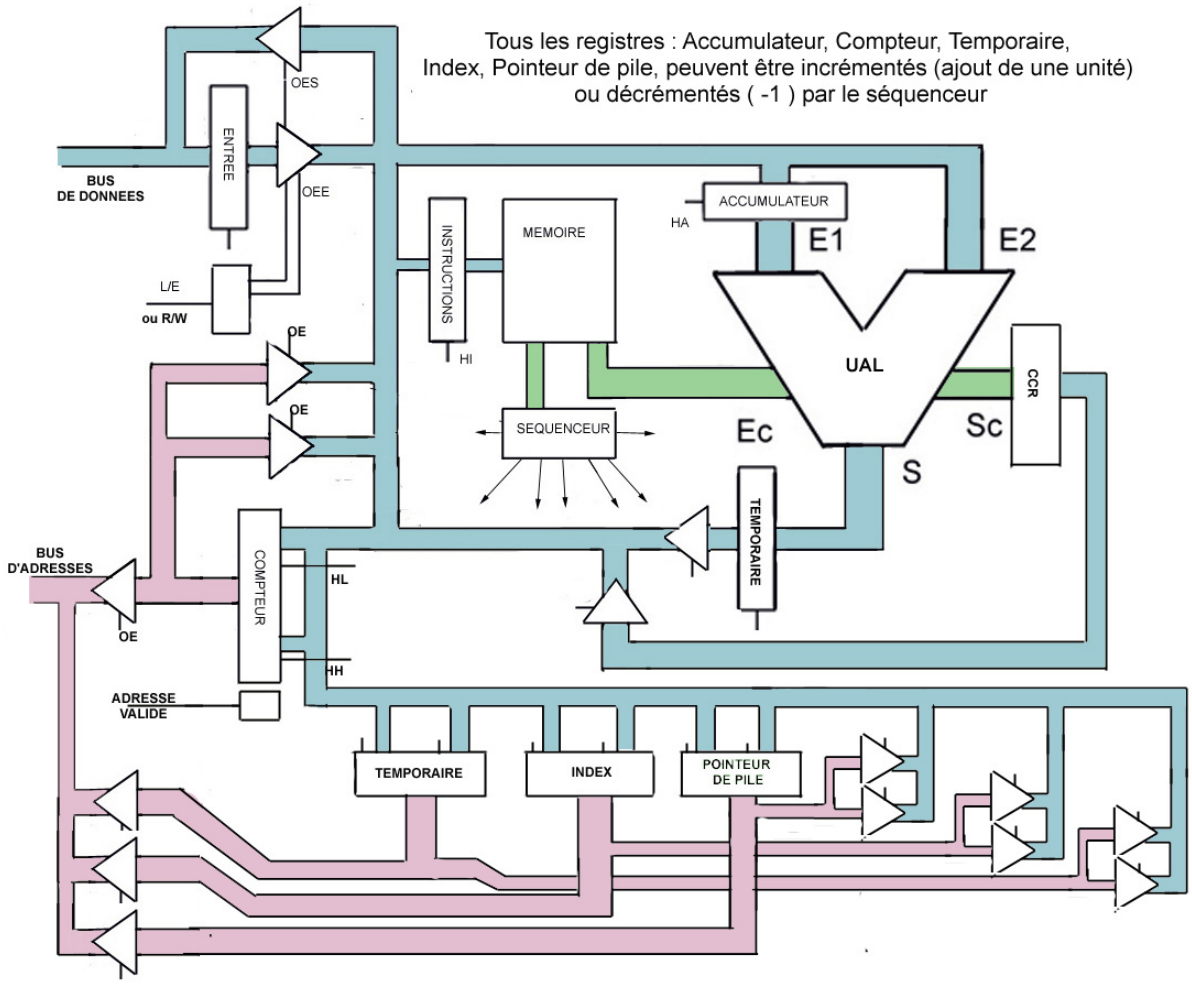
NOTION DE PROGRAMMATION

1/ Complétons notre microprocesseur

Nous avons , dans les leçons précédentes décrit un microprocesseur théorique, cependant il s'inspire du 6800, premier microprocesseur conçu chez Motorola. Le microprocesseur 6800 en fait comporte deux accumulateurs A et B , ces accumulateurs sont connectés l'un et l'autre au bus de données et à la même entrée de l'UAL , la quasi totalité des instructions peut s'exécuter avec l'un ou l'autre. Nous nous contenterons d'un accu. Le 6800 possède également deux registres de 16 bits l'index et le pointeur de pile. L'index est utilisé dans les instructions courantes nous allons l'étudier dans cette leçon. Le pointeur de pile est un registre utilisé dans la gestion des interruptions. Nous étudierons les interruptions ultérieurement.



