Introduction aux microprocesseurs & Présentation du μC 9s12

Module Info 2

Sebastien.Kramm@univ-rouen.fr

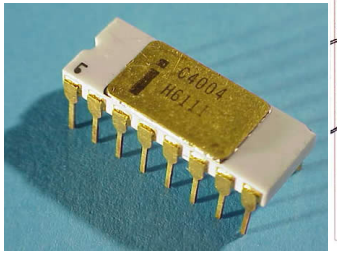
IUT GEII Rouen

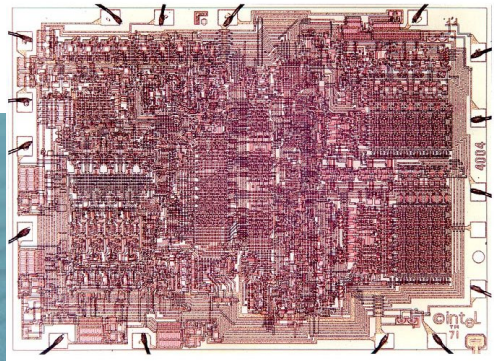
2013-2014

Sommaire

Historique

- 1971 : Intel 4004 : 4 bits, 2300 transistors, 108 kHz

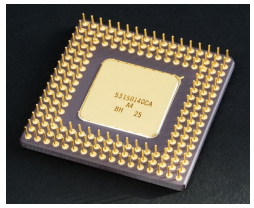


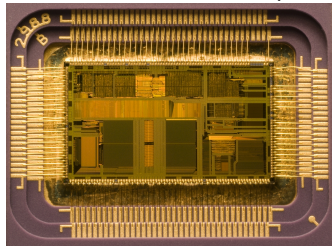


- 1978 : Intel 8088/8086, 16 bits, architecture "x86", utilisé dans les "IBM PC"
- 1985 : Intel 80386 (32 bits)

Historique

- 1989 : Intel 486 ($>$ 1 million transistors)



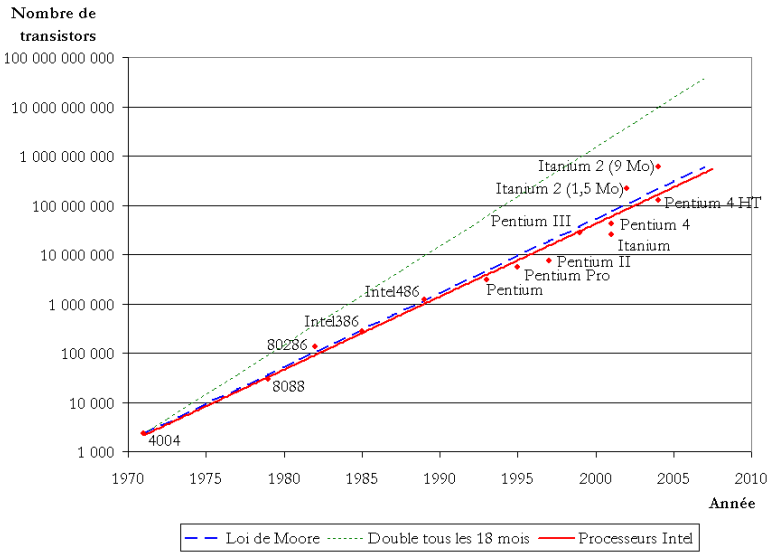


taille réelle : 12 x 7 mm

- 1993 : Intel Pentium
- 2000 : premier processeur x86-64 bits (AMD)
- 2002 : Intel Pentium 4 (Hyper-Threading)
- 2005 : AMD/Athlon 64 X2, premier processeur x86 dual-core.

