[1] 1*00 ----|---X---X 3
```

suppression des lignes inutiles... aucune
règle 1 : IP obligatoires... [0] [1]
terminé !

f = a2!a1!a0 + a3!a1!a0

simplot@brunehaut--/ens/a.../old/C/opt]

---

---

---

---

---

---

---

---

---

---

## Exemple d'implémentation (5/7)

- ☐ Problème de cette implémentation :
  - ↪ Le périphérique peut monopoliser le CPU...
  - ↪ Il faudrait mémoriser les INT au fur et à mesure qu'elles arrivent
    - => utilisation d'un buffer
- ☐ Buffer = FIFO (first in first out)
  - ↪ (rappel, la pile = LIFO)
  - ↪ Utilisation de registres chaînés les uns avec les autres




---

---

---

---

---

---

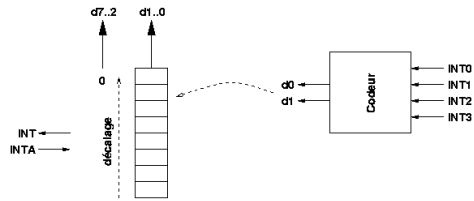
---

---

---

---

## Exemple d'implémentation (6/7)



- Pbs =
- ↳ Où stocker dans mes huit registres ?
  - ↳ Comment informer mon périphérique qu'il a été enregistré ?
  - ↳ Comment faire la différence entre une case vide et l'interruption 0 ?

D. SIMPLOT - Architecture des Ordinateurs



154

---

---

---

---

---

---

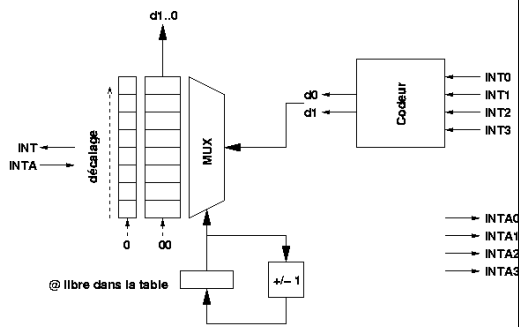
---

---

---

---

## Exemple d'implémentation (7/7)



D. SIMPLOT - Architecture des Ordinateurs



155

---

---

---

---

---

---

---

---

---

---

D. SIMPLOT - Architecture des Ordinateurs



156

---

---

---

---

---

---

---

---

---

---

# Architecture des Ordinateurs

## Partie II : Microprocesseur

### 3. Interruptions et DMA (suite)

David Simplot  
simplot@fil.univ-lille1.fr



---

---

---

---

---

---

---

---

## Au sommaire...

- ▣ Introduction
- ▣ Interruptions matérielles
- ▣ Interruptions logicielles
- ▣ Direct Access Memory
- ▣ **Exemple d'implémentation**

D. SIMPLOT - Architecture des Ordinateurs



158

---

---

---

---

---

---

---

---

## Objectifs

- ▣ Comment implémenter un gestionnaire d'interruption ?
- ▣ Transition vers le chapitre 4
  - ↪ Microprogrammation
- ▣ Unité de contrôle plus simple que celle d'un microprocesseur :
  - ↪ Automates,
  - ↪ Premier pas vers la microprogrammation...

D. SIMPLOT - Architecture des Ordinateurs



159

---

---

---

---

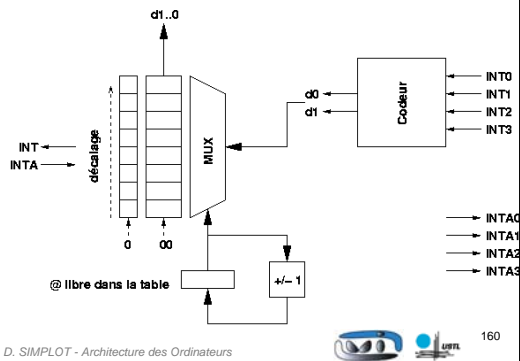
---

---

---

---

## Exemple d'implémentation (1/15)




---

---

---

---

---

---

---

---

---

---

## Exemple d'implémentation (2/15)

### (1) déterminer le comportement

- ☐ Que doit faire le gestionnaire d'interruption ?
  - ☞ Sur réception d'un signal INTx : Mémoriser dans le buffer
    - Y a-t-il une place libre ?
      - Si oui, mémoriser puis incrémenter le compteur il faut également envoyer le signal INTAx
      - Si non, rien à faire... attendre qu'une place se libère...
  - ☞ Sur réception d'un signal INTA : Présenter le numéro d'interruption
    - Attendre que INTA repasse à zéro
    - Décaler les registres du buffer et décrémenter le compteur

---

---

---

---

---

---

---

---

---

---

## Exemple d'implémentation (3/15)

### (1) déterminer le comportement (suite)

- ☐ On utilise un automate :
  -
- ☐ Pb : si le programmeur ne baisse pas tout de suite son INT, il sera mémorisé plusieurs fois...
  - ☞ On ajoute un état de temporisation...

---

---

---

---

---

---

---

---

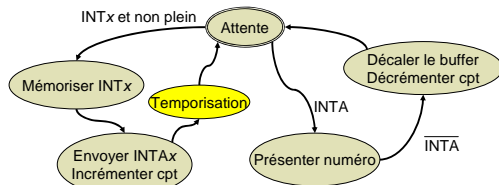
---

---

## Exemple d'implémentation (4/15)

### (1) déterminer le comportement (suite)

- ▣ Nouvel automate...



- ▣ Est-il utile de perdre un cycle d'horloge à chaque mémorisation?
  - ↪ Si il y a un INTA, on peut passer directement à son traitement...

D. SIMPLOT - Architecture des Ordinateurs



163

---

---

---

---

---

---

---

---

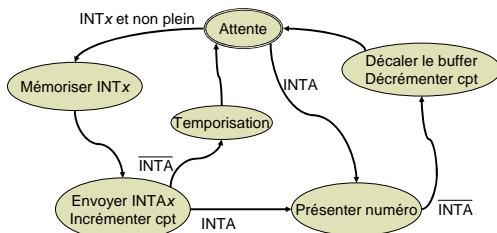
---

---

## Exemple d'implémentation (5/15)

### (1) déterminer le comportement (suite)

- ▣ On court-circuite la temporisation :



D. SIMPLOT - Architecture des Ordinateurs



164

---

---

---

---

---

---

---

---

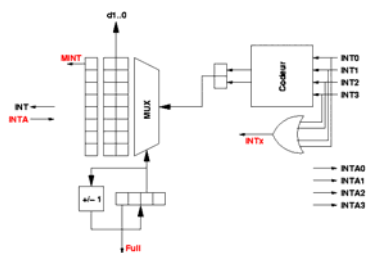
---

---

## Exemple d'implémentation (6/15)

### (2) déterminer les entrées/sorties

- ▣ Déterminer les entrées de l'unité de contrôle
  - ↪ De quoi doit-elle tenir compte ?



D. SIMPLOT - Architecture des Ordinateurs



165

---

---

---

---

---

---

---

---

---

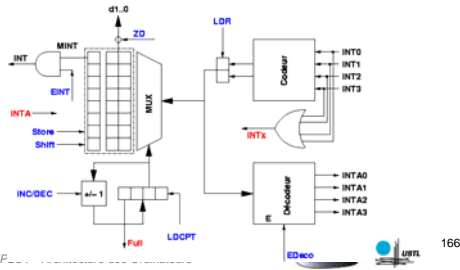
---

## Exemple d'implémentation (7/15)

### (2) déterminer les entrées/sorties (suite)

☐ Déterminer les signaux de sortie de l'UC

↳ Quels signaux doit-elle générer ?




---

---

---

---

---

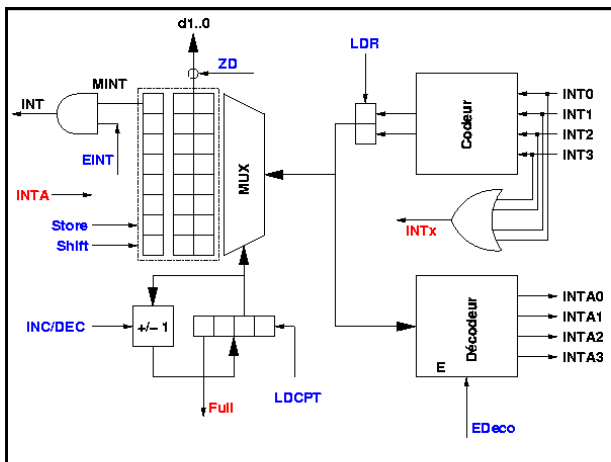
---

---

---

---

---




---

---

---

---

---

---

---

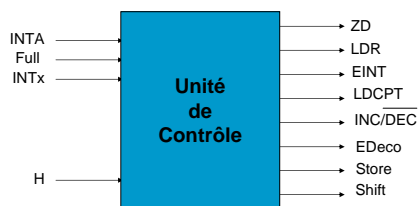
---

---

---

## Exemple d'implémentation (8/15)

### (2) déterminer les entrées/sorties




---

---

---

---

---

---

---

---

---

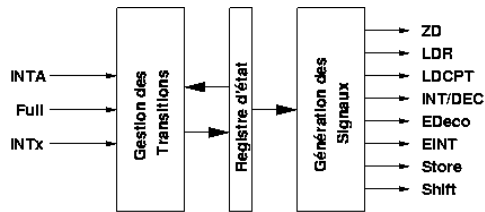
---



## Exemple d'implémentation (9/15)

### (3) choix de la technique d'implémentation

☐ Implémentation « câblée » :



D. SIMPLOT - Architecture des Ordinateurs



169

---

---

---

---

---

---

---

---

---

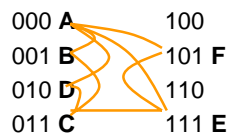
---

## Exemple d'implémentation (10/15)

### (4) Codage des états

☐ Dans notre automate, on a 6 états

- au minimum 3 bits dans le registre d'états
- Il faut attribuer une configuration de ces 3 bits pour chacun des états
  - Critère : minimiser la distance (en nombre de bits différents) entre deux états reliés entre eux...



D. SIMPLOT - Architecture des Ordinateurs



170

---

---

---

---

---

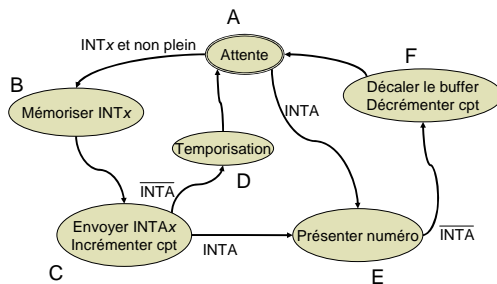
---

---

---

---

---




---

---

---

---

---

---

---

---

---

---

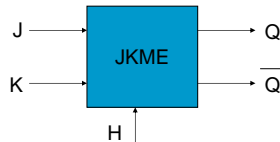
