

**2022-2023**

**Cycle Ingénieur  
1ère Année**

**IE.1102-IE.1202**

*Frédéric AMIEL*

# **Architecture des ordinateurs**

**COURS et TPs**

*Frédéric AMIEL*

**2022-2023**





## Table des matières

Bibliographie.....	p 4
Un peu d'histoire.....	p 5
Cours 1	
Composants élémentaires.....	p 12
Cours 2	
Arithmétique.....	p 43
Architecture externe.....	p 53
Cours 3	
Présentation de l'ARM – 1.....	p 64
Cours 4	
<i>Présentation de l'ARM – 2.....</i>	p 90
Cours 5	
<i>Les circuits périphériques.....</i>	p 110
TD1 .....	p136
TD2 .....	p137
TD3 .....	p138
TD4 .....	p139
TD5 .....	p140
TP1 .....	p141
TP2 .....	p145
TP3 .....	p151
TP4 .....	p155
TP5 .....	p159
Exam 2021-2022 S2 .....	p164
TP Examen Blanc .....	p172
ANNEXE .....	p174
Schéma de la carte IAR .....	p175
Adresses des périphériques STR730 .....	p 176
Fonctionnement des ports parallèles .....	p 178
Instructions ARM7.....	p182

## Bibliographie :

### Historique :

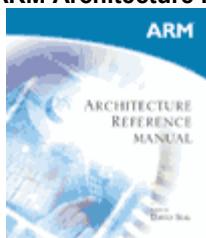
<i>Histoire de l'informatique</i>	P. Breton
<i>La genèse des calculateurs électroniques</i>	B. Randell
<i>Histoire du calcul digital</i>	Brian
Différents sites WEB anglophones proposent des pages sur l'histoire de l'informatique.	
Les sites des grandes sociétés ont généralement également un chapitre histoire. Par exemple, l'invention du microprocesseur par INTEL est relatée sur le WEB :	
<a href="http://www.intel.com/intel/museum/25anniv/index.htm">http://www.intel.com/intel/museum/25anniv/index.htm</a>	

### Architecture des ordinateurs :

Computer organization and design	Patterson / Hennessy
Architecture de l'ordinateur – cours et exercices	Andrew Tanenbaum
Compilateurs, Principes Techniques et Outils	Aho R. Sethi J.Ullman
Understanding and Writing Compilers	R. Bornat

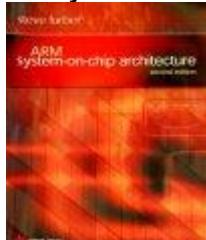
### Processeur ARM :

#### ARM Architecture Reference Manual



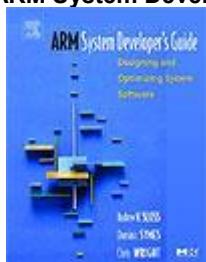
Second Edition, edited by David Seal : Addison-Wesley :  
ISBN 0-201-73719-1  
(Known as the "ARM ARM". ARM Doc No.: DDI-0100)

#### ARM System-on-chip Architecture



In English language, by Steve Furber  
Second Edition  
published by Addison Wesley  
ISBN 0-201-67519-6

#### ARM System Developer's Guide



Hardbound, ISBN: 1-55860-874-5, 704 pages, publication date: 2004  
Imprint: MORGAN KAUFFMAN

[www.arm.com](http://www.arm.com)

## 1. Un peu d'histoire

### 1.1. INTRODUCTION

Luxor, deux mille ans avant notre ère. Le prêtre prononce les incantations rituelles devant le brasier, les portes du temple s'ouvrent lentement par la seule grâce du dieu. Dès cette époque, un mécanisme entièrement automatique permettait de traiter, de manière très rudimentaire, une information. L'allumage du feu, déclencheait un courant d'eau capable de pousser les portes.

Les microprocesseurs d'aujourd'hui, utilisés par exemple dans des machines de contrôle d'accès, effectuent le même type de tâche, avec leurs mégaoctets de mémoire, et leurs millions de transistors.

Entre ces deux extrêmes, la machine à traiter l'information n'a cessé de se développer. Les microprocesseurs sont aujourd'hui au cœur de la plupart des appareils, voitures, domotique, ordinateurs personnels, serveurs, super-ordinateurs ....

La puissance de ces microprocesseurs n'a cessé d'augmenter depuis leur création, suivant la progression exponentielle de la loi de Moore formulée en 1965...

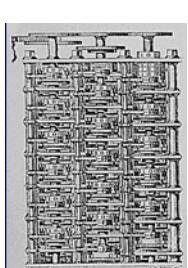
Les microprocesseurs sont champions du monde sur la plupart des jeux, ils reconnaissent les images de façon plus performante que les êtres humains, leur usage est multiforme, et quasi toutes les conceptions électroniques en utilisent.

Ce cours a pour objet de décrire les principes fondamentaux de leur fonctionnement, afin de savoir les choisir, manipuler, programmer, avec efficacité et pertinence.

### 1.2. LES PREMIERES MACHINES A CALCULER

C'est en 1623 qu'apparaît la première "horloge à calcul" de W Schickard. Blaise Pascal inventera la PASCALE ou PASCALINE en 1642, pour son père collecteur d'impôts dans la région de ROUEN.

Différents constructeurs réalisent des calculatrices mécaniques mais il faut attendre Charles BABBAGE (1792-1871) pour la conception de deux machines révolutionnaires pour l'époque. En avance sur la technologie de son époque, ses machines ne furent pas réalisées de son vivant, mais Babbage est considéré comme le père de l'informatique. Sa deuxième machine pose les principes fondamentaux de ce que seront plus tard les ordinateurs. (Unité de calcul, unité mémoire...). Ada Lovelace qui conçue les programmes adaptés à ces machines est la première programmeuse de l'histoire.

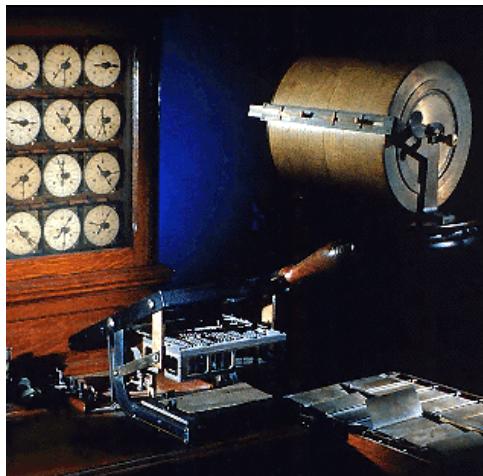


**Figure 1 : La machine à différences - E Babbage - Ada Lovelace**

A la même époque, Jacquard réalise ses métiers à tisser mécaniques, avec programmes à cartons perforés.

C'est le 20ème siècle, avec l'avènement de l'électricité puis bien sûr de l'électronique qui permettra la construction des ordinateurs.

En 1880, H Hollerith, le futur fondateur de la société IBM, fabrique le pantographe, système électrique permettant de 'lire' une carte perforée et d'actionner des compteurs. Cette machine est exploitée pour le recensement des Etats-Unis, et permet d'annoncer le chiffre de 62 622 250 habitants en 6 semaines. Le pantographe démontrait sa redoutable efficacité, le dénombrement précédent, ayant demandé 8 ans de dépouillement.



**Figure 2 : Pantographe et compteurs totalisateurs - IBM**

Chaque individu se voit attribué une carte qui est perforée ou non, suivant la catégorie d'âge, sexe, catégorie sociale...

Les cartes sont collectées et passent sous le pantographe, sorte de planche à clous connectés à différents compteurs électromécaniques, qui totalisent les différentes informations.

La carte perforée entre dans l'histoire. Le système est étendu dans les années suivantes aux traitements statistiques, à la gestion du personnel dans une entreprise, etc.

Cette première utilisation de "machine de traitement de l'information" répond à des demandes du type : nombreuses données, calculs simples. Les ordinateurs, en effet, seront employés majoritairement à ce genre de tâche dans les années suivantes.

La poursuite dans cette voie, associant les cartes perforées et totalisation donnera ces machines utilisées pour la gestion comptable des entreprises. La mécanographie (machines à relais) ne disparaissait qu'à la fin des années 60.

Remarques :

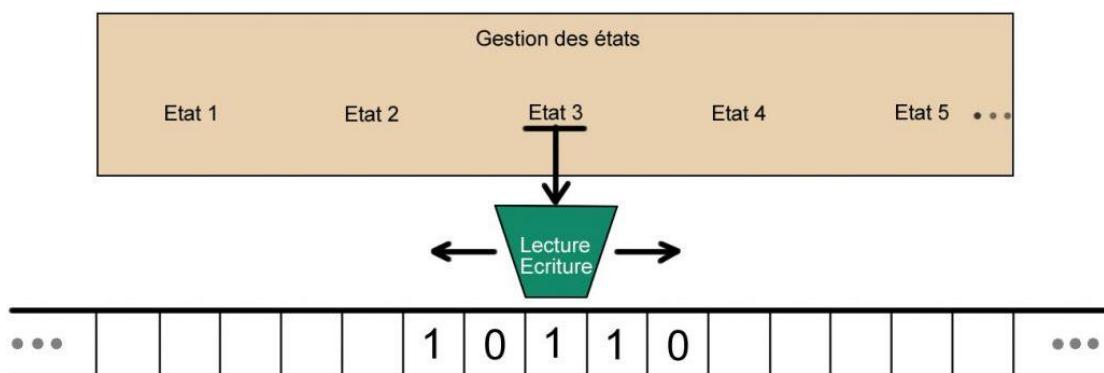
Le code binaire a été inventé par le philosophe F Bacon comme système de cryptographie pour passer des messages diplomatiques.

Le mot algorithme vient de la traduction en latin d'un traité d'algèbre écrit par un mathématicien arabe : Al-Khwarizmi. Le traité commence par ces mots : Algoritmi dixit... La notion est ancienne, mais c'est Turing qui formalisera le terme.

Avec la thèse de Shannon sur l'application de l'algèbre de BOOLE (1815- 1864) aux circuits de commutations électriques. La théorie de l'information naît, et le BIT (BInary digiT) est défini comme étant la plus petite quantité d'information. L'information est définie comme mesure de l'entropie.

La machine de Turing :

Entité la plus simple que l'on puisse imaginer pour aboutir à un organe de "traitement automatique de l'information". Cette machine sert de modèle pour étudier les algorithmes.



**Figure 3 : Ruban et mécanisme de la machine de Turing**

Par la suite, Von Neumann définira son architecture d'ordinateur comme machine la plus simple permettant de simuler une machine de Turing.

Concernant les calculs, en 1930, l'analyseur différentiel de l'américain Vannevar (1930) était la machine la plus puissante du monde à faire des calculs scientifiques. Cette machine, toutefois, résolvait des équations différentielles de manière analogique.

Ce sont les années 40 qui vont voir naître l'ordinateur.

Trois types de calculateurs sont alors construits : les calculateurs électro mécaniques, les calculateurs électroniques et les calculateurs analogiques.

### 1.2.1. LES RELAIS

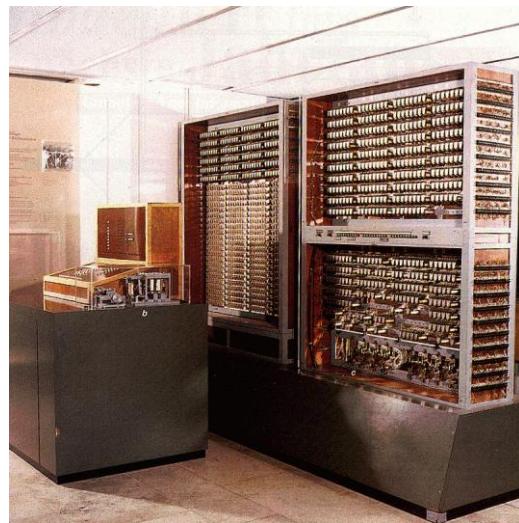
De la technologie mécanique à l'usage de l'électricité avec les relais téléphoniques, la différence est quantitative, on accélère un peu les calculs.

Le premier calculateur à relais, le Model I fut construit en 1939 dans les laboratoires de la compagnie Bell Telephone (qui disposait des milliers de relais nécessaires).



**Figure 4 : Z1 de Zuse**

Zuse, réalise à Berlin ses calculateurs électromécaniques. Le Z3, avec 2600 relais réalise en 1941 une multiplication en 3 à 5 secondes et emmagasine jusqu'à 64 nombres de 22 bits.



**Figure 5 : Zuse Z3**

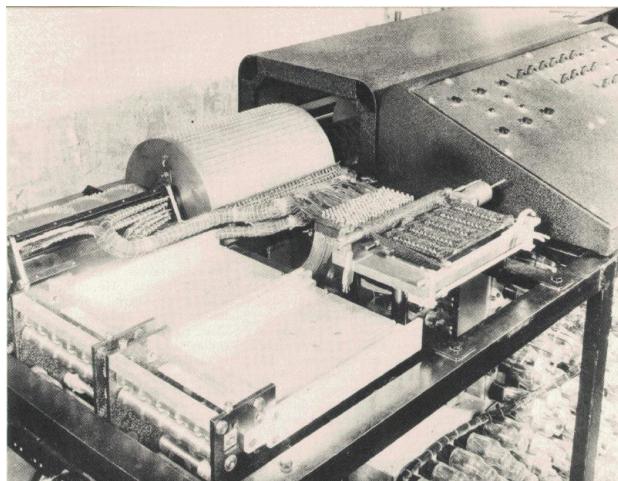
En 1946, le model V comporte 9000 relais, pèse 10 tonnes pour 105 m<sup>2</sup> de surface.

Aiken, utilise les technologies de la mécanographie pour le MARK I, il construit avec la collaboration d'IBM une machine de 5 tonnes et 800 000 éléments qui multiplie 2 nombres décimaux en 3 secondes.

### 1.2.2. L'ELECTRONIQUE

Seule la technologie des tubes à vide va permettre d'améliorer les performances.

L'ABC des américains J. Atanasoff et C Berry est le premier calculateur fonctionnant avec des tubes à vide. Cette machine révolutionnaire utilisait l'algèbre de BOOLE et le système binaire. Très lente (60 Hz) d'horloge, ses performances ne la démarquent pas des systèmes électromécaniques.



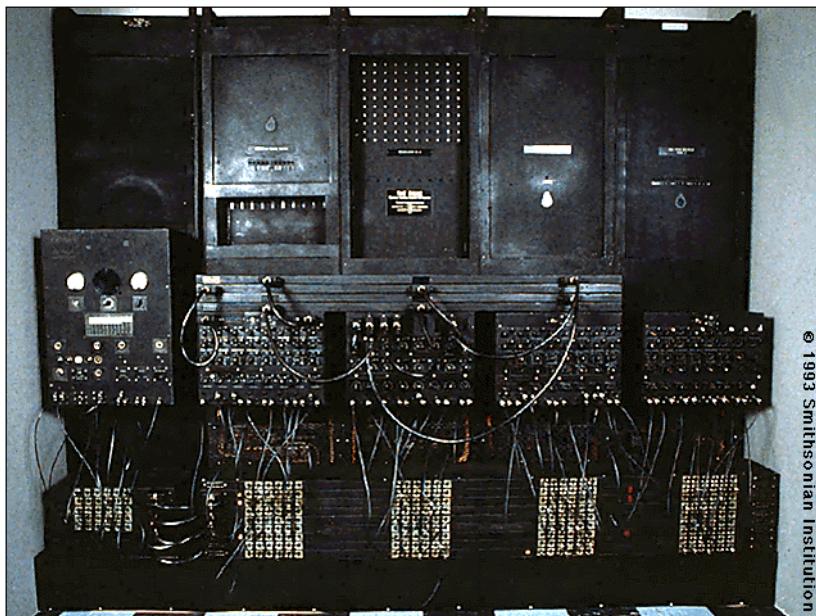
**Figure 6 : ABC**

Les tubes existent depuis plus de 20 ans à l'époque, mais les concepteurs d'ordinateurs sont des logiciels et non des électroniciens. L'ABC tombait fréquemment en panne, ce qui ne permettait pas de l'utiliser pour la résolution des équations linéaires trop gourmandes en chiffres binaires et en temps de calcul.

La deuxième guerre mondiale fut un puissant facteur de motivation les britanniques développèrent leurs machines pour le décryptage des messages secrets de l'adversaire. Les américains utilisaient l'ABC, puis une version de l'analyseur différentiel (analogique) pour le calcul de tables de tir.

C'est dans ce cadre, que fut lancée la construction de l'ENIAC en 1943. Cette machine entra en service en 1945, 3 mois après la reddition des japonais.

Cette machine, conçue par P.Eckert et J.W.Mauchly, est le dernier dinosaure d'avant l'ordinateur moderne. Eckert électronicien radio de formation, pense à sous-alimenter les filaments pour rendre la durée de vie des tubes pratiquement infinie. La machine conçue sur des principes modulaires était très fiable et prouva l'intérêt de l'électronique dans la construction des grandes machines à calculer.



**Figure 7 : ENIAC**

L'horloge de l'ENIAC battait à 200 000 Hz. Lors de sa première démonstration publique, la machine additionna 5000 nombres en 1 seconde. Imposant, ce calculateur pesait 30 tonnes, pour une surface de 160 m<sup>2</sup> et était composé de 17468 tubes, 70000 résistances, 1500 relais 10000 capacités et ... 6000 commutateurs manuels.

La programmation de ce monstre nécessitait la manipulation de plusieurs milliers de ces commutateurs et le branchement de plusieurs centaines de câbles. "La conscience des limites que sa taille et surtout son organisation interne avait atteinte" devait conduire à la naissance des véritables ordinateurs.

Von Neumann (Mathématicien) et Goldstine, (un scientifique militaire qui avait suivi le projet 'PX' le nom de code pour l'ENIAC), définissent alors les concepts de l'ordinateur moderne. En 1945, paraît un texte sur l'organisation d'une nouvelle machine : l'EDVAC. Dans ce texte est définie l'architecture de Von Neumann, qui est encore à la base de la majorité des ordinateurs actuels.

La diffusion des textes sur l'EDVAC (Electronic Discrete Variable Computer) va suivre plusieurs chemins qui aboutiront à la construction de 5 machines principales : l'IAS (Von Neumann), le BINAC (Eckert et Mauchly), l'EDSAC (Wilkes de l'université de Cambridge) et le MARK 1 (Newman de l'université de Manchester).

En 1951, l'UNIVAC (successeur du BINAC) puis l'IBM 701 seront les premiers modèles commercialisés.

L'histoire s'accélère alors et c'est une succession de modèles de plus en plus puissants, avec de plus en plus de mémoires. En Europe, les suédois construisent le BARK, l'Angleterre le MARK 1, la France le CUBA et la série GAMMA de la société BULL.

Remarque : En 1962, P.Dreyfus ingénieur chez BULL invente le mot 'informatique'.

### 1.2.3. LA DEUXIEME GENERATION : LE TRANSISTOR

Le transistor va permettre de réaliser des machines plus puissantes et surtout beaucoup plus fiables. Les quatre premiers modèles seront construits aux Etats-Unis : le SEAC, le TRANSAC, l'AGC Model I, puis le CDC 1604 de la société Control Data.

En 1960, l'IBM 7090 fut la première machine entièrement transistorisée.

Les années soixante vont voir le développement de l'informatique moderne. Les langages évolués se développent (le FORTRAN: FORMula TRANslator a été mis au point entre 1953 et 1956). D'autres langages sont apparus par la suite : le LISP en 1956 le COBOL (1960), l'ALGOL (1960), le PL1 (1964), le PASCAL (1964)...

Au niveau HARDWARE, la société IBM définit toute une série de principes : l'émulation, la mémoire cache, la mémoire virtuelle... La série 360 d'IBM sera le support de ces innovations.

Avec l'EBCDIC, le codage d'un caractère se fait sur 8 bits et non plus sur 6. La plus petite unité directement accessible en mémoire est normalisée à 8 bits, soit un octet (8 bits = 1 byte).

Le premier mini ordinateur date de 1957 : le PDP 1 (Programmed Data Processor model I) de Digital Equipment.

En 1959, J Kilby de la société Texas Instrument créa le premier circuit intégré (assemblant plusieurs transistors sur une puce), initiant une densité de composant toujours plus élevée.

En 1971, INTEL créa le premier microprocesseur.

De 1945 à 1975, trois générations de machines. L'informatique évolue surtout avec la technologie. Elle est naissante, mais ses fondements théoriques ont peu évolué depuis SHANON, TURING et VON NEUMANN.

Les années 80 ont vu la micro-informatique gagner ses lettres de noblesse. Le prix de plus en plus bas des circuits intégrés permet la floraison de machines et d'architectures du plus en plus originales.

### **1.3. INTERET DE L'ASSEMBLEUR**

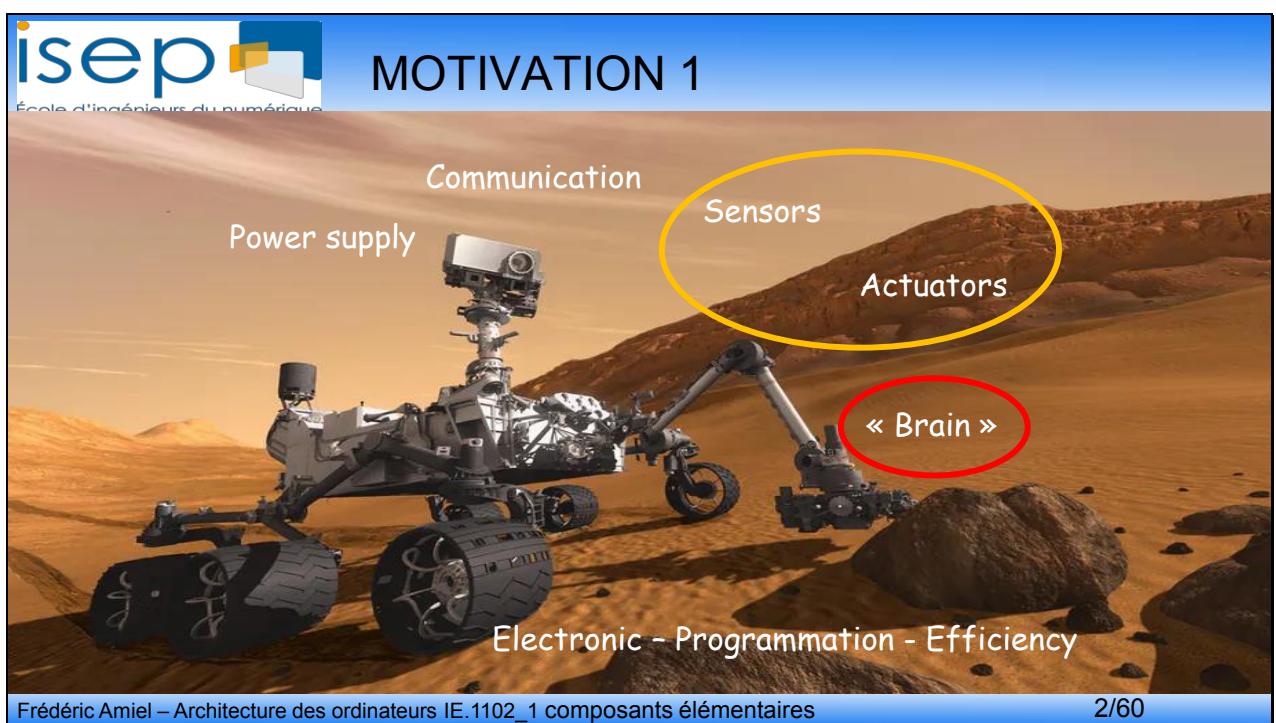
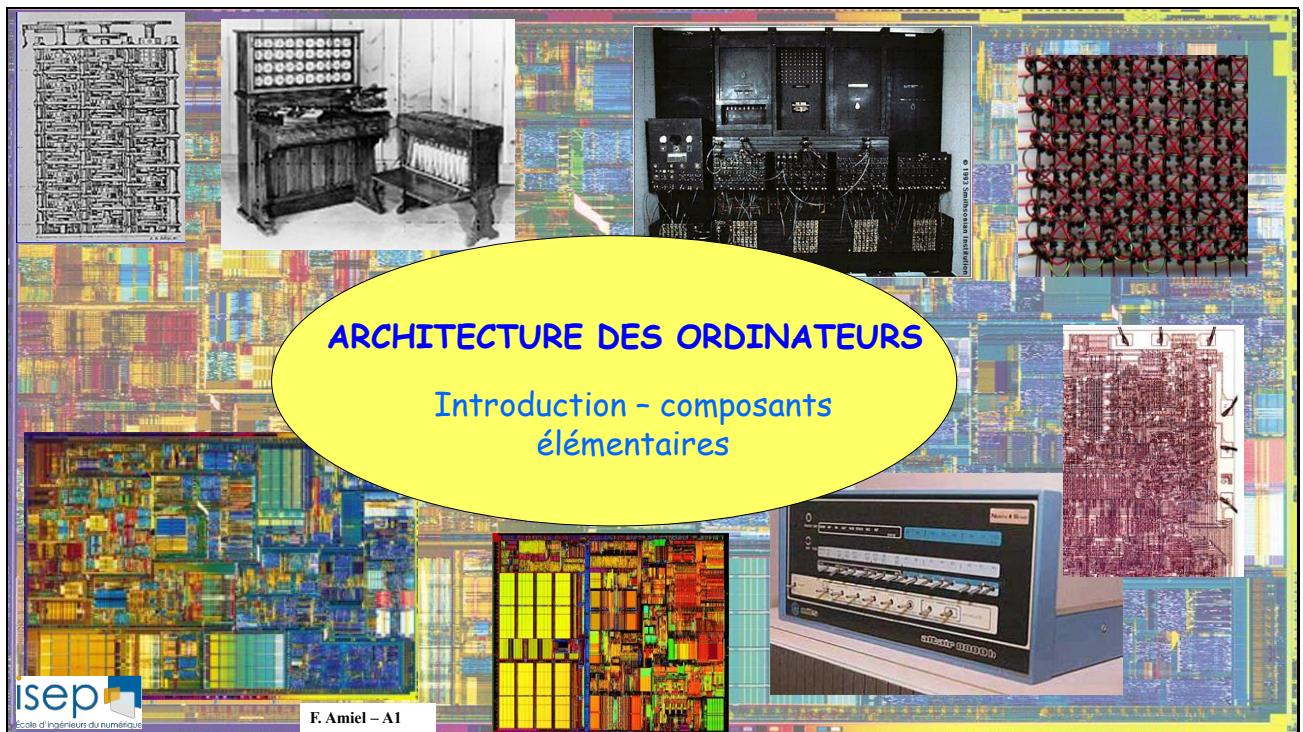
Aujourd'hui, la programmation des ordinateurs s'effectue en utilisant des langages de hauts niveaux.

Ces langages, permettent une approche des applications plus abstraite. Ils permettent d'écrire des programmes avec un minimum de lignes, plus de sûreté, de preuve de fonctionnement. Ils sont portables, c'est à dire qu'on peut en principe les exécuter indifféremment sur une machine ou une autre.

Le langage assembleur, présenté dans ce cours, est le langage bas niveau du processeur. Il n'est pas portable d'un processeur à l'autre, il suppose une connaissance précise du matériel utilisé, il est plutôt « verbeux », c'est à dire qu'il faut beaucoup de lignes pour programmer des fonctions simples. Enfin aujourd'hui, il est extrêmement minoritaire en termes de pourcentage de code écrit à la main.

Quelques raisons qui motivent ce cours sont énumérées ci dessous :

- Tous les programmes, écrits en quelque langage que ce soit, sont traduits en assembleur avant d'être exécutés. Connaître l'assembleur, permet de comprendre précisément comment le programme est exécuté.
- L'assembleur permet de faire le lien entre logiciel et matériel. Un ingénieur en électronique doit avoir un modèle très précis de la machine qui exécute ses programmes.
- L'assembleur est le seul langage pour certaines applications systèmes. Il existe dans les processeurs des instructions spécifiques, permettant de gérer des sécurités, des paramètres liés au processeur qui ne peuvent être exprimées qu'en assembleur. Sur la plupart des processeurs d'autre part, la manipulation des registres systèmes (masque d'interruption, interruptions logicielles...) ne peut s'effectuer que grâce à l'assembleur. Tous les programmes système, disposent donc de fonctions écrites nécessairement en assembleur, même s'ils sont écrits majoritairement en langage C ou C++.
- Certaines applications doivent être codées en assembleur pour optimiser la vitesse d'exécution. (C'est moins vrai sur les architectures parallèles ou les compilateurs effectuent un travail d'optimisation très avancé).
- Parfois, pour des raisons de coût, on programme en assembleur. (Sur des tout petits microcontrôleurs, on cherche à optimiser le nombre de cases mémoires du programme). En effet, si l'assembleur requiert l'écriture manuelle de beaucoup de lignes de codes, les langages évolués génèrent automatiquement un code moins compact, utilisant plus de cases mémoire. Dans ce type d'application, le codage assembleur est en général simple.
- Comprendre l'assembleur d'un processeur spécifique (c'est le but de ce cours) permet de programmer en assembleur n'importe quel processeur après une période d'apprentissage réduite.
- ...



**ISEP** École d'ingénieurs du numérique

## MOTIVATION 2

<https://thenewstack.io/which-programming-languages-use-the-least-electricity/>

Language	DRAM Energy (Joules)	Mem MB
Ada	~6.5	~1.00
C	~2.5	~1.03
C++	~2.5	~1.34
Chapel	~3.5	~1.56
Fortran	~4.5	~1.56
Go	~5.0	~1.89
Haskell	~14.0	~4.40
Ocaml	~7.5	~1.89
Pascal	~13.5	~1.00
Rust	~2.0	~1.00
Swift	~4.0	~1.00

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires      3/60

**ISEP** École d'ingénieurs du numérique

## MOTIVATION 2bis

**Table 4.** Normalized global results for Energy, Time, and Memory

Total			
	Energy	Time	Mem
(c) C	1.00	1.00	1.00
(c) Rust	1.03	1.04	1.05
(c) C++	1.34	1.56	1.17
(c) Ada	1.70	1.85	1.24
(v) Java	1.89	1.89	1.34
(v) Chapel	2.14	2.14	1.47
(v) Pascal	2.18	2.18	1.54
(v) Lisp	2.27	3.14	1.92
(c) Ocaml	2.40	3.09	2.45
(c) Fortran	2.52	3.14	2.57
(c) Swift	2.79	3.40	2.71
(c) Haskell	3.10	3.35	2.80
(v) C*	3.14	4.20	2.82
(c) Go	3.23	4.20	2.85
(i) Dart	3.83	6.30	3.34
(v) F#	4.13	6.52	3.52
(i) JavaScript	4.45	6.67	3.97
(v) Racket	7.01	11.27	4.00
(v) TypeScript	21.50	26.99	4.25
(i) Hack	24.02	27.64	4.59
(i) PHP	29.30	36.71	4.69
(i) Erlang	42.23	43.44	6.01
(i) Lua	43.88	48.20	6.22
(i) Ruby	46.54	59.34	6.72
(i) Ruby	69.91	65.79	7.20
(i) Python	75.88	71.90	8.64
(i) Perl	75.88	82.91	19.84

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires      4/60

**ISEP** École d'ingénieurs du numérique

## Objectifs

- Maîtriser le lien entre informatique et électronique
- Comprendre comment fonctionne un ordinateur
  - Comprendre le fonctionnement d'un microprocesseur
- Impact :
  - Concevoir des systèmes électroniques

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires      5/60

**isep** École d'ingénieurs du numérique

## Organisation du module

- IE.1102-IE.1202
  - Cours et travaux pratiques :
    - Microprocesseur (4 crédits)
      - Bimestre 1
        - 7 séances de cours 1h30
        - 6 séances de TD 1h30
        - Examen QCM 1h30
      - Bimestre 2
        - 6 séances de TP 1h30
        - Contrôle continu
        - Examen TP 40 mns

**isep** École d'ingénieurs du numérique

## Bibliographie

- Livres
  - Computer Architecture : A Quantitative Approach by Hennessy, Goldberg and Patterson
    - Anglais
    - Introduction à l'architecture des ordinateurs
    - Architecture interne des processeurs
  - Architecture des ordinateurs 5eme édition. Hennessy - vuibert.
- Web
  - Sites constructeurs (Intel - Arm - Amd...)
  - Universités (MIT - Inria...)

Ce cours s'intéresse à la programmation assembleur des microprocesseur, et plus spécifiquement du processeur ARM7, c'est-à-dire aux aspects logiciels bas niveau des ordinateurs.

Le cours ARC201 décrit lui l'organisation matérielle des ordinateurs.

**ISEP**  
École d'ingénieurs du numérique

## Objectifs de séance

Introduction - architecture des ordinateurs  
Présentation des mémoires - Différents types de mémoires  
Comprendre les fonctions d'un bus  
Quelques constituants

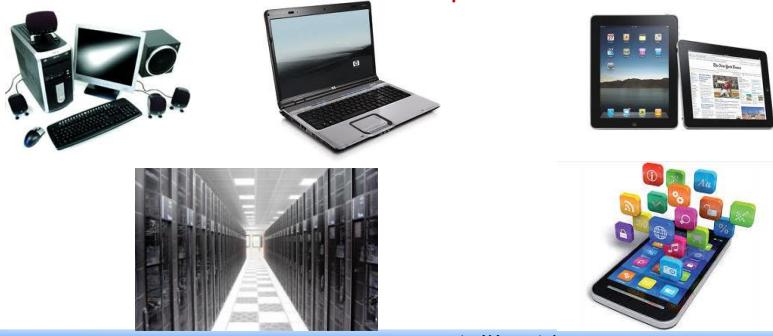
Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires      8/60

Ce premier module constitue un rappel de différentes notions vues lors du cours de logique. Il fait parti de l'introduction au cours sur le processeur ARM.

**ISEP**  
École d'ingénieurs du numérique

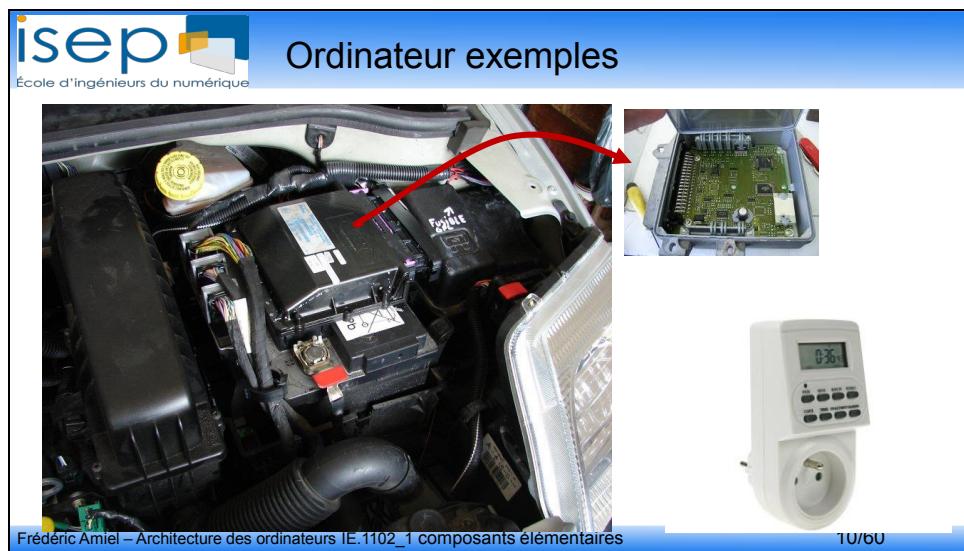
## Ordinateur

**Machine automatique de traitement de l'information, obéissant à des programmes formés par des suites d'opérations arithmétiques et logiques.**



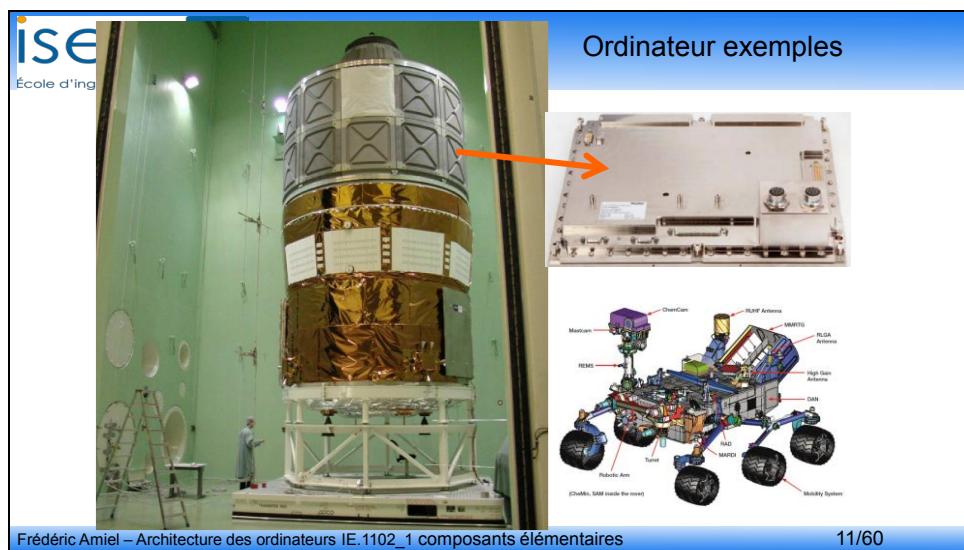
Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires      9/60

Selon cette définition, l'ordinateur est utilisé dans de très nombreuses applications. Ici, les ordinateurs personnels et un super calculateur.



On retrouve des calculateurs embarqués sous le capot des automobiles, des trains, pour assurer différentes tâches qui vont du contrôle de la combustion du moteur à explosion, à la régulation, aux commandes d'allumage des phares, au rythme des essuies glaces ou du clignotant...

De nombreux appareils domestiques embarquent des calculateurs, comme ce programmeur générique, qu'on pourrait retrouver dans un four, une machine à laver etc.



Encore un exemple de calculateur dans le spatial. Chaque domaine d'application ayant ses contraintes de fonctionnement, liées à la fois au contexte d'usage et à l'application :

Il est peu important de « louper » une image sur un film projeté en divx sur un ordinateur personnel. Il est hors de question de louper une « trame de temps » dans le calculateur de bord gérant l'asservissement de la fusée Ariane.

Le robot Curiosity emporte sur la planète Mars un processeur PowerPC 750 (IBM-Motorola) cadencé à 200MHz. Dans ce cas, pour des raisons de fiabilité, le système est redondé.

**ISEP**  
École d'ingénieurs du numérique

## Calculateurs



La médecine utilise des calculateurs au sein d'appareils implantés.



Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires 12/60

L'électronique médicale est un autre exemple d'utilisation de calculateurs embarqués. La contrainte ici est la faible consommation et la miniaturisation des systèmes. Le propos de ces implants étant de mesurer, stocker, communiquer des informations (des données physiologiques), et parfois d'activer des actionneurs (chocs électriques dans le cas du pacemaker) selon un critère programmé.

**ISEP**  
École d'ingénieurs du numérique

## Calculateurs

(presque) TOUS ces calculateurs sont basés sur :

# L'ARCHITECTURE DE VON NEUMANN

Cette architecture permet de créer des machines :

- Avec des claviers et des écrans (ou sans)
- Gérant des sons (enregistrement, restitution, traitement)
- Lisant des capteurs (pression, inclinaison..)
- ...

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires 13/60



**Calculateur**

**Machine automatique de traitement de l'information, obéissant à des programmes formés par des suites d'opérations arithmétiques et logiques.**

Ce cours s'intéresse à la conception de tels systèmes. Il doit faire le lien entre l'électronique et l'informatique.

**Electronique** : Partie de la physique qui étudie et utilise les variations de grandeurs électriques (champs électromagnétiques, charges électriques, courants et tensions électriques) pour capter, transmettre et exploiter de l'information ; technique dérivant de cette science.

**Informatique** : Domaine d'activité scientifique, technique et industriel concernant le traitement automatique de l'information par des machines : des calculateurs, des systèmes embarqués, des ordinateurs, des robots, des automates, etc.

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires      14/60

Par électronique on entend le MATERIEL : (HARDWARE)  
Par informatique on entend le LOGICIEL : (SOFTWARE)



**Calculateurs**

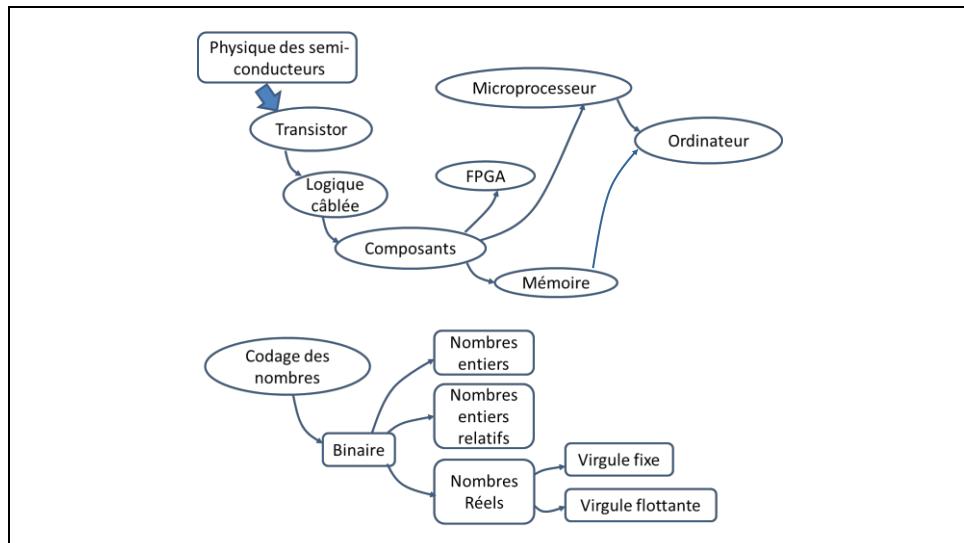
**Machine automatique de traitement de l'information, obéissant à des programmes formés par des suites d'opérations arithmétiques et logiques.**

On va s'intéresser à la façon dont ces programmes sont interprétés et exécutés par l'électronique.

On ne va pas décrire COMPLÈTEMENT l'électronique qui permet de lire des programmes (enseignement spécifique à certains parcours)

Ce cours va décrire l'exécution des logiciels au plus bas niveau (à la lisière de la transition vers l'électronique).

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires      15/60



## ISEP Objectifs généraux du module

École d'ingénieurs du numérique

- Etre capable de programmer une application simple
- Concevoir et dimensionner un calculateur pour une application
- Connaitre les paramètres permettant de choisir un calculateur réalisant une fonction.
- Optimiser la programmation selon différents critères
  - Consommation
  - Rapidité d'exécution
  - Simplicité de développement

**ASSEMBLEUR**  
(Langage machine)  
  
Langage C

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires      17/60

## ISEP Architecture Von Neumann

École d'ingénieurs du numérique

**Machine automatique de traitement de l'information, obéissant à des programmes formés par des suites d'opérations arithmétiques et logiques.**

The diagram shows the Von Neumann architecture with the following components and connections:

- Mémoire** (Memory) is connected to the **Processeur** (Processor).
- The **Processeur** contains an **ALU** (Arithmetic Logic Unit) and a **Register**.
- The **Processeur** is connected to three **E/S** (Input/Output) units.
- A **Bus** connects the **Mémoire**, the **Processeur**, and the **E/S** units.
- An external **Keyboard** and **Monitor** are connected to the **E/S** units.

**Le bus permet l'échange d'information entre ces différents éléments**

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires      18/60

L'INFORMATION entre et sort de l'ordinateur par le biais des circuits périphériques qui permettent de communiquer avec le monde extérieur.

La MÉMOIRE contient les programmes ainsi que les données enregistrées par la machine ou utiles pour le traitement de l'information.

Le PROCESSEUR exécute les opérations logiques requises, les unes après les autres.

**ISEP** École d'ingénieurs du numérique

## Architecture Von Neumann

Les performances d'un calculateur dépendent essentiellement de :

- Son modèle de processeur
  - (Puissance de calcul - consommation...)
- La fréquence de fonctionnement
- La taille mémoire

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires 19/60

Avec l'architecture de VON NEUMANN UNE seule opération est exécutée à un instant donnée. Cette instruction est lue par le processeur dans mémoire puis exécutée.

**ISEP** École d'ingénieurs du numérique

## Architecture des ordinateurs

Pour continuer à décrire l'architecture VON NEUMANN il est nécessaire de décrire ce que sont :

- Le code binaire
- Les mémoires
- Les opérations arithmétiques et logiques
- Le codage des chiffres des symboles

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires 20/60

**isep** École d'ingénieurs du numérique

## Binaire

Dans les calculateurs, TOUTE l'information est codée en BINAIRE.  
(Sous forme de 0 et de 1).

**Avantages :**

- Les calculs sont simples à réaliser (cf partie arithmétique)
- Il y a une bonne IMMUNITÉ au BRUIT (cf électronique)
- Le stockage, le transfert d'information peut s'effectuer avec une consommation très réduite (cf électronique)

**Inconvénients:**

- Il faut beaucoup de cellules binaires pour représenter une information simple
- Il faut souvent utiliser des codes intermédiaires pour représenter l'information vis-à-vis des êtres humains

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires      21/60

Dans les circuits électroniques on code l'état 1 et l'état 0 par des niveaux de tension. Le « bruit » induit par couplage capacitif ou inductif avec des signaux proches ne devrait pas provoquer de confusion de niveaux.

Pour améliorer l'immunité au bruit, il est intéressant d'adopter des tensions élevées. (On utilise des tensions plus élevées pour les signaux de communication à l'extérieur des composants). (0-5v / 0-3,3v / ...)

Pour diminuer la consommation il est intéressant d'adopter des tensions faibles. (On utilise des tensions faibles à l'intérieur des composants). (0-1,8v / 0-1,2v / 0 – 0,9v / 0 – 0,45v / ....)

**isep** École d'ingénieurs du numérique

## Binaire

Tension V

Séries LV, LVC, LVT, ALVC

Voh Min	2,2 V
Vih Min	2 V
Vol Max	0,55 V
Vil Max	0,8V

**Pas de confusion de niveau**      Immunité au bruit maximum

**Puissance dissipée dans les transistors idéalement nulle**

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires      22/60

Avec des transistors parfaits, le courant est nul lorsqu'ils sont bloqués, la tension à leurs bornes est nul lorsqu'ils sont ouverts : Ils ne dissipent pas de puissance. Les logiques multivaluées permettent de mémoriser plus d'un bit par transistor (le transistor est ON / OFF / Semi-ON ...). Dans ce cas, l'immunité au bruit est plus faible, et le composant consomme de l'énergie.

La logique binaire permet également d'implémenter les opérations arithmétiques plus simplement.

## Binaire

La représentation binaire fera l'objet du chapitre suivant

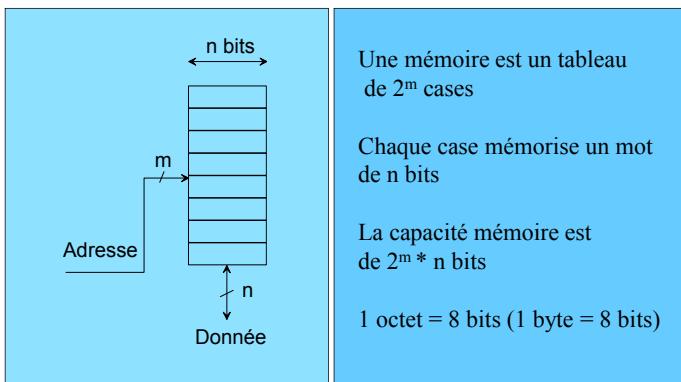
Les instructions du processeur seront codées en binaire

Les calculs seront effectués en logique binaire

Toutes les données des utilisateurs (images, sons, etc) seront stockées en binaire

Les circuits mémoire gardent l'information

## Mémoire - principe



D'un point de vue logique, une mémoire est organisée sous la forme d'un tableau de mots logiques. A un instant donné on ne peut lire ou écrire qu'à un seul endroit (mot) de ce tableau.