Loi de Moore (1975)

"*Le nombre de transistors double tous les 2 ans*"



Microcontroleur vs. Microprocesseur

- Microprocesseur :
  - composant généraliste, optimisé pour la puissance de calcul et la vitesse,
  - ne peut fonctionner qu'accompagné de son *chipset*.
- Microcontrôleur : adapté aux applications embarquées, capable de fonctionnement autonome.
  $\Rightarrow$ intègre un microprocesseur, plus :
  - Mémoire
    - RAM
    - ROM, EEPROM, Flash
  - Gestionnaire périphériques
    - Communication
    - Acquisition (capteur, organe de commande, ...)
    - Pilotage actionneurs et afficheurs

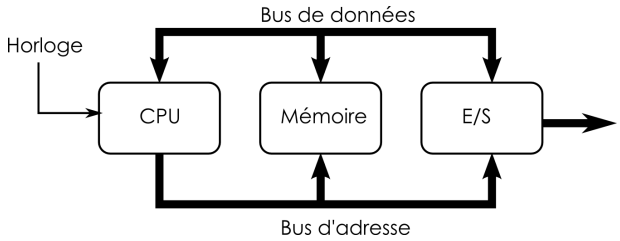
Evolutions et perspectives

- L'augmentation de la puissance de calcul s'est faite principalement sur deux axes :
  - Augmentation de la fréquence d'horloge. Aujourd'hui : $\sim$ 1GHz.
  - Augmentation de la densité d'intégration (finesse de gravure). Aujourd'hui : <50 nm

  $\Rightarrow$ Problème : la dissipation de chaleur ($\sim$ 100 W) devient trop importante.
- Solutions actuelles :
  - *pipeline* d'exécution,
  - architecture multicœur.

Sommaire

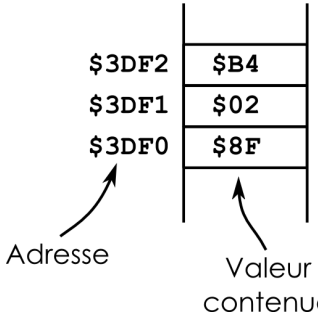
Systeme μ-informatique



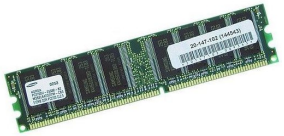
- 3 éléments fondamentaux :
  - CPU : *Central Processing Unit*
  - Mémoire : contient le programme et les données
  - E/S : interface avec l'extérieur
- Bus : interconnexions entre composants sur 'n' fils.
  $\Rightarrow$ permet de transférer une valeur numérique codée en binaire.
  - Le bus d'adresse est unidirectionnel, et piloté par le CPU.
  - Le bus de données est bidirectionnel.
- Un microcontroleur contient ces 3 sous-ensembles sur la même puce.

La mémoire

- Aspect logique : empilement de "cases mémoires", numérotées (adresse).
  - quantité : fonction de la taille du bus d'adresse
  - valeur contenue : octet (8 bits) (convention quasi-universelle)



- Aspect physique : organisée en mots de 8, 16, 32, ... 128 bits.
- Deux types :
  - mémoire "vive" (contenu non persistant sans alimentation)
    $\Rightarrow$ RAM (*Random Access Memory*)
    de type DRAM (*Dynamic*) ou SRAM (*Static*).
  - mémoire "morte" $\Rightarrow$ ROM (*Read Only Memory*)



Taille des bus

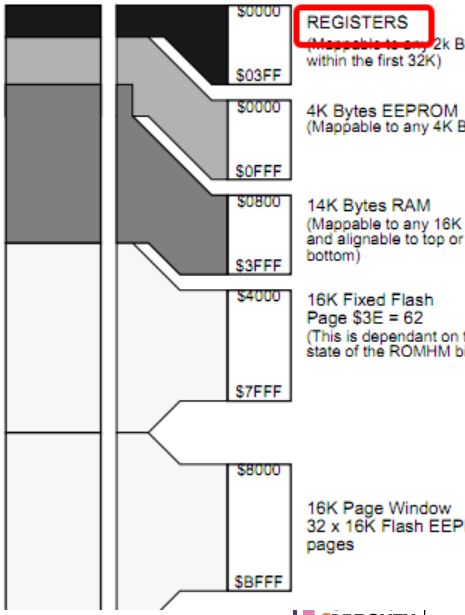
- La taille du bus de données détermine la **catégorie** du processeur (8 bits, 16 bits, 32 bits, ...).
- La taille du bus d'adresse détermine l'**espace mémoire adressable**
  $\Rightarrow$ combien de "cases mémoire" pourront être adressés (octets).
  - 16 bits $\rightarrow 2^{16} = 65\,536$ octets $= 65{,}5$ ko (64×1024)
  - 20 bits $\rightarrow 2^{20} = 1\,048\,576$ octets $= 1{,}04$ Mo (1024×1024)
  - 32 bits $\rightarrow 2^{32} = 4\,294\,967\,296$ octets $= 4{,}29$ Go

Attention

La notation 1024 octets = 1 ko est **abusive**, mais parfois encore utilisée en pratique.

