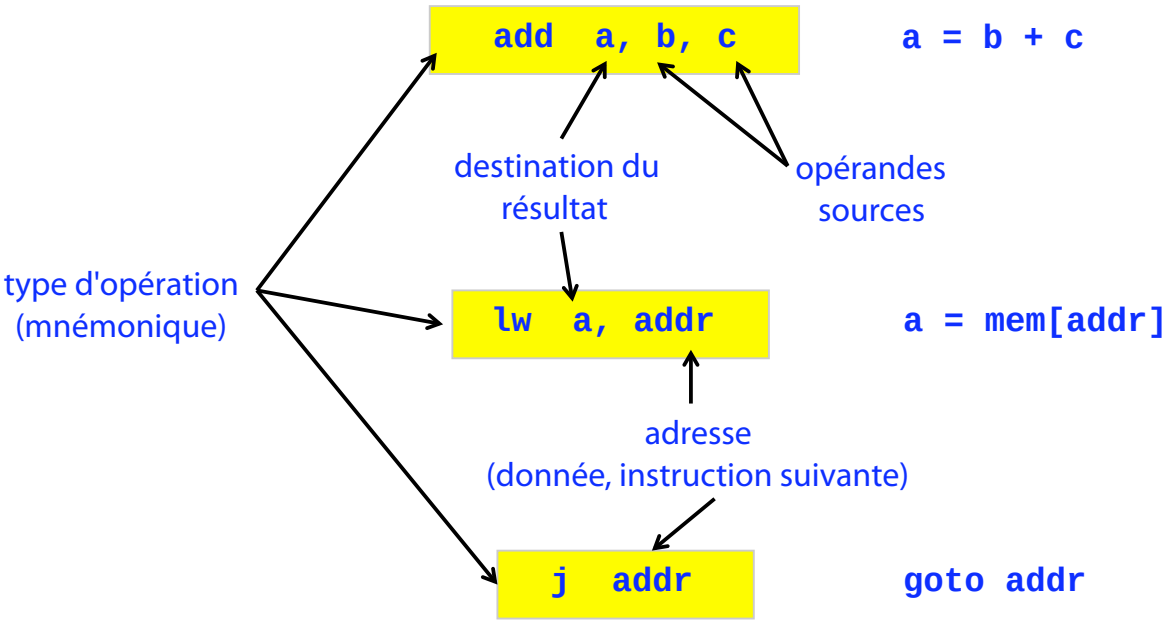
Exécution des instructions machine

Eduardo Sanchez
EPFL

Exemple: le processeur MIPS

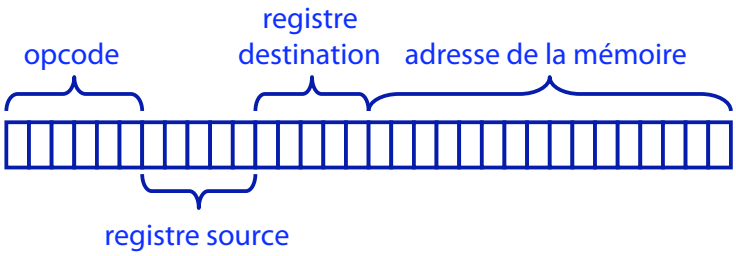


- Le processeur MIPS possède 32 registres
- Les opérations arithmétiques et logiques se font toujours entre registres
- Les seules instructions qui utilisent la mémoire comme opérande sont **`lw`** (load) et **`sw`** (store)
- La façon de donner l'adresse d'un opérande est appelée le mode d'adressage. Le processeur MIPS connaît cinq modes d'adressage:
  - registre
  - immédiat
  - direct ou absolu
  - indirect
  - déplacement ou relatif

- Exemples des modes d'adressage:
  - registre:
    ```
    add   $s0,$s1,$s2          s0=s1+s2
    ```
  - immédiat:
    ```
    add   $s0,$s1,123          s0=s1+123
    ```
  - direct ou absolu:
    ```
    add   $s0,$s1,(1234)       s0=s1+mem[1234]
    ```
  - indirect:
    ```
    add   $s0,$s1,($s2)        s0=s1+mem[s2]
    ```
  - déplacement ou relatif:
    ```
    add   $s0,$s1,123($s2)     s0=s1+mem[s2+123]
    ```