Les entrées sont donc : l'adresse, qui définit le numéro de la case que l'on souhaite lire ou écrire. La donnée, qui véhicule le mot lu ou écrit. Les informations de contrôle, qui permettent de savoir si on lit ou si on écrit, ainsi que les conditions de validation du composant.

**isep** École d'ingénieurs du numérique

## Architecture mémoire

Dans l'architecture VON NEUMANN le processeur échange des informations avec la mémoire et les périphériques par un BUS UNIQUE.

Le processeur voit la mémoire comme un espace continu de cases de taille fixe.

Chaque processeur est caractérisé par le nombre de cases mémoires qu'il peut « voir » et la taille de ces cases

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires      25/60

**isep** École d'ingénieurs du numérique

## Espace mémoire

Ce processeur dispose d'un espace total 16 bits : 0 - 65536 ( $2^{16}$ )

Activation → Mémoire0 De 0 à 32767      Mémoire1 De 32768 à 49151      Périf 1 De 61440 à 65536

Exemple : lecture d'une information

Le processeur demande le contenu de l'adresse 10234  
Le composant Mémoire 0 est sélectionné  
Le composant Mémoire 1 délivre le contenu de la case

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires      26/60

**isep** École d'ingénieurs du numérique

## Espace mémoire EXEMPLES

- Les processeurs des PC disposent d'un espace mémoire virtuel de 64 bits d'adresse (le bus mémoire fait en général 43 bits).
  - Combien de cases mémoires peuvent ils sélectionner ?
- Le processeur ARM cortex M4 a un bus d'adresse de 32 bits
  - Combien de cases mémoire peut il sélectionner ?
- Le processeur PIC 16F1454 a un bus d'adresse de 16 bits.
  - Combien de cases mémoire peut il sélectionner ?

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires      27/60

**ISEP**  
École d'ingénieurs du numérique

## Espace mémoire - processeur

Vu du processeur l'espace mémoire est continu

- En pratique cet espace adresse des composants différents. Parfois aucun composant ne répond à une adresse donnée
- C'est la carte mémoire de l'ordinateur :
  - Recherche : Gestionnaire de périphériques
  - → affichage → ressources par type → mémoire
- Parmi les composants il y a de la mémoire qui contient :
  - Des programmes
  - Les données

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires      28/60

**ISEP**  
École d'ingénieurs du numérique

## Carte mémoire

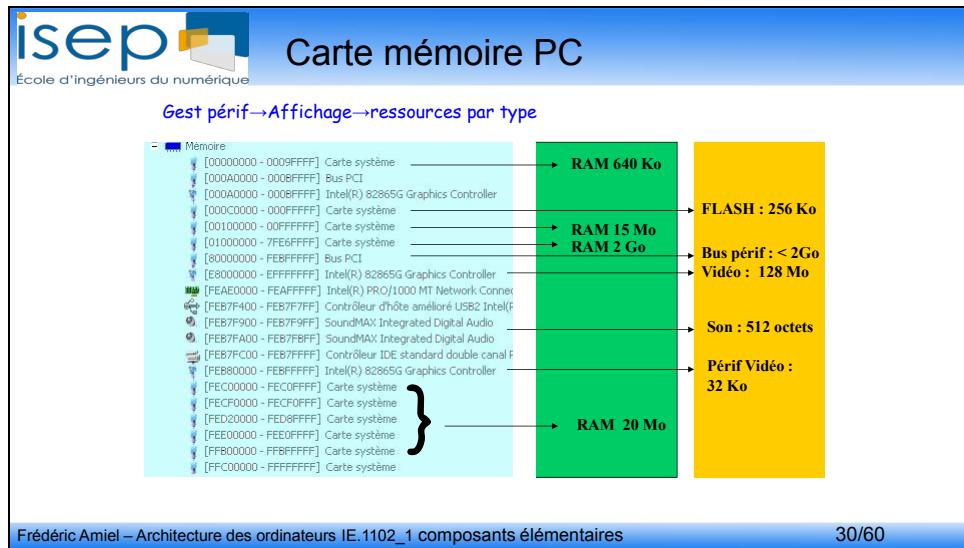
Les composants se partagent l'espace mémoire  
 Les zones d'adresses des composants ne se recouvrent pas  
 Chaque composant occupe plusieurs adresses mémoire

Exemple système élémentaire bus d'adresse de 16 bits

	ffff	c000	bfff	8000	7fff	0000	
FLASH							Vecteur RESET / IT
PERIPH							Programmes 16 Ko
SRAM							Zone d'E/S 16 Ko
							Mémoire vive : Tableaux / données ... 32 Ko

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires      29/60

La carte mémoire représente l'espace mémoire complet vu du processeur, avec les différentes zones occupées par les différents composants connectés au bus.

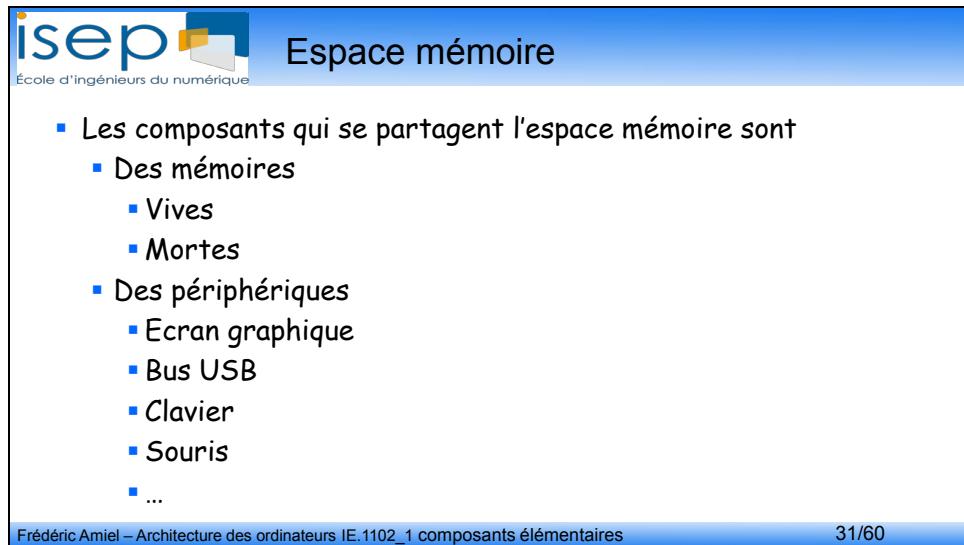


Les adresses générées par le processeur des PC sont définies ici sur 32 bits, ce qui permet de référencer  $2^{32} \sim 4.10^9$  cases différentes.

Différents composants sont attribués à différentes zones.

La zone « bus PCI » représente un bus périphérique permettant d'ajouter des cartes sur des connecteurs dans la machine. Si on insère une carte sur un de ces connecteur, un nouveau composant attribué à une plage d'adresse vient compléter la carte.

L'allocation d'une plage d'adresse à un composant peut s'effectuer de façon statique ou dynamique.



**Mémoires**

- Mémoires mortes
  - ROM
    - Rom - Eprom - Uveprom - Flash - EEPROM

Le contenu est préservé lors des coupures d'alim  
Le temps de lecture <> temps d'écriture

- Mémoires vives
  - SRAM
    - NVRAM
  - DRAM
    - DRAM - VRAM - SDRAM - DRAM EDO- RAMBUS...

Le contenu est perdu lors des coupures d'alim  
Le temps de lecture ~ temps d'écriture

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires      32/60

Les composants mémoires sont d'autres éléments constitutifs d'un ordinateur. On trouve des mémoires mortes et des mémoires vives à l'intérieur d'un processeur et plus généralement comme éléments nécessaire à l'organisation d'un ordinateur.

VRAM – dual ported RAM

NVRAM : No Volatile RAM (RAM avec une pile)

SDRAM - Synchronous dynamic random access memory (SDRAM)

Extended Data Out (EDO)

**Mémoire : Schéma externe**

CS : Chip Select : Le bus de donnée est en HiZ si cette entrée est inactive

R/W : Opération d 'écriture ou de lecture (RAM seulement)

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires      33/60

L'entrée CS permet de valider ou non le composant.

Si la mémoire est une RAM (lecture et écriture possibles), le signal R/S indique si l'opération en cours est une écriture ou une lecture. Si le composant est une ROM (lecture seulement), ce signal est remplacé par un OE (Output Enable) qui permet de valider le bus de donnée du composant en sortie. (Sinon, l'état est en TRI-STATE).

**Transistor MOS**

Dopage : <https://www.youtube.com/watch?v=7ukbKVHnac4> Transistor : [https://www.youtube.com/watch?v=tz62t-q\\_KEc](https://www.youtube.com/watch?v=tz62t-q_KEc)

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires 34/60

**INVERSEUR**

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires 35/60

**PORTE LOGIQUE NOR**

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

NOR

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires 36/60

**ISEP**  
École d'ingénieurs du numérique

## FLASH

From Computer Desktop Encyclopedia  
© 2005 The Computer Language Co., Inc.

**EEPROM and Flash Transistor**

The diagram illustrates the cross-section of an EEPROM cell. It shows a floating gate sandwiched between a control gate and a silicon dioxide insulator. The floating gate is connected to a source and drain electrode through metal tracks. Below the floating gate is a p-channel transistor, and above it is an n-type silicon substrate. The diagram also shows the n-channel and p-type silicon regions. A red arrow indicates the flow of electrons from the floating gate into the substrate during the erase process.

La grille flottante stocke les charges, ce qui contrôle le passage du courant entre le drain et la source du transistor

L'effet Fowler-Nordheim (effet tunnel) permet de faire transiter les charges à travers un isolant en appliquant des potentiels élevés

10 000 – 100 000 cycles d'effacement

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires      37/60

La technologie FLASH, initialement inventée par TOSHIBA permet de réinitialiser le contenu d'une zone mémoire par application d'un champ électrique. (Tension d'une vingtaine de volts).

**ISEP**  
École d'ingénieurs du numérique

## FLASH / EPROM

Dans tous les cas, il y a un processus d'effacement qui permet d'ôter les charges électriques de la grille flottante.

Dans la technologie EEPROM (Electricaly Erasable Programmable Read Only Memory) cet effacement peut s'effectuer case par case.

Dans la technologie FLASH, l'effacement s'effectue par blocs (ensemble de cases)

12	Effacement	12	Effacement
56	Programmation	56	Programmation
39		39	
45	EEPROM	0xFF	FLASH
93		0x45	
83		0xFF	
32		32	

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires      38/60

La modification d'une cellule mémoire requiert l'effacement de la zone concernée et la programmation des cellules effacées. Cette procédure est considérablement plus longue que l'opération de lecture.  
Avec cette technologie, le contenu du composant est préservé lors des arrêts d'alimentation.

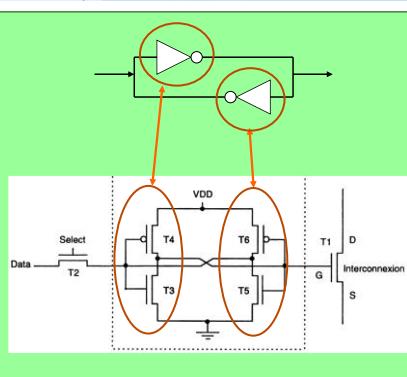
## ROM conclusion

- Gardent l'information lors des coupures d'alim
- $T_{prog} \gg T_{lecture}$ 
  - $T_{prog} = qq \text{ ms / mot}$
  - $T_{lect} = qq \text{ 10ns / mot}$
  - $T_{prog} = 100000 * T_{lect} (\text{EEPROM})$

Aujourd'hui, il n'existe pas de composants commerciaux permettant de conserver une information lors des arrêts d'alimentation et disposant d'une vitesse et d'un nombre de cycle d'écriture aussi rapide et important que la vitesse et le nombre de cycle de lecture.

Les composants NVRAM (non volatile ram) qui intègrent une pile de sauvegarde simulent cette fonction.

## SRAM



- SRAM
- 1 bit = 4 transistors
  - Rapide
  - Peu dense
  - $T_{write} = T_{read}$

Les cellules mémoires SRAM (Static Ram) mémorisent un bit avec 4 transistors. Cette technologie est plus dense et plus lente qu'un banc de registre. Elle permet de réaliser des composants de plusieurs millions de cases mémoires (32 M octets – 2006).

**ISEP**  
École d'ingénieurs du numérique

## DRAM

On utilise la capacité grille canal du transistor

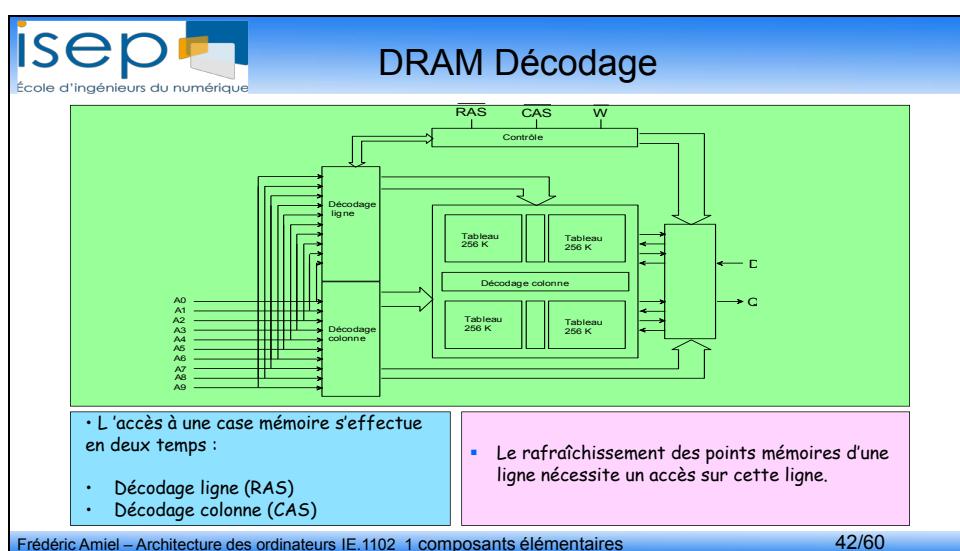
- Le point mémoire est un condensateur
- Le condensateur occupe la même taille qu'un transistor
- Le condensateur se décharge => refresh
- 1 bit = 1 transistor
- Dense

Fig. 11-4 A dynamic memory cell built with MOS transistors. The capacitor is the MOS transistor's input capacitance. A charge stored on this capacitor makes the cell hold a logic 1. No charge is a logic 0.

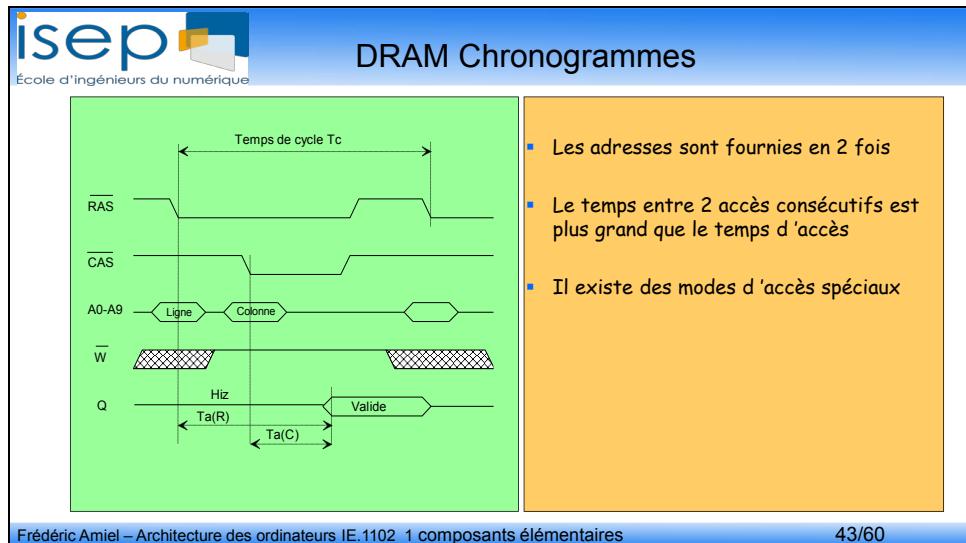
Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires

41/60

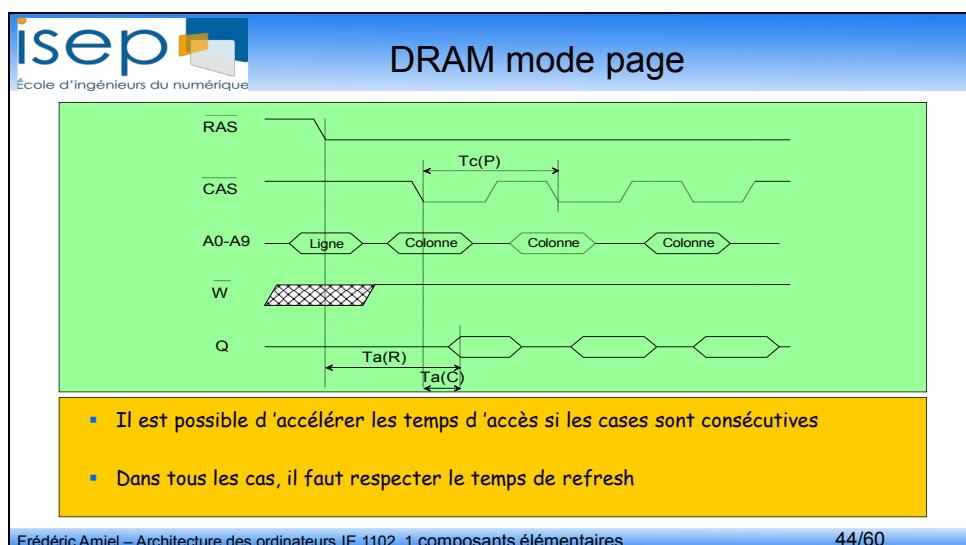
Les mémoires dynamiques utilisent un seul transistor pour mémoriser l'état d'un bit. En logique binaire, on ne peut pas faire plus dense. Plusieurs recherches essayent de réaliser une plus grande densité en utilisant des logiques à plusieurs états. Pour l'instant, il n'existe pas de produit commerciaux utilisant ce principe.



Le mot mémoire est sélectionné par une adresse ligne et une adresse colonne. D'autre part, comme le point mémoire est constitué par un condensateur qui se décharge, il faut « rafraîchir » la mémoire sous peine de voir les données disparaître. Ce rafraîchissement est automatique lors d'une opération de lecture. Les circuits de gestion des mémoires dynamiques gèrent cette opération qui diminue le débit mémoire possible.



L'adresse est fournie en deux fois : Adresse ligne et adresse colonne. Différents modes permettent d'accélérer le débit mémoire.



Par exemple, lorsqu'on lit 2 mots mémoires consécutifs, on peut conserver l'adresse ligne et ne modifier que l'adresse de la colonne. Le temps d'accès est alors plus rapide. C'est le processeur qui est alors chargé d'anticiper ses lectures mémoires en effectuant des accès « BURST ». (En réalité, c'est le gestionnaire de cache qui déclanche l'accès à une suite de mots consécutifs en mémoire).

**ISEP**  
École d'ingénieurs du numérique

## DRAM Intégration

8 puces de chaque coté

SDRAM - DDRAM (DDR2-DDR3) - DRAM EDO - RDRAM

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires

45/60

Ces composants mémoires ont une organisation très régulière. La puissance dynamique dissipée est d'autre part bien distribuée. Ils intègrent le maximum de transistors possibles de la technologie.

Aujourd'hui, un banc mémoire de PC consomme environ 100w (plus de 20 Ampères).

**ISEP**  
École d'ingénieurs du numérique

## FeRAM

### Mémoire FerroElectriques

Titano-Zirconate de Plomb

- Same density as DRAM
- Low power
- $10^{14}$  cycles

Example: MSP430Fr5720  
16KB FeRAM

La mémoire FRAM consomme  $9 \mu\text{A}$  pour 12 Ko/s  
La mémoire Flash consomme  $2\,200 \mu\text{A}$  pour 12 Ko/s (TI)

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires

46/60



## RAM Conclusions

- Ne retiennent pas l'information en cas de coupure de l'alimentation.
- Temps d'écriture ~ temps de lecture
- Les DRAM nécessitent un rafraîchissement
- Les DRAM disposent de mode d'accès rapides

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires      47/60

Les NVRAM intègrent une pile lithium sur le composant, ce qui permet de simuler le comportement d'une ROM disposant d'une grande rapidité d'écriture. La durée de rétention est alors d'environ 10 ans.



## Hiérarchie mémoire

- Les registres sont les plus rapides. On les trouve au cœur des unités de calcul
- Les mémoires statiques sont utilisées :
  - Dans les systèmes ne nécessitant pas beaucoup de mémoire (et chez les militaires pour des raisons de sécurité de fonctionnement)
  - Comme mémoire cache rapide (L1-L2)
- Les mémoires dynamiques sont utilisées comme mémoire centrale (principale)
- Les disques magnétiques sont considérablement plus lents que les mémoires à semi-conducteurs
- Les mémoires mortes contiennent les programmes d'initialisation (exécutés lors de la mise sous tension).

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires      48/60

Ces différents types de mémoires sont utilisés pour des fonctions différentes dans un ordinateur.

Les mémoires dynamiques étant les plus denses, on les utilise dans les gros systèmes (plusieurs centaines de Méga Octets de mémoire) comme composant de mémoire centrale.

Selon la fréquence horloge (la performance de l'ordinateur), les mémoires dynamiques ne peuvent fonctionner à la vitesse du processeur. On utilise alors une hiérarchie mémoire (cours ARC 201).

Les disques magnétiques sont considérablement plus lents que ces composants, ils sont gérés comme des entrées sorties. (ARC 201).

**isep**  
École d'ingénieurs du numérique

## Naissance du microprocesseur

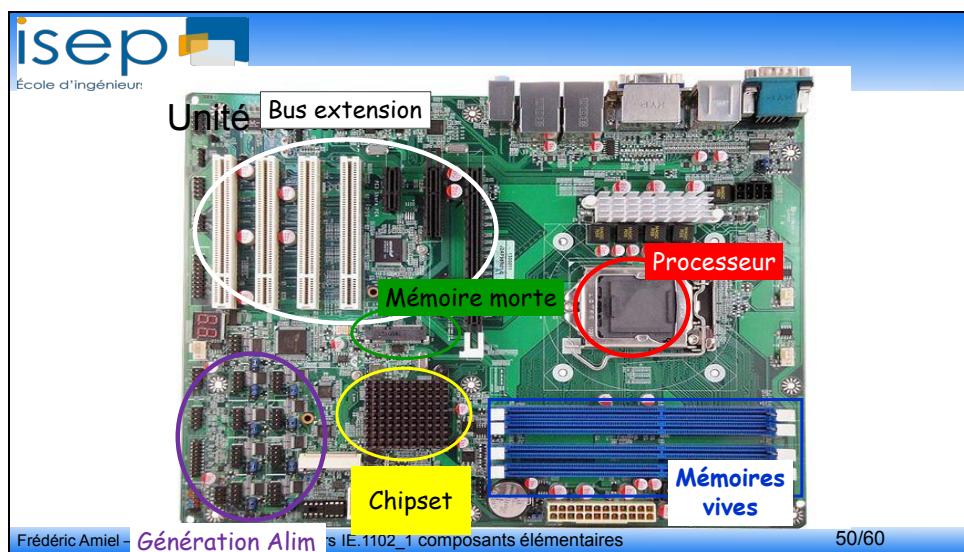
Logicom, une société japonaise commande la conception des composants d'une calculatrice à une société américaine.

**4004**  
1971  
4 bits  
2 300 transistors  
60 000 opérations / secondes.  
**INTEL**

Ted Hoff, Stan Mazor and Federico Faggin; and patent no 3 753 011

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires      49/60

Le transistor est né en 1949. Au début des années 60, on intègre plusieurs transistors sur une même puce de silicium ce qui constitue les premiers circuits intégrés. EN 1971, INTEL réalise le premier microprocesseur.



Sur une carte mère actuelle, les différents éléments qui constituent l'ordinateur sont repérables. Le processeur gère l'ensemble des opérations. La mémoire vive contient les programmes et les données en cours. La mémoire morte permet le démarrage du système. Le chipset regroupe différents composants permettant les entrées sorties (échanges avec l'extérieur).

ISEP  
École d'ingénieurs du numérique

micropuceur

```

/* divide D by r using shifts and odds */
if (in_r < (1 << (B_bits - F_bits - 1)))
    nShifts = F_bits;
else
    nShifts = F_bits - 1;
A = in_D;
M = in_r << nShifts;
target = 0;
for (i = nShifts; i >= 0)
{
    if (A >= M)
    {
        A -= M;
        target++;
        if (i == 0) break;
        A <<= 1; target <<= 1;
    }
    else
        M = total << (B_bits - F_bits - 1);
    if (A >= M) (A -= M; in_r += 1);
}
#define UNROLL_MJM B_bits - 1
#define UNROLL_CODE A << 1; in_r <<= 1;
if (A >= M) (A -= M; in_r += 1);
#include "unroll.h"
A = in_D;
target = 0;
if (in_r < (1 << (B_bits - F_bits - 1)))
    (M = in_r << F_bits;
     if (A >= M) (A -= M; target++,
                  A <<= 1; target <<= 1;
     )
    else
        (M = in_r << (F_bits - 1);
    )

```

IC 9: fulladder > fulladder [0]

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires

51/60

Le programme exécuté par l'ordinateur est placé dans la mémoire vive. Ce programme est lu par le processeur. Ce processeur est décrit sous la forme de macro blocs, qui sont réalisés par des ensembles de portes logiques. Ces portes logiques sont elles mêmes constituées par des assemblages de transistors.

ISEP  
École d'ingénieurs du numérique

## ALU

ALU

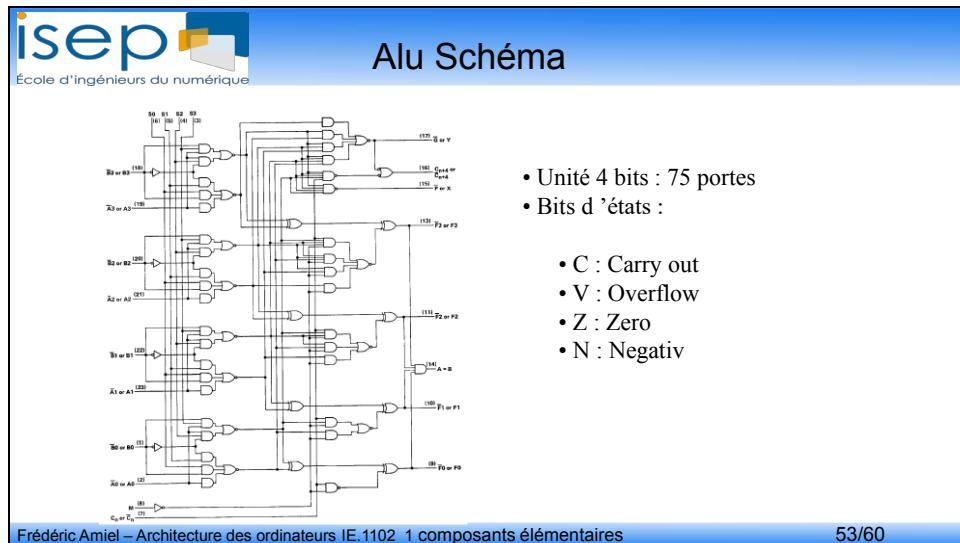
A      n  
B      n  
Code Op      m  
Flags  
OUT

- Unité Arithmétique et Logique
- Circuit purement combinatoire
- Opérations arithmétiques et logiques

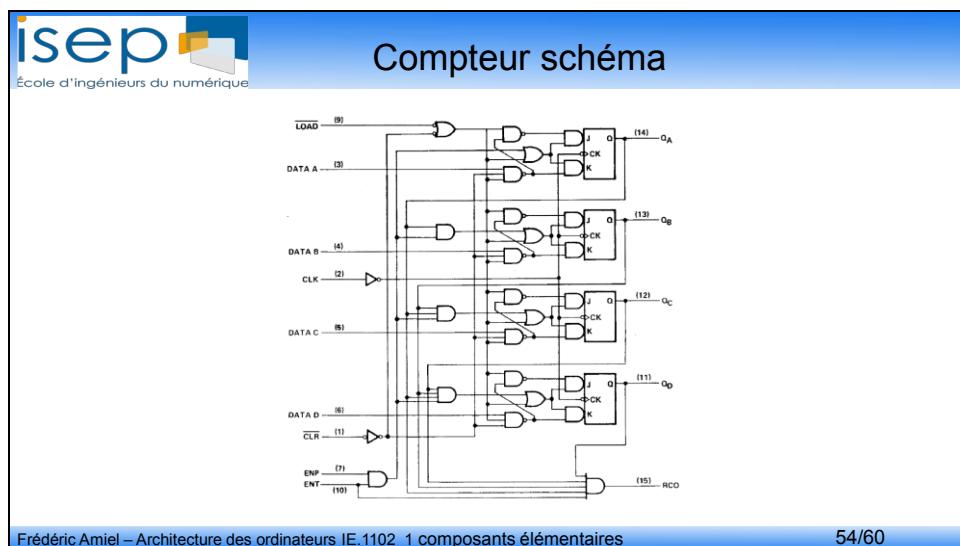
Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires

52/60

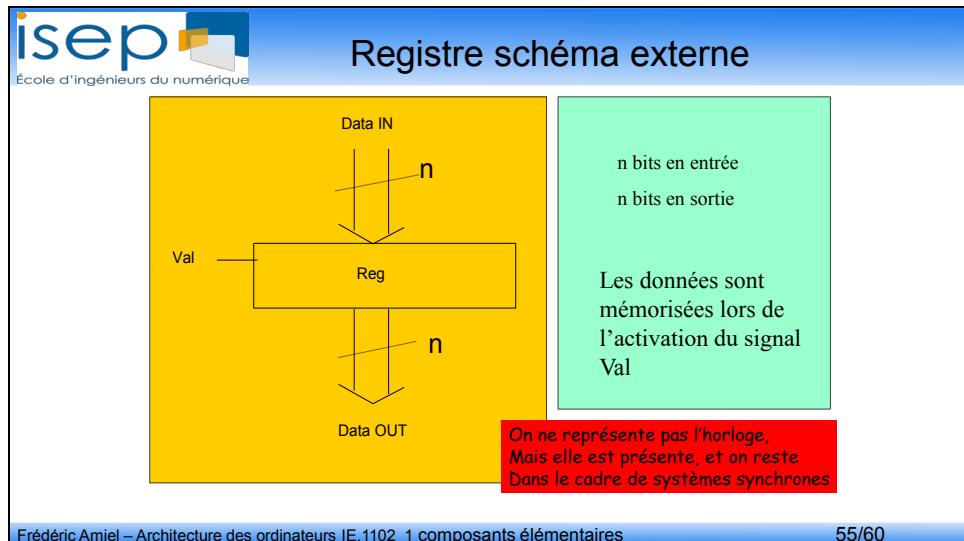
Parmi les blocs qui constituent un processeur, on trouve l'unité arithmétique et logique. Ce composant combinatoire permet de réaliser des opérations sur des nombres codés en binaire ou en complément à deux.



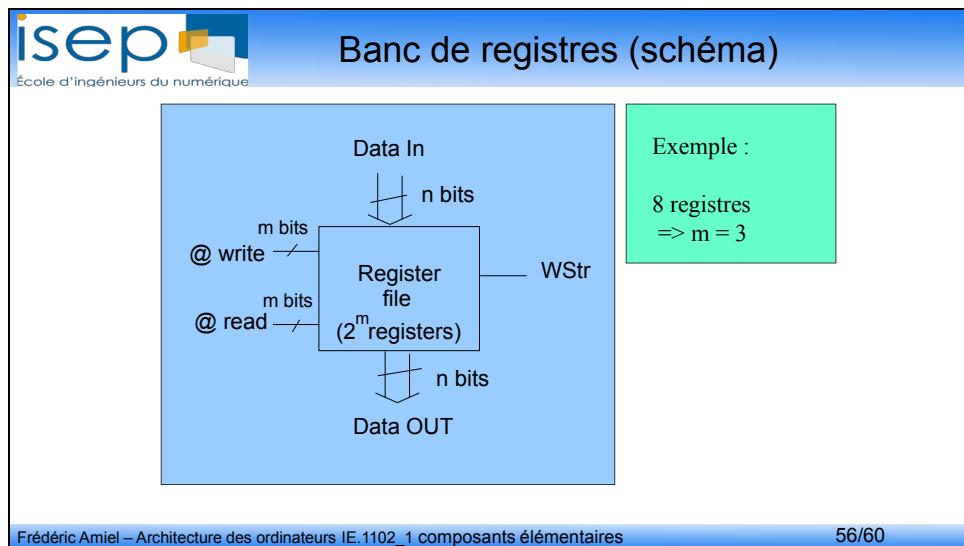
Le schéma logique d'une ALU classique (sans multiplicateur), ne comporte pas beaucoup de portes logiques.



Tous les éléments séquentiels que l'on trouve à l'intérieur d'un processeur sont construits en logique synchrone. La sortie est établie après un temps fixe qui suit le front de l'horloge. Ce type de conception permet de borner de façon absolue la fréquence de l'horloge.



On simplifie la représentation de ces éléments en utilisant un schéma synoptique. On fait en général abstraction du signal d'horloge sur cette représentation. Ce signal est pourtant bien présent et très important au niveau inférieur de représentation.



Ce banc de registre est également conçu en logique synchrone.

**ISEP**  
École d'ingénieurs du numérique

## Complexité

Banc de registres :

4 registres 8 bits => 1000 transistors

Vitesse :

- Lecture :  $T_{mux}$
- Ecriture :  $T_{mux} + t_{su} + t_h + t_w$

=>

- rapide
- Peu dense

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires      57/60

Les bancs de registres permettent de mémoriser des mots binaires. Chaque bit est mémorisé dans une bascule D elle-même constituée d'une vingtaine de transistors. Cette technique très rapide est peu dense : les composants mémoires permettent de mémoriser des bits avec moins de transistors.

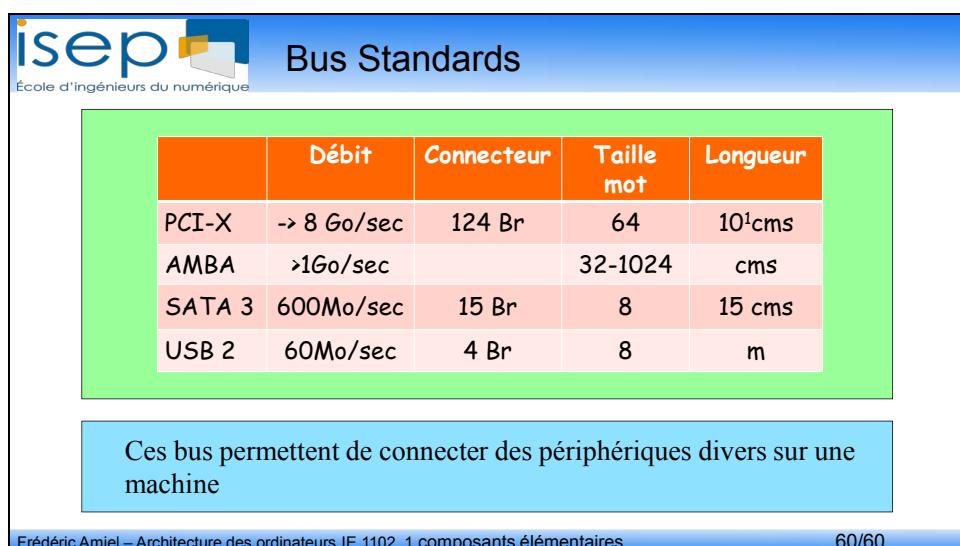
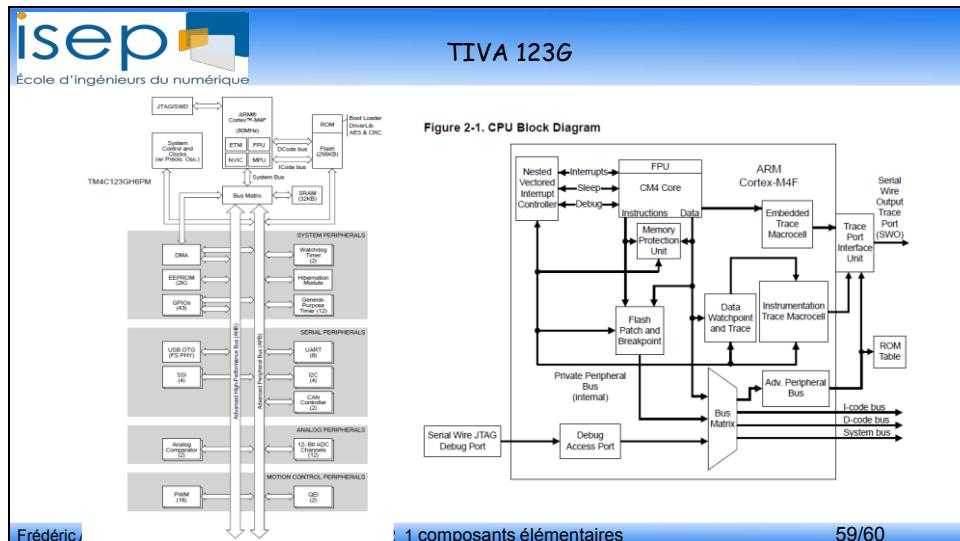
**ISEP**  
École d'ingénieurs du numérique

## Les BUS

- Les bus assurent le transfert de l'information d'un composant à l'autre
  - Bus = multiplexeur
- Il y a un seul écrivain et un seul lecteur à un instant donné (pas de diffusion).

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires      58/60

Les bus permettent de transférer l'information d'un composant vers un autre. Le bus principal se décompose en « sous bus » : Bus d'adresse, bus de données, bus de contrôle.  
 La description du bus comprend la définition et les chronogramme d'un grand nombres de signaux. (ARC201).



Le bus PCI : peripheral Component Interconnect permet d'ajouter des périphériques sur une carte mère par exemple.

Le bus AMBA est un bus interne à un composant (c'est le bus utilisé par le processeur ARM).

Le bus SATA (Serial ATA) permet la connexion aux périphériques d'entrées / sorties rapides (disque dur, lecteur DVD...).

USB 2 échange à 480 Mbits/sec et USB3 peut monter à 600 Mo/sec (mode High Speed)

**isep**  
École d'ingénieurs du numérique

## Bus processeur

- Spécifique à chaque processeur
  - Desktop (INTEL) :
    - Bus interne : 24 Go /sec
    - Bus externe :  $800 * 8 = 6,4$  Go/s
  - ARM Cortex M3:
    - 320 Mo/sec
  - ATMELE ATmega328 (Arduino)
    - 16Mo/sec

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires      61/60

Suivant les processeurs les besoins en communication d'un élément à l'autre sont très différents. Le débit du bus est une des mesures de la performance d'un système.

**isep**  
École d'ingénieurs du numérique

## Auto test 1/2

- Citer un type de mémoire vive, un type de mémoire morte.
- Les SRAM sont plus rapides que les DRAM ?
- Les programmes de démarrage des ordinateurs sont dans des DRAM ?
  - Pourquoi ?
- A quelle condition peut-on connecter des sorties logiques ensembles ?
- Une ALU est un composant séquentiel ?
- Les mémoires SRAM sont complexes à utiliser ?
- Quel avantage y a-t-il à utiliser des portes 3 états / multiplexeurs ?
- On peut utiliser une ROM pour réaliser des fonctions combinatoires ?

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires      62/60

Les mémoires ROMs peuvent être utilisées comme composants combinatoires en utilisant les broches d'adresses comme des entrées, et le bus de donnée comme sortie.



ISEP  
École d'ingénieurs du numérique

## Auto – test 2/2

- Une mémoire a 20 bits d'adresse et des mots de 16 bits. Quelle est sa capacité en Ko ou Mo ?
- Avec un cycle de transfert sur 2 périodes horloge à 40 MHz, un bus de 32 bits, quelle est le débit du bus en Mo/sec ?

Frédéric Amiel – Architecture des ordinateurs IE.1102\_1 composants élémentaires

63/60

La capacité des mémoires est généralement donnée en octets indépendamment de la taille du mot mémoire.

Le rythme d'échange du bus est fixé par l'horloge du système. Selon le bus et la rapidité du composant sélectionné un échange de données dure 1, 2, ou bien plus de période horloge..

## Microprocesseur

IE.1102\_2

Codage des nombres – Eléments  
d’architecture des ordinateurs

IE.1102\_2 : Rappels arithmétiques – Architecture

EAMIET

1/59

## Objectifs

- Connaître les précisions et les différents formats de nombres
- Comprendre l’architecture de base d’un ordinateur
- Savoir ce qu’est le modèle de programmation d’un processeur

IE.1102\_2 : Rappels arithmétiques – Architecture

EAMIET

2/59

## Définitions

- Bit : Unité élémentaire d’information [0 ou 1]
- Byte : octet : 8 bits
- Digit : Un symbole de la base (Nombre sur 2 digits)
- Hexadécimal : base 16, les digits sont 0-9 a-f
- 0x : marque de la base 16 :  $0x10 = 16$

IE.1102\_2 : Rappels arithmétiques – Architecture

EAMIET

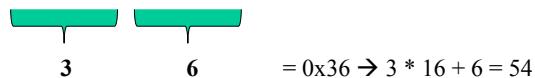
3/59

## Binaire

- Le code binaire permet de coder des nombres entiers naturels.

$$2^7 \ 2^6 \ 2^5 \ 2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0$$

$$0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0 = 32 + 16 + 4 + 2 = 54$$



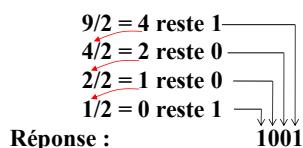
Binary representation of 54: 0 0 1 1 0 1 1 0  
 3                6  
 $= 0x36 \rightarrow 3 * 16 + 6 = 54$

**La notation hexadécimale est plus dense  
 Il est facile de passer de la base 2 à la base 16 et vice versa**

## Conversions décimal / binaire

Le reste des divisions successives par 2 donne les bits à partir de la droite.

**Exemple : convertir 9 en binaire :**



9/2 = 4 reste 1  
 4/2 = 2 reste 0  
 2/2 = 1 reste 0  
 1/2 = 0 reste 1  
 Réponse : 1001

## Hexadécimal

La notation hexadécimale permet d'écrire des nombres en base 16.  
 Pour cela on utilise les symboles :

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Le nombre 10 s'écrit 0xa en hexadécimal  
 Le nombre 17 s'écrit 0xb en hexadécimal

0x indique que le  
 nombre est écrit en  
 base 16

## Hexadécimal

- $0000\ 1000_b = 0x08 = 0x8 = 8$
- $0010\ 1100_b = 0x2c = 2*16 + 12 = 44$
- $0x6d = 0110\ 1101_b$   
 $= 6*16 + 13 = 69$
- $137 = 8*16 + 9 = 0x89 = 1000\ 1001_b$

IE.1102\_2 : Rappels arithmétiques – Architecture

FAMIL

7/59

## Complément à deux

- On construit les nombres négatifs de façon à utiliser les mêmes opérateurs qu'en binaire :

$$x + (-x) = 0$$

Le nombre de bit est défini à l'avance.

C = Carry :  
N'a pas de sens en complément à 2

Exemple : sur 8 bits :

$$1 + (-1) = 0 \quad \text{Or } 0000\ 0001_b + 1111\ 1111_b = 1\ 0000\ 0000_b$$

$$\begin{array}{r} 1 \\ + \\ -1 \\ \hline 0 \end{array}$$

IE.1102\_2 : Rappels arithmétiques – Architecture

FAMIL

8/59

## Analogie : Complément à 10

On considère des nombres en base 10 codé sur un digit :

- 0 représente 0  
 1 représente 1  
 2 représente 2  
 3 représente 3  
 4 représente 4...

$$x + -x = 10 \quad 1 + (-1) = 0 \quad 1 + 9 = 10$$

9 représente -1

- 9 représente -1  
 8 représente -2  
 7 représente -3  
 6 représente -4  
 5 représente -5...

Exemple :  $4 + 4 + 4 - 5 - 5 = ?$

$$4 + 4 + 4 + (-5) + (-5) = ?$$

$$4 + 4 + 4 + 5 + 5 = ?$$

$$\begin{array}{r} 4+4+4+5+5=22 \\ \swarrow 8 \rightarrow -2 \quad \searrow -2+4=2 \\ 2+5+5=12 \\ 7 \rightarrow -3+5=2 \end{array}$$

IE.1102\_2 : Rappels arithmétiques – Architecture

FAMIL

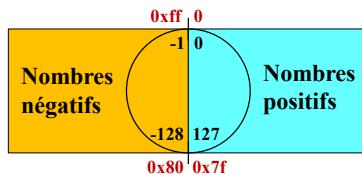
9/59

## Complément à deux

Exemple sur 8 bits :  $27 = 0001\ 1001_b = 0x1b$

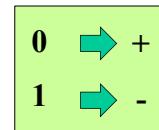
$$27 + (-27) = 1\ 0000\ 0000_b = 256 = 0x100$$

$$-27 = 0x100 - 0x1b = 0xe5$$



## Complément à 2

- 8 bits :  
 $-128 \leq n \leq 127 \leftrightarrow 0x80 \leq n \leq 0x7f$
- 16 bits  
 $-32768 \leq n \leq 32767 \leftrightarrow 0x8000 \leq n \leq 0x7fff$
- 32 bits  
 $-2\ 147\ 483\ 648 < n < 2\ 147\ 483\ 647$   
 $\leftrightarrow 0x8000\ 0000 \leq n \leq 0x7fff\ ffff$
- Le bit le plus à gauche porte le signe



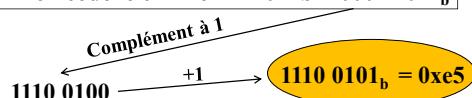
## Calcul de l'opposé en /2

- Passage d'un nombre à son opposé :

Sur 8 bits :

$$\begin{aligned} \text{codage de } (-n) &= 0x100 - n \\ c(-n) &= 0xff - n + 1 \\ &= \text{not}(n) + 1 \end{aligned}$$

**Exemple : représentation de -27 on code le chiffre  $27 = 0x1b = 0001\ 1011_b$**



## Passage d'une précision à une autre

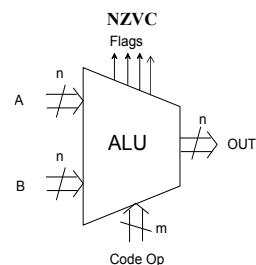
En complément à deux, le bit de signe est infini à gauche.

Décimal	Hexa 8 bits	16 bits	32 bits
123	0x7B	0x007B	0x0000007B
-123	0x85	0xFF85	0xFFFFF85

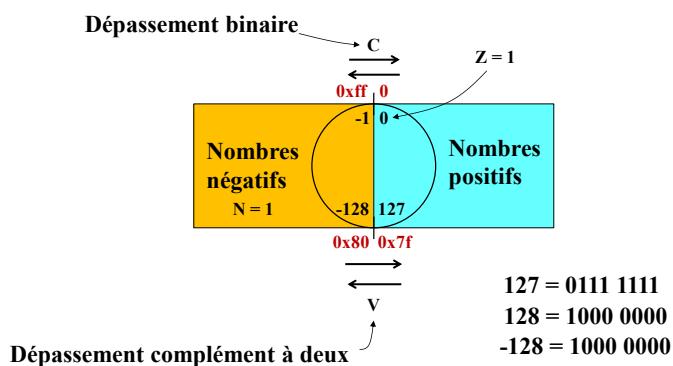
## Indicateurs

- Lors d'une opération, l'unité arithmétique et logique génère des indicateurs :

- N : Indique que le résultat est négatif
- Z : Indique que le résultat est nul
- V : Indique un débordement en /2
- C : Indique une retenue (binaire)



## Drapeaux (8 bits)



## Drapeaux – implémentation des tests

**if (n < 6)**        **n - 6**    **Test du drapeau N**

...

