

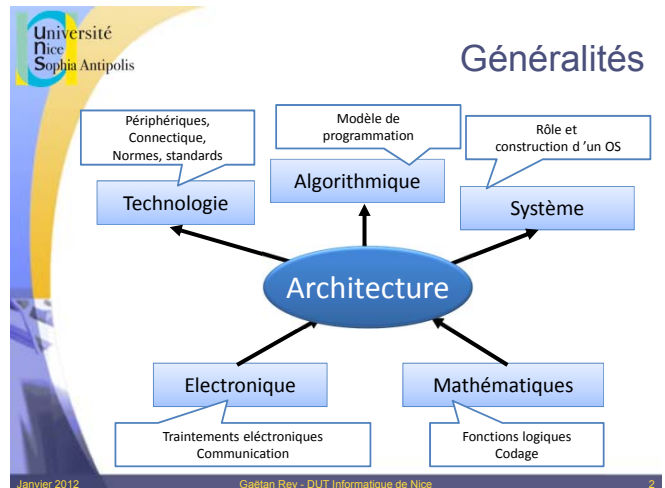
Université
Nice
Sophia Antipolis

Architecture des ordinateurs (ASR 1 et ASR2)

Gaëtan Rey

Département Informatique - IUT de Nice - Côte d'Azur
<http://users.polytech.unice.fr/~rey/cours/archi.html>
Gaetan.Rey@unice.fr



Université
Nice
Sophia Antipolis

Objectifs du Cours

- ✓ De quoi est composé un ordinateur ?
- ✓ Quels sont les modèles sous-jacents au fonctionnement d'une machine ?
- ✓ Comment s'exécutent les programmes ?
- ✓ Quel est le lien entre le logiciel et le matériel ?
- ✓ Comment fonctionnent les divers périphériques ?

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 3

Université
Nice
Sophia Antipolis

Organisation du Cours

- ✓ Les séances
 - 14 x 1h30 de cours
 - 10 x 3h00 de TP
- ✓ Les évaluations
 - Un Devoir Surveillé en fin de module
 - Un contrôle continu
 - Un projet (optionnel)
 - Une évaluation intermédiaire sous forme d'examen
 - Une évaluation du travail effectué lors des séances de TP
 - Une évaluation du travail effectué en dehors des séances de TP
 - Un contrôle possible à chaque séance de cours et ou de TP

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 4

Université
Nice
Sophia Antipolis

Projet d'Architecture (1)

- ✓ Optionnel
 - Un bonus en contrôle continu
- ✓ Présentation orale d'un sujet en rapport avec le cours
 - Sujet technique
 - Sujet non traités en cours
- ✓ Consignes
 - Groupe de 2 à 4 personnes
 - Durée présentation de 5 min x nombre de personnes du groupe
- ✓ Publication en ligne des meilleures présentations
 - Objectif est de créer une mini revue technique

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 5

Université
Nice
Sophia Antipolis

Projet d'Architecture (2)

- ✓ S'inscrire avant le 1/02 à 12h00
 - Donner la liste des membres
 - Proposer deux ou trois sujets
- ✓ Suivi du projet
 - <http://users.polytech.unice.fr/~rey/cours/archiprojet.html>

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 6

Plan

- ✓ Introduction
- ✓ Représentation de l'information
- ✓ Arithmétique binaire
- ✓ Algèbre de Boole
- ✓ Circuits séquentiels/Automates
- ✓ Architecture type Von Neumann
- ✓ Couche d'assemblage / Assembleur 8086
- ✓ Mécanismes d'interruptions
- ✓ Fonctions avancées de processeurs modernes
- ✓ Gestion de la mémoire
- ✓ Traduction de programmes, compilation édition de liens, chargement

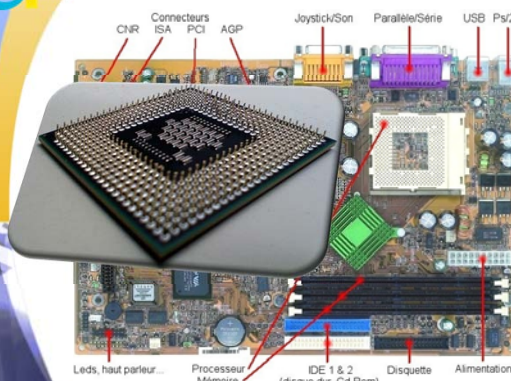
Ordinateur

- ✓ Définitions
 - Machine capable d'effectuer automatiquement des opérations arithmétiques et logiques à partir de programmes définissant la séquence de ces opérations.
 - Un ensemble de circuits électroniques permettant de manipuler des données sous forme binaire, ou bits.
- ✓ But d'un ordinateur
 - exécuter des séquences de calculs

Décomposition matérielle d'un ordinateur

- ✓ Unité centrale
 - Carte mère, carte graphique, disque dur, ...
- ✓ Périphériques d'entrée
 - Souris, clavier, ...
- ✓ Périphériques de sorties
 - Ecrans, Haut-parleur
- ✓ Périphériques de stockage
 - Disquette, lecteur CD-ROM, DVD-ROM, ...

La carte mère d'un PC



La carte mère d'un PC



Terminologie (1)

- ✓ **Bit** = Unité de mesure en informatique désignant la quantité élémentaire d'information représentée par un chiffre du système binaire (Binary digit)
- ✓ **Octet** = Unité de mesure en informatique mesurant la quantité de données (Byte)
 - 1 octet = 8 bits
 - Symbole de l'octet est la lettre « o » minuscule (« B » majuscule en anglais pour le Byte).
- ✓ *Un bit est différent d'un octet*
- ✓ *A bit is different to a byte*



Terminologie (2)

- ✓ **CPU** = Processeur (Central Processing Unit)
- ✓ **GPU** = Processeur Graphique (Graphics Processing Unit)
- ✓ **PPU** = Processeur Physique (Physics Processing Unit)
- ✓ **TPU** = Processeur de Temps (Time Processor Unit)
- ✓ **GPGPU** = Calcul générique sur processeur graphique (General-purpose computing on graphics processing units)

Terminologie (3)

- ✓ **CISC** = Famille de processeur qui a des instructions aussi proches que possible d'un langage de haut niveau (Complex Instruction Set Computer)
- ✓ **RISC** = Famille de processeur qui a des instructions plus simples et une structure permettant une exécution très rapide (Reduced Instructions Set Computer)
- ✓ **DSP** = Processeur spécialisé pour les calculs liés au traitement de signaux (Digital Signal Processor)

Terminologie (4)

- ✓ **ALU** = Unité Arithmétique et Logique ou UAL (*Arithmetic and Logical Unit*)
- ✓ **FPU** = Unité de calcul en virgule flottante (Floating Point Unit)
- ✓ **MIPS** = Millions d'instructions par seconde
- ✓ **CPI** = Cycles par instruction
- ✓ **MSB** = Bit de poids fort (Most Significant Bit)
- ✓ **LSB** = Bit de poids faible (Least Significant Bit)

Terminologie (5)

- ✓ Préfixes système international d'unités

- ✓ Préfixes de la commission électrotechnique internationale IEC 60027-2

Nom	Symbole	Ordre de grandeur SI décimal
unité		$10^0 = 1$
kilo	kK	$10^3 = 1\ 000$
méga	M	$10^6 = 1\ 000\ 000$
giga	G	$10^9 = 1\ 000\ 000\ 000$
téra	T	$10^{12} = 1\ 000\ 000\ 000\ 000$
péta	P	$10^{15} = 1\ 000\ 000\ 000\ 000\ 000$
exa	E	$10^{18} = 1\ 000\ 000\ 000\ 000\ 000\ 000$
zetta	Z	$10^{21} = 1\ 000\ 000\ 000\ 000\ 000\ 000\ 000$
yotta	Y	$10^{24} = 1\ 000\ 000\ 000\ 000\ 000\ 000\ 000\ 000$

Nom	Symbole	Facteur
kibi	Ki	$2^{10} = 1\ 024$
mébi	Mi	$2^{20} = 1\ 048\ 576$
gibi	Gi	$2^{30} = 1\ 073\ 741\ 824$
tébi	Ti	$2^{40} = 1\ 099\ 511\ 627\ 776$
pébi	Pi	$2^{50} = 1\ 125\ 899\ 906\ 842\ 624$
exbi	Ei	$2^{60} = 1\ 152\ 921\ 504\ 606\ 846\ 976$
zébi	Zi	$2^{70} = 1\ 180\ 591\ 620\ 717\ 411\ 303\ 424$
yobi	Yi	$2^{80} = 1\ 208\ 925\ 819\ 614\ 629\ 174\ 706\ 176$

Historique

- ✓ <http://www.histoire-informatique.org/>
- ✓ <http://histoire.info.online.fr/>

Génération	Dates	Technologie	Opérations/s
1	1946-57	Tubes à vide	40k
2	1958-64	Transistors	200K
3	1965-71	SSI-MSI	1M
4	1972-77	LSI	10M
5	1978-	VLSI	100M

Le processeur

- ✓ Le processeur a été inventé par Intel (avec le modèle 4004) en 1971.
 - Il est chargé de traiter les informations et d'exécuter les instructions.
 - Il communique avec le reste de l'ordinateur via un langage binaire.
- ✓ Le processeur est rythmé par une horloge (quartz) cadencée plus ou moins rapidement
 - On exprime la fréquence en Hz (actuellement GHz).
- ✓ A chaque impulsion d'horloge, le processeur lit l'instruction stockée généralement dans un registre d'instruction et exécute l'instruction.

Université Nice Sophia Antipolis

Fabrication des processeurs

- ✓ Les processeurs sont gravés sur des plaques appelées Wafers.
- ✓ La finesse de gravure est différentes selon les séries de processeurs (mesurée en μm ou nm).
 - Les processeurs actuels sont gravés en 45 nm.
- ✓ Diminuer la finesse de gravure permet de
 - Produire plus de processeurs à la fois sur un Wafer
 - Diminution du coût de fabrication
 - Diminution de la consommation d'énergie du processeur et donc de la quantité de chaleur produite
 - Monter plus haut en fréquence.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 19

Université Nice Sophia Antipolis

Un Wafer

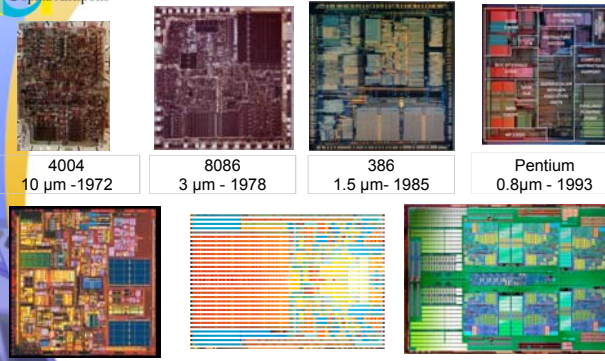
- ✓ Galette de silicium sur lequel est « gravé » un ensemble de processeur
 - <http://www.x86-secret.com/popups/articleswindow.php?id=64>
 - http://www.onversity.com/cgi-bin/progactu/actu_aff.cgi?Eudo=TUYhjkW&P=356#10
 - <http://www.matbe.com/articles/lire/375/cpu-avancees-des-technologies-de-fabrication/>



Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 20

Université Nice Sophia Antipolis

Microprocesseurs



4004 10 μm - 1972	8086 3 μm - 1978	386 1.5 μm - 1985	Pentium 0.8 μm - 1993
Pentium 4 (Northwood) 0.13 μm - 2001	Core 2 (Penryn) 0.045 μm - 2007	Opteron (hexaCore) 0.045 μm - 2009	

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 21

Université Nice Sophia Antipolis

Plan

- ✓ Introduction
- ✓ Représentation de l'information
- ✓ Arithmétique binaire
- ✓ Algèbre de Boole
- ✓ Circuits séquentiels/Automates
- ✓ Architecture type Von Neumann
- ✓ Couche d'assemblage / Assembleur 8086
- ✓ Mécanismes d'interruptions
- ✓ Fonctions avancées de processeurs modernes
- ✓ Gestion de la mémoire
- ✓ Traduction de programmes, compilation édition de liens, chargement

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 22

Université Nice Sophia Antipolis

Représentation des données

- ✓ Différents types d'informations
 - Textes, nombres, ...
- ✓ L'ordinateur ne connaît que le binaire
 - Toutes informations est une suite de 0 et de 1
- ✓ Nécessité d'établir un codage de l'information en binaire c.-à-d. d'établir une correspondance entre la représentation externe de l'information et la représentations interne dans la machine

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 23

Université Nice Sophia Antipolis

Représentation d'un nombre en base b

- ✓ Représentation utilisée par les humains
 - Base 10 ou décimale
 - 10 symboles utilisés : 0,1,2,3,4,5,6,7,8,9
- ✓ Représentation utilisée par les ordinateurs
 - Base 2 ou binaire, base hexadécimale
- ✓ Pour un représentation un nombre en base b, on utilise b symboles :
 - Base 2 : 0,1
 - Base 8 : 0,1,2,3,4,5,6,7
 - Base 16 : 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 24

Université Nice Sophia Antipolis

Représentation d'un nombre en base 10

✓ Comment représentons nous un nombre en base 10 ?

- Exemple 2568
 - millier
 - centaine
 - dizaine
 - unité

$$2568 = 2000 + 500 + 60 + 8$$

$$= 2 \times 1000 + 5 \times 100 + 6 \times 10 + 8 \times 1$$

$$= 2 \times 10^3 + 5 \times 10^2 + 6 \times 10^1 + 8 \times 10^0$$

poids base 10 rang

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 25

Université Nice Sophia Antipolis

Représentation d'un nombre en base b

✓ $N = a_n b^n + a_{n-1} b^{n-1} + \dots + a_1 b^1 + a_0 b^0$

Poids fort Poids faible

avec a_i, b^i où i est le rang de a_i , b est la base et a_i est un symbole de la base (ou sa valeur en base 10).

✓ On appelle a_i le poids.

- a_0 est le chiffre de poids faible,
- a_n est le chiffre de poids fort

✓ Par convention, on note les poids fort à gauche.

- Exemple : $(125)_{10} = 1 \times 10^2 + 2 \times 10^1 + 5 \times 10^0$

indique que le nombre est en base 10

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 26

Université Nice Sophia Antipolis

Exemples

✓ $(354)_{10} = 3 \times 10^2 + 5 \times 10^1 + 4 \times 10^0$

✓ $(01100110)_2 = 0 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$

$$= 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^2 + 1 \times 2^1$$

$$= 64 + 32 + 4 + 2 = 102$$

✓ $(1045)_8 = 1 \times 8^3 + 0 \times 8^2 + 4 \times 8^1 + 5 \times 8^0$

$$= 512 + 0 + 32 + 5 = 549$$

✓ $(7BE)_{16} = 7 \times 16^2 + 11 \times 16^1 + 14 \times 16^0$

$$= 1792 + 176 + 14 = 1982$$

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 27

Université Nice Sophia Antipolis

Passage de la base 10 à la base n

✓ Divisions successives du nombre par n pour la partie entière

- Exemple $20_{(10)} = 10100_{(2)}$

Bit poids faible Bit poids fort

- Exemple $589_{(10)} = 24D_{(16)}$

poids faible il faut convertir 13 en D poids fort

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 28

Université Nice Sophia Antipolis

Base 16 et Base 2

✓ Puisque 16 est une puissance de 2 ($16 = 2^4$), il existe un lien privilégié entre ces bases.

✓ Il existe donc une astuce permettant de passer d'une base à l'autre très facilement

- Base 16 vers base 2 : chaque symbole en base 16 se représente sur 4 bits

- $123_{(16)} = 0001\ 0010\ 0011_{(2)} = 1\ 0010\ 0011_{(2)}$
- $A8F_{(16)} = 1010\ 1000\ 1111_{(2)}$
- $E5B_{(16)} = 1110\ 0101\ 1011_{(2)}$

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 29

Université Nice Sophia Antipolis

Base 16 et Base 2

✓ Puisque 16 est une puissance de 2 ($16 = 2^4$), il existe un lien privilégié entre ces bases.

✓ Il existe donc une astuce permettant de passer d'une base à l'autre très facilement

- Base 16 vers base 2 : chaque symbole en base 16 se représente sur 4 bits
- Base 2 vers base 16 : on associe 4 bits consécutifs pour obtenir un symbole en base 16

- $100\ 1100\ 1001_{(2)} = 0100\ 1100\ 1001_{(2)} = 4C9_{(16)}$
- $110\ 0111\ 1101_{(2)} = 0110\ 0111\ 1101_{(2)} = 67D_{(16)}$

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 30

Base 8 et Base 2

- ✓ Puisque 8 est une puissance de 2 ($8 = 2^3$), il existe un lien privilégié entre ces bases.
- ✓ Il existe donc une astuce permettant de passer d'une base à l'autre très facilement
 - Base 8 vers base 2 : chaque symbole en base 8 se représente sur 3 bits

$$\begin{aligned}
 123_{(8)} &= \begin{array}{ccc} 001 & 010 & 011_{(2)} \\ 1 & 2 & 3 \end{array} = 101\ 0011_{(2)} \\
 765_{(8)} &= \begin{array}{ccc} 111 & 110 & 101_{(2)} \\ 7 & 6 & 5 \end{array} = 1\ 1111\ 0101_{(2)} \\
 402_{(8)} &= \begin{array}{ccc} 100 & 000 & 010_{(2)} \\ 4 & 0 & 2 \end{array} = 1\ 0000\ 0010_{(2)}
 \end{aligned}$$

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

31

Base 8 et Base 2

- ✓ Puisque 8 est une puissance de 2 ($8 = 2^3$), il existe un lien privilégié entre ces bases.
- ✓ Il existe donc une astuce permettant de passer d'une base à l'autre très facilement
 - Base 8 vers base 2 : chaque symbole en base 8 se représente sur 3 bits
 - Base 2 vers base 8 : on associe 3 bits consécutifs pour obtenir un symbole en base 8

$$\begin{aligned}
 1011\ 0011_{(2)} &= \begin{array}{ccc} 010 & 110 & 011_{(2)} \\ 2 & 6 & 3 \end{array} = 263_{(8)} \\
 1\ 0001\ 1101\ 0001_{(2)} &= \begin{array}{ccccccc} 001 & 000 & 111 & 010 & 001_{(2)} \\ 1 & 0 & 7 & 2 & 1 \end{array} = 10721_{(8)}
 \end{aligned}$$

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

32

Tableau récapitulatif

Base 10 (décimal)	Base 2 (binaire)	Base 8 (octale)	Base 16 (Hexadécimale)
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

33

Représentation des nombres fractionnaires

- ✓ C'est-à-dire ceux comportant des chiffres après la virgule
 - En base 10
 - $98,765 = 9 \times 10^1 + 8 \times 10^0 + 7 \times 10^{-1} + 6 \times 10^{-2} + 5 \times 10^{-3}$
 - En base n (cas général)
 - $a_n \dots a_0 \dots a_{-p} = a_n b^n + \dots + a_0 b^0 + \dots + a_{-p} b^{-p}$
 - Exemples
 - $10,101_{(2)} = 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} = 2 + 0.5 + 0.125 = 2.625_{(10)}$
 - $2,07_{(8)} = 2 \times 8^0 + 7 \times 8^{-2} = 2 + 0,109375 = 2,109375_{(10)}$
 - $F,5_{(16)} = 15 \times 16^0 + 5 \times 16^{-1} = 15 + 0.3125 = 15.3125_{(10)}$

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

34

Représentation des nombres fractionnaires

- ✓ Comment passer de la base 10 à la base n ?
 - On multiplie la partie fractionnaire par n en répétant l'opération sur la partie fractionnaire du produit jusqu'à ce qu'elle soit nulle (ou que la précision souhaitée soit atteinte)
- ✓ Exemple de la base 10 à la base 2
 - $54,25_{(10)} = 54_{(10)} + 0,25_{(10)} = 110110,01$

$$\begin{aligned}
 54_{(10)} &= 110110 \\
 0,25 \times 2 &= 0,50 \rightarrow 0 \\
 0,50 \times 2 &= 1,00 \rightarrow 1 \\
 0,00 \times 2 &= 0,00 \rightarrow 0
 \end{aligned}$$

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

35

Représentation des nombres fractionnaires

- ✓ Comment passer de la base 10 à la base n ?
 - On multiplie la partie fractionnaire par n en répétant l'opération sur la partie fractionnaire du produit jusqu'à ce qu'elle soit nulle (ou que la précision souhaitée soit atteinte)
- ✓ Exemple de la base 10 à la base 8
 - $20,35_{(10)} = 20_{(10)} + 0,35_{(10)} = 24,2631$

$$\begin{aligned}
 20_{(10)} &= 24_{(8)} \\
 0,35 \times 8 &= 2,80 \rightarrow 2 \\
 0,80 \times 8 &= 6,40 \rightarrow 6 \\
 0,40 \times 8 &= 3,20 \rightarrow 3 \\
 0,20 \times 8 &= 1,60 \rightarrow 1
 \end{aligned}$$

Janvier 2012

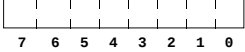
Gaëtan Rey - DUT Informatique de Nice

36

Université Nice Sophia Antipolis

Représentation des nombres

- ✓ Ordinateurs travaillent sur un nombre fixe de bits
- ✓ La notion d'octet
 - un octet = 8 bits
- ✓ La notion de mot
 - Un mot est l'unité d'information de base. Un mot est un ensemble de n bits qui représente une parmi 2^n valeurs.
 - Intel 80386 et 80486 : mot = 32 bits
 - Pentium 4 : mot = 128 bits



Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 37

Université Nice Sophia Antipolis

Représentation des nombres négatifs

- ✓ En décimal on utilise le signe “ – ”
 - Positif : 248 / Négatif : - 248
- ✓ Comment représenter un nombre négatif dans un ordinateur ?
- ✓ Contraintes
 - En binaire : seul 0 et 1 sont utilisables
 - Utilisation d'un nombre fixe de bits
 - Opérations arithmétiques aisées

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 38

Université Nice Sophia Antipolis

Méthode Signe + Valeur absolue

- ✓ Le bit de poids fort indique le signe
 - 0 si positif, 1 si négatif
- ✓ Les autres bits indiquent la valeur absolue
 - Représentation Binaire Non Signé (RNBS)
- ✓ Illustration sur 8 bits : 1 bit de signe et 7 bits pour les valeurs : $\pm [0, 127]$ soit $[-127, 127]$
 - $114_{(10)} = 0 \ 1110010_{(2)} = 0111 \ 0010_{(2)}$
Signe / Valeur absolue
 - $-100_{(10)} = 1 \ 1100100_{(2)} = 1110 \ 0100_{(2)}$
Signe / Valeur absolue
 - $0_{(10)} = 0 \ 0000000_{(2)} = 0000 \ 0000_{(2)}$
 - $-0_{(10)} = 1 \ 0000000_{(2)} = 1000 \ 0000_{(2)}$



Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 39

Université Nice Sophia Antipolis

Méthode Complément à 1

- ✓ On dispose également d'un bit de signe comme dans la méthode « signe + valeur absolue » mais pour la valeur négative on prend le complément à 1 de la valeur absolue.
 - Pour la valeur positive d'un nombre on garde la même valeur que dans la méthode « signe + valeur absolue »
 - $114_{(10)} = 0 \ 1110010_{(2)} = 0111 \ 0010_{(2)}$
 - Pour la valeur négative d'un nombre on inverse les valeurs des bits de la valeur absolue (0 devient 1 et 1 est remplacé par 0) et on a 1 pour le bit de signe
 - $-114_{(10)} = 1 \ 0001101_{(2)} = 1000 \ 1101_{(2)}$
Signe / Complément à 1 de la valeur absolue

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 40

Université Nice Sophia Antipolis

Méthode Complément à 1

- ✓ Illustration sur 8 bits : 1 bit de signe et 7 bits pour les valeurs : $\pm [0, 127]$ soit $[-127, 127]$
 - $100_{(10)} = 0 \ 1100100_{(2)} = 0110 \ 0100_{(2)}$
 - $-100_{(10)} = 1 \ 0011011_{(2)} = 1001 \ 1011_{(2)}$
Signe / Complément à 1 de la valeur absolue
 - $53_{(10)} = 0 \ 0110101_{(2)} = 0011 \ 0101_{(2)}$
 - $-53_{(10)} = 1 \ 1001010_{(2)} = 1100 \ 1010_{(2)}$
Signe / Complément à 1 de la valeur absolue
 - $0_{(10)} = 0 \ 0000000_{(2)} = 0000 \ 0000_{(2)}$
 - $-0_{(10)} = 1 \ 1111111_{(2)} = 1111 \ 1111_{(2)}$
Signe / Complément à 1 de la valeur absolue



Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 41

Université Nice Sophia Antipolis

Méthode Complément à 2

- ✓ On dispose également d'un bit de signe comme dans la méthode « signe + valeur absolue » mais pour la valeur négative on prend le complément à 2 de la valeur absolue.
 - Pour la valeur positive d'un nombre on garde la même valeur que dans la méthode « signe + valeur absolue »
 - $114_{(10)} = 0 \ 1110010_{(2)} = 0111 \ 0010_{(2)}$
 - Pour la valeur négative d'un nombre on calcule le complément à 1 et on ajoute 1 au résultat.
 - $-114_{(10)} = 1 \ 0001101_{(2)} + 1_{(2)} = 1000 \ 1110_{(2)}$
Signe / Complément à 1 de la valeur absolue Complément à 2

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 42

Méthode Complément à 2

- ✓ Illustration sur 8 bits : $[-128, 127]$
 - $100_{(10)} = 0\ 1100100_{(2)} = 0110\ 0100_{(2)}$
 - $-100_{(10)} = 1\ 0011011_{(2)} + 1_{(2)} = 1001\ 1100_{(2)}$
 - Signe / Complément à 1 Complément à 2
 - $53_{(10)} = 0\ 0110101_{(2)} = 0011\ 0101_{(2)}$
 - $-53_{(10)} = 1\ 1001010_{(2)} + 1_{(2)} = 1100\ 1011_{(2)}$
 - Signe / Complément à 1 Complément à 2
 - $0_{(10)} = 0\ 0000000_{(2)} = 0000\ 0000_{(2)}$
 - $-0_{(10)} = 1\ 1111111_{(2)} + 1_{(2)} = 1\ 0000\ 0000_{(2)}$
 - Signe / Complément à 1 Complément à 2
 - $-128_{(10)} = 1000\ 0000_{(2)}$ (on ne garde que 8 bits)

Méthode Excédent à 2^{n-1}

- ✓ Pour un nombre de n bits, on représente les nombres en leur additionnant 2^{n-1} .
 - $[-128, 127] \rightarrow [0, 255]$
- ✓ Exemple sur 8 bits (on additionne $2^7 = 128$)
 - -3 devient $-3 + 128 = 125_{(10)} = 01111101_{(2)}$
- ✓ Astuce pour le cas de l'excédent à 2^{n-1}
 - On calcule le complément à 2 du nombre et on inverse le bit de signe.

Représentation des nombres en virgule flottante

- ✓ Représenter une valeur non entière sur un nombre de bits donnés
 - Approximation des nombres réels
- ✓ Ils sont notés de la manière suivante : s.m.b^e ou
 - s est le signe (+1 ou -1)
 - m est la mantisse
 - Une mantisse normalisée n'a qu'un seul chiffre (différent de 0) avant la virgule
 - e est l'exposant, et
 - b la base

Représentation des nombres en virgule flottante

- ✓ Exemple en base 10
 - $52.23_{(10)} = +\ 5.223 \times 10^1$
 - s m b e
- ✓ Exemples en base 2
 - $7.25_{(10)} = 111.01_{(2)} = +\ 1.1101 \times 2^2_{(2)}$
 - s m b e
 - $-16,625_{(10)} = -10000,101_{(2)} = -\ 1,0000101 \times 2^4_{(2)}$
 - s m b e

La norme IEEE 754 (1985)

- ✓ Standard pour la représentation des nombres à virgule flottante en binaire
 - Portabilité entre machines et langages
 - Reproductibilité des calculs
 - Représentation des nombres spéciaux
 - Procédures d'arrondi
- ✓ Précision
 - Simple sur 32 bits (type float)
 - Double sur 64 bits (type double)
 - Étendue sur 80 bits
- ✓ IEEE 754 Converter
 - <http://www.h-schmidt.net/FloatApplet/IEEE754.html>

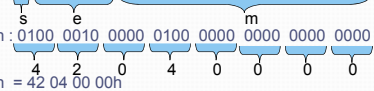
La norme IEEE 754 (1985)

- ✓ Précision simple sur 32 bits
 - Signe sur 1 bit (0 pour le + et 1 pour le -)
 - Exposant sur 8 bits en excédent 127
 - Mais 0000 0000 et 1111 1111 sont interdits
 - Mantisse sur 23 bits
 - En Représentation Binaire Non Signé,
 - Normalisé sans représentation du 1 de gauche
 - Arrondie
- ✓ Précision double sur 64 bits
 - Signe sur 1 bit
 - Exposant sur 11 bits en excédent 1023
 - Mantisse sur 52 bits

Université Nice Sophia Antipolis

La norme IEEE 754 (1985)

✓ Exemple en simple précision

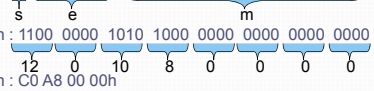
- $33.0_{(10)} = 100001.0_{(2)} = 1.00001 \times 2^5_{(2)} \text{ normalisé}$
 - Signe sur 1 bit : 0 car +
 - Exposant sur 8 bits : 1000 0100
 - Mantisse sur 23 bits : 0000 1000 0000 0000 0000 000
 - Représentation : 0 1000 0100 0000 1000 0000 0000 000
- Représentation : 
- Représentation = 42 04 00 00h

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 49

Université Nice Sophia Antipolis

La norme IEEE 754 (1985)

✓ Exemple en simple précision

- $-5.25_{(10)} = -101.01_{(2)} = -1.0101 \times 2^2_{(2)} \text{ normalisé}$
 - Signe sur 1 bit : 1 car -
 - Exposant sur 8 bits : 1000 0001
 - Mantisse sur 23 bits : 0101 0000 0000 0000 0000 000
 - Représentation : 1 1000 0001 0101 0000 0000 0000 000
- Représentation : 
- Représentation : C0 A8 00 00h

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 50

Université Nice Sophia Antipolis

Représentation des Caractères

✓ De nombreuses représentations existent

✓ Exemples :

- 1968 ASCII (Ansi X3.4, 128 caractères)
- 1972 ASCII localisé
- 1985 ASCII étendu (ISO-8859-1) : 256 caractères, prise en compte des langues européennes
- 1993 Unicode 1.1 (ISO 10646-1:1993)
- 2000 Unicode 3.0 (ISO 10646-1:2000)

✓ Chaque langage utilise sa représentation

- En C : une chaîne est une suite consécutive d'octets (au format ASCII) terminée par le caractère '\0',
- en Pascal : une chaîne a une longueur maximum de 255 caractères, car le premier octet de la chaîne code pour la longueur.