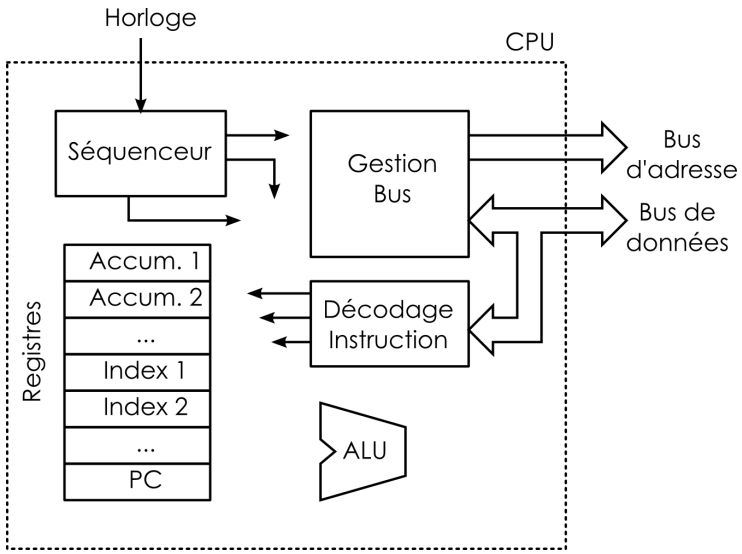
Plan mémoire et E/S

- Sur microcontroleur, les E/S sont souvent "mappées" sur la mémoire : on accède aux registres de contrôles(*) comme à une case mémoire.



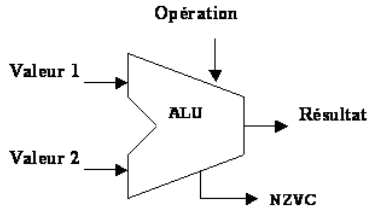
(*) Registres permettant le contrôle des différents blocs d'E/S du microcontroleur, à distinguer des registres-CPU (voir plus loin).

CPU : structure



- Les différents élements sont interconnectables.

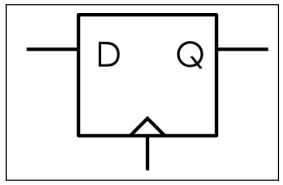
ALU : *Arithmetic and Logic Unit*



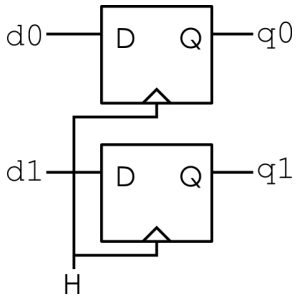
- Caractéristique fondamentale : taille des données (8, 16, 32, ... bits).
- Opérations
  - Logiques : AND, OR, EXOR, complément à 1, à 2, décalages.
  - Arithmétiques
- Bits d'état (NZVC) : fournissent une information sur le résultat.
  - N (*Negative*) : recopie du bit de poids fort, signale un nombre négatif en arithmétique signée
  - Z (*Zero*) : signale un résultat = 0
  - V (*oVerflow*) : retenue (arithmétique signée)
  - C (*Carry*) : retenue (arithmétique non-signée)

Registres / CPU

- Rappel : bascule D
  Fonctionnement : "*Q recopie D sur le front d'horloge*"
  $\Rightarrow$ mémoire 1 bit.

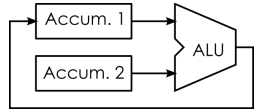


- Assemblage de 2 bascules : mémoire 2 bits.
- Un registre *n* bits est un assemblage de *n* bascules D, synchronisées par un signal commun.