**Exemple n = 4**        **n - 6 = -2 et N est positionné**

**if (n == 6)**        **n - 6**    **Test du drapeau Z**

...

**if (n >= 6)**        **n - 6**    **Test des drapeaux Z OU /N**

...

**Exemple n = 9**        **n - 6 = 3 et /N est positionné**

**Exemple n = 6**        **n - 6 = 0 et Z est positionné**

## Nombres à virgule

On représente les nombres en base 2 en utilisant la notation à position :

Décimal	2 <sup>2</sup>	2 <sup>1</sup>	2 <sup>0</sup>
5	1	0	1

Avec la notation à position, on peut écrire des chiffres fractionnaires en base 2 :

Décimal	2 <sup>2</sup>	2 <sup>1</sup>	2 <sup>0</sup>	2 <sup>-1</sup>
5,5	1	0	1	1

Certains nombres requièrent un nombre infini de digits

Décimal	2 <sup>2</sup>	2 <sup>1</sup>	2 <sup>0</sup>	2 <sup>-1</sup>	2 <sup>-2</sup>	2 <sup>-3</sup>	2 <sup>-4</sup>	2 <sup>-5</sup>
3,3	0	1	1	0	1	0	1	0

= 3,3125

Selon le nombre de bits utilisés on a une approximation du nombre

## Virgule flottante

Tous les nombres peuvent s'exprimer sous la forme suivante :

$$353 = 3.53 \times 10^2$$

Dans cet exemple, 3,53 s'appelle la mantisso  
Le chiffre 2 s'appelle l'exposant

Le chiffre 10 est alors la base d'usage.

## Virgule flottante

En base 2 on note les nombres sous la forme :

$$1, \text{mantisso} * 2^{\text{exposant}}$$

$$3,5 \rightarrow 11,1 \text{ en base 2}$$

$$3,5 \rightarrow 1,11 * 2^1 \rightarrow \begin{array}{l} \text{Mantisso : } 111_b \\ \text{Exposant : } 2 \rightarrow 1_b \end{array}$$

## Virgule Flottante

Normalisé IEEE 754

8 bits exposant en complément à 2

23 bits mantisse (23 + 1)

1 bit de signe

Seeeeeeeemmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm

S : Signe mantisse : 0 → positif / 1 → négatif

e : exposant en binaire décalé (n+127)

m : mantisse

$$C = 1, \text{mmmm..} * 2^{\text{exp}}$$

Mantisso : Binaire avec bit de signe

Exposant : Binaire décalé

$$3,5 \rightarrow 1,11 * 2^1 \rightarrow 1,1100000000000000000000000000000 * 2^{10000000}$$

Inutile de noter ce 1

$$3,5 \rightarrow 0\ 1000000\ 1100000000000000000000000000000$$

## Virgule Flottante

Les opérations élémentaires + - \* / deviennent plus complexes

Addition (par exemple 87 + 6742)

$$8,7 \cdot 10^1 + 6,742 \cdot 10^3 \rightarrow \text{Les exposants sont différents} \rightarrow \text{Alignement des mantisses}$$

$$1 \neq 3 \rightarrow 3 - 1 = 2 \rightarrow 8,7 \cdot 10^1 + 674,2 \cdot 10^1$$

$$8,7 \cdot 10^1 + 674,2 \cdot 10^1 \rightarrow \text{Opération} \rightarrow 682,9 \cdot 10^1 \rightarrow \text{Normalisation}$$

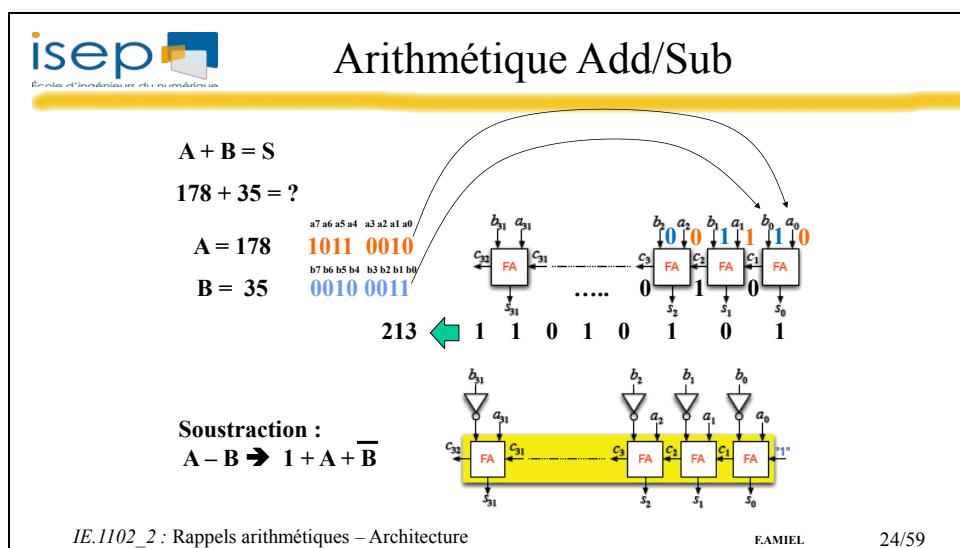
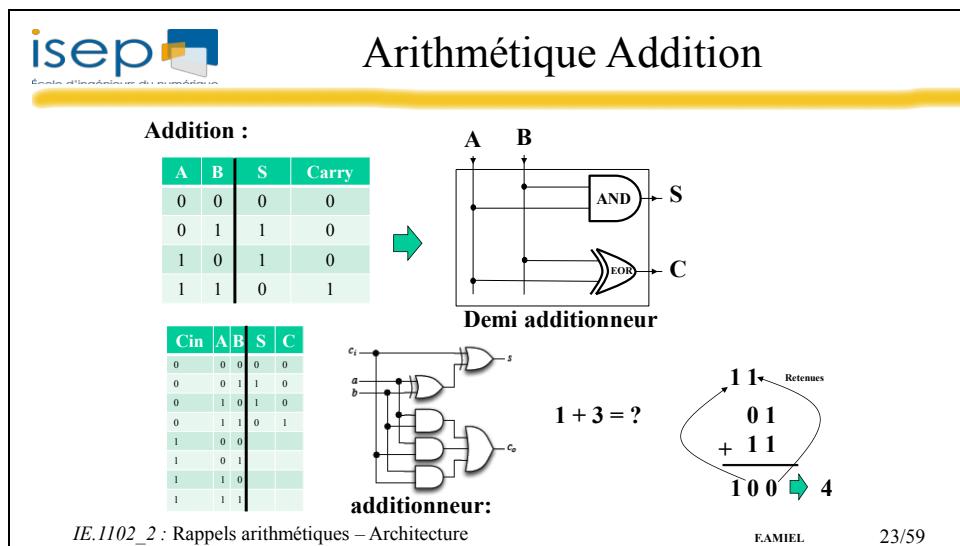
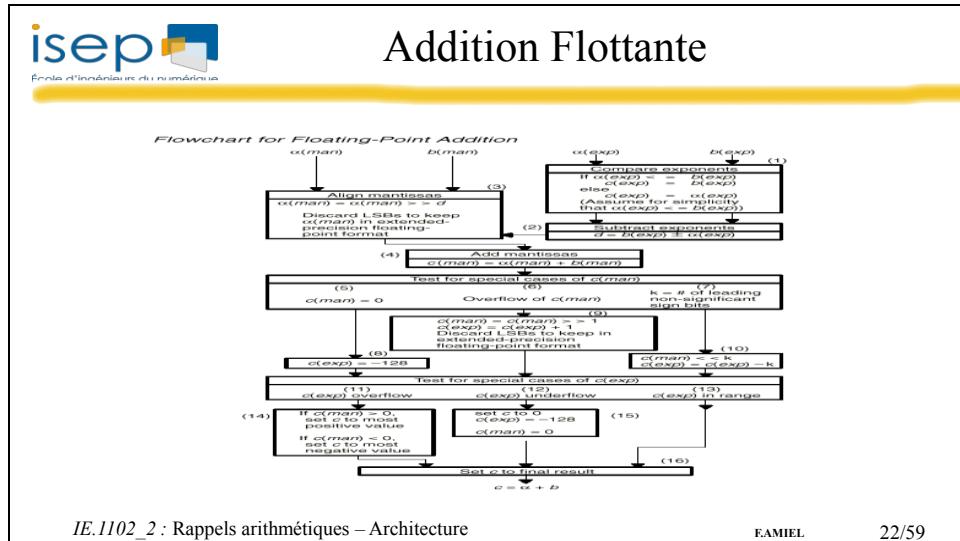
On décale le chiffre de 2 positions

**Virgule flottante :**

Alignement des mantisses → Opération → Normalisation

**Complément à 2 :** Opération

Le calcul en virgule flottante demande plus d'opérations élémentaires que le calcul entier



**isep** École d'ingénieurs de l'université

## Multiplication

$A * B = S$

$178 * 35 = ?$

$A = 178$        $B = 35$

$178 = \begin{array}{r} a_7 \ a_6 \ a_5 \ a_4 \ a_3 \ a_2 \ a_1 \ a_0 \\ 1 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \\ b_7 \ b_6 \ b_5 \ b_4 \ b_3 \ b_2 \ b_1 \ b_0 \\ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 1 \end{array}$

$35 = \begin{array}{r} 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 1 \\ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \\ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \\ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \\ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \\ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \\ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \end{array}$

**Table de vérité**

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

**AND**

**Addition**

**La multiplication est une succession de ET logique et d'ADDITION**

**6230**

IE.1102\_2 : Rappels arithmétiques – Architecture      FAMILIEL      25/59

**isep** École d'ingénieurs de l'université

## Arithmétique saturée

- Avec cette représentation un débordement de l'arithmétique génère des perturbations importantes sur le signal
- On peut configurer l'unité de calcul pour saturer les chiffres : A appliquer lors des programmes de traitement du signal

IE.1102\_2 : Rappels arithmétiques – Architecture      FAMILIEL      26/59

Le débordement des calculs amènerait des diracs dans le signal. Les unités de calculs des DSP doivent gérer la saturation.

**isep** École Supérieure de l'Informatique

## ALU – Différents processeurs

<b>De base : Nombres entiers</b>	<b>Sous programmes</b>
<ul style="list-style-type: none"> <li>• addition</li> <li>• soustraction</li> <li>• opérations logiques</li> </ul>	Multiplication Division Opérations flottantes
<b>Plus avancé : Nombres entiers</b>	<b>Sous programmes</b>
<ul style="list-style-type: none"> <li>• addition</li> <li>• Soustraction</li> <li>• multiplication</li> <li>• opérations logiques</li> </ul>	Division entière Opérations flottantes
<b>Plus avancé : Nombres entiers et nombres flottants</b>	<b>Plus avancé : Opérations vectorielles</b>
<b>La division n'est pas toujours câblée</b>	<b>SIMD</b>

IE.1102\_2 : Rappels arithmétiques – Architecture      FAMILI      27/59

Selon le modèle, les processeurs disposent de plus ou moins d'opérations réalisées par le matériel. Sur les modèles les plus performants, l'unité d'exécution comprend plusieurs opérateurs arithmétiques câblés capables de traiter des nombres flottants.

La différence entre un opérateur câblé et une opération réalisée par un sous programme est que le temps d'exécution est bien plus grand lorsque l'opération est réalisée par logiciel.

**isep** École Supérieure de l'Informatique

## Auto test

- Donner un chiffre  $> 0x92$  et un autre  $< 0x92$  en complément à 2 sur 8 bits.
- Coder le chiffre -18 en complément à 2  $18 \rightarrow 0x32 \rightarrow 18 \rightarrow 1100\ 1101\_{\text{+}2}$
- Dans l'opération  $254 + 2$  quels sont les états des indicateurs N Z V C ?
- Dans l'opération  $130 - 12$  quels sont les états des indicateurs NZVC ?  $130 - 12 \rightarrow 1100\ 1100 \rightarrow \text{N} \& \text{V} \& \text{C}$
- Coder le chiffre 4,78 en binaire  $0.00$ ,  $- 11,625 \rightarrow 1011, 101000$
- Coder le chiffre -11,625 en flottant avec la norme IEEE754  $0.25 \rightarrow 0.01000... \times 2^2$
- Coder le chiffre 0,25 avec la norme IEEE 754  $0.25 \rightarrow 10000 \times 2^2$
- Quel est le code ASCII du caractère « 7 »
- Quel est le caractère de code ASCII 64 ?
- Coder le chiffre 4.75 en virgule fixe puis en virgule flottante
- Coder le chiffre -1,25 en virgule fixe puis en virgule flottante

IE.1102\_2 : Rappels arithmétiques – Architecture      FAMILI      28/59

# Code ASCII

## Pd Faible

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Pd	0 NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
F	1 DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
O	2 SP	!	"	#	£	%	&	'	( )	*	+	,	-	.	/	
R	3 0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
T	4 @	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
	5 P	Q	R	S	T	U	V	W	X	Y	Z	[ \ ]	^	-		
	6 `	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
	7 p	q	r	s	t	u	v	w	x	y	z	{   }			~	DEL

SOH : Start of head   SOT : Start of text

DC1 : Dev ctrl 1 (XON)

DC2 : Dev Ctrl 2

ETX : End of text   EOT : End of trans

DC3 : Dev ctrl 3 (XOFF)

DC4 : Dev Ctrl 4

ENQ : Enquiry   ACK : Acknoledge

NAK : Negativ Ack   SYN : Synchronous Idle

BEL : Bell (Beeper)   BS : Back Space

ETB : End Trans Block

CAN : Cancel

HT : Horizontal Tab   LF : Line Feed

EM : End of medium

SUB : Substitute

VT : Vertical Tab   FF : Form Feed

ESC : Escape   FS : File separator

CR : Carriage Return   SO : Shift Out

GS : Group Separator

RS : Record Separator

SI : Shift In   DLE : Data link escape

US : Record Separator

US : Unit Separator

DEL : Delete   SP : Caractère blanc (Espace)

**isep** École d'ingénieurs et de managers

## Architecture de Von Neumann

The diagram illustrates the Von Neumann architecture. It features a central horizontal bus labeled "Bus unique". Three main components are connected to this bus: "Processeur" (Processor), "Mémoire unique" (Unique Memory containing "Prog + Données"), and "Périphériques" (Peripherals). Below the bus, the text "Echanges" (Exchanges) is written.

Les codes programmes sont placés au même endroit que les données (opérandes).

La mémoire contient les instructions et les opérandes

Le processeur est le maître du bus. C'est lui qui séquence les échanges

IE.1102\_2 : Rappels arithmétiques – Architecture FAMIL 29/59

**isep** École d'ingénieurs et de managers

## Architecture de Von Neumann

The diagram shows a detailed view of the Von Neumann architecture. The "Processeur" (Processor) sends "Adresses" (Addresses), "Données" (Data), and "Contrôle" (Control) signals to the "Decod" (Decoder). The "Decod" provides control signals to "Mémoire 1", "Mémoire 2", and "E/S" (Input/Output). "Mémoire 1" and "Mémoire 2" provide data to the "Processeur". The "Plan mémoire" (Memory map) is shown as a table:

Adresse mémoire	FFFF	ROM
0000		RAM
		I/O
		LIBRE

Le processeur écrit les adresses  
Le processeur lit ou écrit les données  
Le processeur gère des signaux de contrôle qui régulent les transferts d'information

IE.1102\_2 : Rappels arithmétiques – Architecture FAMIL 30/59

**isep** École d'ingénieurs et de managers

## Vue du processeur

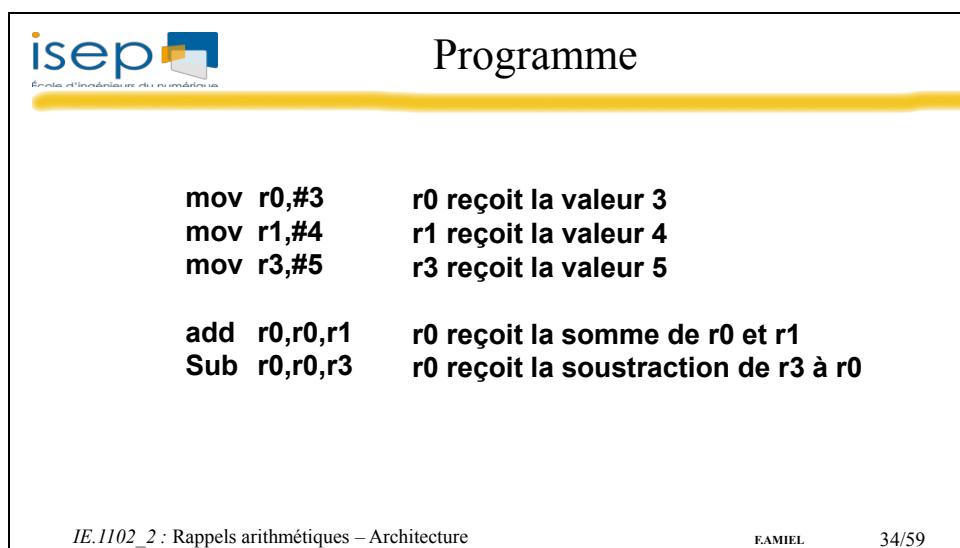
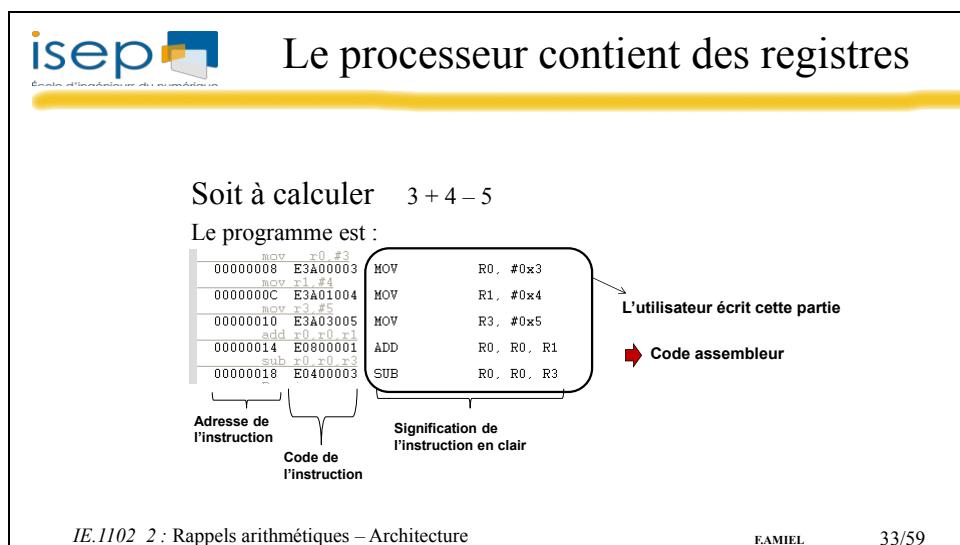
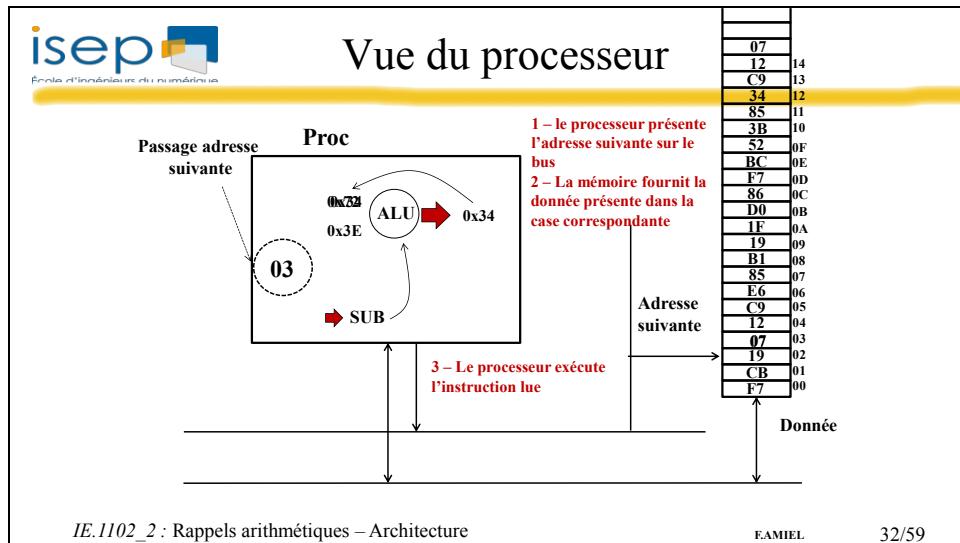
The diagram illustrates the internal operation of the processor. It shows the "Proc" (Processor) performing an "ADD" operation. The ALU takes two values from memory (0x32 and 0x3E) and produces the result 0x72. The processor presents the address on the bus. The memory provides the data at the corresponding address. The processor executes the instruction. To the right, a memory map table is shown:

Adresse	Donnée
07	14
12	C9
13	34
12	85
11	3B
10	52
0F	BC
0E	F7
0D	86
0C	D0
0B	1F
0A	19
09	B1
08	85
07	E6
06	C9
05	12
04	07
03	19
02	CB
01	F7
00	

Valeurs déjà dans le processeur Proc

1 – le processeur présente l'adresse sur le bus  
2 – La mémoire fournit la donnée présente dans la case correspondante  
3 – Le processeur exécute l'instruction lue

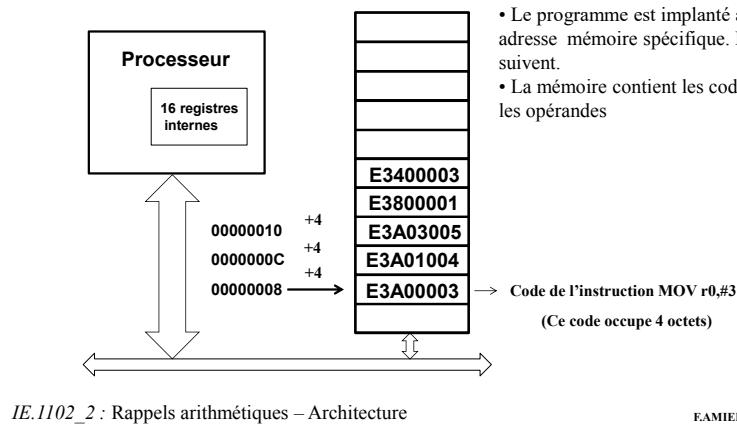
IE.1102\_2 : Rappels arithmétiques – Architecture FAMIL 31/59



## Définitions

- Le programme en langage machine s'appelle *assembleur*
- On passe des codes machines au code binaire par une opération appelée *assemblage*
- Les mots qui désignent une opération sont appelés *mémoriques*
- Les données concernées par les opérations sont appelées *opérandes*

## Codage en mémoire



## Processeurs

- Les processeurs lisent la mémoire et exécutent les instructions qui s'y trouvent
- Tous les processeurs disposent de registres internes
  - PC : Program Counter
  - Data register : Contenant les opérandes
  - Address register : @ des opérandes
- Parfois les registres d'adresses et de données sont unifiés
- On définit un processeur par :
  - Son jeu de registres accessibles par le programmeur
  - Ses instructions
  - Ses modes d'adressage

**Modèle de prog ARM**

Registres généraux

Chaque registre fait 32 bits

L'espace d'adressage est en octet

- Stack register (pointeur de pile)
- Link register (registre de lien)
- Program counter
- Current Program Status register

IE.1102\_2 : Rappels arithmétiques – Architecture      FAMILI      38/59

**Définitions**

- Jeu d'instruction :
  - Ensemble des instructions (mnémoniques) que peut exécuter le processeur
- Mode d'adressage :
  - Techniques pour désigner l'opérande

IE.1102\_2 : Rappels arithmétiques – Architecture      FAMILI      39/59

**Jeu d'instruction**

BLT <	
BEG =	BNE ≠
BGT >	BPL
BLE ≤	
BGE ≥	
LSR	RSR
DIV	NEC

- Les instructions sont de différents types :
  - Arithmétiques et logiques
    - Telles que ADD, SUB, AND... *MUL, AND, ORR, EOR, ISL, ADC, SBC*
  - Chargement / rangement
    - Pour chercher un opérande en mémoire pour le placer dans le processeur ou pour remettre un opérande du processeur vers la mémoire
  - Branchements
    - Pour se déplacer dans le code
  - Instructions systèmes
    - ces instructions permettent de contrôler le fonctionnement du processeur. (reset, lode veille...)

IE.1102\_2 : Rappels arithmétiques – Architecture      FAMILI      40/59

**isep**  
École d'ingénieurs du numérique

## Jeu d'instruction

Instructions arithmétiques / logiques	Chargement / rangement
Instructions de branchements (conditions)	
Instructions systèmes	

IE.1102\_2 : Rappels arithmétiques – Architecture      FAMILI      41/59

ADD ADC NOT (SUB SBC) AND (ORR) (EOR) ADC SBC (CMP) NEG MUL DIV  
 LDR STR  
 BRA BEQ BNE BLE BLT BGT BGE BMI BME BHI BHE  
 STOP RESET WAIT RTE

**isep**  
École d'ingénieurs du numérique

## Fréquence d'instruction

Rang	Instruction 80x86	Moyenne sur entiers (% du total exécuté)
1	chargement	22%
2	branchement conditionnel	20%
3	comparaison	16%
4	rangement	12%
5	addition	8%
6	et	6%
7	soustraction	5%
8	transfert registre-registre	4%
9	appel	1%
10	retour	1%
	Total	96%

**FIGURE 2.11 Les dix premières instructions du 80x86.** Ces pourcentages sont la moyenne des même cinq programmes SPECint92 que dans la figure 2.7 de la page 84 Hennessy - Patterson

IE.1102\_2 : Rappels arithmétiques – Architecture      FAMILI      42/59

**isep** École d'ingénierie de l'informatique

## Jeu d'instruction PIC

Mnemonic, Operands	Description	Cycles	12-Bit Opcode	Status Affected	Notes
ADDWF f,d	Add W and f	1	0001 11df ffff	C,D,C,Z	1,2,4
ANDWF f,d	AND W with f	1	0001 01df ffff	Z	2,4
CLRF f	Clear f	1	0000 011f ffff	Z	4
CLRW	- Clear W	1	0000 000f ffff	Z	
COMF f, d	Complement f	1	0010 01df ffff	Z	
DECf f, d	Decrement f	1	0000 11df ffff	None	2,4
DECF f, d	Decrement f, Skip if 0	1 (2)	0000 11df ffff	None	2,4
DIOSFSZ f, d	-	1 (2)	0011 01df ffff	C	2,4
INCF f, d	Increment f	1	0010 10df ffff	None	2,4
INCFSZ f, d	Increment f, Skip if 0	1 (2)	0011 11df ffff	None	2,4
IORWF f, d	Inclusive OR W with f	1	0000 00df ffff	Z	2,4
MOVE f, d	Move f	1	0010 00df ffff	Z	2,4
MOVWF f	Move W to f	1	0000 001f ffff	None	1,4
NOOP	- No Operation	1	0000 000f ffff	None	
RLF f, d	Rotate left f through Carry	1	0011 01df ffff	C	2,4
RHF f, d	Rotate right f through Carry	1	0011 00df ffff	C	2,4
SUBWF f, d	Subtract W from f	1	0000 10df ffff	C,D,C,Z	2,4
SWAPF f, d	Swap f	1	0011 10df ffff	None	2,4
XORWF f, d	Exclusive OR W with f	1	0001 10df ffff	Z	2,4
<b>BIT-ORIENTED FILE REGISTER OPERATIONS</b>					
BCF f, b	Bit Clear f	1	0100 kb0f ffff	None	2,4
BSF f, b	Bit Set f	1	0101 kb0f ffff	None	2,4
BTFSZ f, b	Bit Test f, Skip if Clear	1 (2)	0110 kb0f ffff	None	
BTFFS f, b	Bit Test f, Skip if Set	1 (2)	0111 kb0f ffff	None	
<b>LITERAL AND CONTROL OPERATIONS</b>					
ANDLW k	AND Literal with W	1	1110 kkkk kkkk	Z	
CALL k	Call subroutine	2	1001 kkkk kkkk	None	1
CLRWDAT k	Clear Watchdog Timer	1	0000 0000 0100	TO,PD	
GOTO k	Unconditional branch	2	101k kkkk kkkk	None	
IOFW	Indirect OR Literal with W	1	1111 kkkk kkkk	Z	
MOVLW k	Move Literal to W	1	1100 kkkk kkkk	None	
OPTION	- Load OPTION register	1	0000 0000 0010	None	
RETlw	- Return from subroutine in W	2	1000 0000 0000	None	
SLEEP	- Go into standby mode	1	0000 0000 0011	TO,PD	
TRIS f	Load TRIS register	1	0000 0000 0fff	None	2
XORLW k	Exclusive OR Literal to W	1	1111 kkkk kkkk	Z	3

IE.1102\_2 : Rappels arithmétiques – Architecture

EAMIEL 43/59

**isep** École d'ingénierie de l'informatique

## Mode d'adressage

- Les instructions de chargement / rangement permettent de lire / d'écrire une donnée en mémoire
- L'adresse de la donnée lue / écrite peut être spécifiée de multiples façons
  - Accès simple à une adresse fixe
  - Accès à une donnée placée dans un tableau à l'indice X
  - Incrémantion / décrémentation automatique de l'index
  - ....

IE.1102\_2 : Rappels arithmétiques – Architecture

EAMIEL 44/59

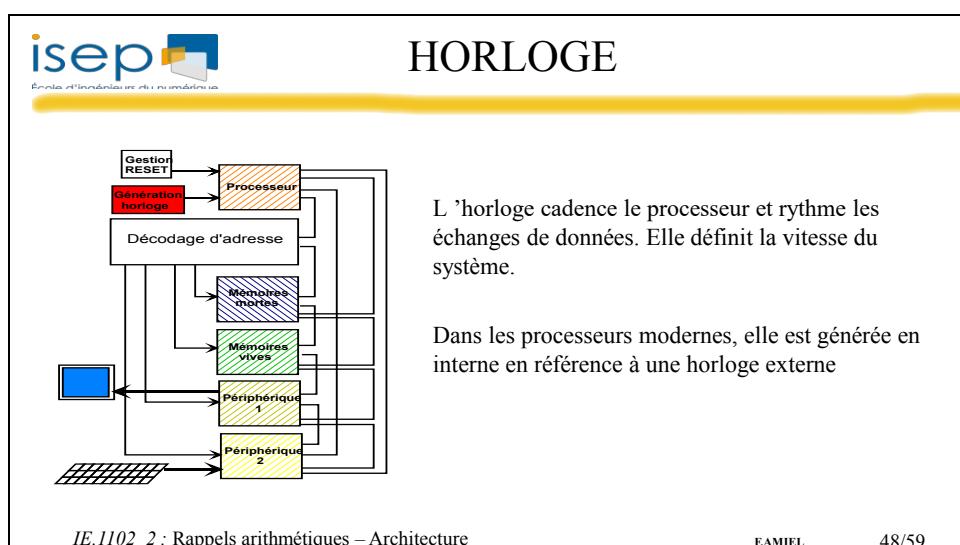
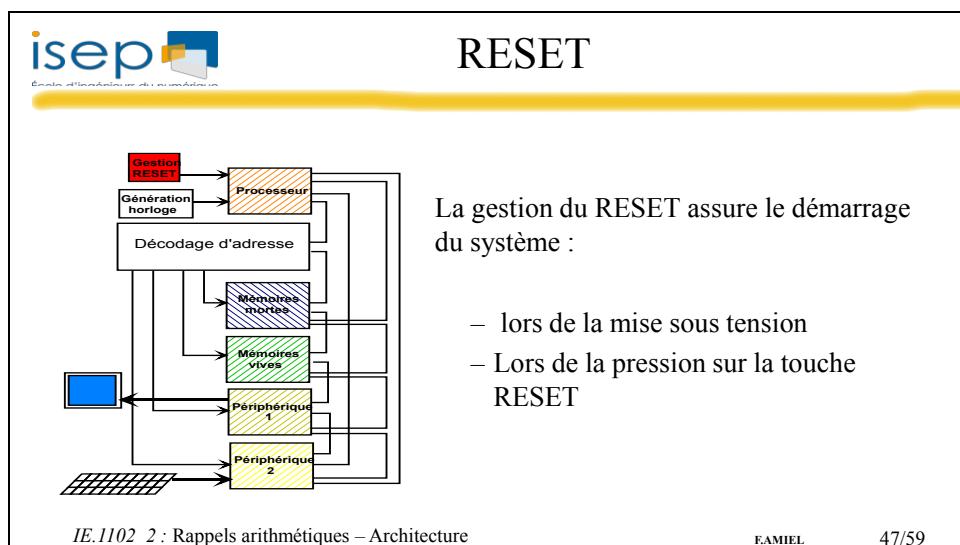
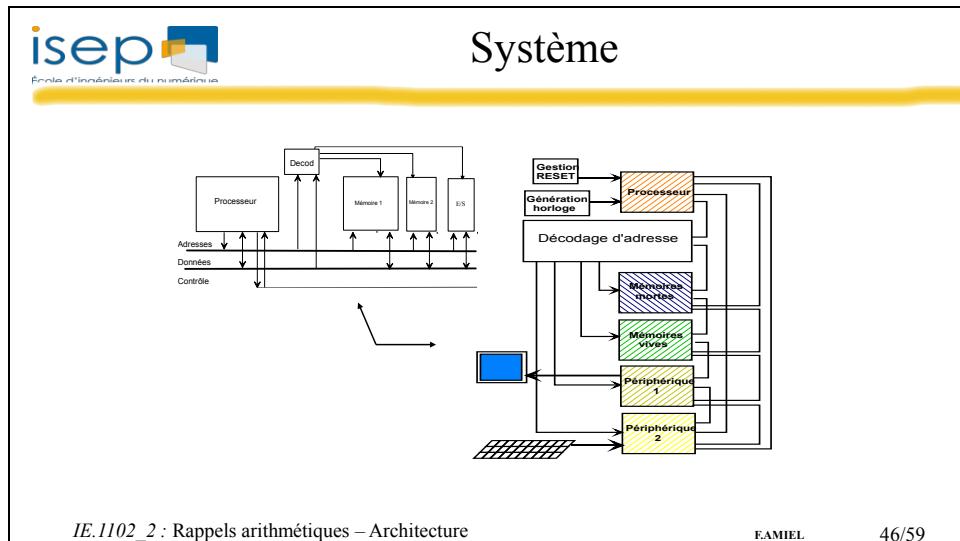
**isep** École d'ingénierie de l'informatique

## Auto test1/1

- Quel est le rôle du registre PC ?
- Que contiennent les registres de données ?
- Le microprocesseur 68k peut adresser 16 777 216 cases mémoire. Combien de bits ont ses registres d'adresse au minimum ?
- A quoi sert le signal horloge
- Quelle est la différence entre les registres du processeur et la mémoire ?
- Comment peut-on faire une multiplication si l'instructions de multiplication n'existe pas dans le processeur ?

IE.1102\_2 : Rappels arithmétiques – Architecture

EAMIEL 45/59



**Processeur**

The diagram illustrates the architecture of a computer system centered around a Processor. The Processor is connected to several components via buses:

- Address Bus:** Connects to a **Décodage d'adresse** (Address Decoding) block, which then connects to **Mémoires mortes** (Dead Memory), **Mémoires vives** (Live Memory), and **Périphérique 1** and **Périphérique 2**.
- Data Bus:** Connects to the Processor and **Périphérique 1** and **Périphérique 2**.
- Control Bus:** Connects to the Processor, **Gestion RESET** (RESET Management), **Génération horloge** (Clock Generation), and **Décodage d'adresse**.

*IE.1102\_2 : Rappels arithmétiques – Architecture*      FAMILIEN      49/59

**Bus de données**

The diagram shows the data bus connections in red, indicating the path of data transfer between the Processor and the external components.

*IE.1102\_2 : Rappels arithmétiques – Architecture*      FAMILIEN      50/59

**Bus d'adresse**

The diagram shows the address bus connections in red, indicating the path of address signals from the Processor to the memory and peripherals.

- Le bus d'adresse porte le numéro de composant et le numéro de case mémoire concernée dans le composant.
- Sur le 68k, ce bus fait 23 bits de large
  - $2^{23} = 8 \cdot 10^6$  cases soit  $16 \cdot 10^6$  octets adressables au maximum
- Processeur ARM 7 : 32 bits
  - $2^{32} = 4 \cdot 10^9$  cases soit 4Gos adressables

*IE.1102\_2 : Rappels arithmétiques – Architecture*      FAMILIEN      51/59

**Décodage d'adresse**

The diagram illustrates the address decoding architecture. At the top, a 'Gestion RESET' (Reset Management) block and a 'Génération horloge' (Clock Generation) block feed into a 'Processeur' (Processor). Below the processor is a 'Décodage d'adresse' (Address Decoding) block. This block has three main output paths: one to 'Mémoires mortes' (Dead Memories), another to 'Mémoires vives' (Live Memories), and a third to two peripheral components labeled 'Périphérique 1' and 'Périphérique 2'. A blue square component at the bottom left also receives connections from the address decoding block.

- Le décodeur d'adresse active les composants concernés suivant le numéro de la case souhaité par le processeur.
- Cette fonction est assurée par un ensemble de portes logiques faisant partie de la GLU d'une carte (General Logic Unit)

IE.1102\_2 : Rappels arithmétiques – Architecture

EAMIET

52/59

**Mémoire morte**

This diagram shows the architecture for 'Mémoire morte'. It features a similar setup to the previous diagram, with a 'Gestion RESET', 'Génération horloge', and 'Processeur' at the top. The 'Décodage d'adresse' block is present, but its outputs are directed solely to the 'Mémoires mortes' (Dead Memories) and the two peripheral components ('Périphérique 1' and 'Périphérique 2'). The blue square component is also connected to the address decoding block.

- La mémoire morte contient les programmes de démarrage du système, et les programmes liés aux périphériques spécifiques de la carte (BIOS)
- Dans un système embarqué, la mémoire morte contient les programmes applicatifs

IE.1102\_2 : Rappels arithmétiques – Architecture

EAMIET

53/59

**Mémoire vive**

The diagram for 'Mémoire vive' follows the same basic structure. The 'Gestion RESET', 'Génération horloge', and 'Processeur' are at the top, with the 'Décodage d'adresse' block in the middle. However, the outputs of the address decoder are now directed to the 'Mémoires vives' (Live Memories) and the two peripheral components ('Périphérique 1' and 'Périphérique 2'). The blue square component remains connected to the address decoding block.

- La mémoire vive contient les variables utilisées par les programmes.
- Dans les stations de travail par exemple, les programmes utilisateurs sont placés dans la mémoire vive.

IE.1102\_2 : Rappels arithmétiques – Architecture

EAMIET

54/59

**isep**  
École d'ingénieurs et de management

## Périphériques

```

    graph TD
        GestionRESET[Gestion RESET] --> Processeur[Processeur]
        GH[Génération horloge] --> Processeur
        DA[Décodage d'adresse] --> Processeur
        DA --> MM[Mémoires mortes]
        DA --> MV[Mémoires vives]
        DA --> P1[Pérophérique 1]
        DA --> P2[Pérophérique 2]
        Processeur --> MM
        Processeur --> MV
        Processeur --> P1
        Processeur --> P2
        MM --> P1
        MV --> P2
        P1 --> Ecran[Écran]
        P2 --> Clavier[Clavier]
    
```

IE.1102\_2 : Rappels arithmétiques – Architecture

EAMIEL 55/59

- Les périphériques font la liaison avec le monde extérieur. Ils assurent la communication de différentes façons.
- Dans l'exemple présenté, l'un des périphériques gère un écran, l'autre un clavier.

**isep**  
École d'ingénieurs et de management

## Exemple

The diagram shows the same architecture as the first slide, but with red lines highlighting the control bus. A red line connects the processor's address bus to the memory modules (Mémoires mortes and Mémoires vives). Another red line connects the processor's control bus to both memory modules and the two peripheral devices (Pérophérique 1 and Pérophérique 2).

IE.1102\_2 : Rappels arithmétiques – Architecture

EAMIEL 56/59

- Le processeur génère une adresse qui concerne la mémoire morte (par exemple, il lit un code opération).
- Le décodeur active la mémoire morte, qui fournit la donnée (code op)
- C'est le bus de contrôle, qui n'est pas représenté ici qui porte les signaux de lecture ou d'écriture

**isep**  
École d'ingénieurs et de management

## Processeurs

**Les performances des processeurs sont très variées**

**Automatismes simples**

Processeurs 4 ou 8 bits

**IHM – Usage général - Embarqué**

Processeurs 32 bits (ARM)

IE.1102\_2 : Rappels arithmétiques – Architecture

EAMIEL 57/59

**isep**  
École d'ingénieurs du numérique

## Processeurs

**Processeurs hautes performances**



Processeurs 32 – 64 bits  
Multi-cœurs (tegra3 :  
4 cœurs ARM Cortex A9)

**Processeurs spécialisés**



DSP – GPU - SoC

IE.1102\_2 : Rappels arithmétiques – Architecture      FAMIEL      58/59

**isep**  
École d'ingénieurs du numérique

## Auto test

- Quelle est la fonction du décodage d'adresse sur une carte ?
- Pourquoi trouve-t-on de la mémoire morte dans un système ?
- Pourquoi trouve-t-on de la mémoire vive dans un système ?
- Quelle est la fonction d'un périphérique ?

IE.1102\_2 : Rappels arithmétiques – Architecture      FAMIEL      59/59



**isep**  
École d'ingénieurs du numérique

## Objectifs

- Comprendre le fonctionnement logiciel d'un processeur
- Apprendre le modèle de programmation de l'ARM
- Connaitre les principaux codes opération d'un processeur
- Avoir un aperçu des modes d'adressage

Frédéric Amiel – Architecture des ordinateurs – ARM 2/52

Ce module décrit l'organisation interne du processeur ARM7, ainsi que les instructions principales de ce processeur et leur mode de fonctionnement.

**isep** École d'ingénieurs du numérique

## ARM Ltd

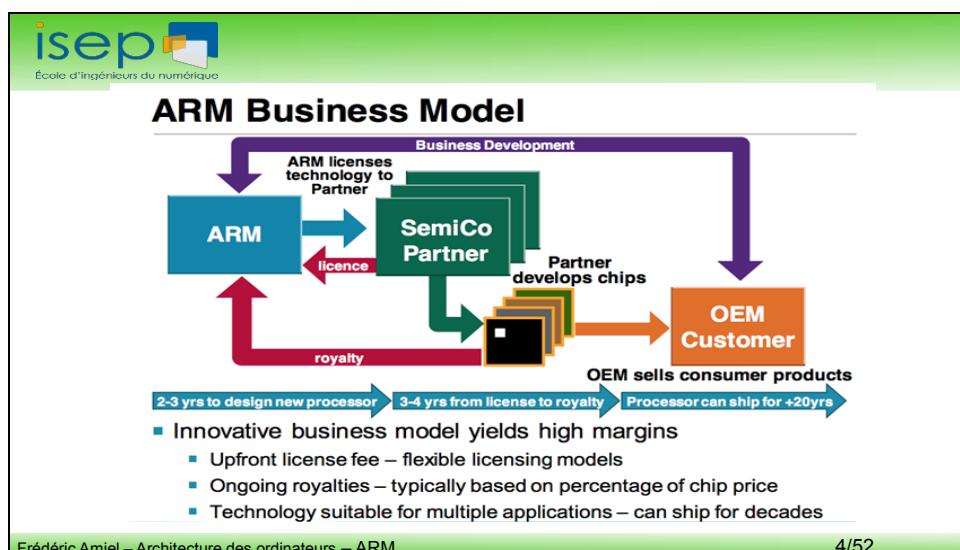
- British company, founded in November 1990 in Cambridge (UK)
  - Spin out of Acorn Computers
  - Listed since April 1998 (ARM NASDAQ: ARMHY)
- ARM designs a range of 32-bits RISC processor cores
  - ARM stands for « Advanced RISC Machines »
- ARM sells licenses to its semiconductor industry partners. These licensees fabricate and sell RISC core based integrated circuits to their customers.
  - ARM does not fabricate silicon itself!!!
- Also develop technologies to assist with the design-in of the ARM Architecture
  - Software tools, development boards, debug hardware, bus architectures (AMBA), peripherals, etc...



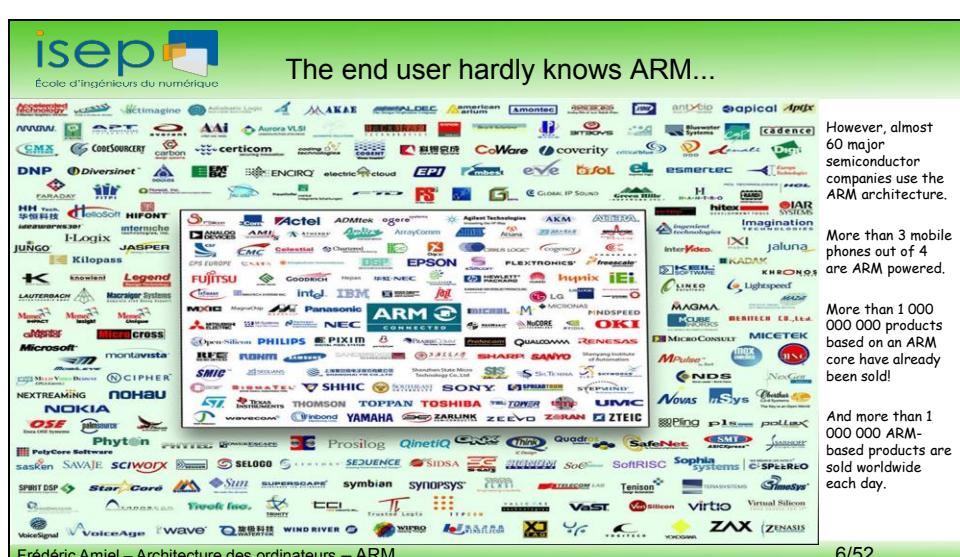
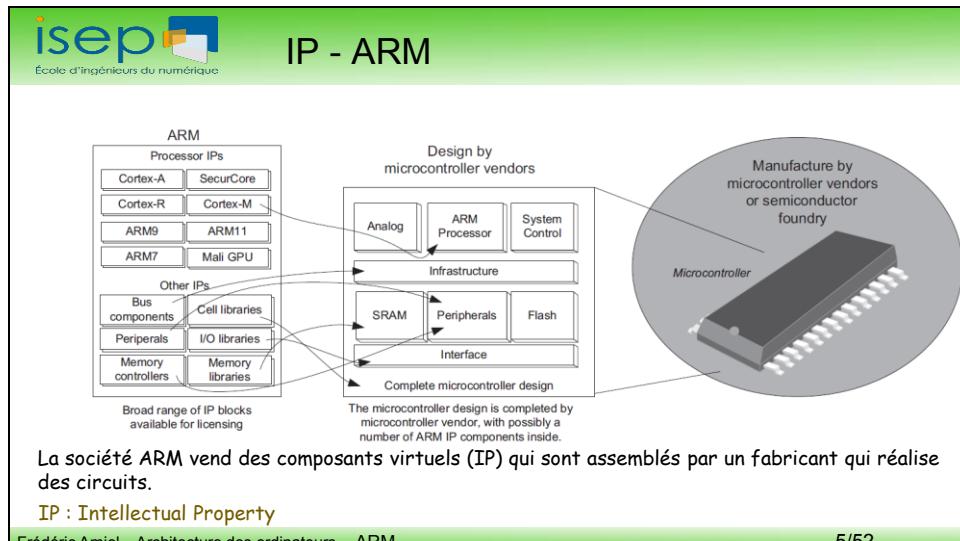
**A dvanced  
R ISC  
M achines**

Frédéric Amiel – Architecture des ordinateurs – ARM

La société ARM est anglaise. Cette société ne vend pas de silicium mais des licences, ainsi qu'un ensemble d'outils logiciels permettant d'utiliser les processeurs. Les clients intègrent le processeur ARM dans leur produit, de façon à construire des « System On Chip » dédiés à une utilisation précise.