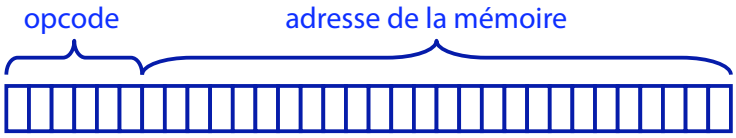
Format I: transfert de données et branchements

```
LW $t0,32($s3)
```



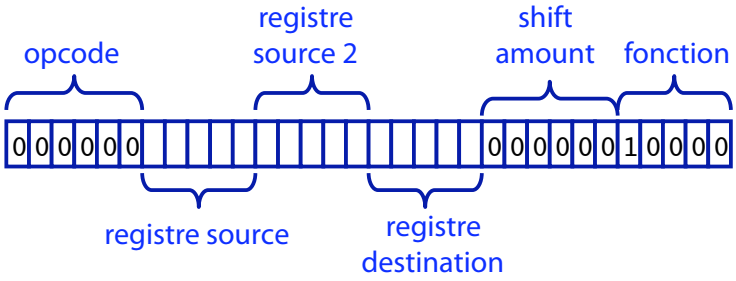
Format J: saut

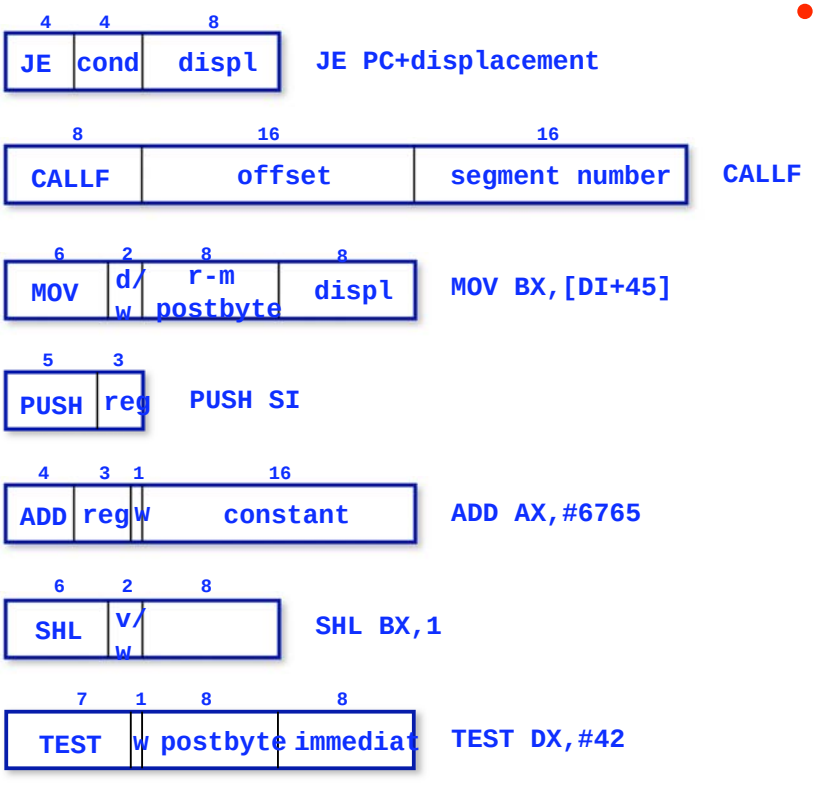
```
J addr
```



Format R: opérations arithmétiques

```
ADD $t1,$t2,$t0
```





- Le format de l'architecture x86 est complètement différent:
  - codage des instructions: 1 à 17 bytes
  - l'opcode peut contenir un bit qui indique si l'opérande a 8 ou 32 bits (bit w)
  - le bit d indique la direction du transfert: registre ⇔ memoire
  - certains opcodes contiennent le type d'adressage et le registre utilisé
  - quelques instructions utilisent un *post-byte* qui indique le type d'adressage