Registres / CPU

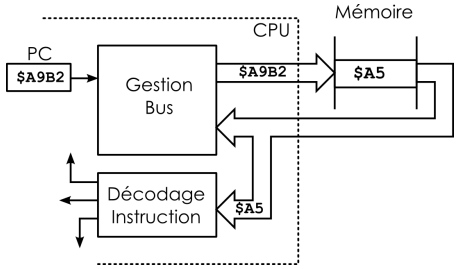
- Utilisés pour mémoriser les données utilisées par l'ALU.
- On distingue
  - les **accumulateurs**, utilisés pour les calculs,
  - les registres d'**index**, pour gérer des pointeurs,
  - les registres système :
    - PC *Program Counter* : contient l'adresse de l'instruction en cours d'exécution,
    - Pointeur de Pile (SP : *Stack Pointer*),
    - d'autres registres système.
- Les calculs se font toujours via des registres.
- Par exemple, addition des deux accumulateurs :
  Acc1 $\leftarrow$ Acc1 + Acc2

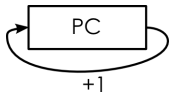


Exécution d'une instruction

- L'exécution d'une instruction se fait en plusieurs cycles d'horloge : le **séquenceur** pilote son déroulement.
- La durée d'exécution est fonction de la complexité de l'opération (3, 4, 5, 6 ... cycles d'horloge, voire plus).

1. lecture en mémoire à l'adresse indiquée par PC du code opératoire de l'instruction,
2. décodage et configuration des interconnexions internes,
3. exécution,
4. incrémentation compteur ordinal.





Jeu d'instructions

- Défini par le constructeur, et propre à la famille du processeur.
- 4 catégories d'instructions :
  1. Transfert de données (mémoire↔registre, registre ↔ registre, mémoire ↔ mémoire).
  2. Opérations arithmétiques et logiques (impliquant l'ALU).
  3. Contrôle de séquence : branchements / sauts (déroutement de l'exécution).
  4. Instructions système diverses.
- Le processeur ne reconnaît que le **code opératoire** : valeur binaire unique qui correspond au code de cette instruction.
- On désigne les instructions par un **mnémonique**.
  Exemple : *Load Accumulator A*, noté `ldaa` : charge le registre-accumulateur A avec une valeur.

Implantation d'une instruction en mémoire

- Le nombre d'octets occupé par une instruction est variable.
- Certaines instructions ont besoin d'une **opérande** (donnée sur laquelle travailler), qui peut être fournie de différentes façons (cf. cours suivant sur les modes d'adressages).
- Par exemple (processeur 9s12) :
  - `inx` : incrémentation du registre X :
    ⇒ 1 octet : $08
  - `ldaa #1` : charge le registre A avec la valeur 1 :
    ⇒ 2 octets (instruction + donnée) : $86 $01
  - `movb $1234,$5678` : copie l'octet situé en $1234 dans la case-mémoire $5678 :
    ⇒ 6 octets : $18 $0C $12 $34 $56 $78

Typologie des architectures

- Deux approches se sont opposées :
  - Approche "traditionnelle" **CISC** (*Complex Instruction Set Computer*)
    → jeu d'instruction de plus en plus étoffé au fur et à mesure des évolutions.
    → richesse fonctionnelle, mais complexité du design, entraînant une limitation des performances.
  - Approche **RISC** (*Reduced Instruction-Set Computer*)
    → jeu d'instructions très réduit (*design* plus simple),
    → chaque instruction s'exécute en 1 cycle d'horloge,
    → moins de transistors, donc plus de place pour d'autres fonctionnalités (mémoire cache, ...)
    → ecriture du *software* plus complexe (en assembleur).
- Il existe des approches intermédiaires (Freescale ColdFire : *Variable Length* RISC) .
- Aujourd'hui, les architectures des CPU modernes rendent cette distinction moins pertinente.

Pipeline d'exécution (1)

- Soit un processeur où 5 cycles sont nécessaires pour exécuter une instruction :
  1. `IF` (*Instruction Fetch*) : charge l'instruction depuis la mémoire,
  2. `ID` (*Instruction Decode*) : décode l'instruction,
  3. `EX` : exécute l'instruction (via l'ALU),
  4. `MEM` (*Memory*) : transfert registre $\leftrightarrow$ mémoire,
  5. `WB` (*Write Back*) : stocke le résultat dans un registre.
- Pour chaque instruction, les différentes étapes vont utiliser 1 cycle d'horloge :

Ex : 3 instructions $\Rightarrow$ 15 cycles d'horloge.