## Exemple d'implémentation (11/15)

### (5) Gestion des transitions

- Pour stocker le registre d'états, on utilise des bascules JKME
- Vues en TD

- Pour mémoire :

| J | K | Q <sub>n+1</sub> |
|---|---|------------------|
| 0 | 0 | Q <sub>n</sub>   |
| 0 | 1 | 0                |
| 1 | 0 | 1                |
| 1 | 1 | Q <sub>n</sub>   |



Mémorisation sur un front montant (ou descendant au choix)

D. SIMPLOT - Architecture des Ordinateurs



172

---

---

---

---

---

---

---

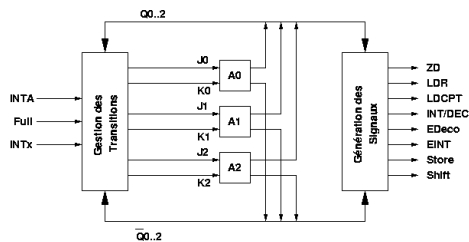
---

---

---

## Exemple d'implémentation (12/15)

### (5) Gestion des transitions (suite)



- On doit donc juste générer les J0..2 et K0..2 en fonction de Q0..2 et des entrées...

D. SIMPLOT - Architecture des Ordinateurs



173

---

---

---

---

---

---

---

---

---

---

## Exemple d'implémentation (13/15)

### (5) Gestion des transitions (suite)

| Etat | Q2..0 | INTA | Full | INTx | état suivant |
|------|-------|------|------|------|--------------|
| A    | 000   | *    | 0    | 1    | B 001        |
|      |       | 1    | 1    | *    | E 111        |
|      |       | 1    | 0    | 0    | E 111        |
|      |       | 0    | 1    | *    | A 000        |
|      |       | 0    | 0    | 0    | A 000        |
| B    | 001   | *    | *    | *    | C 011        |
| C    | 011   | 0    | *    | *    | D 010        |
|      |       | 1    | *    | *    | E 111        |
| D    | 010   | *    | *    | *    | A 000        |
| E    | 111   | 0    | *    | *    | F 101        |
|      |       | 1    | *    | *    | E 111        |
| F    | 101   | *    | *    | *    | A 000        |

D. SIMPLOT - Architecture des Ordinateurs



174

---

---

---

---

---

---

---

---

---

---

### Exemple d'implémentation (14/15) (5) Gestion des transitions (suite)

| Etat | Q2..0 | INTA | Full | INTx | état suivant | JK2   | JK1 | JK0  |    |
|------|-------|------|------|------|--------------|-------|-----|------|----|
| A    | 000   | *    | 0    | 1    | B 001        | 0*    | 0*  | 1*   |    |
|      |       |      | 1    | *    | E 111        | 1*    | 1*  | 1*   |    |
|      |       |      | 1    | 0    | 0            | E 111 | 1*  | 1*   | 1* |
|      |       |      | 0    | 1    | *            | A 000 | 0*  | 0*   | 0* |
|      |       |      | 0    | 0    | 0            | A 000 | 0*  | 0*   | 0* |
| B    | 001   | *    | *    | *    | C 011        | 0*    | 1*  | *0   |    |
| C    | 011   | 0    | *    | *    | D 010        | 0*    | *0  | *1   |    |
|      |       |      | 1    | *    | *            | E 111 | 1*  | *0   | *0 |
| D    | 010   | *    | *    | *    | A 000        | 0*    | *1  | 0*   |    |
| E    | 111   | 0    | *    | *    | F 101        | *0    | *1  | *0   |    |
|      |       |      | 1    | *    | *            | E 111 | *0  | *0   | *0 |
| F    | 101   | *    | *    | *    | A 000        | *1    | 0*  | 1751 |    |

D. SIMPLOT - Architecture des Ordinateurs




---

---

---

---

---

---

---

---

### Exemple d'implémentation (15/15) (6) Génération des signaux

- Pour chaque état, on regarde les signaux générés

D. SIMPLOT - Architecture des Ordinateurs



176

---

---

---

---

---

---

---

---

| Eta t | Q2..0 | Z D | LD R | EIN T | LDCP T | INT!/DE C | EDec o | Stor e | Shif t |
|-------|-------|-----|------|-------|--------|-----------|--------|--------|--------|
| A     | 000   | 0   | *    | 1     | 0      | *         | 0      | 0      | 0      |
| B     | 001   | 0   | 1    | 1     | 0      | 1         | 0      | 1      | 0      |
| C     | 011   | 0   | 0    | 1     | 1      | 1         | 1      | 0      | 0      |
| D     | 010   | 0   | *    | 1     | 0      | *         | 0      | 0      | 0      |
| E     | 111   | 1   | *    | 1     | 0      | 0         | 0      | 0      | 0      |
| F     | 101   | 0   | *    | 0     | 1      | 0         | 0      | 0      | 1      |

---

---

---

---

---

---

---

---

## Conclusion

- ▣ La méthode d'implémentation « câblée » des automates est propre aux  $\mu$ P de type RISC :
  - ↪ RISC = Reduced Instruction Set Computer
    - E.g. PowerPC, MIPS, ...
  - ↪ Le format des instructions est uniforme et on peut facilement « tirer » des câbles à partir du registre IR
  - ↪ La plupart des instructions se font en un cycle et sont donc facilement représentable sous forme d'automates
- ▣ Pour les CISC...
  - ↪ Complex Instruction Set Computer
    - E.g. Pentium x86, Motorola 68xxx, ...
  - ↪ Automate monstrueux  $\Rightarrow$  il faut une autre méthode...



---

---

---

---

---

---

---

---



---

---

---

---

---

---

---

---



---

---

---

---

---

---

---

---

# Architecture des Ordinateurs

## Partie II : Microprocesseur

### 4. Unité de contrôle et Microprogrammation

David Simplot  
simplot@fil.univ-lille1.fr



---

---

---

---

---

---

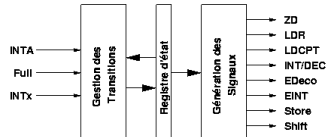
---

---

## Objectifs

Comment sont réalisées les Unités de Contrôle dans les microprocesseurs ?

- Architecture simple (e.g. type RISC)
  - ⇒ implémentation câblée
  - Automate



- Architecture + complexe ???

D. SIMPLOT - Architecture des Ordinateurs



---

---

---

---

---

---

---

---

## Au sommaire...

- Microprogrammation verticale**
- Microprogrammation horizontale
- Améliorations des performances

D. SIMPLOT - Architecture des Ordinateurs



---

---

---

---

---

---

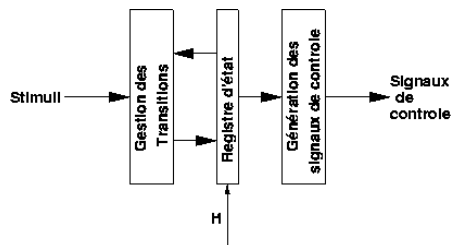
---

---

## Microprogrammation « verticale »

(1/3)

- ▣ Avec une implémentation câblée de type automate, on avait :



D. SIMPLOT - Architecture des Ordinateurs

184

---

---

---