Repertoire d'instructions

- Instructions arithmétiques et logiques

Instruction	Example	Meaning
add	`add $s1, $s2, $s3`	$s1 = $s2 + $s3
add w/o overflow	`addu $s1, $s2, $s3`	$s1 = $s2 + $s3
add immediate	`addi $s1, $s2, 10`	$s1 = $s2 + 10
add imm. w/o overflow	`addiu $s1, $s2, 10`	$s1 = $s2 + 10
AND	`and $s1, $s2, $s3`	$s1 = $s2 ∧ $s3
AND immediate	`andi $s1, $s2, 0xFD`	$s1 = $s2 ∧ 0xFD
NOR	`nor $s1, $s2, $s3`	$s1 = ¬ ($s2 ∨ $s3)
NOT*	`not $s1, $s2`	$s1 = ¬ $s2
OR	`or $s1, $s2, $s3`	$s1 = $s2 ∨ $s3
OR immediate	`ori $s1, $s2, 0xFD`	$s1 = $s2 ∧ 0xFD
shift left logical	`sll $s1, $s2, 4`	$s1 = $s2 << 4
shift left logical var.	`sllv $s1, $s2, $s3`	$s1 = $s2 << $s3
shift right logical	`srl $s1, $s2, 4`	$s1 = $s2 >> 4
shift right logical var.	`srlv $s1, $s2, $s3`	$s1 = $s2 >> $s3
subtract	`sub $s1, $s2, $s3`	$s1 = $s2 – $s3
subtract w/o overflow	`subu $s1, $s2, $s3`	$s1 = $s2 – $s3
subtract immediate*	`subi $s1, $s2, 10`	$s1 = $s2 – 10
exclusive OR	`xor $s1, $s2, $s3`	$s1 = $s2 ⊕ $s3
exclusive OR imm.	`xori $s1, $s2, 0xFD`	$s1 = $s2 ⊕ 0xFD

- Instructions de comparaison

Instruction	Example	Meaning
set on less than	`slt $s1, $s2, $s3`	if ($s2 < $s3) $s1 = 1; else $s1 = 0
set on less than unsgn.	`sltu $s1, $s2, $s3`	if ($s2 < $s3) $s1 = 1; else $s1 = 0
set on less than imm.	`slti $s1, $s2, 10`	if ($s2 < 10) $s1 = 1; else $s1 = 0
slt immediate unsgn.	`sltiu $s1, $s2, 10`	if ($s2 < 10) $s1 = 1; else $s1 = 0

- Instructions de saut conditionnel

Instruction	Example	Meaning
branch on equal	`beq $s1, $s2, L`	if ($s1 == $s2) go to L
branch on not equal	`bne $s1, $s2, L`	if ($s1 != $s2) go to L
branch on greater than equal zero	`bgez $s1, L`	if ($s1 $\geq$ 0) go to L
branch on greater than zero	`bgtz $s1, L`	if ($s1 > 0) go to L
branch on less than equal zero	`blez $s1, L`	if ($s1 $\leq$ 0) go to L
br. on less than zero	`bltz $s1, L`	if ($s1 < 0) go to L
branch on greater than equal zero and link	`bgezal $s1, L`	if ($s1 $\geq$ 0) $ra = PC + 4; go to L
branch on less than zero and link	`bltzal $s1, L`	if ($s1 < 0) $ra = PC + 4; go to L

- Instructions de saut inconditionnel

Instruction	Example	Meaning
jump	`j 0x2500`	go to addr 10,000
jump and link	`jal 0x2500`	$ra = PC + 4; go to addr 10,000
jump and link register	`jalr $s1, $s2`	$s2 = PC + 4; go to $s1
jump register	`jr $ra`	go to $ra

- Instructions d'accès à la mémoire

Instruction	Example	Meaning
load byte (sign-extended)	`lb $s1, 96($s2)`	$s1 = Memory[$s2 + 96]
load unsigned byte	`lbu $s1, 96($s2)`	$s1 = Memory[$s2 + 96]
load halfword (sign-extended)	`lh $s1, 96($s2)`	$s1 = Memory[$s2 + 96]
load unsigned halfword	`lhu $s1, 96($s2)`	$s1 = Memory[$s2 + 96]
load word	`lw $s1, 96($s2)`	$s1 = Memory[$s2 + 96]
store byte	`sb $s1, 96($s2)`	Memory[$s2 + 96] = $s1
store halfword	`sh $s1, 96($s2)`	Memory[$s2 + 96] = $s1
store word	`sw $s1, 96($s2)`	Memory[$s2 + 96] = $s1

