Micro-contrôleur Motorola 68HC11

Plan du cours:

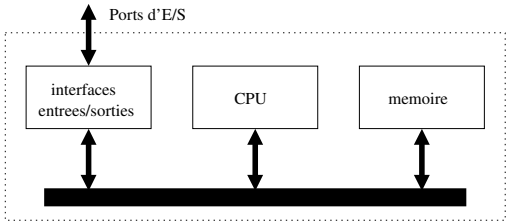
- I-Introduction - Architecture d'un micro-contrôleur
- II-Codage des informations
- III-Structure du 68HC11
- IV-Jeu d'instructions
- V-Entrées/Sorties
- VI-Système de développement. Platine Controlboy

I-Introduction

I-1 Architecture d'un micro-contrôleur

Un micro-contrôleur intègre sur un même circuit:

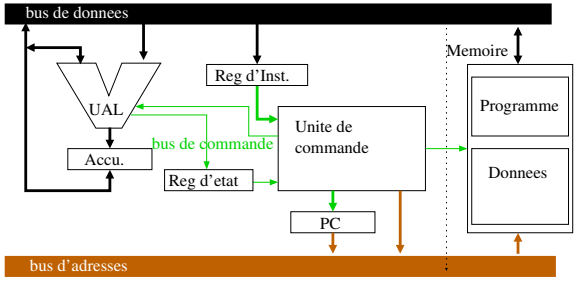
- Une unité centrale ou micro-processeur (CPU)
- De la mémoire (RAM, ROM, EPROM, EEPROM,...)
- Des interfaces d'entrées/sorties



I-2 Unité centrale

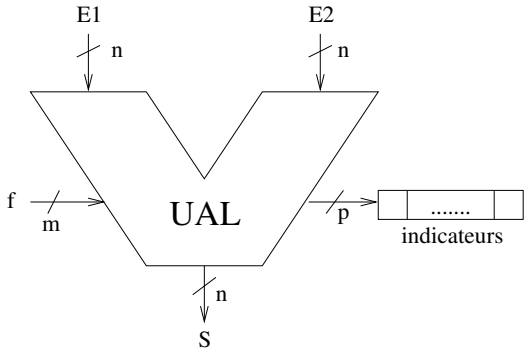
Une unité centrale est constituée:

- D'une unité arithmétique et logique (UAL).
- De registres internes
- D'une unité de commande (décodage et contrôle de l'exécution)
- De bus (adresses, données, commandes)



Unité Arithmétique et logique

Au coeur du processeur, l'UAL effectue le traitement des informations.



C'est un circuit combinatoire qui produit un résultat (S) sur n bits fonction des données présentes sur ses entrées (E1 et E2) et de la fonction à réaliser (f) et met à jour des indicateurs.

I-3 Mémoire

Principe: organisée en octet. Chaque octet de la mémoire possède une adresse codée sur m bits.

Espace adressable: Nombre d'adresses disponibles égal à 2^m.

➳ Mémoire vive: (Utilisée pour les données)

RAM, DRAM, SRAM,...: Utilisable en lecture et écriture. Volatile.

➳ Mémoires mortes: (utilisée pour les programmes)

ROM, PROM: mémoire programmée par le fabricant (ROM) où par l'utilisateur mais une seule fois (PROM). Non volatile. Accessible seulement en lecture.

EPROM: reprogrammable par exposition à des UV. Non-volatile. Accessible en lecture seulement.

EEPROM: effaçable électriquement. Non-volatile. Rapide en lecture mais lente en écriture.

II-Codage des informations

II-1 Introduction

Un ordinateur ne manipule que des informations binaires.

⇨ Nécessité d'un codage des informations. Codage des nombres, codage des caractères.

Unités utilisées dans un ordinateur: bit, octet (8 bits), multiples d'octets (16 bits, 32 bits, 64 bits, ...)

Codage: convention qui permet de représenter une information sous forme d'une suite de 0 et de 1.

Notation hexadécimale: permet de représenter une valeur binaire de manière compacte.

I-4 Interfaces d'E/S intégrées

- ⇨ interfaces parallèles pour la connection des E/S (signaux logiques).
- ⇨ interfaces série pour la communication.
- ⇨ convertisseurs analogique/numérique pour le traitement de signaux analogiques.
- ⇨ timers pour générer ou mesurer des signaux périodiques (signaux rectangulaires).

Accessibles sur des broches du circuit (ports).

Des registres spécialisés situés dans l'espace adressable (pas d'instructions spéciales) permettent de lire où d'écrire des valeurs sur les ports et de programmer les interfaces (par exemple la cadence de transmission de la ligne série).

II-2 Numération positionnelle

Principe: Une quantité est représenté par une suite de symboles. Le poids de chacun des symboles dépend de sa position dans la suite.

➳ **base 10 (décimale)** 10 Symboles: $\{$ 0, 1, 2, ... , 9$\}$

Poids: puissances de 10 en partant de 0, de la gauche vers la droite.

Exemple: 127$= 1 \times 10^2 + 2 \times 10^1 + 7 \times 10^0$

➳ **base 2 (binaire)** 2 symboles $\{0, 1\}$

Poids: puissances de 2 en partant de 0, de la gauche vers la droite.

Exemple: 1011b$= 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$ (soit 11 en décimal)

➳ **base 16 (hexadécimale)** 16 symboles $\{$0,...,9 , 1, A,..., F$\}$

Poids: puissance de 16. Exemple: 3Ah$= 3 \times 16^1 + 10 \times 16^0$ (soit 58 en décimal)

II-3 Changement de base

- **conversion binaire ➟ hexadécimal:**
  on regroupe les bits par groupes de 4 et on cherche le symbole hexadécimal de chaque groupe.
  Ex: 0010 1100b ➟ 2Ch
- **conversion hexadécimal ➟ binaire:**
  il faut décomposer chaque symbole hexadécimal en groupe de 4 bits.
  Ex: 2Ch ➟ 0010 1100b
- **conversion binaire ➟ décimal:**
  chaque bit doit être multiplié par son poids (puissance de 2).
  Ex: 0010 1100b ➟ $2^5 + 2^3 + 2^2$ =44d

II-4 Entiers naturels

Soit A=$a_{n-1}a_{n-2}...a_2a_1a_0$ un mot-code de n bits en binaire.

Ce mot-code A représente l'entier naturel codé sur n bits:

$$N(A) = \sum_{i=0}^{n-1} a_i 2^i$$

Dans un ordinateur n est un multiple de 8 bits.

n	plage décimal	hexadécimal
8	0 à 255	00h à FFh
16	0 à 65535	0000h à FFFFh
24	0 à 16 777 215	000000 à FFFFFF

- **conversion décimal ➟ binaire:**
  On effectue des divisions successives par 2
  Ex: 44/2=22, r=b0=0, 22/2=11, r=b1=0, 11/2=5, r=b2=1, 5/2=2, r=b3=1, 2/2=1, r=b4=0, 1/2=0, r=b5=1 donc 44d ➟ b5b4b3b2b1b0=101100b
- **conversion hexadécimal ➟ décimal:**
  chaque symbole doit être multiplié par son poids (puissance de 16).
  Ex: 2Ch ➟ $2 \times 16 + 12$ =44d
- **conversion décimal ➟ hexadécimal:**
  On effectue des divisions successives par 16.
  Ex: 44/16=2, r=12, 2/16=0, r=2 donc 44d ➟ 2Ch

- **Décodage** d'un entier naturel: soit A le mot-code représentant l'entier $N(A)$.
  La valeur décimale de $N(A)$ est obtenue en appliquant la définition:
  Exemple (codage sur 8 bits): A=0010 1011
  $N(A) = 2^0 + 2^1 + 2^3 + 2^5 = 1 + 2 + 8 + 32 = 43$
- **Codage:** soit l'entier naturel $N(A)$. Son code A est obtenu par des divisions successives par 2:

Exemple: $N(A) = 43$

valeur	quotient	reste	bit
43	21	1	b0
21	10	1	b1
10	5	0	b2
5	2	1	b3
2	1	0	b4
1	0	1	b5

D'ou: A=0010 1011

II-5 Entiers relatifs

Principe: codage en complément à 2. Le bit le plus à gauche (bit le plus significatif) a un poids négatif. Soit Z l'entier relatif codé par A:

$$Z(A) = -a^{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i$$

Conséquence: si $a_{n-1} = 1$, $Z(A)$ est un nombre négatif quel que soient les valeurs des autres bits (a_0 à a_{n-2}). Inversement, si $a_{n-1} = 0$, $Z(A)$ est positif.

⇨ Le signe d'un entier relatif est donné par la valeur du bit le plus significatif de son code (0: positif, 1: négatif)

Exemple: 1001 1111b représente un entier négatif.

Rêgles de codage des entiers relatifs:

Soit Z un entier relatif à coder, soit A son code.

Si Z≥0, le code de Z est le même que celui de l'entier naturel N=Z.

Si Z<0, il faut chercher le complément à 2 du code de sa valeur absolue (codée comme un entier naturel).

A	N(A)	$\widetilde{N}$(A)	Z(A)	A	N(A)	$\widetilde{N}$(A)	Z(A)
0000	0	16	0	1000	8	8	-8
0001	1	15	1	1001	9	7	-7
0010	2	14	2	1010	10	6	-6
0011	3	13	3	1011	11	5	-5
0100	4	12	4	1100	12	4	-4
0101	5	11	5	1101	13	3	-3
0110	6	10	6	1110	14	2	-2
0111	7	9	7	1111	15	1	-1

le complément à 2 d'un nombre est obtenu en rajoutant 1 à son complément vrai.

Complément à 2

soit $\widetilde{X}$ le complément à 2 de X, par définition, $\widetilde{X}$ est tel que $\widetilde{X} + X = 2^n$.

On peut encore écrire $Z(A)$ sous la forme:

$$Z(A) = -a^{n-1}2^n + \sum_{i=0}^{n-1} a_i 2^i = -a^{n-1}2^n + N(A)$$

D'ou, si $a_{n-1} = 0$, $Z(A) = N(A)$.

⇨ Le code d'un entier positif $Z(A)$ est le même que celui de l'entier naturel $N(A)$

Si, par contre, $a_{n-1} = 1$, $Z(A) < 0$ et $|Z(A)| = 2^n - N(A) = \widetilde{N}(A)$.

⇨ Le code d'un entier négatif $Z(A)$ est le même que celui du complément à 2 de l'entier naturel égal à sa valeur absolue.

II-6 Addition et débordement

Les valeurs représentables dépendent du nombre de bits choisi pour le codage. Le processeur doit détecter si une le résultat d'une opération est représentable ou pas.

Soit S= $s_{n-1}...s_0$ le code du résultat de l'addition sur n bits de A= $a_{n-1}...a_0$ avec B= $b_{n-1}...b_0$. L'exactitude du résultat dépend du contexte:
non signé⟶indicateur C, ou signé ⟶indicateur V.

a_{n-1}	b_{n-1}	s_{n-1}	C	V
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	0	0
1	0	0	1	0
1	0	1	0	0
1	1	0	1	1
1	1	1	1	0

II-7 Codage des réels en virgule fixe

Le principe consiste à fixer la position de la virgule et à donner aux bits à droite de cette position un poids en puissance de 2 négative: $2^{-1}, 2^{-2}, \ldots$

Soit A le mot code représentant le réel $X(A)$ sur n bits dont m après la virgule:

$$X(A) = -a^{n-1}2^{n-1-m} + \sum_{i=0}^{n-2} a_i 2^{i-m}$$

Exemple avec n=8 et m=3: le code 00101 100b représente le réel:
$2^2 + 2^0 + 2^{-1} = 4 + 1 + 0.5 = 5.5$

On peut aussi écrire:

$$X(A) = 2^{-m} Z(A)$$

II-8 Opérations logiques et masques

- **opérateur OU**

  Permet de mettre à 1 un bit ou un groupe de bits sans modifier les autres.

  Ex: xxxx xxxx + 0000 1000 = xxxx 1xxx. Met à 1 le bit b3.

- **opérateur ET**

  Permet d'isoler un bit ou un groupe de bits.

  Ex: xxxx xxxx & 0000 1000 = 0000 x000. Isole le bit b3.

  Permet aussi de mettre à 0 un bit ou un groupe de bits.

  Ex: xxxx xxxx & 1111 0111 = xxxx 0xxx. Met à zéro le bit b3.

- **opérateur OU EXCLUSIF**

  Permet d'inverser la valeur d'un bit ou d'un groupe de bits.

  Ex: xxxx xxxx $\oplus$ 0000 1000 = xxxx $\bar{x}$xxx

II-9 Codage des caractères

code	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E
0x00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	NP	CR	SO
0x10	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS
0x20	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.
0x30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>
0x40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N
0x50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^
0x60	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n
0x70	p	q	r	s	t	u	v	w	x	y	z	{	—	}	~

Les caractères 0 a 31 et 127 sont des caractères de contrôle.

Le 8ème bit peut avoir divers usages:

- contrôle de parité
- définition de caractères étendus (accentués par exemple).

Caractères de contrôle et propriétés du code ascii

⇨ Les lettres de l'alphabet sont dans l'ordre croissant et se suivent: Un tri alphabétique se ramène à un tri numérique.

⇨ Le passage de minuscule en majuscule et vice-versa se fait simplement en inversant le bit 5 du code (soit + ou - $20)

⇨ Le passage d'un chiffre à son code ASCII consiste à rajouter $30 à ce chiffre.

Quelques caractères de contrôles:

NUL	absence de caractère	SOH	début d'entète	STX	début de texte
ETX	fin de texte	EOT	fin de transmission	ENQ	interrogation
ACQ	accusé réception	BEL	sonnette	BS	retour arrière
HT	tabulation horizontale	LF	ligne suivante	VT	tabulation verticale
FF	page suivante	CR	retour chariot	NAK	reçu avec erreur
SYN	synchronisation	CAN	annulation	SUB	substitution
ESC	échappement	FS	séparateur de fichier	GS	séparateur de groupe
RS	séparateur d'article	US	séparateur d'unité	DEL	effaçer

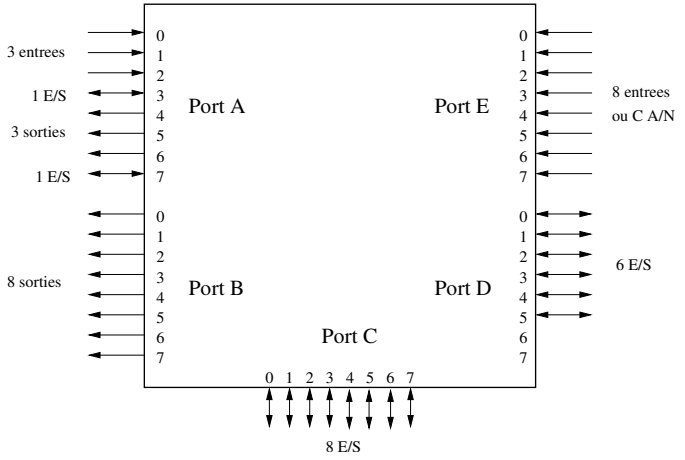
III-Structure du 68HC11

III-1 Présentation

Le 68HC11 série E est un micro-contrôleur développé par Motorola (devenu freescale) construit autour d'une unité centrale 8 bits. Il possède:

- un espace adressable de 64 Ko (bus d'adresses de 16 bits).
- Un convertisseur analogique/numérique 8 bits multiplexé sur 8 canaux.
- Une interface de communication série asynchrone (RS232)
- Une interface de communication série synchrone
- Un timer 16 bits avec 3 entrées de capture et 4 sorties de comparaison plus 1 E/S programmable
- 22 sources d'interruptions (internes et externes)
- 38 broches d'E/S polyvalentes réparties sur 5 ports.

III-2 Ports du 68HC811E2

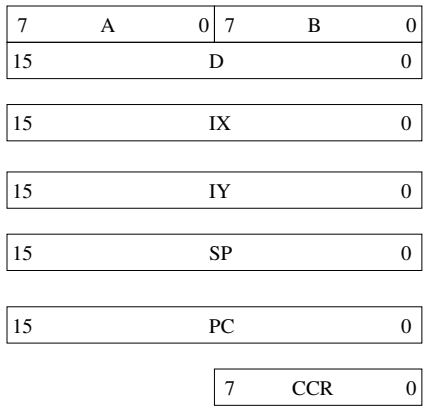


III-3 Brochage du 68HC11E2

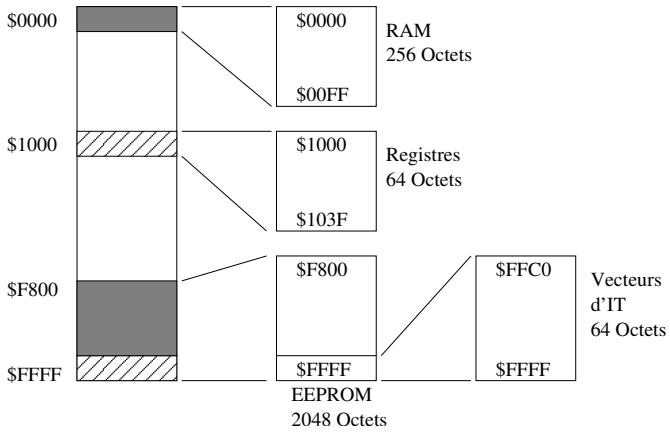
Le 68HC11E2 est inclus dans un boitier à 52 broches. En plus des 38 E/S des ports, il possède 14 broches:

- ⇨ **VSS** et **VDD**: alimentation VDD=5V, VSS=0.
- ⇨ **XTAL** et **EXTAL**: connection d'un quartz ou d'une horloge externe.
- ⇨ **E** et **4XOUT**: sorties d'horloge.
- ⇨ $\overline{\text{IRQ}}$ et $\overline{\text{XIRQ}}$: entrées d'interruptions externes.
- ⇨ **VRH** et **VRL**: tension de référence pour la conversion A/N. VRH$<$VDD.
- ⇨ **MODA** et **MODB**: mode de fonctionnement.
- ⇨ $\overline{\text{RESET}}$ signal d'initialisation.
- ⇨ **STRA** et **STRB**: en mode normal (monopuce), signaux de communication avec une interface parallèle.

III-4 Registres internes



III-5 Plan mémoire



III-6 Registre d'état (CCR)

S	X	H	I	N	Z	V	C

- **C**: Carry: Mis à 1 en cas de retenue lors d'une opération arithmétique. Indique un débordement en contexte non-signé.
- **V**: Overflow: Débordement en contexe signé.
- **Z**: Zero: Mis à 1 si le résultat d'une opération vaut zéro.
- **N**: Negative: Mis à 1 pour un résultat négatif (contexte signé) soit bit b7=1.
- **I**: Interrupt mask: Mis à 1 pour interdire les interruptions masquables.
- **H**: Half-Carry: Demi-retenue. Utilisée pour les opérations en DCB.
- **X**: X interrupt mask: Interruption non-masquable (entrée $\overline{\text{XIRQ}}$). Une fois mis à 0, ne peut plus être mis à 1.
- **S**: Stop disable: mis à 1 pour inactiver l'instruction `stop` (valeur par défaut au reset).

III-7 Jeu d'instructions

Classement en 5 grandes familles:

- **Instructions de traitement:** regroupent les opérations arithmétiques et logiques sur les données. exécutées entre registres où entre registre et mémoire, le résultat étant stocké dans un registre.
- **Instructions de chargement/rangement** Pour être traitées, les données doivent être chargées dans les registres et les résultats stockés en mémoire.
  chargement (*load*): registre ⟵ mémoire.
  rangement (*store*): mémoire ⟵ registre.
- **Instructions de branchement** Par défaut les instructions s'exécutent séquentiellement dans l'ordre de rangement en mémoire.
  Les branchements permettent de dérouter un programme de manière conditionnelle.

- **Sous-programmes, pile et interruptions** Les instructions de cette catégorie permettent
  - de gérer des sous-programmes (instructions d'appel et de retour)
  - de gérer des interruptions (sous-programmes particuliers).
  - de gérer explicitement une structure de données de type pile (utilisée de manière implicite par les sous-programmes).
- **Instructions diverses** On range dans cette catégorie les instructions systèmes où des instructions inclassables par ailleurs (Exemple: l'instruction `nop` qui ne fait rien!)

IV-Jeu d'instructions

IV-1 Instructions de chargement/rangement

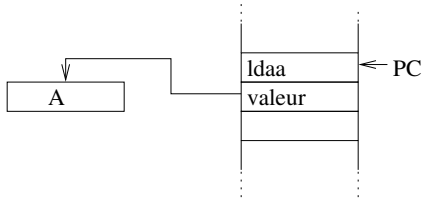
Les opérations sur les données réalisées par l'UAL du 68HC11 nécessitent en général 2 opérandes dont 1 au moins doit être contenu dans un registre interne du micro-contrôleur.

Les instructions de chargement permettent de transférer des données en mémoire vers un registre.

Inversement, les instructions de rangement permettent d'effectuer le transfert inverse (copie en mémoire du contenu d'un registre).

Adressage immédiat

➳ la valeur est codée à la suite de l'instruction dans le programme.



Notation: `ldaa #valeur`

Instructions de chargement

Objectif: charger un registre avec une valeur.

Registres concernés:

`A, B` (8 bits) `D, IX, IY, SP` (16 bits)

Instructions: `ldaa, ldab, ldd, ldx, ldy, lds`

Pour un regitre 8 bits la valeur est codée sur un octet, pour un registe 16 bits, elle est codée sur deux octets en commençant par les poids forts.

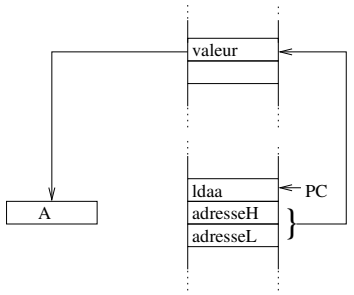
Différentes manières d'accéder à une valeur (modes d'adressage):

- immédiat,
- direct,
- étendu,
- indexé.

Adressage direct et étendu

➳ Adressage direct: l'octet suivant l'instruction représente l'adresse mémoire ou est stockée la valeur. L'accès est limitée aux adresses $0000 à $00FF.

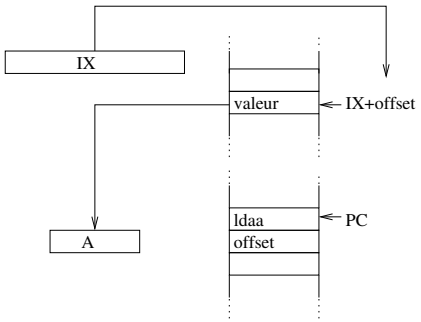
➳ Adressage étendu: comme l'adressage direct mais l'adresse est codée sur les deux octets suivant l'instruction (poids fort d'abord).



Notation: `ldaa $adresse`

Adressage indirect ou indexé

- l'adresse est obtenue en additionnant la valeur contenue dans un regitre d'index (`IX` ou `IY`) avec un *offset* codé dans le programme sous forme d'un entier non-signé sur 8 bits.



Notation: `ldaa offset,X`

- `tsy` Transfère le contenu du pointeur de pile SP incrémenté de 1 dans le registre d'index IY. `IY←SP+1`.
- `txs` Transfère le contenu du registre d'index IX décrémenté de 1 dans le pointeur de pile SP. `SP←IX-1`.
- `tys` Transfère le contenu du registre d'index IY décrémenté de 1 dans le pointeur de pile SP. `SP←IY-1`.
- `xgdx` Echange le contenu de D avec IX. `D⇔IX`.
- `xgdy` Echange le contenu de D avec IY. `D⇔IY`

Transferts entre registres

Le 68HC11 possède des instructions de transferts entre registres que l'on peut classer dans les instructions de chargement/rangement.

Ce sont des instructions sans arguments (codées sur 1 octet). On parle d'adressage implicite.

- `tab` Transfère le contenu de l'accumulateur A dans l'accumulateur B. `B←A`.
- `tba` Transfère le contenu de l'accumulateur B dans l'accumulateur A. `A←B`.
- `tap` Transfère le contenu de l'accumulateur A dans le registre de condition CCR. `CCR←A`.
- `tpa` Transfère le contenu du registre de condition CCR dans l'accumulateur A. `A←CCR`.
- `tsx` Transfère le contenu du pointeur de pile SP incrémenté de 1 dans le registre d'index IX. `IX←SP+1`.

Instructions de rangement

Objectif: ranger le contenu d'un registre en mémoire.

Registres concernés:

`A, B` (8 bits)

`D, IX, IY, SP` (16 bits)

Instructions: `staa, stab, std, stx, sty, sts`

Modes d'adressage:

- direct,
- étendu,
- indexé.

Exemple: `std $10`: copie la valeur contenue dans `D` a l'adresse $0010 pour les poids forts et $0011 pour les poids faibles. Equivalent aux deux instructions exécutées en séquence: `staa $10, stab $11`.

IV-2 Instructions de traitement

Le traitement des informations est effectué par l'Unité Arithmétique et Logique. Il s'agit d'opérations arithmétiques et logiques.

Les instructions de traitement peuvent être classées de différentes manières. La classification qui a été choisie pour ce cours est la suivante:

- Opérations sur deux opérandes dont un en mémoire.
- Opérations sur deux opérandes entre registres.
- Opérations sur un seul opérande.
- Opérations diverses.

On note R un registre et M une valeur stockée en mémoire.

Fonction	Effet	Instructions
Addition sans retenue	R ⟵ R+M	`adda, addb, addd`
Addition avec retenue	R ⟵ R+M+C	`adca, adcb`
Soustraction	R ⟵ R − M	`suba, subb, subd`
Soustraction avec retenue	R ⟵ R − M − C	`sbca, sbcb`
Comparaison	R − M	`cmpa, cmpb, cpd, cpx`
ET bit à bit	R ