

# Real-Time Operating System avec des Microcontrôleurs

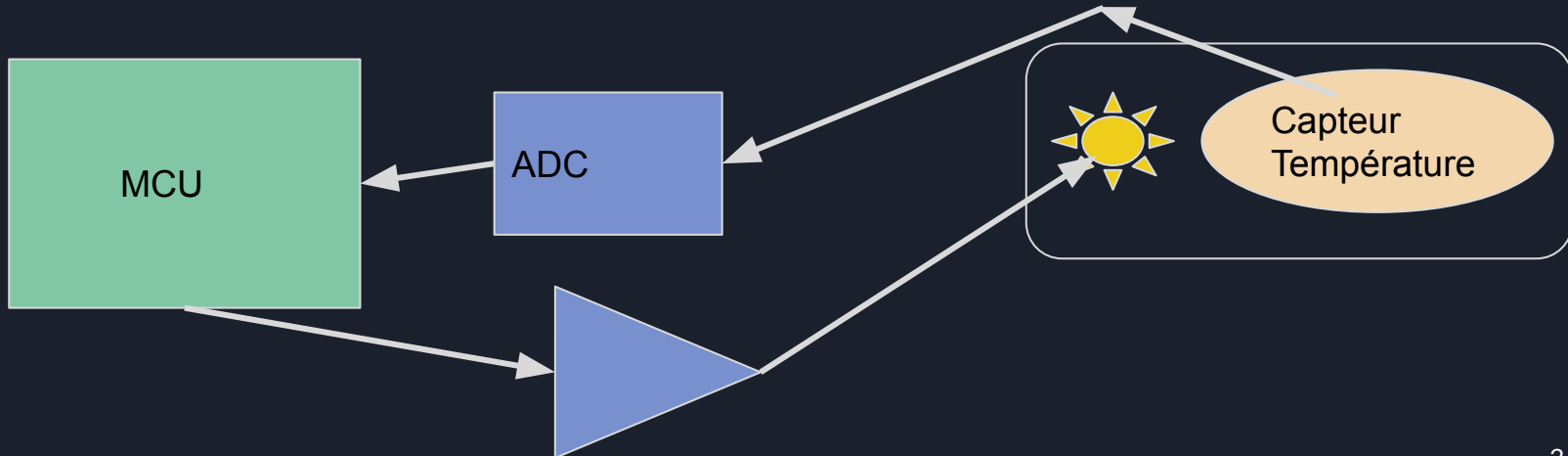


# Plan

- Les systèmes embarqués temps réel
- Comprendre les RTOS tasks
- Les microprocesseurs et microcontrôleurs
- Les présupposés sur l'ordinateur
- MicroController architecture et la notion d'assembleur
- Le Langage Assembleur
- General Purpose IO et I/O avec un Seven Led Segment
- Utilisation avancée du compilateur
- Pointeur avancé
- Les instructions de contrôles avancée
- La mémoire
- Les fonctions avancées
- Les structures
- Les fonctions avec un nombre d'argument variables
- Exploration de la bibliothèque des fonctions
- Timer programming ?

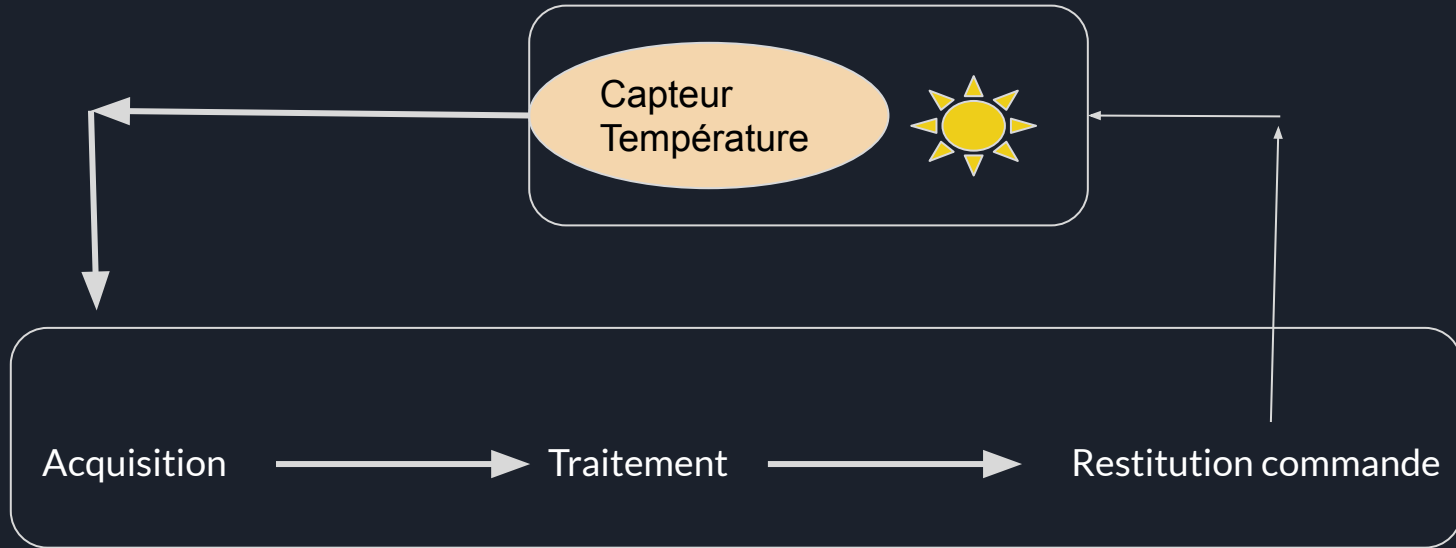
# Les systèmes embarqués temps-réel

- La notion de système embarqué temps-réel : système informatique et électronique autonome avec une tâche spécifique.  
Quand ne pas délivrer la tâche dans un temps donnée est considéré comme une erreur du système.



# Les systèmes embarqués temps-réel

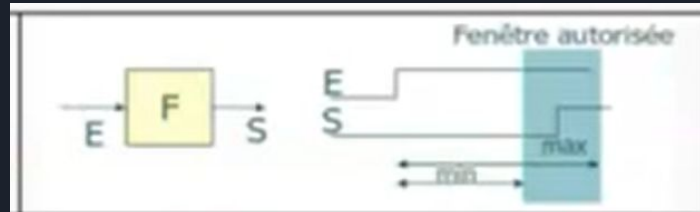
- exactitude logique
- exactitude temporelle



# Les systèmes embarqués temps-réel


- Exemples de contraintes de temps :

Echéance : Terminaison au plus tôt ou plus tard



Cohérence : instants de production des résultats





# Les systèmes embarqués temps-réel : Exemple

- 2 % processeurs pour les PC
- Domaine d'application :
  - Contrôle de processus industriels
  - Transport
    - Avionique
    - Trains, Automobile (ABS...)
    - Contrôle de navigation
  - Télécommunication
    - Satellites
    - GPS
    - Téléphone Mobile



# Les systèmes embarqués temps-réel

- Domaine d'application :

Distributeur billets :

- information touches / écrans tactiles + connections banque
- décision restitution des billets
- compte nombre billets restants et informe la banque

Une carte à puce:

- reçoit information lecteur cartes
- décision validation du code
- compte nombre tentative et blocage possible



# Les systèmes embarqués temps-réel

Système embarqué :

- réagit stimuli extérieurs
- prends des décisions -> stimuli + état interne

Classification des systèmes temps réel

- Réactifs :
  - interactions permanentes avec l'environnement
  - En réponse aux stimuli le système provoque des réactions
  - Le système ne travaille que lors de l'élaboration des réactions
  - les instants de production des résultats sont contraints par la dynamique du procédé
- Transformationnels
  - Faible couplage avec l'environnement.
  - Les données entrées sont prises à l'initiative du système.
  - Les résultats sont engendrés à l'initiative du système.
  - Les traitements internes sont généralement importants (traitements algorithmiques).
- Interactifs
  - les stimuli provoquent des réactions
  - leur prise en compte reste à l'initiative du système





# Les systèmes embarqués temps-réel : nature des Traitements

Nature des Traitements :

- Sporadiques
  - Arrêt d'urgence
  - Changement de mode de fonctionnement
  - Traitement par interruption
- Périodiques
  - Acquisition de capteurs (image, poids, température)
  - Calcul de position
  - Calcul de consigne



# Les systèmes embarqués temps-réel

Les types d'interactions avec l'environnement :

- les **événements** : transition d'état
- les **mesures** : données reçues de l'environnement extérieure
- les **commandes** : données émises par le système

Exigences des systèmes :

- Contraintes temporelles
- Parallélisme
- Prévisibilité
- Sûreté



# Les systèmes embarqués temps-réel

Les trois catégories des systèmes temps-réel :

- Hard real-time systems

Non respect d'une contrainte temporelle entraîne conséquences dramatiques ( Intégrité compromise)

- Firm real-time systems

Plusieurs non respect peuvent entraîner des conséquences dramatiques.

- Soft Real-time systems

Non-Respect des contraintes entraînent une dégradation du service

Exemples.



# Les systèmes embarqués temps-réel

Les différents types de systèmes temps réel :

- Hardware :
  - ASIC (circuit intégré spécifique à l'application)
  - PLD (dispositifs de logique programmable)
    - > CPLD Complex programmable logic devices
    - > FGPA Field-programmable gate arrays

Peu flexible, expertise, cost...

- MCU (MicroController Unit) : Flexibilité , Coût, Taille, Facilité d'utilisation...

Bare-metal firmware

RTOS based firmware

RTOS based software



# Les systèmes embarqués temps-réel

OS = operating system : système d'exploitation général qui est conçu pour gérer les ressources d'un ordinateur, telles que le processeur, la mémoire, les périphériques d'entrée et de sortie et les fichiers.

RTOS : système d'exploitation conçu pour gérer des tâches en temps réel, c'est-à-dire des tâches qui doivent être exécutées dans des délais stricts.

Environnement de programme. Write. Maintien.

Abstraction du matériel.

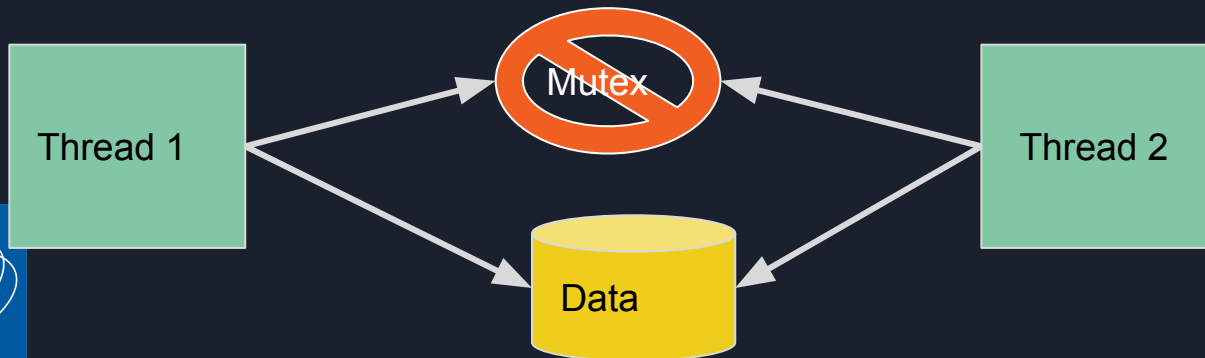
Donne accès à des objets qui accomplissent des comportements complexes.

Exemple : Threads And Mutexes

# Les systèmes embarqués temps-réel

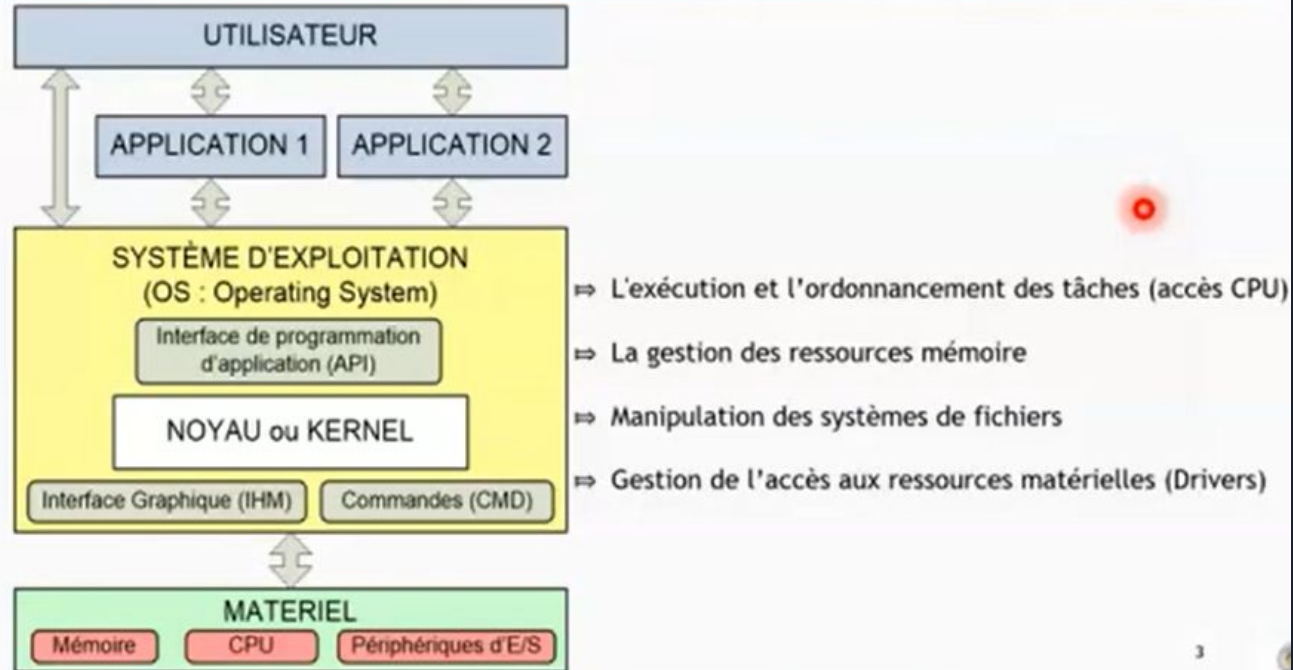
## Exemple : Threads And Mutexes

- moins complexes
- plus facile à comprendre
- meilleure intégration, portabilité :-> abstraction hardware.