Un **OEM** : Original Equipment Manufacturer, désigne une entreprise chargée de la fabrication de pièces détachées pour une autre entité (entreprise, assembleur, etc.).



Beaucoup de sociétés utilisent le processeur ARM au cœur de leurs produits. En particulier, on trouve un ARM dans plus de 70% des téléphones portables. (En association avec un DSP).

**ISEP**  
École d'ingénieurs du numérique

## ARM Pro / Cons

**LOW POWER**  
Up to 11000 MIPS/Watt (0.25um/0.9V)  
Suited for embedded applications

**BUT**

Fees for ISA	No fees for ISA
Fees for microarchitecture	Fees for microarchitecture
Warranty & indemnification (limited)	Warranty & indemnification (limited)
Classic commercial IP license	RISC-V commercial IP license
	RISC-V open source IP license

32 / 64 bits architecture  
SIMD extension (NEON)

Concerning high end cpu, ARM neoverse is ranked 57 (2021)

<https://www.cpubenchmark.net/>

**RISC-V**

Frédéric Amiel – Architecture des ordinateurs – ARM      7/52

Ce processeur est architecturé comme un composant 32 bits, mais peut travailler en mode 16 bits. Il permet de répondre à une grande variété de besoin. Il est particulièrement réputé pour sa faible consommation (Million d'instruction par Seconde par Watt : MIPS/Watts).

**ISEP**  
École d'ingénieurs du numérique

## Targeted Markets

**Arm's long reach**  
Global available market for Arm chip technology, \$bn

Market	2017 (\$bn)	2026 Forecast (\$bn)
Internet of Things	~25	~45
Mobile-computing processors	~20	~32
Vehicles	~12	~30
Energy-efficient servers	~18	~22
Network infrastructure	~15	~20
Other	~30	~38

Source: Sanford C. Bernstein

Arm market share, 2017, %

Market	Arm Market Share (%)
Internet of Things	40
Mobile-computing processors	90
Vehicles	20
Energy-efficient servers	<1
Network infrastructure	20
Other	40

The Economist      8/52

Ce processeur se décline en plusieurs gammes et se trouve

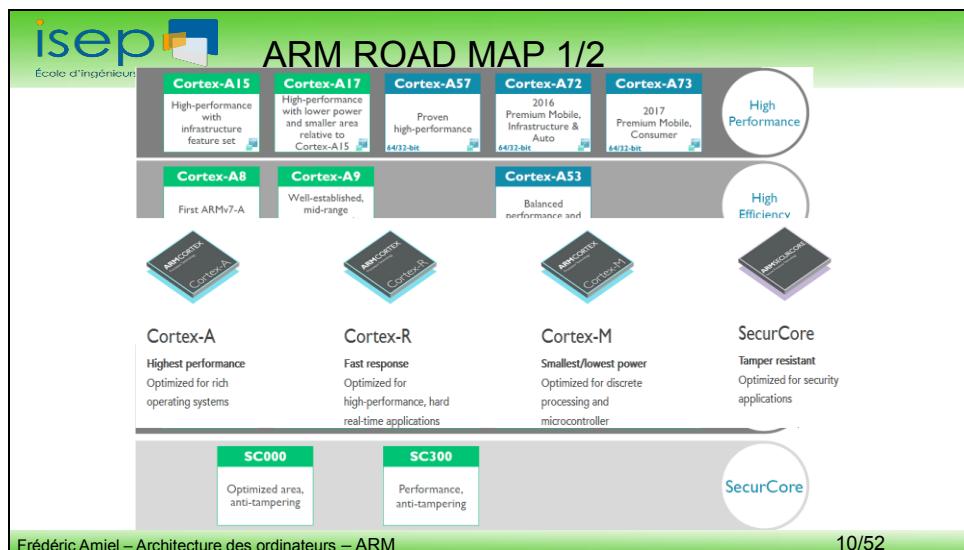
**isep** École d'ingénieurs du numérique

## ARM History

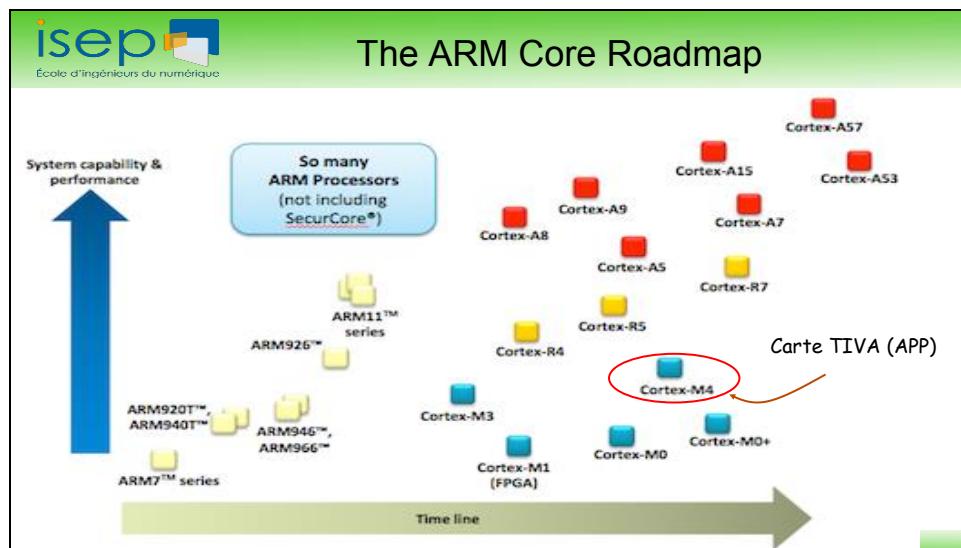
- 1983-1985 : Acorn fonde les bases d'un microprocesseur RISC pour des applications commerciales
- 1985 : Le premier prototype ARM1
- 1990 : Création de Advance Risc Machine Ltd
- 1993 : Annonce du processeur ARM7
- 1995 : premier composant ARM7 (Texas Instrument Villeneuve loubey)

Frédéric Amiel – Architecture des ordinateurs – ARM      9/52

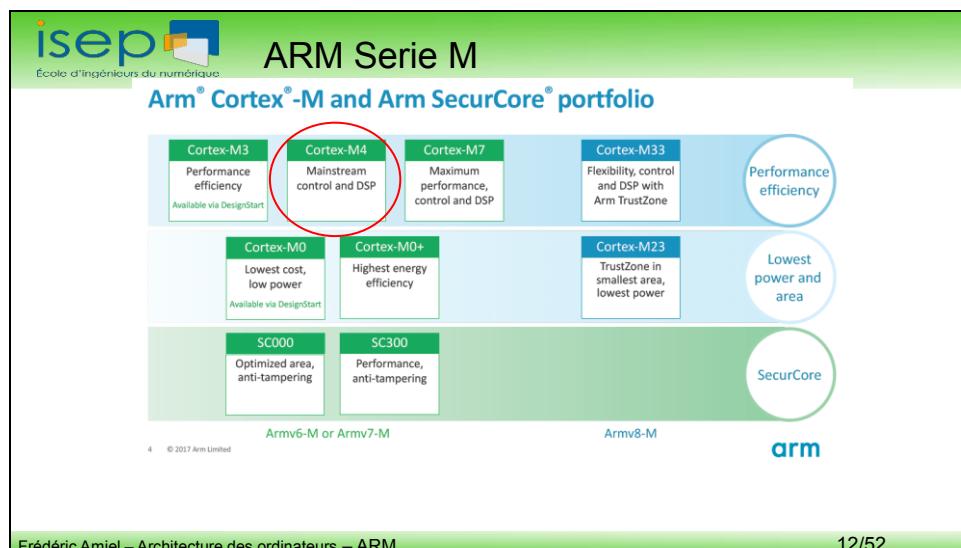
Le premier composant ARM à été réalisé par la société TEXAS INSTRUMENT en France en 1995.



Cortex A pour les systèmes performants (raspberry PI....)  
Cortex R pour les processeurs réactif (automatique)  
Cortex M pour les applications IoT



L'arm7 est conçu pour fonctionner à des fréquences de l'ordre de 50 MHz. Les processeurs de gamme plus élevée fonctionnent à des fréquences supérieures. Le processeur ARM9 intègre une unité de calcul flottant. Le processeur ARM11 est super scalaire. Les processeurs Cortex sont spécialisés pour exécuter au mieux certains systèmes d'exploitation.

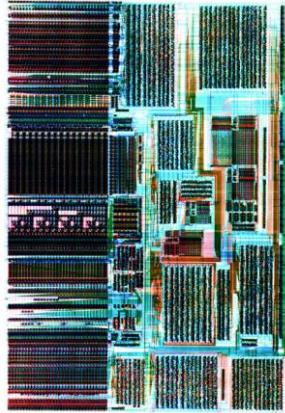


M0 – M55

**ISEP**  
École d'ingénieurs du numérique

## ARM 7 Généralités

- Processeur RISC 32 bits
  - Architecture load/store
- 75000 Transistors
- 51 instructions
- 9 modes d'adressage
- Pipeline 3 étages
- 7 modes d'exécution
- ~66 MHz
- CPI ~ 1,9
- 60 Mips (??)
- Von Neumann architecture



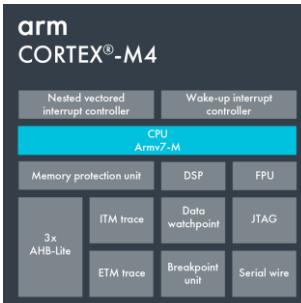
Frédéric Amiel – Architecture des ordinateurs – ARM

Le processeur ARM7 un processeur d'usage général. Son pipeline à 3 étages lui assure un CPI : Clock Per Instruction de 1,9. (1,9 Cycles horloge en moyenne pour une instruction).

**ISEP**  
École d'ingénieurs du numérique

## Cortex M4

ISA Support	<u>Thumb/Thumb-2</u>
Pipeline	3-stage + branch speculation
DSP Extension	Single cycle 16/32-bit MAC Single cycle dual 16-bit MAC 8/16-bit SIMD arithmetic Hardware Divide (2-12 Cycles)
Floating-Point Unit	Optional single precision floating point unit IEEE 754 compliant
Memory Protection	Optional 8 region MPU with sub regions and background region
Bit Manipulation	Integrated Bit Field Processing Instructions & Bus Level Bit Banding
Interrupts	Non-maskable Interrupt (NMI) + 1 to 240 physical interrupts
Interrupt Priority Levels	8 to 256 priority levels
80-MHz operation; 100 DMIPS performance	



Frédéric Amiel – Architecture des ordinateurs – ARM      14/52

**isep**  
École d'ingénieurs du numérique

## ARM7 TDMI - S

- ARM7 : Advance Risc Machine cœur ARM v4T
  - T : Extension Thumb du jeu d'instruction (16 bits)
  - D : Extension Debug
  - M : Multiplieur étendu :  $32 \times 32 \Rightarrow 64$  bits
  - I : Extension interne « Embedded ICE » (debug)
  - S : Cœur synthétisable

Frédéric Amiel – Architecture des ordinateurs – ARM      15/52

Lors des travaux pratiques, on utilise l'ARM7 TDMI.

**isep**  
École d'ingénieurs du numérique

## Taille de données

- Le processeur ARM est une architecture RISC 32-bits.
- Sur ce processeur
  - Byte indique une donnée codée sur 8 bits
  - Halfword indique une donnée 16 bits (2 bytes)
  - Word indique une donnée 32 bits (4 bytes)
- La plupart des processeurs ARM implémentent 2 jeux d'instructions
  - Le jeu 32-bit « ARM Instruction Set » (celui que nous allons étudier)
  - Le jeu 16-bit « Thumb II Instruction Set »
- Le cœur Jazelle dispose d'instructions Java (Java bytecodes).

A dvanced  
R ISC  
M achines

Frédéric Amiel – Architecture des ordinateurs – ARM      16/52

Les mots font 32 bits. Il est possible toutefois d'effectuer des accès mémoires par octets.

Lors de ce cours, on ne s'intéresse qu'au fonctionnement sur 32 bits.

**isep**  
École d'ingénieurs du numérique

## Processor Modes

- Le processeur ARM dispose de 7 modes d'opération :
- **User** : Mode sans priviléges : celui de la plupart des tâches
- **FIQ** : on y entre lorsque une interruption « Fast Interrupt » est activée (IT prioritaire).
- **IRQ** : On y entre lorsqu'une interruption est activée.
- **Supervisor** : Mode lors du RESET ou lorsqu'une instruction d'exception logiciel (instruction SWI) est activée.
- **Abort** : utilisé pour gérer les violations d'accès mémoire.
- **Undef** : Permet de gérer les instructions non définies.
- **System** : Mode privilégié

Frédéric Amiel – Architecture des ordinateurs – ARM      17/52

Le processeur dispose de 7 modes d'exécution. Pour la suite, on sera essentiellement en mode USER.

**isep**  
École d'ingénieurs du numérique

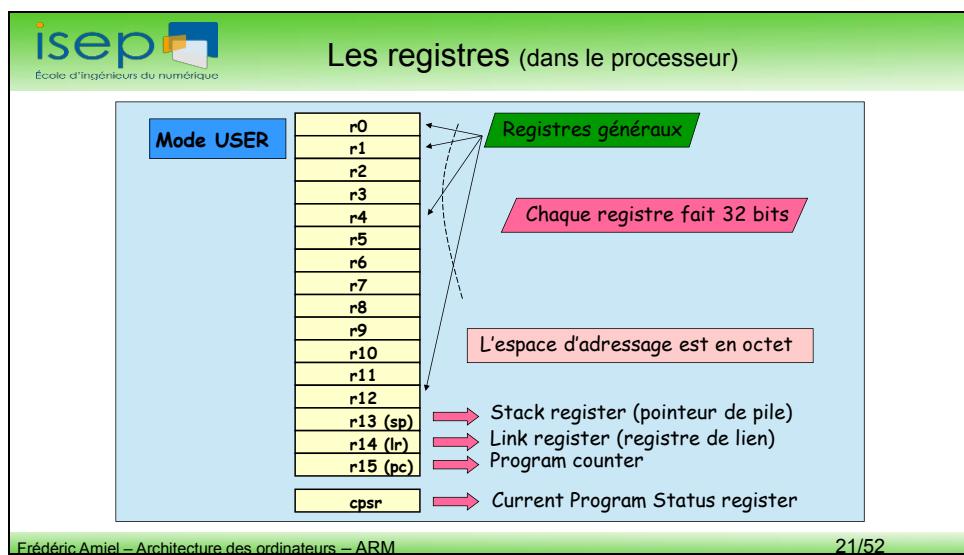
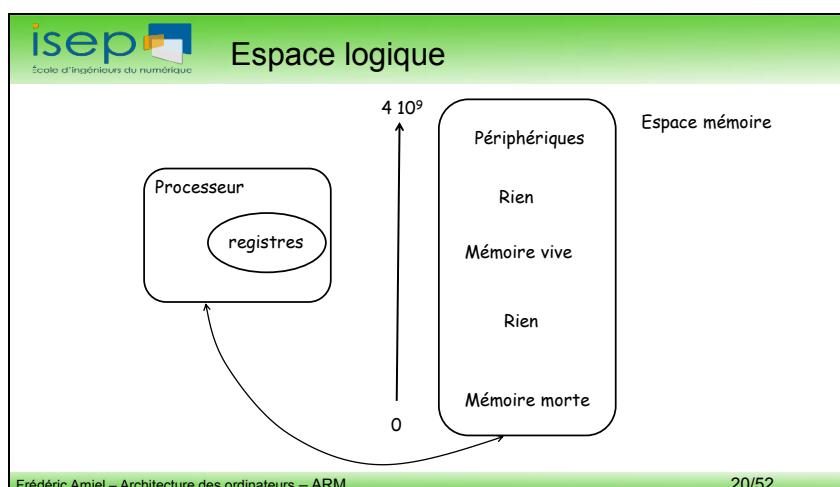
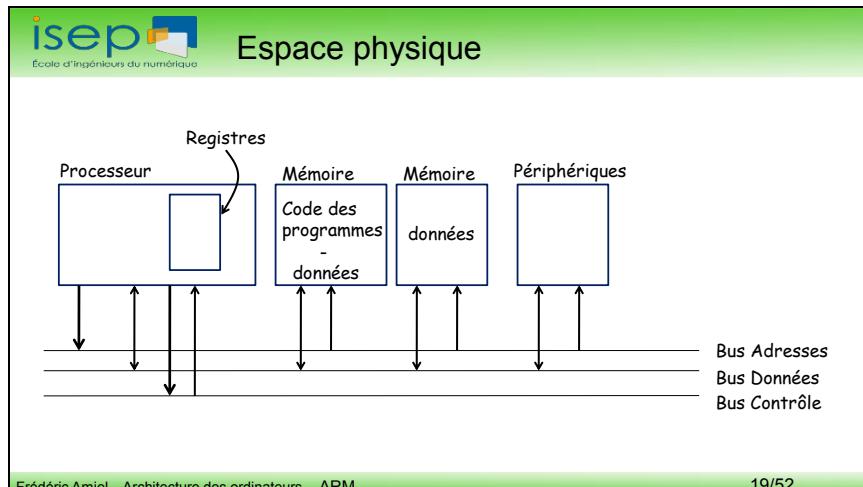
## Code ARM - Thumb

- Lorsque le processeur exécute du code ARM
  - Les instructions sont codées sur 32 bits
  - Les instructions doivent être alignées sur des mots
  - La valeur du PC est stockée dans les bits [32:2]. Les bits [1:0] ne sont pas définis ⇒ les instructions sont alignées suivant des mots (tous les 4 octets).
- Lorsque le processeur exécute du code Thumb II
  - Les instructions sont codées sur 16 bits ou 32 bits
  - Les instructions sont alignées par demi-mots
  - Le bit 0 du PC n'est pas défini

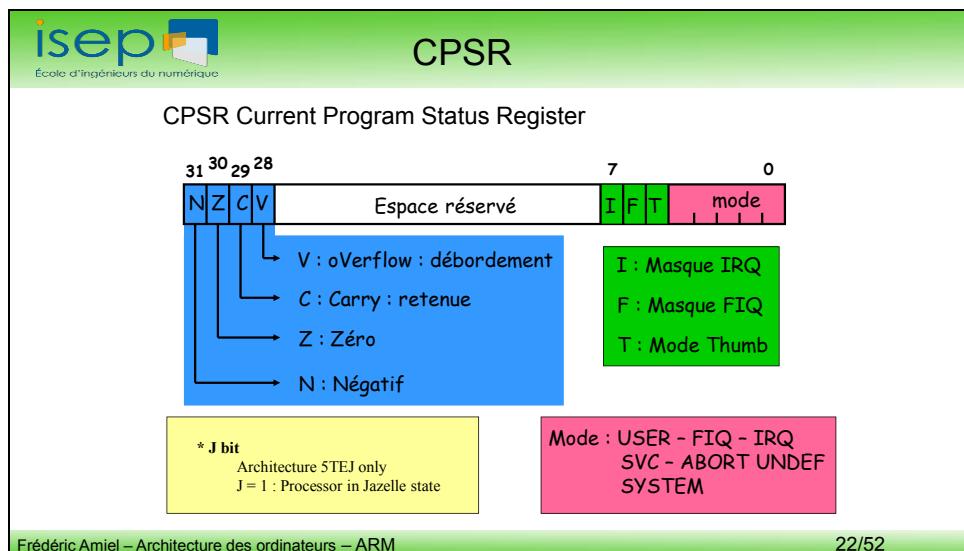
**Nous utiliserons essentiellement le mode ARM**

Frédéric Amiel – Architecture des ordinateurs – ARM      18/52

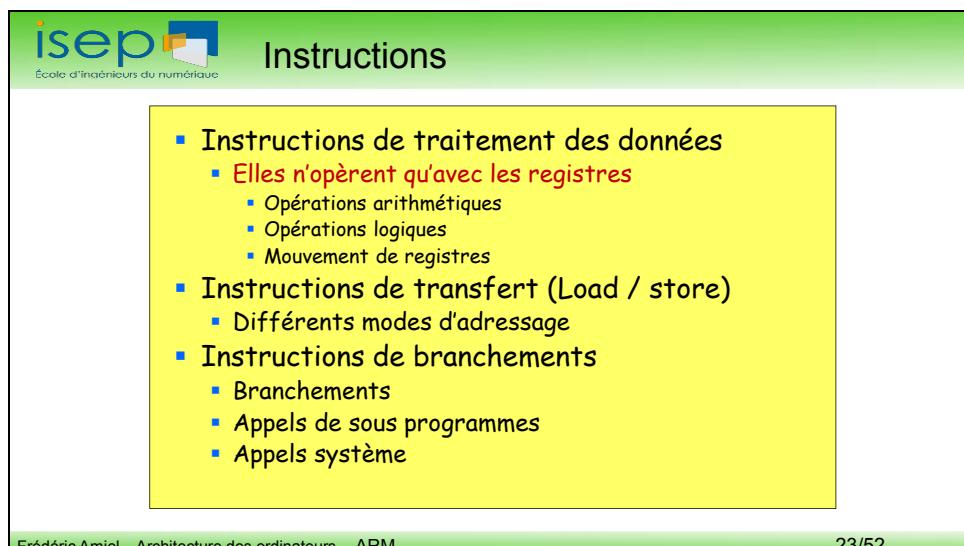
Le mode 16 bits permet de produire un code plus petit avec des performances plus faibles. Ce mode THUMB permet au processeur ARM de se placer aussi sur le marché des processeurs 16 bits.



Le processeur dispose de 16 registres généraux utilisables comme registres de données (permettant de délivrer ou de recevoir des opérandes) et d'adresse (permettant de définir l'adresse : le numéro de case d'une donnée en mémoire).



CPSR : Current program Status Resigter. Ce registre contient l'état « courant » du processeur.



Dans les processeurs RISC, les instructions ne s'appliquent qu'aux opérandes disponibles dans les registres du processeur.

**isep** École d'ingénieurs du numérique

## Instructions

Le format général des instructions est le suivant

Inst {cond}{S}    Rd,Rn,N    ↔    ADD r1,r2,r3

Mnémonique de l'instruction

Il y a au maximum 3 opérandes

Rd : registre de destination

Rn : registre source

N : source

Les instructions modifient ou non les bits d'états : {S}

Toutes les instructions sont conditionnelles : Elles ne s'exécutent que si les bits de l'UAL sont positionnés comme désirés (prédictat)

On peut décaler le dernier opérande de chaque instruction (Il faut ajouter un suffixe)

Frédéric Amiel – Architecture des ordinateurs – ARM      24/52

Les instructions de traitements des données acceptent 2 opérandes en entrée, et génèrent un résultat. Les opérandes sources sont :

- un registre
- un registre ou un nombre.

L'opérande destination est toujours un registre.

**isep** École d'ingénieurs du numérique

## Instructions de traitement

ADD r1,r2,r3    r1 ← r3 + r2  
ADD r1,r1,r1    r1 ← r1 + r1

Mode registre direct  
Les opérandes sont les **registres**

Seul l'opérande 2 peut être : un registre OU une valeur

ADD r1,r2,#2    r1 ← 2 + r2

8 bits max    Adressage immédiat : #  
L'opérande est la valeur spécifiée

Une seule donnée immédiate par opération

Par défaut les instructions n'activent pas les bits d'état de l'UAL  
Le suffixe S permet de les activer

ADDS r1,r2,r3    r1 ← r3 + r2 / N Z C V activés

Frédéric Amiel – Architecture des ordinateurs – ARM      25/52

Lorsqu'un des opérande est un nombre, il faut placer un « # » devant. Dans ce cas, ce nombre est limité à 8 bits par qu'il est placé dans le mot binaire qui code l'instruction complète. Ce mot fait 32 bits et contient l'ensemble des informations qui représentent l'instruction.

**isep** École d'ingénieurs du numérique **Instruction MOV**

<b>MOV</b> : place une valeur 32 bits dans un registre $MOV\ r4,r3\quad r4 \leftarrow r3$ <b>MOVN</b> : Place le complément à 1 (not) d'une valeur dans un registre $MOVN\ r4,r3\quad \sim r4 \leftarrow r3$	
<b>Exemples :</b> $r5 = 3$ $r7 = 8$	$r5 =$ $MOV\ r7,r5\quad r7 =$ $r5 =$ $MOVN\ r7,r5\quad r7 =$ $r7 = 0 / Z \leftarrow 1$ <span style="background-color: #00008B; color: white; padding: 2px;">Instructions de traitement</span>

Frédéric Amiel – Architecture des ordinateurs – ARM 26/52

L'instruction MOV effectue une copie de la source vers la destination. Cette instruction ne nécessite que deux opérandes.

**isep** École d'ingénieurs du numérique **Instructions arithmétiques**

Inst {cond}{S} Rd,Rn,N		
<u><a href="#">ADC</a></u>	Addition de 2 valeurs 32 bits avec la retenue	$rd \leftarrow rn + N + C$
<u><a href="#">ADD</a></u>	Addition de 2 valeurs 32 bits	$rd \leftarrow rn + N$
<u><a href="#">RSB</a></u>	Soustraction inverse de 2 valeurs	$rd \leftarrow N - rn$
<u><a href="#">RSC</a></u>	Soustraction inverse de 2 valeurs avec retenue	$rd \leftarrow N - rn - !C$
<u><a href="#">SBC</a></u>	Soustraction 2 valeurs avec retenue	$rd \leftarrow rn - N - !C$
<u><a href="#">SUB</a></u>	Soustraction de 2 valeurs 32 bits	$rd \leftarrow rn - N$

<b>Exemples :</b> $r0 = 0$ $r1 = 2$ $r2 = 5$	$SUB\ r0,r1,r2\quad r0 =$ $RSB\ r0,r1,#0\quad r0 = -r1 = 0xffffffff$ $ADD\ r0,r1,#4\quad r0 = r1 + 4 = 6$
---	---

Frédéric Amiel – Architecture des ordinateurs – ARM 27/52

Les instructions arithmétiques utilisent 2 opérandes en entrée, et génèrent un résultat.

**isep** École d'ingénieurs du numérique

## Instructions logiques

Inst {cond}{S} Rd,Rn,N		
<u>AND</u>	ET logique (mise de bits à 0)	$rd \leftarrow rn \& N$
<u>ORR</u>	OU logique (mise de bits = 1)	$rd \leftarrow rn   N$
<u>EOR</u>	OU exclusif (inversion de bits)	$rd \leftarrow N \oplus rn$
<u>BIC</u>	Mise à 0 de bits sélectionnés	$rd \leftarrow rn \& \sim N$

Exemples :

r0 = 0  
r1 = 0x02040608  
r2 = 0x10305070

→

ORR r0,r1,r2 r0 =
AND r0,r1,0xff r0 =
EOR r0,r1,#0xf r0 =
BIC r0,r2,#0xf0 r0 =

Frédéric Amiel – Architecture des ordinateurs – ARM

28/52

Les instructions logiques fondamentales sont le AND, le OR et le NOT.

**isep** École d'ingénieurs du numérique

## Instructions de comparaison

Inst {cond}{S} Rn,N		
Pas de registre destination		Les drapeaux sont levés
<u>CMN</u>	Comparaison avec l'inverse	$rn + N$
<u>CMP</u>	Comparaison	$rn - N$
<u>TEQ</u>	Test de l'égalité de 2 valeurs	$rn \oplus N$
<u>TST</u>	Test d'une valeur	$rn \& N$

Exemples : Permet un branchement conditionnel par la suite  
si  $\gg \geq < \leq == \dots$

→

CMP r0,#2 r0 $\leftrightarrow$ 2
TEQ r0,#4 r0 == 4 ?
TST r9,r3

Frédéric Amiel – Architecture des ordinateurs – ARM

29/52

Les instructions de comparaison positionnent les drapeaux sans modifier de registre. Elles sont utilisées pour implémenter les tests, les instructions conditionnelles, les branchements conditionnels.

**isep**  
École d'ingénieurs du numérique

## Multiplication simple précision

<code>mla Rd,Rm,rs,rn mul rd,rm,rs</code>		
<b>MLA</b>	Multiplication et accumulation	$rd \leftarrow (rm * rs) + rn$
<b>MUL</b>	Multiplication	$rd \leftarrow rm * rs$

MLA permet d'implémenter des filtres numériques

**Exemples :**

$r0 = 0$	<b>MUL</b>	$r0,r1,r2$	$r0 =$
$r1 = 2$			
$r2 = 3$			
$r3 = 4$	<b>MLA</b>	$r0,r1,r2,r3$	$r0 =$

Frédéric Amiel – Architecture des ordinateurs – ARM      30/52

La multiplication de deux nombres 32 bits génère un résultat sur 64 bits qui est tronqué dans cette instruction. La multiplication / accumulation est une opération fondamentale en traitement numérique du signal.

**isep**  
École d'ingénieurs du numérique

## Multiplication double précision

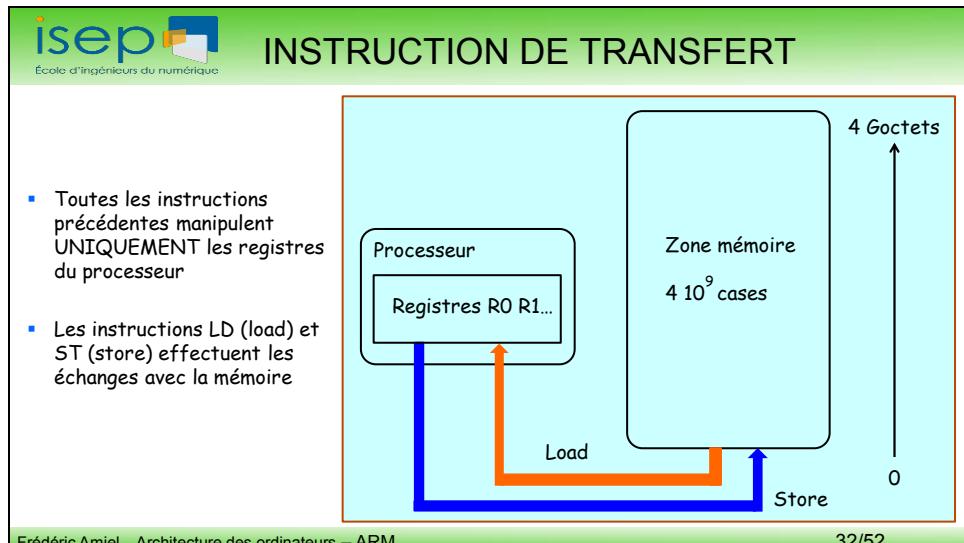
<code>Inst {cond}{S} RdLo,RdHi,Rm,Rs</code>		
<b>SMLAL</b>	Multiplication signée et accumulation long	$[RdHi,RdLo] = [RdHi,RdLo] + Rm * Rs$
<b>SMULL</b>	Multiplication signée long	$[RdHi,RdLo] = Rm * Rs$
<b>UMLAL</b>	Multiplication non signée et accumulation long	$[RdHi,RdLo] = [RdHi,RdLo] + Rm * Rs$
<b>UMULL</b>	Multiplication non signée long	$[RdHi,RdLo] = Rm * Rs$

**Exemple :**

$r0 = 0$	<b>UMULL</b>	$r0,r1,r2,r3$
$r1 = 0$		
$r2 = 0xf0000002$		
$r3 = 0x00000002$		
$r0 = 0xe0000004 \rightarrow RdLo$		
$r1 = 0x00000001 \rightarrow RdHi$		

Frédéric Amiel – Architecture des ordinateurs – ARM      31/52

Ces instructions permettent de réaliser des multiplications avec des résultats sur 64 bits.



**isep**  
École d'ingénieurs du numérique

## Load – Store

Instructions de transfert

LDR   STR {cond}{B} Rd,addressing		
LDR	Chargement d'un mot de la mémoire dans un registre	$rd \leftarrow \text{mem32}[address]$
STR	Sauvegarde d'un registre dans la mémoire	$rd \rightarrow \text{mem32}[address]$
LDRB	Chargement d'un octet de la mémoire dans un registre	$rd \leftarrow \text{mem8}[address]$
STRB	Sauvegarde d'un octet d'un registre dans la mémoire	$rd \rightarrow \text{mem32}[address]$
LDRH	Chargement d'un demi mot de la mémoire dans un registre	$rd \leftarrow \text{mem16}[address]$
STRH	Sauvegarde d'un demi mot d'un registre dans la mémoire	$rd \rightarrow \text{mem16}[address]$
LDRSH	Chargement d'un octet signé de la mémoire dans un registre	$rd \leftarrow \text{mem16}[address]$
STRSH	Sauvegarde d'un octet signé d'un registre dans la mémoire	$rd \rightarrow \text{mem16}[address]$

Frédéric Amiel – Architecture des ordinateurs – ARM

33/52

L'instruction LOAD charge les registres du processeur. L'instruction STORE les sauvegarde en mémoire.

**isep**  
École d'ingénieurs du numérique

## Décalages

- Toutes les instructions peuvent recevoir un opérande décalé :
 

Logical Shift Left / right	Arithmetic Shift LSL / LSR (Logical Shift Left / right)
Rotate Left / right	

LSL		Les bits sont décalés à gauche du nombre de position spécifié
LSR 0 →		Les bits sont décalés à droite du nombre de position spécifié
ASR		Les bits sont décalés à gauche du nombre de position spécifié
ROR		Les bits sont décalés à gauche du nombre de position spécifié
ROL		Les bits sont décalés à gauche du nombre de position spécifié

Frédéric Amiel – Architecture des ordinateurs – ARM      34/52

Le décalage logique à droite (LSR) permet de diviser un nombre binaire par une puissance de 2. Le décalage logique à gauche (LSL) permet de multiplier un nombre par une puissance de 2.

Le décalage arithmétique à droite (ASR) conserve le bit de signe. Cette instruction permet de diviser un nombre codé en complément à 2 par une puissance de 2. Ces instructions s'ajoutent à une instruction de traitement et concernent un des opérandes d'entrée.

**isep**  
École d'ingénieurs du numérique

## Décalages exemples

Les instructions de décalage sont en suffixe.  
C'est l'opérande 2 qui subit le décalage.

MOV r7,r5,LSL #2  r7 ← r5*4
ADD r0,r1,r1,LSL #1  r0 ← r1*3 ( r1 + 2*r1 )
Quels est l'intérêt des décalages à droite / à gauche ? Quels est l'intérêt du mode arithmétique / logique ?

Frédéric Amiel – Architecture des ordinateurs – ARM      35/52

Les instructions de traitement de données acceptent un opérande qui peut subir un décalage. Lorsqu'on veut appliquer un décalage uniquement à un opérande, il faut utiliser l'instruction MOV et utiliser le décalage d'opérande.

**isep** École d'ingénieurs du numérique Mode Thumb2

Le mode THUMB2 permet de simplifier le code.

**ADD R3,#4**  
Dans ce cas R3 est opérande source et opérande destination :  
 $R3 \leftarrow R3+4$

Les opérateurs de décalage deviennent alors des instructions à part entière  
**LSR R4,#2**     $R4 \leftarrow R4$  décalé à droite de 2 bits

Frédéric Amiel – Architecture des ordinateurs – ARM      36/52

**isep** École d'ingénieurs du numérique Modes d'adressage

- Les instructions arithmétiques et logiques ne fonctionnent qu'entre les registres
- Les instructions Load et Store permettent de lire et d'écrire des valeurs de la mémoire

PC → Mémoire (ADD, SUB, LDR, 87012012) → Instructions de transfert ← Mémoire (87012012)

Lecture instruction  
Lecture instruction  
Lecture instruction  
Lecture donnée

Frédéric Amiel – Architecture des ordinateurs – ARM      37/52

Les instructions LDR et STR s'exécutent obligatoirement en plusieurs cycles bus.

**isep**  
École d'ingénieurs du numérique

## Mode d'adressage

- Le mode d'adressage est la façon de définir une adresse mémoire
- Sur le processeur ARM, les modes d'adressage ne concernent que les instructions LDR / STR
- Les modes d'adressages permettent d'implémenter les accès aux variables, à des tableaux...

Instructions de transfert

Frédéric Amiel – Architecture des ordinateurs – ARM      38/52

Les modes d'adressage décrivent la localisation des opérandes. Ces modes d'adressage ne concernent que les instructions de chargement / rangement et les branchements.

**isep**  
École d'ingénieurs du numérique

## Adressage indirect

Le seul mode d'adressage du processeur ARM est l'adressage INDIRECT qui se décline de plusieurs manières

<code>LDR r0,[r1]</code> Le contenu des 4 cases mémoires de première adresse définie dans r1 est placée dans r0 (32 bits)
<code>STR r0,[r1]</code> Le contenu de r0 est placé dans les 4 cases mémoire à partir de l'adresse définie par r1.

Instructions de transfert

Frédéric Amiel – Architecture des ordinateurs – ARM      39/52

Dans le mode d'adressage indirect par registre, l'opérande est dans la case mémoire dont d'adresse est définie par un registre (qui sert de pointeur mémoire).

**isep**  
École d'ingénieurs du numérique

## Adressage indirect avec index

Préindexé avec écriture	mem[base+offset]	ldr r0,[r1,#4]!
Préindexé	mem[base+offset]	ldr r0,[r1,#4]
Postindexé	mem[base]	ldr r0,[r1], #4

Le « ! » indique que l'adresse est actualisée dans le registre d'adresse :

**Exemple :**

```

r0 = 0
r1 = 0x00009000
mem[0x9000] = 0x01010101
mem[0x9004] = 0x02020202
  
```

→

```

ldr r0,[r1,#4]!
r0 = 0x02020202
r1 = 0x00009004
  
```

**Instructions de transfert**

Frédéric Amiel – Architecture des ordinateurs – ARM      40/52

Différentes options permettent de définir l'adresse de l'opérande. Ces techniques permettent de parcourir des tableaux, de définir une adresse de base d'une zone de différents paramètres etc.

**isep**  
École d'ingénieurs du numérique

## Adressage indexé exemples

**Exemple :**

```

r0 = 0
r1 = 0x00009000
mem[0x9000] = 0x01010101
mem[0x9004] = 0x02020202
  
```

**Pré indexé**

```

ldr r0,[r1,#4]
  
```

→

```

r0 = 0x02020202
r1 = 0x00009000
  
```

**Post indexé**

```

ldr r0,[r1],#4
  
```

→

```

r0 = 0x01010101
r1 = 0x00009004
  
```

**Instructions de transfert**

Frédéric Amiel – Architecture des ordinateurs – ARM      41/52

Ce mode avec incrément automatique permet de parcourir un tableau de données placées les unes à la suite des autres.

**isep**  
École d'ingénieurs du numérique

## Adressage indexé

Il est possible d'utiliser un autre registre comme offset

Préindexé avec offset immédiat	[Rn, # ± offset_12]
Préindexé avec offset registre	[Rn, ± Rm]
Préindexé avec registre décalé	[Rn, ± Rm, shift #shift_imm]
Préindexé avec offset et réécriture	[Rn, # ± offset_12] !
Préindexé avec registre et réécriture	[Rn, ± Rm] !
Préindexé avec registre décalé réécriture	[Rn, ± Rm, shift #shift_imm] !
Postindexé avec offset immédiat	[Rn], # ± offset_12
Post indexé avec registre	[Rn], ± Rm
Post indexé avec registre décalé	[Rn], ± Rm, shift #shift_imm

Instructions de transfert

Frédéric Amiel – Architecture des ordinateurs – ARM      42/52

Ces différentes options répondent à différents besoins applicatifs de parcours de tableau, ou d'adressage de variable.

**isep**  
École d'ingénieurs du numérique

## Adressage indexé exemples

Instruction	r0 =	r1 +=
ldr r0,[r1, #4]!	mem[r1+4]	0x4
ldr r0,[r1,r2]!	mem[r1+r2]	r2
ldr r0,[r1,#0x4]	mem[r1+4]	0
ldr [r0,-r2, LSR #04]	mem [r1-(r2 lsr 4)]	0
ldr r0,[r1],#04	Mem[r1]	4
ldr r0,[r1],r2	Mem[r1]	r2

Instructions de transfert

Frédéric Amiel – Architecture des ordinateurs – ARM      43/52

**isep**  
École d'ingénieurs du numérique

## Load / Store

Les instructions Load / Store permettent d'accéder aux données et de gérer des tableaux

Soit  $i = 2$  stocké dans R3  
Dans cet exemple le tableau débute en 0x2300

Tab[i] = 5;	
Ldr r10,=0x2300	R10 0x2300
Mov r0,#5	R0 5
Str r0,[R10,r3]	R3 2

Code assembleur

Mémoire

0x230C Tab[3]  
0x2308 Tab[2]  
0x2304 Tab[1]  
0x2300 Tab[0]

R10 → 0x2300

Frédéric Amiel – Architecture des ordinateurs – ARM 44/52

Les modes d'adressage permettent d'accéder à des données dans des tableaux.

**isep**  
École d'ingénieurs du numérique

## LDM / STM

Les instructions LDM et STM permettent de charger / ranger plusieurs registres.  
Ces instructions se déclinent avec un suffixe :

IA	Increment After
IB	Increment before
DA	Decrement after
DB	Decrement before

Ldmia r0!,{r1-r3} : Les registres r1 à r3 prennent les valeurs du contenu des cases mémoires pointées par r0. R0 est ensuite incrémenté de 12.

Instructions de transfert

Frédéric Amiel – Architecture des ordinateurs – ARM 45/52

Ces instructions sont utilisées pour sauvegarder et restituer des registres avant et après un appel de sous programme par exemple.

**isep** École d'ingénieurs du numérique

## Chargement d'une constante

LDR : Load constant

Il s'agit d'une pseudo instruction permettant de placer une valeur dans un registre.

LDR r0,=0xff	↔	MOV r0,#0xff
LDR r0,=12345678	↔	LDR r0,[pc,#offset_12]

L'instruction référence l'adresse de la constante à placer dans r0, relativement à l'adresse de l'instruction

Frédéric Amiel – Architecture des ordinateurs – ARM      46/52

Le chargement d'une constante est particulièrement utilisé lors des programmes écrits en assembleur.

On utilise l'adressage indirect, mais une subtilité d'écriture permet de simplifier la syntaxe.

**isep** École d'ingénieurs du numérique

## Instructions de branchement

B {L}{<cond>} <adresse cible>

B : branchement simple  
BL : branchement avec lien (permet un retour automatique)  
Dans ce cas, le registre r15 (adresse de l'instruction suivante) est copié avant le branchement dans le registre r14 (r14 = lr = registre de lien).

**Exemple :**

B      ETIQ ; Branchement systématique	...	...
ETIQ    ..		
BL      SOUS_PROG ; Appel de sous programme	...	...

Frédéric Amiel – Architecture des ordinateurs – ARM      47/52

L'instruction BL correspond à l'appel de sous programme. Elle est donc très différente de l'instruction B qui correspond à un branchement simple.

Le fonctionnement détaillé de l'instruction BL est expliqué dans la suite de ce cours.

**isep**  
École d'ingénieurs du numérique

## Instructions de branchement

B {L}{<cond>} <adresse cible>

Le champ <condition> permet d'effectuer des branchements conditionnels

**Exemple :**

```
for (i=10; i != 0; i--);
```

Exécution d'une boucle 10 fois

```

MOV r0,#10      ; Initialisation compteur de boucle
BCL ..
SUBS r0,#1      ; Décrémentation compteur
BNE BCL          ; Si compteur <> 0 alors branchement
...              ; Sinon on continue

```

Frédéric Amiel – Architecture des ordinateurs – ARM      48/52

Comme toutes les instructions, les branchements peuvent être conditionnels. Ces branchements conditionnels permettent d'implémenter les tests.

**isep**  
École d'ingénieurs du numérique

## Instructions de branchement

B {L}{<cond>} <adresse cible>

Les plus courantes :

**Conditions :**

EQ : Equal / Equal 0	→ Z set
NE : Non equal	→ Z clear
HI : Plus grand (non signé)	→ C set and Z clear
LS : lower or same (non signé)	→ C clear or Z set
GE : Greater than or equal	→ N equal V
LE : Less than or equal	→ N non equal V

Frédéric Amiel – Architecture des ordinateurs – ARM      49/52

On général, on utilise une arithmétique signée (nombres codés en complément à 2).

**isep** École d'ingénieurs du numérique

## Instructions de branchement

B {L}{<cond>} <adresse cible>
-------------------------------

**Conditions :**

CS/HS : Carry set / Higher or Same	→ C set
CC/LO : Carry clear / Lower or same	→ C clear
MI : Minus / Negativ	→ N set
PL : Plus / Positiv or zero	→ N clear
VS : Overflow	→ V set
VC : No overflow	→ V clear
AL : Alway	→ Any
NV : Never	→ none

Les BMI – BPL etc permettent d'effectuer des tests entre des nombres codés en binaire pur.

**isep** École d'ingénieurs du numérique

## Jeu conditionnel

- Toutes les instructions sont conditionnelles
- Par défaut {AL} (Always) : ADD ↔ ADDAL

CMP	r0,#4	
ADDEQ	R1,R1,#1	Si R0 == 4 alors R1 ← R1+1

```

    cmp   r0,#'a'
    cmpne r0,#'e'
    cmpne r0,#'i'
    cmpne r0,#'o'      Si R0 == 'a' ou 'e' ou 'i' ou 'o' ou 'u' ou 'y'
    cmpne r0,#'u'      alors R1 ← R1+1
    cmpne r0,#'y'

    Addeq r1,r1,#1 → Si Z est positionné alors on incrémenté R1
    ↙ EQ = Test d'égalité : Z = 1
  
```

**isep**   
École d'ingénieurs du numérique

## Autotest

Quelle est la signification des instructions suivantes :

```
Sub    r0,r1,#5
add   r0,r1,r3
Ands  r4,r4,#0x20
addeq r5,r5,r6
ldr   r0,[r1],#4
str   r0,[r1]
```

Quelles instructions permettent de réaliser les opérations suivantes :

- Additionner le contenu de R0 avec lui-même, placer le résultat dans R1
- Soustraire R4 de R6, placer le résultat dans R3
- Placer le contenu de R3 dans R4
- Placer le chiffre 47 dans R10
- Placer le contenu de R0 dans la case mémoire dont l'adresse est dans R10

Frédéric Amiel – Architecture des ordinateurs – ARM      52/52



- Connaître les conditions (instructions conditionnelles / branchement).
- Comprendre le codage des instructions
- Comprendre les appels de fonctions
- Connaître les durées d'exécution

**isep**

## Codage des instructions

Les conditions 1/4

**Toutes les instructions de l'ARM se codent sur UN SEUL mot mémoire**

Selon la condition, l'instruction est exécutée ou non.

31	28 27	cond	0

Opcode	Mnemonic extension	Interpretation	Status flag for execution
0000	EQ	Equal / Equals zero	Z set
0001	NE	Not equal	Z clear
0010	CS/HS	Carry set/ unsigned $\geq$	C set
0011	CC/LO	Carry clear / unsigned $<$	C clear

HS : higher or same. LO : Lower.  
Le test ne prend en compte que la retenue : binaire pur (unsigned).

Frédéric Amiel – Architecture des ordinateurs – ARM

3/40

On construit le code des instruction par champs de bits. Chaque groupe de bit a une signification interprétée par le décodeur d'instruction du processeur ARM. Dans les transparents qui suivent, on énumère ces différents champs de bits, de façon à L'exécution de toutes les instructions est conditionnée à l'état des drapeaux.

**isep**

## Codage des instructions

Les conditions 2/4

**Toutes les instructions de l'ARM se codent sur UN SEUL mot mémoire**

Selon la condition, l'instruction est exécutée ou non.

31	28 27	cond	0

Opcode	Mnemonic extension	Interpretation	Status flag for execution
0100	MI	Minus / négativ	N set
0101	PL	Plus / $\geq$	N clear
0110	VS	Overflow	V set
0111	VC	No overflow	V clear

Frédéric Amiel – Architecture des ordinateurs – ARM

4/40

MI et PL référencent des tests concernant le codage binaire (non signé)

**isep**

## Codage des instructions

### Les conditions 3/4

Toutes les instructions de l'ARM se codent sur UN SEUL mot mémoire

Selon la condition, l'instruction est exécutée ou non.