Registres

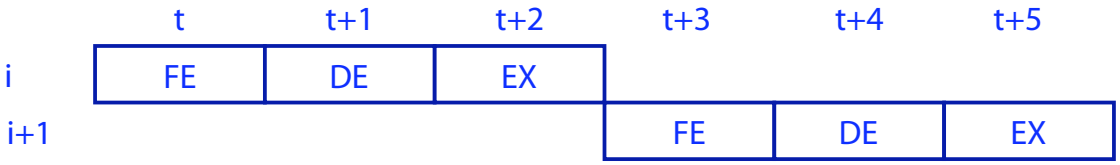
Name	Register number	Usage
$zero	0	the constant value 0
$at	1	reserved for the assembler
$v0–$v1	2–3	values for results and expression evaluation
$a0–$a3	4–7	arguments
$t0–$t7	8–15	temporaries
$s0–$s7	16–23	saved
$t8–$t9	24–25	more temporaries
$k0–$k1	26–27	reserved for the operating system
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address

Exécution temporelle des instructions

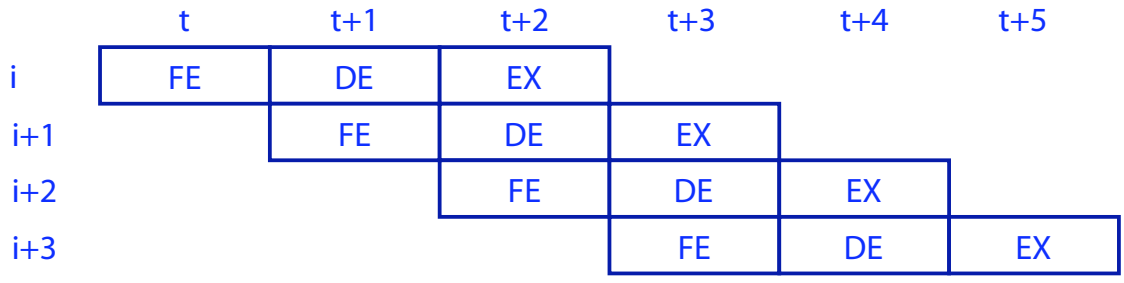
- Toute instruction d'un processeur est exécutée en trois phases:
  - *fetch*: lecture de l'instruction en mémoire, à l'adresse donnée par le PC. L'instruction est stockée dans le registre IR
  - *decode*: identification de l'instruction
  - *execute*: exécution des opérations nécessaires à l'instruction
- En général, l'exécution de chaque phase de l'instruction demande au moins un cycle d'horloge. La phase execute peut demander plus d'un cycle, en fonction de la complexité de l'instruction: si une instruction de déplacement peut se faire en un seul cycle, une instruction de division a besoin de plusieurs dizaines de cycles

- Les vitesses d'horloge des processeurs actuels vont de plusieurs mégahertz (MHz) à plusieurs gigahertz (GHz) pour les plus récents
- Il n'y a pas beaucoup de sens à comparer les performances de deux processeurs en comparant seulement les vitesses d'horloge. En effet, pour des familles différentes, la quantité de travail effectuée en un cycle peut être très différente. Il est plus réaliste de mesurer la performance en comparant les temps d'exécution pour un même programme. Les programmes utilisés pour effectuer ces mesures sont appelés benchmarks
- L'ensemble de benchmarks le plus utilisé commercialement est le SPEC (*Standard Performance Evaluation Corporation*)

- Une augmentation de performance sans augmentation de la vitesse d'horloge est obtenue grâce au pipeline
- Dans un processeur sans pipeline, l'exécution d'une instruction ne commence pas avant la fin d'exécution de l'instruction précédente



- Dans un processeur à pipeline, à la fin de chaque phase démarre l'exécution d'une nouvelle instruction