# Les systèmes embarqués temps-réel

## Le Système d'exploitation





# Les systèmes embarqués temps-réel

Bilan :

- 1. Les systèmes temps-réel ont-ils besoin d'être rapides ?
- 2. Un RTOS est-il la meilleure solution pour les systèmes temps réel ?
- 3. RTOS based firmware est-il le seul moyen de satisfaire les échéances des systèmes temps-réel ?
- 4. Qu'est ce qu'un système temps réel ?
- 5. Nomme au moins 2 systèmes temps réel.
- 6. Quand est-il approprié d'utiliser un RTOS pour satisfaire les échéances des systèmes temps-réel ?





# Comprendre les RTOS tasks : Superloop vs RTOS

## Superloop :

- Le superloop est une méthode de programmation de microcontrôleurs qui permet de gérer plusieurs tâches sans utiliser de système d'exploitation en temps réel (RTOS).
- Dans un superloop, toutes les tâches sont exécutées dans une boucle principale qui les parcourt en séquence.
- Les tâches sont généralement écrites sous forme de fonctions, qui sont appelées dans la boucle principale.
- Les tâches peuvent être exécutées à intervalles réguliers ou en réponse à des événements externes.
- Le superloop est une méthode simple et efficace pour gérer plusieurs tâches sur des microcontrôleurs à faible coût et à faible puissance.



# Comprendre les RTOS tasks : Superloop vs RTOS

## RTOS Task :

- Un système d'exploitation en temps réel (RTOS) est un logiciel qui permet de gérer efficacement plusieurs tâches simultanément.
- Les tâches dans un RTOS sont exécutées de manière indépendante, avec leur propre pile et leur propre contexte d'exécution.
- Les tâches peuvent être priorisées et planifiées par le RTOS pour assurer un traitement rapide et précis des événements.
- Les tâches peuvent également communiquer entre elles en utilisant des mécanismes tels que les sémaphores, les files d'attente et les événements.
- Les RTOS sont couramment utilisés dans les systèmes embarqués et les applications en temps réel pour garantir une fiabilité et une précision maximales.



# Comprendre les RTOS tasks : Introduction

## Communication

### Queues:

- Les files d'attente sont des mécanismes de communication entre les tâches dans un système d'exploitation en temps réel.
- Elles sont utilisées pour échanger des données entre les tâches.
- Les tâches peuvent ajouter des éléments à la file d'attente ou en retirer en fonction de leur état d'exécution et de leur priorité.
- Les files d'attente peuvent être utilisées pour échanger des données entre les tâches dans un système d'exploitation en temps réel.
- Les files d'attente sont souvent utilisées pour transférer des données d'une tâche à une autre.



# Comprendre les RTOS tasks : Introduction

## Communication

### Événements:

- Les événements sont des signaux ou des notifications envoyés entre les tâches pour signaler la survenue d'un événement.
- Ils sont utilisés pour déclencher une action spécifique dans une ou plusieurs tâches.
- Les événements sont souvent utilisés pour coordonner les tâches dans un système d'exploitation en temps réel.
- Ils permettent aux tâches à faible priorité d'être informées des événements qui se produisent, afin qu'elles puissent effectuer une action spécifique en réponse.
- Les événements peuvent être utilisés pour synchroniser les tâches et pour déclencher des actions en temps réel.



# Comprendre les RTOS tasks : Introduction

## Communication

### Semaphores:

- Les sémaphores sont des objets de synchronisation utilisés dans les systèmes d'exploitation en temps réel pour coordonner l'accès à des ressources partagées.
- Ils sont utilisés pour gérer l'accès à une ressource partagée par plusieurs tâches.
- Les tâches doivent acquérir le sémaphore associé à la ressource avant d'y accéder.
- Si le compteur du sémaphore est déjà à zéro, la tâche doit attendre jusqu'à ce que le sémaphore soit à nouveau disponible.
- Lorsqu'une tâche a fini d'utiliser la ressource partagée, elle doit libérer le sémaphore pour permettre à d'autres tâches d'y accéder.



# Comprendre les RTOS tasks : les tâches

- Architecture mono-tâche :

Suffit de répéter indéfiniment la suite des tâches :

- Attendre stimulus
- Agir

- Architecture multi-tâches :

Exécution pseudo-parallèle de plusieurs tâches .

Difficultés accès au processeurs, mémoire périphériques.

Besoin d'ordonnancement

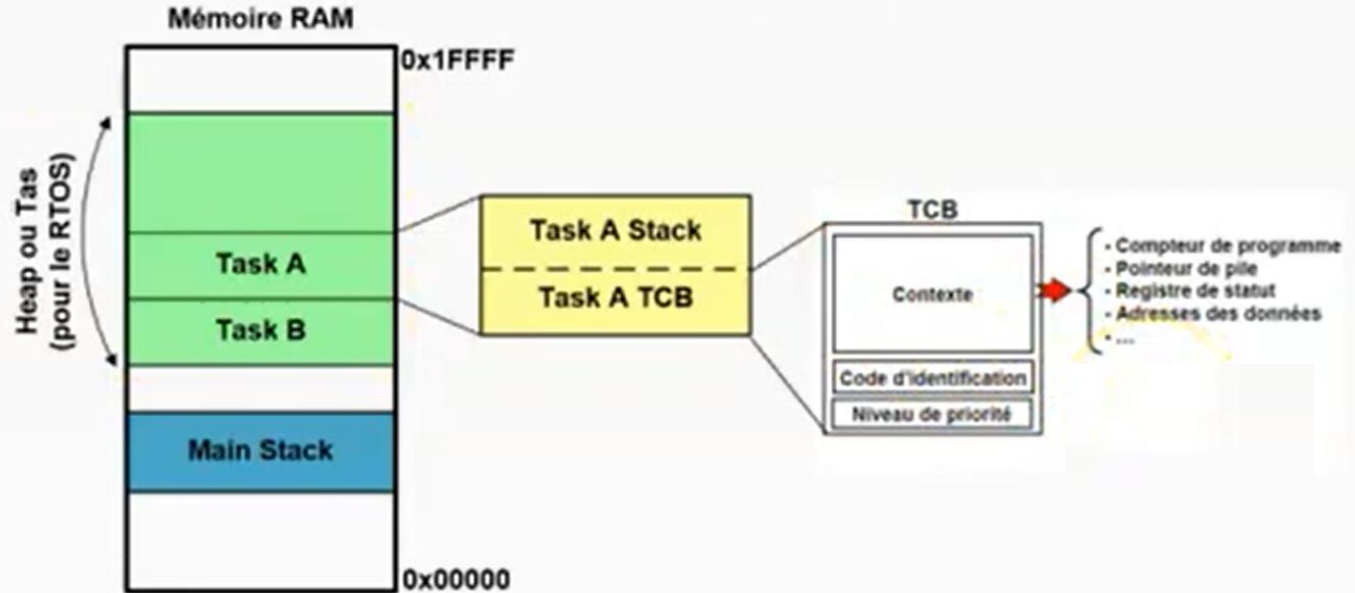
Qu'est qu'une tâche ?

Fonction C avec une boucle infinie.

TCB Task Control Block :

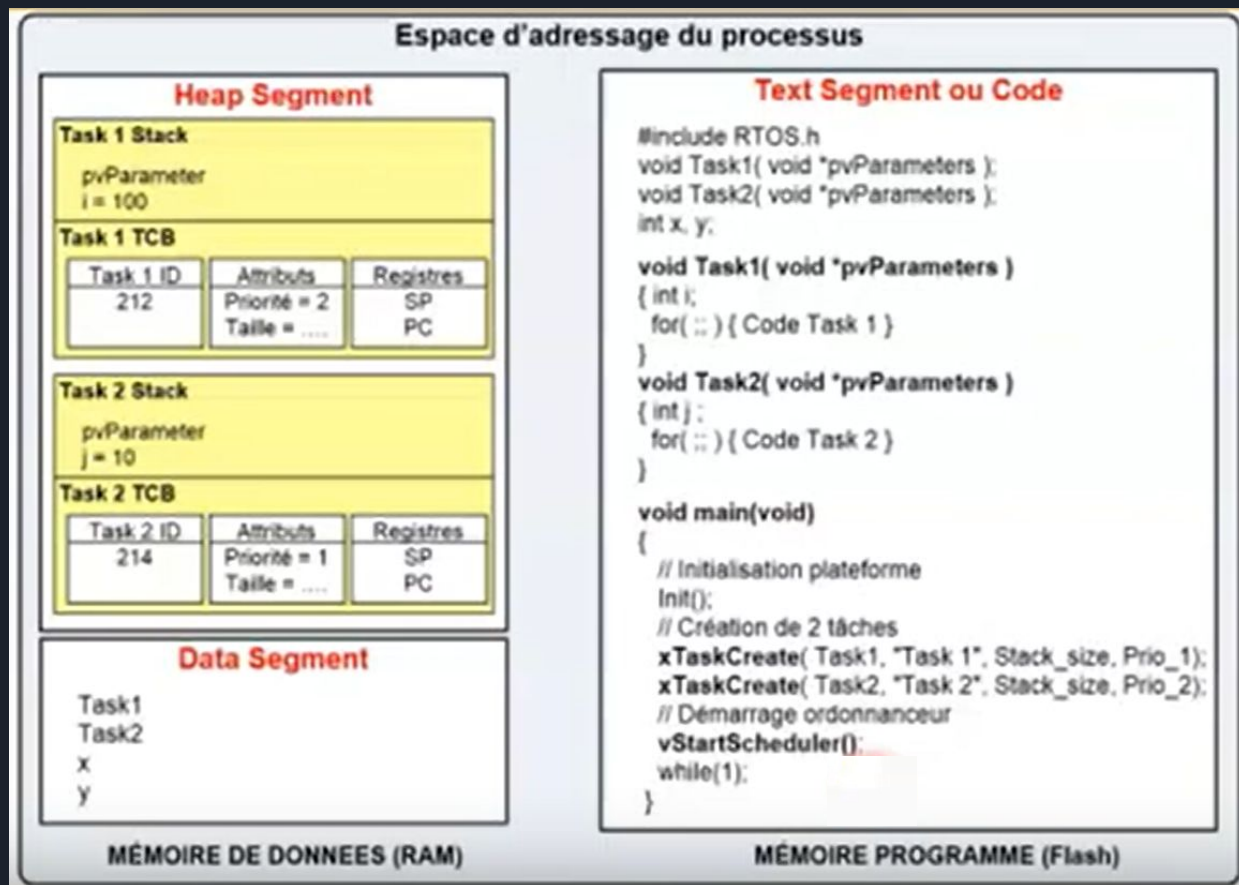
- identifiant,
- priorité,
- un état (prête, bloqué..
- un contexte (compteur qui pointe vers le code, pointeur de pile SP)

# Comprendre les RTOS tasks : les tâches





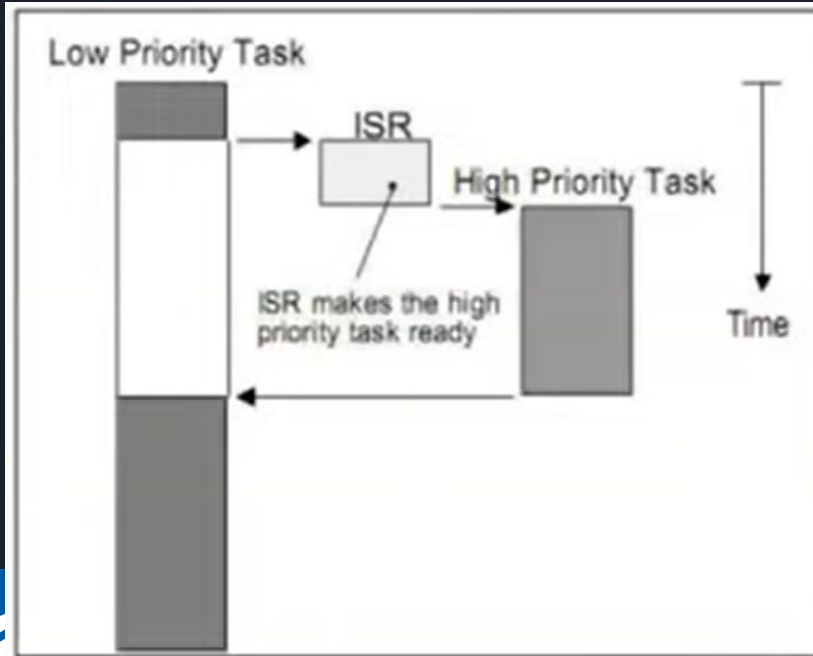
# Comprendre les RTOS tasks : les tâches et la mémoire avec un OS



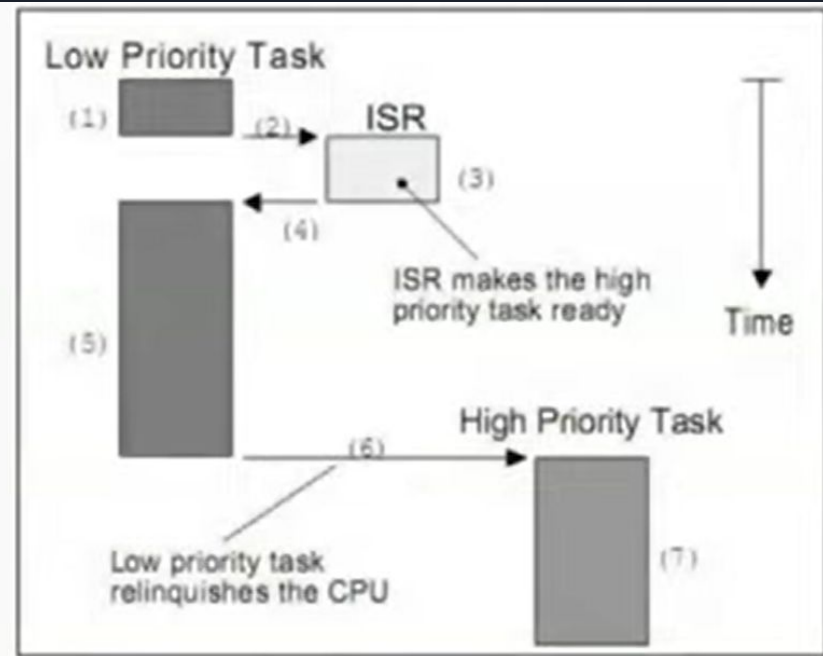


# Comprendre les RTOS tasks : les types de noyau

Préemptif



Non Préemptif ( Collaboratif )





# Comprendre les RTOS tasks : les types de noyau



Quantum fixe pour chaque tâche.  
Temps tâche  $\leq$  quantum

Caractéristique :

- Long processus n'entraîne pas de retard.
- Pseudo parallèle.
- Choix du quantum important.



# Comprendre les RTOS tasks : les types de tâches

Tâches périodiques : définies par la période d'exécution : Observer un capteur à intervalles réguliers.