---

---

---

---

---

---

---

## Microprogrammation « verticale »

(2/3)

- ▣ L'idée est de faciliter l'implémentation de l'automate en stockant les transitions en mémoire...

- ↪ On a vu dans l'exemple que
  - Nouvel état = f(état courant, stimuli)
- ↪ État courant + stimuli forment une @ mémoire

**Mémoire de type ROM interne à l'unité de contrôle !**



- ↪ A cette @ mémoire, est inscrit le nouvel état !
  - Plus rapide
  - Mais prend plus de place sur le chip

D. SIMPLOT - Architecture des Ordinateurs

185

---

---

---

---

---

---

---

---

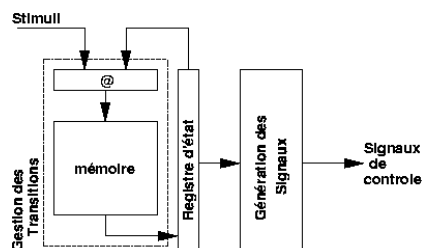
---

---

## Microprogrammation « verticale »

(3/3)

- ▣ Ceci donne :



D. SIMPLOT - Architecture des Ordinateurs

186

---

---

---

---

---

---

---

---

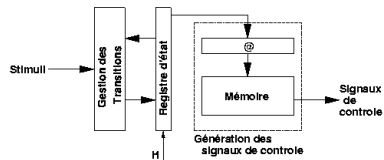
---

---

## Microprogrammation « horizontale »

(1/7)

- ▣ L'idée est de faire la même chose avec les signaux de contrôle générés à partir de l'état...
- ▣ On dispose d'une mémoire de micro-instructions
  - L'état de l'automate est l'adresse du microprogramme à exécuter
  - On parle de pointeur de microprogramme et non plus d'état de l'automate



D. SIMPLOT - Architecture des Ordinateurs

187

---

---

---

---

---

---

---

---

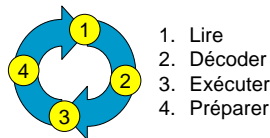
---

---

## Microprogrammation « horizontale »

(2/7)

- ▣ Dans un microprocesseur, la plupart des chemins des chemins dans l'automate sont des séquences
- ▣ Le seul choix est fait lors du décodage de l'instruction (phase 2)



D. SIMPLOT - Architecture des Ordinateurs

188

---

---

---

---

---

---

---

---

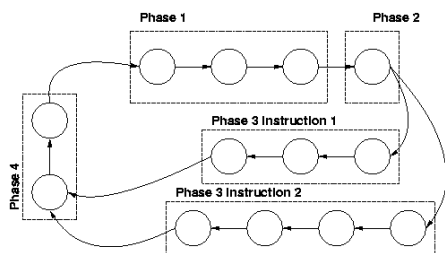
---

---

## Microprogrammation « horizontale »

(3/7)

- ▣ Automate de l'unité de contrôle d'un microprocesseur :



D. SIMPLOT - Architecture des Ordinateurs

189

---

---

---

---

---

---

---

---

---

---

## Microprogrammation « horizontale » (4/7)

Chaque ligne de la mémoire de microprogramme est une micro-instruction

- ↪ Directement reliée aux transistors générant les signaux de contrôle
- ↪ Implicitement, la micro-instruction suivante est à l'adresse suivante
- ↪ On a une colonne particulière pour dire que c'est au décodeur d'instruction de générer l'adresse de la micro-instruction suivante

---

---

---

---

---

---

---

---

---

---

## Microprogrammation « horizontale » (5/7)

Les séquences de micro-instructions sont celles que l'on a vu dans le chapitre 2 :

- ↪ Phase 1 :
  - PCout – LDMAR – LDX
  - Read – INCX – LDY
  - Yout – LDPC – WaitMemory
  - MDRout – LDIR
- ↪ Phase 2 :
  - Mettre dans le pointeur de micro-programme l'@ de la séquence correspondant à l'instruction

---

---

---

---

---

---

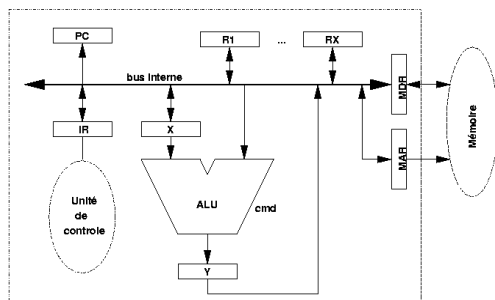
---

---

---

---

## Microprogrammation « horizontale » (6/7)




---

---

---

---

---

---

---

---

---

---



## Microprogrammation « horizontale » (7/7)

- ↪ Phase 3 : (exemple pour MOV R1, AA)
  - 3.1 lecture argument + incrémentation PC
    - PCout – LDMAR – LDX
    - Read – INCX – LDY
    - Yout – LDPC - WaitMemory
  - 3.2 exécuter l'instruction
    - R1out – LDX
    - MDRout – ADD – LDY
    - Yout – LDR1

D. SIMPLOT - Architecture des Ordinateurs



193

---

---

---

---

---

---

---

---

---

---

## Améliorations des performances (1/15) Mesure des performances

- ☐ Performance = vitesse de traitement

$$\begin{aligned} \text{Temps/t\^ache} \\ = \\ \text{instructions/t\^ache} \\ \times \\ \text{cycles/instruction} \\ \times \\ \text{temps/cycle} \end{aligned}$$

D. SIMPLOT - Architecture des Ordinateurs



194

---

---

---

---

---

---

---

---

---

---

## Améliorations des performances (2/15) Mesure des performances (suite)

$$\text{Temps/t\^ache} = \text{instructions/t\^ache} \times \text{cycles/instruction} \times \text{temps/cycle}$$

- ☐ Instructions/t\^ache

- ↪ Dépend :
  - Du jeu d'instructions (RISC/CISC)
  - Algorithme pour réaliser la tâche
  - Niveau d'optimisation du compilateur

- ☐ Cycles/instruction

- ↪ Dépend :
  - De la complexité des instructions utilisées (RISC/CISC)
  - Optimisation du compilateur (choix des instructions)

D. SIMPLOT - Architecture des Ordinateurs



195

---

---

---

---

---

---

---

---

---

---

## Améliorations des performances (3/15) Mesure des performances (suite)

Temps/tâche = instructions/tâche × cycles/instruction × temps/cycle

### Temps/cycle

- ↳ Directement dérivé de la fréquence de l'horloge
- ↳ Dépend :
  - Complexité de l'architecture
  - Technologie

---

---

---

---

---

---

---

---

---

---

## Améliorations des performances (4/15) Philosophies CISC/RISC

### CISC

- ↳ Jeu d'instructions avec un grand nombre d'instructions
  - E.g. instructions MMX
- ↳ de nombreux modes d'adressage
  - E.g. adressages indexés... la plupart des instructions peuvent adresser la mémoire
- ↳ Soucis de compatibilité avec les générations précédentes
  - « compatibilité ascendante »

### RISC (début des années 80 CRAY/IBM)

- ↳ Jeu d'instructions limité dans le but de minimiser le temps d'exécution
  - E.g. seuls les instructions LOAD et STORE adressent la mém.

---

---

---

---

---

---

---

---

---

---

## Améliorations des performances (5/15) Philosophies CISC/RISC (suite)

| Caractéristique               | RISC       | CISC     |
|-------------------------------|------------|----------|
| Nbre d'instructions           | <100       | >200     |
| Nbre de modes d'adressage     | 1 à 2      | 5 à 20   |
| Nbre de format d'instructions | 1 à 2      | 3+       |