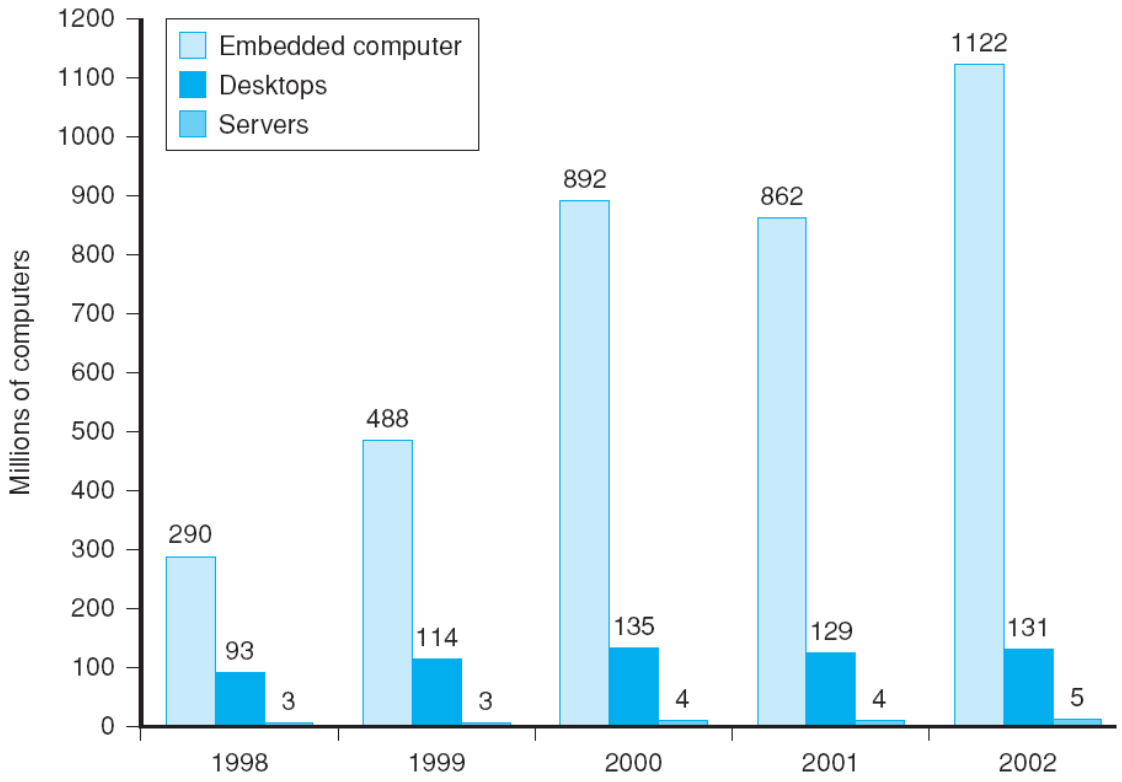
- Le pipeline peut être vu comme un premier pas vers le parallélisme. Toutefois, le vrai parallélisme est atteint seulement avec plus d'une unité de traitement
- Les processeurs superscalaires peuvent chercher plusieurs instructions à la fois, exécutées en parallèle par plusieurs unités de traitement
- Dans les systèmes multiprocesseurs, plusieurs séquences d'instructions sont exécutées en parallèle sur différents ensembles de données: c'est les architectures MIMD (*multiple-instruction stream, multiple-data stream*) appelées ainsi par opposition aux traditionnelles SISD (*single-instruction stream, single-data stream*)
- Un autre type de parallélisme est obtenu avec l'architecture SIMD (*single-instruction stream, multiple-data stream*), où plusieurs unités de traitement exécutent en parallèle la même séquence d'instructions, mais chacune sur son propre ensemble de données

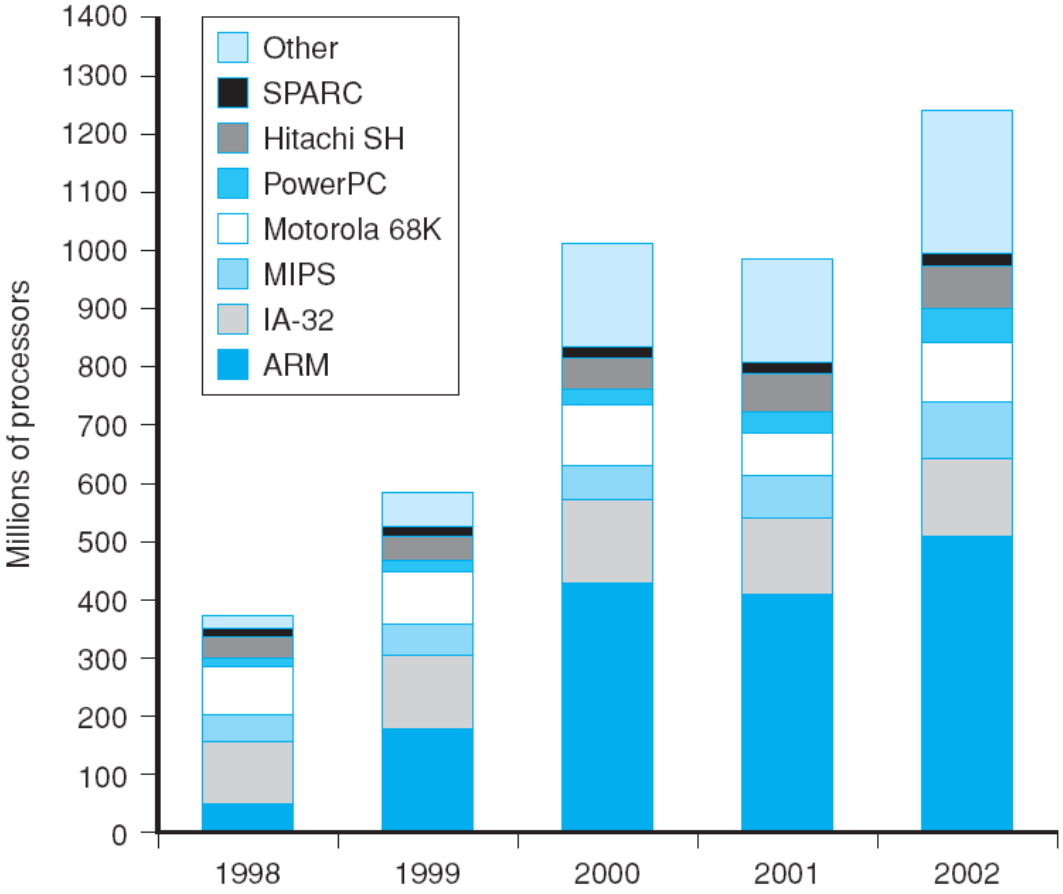
Processeurs actuels (10.08)