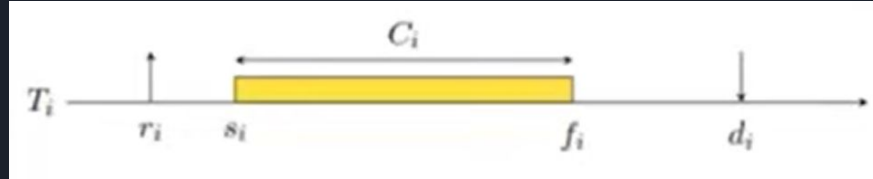
Tâches apériodiques : tâches exécutées à intervalle irréguliers.

Des tâches indépendantes : tâches dont l'ordre d'exécution peut être quelconque.

Des tâches dépendantes : tâches avec des contraintes de précedence.



# Comprendre les RTOS tasks : les caractéristique d'une tâche



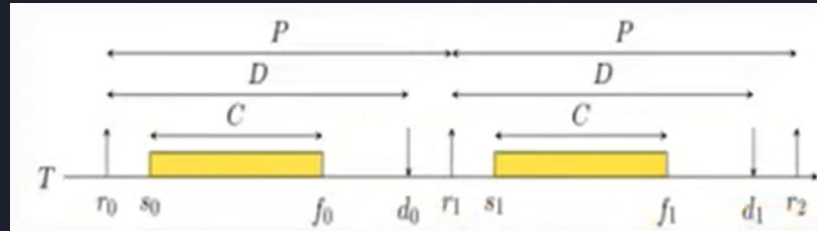
<b>r</b>	date de réveil
<b>C</b>	durée d'exécution (temps processeur)
<b>D</b>	délai critique
<b>d</b>	date de fin pour un r donnée $\rightarrow d = r + D$
<b>s</b>	date début exécution
<b>f</b>	date fin exécution
<b>tr = f - r</b>	temps de réponse. Rmq : $D - tr \geq 0 \Rightarrow$ échéance respectée
<b>L = D - C</b>	laxité nominal (retard max)



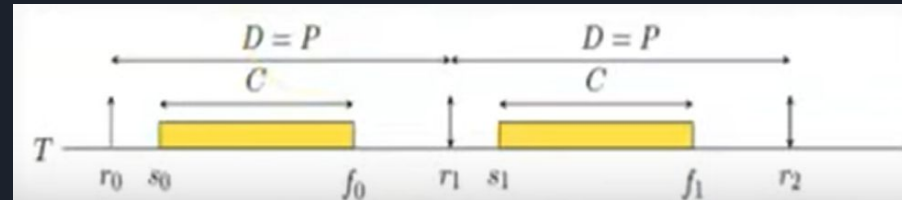
# Comprendre les RTOS tasks : les types de tâches 2

Tâches apériodiques : tuple  $T(r, C, D)$

Tâches périodiques : quadruplet  $T(r_0, C, D, P)$



Des tâches à échéance sur requête





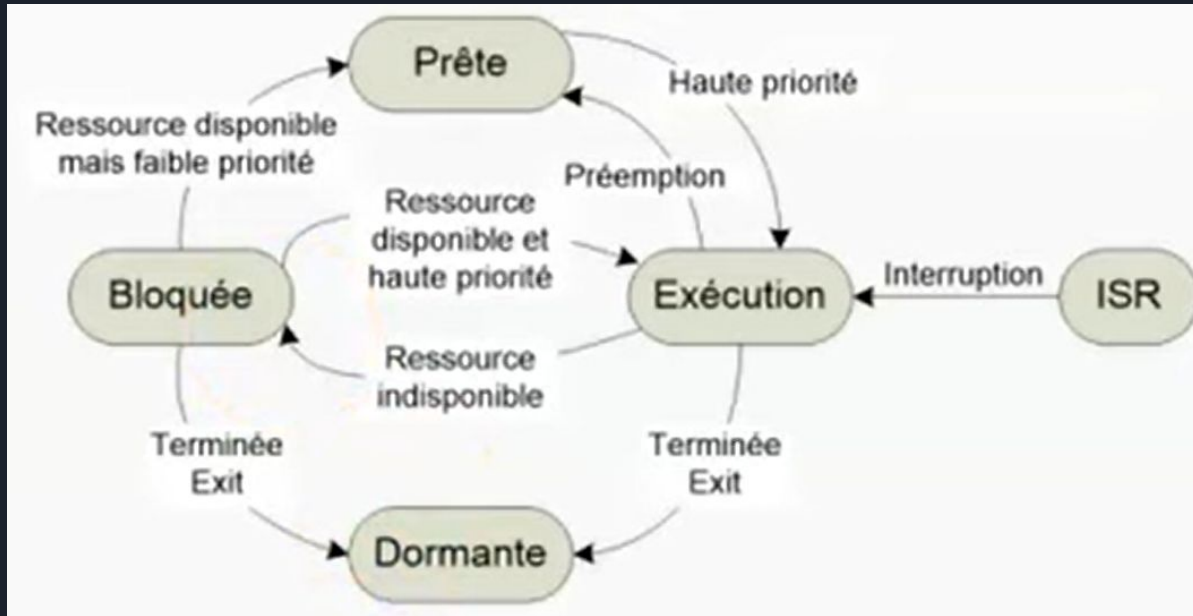
# Comprendre les RTOS tasks : les priorités des tâches

Plus une tâche est importante et critique, plus sa valeur est élevée.

Priorité des tâches peuvent être :

- Statiques :  
Priorités fixes.
- Dynamiques :  
Priorités non-fixes

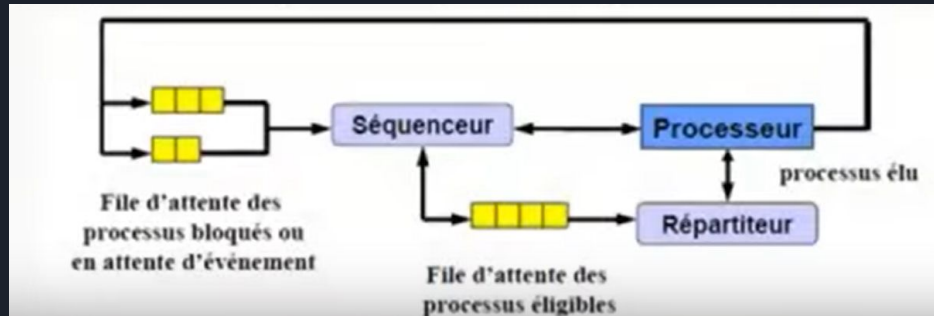
# Comprendre les RTOS tasks : les états des tâches.



# Comprendre les RTOS tasks : le gestionnaire des tâches (Scheduler)

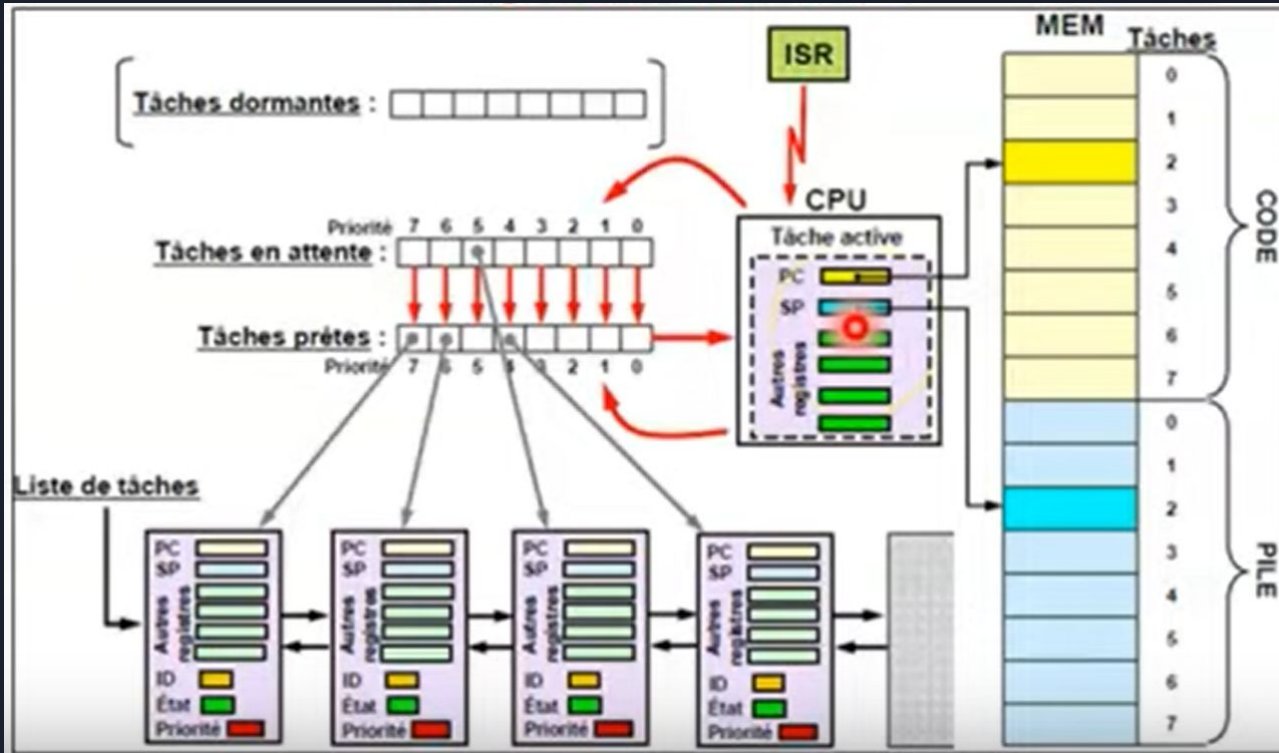
Noyau utilisent 2 entités :

- Ordonnanceur (scheduler) . Choisit l'ordre d'exécution des tâches selon une politique d'ordonnancement grâce aux informations TCB
- Répartiteur (dispatcher). Effectue le changement de contexte :
  - Conserver état des registres quand une tâche est suspendue
  - Sauvegarder le contexte de la pile
  - Restauration du contexte depuis le TCB





# Comprendre les RTOS tasks : le gestionnaire des tâches (Scheduler)



# Comprendre les RTOS tasks : les algorithmes d'ordonnancement.

Hypothèses :

- Respect des échéances des tâches.
- Dépend des caractéristique :
  - indépendantes ou dépendantes - Périodiques ou apériodiques
  - partageant ou non des ressources - Échangeant ou non des informations.

pour tâche périodique

Ti	Tâche périodique
Ci	temps exécution
Pi	Période
Di	échéance relative à la tâche Ti
H1	instances d'une tâche périodiques
H2	instances avec $C = D$
H3	Tâches indépendantes.

Facteur d'occupation : fraction temps processeur pour l'exécution d'une tâche  $U_i = C_i / P_i$



# Comprendre les RTOS tasks : les algorithmes d'ordonnancement.

- 1<sup>er</sup> algorithme : Algorithm Rate Monotonic Assignment (RMA) :  
Condition : priorités tâches fixes

Règle : Plus la période est courte plus la priorité est haute.

Caractéristiques = préemptif.

Condition Suffisante :

$$\sum_{i=1}^n \frac{C_i}{P_i} < n \left( 2^{1/n} - 1 \right)$$

$n=2 \Rightarrow 82,8\%$

$n=3 \Rightarrow 78\%$

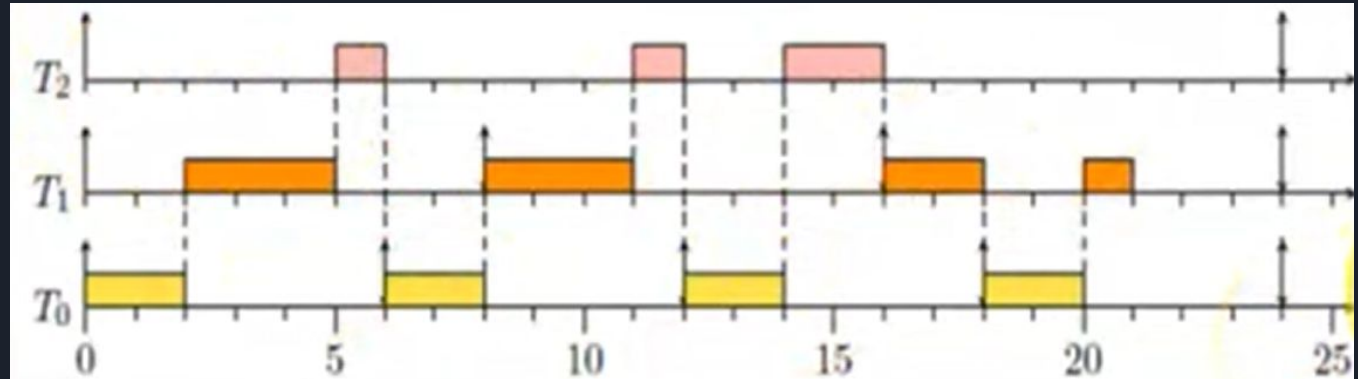
$n = \text{infini} \Rightarrow 70\%$

# Comprendre les RTOS tasks : les algorithmes d'ordonnancement.

Exemple :

$$\sum_{i=1}^n \frac{C_i}{P_i} = 87,5\%$$

Tâche	Coût	Période
$T_0$	2	6
$T_1$	3	8
$T_2$	4	24





# Comprendre les RTOS tasks : les algorithmes d'ordonnancement.

Exercice:

$$\sum_{i=1}^n \frac{C_i}{P_i} = \%$$

Tâche	Coût	Période
$T_0$	3	20
$T_1$	2	5
$T_2$	2	10

Périodicité de  $T_0$ ,  $T_1$ ,  $T_2$

# Comprendre les RTOS tasks : les algorithmes d'ordonnancement.

Exercice:

$$\sum_{i=1}^n \frac{C_i}{P_i} = 75\%$$

Tâche	Coût	Période
$T_0$	3	20
$T_1$	2	5
$T_2$	2	10

Périodicité de  $T_0$ ,  $T_1$ ,  $T_2$

$$\begin{aligned} T_0 &= 9 \\ T_1 &= 2 \\ T_2 &= 4 \end{aligned}$$



# Comprendre les RTOS tasks : les algorithmes d'ordonnancement : 2

- 2ème algorithme : Algorithm Deadline Monotonic priority Assignment (DMA) :  
Condition : priorités tâches fixes

Règle : Plus l'échéance est courte plus la priorité est haute.