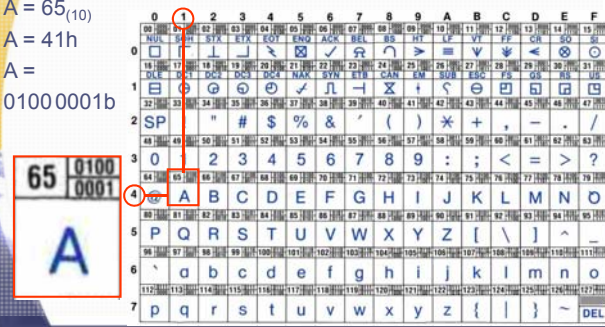
Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 51

Université Nice Sophia Antipolis

American Standard Code for Information Interchange

✓ Table des codes ASCII (7 bits, Norme ISO 646)

A = $65_{(10)}$
A = 41h
A = 0100 0001b



Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 52

Université Nice Sophia Antipolis

Plan

- ✓ Introduction
- ✓ Représentation de l'information
- ✓ Arithmétique binaire
- ✓ Algèbre de Boole
- ✓ Circuits séquentiels/Automates
- ✓ Architecture type Von Neumann
- ✓ Couche d'assemblage / Assembleur 8086
- ✓ Mécanismes d'interruptions
- ✓ Fonctions avancées de processeurs modernes
- ✓ Gestion de la mémoire
- ✓ Traduction de programmes, compilation édition de liens, chargement

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 53

Université Nice Sophia Antipolis

Addition binaire (RBNS)

✓ Elle est identique à l'addition décimale, excepté qu'un bit de retenue est levé lorsque la somme est supérieure à 1 au lieu de 9

✓ Exemples sur 8 bits

$$\begin{array}{r} 00010100 \quad 20 \\ + 00000111 \quad 7 \\ \hline 00011011 \quad 27 \end{array}$$

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 54

Université Nice Sophia Antipolis

Addition binaire (Complément à 2)

- ✓ On utilise des nombres en complément à 2 pour les nombres négatifs
- ✓ Exemples sur 5 bits

1		
00100	4	
+ 00111	+ 7	
01011	11	

- ✓ Overflow = Carry XOR Retenue_{n-2→n-1}

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 55

Université Nice Sophia Antipolis

Soustraction binaire (Complément à 2)

- ✓ Elle est plus ou moins identique à la soustraction décimale
- ✓ Exemples sur 6 bits

010111	23
- 000101	- 5
010010	18

- Complément à 2 vers Complément à 1:
 - 110001 - 1 → 110000
- Complément à 1 à signe + valeur absol
 - 110000 → 1 01111 = -15

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 56

Université Nice Sophia Antipolis

Bilan

- ✓ Addition
 - en RBNS
 - Résultat cohérent si dans l'intervalle $[0, 2^n - 1]$
 - Dépassement signalé par le positionnement du Carry à 1
 - en Complément à 2
 - Résultat cohérent si dans l'intervalle $[-2^{n-1}, 2^{n-1} - 1]$
 - Dépassement signalé par le positionnement de l'Overflow à 1
- ✓ Soustraction
 - Pourrait être réalisé par l'addition de son opposé en Complément à 2 (sauf pour -128)
- ✓ Rappel
 - Overflow = Carry XOR Retenue_{n-2→n-1}

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 57

Université Nice Sophia Antipolis

Plan

- ✓ Introduction
- ✓ Représentation de l'information
- ✓ Arithmétique binaire
- ✓ Algèbre de Boole
- ✓ Circuits séquentiels/Automates
- ✓ Architecture type Von Neumann
- ✓ Couche d'assemblage / Assembleur 8086
- ✓ Mécanismes d'interruptions
- ✓ Fonctions avancées de processeurs modernes
- ✓ Gestion de la mémoire
- ✓ Traduction de programmes, compilation édition de liens, chargement

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 58

Université Nice Sophia Antipolis

Algèbre booléenne

- ✓ Variable booléenne
 - Souvent nommée x, y ou z
 - 2 valeurs possible : vrai (1) ou faux (0)
- ✓ Opération booléenne
 - NON : noté $\neg x$ ou $\bar{x}$ ou $|x$

X	Vrai	Faux
$\neg X$	Faux	Vrai
 - OU inclusif : noté $x+y$ ou $x \vee y$

X\Y	Faux	Vrai
Faux	Faux	Vrai
Vrai	Vrai	Vrai
 - ET : noté $x.y$ ou xy ou $x \wedge y$

X\Y	Faux	Vrai
Faux	Faux	Faux
Vrai	Faux	Vrai
 - XOR exclusif : noté $x \oplus y$

X\Y	Faux	Vrai
Faux	Faux	Vrai
Vrai	Vrai	Faux

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 59

Université Nice Sophia Antipolis

Propriétés des opérations (1)

- ✓ Commutativité
 - $X.Y = Y.X$
 - $X+Y = Y+X$
- ✓ Associativité
 - $X.(Y.Z) = (X.Y).Z$
 - $X+(Y+Z) = (X+Y)+Z$
- ✓ Distributivité
 - $X.(Y+Z) = X.Y + X.Z$
 - $X+(Y.Z) = (X+Y).(X+Z)$
- ✓ Identité
 - $1.X = X$
 - $0+X = X$
- ✓ Domination / Nullité
 - $0.X = 0$
 - $1+X = 1$

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 60

Université Nice Sophia Antipolis

Propriétés des opérations (2)

- ✓ Idempotence
 - $X.X = X$
 - $X+X = X$
- ✓ Complément / Inversion
 - $\neg X.X = 0$
 - $\neg X+X = 1$
- ✓ Double négation
 - $\neg \neg X = X$
- ✓ Lois de Morgan
 - $\neg(X+Y) = \neg X . \neg Y$
 - $\neg(X.Y) = \neg X + \neg Y$
- ✓ Absorption
 - $X.(X+Y) = X$
 - $X+(X.Y) = X$

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 61

Université Nice Sophia Antipolis

Exemples de simplifications

- ✓ $X(\neg X+Y) = \neg X.X + X.Y = X.Y$
- ✓ $(X+Y).(X+\neg Y) = X.X + X.\neg Y + Y.X + Y.\neg Y$
 $= X + X(\neg Y+Y) + 0$
 $= X + X.1 = X+X = X$
- ✓ $\neg((A.B+A.D).(\neg B+\neg C)) = \neg(A.B+A.D)+\neg(\neg B+\neg C)$
 $= \neg(A.B).\neg(A.D) + B.C$
 $= (\neg A+\neg B).(\neg A+\neg D) + B.C$

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 62

Université Nice Sophia Antipolis

Fonctions Booléennes

- ✓ On appelle fonction booléenne toutes fonctions qui possèdent uniquement des variables booléennes et dont le résultat est un booléen.
- ✓ Une fonction booléenne f de n variables n'admet que 2^n combinaisons possibles de valeurs d'entrée
 - Elle peut être décrite complètement au moyen d'une table de 2^n ligne, chaque ligne indiquant la valeur de f pour une combinaison spécifique de valeurs d'entrée
 - On appelle une telle table : une table de vérité

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 63

Université Nice Sophia Antipolis

Table de vérité

- ✓ Soit $S = f(a,b,c)$ une fonction booléenne à 3 variables.
 - Donc $2^3 = 8$ combinaisons possibles
- ✓ Fonction logique :

$$S = \bar{A}.\bar{B}.C + \bar{A}.B.\bar{C} + \bar{A}.B.C + A.\bar{B}.\bar{C}$$

$$S = \bar{A}.\bar{B}.C + \bar{A}.B + A.\bar{B}.\bar{C}$$
- ✓ Nécessite une simplification

Entrées			Sortie
a	b	c	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 64

Université Nice Sophia Antipolis

Tableau de Karnaugh (1)

- ✓ Un tableau de Karnaugh est une table de vérité
 - qui sert à d'obtenir l'expression logique la plus simple de la fonction booléenne qu'elle représente
 - dans laquelle les différentes possibilités des entrées sont classées en code GRAY.
 - 0 puis 1 dans le cas d'une variable,
 - 00, 01, 11, 10 pour 2 variables.
 - On peut remarquer qu'en passant d'une case à une case adjacente, une seule variable a changé.

	C	0	0	1	1
A	B	0	1	1	0
0					
1					

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 65

Université Nice Sophia Antipolis

Tableau de Karnaugh (2)

- ✓ Pour écrire l'expression de la fonction booléenne f recherchée, on doit construire des « paquets » à l'aide des cases adjacentes contenant la valeur 1 en respectant les règles suivantes
 - Le nombre de cases dans chaque paquet doit être une puissance de 2
 - Chaque paquet doit être « centré » autour d'un axe de symétrie
 - Chaque case contenant un 1 doit appartenir à au moins un paquet
 - Chaque paquet doit être le plus grand possible

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 66

Exemple (1)

- ✓ Soit la fonction $f(x,y,z,t)$ dont le tableau de Karnaugh est le suivant :

xy \ z	t				
	0	0	1	1	
00	1	1	0	0	Axes de symétrie
01	0	0	1	1	
11	0	0	1	1	
10	1	1	0	0	

- ✓ Dans le paquet 1, y vaut toujours 0 et z vaut toujours 0
→ monôme = $\neg y \cdot \neg z$
- ✓ Dans le paquet 2, y vaut toujours 1 et z vaut toujours 1
→ monôme = $y \cdot z$
- Donc ce tableau de Karnaugh décrit la fonction
 $f = y \cdot z + \neg y \cdot \neg z = y \oplus z$

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

67

Exemple (2)

- ✓ Tableau à deux variables d'entrées

A \ B	0	1
	0	1
0	0	1
1	0	1

$$f = B$$

- ✓ Tableau à trois variables d'entrées

A \ B C	00	01	11	10
	0	1	1	0
0	1			1
1	1		1	1

$$f = \neg B + A \cdot C$$

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

68

Exemple (3)

- ✓ Tableau à quatre variables d'entrées

A B \ D C	00	01	11	10
	0	1	1	0
00		1	1	1
01		1	1	
11				
10				1

$$S = C \cdot \neg A + D \cdot \neg C \cdot \neg B$$

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

69

Exemple (4)

- ✓ Tableau à quatre variables d'entrées avec des états non utilisés

- **Etats non utilisés** : états qui existent en théorie mais qui en pratique ne peuvent jamais apparaître pour des raisons physiques ou mécaniques.

A B \ D C	00	01	11	10
	0	1	1	0
00	1	1		
01	1	-	-	-
11		1	1	
10				

$$S = \neg A \cdot \neg D + B \cdot C$$

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

70

Plan

- ✓ Introduction
- ✓ Représentation de l'information
- ✓ Arithmétique binaire
- ✓ Algèbre de Boole
- ✓ **Circuits séquentiels/Automates**
- ✓ Architecture type Von Neumann
- ✓ Couche d'assemblage / Assembleur 8086
- ✓ Mécanismes d'interruptions
- ✓ Fonctions avancées de processeurs modernes
- ✓ Gestion de la mémoire
- ✓ Traduction de programmes, compilation édition de liens, chargement

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

71

Portes logiques (1)

- ✓ Élément de base de la logique numérique
- ✓ Les fonctions logiques sont implémentées par l'interconnexion de portes
- ✓ Fonctionnent sur des signaux électriques
 - Lorsque la valeur d'entrée change, le signal de sortie correct apparaît presque instantanément.
 - Retardé par le délai de propagation des signaux au travers de la porte (délai de porte)

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

72

Université Nice Sophia Antipolis

Portes logiques (2)

- ✓ Matérialisent les opérateurs de bases
- ✓ 2, 3, 4 ou même d'avantage d'entrées

ET/ AND

OU/ OR

NOT

Non ET/ NAND

Non OU/ NOR

XOR

Janvier 2012 Gaétan Rey - DUT Informatique de Nice 73

Université Nice Sophia Antipolis

Circuits combinatoires (1)

- ✓ C'est un ensemble de portes interconnectées dont les valeurs de sortie ne dépendent que des valeurs d'entrée à cet instant
- ✓ Il se compose de n entrée binaires et de m sorties binaires

Sortie = f(Entrée)

Janvier 2012 Gaétan Rey - DUT Informatique de Nice 74

Université Nice Sophia Antipolis

Circuits combinatoires (2)

- ✓ On peut le définir de 3 manières
 - Table de vérité
 - Symboles graphiques (logigrammes)
 - Equations booléennes
- ✓ Exemple : Circuit à 2 entrées E1 et E2 :

E1	E2	0	1
0	0	1	0
1	0	0	1

Janvier 2012 Gaétan Rey - DUT Informatique de Nice 75

Université Nice Sophia Antipolis

Exemple de circuit logique (1)

$(A + B).(A + \overline{C})$

Janvier 2012 Gaétan Rey - DUT Informatique de Nice 76

Université Nice Sophia Antipolis

Exemples de circuits logiques (2)

$S_1 = AB + C.DE + F$

$S_2 = \overline{(\overline{A.B}).(\overline{C.D.E}).F}$

Janvier 2012 Gaétan Rey - DUT Informatique de Nice 77

Université Nice Sophia Antipolis

Additionneur (1)

- ✓ Comment faire un additionneur 8 bits ?

10
+ 12

22

- ✓ Etapes
 - 1) faire un demi-additionneur 1bit
 - 2) faire un additionneur 1 bit
 - 3) faire un additionneur 8 bits

Janvier 2012 Gaétan Rey - DUT Informatique de Nice 78

Université Nice Sophia Antipolis

Demi-additionneur binaire

✓ Circuit logique à deux entrées (A et B) et deux sorties (S et R).

$R_n = A_n B_n$

$S_n = \neg A_n B_n + A_n \neg B_n$

A	B	R	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 79

Université Nice Sophia Antipolis

Additionneur complet

A _n	B _n	R _{n-1}	R _n	S _n
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

$R_n = A_n B_n + B_n R_{n-1} + A_n R_{n-1}$

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 80

Université Nice Sophia Antipolis

Additionneur complet

A _n	B _n	R _{n-1}	R _n	S _n
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

$S_n = \neg R_{n-1} (\neg A_n B_n + A_n \neg B_n) + R_{n-1} (\neg A_n \neg B_n + A_n B_n)$

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 81

Université Nice Sophia Antipolis

Additionneur complet

A _n	B _n	R _{n-1}	R _n	S _n
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 82

Université Nice Sophia Antipolis

Additionneur (2)

✓ un additionneur 8 bits

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 83

Université Nice Sophia Antipolis

Circuits logiques

✓ Sorties 3 états :

- 0 ou 1 (Si E = 1) ou ouvert (Si E = 0)
- Le signal E est le signal de validation de la sortie S

$S = A.E$

✓ Les sorties des circuits logiques nommés mémoires sont toutes de type 3 états

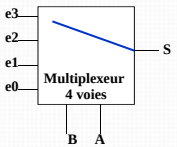
✓ Par l'intermédiaire de circuits nommés BUS le Microprocesseur peut accéder aux contenus de chaque circuit mémoire.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 84

Université Nice Sophia Antipolis

Circuits combinatoires (1)

- ✓ Multiplexeur (ici 4 voies)
 - circuit permettant de concentrer sur une même voie de transmission différents types de liaisons en sélectionnant une entrée parmi N
 - L'entrée e_i est propagée sur la sortie S suivant la valeur de A et de B.



B	A	S
0	0	e0
0	1	e1
1	0	e2
1	1	e3

$$S = e0.\bar{B}.\bar{A} + e1.\bar{B}.A + e2.B.\bar{A} + e3.B.A$$

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 85

Université Nice Sophia Antipolis

Circuits combinatoires (2)

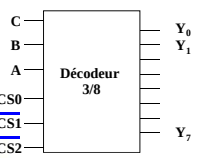
- ✓ Codeurs décodeurs n vers 2^n :
 - Commander un organe différent parmi 2^n au moyen de n bits.
 - Il allume une et une seule sortie à la fois, celle correspondant à la valeur binaire donnée en entrée
- ✓ Exemple avec un décodeur 3 vers 8
 - 3 entrées (A,B,C)
 - 8 sorties (2^3) de Y_0 à Y_7
 - Si $A=0, B=1$ et $C=1 \rightarrow 110_{(2)} = 6_{(10)} \rightarrow Y_6$ est activé

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 86

Université Nice Sophia Antipolis

Circuits combinatoires (3)

- ✓ Codeurs décodeurs 3 vers 8 avec contrôle
 - Les lignes CS_i permettent de valider le circuit. Ici pour que le circuit fonctionne ces lignes doivent être à vrai c'est à dire à 100.
 - Ces lignes permettent des montage plus complexes à plusieurs décodeurs.



$Y_0 = \bar{C}.\bar{B}.\bar{A}$	$Y_0 = \bar{C}.\bar{B}.\bar{A}.CS0.\bar{CS1}.\bar{CS2}$
$Y_1 = \bar{C}.\bar{B}.A$	$Y_1 = \bar{C}.\bar{B}.A.CS0.\bar{CS1}.\bar{CS2}$
$Y_7 = C.B.A$	$Y_7 = C.B.A.CS0.CS1.CS2$

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 87

Université Nice Sophia Antipolis

Plan

- ✓ Introduction
- ✓ Représentation de l'information
- ✓ Arithmétique binaire
- ✓ Algèbre de Boole
- ✓ Circuits séquentiels/Automates
- ✓ **Architecture type Von Neumann**
- ✓ Couche d'assemblage / Assembleur 8086
- ✓ Mécanismes d'interruptions
- ✓ Fonctions avancées de processeurs modernes
- ✓ Gestion de la mémoire
- ✓ Traduction de programmes, compilation édition de liens, chargement

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 88

Université Nice Sophia Antipolis

Architecture de Von Neumann (1)

- ✓ Le modèle d'architecture de la plupart des ordinateurs actuels provient du travail effectué par John Von Neumann
 - Il propose la première description d'un ordinateur dont le programme est stocké dans sa mémoire
- ✓ Pour traiter une information, le microprocesseur seul ne suffit pas. Il faut l'insérer au sein d'un système de traitement de l'information.

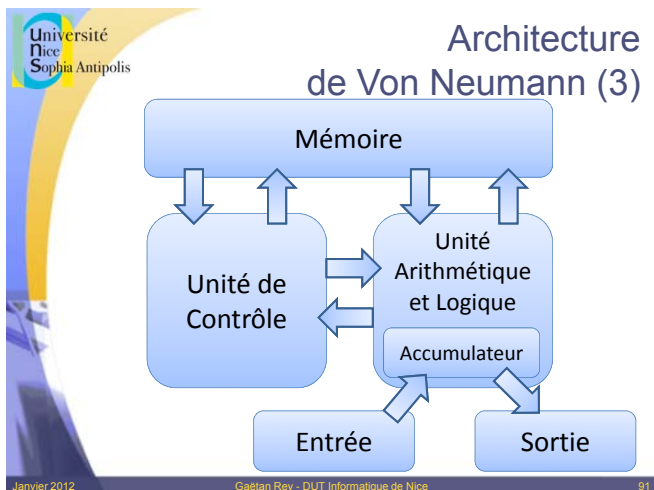
Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 89

Université Nice Sophia Antipolis

Architecture de Von Neumann (2)

- ✓ Elle décompose l'ordinateur en 4 parties
 - **L'unité de contrôle** ou unité de commande, chargée du séquençage des opérations.
 - **L'unité de traitement** ou unité arithmétique et logique (UAL) : son rôle est d'effectuer les opérations de base.
 - **La mémoire** qui contient à la fois les données et le programme.
 - La mémoire se divise entre mémoire volatile (programmes et données en cours de fonctionnement) et mémoire permanente (programmes et données de base de la machine).
 - **Les dispositifs d'entrée-sortie**, qui permettent de communiquer avec le monde extérieur.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 90



Université Nice Sophia Antipolis

Les bus (1)

- ✓ Un bus est un ensemble de fils qui assure
 - la transmission du même type d'information et
 - la communication entre les divers composants.
- ✓ On distingue 3 grandes catégories de Bus
 - **bus de données** : bidirectionnel qui assure le transfert des informations entre le microprocesseur et son environnement, et inversement.
 - Son nombre de lignes est égal à la capacité de traitement du microprocesseur.
 - bus d'adresses ...
 - bus de commande ...

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 92

Université Nice Sophia Antipolis

Les bus (2)

- ✓ On distingue 3 grandes catégories de Bus (suite)
 - bus de données ...
 - **bus d'adresses** : unidirectionnel qui permet la sélection des informations à traiter dans un espace mémoire (ou espace adressable) qui peut avoir 2^n emplacements, avec n = nombre de conducteurs du bus d'adresses.
 - **bus de commande** : constitué par quelques conducteurs qui assurent la synchronisation des flux d'informations sur les bus des données et des adresses.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 93

Université Nice Sophia Antipolis

L'unité de commande (1)

- ✓ **L'unité de commande**
 - Elle séquence le déroulement des instructions.
 - Elle effectue la recherche en mémoire de l'instruction, le décodage, l'exécution et la préparation de l'instruction suivante.
- ✓ **Le Compteur Ordinal (ou de programme) (CO)**
 - Il contient l'adresse de la prochaine instruction à exécuter.
 - Au chargement du programme, contient l'adresse de la 1ère instruction. Il est modifié :
 - soit par incrémentation automatique dans le cas où les adresses des instructions se suivent.
 - soit par chargement de l'adresse de branchement dans le cas de sauts programmés.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 94

Université Nice Sophia Antipolis

L'unité de commande (2)

- ✓ **Registre d'instruction et décodeur d'instruction**
 - chacune des instructions à exécuter est rangée dans le registre instruction puis est décodée par le décodeur d'instruction.
- ✓ **Séquenceur**
 - Il organise l'exécution des instructions au rythme d'une horloge.
 - Il élabore les signaux de synchronisation internes ou externes (bus de commandes) du microprocesseur

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 95

Université Nice Sophia Antipolis

L'unité de commande (3)

- ✓ **Phase de recherche de l'instruction**
 - Le programme est en mémoire centrale
 - Lancement du programme
 - Compteur Ordinal ← Adresse de la 1ère instruction (OS)
 - Le séquenceur va générer les microcommandes pour placer l'instruction dans le Registre d'Instruction
 - La microcommande pour incrémenter le compteur ordinal
 - Compteur Ordinal ← Adresse de la prochaine instruction

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 96

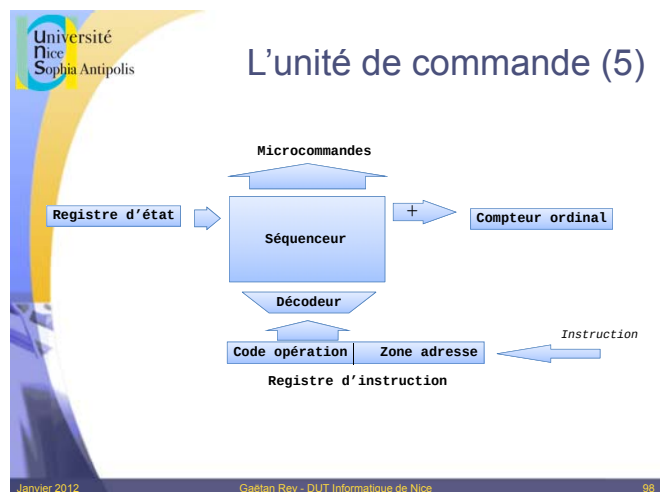
Université Nice Sophia Antipolis

L'unité de commande (4)

- ✓ Phase de **traitement de l'instruction**
 - Le décodeur analyse la zone opération de l'instruction
 - Le séquenceur exécute les microcommandes correspondantes
 - Exemple : l'addition

```
MOV AL,[0200]
ADD AL,[0300]
MOV [0400],AL
```

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 97

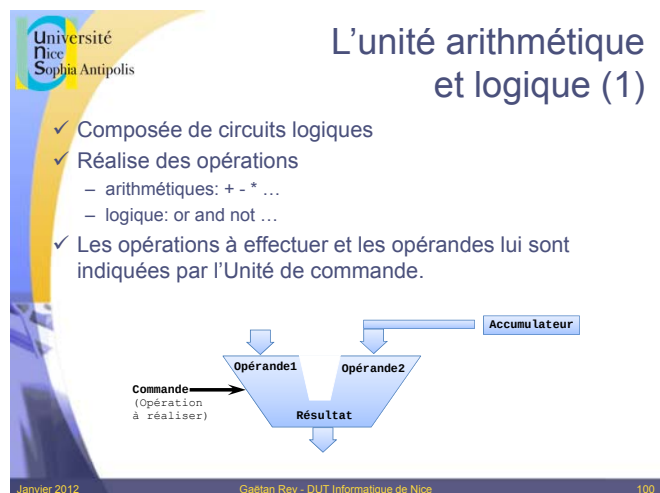


Université Nice Sophia Antipolis

Unité de traitement

- ✓ UAL
 - circuit complexe qui assure les fonctions logiques (AND, OR, CMP...) ou arithmétiques (ADD, SUB, INC...)
- ✓ Registre d'état
 - Indicateur (généralement composé de 8 bits) dont l'état dépend du résultat de la dernière opération effectuée.
 - Indicateurs d'état ou flags.
- ✓ Les accumulateurs
 - registres de travail qui servent à stocker les opérandes et les résultats à la fin de l'opération.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 99



Université Nice Sophia Antipolis

L'unité arithmétique et logique (2)

- ✓ UAL et registre d'état
 - Exemple: Overflow, Zéro, Carry

Instruction de comparaison en assembleur 8086

↑

CMP AL, 0

} Si AL = 0
alors ZF=1
sinon ZF=0
fini

- Flag ZF du registre d'état

- ✓ NB: Une UAL fait un nombre limité d'opérations → fonctions manquantes à programmer

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 101

Université Nice Sophia Antipolis

Registre d'état (1)

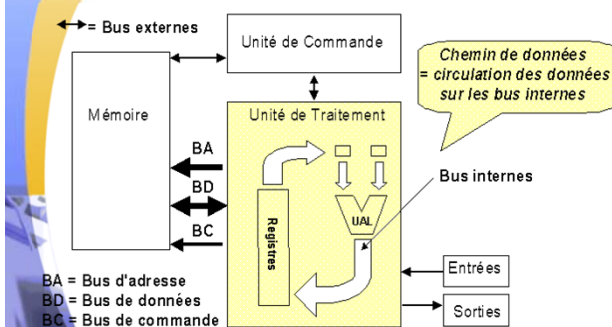
- ✓ Les instructions de branchement conditionnels utilisent les indicateurs : bits spéciaux positionnés par l'UAL après certaines opérations.
- ✓ ZF (Zero Flag)
 - mis à 1 lorsque le résultat de la dernière opération est zéro. Sinon, ZF est positionné à 0.
- ✓ CF (Carry Flag)
 - Indicateur de report (retenue). Il est positionné par les instructions ADD, SUB et CMP (entiers naturels).
 - CF = 1 s'il y a une retenue

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 102

Registre d'état (2)

- ✓ **SF (Sign Flag)**
 - Positionné à 1 si le bit de poids fort du résultat d'une addition ou soustraction est 1; sinon SF=0. Utile pour la manipulation des entiers relatifs, le bit de poids fort donne le signe du résultat.
- ✓ **OF (Overflow Flag) Indicateur de débordement**
 - OF=1 si le résultat d'une addition ou soustraction donne un nombre qui n'est pas codable en relatif dans l'accumulateur (par exemple si l'addition de 2 nombres positifs donne un codage négatif).
- ✓ **Exemple**
 - CMP = SUB, mais ne stocke pas le résultat de la soustraction (positionner les indicateurs)
 - CMP AX, 5
 - ZF = 1 si AX contient 5, et ZF = 0 si AX ≠ 5.

Architecture de Von Neumann (4)



La mémoire (1)

- ✓ Une mémoire peut être représentée comme une armoire de rangement constituée de différents tiroirs.
 - Chaque tiroir représente alors une case mémoire qui peut contenir un seul élément : des données.
 - Une adresse identifie la case mémoire et chaque donnée est accessible grâce à son adresse.

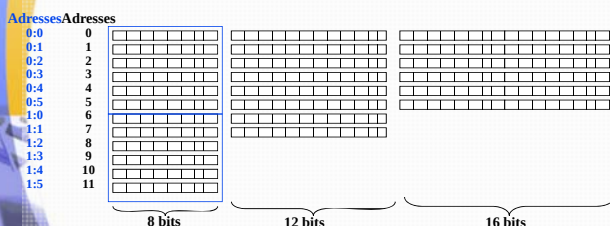
Adresse	Case mémoire
5=101	00000000
4=100	00000000
3=011	00000000
2=010	11101110
1=001	10001110
0=000	00001000

La mémoire (2)

- ✓ **Stocke** des informations utilisées par le processeur
- ✓ **Juxtaposition** de cellules (la plus petite quantité de mémoire adressable).
 - Chaque cellule est numérotée (**adresse**)
 - Généralement les mémoires sont adressables par octets
- ✓ **Taille d'une cellule**
 - octet : 8 bits (byte)
 - word : 16 bits,
 - Double word : 32 bits,
 - Quad word 64 bits : en fonction de machines (16 ou 32 bits)
- ✓ **Notation :**
 - Si M désigne une cellule mémoire
 - [M] représente son contenu

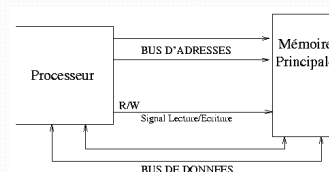
Organisation d'une mémoire

- ✓ **Mémoire de 96 bits :**
 - Organisation en cellules élémentaires 8bits, 12 bits ou 16 bits :



Opérations effectuées sur les mémoires

- ✓ **Lecture (adresse)**
 - lit l'emplacement donné par adresse et ne modifie pas l'emplacement mémoire.
- ✓ **Écriture (valeur, adresse)**
 - met à l'emplacement adresse la valeur placée en paramètre



Université
Nice
Sophia Antipolis

Exemple d'accès mémoire

- ✓ « Ajouter 5 au contenu de la case mémoire d'adresse 180 »
 1. Le processeur lit et décode l'instruction;
 2. le processeur demande à la mémoire la contenu de l'emplacement 180;
 3. la valeur lue est rangée dans l'accumulateur;
 4. l'unité de traitement (UAL) ajoute 5 au contenu de l'accumulateur;
 5. le contenu de l'accumulateur est écrit en mémoire à l'adresse 180.
- ✓ C'est l'unité de commande qui déclenche chacune de ces actions dans l'ordre. L'addition est effectuée par l'UAL.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 109

Université
Nice
Sophia Antipolis

Caractéristiques des mémoires (1)

- ✓ Capacité : taille de la mémoire en bits (ou octets).
- ✓ Format des données : nombre de bits que l'on peut mémoriser par case mémoire.
- ✓ Volatilité
 - laps de temps durant lequel elle maintient les informations
 - Alimentation des mémoires : l'information stockée est volatile si elle risque d'être altérée par un défaut d'alimentation électrique et non volatile dans le cas contraire.
 - mémoire de travail de l'ordinateur, mémoire sur support magnétique

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 110

Université
Nice
Sophia Antipolis

Caractéristiques des mémoires (2)

- ✓ Temps d'accès
 - temps pour accéder à l'information une fois que l'instruction a été lancée.
 - de l'ordre de la ns pour les mémoires actuelles et de la ms pour les supports magnétiques
- ✓ Type d'accès
 - accès direct à l'information, accès aux mots-mémoire par leur adresse.
 - accès séquentiel
- ✓ Temps de cycle : intervalle minimum entre 2 demandes successives de lecture/écriture.
- ✓ Débit : nombre maximum d'informations lues ou écrites par seconde

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 111

Université
Nice
Sophia Antipolis

Hierarchie des mémoires (1)

Registres : situés au niveau du processeur, stockage des opérandes et des résultats
Mémoire cache : mémoire rapide destinée à accélérer l'accès de la mémoire principale.
Mémoire principale : contient les programmes (instructions et données)
Mémoire d'appui : mémoire intermédiaire entre mémoire principale et mémoire de masse, mémoire de cache.
Mémoire de masse : stockage permanent, sauvegarde. HDD, CDROM...