| Nbre cycles/instructions      | ~1         | 3 à 10   |
| Accès à la mémoire            | load/store | ~ toutes |
| Nbre registres                | 32+        | 2 à 16   |
| Réalisation µP                | câblé      | µ-pgm    |
| Logique pour décodage         | 10%        | 50%      |

---

---

---

---

---

---

---

---

---

---

## Améliorations des performances (6/15) Convergence RISC/CISC

- ▣ Convergence des performances CISC/RISC
  - ↪ La plupart des technologies RISC sont reprises dans les architectures CISC.
- ▣ Certains microprocesseurs CISC (e.g. Pentium II+) traduisent les instructions en suite de micro-instructions et fonctionnent comme un processeur RISC sur ces micro-instructions
  - ↪ Implémentation câblée

D. SIMPLOT - Architecture des Ordinateurs



199

---

---

---

---

---

---

---

---

## Améliorations des performances (7/15) Parallélisation des micro-instructions

- ▣ Dans l'écriture des séquences de micro-instructions correspondant à une instructions
  - ↪ On a mis en // différentes micro-instructions.
- ▣ En prenant plusieurs instructions d'un coup (typiquement 3 ou 4), on peut paralléliser les différentes micro-instructions
  - ↪ Pb. Des branchements peuvent intervenir et rendre faux les opérations déjà exécuter
  - ↪ ⇒ anticipation de branchement (branchement prédictifs)
  - ↪ ⇒ il faut pouvoir annuler une opération
  - ↪ « Out of Order »

D. SIMPLOT - Architecture des Ordinateurs



200

---

---

---

---

---

---

---

---

## Améliorations des performances (8/15) Parallélisation des micro-instructions (suite)

- ▣ Out of Order
  - ↪ Vient des CPU « super-scalaires »
    - 1: Add r1, r2 -> r8
    - 2: Sub r8, r3 -> r3
    - 3: Add r4, r5 -> r8
    - 4: Sub r8, r6 -> r6
  - ↪ Les instructions 1 et 3 peuvent être exécutées en parallèle si r8 est renommé

|                  |                  |
|------------------|------------------|
| Add r1, r2 -> r8 | Add r4, r5 -> r9 |
| Sub r8, r3 -> r3 | sub r9, r6 -> r6 |

D. SIMPLOT - Architecture des Ordinateurs



201

---

---

---

---

---

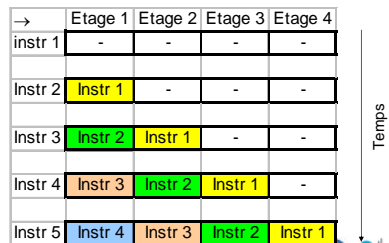
---

---

---

## Améliorations des performances (9/15) Parallélisation des micro-instructions (suite)

- Technique RISC : « pipe-line »
  - Microprocesseur en étage (travail à la chaîne)



D. SIMPLOT - Architecture des Ordinateurs

202

---

---

---

---

---

---

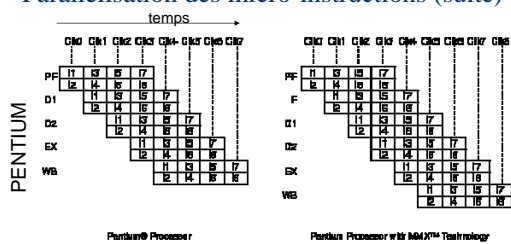
---

---

---

---

## Améliorations des performances (10/15) Parallélisation des micro-instructions (suite)



- NOTE: 11 refers to instruction n+1
- PF=Prefetch, F=Fetch, D1=Instruction decode, D2=Address Generate, EX=Execute, WB=Writeback

D. SIMPLOT - Architecture des Ordinateurs

203

---

---

---

---

---

---

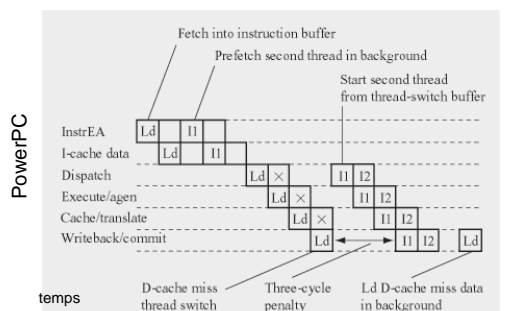
---

---

---

---

## Améliorations des performances (11/15) Parallélisation des micro-instructions (suite)



D. SIMPLOT - Architecture des Ordinateurs

204

---

---

---

---

---

---

---

---

---

---

## Améliorations des performances (12/15) Architecture IA-64

- ❑ Accord entre Intel et HP pour mettre au point une architecture 64 bits haut de gamme
- ❑ IA-32 + HP PA ⇒ IA-64 (Itanium)
- ❑ Architecture « CISC » ?
  - ↳ Multi-étages (une dizaine)
  - ↳ Compatible IA-32 avec génération de micro-code parallélisé à la volée (RISC)
    - Défauts de l'IA-32 : nbre de registres limité + calcul flottant lents + capacité mémoire limitée à 4 Go ☹
  - ↳ Mode IA-64
    - EPIC : Explicitly Parallel Instruction Computing

D. SIMPLOT - Architecture des Ordinateurs



205

---

---

---

---

---

---

---

---

---

---

## Améliorations des performances (13/15) Architecture IA-64 (suite)

- ❑ Techniques utilisées :
  - ↳ Parallélisation des instructions
  - ↳ Anticipation de branchements
    - Ne pas briser le pipe-line et les instructions en cours
  - ↳ Prédiction des chargements
    - Les chargements en mémoire sont très pénalisants et le sont de plus en plus avec l'accélération des µP
- ❑ Nombreux registres 64 bits (128 !)
- ❑ Les instructions sont regroupées en paquets (*bundle*) de 3 instructions (certains disent de 1 à 9+ ?)
  - ↳ Descripteur permettant d'anticiper et vérifier que les chargements ont été fait (prédiction des chargements) ainsi que tester s'il faut exécuter le bloc (anticipation des branchement)

D. SIMPLOT - Architecture des Ordinateurs



206

---

---

---

---

---

---

---

---

---

---

## Améliorations des performances (14/15) Architecture IA-64 (suite)

- ❑ 10 étages de l'itanium :



D. SIMPLOT - Architecture des Ordinateurs



207

---

---

---

---

---

---

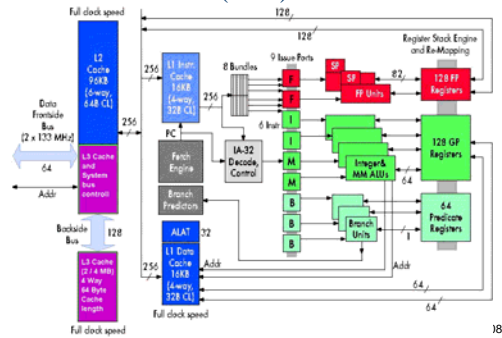
---

---

---

---

## Améliorations des performances (15/15) Architecture IA-64 (suite)



D. SIMPLOT - Architecture des Ordinateurs

---

---

---

---

---

---

---

---

## Conclusion

- ☐ On a vu :
  - ↪ Architecture interne d'un microprocesseur
  - ↪ Les chemins de données
  - ↪ La mémoire
  - ↪ La gestion des périphériques avec les interruptions et les DMA
  - ↪ Les optimisations possibles
- ☐ Reste à voir :
  - ↪ Liens entre matériel et logiciel (Partie III)
    - Système d'exploitation, génération de code,...
  - ↪ Gestion de la mémoire, des entrées/sorties (Partie IV)

D. SIMPLOT - Architecture des Ordinateurs




---

---

---

---

---

---

---

---

D. SIMPLOT - Architecture des Ordinateurs




---

---

---

---

---

---

---

---

# Architecture des Ordinateurs

## Partie III : Liens avec le système d'exploitation

### 1. Modèles d'exécution

David Simplot  
simplot@fil.univ-lille1.fr



---

---

---

---