Caractéristiques = préemptif, mais optimal pour  $D_i < P_i$

Condition Suffisante :

$$\sum_{i=1}^n \frac{C_i}{D_i} < n \left( 2^{1/n} - 1 \right)$$

$n=2 \Rightarrow 82,8\%$

$n=3 \Rightarrow 78\%$

$n = \text{infini} \Rightarrow 70\%$

# Comprendre les RTOS tasks : les algorithmes d'ordonnancement : 2

Exemple :

$$\sum_{i=1}^n \frac{C_i}{P_i} = 87,5\%$$

$$\sum_{i=1}^n \frac{C_i}{D_i} = 135\%$$

Tâche	Coût	Période	Echéance
$T_0$	2	6	5
$T_1$	3	8	4
$T_2$	4	24	20

Périodicité de  $T_0$ ,  $T_1$ ,  $T_2$



# Comprendre les RTOS tasks : les algorithmes d'ordonnancement : 2

Exemple :

$$\sum_{i=1}^n \frac{Ci}{Pi} = \%$$

$$\sum_{i=1}^n \frac{Ci}{Di} = \%$$

Tâche	Coût	Période	Echéance
$T_0$	3	20	7
$T_1$	2	5	4
$T_2$	2	10	9

Périodicité de  $T_0$ ,  $T_1$ ,  $T_2$

# Comprendre les RTOS tasks : les algorithmes d'ordonnancement : 2

Exemple :

$$\sum_{i=1}^n \frac{Ci}{Pi} = 75\%$$

$$\sum_{i=1}^n \frac{Ci}{Di} = 115\%$$

Tâche	Coût	Période	Echéance
$T_0$	3	20	7
$T_1$	2	5	4
$T_2$	2	10	9

Périodicité de  $T_0$ ,  $T_1$ ,  $T_2$

$$\begin{aligned} T_0 &= 5 \\ T_1 &= 2 \\ T_2 &= 9 \end{aligned}$$



# Comprendre les RTOS tasks : les algorithmes d'ordonnancement : 3

- 3ème algorithme : Earliest Deadline First (EDF) :  
Condition : priorités tâches non fixes

Règle : A chaque fois qu'une tâche est réveillée, l'ordonnanceur réévalue les tâches prêtes et prend l'échéance la plus courte.

Caractéristiques = préemptif, mais optimal pour les tâches périodiques et apériodiques  
Condition Suffisante :

$$\sum_{i=1}^n \frac{C_i}{D_i} \leq 1$$

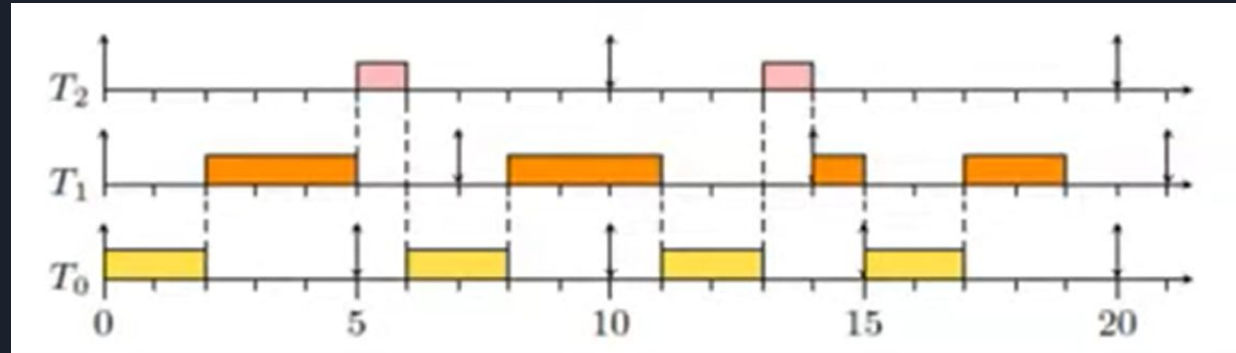
# Comprendre les RTOS tasks : les algorithmes d'ordonnancement : 3

Exemple :

$$\sum_{i=1}^n \frac{C_i}{P_i} = \%$$

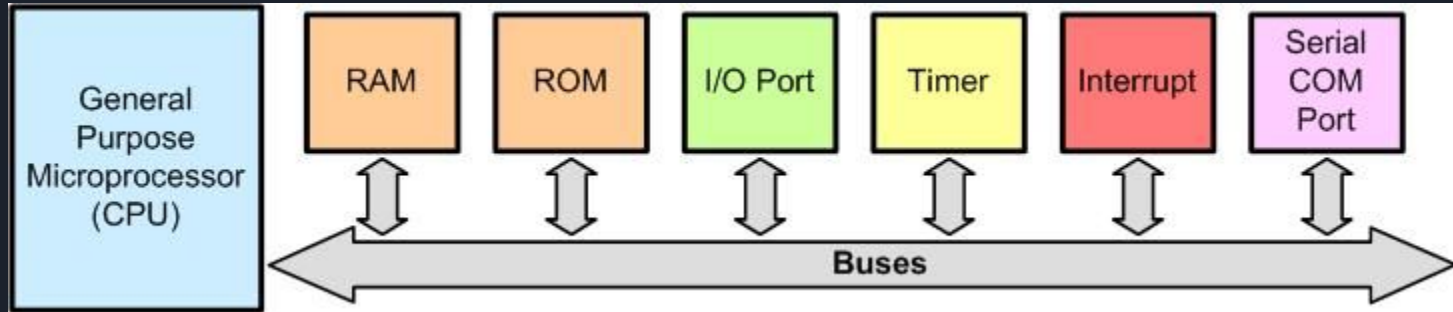
$$\sum_{i=1}^n \frac{C_i}{D_i} = \%$$

Tâche	Coût	Période	Echéance
$T_0$	2	5	5
$T_1$	3	7	7
$T_2$	1	10	10



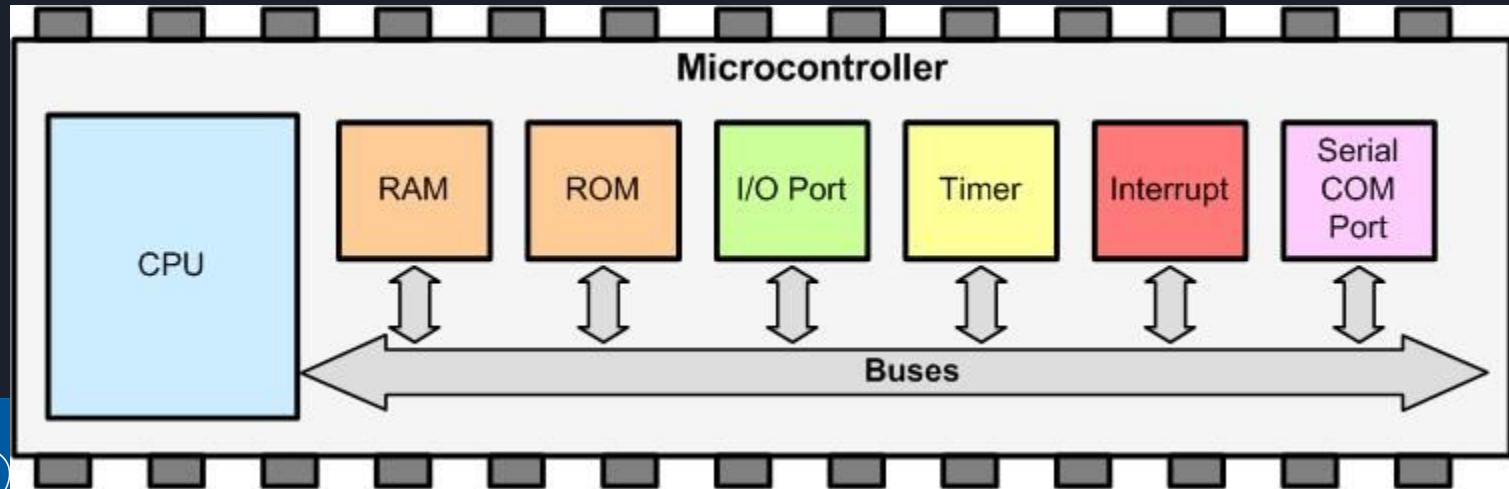
# Les microprocesseurs et microcontrôleurs

- Generale Purpose Microprocessor



# Les microprocesseurs et microcontrôleurs

- Micro-Contrôleurs





# Les microprocesseurs et microcontrôleurs

- Différence GPOS et RTOS

GPOS	RTOS
Interface Utilisateur	Embarqué
Exécution d'applications	Déterministe
Gestion mémoire dynamique	Gestion temps réel des tâches
Exemple : Windows Linux	Préemptif
	Exemple : freeRTOS



# Les microprocesseurs et microcontrôleurs

Choisir un microcontrôleur :

- Caractéristiques de la puce :
  - Vitesse,
  - Encombrement,
  - consommation
  - Taille RAM / ROM
  - Périphérique
  - Nombre de ports I/O, et l'horloge
  - Adaptabilité
  - Prix
- Environnement de développement
- Disponibilité



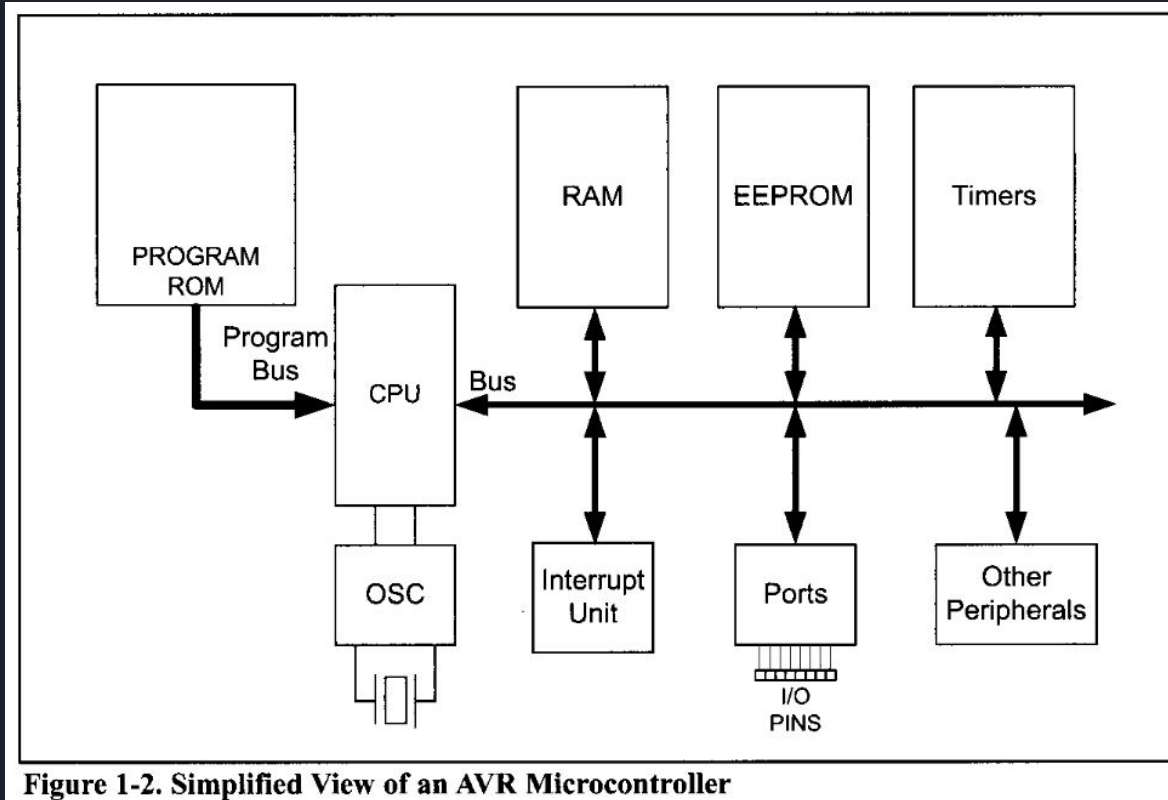


# Les microprocesseurs et microcontrôleurs

Bilan :

- Microcontrôleur moins cher que General Purpose Microprocessors ?
- Un microcontrôleur a normalement quels composants “on chip”, sur la puce ?
  - a) RAM      b) ROM      c) I/O
- Un General Purpose Microprocessors a normalement quels composants attachés ?
  - a) RAM      b) ROM      c) I/O
- Un système embarqué est aussi appelé un système dédié. Pourquoi ?
- Qu'est ce que le terme “embedded” signifie ?

# Les microprocesseurs et microcontrôleurs





# Les présupposés :

## conversion de base 2, 10, 16

- Nombre et systèmes de codage
- Conversion de décimal à binaire : diviser par 2 et conserver le reste de la division euclidienne jusqu'à zéro

### 25 à binaire

Quotient	Remainder
$25/2 = 12$	1 LSB (least significant bit)
$12/2 = 6$	0
$6/2 = 3$	0
$3/2 = 1$	1
$1/2 = 0$	1 MSB (most significant bit)

$$25_{10} = 11001_2$$



# Les présupposés :

## conversion de base 2, 10, 16

- Nombre et systèmes de codage
- Conversion de décimal à binaire : diviser par 2 et conserver le reste de la division euclidienne jusqu'à zéro

### 25 à binaire

Quotient	Remainder
$25/2 = 12$	1 LSB (least significant bit)
$12/2 = 6$	0
$6/2 = 3$	0
$3/2 = 1$	1
$1/2 = 0$	1 MSB (most significant bit)

25 -> 11001

# Les présupposés :

## conversion de base 2, 10, 16

- Nombre et systèmes de codage
- Explication base 10 des nombres décimaux

$$\begin{array}{rcl} 740683_{10} & = & \\ 3 \times 10^0 & = & 3 \\ 8 \times 10^1 & = & 80 \\ 6 \times 10^2 & = & 600 \\ 0 \times 10^3 & = & 0000 \\ 4 \times 10^4 & = & 40000 \\ 7 \times 10^5 & = & \underline{700000} \\ & & 740683 \end{array}$$

# Les présupposés :

## conversion de base 2, 10, 16

- Nombre et systèmes de codage
- Conversion binaire -> décimal

$740683_{10} =$		
$3 \times 10^0$	=	3
$8 \times 10^1$	=	80
$6 \times 10^2$	=	600
$0 \times 10^3$	=	0000
$4 \times 10^4$	=	40000
$7 \times 10^5$	=	<u>700000</u>
		740683

$110101_2 =$			<b>Decimal</b>	<b>Binary</b>
$1 \times 2^0$	=	$1 \times 1$	= 1	1
$0 \times 2^1$	=	$0 \times 2$	= 0	00
$1 \times 2^2$	=	$1 \times 4$	= 4	100
$0 \times 2^3$	=	$0 \times 8$	= 0	0000
$1 \times 2^4$	=	$1 \times 16$	= 16	10000
$1 \times 2^5$	=	$1 \times 32$	= <u>32</u>	<u>100000</u>
			53	110101