	IBM Power6	Sun UltraSPARC T2+	Fujitsu SPARC64 VII
Architecture	2x2x64-bit	8x8x64-bit	4x2x64-bit
Fréquence (GHz)	5	1.4	2.52
Superscalaire	7	16	4
Pipeline	13	8int/12fp	15
Désordre (instr)	limited	0	64
Technologie (nm-metal)	65-10	65	65-11
Taille (mm^2)	341	342	400
Transistors (M)	790	503	600
Consommation (W)	100	95	135
Cache (L1-L2-L3 I/D)	2x64K/64K-2x4M-32M	8x8K/16K-4M-NA	4x64K/64K-6M-NA
SPECint/fp2006[cores]	15.8/20.1[1]	NA	10.5/25[64]
SPECint/fp2006rate[cores]	1837/1822[64]	142/111[16]	2088/1861[256]

	Intel 6-core Xeon X7460	AMD 4-core Opteron 8360SE	Itanium 9150M
Architecture	6x1x32/64-bit	4x1x32/64-bit	2x2x64-bit
Fréquence (GHz)	2.67	2.5	1.67
Superscalaire	1 complex + 3 simple	3	6
Pipeline	14	12int/17fp	8
Désordre (instr)	96	72	0
Technologie (nm-metal)	45	65-11	90-7
Taille (mm^2)	503	283	596
Transistors (M)	1900	463	1720
Consommation (W)	130	105	104
Cache (L1-L2-L3 I/D)	6x32K/32K-3x3M-16M	4x64K/64K-4x512K-2M	2x16K/16K-1M/256K-12M
SPECint/fp2006[cores]	22.0/22.3[24]	14.4/18.5[8]	NA
SPECint/fp2006rate[cores]	274/142[24]	170/156[16]	2893/NA[256]