Pipeline d'exécution (2)

- Idée : Si chaque étape correspond à un bloc hardware différent, on peut l'utiliser pour une autre instruction une fois terminée son utilisation pour l'instruction en cours.
- La durée d'exécution globale est raccourcie :

Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

source:wikimedia/Poil

Ex : 3 instructions $\Rightarrow$ 7 cycles d'horloge.

- Inconvénient : l'exécution de certaines instructions dépend parfois du résultat des précédentes.

Sommaire

Aspect matériel

- La **cible** n'a souvent que peu de capacités d'E/S. Il est donc en général **impossible** de développer le logiciel directement dessus.

- De même, le programme ne pourra pas utiliser les fonctions correspondantes...

```
printf( "coucou\n" );
scanf("%d", &a );
```

Aspect matériel

- On utilise un **hôte** (un PC), connecté à la cible, qui est utilisé pour :
  - développer le **programme applicatif** (*firmware*),
  - le télécharger dans la mémoire de la cible,
  - débogguer à distance,
  - faire la programmation définitive de la ROM de la carte.
- L'hôte disposera des différents outils logiciels correspondants.

hôte cible

Développement logiciel

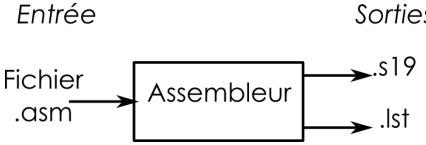
- Irréaliste d'écrire un programme directement en code opératoires !
- 3 approches possibles :
  - Assembleur (langage natif du composant)
  - Langage évolué : C, Basic ("assembleur amélioré"), . . . ,
  - Langage de haut niveau (approche objet) : C++, java, C#, . . .
- Choix du langage : dicté par le contexte (type de produit à développer, compétences en interne dans l'entreprise, etc.)

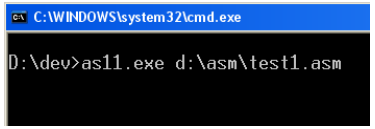
Attention

- Le terme "assembleur" désigne à la fois :
  - Le **langage de programmation**, spécifique à chaque famille de processeur.
  - Le **logiciel** effectuant la transformation du fichier texte en codes opératoires.

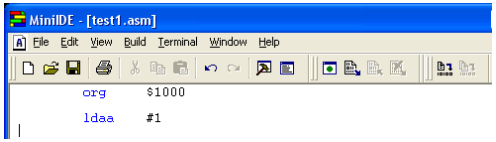
Assembleur (logiciel)

- Le fichier source est passé en entrée à un exécutable.
- L'assembleur produit :
  - un fichier contenant les codes opératoires ("*binaires*"), à télécharger dans la cible (format .s19 ou autre).
  - un fichier "listing", principalement destiné à la vérification.





- Des interfaces graphiques permettent souvent de s'affranchir de la ligne de commande.



Programmation en assembleur (exemple : Freescale)

- 3 champs, séparés par une **tabulation** :
  - col. 1 : **étiquette**
  - col. 2 : **instruction** (mnémonique) ou **directive assembleur**
  - col. 3 : **opérande**

```
; sous-prog de calcul
        org     $1000

DEBUT   ldab    #2      ; comment
        ldx     #TABLE  ; initial
LOOP    ldaa    0,x
        bsr     mon_sp
        inx
        dey
        bne     LOOP
        rts
```

- La première colonne permet d'assigner un point du programme à un symbole (utile pour les branchements).
- Les instructions s'exécutent séquentiellement, sauf en cas de branchement (instruction `bra` ou autre).
- On doit indiquer à quel emplacement de la mémoire on implante le programme :
  → directive assembleur "`org`" (origine).

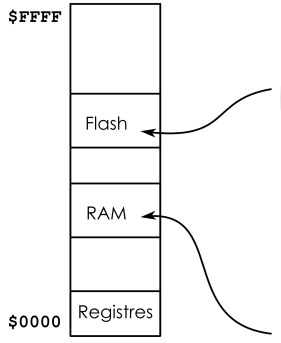
Structure générale d'un source en assembleur

- Impératif : bien structurer son programme
  - Séparation des différents blocs
  - Commentaires
  - Un programme destiné à une cible embarquée comprend :
    - du code exécutable ⇒ en ROM
    - des données constantes (par exemple, table de conversion) ⇒ en ROM
    - des variables ⇒ en RAM
  - Les données constantes ne doivent pas être mélangées au programme, mais être dans une zone mémoire séparée.
  - Ces différentes sections doivent se retrouver dans le source, on les implante avec la **directive assembleur** `org`.