# Les présupposés :

## conversion de base 2, 10, 16

- Nombre et systèmes de codage
- Conversion binaire -> décimal

$16 = 2^5$	$8 = 2^4$	$4 = 2^2$	$2 = 2^1$	$1 = 2^0$
1	1	0	0	1

$32 = 2^6$	$16 = 2^5$	$8 = 2^4$	$4 = 2^2$	$2 = 2^1$	$1 = 2^0$
1	1	1	0	0	1



# Les présupposés :

## conversion de base 2, 10, 16

- Nombre et systèmes de codage
- Hexadécimal

**Table 0-1: Base 16  
Number System**

Decimal	Binary	Hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F





# Les présupposés :

## conversion de base 2, 10, 16

- Nombre et systèmes de codage
- Binaire -> Hexadécimal

	<b>100111110101</b>	
<b>1001</b>	<b>1111</b>	<b>0101</b>
<b>9</b>	<b>F</b>	<b>5</b>
	<b>9F5</b>	



# Les présupposés :

## conversion de base 2, 10, 16

- Nombre et systèmes de codage
- Hexadécimal -> Binaire

	<b>29B</b>	
<b>2</b>	<b>9</b>	<b>B</b>
<b>0010</b>	<b>1001</b>	<b>1011</b>
	<b>001010011011</b>	



# Les présupposés :

## conversion de base 2, 10, 16

- Nombre et systèmes de codage
- Décimal -> hexadécimal : 45

$32 = 2^6$	$16 = 2^5$	$8 = 2^4$	$4 = 2^2$	$2 = 2^1$	$1 = 2^0$
------------	------------	-----------	-----------	-----------	-----------



# Les présupposés :

## conversion de base 2, 10, 16

- Nombre et systèmes de codage
- Décimal -> hexadécimal : 45

$32 = 2^6$	$16 = 2^5$	$8 = 2^4$	$4 = 2^2$	$2 = 2^1$	$1 = 2^0$
1	0	1	1	0	1
	0011				1101
	2				D
					2D hex



# Les présupposés :

## conversion de base 2, 10, 16

- Nombre et systèmes de codage
- Décimal -> hexadécimal : 629

512	256	128	64	$32 = 2^6$	$16 = 2^5$	$8 = 2^4$	$4 = 2^2$	$2 = 2^1$	$1 = 2^0$
-----	-----	-----	----	------------	------------	-----------	-----------	-----------	-----------



# Les présupposés :

## conversion de base 2, 10, 16

- Nombre et systèmes de codage
- Décimal -> hexadécimal : 629

512	256	128	64	$32 = 2^6$	$16 = 2^5$	$8 = 2^4$	$4 = 2^2$	$2 = 2^1$	$1 = 2^0$
1	0	0	1	1	1	0	1	0	1
	0010				0111				0101
	2				7				5



# Les présupposés :

## conversion de base 2, 10, 16

- Nombre et systèmes de codage
- hexadécimal -> décimale : 6B2

512	256	128	64	$32 = 2^6$	$16 = 2^5$	$8 = 2^4$	$4 = 2^2$	$2 = 2^1$	$1 = 2^0$
-----	-----	-----	----	------------	------------	-----------	-----------	-----------	-----------

# Les présupposés :

## conversion de base 2, 10, 16

- Nombre et systèmes de codage
- hexadécimal -> décimale : 6B2

		6				B				2
		0110				1011				0010
1	1	0	1	0	1	1	0	0	1	0
1024	512	256	128	64	$32 = 2^6$	$16 = 2^5$	$8 = 2^4$	$4 = 2^3$	$2 = 2^2$	$1 = 2^1$
										1712



# Les présupposés :

## conversion de base 2, 10, 16

- Nombre et systèmes de codage
- Addition binaire

***Table 0-3: Binary Addition***

<b>A + B</b>	<b>Carry</b>	<b>Sum</b>
0 + 0	0	0
0 + 1	0	1
1 + 0	0	1
1 + 1	1	0

<i>Binary</i>	<i>Decimal</i>
1101	13
+ 1001	9
10110	22

# Les présupposés :

## conversion de base 2, 10, 16

- Nombre et systèmes de codage
- Soustraction  $\rightarrow x - y \rightarrow x + \text{Complément à 2 de } y$
- Complément à 2 : inverser les bits et ajouter 1.

10011101	binary number
01100010	1's complement
+      1	
<hr/>	
01100011	2's complement



## Les présupposés :

### conversion de base 2, 10, 16

- Nombre et systèmes de codage
- Addition hexadécimale

23D9  
+ 94BE  
B897

LSD:  $9 + 14 = 23$   
 $1 + 13 + 11 = 25$   
 $1 + 3 + 4 = 8$   
MSD:  $2 + 9 = B$

$23 - 16 = 7$  with a carry  
 $25 - 16 = 9$  with a carry

# Les présupposés :

## conversion de base 2, 10, 16

- Nombre et systèmes de codage
- Subtraction hexadécimale : En soustrayant deux nombres hexadécimaux, si le deuxième chiffre est supérieur au premier, empruntez 16 du chiffre précédent.

$$\begin{array}{r} 59F \\ - 2B8 \\ \hline 2E7 \end{array}$$

$$\begin{array}{l} \text{LSD: } 8 \text{ from } 15 = 7 \\ 11 \text{ from } 25 (9 + 16) = 14 (E) \\ 2 \text{ from } 4 (5 - 1) = 2 \end{array}$$



# Les présupposés :

## Conversion de base 2, 10, 16

- Nombre et systèmes de codage
- Bilan :
  1. Pourquoi les ordinateurs utilisent-ils le système binaire au lieu du système décimal ?
  2. Convertissez 34 en binaire et en hexadécimal.
  3. Convertissez 110101 en hexadécimal et décimal.
  4. Effectuez l'addition binaire :  $101100 + 101$ .
  5. Convertissez  $101100$  en sa représentation en complément à 2.
  6. Ajouter  $36BH + F6H$ .

# Les présupposés : Logique Binaire

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1



## Logical OR Function

Inputs		Output
X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1



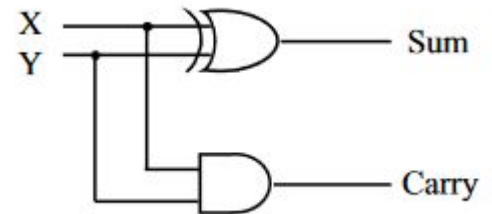
## Logical XOR Function

Inputs		Output
X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0



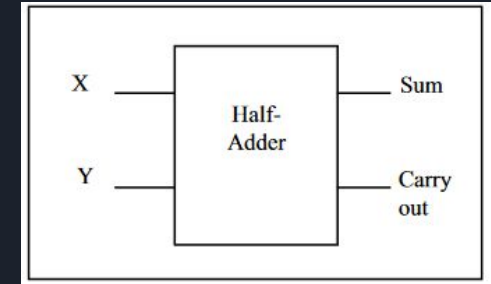
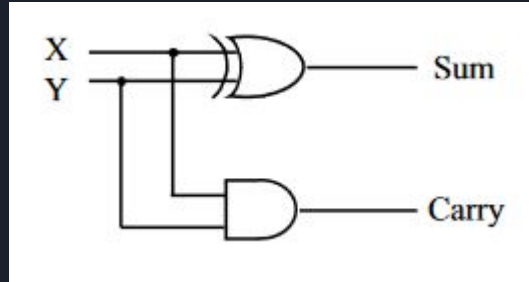
## Half-adder

	Carry	Sum
0 + 0 =	0	0
0 + 1 =	0	1
1 + 0 =	0	1
1 + 1 =	1	0

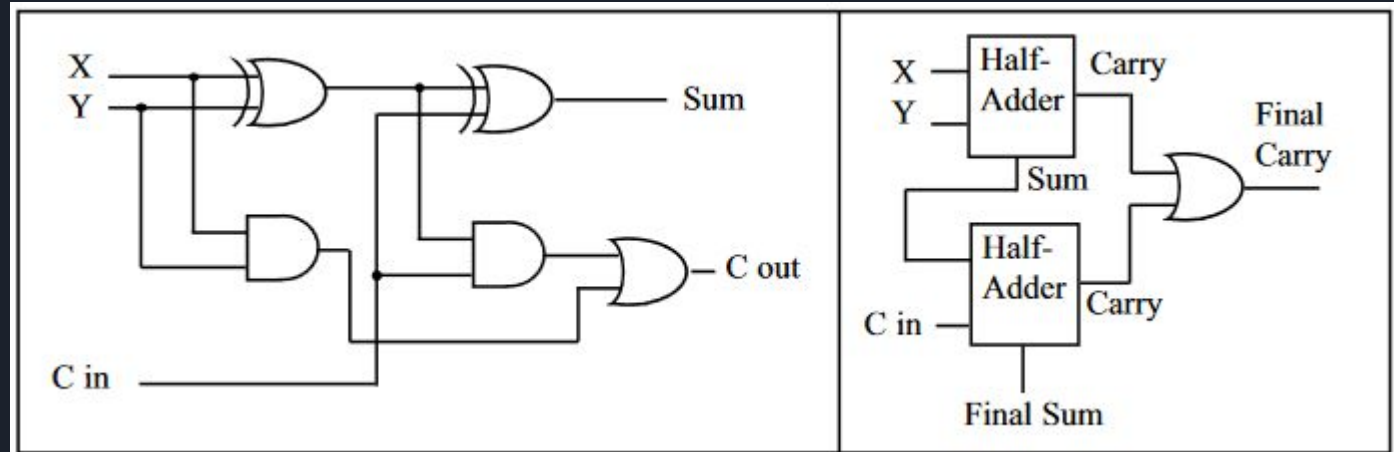


# Les présupposés : Logique Binaire

Half-adder



Full adder





# Les présupposés : Logique Binaire

## Bilan

1. L'opération logique \_\_\_\_\_ donne une sortie 1 lorsque toutes les entrées sont 1.
2. L'opération logique \_\_\_\_\_ donne une sortie 1 lorsqu'une ou plusieurs de ses entrées est 1.
3. L'opération logique \_\_\_\_\_ est souvent utilisée pour comparer deux entrées afin de déterminer si ils ont la même valeur.



# Les présupposés : La mémoire

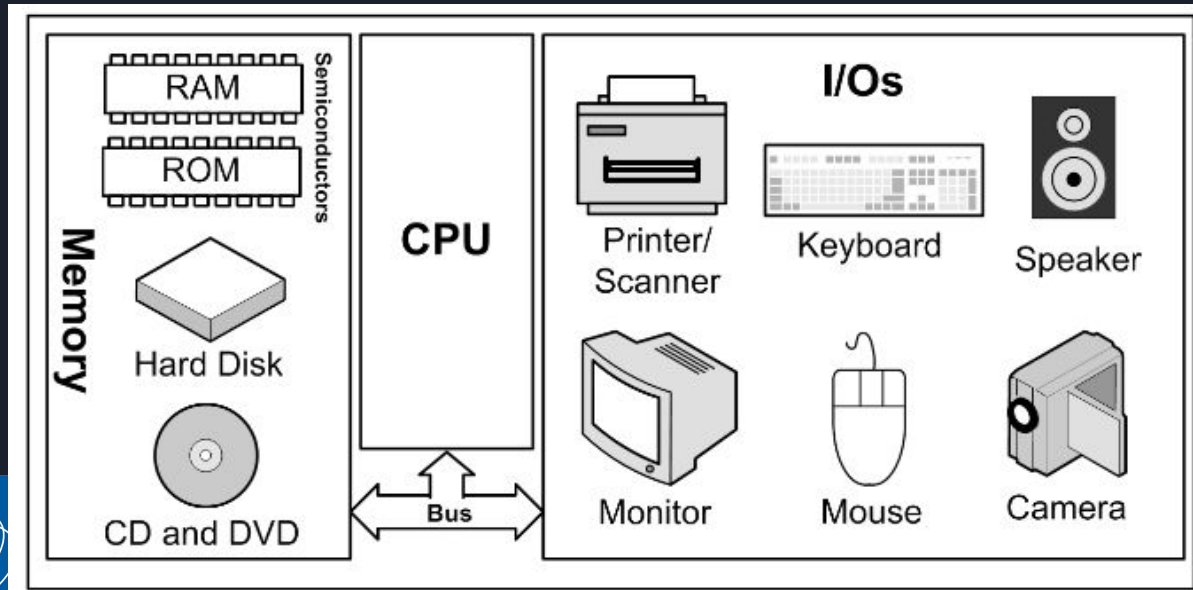
Bit	0
Nibble	0000
Byte	0000 0000
Word	0000 0000 0000 0000

Kilobyte =  $2^{10}$  -> 1024

Megabyte =  $2^{20}$  -> 1 048 576

# Les présupposés : La mémoire

Internal organization of computers





# Les présupposés : La mémoire

La capacité mémoire : nombre de bits

L'organisation de la mémoire :

- Une puce mémoire contient  $2^x$  emplacements, où  $x$  est le nombre de pins d'adresse.
- Chaque emplacement contient des bits  $y$ , où  $y$  est le nombre de pins de données sur la puce.
- La puce entière contiendra  $2^x \times y$  bits, où  $x$  est le nombre de broches d'adresse et  $y$  est le nombre de broches de données sur la puce

La vitesse:

- access time (nano s / micro s)

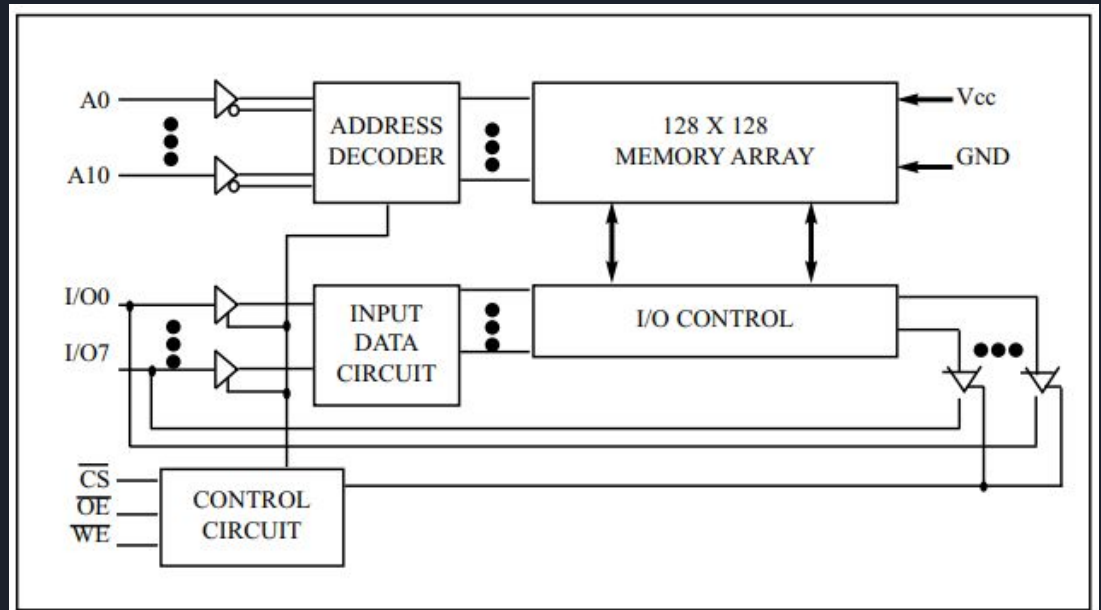
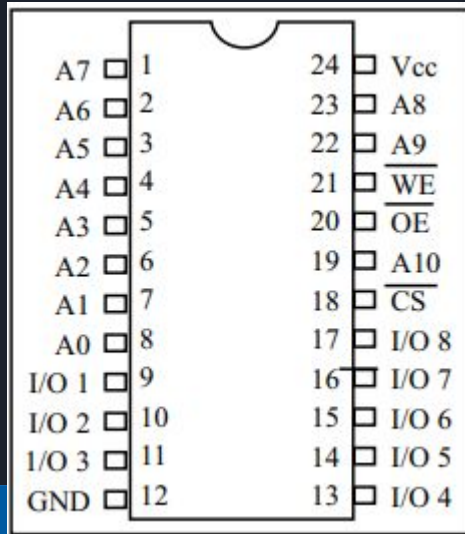
Exemple : organisation et capacité ?