---

---

---

---

## Objectifs

- ▣ Faire le lien entre le matériel et ce que vous faites « au-dessus »
  - ↪ Système d'exploitation
  - ↪ Modèle d'exécution
    - Langage natif
    - Machine virtuelle
  - ↪ Compilateurs
  - ↪ Outils d'aide à la conception et de mise au point

D. SIMPLOT - Architecture des Ordinateurs



212

---

---

---

---

---

---

---

---

## Au sommaire...

- ▣ **Rôle d'un système d'exploitation**
- ▣ Chaîne de compilation
- ▣ Machines virtuelles

D. SIMPLOT - Architecture des Ordinateurs



213

---

---

---

---

---

---

---

---

## Rôle d'un Système d'Exploitation (1/3)

- ❑ Système d'Exploitation = *Operating System*
- ❑ Présenter au programmes une abstraction du matériel
  - ↪ Piloter un périphérique = très compliqué
    - Pilote = *Driver*
  - ↪ => le système d'exploitation propose une HAL
    - Hardware Abstraction Layer
    - Ex. fichiers, objets
- ❑ L'interface entre le système d'exploitation et les programmes de l'utilisateur est constitué d'un ensemble d'« instructions étendues » fournies par le système d'exploitation
  - ↪ Appels système

D. SIMPLOT - Architecture des Ordinateurs



214

---

---

---

---

---

---

---

---

## Rôle d'un Système d'Exploitation (2/3)

- ❑ Gestion des processus
  - ↪ Multi-tâches
    - Préemptif, non préemptif
  - ↪ Ordonnanceur de processus
  - ↪ Partage des ressources
  - ↪ Communications inter-processus
- ❑ Les entrées/sorties (*Voir Partie IV*)
  - ↪ Interruptions, DMA
  - ↪ « Bufferisation » des E/S
- ❑ Gestion de la mémoire (*Voir Partie IV*)
- ❑ Système de fichiers

D. SIMPLOT - Architecture des Ordinateurs



215

---

---

---

---

---

---

---

---

## Rôle d'un Système d'Exploitation (3/3)

- ❑ Chargement d'applications
  - ↪ Passer de l'état « **fichier exécutable** » à l'état « **programme qui s'exécute** » ☺
  - ↪ Le fichier contient une suite d'octets correspondant au programme en langage machine ainsi qu'aux données
    - En plus, on a des informations sur le programme
      - L'adresse de début du programme
      - La mémoire nécessaire
      - Les bibliothèques dynamiques utilisées
      - ...
  - ↪ C'est le rôle du système d'exploitation d'effectuer le chargement
    - Copie du « code » en mémoire + liens + exécution

D. SIMPLOT - Architecture des Ordinateurs



216

---

---

---

---

---

---

---

---



## Chaîne de Compilation (1/14)

- ▣ Programme écrit en langage de « haut niveau »
  - ↳ Fortran, Pascal, Ada, Cobol, C, C++...
- ▣ Transformer le « **programme source** » en « **fichier exécutable** », c'est le rôle de la compilation.
- ▣ On distingue deux phases :
  - ↳ Compilation
  - ↳ Edition de liens




---

---

---

---

---

---

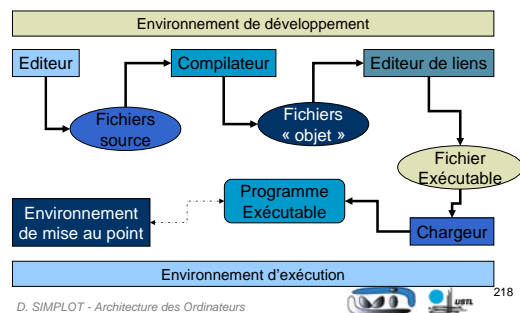
---

---

---

---

## Chaîne de Compilation (2/14)




---

---

---

---

---

---

---

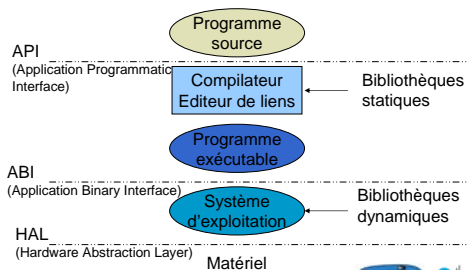
---

---

---

## Chaîne de Compilation (3/14) Niveaux de compatibilité

- ▣ On distingue plusieurs niveaux de compatibilité




---

---

---

---

---

---

---

---

---

---

## Chaîne de Compilation (4/14)

### Niveaux de compatibilité (suite)

- ▣ Compatibilité source
  - ↪ Respect de l'API proposée
  - ↪ Repose généralement sur des normes
    - Ex. ANSI, POSIX, X-OPEN
  - ↪ Le portage d'un programme implique la recompilation des sources
  - ↪ Pb. de propriété industrielle
    - Open Source

D. SIMPLOT - Architecture des Ordinateurs



220

---

---

---

---

---

---

---

---

## Chaîne de Compilation (5/14)

### Niveaux de compatibilité (suite)

- ▣ Compatibilité binaire niveau application
  - ↪ Programme sous forme de « binaires »
    - Programmes compilés dans un environnement déterminé sous forme chargeable
    - Plate-forme proposant l'interface ABI utilisée
  - ↪ Pb. pour les nouvelles architecture :
    - Coût de développement
    - ⇒ compatibilité binaire ascendante des plate-formes (matériel et logiciel)
  - ↪ Compatibilité binaire :
    - Architecture du processeur
    - Convention d'adressage et de communication (OS)
    - Interface avec l'OS et les bibliothèques (.so ou DLL)
    - Conventions de représentation des données (taille, LE, ou BE)

D. SIMPLOT - Architecture des Ordinateurs



221

---

---

---

---

---

---

---

---

## Chaîne de Compilation (6/14)

### Niveaux de compatibilité (suite)

- ▣ Compatibilité binaire niveau OS
  - ↪ Portabilité des != OS sur != plateformes
    - On travaille sur l'interface OS/matériel
    - HAL = Hardware Abstraction Layer

D. SIMPLOT - Architecture des Ordinateurs



222

---

---

---

---

---

---

---

---

## Chaîne de Compilation (7/14)

### Exemple GNU

#### ▣ Chaîne de compilation du projet GNU

- ↪ Le **projet GNU** a été lancé en 1984 afin de développer un système d'exploitation complet, semblable à Unix et qui soit un **logiciel libre**: le système GNU. (« GNU » est l'acronyme récuratif the « GNU's Not Unix »; on le prononce « gnou » avec un G audible) Des variantes du système d'exploitation GNU, basées sur le noyau « Linux », sont utilisées largement à présent; bien que ces systèmes soient communément appelés par le terme « Linux », ils le seraient plus exactement par « GNU/Linux ».



D. SIMPLOT - Architecture des Ordinateurs



223

---

---

---

---

---

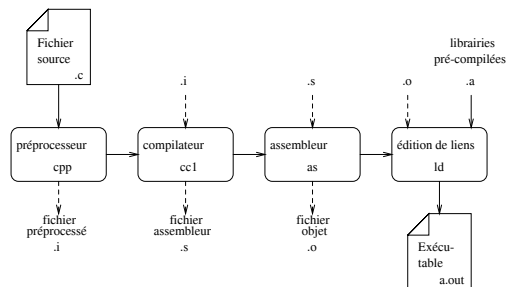
---

---

---

## Chaîne de Compilation (8/14)

### Exemple GNU (suite)



D. SIMPLOT - Architecture des Ordinateurs



224

---

---

---

---

---

---

---

---

## Chaîne de Compilation (9/14)

### Exemple GNU (suite)

#### ▣ Programme C très très simple :