31	28 27	cond	0

Opcode	Mnemonic extension	Interpretation	Status flag for execution
1000	HI	Unsigned >	C set Z clear
1001	LS	Unsigned lower	C clear or Z clear
1010	GE	Signed $\geq$	N equal V
1011	LT	Signed $<$	N not equal V

HI : Higher / LS : Less or Same / GE : Greater or equal / LT : Less than

**isep**

## Codage des instructions

### Les conditions 4/4

Toutes les instructions de l'ARM se codent sur UN SEUL mot mémoire

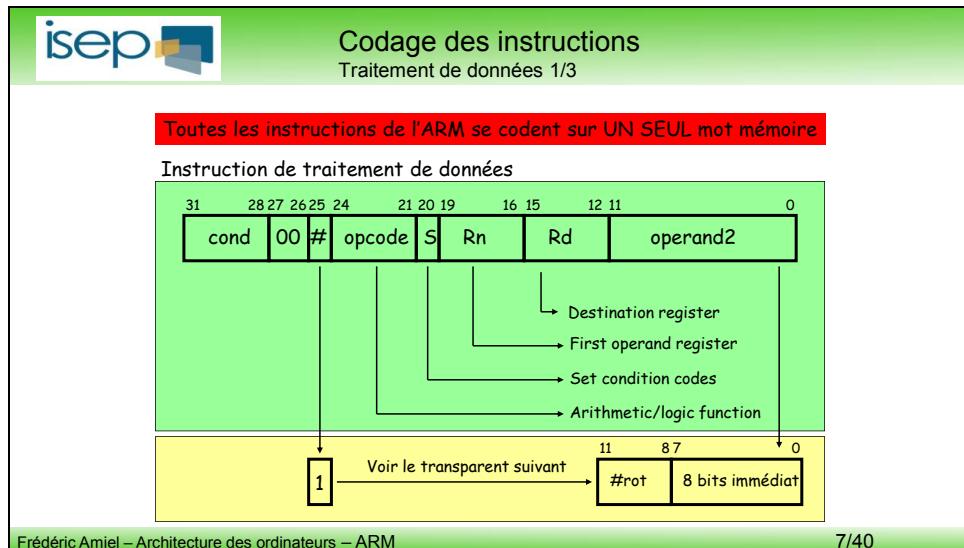
Selon la condition, l'instruction est exécutée ou non.

31	28 27	cond	0

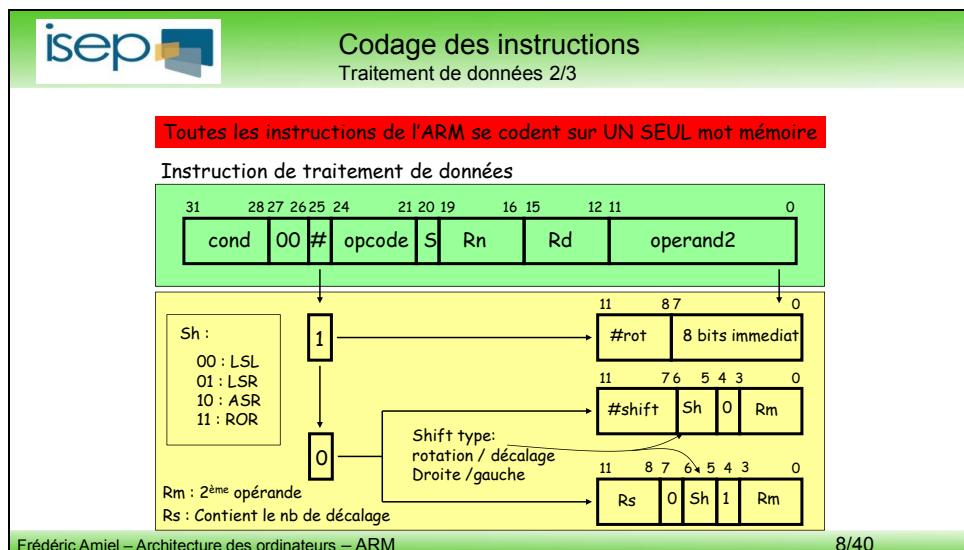
Opcode	Mnemonic extension	Interpretation	Status flag for execution
1100	GT	Signed $>$	Z clear and N = V
1101	LE	Signed $\leq$	Z set or N not equal V
1110	AL	Always	any
1111	NV	Never	none

GT : Greater / LE : Less or Equal

Par défaut, la condition « AL » (toujours) est utilisée.



Le champ opcode indique la nature de l'opération (transparents 10-11). Rd correspond à un des registres d'entrée. Rd, au registre qui va recevoir le résultat. L'opérande 2 peut être un registre ou une valeur immédiate. Dans ce cas, cette valeur est limitée à 8 bits, mais accepte une rotation.



Lorsque l'opérande 2 est un registre, on peut appliquer à ce registre des décalages ou rotation. Le nombre de bit décalés peut être précisé dans l'instruction ou être contenu dans un registre.

**isep**

### Codage des instructions

Traitement de données 3/3

Exemple : Codage de : ADD R0,R1,R2 (R0 = R1+R2)

31	28 27	26 25	24	21 20 19	16 15	12 11	0
cond	00 #	opcode	S	Rn	Rd	operand2	

Cond Always : xxxx  
 Opcode ADD : xxxx  
 Rn R1  
 Rd R0  
 Operand2 décalage de 0 appliquée à R2 :

11	7 6	5 4 3	0
#shift	Sh	0	Rm

31	00	0
0x		

Frédéric Amiel – Architecture des ordinateurs – ARM 9/40

Le 00 des bits 26 et 27 indiquent qu'on a affaire à une instruction de traitement de données.

**isep**

### Opcode 1/2

31	28 27	26 25	24	21 20 19	16 15	12 11	0
cond	00 #	opcode	S	Rn	Rd	operand2	

0000	AND	ET logique	Rd = rn and op
0001	EOR	Ou exclusif	Rd = rn eor op
0010	SUB	Soustraction	Rd = rn - op
0011	RSB	Soustraction inverse	Rd = op - rn
0100	ADD	Addition	Rd = rn + op
0101	ADC	Addition avec retenue	Rd = rn+op+c
0110	SBC	Soustraction avec retenue	Rd = rn-op+c-1
0111	RSC	Soustraction inverse avec R	Rd = op-rn+c-1

Frédéric Amiel – Architecture des ordinateurs – ARM 10/40

**isep**

### Opcode 2/2

31	28 27	26 25	24	21 20 19	16 15	12 11	0
cond	00 #	opcode	S	Rn	Rd	operand2	

1000	TST	Test	Cond : rn and op
1001	TEQ	Test égalité	Cond : rn eor op
1010	CMP	Comparaison	Cond : Rn-op
1011	CMN	Compare négatif	Cond : rn+op
1100	ORR	OU logique	Rd = rn OU op
1101	MOV	Mouvement de donnée	Rd = op
1110	BIC	Bit clear (mise de bit à 0)	Rd = rn and not op
1111	MVN	Move négatif	Rd = not op

Frédéric Amiel – Architecture des ordinateurs – ARM 11/40

**isep**

### Codage des instructions Instruction de branchement 1/2

Branchement avec lien (voir plus loin).

Cette configuration indique que c'est une instruction de branchement

L'adresse des instructions est modulo 4 (chaque instruction est codée sur 4 octets).  
Avec 24 bits de déplacement, les sauts de ce type sont limités à  $\pm 2^{23}$  adresses de différence soit  $\pm 2^{25}$  octets =  $\pm 32$  octets

Les bits 26 et 27 ne sont plus à 00.

**isep**

### Codage des instructions Instruction de branchement 2/2

Adresse Mémoire

202C	
2028	
2024	
2020	
201C	
2018	
2014	Suite du prog
2010	
200C	
2008	
2004	B *+ 12
2000	

L'offset de +12 se code +3  
(3 cases de 4 octets plus loin).

Ces cases sont sautées

L'offset permet des déplacements relatifs par rapport à l'endroit courant. Il est impossible d'indiquer un endroit situé trop « loin » de cette façon.

- Dans quel cas effectue-t-on des branchements ?
  - Goto
  - Boucles
  - Test ?
  - Appel de sous programmes
  - ...?
- Énumérer les raisons d'effectuer un branchement

Les instructions de branchements sont très fréquentes dans un programme.

- Les branchements conditionnels permettent d'implémenter les tests :

If (i == 4)	Cmp R0,#4
{	Bne adr1
code 1	... code 1
}	B suit
Else	Adr1
{	... code2
code 2	suit
}	

Sur le processeur ARM, il est possible d'implémenter les tests de différentes façon (exécution conditionnelles). Les instructions de branchement conditionnel sont les plus générales.

**isep**

## Instructions conditionnelles

- On peut aussi utiliser les instructions conditionnelles

```

if (i > 0)          teq    r0,#0
    i += 5;        addgt  r0,r0,#5
...
...
if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u' || c == 'y')
    vowel++;
    Teq    r0,#'a'
    Teqne r0,#'e'
    Teqne r0,#'i'
    Teqne r0,#'o'
    Teqne r0,#'u'
    Teqne r0,#'y'
    Addeq r1,r1,#1
  
```

Dans ce cas, le processeur parcourt l'ensemble des instructions même si le premier test indique l'égalité.

Les instructions conditionnelles sont à utiliser avec mesure

Frédéric Amiel – Architecture des ordinateurs – ARM 16/40

Lorsqu'une instruction à exécution conditionnelle n'est pas exécutée, le processeur la lit tout de même. Le temps d'exécution est d'au moins un cycle bus.

**isep**

## Codage instructions

31	28	27	26	25	24	23	22	21	20	19	16	15	12	11	8	7	5	4	3	0
Cond	0	0	I	Opocde			S	Rn	Rd	Operand 2										
Cond	0 0 0 0 0 0					A	S	Rd	Rn	Rs	1 0 0 1		Rm							
Cond	1 1 1 1					ignored by processor														
Cond	0	1	I	P	U	B	W	L	Rn	Rd	offset									
Cond	0 1 1					XXXXXXXXXXXXXXXXXXXXXX													1	XXXX
Cond	1 0 0					P	U	S	W	L	Rn	Register List								
Cond	1 0 1					L	offset													

Data Processing  
PSR Transfer  
Multiply  
Software interrupt  
Single Data Transfer  
Undefined  
Block Data Transfer  
Branch

Frédéric Amiel – Architecture des ordinateurs – ARM 17/40

**isep**

## Notation symbolique

- Les programmes assembleurs suivent les règles suivantes :

Un programme est constitué de lignes.

Chaque ligne contient :

- Une zone label permettant de repérer la ligne
- Une zone mnémonique / opérandes
- Une zone commentaires

On peut définir des symboles de façon à améliorer la lisibilité

Les programmes sont placés dans la mémoire. Le processeur vient lire les instructions et les exécute.

On écrit un programme assembleur dans un fichier texte en suivant une syntaxe.

**isep**

## Syntaxe assembleur

Les lignes assembleur ont le format suivant :

```
{<label>}      {<instruction>}      ; commentaire
{<symbol>}    <directive>        ; commentaire
```

<instruction> est une instruction de l'ARM  
 <label> est le nom d'un symbole qui référence l'adresse d'une instruction  
 <directive> est une directive d'assemblage  
 <symbol> est le nom d'un symbole utilisé par <directive>  
 De plus chaque programme est localisé dans une « zone » le mot clé AREA indique une nouvelle zone.

Les instructions s'écrivent sur une seule ligne.

Le premier caractère à gauche dans chaque ligne est un label et non une instruction.  
 Les instructions s'écrivent donc plus à « droite » (le premier caractère de la ligne peut alors être un espace ou une tabulation).

**isep** Syntaxe assembleur

{<label>} {<symbol>}	{<instruction>} {<directive>}	; commentaire ; commentaire
----------------------	-------------------------------	--------------------------------

Les différents champs sont séparés par des caractères <space> ou <TAB>.

Le champs <label> ou <symbol> peut être vide : les instructions ne peuvent jamais s'écrire sur la première colonne, elles doivent être précédées par un <space> ou <TAB>.

La section commentaire est optionnelle.

Les lignes peuvent être vides de façon à clarifier le code.

Frédéric Amiel – Architecture des ordinateurs – ARM      20/40

Le caractère « ; » en début de ligne indique que la suite (de la ligne) est un commentaire.

**isep** Programme Assembleur

- The following is a simple example which illustrates some of the core constituents of an ARM assembler module:

```

AREA Example, CODE, READONLY
ENTRY

start
    MOV    r0, #15
    MOV    r1, #20
    BL     firstfunc
    SWI   0x11
firstfunc
    ADD    r0, r0, r1
    MOV    pc, lr
END

        ; name this block of code
        ; mark first instruction
        ; to execute
        ; Set up parameters
        ; Call subroutine
        ; terminate
        ; Subroutine firstfunc
        ; r0 = r0 + r1
        ; Return from subroutine
        ; with result in r0
        ; mark end of file

```

The diagram shows an ARM assembly program with annotations:

- start**: A label at the beginning of the code.
- firstfunc**: A label defining a subroutine.
- END**: The end of the program.
- label**: A box pointing to the label "firstfunc".
- opcode**: A box pointing to the instruction "BL firstfunc".
- operands**: A box pointing to the instruction "ADD r0, r0, r1".
- comment**: A box pointing to the comment " ; mark end of file".

Frédéric Amiel – Architecture des ordinateurs – ARM      21/40

Les ; définissent le début de la zone commentaire.

**Description exemple**

<pre> Start     mov r0, #15     mov r1,#20     bl  firstfunc     swi 0x11  Firstfunc     add r0,r0,r1     mov pc,lr </pre>	<p>La routine principale &lt;start&gt; charge les valeurs 15 et 20 dans les registres r0 et r1.</p>
	<p>Le programme appelle le sous programme &lt;firstfunc&gt; par l'instruction BL. (Cette instruction est détaillée par la suite).</p>
	<p>Ce sous programme ajoute 2 paramètres et place le résultat dans r0.</p>
	<p>La fin de sous programme est effectuée en rechargeant le pc par la valeur du lr.</p>
	<p>Après le retour du sous programme, le programme principal appelle l'interruption logicielle 11 (L'instruction SWI est décrite en détail par la suite).</p>

Frédéric Amiel – Architecture des ordinateurs – ARM      22/40

主例程<start>将值 15 和 20 加载到寄存器 r0 和 r1 中。

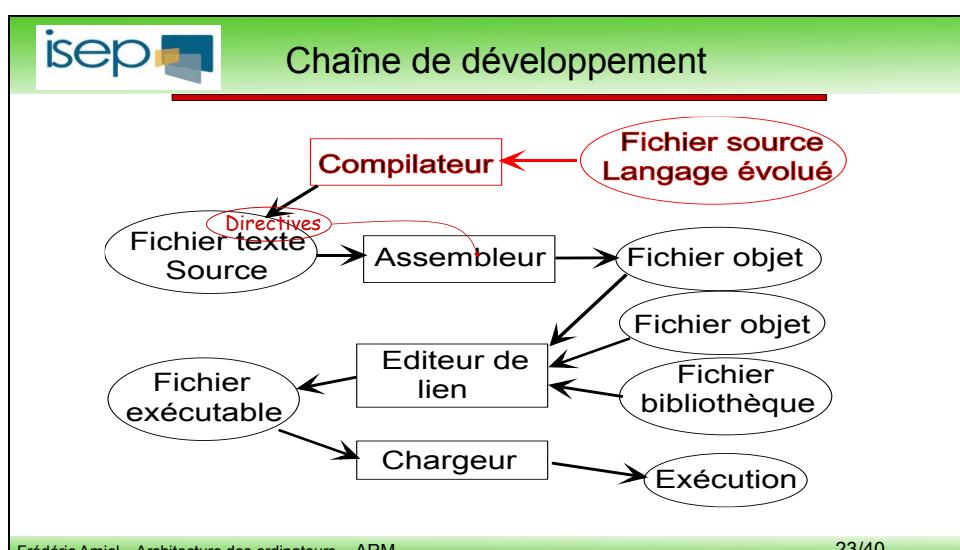
程序<firstfunc>通过 BL 语句调用子例程。 (此说明详述如下)。

此子例程添加 2 个参数并将结果放在 r0 中。

子程序的结束是通过按 lr 的值重新加载 pc 来执行的。

子例程返回后，主程序调用软件中断 11 (SWI 指令将在后面详细描述)

Les programmes sont constitués d'une suite de lignes.



L'entrée des programmes s'effectue par un texte source assembleur. Les compilateurs génèrent ce langage, avec une sortie texte optionnelle.

**isep**

## Pseudo instructions

- AREA
  - Permet de définir une zone logique qui sera affectée à une adresse particulière. (mémoire vive - mémoire morte...).
- ORG
  - Définit une adresse d'implantation (suivant les assembleurs)
- ENTRY
  - Définit le point d'entrée d'un programme
- EQU
  - Permet de définir un symbole (équivalent du #define en C).
- END
  - Indique la fin de l'assemblage
- ALIGN
  - Impose l'alignement sur une adresse modulo 4
- DCB - DCW - DCD - DCQ - DCI
  - Define Constant <Byte> <Half word> <Word> : Écriture d'octets - mots de 16 bits - mot de 32 bits - mots de 64 bits - Entier en mémoire

Frédéric Amiel – Architecture des ordinateurs – ARM

24/40

AREA • 允许您定义将分配给特定地址的逻辑区域。 (内存 – 只读存储器...) • ORG • 定义位置地址 (取决于汇编程序) • ENTRY • 定义 EQU • 程序•的入口点 允许您定义符号 (相当于 C 中的#define) 。 • 结束 指示对齐程序集•的结束 • 强制对齐•到模地址 4 • DCB-DCW-DCD-DCQ-DCI • 定义常量 <Byte><半字>: 字节写入 – 16 位字 – 32 位字 – <Word> 64 位字 – 内存中的整数

Différents mots clés permettent au programme effectuant l'assemblage de prendre en compte certaines directives.

**isep**

## EQU

Les pseudo instructions sont des directives pour l'assembleur. Le processeur ne les connaît pas.

- EQU est utilisé pour définir des noms de constantes significatifs

```

nb_boucl    equ    10
...
tempo        mov    r0,nb_boucl   ; initialisation de R0 à 10
bcl          subs   r0,r0,#1     ; r0-1
                bne   bcl       ; si r0 <> 0 , rebouclage
...

```

Remarque :

Cette fonction « tempo » effectue une temporisation. Elle dépend de la vitesse de l'horloge du processeur, ce n'est pas une fonction « portable ».

Frédéric Amiel – Architecture des ordinateurs – ARM

25/40

La pseudo instruction EQU permet de clarifier la lecture des programmes.

**isep** DCx - SPACE

- **DCx** permet de définir des données initialisées en mémoire.
- **SPACE** permet de réserver de la place en mémoire.

<pre>Constant_number dcd 0x12345678 Message dcb « Hello world »,10,13,0 table space 10</pre>	<p>La mémoire est initialisée avec les valeurs spécifiées</p> <p>Cette instruction réserve 10 octets libres en mémoire</p>
--	--

Frédéric Amiel – Architecture des ordinateurs – ARM      26/40

Ces directives agissent directement sur les informations binaires générées par l'assembleur.

**isep** Pseudo instructions LDR / ADR

Il est impossible sur le processeur ARM d'initialiser directement un registre par une valeur 32 bits. <expliquer pourquoi>

Si on souhaite le faire, on utilise un chargement indirect. La pseudo instruction LDR simplifie l'écriture.

1) Initialisation avec une valeur pouvant être codée sur 8 bits : MOV

```
MOV r0,#0,65 ; R0 ← 0x41
MOV r1,0x10 LSL 8 ; R0 ← 0x 1000
```

2) Initialisation avec une valeur 32 bits : chargement indirect

```
LDR r0,[pc, #constant_number-8-{PC}] ; R0 ← 0x12345678
...
Constant_number
DCD 0x12345678
```

Frédéric Amiel – Architecture des ordinateurs – ARM      27/40

**isep**

## Pseudo instruction LDR / ADR

- Pour simplifier l'écriture précédente, on peut utiliser la pseudo instruction LDR

Pseudo instruction	Instruction réellement employée
LDR r0,=0xff	MOV r0,#0xff
LDR r1,-0x12345678	ldr r1,[pc,#offset_12]

- On utilise aussi l'instruction LDR pour initialiser un pointeur

Pseudo instruction	Instruction réellement employée
LDR r0,=table	LDR R0,[pc,#table]
...	
table	
DCD 0x81,0x42,0x24,0x18	

Frédéric Amiel – Architecture des ordinateurs – ARM      28/40

**isep**

## Appels de fonctions

- Il est fréquent d'utiliser plusieurs fois le même code à des endroits différents du programme
  - Affichage d'un caractère
  - Sous partie d'un calcul
  - ....
- Les programmes comprennent en général de nombreux appels imbriqués

Frédéric Amiel – Architecture des ordinateurs – ARM      29/40

Les appels de sous programmes sont très fréquents dans les programmes.

**isep** Appels de fonctions

- Il faut sauvegarder l'adresse de retour avant l'appel

```

graph TD
    PC --> BL[BL]
    BL --> affich[affich]
    affich --> Mov[Mov]
    affich --> Retour((Retour au programme appelant))
    Mov --> pc_lr[pc,lr]
    pc_lr --> BL
  
```

L'instruction BL (branch and link) sauvegarde l'adresse de l'instruction suivante (PC = R15) dans le registre LR = r14

Frédéric Amiel – Architecture des ordinateurs – ARM 30/40

**isep** Appels imbriqués

- Lors des appels imbriqués il faut mémoriser plusieurs adresses de retour :

```

graph TD
    PC --> BL_feuille1[BL feuille_1]
    BL_feuille1 --> feuille1[feuille_1]
    feuille1 --> BL_feuille2[BL feuille_2]
    BL_feuille2 --> mov_pc_lr1[mov pc,lr]
    mov_pc_lr1 --> BL_feuille1
    mov_pc_lr2[mov pc,lr] --> feuille1
  
```

Au niveau du deuxième sous programme, il faut sauvegarder le registre LR.

Frédéric Amiel – Architecture des ordinateurs – ARM 31/40

Dès qu'il y a plus d'un niveau d'imbrication, il faut mémoriser l'ancienne adresse de retour.

**isep**

## Instruction BL

Bcl	Mov r0,#0x41 ; r0 = 'A'
	Bl putc ; Appel de la fonction putc
	Add r0,r0,1 ; Caractère suivant
	Teq r0,#'J' ; test r0 = code ascii de 'J'
	Bne bcl ; si différent, bouclage
...	

L'instruction BL permet d'effectuer des appels de sous programmes

L'adresse de retour est sauvegardée dans le registre R14 (Link register : LR).

Cette instruction ne gère pas les appels de sous programmes à plusieurs niveaux

Frédéric Amiel – Architecture des ordinateurs – ARM      32/40

L'instruction BL effectue l'appel de sous programmes :  
(Branch and Link)

**isep**

## Gestion de pile

- La pile peut recevoir plusieurs adresses
- La pile ressort les adresses
- La dernière adresse entrée est la première à ressortir
- Pile LIFO

Last In First Out

Pile des adresses de retour

Frédéric Amiel – Architecture des ordinateurs – ARM      33/40

Les adresses de retour doivent être placées dans une pile LIFO.

**Pile et pointeur de pile**

- La pile est une mémoire LIFO
- Sur certains processeur elle est ajoutée au jeu de registres internes :

The diagram illustrates a processor architecture where the stack is implemented within the internal register set. A central box labeled "Processeur" contains several horizontal slots representing registers. To the left, a group of four arrows points from the text "Jeu de registres internes" to these slots. To the right, another arrow points from the text "Zone de pile" to a separate rectangular box labeled "Zone de pile". Below the processor is a horizontal line labeled "Bus de communication".

Dans ce cas, la pile a une taille fixe déterminée par le constructeur du processeur

→ Le nombre de niveaux de sous programme est limité  
→ Microcontrôleurs

Frédéric Amiel – Architecture des ordinateurs – ARM      34/40

**Pointeur de pile**

- Dans les processeurs à usage généraux, la pile est placée en mémoire référencé par un registre spécifique du processeur : le pointeur de pile.
- Pour le processeur ARM, le pointeur de pile est le registre R13

This diagram shows a more general architecture. On the left, a "Processeur" box contains a "Registre" slot and a "Pointeur de pile" slot. Arrows from "Jeu de registres internes" point to both the "Registre" and "Pointeur de pile" slots. A "Bus de communication" line connects the processor to a "Mémoire" box on the right. The "Mémoire" box contains a vertical stack of memory blocks, with the top one labeled "Zone mémoire consacrée à la pile".

Frédéric Amiel – Architecture des ordinateurs – ARM      35/40

Le registre R13 est usuellement utilisée pour pointer la zone mémoire contenant les adresses de retour (pile LIFO). On l'appelle aussi SP (Stack Pointer)

**Appel de sous programme**

- Lors des sauts vers les sous programmes, l'adresse de retour est placée dans la pile

The diagram illustrates the state of memory during a subroutine call. On the left, a 'Processeur' (Processor) has a stack pointer (SP) pointing to a stack frame in 'Mémoire' (Memory). The stack frame contains the address of the next instruction ('addrret') and a return address ('+') which points back to the caller. On the right, a code snippet titled 'Feuille\_2' shows the assembly instructions: 'stmfd sp!,{regs,lr}' followed by three ellipses, and then 'ldmfd sp!,{regs,pc}'.

STMFD : STORE Multiple Full Descending Stack  
LDMFD : LOAD Multiple Full Descending Stack

Frédéric Amiel – Architecture des ordinateurs – ARM      36/40

En début de sous programme, on sauvegarde les différents registres qui vont être utilisés par la fonction en cours dans la pile LIFO.

En fin de sous programme, on les restitue, de façon à ce qu'ils reprennent la valeur qu'ils avaient dans le programme appelant.

Les instructions STM et LDM (Store Multiple et Load Multiple) permettent d'effectuer ces sauvegardes / restitutions.

L'instruction stmfda sauvegarde dans la mémoire les registres et le registre R14 (LR) qui contient l'adresse de retour.

La dernière instruction de la fonction : ldmfd restitue les registres sauvegardés et le registre PC (r15) en supplément, ce qui permet de revenir au programme appelant.

**Initialisation du SP**

- Dans les programmes utilisant la pile, il faut initialiser le SP avec une adresse pointant de la RAM disponible.
- Sur le processeur ARM
  - L'instruction BL est l'appel de sous prog
  - L'instruction mov pc,lr est le retour
  - Des le deuxième niveau de sous programme, il faut sauvegarder le registre LR dans une LIFO en mémoire
  - Les instructions STMFD et LDMFD permettent de gérer une pile LIFO en mémoire

```

ldr    r13,=esp_pile ; init du SP
bl    tempo           ; appel de sous prog «
...
esp_pile
space     100          ; Reservation de 100
                      ; octets pour la pile

```

Frédéric Amiel – Architecture des ordinateurs – ARM      37/40

Durée d'exécution		
Instruction	Nb cycle	commentaire
ALU (Instruction arith / logique)	1	+1 si shift +2 si Rd est pc
B, BL, BX	3	
CDP (co processor)	1+B	
LDC (co processor)	1+B+N	
LDR/B/H...	3	+2 si rd est PC
LDM	2+N	
MCR (co processor)	2+B	
MLA	2+M	Mul Acc
xMLAL	3+M	

Frédéric Amiel – Architecture des ordinateurs – ARM

38/40

ALU représente les instructions utilisant l'ALU comme : add, sub etc... (instructions de traitement de données).

Le processeur ARM7 utilise un pipeline à 3 étages. (Cf cours ARC201). C'est pour cette raison que chaque fois qu'on touche au registre PC, on perd 3 cycles horloge. (instruction B par exemple).

Durée d'exécution		
Instruction	Nb cycle	commentaire
MRC	3+B	
MRS, MSR	1	Mov / PSR
MUL	1+M	
xMUL	2+M	
STC	1+B+N	
STR/B/H	2	
STM	1+N	
SWI	3	
SWP/B	4	Swap register / memory

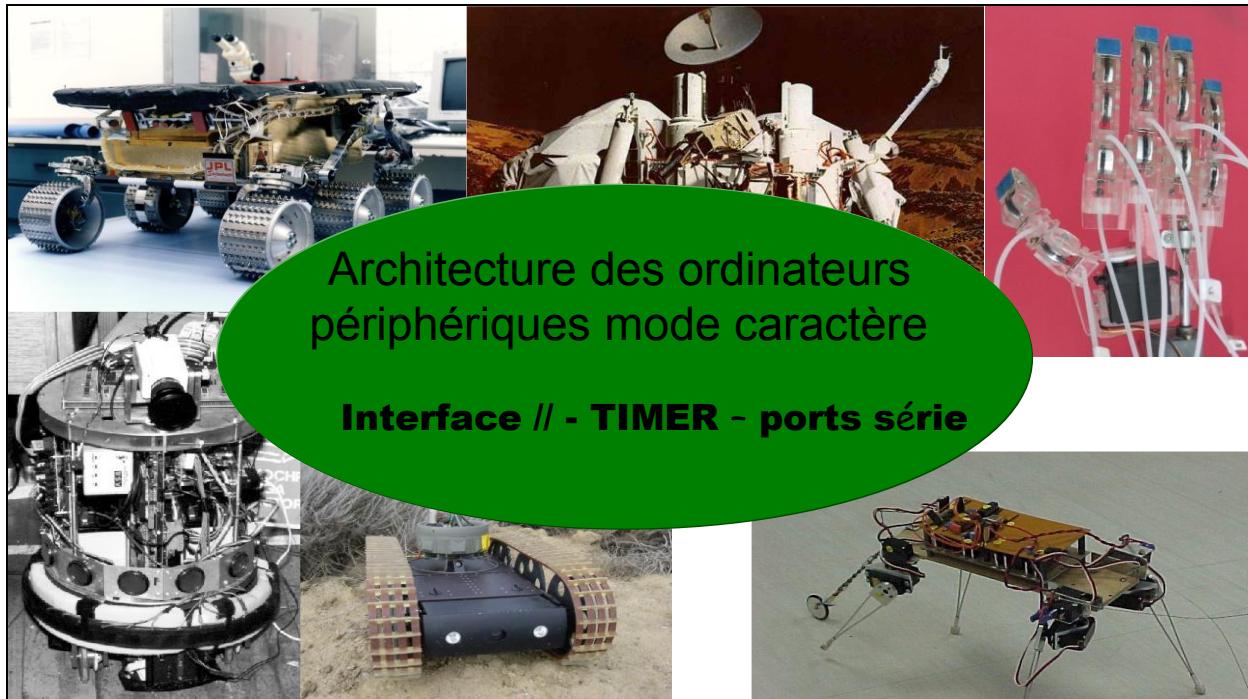
Frédéric Amiel – Architecture des ordinateurs – ARM

39/40

**isep** Auto test

- Quelle instruction correspond au code:
  - 0x00000000
  - 0x12345678
- Ecrire de deux façons différentes un programme permettant de tester R0
  - Si r0 == 5 alors r1 = 4
  - Sinon r1 = 3
- Quel est le contenu de R0 après :

```
Ldr r2,=0x12345678
Str r2,[r3]           // r2 → M[r3]
Ldrb r0,2[r3]
```



## Objectifs

- Connaître la notion de périphérique mode caractère et de périphérique mode bloc
- Comprendre la programmation d'un interface parallèle
- Comprendre la notion de TIMER
- Comprendre le fonctionnement de quelques ports série
- Pooling
- Introduction à la notion d'interruption

**isep**  
École d'ingénieurs du numérique

## Périphériques

Les périphériques permettent au système de communiquer avec le monde extérieur

- Saisir des informations
- Provoquer des actions

Différents périphériques permettent de :

- Lire des états logiques / analogiques
- Générer des états logiques / analogiques
- Communiquer avec d'autres circuits
- etc

Frédéric Amiel – Architecture des ordinateurs – ARM      3/52

**isep**  
École d'ingénieurs du numérique

## Microcontrôleur/processeur

Un microcontrôleur est un composant qui contient un (ou plusieurs) processeur associés à de la mémoire (morte et vive); des périphériques permettant la communication avec le monde extérieur et des circuits de gestion (fréquence de fonctionnement, circuiterie de RESET)

Ce composant permet de disposer d'un système complet avec un minimum de composants annexes

PERIPHERIQUES

Frédéric Amiel – Architecture des ordinateurs – ARM      4/52

**isep**  
École d'ingénieurs du numérique

## Mode caractère

- Les ordinateurs disposent de 2 types de périphériques

<ul style="list-style-type: none"> <li>• Mode bloc : l'échange de donnée s'effectue par bloc de qq <math>10^2</math> - <math>10^3</math> octets           <ul style="list-style-type: none"> <li>• Disques</li> <li>• Réseau</li> <li>• Ecran graphique</li> <li>• ...</li> </ul> </li> </ul> <p style="margin-top: 20px;">DMA operation (DIRECT MEMORY ACCESS) (Pas abordé dans ce cours)</p>	<ul style="list-style-type: none"> <li>• Mode caractère : L'échange de données s'effectue octet par octet:           <ul style="list-style-type: none"> <li>• Console texte</li> <li>• Souris</li> <li>• Ligne faible débit</li> <li>• Entrées sorties TOR</li> <li>• ...</li> </ul> </li> </ul> <p style="margin-top: 20px;">Le processeur gère l'échange de donnée</p>
--	--

Frédéric Amiel – Architecture des ordinateurs – ARM      5/52

**isep**  
École d'ingénieurs du numérique

## Plan mémoire unique

- Périphériques mappés dans l'espace mémoire :

Le processeur ARM « voit » un espace mémoire de 4 Goctets (32 bits)

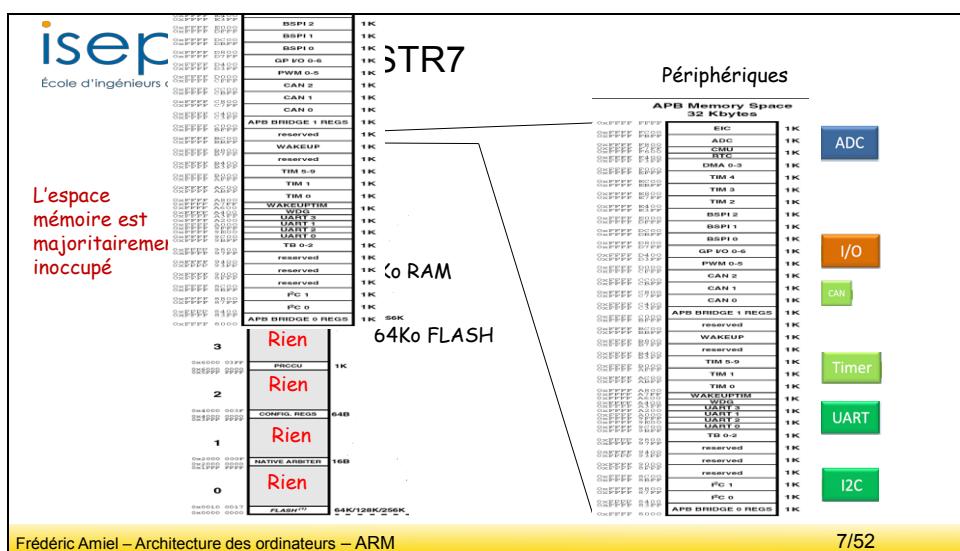
(on peut spécifier 4  $10^9$  adresses différentes) (les registres font 32 bits)

Ces 4 Goctets référencent :

De la mémoire morte  
De la mémoire vive  
Des composants d'entrées / sorties

On accède à un composant d'E/S par son adresse

Chaque périphérique se programme par plusieurs « registres périphériques » vus comme des cases mémoire. Le processeur accède à ces registres en écrivant ou en lisant aux adresses mémoire où est implanté le périphérique.



Le processeur ARM7 peut référencer  $4 \cdot 10^9$  adresses mémoires.

Le composant utilisé en TP ne dispose que de 64Ko de mémoire FLASH, 16 Ko de mémoire vive, et l'ensemble des périphériques est regroupé dans un espace mémoire qui occupe 32 Ko. Le reste des cases mémoire accessible n'est pas employé.

开始

插入

绘图

设计

切换

动画

幻灯片放映

审阅

视图



B

I

U

aA

|

|

|

|

|

|

|

|

|

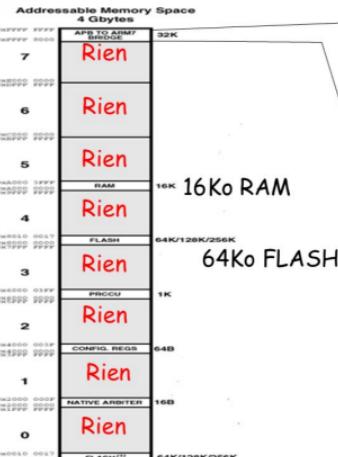
点击此处试用格式刷



## MAP STR7

## Périphériques

L'espace mémoire est majoritairement inoccupé



APB Memory Space 32 Kbytes	
0xFFFFF_FFFF	EIC
0xFFFFF_FDDP	ADC
0xFFFFE_E200	CMU
0xFFFFE_E400	PLL
0xFFFFE_E500	DMA 0-3
0xFFFFE_E900	TIM 4
0xFFFFE_E900	TIM 3
0xFFFFE_E900	TIM 2
0xFFFFE_E900	BSPI 2
0xFFFFE_E900	BSPI 1
0xFFFFE_E900	BSPI 0
0xFFFFE_E900	GP I/O 0-6
0xFFFFE_E900	PWM 0-5
0xFFFFE_E900	CAN 2
0xFFFFE_E900	CAN 1
0xFFFFE_E900	CAN 0
0xFFFFE_E900	APB BRIDGE 1 REGS
reserved	reserved
reserved	WAKEUP
reserved	reserved
reserved	TIM 5-6
reserved	TRNG
reserved	TIM 0
reserved	WAKEUP/TIM
reserved	WDOG
reserved	UART 3
reserved	UART 2
reserved	UART 0
SFFFFFF_8000	TB 0-2
SFFFFFF_8000	reserved
SFFFFFF_8000	reserved
SFFFFFF_8000	reserved
SFFFFFF_8000	PC 1
SFFFFFF_8000	PC 0
SFFFFFF_8000	APB BRIDGE 0 REGS

ADC

I/O

CAN

Timer

UART

I2C

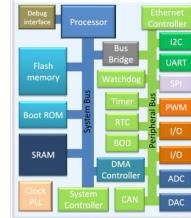
Frédéric Amiel – Architecture des ordinateurs – ARM

7/52

**isep** École d'ingénieurs du numérique

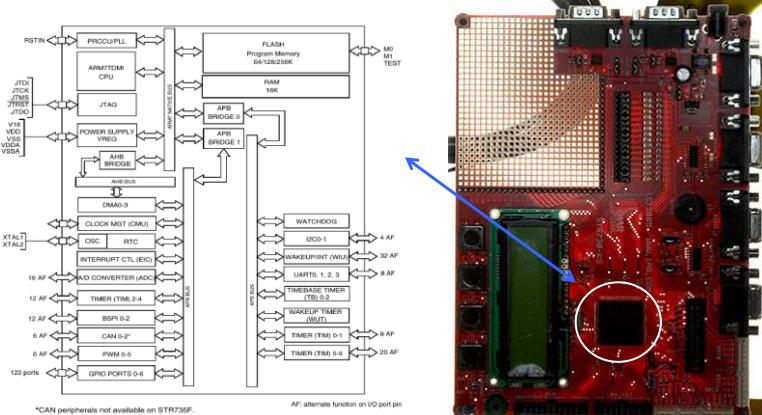
## Microcontrôleur

- Un microcontrôleur est un composant qui comprend
  - 1 ou plusieurs processeurs
  - De la mémoire morte (de la FLASH et parfois un peu d'EEPROM)
  - De la mémoire vive
  - Plusieurs types de périphériques d'entrées/sorties
    - Entrées/Sortie logique
    - Entrée Analogique
    - Bus de communication (I2C - SPI - USB - UART - CAN ...)
  - Une circuiterie de génération d'horloge et de reset
  - ...
- Un microcontrôleur permet de disposer d'un système complet sur un seul composant



**isep** École d'ingénieurs du numérique

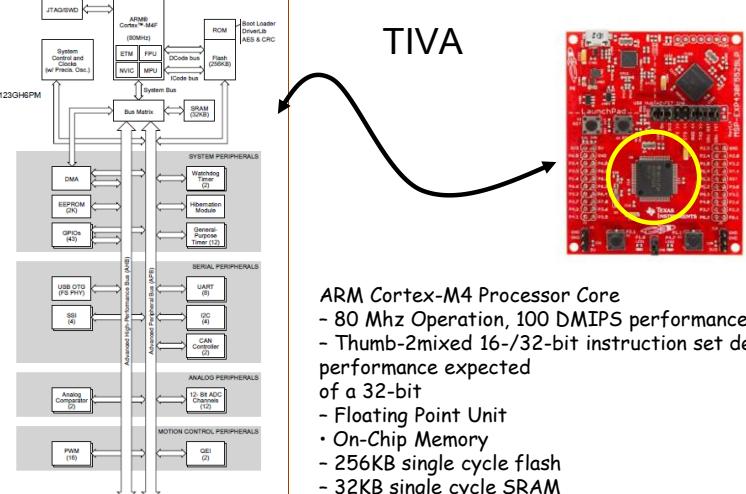
## STR7



\*CAN peripherals not available on STR735F.

AF: alternate function on I/O port pin

**TIVA**



**ARM Cortex-M4 Processor Core**

- 80 Mhz Operation, 100 DMIPS performance
- Thumb-2mixed 16-/32-bit instruction set delivers the high performance expected of a 32-bit
- Floating Point Unit
- On-Chip Memory
- 256KB single cycle flash
- 32KB single cycle SRAM

**Broche - périphérique**

Une même broche du microcontrôleur peut être affectée à différents périphériques

Frédéric Amiel – Architecture des ordinateurs – ARM      11/52

**Périphériques - programmation**

- De façon générale, les microcontrôleurs disposent en interne de beaucoup de périphériques. Chaque périphérique peut être « éteint » ou « actif »

Frédéric Amiel – Architecture des ordinateurs – ARM      12/52

**GPIO - Validation**

Un port d'entrées / sorties dispose de 8 ou 16 broches  
De façon générale il faut activer le port. Sur Le STR7 le port est systématiquement actif.  
Sur TIVA chaque périphérique (ici le GPIO) dispose d'une horloge qu'on peut activer ou non.  
(Clock gating)

General-Purpose Input/Output Run Mode Clock Gating Control

I/O

Les périphériques non utilisés ne sont pas activés : ils ne consomment pas (ou peu)

Chaque bit du registre RCGCGPIO permet d'activer un port.  
Ce registre est disponible à l'adresse 0x0400F.E000

General-Purpose Input/Output Run Mode Clock Gating Control (RCGCGPIO)  
Base 0x400F.E000  
Offset 0x00000000  
Type RW, reset 0x00000000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Type	RO														
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Type	RO														
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Port F    Port C    Port A    Port E    Port B

Frédéric Amiel – Architecture des ordinateurs – ARM      13/52

## General Purpose Input Output - GPIO – (Port parallèle)

- Ce périphérique de base permet par programme de :

- De générer un état logique (0 ou 1)
- De capter un état logique (0 ou 1)

I/O

Sur une broche spécifique du composant

Le STR7 gère de 122 entrées/sorties GPIO

La carte TIVA : 43

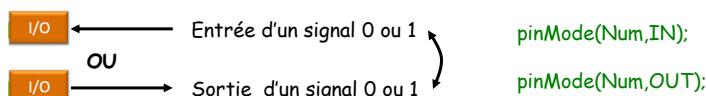
Le microcontrôleur de l'ARDUINO UNO : 14

Raspberry PI : 32

...

## GPIO - Programmation

De façon générale chaque broche du GPIO peut se programmer en entrée ou en sortie.

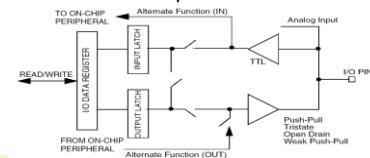


Un registre de direction permet de déterminer le sens (entrée ou sortie) pour chaque broche :  
GPIODIR

Un registre de donnée permet d'affecter un état ou de lire un état sur chaque broche :  
GPIODATA

Chaque broche du port d'entrée/sortie correspond à un bit précis de ces registres

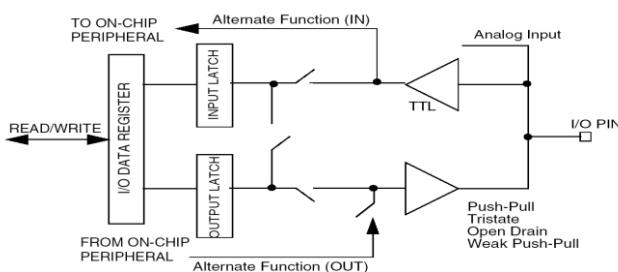
On programme 8 bits ou 16 bits à la fois



## GPIO – Configuration

Pour le GPIO la broche peut être :

I/O



Les périphériques sont accessibles par programmation à des adresses mémoire définies par le constructeur du microcontrôleur

STR7 : PORT 0 adresse : 0xFFFFFD400  
TIVA PORT F : adresse : 0x4002.5000

GPIO : General Purpose Input Output. Ce périphérique permet de lire ou d'écrire des états logiques (tout ou rien). C'est le périphérique le plus fondamental. On peut par exemple lire un chiffre 8 bits en utilisant 8 broches GPIO chacune connectée à un bit du chiffre à lire.

**isep**  
École d'ingénieurs du numérique

## Programmation direction

I/O

- Les E/S du GPIO sont groupées par PORT
- Un PORT fait 8 bits ou 16 bits selon le constructeur
- On configure chaque bit en entrée (0) ou en sortie (1)

b15 b14 b13 b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0	Exemple : Port 2 du STR7 (16 bits) 0x00FF
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1	
8 broches IN      8 broches OUT	
b7 b6 b5 b4 b3 b2 b1 b0	Exemple : PortA TIVA PA0 à PA4 en entrée PA5 à PA7 en sortie
1 1 1 0 0 0 0 0	0xE0
3 OUT      5 IN	

Frédéric Amiel – Architecture des ordinateurs – ARM      17/52

**isep**  
École d'ingénieurs du numérique

## Lecture/Ecriture d'un bit

I/O

- LECTURE :**
  - On lit l'état du port ENTIER
  - On sélectionne l'état du bit qui nous intéresse pour le tester

```
Ldr r10,0xfffffD40C ; Adresse du port 0 (STR7)
Ldrb r0,[R10]          ; Lecture du port
Ands r0,#0x8           ; Selon le bit 3 r0=0 ou r0≠0
```
- Ecriture :**
  - On lit l'état du port ENTIER dans un registre du processeur
  - On modifie l'état du bit qui nous intéresse
  - On écrit le résultat sur le port

```
Ldr r10,0xfffffD40C ; Adresse du port 0 (STR7)
Ldrb r0,[R10]          ; Lecture du port
Orr r0,0x8            ; Mise à 1 du bit 3
Str r0,[R10]           ; Ecriture dans le port
```

Frédéric Amiel – Architecture des ordinateurs – ARM      18/52

**isep**  
École d'ingénieurs du numérique

## Lecture/Ecriture d'un bit

I/O

- **LECTURE :**
  - On lit l'état du port ENTIER
  - On sélectionne l'état du bit qui nous intéresse pour le tester

```
int *p = 0xffffd400c
val *p;
val = val & 0x8;
```
- **Ecriture :**
  - On lit l'état du port ENTIER dans un registre du processeur
  - On modifie l'état du bit qui nous intéresse
  - On écrit le résultat sur le port

```
int *p = 0xffffd400c
val = *p;
val = val | 0x8;
*p = val;
```

Frédéric Amiel – Architecture des ordinateurs – ARM      19/52

**isep**  
École d'ingénieurs du numérique

## Port parallèle (GPIO)

I/O

- Les ports parallèles sont utilisés pour les entrées / sortie TOR (tout ou rien)
- OUT
  - Commandes de mécanismes
    - Lumières
    - Moteurs
    - Vannes
    - ...
- IN
  - Lectures de capteurs
    - Interrupteurs
    - Mesures physiques
    - ...
- IN/OUT
  - Gérer des communications

Frédéric Amiel – Architecture des ordinateurs – ARM      20/52

Les ports parallèles peuvent lire ou écrire plusieurs bits en même temps. On peut aussi s'en servir pour récupérer des chiffres qui représentent des mesures issues de capteurs.