12 adresses pins, 4 data pins

512k memory chip, 8 pins of data

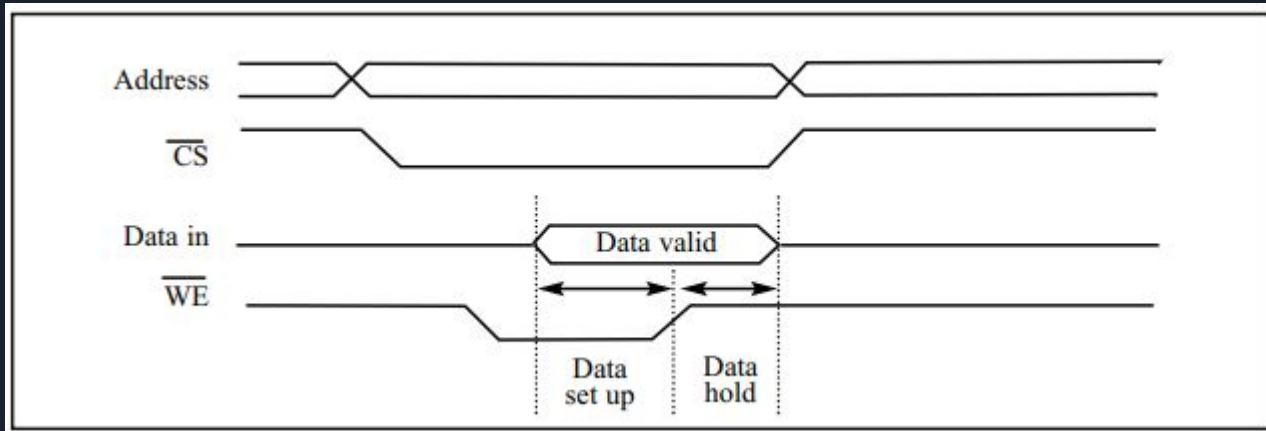
# Les présupposés : La mémoire

Exemple Static RAM : 2k x 8



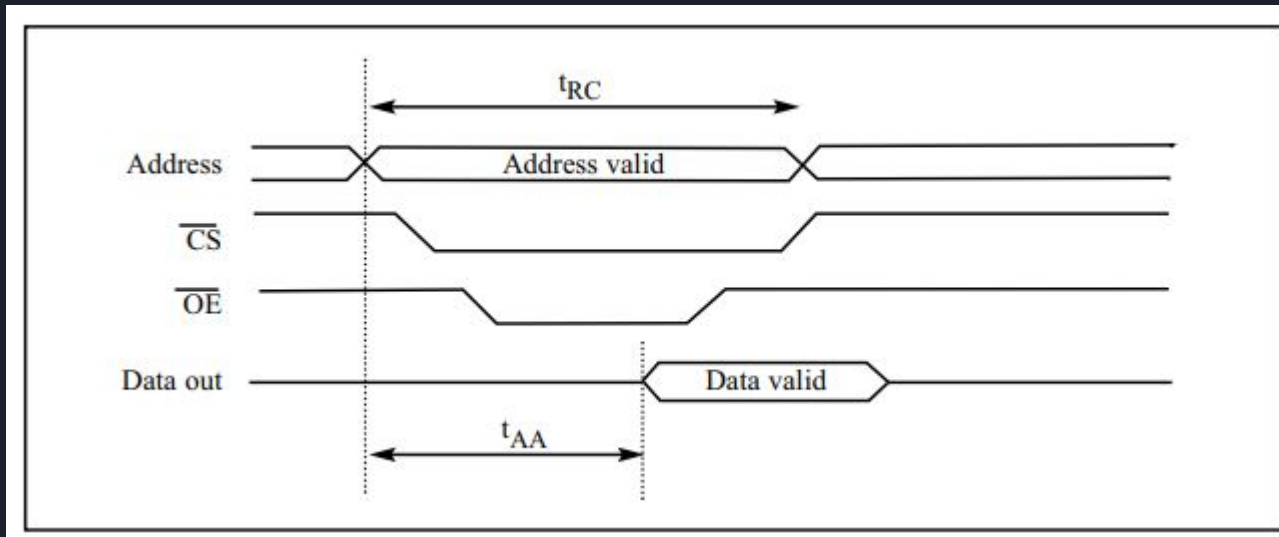
# Les présupposés : La mémoire

Exemple Static RAM :  
Write



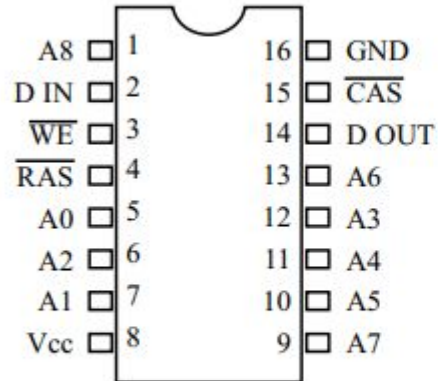
# Les présupposés : La mémoire

Exemple Static RAM :  
Read

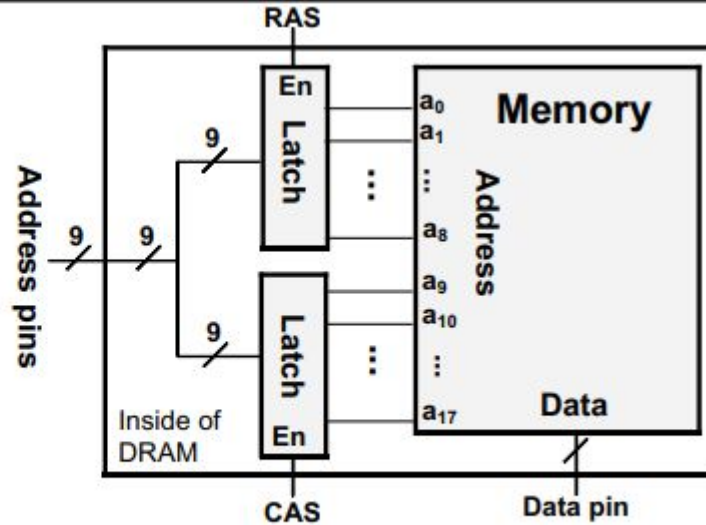


# Les présupposés : La mémoire

Exemple dynamic RAM  
Read



(a) pinout



(b) internal view



# Les présupposés : La mémoire

Exemple différence de pins :

a) 16k x 4 DRAM

b) 16k x 4 SRAM





# Les présupposés : La mémoire

Exemple différence de pins d'adresse nécessaire :

a) 16k x 4 DRAM

b) 16k x 4 SRAM

a) 7 A0-A6 + 2 pins RAS, CAS

b) 14 A0 - A13.



# Les présupposés : La mémoire

Bilan :

1. Combien d'octets font 24 kilooctets?
2. Que signifie « RAM »? Comment est-il utilisé dans les systèmes informatiques?
3. Que signifie « ROM »? Comment est-il utilisé dans les systèmes informatiques?
4. Pourquoi la mémoire vive est-elle appelée mémoire volatile?
5. Énumérez les trois principales composantes d'un système informatique.
6. Que signifie « CPU »? Expliquez sa fonction dans un ordinateur.
7. La vitesse de la mémoire semi-conductrice se situe dans la plage
  - (a) microsecondes
  - (b) millisecondes
  - (c) nanosecondes
  - (d) picosecondes



# Les présupposés : La mémoire

Bilan :

1. Trouver l'organisation et la capacité de la puce pour chaque ROM avec le nombre indiqué de adresses et broches de données.

- a) 14 adresses, 8 data
- b) 16 adresses, 8 data
- c) 12 adresses, 8 data

2. Trouver l'organisation et la capacité de la puce pour chaque mémoire vive avec le nombre indiqué d'adresses et broches de données.

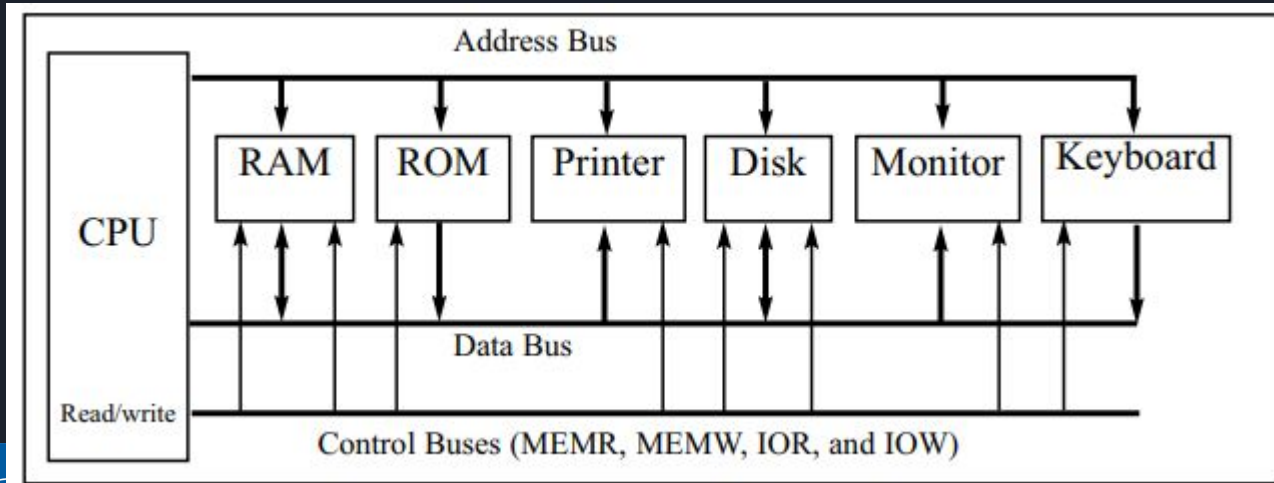
- (a) 11 adresses, 1 data SRAM (b) 13 adresses, 4 data SRAM
- (c) 8 adresses, 4 data DRAM (d) 9 adresse, 1 data DRAM

3. Trouver la capacité et le nombre de broches réservées pour l'adresse et les données pour les puces de mémoire avec les organisations suivantes.

- (a) 16K 4 SRAM
- (d) 256 K 4 SRAM

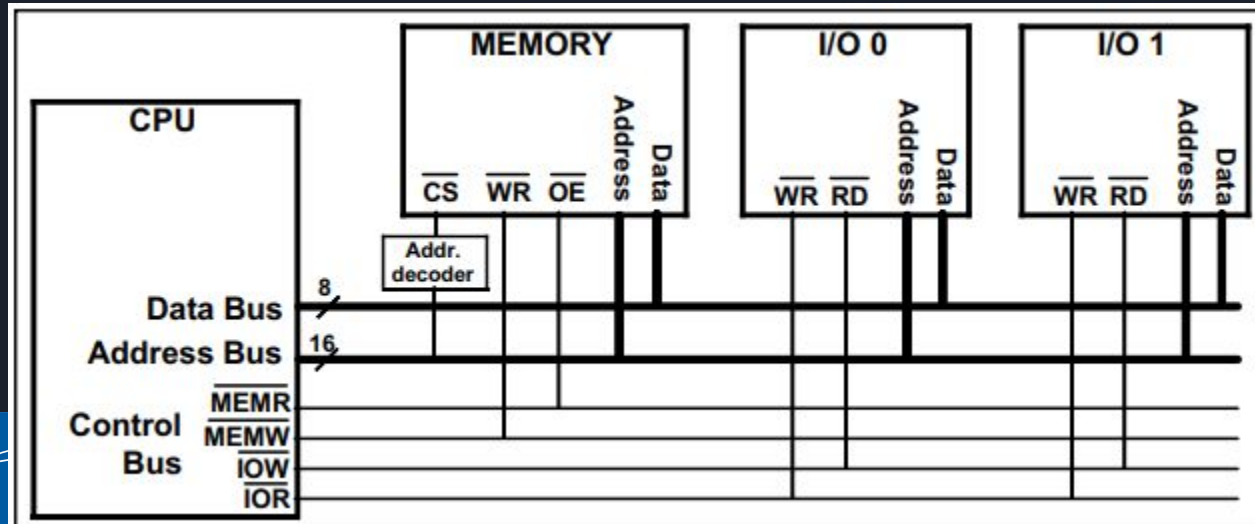
# Les présupposés : Le bus et le décodage d'adresse