Subdivisions d'un programme embarqué

- Le code exécutable se divise en :
  - un bloc "initialisations", exécutée une seule fois au démarrage,
  - une "boucle principale", rebouclant à l'infini,
  - des sous-programmes (fonctions) et routines d'interruptions.

Structure générale d'un source en assembleur




```
; ---------------------------------------
; Moteur PAP en marche alternée, nbe de pas comma
; DUPONT Jean
; date : 2011 - 03 - 22
; ---------------------------------------

; declaration symboles
VAL1    equ     10

        org     MEM_FLASH
;-----------------------------------
; INITIALISATIONS
        ldaa    #$0f            ;config portc (cl
        staa    DDRC
        clr     PORTC
        ...

;-------------------------------------------
; BOUCLE PRINCIPALE
PP      ldaa    MARCHE          ;si = 0, moteur à
        beq     PP

        bsr     SP
        bsr     TEMPO

        bra     PP
;-------------------------------------------
; SOUS-PROGRAMMES
sp1     ldaa    0,x
        staa    PORTB
        ...
        rts
* ---------------------------------------
; DONNEES
        org     MEM_RAM
table   fcb     %0101,%1100,%1010,%0011
SENS    rmb     1
* ---------------------------------------
```


Instructions de branchement

- On spécifie l'étiquette à laquelle on souhaite dérouter le programme : `bra debut` ⇒ saut à l'étiquette "debut"
- On distingue :
  - **branchement inconditionnel** : le déroutement est systématique.
    - `bra` (*Branch Always*)
    - `bsr` (*Branch to Sub Routine*)
  - **branchement conditionnel** : le déroutement ne s'effectue que **si** une certaine condition sur les bits NZVC est remplie.
    - `bne` (*Branch if Not Equal*) ⇒ branchement si Z=0
    - `beq` (*Branch if EQual*) ⇒ branchement si Z=1
    - etc. (voir TP)
- Ces derniers seront donc placés **immédiatement** à la suite d'une **instruction de comparaison** (ou d'une instruction qui modifie ces bits).

Branchements inconditionnels

- En C :

```
1  while( 1 )
2  {
3      n++;
4      MaFonction();
5  }
```

- En assembleur (*Freescale*)

```
1  BOUCLE  inc   variable_n
2          bsr   MaFonction
3          bra   BOUCLE
4
```

  - `inc` : Incrémentation d'une variable (d'une case mémoire)
  - en C : **fonction** ⇔ en assembleur : **sous-programme** (*subroutine*)
- Sous-programmes : en assembleur, il est **impératif** d'ajouter à la fin une instruction de **retour de sous programme** (RTS : *ReTurn from Subroutine*).

```
MaFonction ...
                ...
               rts
```


Branchements conditionnels

- En C :

```
1     if( var1 == 2 )
2        MaFonction();
3     LaSuite();
4
```

- En assembleur (*Freescale*)

```
1  BOUCLE  ldaa  var1
2          cmpa  #2
3          bne   etape2
4          bsr   MaFonction
5  etape2  bsr   LaSuite
```

  - `ldaa` : (*LoaD Accumulator A*) : charge A avec la valeur fournie.
  - `cmpa` : (*CoMPare accumulator A*) : effectue la soustraction (A-val), et positionne NZVC en fonction du résultat.

Implémentation de boucles de type "for"

- On utilise un registre ou une variable comme **décompteur**.
- On le décrémente dans le corps de la boucle.
- On utilise un branchement conditionnel pour détecter l'arrivée à 0.
- Par exemple :

