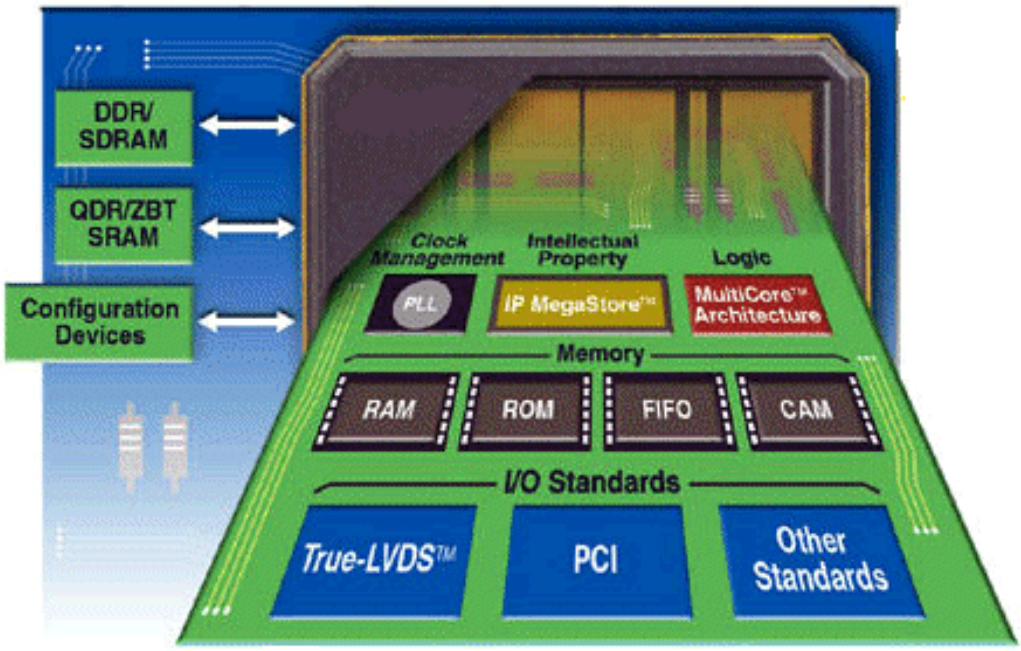
Microprocesseurs microcontrôleurs et DSP



Jean Louis BOIZARD
Maître de conférences IUFM Midi-Pyrénées

IUP AISEM L3
Version 4.0

Module 3 « Microprocesseurs, microcontrôleurs et DSP

1) Les microprocesseurs
    - 1.1) Historique et évolution
    - 1.2) Domaines d'utilisation
    - 1.3) Architecture d'un système à µP
    - 1.4) Architecture interne d'un µP
    - 1.5) Architectures RISC et CISC

2) Les microcontrôleurs
    - 2.1 Exemples d'architectures internes (ATMEL AT90S8515, Motorola 68HC11)

3) Les DSP
    - 3.1 Architecture interne

4) Les méthodes et outils de développement
    - 4.1 Les langages de programmation
    - 4.2 Les simulateurs
    - 4.3 Les émulateurs « In circuit »
    - 4.4 Les émulateurs « temps réel » à mémoire trace
    - 4.5 Les émulateurs « low cost »

5) Le microcontrôleur ATMEL AT90S8535
    - 5.1 Présentation
    - 5.2 Organisation de la mémoire
    - 5.3 Architecture du microprocesseur
    - 5.4 Reset, interruptions et veille
    - 5.5 Le jeu d'instructions
    - 5.6 Les directives d'assemblage
    - 5.7 Les périphériques

Avant-propos

Le présent document a pour objectif principal de démystifier le fonctionnement des microprocesseurs, microcontrôleurs et processeurs de signaux (DSP).

L'évolution considérable des outils de développement fait qu'il est tout à fait possible de construire aujourd'hui des applications à base de ces composants sans en avoir a priori une connaissance intime.

Afin d'aborder le contenu de ce cours dans les meilleures conditions, le lecteur aura avantage à revoir/assimiler les points suivants :

- Numération binaire et hexadécimale
- logique combinatoire et tables de vérité
- logique séquentielle
- les bascules de type D

1) LE MICROPROCESSEUR

1.1) Historique

Le premier microprocesseur (Intel 4004) a été inventé en 1971. C'était un processeur ayant un bus de données de 4 bits. Sa fréquence de travail était de 108 Khz et il comportait 2300 transistors. Avec l'évolution de la technologie et l'arrivée des circuits sub-microniques (tracés inférieurs au micron), les performances des circuits électroniques ont décuplé tant par leur vitesse de traitement que par leur niveau d'intégration. Le tableau ci-dessous donne un aperçu de cette évolution (source INTEL).

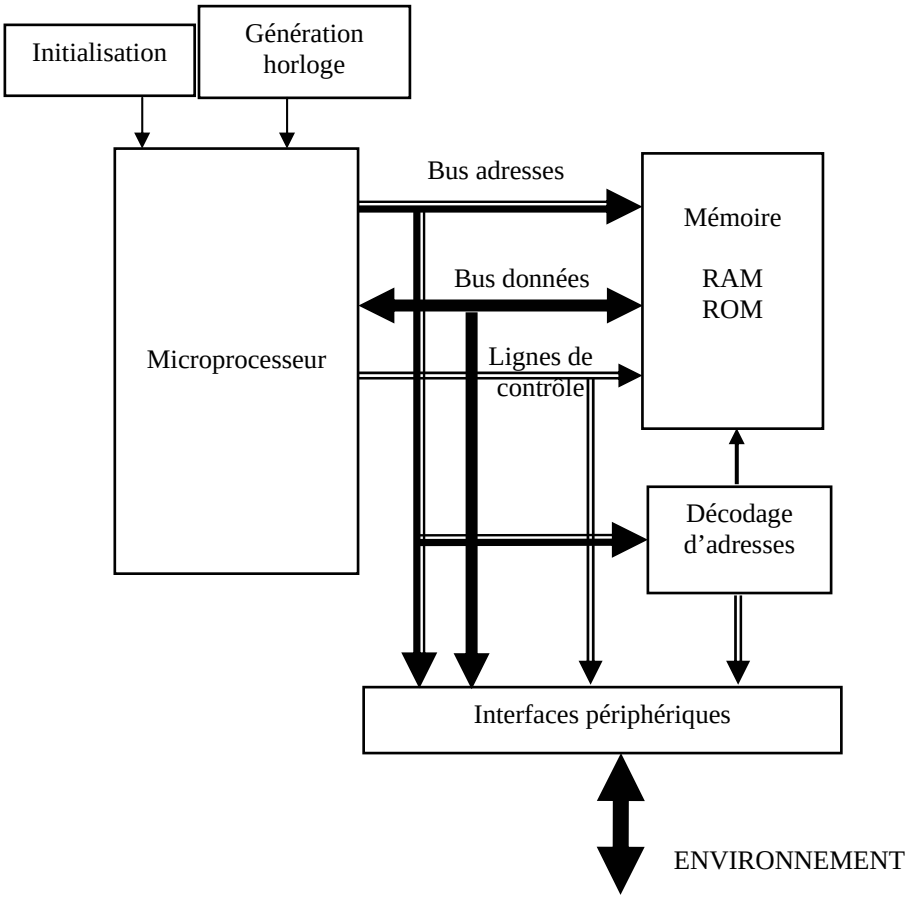
année	référence	Bus données	fréquence	Nbre de transistors
1971	4004	4	108 Khz	2300
1974	8080	8	2 Mhz	4500
1978	8086	16	10Mhz	29 000
1985	386	32	40Mhz	275 000
1993	Pentium	32	75Mhz	3.5 M
1995	Pentium Pro	32	150Mhz	5.5 M
1997	Pentium II	32	233Mhz	7.5 M
1999	Celeron	32	850Mhz	9.5 M
2000	Pentium IV	64	1.5 Ghz	55 M
2004	Pentium IV série 600	64	3.73 Ghz	169 M

1.2) Domaines d'utilisation

De part leur facilité de mise en œuvre et leur coût de revient très bas les microprocesseurs (microcontrôleurs) sont utilisés dans de nombreux domaines :

- Automobile (ex : Volvo S80 équipée de 11 µC/µP)
- Aéronautique (ex : calculateur de vol Airbus A320)
- Electroménager (four électrique, micro-onde, lave-linge, …)
- Machines outils
- Mesure et régulation (oscilloscopes numériques, multimètres, PID, …)
- Stations de travail (PC, station SUN, …)

1.3) ARCHITECTURE D'UN SYSTEME A µP (CONFIGURATION MINIMUM)



L'architecture matérielle minimum d'un système à microprocesseur est représentée sur la figure précédente. Celle-ci comporte :

- le microprocesseur
- un circuit d'initialisation
- un générateur d'horloge
- une mémoire à lecture seule (ROM)
- une mémoire à lecture/écriture (RAM)
- un dispositif de décodage d'adresses
- des interfaces de périphériques.
- des bus de communication

1.3.1) le microprocesseur

C'est un **automate séquentiel** dont le rôle est la **lecture, le décodage** puis **l'exécution** des instructions présentes en mémoire (ROM ou RAM) et qui constituent le **programme.** Son fonctionnement sera vu au chapitre 1.3.