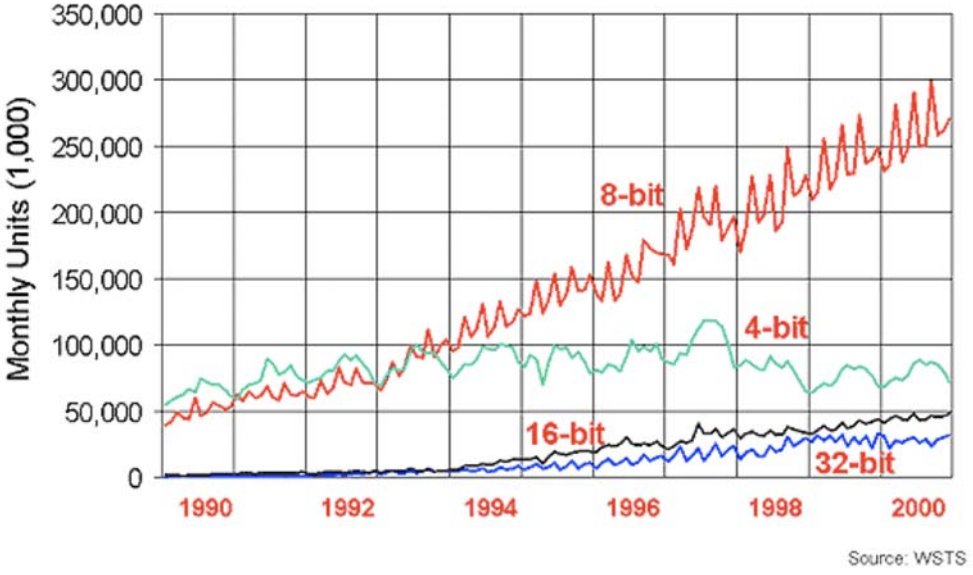
Systèmes embarqués





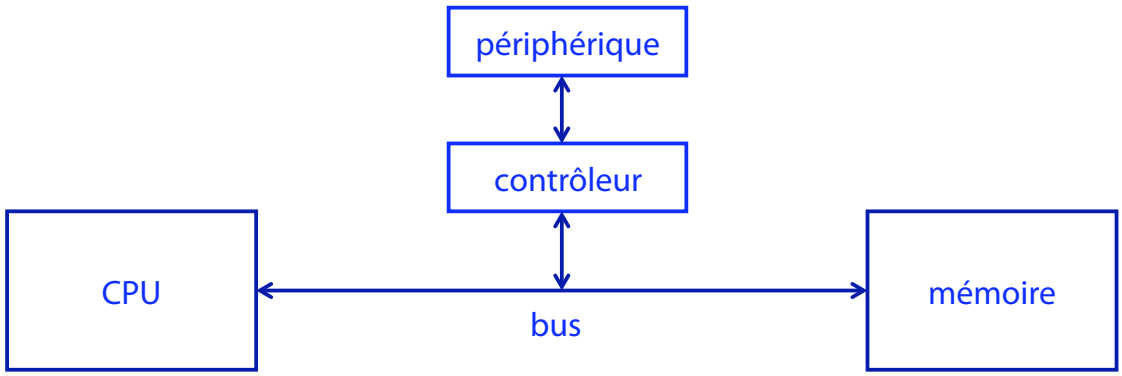
Microprocessor Unit Sales

All types, all markets worldwide



Communication

- La communication entre le processeur et les dispositifs d'entrée/ sortie se fait normalement par le biais d'un dispositif appelé un contrôleur
- Le contrôleur réalise la conversion des messages entre le processeur et le périphérique, afin de respecter les formats propres à chaque dispositif



- La communication entre le processeur et le contrôleur se fait de la même manière qu'avec la mémoire: une instruction similaire au STORE envoie une chaîne de bits au contrôleur et une instruction similaire au LOAD envoie une chaîne de bits au processeur
- Chaque contrôleur possède un ensemble d'adresses réservées, appelées ports
- Certains contrôleurs peuvent accéder directement à la mémoire, sans passer par le processeur: ce type d'accès est appelé DMA (*Direct Memory Access*)
- Par exemple, pour lire un secteur d'un disque, le processeur peut envoyer une demande au contrôleur, qui se charge de la tâche pendant que le processeur exécute d'autres opérations

- La communication avec un périphérique est en général à double sens. Un processeur, par exemple, produit des données beaucoup plus rapidement qu'une imprimante n'est capable de les traiter: afin de ne pas perdre des données, il est donc important pour le processeur de connaître l'état de l'imprimante avant chaque envoi
- Le dialogue entre le périphérique et le processeur, via le contrôleur, utilise un mot de status, généré par le périphérique et stocké dans le contrôleur

- La vitesse de transmission de l'information est mesurée en bits par seconde (bps)
- Il y a deux grands types de trajectoires de communication: la parallèle et la sérielle.
- Dans la communication parallèle, plusieurs bits sont transférés en même temps, chacun sur une ligne séparée
- Dans la communication sérielle, on transfère un seul bit à la fois. Cette technique est plus lente, mais demande une implémentation plus simple (donc, moins chère)
- Un exemple bien connu de communication sérielle est l'utilisation des modems (modulateur-démodulateur) pour transférer de l'information via des lignes téléphoniques. Les chaînes de bits sont d'abord transformées en tonalités pour être transférées par la ligne téléphonique et, ensuite, à leur destination, ces tonalités sont de nouveau transformées en chaînes de bits