```
#define MAX 2

int main(void)
{
 int a = MAX;

 a = a + 2;

 return 0;
}
```

D. SIMPLOT - Architecture des Ordinateurs



225

---

---

---

---

---

---

---

---

## Chaîne de Compilation (10/14)

### Exemple GNU (suite)

#### ▣ Première étape : préprocesseur

```
3 "ex.c"
int main(void)
{
 int a = 2;

 a = a + 2;

 return 0;
}
```

D. SIMPLOT - Architecture des Ordinateurs



226

---

---

---

---

---

---

---

---

## Chaîne de Compilation (11/14)

### Exemple GNU (suite)

#### ▣ Deuxième étape : génération de l'assembleur

```
.file "ex.c" "01.01"
.version "01.01"
gcc2_compiled.:
.text
.align 16
.globl main
.type main,@function
main:
 pushl %ebp
 movl %esp, %ebp
 subl $4, %esp
 movl $2, -4(%ebp)
 leal -4(%ebp), %eax
```

D. SIMPLOT - Architecture des Ordinateurs



227

---

---

---

---

---

---

---

---

## Chaîne de Compilation (12/14)

### Exemple GNU (suite)

```
 addl $2, (%eax)
 movl $0, %eax
 movl %ebp, %esp
 popl %ebp
 ret
.Lfel:
 .size main, .Lfel-main
 .ident "GCC: (GNU) 2.96 20000731 (Linux-
Mandrake 8.0 2.96-0.48mdk)"
```

D. SIMPLOT - Architecture des Ordinateurs



228

---

---

---

---

---

---

---

---

## Chaîne de Compilation (13/14)

### Exemple GNU (suite)

#### ▣ Troisième étape : génération du .o