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 112

Université
Nice
Sophia Antipolis

Hierarchie des mémoires (2)

- ✓ Machine idéale
 - taille mémoire illimitée
 - temps d'attente nul
- ✓ Problème de coût => Hiérarchie de mémoires
- ✓ Le processeur va chercher dans les différentes mémoires
- ✓ L'optimisation est faite par le gestionnaire de mémoire

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 113

Université
Nice
Sophia Antipolis

Caractéristiques des mémoires (3)

- ✓ Mémoire vive: RAM
 - Mémoire volatile
 - Cycle de lecture/écriture (@ bus d'adresse, V bus de donnée, CS, R/W)
 - Mémoire vive statique → SRAM
 - Mémoire vive dynamique → DRAM
 - Mémoire vive dynamique Synchrones → SDRAM
- ✓ Mémoire morte: ROM, PROM, EPROM, EEPROM
 - A lecture seule,
 - Conserve l'information
 - ROM : information stockée de manière définitive,
 - PROM : Programmable par l'utilisateur
 - EPROM -EEPROM : Programmable + effacement par UV ou électriquement

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 114

Université
Nice
Sophia Antipolis

La mémoire virtuelle (1)

- ✓ Utilisation de la mémoire secondaire «comme si» c'était de la mémoire centrale
 - Disponible à partir des 80386 et 68030
 - La gestion se fait au niveau du système d'exploitation
- ✓ Système de gestion de mémoire virtuelle très sophistiqué
 - Pagination
 - mono dimensionnelle
 - mappage de l'espace d'adressage sur la mémoire physique.
 - Les zones mémoires lues sur la mémoire secondaire sont les pages

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 115

Université
Nice
Sophia Antipolis

La mémoire virtuelle (2)

- ✓ Utilisation de la mémoire secondaire «comme si» c'était de la mémoire centrale
 - Disponible à partir des 80386 et 68030
 - La gestion se fait au niveau du système d'exploitation
- ✓ Système de gestion de mémoire virtuelle très sophistiqué
 - Segmentation
 - possibilité d'avoir plusieurs espaces d'adressage virtuels
 - chaque espace est indépendant : les segments
 - les segments sont de taille variable (limités à 64k)
 - Sur les 80X86 la segmentation a été utilisée pour résoudre le problème des adresses sur 20 bits

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 116

Université
Nice
Sophia Antipolis

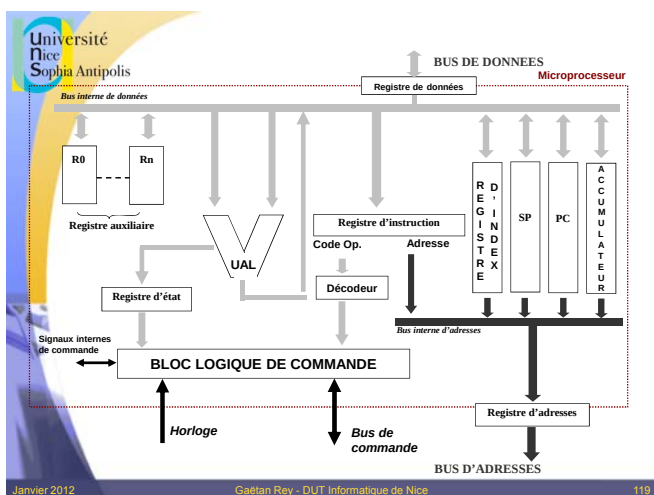
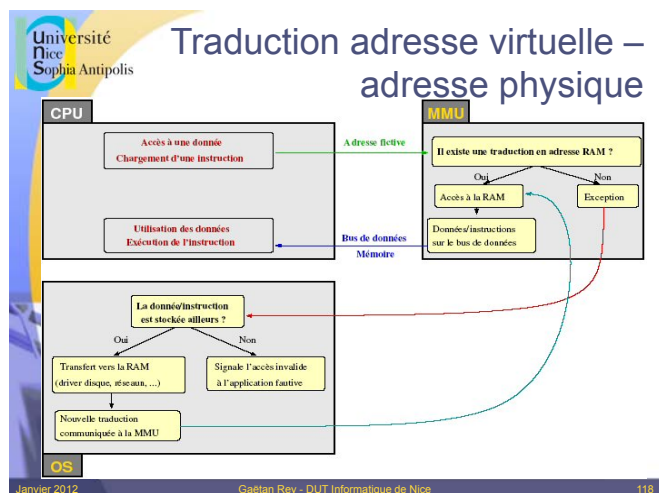
La pagination

- ✓ Séparation entre espace d'adressage et emplacement mémoire
 - Une page est une zone de mémoire de taille fixe (4k)
 - Déplacement des pages de la mémoire secondaire vers la mémoire centrale -> *Pagination*
- ✓ La Pagination
 - se fait au fur et à mesure des besoins
 - Pb du défaut de page
 - La pagination est transparente à l'utilisateur

Réalisation de la pagination

- à la charge du système d'exploitation

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 117



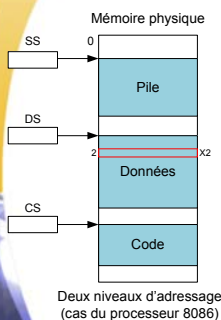
Université
Nice
Sophia Antipolis

Mémoires rapides : registres

- ✓ Mémoire **locale et privée** du processeur
- ✓ Information temporaire
- ✓ Capacité limitée (1 à 128 bits)

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 120

Mémoire segmentée



Les instructions manipulent des adresses réduites, qui désignent des objets à l'intérieur d'un segment.

Dans le cas du 8086, on a plusieurs segments qui servent à stocker le code du programme, ses données statiques, les données de la pile, etc.

Organisation de la mémoire centrale

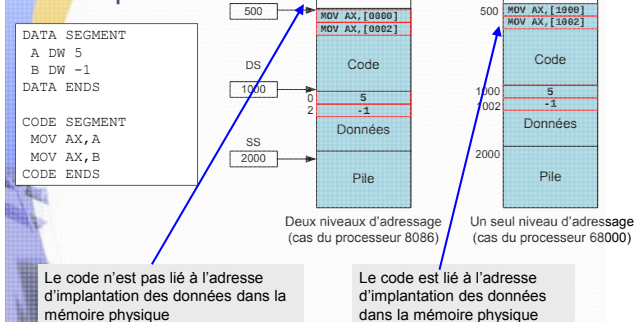
- ✓ Nécessaire d'implanter dans un microprocesseur
 - espace pour le système
 - espace pour l'utilisateur
 - espace pour les entrées /sorties
- ✓ Taille suffisante pour ranger les programmes
- ✓ Séparation entre les différents espaces (utilisateur, système)

Modèle d'organisation de la mémoire

- ✓ Modèle mémoire linéaire (1 niveau d'adressage)
 - valable pour OS qui nécessitent une vitesse performante
 - protection moindre
 - limité dans le cas des applications de plus de 4Go de mémoire
- ✓ Modèle de mémoire segmentée (2 niveaux d'adressage)
 - utilise les registres de segment et quatre niveaux de protection
 - les OS peuvent se protéger des applications
 - le processeur gère la mémoire et vérifie les protections

Machine à 1 et 2 niveaux d'adressages

Exemple



Instructions (1)

- ✓ Une instruction
 - Opération élémentaire
 - Information sur :
 - ce que l'instruction fait (add, sub, mov ...)
 - sur quelles données (AL, AX, 10h ...)
- ✓ les instructions sont spécifiques à un microprocesseur
 - 680XX, 80X86, SPARC ...
 - Exemple
 - Pentium 220 instructions.
 - Techno MMX +57 instructions

Instructions (2)

Codage d'une instruction

Zone Opération	Opérande
----------------	----------

- Zone Opération
 - Code Opération → Codage unique
 - jeu d'instruction limité
- Opérande
 - adresse(s) de la donnée
 - Adressage implicite → utilisation d'un accumulateur
 - Exemple : B0 01 → MOV AL, 01

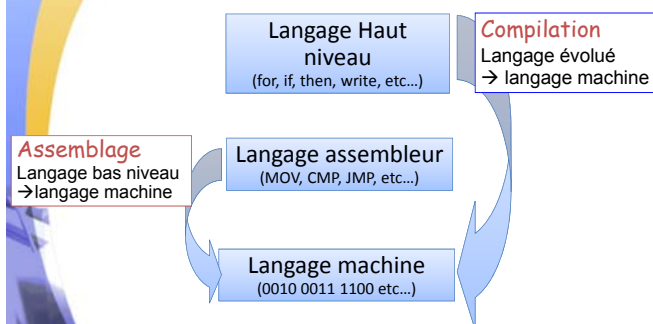
Instructions (3)

- ✓ Les instructions et leurs opérandes (paramètres) sont stockés en mémoire principale
 - La taille totale d'une instruction (nombre de bits nécessaires pour la représenter en mémoire) dépend du type d'instruction et aussi du type d'opérande.
- ✓ Chaque instruction est toujours codée sur un nombre entier d'octets, afin de faciliter son décodage par le processeur. Une instruction est composée de deux champs :
 - le code opération, qui indique au processeur quelle instruction réaliser;
 - le champ opérande qui contient la donnée, ou la référence à une donnée en mémoire (son adresse).

Plan

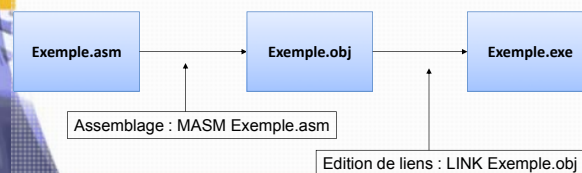
- ✓ Introduction
- ✓ Représentation de l'information
- ✓ Arithmétique binaire
- ✓ Algèbre de Boole
- ✓ Circuits séquentiels/Automates
- ✓ Architecture type Von Neumann
- ✓ **Couche d'assemblage / Assembleur 8086**
- ✓ Mécanismes d'interruptions
- ✓ Fonctions avancées de processeurs modernes
- ✓ Gestion de la mémoire
- ✓ Traduction de programmes, compilation édition de liens, chargement

Définitions



Définitions

- ✓ **Assembleur** : mnémoniques associées au langage machine (JMP, MOV, CMP...)
- ✓ **Langage machine** : ordres (en binaire) compréhensible par un processeur donné.
- ✓ **Exécutable** : suite d'instruction en langage machine



Processeurs 80x86

- ✓ Micro-processeurs 80x86 équipent les PC et compatibles
 - Premiers PC (début 80) = 8086, micro-processeur 16 bits
 - Puis 80286, 80386, 80486, Pentium...
- ✓ Améliorations
 - Augmentation de la fréquence d'horloge, de la largeur des bus d'adresses et de données
 - Ajout de nouvelles instructions et de registres
- ✓ Compatibilité ascendante
 - Un programme écrit dans le langage machine du 286 peut s'exécuter sur un 486 (l'inverse est faux)

Les bus du 8086

- ✓ Les Bus
 - Bus de données : 16 bits
 - Bus d'adresses : 20 bits
- ✓ Exemples

MICROPROCESSEUR	BUS DE DONNEES	BUS D'ADRESSES
8088	8	20
8086	16	20
80286	16	24
80386SX	16	32
80386DX	32	32
80486	32	32
Pentium	64	32

Registres du 8086 (1)

- ✓ Un registre sert à stocker les données nécessaires à l'exécution d'un programme par le microprocesseur.
 - On n'accède pas à un registre avec une adresse, mais avec son appellation.
- ✓ Exemple : l'instruction MOV registre1,registre2
 - elle a pour effet de copier le contenu du registre2 dans le registre1 (Le contenu du registre1 est écrasé).
 - Registre1 et registre2 doivent être de même taille

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

133

Registres du 8086 (2)

- ✓ Registres de travail :
 - **AX** : Accumulateur utilisé pour les opérations arithmétiques.
 - **BX** : Base pour les adressages
 - **CX** : Compteur pour le comptage
 - **DX** : Data pour les données
- ✓ Registres de travail ont une taille de 16 bits

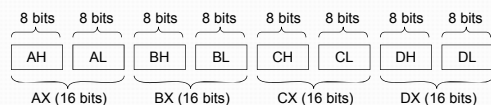
Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

134

Registres du 8086 (3)

- ✓ Les Registres AX, BX, CX et DX (16 bits) peuvent également être utilisés comme 2 registres 8 bits



- ✓ **AX et AL (AH) sont le même registre**
 - Si on écrit dans AL (AH), on efface une partie de AX
 - Il en est de même pour BX et (BL, BH), CX et (CL, CH) et DX et (DL, BDH)

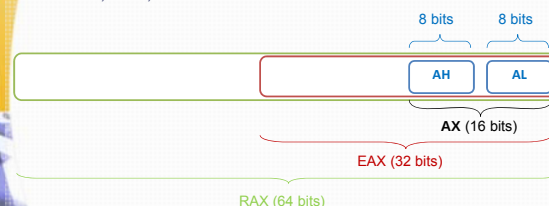
Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

135

Registres du 8086 (4)

- ✓ Depuis les 386 ont dispose également de 4 nouveaux registres de données de 32 bits
 - EAX, EBX, ECX et EDX



Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

136

Registres du 8086 (5)

- ✓ Registres d'offset :
 - **SI** : Source Index utilisé lors d'opérations sur chaînes de caractères, associé à DS.
 - **DI** : Destination Index utilisé lors d'opérations sur les chaînes de caractères, associé à DS. si utilisé comme index et à ES lors de manipulation de chaînes.
 - **IP** : Instruction Pointer associé à CS, il indique la prochaine instruction à exécuter, et ne peut être modifié.
 - **BP** : Base Pointer associé à SS pour accéder aux données de la pile lors d'appels d'un sous-programme.
 - **SP** : Stack Pointer associé à SS, il indique l'adresse du dernier élément de la pile.

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

137

Le registre IP

- ✓ **IP : pointeur d'instruction**
- ✓ Le registre IP du processeur conserve l'adresse de la prochaine instruction à exécuter
- ✓ Le processeur effectue les actions suivantes pour chaque instruction :
 - lire et décoder l'instruction à l'adresse IP;
 - $IP \leftarrow IP + \text{taille de l'instruction};$
 - exécuter l'instruction.

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

138

Université
Nice
Sophia Antipolis

Registres du 8086 (6)

- ✓ Registres de segment :
 - **CS** : Code Segment
 - il contient l'adresse du segment de la mémoire de programme
 - **DS** : Data Segment
 - il contient l'adresse du segment de mémoire de données
 - **SS** : Stack Segment
 - il contient l'adresse du segment de mémoire de la pile
 - **ES** : Extra Segment
 - segment supplémentaire pour adresser des données.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 139

Université
Nice
Sophia Antipolis

Sens des mouvements de données

- ✓ La plupart des instructions spécifient des mouvements de données entre la mémoire principale et le microprocesseur.
- ✓ En langage symbolique, on indique toujours la destination, puis la source.
 - L'instruction **MOV AX, [0110]** transfère le contenu de l'emplacement mémoire 0110H dans l'accumulateur,
 - L'instruction **MOV [0112], AX** transfère le contenu de l'accumulateur dans l'emplacement mémoire 0112.
 - L'instruction MOV s'écrit donc toujours :
 - MOV destination, source

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 140

Université
Nice
Sophia Antipolis

Jeu d'instructions du 8086 (1)

- ✓ Instructions d'affectation :
 - Déclenchent un transfert de données entre l'un des registres du processeur et la mémoire principale (MP)
 - **Lecture en MP** : de la Mémoire Principale vers le CPU
 - **Ecriture en MP** : du CPU vers la Mémoire Principale
 - L'instruction MOV
 - MOV Registre, Registre → Par ex: MOV CX, DX
 - MOV Registre, Mémoire → Par ex: MOV AX, [MEM]
 - MOV Registre, Constante → Par ex: MOV AH, 9
 - MOV Mémoire, Registre → Par ex: MOV [MEM], AX
 - MOV Mémoire, Constante → Par ex: MOV [MEM], 5
 - ...

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 141

Université
Nice
Sophia Antipolis

Jeu d'instructions du 8086 (2)

- ✓ Instructions arithmétiques et logiques
 - Opération entre une donnée et l'accumulateur. Le résultat est placé dans le premier paramètre.
 - L'instruction Addition (**ADD**): $AX \leftarrow AX + \text{donnée}$
 - ADD Registre, Registre → Par ex: ADD AX, BX
 - ADD Registre, Mémoire → Par ex: ADD BX, [MEM]
 - ADD Registre, Constante → Par ex: ADD AH, 12
 - ADD Mémoire, Registre → Par ex: ADD [MEM], BX
 - ADD Mémoire, Constante → Par ex: ADD [MEM], 11
 - L'instruction Soustraction (**SUB**): $AX \leftarrow AX - \text{donnée}$
 - SUB Registre, Registre → Par ex: SUB AX, BX
 - SUB Registre, Mémoire → Par ex: SUB BX, [MEM]
 - SUB Registre, Constante → Par ex: SUB AH, 12
 - SUB Mémoire, Registre → Par ex: SUB [MEM], BX
 - SUB Mémoire, Constante → Par ex: SUB [MEM], 11

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 142

Université
Nice
Sophia Antipolis

Jeu d'instructions du 8086 (3)

- ✓ Instructions arithmétiques et logiques
 - Opération entre une donnée et l'accumulateur. Le résultat est placé dans le premier paramètre.
 - L'instruction Incrémentement (**INC**): $AX \leftarrow AX + 1$
 - INC Registre → Par ex: INC CX
 - INC Mémoire → Par ex: INC [MEM]
 - L'instruction Décrémentement (**DEC**): $AX \leftarrow AX - 1$
 - DEC Registre → Par ex: DEC CX
 - DEC Mémoire → Par ex: DEC [MEM]
 - L'instruction Négation (**NEG**): $AX \leftarrow -AX$
 - NEG Registre → Par ex: NEG CX
 - NEG Mémoire → Par ex: NEG [MEM]

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 143

Université
Nice
Sophia Antipolis

Jeu d'instructions du 8086 (4)

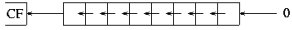
- ✓ Instructions arithmétiques et logiques
 - Les instructions de décalage et de rotations (Sxx / Rxx)
 - Sxx Registre, 1 → Par ex: Sxx DL, 1
 - Sxx Registre, CL → Par ex: Sxx AX, CL
 - Sxx Mémoire, 1 → Par ex: Sxx [MEM], 1
 - Sxx Mémoire, CL → Par ex: Sxx [MEM], CL
 - Décalage vers la gauche ou vers la droite les bits de l'accumulateur. Opérations utilisées
 - pour décoder bit à bit des données
 - pour diviser ou multiplier rapidement par une puissance de 2.
 - décaler AX de n bits vers la gauche revient multiplier AX par 2^n (un décalage vers la droite revient à diviser par 2^n)

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 144

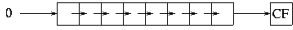
Université
Nice
Sophia Antipolis

Décalage et Rotations (1)

- ✓ **SHL registre, 1 (Shift Left)**
 - Décale les bits du registre d'une position vers la gauche. Le bit de gauche est transféré dans l'indicateur CF. Les bits introduits à droite sont à zéro.



- ✓ **SHR registre, 1 (Shift Right)**
 - Comme SHL mais vers la droite. Le bit de droite est transféré dans CF. SHL et SHR peuvent être utilisé pour multiplier/diviser des entiers *naturels* (et non des relatifs car le bit de signe est perdu)

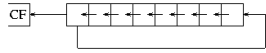


Janvier 2012 Gaétan Rey - DUT Informatique de Nice 145

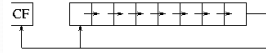
Université
Nice
Sophia Antipolis

Décalage et Rotations (2)

- ✓ **ROL registre, 1 (Rotate Left)**
 - Rotation vers la gauche : le bit de poids fort passe à droite, et est aussi copié dans CF. Les autres bits sont décalés d'une position.



- ✓ **ROR registre, 1 (Rotate Right)**
 - Rotation vers la droite : le bit de poids faible passe à gauche, et est aussi copié dans CF. Les autres bits sont décalés d'une position.



Janvier 2012 Gaétan Rey - DUT Informatique de Nice 146

Université
Nice
Sophia Antipolis

Décalage et Rotations (3)

- ✓ **RCL registre, 1 (Rotate Carry Left)**
 - Rotation vers la gauche en passant par l'indicateur CF. CF prend la place du bit de poids faible; le bit de poids fort part dans CF.



- ✓ **RCR registre, 1 (Rotate Carry Right)**
 - Rotation vers la droite en passant par l'indicateur CF. CF prend la place du bit de poids fort; le bit de poids faible part dans CF.



- ✓ RCL et RCR sont utiles pour lire ou écrire le contenu d'un registre.

Janvier 2012 Gaétan Rey - DUT Informatique de Nice 147

Université
Nice
Sophia Antipolis

Jeu d'instructions du 8086 (5)

- ✓ Instructions arithmétiques et logiques
 - Les instructions logiques (AND, OR, XOR et TEST)
 - AND Registre, Registre → Par ex: AND AX, DX
 - AND Registre, Mémoire → Par ex: AND BX, [MEM]
 - AND Registre, Constante → Par ex: AND AH, 7
 - AND Mémoire, Registre → Par ex: AND [MEM], AX
 - AND Mémoire, Constante → Par ex: AND [MEM], 11
 - Pas de propagation de retenue lors de ces opérations
 - chaque bit du résultat est calculé indépendamment des autres
 - AX <- AX ou BX
 - Les instructions logiques (NOT)
 - AND Registre → Par ex: NOT CX
 - AND Mémoire → Par ex: NOT [MEM]

Janvier 2012 Gaétan Rey - DUT Informatique de Nice 148

Université
Nice
Sophia Antipolis

Opérations logiques

- ✓ **AND**
 - AND est souvent utilisé pour forcer certains bits à 0.
 - Après **AND AX, FF00**, l'octet de poids faible de AX vaut 00, tandis que l'octet de poids fort est inchangé.
- ✓ **OR**
 - OR est souvent utilisé pour forcer certains bits à 1.
 - Après **OR AX, FF00**, l'octet de poids fort de AX vaut FF, tandis que l'octet de poids faible est inchangé.
- ✓ **XOR**
 - XOR est souvent utilisé pour inverser certains bits.
 - Après **XOR AX, FFFF**, tous les bits de AX sont inversés

Janvier 2012 Gaétan Rey - DUT Informatique de Nice 149

Université
Nice
Sophia Antipolis

Jeu d'instructions du 8086 (6)

- ✓ Instructions de comparaison
 - Compare un registre avec une donnée et positionne les indicateurs. La donnée et la valeur du registre ne sont pas modifiées
 - L'instruction Comparaison (**CMP**):
 - CMP Registre, Registre → Par ex: CMP CX, DX
 - CMP Registre, Mémoire → Par ex: CMP AX, [MEM]
 - CMP Registre, Constante → Par ex: CMP AH, 12
 - CMP Mémoire, Registre → Par ex: CMP [MEM], AX
 - CMP Mémoire, Constante → Par ex: CMP [MEM], 11

Janvier 2012 Gaétan Rey - DUT Informatique de Nice 150

Jeu d'instructions du 8086 (7)

- ✓ Instructions de branchement
 - Un programme exécute normalement les instructions les une à la suite des autres. En assembleur, le registre IP indique l'adresse de la prochaine instruction à modifier
 - Une instruction de branchement modifie la valeur de IP pour exécuter une autre instruction (cas des boucles, tests, ...)
 - Il y a 2 catégories de branchement
 - **Branchements inconditionnels** : on effectue toujours le saut
 - **Branchements conditionnels** : on effectue le saut si une condition est satisfaite

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

151

Jeu d'instructions du 8086 (8)

- ✓ Instructions de branchement inconditionnels (JMP):
 - JMP Mémoire → Par ex: JMP [MEM]
 - JMP Offset → Par ex: JMP ADRESSE
- ✓ Instructions de branchement conditionnels (Jxxx):
 - Jxxx label → Par ex: JMP fin
 - Attention, l'espace franchi par les instructions de branchement conditionnels ne doit pas dépasser 127 octets

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

152

Jeu d'instructions du 8086 (9)

- ✓ Instructions de branchement conditionnels (Jxxx):
 - **JE** Jump if Equal (ou JZ) : saut si ZF = 1;
 - **JNE** Jump if Not Equal (ou JNZ) : saut si ZF = 0;
 - **JG** Jump if Greater : saut si ZF = 0 et SF = 0;
 - **JLE** Jump if Lower or Equal : saut si ZF=1 ou SF ≠ OF;
 - **JS** Jump if Sign : saut si SF=1;
 - **JNS** Jump if Not Sign : saut si SF=0;
 - **JA** Jump if Above : saut si CF=0 et ZF=0;
 - **JBE** Jump if Below or Equal : saut si CF=1 ou ZF=1.
 - **JB** Jump if Below : saut si CF=1.
 - **JO** Jump if Overflow : saut si OF=1

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

153

Les Indicateurs (1)

- ✓ Les instructions de branchement conditionnels utilisent les indicateurs, qui sont des bits spéciaux positionnés par l'UAL après certaines opérations.
- ✓ Les indicateurs sont regroupés dans le registre d'état du processeur. Ce registre n'est pas accessible globalement par des instructions ; chaque indicateur est manipulé individuellement par des instructions spécifiques.
- ✓ On se concentrera ici les indicateurs suivant
 - ZF, CF, SF et OF.

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

154

Les Indicateurs (2)

- ✓ ZF Zero Flag
 - Cet indicateur est mis à 1 lorsque le résultat de la dernière opération est zéro.
 - Lorsque l'on vient d'effectuer une soustraction (ou une comparaison), ZF=1 indique que les deux opérandes étaient égaux. Sinon, ZF est positionné à 0.

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

155

Les Indicateurs (3)

- ✓ CF Carry Flag
 - C'est l'indicateur de report (retenue), qui intervient dans les opérations d'addition et de soustractions sur des entiers naturels. Il est positionné en particulier par les instructions ADD, SUB et CMP.
 - CF = 1 s'il y a une retenue après l'addition ou la soustraction du bit de poids fort des opérandes.
 - Exemples (sur 4 bits pour simplifier) :

0100	1100	1111
+ 0110	+ 0110	+ 0001
1010	0010	0000

CF = 0

CF = 1

CF = 1

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

156

Les Indicateurs (4)

✓ SF Sign Flag

- SF est positionné à 1 si le bit de poids fort du résultat d'une addition ou soustraction est 1 ; sinon SF=0.
- SF est utile lorsque l'on manipule des entiers relatifs, car le bit de poids fort donne alors le signe du résultat.
- Exemples (sur 4 bits) :

$\begin{array}{r} 0100 \\ + 0110 \\ \hline 1010 \end{array}$	$\begin{array}{r} 1100 \\ + 0110 \\ \hline 0010 \end{array}$	$\begin{array}{r} 1111 \\ + 0001 \\ \hline 0000 \end{array}$
SF = 1	SF = 0	SF = 0

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

157

Les Indicateurs (5)

✓ OF Overflow Flag

- Indicateur de débordement OF=1 si le résultat d'une addition ou soustraction donne un nombre qui n'est pas codable en relatif dans l'accumulateur
 - Par exemple si l'addition de 2 nombres positifs donne un codage négatif
- Overflow = Carry XOR Retenue_{n-2} → n-1

$\begin{array}{r} 0100 \\ + 0110 \\ \hline 1010 \end{array}$	$\begin{array}{r} 1100 \\ + 0110 \\ \hline 0010 \end{array}$	$\begin{array}{r} 1111 \\ + 0001 \\ \hline 0000 \end{array}$
OF = 1	OF = 0	OF = 0

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

158

Jeu d'instructions du 8086 (10)

✓ Instructions de modification des indicateurs

- L'instruction CLC (sans opérande)
 - Elle met à 0 l'indicateur CF
- L'instruction SLC (sans opérande)
 - Elle met à 1 l'indicateur CF
- L'instruction CMC (sans opérande)
 - Elle inverse l'indicateur CF (opération NOT)

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

159

Correspondance avec le langage C

- ✓ La table suivante établit un parallèle entre les instructions arithmétiques et logiques du 8086 et les opérateurs du langage C (lorsque ces derniers agissent sur des variables non signées).

Opérateur C	Instruction 80x86
+	ADD
-	SUB
<<	SHL
>>	SHR
	OR
&	AND
^	XOR

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

160

Codage des instructions (1)

- ✓ Les instructions et leurs opérandes (paramètres) sont stockés en mémoire principale.
- ✓ La taille totale d'une instruction (nombre de bits nécessaires pour la représenter en mémoire) dépend du type d'instruction et aussi du type d'opérande.
- ✓ Chaque instruction est toujours codée sur un nombre entier d'octets, afin de faciliter son décodage par le processeur.

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

161

Codage des instructions (2)

- ✓ Une instruction est composée de deux champs :

- le **code opération (Obligatoire)**, qui indique au processeur quelle instruction réaliser ;
- le **champ opérande** qui contient la donnée, ou la référence à une donnée en mémoire (son adresse).

champ code opération	champ code opérande
-------------------------	------------------------

- ✓ Selon la manière dont la donnée est spécifiée, c'est à dire selon le mode d'adressage de la donnée, une instruction sera codée par 1, 2, 3 ou 4 octets.