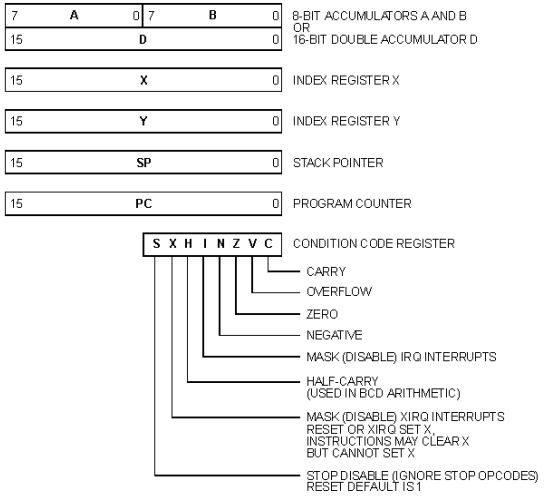
```
1  DEBUT   ldaa  #5
2  BOUCLE  ...           ; ici, la t\^ache à réaliser dans
3          ...           ; le corps de la boucle
4          deca
5          bne   BOUCLE  ; si pas égal à 0, on recommence
6          ...           ; la suite
```

- Attention : il ne faut pas modifier l'accumulateur A dans le corps de la boucle !

Sommaire

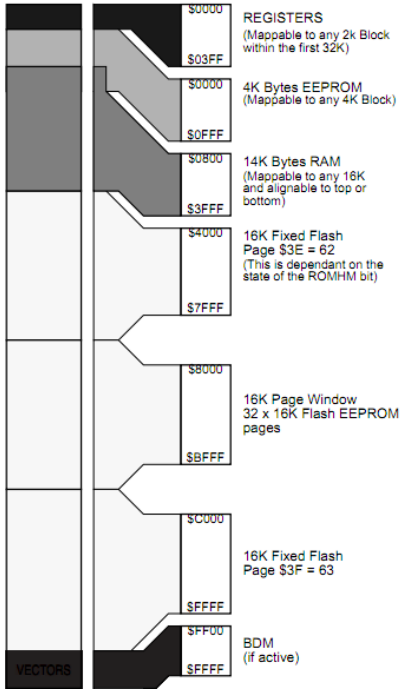
CPU : registres

- 2 accumulateur 8 bits OU 1 accumulateur 16 bits
- 2 registres d'index X et Y (=pointeurs)
- 1 Program Counter (PC)
- 1 pointeur de pile (*Stack Pointer*)
- 1 registre d'état CC



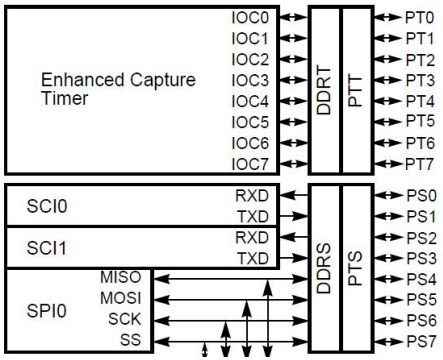
Memory map

- L'espace mémoire adressable est sur 16 bits. (`$0000` à `$ffff`)
- Un système de pages permet d'accéder au 512 ko. de flash.



Broches externes

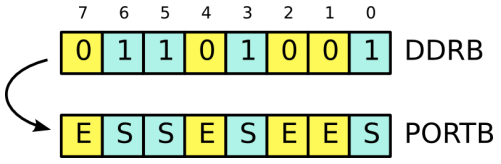
- La plupart des broches sont pilotées
  - soit par un "bloc fonctionnel",
  - soit par un port d'E/S binaire universel (GPIO).
- La sélection entre les deux se fait **automatiquement**.



- Si le bloc *Enhanced Capture Timer* (ECT) est activé, les broches PT0 à PT7 seront contrôlées par celui-ci, sinon par les registres DDRT/PTT.
- Si le bloc SCI0 est activé, les broches PS0 et PS1 seront contrôlées par celui-ci, sinon par les bits 0 et 1 de DDRS & PTS.

Ports GPIO

- Le processeur est doté de plusieurs ports de broches GPIO (*General Purpose Input Output pin*) bidirectionnelles.
- Ces broches sont organisées en **ports** de 8 bits et sont identifiés par une lettre (port A, B, C,...)
- Le **sens** des broches (Entrée ou Sortie) est contrôlé par un bit dans un registre associé au port. (par ex. DDRB : Data Direction Register port B).
  - 0 : entrée
  - 1 : sortie



La valeur du bit (entrée ou sortie) est accessible dans le registre du port

- par une lecture si broche en entrée,
- par une écriture si broche en sortie.