```
➤ 000000: 71 45 4c 46 01 01 01 00 127 069 076 070 001 001 000 000 .ELF....
➤ 000008: 00 00 00 00 00 00 00 00 000 000 000 000 000 000 000 000
➤ 000010: 01 00 03 00 01 00 00 00 001 000 003 000 001 000 000 000
➤ 000018: 00 00 00 00 00 00 00 00 000 000 000 000 000 000 000 000
➤ 000020: e8 00 00 00 00 00 00 00 232 000 000 000 000 000 000 000 è.....
➤ 000028: 34 00 00 00 00 00 28 00 052 000 000 000 000 000 040 000 4.....(
➤ 000030: 09 00 06 00 00 00 00 00 009 000 006 000 000 000 000 000
➤ 000038: 00 00 00 00 00 00 00 00 000 000 000 000 000 000 000 000
➤ 000040: 55 89 e5 83 ec 04 c7 45 085 137 229 131 236 004 199 069 U.ä.i.ÇE
➤ 000048: fe 02 00 00 00 84 45 fe 252 002 000 000 000 141 069 252 U.....Eu
➤ 000050: 83 00 02 b8 00 00 00 00 131 000 002 184 000 000 000 000
➤ 000058: 89 ec 5d c3 08 00 00 00 137 236 093 195 008 000 000 000 ijÄ....
➤ 000060: 00 00 00 00 01 00 00 00 000 000 000 000 001 000 000 000
➤ 000068: 30 31 2e 30 31 00 00 00 048 049 046 048 049 000 000 000 01.01...
➤ 000070: 00 47 43 43 3a 20 28 47 000 071 067 067 058 032 040 071 .GCC: (G
➤ 000078: 4e 55 29 20 32 2e 39 36 078 085 041 032 050 046 057 054 NU) 2.96
➤ ...
```

D. SIMPLOT - Architecture des Ordinateurs



229

---

---

---

---

---

---

---

---

---

---

## Chaîne de Compilation (14/14)

### Exemple GNU (suite)

#### ▣ Génération de l'exécutable

- Prendre les fichiers .o
- Déterminer le point d'entrée du programme
- Faire les liens
  - Edition de liens = Link

#### ▣ On peut mettre des informations permettant de déboguer le programme

D. SIMPLOT - Architecture des Ordinateurs



230

---

---

---

---

---

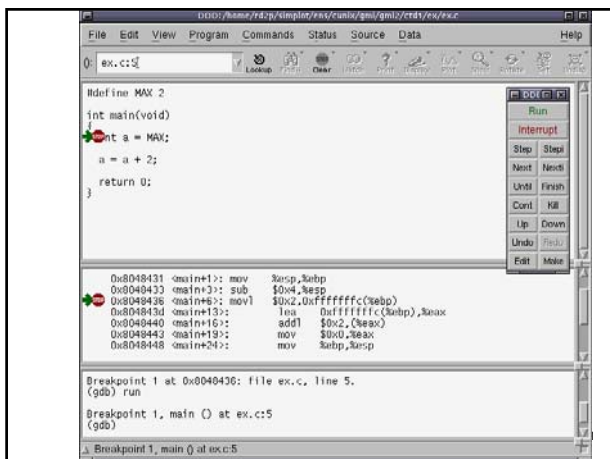
---

---

---

---

---



---

---

---

---

---

---

---

---

---

---

## Machines Virtuelles (1/4)

### Objectif =

- ↪ S'affranchir de la compabilité binaire !
- ↪ On ne compile plus pour une plate-forme physique donnée, mais pour une Machine Virtuelle

### Langage Java

- ↪ Développé par Sun Microsystem
- ↪ S'inspire du C++ pour la syntaxe
- ↪ S'inspire de Smalltalk pour la philosophie
- ↪ Indépendance de la plate-forme matérielle
- ↪ JVM = Java Virtual Machine



**Write once, run everywhere.**

D. SIMPLOT - Architecture des Ordinateurs



232

---

---

---

---

---

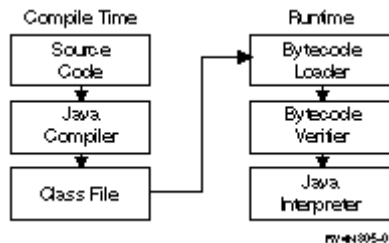
---

---

---

## Machines Virtuelles (2/4)

### Chaîne de compilation et plate-forme d'exécution



D. SIMPLOT - Architecture des Ordinateurs



233

---

---

---

---

---

---

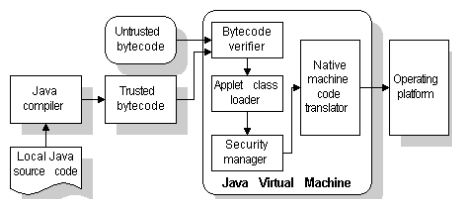
---

---

## Machines Virtuelles (3/4)

### Utilisation de code natif

- ↪ JIT = Just-In-Time compiler



D. SIMPLOT - Architecture des Ordinateurs



234

---

---

---

---

---

---

---

---

## Machines Virtuelles (4/4)

- Java n'est pas le seul langage avec machine virtuelle :
  - Smalltalk
  - VisualBasic (génération de P-code)
  - Langages interprétés en général



---

---

---

---

---

---

---

---

## Conclusion

- Reste à voir les techniques de génération de code



---

---

---

---

---

---

---

---



---

---

---

---

---

---

---

---

# Architecture des Ordinateurs

## Partie III : Liens avec le système d'exploitation

### 2. Génération de code

David Simplot  
simplot@fil.univ-lille1.fr



---

---

---

---

---

---

---

---

## Objectifs

- ▣ Voir la génération de code ainsi que les mécanismes implicites utilisés
  - Gestion des données
  - Gestion des appels de fonctions
  - Gestion de l'allocation dynamique

D. SIMPLOT - Architecture des Ordinateurs



239

---

---

---

---

---

---

---

---

## Au sommaire...

- ▣ **Schéma général d'un compilateur**
- ▣ Organisation de l'espace mémoire
- ▣ Accès aux noms non locaux
- ▣ Passage de paramètres
- ▣ Techniques pour l'allocation dynamique

D. SIMPLOT - Architecture des Ordinateurs



240

---

---

---

---

---

---

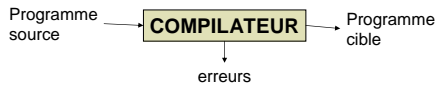
---

---



## Schéma général d'un compilateur (1/8)

- ▣ Vision simplifiée d'un compilateur :



- ▣ Programme source
  - ↪ C, C++, Java, ADA, Cobol, Fortran, Pascal...
- ▣ Programme cible
  - ↪ Programme « binaire » en langage machine
  - ↪ Bytecode (type bytecode Java ou P-Code MS)

D. SIMPLOT - Architecture des Ordinateurs



241

---

---

---

---

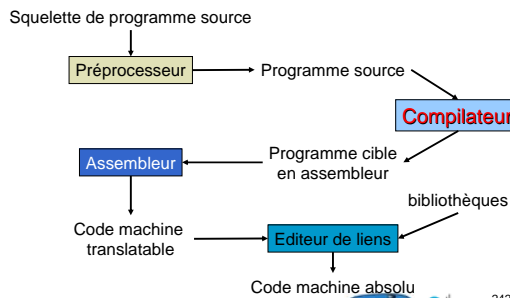
---

---

---

---

## Schéma général d'un compilateur (2/8)



D. SIMPLOT - Architecture des Ordinateurs



242

---

---

---

---

---

---

---

---

## Schéma général d'un compilateur (3/8)

- ▣ Modèle de la compilation par analyse et synthèse

- ↪ Analyse = partitionner le programme et construire une représentation intermédiaire des parties
  - Voir cours d'ASC
    - Analyse lexicale = identifier les tokens
    - Analyse syntaxique = déduire les structures
      - structure d'arbres abstraits
    - Analyse sémantique = contrôle des types, bon utilisation des opérateurs, etc.
- ↪ Synthèse = construire le programme cible désiré à partir de la représentation intermédiaire

D. SIMPLOT - Architecture des Ordinateurs



243

---

---

---

---

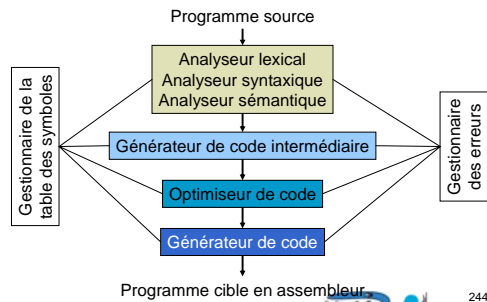
---

---

---

---

## Schéma général d'un compilateur (4/8)




---

---

---

---

---

---

---

---

## Schéma général d'un compilateur (5/8)

- Rôle de la table des symboles
  - ↪ Stocker les identificateurs utilisés dans le programme
    - Déterminer leurs types (variables, fonctions, etc...)
    - Pour les fonctions : le nombre et le type des arguments
    - Déterminer leurs portées dans le programme
- Le code intermédiaire est généré à l'issue de la phase d'analyse sémantique

---

---

---

---

---

---

---

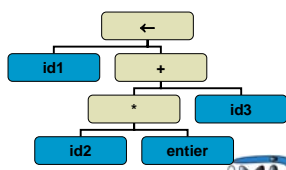
---

## Schéma général d'un compilateur (6/8) Exemple

- Programme source
  - ↪  $note = vrai\_note * 2 + mini$
- Analyseur lexical
  - ↪  $id1 \leftarrow id2 * entier + id3$
- Analyseur syntaxique

Table des symboles

|   |           |       |
|---|-----------|-------|
| 1 | note      | float |
| 2 | vrai_note | float |
| 3 | mini      | float |




---

---

---

---

---

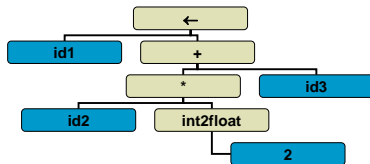
---

---

---

## Schéma général d'un compilateur (7/8) Exemple (suite)

- Analyse sémantique



D. SIMPLOT - Architecture des Ordinateurs



247

---

---

---

---

---

---

---

---

## Schéma général d'un compilateur (8/8) Exemple (suite)

- Générateur de code intermédiaire
  - `tmp1 ← int2float(2)`
  - `tmp2 ← id2 * tmp1`
  - `tmp3 ← tmp2 + id3`
  - `id1 ← tmp3`
- Générateur de code
  - `MOVF R1, id2`
  - `MULF R1, %2.0`
  - `MOVF R2, id3`
  - `ADDF R1, R2`
  - `MOVF id1, R1`
- Optimiseur de code
  - `↪ tmp1 ← id2 * 2.0`
  - `↪ id1 ← tmp1 + id3`

D. SIMPLOT - Architecture des Ordinateurs



248

---

---

---

---

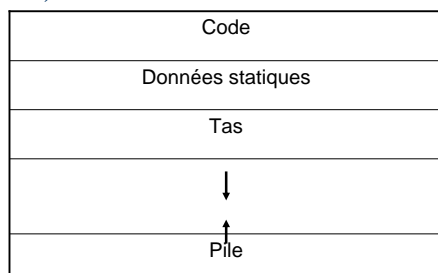
---

---

---

---

## Organisation de l'espace mémoire (1/4)



D. SIMPLOT - Architecture des Ordinateurs



249

---

---

---

---

---

---

---

---

## Organisation de l'espace mémoire (2/4)

### Tas

- ↳ Sert pour l'allocation dynamique de mémoire
  - E.g. le malloc

### Pile

- ↳ Sert pour l'« appel des fonctions »
  - Idem pour les invocations de méthodes en langage objet
- ↳ À chaque appel de fonction, on se sert de cette pile pour mettre un « enregistrement d'activation »
  - Aussi appelé « frame »

D. SIMPLOT - Architecture des Ordinateurs



250

---

---

---

---

---

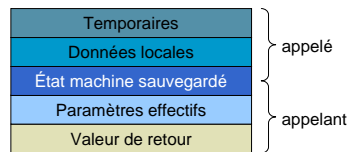
---

---

---

## Organisation de l'espace mémoire (3/4)

### Exemple d'enregistrement d'activation :



- ↳ État machine sauvegardé
  - Appellant : PC lors de l'appel à la sous-routine (CALL/JSR)
  - Appelé : sauvegarde des registres

D. SIMPLOT - Architecture des Ordinateurs



251

---

---

---

---

---

---

---

---

## Organisation de l'espace mémoire (4/4)

### La taille de chacun des champs « données »

- ↳ Aussi bien données locales, paramètres que valeur de retour

### dépend de la représentation en mémoire des types du langage de haut-niveau

### L'accès aux variables locales et aux paramètres passent donc par la pile (par le frame).

### Problème :

- ↳ La taille du programme est limitée  $\Rightarrow$  on majore la taille du tas et de la pile  $\Rightarrow$  gaspillage d'espace mémoire dans la plupart des cas
- ↳ Solution : chaînage de pile et de tas...

D. SIMPLOT - Architecture des Ordinateurs



252

---

---

---

---

---

---

---

---

## Accès aux noms non locaux (1/3)

- Variables globales ou variables de classes
  - Adresse connue à la compilation ou on a un « adresseur » qui connaît l'adresse de réification de la classe

- Variables de bloc