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

162

Code opération : assembleur

Symbole	Code Op.	Octets	
MOV AX, valeur	B8	3	AX ← valeur
MOV AX, [adr]	A1	3	AX ← contenu de l'adresse adr.
MOV [adr], AX	A3	3	range AX à l'adresse adr.
ADD AX, valeur	05	3	AX ← AX + valeur
ADD AX, [adr]	03 06	4	AX ← AX + contenu de adr.
SUB AX, valeur	2D	3	AX ← AX - valeur
SUB AX, [adr]	2B 06	4	AX ← AX - contenu de adr.
SHR AX, 1	D1 E8	2	décale AX à droite.

On utilise des programmes spéciaux, appelés *assembleurs*, pour traduire automatiquement le langage symbolique en code machine.

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

163

Exemple de programme

✓ Programme en langage machine implanté à l'adresse mémoire 0100H

- A1 10 01 03 06 12 01 A3 14 01
- Ce programme additionne le contenu de deux cases mémoire et range le résultat dans une troisième

Adresse	Contenu MP	Langage Symbolique	Explication
0100H	A1 10 01	MOV AX, [0110H]	;Charger AX avec le contenu de 0110H.
0103H	03 06 12 01	ADD AX, [0112H]	;Ajouter le contenu de 0112H à AX (résultat dans AX).
0107H	A3 14 01	MOV [0114H], AX	;Ranger AX en 0114H.

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

164

Modes d'adressage

- ✓ **Adressage** = méthode de localisation des opérandes
- ✓ **Mode d'adressage** = manière d'interpréter les bits d'un champs d'adresse en vue de la localisation de l'opérande

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

165

Adressage implicite (par registre)

- ✓ Pas d'accès mémoire pour les opérandes
- ✓ L'instruction contient seulement le code opération, sur 1 ou 2 octets.

code opération
(1 ou 2 octets)

- ✓ L'instruction porte sur des registres ou spécifie une opération sans opérande
 - Exemple : INC AX

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

166

Adressage immédiat

- ✓ Adressage dans lequel la valeur de l'opérande figure directement dans l'instruction sans avoir besoin de faire un nouvel accès mémoire.

code opération (1 ou 2 octets)	valeur (1 ou 2 octets)
-----------------------------------	---------------------------

- ✓ Exemple :

MOV AL, 10 ; Met la valeur 10 dans AL
ADD AX, 5 ; Ajoute 5 à la valeur contenue dans AX

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

167

Adressage direct

- ✓ Un des opérandes est l'emplacement mémoire dont l'adresse figure dans l'instruction.
 - Une fois l'instruction lue, il faut aller faire un accès mémoire pour obtenir l'opérande désiré.



- ✓ Exemple :

MOV AX, [130h] ; Met la valeur contenue dans l'emplacement d'adresse 130h dans AX

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

168

Université Nice Sophia Antipolis

Adressage indirect

- ✓ L'adresse de la variable est contenue dans un registre de base ou d'index : 2 accès mémoires.

Exemple :

```
MOV BX, offset VAR
MOV AX, [BX] ; Met dans AX la valeur contenue dans l'emplacement mémoire dont l'adresse est spécifiée dans l'emplacement mémoire d'adresse BX.
```

- Le type de la donnée doit être accordé avec le registre utilisé

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 169

Université Nice Sophia Antipolis

Adressage basé

- ✓ Similaire à l'adressage indirect par registre sauf qu'un déplacement est ajouté à la base.
- adresse effective = adresse base + déplacement

Exemple :

```
MOV BX, OFFSET DVAR
MOV AX, [BX+4]
ADD AX, [BP+4]
```

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 170

Université Nice Sophia Antipolis

Adressage indexé

- ✓ On utilise un registre d'index SI ou DI plus un déplacement
- L'adressage est noté par des crochets
- Utile pour le parcours de tableaux

Exemple :

```
MOV SI, 2
MOV AX, T[SI]
```

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 171

Université Nice Sophia Antipolis

Modes d'adressage : résumé

- ✓ Par registre ou implicite
 - Add AX, BX
- ✓ Immédiat
 - Add AX, valeur
- ✓ Direct
 - Add AX, [adresse]
- ✓ Indirect
 - Mov BX, adresse
 - Add AX, [BX]
- ✓ Indexé
 - Add, [adresse+index]

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 172

Université Nice Sophia Antipolis

Temps d'exécution

- ✓ Chaque instruction nécessite un certain nombre de cycles d'horloges pour s'effectuer.
- ✓ Le nombre de cycles dépend de la complexité de l'instruction et aussi du mode d'adressage
 - il est plus long d'accéder à la mémoire principale qu'à un registre du processeur.
- ✓ La durée d'un cycle dépend bien sûr de la fréquence d'horloge de l'ordinateur.
 - Plus l'horloge bat rapidement, plus un cycle est court et plus on exécute un grand nombre d'instructions par seconde.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 173

Université Nice Sophia Antipolis

Structure d'un programme (1)

```
data SEGMENT ; data est le nom du segment de donnees
; directives de declaration de donnees
data ENDS ; fin du segment de donnees

ASSUME DS:data, CS:code

code SEGMENT ; code est le nom du segment d'instructions
debut: ; 1ere instruction, avec l'etiquette debut
; suite d'instructions
code ENDS

END debut ; fin du programme, avec l'etiquette de la premiere instruction.
```

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 174

Structure d'un programme (2)

- ✓ Un programme écrit en assembleur comprend des définitions de données et des instructions, qui s'écrivent chacune sur une ligne de texte.
- ✓ Les données sont déclarées par des directives, mots clef spéciaux que comprend l'assembleur.
- ✓ Les directives qui déclarent des données sont regroupées dans le segment de données, qui est délimité par les directives SEGMENT et ENDS.
- ✓ Les instructions sont placées dans un autre segment, le segment de code.

Structure d'un programme (3)

- ✓ La première instruction du programme (dans le segment d'instruction) doit toujours être repérée par une étiquette.
- ✓ Le fichier doit se terminer par la directive END avec le nom de l'étiquette de la première instruction
 - ceci permet d'indiquer à l'éditeur de liens quelle est la première instruction à exécuter lorsque l'on lance le programme.
- ✓ Les points-virgules indiquent des commentaires.

Déclaration de variables (1)

- ✓ On déclare les variables à l'aide de directives.
- ✓ L'assembleur attribue à chaque variable une adresse.
 - Dans le programme, on repère les variables grâce à leur nom.
- ✓ Les noms des variables (comme les étiquettes) sont composés d'une suite de 31 caractères au maximum, commençant obligatoirement par une lettre.
 - Le nom peut comporter des majuscules, des minuscules, des chiffres, plus les 3 caractères @ ? _
- ✓ Lors de la déclaration d'une variable, on peut lui affecter une valeur initiale.

Déclaration de variables (2)

- ✓ 2 directives permettent de déclarer des variables
 - DB (Define Byte) : 1 octet
 - DW (Define Word) : 2 octets
 - Les valeurs initiales peuvent être données en hexadécimal (terminée par H) ou en binaire (terminée par b)
- ✓ Exemple d'utilisation :

```
data SEGMENT
    entree DW 15 ; 2 octets initialises a 15
    sortie DW ? ; 2 octets non initialises
    cle DB ? ; 1 octet non initialise
    nega DB -1 ; 1 octet initialise a -1
    truc DW 0FOAH ; en hexa
    masque DB 01110000b ; en binaire
data ENDS
```

Déclaration de variables (3)

- ✓ Tableaux
 - Il est aussi possible de déclarer des tableaux, c'est à dire des suite d'octets ou de mots consécutifs.
- ✓ Pour cela, utiliser plusieurs valeurs initiales :

```
data SEGMENT
    machin db 10, 0FH          ; 2 fois 1 octet
    chose db -2, 'ALORS'
```

data ENDS

 - la déclaration de la variable chose :
 - un octet à -2 (=FEH),
 - une suite de caractères.
 - L'assembleur n'impose aucune convention pour la représentation des chaînes de caractères

Déclaration de variables (4)

- ✓ Tableaux
- Après chargement de ce programme, la mémoire aura le contenu suivant :
- | | | |
|----|-----|-----------|
| 10 | 0AH | ← machine |
|----|-----|-----------|

10	0AH	← machin
15	0FH	← machin + 1
-2	FEH	← chose
A	41H	← chose + 1
L	4CH	← chose + 2
O	4FH	← chose + 3
R	52H	← chose + 4
S	53H	← chose + 5

- Pour changer 'ALORS' en 'OLARS', on peut écrire :
MOV chose + 1, 'O'
MOV machin + 5, 'A'
- chose+1 est une constante (valeur connue au moment de l'assemblage) : l'instruction générée par l'assembleur pour
MOV chose+1, AL est MOV [adr], AL .

Université Nice Sophia Antipolis

Déclaration de variables (5)

- ✓ Tableaux et la directive dup
 - on utilise la directive dup lorsque l'on veut déclarer un tableau de n cases soit
 - Non initialisées
 - Toutes initialisées à la même valeur,
 - Exemples


```
tab DB 100 dup (15) ; 100 octets valant 15
zzz DW 10 dup (?) ; 10 mots de 16 bits non initialisés
```
- ✓ Constantes
 - La directive EQU (ou =) permet de déclarer une constante


```
Const1 EQU "Avec du Texte"
Const2 = 1000
```

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 181

Université Nice Sophia Antipolis

Segmentation de la mémoire (1)

- ✓ Largeur du bus d'adresse est de 20 bits
 - Possibilité d'adressage mémoire $2^{20} = 1 \text{ Mo}$
- ✓ Le pointeur d'instruction (IP) fait 16 bits
 - Possibilité d'adresser $2^{16} = 64 \text{ Ko}$
 - Cela ne couvre pas la mémoire
- ✓ On utilise deux registres pour indiquer une adresse au processeur

Registre de Segment
Registre de Déplacement
- ✓ Un segment de mémoire est une zone mémoire adressable avec une valeur fixée du registre de segment
 - Un segment a donc une taille maximale de 64 Ko.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 182

Université Nice Sophia Antipolis

Segmentation de la mémoire (2)

- ✓ Chaque segment débute à l'endroit spécifié par un registre spécial (registre segment)
- ✓ Le déplacement permet de trouver une information à l'intérieur du segment.
- ✓ Exemples
 - lecture du code d'une instruction : CS:IP
 - CS registre segment
 - IP déplacement
 - accès aux données : DS
 - `MOV AX,[1045]` = lecture du mot mémoire d'adresse DS:1045H

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 183

Université Nice Sophia Antipolis

Segmentation de la mémoire (3)

- ✓ Registres de déplacement sert à sélectionner une information dans un segment.
 - Dans le segment de code défini par CS
 - Le compteur de programme IP joue ce rôle.
 - CS:IP permet d'accéder à une information dans le segment de code.
 - Dans les segments de données défini par DS
 - Les deux index SI ou DI jouent ce rôle.
 - Le déplacement peut être aussi une constante.
 - DS:SI ou DS:DI permettent d'accéder à une information dans le segment de données.
 - Dans le segment de pile défini par SS
 - Les registres SP (stack pointer) et BP (base pointer) jouent ce rôle.
 - SS:SP ou SS:BP permettent d'accéder à une information dans le segment de pile.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 184

Université Nice Sophia Antipolis

Segmentation de la mémoire (4)

- ✓ Initialisation des registres segment
 - Dans ce cours, nous n'écrirons pas de programmes utilisant plus de 64 Ko de code et 64 Ko de données, ce qui nous permettra de n'utiliser qu'un seul segment de chaque type.
 - La valeur des registres CS et de DS sera fixée une fois pour toute au début du programme.
 - Le programmeur en assembleur doit se charger de l'initialisation de DS, c'est-à-dire de lui affecter l'adresse du segment de données à utiliser.
 - Par contre, le registre CS sera automatiquement initialisé sur le segment contenant la première instruction au moment du chargement en mémoire du programme

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 185

Université Nice Sophia Antipolis

Segmentation de la mémoire (5)

- ✓ La directive ASSUME permet d'indiquer à l'assembleur quel est le segment de données et celui de code, afin qu'il génère des adresses correctes.
- ✓ Le programme doit commencer, avant toute référence au segment de données, par initialiser le registre segment DS, de la façon suivante :


```
MOV AX, nom_segment_de_donnees
MOV DS, AX
```
- ✓ Il serait plus simple de faire `MOV DS, nom_segment_de_donnees` mais cette instruction n'existe pas.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 186

Université
Nice
Sophia Antipolis

Structure du programme (1)

```

FILE SEGMENT STACK
DW 256 DUP(?)
Base EQU 0
FILE ENDS

DATA SEGMENT
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE, DS:DATA, SS:PILE

main:
MOV AX, DATA
MOV DS, AX

MOV AX, PILE
MOV SS, AX
MOV SP, Base

CODE ENDS
END main

```

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 187

Université
Nice
Sophia Antipolis

Structure du programme (2)

```

FILE SEGMENT STACK
DW 256 DUP(?)
Base:
PILE ENDS

DATA SEGMENT
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE, DS:DATA, SS:PILE

main:
MOV AX, DATA
MOV DS, AX

MOV AX, PILE
MOV SS, AX
MOV SP, Base

CODE ENDS
END main

```

Déclaration du segment de pile

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 188

Université
Nice
Sophia Antipolis

Structure du programme (3)

```

FILE SEGMENT STACK
DW 256 DUP(?)
Base:
PILE ENDS

DATA SEGMENT
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE, DS:DATA, SS:PILE

main:
MOV AX, DATA
MOV DS, AX

MOV AX, PILE
MOV SS, AX
MOV SP, Base

CODE ENDS
END main

```

Déclaration du segment de données

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 189

Université
Nice
Sophia Antipolis

Structure du programme (4)

```

FILE SEGMENT STACK
DW 256 DUP(?)
Base:
PILE ENDS

DATA SEGMENT
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE, DS:DATA, SS:PILE

main:
MOV AX, DATA
MOV DS, AX

MOV AX, PILE
MOV SS, AX
MOV SP, Base

CODE ENDS
END main

```

Déclaration du segment de code
Avec initialisation des registres de segment

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 190

Université
Nice
Sophia Antipolis

Structure du programme (5)

```

FILE SEGMENT STACK
DW 256 DUP(?)
Base:
PILE ENDS

DATA SEGMENT
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE, DS:DATA, SS:PILE

main:
MOV AX, DATA
MOV DS, AX

MOV AX, PILE
MOV SS, AX
MOV SP, Base

CODE ENDS
END main

```

Etiquette du programme principal

Indication de l'étiquette où commence le programme principal

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 191

Université
Nice
Sophia Antipolis

La pile (1)

- ✓ Les piles offrent un nouveau moyen d'accéder à des données en mémoire principale, qui est très utilisé pour stocker temporairement des valeurs.
- ✓ Une pile est une zone de mémoire particulière définie dans son propre segment dont le registre est SS.
- ✓ Un pointeur (SP) qui conserve l'adresse du sommet de la pile.
- ✓ Elle mémorise les adresses d'appel et retour de sous programmes (cf la partie sur les procédures)

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 192

La pile (2)

- ✓ Structure de « rangement » de données
 - zone mémoire spécialisée
 - fonctionnement LIFO (Last In, First Out)
- ✓ 2 instructions pour accéder à la pile



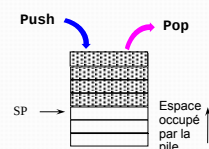
Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

193

La pile (3)

- ✓ Le registre SS (Stack Segment)
 - C'est le registre segment qui contient l'adresse du segment de pile courant
 - Il est initialisé au début du programme et reste fixé par la suite.
- ✓ Le registre SP (Stack Pointer) pointe sur le dernier bloc occupé de la pile



Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

194

La pile (4)

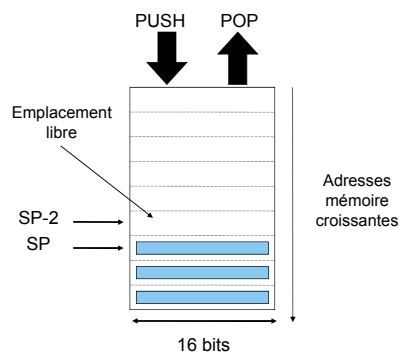
- ✓ PUSH registre
 - Empile le contenu du registre sur la pile.
 - $PUSH : SP \leftarrow SP - 2$
- ✓ POP registre
 - Retire la valeur en haut de la pile et la place dans le registres spécifié.
 - $POP : SP \leftarrow SP + 2$
- ✓ Exemple : transfert de AX vers BX en passant par la pile.
 - $PUSH AX$; empile le contenu de AX sur la pile
 - $POP BX$; dépile le 1^{er} élément dans la pile dans BX

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

195

La pile (5)



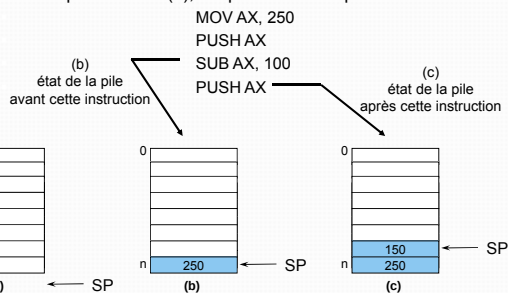
Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

196

La pile (6)

- ✓ Illustration du fonctionnement de la pile
 - Quand la pile est vide (a), SP pointe sous la pile



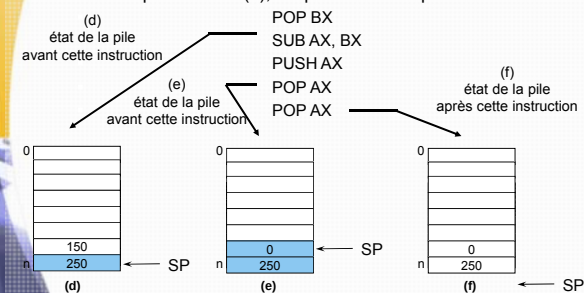
Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

197

La pile (7)

- ✓ Illustration du fonctionnement de la pile
 - Quand la pile est vide (a), SP pointe sous la pile



Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

198

La pile (8)

✓ Déclaration d'une pile

- Pour utiliser une pile en assembleur, il faut déclarer un segment de pile, et y réserver un espace suffisant. Ensuite, il est nécessaire d'initialiser les registres SS et SP pour pointer sous le sommet de la pile.
- Exemple : déclaration d'une pile de 512 octets :

```
seg_pile SEGMENT stack ; mot clef stack car pile
        DW 256 dup (?)
        Base_Pile: ; étiquette base de la pile
seg_pile ENDS
```
- Le mot clef "stack" après la directive SEGMENT, qui indique à l'assembleur qu'il s'agit d'un segment de pile.

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

199

La pile (9)

✓ Initialisation de la pile

- Après les déclarations précédentes, on utilisera la séquence d'initialisation :

```
ASSUME SS:seg_pile
MOV AX, seg_pile
MOV SS, AX ; init Stack Segment
MOV SP, Base_Pile ; pile vide
```
- Le registre SS s'initialise de façon similaire au registre DS ; par contre, on peut accéder directement au registre SP.
- Afin d'initialiser SP, il faut repérer l'adresse du bas de la pile ; c'est le rôle de l'étiquette **Base_Pile**

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

200

Appels systèmes DOS (1)

✓ SAISIR UN CARACTERE

- Cette fonction permet de lire un caractère sur l'entrée standard. Ce caractère est récupéré sous la forme de son code ASCII dans AL.
- Entrée
 - AH = 01h
- Sortie
 - AL = caractère entré au clavier
- Exemple

```
MOV AH,01
INT 21H ; résultat dans AL sous forme de car ASCII
```

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

201

Appels systèmes DOS (2)

✓ AFFICHER UN CARACTERE

- Cette fonction permet de sortir (afficher) un caractère sur la sortie standard. Ce caractère doit être stocké sous la forme de son code ASCII dans DL.
- Entrée
 - AH = 02h
 - DL = Code ASCII du caractère à afficher
- Sortie
 - Aucune
- Exemple

```
MOV DL,'A'
MOV AH,02
INT 21H
```

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

202

Appels systèmes DOS (3)

✓ AFFICHER UNE CHAINE DE CARACTERES

- Cette fonction permet de sortir (afficher) une chaîne de caractères sur la sortie standard. Cette chaîne doit être stockée sous la forme d'une séquence d'octets correspondant aux codes ASCII des caractères composant la chaîne. La fin de la chaîne de caractères doit être signalée à l'aide du caractère '\$'.
- Entrée
 - AH = 09h
 - DS = Adresse de segment de la chaîne de caractères
 - DX = Adresse d'offset de la chaîne de caractères
- Sortie
 - Aucune
- Note: si vous insérez à la suite dans la chaîne les deux caractères ASCII n° 10 et 13; vous obtiendrez un retour à la ligne.

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

203

Les Boucles (1)

✓ IF THEN ELSE

- ✓ Si (ax==1)
- ✓ bx = 10;
- ✓ Sinon {
- ✓ bx = 0;
- ✓ cx = 10;
- ✓ }

✓ En assembleur, on obtient

```
If:    CMP AX, 1
       JNE Else
Then:  MOV BX,10
       JMP EndIf
Else:  MOV BX,0
       MOV CX,10
EndIf:
```

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

204

Université Nice Sophia Antipolis

Les Boucles (2)

✓ La boucle FOR

- bx = 0 ;
- Pour K = 0 jusqu'à 10
- bx = bx + K ;

✓ En assembleur, on obtient

```

MOV BX, 0
MOV CX, 0
For:   CMP CX, 10
        JG EndFor
        ADD BX, CX
        INC CX
        JMP For
EndFor:

```

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 205

Université Nice Sophia Antipolis

Les Boucles (3)

✓ La boucle WHILE

- bx = 5 ;
- Tant que bx > 0
- faire bx = bx - 1 ;

✓ En assembleur, on obtient

```

MOV BX, 5
While : CMP BX, 0
        JLE EndWhile
        DEC BX
        JMP While
EndWhile:

```

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 206

Université Nice Sophia Antipolis

Les Boucles (4)

✓ La boucle REPEAT

- bx = 10 ;
- Répéter
- bx = bx - 1 ;
- jusqu'à bx <= 0

✓ En assembleur, on obtient

```

MOV BX, 10
Repeat1: DEC BX
        CMP BX, 0
        JG Repeat1
Endrepeat1:

```

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 207

Université Nice Sophia Antipolis

Les Boucles (5)

✓ La boucle REPEAT

- bx = 10 ;
- Répéter
- bx = bx - 1 ;
- jusqu'à bx <= 0

✓ En assembleur, on obtient

```

MOV BX, 0
MOV CX, 10
Repeat2: ADD BX, CX
        LOOP Repeat2;
EndRepeat2:

```

- Si le nombre de répétitions est connu et au moins égal à 1
- Ici on utilise BX et CX pour remplacer la variable bx. Donc on doit faire ADD BX, CX

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 208

Université Nice Sophia Antipolis

Les Boucles (6)

✓ Le test SWITCH

```

Switch (bx) {
case 1: ax = 1 ; break;
case 2: ax = 5 ; break ;
case 3: ax = 10; break;
default: ax = 0;
}

```

✓ En assembleur, on obtient

```

CMP BX, 1
JNZ case2
MOV AX, 1
JMP endswitch
case2: CMP BX, 2
        JNZ case3
        MOV AX, 5
        JMP endswitch
case 3: CMP BX, 3
        JNZ default
        MOV AX, 10
        JMP endswitch
default: MOV AX, 0
endswitch:

```

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 209

Université Nice Sophia Antipolis

Spécification de la taille des données

✓ Dans certains cas, l'adressage indirect est ambigu. Par exemple, si l'on écrit l'exemple ci-dessous, l'assembleur ne sait pas si l'instruction concerne 1 ou 2 octets consécutifs.

```
MOV [BX], 0 ; range 0 à l'adresse spécifiée par BX
```

✓ Afin de lever l'ambiguïté, on doit utiliser une directive spécifiant la taille de la donnée à transférer :

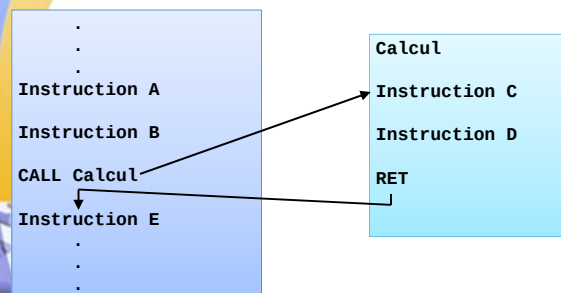
```
MOV byte ptr [BX], val ; concerne 1 octet
MOV word ptr [BX], val ; concerne 1 mot de 2 octets
```

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 210

Procédures (1)

- ✓ Notion de procédure équivalente
 - à la notion de fonction en C
 - à la notion de sous programme dans d'autres langages
- ✓ Une procédure
 - est une suite d'instruction effectuant une action précise
 - Regroupées par commodité
 - Permet d'éviter de les écrire plusieurs fois
 - est repérée par l'adresse de sa première instruction
 - peut être appelée par
 - un programme
 - une autre procédure

Procédures (2)



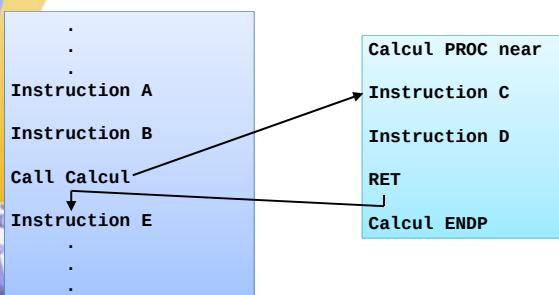
Procédures (3)

- ✓ Appel d'une procédure est effectuée par
CALL adresse_debut_procedure ;
- ✓ La fin d'une procédure est marqué par
RET
- ✓ Déclaration d'une procédure :
Procedure PROC [NEAR/FAR]
...
Procedure ENDP
- ✓ La procédure et programme principal sont
 - dans le même segment de code : NEAR
 - dans 2 segments différents : FAR

Procédures (4)

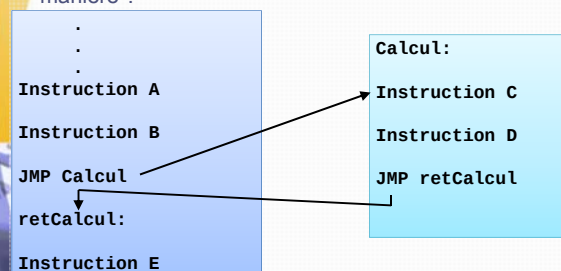
- ✓ Au moment de l'appel de la fonction, l'adresse de l'instruction suivante est sauvegardée dans la pile
 - CALL = sauvegarde de IP
- ✓ A la fin de la procédure, l'unité de traitement récupère les valeurs sauvegardées pour retourner au programme principal
 - RET = dépilement de IP
- ✓ Si la procédure est de type FAR, CS est aussi sauvegardé sur la pile lors du CALL, RET dépile IP puis CS

Procédures (5)



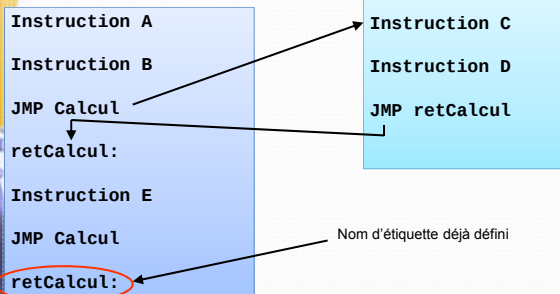
Exercice (1)

- ✓ Est-ce qu'on pourrait simuler une procédure de cette manière ?



Réponse (1)

- ✓ Est-ce qu'on pourrait simuler une procédure de cette manière ?