3 types de bus

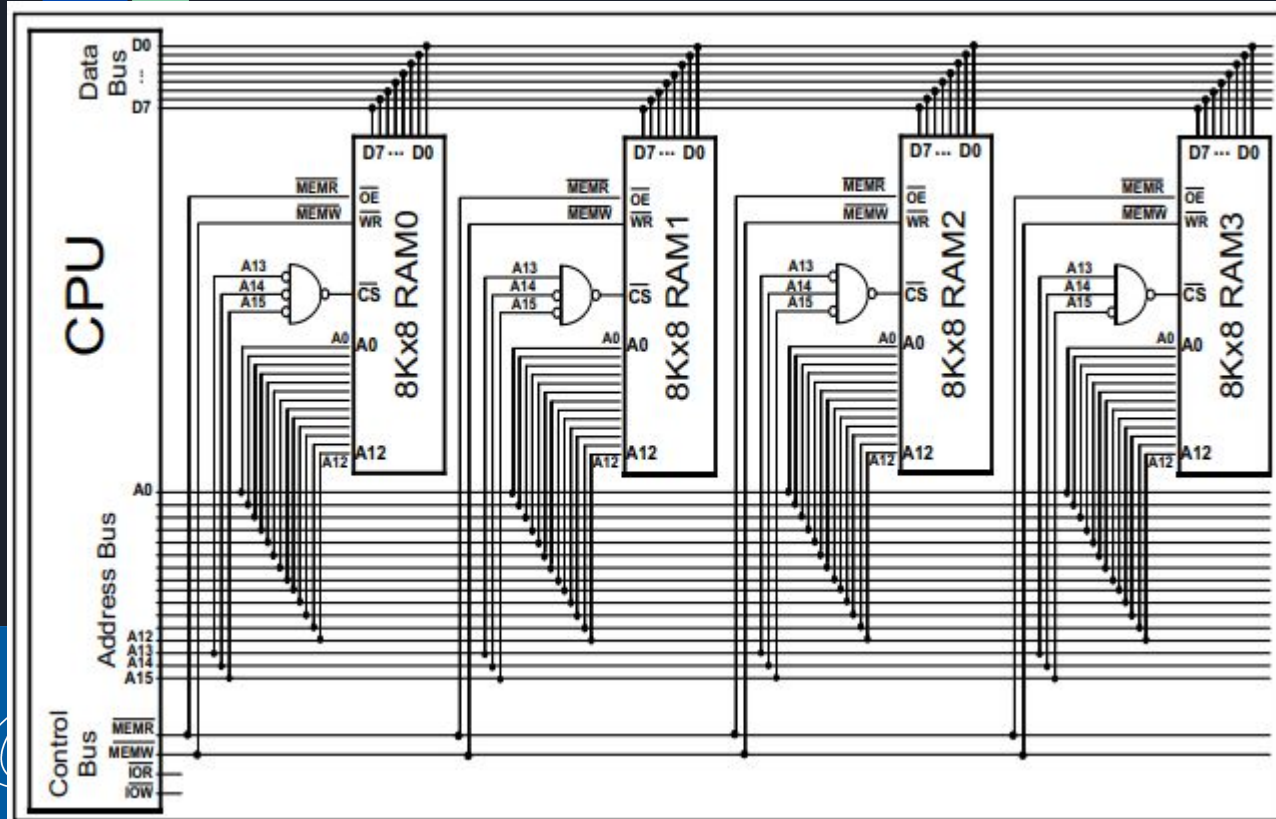


# Les présupposés : Le bus et le décodage d'adresse

Signal de contrôle pour séparer I/O et mémoire



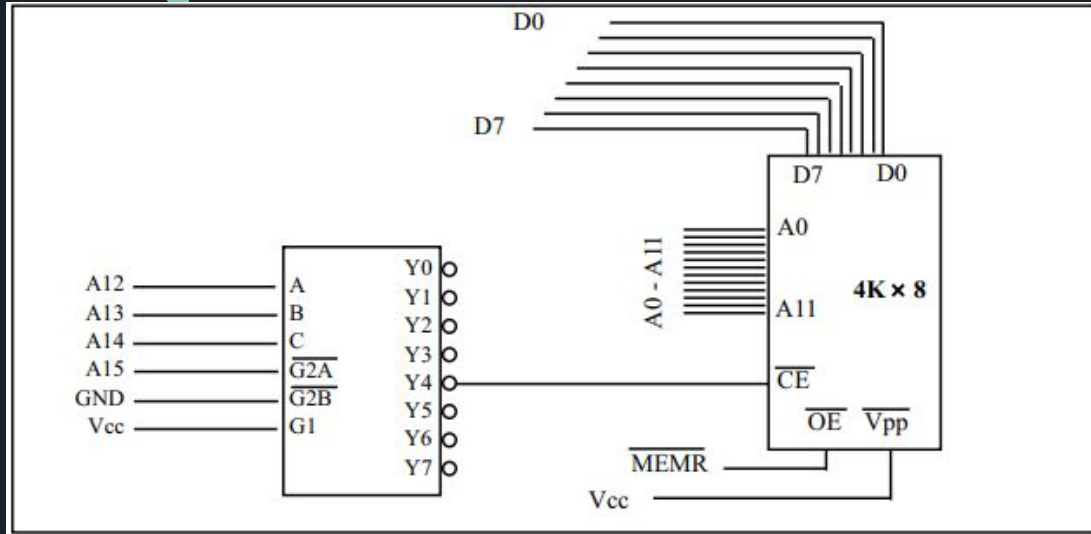
# Les présupposés : Le bus et le décodage d'adresse : Logic gate adress



Address Range	
0000H-1FFFH	RAM 0
2000H-3FFFH	RAM 1
4000H-5FFFH	RAM 2
6000H-7FFFH	RAM 3
8000H-FFFFH	Not used

4000 -> 4FFFF

# Les présupposés : Le bus et le décodage d'adresse : 74LS138 3-8 décodeur



Inputs				Outputs									
Enable		Select											
G1	G2	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7	
X	H	X	X	X	H	H	H	H	H	H	H	H	
L	X	X	X	X	H	H	H	H	H	H	H	H	
H	L	L	L	L	L	H	H	H	H	H	H	H	
H	L	L	L	H	H	L	H	H	H	H	H	H	
H	L	L	H	L	H	H	L	H	H	H	H	H	
H	L	L	H	H	H	H	H	L	H	H	H	H	
H	L	H	L	L	H	H	H	H	L	H	H	H	
H	L	H	L	H	H	H	H	H	H	L	H	H	
H	L	H	H	L	H	H	H	H	H	H	L	H	
H	L	H	H	H	H	H	H	H	H	H	H	L	

Y4

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1

4000 -> 4FFFF

# Les présupposés : Le bus et le décodage d'adresse

Bilan :

1. Énumérez les trois types de bus qui se trouvent dans les systèmes informatiques et énoncez brièvement leur objectif.
2. Indiquez lequel des éléments suivants est unidirectionnel et lequel est bidirectionnel :  
a) bus de données b) bus d'adresse
3. Si un bus d'adresse pour un ordinateur donné compte 16 lignes, quel est le montant maximal de mémoire accessible (chaque emplacement de mémoire est 8 bits) ?
4. Un bloc mémoire donné utilise des adresses 4000H–7FFFH. Combien de kilooctets est-ce blocage de mémoire?

5. Quelle est la fourchette des adresses attribuées à Y5?

Inputs				Outputs								
Enable		Select										
G1	G2	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
H	L	H	L	L	H	H	H	H	L	H	H	H



# Les présupposés : Le bus et le décodage d'adresse

Bilan :

1. Énumérez les trois types de bus qui se trouvent dans les systèmes informatiques et énoncez brièvement leur objectif.
2. Indiquez lequel des éléments suivants est unidirectionnel et lequel est bidirectionnel :  
a) bus de données b) bus d'adresse
3. Si un bus d'adresse pour un ordinateur donné compte 16 lignes, quel est le montant maximal de mémoire accessible (chaque emplacement de mémoire est 8 bits) ?
4. Un bloc mémoire donné utilise des adresses 4000H–7FFFH. Combien de kilooctets est-ce blocage de mémoire?

5. Quelle est la fourchette des adresses attribuées à Y4?

Inputs				Outputs							
Enable		Select									
G1	G2	C	B A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
H	L	H	L	H	H	H	H	L	H	H	H



# Les présupposés : CPU architecture

Inside a CPU : Central Processing Unit

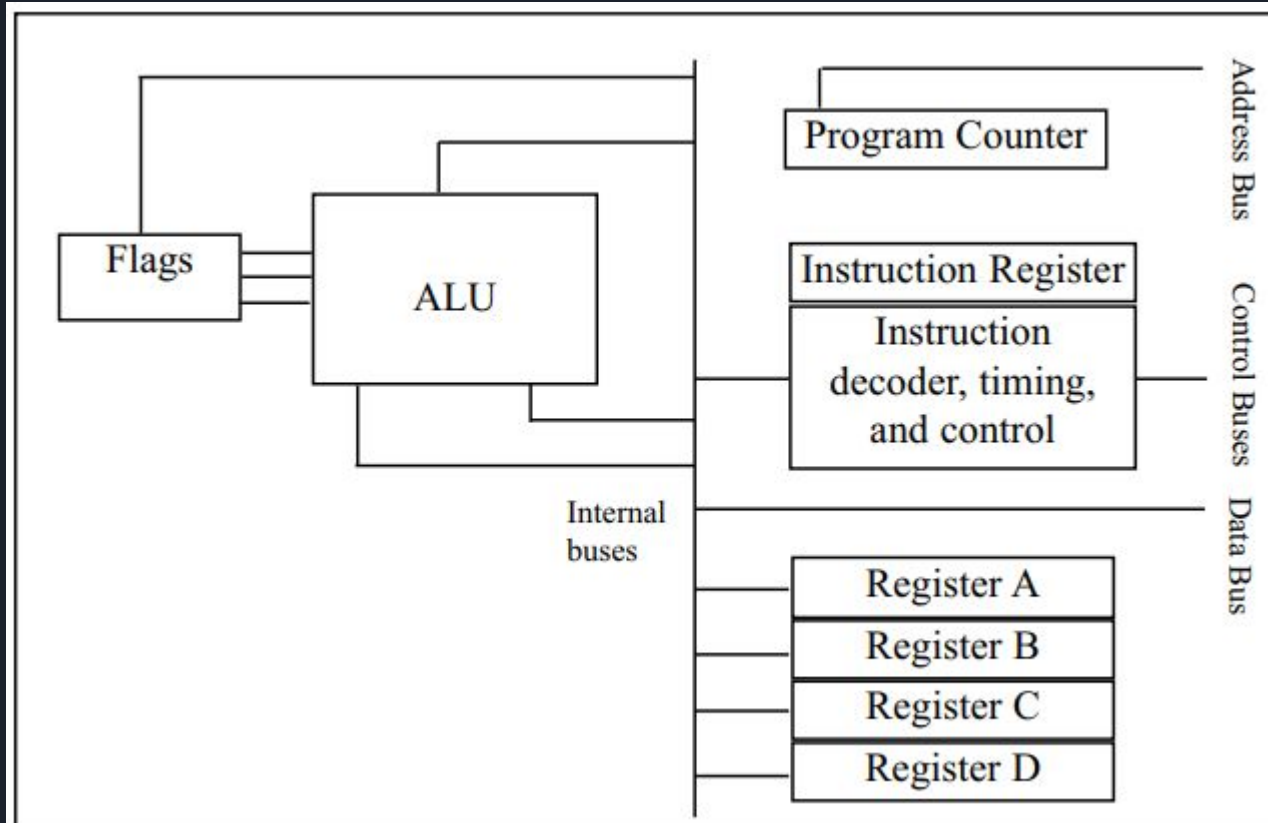
Le programme en mémoire donne des instructions pour faire des actions.

- contrôler un robot
- addition des chiffres.

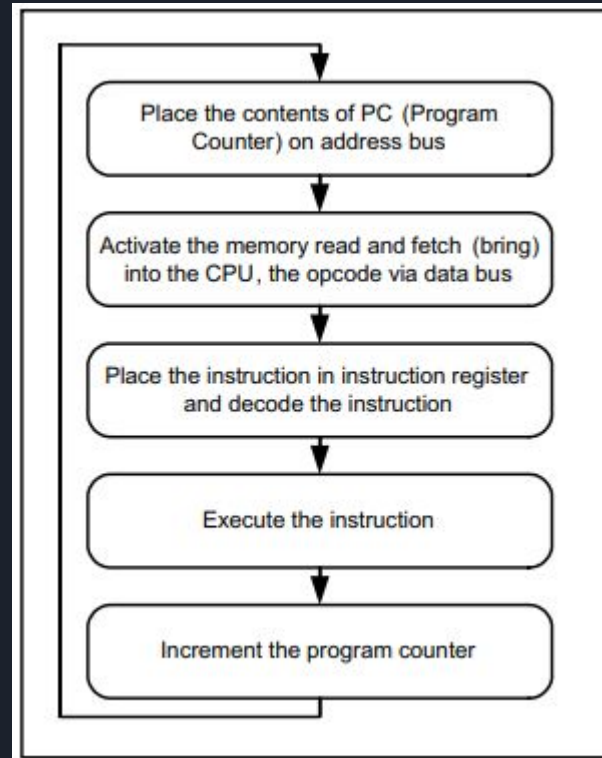
CPU = récupère les instructions de la mémoire et les exécute.

.

# Les présupposés : CPU architecture



# Les présupposés : CPU architecture, les instructions



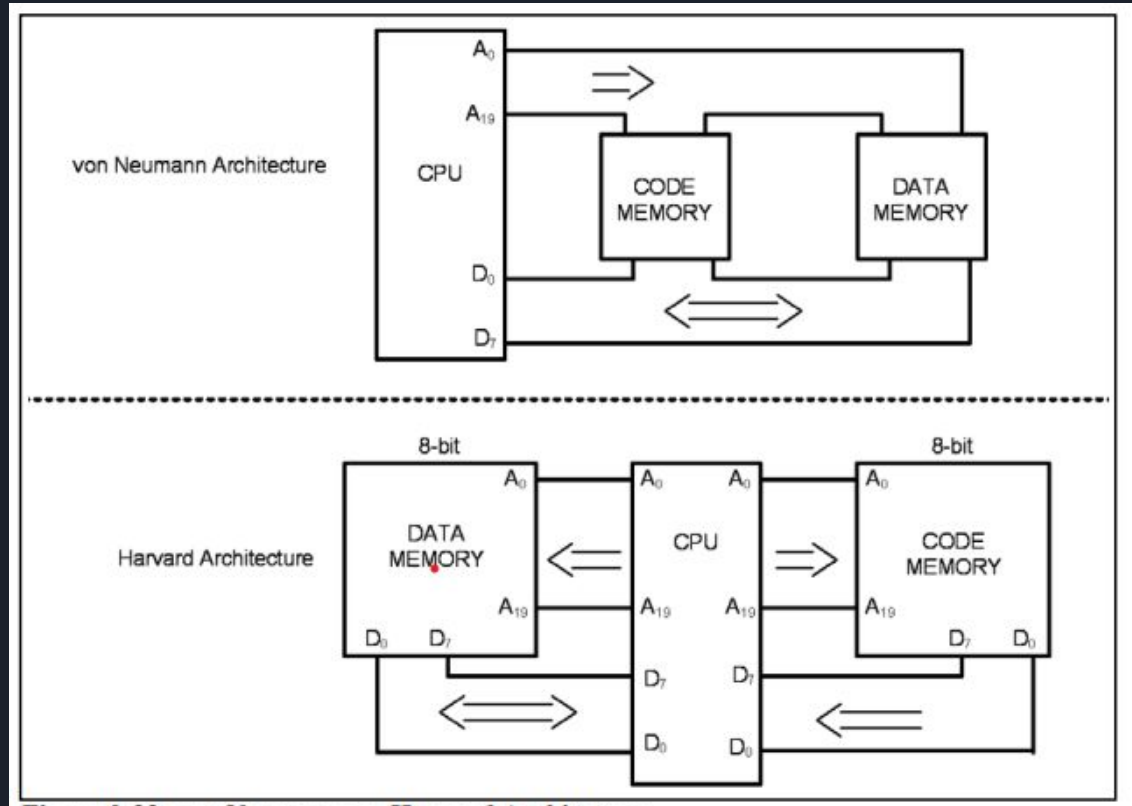


# Les présupposés : CPU architecture, les instructions décomposées

Action	Code	Data
Move value 21H into register A	B0H	21H
Add value 42H to register A	04H	42H
Add value 12H to register A	04H	12H

<b>Memory address</b>	<b>Contents of memory address</b>
1400	(B0)code for moving a value to register A
1401	(21)value to be moved
1402	(04)code for adding a value to register A
1403	(42)value to be added
1404	(04)code for adding a value to register A
1405	(12)value to be added
1406	(F4)code for halt

# Les présupposés : CPU architectures





# Les présupposés : CPU architecture,

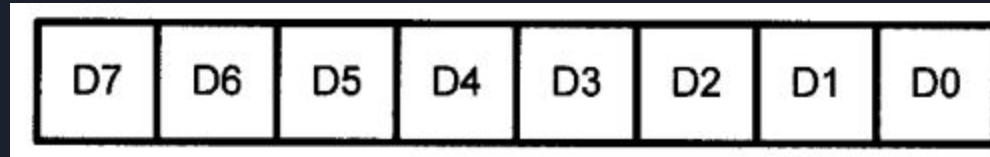
Bilan :

1. Que signifie « ALU » ? Quel est son but?
2. Comment les registres sont-ils utilisés dans les systèmes informatiques ?
3. A quoi sert le compteur de programmes ?
4. A quoi sert le décodeur d'instructions ?
5. Vrai ou faux. L'architecture Harvard utilise la même adresse et les mêmes bus de données pour récupérer les deux codes et données.



# MicroController architecture et la notion d'assembleur : Les General Purpose Register

Register = stocke data temporairement . -> 8 bits ici



.Most significant bit

least significant bit





# MicroController architecture et la notion d'assembleur : Les General Purpose Register

Adresse la plus basse

R0
R1
R2
⋮
R14
• R15
R16
R17
R18
⋮
R30
R31



# MicroController architecture et la notion d'assembleur : Les General Purpose Register

Instruction LDI :

```
LDI  Rd, K           // stocke dans le registre Rd la valeur K.  
                        // d entre 16 et 31  
                        // K entre 0 et 255
```

I = immediate

Convention : \$50, 0x50 -> hexadécimale  
60 -> décimale

0x5 -> 0x05

LDI Rd, 0x505 -> erreur.



# Les General Purpose Register

Instruction ADD :

ADD Rd, Rr ;            // ADD Rr à Rd et stocke le résultat dans Rd

LDI R16, 0x25

LDI R17, 0x34

ADD R16, R17

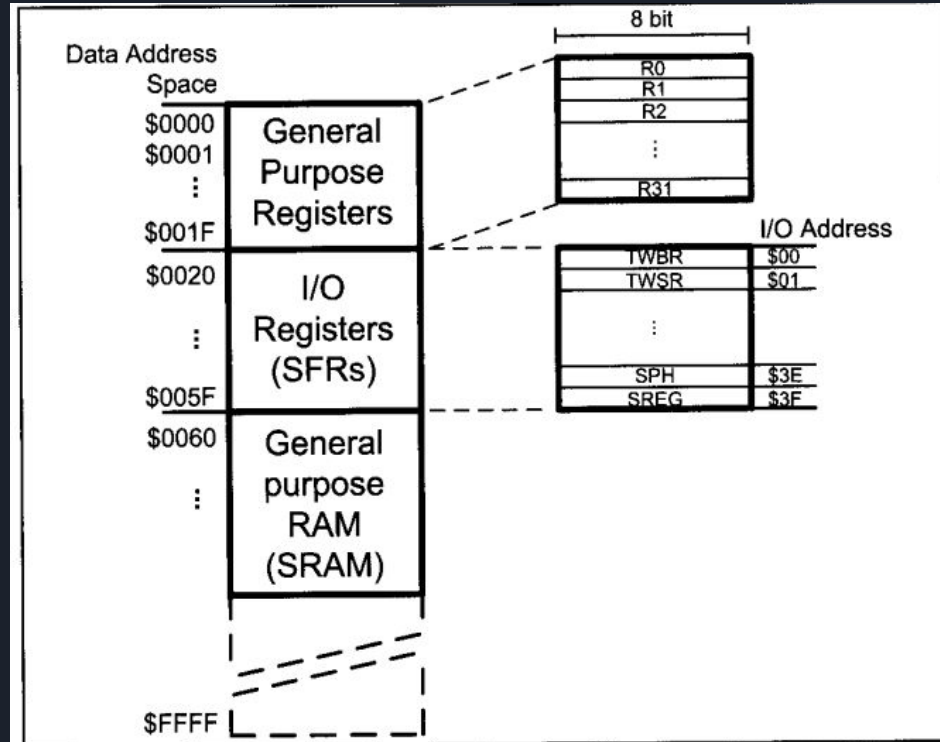


# MicroController architecture et la notion d'assembleur : Les General Purpose Register

## Bilan

1. Instruction pour mettre 0x34 dans le registre R29.
2. Instructions pour additionner 0x16 et 0xCD de stocker le résultat dans R19.
3. Aucune valeur ne peut être placée directement dans le General Purpose Register. Vrai ou Faux.
4. La valeur maximum stockée dans un registre 8 bits ? en hexadécimal ?

# MicroController architecture et la notion d'assembleur : La mémoire





# MicroController architecture et la notion d'assembleur : La mémoire

## Bilan

1. Vrai ou faux. Les registres d'I/O sont utilisés pour stocker des données.
2. Les GPRs avec les registres I/O et la SRAM sont appelés \_\_\_\_
3. Les registres d'I/O sont de \_\_\_\_\_ bits.
4. L'espace de mémoire de données est divisé en \_\_\_\_ pièces.



# MicroController architecture et la notion d'assembleur : Les instructions mémoires

L'instruction : LDS Load direct from Data Space

LDS Rd, K    //K adresse  
              //register

LDS R5, 0x200

Exemple : Addition de la valeur à l'adresse 0x300 à celle de l'adresse 0x302 :

LDS R0, 0x300  
LDS R1, 0x302  
ADD R1, R0

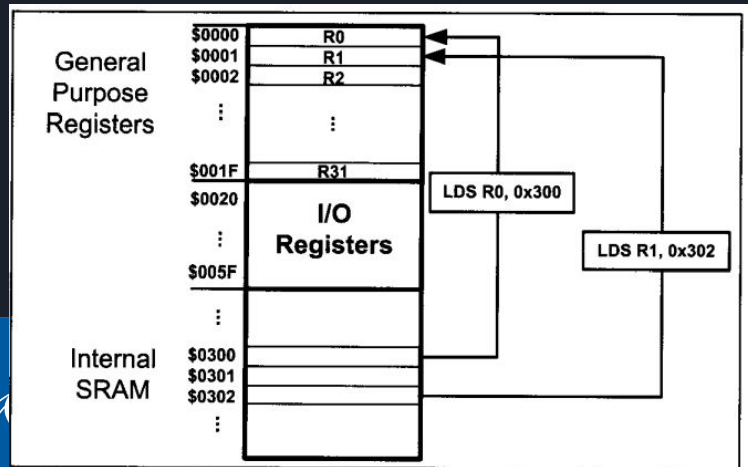
# MicroController architecture et la notion d'assembleur : Les instructions mémoires

Exemple : Addition de la valeur à l'adresse 0x300 à celle de l'adresse 0x302 :

LDS R0, 0x300

LDS R1, 0x302

ADD R1, R0



	R0	R1	Loc \$300	Loc \$302
Before LDS R0,0x300	?	?	$\alpha$	$\beta$
After LDS R0,0x300	$\alpha$	?	$\alpha$	$\beta$
After LDS R1,0x302	$\alpha$	$\beta$	$\alpha$	$\beta$
After ADD R0, R1	$\alpha + \beta$	$\beta$	$\alpha$	$\beta$





# MicroController architecture et la notion d'assembleur : Les instructions mémoires

L'instruction : STS STore direct to data Space

STS K, Rr    // K adresse

              //Rr register

STS 0x200, R0

Exemple : Copie la valeur à l'adresse 0x99 aux adresses 0x200 à 0x203

LDS R0, 0x99

STS 0x203, R0

STS 0x202, R0

STS 0x201, R0

STS 0x200, R0

Address	Data
\$200	0x99
\$201	0x99
\$202	0x99
\$203	0x99

# MicroController architecture et la notion d'assembleur : Les instructions mémoires

L'instruction : IN for I/O location

IN Rd, A; // store an I/O à l'adresse A dans Rd.

Address		Name
Mem.	I/O	
\$20	\$00	TWBR
\$21	\$01	TWSR
\$22	\$02	TWAR
\$23	\$03	TWDR
\$24	\$04	ADCL
\$25	\$05	ADCH
\$26	\$06	ADCSRA
\$27	\$07	ADMUX
\$28	\$08	ACSR
\$29	\$09	UBRRL
\$2A	\$0A	UCSRB
\$2B	\$0B	UCSRA
\$2C	\$0C	UDR
\$2D	\$0D	SPCR
\$2E	\$0E	SPSR
\$2F	\$0F	SPDR
\$30	\$10	PIND
\$31	\$11	DDRD
\$32	\$12	PORTD
\$33	\$13	PINC
\$34	\$14	DDRC
\$35	\$15	PORTC

Address		Name
Mem.	I/O	
\$36	\$16	PINB
\$37	\$17	DDRB
\$38	\$18	PORTB
\$39	\$19	PINA
\$3A	\$1A	DDRA
\$3B	\$1B	PORTA
\$3C	\$1C	EECR
\$3D	\$1D	EEDR
\$3E	\$1E	EEARL
\$3F	\$1F	EEARH
\$40	\$20	UBRRC
		UBRRH
\$41	\$21	WDTCSR
\$42	\$22	ASSR
\$43	\$23	OCR2
\$44	\$24	TCNT2
\$45	\$25	TCCR2
\$46	\$26	ICR1L
\$47	\$27	ICR1H
\$48	\$28	OCR1BL
\$49	\$29	OCR1BH
\$4A	\$2A	OCR1AL

Address		Name
Mem.	I/O	
\$4B	\$2B	OCR1AH
\$4C	\$2C	TCNT1L
\$4D	\$2D	TCNT1H
\$4E	\$2E	TCCR1B
\$4F	\$2F	TCCR1A
\$50	\$30	SFIOR
\$51	\$31	OCDR
		OSCCAL
\$52	\$32	TCNT0
\$53	\$33	TCCR0
\$54	\$34	MCUCSR
\$55	\$35	MCUCR
\$56	\$36	TWCR
\$57	\$37	SPMCR
\$58	\$38	TIFR
\$59	\$39	TIMSK
\$5A	\$3A	GIFR
\$5B	\$3B	GICR
\$5C	\$3C	OCR0
\$5D	\$3D	SPL
\$5E	\$3E	SPH
\$5F	\$3F	SREG

# MicroController architecture et la notion d'assembleur : Les instructions mémoires

```
LDI    R16, 0x99    ;load R16 with value 0x99
STS    0x212, R16
LDI    R16, 0x85    ;load R16 with value 0x85
STS    0x213, R16
LDI    R16, 0x3F    ;load R16 with value 0x3F
STS    0x214, R16
LDI    R16, 0x63    ;load R16 with value 0x63
STS    0x215, R16
LDI    R16, 0x12    ;load R16 with value 0x12
STS    0x216, R16
```

Address	Data
\$212	0x99
\$213	0x85
\$214	0x3F
\$215	0x63
\$216	0x12

# MicroController architecture et la notion d'assembleur : Les instructions mémoires

```
LDI    R20, 5      ;load R20 with 5
LDI    R21, 2      ;load R21 with 2
ADD    R20, R21     ;add R21 to R20
ADD    R20, R21     ;add R21 to R20
STS    0x120, R20   ;store in location 0x120 the contents of R20
```

<u>Location</u>	<u>Data</u>	<u>Location</u>	<u>Data</u>	<u>Location</u>	<u>Data</u>	<u>Location</u>	<u>Data</u>	<u>Location</u>	<u>Data</u>
R20	5	R20	5	R20	7	R20	9	R20	9
R21		R21	2	R21	2	R21	2	R21	2
0x120		0x120		0x120		0x120		0x120	9
After LDI R20, 5		After LDI R21, 2		After ADD R20, R21		After ADD R20, R21		After STS 0x120, R20	



# MicroController architecture et la notion d'assembleur : Les instructions mémoires

Instruction	Définition
OUT A, Rr	stocke I/O location la valeur de Rr
MOV Rd, Rr	copie Rr dans Rd
INC Rd	incrémente Rd
SUB Rd , Rr	$Rd = Rd - Rr$
DEC Rd	décrémente Rd
COM Rd	inverse les bits 0x55 -> 0xAA



# MicroController architecture et la notion d'assembleur : Les instructions mémoires

Exemple programme qui inverse continuellement le Port B à l'adresse 0x55 :

```
LDI    R20, 0x55
OUT     PORTB, R20
L1:    COM    R20
OUT     PORTB, R20
JMP     L1
```



# MicroController architecture et la notion d'assembleur : Les instructions mémoires

Bilan :

1. Vrai ou faux. Aucune valeur ne peut être chargée directement dans la SRAM interne.
2. Écrivez des instructions pour charger la valeur 0x95 dans le registre d'E/S SPL.
3. Rédigez des instructions pour ajouter 2 au contenu de R18.
4. Écrivez des instructions pour additionner les valeurs 0x16 et 0xCD. Placez le résultat à l'emplacement 0x400 de la mémoire de données.
5. Quelle est la plus grande valeur hexadécimale pouvant être déplacée vers un emplacement de la mémoire de données ? Quel est l'équivalent décimal de la valeur hexadécimale ?
6. "ADD R16, R3" met le résultat en \_\_\_\_
7. Que fait "OUT OCRO, R23" ?
8. Quelle est l'erreur "STS OCRO, R23" ?