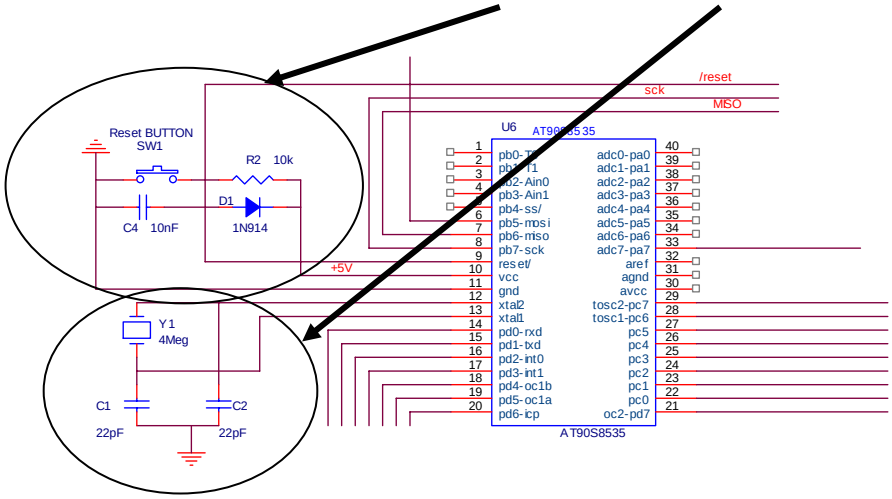
1.3.2) le circuit d'initialisation (reset)

A la mise sous tension, il empêche le démarrage du microprocesseur tant que les alimentations en énergie ne sont pas stabilisées . En général le circuit de « reset » bloque le microprocesseur quelques millisecondes à partir de sa mise sous tension (cette durée dépend en fait du type de µprocesseur et est donnée par la documentation du constructeur).
En fonctionnement normal, il permet de réinitialiser le système.
Généralement le signal d'initialisation (reset) est obtenu à partir d'un circuit électrique mettant en œuvre un réseau RC. Le choix des valeurs de R et C permet de déterminer la durée à l'état actif du signal. On trouve également dans le commerce beaucoup de circuits spécialisés assurant cette fonction (ex : Motorola MC 34064, Maxim Max-6475, …).

1.3.3) le générateur d'horloge

C'est lui qui assure le cadencement de l'exécution des instructions. Plus la fréquence d'horloge est élevée, plus les instructions seront exécutées rapidement. La puissance d'un microprocesseur s'exprime en MIPS (Millions d'instructions par seconde) mais cette grandeur est très insuffisante pour estimer les performances d'un système.
Nota : dans les systèmes récents dont la fréquence de travail est très élevée, l'horloge est obtenue en multipliant la fréquence de l'horloge de base (issue d'un quartz). Cette opération est réalisée par une boucle à verrouillage de phase implantée sur la puce du processeur.

Exemple de circuits de reset et de génération d'horloge



1.3.4) la mémoire ROM

C'est une mémoire à **lecture seule** (Read Only Memory). Dans les microsystèmes, c'est elle qui en général contient le programme à exécuter. Dans les systèmes plus complexes (stations de travail, PC, ..) le programme, stocké en mémoires de masse (disque dur, CD-rom, …), est chargé dans la mémoire vive (RAM) avant d'être exécuté.

Fonctionnellement la mémoire ROM a pour rôle de fournir sur son « buffer » de sortie un mot binaire correspondant au contenu de la case pointée par l'adresse d'entrée. Une mémoire est spécifiée par sa capacité et son organisation.

Exemple :

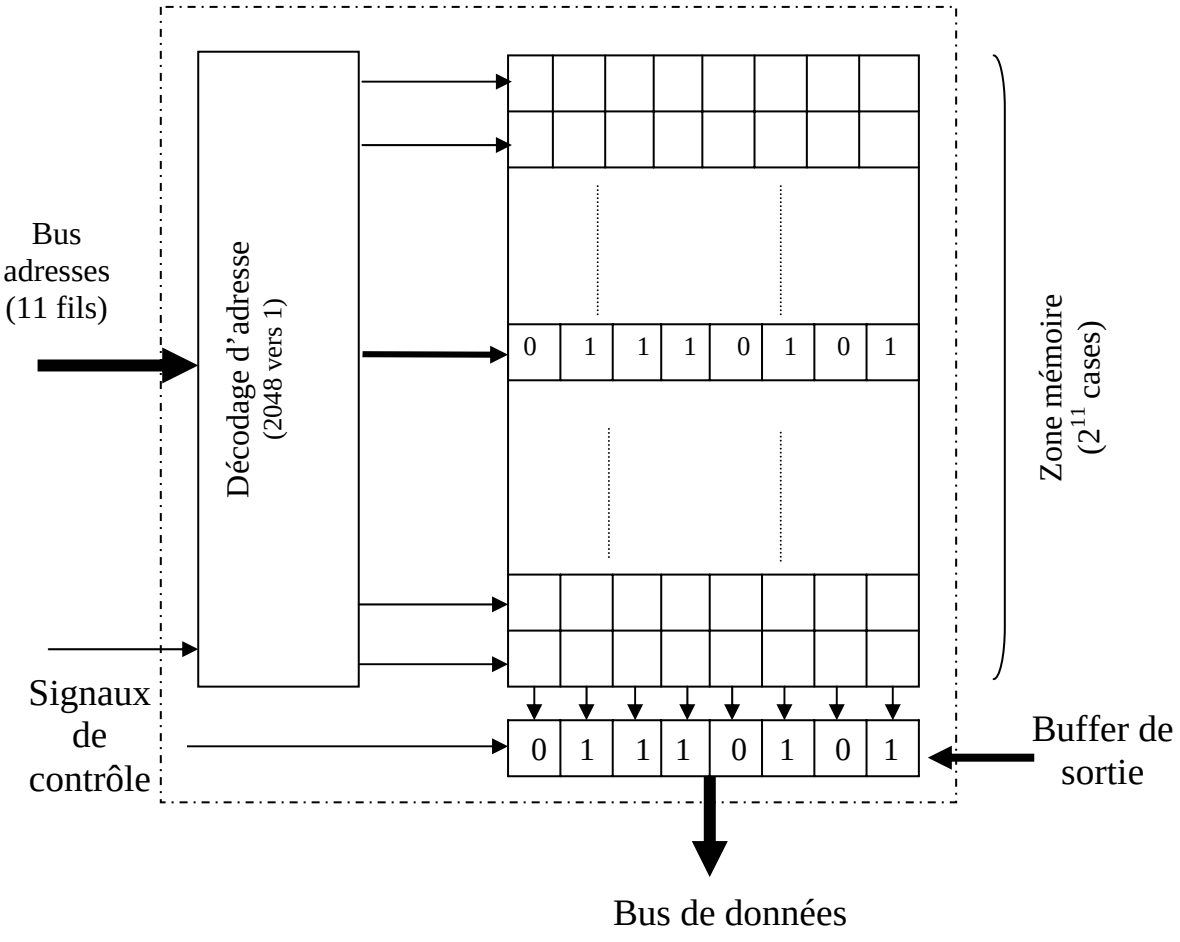
- une mémoire de 32K * 8 signifie que celle-ci contient 32K mots de 8 bits. Pour adresser toutes les cases de cette mémoire, le bus d'adresse devra avoir une largeur de 15 bits (2^{15} combinaisons soit 32768).
-

Nota : 1Koctets = 1024 octets

Remarque :

La mémoire ROM conserve son contenu lorsque son alimentation est coupée.

Ex : ARCHITECTURE INTERNE d'une MEMOIRE ROM de 2Ko



1.3.5) La mémoire RAM

La mémoire RAM (Random Access Memory) est une mémoire à accès aléatoire qui peut être **lue** ou **écrite** indéfiniment. Dans les microsystèmes, elle est souvent réservée pour stocker des

variables, des résultats intermédiaires et sert également de « pile » pour le processeur. Comparativement à la ROM, le buffer de sortie est **bidirectionnel** et c'est le signal de lecture/écriture issu des lignes de contrôle qui fixe le sens de circulation des données.

Remarque :
La mémoire RAM perd son contenu lorsque son alimentation est coupée.

1.3.6) Les bus de communications

- le bus d'adresses

Ce sont des lignes physiques (équipotentielles) qui véhiculent les adresses générées par le microprocesseur. La taille du bus fixe sa capacité d'adressage. Dans les systèmes monoprocesseurs et selon leurs types, le bus d'adresses est souvent unidirectionnel.

Exemples :
Motorola 6800 : bus adresses 16 bits => 65536 adresses possibles (2^{16} combinaisons).
Motorola 68000 : bus adresses 20 bits => 1 Million adresses possibles (2^{20} combinaisons).
Intel 80386 : bus adresses 32 bits => 4 G adresses possibles (2^{32} combinaisons).

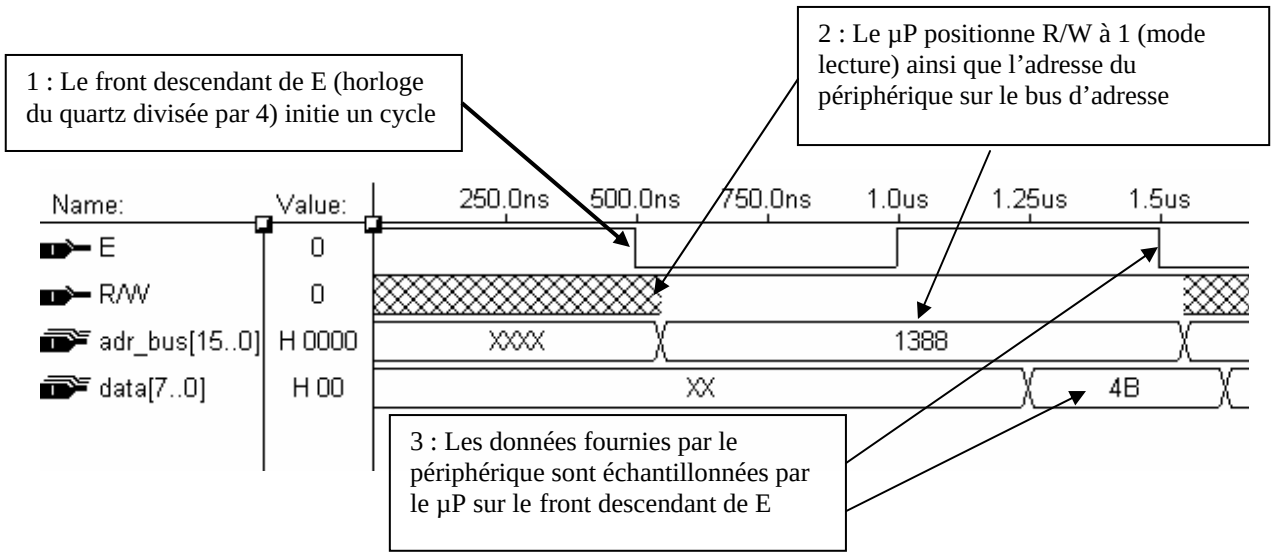
- le bus de données

Ce sont des lignes physiques (équipotentielles) qui véhiculent les données. Ce bus est bidirectionnel (les données peuvent entrer ou sortir du µP suivant qu'il est en mode lecture ou écriture). La taille de ce bus correspond à la taille du bus de données interne au µP.