```
int main()
{
 int a=0;
 int b=3;
 {
 float b=1.5;
 ...
 }
}
```

Bloc 1

Bloc 0

D. SIMPLOT - Architecture des Ordinateurs



253

---

---

---

---

---

---

---

---

## Accès aux noms non locaux (2/3)

- On renomme les variables

```
int main() int main()
{ {
 int a=0; int a=0;
 int b=3; int b0=3;
 { {
 float b=1.5; float b1=1.5;

 } }
} }
```

D. SIMPLOT - Architecture des Ordinateurs



254

---

---

---

---

---

---

---

---

## Accès aux noms non locaux (3/3)

- C'est à la charge de l'optimiseur de déterminer la durée de vie des variables et de réutiliser l'espace mémoire si possible

D. SIMPLOT - Architecture des Ordinateurs



255

---

---

---

---

---

---

---

---

## Passage de paramètres (1/7)

- ▣ Passage par valeur
  - ↪ Ce qui est mis dans le frame, c'est une copie de la valeur de l'argument
- ▣ Passage par référence
  - ↪ On met dans le frame l'adresse de l'argument
  - ↪ Il s'agit d'un pointeur
  - ↪ Cette adresse pointe sur une données accessible par la fonction appelante
- ▣ Passage par donnée-résultat
  - ↪ Mixe des deux précédents
  - ↪ On donne un pointeur et un peut éventuellement utiliser la valeur stockée à l'adresse passée en paramètre

D. SIMPLOT - Architecture des Ordinateurs



256

---

---

---

---

---

---

---

---

## Passage de paramètres (2/7) Exemple

```
void swap(int *a, int *b) int min(int a, int b)
{
 int tmp; {
 tmp = *a; if (a<b)
 *a = *b; return a;
 *b = *a; return b;
} }

void tri(int *a, int *b) main()
{
 if (*a > *b) swap(a, b) {
} int x=7, y=5, z;
 z=min(x, 3);
 tri(&x, &y);
 }

```

D. SIMPLOT - Architecture des Ordinateurs



257

---

---

---

---

---

---

---

---

## Passage de paramètres (3/7) Exemple (suite)

```
▣ Function main push x ; arg1
 .return none tmp1 ← 3
 .param push tmp1 ; arg2
 .locals
 x int
 y int
 z int
 .temp
 tmp1 db 4 dup(?)
 .code
 x ← 7
 y ← 5
 sp ← sp + 4 ; réservation ; retour
 call min ; arg1
 push tmp1 ; arg2
 tmp1 ← @y
 push tmp1 ; arg2
 call tri
 end function ; fin main

```

D. SIMPLOT - Architecture des Ordinateurs



258

---

---

---

---

---

---

---

---

## Passage de paramètres (4/7) Exemple (suite)

```
■ Function min
.return int
.param
 a int
 b int
.locals
.temp
 tmp1 int
 tmp2 int
.code
 tmp1 ← a
 tmp2 ← b

 cmp tmp1, tmp2
 jb min_suite
 return ← tmp1 ; retour
 jmp min_fin
min_suite:
 return ← tmp2 ; retour
min_fin:
 end function ; fin min
```

D. SIMPLOT - Architecture des Ordinateurs



259

---

---

---

---

---

---

---

---

---

---

## Passage de paramètres (5/7) Exemple (suite)

```
■ Appel de la fonction min
```

D. SIMPLOT - Architecture des Ordinateurs



260

---

---

---

---

---

---

---

---

---

---

## Passage de paramètres (6/7) Exemple (suite)

```
■ Function tri
.return none
.param
 a int*
 b int*
.locals
.temp
 tmp1 int
 tmp2 int

 .code
 tmp1 ← [a]
 tmp2 ← [b]
 cmp tmp1, tmp2
 jle fin
 push a ; arg #1
 push b ; arg #2
 call swap
 fin:
 End function
```

D. SIMPLOT - Architecture des Ordinateurs



261

---

---

---

---

---

---

---

---

---

---

## Passage de paramètres (7/7)

### Exemple (suite)

- ▣ **Function swap**

```
.return none
.param
a int*
b int*
.locals
tmp int
.temp
tmp1 int

.code
tmp ← [a]
tmp1 ← [b]
[a] ← tmp1
[b] ← tmp
End function
```
- ▣ **Exécution de la fonction swap**

D. SIMPLOT - Architecture des Ordinateurs



262

---

---

---

---

---

---

---

---

---

---

## Techniques pour l'allocation dynamique

- ▣ **But** : construire des données/objets persistant à la destruction du contexte d'une fonction
- ▣ Il s'agit de « gérer » le tas
- ▣ Implémentation « bitmap »
- ▣ Implémentation par « chaînage » des emplacements libres
  - ↪ On regroupe par « slots » d'une taille fixée
  - ↪ On chaîne les emplacements vides

D. SIMPLOT - Architecture des Ordinateurs



263

---

---

---

---

---

---

---

---

---

---

## Conclusion

- ▣ **Optimisation du code intermédiaire**
  - ↪ Durée de vie des variables
- ▣ **Génération de code natif**
  - ↪ Encore une phase d'optimisation
    - Pour utiliser les instructions les moins coûteuses du microprocesseur
    - À chaque instruction, on peut attribuer un coût qui est le nombre de cycles nécessaire pour faire l'instruction
      - De plus en plus difficile avec les processeurs optimisants
  - ↪ Réécriture avec des règles de traductions des instructions du code intermédiaire
  - ↪ Cross-compileur ?

D. SIMPLOT - Architecture des Ordinateurs



264

---

---

---

---

---

---

---

---

---

---