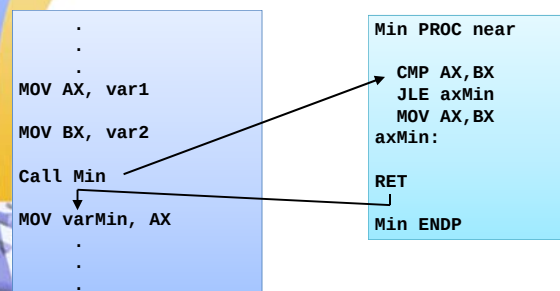
Passage de paramètres

- ✓ En général, une procédure
 - effectue un traitement sur des données (*paramètres*) qui sont fournies par le programme appelant, et
 - produit un résultat qui est transmis à ce programme.
- ✓ 2 méthodes de passage de paramètres :
 - par registre
 - par la pile

Passage par registre (1)

- ✓ Passage de paramètres par registre
 - les paramètres d'entrée de la procédure sont mis dans des registres avant l'appel de la procédure
 - les paramètres de sortie sont aussi rangés dans des registres
- ✓ Avantages
 - rapidité,
 - simplicité de gestion
- ✓ Inconvénient
 - peu de registres

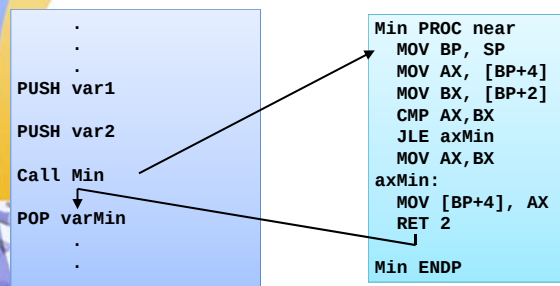
Passage par registre (2)

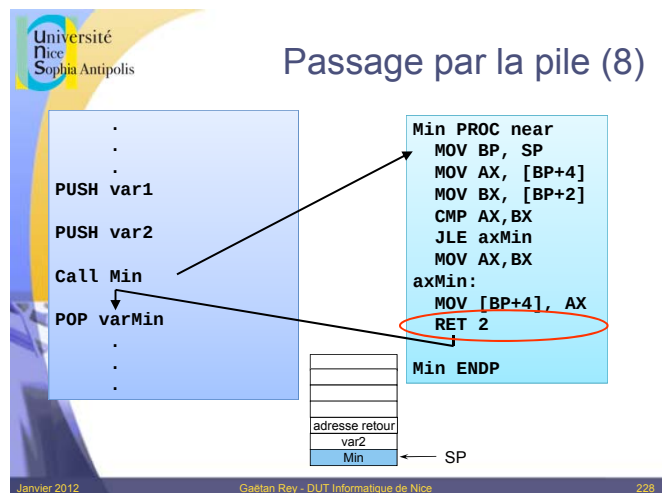
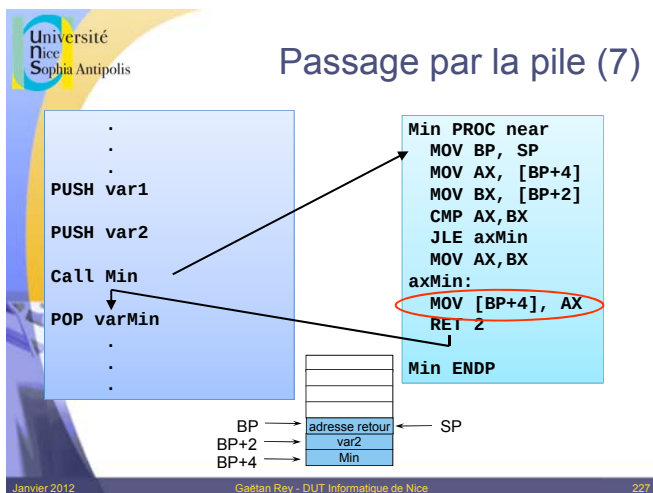
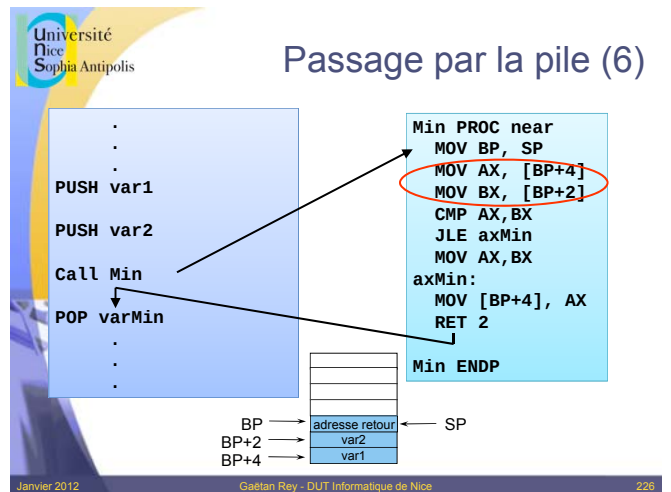
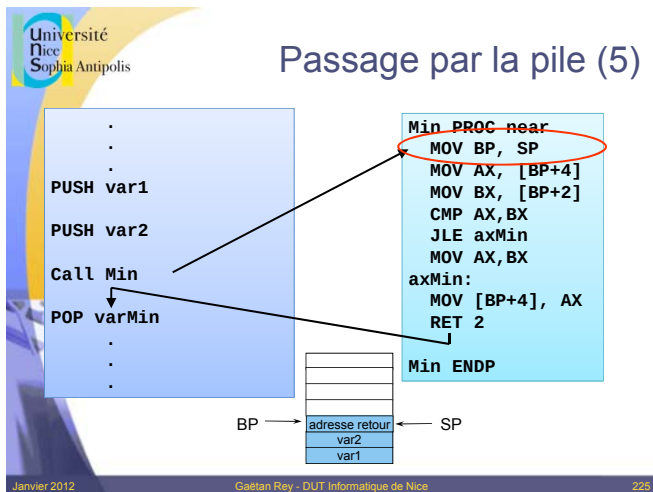
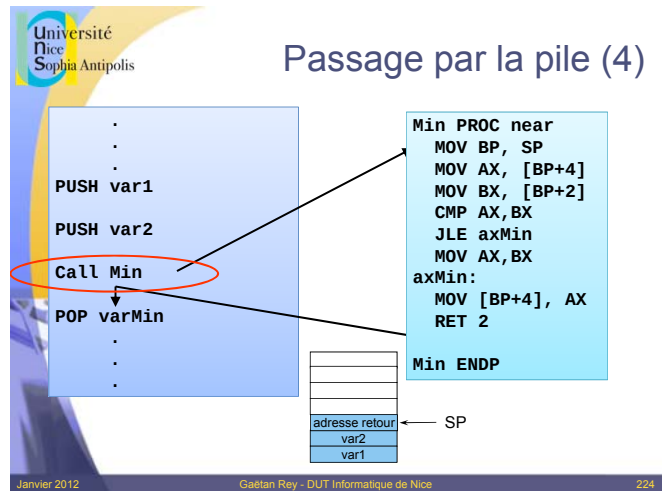
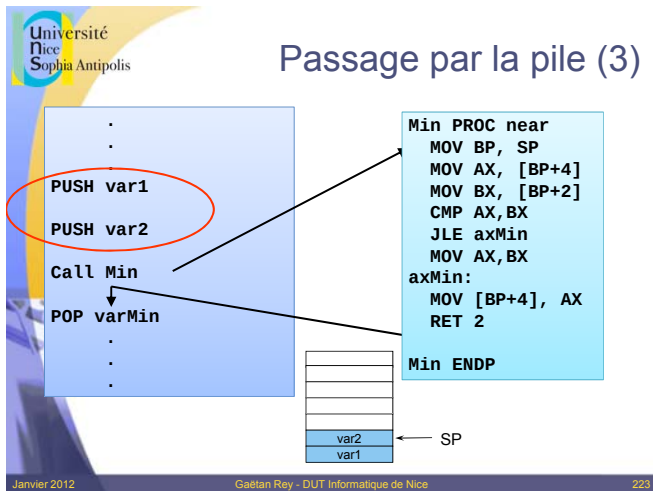


Passage par la pile (1)

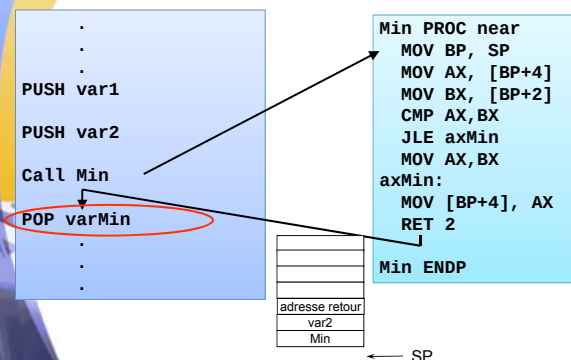
- ✓ Passage de paramètres par la pile
 - Les paramètres d'entrée sont empilés avant l'appel de la procédure
 - Les paramètres de sortie sont dépilés par le programme principal
- ✓ Avantage
 - pas de limite au nombre de paramètres
- ✓ Inconvénient
 - récupération des paramètres plus « lourde »

Passage par la pile (2)





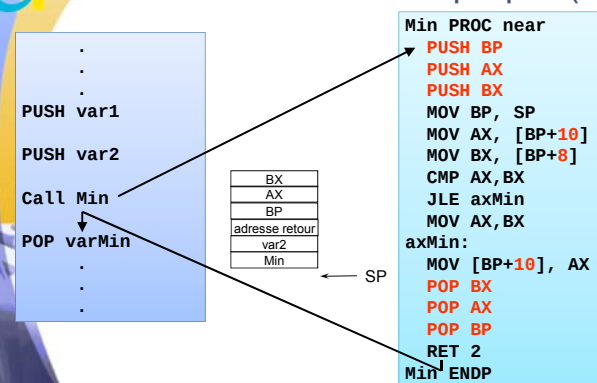
Passage par la pile (9)



Procédure propre (1)

- ✓ La procédure ne doit pas modifier les valeurs des registres du point de vue du programme appelant
 - Sauvegarder les registres au début la procédure.
 - Exécuté la procédure
 - Restaurer les valeurs des registres
- ✓ La sauvegarde des registres se fait en utilisant la pile.

Procédure propre (2)



Plan

- ✓ Introduction
- ✓ Représentation de l'information
- ✓ Arithmétique binaire
- ✓ Algèbre de Boole
- ✓ Circuits séquentiels/Automates
- ✓ Architecture type Von Neumann
- ✓ Couche d'assemblage / Assembleur 8086
- ✓ **Mécanismes d'interruptions**
- ✓ Fonctions avancées de processeurs modernes
- ✓ Gestion de la mémoire
- ✓ Traduction de programmes, compilation édition de liens, chargement

Système d'exploitation (1)

- ✓ Le système d'exploitation d'un ordinateur est le programme qui permet d'accéder aux ressources matérielles de cet ordinateur. Ces ressources matérielles sont essentiellement les organes d'entrées/sorties :
 - clavier, écran, liaisons réseau, imprimante, disque dur...
- ✓ Les périphériques d'entrées/sorties varient d'un modèle d'ordinateur à l'autre. De ce fait, les instructions à exécuter pour piloter tel périphérique diffèrent d'un ordinateur à l'autre.

Système d'exploitation (2)

- ✓ Le rôle principal du système d'exploitation est d'isoler les programmes des détails du matériel.
- ✓ C'est le système d'exploitation qui doit connaître les détails du matériel.
- ✓ Cette répartition des rôles simplifie grandement l'écriture des applications
- ✓ Le système d'exploitation est donc un programme complexe, lié à la configuration matérielle de la machine.

Le BIOS

- ✓ Le BIOS constitue la couche basse de tous les systèmes d'exploitations sur PC.
- ✓ Il est responsable de la gestion du matériel :
 - clavier, écran, disques durs, liaisons séries et parallèles.
- ✓ Le programme du BIOS se trouve en mémoire morte (ROM).
- ✓ Chaque modèle de PC est vendu avec une version du BIOS adapté à sa configuration matérielle.

Les fonctions du BIOS

- ✓ On peut considérer le BIOS comme une librairie de fonctions.
 - Chaque fonction effectue une tâche bien précise, comme par exemple afficher un caractère donné sur l'écran.
 - L'appel de l'une de ces fonctions constitue un *appel système*.
- ✓ Le mécanisme d'appel est différent des appels de fonctions (CALL).
 - Il a été prévu de pouvoir modifier le comportement du BIOS en cours d'utilisation, (gestion d'un nouveau périphérique).
 - Le code du BIOS étant en mémoire morte, il n'est pas modifiable.
 - De plus, le BIOS étant différent d'un ordinateur à l'autre, les adresses des fonctions changent...

Vecteurs d'interruptions (1)

- ✓ Le problème est résolu par l'utilisation d'une table d'indirection, la table des vecteurs d'interruptions.
- ✓ Cette table est placée en mémoire principale (RAM), et contient les adresses (en ROM ou en RAM) des fonctions du BIOS.
 - Elle est implantée à partir de l'adresse 0000 0000H (première case mémoire) et est initialisé par le BIOS lui même au moment du démarrage du PC (boot).
- ✓ Adresse contenu


```
0000 adresse de la première fonction du BIOS
0004 adresse de la deuxième fonction du BIOS
... ..
```
- ✓ Chaque élément de la table occupe 4 octets (adresse 32 bits). La table a 256 éléments (et occupe donc 1Ko).

Vecteurs d'interruptions (2)

- ✓ Par exemple, si l'on sait que la fonction du BIOS qui affiche un caractère est la 33ième, on va l'appeler en lisant la 33ième ligne de la table, puis en allant exécuter les instructions à l'adresse trouvée.
- ✓ Adresse contenu


```
... ..
0084H F1234560H (car 4x33 = 84H) .
... ..
```
- ✓ La table des vecteurs d'interruptions contient des valeurs différentes pour chaque version de BIOS, et peut être modifiée pour pointer sur du code en mémoire principale, modifiant alors le BIOS existant.

Appel système (1)

- ✓ L'instruction INT n permet d'appeler la n-ième fonction de la table des vecteurs d'interruptions.
- ✓ n est un entier compris entre 0 et 255 (1 octet), car il y a 256 vecteurs d'interruptions dans la table.
- ✓ L'instruction INT n est semblable à l'instruction CALL, sauf
 - que l'adresse de destination est donnée par la table des vecteurs d'interruptions, et
 - que les indicateurs sont automatiquement sauvegardés sur la pile.
- ✓ De plus, l'adresse de retour complète (32 bits) est empiilée, car le traitant d'interruption n'est pas dans le même segment de code que le programme appelant.

Appel système (2)

- ✓ Le déroulement de INT n se passe comme suit :
 1. sauvegarde les indicateurs du registre d'état sur la pile (les indicateurs sont regroupés dans un mot de 16 bits) ;
 2. sauvegarde CS et IP sur la pile ;
 3. CS et IP sont chargés avec la valeur lue à l'adresse 4n, n étant le paramètre de INT. L'exécution continue donc au début du traitant d'interruption.

Traitants d'interruptions

- ✓ Un traitant d'interruption est une fonction appelée via la table des vecteurs d'interruption par l'instruction INT n.
- ✓ Les traitants d'interruptions sont très similaires à des procédures ordinaires, sauf qu'ils doivent se terminer par l'instruction IRET au lieu de RET.
- ✓ L'instruction IRET est très similaire à RET, sauf
 - que CS et IP sont dépilés, et
 - que tous les indicateurs sont restaurés à leur anciennes valeurs, sauvegardées sur la pile par INT n.
- ✓ Notons que les éventuels paramètres du traitant d'interruption sont toujours passés par registre.

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

241

Le DOS

- ✓ Le système DOS (*Disk Operating System*, système d'exploitation de disque) repose sur le BIOS, dont il appelle les fonctions pour interagir avec le matériel.
- ✓ Les fonctions du DOS s'utilisent comme celles du BIOS, via des vecteurs d'interruptions. Elles offrent des fonctionnalités de plus haut niveau que le BIOS (entrées/sorties, ouverture de fichiers sur disque, etc.).
- ✓ Les fonctions du DOS s'appellent toutes à l'aide du vecteur 21H. La valeur du registre AH permet d'indiquer quelle est la fonction que l'on appelle :

```
MOV AH, numero_fonction
INT 21H
```

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

242

Notion d'interruption (1)

- ✓ Lorsqu'une machine fonctionne, sous le contrôle d'un système d'exploitation, l'unité centrale est en permanence susceptible d'exécuter un programme.
- ✓ Le problème se pose alors de savoir à quel moment l'unité centrale va pouvoir prendre en compte les événements extérieurs à la séquence d'instructions qu'elle exécute :
 - requête d'un périphérique,
 - frappe d'une touche sur un clavier,
 - insertion d'une disquette,
 - passage d'un objet devant un capteur, ...

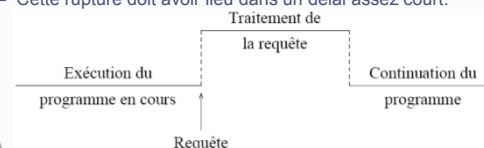
Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

243

Notion d'interruption (2)

- ✓ A chacun de ces événements correspond une tâche à exécuter par l'unité centrale.
 - Cette tâche est codée sous forme d'une procédure du système d'exploitation.
- ✓ Pour pouvoir exécuter cette procédure il faut que se produise une rupture de séquence.
 - Cette rupture doit avoir lieu dans un délai assez court.



Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

244

Notion d'interruption (3)

- ✓ Comment signaler au CPU un événement asynchrone ?
- ✓ Une première approche (dite des drapeaux) consiste à tester périodiquement en séquence chaque unité.
 - Cette technique n'est pas très économique en temps, elle demande une attention constante du CPU et la liste exhaustive des intervenants à tester.
- ✓ Dans une autre approche les machines peuvent être dotées d'un système d'interruptions.
 - Cette technique consiste à pouvoir interrompre "brutalement" l'exécution du programme en cours à la fin de l'instruction courante.
 - Après sauvegarde de son état présent, le contrôle de l'unité centrale est confié à un sous-programme du système dépendant de la nature de l'interruption.

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

245

Notion d'interruption (4)

- ✓ Une interruption est signalée au processeur par un signal électrique sur une borne spéciale.
 - Lors de la réception de ce signal, le processeur "traite" l'interruption dès la fin de l'instruction qu'il était en train d'exécuter.
- ✓ Le traitement de l'interruption consiste soit :
 - à l'ignorer et passer normalement à l'instruction suivante.
 - à exécuter un *traitant d'interruption* (interrupt handler).

Janvier 2012

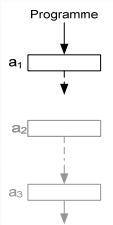
Gaëtan Rey - DUT Informatique de Nice

246

Université Nice Sophia Antipolis

Principes des interruptions (1)

- ✓ Le processeur exécute les instructions machine d'un programme
 - Il vient de terminer l'exécution de l'instruction a1
 - Il va normalement passer à l'instruction a2

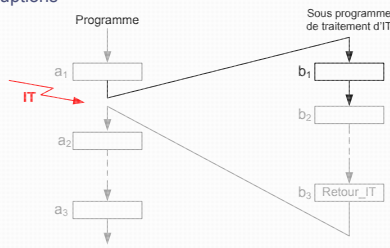


Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 247

Université Nice Sophia Antipolis

Principes des interruptions (2)

- ✓ Le processeur reçoit une interruption
 - Il empile l'adresse de l'instruction qu'il devait exécuter
 - Il se déroute à la 1ère instruction de la fonction du traitement d'interruptions

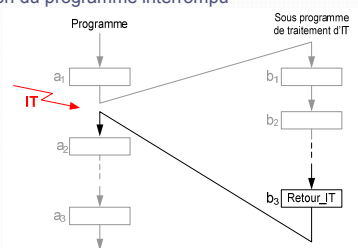


Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 248

Université Nice Sophia Antipolis

Principes des interruptions (3)

- ✓ A la fin de la fonction du traitement d'interruptions
 - IRET instruction spécifique
 - Dépile l'adresse précédemment sauvegarder, et reprend l'exécution du programme interrompu

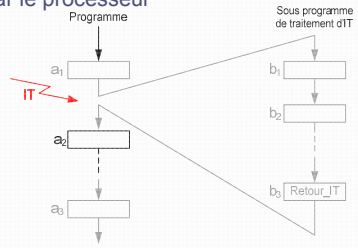


Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 249

Université Nice Sophia Antipolis

Principes des interruptions (4)

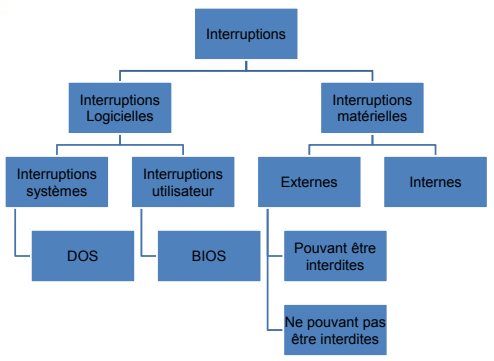
- ✓ Le programme a repris son cours d'exécution normal. L'exécution du sous programme de traitement d'interruptions a été « invisible » pour le programme exécuté par le processeur



Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 250

Université Nice Sophia Antipolis

Les différents types d'interruptions



Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 251

Université Nice Sophia Antipolis

Caractéristiques des Interruptions

- ✓ Priorités des interruptions
 - Classement par ordre de priorités.
Priorité IT interne > Priorité IT matérielles > Priorité IT logicielles
- ✓ Visibilité des interruptions
 - Interruptions masquables
 - Interruptions non masquables
 - traitées immédiatement
 - possèdent généralement une ligne de requête dédiée et un numéro de vecteur fixe.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 252

Université
Nice
Sophia Antipolis

Reconnaissance des interruptions

- ✓ Il existe différents moyens physique pour déterminer la source d'une interruption
 - Interruptions multi-niveaux
 - Interruption en ligne unique
 - Interruption vectorisée

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 253

Université
Nice
Sophia Antipolis

Interruptions multi-niveaux

- ✓ Chaque équipement est relié à une entrée d'IT particulière sur le microprocesseur.

- ✓ **Avantage**
 - solution techniquement simple
- ✓ **Inconvénients**
 - coûteuse en broches d'entrée du processeur,
 - pas très portable

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 254

Université
Nice
Sophia Antipolis

Interruption en ligne unique (1)

- ✓ Suppose que chaque demandeur dispose d'un bit d'indication de demande que le CPU peut aller lire.
 - Lorsqu'une interruption est acceptée, le CPU va exécuter un programme débutant à une adresse fixe et qui a pour but de tester un à un les bits d'indication de demande puis de sauter au programme correspondant à la première demande active rencontrée.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 255

Université
Nice
Sophia Antipolis

Interruption en ligne unique (2)

- ✓ La scrutation est lente lorsque le nombre de demandeurs est grand.
- ✓ Par contre, le problème de priorité est directement résolu, puisque l'ordre dans lequel les demandeurs sont testés nous donne l'ordre de priorité décroissant.

- ✓ **Avantage**
 - une seule ligne d'IT sur le processeur
- ✓ **Inconvénient**
 - scrutation des périphériques pour déterminer le générateur de l'interruption

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 256

Université
Nice
Sophia Antipolis

Interruption vectorisée (1)

- ✓ Est une méthode dans laquelle le CPU envoie un signal d'acceptation d'interruption *INTA_L (Interrupt Acknowledge)*, à la suite de quoi un demandeur s'identifie en envoyant son vecteur spécifique.
- ✓ Ce vecteur, envoyé à travers le bus de données, peut être
 - une instruction (généralement un appel à un sous-programme),
 - une adresse,
 - une partie d'une adresse seulement.
- ✓ En utilisant la vectorisation, le temps de réponse à une interruption ne dépend pas du nombre de demandeurs potentiels et reste toujours inférieur à celui obtenu par scrutation.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 257

Université
Nice
Sophia Antipolis

Interruption vectorisée (2)

- ✓ Par contre, non seulement il faudra du matériel pour générer le vecteur, mais encore faudra-t-il résoudre le problème des priorités.
- ✓ **Avantage**
 - le microprocesseur reconnaît immédiatement le périphérique qui a déclenché l'interruption
- ✓ **Inconvénient**
 - il est nécessaire de gérer des priorités (dépôts simultanés de 2 vecteurs sur le bus)

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 258

Université
Nice
Sophia Antipolis

Table des vecteurs d'interruptions (1)

- ✓ Le 8086 supporte 256 interruptions différentes, chacune étant identifiée par un numéro compris entre 0 et FFh.
- ✓ L'adresse de début d'une routine d'interruption est appelée Vecteur d'interruption ou traitant d'interruption.
- ✓ Toutes les adresses (logique CS:IP) de début de ces vecteurs sont rangées dans une table et constituent donc la Table des Vecteurs d'Interruption.
 - Cette table se charge en RAM à partir de l'adresse la plus basse (0000h).
 - Chaque vecteur d'interruption est stocké sur 4 octets (2 pour CS et 2 pour IP)

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 259

Université
Nice
Sophia Antipolis

Table des vecteurs d'interruptions (2)

Pour « installer » une routine de traitement pour l'interruption n°i, il suffit de modifier le contenu de cette table

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 260

Université
Nice
Sophia Antipolis

Fonctionnement sur le 8086

- ✓ état du 8086 avant réception de l'IT n°i

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 261

Université
Nice
Sophia Antipolis

Prise en compte de l'IT numéro i

On sauve le registre d'état du 8086, le contenu du registre de segment CS et la valeur du PC dans la pile.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 262

Université
Nice
Sophia Antipolis

Prise en compte de l'IT numéro i

On récupère dans la table d'IT de nouvelles valeurs pour PC et CS, qui permettent d'accéder au SP_IT n°i.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 263

Université
Nice
Sophia Antipolis

Interruptions matérielles sur PC

- ✓ Les processeurs de la famille 80x86 possèdent trois bornes pour gérer les interruptions
 - NMI (Non Maskable Interrupt) sert à envoyer une interruption non masquable.
 - Le processeur ne peut pas ignorer ce signal et va exécuter le traitant donné par le vecteur 02H.
 - Ce signal est normalement utilisé pour détecter les erreurs matérielles (par exemple une erreur mémoire).
 - INTR (Interrupt Request) demande une interruption masquable.
 - Utilisée pour indiquer au CPU l'arrivée d'une interruption
 - INTA (Interrupt Acknowledge) est mis à 0 lorsque le processeur traite l'interruption signalée par INTR
 - C'est-à-dire qu'elle n'est plus masquée

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 264

Indicateur IF

- ✓ IF : (Interrupt Flag) est un indicateur spécial du registre d'état
- ✓ A un instant donné les interruptions sont soit masquées, soit autorisées suivant l'état de IF
 - Si IF = 1, le processeur accepte les demandes d'interruptions masquables : il les traite immédiatement
→ Interruptions démasquées
 - Si IF = 0, le processeur ignore les interruptions
→ Interruptions masquées
- ✓ L'indicateur IF peut être modifié à l'aide de ces deux instructions :
 - STI (SeT IF) : $IF \leftarrow 1$
 - CLI (CLear IF) : $IF \leftarrow 0$

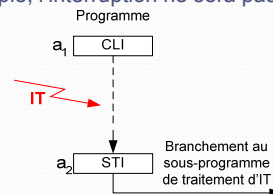
Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

265

Masquage/démasquage d'IT

- ✓ Si une interruption survient pendant que les IT sont masquées, celle-ci n'est pas perdue, elle sera prise en compte dès le démasquage des IT.
- ✓ Sur l'exemple, l'interruption ne sera pas prise en compte.



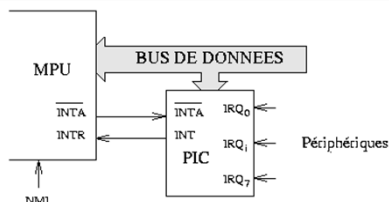
Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

266

Contrôleur d'interruptions

- ✓ L'ordinateur est relié à plusieurs périphériques, et pourtant un seul signal de demande d'interruptions (INTR)
- ✓ Le contrôleur d'interruptions
 - PIC : Programmable Interrupt Controller



Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

267

Communication avec le contrôleur

- ✓ Le processeur communique avec le contrôleur
 - Il donne des ordres au contrôleur (lancer moteur ...)
 - Il échange les données (UC -> contrôleur, contrôleur -> UC)
 - Il doit récupérer des informations sur le déroulement des opérations (erreur, état du contrôleur, etc.)
- ✓ Le processeur est doté d'instructions permettant
 - D'envoyer une information vers le contrôleur
 - Instruction OUT sur le 8086 : écrit le contenu de AL à l'adresse spécifiée de l'espace d'entrées/sorties
 - De lire une information venant du contrôleur
 - Instruction IN sur le 8086 : lit l'octet d'adresse spécifiée dans l'espace d'entrées/sorties et le transfère dans le registre AL.

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

268

Déroulement d'une interruption masquable (1)

1. Un signal INT est émis par un périphérique (ou plutôt par l'interface gérant celui-ci).
2. Le contrôleur d'interruptions reçoit ce signal sur une de ses bornes IRQi. Dès que cela est possible (suivant les autres interruptions en attente de traitement), le contrôleur envoie un signal sur sa borne INT.
3. Le MPU prend en compte le signal sur sa borne INTR après avoir achevé l'exécution de l'instruction en cours. Si l'indicateur IF=0, le signal est ignoré, sinon, la demande d'interruption est acceptée.

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

269

Déroulement d'une interruption masquable (2)

4. Si la demande est acceptée, le MPU met sa sortie $\overline{INTA}$ au niveau 0 pendant 2 cycles d'horloge, pour indiquer au contrôleur qu'il prend en compte sa demande.
5. En réponse, le contrôleur d'interruption place le numéro de l'interruption associé à la borne IRQi sur le bus de données.

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

270

Université Nice Sophia Antipolis

Déroulement d'une interruption masquable (3)

6. Le processeur lit le numéro de l'interruption sur le bus de données et l'utilise pour trouver le vecteur d'interruption. Ensuite, tout se passe comme pour un appel système, le processeur :

- sauvegarde les indicateurs du registre d'état sur la pile;
- met l'indicateur IF à 0 (masque les interruptions suivantes);
- sauvegarde CS et IP sur la pile;
- cherche dans la table des vecteurs d'interruptions l'adresse du traitant d'interruption, qu'il charge dans CS:IP.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 271

Université Nice Sophia Antipolis

Déroulement d'une interruption masquable (4)

7. La procédure traitant l'interruption se déroule. Pendant ce temps, les interruptions sont masquées (IF=0).

- Si le traitement est long, on peut dans certains cas ré-autoriser les interruptions avec l'instruction STI.