Les trois registres temporaire, index, pointeur de pile peuvent se substituer au compteur sur le bus d'adresses ils peuvent être lus par l' UAL et chargés octet par octet.

2/ Structure d'une instruction

Nous allons étudier les instructions en **langage assembleur** . L'assembleur est un logiciel qui va transformer les **mnémoniques** qui sont des abréviations conventionnelles à la structure rigoureuse en **codes machine** composés uniquement de nombres binaires de 8 bits. Par exemple pour le 6800 :

2A/ charger l'accumulateur A avec la valeur hexadécimale 2B s'écrit en assembleur :

LDAA #0X2B

LDAA est le mnémonique de LOAD Accumulator A
signifie que le nombre qui suit est l'opérande (opérande est du genre masculin on dit un opérande), c'est à dire que c'est le nombre à charger dans l'accumulateur
0X signifie que le nombre qui suit est en hexadécimal
2B est le nombre hexadécimal à charger

Soumis au logiciel assembleur l'instruction devient 86 2B.
86 est le **code machine** de l'instruction, 2B est l'opérande. L'instruction comporte deux octets le microprocesseur devra faire deux lectures mémoire , le bus de données est de 8 bits, avant d'exécuter le travail

2B/ charger l'accumulateur A avec l'octet situé à l'adresse 0X2000

LDAA 0X2000

L'assembleur va nous donner

B6 20 00

B6 est le code machine de l'instruction dans ce mode d'adressage 20 00 l'adresse de l'opérande . L'instruction comporte trois octets en mémoire, le microprocesseur va faire 3 opérations de lecture mémoire pour acquérir la totalité de l'instruction , il va charger l'adresse 2000 dans son registre temporaire placé celui-ci sur le bus d'adresse et va faire une opération de lecture de cette adresse puis charger l'opérande ainsi acquise dans l'accumulateur

3/ Les modes d'adressage

On appelle adressage la technique qui va permettre soit de trouver la donnée soit sa destination. Passons en revue les différents modes de base:

Adressage implicite : L'instruction contient implicitement l'adresse par exemple mettre à 0 l'accumulateur A

Instruction en assembleur	CLRA	Clear Accumulateur A
code machine	4F	01001111

Adressage immédiat Le mnémonique de l'instruction est suivi de l'opérande précédé d'un dièse (#) . Exemple : charger l'index avec le nombre 0X1000

Instruction en assembleur	LDX #0X1000	Load index avec 0X1000
Code machine	CE 1000	11001110 00010000 00000000

Adressage étendu Le mnémonique de l'instruction est suivi de l'adresse de l'opérande . Exemple : charger l'index avec le nombre situé à l'adresse 0X1000 (et 1001) puisque l'index est un registre 16 bits

Instruction en assembleur	LDX 0X1000	Load index avec(0X1000)
Code machine	FE 1000	11111110 00010000 00000000

Adressage indexé Le mnémonique de l'instruction est suivi d'un nombre qui additionné au contenu de l'index génèrera l'adresse de l'opérande . Exemple : charger l'index avec le nombre situé à l'adresse 0X102B sachant que l'index contient le nombre 0X1000

Instruction en assembleur	LDX 0X2B	Load index avec(index) + 0X2B
Code machine	EE 2B	11101110 00010000 00000000

Adressage relatif Ce mode d'adressage n'est utilisé que dans les instructions de saut dans la mesure ou l'amplitude du saut est comprise **dans la fourchette –128 +127** . Ainsi lorsque le microprocesseur doit sauter plusieurs instruction dans un programme, on utilise les instructions de branchement ou de branchement conditionnel. Généralement on utilise les étiquettes nous verrons cette technique ultérieurement. Nous allons ici supposer un saut de 0X10 adresses à effectuer lorsque l'instruction a donné un résultat nul (Z=1)

Instruction en assembleur	BEQ 0X10	Si le bit Z = 1 ajouter 0X10 au contenu du compteur de programme
Code machine	27 10	00100111 00010000

Suivant le microprocesseur que l'on est appelé à utiliser, on retrouvera ces modes d'adressage avec des appellations qui pourront être différentes.

4/ Notion de programme

Exemple :

Réalisation d'une temporisation

La technique consiste à charger une ou plusieurs cases mémoire ou un registre et de le décrémenter jusqu'à ce qu'il arrive à 0 . Pour cela on va réaliser une boucle utilisons l'index comme registre à décrémenter

Ecriture du programme :

```
Tempo :   LDX     #0XFFFF   Chargement de l'index avec FFFF
Dec :     DEX               Contenu de l'index (index) moins 1  résultat dans l'index
          BNE     Dec       Si (index) ≠ 0 retourner à Dec
          Suite du programme
```

Etude du programme :

1ère instruction

Tempo est une étiquette (chaîne de caractères), l'assembleur va repérer l'adresse de l'instruction qui suit, il établit une table de correspondance étiquettes/adresse. Toute référence à Tempo dans le programme sera remplacée par l'adresse de l'instruction par l'assembleur

En suite nous trouvons LDX 0XFFFF et dans le champ commentaire « Chargement de l'index avec FFFF » , ce champ n'est pas interprété par l'assembleur

2ème instruction

Dec nouvelle étiquette suivie de DEX et du champ commentaire « Contenu de l'index (index) moins 1 résultat dans l'index

3ème instruction

BNE Dec si le bit Z=0 remonter à l'adresse Dec si Z=1 poursuivre le programme à l'instruction suivante. L'assembleur va calculer le saut à effectuer comme ce saut doit être négatif, l'amplitude du saut sera exprimé en complément à 2.

Le microprocesseur va donc effectuer 65536 fois la boucle DEX, BNE pour évaluer le temps que va mettre le microprocesseur pour sortir de la boucle, il nous faut connaître le nombre de microcycles pour chacune des instructions. Il faut consulter la documentation du microprocesseur utilisé. La documentation du 6800 nous donne :

```
LDX 0XFFFF  3 ~
DEX         4 ~
BNE Dec     4 ~
```

La durée de la temporisation en microcycles sera :

3 + [(4+4)*65536]=524291 microcycles

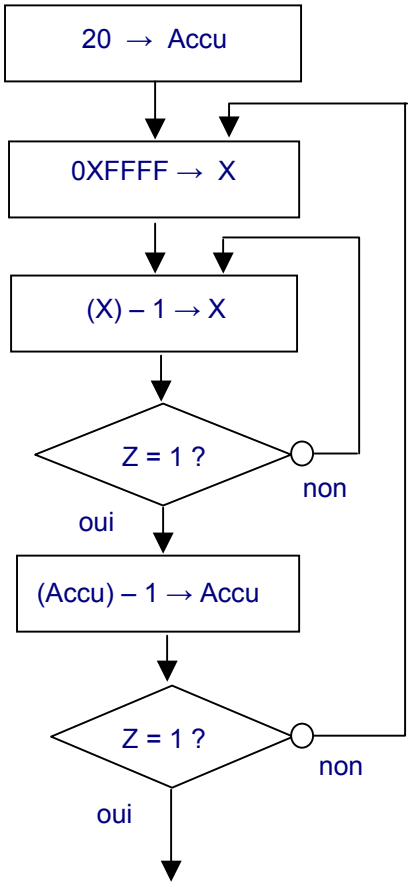
en prenant 1µs comme durée d'un microcycle, la temporisation sera de 524 291 µs soit environ 0,5 seconde. Si nous voulons une temporisation de 10 secondes nous devrons l'exécuter 20 fois

Temporisation de 10 secondes

```
Debut :   LDAA    # 0X14      mettre 20 dans l'accu A
Tempo     LDX     # 0XFFFF    mettre 65535 dans l'index
Dec       DEX                decrémentation de l'index
          BNE     Dec        Si l'index n'est pas vide décrémenter à nouveau
          DECA               Si l'index est vide décrémenter l'accu
          BNE     Tempo      Si l'acccu n'est pas vide recharger l'index
          Suite
```

Pour concevoir un programme, il est possible de mettre ses idées sous forme d'un organigramme ; L'organigramme présente l'avantage d'être clair à comprendre et permettra une transposition plus facile si la mise en œuvre doit se faire avec un autre microprocesseur que celui prévu au départ.

Organigramme



L'organigramme ci-dessus représente notre temporisation de 10 secondes étudiée précédemment. Les losanges représentent les **instructions conditionnelles** une question est posée (Z=1 ?) lorsque la réponse est négative un **branchement** est effectué. Dans le programme ci-dessus il y a deux boucles l'une à l'intérieur de l'autre. La question posée dans le losange est la **condition de sortie de boucle**.

Petit complément sur l'étiquette

Une étiquette permet de concevoir un programme sans savoir à quelle adresse il sera placé. Si l'on place une étiquette en début de programme, on peut provisoirement donner une équivalence étiquette/adresse pour un assemblage provisoire et tester le programme sur un système de développement. Le programme une fois mis au point pourra être incorporé dans un programme plus vaste et réassemblé. Les étiquettes permettent également de ne pas avoir à calculer l'amplitude des saut dans les branchements, cela rend facile l'introduction ou la suppression d'instructions dans la boucle.

5/ Les instructions

Le jeu d'instructions d'un microprocesseur comporte toujours les mêmes instructions de base . Le 6800 offre 107 mnémoniques , le 6809 en propose 138, mais en fait il y a peu d'instructions vraiment nouvelle en effet dans le 6800 possède deux accumulateurs A et B on trouvera les mnémoniques de chargement LDAA et LDAB , dans le 6809 les deux accumulateurs A et B peuvent se concaténer en un seul accumulateur 16 bits qui s'appelle alors D il y a donc trois mnémoniques de chargement LDA, LDB, LDD. Cependant de nouvelles instructions vont commencer à faciliter la vie des programmeurs notamment la multiplication . Il faudra attendre le 68000 pour la division . Le 68000 va apporter une grande souplesse au programmeur puisqu'il va disposer de 8 registres de données équivalents des accumulateurs mais d'une capacité de 32 bits. Et un jeu de 125 instructions Le 68020 environ 150 etc

Il est évident que plus le jeu d'instruction est important et plus les instructions sont complexes plus le décodages des instructions et le séquenceur seront complexe ce qui sera au détriment de la vitesse d'exécution. Pour accroître la rapidité les concepteurs ont réduit le jeu d'instructions de certains microprocesseurs , ce sont les microprocesseurs **RISC** (Reducts Instruction Set Computer) par opposition à **CISC** (Complex Instruction Set Computer) .

Les familles d'instructions :

Les instructions de chargement

Ce sont les « **LOAD** » ou les « **MOVE** »on les trouve dans tous les microprocesseurs ils s'appliquent généralement à tous les registres du modèle de programmation (ou presque) **Le modèle de programmation** est l'ensemble des registres accessibles au programmeur.

Egalement les mise à 0 Clear « **CLR** »

Les instructions arithmétiques

On y trouve les additions « **ADD** » et les soustractions « **SUB** » l'exécutions de ces instructions généralement influencent les bits du registre CCR qu'on appelle aussi registre d'état tels que C (report ou carry) Z (zéro) N (Négatif) V(dépassement ou overflow) et d'autres suivant les microprocesseurs

Les instructions d'incrémentation « **INC** » augmenter le nombre d'une unité ou de décrémentation « **DEC** » réduire d'une unité

Les instructions de comparaison « **CMP** » qui sont équivalentes à une soustraction mais le résultat ne fait que positionner les bits de CCR, l'octet n'est pas modifié

Faire le complément à 2 d'un nombre l'instruction « **NEG** »

Plus rarement les multiplications « **MUL** » et les divisions « **DIV** » en binaire signé et non signé

Les instruction logiques

Le ET logiques « **AND** » OU logiques « **OR** » les OU Exclusifs « **EOR** » ou bien « **XOR** »

Le ET logique peut servir à remettre à 0 un bit sur un octet

Exemple un octet en mémoire 1010 0111 pour remettre à 0 le deuxième bit on pourra faire un ET logique avec 1111 1101

	1010 0111
et	1111 1101
donne	1010 0101

Le OU logique peut servir à mettre à 1 un bit dans un octet

Exemple un octet en mémoire 1010 0111 pour remettre à 1 le quatrième bit on pourra faire un OU logique avec 0000 1000

	1010 0111
et	0000 1000
donne	1010 1111

Le OU Exclusif peut être du même usage que la soustraction dans certains cas mais en plus , appliqué deux fois de suite au même octet il s'annule

Exemple on s'interroge , un octet en mémoire est-il égal à 1010 0111, supposons que cet octet soit 1010 1111

	1010 1111	
ou ex	1010 0111	
donne	0000 1000	une égalité aurait donné 0000 0000 et Z=1
ou ex	1010 0111	
donne	1010 1111	l'octet d'origine est rétabli

Les instructions de décalage

Généralement on trouvera deux familles , les décalages Logiques et Arithmétiques

Décalage logique, le nombre concerné est décalé de 1 ou N rangs à droite Logical Shift Right « **LSR** » des 0 sont introduits à gauche généralement le bit éjecté à droite passe dans le bit C

Exemple :

1100 0011			
faire	0 →	1100 0011 →	C
résultat		0110 0001	1

Décalage logique, le nombre concerné est décalé de 1 ou N rangs à gauche Logical Shift Left « **LSL** » des 0 sont introduits à droite généralement le bit éjecté à gauche passe dans le bit C

Exemple :

1100 0011			
faire	C ←	1100 0011 ←	0
résultat	1	1000 0110	

Décalage arithmétique. Les décalages peuvent être utilisés pour faire des multiplications ou des divisions

Le nombre	0000 0110	soit 6 en base 10
Décalé à droite	0000 0011	soit 3 en base 10
Décalé à gauche	0000 1100	soit 12 en base 10

Le décalage marche bien mais :

Le nombre	1100 0000	soit 192 en base 10 ou –64 en binaire signé (*)
Décalé à droite	0110 0000	soit 96 en base 10 ou +96 en binaire signé

En binaire signé ce décalage à droite ne marche pas

Le décalage arithmétique **ASR** Arithmetic Shift Right. Le décalage arithmétique à droite au lieu d'introduire un 0 lors du décalage introduit le même bit que celui qui est décalé

Exemple

Le nombre	1100 0000	–64 en binaire signé
Décalé à droite	1110 0000	–32 en binaire signé

(*) Rappel :

Un nombre négatif tel que 1100 0000, pour en connaître la valeur on en fait le complément à 2 et on obtient la valeur positive . Pour faire le complément à 2 on inverse tous les bits et on ajoute 1

Ainsi	1100 0000
Bits inversés	0011 1111
Plus 1	0100 0000 soit 64 en base 10 le nombre d'origine est donc – 64

Les rotations à droite ou à gauche **ROR** ou **ROL** le nombre est décalé à droite le bit C devient le bit de poids fort et le bit de poids faible est éjecté dans C. A gauche le bit C passe dans le bit de poids faible, et le bit de poids fort passe dans C

Les branchements et sauts

Dans le cas Motorola les sauts ou jump (**JMP**) sont inconditionnels et ne se font pas en adressage relatif (déplacement par rapport au compteur de programme) les branchements se font en adressage relatif le mnémonique est accompagné du nombre (positif ou négatif) qui sera additionné au compteur de programme pour pointer la nouvelle instruction. Ils peuvent être inconditionnels « **BRA** » Branch Always ou conditionnés par l'état d'un bit de CCR par exemple Bcc branch if carry clear (C=0) ou Bcs Branch if carry set (C=1)

D'autre instructions peuvent concerner les sous-programmes et la gestion de la pile, ce sera l'objet des leçons suivantes