Exemples :
Motorola 6800 : bus de données 8 bits.
Motorola 68000 : bus de données 16 bits.
Motorola DSP 56000 : bus de données 24 bits.
Motorola 68020 : bus de données 32 bits.

- Le bus de contrôle

Il est de taille variable suivant la complexité du processeur. C'est lui qui synchronise les échanges entre le processeur et ses périphériques. Le chronogramme ci-dessous illustre un cycle de lecture d'un périphérique du microprocesseur Motorola 6809.



1.3.7) Les interfaces de périphériques

Ce sont souvent des circuits intégrés spécialisés directement interfaçables avec le microprocesseur et qui assurent les liens avec les unités matérielles du système.

Exemples :

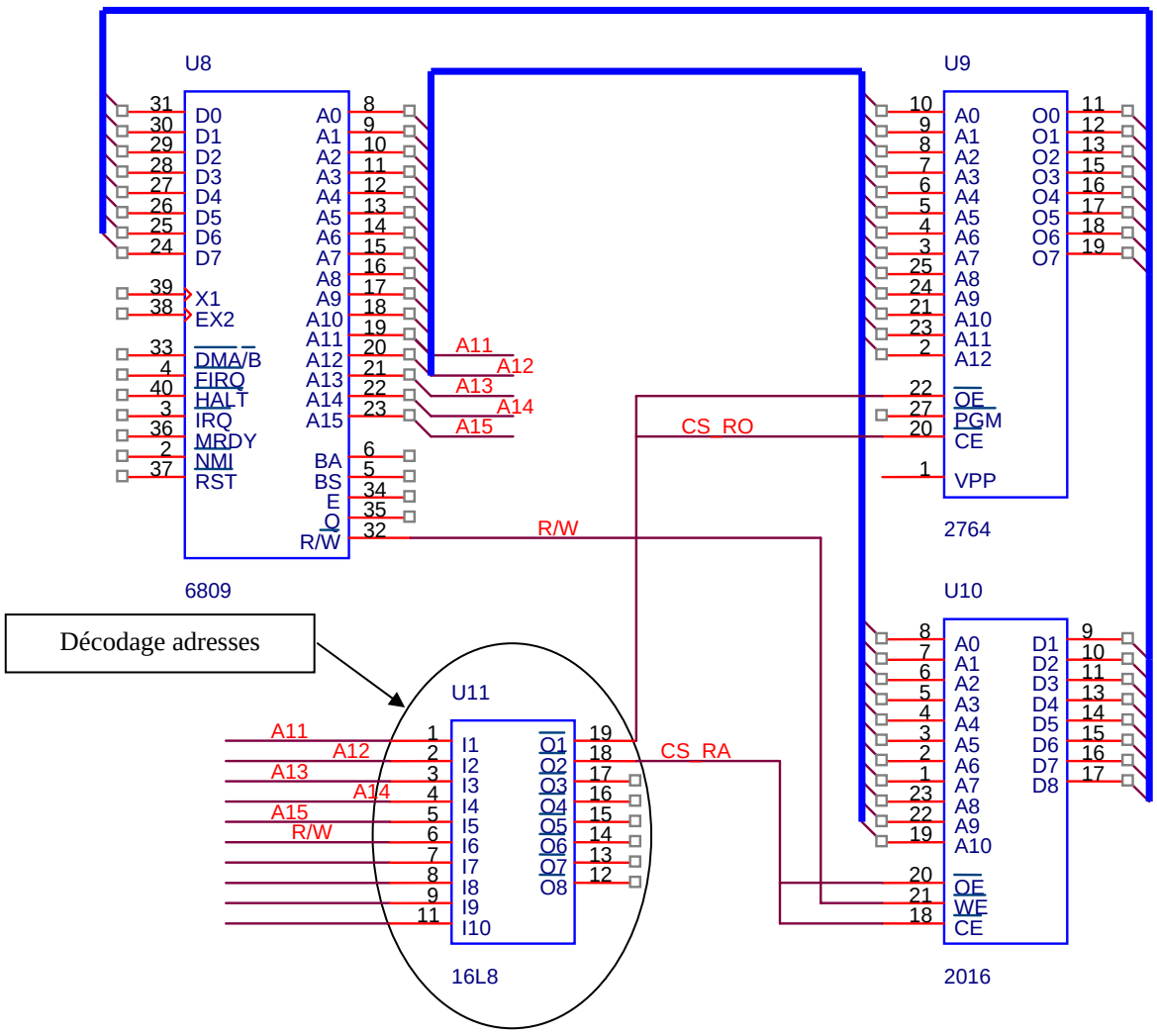
- interface IDE pour lecteur de disquettes
- interface SCSI pour disque dur
- entrées/sorties numériques et analogiques
- interfaces vidéo, claviers, bus USB
- etc…

Chaque interface dispose d'un champ adresse qui lui est propre. Ce champ adresse est fourni par **le bloc de décodage d'adresse**.

1.3.8) Le décodage d'adresse

Il a pour but d'**assigner** à chaque bloc mémoire et aux différents périphériques qui composent le système un emplacement et un espace dans le champ adressable par le microprocesseur.

DECODAGE D'ADRESSES



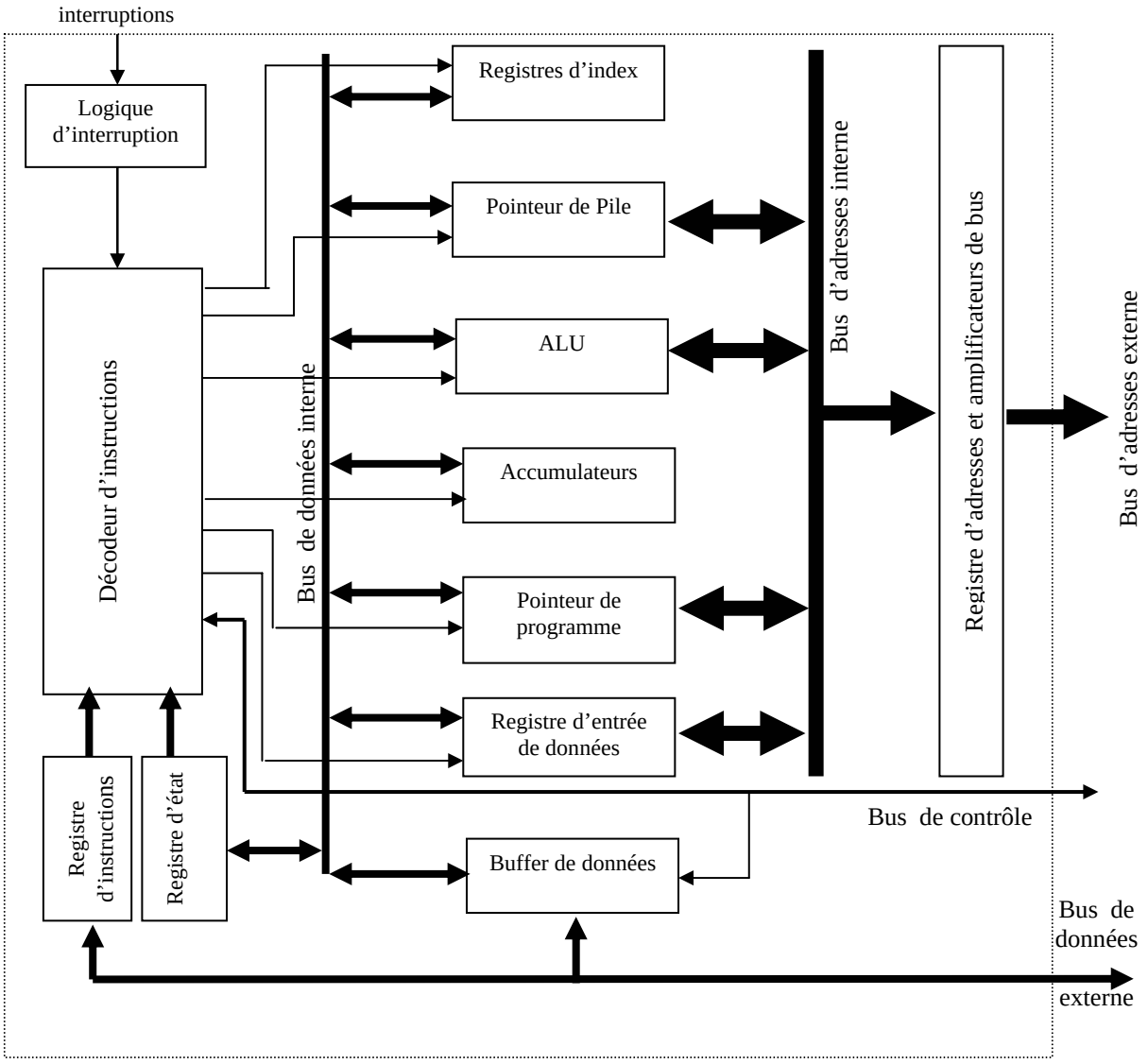
Exercice :

Déterminer les équations de CS_RAM et CS_ROM pour que les mémoires RAM et ROM débutent respectivement aux adresses :

- $(A000)_H$ pour la ROM (circuit U9)
- $(5000)_H$ pour la RAM (circuit U10)

Quel est l'espace occupé par chaque mémoire. Donner l'adresse de fin correspondante. (rep : BFFF pour la ROM et 57FF pour la RAM).

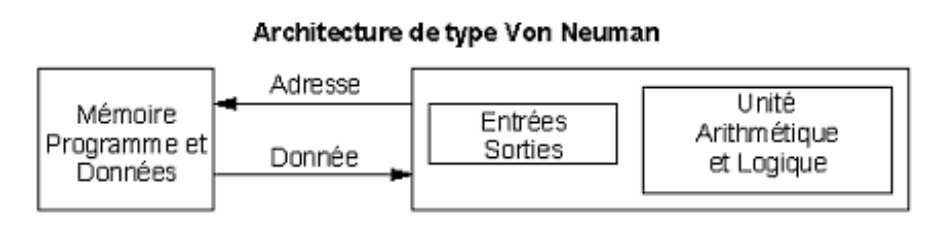
1.4) ARCHITECTURE INTERNE D'UN MICROPROCESSEUR



L'architecture interne du microprocesseur ci-dessus est une architecture de type « Von Neuman [(1)] » (**un seul bus de données**). Elle se compose essentiellement :

- D'une unité de décodage d'instructions

- D’un ensemble de registres
- D’amplificateurs de lignes (buffers de données et d’adresses)
- D’une unité arithmétique et logique.

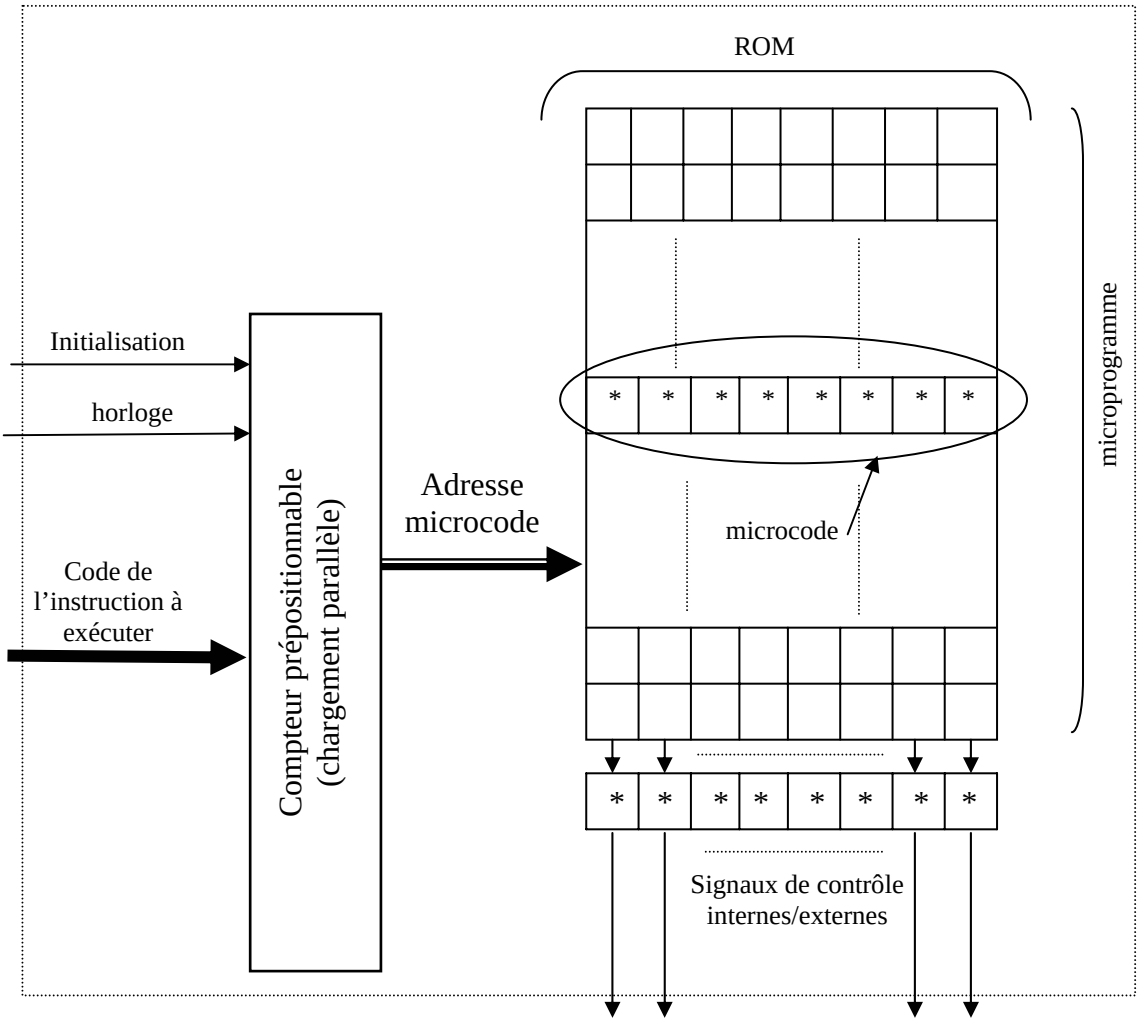


(1) Von Neuman : Mathématicien américain (Budapest 1903-Washington 1957)

1.4.1) L’unité de décodage d’instructions

L’unité de décodage d’instructions a pour rôle de générer en séquence les différents signaux de fonctionnement internes et externes au µP correspondant à l’instruction à traiter. Le principe peut être vu à partir du synoptique suivant (mais d’autres types de réalisation sont possibles) :

ARCHITECTURE DU DECODEUR D’INSTRUCTION



Nota :

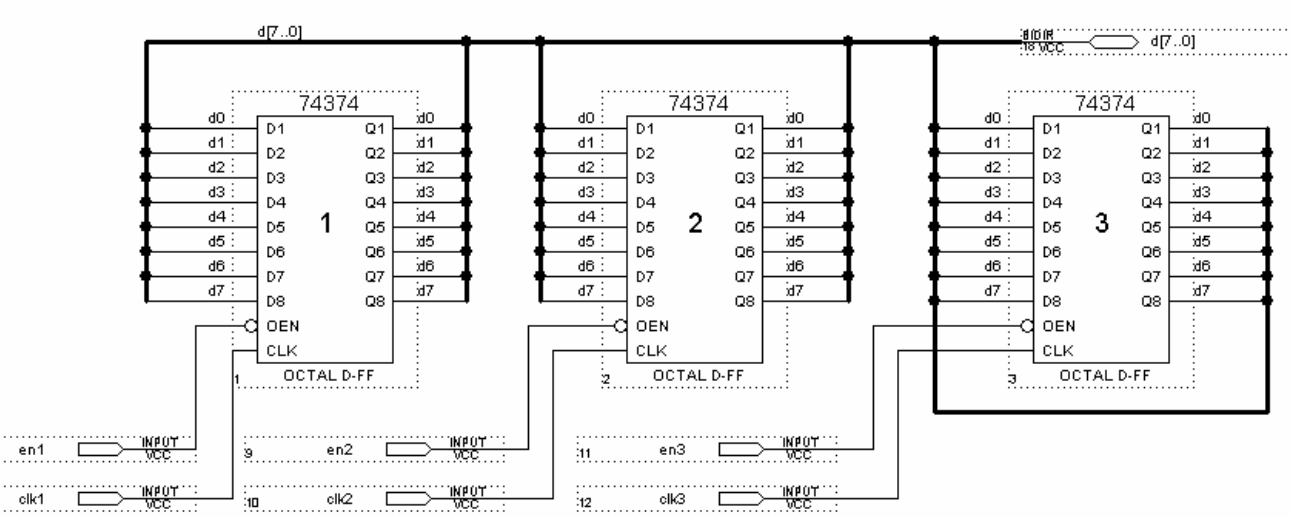
Dans ce dessin la taille de la mémoire des microcodes a été ramenée à 8 bits par simplification. Cette taille est en fait très variable et dépend de la complexité du microprocesseur.

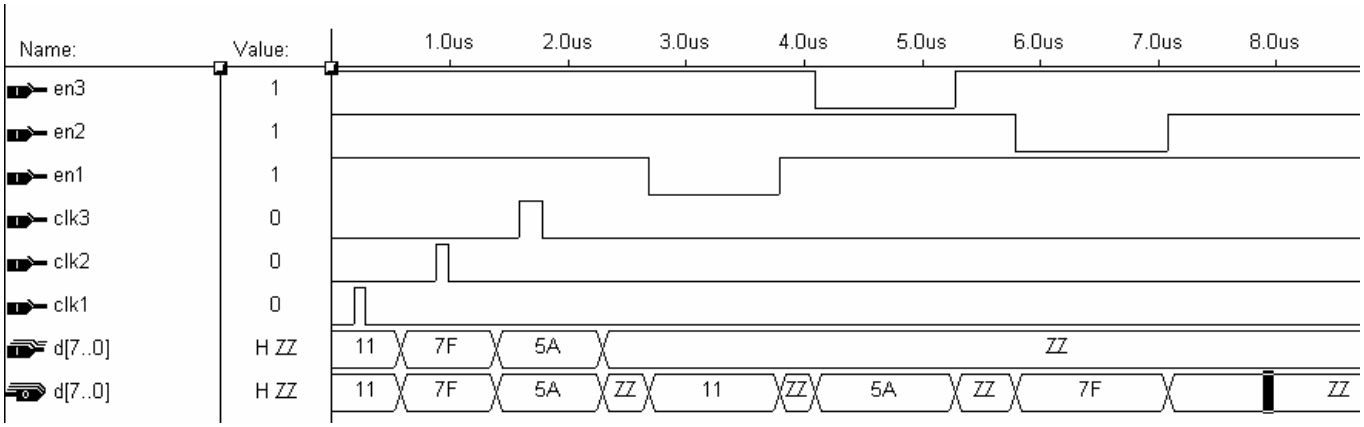
Fonctionnement :
Le code de l'instruction à exécuter est préchargé dans un compteur prépositionnable. Celui-ci adresse par conséquent une zone de la ROM interne du µP dans laquelle se trouve le microprogramme associé à cette instruction. Ce microprogramme est une suite de microcodes correspondant directement aux signaux à générer. Ainsi le séquencement des signaux est obtenu par incrémentation du compteur à partir de l'horloge qu'il reçoit. A la fin de l'exécution de l'instruction, un signal vient repositionner le compteur à une adresse correspondant à un cycle de recherche de nouvelle instruction.

1.4.2) Les registres

Les registres sont réalisés à partir de bascules type D ayant des sorties « haute impédance ». Leur rôle est de mémoriser (stocker) à des instants précis les données qui circulent sur le bus. Les signaux qui contrôlent les registres (horloge, commande « haute impédance ») sont générés par l'unité de décodage d'instruction.

Le schéma qui suit en illustre le principe.





Les signaux référencés **clk_i** assurent, sur front montant, l'écriture dans le registre **i** de la valeur présente sur le bus à cet instant.

Les signaux référencés **en_i** permettent, lorsqu'ils sont à **1,** la déconnexion du registre **i** du bus. En conséquence deux signaux ou plus ne doivent pas être mis à **0** simultanément sous peine de mettre en court circuit les étages de sortie des registres.

Dans l'exemple présenté, le registre 1 se charge à la valeur « 11 », le registre 2 à la valeur « 7F » et le registre 3 à la valeur « 5A ».

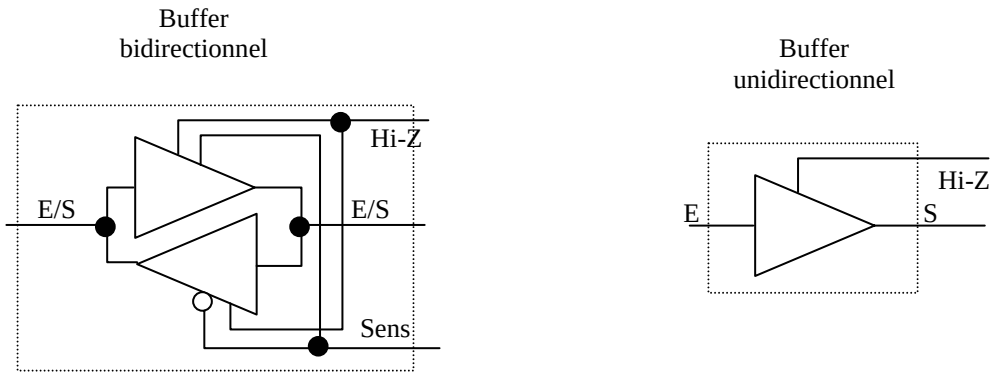
Si on connecte le registre 1 au bus par le biais du signal « en1 » on trouve alors sur le bus la valeur mémorisée « 11 » puis en répétant l'opération pour le registre 3 on trouve « 5A » et « 7F » pour le registre 2.

C'est la seule méthode de circulation de l'information à ce jour entre un processeur et ses périphériques directs.

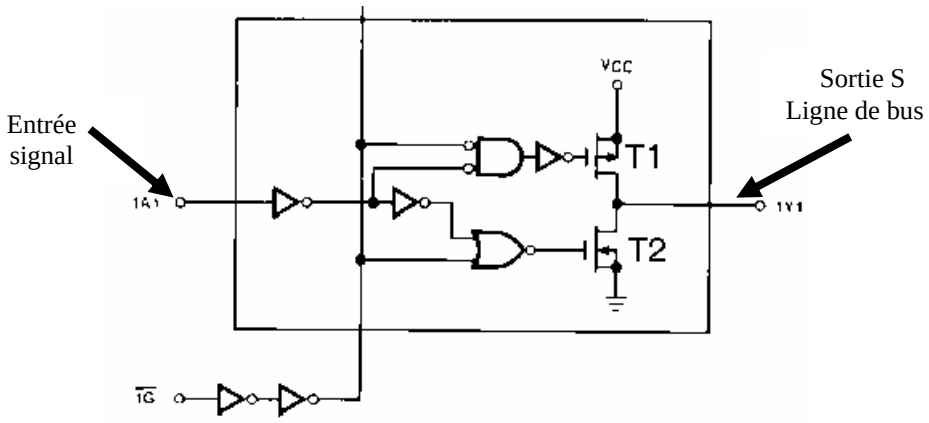
(« périphérique » est pris ici au sens large et peut signifier également un autre processeur ou de la mémoire).

1.4.3) Les amplificateurs de ligne

Ces circuits sont aussi appelés des « buffers ». Ils ont pour rôle de fournir le courant nécessaire à la charge que représente le bus. Ils disposent en général d'une commande qui permet de mettre leur sortie à haute impédance (Hi-Z).
Lorsque les bus sont de type bidirectionnels, une commande permet de choisir le sens de transfert des données.



Principe de la mise à haute impédance (buffer de ligne type 74HC244) :

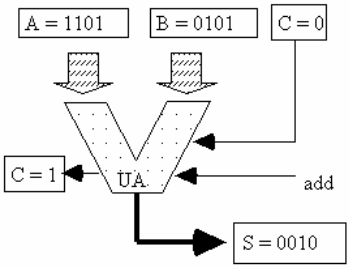


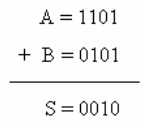
- **T1 ON et T2 OFF** : Le circuit force la ligne de bus à **1 (Vcc)**
- **T1 OFF et T2 ON** : Le circuit force la ligne de bus à **0 (Gnd)**
- **T1 OFF et T2 OFF** : La ligne de bus est ouverte ou « à haute impédance » **(Hi-Z).**

1.4.4) L'unité arithmétique et logique

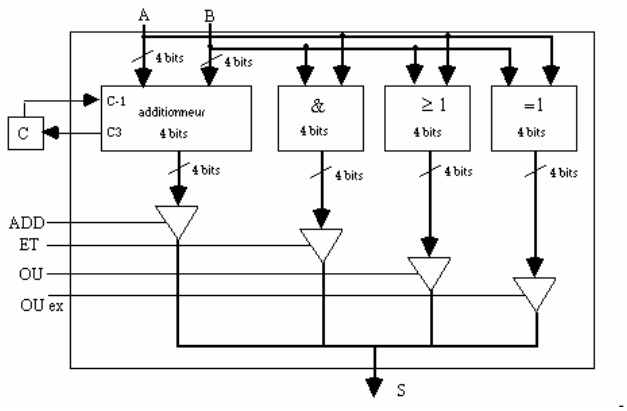
L'unité arithmétique et logique a pour but l'exécution d'opérations élémentaires sur les données. Les opérations de base supportées sont :

- l'addition
- la soustraction
- la complémentation
- le ET , le OU, le XOR logiques …





Exemple : addition (sur 4 bits)



Synoptique de réalisation matérielle

1.5) Architectures RISC et CISC

Les premières générations de microprocesseurs étaient à architecture CISC (Complex Instructions Set Computer) ce qui nécessitait plusieurs cycles d'horloge de base (fournie par le quartz) pour exécuter une instruction. Au fil du temps les concepteurs de circuits se sont aperçus que 80% des traitements effectués faisaient appel à 20% des instructions du µP. L'idée est donc venue de développer des microprocesseurs possédant un jeu d'instructions

réduit (RISC : Reduced Instruction Set Computer) mais exécutable dans un temps très court (souvent en un seul cycle d'horloge). Ces microprocesseurs sont souvent à architecture de « **Harvard** » et disposent d'un mode de fonctionnement en « **pipeline** ». Ces notions seront vues au chapitre 3.
Aujourd'hui les deux types de processeurs cohabitent et leur utilisation est fonction des applications visées (station de travail, serveur, système temps réel …).
Notons toutefois que si les architectures RISC sont plus simples à réaliser sur le plan matériel, les programmes assembleurs et les compilateurs associés sont plus compliqués à réaliser.

Exemple :

1) instruction d'addition de 2 registres (micro RISC Atmel AT90S8535)

   ADD Rd, Rr Rd <= Rd + Rr nécessite un cycle d'horloge (125ns pour une horloge à 8Mhz).

2) instruction d'addition de 2 registres (micro CISC Motorola 68HC11)

   ABA A <= A + B nécessite deux cycles d'horloge (250ns pour une horloge à 8Mhz).

	RISC	CISC
Avantages	Temps d'exécution court, hardware plus simple, coût puce électronique réduit	Instructions puissantes, programme plus simple à écrire, moins de lignes de programme, compilateur simple à concevoir
Inconvénients	Programme plus complexe à écrire, plus de lignes, compilateur difficile à concevoir	Temps exécution long, hardware plus compliqué et plus coûteux

2) LES MICROCONTROLEURS

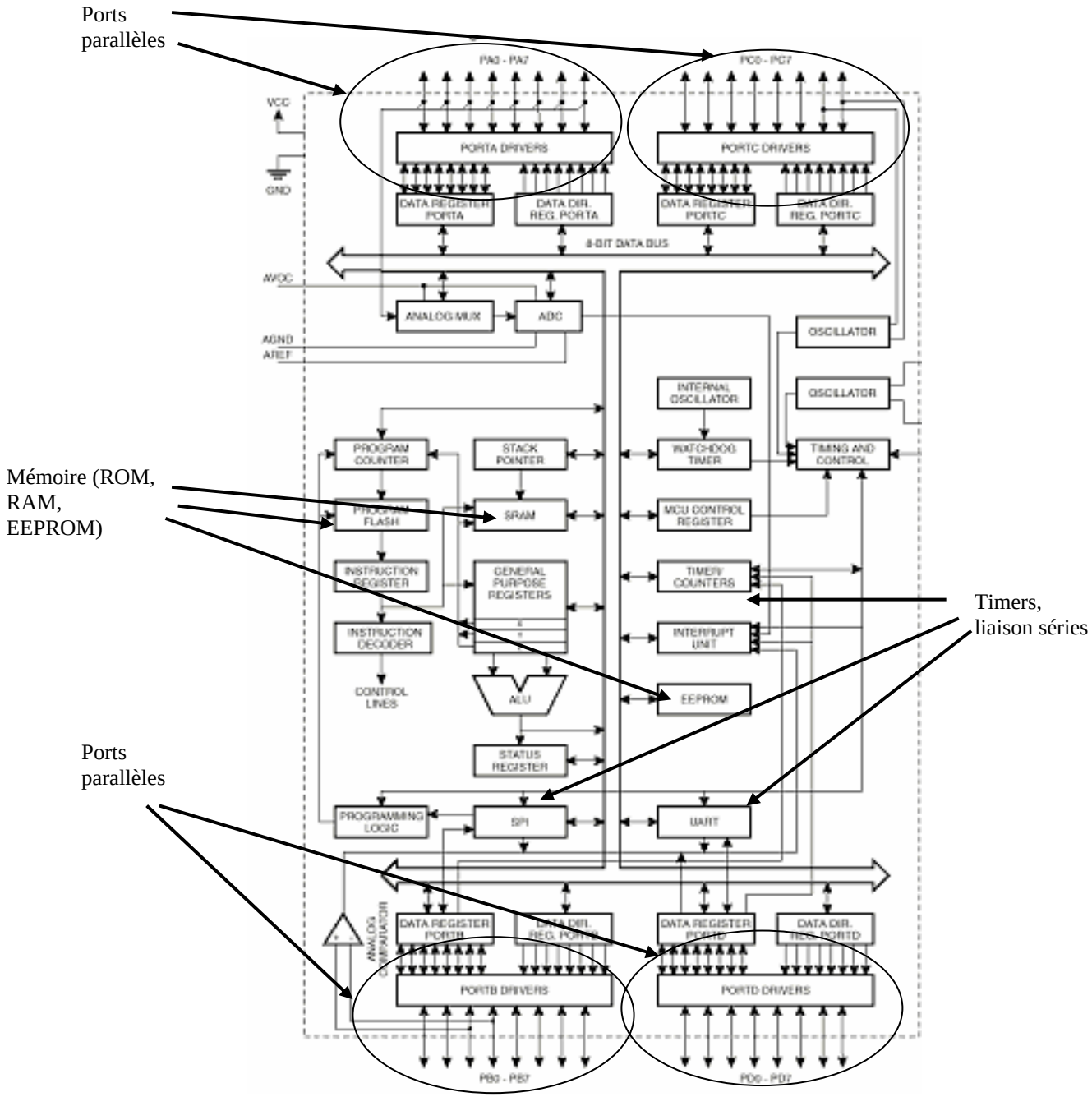
L'apparition des microcontrôleurs résulte de l'évolution des niveaux d'intégration des transistors (des portes) dans les circuits intégrés. Il est devenu possible de rajouter sur la puce de silicium dédiée au processeur un ensemble de ressources telles que :

- de la mémoire ROM, EPROM, EEPROM, FlashEPROM
- de la mémoire SRAM
- du décodage d'adresse
- quelques périphériques (entrées-sorties numériques, liaison série RS232C, timers…)

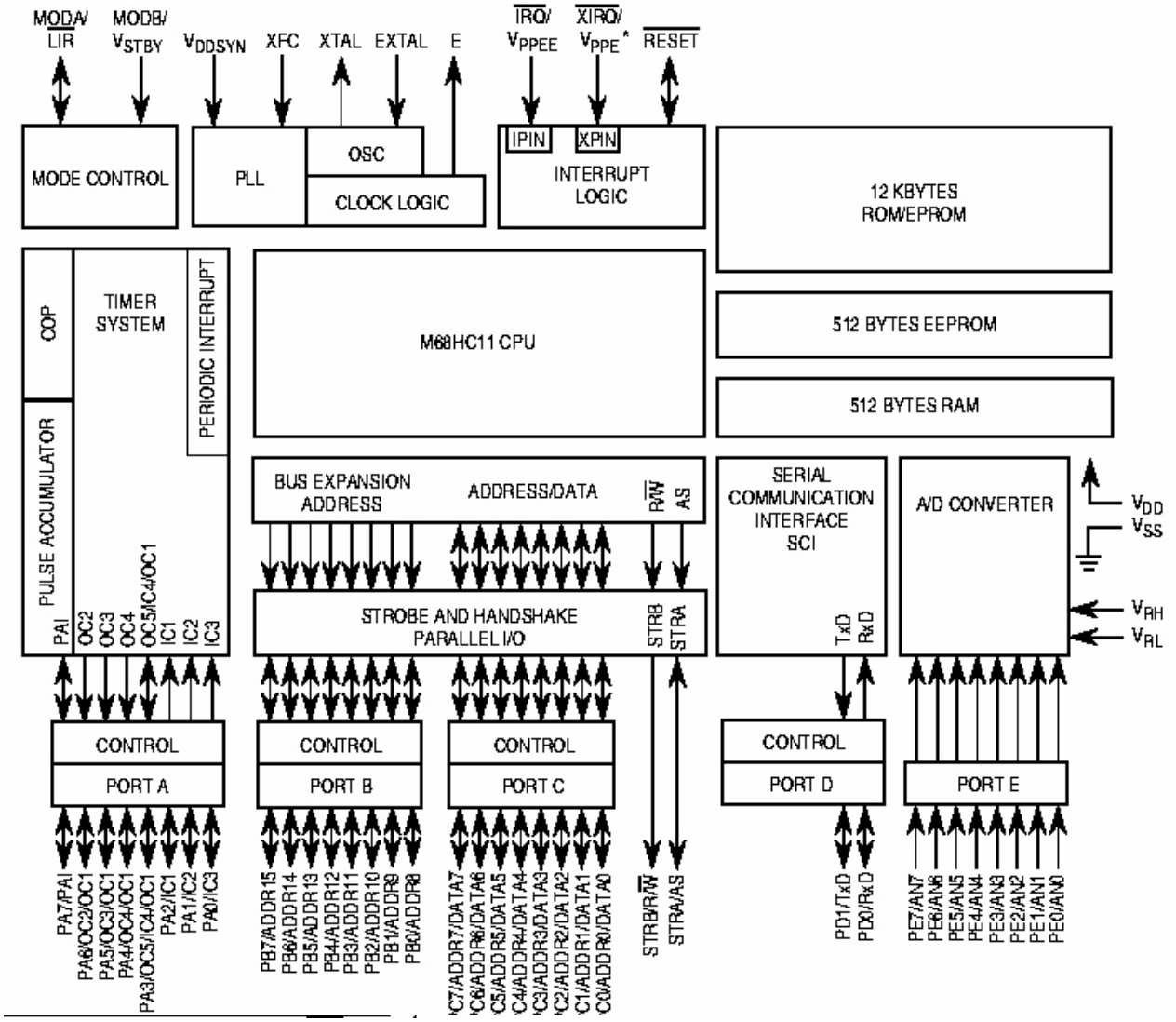
Les microcontrôleurs récents intégrent des périphériques plus complexes tels que:

- des convertisseurs N/A et A/N
- des interfaces bus I2C, bus CAN, VAN, USB …
- des interfaces type réseaux éthernet…
- des contrôleurs de mémoire dynamique…

Exemple d'architecture interne de microcontrôleur (Atmel AT90S8535) :



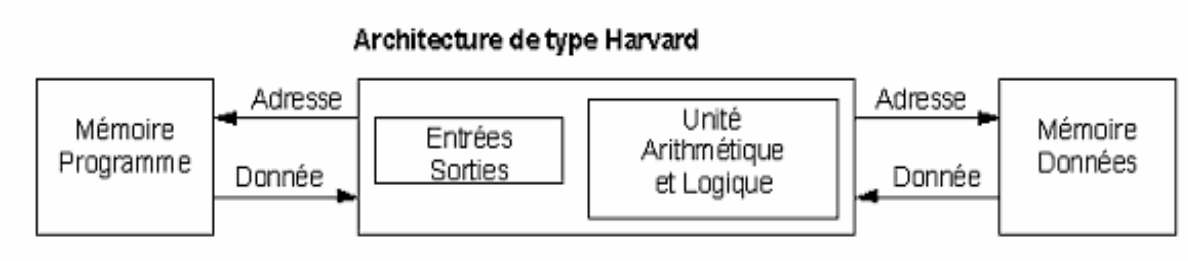
Autre exemple : microcontrôleur motorola 68HC11



3) Les DSP (Digital Signal Processor)

Ce sont des microprocesseurs/microcontrôleurs dont l'architecture interne a été adaptée pour le traitement du signal. Ils sont en général basés sur une architecture de **HARVARD** qui fait apparaître des bus de programme et de données distincts. Ils disposent également de la multiplication-accumulation câblée et les instructions sont essentiellement exécutées en un cycle d'horloge (mode pipeline). Certains DSP (DSP56001 de Motorola par exemple) peuvent exécuter, suivant le cas, jusqu'à 3 instructions simultanément. Un certain nombre de ressources et de périphériques sont intégrés sur la même puce afin d'optimiser les temps calcul. On trouve ainsi :

- De la mémoire programme (ROM ou RAM) à accès rapide (10 à 15 ns)
- De la mémoire données (ROM ou RAM) à accès rapide souvent scindée en deux banques pour traiter les nombres complexes.
- Un accès DMA (Direct Memory Access) pour le transfert rapide de données
- Un mode d'adressage par inversion de bits, servant à réorganiser les échantillons de sortie de Transformées de Fourier Rapides (TFR ou FFT).
- Une architecture à pipe-line permettant de paralléliser le maximum d'opérations élémentaires (préchargement d'instruction, chargement des données, exécution d'opérations arithmétiques/flottantes, stockage des résultats)
- Des entrées-sorties similaires à celles d'un microcontrôleur (liaison série SPI, RS232C, timers, CNA et CAN …).



Exécution des instructions sans pipe-line :

	Clock cycle							
	1	2	3	4	5	6	7	8
Instruct. fetch	I1				I2			
Decode		I1				I2		
Data read/write			I1				I2	
Execute				I1				I2

Sans pipe-line, il faut 4 cycles d'horloge pour exécuter une instruction.

Et avec pipe-line :

	Clock cycle							
	1	2	3	4	5	6	7	8
Instruct. fetch	I1	I2	I3	I4	I5	I6	I7	I8
Decode		I1	I2	I3	I4	I5	I6	I7
Data read/wite			I1	I2	I3	I4	I5	I6
Execute				I1	I2	I3	I4	I5

Il faut en moyenne un cycle d'horloge pour exécuter une instruction. Cependant cette architecture rend la gestion des sauts de programmes plus compliquée.

Les DSP se partagent en deux grandes familles :

- Les DSP à virgule fixe
- Les DSP à virgule flottante.

Ces derniers, plus coûteux, ne nécessitent pas de gérer les débordements d'accumulateurs lors des calculs arithmétiques et sont par conséquent plus simples d'emploi.

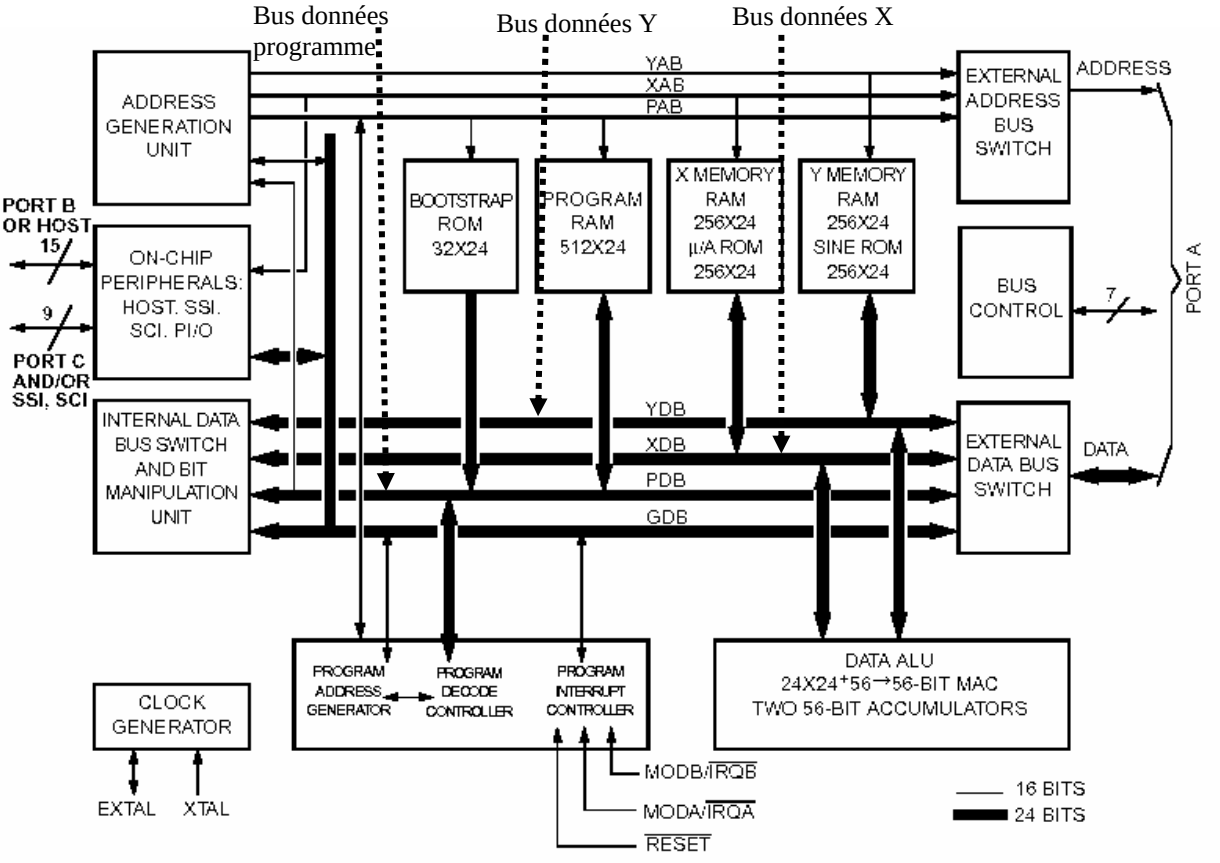


Figure 1. DSP56001 Block Diagram

4) LES METHODES ET OUTILS DE DEVELOPPEMENT

Compte tenu de la complexité des microprocesseurs, microcontrôleurs et DSP, le matériel courant de laboratoire n'est plus adapté pour la mise en oeuvre de tels composants. Un certain nombre d'outils plus ou moins performants s'avèrent indispensables pour effectuer la mise en œuvre du circuit voire tester le bon fonctionnement de la carte électronique réalisée. Parmi ces outils on peut citer :

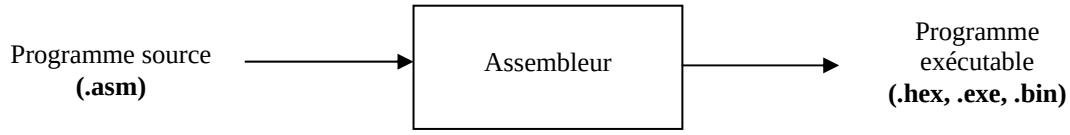
- les langages de programmation
- les simulateurs logiciels
- les émulateurs « in circuit »
- les émulateurs « temps réel » à mémoire trace
- les émulateurs « low cost »

4.1) Les langages de programmation

Les µP/µC/DSP sont des composants qui exécutent un programme (ces programmes sont constitués de suites d'instructions). Se pose alors le problème du choix du langage de programmation : langage de bas niveau (assembleur) ou langage de haut niveau (C, Basic, Pascal,…).

4.1.1) L'assembleur

Le langage de bas niveau (assembleur) est un langage qui **est directement compréhensible et exécutable** par le processeur. Pour des commodités d'utilisation, le programmeur manipule des instructions représentées par des **mnémoniques**. A chaque mnémonique correspond un code hexadécimal. C'est le rôle du **programme assembleur** de transformer le programme écrit en mnémonique en programme hexadécimal (ou binaire) directement exécutable.

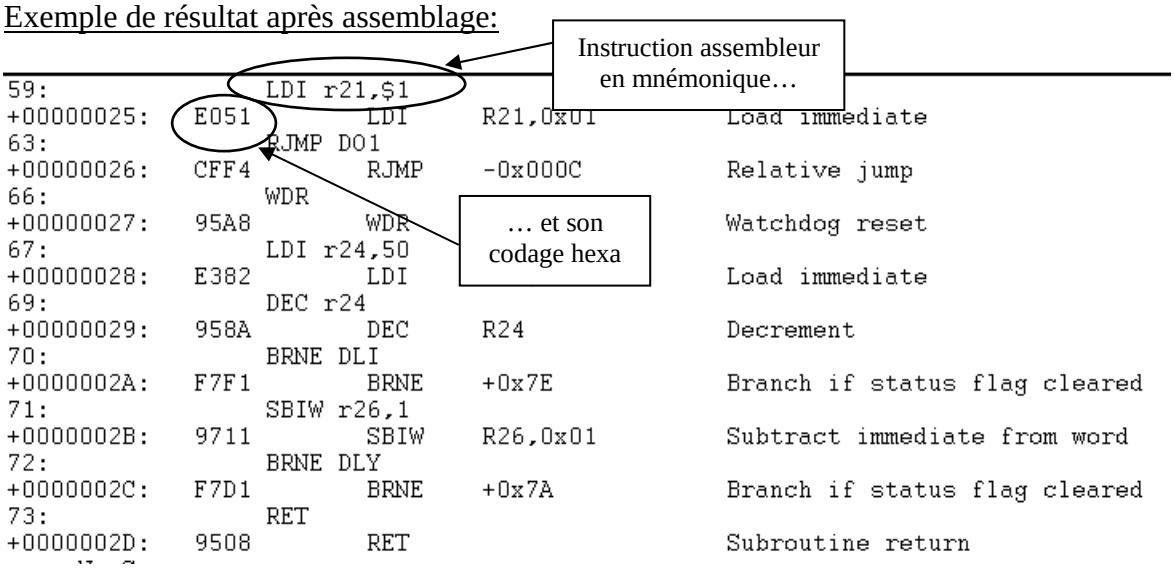


Exemple de programme assembleur source (micro Atmel AT90S8535) :

```
debut:      clr temp          ;temp=0
            clr th            ;th=0
            clr NBR           ;NBR=0
            clr cycle         ;cycle=0

            ldi cycle,50      ;mesure sur 50 périodes
att1:       in  temp,signal   ;test si le signal est au nvl 1
            andi temp,1
            cpi temp,0
            breq att1         ; si le nvl est 0 alors attente
incr1:      inc th            ; sinon on incrémente th
            in temp,signal    ;tant que le signal est au nvl 1, on incrémente th.
             andi temp,1
            cpi temp,1
            breq incr1
            …
```

Exemple de résultat après assemblage:



4.1.2) Les langages de haut niveau

Ils permettent de s'affranchir de la connaissance du langage assembleur qui est spécifique à une famille de µP/µC. Ils ne nécessitent pas la connaissance approfondie de l'architecture interne du microcontrôleur/microprocesseur utilisé (registres, ressources diverses…). C'est le **compilateur** qui se charge de **traduire** les instructions de haut niveau en instructions assembleur puis en code machine. La connaissance intime du processeur est laissée à ceux qui développent les compilateurs.

Nota:
Lorsque le compilateur est exécuté sur une machine hôte dont le processeur est différent de la machine cible, on parle de cross-compilateur (compilateur croisé).

4.1.3) Avantages et inconvénients des langages assembleurs et des langages évolués

ASSEMBLEUR	LANGAGE DE HAUT NIVEAU
Inconvénients: - souvent spécifique au µP utilisé - nécessite la connaissance de l'architecture interne - maintenabilité difficile - évolutivité difficile - peu de pérennité des investissements - nécessite une formation spécifique Avantages: - code généré très compact - vitesse d'exécution du programme	Inconvénients: - code généré moins compact (dépend des performances du compilateur) - vitesse d'exécution plus lente Avantages: - langage commun indépendant du µP/µC - ne nécessite pas une connaissance forte de l'architecture du processeur - programmes plus facilement maintenables et évolutifs - meilleure pérennité des investissements (si changement de processeur) - apprentissage du langage aisé

CONCLUSION:

A l'exception des cas où une rapidité d'exécution ou compacité du code est recherchée, les développements se font autant que possible en langage de haut niveau (C, C^{++} principalement, éventuellement JAVA avec l'apparition récente de quelques compilateurs). Pratiquement on effectue un mixage des deux méthodes: modules assembleurs pour les parties requérant de la vitesse d'exécution et langage de haut niveau pour le reste du programme.
Il est à noter toutefois que pour avoir un bon niveau d'expertise des systèmes temps réel la connaissance de l'assembleur peut s'avérer incontournable.

4.2) Les simulateurs logiciels

Ce sont des programmes informatiques exécutés sur des stations de travail ou PC de bureau et qui permettent de simuler les instructions du programme et l'évolution de l'ensemble des ressources internes du composant.
Ces simulateurs ne permettent pas de faire de la simulation « temps réel » mais s'avèrent suffisants pour beaucoup d'applications. Il est possible toutefois de prendre en compte des événements extérieurs au composant en créant des fichiers de stimulis mais cette méthode reste fastidieuse.
Une fois le programme mis au point (simulé et testé), celui-ci est téléchargé dans l'application par le port série ou le port // du PC. D'autres types de liaisons peuvent être utilisées pour cette opération notamment la liaison « JTAG » qui se répend de plus en plus ou la liaison « BDM » propre à la société Motorola.

4.3) Les émulateurs « in circuit »

Ces émulateurs ont fait leur apparition assez récemment. Ils sont la conséquence de l'évolution rapide des niveaux d'intégration dans les circuits électroniques et du fait que les ressources des microcontrôleurs ne sont plus accessibles de l'extérieur. En effet, la partie

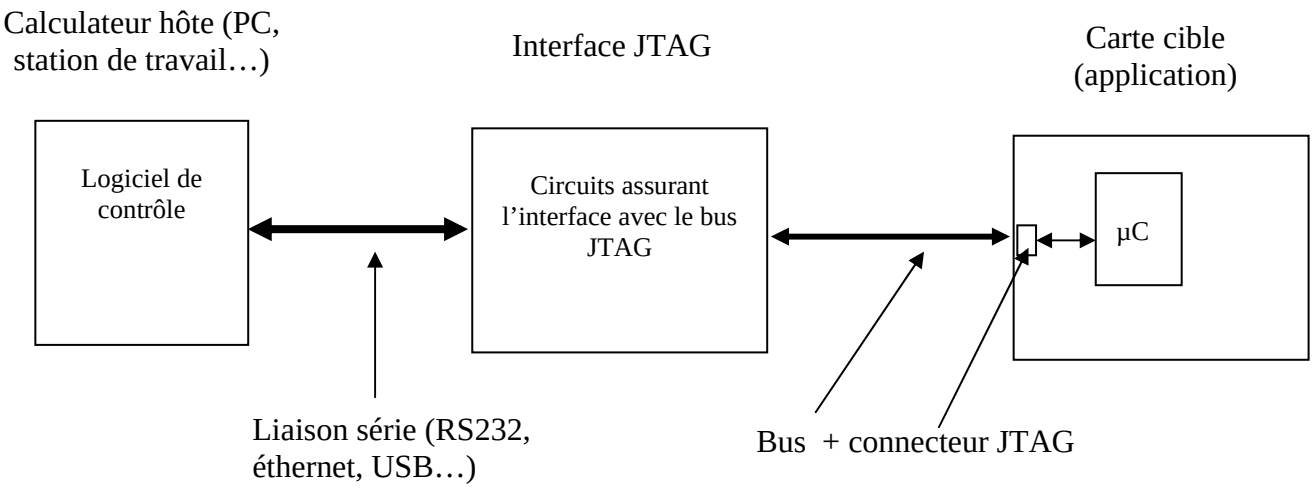
matérielle nécessaire à l'émulation des microcontrôleurs devenant peu importante en regard des ressources intégrées dans ces composants, il était intéressant que celle-ci soit également intégrée sur la même puce de silicium.

Le principe d'un émulateur « in circuit » est le suivant :

Le programme d'application est **téléchargé dans la mémoire du composant** (souvent de la FlashEprom) par une liaison spécialisée (JTAG, BDM ou autre). Si l'utilisateur a mis un point d'arrêt dans son programme**, l'adresse** de l'instruction qui suit le point d'arrêt est **stockée** dans un registre spécifique du composant puis **comparée** en temps réel à la valeur courante du compteur de programme. Lorsqu'il y a **égalité** entre les deux valeurs, le fonctionnement du microcontrôleur est stoppé et l'état des ressources internes est renvoyé au calculateur hôte par la liaison spécialisée pour affichage. Le programme s'exécute par conséquent sur **la machine cible** (et non sur la machine hôte comme dans le cas précédent (cf 4.1)) à la **vitesse réelle**.

Ce type d 'émulateur est un bon compromis entre les performances et son prix de revient. Il n'est pas disponible cependant sur les microcontrôleurs d'entrée de gamme.
Il est à noter également que toute modification du programme applicatif entraine un effacement et une reprogrammation de la mémoire Flash du composant, laquelle n'accepte pas indéfiniment des cycles d'effacement et de reprogrammation.

INTERCONNEXION d'un EMULATEUR JTAG
(Le µC est <u>présent</u> sur la carte cible)



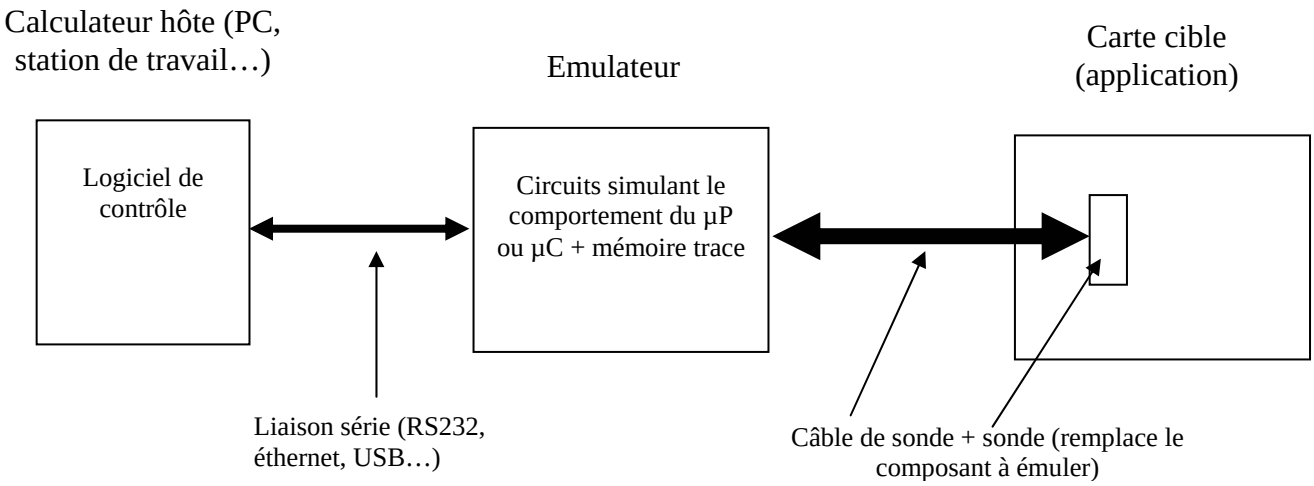
4.4) Les émulateurs « temps réel » à mémoire trace

A l'inverse des simulateurs logiciels, les émulateurs « temps réel » à mémoire trace sont des simulateurs **comportementaux matériels**. Il s'agit d'un ensemble de circuits électroniques (une version « éclatée » du µP/µC/DSP) ayant le même comportement que le circuit à émuler auxquels on a associé de la **mémoire trace** (RAM).

Mise en œuvre :

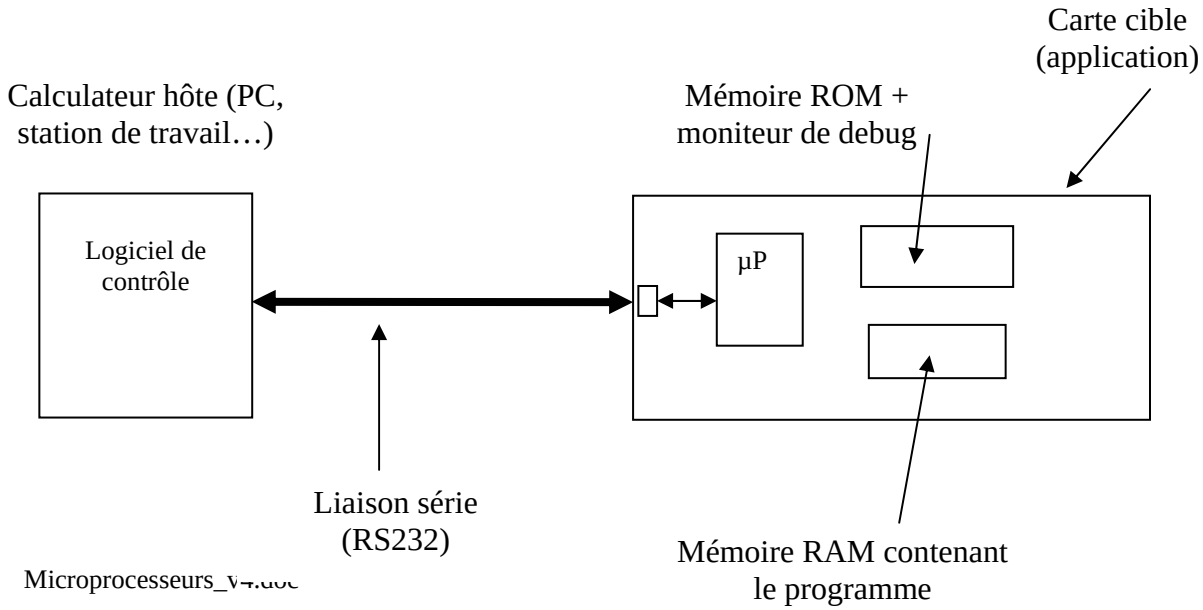
A partir d'un PC hôte, le programme est téléchargé dans l'émulateur. Une sonde **remplace** le composant à émuler sur la carte cible. Le programme est exécuté en temps réel par l'émulateur et la mémoire trace « espionne » et enregistre l'activité des différents circuits. Lorsque le processeur rencontre un point d'arrêt dans le programme, le calculateur hôte reprend la main et a accès, en analysant le contenu de la mémoire trace, à **l'historique** de l'exécution du programme et de l'évolution des ressources.
C'est l'émulateur le plus performant et le plus coûteux que l'on puisse rencontrer actuellement sur le marché. Il est surtout utilisé pour la mise au point matérielle de la carte cible et la mise au point de programmes complexes faisant appel à des entrées-sorties.

INTERCONNEXION d'un EMULATEUR CLASSIQUE
(Le µP/µC/DSP est **<u>remplacé</u>** par une sonde)



4.5) Les émulateurs « low cost »

On les nomme ainsi car ils font appel à peu de matériel. L'architecture d'un tel émulateur est représentée ci-dessous.



Cet émulateur impose lors de la conception de la carte cible d'intégrer une mémoire ROM qui contiendra un programme appelé « moniteur de debugg » et une RAM qui contiendra le programme applicatif à tester.

Lors de la mise sous tension de la carte cible, le microprocesseur exécute le programme correspondant au moniteur de debugg. Celui-ci permet alors de :

- gérer la liaison série avec le PC ou la station de travail
- exécuter les commandes issues de la station (téléchargement du programme applicatif dans la RAM, mise en place de points d'arrêt, …)
- retourner l'état des registres internes du µP vers le calculateur hôte
- …

Les avantages :

- Le programme applicatif est exécuté par le µP de la carte cible à **vitesse réelle**.
- Le programme peut être modifié et téléchargé indéfiniment.
- Pas besoin d'émulateur matériel.
- Possibilité de mettre des points d'arrêt
- Faible coût

Les inconvénients :

- Carte cible plus imposante que ne le nécessite l'application.
- Ajout d'une mémoire RAM + ROM de debug
- Liaison série (RS232) non disponible pour l'application
- Le plan mémoire n'est pas celui de la version définitive.