8. La procédure se termine par l'instruction IRET, qui restaure CS, IP et les indicateurs à partir de la pile, ce qui permet de reprendre le programme qui avait été interrompu.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 272

Université Nice Sophia Antipolis

Entrées/sorties par interruption

- ✓ Les périphériques qui reçoivent des données de l'extérieur mettent en oeuvre un mécanisme d'interruption : clavier, liaisons séries (modem, souris...) et parallèles (imprimantes), interfaces réseau, contrôleurs de disques durs et CD-ROMS, etc.
- ✓ Nous allons voir 2 méthodes pour gérer ces E/S :
 - Attente active
 - E/S par Interruption

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 273

Université Nice Sophia Antipolis

Informations sur le contrôleur

- ✓ r.entrée : stocke les caractères reçus de l'extérieur
- ✓ r.sortie : stocke les caractères à envoyer vers l'extérieur
- ✓ r.etat : contient des informations sur l'état du coupleur
 - État prêt_à_emettre signifie que r.sortie est vide
 - État car_reçu signifiant que r.entrée est plein

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 274

Université Nice Sophia Antipolis

E/S par attente active: polling (1)

- ✓ Mise en oeuvre de la sortie d'un caractère :
 - Problème : Il ne faut pas lancer de nouvelle sortie (c.a.d écrire dans r.sortie) si la sortie précédente n'est pas terminée.
 - Solution : On va lire l'état du registre r.etat et ne lancer une sortie que si r.etat = prêt_à_emettre

```

procédure ecrire(car c)
var etatcoupleur état;
début
  lire r.etat;
  tantque r.etat ≠ prêt_à_emettre faire
    lire r.etat;
  refaire;
  mettre c dans r.sortie;
fin;

```

Attente active (polling)

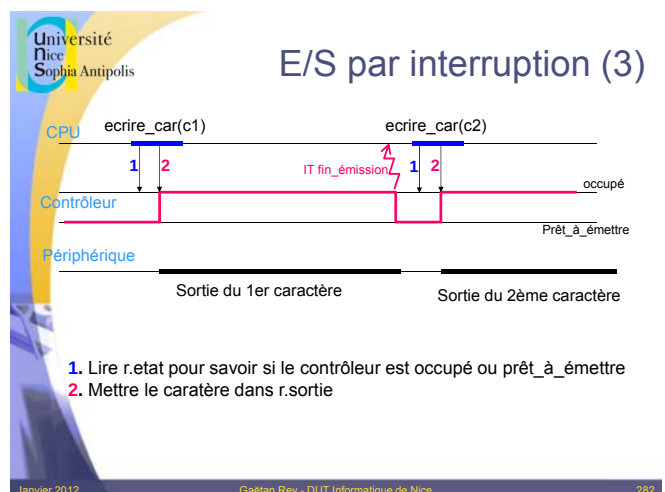
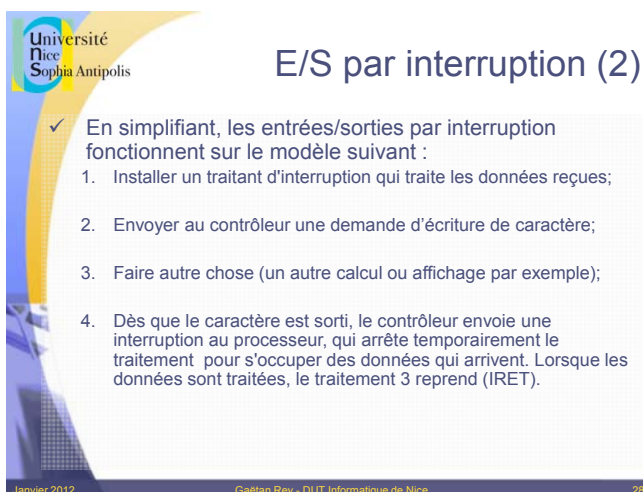
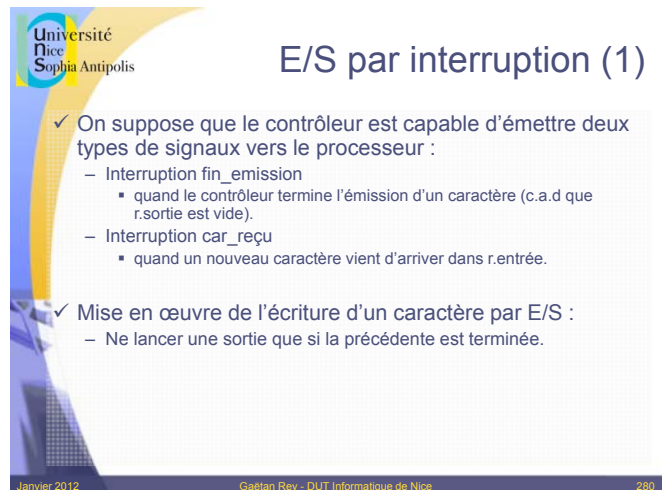
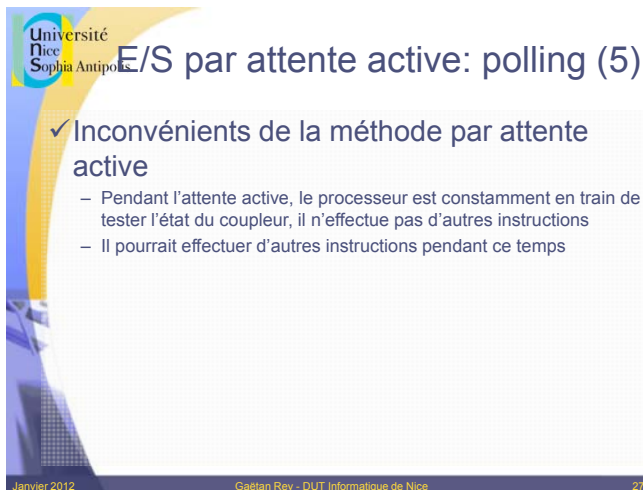
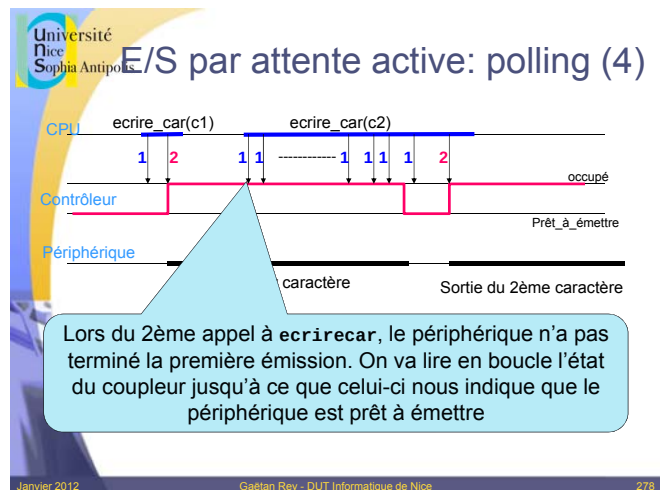
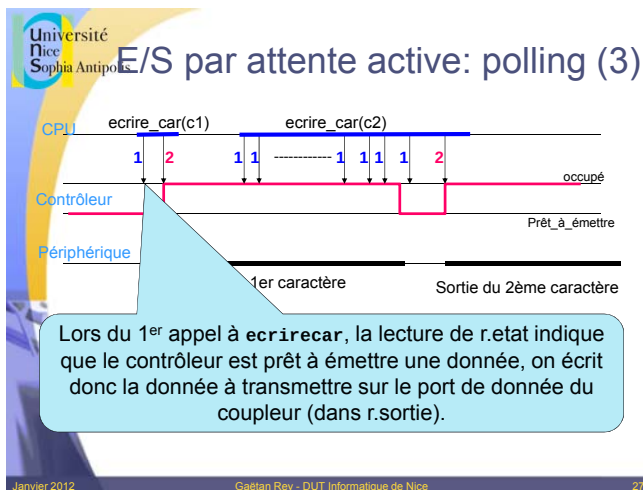
Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 275

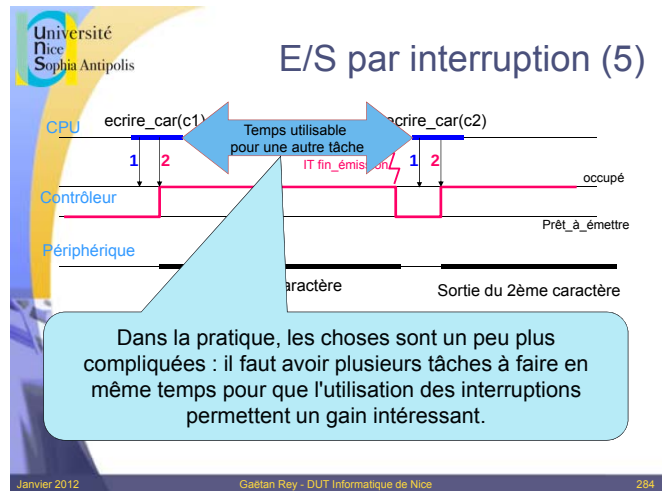
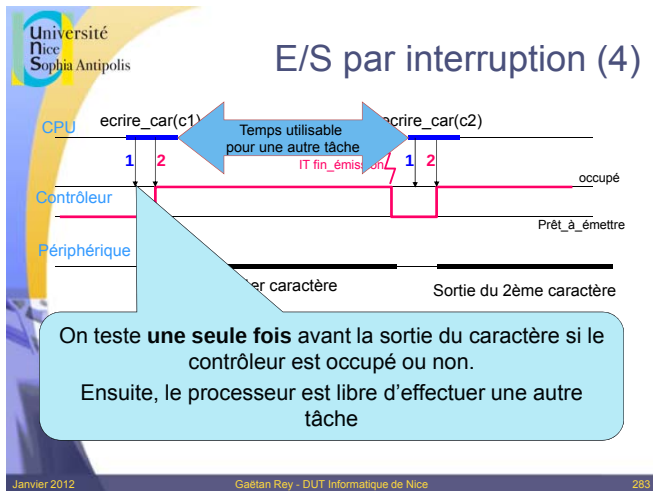
Université Nice Sophia Antipolis

E/S par attente active: polling (2)

1. Lire r.etat pour savoir si le contrôleur est occupé ou prêt_à_emettre
2. Mettre le caractère dans r.sortie

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 276



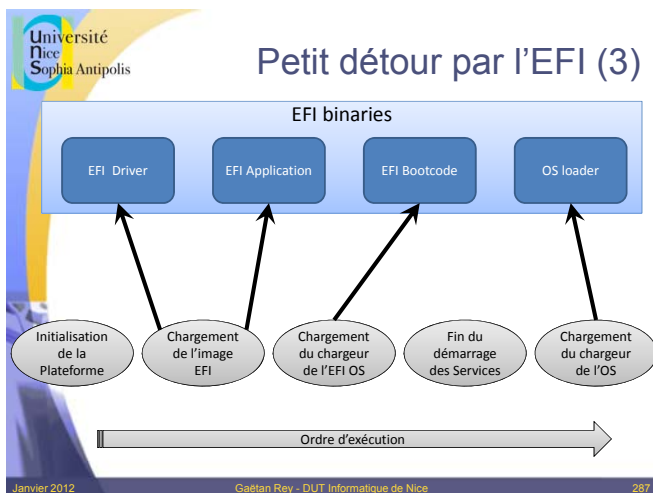
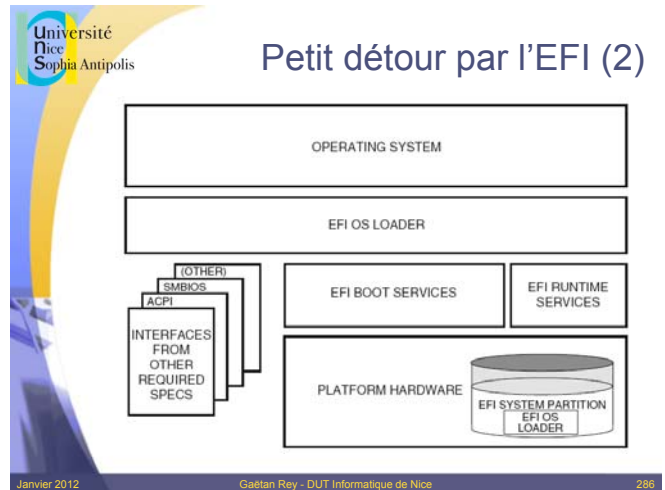


Université Nice Sophia Antipolis

Petit détour par l'EFI (1)

- ✓ UEFI (Unified Extensible Firmware Interface) est une spécification décrivant une interface qui permet de transférer le contrôle du système de pré-démarrage de l'environnement à un système d'exploitation.
- ✓ UEFI fournit une interface claire entre le système d'exploitation et la hardware au démarrage.
- ✓ La spécification UEFI est basée sur la spécification EFI 1.10 publié par Intel avec des corrections et des modifications géré par l'Unified EFI Forum.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 285



Université Nice Sophia Antipolis

Plan

- ✓ Introduction
- ✓ Représentation de l'information
- ✓ Arithmétique binaire
- ✓ Algèbre de Boole
- ✓ Circuits séquentiels/Automates
- ✓ Architecture type Von Neumann
- ✓ Couche d'assemblage / Assembleur 8086
- ✓ Mécanismes d'interruptions
- ✓ **Fonctions avancées de processeurs modernes**
- ✓ Gestion de la mémoire
- ✓ Traduction de programmes, compilation édition de liens, chargement

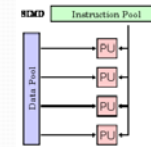
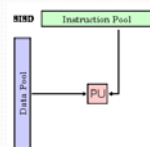
Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 288

Single Instruction, Multiple Data

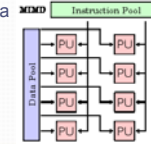
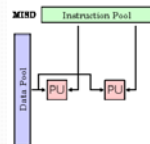
- ✓ Les applications multimédias et de communication devenant de plus en plus gourmandes en calcul.
- ✓ En 1996, Intel développe la technologie MMX, (AMD crée 3Dnow).
- ✓ Exemple, changer la couleur d'une image 1024x768 pixels en 65536 couleurs (chaque pixel occupe donc 16 bit)
 - il faut demander au processeur d'exécuter 1024*768 fois la même instruction.

SISD, SIMD, MISD et MIMD

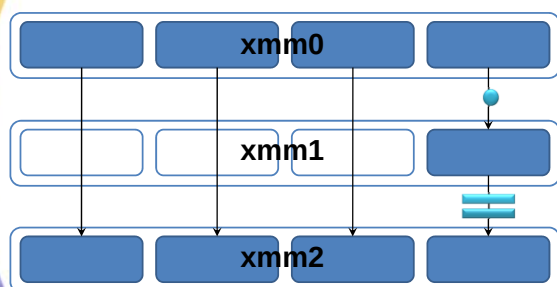
- ✓ Single Instruction, Single Data
- ✓ Single Instruction, Multiple Data



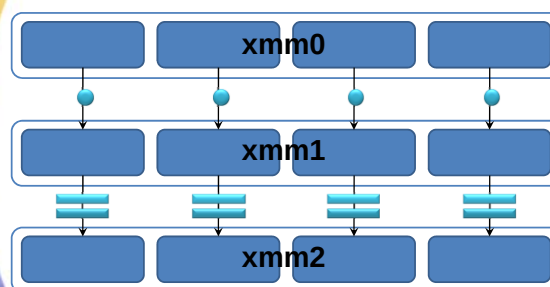
- ✓ Multiple Instruction, Single Data
- ✓ Multiple Instruction, Multiple Data



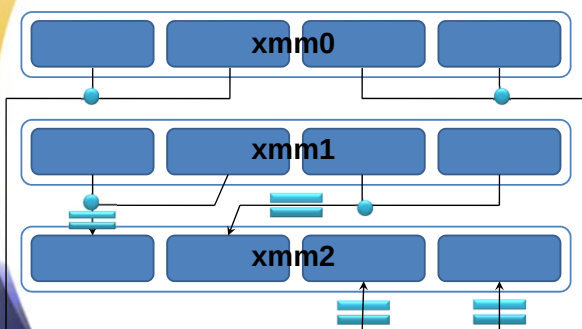
Opération scalaire



Opération vectorielle



Opération vectorielle horizontale



MultiMédia eXtended (1)

- ✓ Les processeurs MMX possèdent 8 registres supplémentaires de 64 bits, nommés MMX0 à MMX7
 - utilisables comme des registres d'usage général,
 - ne peuvent pas être utilisés pour effectuer de l'adressage indexé.

Registre Virgule Flottante 80 bits (*Floating Point Register*)

Registre MMX 64 bits

79	64	63	MM7	0
79	64	63	MM6	0
79	64	63	MM5	0
79	64	63	MM4	0
79	64	63	MM3	0
79	64	63	MM2	0
79	64	63	MM1	0
79	64	63	MM0	0

MultiMédia eXtended (2)

- ✓ Les 57 instructions MMX sont de type SIMD.
 - Elles peuvent agir ainsi sur 8 octets, ou 4 mots, ou 2 double-mots ou 1 quadruple mot de 64 bits.
 - Elles permettent de faire toute sortes d'opérations telles les opérations arithmétiques, logiques, des comparaisons, des conversions, des décalages et le chargement des registres MMX
- ✓ Attention, les instructions MMX ne travaillent que sur des entiers !!!



3DNow!

- ✓ Développé par AMD en 1997 avec le K6-2
- ✓ Elle regroupe 21 instructions et 8 registres 64 Bits FP / MMX
- ✓ 2 opérations entières ou flottantes par cycle d'horloge
- ✓ 2 instructions 3DNow! par cycle d'horloge
- ✓ L'équivalent d'Intel au 3Dnow! est le SSE qui est similaire mais incompatible

Streaming SIMD Extensions

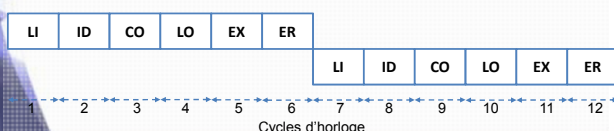
- ✓ L'architecture SSE
 - complète le jeu d'instructions MMX avec des opérations de manipulation des entiers, sur les registres MMX existants.
 - ajoute les instructions qui manquaient à la technologie MMX : la manipulation des nombres flottants
 - ajoute des instructions de contrôle du comportement des caches L1/L2, et des instructions de manipulation d'un nouveau registre d'état.
- ✓ L'architecture SSE ajoute 8 registres 128 bits (XMM0 à XMM7), à accès aléatoire,
 - contenant des paquets de quatre double-mots (flottant simple précision).

SSE – les versions suivantes

- ✓ SSE2 = SSE1 + 144 instructions
 - Traitement des flottants en double précision
 - Nouvelles instructions de traitement sur les entiers
 - Gestion du cache et de contrôle de lecture et d'écriture en mémoire
- ✓ SSE3 = SSE2 + 13 instructions
 - Traitement horizontal des registres.
 - Optimisation des applications multi-threadées (moniteur).
- ✓ SSSE3 = SSE3 + 16 instructions
- ✓ SSE4 = SSSE3 + 50 instructions
 - Traitements autres que multimédia (CRC32, comparaison de chaînes, ...)

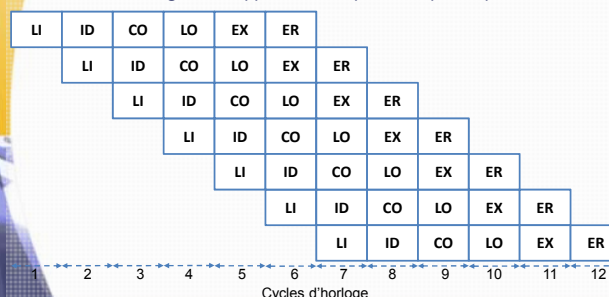
Pipeline (1)

- ✓ Dans un processeur l'exécution d'une instruction est découpé en une succession d'étapes
 - Lecture de l'instruction (LI)
 - Décodage de l'instruction (DI)
 - Calcul des opérandes (CO)
 - Lecture des opérandes (LO)
 - Exécution de l'instruction (EX)
 - Ecriture du résultat (ER)



Pipeline (2)

- ✓ Si on utilise un pipeline
 - Besoin de registres supplémentaire pour chaque étape



Université Nice Sophia Antipolis

Architecture Superscalaire

✓ Elle contient plusieurs pipelines en parallèle

Cycles d'horloge

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 301

Université Nice Sophia Antipolis

Performance du pipeline (1)

Cycles d'horloge

✓ $T = \text{temps de cycle du pipeline}$

$$\tau = \max_i [\tau_i] = \tau_m + d$$

✓ $T_m = \text{délai maximum d'un étage}$

✓ $d = \text{délai entre les étages du pipeline } (T_m \gg d)$

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 302

Université Nice Sophia Antipolis

Performance du pipeline (2)

Cycles d'horloge

✓ Quel est le temps d'exécution de n instructions en fonction de k (le nombre d'étages), de T (durée d'un cycle) dans le cas où :

- il n'y a pas de pipeline ?
- il y a un pipeline et
 - Les instructions sont indépendantes et il n'y a pas de branchement ?
 - Les instructions sont indépendantes et il y a des branchements ?

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 303

Université Nice Sophia Antipolis

Performance du pipeline (3)

✓ Cas sans pipeline pour n instructions

- $T_{sp} = nkT$

Cycles d'horloge

✓ Soit k , le nombre d'étages du pipeline

- Le temps de la première instruction est $T_{i1} = k \cdot T$
- Le temps de chacune des instructions suivantes est $T_{ij} = T$

✓ Le temps de n instructions sans branchement est

- $T_{i1 \rightarrow in} = (k + (n-1)) T$

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 304

Université Nice Sophia Antipolis

Performance du pipeline (4)

✓ Facteur d'accélération du pipeline

- $S_n = T_{sp} / T_{i1 \rightarrow in} = nkT / (k + (n-1)) T = nk / k + (n-1)$

Facteur d'accélération

— $k = 6$ étages
— $k = 9$ étages
— $k = 12$ étages

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 305

Université Nice Sophia Antipolis

Pipeline & branchement

Instruction de branchement et saut effectué

Pénalité de branchement (P)

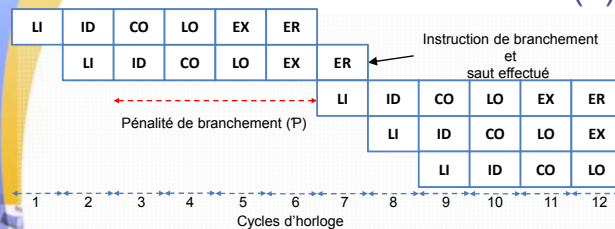
Cycles d'horloge

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 306

Performance du pipeline avec branchement (1)

- ✓ Le temps de n instructions **avec** branchement est
 - $T_{i1 \rightarrow in} = T_{i1 \rightarrow in} + P_{i1 \rightarrow in}$
- ✓ Le temps de n instructions **sans** branchement est
 - $T_{i1 \rightarrow in} = (k + (n-1)) T$
- ✓ Le temps de n instructions sans branchement est
 - $P_{i1 \rightarrow in} = (P * Nb) T$
 - P = la pénalité d'un branchement
 - Nb = nombre de branchements
- ✓ Le temps de n instructions **avec** branchement est
 - $T_{i1 \rightarrow in} = (k + (n-1)) T + (P * Nb) T = (k + (n-1) + P * Nb) T$

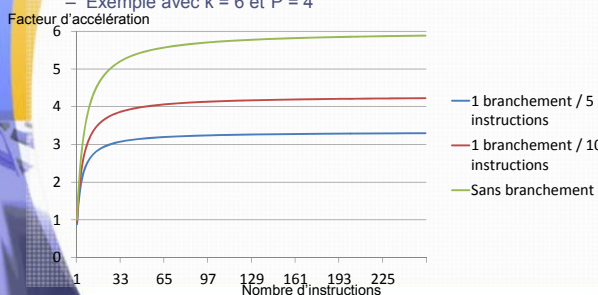
Performance du pipeline avec branchement (2)



- ✓ $k = 6$; $P = 4$; $n = 5$ et il y a 1 branchement toutes les 5 instructions
- ✓ $T_{i1 \rightarrow i_n}^b = (k + (n-1) + P * Nb) T = (6 + (5-1) + 4 (5*1/5)) T$
- ✓ $T_{i1 \rightarrow i_n}^b = 14 T$

Performance du pipeline avec branchement (3)

- ✓ Facteur d'accélération du pipeline
 - $S_n = T_{sp} / T_{i1 \rightarrow in} = nk / k + (n-1) + P * Nb$
 - Exemple avec $k = 6$ et $P = 4$



Traitements des branchements

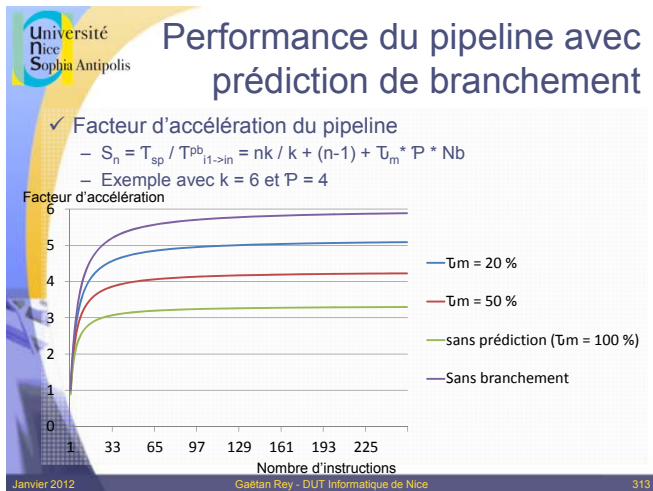
- ✓ Comment diminuer la pénalité d'un branchement conditionnel ?
- ✓ Plusieurs flots
 - Dupliquer le pipeline (au moins le début)
 - Problèmes :
 - Augmente les délais registres et mémoires
 - Possiblement d'autres branchements
- ✓ Destination de branchement pré chargée
 - Si détection d'un branchement conditionnel, pré chargement de l'instruction suivante et de la destination

Tampon de boucle

- ✓ Mémoire à très haute vitesse gérée par l'étage LI
- ✓ Contient les n dernières instructions en séquence
 - Si le branchement pointe sur ces instructions (cas de la boucle) => on lit directement les instructions depuis le tampon
- ✓ Avantage
 - Diminution du temps d'accès des instructions des boucles
- ✓ Problème
 - Couteux
 - Doit avoir une taille suffisante pour contenir toutes les instructions de la boucle

Prédiction de branchements

- ✓ Repérer les instructions de branchement conditionnel et à prédire si le branchement est pris ou non.
- ✓ Différentes techniques de branchements :
 - Prédit jamais pris,
 - Prédit toujours pris,
 - Prédit selon code op,
 - Commutateur pris/non pris,
 - Table d'historique des branchements.
- ✓ $T_{i1->iN}^{pb} = (k + (n-1)) T + (\bar{T}_m * P * Nb) T$
- ✓ $T_{i1->iN}^{pb} = (k + (n-1)) T + \bar{T}_m * P * Nb) T$
 - Avec $\bar{T}_m$ le taux de mauvaises prédictions



Université Nice Sophia Antipolis

Prédiction de branchements Prédit jamais/toujours pris

- ✓ Approche statique
- ✓ Elle fait l'hypothèse que le branchement ne sera pas pris ou toujours pris.
 - Exemple le 68020 : jamais pris
- ✓ Analyse du code
 - En moyenne 50% des branchements sont de type conditionnel
- ✓ Prédit toujours pris > Prédit jamais prit
 - Vrai si le coût de pré-chargement est identique

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 314

Université Nice Sophia Antipolis

Prédiction de branchements Prédit selon code op

- ✓ Approche statique
- ✓ Branchement pris en fonction du code opération du branchement
 - Toujours pris si branchement inconditionnel
 - Ça dépend pour les branchements conditionnels
- ✓ Taux de réussite > 75%

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 315

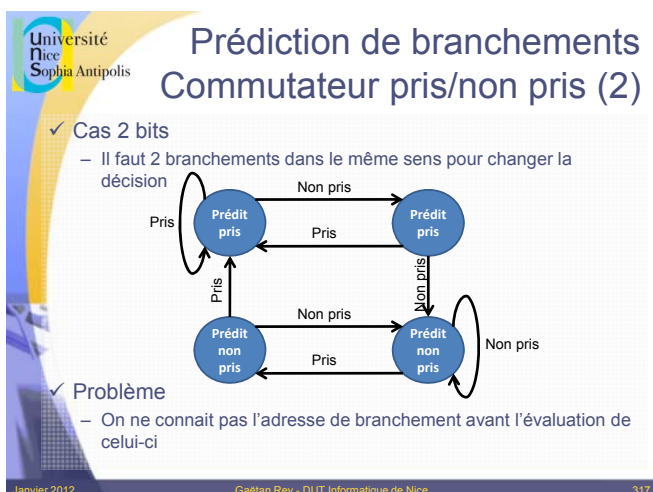
Université Nice Sophia Antipolis

Prédiction de branchements Commutateur pris/non pris (1)

- ✓ Approche dynamique
- ✓ Objectif
 - Améliorer la prédiction en gardant l'historique des branchements
- ✓ Utilisation d'une mémoire à très haute vitesse
 - Couple instruction (adresse) / compteur (quelques bits)
- ✓ Cas 1 bit
 - On fait comme la fois précédente
 - Si erreur on modifie la table

Adresse de l'instruction	Commutateur 1 bit
0AB5 1245h	1 ← Pris
0BE0 8574h	0 ← Non pris
0EB6 4512h	0 ← Non pris

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 316



Université Nice Sophia Antipolis

Prédiction de branchements Table d'historique des branchements

- ✓ Approche dynamique
- ✓ Petite mémoire cache dans l'étage LI
- ✓ 3 éléments par entrée
 - Adresse de l'instruction branchement
 - Bits d'historique
 - Instruction destination ou adresse de l'instruction destination

Adresse de l'instruction	Commutateur 2 bits	Adresse destination
0AB5 1245h	10	0AB5 1345h
0BE0 8574h	01	0BE0 6774h
0EB6 4512h	00	0EB6 0012h

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 318

Plan

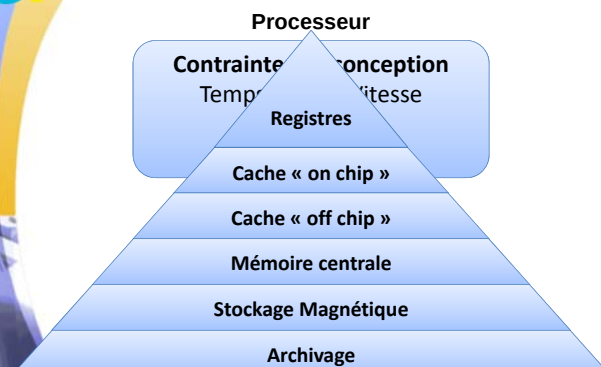
- ✓ Introduction
- ✓ Représentation de l'information
- ✓ Arithmétique binaire
- ✓ Algèbre de Boole
- ✓ Circuits séquentiels/Automates
- ✓ Architecture type Von Neumann
- ✓ Couche d'assemblage / Assembleur 8086
- ✓ Mécanismes d'interruptions
- ✓ Fonctions avancées de processeurs modernes
- ✓ **Gestion de la mémoire**
- ✓ Traduction de programmes, compilation édition de liens, chargement

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

319

Mémoire



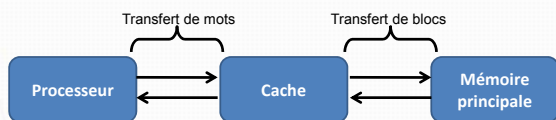
Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

320

Cache

- ✓ Objectifs
 - Grande vitesse
 - Taille importante
 - Prix pas trop élevé

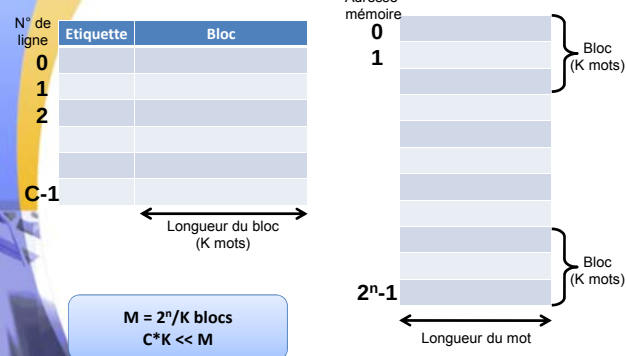


Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

321

Organisation d'un cache



Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

322

Fonction de correspondance

- ✓ Nombre de Lignes du cache < Nombre de Blocs
 - Comment faire la correspondance entre les lignes et les blocs ?
- ✓ 3 techniques de correspondances
 - Directe
 - Associative
 - Associative par ensemble
- ✓ La technique utilisée détermine l'organisation du cache
- ✓ Exemple
 - Cache de 64 Ko avec un bloc de 4 octets
 - Mémoire 16 Mo (bus d'adresse 24 bits),

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

323

Correspondance directe (1)

- ✓ Chaque bloc mémoire est associé à une seule ligne
- ✓ Simple et peu coûteux
- ✓ Problème de concurrence entre 2 blocs sur une même ligne
- ✓ Fonction de correspondance : $i = j \text{ modulo } m$
 - $i = n^\circ$ de la ligne de cache
 - $j = n^\circ$ du bloc
 - $m = \text{nb de ligne de cache}$



Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

324

Université Nice Sophia Antipolis

Correspondance directe (2)

- ✓ Récapitulatif
 - Longueur de l'adresse = $(s + w)$ bits = 24 bits
 - Nombre d'unités adressables = 2^{s+w} octets = 2^{24} octets
 - Taille du bloc = 2^w octets = 2^2 octets = 4 octets
 - Taille de la ligne = 2^w octets = 2^2 octets = 4 octets
 - Nombre de blocs dans la mémoire = $2^{s+w} / 2^w = 2^s$ octets = 2^{22} o
 - Nombre de lignes dans le cache = $m = 2^r = 2^{14}$ lignes
 - Taille de l'étiquette = $(s - r)$ bits = 8 bits
- ✓ Exemple
 - Cache de 64 Ko avec un bloc de 4 octets (2^{14} lignes)
 - Mémoire 16 Mo (bus d'adresse 24 bits), soit 2^{22} blocs

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 325

Université Nice Sophia Antipolis

Correspondance directe (3)

Adresse mémoire en binaire

0001 0000	0011 0011 1001 11	00
Étiquette	Ligne + mot	

Diagram illustrating direct mapping between memory and cache. The memory is divided into blocks of 4 octets. The cache is divided into lines of 4 octets. The mapping table shows the correspondence between memory blocks and cache lines.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 326

Université Nice Sophia Antipolis

Correspondance associative (1)

- ✓ Eviter le problème d'association bloc-ligne
- ✓ Adresse du bloc = adresse de l'étiquette

Adresse bloc (s bits)

(w bits)

Adresse Etiquette
Adresse mots dans le bloc
- ✓ Opération de lecture
 - Parcours de l'ensemble de la table
- ✓ Défaut de cache
 - besoin d'un algorithme de remplacement ...

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 327

Université Nice Sophia Antipolis

Correspondance associative (2)

- ✓ Récapitulatif
 - Longueur de l'adresse = $(s + w)$ bits = 24 bits
 - Nombre d'unités adressables = 2^{s+w} octets = 2^{24} octets
 - Taille du bloc = 2^w octets = 2^2 octets = 4 octets
 - Taille de la ligne = 2^w octets = 2^2 octets = 4 octets
 - Nombre de blocs dans la mémoire = $2^{s+w} / 2^w = 2^s$ octets = 2^{22} o
 - Nombre de lignes dans le cache => indéterminé
 - Taille de l'étiquette = s bits = 22 bits
- ✓ Exemple
 - Cache de 64 Ko avec un bloc de 4 octets (2^{14} lignes)
 - Mémoire 16 Mo (bus d'adresse 24 bits), soit 2^{22} blocs

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 328

Université Nice Sophia Antipolis

Correspondance associative (3)

Adresse

000000	13579246
000004	

Données

Adresse mémoire en binaire

00 0101 1000 1100 1110 0111	00
Étiquette	mot

Diagram illustrating associative mapping between memory and cache. The memory is divided into blocks of 4 octets. The cache is divided into lines of 4 octets. The mapping table shows the correspondence between memory blocks and cache lines.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 329

Université Nice Sophia Antipolis

Correspondance associative par ensemble (1)

- ✓ Compromis entre les 2 approches précédentes
 - Profiter des avantages des 2 approches
 - Minimiser leurs inconvénients
- ✓ Le cache est divisé en V ensemble de K lignes
 - $m = k * v$
 - Fonction de correspondance : $i = j \text{ modulo } v$
 - $i = n^\circ$ d'ensemble du cache
 - $j = n^\circ$ du bloc
 - $m = \text{nb de ligne de cache}$
- ✓ Le bloc est alors placer dans n'importe quelle ligne de ensemble identifié

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 330

Université Nice Sophia Antipolis

Correspondance associative par ensemble (2)

- ✓ Récapitulatif
 - Longueur de l'adresse = $(s + w)$ bits = 24 bits
 - Nombre d'unités adressables = 2^{s+w} octets = 2^{24} octets
 - Taille du bloc = 2^w octets = 2^2 octets = 4 octets
 - Taille de la ligne = 2^w octets = 2^2 octets = 4 octets
 - Nombre de blocs dans la mémoire = $2^{s+w} / 2^w = 2^s$ octets = 2^{22} o
 - **Nombre de ligne dans l'ensemble = k**
 - **Nombre d'ensemble $v = 2^d$**
 - **Nombre de lignes dans le cache = $k * v = k * 2^d$**
 - **Taille de l'étiquette = $(s - d)$ bits**
- ✓ Exemple
 - Cache de 64 Ko avec un bloc de 4 octets (2^{14} lignes)
 - Mémoire 16 Mo (bus d'adresse 24 bits), soit 2^{22} blocs

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 331

Université Nice Sophia Antipolis

Correspondance associative par ensemble (3)

Adresse mémoire en binaire

0 0010 1100 0 1100 1110 0111 00

Étiquette Ensemble mot

Cache

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 332

Université Nice Sophia Antipolis

Algorithmes de remplacement

- ✓ Quel bloc enlever quand on charge un nouveau bloc ?
 - Correspondance direct : pas de choix, problème résolu
 - Correspondance associative : au choix dans le cache
 - Correspondance associative k voies : au choix dans l'ensemble
- ✓ Algorithmes implémentées en hardware
 - Aléatoire : un remplace un bloc de manière aléatoire
 - FIFO (First In First Out) : remplace le bloc le plus ancien
 - LFU (Least Frequently Used) : remplace le bloc le moins référencé.
 - LRU (Least Recently Used) : remplace le bloc qui se trouve dans le cache depuis le plus longtemps sans avoir été référencé

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 333

Université Nice Sophia Antipolis

Stratégie d'écriture

- ✓ Quand on remplace un bloc, que doit on faire du bloc qui se trouve dans le cache ?
 - Si bloc non modifié : rien à faire
 - Si bloc modifié : sauvegarder le résultat en mémoire
 - Problème : cohérence mémoire <-> cache

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 334

Université Nice Sophia Antipolis

Autres éléments sur les caches

- ✓ Taille des blocs
 - Plus les blocs sont gros, moins il y en aura en mémoire, plus on remplacera des blocs récents
 - Plus les blocs sont gros, plus on charge de données éloignées de celles demandées.
- ✓ Caches multi-niveaux
 - Actuellement L1, L2 et L3
 - Inclusif (core 2 duro, core i7) ou (pseudo-)exclusif (Phenom)
- ✓ Caches unifiés ou séparés
 - Unifiés : le cache mélange les données et les instructions
 - Séparés : un cache pour les instructions, un autre pour les données

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 335

Université Nice Sophia Antipolis

Plan

- ✓ Introduction
- ✓ Représentation de l'information
- ✓ Arithmétique binaire
- ✓ Algèbre de Boole
- ✓ Circuits séquentiels/Automates
- ✓ Architecture type Von Neumann
- ✓ Couche d'assemblage / Assembleur 8086
- ✓ Mécanismes d'interruptions
- ✓ Fonctions avancées de processeurs modernes
- ✓ Gestion de la mémoire
- ✓ Traduction de programmes, compilation édition de liens, chargement

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 336

Chaîne d'exécution

- ✓ Assembleur/Traducteur (exemple : MASM, TASM)
 - L'assembleur est chargé d'effectuer la traduction d'un module source (.asm) en module objet (.obj).
- ✓ Éditeur de Lien (exemple : LINK)
 - L'éditeur de lien est chargé de regrouper plusieurs modules objets (.obj) afin de produire un programme « exécutable » (.exe)
- ✓ Chargeur (exemple : MS-DOS)
 - Lorsque on demande au système de lancer un exécutable, celui-ci fait appel au « chargeur ».
 - Son rôle est de copier le programme en mémoire, et d'initialiser certains registres (compteur ordinal entres autres)

Assembleur (1)

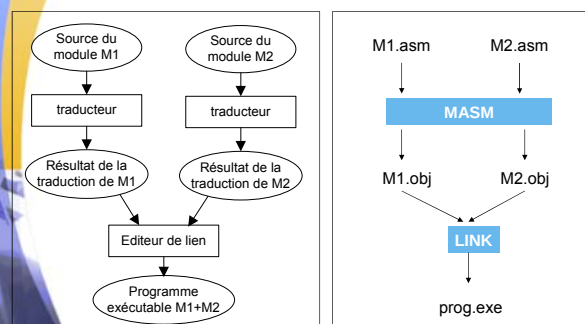
- ✓ L'opération d'assemblage traduit chaque instruction du programme source en une instruction machine.
- ✓ Le résultat de l'assemblage est enregistré dans un fichier avec l'extension .OBJ (fichier objet).
- ✓ Le fichier .OBJ n'est pas directement exécutable. En effet, il arrive fréquemment que l'on construise un programme exécutable à partir de plusieurs fichiers sources.
- ✓ Il faut « relier » les fichiers objets à l'aide d'un utilitaire nommé éditeur de lien (même si l'on en a qu'un seul).

Assembleur (2)

- ✓ L'éditeur de liens fabrique un fichier exécutable , avec l'extension .EXE.
- ✓ Le fichier .EXE est directement exécutable.
- ✓ Un utilitaire spécial du système d'exploitation (MSDOS ici), le chargeur est responsable de la lecture du fichier exécutable, de son implantation en mémoire principale, puis du lancement du programme.

Compilation, édition de liens

Exemple d'un programme constitué de 2 fichiers/modules M1 et M2



Définitions

- ✓ Objet physique : objet directement manipulable par la machine.
 - valeurs, emplacements mémoire, instructions, etc.
- ✓ Objet logique : objet manipulé par le programme
 - variables, constantes, sous-programmes, etc.
- ✓ La liaison consiste à associer à chaque objet logique du programme un objet physique de la machine.
- ✓ Le problème de la liaison se ramène à associer une adresse à chaque identificateur du programme
 - Certaines adresses ne sont calculées qu'à l'exécution, on va s'intéresser à la partie du travail effectuée avant l'exécution

Liaison : exemple (1)

- ✓ Accéder à une variable globale/statique ?
 - On utilise une instruction machine qui contient une adresse
 - Adressage direct (variable simple) ou indexé (tableau statique)
 - Liaison = associer une adresse à l'identificateur de la variable
- ✓ Accéder à une variable locale/paramètre d'un Sous Programme ?
 - Ces variables sont stockées dans la zone de pile,
 - Instruction machine qui contient un déplacement (basé)
 - Liaison = associer un déplacement à l'identificateur

Liaison : exemple (2)

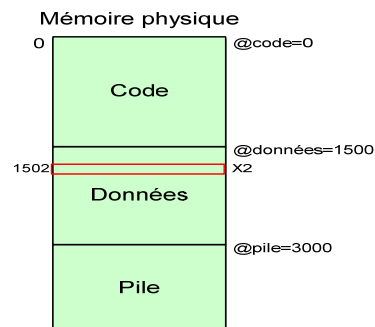
- ✓ Accéder à une variable dans le tas ?
 - Les objets du tas sont créés/détruits pendant l'exécution
 - Pas d'identificateur désignant directement un élément du tas
- ✓ Accéder à une instruction ?
 - Liaison = adresse de l'instruction dans le programme

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

343

Machine à un niveau d'adressage



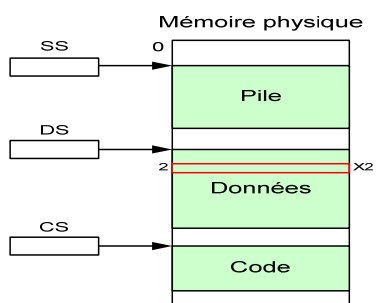
Un seul niveau d'adressage
(processeur 68000)

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

344

Machine à deux niveaux d'adressage



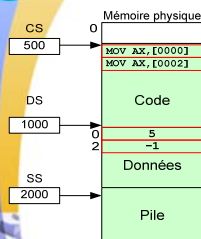
Deux niveaux d'adressage
(cas du processeur 8086)

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

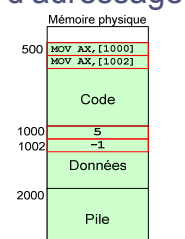
345

Machine à 1 et 2 niveaux d'adressages



Deux niveaux d'adressage
(cas du processeur 8086)

Le code n'est pas lié à l'adresse
d'implantation des données dans la
mémoire physique



Un seul niveau d'adressage
(cas du processeur 68000)

Le code est lié à l'adresse
d'implantation des données dans la
mémoire physique

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

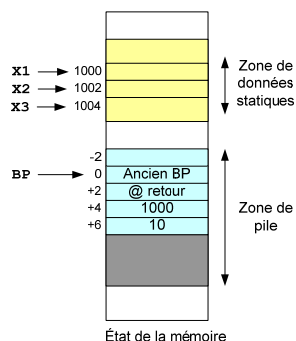
346

Exemple (1)

MOV [1000], 5 adressage direct
MOV [1004], 8 adressage direct
PUSH 10 adressage immédiat
CALL SP adressage immédiat

MOV AX, [BP+6] adressage basé/BP
MOV [1002], AX adressage direct
MOV AX, 0 adressage par valeur
MOV [BP-2], AX adressage basé/BP

MOV AX, 0
MOV BX, [BP+4] adressage basé/BP
MOV [BX], AX adressage indirect



Comment traduire (1) (2) et (3) ?

Janvier 2012

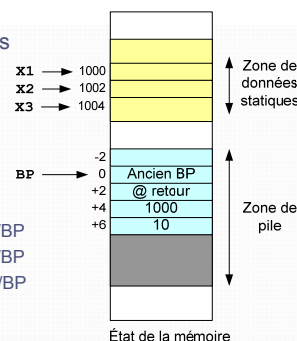
Gaëtan Rey - DUT Informatique de Nice

347

Exemple (2)

- ✓ On en déduit les informations de liaisons

- X1 → 1000 (adresse) direct
- X2 → 1002 (adresse) direct
- X3 → 1004 (adresse) direct
- Y → -2 (déplacement) basé/BP
- V → +6 (déplacement) basé/BP
- R → +4 (déplacement) basé/BP



Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

348

Mécanisme de liaison

Module M1

```

var ent X1, X2, X3 ;
procédure SP(ent V,ptr ent R)
var ent Y ;
Début
  (...)
  X2:=V; (1)
  Y:=0; (2)
  R:=0; (3)
  (...)
fin;

extern procédure SP2(ent a);
début
  (...)
  X1 :=5; X3 :=8;
  SP(10, @X1);
  SP2(10);
  (...)
fin;

```

référence externe :
L'objet est utilisé dans le module, mais défini à l'extérieur

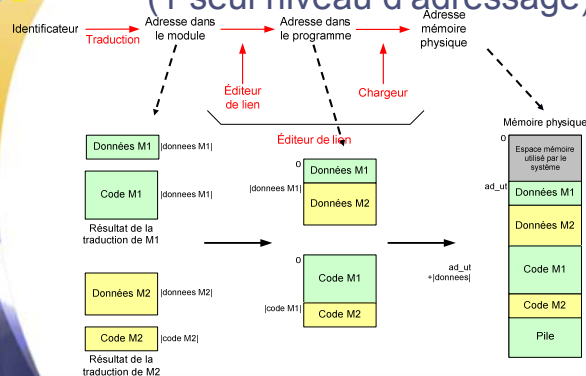
Mécanisme de liaison (3)

- ✓ Le compilateur ne connaît qu'un module à la fois :
 - Les adresses produites sont des adresses à l'intérieur de chaque module formant le programme.
 - Il ne peut pas traduire l'accès aux objets externes (ex : SP2)
- ✓ Le rôle de l'éditeur de liens sera de :
 - Regrouper le code de M1 et M2 pour former le code final
 - Regrouper les données statiques de M1 et M2 pour obtenir la zone de données statiques finale
 - Résoudre les références externes
- ✓ L'éditeur de liens a donc transformé les adresses dans le module en adresses dans le programme

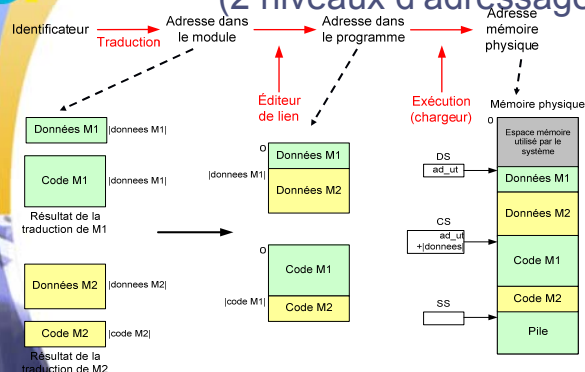
Mécanisme de liaison (4)

- ✓ Pour les données, le regroupement de modules implique des translations d'adresses qui sont fonction de la taille des modules formant le programme
 - L'adresse de u dans le module M2 est 0
 - L'adresse de u dans le programme complet est 6
- ✓ Il reste encore à traduire les adresses dans le programme avec l'adresse d'implantation du programme en mémoire. Ceci peut être réalisée soit par
 - l'éditeur de liens (qui effectue les deux translations en même temps)
 - le chargeur (qui implante le programme en mémoire)

Exemple liaison (1 seul niveau d'adressage)



Exemple liaison (2 niveaux d'adressage)



Exercise

- ✓ Donner le résultat de la traduction complète de l'exemple vu précédemment, en indiquant pour chaque instruction si elle nécessite une translation.
 - On suppose qu'il y a séparation du code et des données
 - Les instructions occupent toutes 4 octets
- ✓ Indiquer le travail de translation effectué et le résultat de l'édition de lien dans le cas où la taille du système est de
 - adresse d'implantation du programme en mémoire = 1000
 - code M1 = 150,
 - code M2 = 40

Analyse lexicale

- ✓ Principe
 - On découpe le texte du programme source en unités lexicales
 - Celles-ci correspondent aux non terminaux de la grammaire.
- ✓ Exemple : `MOV AX,125`

avec

<ident> = lettre {lettre | chiffre}*
<constante> = chiffre*
- ✓ Les critères de découpages peuvent être +/- précis :
 - Purement lexical : ce qu'on vient de voir
 - Sémantique (on va différencier les identificateurs des directives et des mnémoniques d'instructions en consultant des tables)

Grammaire (1)

- ✓ Formalisme permettant de décrire la structure des programmes
 - Décrit la syntaxe → en décrit aussi indirectement la sémantique
- ✓ Une grammaire est formée d'un ensemble de règles de production et d'un symbole de départ
 - Chaque règle est formée de 2 parties
 - La partie gauche : le nom de la construction syntaxique
 - La partie droite : une forme possible de la construction syntaxique

Grammaire (2)

- ✓ Exemple de grammaire :
 - expression → '(' expression opérateur expression ')'
 - expression → '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
 - opérateur → '+' | '-' | '*' | '/'
- ✓ Exemple de dérivation
 - expression
 - '(' expression opérateur expression ')'
 - '(' '3' opérateur expression ')'
 - '(' '3' '+' expression ')'
 - '(' '3' '+' '(' expression opérateur expression ')')'
 - '(' '3' '+' '(' '2' opérateur expression ')')'
 - '(' '3' '+' '(' '2' '*' expression ')')'
 - '(' '3' '+' '(' '2' '*' '5' ')')'

Analyse syntaxique et production

- ✓ Principe de l'analyse syntaxique :
 - On vérifie que les unités lexicales se présentent dans un ordre conforme à la grammaire du langage
 - Exemple :
 - `MOV AX,0` est correct
 - `MOV BX+1,0` est incorrect (non reconnu par la grammaire)
- ✓ Principe de la production :
 - À partir des unités lexicales reconnues syntaxiquement correctes, on produit le « module objet »
- ✓ Remarque :
 - Dans le cas d'un langage d'assemblage, la simplicité de la grammaire permet de réaliser l'analyse syntaxique et la production en utilisant un automate d'état fini

Exemple : MOV AX,12

- ✓ Analyse lexicale :
 - ' ' (Séparateur : Espace)
 - MOV (symbole) : correspond à un code d'instruction
 - ' ' (Séparateur : Espace)
 - AX (symbole) : correspond à un code de registre
 - , (Séparateur : Virgule)
 - 12 (12)
- ✓ Analyse syntaxique et production

- A1 : produit la suite de bits correspondant au code de l'instruction MOV (table)
 - A2 : produit la suite de bits codant AX (table)
 - A3 : converti les caractères '1' et '2' en 0...01100 et produit cette suite de bits

Définitions (1)

- ✓ Identificateur
 - Symbole qui désigne un objet défini par le programmeur
- ✓ Définition d'un identificateur
 - Occurrence qui permet d'associer un objet à un identificateur
- ✓ Référence à un identificateur
 - Occurrence d'un ident. demandant l'utilisation de l'objet désigné
- ✓ Référence en avant
 - Référence faite avant la définition de l'identificateur

Définitions (2)

- ✓ Référence externe
 - Référence faite à un identificateur défini dans un module externe
- ✓ Expression calculable
 - Expression dont tous les éléments sont définis à la rencontre de l'expression
- ✓ Expression absolue/translatable
 - Expression dont le résultat est (in)dépendant de l'implantation du programme en mémoire

Traitement des identificateurs

- ✓ L'objectif de la traduction :
 - Produire une suite de bits qui est chargée en mémoire et interprétée par la machine pour réaliser le calcul demandé
- ✓ Elle ne produit pas directement cette suite de bits
 - Certaines parties du programme ne sont pas connues du traducteur (assemblage séparé)
 - Cela peut-être inutile, par exemple dans le cas d'une directive de type DW 100 DUP (?).
 - Inutile de produire 100 mots ayant une valeur choisie au hasard
 - Il suffit de conserver l'information « ici il faut sauter 100 mots »
- ✓ L'assembleur produit une description (image) de la mémoire au moment de l'exécution

Construction de l'Image de la mémoire

- ✓ Compteur d'emplacement (CE)
 - Adresse dans le module du prochain emplacement à réserver dans l'image de la mémoire
 - Correspond au nombre d'emplacements déjà réservés
- ✓ Vecteur de translation
 - Informations indiquant si le contenu de l'emplacement doit être traduit ou non

Au fur et à mesure de la traduction on va remplir l'image du programme et le vecteur de translation, puis mettre à jour le compteur d'emplacement.

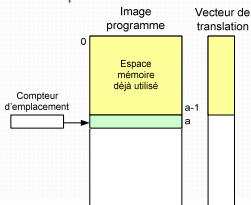


Table des identificateurs (1)

- ✓ Un identificateur désigne :
 - Un emplacement mémoire (donnée ou instruction)
 - Une valeur (qui peut être une adresse)
- ✓ On lui associe certaines informations
 - Celles-ci doivent décrire l'objet physique associé
 - Si c'est un emplacement mémoire, on lui associe une adresse
 - Si c'est une valeur, on lui associe sa représentation binaire
 - Il faut aussi indiquer si l'objet physique est translatable

Table des identificateurs (2)

- ✓ Ces informations de liaison sont stockées dans la table des identificateurs, qui contient :
 - L'identificateur (son nom)
 - La description de l'objet désigné
 - L'information associée (adresse de l'empl., valeur)
 - Son type (absolu, translatable)

Définition d'un identificateur (1)

- ✓ Avant référence
 - Quand on traite la définition d'un identificateur :
 - vérifier qu'il n'y figure pas déjà avec une autre définition (sinon on a une erreur de double définition)
 - Créer une entrée dans la table des identificateurs (on note cette table TAB_IDENT) et on l'initialise
 - Le contenu de cette entrée dépend de la directive associée à l'identificateur
 - Cas d'une directive {DB,DW} DUP (?)
 - Information associée = valeur du compteur d'emplacement
 - Type = translatable
 - Cas d'une directive INSTRUCTION
 - Information associée = valeur du compteur d'emplacement
 - Type = translatable

Université Nice Sophia Antipolis

Définition d'un identificateur (2)

- ✓ Avant référence
 - Cas d'une directive EQU
 - Information associée = info associée à l'opérande
 - Type = type de l'opérande
 - Il faut donc « aller voir » l'opérande
 - Si le type de l'opérande est une notation de constante
 - Information associée = représentation binaire de la valeur
 - Type = absolu
 - Si le type de l'opérande est un identificateur
 - On recherche cet identificateur dans TAB_IDENT et on recopie l'info associée

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 367

Université Nice Sophia Antipolis

Définition d'un identificateur (3)

- ✓ Avant référence
 - On recherche l'identificateur dans TAB_IDENT, et
 - On le trouve (on se place dans le cas où la définition a eu lieu)
 - on se sert des informations trouvées pour faire la traduction
 - La traduction va dépendre du contexte de la référence
 - Contexte : DB,DW <ident> DUP (?)
 - L'information associée doit-être absolue
 - Info trouvée dans TAB_IDENT = nombre d'emplacement à réserver

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 368

Université Nice Sophia Antipolis

Définition d'un identificateur (4)

- ✓ Avant référence
 - Contexte : DB,DW <ident>
 - Info trouvée dans TAB_IDENT = valeur à mettre dans l'emplacement réservé
 - Type = à mettre dans le vecteur de translation
 - Contexte : EQU <ident> : voir la définition
 - Contexte : INSTRUCTION
 - Info trouvée dans TAB_IDENT = valeur à mettre dans l'emplacement réservé
 - Type = à mettre dans le vecteur de translation

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 369

Université Nice Sophia Antipolis

Exemple

- ✓ Soit la séquence de déclaration d'identificateurs suivante


```
X DB 2 DUP(?)
N EQU 3
Y DB N DUP(?)
C1 EQU N
C2 EQU Y
```
- ✓ Remplir la table des identificateur correspondante

Image mémoire

?
?
?
?

ident	info	type
X		
N		
Y		
C1		
C2		

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 370

Université Nice Sophia Antipolis

Traitement des références avant

- ✓ Exemples


```
JMP saut (1)
...
saut: MOV AX, ...
```

```
MOV AX, [Y] (1)
...
Y DB 10
```

En (1), on ne peut pas traduire la référence à "saut" car on a pas encore vu sa définition. On effectuera cette traduction plus tard

- ✓ On va voir deux techniques pour résoudre ce problème
 - L'assemblage en deux passes, dans lequel on traite d'abord toutes les définitions, pour seulement ensuite s'occuper des références.
 - L'assembleur à pseudo simple passage, qui essaie de gérer la définition et références en même temps.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 371

Université Nice Sophia Antipolis

Assemblage à deux passages (1)

- ✓ On procède en deux étapes (passes) distinctes
 - On effectue d'abord le traitement des définitions des identificateurs, ce qui va permettre de remplir TAB_IDENT
 - Une fois TAB_IDENT remplie, on peut alors traiter les références aux identificateurs (ils sont tous définis)

Assemblage

```

graph LR
    MS([Module source]) --> AP1[Assembleur 1ère passe]
    AP1 --> CI([Code intermédiaire])
    CI --> AP2[Assembleur 2ème passe]
    AP2 --> MO([Module objet])
  
```

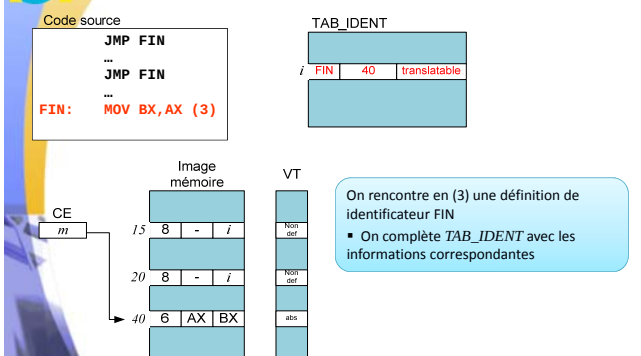
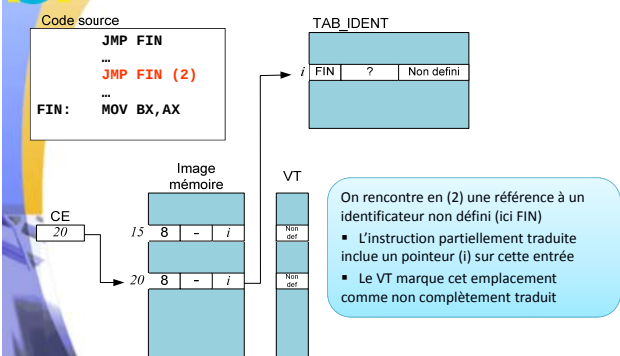
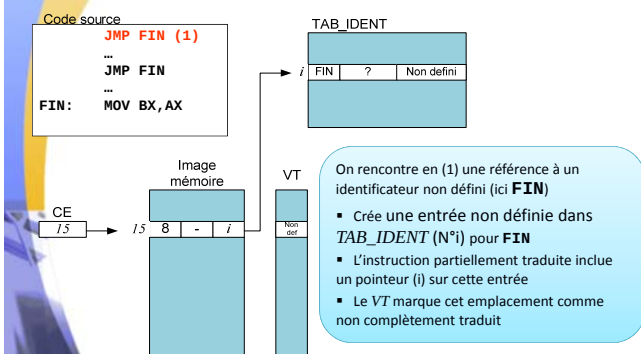
- ✓ Pour réaliser l'étape (1), on ne peut pas se contenter d'analyser la partie étiquette (partie gauche):
 - X EQU 20 ; à analyser pour donner sa valeur à X
 - Y DW 2 DUP (?) ; à analyser pour incrémenter le Compteur d'emplacement et donner des valeurs aux identificateurs suivants.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 372

- ✓ On a deux solutions extrêmes :
 - Le 1er passage ne fait que remplir TAB_IDENT
 - Le code intermédiaire correspond donc au code source
 - Au 2ème passage on fait toute la traduction, sauf le traitement de la partie étiquette.
 - Le 1er passage fait tout sauf le traitement des références en avant.
 - Le code intermédiaire est donc très proche du code objet
 - Au 2ème passage, il ne reste à traiter que les références en avant.
 - Dans la solution 2, lors d'une référence avant, on ne sait pas traduire l'identificateur au 1er passage,
 - Il faut donc conserver dans le code intermédiaire l'information qui permettra de le traduire à la 2ème passe.

- ✓ Quelle information doit-on conserver ?
 - A priori, il faut conserver le nom de l'identificateur référencé
- ✓ Où cette information va-t-elle être conservé ?
 - Pas directement dans le code, car ce n'est pas pratique
 - L'identificateur à une taille variable (chaîne de caractère)
 - Si l'ident est utilisé plusieurs fois on aura des redondances.
- ✓ Solution : on va conserver dans le code objet un pointeur sur l'identificateur.

- ✓ Comment fait-on ?
 1. Le nom de l'ident est stocké dans TAB_IDENT (en notant qu'il s'agit d'un ident non-défini)
 2. On conserve dans le code intermédiaire un pointeur sur l'entrée de TAB_IDENT contenant notre identificateur
 3. On note dans le Vecteur de Translation que la traduction de cet emplacement n'est pas terminée.
 4. Au 2ème passage, on va parcourir le Vecteur de Translation pour savoir quelles sont les références à compléter, et où trouver l'info correspondante



Université Nice Sophia Antipolis

Exemple (4)

Code source

```

JMP FIN
...
JMP FIN
...
FIN: MOV BX, AX
  
```

TAB_IDENT

i	FIN	40	translatable

Image mémoire

15	8	-	40
20	8	-	40
40	6	AX	BX

VT

trans
trans
abs

Pendant la 2^{ème} passe, on parcourt le VT, et on complète les emplacements non définis.

CE 40

Janvier 2012 Gaétan Rey - DUT Informatique de Nice 379

Université Nice Sophia Antipolis

Assembleur à pseudo-simple passage (1)

✓ Principe

- On veut résoudre les références en avant sans avoir à créer un code intermédiaire que l'on relit entièrement.
- On va produire le code objet en revenant en arrière pour le compléter si besoin.

✓ Mise en oeuvre

- Pour chaque symbole « référence en avant » on tient à jour une liste de tous les emplacements du code objet dans lesquels on a besoin de ranger la valeur du symbole
 - En fait une « liste des trous non traduits »

Janvier 2012 Gaétan Rey - DUT Informatique de Nice 380

Université Nice Sophia Antipolis

Assembleur à pseudo-simple passage (2)

✓ Fonctionnement

- À la 1^{ère} référence en avant, on crée une liste des trous pour cet identificateur (la liste ne contenant qu'un élément)
- À chaque nouvelle référence en avant, on ajoute l'emplacement correspondant à la liste.
- Dès que le symbole est défini, on remonte cette liste pour compléter le code objet avec la valeur du symbole

✓ Exemple :

```

(1) MOV AX, S
(2) MOV BX, S
(3) MOV CX, S
(4) S DB 10
  
```

Janvier 2012 Gaétan Rey - DUT Informatique de Nice 381

Université Nice Sophia Antipolis

Exemple (1)

(1) MOV AX, S
(2) MOV BX, S
(3) MOV CX, S
(4) S DB 10

On rencontre en (1) une référence à l'ident S :

- on crée une entrée vide dans TAB_IDENT
- on y ajoute un pointeur sur l'emplacement non traduit

TAB_IDENT

S	i	Non défini
---	---	------------

Image mémoire

i	MOV	AX	direct	-
---	-----	----	--------	---

VT

Janvier 2012 Gaétan Rey - DUT Informatique de Nice 382

Université Nice Sophia Antipolis

Exemple (2)

(1) MOV AX, S
(2) MOV BX, S
(3) MOV CX, S
(4) S DB 10

On rencontre en (2) une référence à l'identificateur S :

- on ajoute un pointeur sur l'emplacement i dans l'emplacement j.
- on met à jour le pointeur de TAB_IDENT à j.

TAB_IDENT

S	j	Non défini
---	---	------------

Image mémoire

i	MOV	AX	direct	-
j	MOV	BX	direct	i

VT

Janvier 2012 Gaétan Rey - DUT Informatique de Nice 383

Université Nice Sophia Antipolis

Exemple (3)

(1) MOV AX, S
(2) MOV BX, S
(3) MOV CX, S
(4) S DB 10

On rencontre en (3) une référence à l'identificateur S :

- on ajoute un pointeur sur l'emplacement j dans l'emplacement k.
- on met à jour le pointeur de TAB_IDENT à k.

TAB_IDENT

S	k	Non défini
---	---	------------

Image code

i	MOV	AX	direct	-
j	MOV	BX	direct	i
k	MOV	CX	direct	j

VT

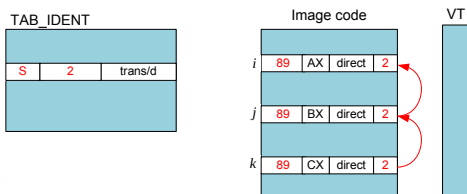
Janvier 2012 Gaétan Rey - DUT Informatique de Nice 384

Exemple (4)

(1) MOV AX, S
(2) MOV BX, S
(3) MOV CX, S
(4) S DB 10

On rencontre en (4) une définition pour S :

- on part du pointeur stocké dans TAB_IDENT pour retrouver tous les emplacements où la référence à S n'a pas été traduite.



Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

385

Comparaison des méthodes

- ✓ Les deux critères sont rapidité et place mémoire.
 - Le programme source et le code objet sont stockés sur le disque
 - On y accède donc plus lentement qu'en mémoire
- ✓ Approche à deux passages
 - On fait deux passages, donc a priori le traitement est plus long.
 - On a besoin que d'une seule ligne de code source et de code objet à la fois (peu de place)
- ✓ Approche à pseudo-simple passage
 - Une seule lecture, donc a priori traitement plus rapide
 - MAIS, il faut revenir en arrière sur le code objet, donc il faut pouvoir tout garder en mémoire (sinon on doit faire des accès disques très lents)

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

386

Traitement d'une définition externe

- ✓ Définition externe :
 - Définition d'un identificateur qui sera accessible à l'extérieur du module.
- ✓ Traitement effectué par l'assembleur
 - On stocke l'identificateur associé à la définition externe dans une table appelée TAB_ENTREE (table des points d'entrées)
 - On recopie dans TAB_ENTREE les informations de TAB_IDENT concernant cet identificateur.
 - NB : TAB_ENTREE fait partie du résultat de l'assemblage, et sera donc intégrée au code objet (TAB_IDENT reste interne)

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

387

Traitement d'une référence externe

- ✓ Référence externe :
 - Référence à un identificateur qui est défini dans un autre module, et qu'on ne peut donc pas traduire tout de suite.
- ✓ Traitement effectué par l'assembleur
 - On va conserver cette références externes : on stocke le nom de son identificateur dans la table TABLE_EXT (appelée table des références externes)
 - Au lieu de traduire la référence, on met en place un pointeur sur l'identificateur stocké dans la table des références externes.
 - On note dans le vecteur de translation (VT) que l'emplacement est non traduit (on marque ext).

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

388

Bilan : définition des identificateurs

- ✓ Table des identificateurs (TAB_IDENT)
 - Contenu = (nom_ident, info associée, type, translation)
 - Type = (mot | octet | instr | valeur)
 - Translation = (abs | trans | non def)
- ✓ Table des point d'entrées (TAB_ENTREE)
 - (nom_ident, info associée, type, translation)
- ✓ Table des externes (TAB_EXT)
 - Elle contient seulement (nom_ident)
- ✓ Code objet
 - Résultat de la traduction
 - Vecteur de translation = (abs, trans, ext, {non défini})

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

389

Bilan : traitement des externes

- ✓ À la rencontre de la directive EXTERNE
 - On stocke l'identificateur dans TAB_EXT
- ✓ À la rencontre de la directive PUBLIQUE
 - On stocke l'identificateur dans TAB_ENTREE

Janvier 2012

Gaëtan Rey - DUT Informatique de Nice

390

Université
Nice
Sophia Antipolis

Bilan : traitement des définitions

- ✓ Ident $\notin$ TAB_EXT et Ident $\notin$ TAB_IDENT
 - Définition d'un identificateur avant utilisation
- ✓ Ident $\in$ TAB_EXT
 - Erreur ! Identificateur à la fois local et externe !
- ✓ Ident $\in$ TAB_IDENT et définition
 - Erreur ! Des identificateurs locaux ne peuvent avoir le même nom
- ✓ Ident $\in$ TAB_IDENT, et non définition
 - Définition d'un identificateur pour lequel on a eu des références en avant

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 391

Université
Nice
Sophia Antipolis

Bilan : traitement des références

- ✓ Ident $\in$ TAB_EXT
 - C'est une référence à un identificateur externe
- ✓ Ident est défini, $\in$ TAB_IDENT et $\notin$ TAB_EXT
 - C'est une référence à un identificateur local défini.
- ✓ Ident est non défini, $\in$ TAB_IDENT et $\notin$ TAB_EXT
 - C'est une nouvelle référence en avant.
- ✓ Ident $\notin$ TAB_IDENT et $\notin$ TAB_EXT
 - C'est la première référence en avant à un identificateur local.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 392

Université
Nice
Sophia Antipolis

Bilan : résultats de la traduction

- ✓ TAB_EXT : table des références externes.
- ✓ TAB_ENTREE : table des définitions externes.
- ✓ Image mémoire + Vecteur de Translation.
 - Taille de l'image mémoire
 - Nombre de références externes,
 - Nombre de définitions externes.
- ✓ Adresse de la 1ère instruction à exécuter.
- ✓ Toutes ces informations sont stockées dans le fichier objet produit par l'assembleur.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 393

Université
Nice
Sophia Antipolis

Travail effectué par l'éditeur de liens (1)

- ✓ L'éditeur de lien (EdL)
 - Prend en entrée les fichiers objets résultant de l'assemblage des modules du programme
 - Produit en résultat un programme relié contenant la traduction complète de tout le programme.
 - Dans certains cas ce programme est directement exécutable,
 - Dans d'autres cas il y a une translation à faire avant de pouvoir exécuter le programme.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 394

Université
Nice
Sophia Antipolis

Travail effectué par l'éditeur de liens (2)

- ✓ Dans tous les cas, l'éditeur de lien a deux tâches
 - Regrouper les différents modules et faire les translations d'adresses rendues nécessaires par ce regroupement
 - Traduire les références externes laissées par l'assembleur.

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 395

Université
Nice
Sophia Antipolis

Regroupement des modules (1)

- ✓ Exemple 1 : une section par module
 - Code et données mélangées, un seul niveau d'adressage
 - Pour chaque module, le traducteur a produit une image de la mémoire du module.
 - Utilise des adresses dans le module
 - Contient un vecteur de translation
- ✓ On se donne trois modules M1, M2 et M3
 - On connaît la taille des modules (notée |M1|, |M2| et |M3|)
 - On connaît l'adresse d'implantation en mémoire

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 396

Regroupement des modules (2)

Image de la mémoire de M1
0
JM1-1

Image de la mémoire de M2
0
JM2-1

Image de la mémoire de M3
0
JM3-1

Regroupement des modules

Image de la mémoire du programme

Image de la mémoire de M1
 ad_ut

Image de la mémoire de M2
 $ad_ut + JM1$

Image de la mémoire de M3
 $ad_ut + JM1 + JM2$

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 397

Exemple avec une mémoire non segmentée

Module source M1

```
EXTRN X, Y, Z
V DW 3 DUP(?)
MOV AX, X
MOV BX, V
MOV CX, Y
MOV DX, Z
```

Module source M2

```
PUBLIC X, Y, Z
X DW 2 DUP(?)
Y DW 1 DUP(?)
Z EQU -1
MOV AX, X
MOV CX, Y
MOV DX, Z
```

Module objet M1

Image de la mémoire

0			
2			
4			
6	3	0	1
10	3	1	0
14	3	2	2
18	1	3	3

Vecteur de translation

V	
V	
V	
externe	
translatable	
externe	
externe	

Table des externes

1	X
2	Y
3	Z

Module objet M2

Image de la mémoire

0			
2			
4			
6	3	0	0
10	3	2	4
14	1	3	-1

Vecteur de translation

X	
X	
Y	
translatable	
translatable	
absolu	

Table des points d'entrée

X	0	translatable
Y	4	translatable
Z	-1	absolu

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 398

Regroupement des modules (3)

- On suppose $ad_ut = 1500$
- Résultat de la fusion des modules
 - Table d'implantation des modules
 - Image mémoire des modules regroupés

Table d'implantation

Nom	M1	M2
Adresse	1500	1522

Image de la mémoire du programme

1500			
1502			
1504			
1506	3	0	1522
1510	3	1	1500
1514	3	2	1526
1518	1	3	-1
1522			
1524			
1526			
1528	3	0	1522
1532	3	2	1526
1536	1	3	-1

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 399

Traduction des références externes (1)

- On construit un dictionnaire global
 - On regroupe pour cela les tables de définitions externes des modules du programme.
 - Les informations contenues dans ces tables sont traduites en se basant sur la table d'implantation.

Résultat de l'assemblage de M1

Image mémoire

0		
	k	

TAB_EXT

INIT	
------	--

Résultat de l'assemblage de M2

Image mémoire

0		

TAB_ENTREE

INIT	t/c	1500
------	-----	------

Construit pendant l'édition de liens

DICT_GLOBAL

INIT	t/c	2500
------	-----	------

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 400

Traduction des références externes (2)

- On cherche la définition des références non traduites
 - On recoupe les définitions du dictionnaire global (DICT_GLOB) avec celles des tables de références externes (TAB_EXT)

Résultat de l'assemblage de M1

Image mémoire

0		
	k	

TAB_EXT

INIT	
------	--

Résultat de l'assemblage de M2

Image mémoire

0		

TAB_ENTREE

INIT	t/c	1500
------	-----	------

Construit pendant l'édition de liens

DICT_GLOBAL

INIT	t/c	2500
------	-----	------

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 401

Traduction des références externes (3)

- On effectue la traduction
 - On utilise les informations du dictionnaire global

Résultat de l'assemblage de M1

Image mémoire

0		
	k	

TAB_EXT

INIT	
------	--

Résultat de l'assemblage de M2

Image mémoire

0		

TAB_ENTREE

INIT	t/c	1500
------	-----	------

Construit pendant l'édition de liens

DICT_GLOBAL

INIT	t/c	2500
------	-----	------

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 402

Exemple (1)

- ✓ assemblage du fichier assep3.asm
 - le fichier objet (.OBJ) n'est pas directement lisible
 - on va regarder le contenu du fichier .lst
 - toutes les références n'ont pas été traduites

```

0000      .MODEL SMALL
0000      .STACK
0000      .CODE
0000      EXTRN INIT:NEAR, RANGER:NEAR
0000      DATA
0000      31      C1      DB      '1'
0000      = 0030      C2      EQU      '0'
0001      .CODE
0001      0000      JUMP     INIT
0001      0000      MOV     AX, 8DATA
0001      0000      MOV     DS, AX
0001      0000      CALL    INIT
0001      0000      MOV     AX, C1
0001      0000      CALL    RANGER
0001      0000      MOV     AX, C2
0001      0000      CALL    RANGER
0001      0000      MOV     AX, 4CH
0001      0000      INT     21H
0001      0000      INT     21H

```

e : référence externe
r : référence translatable (relative)
s : initialisation du segment

Exemple (2)

- ✓ L'assemblage séparé produit trois images de modules
 - Assep1: données (0 to 12), code ((B)16 to 26)
 - Assep2: code ((15)16 to 22)
 - Assep3: données (0 to 1), code ((16)16 to 23)
- ✓ Le travail de l'édition de liens va être double
 - Il doit regrouper (fusionner les modules)
 - Il doit résoudre les références externes

Regroupement des modules (4)

- ✓ On va construire une table d'implantation des modules
 - Indique l'adresse de début des modules dans le programme
 - Elle dépend de l'ordre dans lequel les modules sont liés
 - Elle dépend de la taille des modules
 - On l'utilise pour effectuer les translations
- ✓ Exemple :
 - link assep1 assep2 assep3

nom	assep1	assep2	assep3
code	0	1A ₁₆	30 ₁₆
données	0	0	15 ₁₆

Le chargeur

- ✓ Principe
 - C'est un programme qui fait partie du système d'exploitation
 - Appelé à chaque fois que l'utilisateur veut exécuter un programme
 - Par exemple lorsque l'on saisit le nom d'un fichier exécutable dans une fenêtre de commande
- ✓ Fonctionnement
 - Détermine où il y a de la mémoire libre sur la machine pour implanter le programme
 - Lit le programme à partir de son support de stockage et le ranger en mémoire (avec éventuellement une translation)
 - Configure la machine pour que le programme utilisateur puisse d'exécuter correctement

Exemple du 8086 (1)

- ✓ L'Édition de liens produit du code exécutable
 - Toutes les références sont traduites
 - Les adresses sont celles figurant dans les instructions
 - Le programme est indépendant de l'endroit où il est implanté
 - La transformation d'adresse a lieu à l'exécution

Exemple du 8086 (2)

- ✓ La configuration de la machine va consister à
 - Initialiser les registres de segment CS,DS et SS
 - Lancer l'exécution (initialiser IP)

Machine sans registres de segment (1)

✓ Le programme contient des adresses physiques

- Il dépend de l'endroit où il est implanté en mémoire
- Il n'y a pas de transformations d'adresses à l'exécution
- Exemple :

Implantation en 1000

```

0
1000
MOV AL, [1000]

```

Implantation en 1500

```

0
1500
MOV AL, [1500]

```

✓ On a deux approches ...

Machine sans registres de segment (2)

avant exécution

pendant exécution

avant exécution

pendant exécution

ident → ASS → adresses dans le module → EDL → adresses dans le programme → chargeur → adresses physiques → exécution

ident → ASS → adresses dans le module → EDL → Adresse physique → chargeur → adresses relatives → exécution

• L'EdL fait la translation due au regroupement des modules

• Le chargeur fait la translation due à l'implantation (le résultat de l'EdL contient un VT)

• L'EdL fait les deux translations à la fois.

• Le résultat de l'EdL ne contient pas de VT.

Fichiers .obj

✓ Exemple : représentation en hexa du fichier ASSEP2.OBJ

```

00000000 80 0c 00 0a 61 73 73 65 70 32 2e 61 73 6d ad 88 ....assep2.asm-
00000010 1f 00 00 00 54 75 72 62 6f 20 41 73 73 65 6d 62 ....Turbo Assemb
00000020 6c 65 72 20 20 56 65 72 73 69 6f 6e 20 32 2e 30 ler Version 2.0
00000030 b9 88 12 00 40 e9 00 7e 41 32 0a 61 73 73 65 70 9...!~A2,assep
00000040 32 2e 61 73 6d 85 88 03 00 40 e9 4c 96 02 00 02.asm...!H...
00000050 68 88 03 00 40 a1 94 96 0c 00 05 5f 54 45 58 54 h...!...TEXT
00000060 04 43 4f 44 45 96 98 07 00 48 16 00 02 03 01 fd .CODE...H...}
00000070 96 0c 00 05 5f 44 41 54 41 04 44 41 54 41 c2 98 ....DATA.DATAB.
00000080 07 00 48 00 00 04 05 01 0f 96 04 00 05 53 54 41 .H...STAC
00000090 43 4b 05 53 54 41 43 4b 67 98 07 00 74 00 04 06 CK.STACKg...t...
000000a0 07 01 db 96 08 00 06 44 47 52 4f 55 50 8b 9a 06 [...].DGROUP...
000000b0 00 08 ff 03 ff 02 55 8c 13 00 04 54 4d 41 58 00 ....U...TMAX.
000000c0 05 4e 42 43 41 52 00 03 54 41 42 00 9d 88 04 00 .NBCAR.TAB.^...
000000d0 00 01 06 52 41 4e 47 45 52 00 00 9d 88 04 00 ....RANGER...
000000e0 40 a2 01 91 a0 1a 00 01 00 00 8b 36 00 00 81 fe 8".....6...~
000000f0 00 00 74 0b 88 84 00 00 83 c6 01 89 36 00 00 c3 .t...F.6.C
00000100 ae 9c 14 00 c4 02 16 01 02 c4 06 56 01 c4 0c 16 ...D...D.V.D..
00000110 01 03 c4 13 16 01 02 76 8a 02 00 00 74 ...D...v...t

```

Tables des externes

✓ Début de la table :

✓ Chaque identificateur :

8C xx xx

Taille de la table

xx xx xx 00

Taille en caractères de l'ident identificateur Marqueur de fin

```

00000000 80 0c 00 0a 61 73 73 65 70 32 2e 61 73 6d ad 88 ....assep2.asm-
00000010 1f 00 00 00 54 75 72 62 6f 20 41 73 73 65 6d 62 ....Turbo Assemb
00000020 6c 65 72 20 20 56 65 72 73 69 6f 6e 20 32 2e 30 ler Version 2.0
00000030 b9 88 12 00 40 e9 00 7e 41 32 0a 61 73 73 65 70 9...!~A2,assep
00000040 32 2e 61 73 6d 85 88 03 00 40 e9 4c 96 02 00 02.asm...!H...
00000050 68 88 03 00 40 a1 94 96 0c 00 05 5f 54 45 58 54 h...!...TEXT
00000060 04 43 4f 44 45 96 98 07 00 48 16 00 02 03 01 fd .CODE...H...}
00000070 96 0c 00 05 5f 44 41 54 41 04 44 41 54 41 c2 98 ....DATA.DATAB.
00000080 07 00 48 00 00 04 05 01 0f 96 04 00 05 53 54 41 .H...STAC
00000090 43 4b 05 53 54 41 43 4b 67 98 07 00 74 00 04 06 CK.STACKg...t...
000000a0 07 01 db 96 08 00 06 44 47 52 4f 55 50 8b 9a 06 [...].DGROUP...
000000b0 00 08 ff 03 ff 02 55 8c 13 00 04 54 4d 41 58 00 ....U...TMAX.
000000c0 05 4e 42 43 41 52 00 03 54 41 42 00 9d 88 04 00 .NBCAR.TAB.^...
000000d0 00 01 06 52 41 4e 47 45 52 00 00 9d 88 04 00 ....RANGER...
000000e0 40 a2 01 91 a0 1a 00 01 00 00 8b 36 00 00 81 fe 8".....6...~
000000f0 00 00 74 0b 88 84 00 00 83 c6 01 89 36 00 00 c3 .t...F.6.C
00000100 ae 9c 14 00 c4 02 16 01 02 c4 06 56 01 c4 0c 16 ...D...D.V.D..
00000110 01 03 c4 13 16 01 02 76 8a 02 00 00 74 ...D...v...t

```

Tables des publics

✓ Pour chaque identificateur :

90 xx xx xx xx xx xx xx 00

Taille de l'entrée Segment 01 : code 02 : données Taille de l'ident identificateur Marqueur de fin

```

00000000 80 0c 00 0a 61 73 73 65 70 32 2e 61 73 6d ad 88 ....assep2.asm-
00000010 1f 00 00 00 54 75 72 62 6f 20 41 73 73 65 6d 62 ....Turbo Assemb
00000020 6c 65 72 20 20 56 65 72 73 69 6f 6e 20 32 2e 30 ler Version 2.0
00000030 b9 88 12 00 40 e9 00 7e 41 32 0a 61 73 73 65 70 9...!~A2,assep
00000040 32 2e 61 73 6d 85 88 03 00 40 e9 4c 96 02 00 02.asm...!H...
00000050 68 88 03 00 40 a1 94 96 0c 00 05 5f 54 45 58 54 h...!...TEXT
00000060 04 43 4f 44 45 96 98 07 00 48 16 00 02 03 01 fd .CODE...H...}
00000070 96 0c 00 05 5f 44 41 54 41 04 44 41 54 41 c2 98 ....DATA.DATAB.
00000080 07 00 48 00 00 04 05 01 0f 96 04 00 05 53 54 41 .H...STAC
00000090 43 4b 05 53 54 41 43 4b 67 98 07 00 74 00 04 06 CK.STACKg...t...
000000a0 07 01 db 96 08 00 06 44 47 52 4f 55 50 8b 9a 06 [...].DGROUP...
000000b0 00 08 ff 03 ff 02 55 8c 13 00 04 54 4d 41 58 00 ....U...TMAX.
000000c0 05 4e 42 43 41 52 00 03 54 41 42 00 9d 88 04 00 .NBCAR.TAB.^...
000000d0 00 01 06 52 41 4e 47 45 52 00 00 9d 88 04 00 ....RANGER...
000000e0 40 a2 01 91 a0 1a 00 01 00 00 8b 36 00 00 81 fe 8".....6...~
000000f0 00 00 74 0b 88 84 00 00 83 c6 01 89 36 00 00 c3 .t...F.6.C
00000100 ae 9c 14 00 c4 02 16 01 02 c4 06 56 01 c4 0c 16 ...D...D.V.D..
00000110 01 03 c4 13 16 01 02 76 8a 02 00 00 74 ...D...v...t

```

Valeurs produites

✓ Traduction des instructions et données initialisées

– Les parties non initialisées ne sont pas traduites

A0 xx xx xx xx xx

Taille Segment 01 : code 02 : données Adresse Valeur

```

00000000 80 0c 00 0a 61 73 73 65 70 32 2e 61 73 6d ad 88 ....assep2.asm-
00000010 1f 00 00 00 54 75 72 62 6f 20 41 73 73 65 6d 62 ....Turbo Assemb
00000020 6c 65 72 20 20 56 65 72 73 69 6f 6e 20 32 2e 30 ler Version 2.0
00000030 b9 88 12 00 40 e9 00 7e 41 32 0a 61 73 73 65 70 9...!~A2,assep
00000040 32 2e 61 73 6d 85 88 03 00 40 e9 4c 96 02 00 02.asm...!H...
00000050 68 88 03 00 40 a1 94 96 0c 00 05 5f 54 45 58 54 h...!...TEXT
00000060 04 43 4f 44 45 96 98 07 00 48 16 00 02 03 01 fd .CODE...H...}
00000070 96 0c 00 05 5f 44 41 54 41 04 44 41 54 41 c2 98 ....DATA.DATAB.
00000080 07 00 48 00 00 04 05 01 0f 96 04 00 05 53 54 41 .H...STAC
00000090 43 4b 05 53 54 41 43 4b 67 98 07 00 74 00 04 06 CK.STACKg...t...
000000a0 07 01 db 96 08 00 06 44 47 52 4f 55 50 8b 9a 06 [...].DGROUP...
000000b0 00 08 ff 03 ff 02 55 8c 13 00 04 54 4d 41 58 00 ....U...TMAX.
000000c0 05 4e 42 43 41 52 00 03 54 41 42 00 9d 88 04 00 .NBCAR.TAB.^...
000000d0 00 01 06 52 41 4e 47 45 52 00 00 9d 88 04 00 ....RANGER...
000000e0 40 a2 01 91 a0 1a 00 01 00 00 8b 36 00 00 81 fe 8".....6...~
000000f0 00 00 74 0b 88 84 00 00 83 c6 01 89 36 00 00 c3 .t...F.6.C
00000100 ae 9c 14 00 c4 02 16 01 02 c4 06 56 01 c4 0c 16 ...D...D.V.D..
00000110 01 03 c4 13 16 01 02 76 8a 02 00 00 74 ...D...v...t

```

Université Nice Sophia Antipolis

Vecteurs de translation

- ✓ Début de la table en 9C xx xx
 - Seuls les emplacements à traduire et les références externes sont décrites
 - Pour chaque valeur à modifier ou à compléter :

```

00000000 80 0c 00 0a 61 73 73 65 70 32 2e 61 73 6d ad 88 ....assep2.asm-.
00000010 1f 00 00 00 54 75 72 62 6f 20 41 73 73 65 6d 62 ....Turbo Assemb
00000020 6c 65 72 20 20 56 65 72 73 69 6f 6e 20 32 2e 30 1er Version 2.0
00000030 b9 88 12 00 40 e9 00 7e 41 32 0a 61 73 73 65 70 9...@i.-A2.assep
00000040 32 2e 61 73 6d 85 88 03 00 40 e9 4c 96 02 00 00 2.asm...@iL....
00000050 68 88 03 00 40 a1 94 96 0c 00 05 5f 54 45 58 54 h...@!....._TEXT
00000060 04 43 4f 44 45 96 98 07 00 48 16 00 02 03 01 fd .CODE....H.....}
00000070 96 0c 00 05 5f 44 41 54 41 04 44 41 54 41 c2 98 ....DATA.DATAB.
00000080 07 00 48 00 00 04 05 01 0f 96 04 00 05 53 54 41 ..H.....STA
00000090 43 4b 05 53 54 41 43 4b 67 98 07 00 74 00 04 06 CK.STACKg...t...
000000a0 07 01 db 96 08 00 06 44 47 52 4f 55 50 8b 9a 06 ..[.....DGROUP...
000000b0 00 08 ff 03 ff 02 55 8c 13 00 04 54 4d 41 58 00 .....U...TMXK.
000000c0 05 4e 42 43 41 52 00 03 54 41 42 00 de 90 0d 00 .NBCAR..TAB.^...
000000d0 00 01 06 52 41 4e 47 45 52 00 00 9d 88 04 00 ...RANGER.....
000000e0 40 a2 01 91 a0 1a 00 01 00 00 8b 36 00 00 81 fe @".....6...~
000000f0 00 00 74 0b 88 84 00 00 83 c6 01 89 36 00 00 c3 .t.....F..6..C
00000100 ae 9c 14 00 c4 02 16 01 02 c4 06 56 01 c4 0c 16 .....D....D.V.D..
00000110 01 03 c4 13 16 01 02 76 8a 02 00 00 74 .....v.....t

```

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 415

Université Nice Sophia Antipolis

Edition de liens dynamique (1)

- ✓ Effectuer la liaison au moment du premier appel
 - Initialement utilisé dans MULTICS
- ✓ Sous Windows : DLL (Dynamic Link Library)
 - Elles contiennent
 - des procédures
 - des données
 - des procédures et des données
 - Elles sont utilisées pour le partage de bibliothèques et/ou de données entre différents processus
 - Elles existent aussi sous d'autres formes
 - .drv : Pilotes (Drivers)
 - .fon : Polices (fonts)

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 416

Université Nice Sophia Antipolis

Edition de liens dynamique (2)

- ✓ La construction d'une DLL est très proche de la conception d'un programme binaire
 - Un bit de signalisation indique que c'est une DLL
 - Les informations d'en-tête sont différentes
 - Ne peut démarrer ni s'exécuter d'elle-même
- ✓ Une DLL contient des procédures indépendantes de celle de la bibliothèque
 - Une pour la liaison, une pour le détachement
- ✓ Avantages des DLL
 - Economie de mémoire (RAM et Disque)
 - Facilité de mise à jour et/ou de corrections

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 417

Université Nice Sophia Antipolis

Edition de liens dynamique (3)

- ✓ Deux manières différentes de se lier à une DLL
 - Liaison implicite
 - Le programme est statiquement lié à un fichier spécial appelé bibliothèque d'importation
 - La bibliothèque d'importation fournit la clef pour accéder à une DLL
 - Charge la DLL à l'exécution du programme
 - Liaison explicite
 - Pas de bibliothèque d'importation
 - La DLL n'est pas chargée en temps que le programme mais pendant son exécution

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 418

Université Nice Sophia Antipolis

Références bibliographique (1)

- ✓ « Architecture de l'ordinateur », Andrew Tanenbaum chez Pearson Education (5ème édition)
- ✓ « Organisation et architecture de l'ordinateur », William Stallings chez Pearson Education (6ème édition)
- ✓ « Programmer en Assembleur sur PC », Holger Schakel chez Micro application
- ✓ « Compilateur », Dick Grune, Henri E. Bal chez Dunod
- ✓ « Architecture des Ordinateurs », Cecile Germain et Daniel Etienne

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 419

Université Nice Sophia Antipolis

Références bibliographique (2)

- ✓ « Une introduction au langage assembleur », Djamel Rebaïne
- ✓ « Architecture des ordinateurs », Emmanuel Viennet
- ✓ « Architecture », Michel Meynard
- ✓ « Introduction à la programmation en Assembleur », Pierre Jourlin
- ✓ « ASR 1 » et « ASR 2 », Joanna Moulrierac
- ✓ <http://www.commentcamarche.net/>

Janvier 2012 Gaëtan Rey - DUT Informatique de Nice 420