**isep**  
Ecole d'ingénieurs du numérique

## Timer

Timer

- Les TIMER disposent de leur propre horloge (quartz indépendant), ils permettent d'avoir une référence de temps utilisable pour :
  - Mesurer des temps
  - Générer des signaux carrés paramétrables
    - (Fréquence et rapport cyclique)
  - Mesurer des durées d'impulsions
  - Et également de :
    - Compter des impulsions

Frédéric Amiel – Architecture des ordinateurs – ARM      21/52

Les timers sont des périphériques indépendants permettant de gérer le temps indépendamment de la vitesse d'exécution du processeur.

**isep**  
Ecole d'ingénieurs du numérique

## Timer

Timer

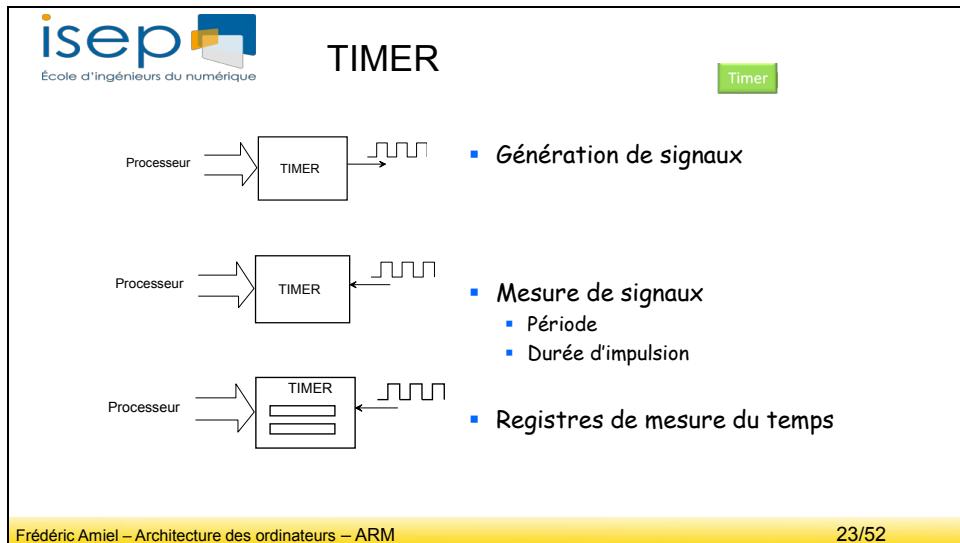
- Les Timer permettent d'avoir une référence de temps indépendante de l'horloge du processeur
- L'utilisation de Timer ne requiert pas de temps processeur.

Exemple : génération d'un signal carré :

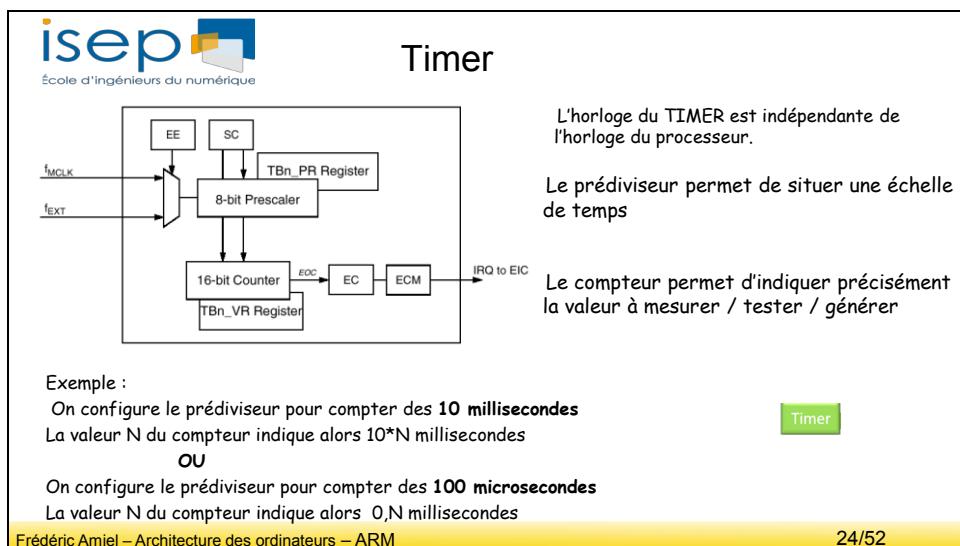
<p>Avec un GPIO : le processeur écrit un 0, boucle un moment, écrit un 1 boucle un moment et recommence</p> <pre>Sans timer : Bcl strh R0,PD0 bl tempo strh r1,PD0 bl tempo B bcl</pre> <p>Le processeur est actif en permanence : Il n'est pas utilisable par une autre tâche Il consomme de l'énergie</p>	<p>Avec un TIMER : le processeur programme le TIMER pour générer un carré avec la fréquence désirée</p> <pre>Avec Timer : strh r0,ocar strh r1,ocbr</pre> <p>Le processeur n'est plus mobilisé pour générer le signal</p>
---	---

Frédéric Amiel – Architecture des ordinateurs – ARM      22/52

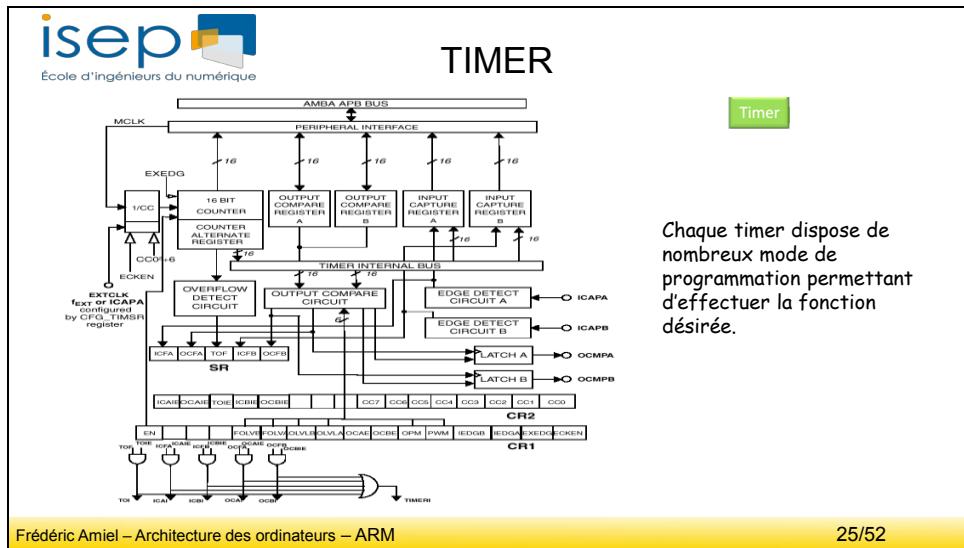
La programmation d'un timer consiste essentiellement à configurer le composant.



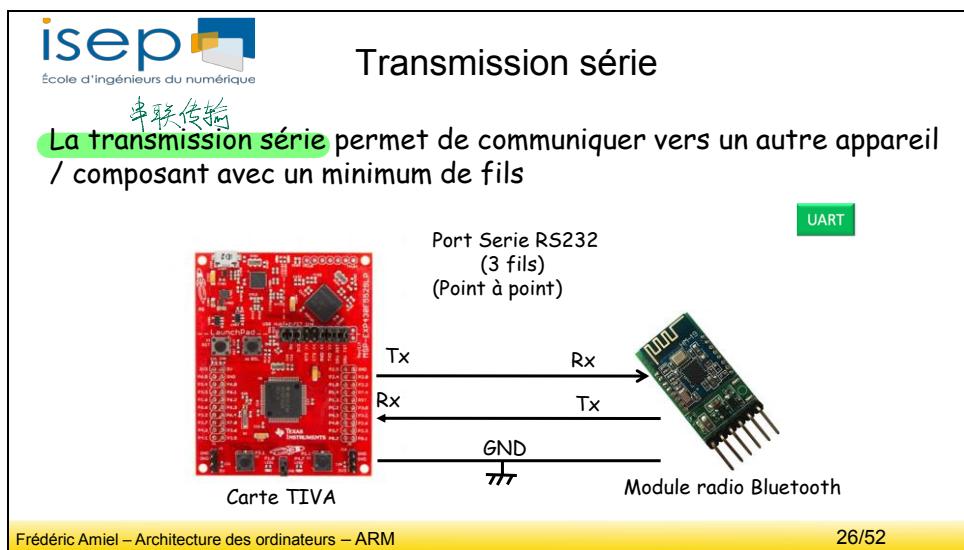
Différentes fonctionnalités sont possibles. Les timers permettent non seulement de lire le temps par programme, mais aussi de générer ou de prendre en compte des signaux extérieurs.



EN général ces composants permettent un « réglage fin » par un registre de comptage, et une « échelle de temps » par l'intermédiaire d'un prédiviseur.



Plusieurs registres sont utilisés pour couvrir les différentes possibilités du périphérique.



Le protocole de transmission série est très connu et très utilisé.

**isep**  
École d'ingénieurs du numérique

## Transmission série

La transmission série permet de communiquer vers un autre appareil / composant avec un minimum de fils

Carte TIVA

Port I2C (3 fils)  
Plusieurs périphériques

SCL - SDA - GND

Afficheur LCD  
Accéléromètre

I2C

Frédéric Amiel – Architecture des ordinateurs – ARM

27/52

Le protocole de transmission série est très connu et très utilisé.

**isep**  
École d'ingénieurs du numérique

## Communication série

- RS232 :
  - ordinateur / écran clavier texte
  - ordinateur / périphérique bas débit
    - GPS
    - Module radio bas débit
    - ....
- I2C
  - ordinateur / afficheur
  - ordinateur / Capteurs
  - ...
- SPI
  - Ordinateur / convertisseur AD/DA
  - ...

Plusieurs mètres

Différents interfaces électriques normalisés

Débit de 100 bp/s à 110Kbp/s

Point à point

Qq dizaines de centimètres  
(composants sur la même carte)

Débit de 100 - 400 Kbps (quelques modes plus rapides  
(5Mbp/s))

127 périphériques adressables

Quelques centimètres

Débit de 1,5 - 3 Mbp/s (20Mbp/s)

2 à 4 périphériques sélectionnables

UART

I2C

Frédéric Amiel – Architecture des ordinateurs – ARM

28/52

**isep** École d'ingénieurs du numérique

## Transmission RS232

UART

- Transmission asynchrone (on ne transmet pas l'horloge)
- On définit à l'avance la vitesse et le format
  - Exemple 7 bits parité paire / Transmission du chiffre 0x53 = 101 0011

Bit de start : Indique le début de transmission d'un mot  
Le chiffre est transmis en commençant par le bit 0  
Le bit de parité paire est à 0 (cf slide 31)  
Bit de stop : retour à l'état initial

Frédéric Amiel – Architecture des ordinateurs – ARM      29/52

**isep**

## Reception RS232

UART

Bit de start détecté : Attente  $T/2$   
Echantillonnage tous les  $T$   
Echantillonnage parité  
Echantillonnage bit de stop

L'horloge entre émetteur et récepteur n'étant pas commune il y a un petit décalage  
Fonctionne si  $\pm \frac{T}{10T} = 5\%$  de précision entre les horloges

Frédéric Amiel – Architecture des ordinateurs – ARM      30/52

Ce protocole utilise un overhead (bit de start, bit de stop, **bit de parité**) important (30% du message), ce qui en fait un système robuste. Il est très adapté aux **échanges bas débits**.  
奈何技术化  
低速交换

**isep**  
École d'ingénieurs du numérique

## Contrôle : bit de parité

UART

- Le bit de parité permet de contrôler qu'il n'y a pas eu une erreur sur un bit :

Un éventuel parasite modifie le nombre de bit à 1 :

**Parité paire :**  
Le nombre de bit à 1 du message est pair

**Parité impaire :**  
Le nombre de bit à 1 du message est impair

5 bits à 1 normalement : le bit de parité paire est placé à 1 par l'émetteur

Le récepteur constate une erreur liée à un parasite

Frédéric Amiel – Architecture des ordinateurs – ARM

31/52

Le bit de parité permet de vérifier l'intégrité de la transmission parasité par un (et un seul) événement externe.

**isep**  
École d'ingénieurs du numérique

## Parité

UART

- L'émetteur a placé le bit de parité à 1
- Le récepteur reçoit un 1 à la place du 0 au bit 3. Le récepteur calcule un bit de parité paire à 0.
- L'incohérence permet de détecter l'erreur de transmission.

Frédéric Amiel – Architecture des ordinateurs – ARM

32/52

Le bit de parité permet de détecter l'erreur de transmission, un protocole de plus haut niveau doit alors gérer cette erreur.

**ISEP**  
École d'ingénieurs du numérique

## RS232 Paramètres

- Vitesse en bits par seconde (bps) ou en bauds (nombre de symboles par secondes, ici les symboles sont des bits).
  - 75 bauds (Minitel - Utilisateur → Serveur)
  - 150 - 300 - 600
  - 1200 bauds (Serveur → Minitel) - 2400 - 4800 (Fax) - 9600
  - 19200 (Console mode texte) 28k - 33k - 38k - 56k - 64k - 110 k
  - ... : Remarque : 9600 Bauds ≈ 960 caractères / sec : 7 bits, parité paire
- Nb de bits transmis : 5 - 6 - 7 - 8 - 9 (**Ascii** ou **Binaire**)
- Parité :
  - Paire : Le nombre à 1 doit être paire
  - Impaire : Le nombre de bit à 1 doit être impair
  - No : Pas de parité
- Nb de bit de Stop : 1 - 1.5 - 2

UART

Frédéric Amiel – Architecture des ordinateurs – ARM      33/52

En mode texte, on utilise classiquement du 7 bits, parité paire à 19200 bauds.  
En mode binaire, du 8 bits, sans parité à 19200 bauds ou au-delà.

**ISEP**  
École d'ingénieurs du numérique

## Interface ligne RS232

- Différents interfaces permettent de s'adapter à des milieux plus ou moins parasités
- RS232 : 0 logique = +3 / +18v  
1 logique = -3 / -18 v

UART

L'immunité au bruit est meilleure avec une dynamique de tension plus grande  
Généralement : ± 12v

Frédéric Amiel – Architecture des ordinateurs – ARM      34/52

L'interface électrique permet d'augmenter le rapport **signal / bruit**.

**ISEP**  
Ecole d'ingénieurs du numérique

## RS485

- La transmission s'effectue par des signaux différentiels :

Le parasite agit sur les 2 fils de transmission, la tension différentielle n'est pas perturbée

Dans ce cas on augmente le nombre de fils  
(USB)

UART

Frédéric Amiel – Architecture des ordinateurs – ARM      35/52

L'interface RS485 est très robuste vis-à-vis des perturbations électriques extérieures.

**ISEP**  
Ecole d'ingénieurs du numérique

## Protocole

- Le protocole permet d'éviter la saturation du récepteur :

- Transmission binaire : protocole matériel RTS/CTS
- Transmission texte : protocole XON / XOFF

UART

Emetteur

Récepteur

alkjh csdfpjh  
sdlmfh sdmfh  
sd fplkhsd m  
sdmkijfhmdlh  
ldskfmldskfjm

Le récepteur est saturé (exemple :  
imprimante : plus de papier)

Frédéric Amiel – Architecture des ordinateurs – ARM      36/52

Le protocole permet de régler les saturations du récepteur. Il existe un protocole qui se mêle aux données (code ASCII) et un protocole matériel.

**XON/XOFF**

- Pour des données texte (ASCII), 2 caractères permettent au récepteur :
  - De demander à l'émetteur de suspendre les émissions
    - XOFF (Code ascii 19 : ^S)
  - De demander à l'émetteur de reprendre les émissions
    - XON (Code ascii : 17 : ^Q)

UART

Suspension de l émission

Les 2 machines peuvent émettre et  
Recevoir Les données échangées  
sont ASCII

Frédéric Amiel – Architecture des ordinateurs – ARM      37/52

Le protocole xon xoff utilise des caractères réservé dans le code ASCII.

**Programmation**

UART

- Différents registres permettent de contrôler ce port série.
  - Un registre permettant de définir la vitesse.
  - Un registre de commande définissant le format des trames
  - Un registre d'état permettant de savoir si les buffers de réception et d'émission sont libres, de vérifier qu'il n'y a pas eut d'erreurs...
  - Un registre de donnée permettant de lire ou de recevoir les octets

Frédéric Amiel – Architecture des ordinateurs – ARM      38/52

Concernant la programmation, on configure dans un premier temps la communication. Les données reçues et à transmettre sont alors lues / écrite par le processeur dans un registre. Le synchronisation permettant de savoir si la donnée est nouvelle ou non peut s'effectuer par pooling ou interruption. (Le poolong consiste à scruter un drapeau, la technique des interruption est décrite plus loin dans le cours).

**ISEP**  
École d'ingénieurs du numérique

## UART

UART : Universal Asynchronous Receiver Transmitter

Ces périphériques permettent de gérer les transmissions série. Ils peuvent intégrer des protocoles de bas niveau.

Addr. Offset	Register Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	UART_BaudRate																
4	UART_TxBuffer	UART_BaudRate															
8	UART_RxBuffer	UART_TxBuffer															
C	UART_CR	Reserved	FIFOEnable	Request	Retries	Run	LongBreak	Promisc	Stop	Bits	Mode						
10	UART_IER	UART_RxBuffer															
14	UART_SR	UART_IER															
1C	UART_TOR	UART_SR															
20	UART_TxReset	UART_TOR															
24	UART_RxReset	UART_TxReset															

Différentes cases mémoires (on parle alors de registres périphériques) permettent de gérer les différents paramètres.

Frédéric Amiel – Architecture des ordinateurs – ARM      39/52

Le nom de l'interface série asynchrone est « UART ».

**ISEP**  
École d'ingénieurs du numérique

## FIFO de communication

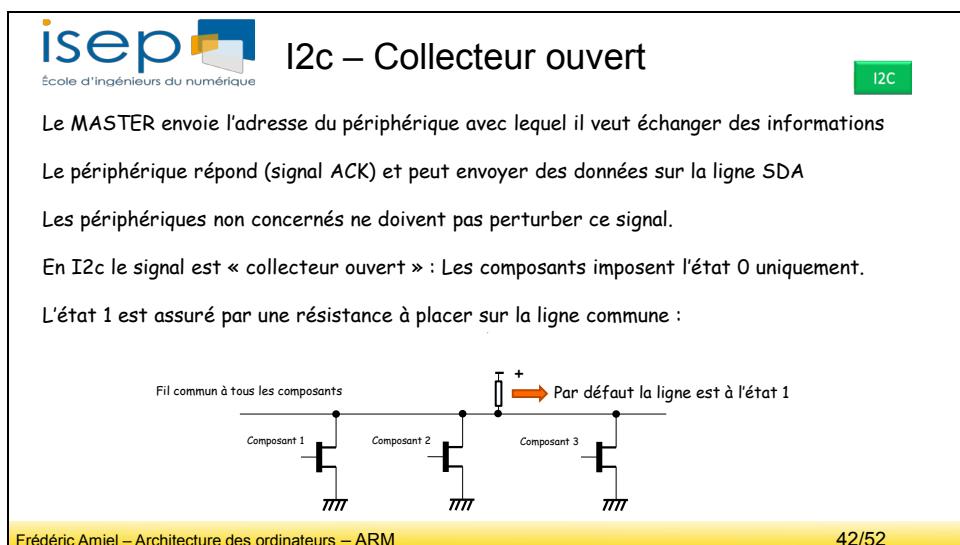
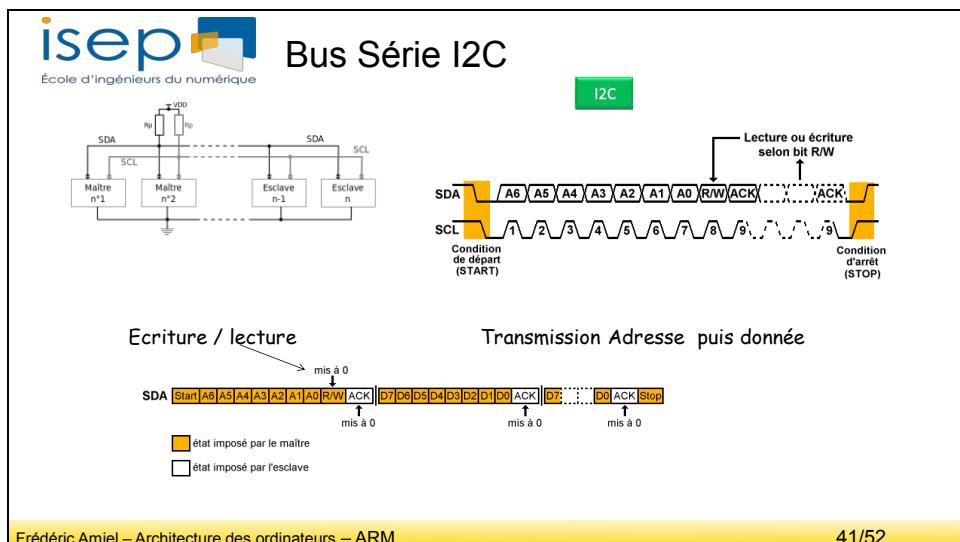
UART

- Les périphériques qui gèrent des transmissions sont parfois bufferisés.

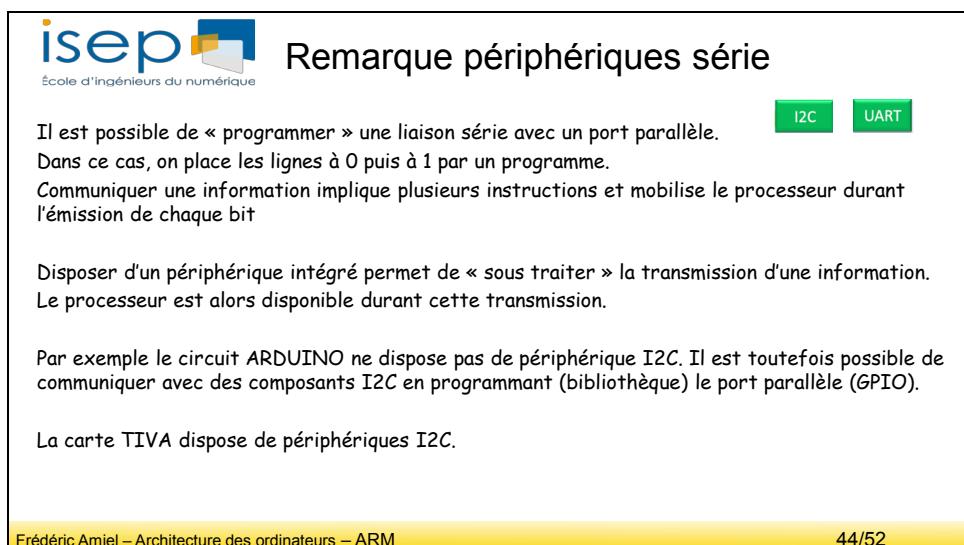
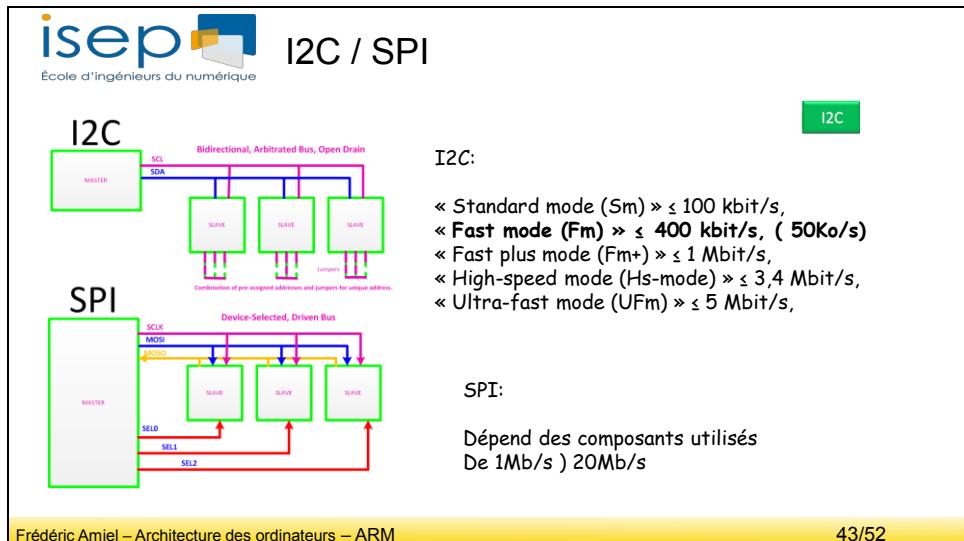
De cette façon, l'application peut être moins réactive.  
Il est plus facile de garantir une bonne réception / transmission des données.

Frédéric Amiel – Architecture des ordinateurs – ARM      40/52

Les fifos intégrées parfois dans l'UART permettent d'effectuer les échanges par blocs de données.



La technique du collecteur ouvert permet de connecter plusieurs sorties logiques ensembles. De cette façon chaque composant peut imposer un 0. Si un autre périphérique place la ligne à 1 il n'y a pas de court circuit (Le courant est limité par la résistance – en général de qq KΩ).



« Emuler » un port série avec un port parallèle est une solution simple qui convient si on ne gère pas plusieurs communications (plusieurs liaisons différentes) simultanément.

## Gestion d'un événement

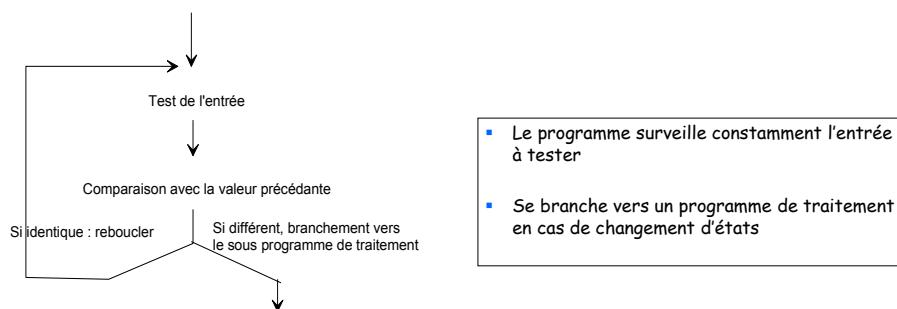
- Par Pooling
- Par Interruption

Événement extérieur : C'est le changement d'état d'une entrée

- Basculement d'un interrupteur
- Tick horloge
- Avancée d'une souris
- Pression d'une touche au clavier
- Fin de transmission d'un message sur une ligne de communication
- Réception d'un message ..

La gestion des entrées / sorties fait apparaître un nouveau principe : les interruptions.

## Le pooling :



Le pooling reste la technique la plus simple.

## Pooling

**Avantages :**

- Facile à programmer

**Inconvénients :**

- Le temps de réaction peut être très long si on surveille beaucoup d'événements
- Le processeur est utilisé pour tester un événement survient peu souvent
- Certaines entrées demandent une fréquence de scrutation élevée, d'autre, une fréquence de scrutation faible : la programmation de la surveillance de différents événements est compliquée.
- L'ajout d'un nouvel événement implique de modifier le programme principal.

Mais il est difficile de programmer un pooling conjoint sur plusieurs périphériques par exemple.

## Pooling - conclusion :

**A utiliser dans un système simple :**

- Peu d'entrées à surveiller
- Processeur n'ayant rien d'autre à faire

**A proscrire dans les systèmes multitâches**

- où le processeur est une ressource importante
- Lorsqu'il y a beaucoup d'entrées de même nature à surveiller si le temps de traitement est critique
- Dès qu'il y a des entrées de nature différente à surveiller

Le pooling utilise le processeur même lorsque qu'il n'y a pas d'événement à traiter. Ce n'est pas une méthode « basse consommation ».

**isep**  
École d'ingénieurs du numérique

## Interruption

```

for(i=1;i<numvertex;i++)
{
    InitSegmentCouraud(*curpt,*nexpt);
    curpt++;
    nexpt++;
}
nexpt = vertexlist;
InitSegmentCouraud(*curpt,*nexpt);
if(miny>0) miny=0;
Arrêt du
programme
en cours
    ...
Void interrupt Ges_Clav()
    ...
}
if(maxy>Ymax) maxy=Ymax-1;
for(i=miny;i<=maxy;i++)
{
    if(endx[i]==INFINI)
        endx[i]=startx[i];
    HlineCouraud(startx[i],startcol[i],endx[i],endcol[i]);
}
    
```

Priorité ? Adresse des routines d'interruption ?

Frédéric Amiel – Architecture des ordinateurs – ARM      49/52

Les processeurs gèrent des broches particulières qui permettent de gérer les entrées / sorties de façon « asynchrone ».

**isep**  
École d'ingénieurs du numérique

## Interruption processeur ARM

Adresse	Interruption	Signification	Offset
0	Reset	Démarrage du processeur	-
4	Instruction SVC	Cette instruction permet de générer une IT logicielle	0
8	Lecture instruction impossible	Pas de mémoire à l'endroit du code	0
12 (0x0C)	Lecture donnée impossible	Pas de mémoire à l'endroit d'une donnée	8
16 (0x10)	Réserve	-	
20 (0x14)	IRQ (Interrupt Request)	Interruption matérielle	4
24 (0x18)	FIQ (Fast Interrupt Request)	Interruption matérielle rapide	4

Lors de l'arrivée d'une interruption, le registre PC (R15) prend la valeur associée.  
Il est essentiel d'avoir de la mémoire à cet endroit. Le contenu de chaque adresse est alors un branchement vers le programme d'interruption associé (B)

Frédéric Amiel – Architecture des ordinateurs – ARM      50/52

Il existe différents types d'interruptions.

**isep**  
École d'ingénieurs du numérique

## Priorité / masque

Le signal RESET permet l'initialisation du processeur

Les interruptions IRQ et FIQ peuvent être masquées

Priorité	Interruption
Plus haute	Reset
	+ Data abort +
	IRQ
	IRQ
	- Prefetch abort -
Plus basse	Interruption logique et instruction indéfinie (ne peuvent pas survenir en même temps... pourquoi?)

Mode : USER - FIQ - IRQ  
SVC - ABORT UNDEF  
SYSTEM

Frédéric Amiel – Architecture des ordinateurs – ARM

51/52

**isep**  
École d'ingénieurs du numérique

## Gestion des ITs

Un contrôleur indépendant permet de gérer la priorité des différentes interruptions générées par les périphériques.

NVIC : Nested Vectored Interrupt Controller

Le microcontrôleur Cortex M4 peut ainsi gérer jusqu'à 240 sources d'interruptions

Frédéric Amiel – Architecture des ordinateurs – ARM

52/52

**isep**  
École d'ingénieurs du numérique

## Instructions spéciales / Vecteur IT

Chaque interruption est identifiée par un numéro : le Vecteur d'interruption.

La table des vecteurs d'IT est située à partir de l'adresse 0 pour le processeur ARM

Chaque « vecteur » occupe 4 octets

Exemple de table d'interruption

```
COMMON INTVEC:CODE      ; Déclaration d'un segment de code
B      main             ; Code au RESET
nop
nop
nop
nop
B      irq              ; Code lors d'un IRQ
```

Frédéric Amiel – Architecture des ordinateurs – ARM

53/52

**isep**  
École d'ingénieurs du numérique

## Mode du processeur

CPSR (poids faible)

Les 5 bits de poids faible du registre indiquent le mode d'exécution en cours

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8_fiq	R8	R8	R8	R8
R9	R9_fiq	R9	R9	R9	R9
R10	R10_fiq	R10	R10	R10	R10
R11	R11_fiq	R11	R11	R11	R11
R12	R12_fiq	R12	R12	R12	R12
R13	R13_fiq	R13_svc	R13_abt	R13_irq	R13_und
R14	R14_fiq	R14_svc	R14_abt	R14_irq	R14_und
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)

M[4:0]      Mode

%10000	User
%10001	FIQ
%10010	IRQ
%10011	SVC (supervisor)
%10111	Abort
%11011	Undefined
%11111	System (ARMv4+)

Lors de l'IT le processeur change de mode.  
Certains registres sont spécifiques au mode courant.

L'instruction MSR permet d'écrire dans le SR :

```
msr cpsr,r1
```

Frédéric Amiel – Architecture des ordinateurs – ARM

54/52

**isep**  
École d'ingénieurs du numérique

## Retour d'interruption

Lorsque l'interruption est activée le registre CPSR change automatiquement de valeur.

- Le masque d'IT est levé
- Le mode est actualisé

A la fin du programme d'interruption est terminé il faut :

- Remettre le masque d'IT tel qu'il était
- Changer le mode
- Actualiser le PC pour revenir

Frédéric Amiel – Architecture des ordinateurs – ARM

55/52

**isep**  
École d'ingénieurs du numérique

## Next

- Bus operations
- Interrupts
- DMA operations
- Cache memory
- Virtual memory
- Pipeline / Multiple exécution unit
- Multicore
- GPU / tensor operation
- Real time systems

Frédéric Amiel – Architecture des ordinateurs – ARM

56/52

## Auto test 1

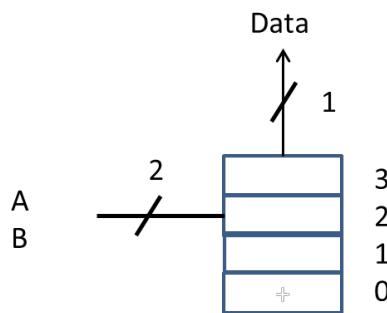
- On réalise une carte de transmission de données à 20Mbits/s. De quel type de périphérique s'agit-il ?
- On souhaite commander un ascenseur, quel type d'interface faut-il ? Justifiez
- Quel est l'avantage à utiliser un TIMER par rapport à une boucle de programme pour faire une temporisation ? (au moins 2 arguments).
- On transmet des données à 9600 bauds. Combien de temps faut-il pour transférer 1 Mo ?
- Quelle est la durée de transmission d'un bit avec ces paramètres ?

## Auto test 2

- Comment sélectionne-t-on un périphérique spécifique avec l'interface i2c ?
- Quel est la différence entre I2C / SPI et UART ?
- Qu'est ce qu'une interruption ?

**Architecture des ordinateurs - TD 1**

A] On souhaite utiliser une mémoire morte pour réaliser une fonction logique avec le câblage suivant :



Q1) Quel est le contenu des cases 0, 1, 2 et 3 pour réaliser

- une fonction logique AND
- une fonction logique EOR (OU exclusif)

On souhaite maintenant réaliser un additionneur capable de traiter deux entrées chacune codées sur 2 bits.

Q2) Quelle est la plage de variation des chiffres à l'entrée ?

Q3) Combien de bits faut-il pour exprimer le résultat ?

Q4) Proposer un schéma avec une mémoire morte permettant de réaliser cette fonction

Q5) quel est le contenu de cette mémoire morte ?

On ajoute un bit supplémentaire pour gérer le fait qu'il y ait une retenue en entrée.

Q6) Quelle modification faut-il apporter à la mémoire ?

Q7) Proposer une solution permettant de réaliser un additionneur acceptant des chiffres codés sur 4 bits en entrée.

Q8) Quelles sont les plages de variation des chiffres dans ce cas.

B] On utilise un processeur raccordé à une mémoire par un bus d'adresse de 24 bits et un bus de donnée de 16 bits.

Q9) Quelle est la taille de l'espace d'adressage en Ko ou Mo ?

Le processeur est cadencé par une horloge à 100 MHz. Il lui faut 2 coups d'horloge pour cadencer un échange bus.

Q10) Quelle est le débit du bus en Ko/sec ou Mo/sec ?

On souhaite placer une mémoire RAM de 1 Mocets sur ce bus et une mémoire morte de 512 Ko.

Q11) Proposer une carte mémoire de ce système.

C] On dispose d'une mémoire contenant deux chiffres dans les cases 0 et 1.

Q12 Décrire les opérations bus permettant d'additionner les deux chiffres et de placer le résultat dans la case 2.

**Architecture des ordinateurs TD 2 :**

I) Convertir les nombres suivants de binaire pur vers décimal :

00010010  
10101111  
11110000  
00001111  
10101100

II) On considère maintenant que ces nombres sont codés en complément à 2 sur 8 bits. Quelle est leur valeur en décimal ?

III) Indiquer la représentation hexadécimale de ces nombres.

IV) Représenter les nombres suivants de base 10 vers la base 16 puis en binaire :

14	38	364
-13	-5	-200

V) Sans faire les calculs, indiquer l'état des indicateurs N Z V et C pour les opérations suivantes (réalisées avec une alu 8 bits).

12 + 5	0x70 + 0x20
12 - 5	0x90 - 0x20
4 - 6	130 - 4
120 + 32	200 + 70

VI) On dispose des nombres suivants représentés en hexadécimal et codant des nombres en complément à 2 sur 8 bits. On souhaite obtenir leur représentation en complément à 2 sur 16 bits puis sur 32 bits. Indiquer de plus leur valeur en base 10.

0x12  
0x90  
0xFF

VII) Coder les chiffres suivants en virgule flottante : 5,75 / -11,7 / 0,3

VIII) Expliquer pourquoi l'addition de 2 nombres en virgule flottante requière plus de transistors qu'une addition de nombres entiers.

**IE 1102 - TD3 :**

Tous les registres du processeur contiennent le chiffre 0 au départ.

Q1) Que contient R0 après :

a) mov r0,#0x1f	f) ldr mov	r0,=0x1234 r0,r0 lsl 2
b) mov r1,#0x4 add r0,r0,r1	g) ldr ldr	r0,=0x12345678 r1,=0x87654321
c) mov r0,#0x5a mov r1,#0x0f eor r0,r0,r1	and sub	r0,r0,r1 r0,r0,r0
d) mov r1,#20 add r0,r1,#10	h) ldr mov	r0,=0x12345678 r0,r0 ror #4
e) mov r0,#0x12 sub r0,r0,#12 add r0,r0,r0	i) mov add	r0,#10 r0,r0,r0 lsr #1

Q2) Ecrire un programme simple pour réaliser :

- a) R8 = 0x3
- b) R7 = 10 \* 5
- c) R6 = r9 – 6
- d) R5 = 6 – r9

Q3) Expliquer le rôle du registre r15

Où est le registre r4

## Travaux Dirigés 4

Q1) On souhaite additionner 2 valeurs 64 bits. La première valeur est placée dans R0/R1  
La deuxième valeur dans R2/R3.

Q2) Faire un programme permettant de lire 10 cases mémoires, et de multiplier leur valeur par 2.

Proposer 2 versions :

- Sans boucle
- Avec une boucle

Q3) Expliquer le rôle et coder en binaire les instructions :

ADD r0,r1,r2	MOV R0, #5
ADDTGT r0,r1,r2	mov r0,#5 ror 4
ADDLTS r0,r1,r2	B +1000

Q4) Ecrire en assembleur un programme de transcodage d'un nombre compris entre 0 et 15 en un caractère ascii en utilisant une TABLE de correspondance.

## Travaux Dirigés 5

On souhaite faire défiler une led allumée sur un port parallèle.

Q1) Définir la configuration du registre de direction.

Q2) Ecrire le programme principal permettant le défilement sur 8 bits d'une lumière allumée.

Q4) Programmer une routine de temporisation d'une seconde exactement en considérant qu'on utilise une horloge à 50 MHz.

Q5) Représenter la transmission du caractère 'A' en format 7 bits parité paire.

Q6) On dispose d'un QUARTZ de 32,768 MHz en horloge d'un TIMER.

On souhaite générer un événement toutes les millisecondes. En utilisant les registres du cours : pré diviseur 8 bits et registre de comptage 16 bits. Proposer une configuration.

Q7) On souhaite dialoguer avec un port série avec les éléments suivants :

1. Un convertisseur AD 8 bits à  $10^6$  échantillons par seconde.
2. Trois afficheurs à cristaux liquides de 16 caractères chacun
3. Un GPS

Proposer un type de port série pour ces 3 cas et expliquer votre critère de choix (les avantages de votre solution).

## ARM – TP1

Le but de ce TP est de prendre en main l'environnement de développement ARM. De programmer quelques instructions arithmétiques et logiques simples, et quelques transferts mémoire. Créer un fichier PDF en utilisant le template fourni sous Moodle et déposez le dans la boîte de dépôt correspondante à votre groupe.

### ***Préparation :***

Q1) Coder les chiffres suivants de décimal vers hexadécimal :

100                  10000

Q2) Coder les chiffres suivants de hexadécimal vers décimal :

0x1234              0x12345678

Q3) Expliquer le fonctionnement du programme suivant :

```

B      main
main    NOP
      B main
  
```

### ***A) Manipulation – partie 1 – Environnement et représentation***

Dans cette section, vous allez apprendre à utiliser l'environnement pour créer un projet, éditer un programme, l'exécuter instruction après instruction, visualiser le contenu des registres et de la mémoire, vérifier le fonctionnement des drapeaux N Z et C.

Utiliser le navigateur pour aller sous :

**D:\TP\_A1**

Créer un répertoire avec votre nom de binôme.

**D:\TP\_A1\Votre\_Nom**

Sous ce répertoire, créer un autre répertoire lab1 :

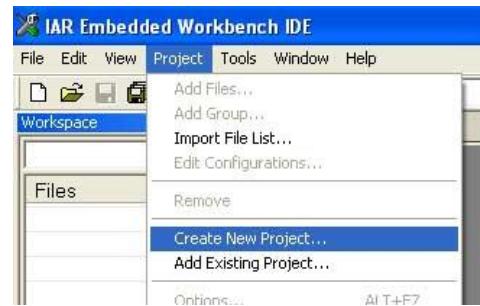
**D:\TP\_A1\Votre\_Nom\lab1**

C'est dans ce répertoire que vous sauvegarderez votre travail concernant le TP1. Si vous souhaitez copier votre travail sur une clé de sauvegarde, vous devez copier le répertoire lab1.

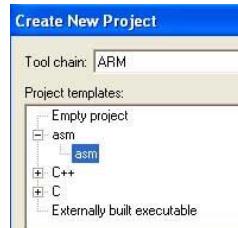
Lancer l'environnement de développement en cliquant sur l'icône :



Faire : **Project -> create New Project**



Puis créer un projet de type asm :



Enregistrer sous le dossier : **d:\TP\_A1\Votre\_Nom\lab1\** - vous pouvez utiliser TP1 comme nom de fichier.

**Afin de sauvegarder l'environnement cliquez sur l'icône :**



Utilisez **TP1** comme nom de fichier.

Cette sauvegarde vous permettra ultérieurement de retrouver votre travail et votre environnement.

Vous disposez alors d'un programme de base au milieu duquel vous allez insérer votre code :

```

NAME main
PUBLIC main
COMMON INTVEC:CODE
CODE32
B main
RSEG ICODE:CODE
CODE32

main    NOP
B      main
END    main
  
```

Insérer le programme suivant dans le code proposé, sous l'instruction NOP.

```

ldr      r0,=100
ldr      r1,=10000
Add     r2,r1,r0,r4
  
```

**Project → make**

L'erreur de syntaxe est repérée par une croix rouge.

#### Q4) Expliquer et corriger l'erreur :

La ligne devient : Add r2,r1,r0

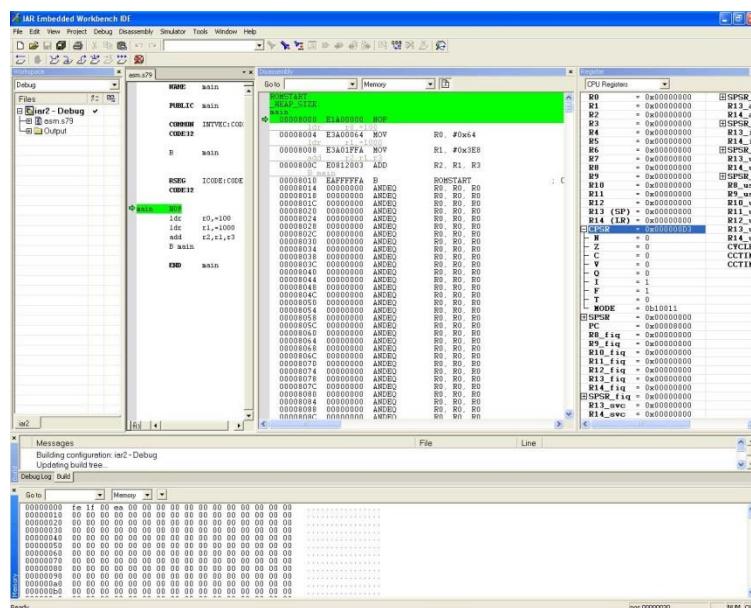
Assembler ce programme, et lancer le debugger : **Project → Debug** (raccourci : ^D)

Faites apparaître les registres et la mémoire par :

**View → Registers** et **View→Memory**

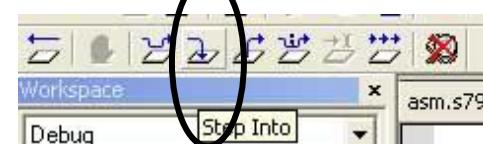
Développer le registre CPSR en cliquant sur le +

Redimensionner les fenêtres et passer en plein écran de façon à voir à peu près :



Identifier les registres, et les drapeaux de l'ALU.

Exécuter le programme en pas à pas :



(Raccourci : F11)

Constater l'exécution correcte en regardant l'évolution des registres.

#### Q5) Comment instructions suivantes ont-elles été transformées ?

Ldr r0,=100 et ldr r1,=10000 ?

**Q6) Rappeler dans quelles conditions on peut utiliser l'instruction MOV pour affecter une valeur à un registre.**

Ajouter alors les lignes suivantes après l'instruction add de votre programme :

```
ldr r10,=0x9000
str r2,[r10]
```

Vérifier l'écriture en mémoire 0x9000 en utilisant la fenêtre memory. (Go to 0x9000)

**Q7) Expliquer le codage du chiffre 0x2774 en mémoire 0x9000.**

Modifier votre code de départ.

Pour simplifier l'écriture, écrivez le programme en utilisant la pseudo-instruction EQU, qui permet de définir des symboles.

Avant la ligne « main nop » ajouter les deux lignes suivantes :

```
NB1 equ 10
NB2 equ 10000
```

Dans le programme, modifier les deux lignes du programme précédent de façon à obtenir :

```
ldr r0,NB1
ldr r1,NB2
```

On rappelle que les instructions ne lèvent les drapeaux qu'avec le suffixe S (adds).

**Q8) Proposer 2 valeurs à placer dans r0 et r1 pour que l'addition lève le drapeau Z .****Q9) Proposer 2 valeurs permettant de lever V.****Q10) Proposer 2 valeurs permettant de lever N.**

## ARM – TP2

Lors de ce TP vous devez :

- comprendre la différence entre le processeur et la mémoire au travers des instructions load et store (LDR et STR).
- Comprendre la programmation des boucles
- Comprendre le travail d'un compilateur C

### A) Manipulation – partie 1 – Environnement et représentation

Replacez-vous sous votre répertoire de travail

**D:\TP\_A1**

Puis aller sur le répertoire de votre nom de binôme s'il existe. Vous devez le créer si ce n'est pas le cas.

**D:\TP\_A1\Votre\_Nom**

Sous ce répertoire, créer un autre répertoire lab2 :

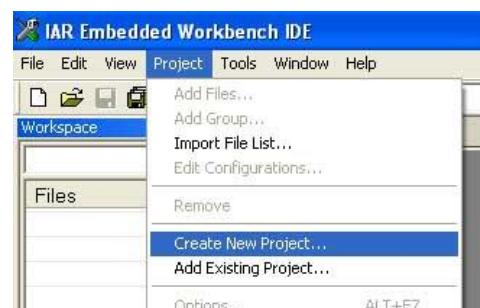
**D:\TP\_A1\Votre\_Nom\lab2**

C'est dans ce répertoire que vous sauvegarderez votre travail concernant le TP2. Si vous souhaitez copier votre travail sur une clé de sauvegarde, vous devez copier le répertoire lab2.

Lancer l'environnement de développement en cliquant sur l'icône :



Faire : **Project -> create New Project**



Puis créer un projet de type asm :



Enregistrer sous : **d:\TP\_A1\Votre\_Nom\lab2** - vous pouvez utiliser TP2 comme nom de fichier.

Afin de sauvegarder l'environnement cliquez également sur l'icône :



Remplacer tout le code existant par le code suivant :

```

CODE32

NB      equ    10
main
        ldr    r10,=table
        mov    r2,#NB
bcl
        ldrb   r0,[r10]
        mov    r1,r0,lsl #1
        add    r10,r10,#1
        subs   r2,r2,#1
        bne   bcl
ici
        nop
        b      ici
table
        dc8  0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39
END    main

```

### **Project → make**

Permet d'assembler votre programme (raccourci : F7)

Ce programme doit s'assembler sans erreur. Vérifier les indentations sur les lignes qui seraient invalides. (Les étiquettes de programmes doivent se trouver en première colonne).

Lancer le debugger : **Project → Debug** (raccourci : ^D)

Votre programme est alors placé dans la mémoire. Vous devez identifier l'adresse mémoire contenant les instructions et les données (codes instructions) qui s'y trouvent.

**Q1) Quelle est l'adresse de début de votre programme ? Combien d'instructions au total comporte ce programme ?**

**Q2) Quelle est le code de l'instruction :**      **MOV**      **R10,0x24**

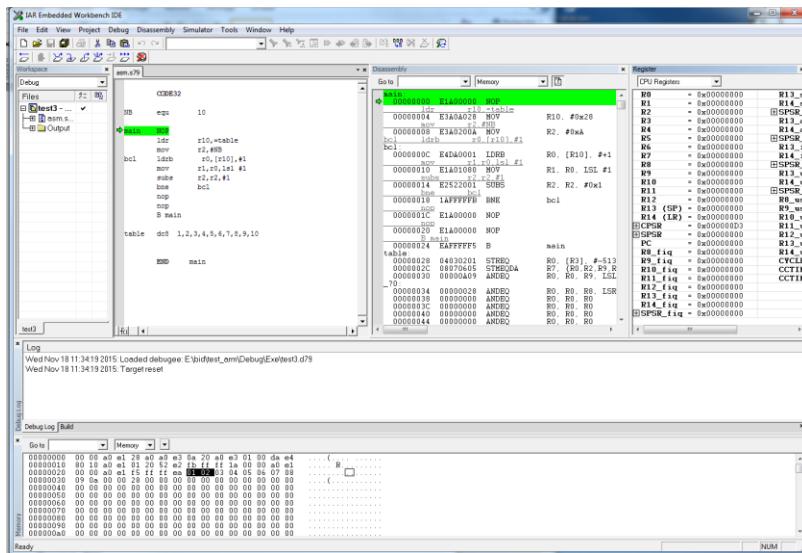
**Q3) Quelle est l'adresse de l'instruction :**      **MOV**      **R2,#0xA**

**Q4) combien faut-il d'octets pour coder l'instruction MOV R2,#0A**

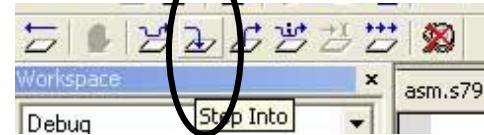
Si votre environnement n'en dispose pas déjà, faites apparaître les registres et la mémoire.

**View → Registers** et **View→Memory**

Redimensionner les fenêtres et passer en plein écran de façon à voir à peu près :



Exécuter le programme en pas à pas :



(Raccourci : F11)

jusqu'à ce que vous arriviez sur les instructions nop.

**Q5) Quel est le registre qui compte (ou décompte) le nombre de boucles exécutées ?**

**Q6) Quelle est la signification et le rôle de l'instruction BNE ?**

**Q7) Quelle est la ligne qui spécifie le nombre de boucles ?**

Relancer le chargement du programme



Les données [0x30,0x31,0x32 etc...] sont placées en mémoire et référencées par l'étiquette table. Repérer ces données dans la fenêtre de visualisation mémoire.

		Memory															
Go to		00 00 a0 e1 24 a0 a0 e3 0a 20 a0 e3 00 00 da e5	...	\$	.....												
00000000	00 00 a0 e1 24 a0 a0 e3 0a 20 a0 e3 00 00 da e5	.....\$.....															
00000010	80 10 a0 e1 01 a0 8a e2 01 20 52 e2 fa ff 1a	.....R.....															
00000020	f6 ff ff ea 30 31 32 33 34 35 36 37 38 39 00 00	.....0123456789..															
00000030	24 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	\$.....															
00000040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....															
00000050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....															
00000060	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....															
00000070	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....															
00000080	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....															
00000090	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....															
000000a0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....															
000000b0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....															

**Q8) A partir de quelle adresse repérés par l'étiquette table les valeurs 0x30,0x31,0x32, etc sont-elles stockées ?**

Exécuter le programme en pas à pas.

**Q9) Que contient le registre R0 puis R1 lors de chaque boucle et pourquoi ?**

**Q10) Que contient le registre R10 ?**

On va maintenant écrire le résultat en mémoire à la place des données de départ. Pour écrire en mémoire, il faut utiliser l'instruction STRB (store Byte).

**Q11) A quel endroit doit-on placer cette instruction STRB et quelle est sa syntaxe ? Ecrire et placer l'instruction STRB avec les opérandes adéquats.**

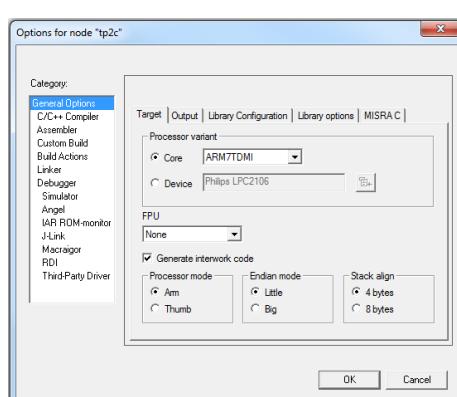
On va maintenant écrire ce programme en langage C.

Fermer le projet (File->Close Workspace)

Puis ouvrir un nouveau projet :

Project→create new project →C→main

Créer un répertoire de travail lab2c sous le répertoire à votre nom de façon à sauver votre projet sous le nom lab2c.c



Entrer le code suivant :

```

char Table[] = {0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,0x38,0x39};

int main()
{
    int i;
    char a,b;

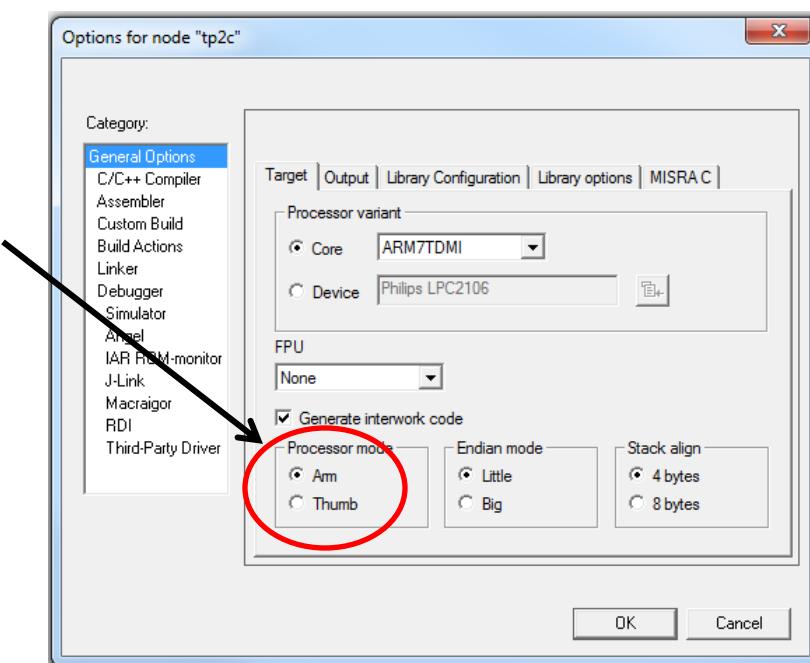
    for (i=0;i<10;i++)
    {
        a = Table[i];
        b = 2 * a;
        Table[i] = b;
    }
    return 0;
}

```

Par défaut le système génère un code THUMB (16 bits) il faut configurer le compilateur pour générer du code ARM (32 bits).

Project → Options... → General Option

Et activer la case **Arm** :



Sauvegarder votre programme sous le nom lab2c :



Compiler le programme.

Lancer le debugger et configurer les fenêtres de façon à visualiser les registres et la mémoire.

Exécuter le programme pas à pas.

**Q12) Quels registres sont utilisés pour les variables : i, a, b ?**

Dans la fenêtre Memory, utiliser le **Go to 0x...** pour visualiser les données du tableau « Table »

Exécuter le programme en pas à pas et vérifier le fonctionnement.

Le pas à pas s'exécute instruction par instruction C en cliquant dans la fenêtre du programme C et en assembleur en cliquant dans la fenêtre affichant le code désassemblé.

**Q13) Combien d'instructions C sont exécutées pour une itération de la boucle for ?**

**Q14) Combien d'instructions assembleur après compilation du programme C sont exécutées pour une itération de la boucle for**

**Q15) Quel est le nombre total d'instructions assembleur après compilation ? (Etiquette \_main et étiquette \_exit). Quel est le rapport entre ce nombre d'instructions et celui trouvé à la question 1 ? (Nombre d'instructions d'un programme écrit directement en assembleur)**

## ARM – TP3

Lors de ce TP vous allez :

- Utiliser une carte d'évaluation et non le simulateur
- Programmer un port parallèle en assembleur

### A) Préparation :

**Q1) On souhaite programmer le port parallèle 0 du str7 en sortie sur ses16 bits. Quel est le contenu de PC0 – PC1 et PC2 ?**

**Q2) Quelle est l'instruction d'appel de sous-programme ? Quelle est la différence entre cette instruction et l'instruction B ? Rappeler le rôle des registres PC et LR.**

**Q3) Indiquer le temps d'exécution des instructions suivantes :**

```
sub    r0,r0,#1           et
bne   xxx
```

**Q4) Avec un cycle horloge de 120 ns (fréquence à 8 MHz) combien de fois faut-il exécuter ces instructions pour avoir un temps d'environ 0,5 secondes ?**

### B) Manipulation

Créer un nouveau répertoire Lab3 :

**D :\TP\_A1\Votre\_Nom\ lab3\**

Créer un nouveau projet de type **asm** sous IAR Embedded Workbench.

(Project→Create New Project)

Enregistrer ce nouveau projet sous le nom **TP3** dans le répertoire lab3

Remplacer **TOUT** le code existant par les lignes suivantes :

```
; ----- Déclaration des symboles -----
org 0xa0000000
CODE32
PORT0 equ 0xffffd400
pc0   equ 0
pc1   equ 4
pc2   equ 8
pdat  equ 12
```

```

; ----- Initialisations -----
main
    bl      init_out           ; Appel du sous-programme d'initialisation des ports d'E/S

    ldr    r0,=0x00FF
    ldr    r1,=0xFF00
bclmain
    strh   r0,[r10,#pdat]
    strh   r1,[r10,#pdat]
    b      bclmain

; Sous-programme initialisation des entrées / sorties en SORTIE

init_out
    ldr    r10,=PORT0
    ldr    r0,=0
    ldr    r1,=0xffff
    strh   r1,[r10,#pc0]
    strh   r0,[r10,#pc1]
    strh   r1,[r10,#pc2]
    mov    pc,lr                ; Fin de sous-programme

end    main

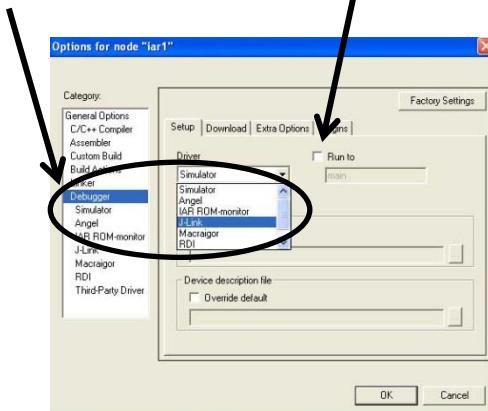
```

L'instruction ORG permet de définir une adresse d'implantation pour le code. Dans ce cas, l'adresse 0xa0000000 référence une zone de 16 Ko de mémoire vive (RAM). C'est donc là que sera placé votre programme.

Il faut maintenant configurer le système pour que votre programme soit chargé sur la carte et non sur un simulateur. Pour cela faire :

Project → Options → Debugger

Sélectionner le driver « J-Link et décocher la case : « Run to » (main) :



Le driver J-Link correspond à la sonde USB connectée à votre PC. Désormais, votre programme sera chargé et exécuté par la carte de développement.

Sauvegarder votre projet.



Assembler et lancer le Debug de votre programme. Exécuter en pas à pas.

**Q5) Que fait ce programme ? Expliquer sommairement le fonctionnement. Pourquoi utiliser des instructions strh plutôt que str (vérifier que le programme fonctionne en remplaçant une instruction strh par une instruction str).**

Lancer alors le programme en RUN :



Ou :

Debug→Go (F5)

**Q6) Expliquer le comportement des Leds.**

Vous allez maintenant écrire un sous-programme de temporisation élémentaire avant la ligne :

« end main »

```
NBR      equ    3
tempo
bbb      ldr    r5,=NBR
         sub    r5,r5,#1
         bne   bbb
         mov    pc,lr
```

Appeler ce sous-programme en insérant l'instruction d'appel dans le code précédent.  
En exécutant ce programme en mode pas à pas, on constate **qu'il ne marche pas.**

**Q7) Indiquer et corriger l'erreur.**

Enfin, en lançant le programme en mode RUN, il n'a pas d'effet sur le comportement des Leds.

**Q8) Quel valeur faut-il associer à NBR pour voir les Leds clignoter ?**

On souhaite modifier maintenant le programme pour voir un chenillard sur les Leds (Une led allumée qui semble se déplacer de la droite vers la gauche cf TP1).

**Q9) Quelle est l'instruction permettant de décaler ?**

Ecrire le programme permettant de décaler le motif 000000000000000X vers la gauche.

**Q10) Ecrire et tester le programme correspondant, opérant un défilement continu de la led allumée (elle apparaît à nouveau à droite lorsqu'elle disparait à gauche).**

## ARM – TP4

Lors de ce TP vous allez :

- Utiliser une carte d'évaluation et non le simulateur
- Programmer un port parallèle en langage C et en assembleur
- Manier le port parallèle en langage C
- Ecrire des fonctions assembleur appelées dans un programme écrit en langage C

### A) Préparation :

**Q1) Quelle instruction assembleur permet de faire une rotation dans le registre R0 ?**

**Q2) Rappeler l'adresse et la taille de la mémoire RAM dans la carte d'évaluation STR730 utilisée en TP**

**Q3) Quelle est la fonction du « linker » (édition de lien) dans le codage d'un programme ?**

En langage C le ET logique s'effectue par l'opérateur noté « & » le OU logique par l'opérateur noté « | ».

Le décalage vers la gauche s'effectue par l'opérateur noté « << » et le décalage vers la droite par l'opérateur « >> ». Il n'y a pas d'opérateur permettant d'effectuer une rotation (le bit qui sort d'un côté rentre de l'autre côté).

**Q4) Quel est l'effet de la ligne C suivante sur la variable val :**

val = val & 1 ;

**Q5) Quel est l'effet de la ligne C suivante sur la variable val :**

val = val | 1 ;

### B) Manipulation

Créer un répertoire lab4 dans le dossier à votre nom.

Sur MOODLE, sous la rubrique « cours architecture A1 » puis sous le répertoire LAB4. Charger les 4 fichiers :

main4.c	used_function.h
used_function.c	lnkarm_ram.xcl

Créer alors un nouveau projet vide de nom TP4.

Sélectionner ce projet TP4 dans la fenêtre « Workspace »

Utiliser le menu

**Project → add Files et ajouter les fichiers main4.c – used\_fonction.c.**

Utiliser le menu

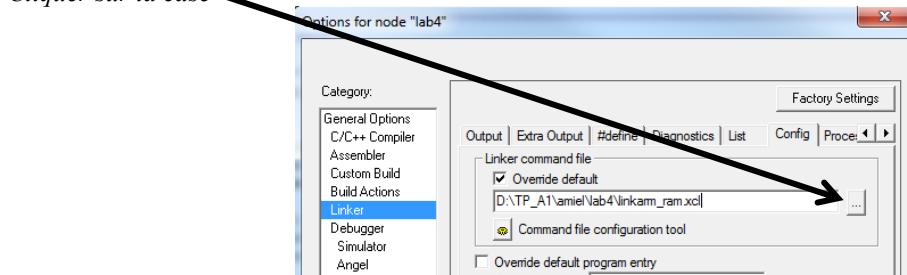
**View Registers**

Pour faire apparaître les registres et dimensionner les fenêtres pour avoir une vue compréhensible des informations affichées.

Utiliser le menu

**Project → Options → linker et aller dans l'onglet Config. Cliquer sur « Override default »**

*Cliquer sur la case*



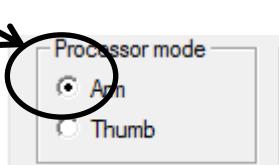
*et sélectionner votre fichier lnkarm\_ram.xcl*

Configurer le débugger sur JLINK (voir TP3)

**Project → Options → Debugger** En laissant activée la case **Run To (main)**

Configurer la compilation pour générer du code ARM (32 bits)

**Project → Options**



Sauvegarder le nouveau projet entier sous le nom tp4 en cliquant sur :



Lancer l'exécution.

**Q6) Que fait ce programme ?**

Ouvrir le fichier lnkarm\_ram.xcl disponible à partir de la fenêtre « Workspace » en déployant la rubrique « output ». Ce fichier est utilisé par le linker lors de la génération du code exécutable.

**Q7) Quelle est la ligne de ce fichier qui indique que le programme sera placé en mémoire RAM ?**

Ouvrir maintenant le fichier « main4.c »

**Q8) Expliquer le rôle et le fonctionnement de la fonction « rotate\_right »**

On souhaite maintenant que le motif décalé soit le suivant :

ooooooooooooxxxxoo

(o représente une led éteinte, et x une led allumée).

**Q9) Programmer la modification (Indiquer la valeur initialisée).**

La fonction « rotate\_right » écrite en langage C est relativement complexe (test, opérations logiques...) on se propose d'écrire une fonction assembleur réalisant la même opération de façon plus simple. (Utilisation de l'opérateur ROR).

La fonction « Aller jusqu'au curseur » vous permet d'identifier simplement les parties de code C ou assembleur qui vous intéressent.



**Q10) Lors de l'appel à la fonction rotate\_right dans quel registre du processeur est placé le paramètre (val) ?**

**Q11) Lors du retour de fonction, dans quel registre est placé la valeur renvoyée ?**

Créer un nouveau fichier et écrire les lignes suivantes :

```
public    rotate
CODE32
RSEG   ICODE:CODE

rotate
    mvn r0,r0
    mov pc,lr
end
```

Le mot clé **public** permet de rendre la fonction « visible » des autres fichiers. CODE32 indique à l’assembleur de générer du code ARM et RSEG indique qu’il faut placer les instructions dans la partie CODE (le fichier de link indique à quelle adresse mémoire se situe la partie CODE).

**Sauvegarder ce fichier sous le nom rotate.asm**

**Q12) Expliquer ce que fait cette fonction.**

Dans la fonction main, vous devez maintenant remplacer l’appel de la fonction rotate\_right par l’appel de la fonction rotate.

**Ajouter le fichier rotate.asm au projet :**

**Project → Add Files** puis sélectionner le fichier rotate.asm (attention au filtre des types de fichier).

**Exécuter et vérifier le bon fonctionnement de la fonction assembleur.**

Vous allez maintenant modifier le code de cette fonction rotate pour implémenter la fonction de **décalage à droite**.

**Q13) Quel est le nouveau code de la fonction rotate permettant de décaler à droite ?**

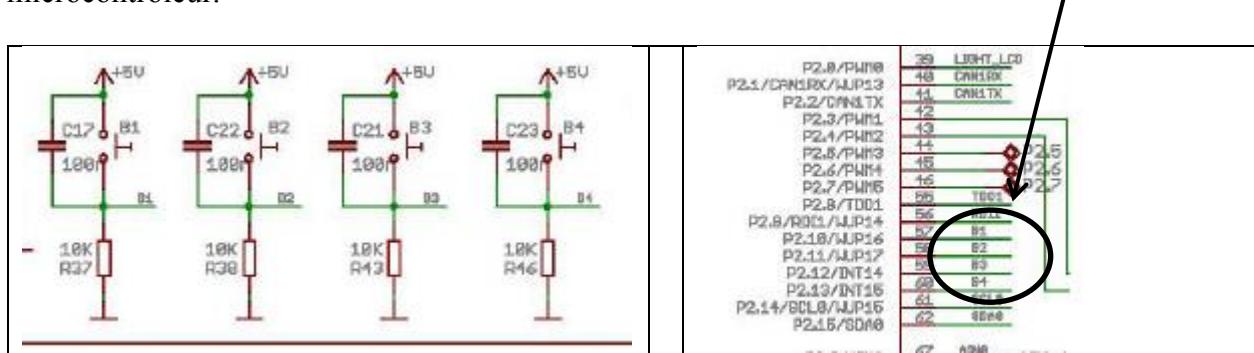
## ARM – TP5

Lors de ce TP vous allez :

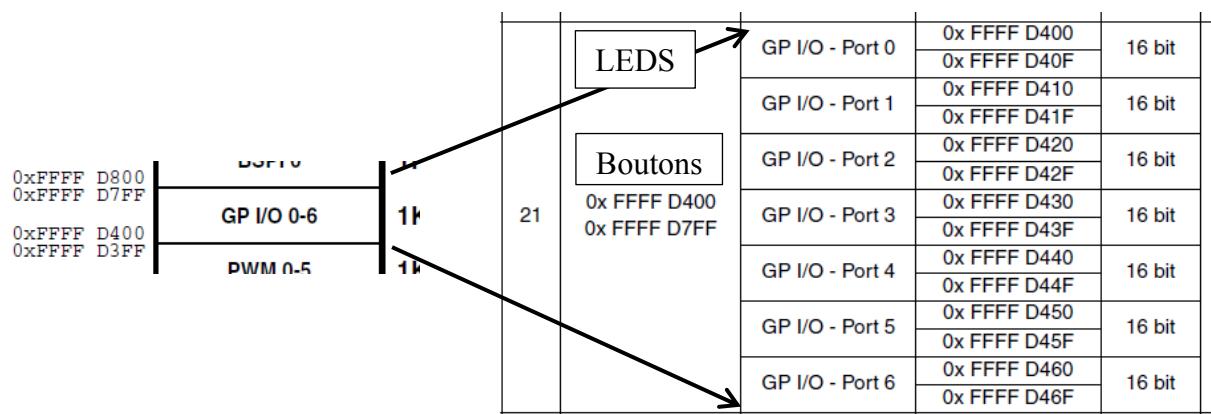
- Utiliser la carte d'évaluation
- Programmer un port parallèle en langage C et en assembleur
- Manier le port parallèle en sortie et en entrée en langage C
- Ecrire des fonctions assembleur appelées dans un programme écrit en langage C
- Lire l'état d'interrupteurs et allumer les LEDs en conséquence

### A) Préparation :

Le schéma de la carte STR730 utilisée lors des travaux pratiques est disponible dans le poly de cours. La figure ci-dessous présente un zoom des interrupteurs et de leur connexion au port du microcontrôleur.



Ces 4 interrupteurs sont donc connectés sur le port 2 sur les bits 10,11,12 et 13.



**Q1) Quelle est l'adresse de base du port 2 ?**

## B) Manipulation

Créer un nouveau répertoire Lab5 :

D :\TP\_A1\Votre\_Nom\ lab5\

Sur MOODLE, sous la rubrique « cours architecture A1 » puis sous le répertoire LAB5. Charger les 4 fichiers :

Main5.c	used_function.h
used_function.c	lnkarm_ram.xcl

Créer alors un nouveau projet vide (empty) de nom TP5.

Sélectionner ce projet TP5 dans la fenêtre « Workspace »

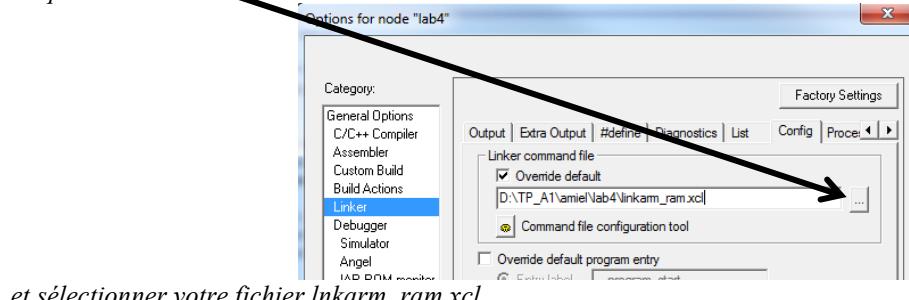
Utiliser le menu

*Project → add Files et ajouter les fichiers main5.c – used\_fonction.c.*

Utiliser le menu

*Project → Options → linker et aller dans l'onglet Config. Cliquer sur « Override default »*

*Cliquer sur la case*



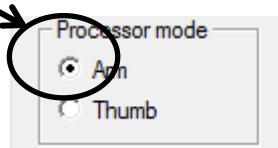
*et sélectionner votre fichier lnkarm\_ram.xcl*

Configurer le débugger sur JLINK (voir TP3)

*Project → Options → Debugger En laissant activée la case Run To (main)*

Configurer la compilation pour générer du code ARM (32 bits)

*Project → Options*



Sauvegarder le nouveau projet entier sous le nom tp5 en cliquant sur :



Lancer le debugger,

Utiliser le menu

#### ***View Registers***

Pour faire apparaître les registres et dimensionner les fenêtres pour avoir une vue compréhensible des informations affichées.

Lancer l'exécution.

Enfin la boucle while devient :

```
while (1)
{
    val = in_port(adr_port2);
    out_port(adr_port0,val); // écriture sur le port de sortie
}
```

Exécuter ce programme.

#### **Q2) Que fait ce programme ?**

On va maintenant écrire une fonction en assembleur acceptant en paramètre d'entrée un numéro d'interrupteur entre 1 et 4 et retournant son état [0 ou 1].

val = button (adr\_port3,num) ;

Modifier la boucle while de la fonction main comme suit :

```
while (1)
{
    val = button(adr_port2,1);
    if (val)
        out_port(adr_port0,0xffff); // Illuminer les leds
    else
        out_port(adr_port0,0x0000); // éteindre les leds
}
```

En début du fichier main5.c il faut déclarer cette nouvelle fonction :

```
#include "used_functions.h"
int button (unsigned int *p,int num);
```

Utiliser alors le menu **File→new** pour créer un nouveau fichier contenant le texte suivant :

```
public    button
CODE32
RSEG   ICODE:CODE

button
nop
nop
mov pc,lr
end
```

Sauvegarder ce fichier sous le nom button.asm et utiliser le menu :

**Project→Add Files..**

Pour ajouter le fichier à votre projet.

**Générer le projet.**

**En utilisant la fonction « aller jusqu'au curseur »**

Lancer l'exécution jusqu'à la fonction button.

**Q3) Quels registres contiennent les paramètres envoyés par la fonction main lors de l'appel de la fonction button ?**

Modifier maintenant la fonction button conformément au code ci-dessous :

```
button
ldr r0,[r0,#12]
cmp r1,#1
moveq r0,r0,lsr #10
ands r0,r0,#1
mov pc,lr
```

Lancer ce code et constater le fonctionnement.

**Q4) quel est la rôle de l'instruction *cmp r1,#1***

**Q5) Quel est l'intérêt du décalage de 10 bits vers la droite (*moveq r0,r0,lsr #10*)**

**Q6) Ajouter des instructions permettant la prise en compte d'une pression sur B2 également**

**Q7) Ajouter des instructions permettant également la prise en compte des boutons B3 et B4. D'autre part, si le numéro de bouton n'est pas compris entre 1 et 4 retourner -1**

**Q8) Modifier alors le programme C de façon à avoir le comportement suivant :**

Pression du bouton 1 : Affichage de ooooooooooooooxxxxx (x = led allumée ; o = led éteinte)

Pression du bouton 2 : Affichage de oooooooooooooxxxoooo (x = led allumée ; o = led éteinte)

Pression du bouton 3 : Affichage de oooxxxxxoooooooooooo (x = led allumée ; o = led éteinte)

Pression du bouton 4 : Affichage de xxxxooooooooooooooo (x = led allumée ; o = led éteinte)

**Architecture des ordinateurs - IE.1102 - A1 - 2021-2022 - Semestre 2**

Sans documents

Sans calculatrice

Pas de points négatifs. Durée 1h30

Documentation nécessaire en fin de sujet.

**A] Nombre et arithmétique :**

Q1) Quel est le codage du chiffre 42 en complément à 2 sur 8 bits représenté en hexa?

- a) 0x2A      b) 0x42      c) 0xD6      d) 0x42

Q2) Quel est le codage du chiffre -21 en complément à 2 sur 8 bits représenté en hexa?

- a) 0xEB      b) 0x21      c) 0x15      d) 0xF1

Q3) Quel est l'état des Flags NZVC après le calcul 145 + 11 sur 8 bits ?

- a) 1001      b) 1000      c) 0001      d) 0010

Q4) Quelle est la représentation décimale du chiffre %11000011 codé en complément à 2 sur 8 bits ?

- a) 33      b) 195      c) -60      d) -61

Q5) Quel chiffre faut-il ajouter à 100 pour faire -100 (codage sur 8 bits complément à 2).

- a) 200      b) 56      c) 128      d) 9

Q6) On veut coder des chiffres entiers entre -10,0 et +9,9 (précision au dixième combien faut-il de bits au minimum pour leur représentation ?

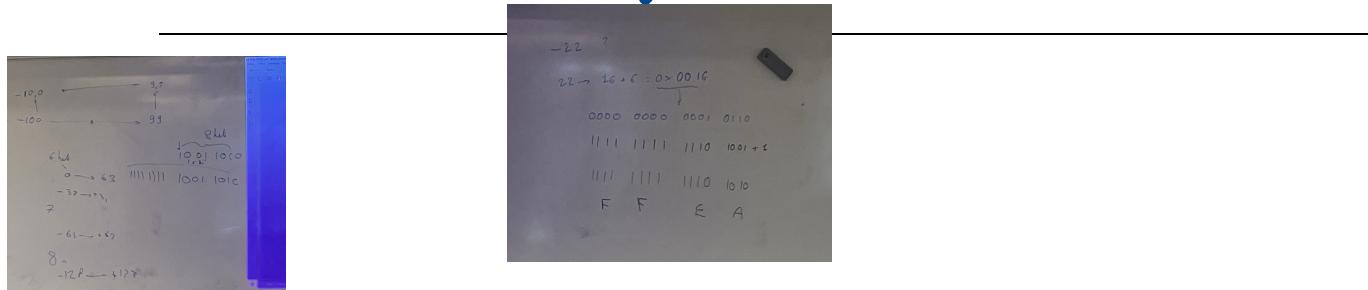
- a) 8      b) 9      c) 10      d) 11

Q7) On lit un convertisseur A/D 12 bits récupérant une température entre -200° et +200° à pleine échelle, quelle est environ la précision de la mesure ?

- a) 1°      b) 0,5°      c) 0,1°      d) 0,05°

Q8) Quel est le codage du chiffre -22 sur 16 bits en complément à 2 ?

- a) 0x0016      b) 0xFF16      c) 0xFFEA      d) 0xFFDE



**B] Architecture / composants**

Q9) Qu'est ce qui peut être aléatoire dans une RAM ?

- a) Le contenu
- b) L'adresse
- c) Le fonctionnement
- d) La donnée

Q10) Dans la technologie FLASH

- a) Il est impossible d'écrire une donnée
- b) Il est impossible de lire une donnée
- c) Le contenu est perdu lors des coupures d'alim
- d) L'écriture se fait par blocs

Q11) Un processeur dispose d'un bus d'adresse de 24 bits, et d'un bus de donnée de 32 bits (4 octets). Quelle est la taille de l'espace adressable ?

- a) 16 Mo
- b) 32 Mo
- c) 64 Mo
- d) 1282 Mo

Q12) Quel composant consomme le plus d'énergie par bit stocké ?

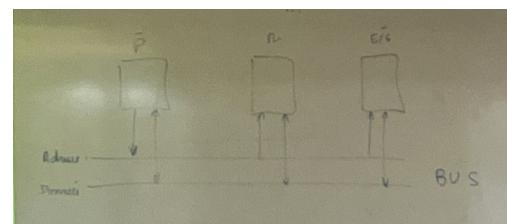
- a) Disque dur
- b) registre
- c) Mémoire FLASH
- d) Mémoire SDRAM

Q13) Quelle proposition est vraie ?

- a) Un registre est plus rapide qu'une SRAM
- b) Une DRAM est plus rapide qu'une SRAM
- a) Une FLASH est plus rapide qu'une DRAM
- b) Une DRAM utilise 4 transistors par bit stocké

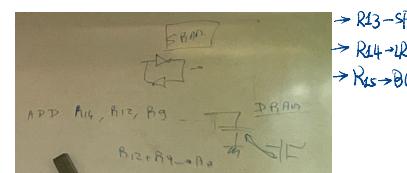
Q14) Dans l'architecture de Von Neuman...

- a) Les périphériques lisent la mémoire
- b) Les périphériques sont lus directement par le processeur
- c) La mémoire gère les écritures
- d) Le bus gère les écritures



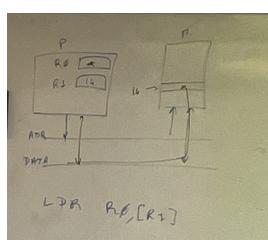
Q15) A quoi sert le registre R14 du processeur ARM ?

- R15 →
- a) Il contient toujours le résultat des opérations arithmétiques et logiques
  - b) A retenir l'adresse du code en cours d'exécution en mémoire
  - c) C'est un registre d'usage général
  - d) A retenir l'adresse de retour d'un sous-programme appelé avec l'instruction BL



Q16) Lors de l'instruction : LDR R0,[R1] L'opérande source est :

- a) Dans le processeur
- c) Dans un périphérique
- b) Dans la mémoire
- d) Dans la pile



## C] Assembleur ARM

Pour les questions suivantes **les registres sont initialisés à 0**. Le contenu des mémoires est aléatoire avant qu'on écrive dedans, et la mémoire autour de l'adresse 0 est de type RAM (lecture / écriture).

Q17) Quel est le contenu du registre R0 après : **mov R0,#-5**

- a) 0x5      b) 0xF5      c) 0xfffffffffb      d) 0xFFFFFFFF5

Q18) Quel est le contenu du registre R0 après : **orr R0,R1,#3** (orr est le OU logique)

- a) -3      b) 0      c) 3      d) 0xFD

$$\begin{array}{r}
 \text{0000} & \text{0000} & \text{R2} \\
 \text{00} \text{ 0000} & \text{0011} & \underline{\text{3}} \\
 \hline
 \text{0000} & \text{0011} = \underline{\text{3}}
 \end{array}$$

Q19) Quel est le contenu de R3 après

**ldr r4,=8  
sub r3,r4,#3**

- a) 5      b) -5      c) -3      d) 3

Q20) Quel est le contenu du registre R0 après :

**and R0,R3,#14**

- a) 0x14      b) 0x0E      c) 0x00000002      d) 0

Q21) Quel est le contenu de R0 après :

**ldr r0,=0x19  
sub r0,r0,r0**

- a) 0xFFFFFE7      b) 0xFFFFFFFED      c) 0      d) 0x19

Q22) Quel est le contenu de r0 après :

**ldr r0,=0x40000000  
adds r0,r0,r0  
adds r0,r0,r0**

- a) 0      b) 0x80000000      c) 0xc0000000      d) E0000000

Q23) Quel est le contenu de r0 après :

**ldr r7,=0x80  
adds r7,r7,r7  
adcs r0,r7,r7**

- a) 0x201      b) 0x200      c) 0x100      d) 0x80

Q24) Quelle est la valeur du registre R0 après

```
ldr    r0,=0x30
ldr    r1,=0x20
sub   r0,r0,r1
```

a) 0xFFFFFFFF

b) 0

c) ✓ 0x10

d) 0x20

Q25) Quel est le contenu de r0 après : (EOR est l'opération de ou exclusif logique)

```
ldr    r0,=0x12345678
ldr    r1,=0x87654321
eor   r0,r1,r0
```

a) 0x022444220

b) 0x95511559

c) 0x9775579

d) 0x77777777

Q26) Quelle est la valeur du registre R0 après

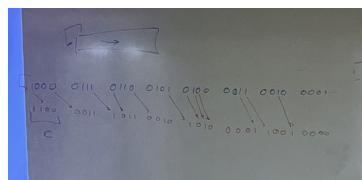
```
ldr    r0,=0x87654321
mov   r0,r0,asr #1
```

a) 0x08765432

b) 0x76543210

c) 0x0ECA8642

d) ✓ 0xC3B2A190



Q27) Quel est le contenu de R0 après : (GT : Greater or Than)

```
mov   r0,#10
cmp   r0,#8
sublt r0,r0,#2
```

a) ✓ 10

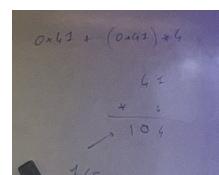
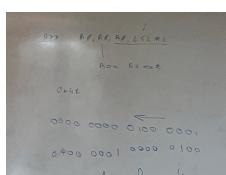
b) 0x10

c) 8

d) 6

Q28) Quel est le contenu de r0 après :

```
mov   r0,#0x41
add   r0,r0,r0,lsl #2
```



a) 0x20

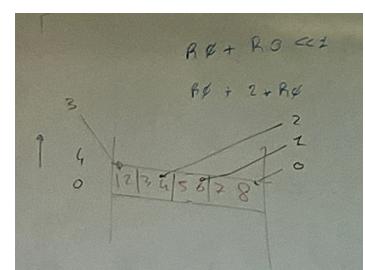
b) ✓ 0x82

c) 0x145

d) 0x249

Q29) On souhaite faire  $r0 * 3$  quel est la ligne qui convient ?

- |  |  |
|--|--|
| a) add r0,r0,r0 lsl #1                                 | b) add r0,r0,r0 lsl #5 $\Rightarrow R0 + 32 * R0$      |
| b) add r0,r0,r0 lsr #2 $\Rightarrow R0 + \frac{R0}{4}$ | d) add r0,r0,r0 lsr #1 $\Rightarrow R0 + \frac{R0}{2}$ |



Q30) Quelle est la valeur du registre R0 après :

```
ldr   r1,=0x12345678
str   r1,[r0]
ldrb  r0,[R0,3]
```

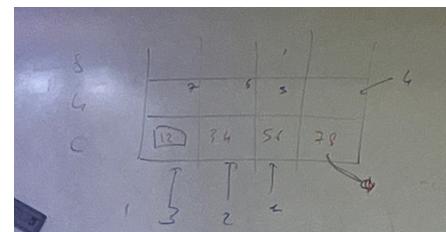
;ldrb : Load byte

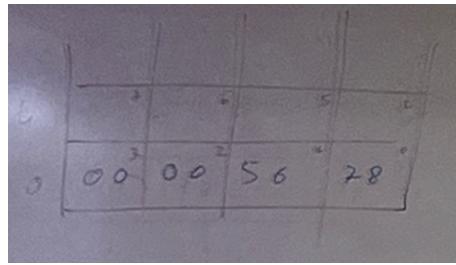
a) ✓ 0x12

b) 0x34

c) 0x56

d) 0x78





Q31) Quelle est la valeur du registre R0 après :

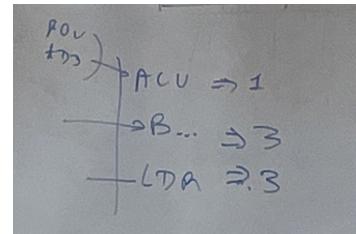
**ldr r1,=0x12345678**  
**strh r1,[r8]** ; strh ; store half word  
**ldr r0,[r8]** ; (ldr : load word)

- a) 0x1234    b) 0x5678    c) 0x12345678    d) 0x56781234

Q32) Quelle est la durée d'exécution du programme suivant en cycle horloge ?

**ldr r0,=3**    —3  
**mov r0,r1**    —1  
**sub r0,r1,r2**    —1  
**bal ici**    —3    ; bal : branch always  
**ici add r0,r0,#3**    —1  
                    —9

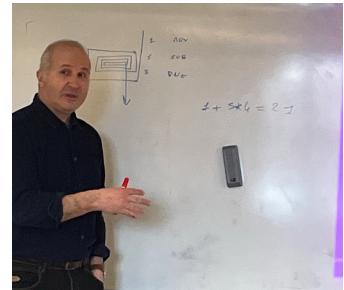
a) 5    b) 7    c) 8    d) 9



Q33) Quelle est la durée d'exécution du programme suivant :

**Ici mov r0,#5**    1 → 1  
**Ici subs r0,r0,#1**    1 } 5  
**Ici bne ici**    3    : (Branch if not equal to 0)  
Fin

- a) 5    b) 15    c) 21    d) 34



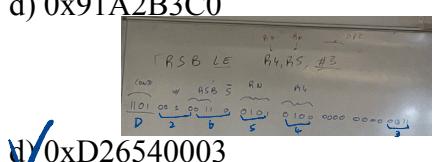
Q34) Quel est le contenu de R0 après :

**Ldr r0,=0x12345678**  
**Mov r0,r0,asr #3**

- a) 0x2468ACF    b) 0x00012345    c) 0x45678000    d) 0x91A2B3C0

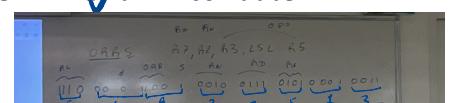
Q35) Coder l'instruction **RSBLE R4,R5,#3**

- a) 0xD25450003    b) 0xE25450003    c) 0xD26450003    d) 0xD26540003



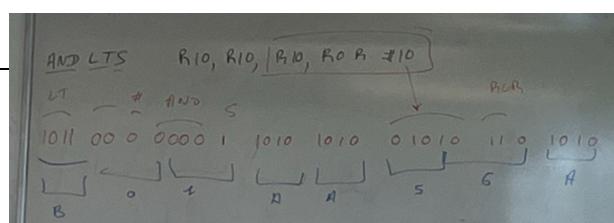
Q36) Coder l'instruction **ORRS R7,R2,R3,LSL R5**

- a) 0xE3927513    b) 0xE1927513    c) 0xE1927515    d) 0xE1727513



Q37) Coder l'instruction **ANDLTS R10,R10,R10,ROR#10**

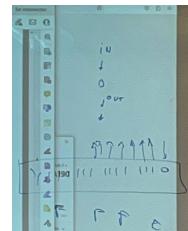
- a) 0xB01AA566    b) 0xB0199569    c) 0xB02BB56B    d) 0xB01AA56A



## D) Périmétries

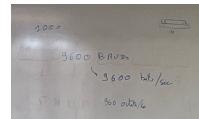
Q38) On programme le port parallèle de la façon suivante : bits 0 en entrée, bit 1-15 en sortie  
Quelle valeur place-t-on dans PC2 ?

- a) 0x0001    b) ✓ 0xFFFF    c) 0xFF00    d) 0x000F



Q39) On transmet 1Koctets de données sur 8 bits sans parité à 9600 bauds. Quel est l'ordre de grandeur du temps de transmission ?

- a) ✓ 1 seconde    b) 10 secondes    c) 20 minutes    d) 2 heures



Q40) Quelle est la valeur du bit de parité paire de la donnée 0x52?    0+52    1000

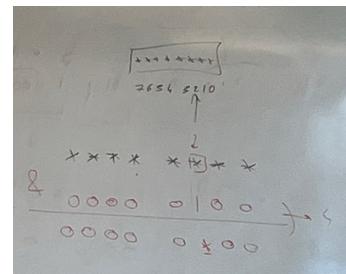
- a) 0    b) ✓ 1    c) 00    d) 11

0101 0010 |

Q41) On souhaite tester un interrupteur placé sur le bit 2 du port parallèle. Que vaut XX ?

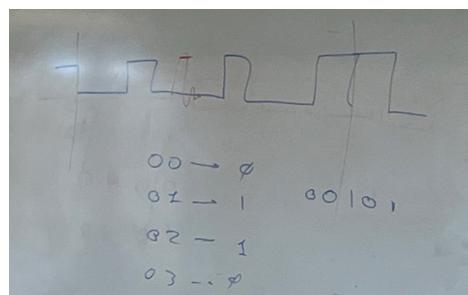
Ldrh r0,[PD]  
Ands r0,#XX  
Bne INTER\_ON

- a) 0    b) 1    c) 2    d) ✓ 4



Q42) On souhaite communiquer avec un module Bluetooth, quel interface faut-il utiliser ?

- a) ✓ Série    b) Timer    c) convertisseur A/D    d) Parallèle



## Codage d'une instruction arithmétique

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<i>cond</i>	0	0	#		<i>op</i>	S		<i>Rn</i>			<i>Rd</i>																				

Les champs variables ont la signification suivante :

*Cond* : condition d'exécution de l'instruction. Les différentes valeurs possibles sont les suivantes :



Condition (cond)				Code Opération (op)			
Valeur	condition	Valeur	condition	Valeur	condition	Valeur	condition
0000	EQ	1000	HI	0000	AND	1000	TST
0001	NE	1001	LS	0001	EOR	1001	TEQ
0010	CS	1010	GE	0010	SUB	1010	CMP
0011	CC	1011	LT	0011	RSB	1011	CMN
0100	MI	1100	GT	0100	ADD	1100	ORR
0101	PL	1101	LE	0101	ADC	1101	MOV
0110	VS	1110	{AL}	0110	SBC	1110	BIC
0111	VC	1111	NV	0111	RSC	1111	MVN

# : Format du 2nd opérande. S'il s'agit d'une valeur immédiate, ce champ est à 1. Sinon, il est à 0.

S : est à 1 si les codes conditions doivent être mis à jour.

*Rn* : numéro du registre opérande 1

*Rd* : numéro du registre destination

*operand2* : second opérande. Trois formats sont possibles :

Opérande **registre, décalage éventuel spécifié par une constante** :

11	10	9	8	7	6	5	4	3	2	1	0
#shift	Sh	0	<i>Rm</i>								

#shift : Nombre de positions de décalage

Sh : Type de décalage (00 = LSL, 01 = LSR, 10 = ASR, 11 = ROR)

*Rm* : Numéro du registre

Opérande **registre, décalage éventuel spécifié par un registre** :

11	10	9	8	7	6	5	4	3	2	1	0
<i>Rs</i>	0	Sh	1	<i>Rm</i>							

*Rs* : Numéro du registre contenant le nombre de positions de décalage

Sh : Type de décalage (00 = LSL, 01 = LSR, 10 = ASR, 11 = ROR)

*Rm* : Numéro du registre

Opérande **immédiat (de la forme valeur\_8\_bits \* 2^2n) : (Bit 25 à 1)**

11	10	9	8	7	6	5	4	3	2	1	0
#rot	value										

#rot : n

value : Valeur sur 8 bits

**Codage d'une instruction de branchement**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond		1	0	1	L	offset																									

Les champs variables ont la signification suivante :

cond : condition du branchement. Les différentes valeurs possibles sont les mêmes que pour les instructions arithmétiques (voir ci-dessus).

L : Si l'adresse suivante doit être mémorisée dans r14 (adresse de retour), ce champ est à 1. Sinon, il est à 0.

offset : déplacement (d) divisé par 4. L'adresse cible du branchement sera calculée comme :  
adresse cible = adresse du branchement (PC) + 8 + d

**Périphérique rappel**

Le sens des échanges se programme par les registres PC0 PC1 et PC2 du périphérique parallèle.

La lecture ou l'écriture du port parallèle s'effectue par le registre PD.

Pour chaque bit, le sens des échanges est programmé comme suit :

PC0	PC1	PC2	Etat
1	0	1	Sortie
1	0	0	Entrée

**Temps d'exécution des instructions :**

ALU	1	+1 si shift +2 si Rd est pc
B, BL, BX	3	
LDR	3	+2 si rd est PC

**TP EXAMEN BLANC**

Aidez-vous du logiciel pour répondre aux questions Q1 à Q10. Faites constater les programmes Q11-Q12.

Q1) Quel est le contenu de R0 après :

```
Ldr    r0,=0x8765
Ldr    r1,=0x1234
eor    r0,r0,r1
```

**R0 =**

Q2) Quel est le contenu de R0 après :

```
Ldr    r0,=0x3452
And   r0,r0,#0x37
```

**R0 =**

Q3) Quel est l'état des drapeaux après:

```
Ldr    r0,=0x8765
Ldr    r1,=0x1234
subs  r0,r0,r1
```

**NZVC =**

Q4) Quel est l'état des drapeaux NZCV après l'opération :

45 – 56

**NZVC =**

Q5) Quel est le résultat de l'opération OU logique entre les mots 0x123456 et 0x654321 ?

**Nbr :**

Q6) Quel est le code de l'instruction : ORR

R1,R2,R3

**Code : 0x**

Q7) Quelle est l'instruction qui se code : 0xE0A10002

(On rappelle que la pseudo instruction dcd

Permet d'écrire un mot de 32 bits dans le code source).

**Instruction :**

Q8) Quel est le code de l'instruction LDR R3,[R10]

**Code : 0x**

Q9) On place R0 à la valeur 0x1234 et R10 à la valeur 0x9000 puis on exécute l'instruction :

STR R0,[R10]

Quel est le contenu de la case 0x9001 ?

(utiliser la fenêtre memory)

**0x9001 :**

Q10) Quel chiffre placer dans Nbr pour lever

V et N après Nbr – 0x45 ?

**Nbr =**Q11 ) Utiliser le programme du TP3 pour illuminer les Leds de la façon suivante :

OOOOOOOOXXXXXXX

X = led allumée

O = Led éteinte

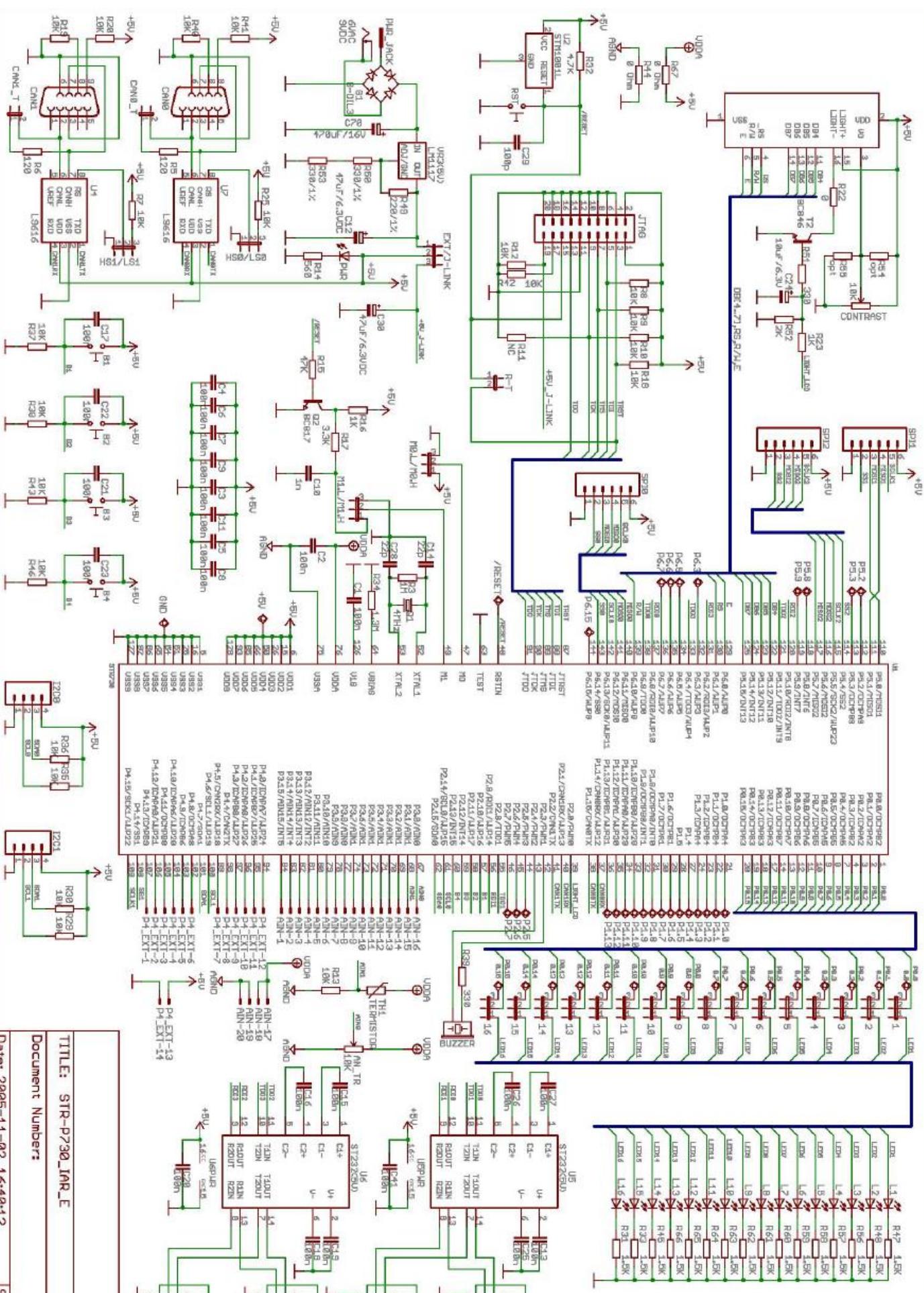
Q12) Faire alterner les LEDs avec le motif ci-dessus et toutes les Leds éteintes avec un rythme d'une demi seconde environ.

## ANNEXE :

- **Schéma de la carte IAR utilisée en TP**
- **Adresse des périphériques du SRT730**
- **Les ports parallèles**
- **Les instructions de l'ARM7**

# Schéma de la carte IAR

IE.1202 - Architecture des ordinateurs



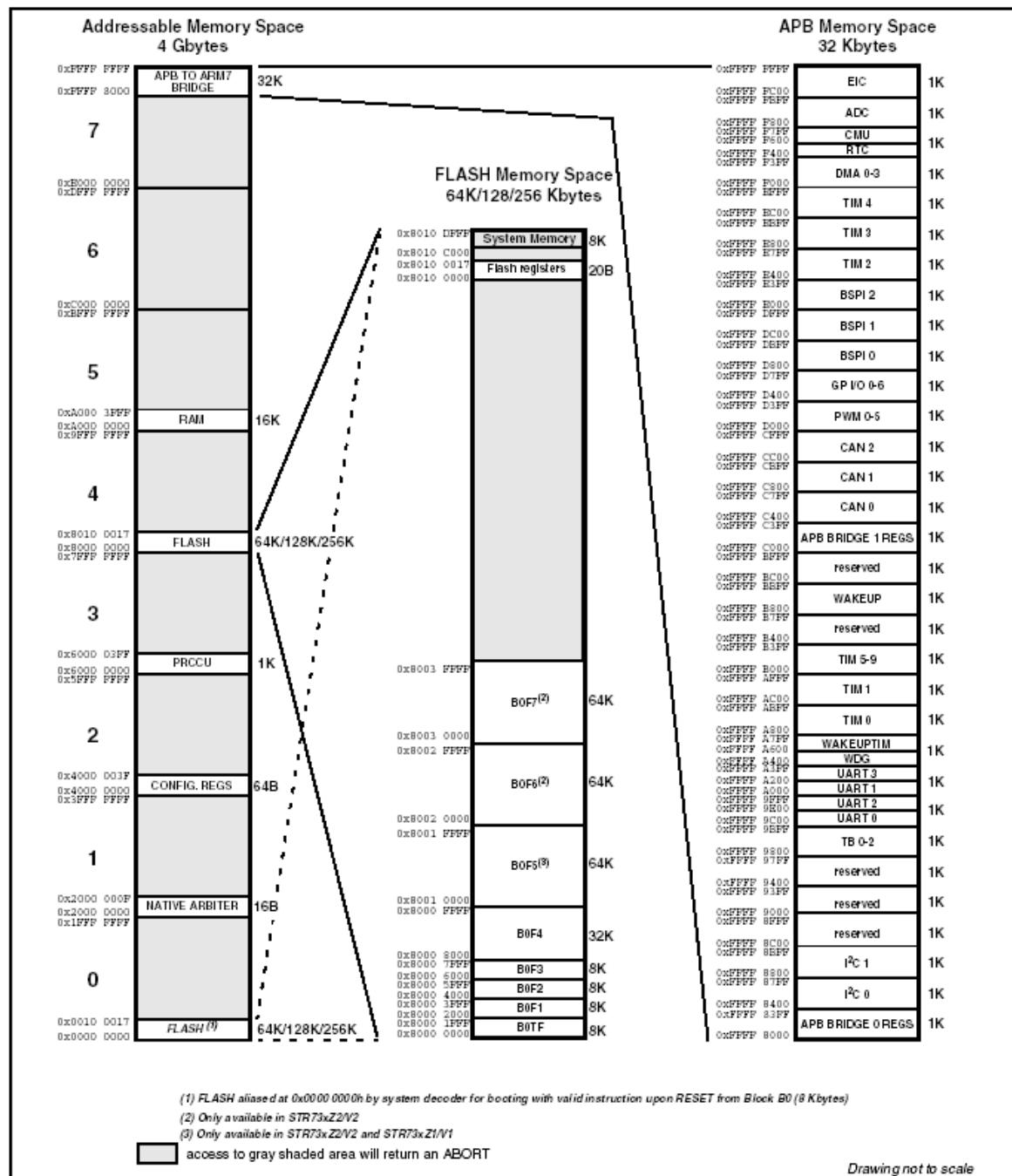
TITLE: STR-P730_IAR_E	Document Number:	REF:
Date: 2005-11-02 16:40:12	Sheet: 1/1	E

# Le composant STR730 – Adresse des périphériques

## STR73xF MICROCONTROLLER - MEMORY

### 2.1.1 Memory Map

Figure 1. Memory Map



**STR73xF MICROCONTROLLER - MEMORY****2.1.1.1 Register Base Addresses****Table 1. Main Block Register base addresses**

STR73x Main Blocks	Base Address	Block Register Map
Configuration Registers (CFG)	0x4000 0000	<a href="#">Section 4.5</a>
Power, Reset and Clock Control Unit (PRCCU)	0x6000 0000	<a href="#">Section 3.6.8</a>
Flash	0x8000 0000	See STR7xx Family Flash Programming Reference Manual

**2.1.1.2 APB Memory Map****Table 2. APB Memory Map**

Sub Page	SubPage Boundary Addresses	Peripheral	Peripheral Boundary Addresses	Bus Access Width	Register Map
0	0xFFFF 8000	APB Bridge 0 Registers	0xFFFF 8000	32-bit	<a href="#">Section 21.2</a>
	0xFFFF 83FF		0xFFFF 800F		
1	0xFFFF 8400	I <sup>2</sup> C 0	0xFFFF 8400	8 bit	<a href="#">Section 17.6</a>
	0xFFFF 87FF		0xFFFF 841F		
2	0xFFFF 8800	I <sup>2</sup> C 1	0xFFFF 8800	8 bit	
	0xFFFF 8BFF		0xFFFF 881F		
3	0xFFFF 8C00	reserved			
	0xFFFF 8FFF				
4	0xFFFF 9000	reserved			
	0xFFFF 93FF				
5	0xFFFF 9400	reserved			
	0xFFFF 97FF				
6	0xFFFF 9800 0xFFFF 9BFF	Timebase (TB) Timer 0	0xFFFF 9800 0xFFFF 9817	16 bit	<a href="#">Section 13.4</a>
			0xFFFF 9900 0xFFFF 9917		
			0xFFFF 9A00 0xFFFF 9A17		
7	0xFFFF 9C00 0xFFFF 9FFF	UART 0	0xFFFF 9C00 0xFFFF 9C27	16 bit	<a href="#">Section 19.5</a>
			0xFFFF 9E00 0xFFFF 9FFF		
		UART 2	0xFFFF A000 0xFFFF A027	16 bit	
8	0xFFFF A000 0xFFFF A3FF	UART 1	0xFFFF A200 0xFFFF A3FF	16 bit	
			0xFFFF A000 0xFFFF A200		

**STR73xF MICROCONTROLLER - MEMORY****Table 2. APB Memory Map**

Sub Page	SubPage Boundary Addresses	Peripheral	Peripheral Boundary Addresses	Bus Access Width	Register Map	
9	0xFFFF A400 0xFFFF A7FF	Watchdog (WDG)	0xFFFF A400	16 bit	<a href="#">Section 12.5</a>	
			0xFFFF A41B			
		Wake-up Timer (WUT)	0xFFFF A600	16 bit	<a href="#">Section 10.6</a>	
			0xFFFF A617			
10	0xFFFF A800 0xFFFF ABFF	TIM 0	0xFFFF A800	16 bit	<a href="#">Section 14.7</a>	
			0xFFFF A81F			
11	0xFFFF AC00 0xFFFF AFFF	TIM 1	0xFFFF AC00	16 bit		
			0xFFFF AC1F			
12	0xFFFF B000 0xFFFF B3FF	TIM 5	0xFFFF B000	16 bit		
			0xFFFF B01F			
		TIM 6	0xFFFF B080	16 bit		
			0xFFFF B09F			
		TIM 7	0xFFFF B100	16 bit		
			0xFFFF B11F			
		TIM 8	0xFFFF B180	16 bit		
			0xFFFF B19F			
		TIM 9	0xFFFF B200	16 bit		
			0xFFFF B21F			
13	0xFFFF B400 0xFFFF B7FF	reserved				
14	0xFFFF B800 0xFFFF BBFF	Wake-up/Interrupt Unit (WIU)	0xFFFF B800 0xFFFF B813	32 bit	<a href="#">Section 7.9.4.6</a>	
15	0xFFFF BC00 0xFFFF BFFF	reserved				
16	0xFFFF C000 0xFFFF C3FF	APB Bridge 1 Registers	0xFFFF C000 0xFFFF C00F	32-bit	<a href="#">Section 21.2</a>	
17	0xFFFF C400 0xFFFF C7FF		0xFFFF C400 0xFFFF C57F			
18	0xFFFF C800 0xFFFF CBFF	CAN 0	0xFFFF C800 0xFFFF C97F	16 bit	<a href="#">Section 16.6</a>	
19	0xFFFF CC00 0xFFFF CFFF	CAN 1	0xFFFF CC00 0xFFFF CFFF	16 bit		
		CAN 2				

**STR73xF MICROCONTROLLER - MEMORY****Table 2. APB Memory Map**

Sub Page	SubPage Boundary Addresses	Peripheral	Peripheral Boundary Addresses	Bus Access Width	Register Map
20	0x FFFF D000 0x FFFF D3FF	PWM 0	0x FFFF D000 0x FFFF D023	16 bit	Section 15.5
		PWM 1	0x FFFF D040 0x FFFF D063	16 bit	
		PWM 2	0x FFFF D080 0x FFFF D0A3	16 bit	
		PWM 3	0x FFFF D0C0 0x FFFF D0E3	16 bit	
		PWM 4	0x FFFF D100 0x FFFF D123	16 bit	
		PWM 5	0x FFFF D140 0x FFFF D163	16 bit	
		GP I/O - Port 0	0x FFFF D400 0x FFFF D40F	16 bit	
21	0x FFFF D400 0x FFFF D7FF	GP I/O - Port 1	0x FFFF D410 0x FFFF D41F	16 bit	Section 6.2.1
		GP I/O - Port 2	0x FFFF D420 0x FFFF D42F	16 bit	
		GP I/O - Port 3	0x FFFF D430 0x FFFF D43F	16 bit	
		GP I/O - Port 4	0x FFFF D440 0x FFFF D44F	16 bit	
		GP I/O - Port 5	0x FFFF D450 0x FFFF D45F	16 bit	
		GP I/O - Port 6	0x FFFF D460 0x FFFF D46F	16 bit	
		BSPI 0	0x FFFF D800 0x FFFF DBFF	16 bit	
22	0x FFFF D800	BSPI 1	0x FFFF D817	16 bit	Section 18.4
23	0x FFFF DC00		0x FFFF DC00 0x FFFF DC17		
24	0x FFFF E000	BSPI 2	0x FFFF E000 0x FFFF E017	16 bit	
25	0x FFFF E400	TIM 2	0x FFFF E400 0x FFFF E41F	16 bit	Section 14.7
26	0x FFFF E800		0x FFFF E800 0x FFFF E81F		
27	0x FFFF EC00	TIM 4	0x FFFF EC00 0x FFFF EC1F	16 bit	

**STR73xF MICROCONTROLLER - MEMORY****Table 2. APB Memory Map**

Sub Page	SubPage Boundary Addresses	Peripheral	Peripheral Boundary Addresses	Bus Access Width	Register Map
28	0x FFFF F000 0x FFFF F3FF	DMA 0	0x FFFF F000 0x FFFF F0FB	16 bit	Section 8.6
		DMA 1	0x FFFF F100 0x FFFF F1FB	16 bit	
		DMA 2	0x FFFF F200 0x FFFF F2FB	16 bit	
		DMA 3	0x FFFF F300 0x FFFF F3FB	16 bit	
29	0x FFFF F400 0x FFFF F7FF	Realtime Clock (RTC)	0x FFFF F400 0x FFFF F427	16 bit	Section 11.5
		Clock Monitor Unit (CMU)	0x FFFF F600 0x FFFF F61F	16 bit	
30	0x FFFF F800 0x FFFF FBFF	Analog/Digital Converter (ADC)	0x FFFF F800 0x FFFF F94F	16 bit	Section 20.5
		Enhanced Interrupt Controller (EIC)	0x FFFF FC00 0x FFFF FD5F	32 bit	
31	0x FFFF FC00 0x FFFF_FFFF		0x FFFF FC00 0x FFFF FD5F	Section 7.7	

# Le composant STR730 – LES PORTS PARALLELES

## 6 I/O PORTS

### 6.1 Functional Description

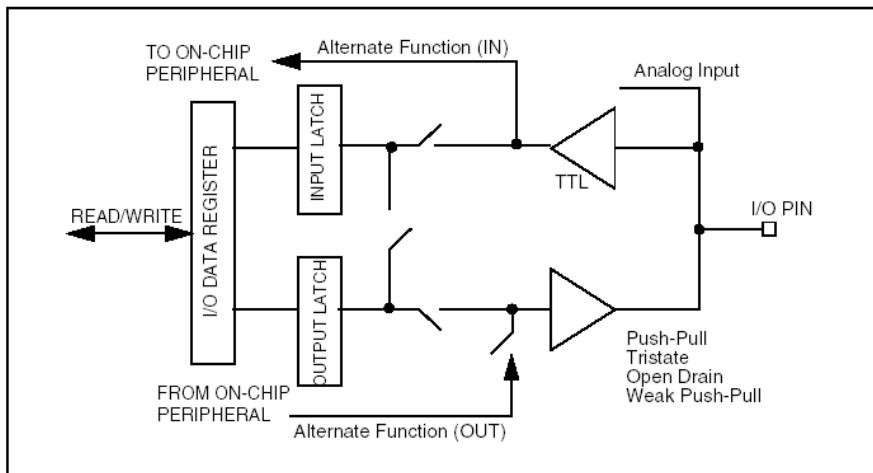
Each of the General Purpose I/O Ports has three 16-bit configuration registers (PC0, PC1, PC2) and one 16-bit Data register (PD).

Subject to the specific hardware characteristics of each I/O port listed in the datasheet, you can configure each port bit individually as input, output, alternate function etc.

Each I/O port bit is freely programmable, however the I/O port registers have to be accessed as 16-bit words. Byte or bit-wise access is not allowed.

[Figure 15](#) shows the basic structure of an I/O Port bit.

**Figure 15. Basic Structure of an I/O Port Bit**



**Table 16. Port Bit Configuration Table**

Port Configuration Registers (bit)	Values							
	PC0(n)	1	0	1	0	1	0	1
PC1(n)	0	0	1	1	0	0	1	1
PC2(n)	0	0	0	0	1	1	1	1
Configuration	HIZ/AIN	IN	reserved	IPUPD	OUT	OUT	AF	AF
Output	TRI	TRI		WP	OD	PP	OD	PP
Input	-	TTL		TTL	TTL	TTL	TTL	TTL

**Notes:**

AF: Alternate Function

OD: Open Drain

AIN: Analog Input

OUT: Output

HIZ: High impedance

PP: Push-Pull

IN: Input

TRI: Tristate

IPUPD: Input Pull Up /Pull Down

TTL: TTL Input levels

WP \*: Weak Pull-Up

PP : Weak Push-Pull

\*) Depending on PD(n) value it behaves as Weak Pull-up (PD=1) or Weak Pull-down (PD=0)

### General Purpose I/O (GPIO)

At reset the I/O ports are configured as general purpose (memory mapped I/O).

When you write to the I/O Data register the data is always loaded in the Output Latch. The Output Latch holds the data to be output while the Input Latch captures the data present on the I/O pin.

A read access to the I/O Data register reads the Input Latch or the Output Latch depending on whether the Port bit is configured as input or output.

### Alternate Function I/O (AF)

The alternate functions for each pin are listed in the datasheet. If you configure a port bit as Alternate Function, this disconnects the output latch and connects the pin to the output signal of an on-chip peripheral.

To use the alternate function, you also have to enable it in the peripheral control registers. Only one alternate function can be used on each pin.

- For AF input, the port bit can be either in Input or AF configuration
- For AF output or input-output, the port bit must be in AF configuration

## STR73xF MICROCONTROLLER - I/O PORTS

### External Interrupts/wake-up lines

Some ports have external interrupt capability (see datasheet). To use external interrupts, the port must be configured in input mode. For more information on interrupts and wake-up lines, refer to [Section 7](#).

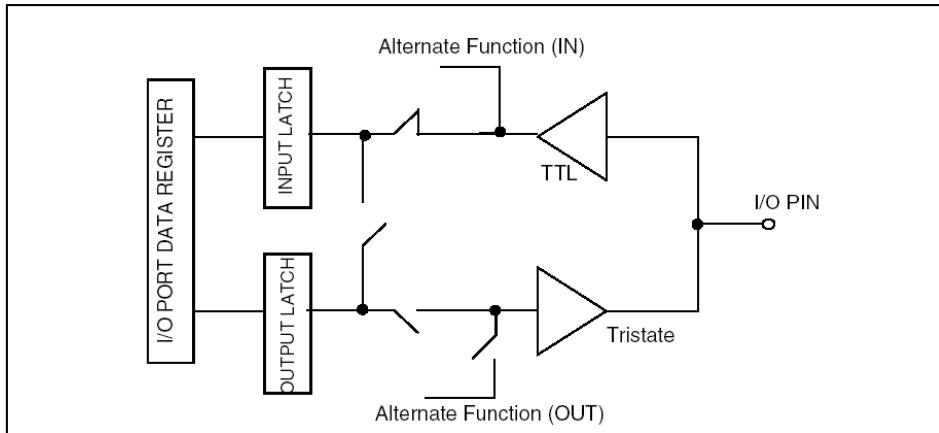
#### 6.1.1 Input Configuration

When the I/O Port is programmed as Input:

- The Output Buffer is forced tristate
- The data present on the I/O pin is sampled into the Input Latch every clock cycle
- A read access to the Data register gets the value in the Input Latch.

The [Figure 16 on page 79](#) shows the Input Configuration of the I/O Port bit.

**Figure 16. Input Configuration**



## STR73xF MICROCONTROLLER - I/O PORTS

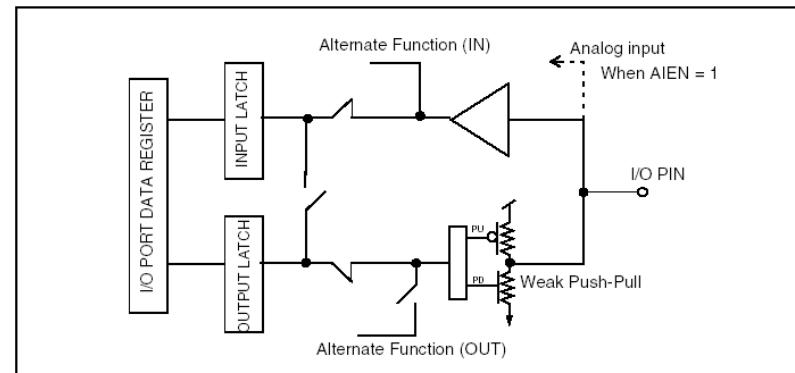
### 6.1.2 Input Pull Up/Pull Down Configuration

When the I/O Port is programmed as Input Pull Up/Pull Down:

- The Output Buffer is turned on in Weak Push-Pull configuration and software can write the appropriate level in the output latch to activate the weak pull-up or pull-down as required.
- The data in the Output Latch drives the I/O pin (a logic zero activates a weak pull-down, a logic one activates a weak pull-up)
- A read access to the I/O Data register gets the Input Latch value.

The [Figure 17](#) shows the Input PUPD Configuration of the I/O Port.

**Figure 17. Input Pull Up/Pull Down Configuration**



---

STR73xF MICROCONTROLLER - I/O PORTS

---

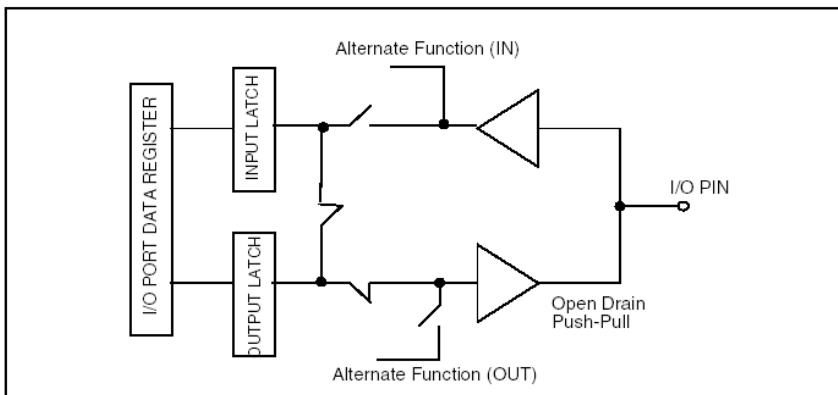
**6.1.3 Output Configuration**

When the I/O Port is programmed as Output:

- The Output Buffer is turned on in Open Drain or Push-Pull configuration
- The data in the Output Latch drives the I/O pin
- A read access to the I/O Data register gets the Output Latch value.

The [Figure 18 on page 81](#) shows the Output Configuration of the I/O Port bit.

**Figure 18. Output Configuration**



---

STR73xF MICROCONTROLLER - I/O PORTS

---

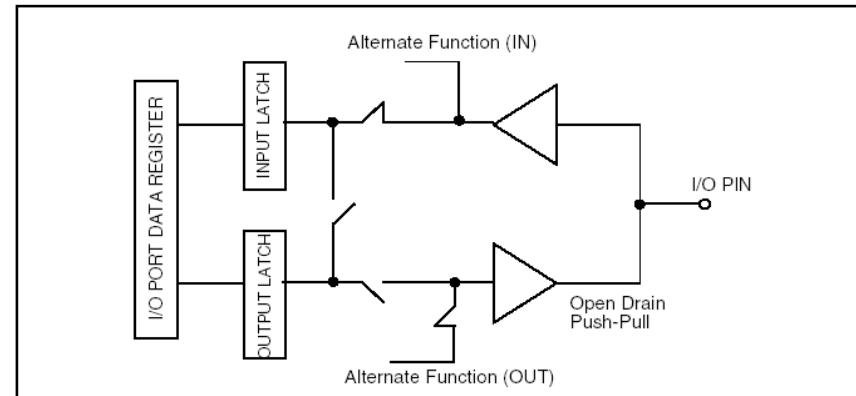
**6.1.4 Alternate Function Configuration**

When the I/O Port is programmed as Alternate Function:

- The Output Buffer is turned on in Open Drain or Push-Pull configuration
- The Output Buffer is driven by the signal coming from the peripheral (alternate function out)
- The data present on the I/O pin is sampled into the Input Latch every clock cycle
- A read access to the Data register gets the value in the Input Latch.

The [Figure 19 on page 82](#) shows the Alternate Function Configuration of the I/O Port bit.

**Figure 19. Alternate Function Configuration**



---

STR73xF MICROCONTROLLER - I/O PORTS

---

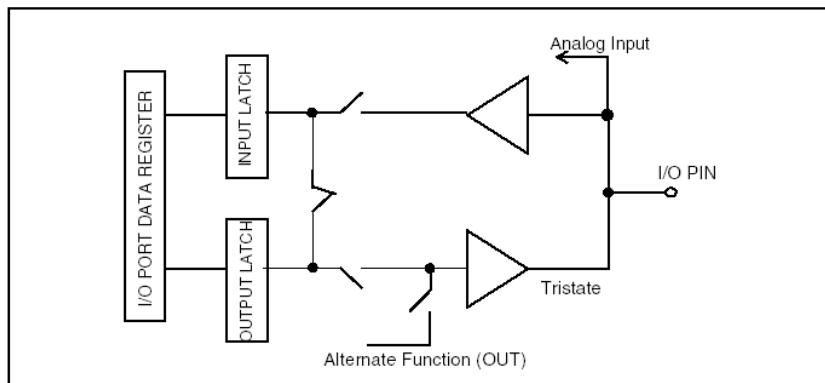
**6.1.5 High impedance-Analog Input Configuration**

When the I/O Port is programmed as High impedance-Analog Input Configuration:

- The Output Buffer is forced tristate
- The Input Buffer is disabled (the Alternate Function Input is forced to a constant value)
- The Analog Input can be input to an Analog peripheral
- A read access to the I/O Data register gets the Output Latch value

The [Figure 20 on page 83](#) shows the High impedance-Analog Input Configuration of the I/O Port bit.

**Figure 20. High impedance-Analog Input Configuration**



---

STR73xF MICROCONTROLLER - I/O PORTS

---

**6.2 Register Description**

The I/O port registers cannot be accessed by byte.

**Port Configuration Register 0 (PC0)**

Address Offset: 00h

Reset value: 0xFFFF

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C015	C014	C013	C012	C011	C010	C09	C08	C07	C06	C05	C04	C03	C02	C01	C00

rw    rw

Bit 15:0 = C0[15:0]: Port Configuration bits

See [Table 16 on page 78](#) to configure the I/O Port.

**Port Configuration Register 1 (PC1)**

Address Offset: 04h

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C115	C114	C113	C112	C111	C110	C19	C18	C17	C16	C15	C14	C13	C12	C11	C10

rw    rw

Bit 15:0 = C1[15:0]: Port Configuration bits

See [Table 16 on page 78](#) to configure the I/O Port.

**Port Configuration Register 2 (PC2)**

Address Offset: 08h

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
C215	C214	C213	C212	C211	C210	C29	C28	C27	C26	C25	C24	C23	C22	C21	C20

rw    rw

Bit 15:0 = C2[15:0]: Port Configuration bits

See [Table 16 on page 78](#) to configure the I/O Port.

## Les instructions de l'ARM7

### *Les instructions arithmétiques et logiques*

#### ADC : Addition with Carry

ADC<suffix> <dest>, <op 1>, <op 2>

$$\text{dest} = \text{op\_1} + \text{op\_2} + \text{carry}$$

ADC ajoute les deux opérandes et place le résultat dans le registre destination. Cette instruction utilise le bit de retenue (C). L'exemple suivant ajoute deux nombres de 128 bits :

Résultat : R0 R1 R2 et R3

Opérande 1 : R4 R5 R6 R7

Opérande 2 : R8 R9 R10 R11

ADDS	R0, R4, R8	; Add low words
ADCS	R1, R5, R9	; Add next word, with carry
ADCS	R2, R6, R10	; Add third word, with carry
ADCS	R3, R7, R11	; Add high word, with carry

Il est nécessaire d'utiliser le suffixe S pour positionner les drapeaux.

#### ADD : Addition

ADD<suffix> <dest>, <op 1>, <op 2>

$$\text{dest} = \text{op\_1} + \text{op\_2}$$

ADD ajoute les deux opérandes et place le résultat dans le registre destination. L'opérande 1 est un registre. L'opérande 3 est un registre, un registre décalé ou une valeur immédiate.

ADD	R0, R1, R2	; R0 = R1 + R2
ADD	R0, R1, #256	; R0 = R1 + 256
ADD	R0, R2, R3,LSL#1	; R0 = R2 + (R3 << 1)

#### AND : Logical AND

AND<suffix> <dest>, <op 1>, <op 2>

$$\text{dest} = \text{op\_1} \text{ AND } \text{op\_2}$$

AND effectue un ET logique entre les deux opérandes d'entrée et place le résultat dans le registre de sortie.

Cette opération est utilisée pour placer un bit spécifique à 0.

AND R0, R0, #3 ; R0 = Keep bits zero and one of R0, discard the rest.

An AND table (result = both):

Op_1	Op_2	Result
------	------	--------

0	0	0
0	1	0
1	0	0
1	1	1

## BIC : Bit Clear

BIC<suffix> <dest>, <op 1>, <op 2>

$$\text{dest} = \text{op\_1 AND } (\text{!op\_2})$$

BIC permet de placer des bits spécifiés à 0. L'opérande 2 est un masque. Les bits à 1 de ce masque indiquent les bits qui placés à 0.

BIC R0, R0, #%1011 ; Clear bits zero, one, and three in R0. Leave the remaining bits alone.

Op_1	Op_2	Result
------	------	--------

0	0	0
0	1	0
1	0	1
1	1	0

## EOR : Logical Exclusive OR

EOR<suffix> <dest>, <op 1>, <op 2>

$$\text{dest} = \text{op\_1 EOR op\_2}$$

EOR effectue un ou exclusive bit à bit entre les deux opérandes.

EOR R0, R0, #3 ; Invert bits zero and one in R0

An EOR table (result = either, but not both):

Op_1	Op_2	Result
------	------	--------

0	0	0
0	1	1
1	0	1
1	1	0

## MOV : Move

MOV<suffix> <dest>, <op 1>

$$\text{dest} = \text{op\_1}$$

MOV charge une valeur dans le registre destination à partir de : un autre registre, un registre décalé, ou une valeur immédiate.

MOV R0, R0	; R0 = R0... NOP instruction
MOV R0, R0, LSL#3	; R0 = R0 * 8

If R15 is the destination, the program counter or flags can be modified. This is used to return to calling code, by moving the contents of the link register into R15:

MOV PC, R14	; Exit to caller
MOVS PC, R14	; Exit to caller preserving flags
(not 32-bit compliant)	

## MVN : Move Negative

MVN<suffix> <dest>, <op 1>

$$\text{dest} = !\text{op\_1}$$

MVN charge une valeur dans le registre de destination à partir d'un registre, d'un registre décalé, ou d'une valeur immédiate. Les bits sont inversés avant d'être transférés, ce qui permet de charger des chiffres négatifs (avec l'opération MOV les valeurs immédiates étant limitées à 8 bits, les chiffres ne peuvent pas être négatifs).

MVN R0, #4	; R0 = -5
MVN R0, #0	; R0 = -1

## ORR : Logical OR

ORR<suffix> <dest>, <op 1>, <op 2>

$$\text{dest} = \text{op\_1 OR op\_2}$$

OR effectue un OU logique entre deux opérandes, et place le résultat dans le registre de destination. Cette opération permet de placer des bits à 1.

ORR R0, R0, #3	; Set bits zero and one in R0
----------------	-------------------------------

An OR table (result = either or both):

Op_1	Op_2	Result
------	------	--------

0	0	0
0	1	1
1	0	1
1	1	1

## RSB : Reverse Subtraction

RSB<suffix> <dest>, <op 1>, <op 2>

$$\text{dest} = \text{op\_2} - \text{op\_1}$$

RSB soustrait l'opérande 1 de l'opérande 2 et place le résultat dans le registre de destination.

```
RSB R0, R1, R2      ; R0 = R2 - R1
RSB R0, R1, #256    ; R0 = 256 - R1
RSB R0, R2, R3,LSL#1 ; R0 = (R3 << 1) - R2
```

## RSC : Reverse Subtraction with Carry

RSC<suffix> <dest>, <op 1>, <op 2>

$$\text{dest} = \text{op\_2} - \text{op\_1} - \text{!carry}$$

RSC soustrait l'opérande 1 et le bit de retenue de l'opérande 2, et place le résultat dans le registre de destination.

## SBC : Subtraction with Carry

SBC<suffix> <dest>, <op 1>, <op 2>

$$\text{dest} = \text{op\_1} - \text{op\_2} - \text{!carry}$$

SBC soustrait l'opérande 2 et le bit de retenue de l'opérande 1 et place le résultat dans le registre de destination.

## SUB : Subtraction

SUB<suffix> <dest>, <op 1>, <op 2>

$$\text{dest} = \text{op\_1} - \text{op\_2}$$

SUB soustrait l'opérande 2 de l'opérande 1 et place le résultat dans le registre de destination.

```
SUB R0, R1, R2      ; R0 = R1 - R2
SUB R0, R1, #256    ; R0 = R1 - 256
SUB R0, R2, R3,LSL#1 ; R0 = R2 - (R3 << 1)
```

## SWP : Swap

SWP<suffix> <dest>, <op 1>, [<op 2>]

SWP échange le contenu d'un registre et d'une case mémoire. L'opérande 2 indique l'adresse de cette case mémoire. Le suffixe B permet d'effectuer un échange d'un octet, sinon, l'échange s'effectue sur un mot.

## *Les instructions de comparaison*

### CMN : Compare Negative

CMN<suffix> <op 1>, <op 2>

$$\text{status} = \text{op\_1} - (-\text{op\_2})$$

CMN compare les 2 opérandes en inversant le signe de l'opérande 2, ce qui permet de faire des comparaisons avec des chiffres négatifs.

CMN R0, #1 ; Compare R0 with -1

### CMP : Compare

CMP<suffix> <op 1>, <op 2>

$$\text{status} = \text{op\_1} - \text{op\_2}$$

CMP compare le contenu de deux registres ou d'un registre et d'une valeur immédiate, et positionne en conséquence les drapeaux d'état (dans CPSR), qui sont en place pour d'éventuelles instructions conditionnelles.

Cette instruction exécute une soustraction, mais sans ranger le résultat.

Les drapeaux se réfèrent à l'opérande 1 devant l'opérande 2, par exemple, le suffixe GT (Greater Than) sera exécuté si l'opérande 1 est plus grand que l'opérande 2.

Pour cette instruction, le suffixe S n'est pas nécessaire. Il est ignoré si on le place.

CMP R0,#3 ; Compare R0 with 3

## TEQ : Test Equivalence

TEQ<suffix> <op 1>, <op 2>

Status = op\_1 EOR op\_2

TEQ est similaire à TST, mais utilise un OU exclusive au lieu d'un ET logique. Cette instruction permet de vérifier l'égalité de 2 opérandes sans positionner le bit de retenue (C) contrairement à l'instruction CMP.

Le suffixe S n'est pas utile.

## TST : Test bits

TST<suffix> <op 1>, <op 2>

Status = op\_1 AND op\_2

TST ne produit pas de résultat, mais positionne les drapeaux du registre d'état. TST permet de tester si un bit particulier est à 1. L'opérande 1 contient le mot à testet, l'opérande 2 est un masque qui indique les bits dont on veut tester s'ils sont à 1.

Le suffixe S n'est pas utile.

TST R0, #1 ; Test if bit zero is set in R0

## *Les instructions de branchement*

### B : Branch

B<suffix> <address>

B est l'instruction de branchement simple. Le processeur « saute » immédiatement à l'adresse spécifiée, et continue l'exécution à cet endroit.

L'offset est alors spécifié en relatif par rapport à l'endroit courant. (24 bits).

Comme les autres, cette instruction se décline avec les conditions (branchements conditionnels).

### BL : Branch with Link

BL<suffix> <address>

BL est l'instruction de branchement vers un sous programme. Dans ce cas, le contenu de R15 (PC) qui référence l'adresse de l'instruction suivante, est copié dans R14 (LR) le registre de lien.

BL switch\_screen\_mode

BL get\_screen\_info

BL load\_palette

## ***Les instructions de chargement / rangement***

### **Single Data Transfer STR - LDR**

Les instructions de transfert simple (STR et LDR) permettent d'échanger des octets, des mots, avec la mémoire.

```
LDR R0, address
STR R0, address
LDRB R0, address
STRB R0, address
```

Ces instructions chargent ou rangent le contenu du registre R0 à l'adresse spécifiée. Le suffixe B indique un échange octet et le suffixe H un échange sur 16 bits.

L'adresse peut être une valeur, un offset, un offset décalé.

STR R0, [Rbase] ; Store R0 at Rbase.

STR R0, [Rbase, Rindex] ; Store R0 at Rbase + Rindex.

STR R0, [Rbase, #index] ; Store R0 at Rbase + index. Index is an immediate value.  
STR R0, [R1, #16] ;would load R0 from R1+16.

STR R0, [Rbase, Rindex]! ; Store R0 at Rbase + Rindex, & write back new address to  
Rbase.

STR R0, [Rbase, #index]! ; Store R0 at Rbase + index, & write back new address to  
; Rbase.

STR R0, [Rbase], Rindex Store R0 at Rbase, & write back Rbase + Rindex to Rbase.

STR R0, [Rbase, Rindex, LSL #2] will store R0 at the address Rbase + (Rindex \* 4)

STR R0, place Will generate a PC-relative offset to 'place', and store R0 there.

Ces instructions supportent les conditions. Dans ce cas, la condition est écrite devant l'indication de taille du transfert :

```
LDREQB Rx, address
```

Le ! indique si l'offset est mis à jour ou non.

LDREQB R0, [R1, #-1] !

## Multiple Data Transfer LDRM STRM

Les instructions LDRM et STRM sont utilisées pour faire des échanges multiples avec la mémoire. Ces instructions permettent de sauvegarder / récupérer les registres qui doivent être sauvegardés dans la pile.

```
STMFD R13!, {R0-R12, R14}.
```

LDMED	LDMIB	Pre-incremental load
LDMFD	LDMIA	Post-incremental load
LDMEA	LDMDB	Pre-decremental load
LDMFA	LDMDA	Post-decremental load
STMFA	STMIB	Pre-incremental store
STMEA	STMIA	Post-incremental store
STMFD	STMDB	Pre-decremental store
STMED	STMDA	Post-decremental store

L'ordre des registres lors de l'écriture n'a pas d'effet.

```
STMFD R13!, {R0, R1}
LDMFD R13!, {R1, R0}
```

Par exemple, ces 2 instructions n'échangent pas le contenu des registres R0 et R1.

L'instruction suivante sauvegarde les registres R0 à R12 dans la pile (La pile est la mémoire pointée par R13).

Le registre R14 qui contient l'adresse de retour du programme actuel est également sauvegardé.

```
STMFD R13!, {R0-R12, R14}
```

L'instruction suivante récupère le contenu des registres et revient directement à la fonction appelante : le PC prend la valeur l'adresse de retour.

```
LDMFD R13!, {R0-R12, PC}
```

## ***Instructions de multiplication***

Ces instructions sont différentes des instructions arithmétiques et logiques classiques par leurs restrictions sur les opérandes :

Tous les opérandes doivent être des registres. Il n'est pas possible d'utiliser une valeur immédiate ou un décalage sur l'opérande 2.

Le registre de destination doit être différent des registres sources.

R15 ne peut pas être un registre de destination.

### **MLA : Multiplication with Accumulate**

MLA<suffix> <dest>, <op 1>, <op 2>, <op 3>

$$\text{dest} = (\text{op\_1} * \text{op\_2}) + \text{op\_3}$$

### **MUL : Multiplication**

MUL<suffix> <dest>, <op 1>, <op 2>

$$\text{dest} = \text{op\_1} * \text{op\_2}$$

MUL effectue une multiplication 32 bits entre deux nombres signés.

```
MUL R2, R0, R1
MOV R0, R2
```

## ***Décalages et rotations***

Les différentes opérations permettent de spécifier une opération de décalage ou de rotation sur l'opérande 2.

Les différentes possibilités sont :

LSL	Logical Shift Left
ASL	Arithmetic Shift Left
LSR	Logical Shift Right
ASR	Arithmetic Shift Right
ROR	Rotate Right
RRX	Rotate Right with Extend (Carry is inserted)

ASL et LSL ont le même effet et peuvent être intervertis.

## Suffixes conditionnels

**EQ** : Equal (par défaut = 0) - **NE** : Non Equal - **GT** : Greater Than (signé) – **LE** : Less or Equal (signé)  
**GE** : greater or Equal (Signé) – **LT** : Less than (signé) – **HI** : Higher (non signé) – **LS** : Lower or same (non signé) – **CS/HS** : Carry set / higher or same (non signé) – **CC/LO** : Carry Clear / Lower (non signé)  
**PL** : Positif or 0 – **VS** : Overflow (V set) – **VC** : non overflow (V clear) – **AL** : always – **NV** : never.

Exemples :

1)

MOV	R0,R1	; R0 ← R1
MOVEQ	R0,#1	; Si r0 = 0 alors R0 ← 1

2)

	mov	r0,#10	
Bcl	sub	r0,r0,#1	; r0 ← r0-1
	Bne	bcl	; Boucle si (tant que) r0 ≠ 0
		...	

## Les pseudos instructions

Ces pseudo-instructions sont des consignes données à l'assembleur.

### ADR : Load Address

ADRL<suffix> <register>, <label>

Cette pseudo instruction permet d'affecter une adresse (un pointeur) à un registre.

ADR	R6,loop	
Mov	r0,3	
Loop	subs	r0,#1
	Bne	R6
		; R6 pointe sur l'adresse référencée par loop

### LDR : Load register

LDR <suffix> <register>,<value>

Cette pseudo instruction permet d'initialiser un registre avec une valeur immédiate sans se soucier de la taille de cette valeur. Elle est traduite selon le cas par une instruction MOV ou LDR avec une indirection.

LDR R0,=3 ; R0 ← 3

LDR R0,=0x12345678 ; R0 ← 0x12345678

**ORG : Origin**

```
ORG <Address>
```

Cette pseudo instruction permet de spécifier une adresse d'implantation pour le code qui suit.

**EQU : Equal**

```
<symbol> EQU <value>
```

Cette pseudo instruction permet de définir des symboles pour la suite du programme.

```
ROUGE equ 0x12
```

```
Mov R0,#ROUGE ; R0 ← 0x12
```

**DC : Define Constant**

DCB (8 bits) – DCW (16 bits) – DCD (32 bits) – DCS (string – taille suivant le cas).

```
DC <value1>,<value2>....
```

Cette pseudo instruction permet de placer des valeurs en mémoire.

```
BUF DCS « Hello world »,10,13,0
ASC DCB 0x41,0x42,0x43
```

**NOP : No Operation**

```
NOP
```

Cette pseudo instruction permet de passer un cycle à ne rien faire. On peut l'implanter de différentes façons, par exemple :

```
MOV R0,R0
```

**OPT : Set Assembler Option**

```
OPT <Value>
```

Cette pseudo instruction permet de spécifier des options vers le programme d'assemblage.

## **ALIGN : Align pointers**

ALIGN

Cette pseudo instruction permet d'aligner les adresses sur un modulo 4.

Sur le système utilisé en TP, on utilise en fait les pseudo instructions :

ALIGNROM x

Ou

ALIGNRAM x

Où x est la puissance de 2 sur laquelle on aligne.

Exemple :

Alignrom 2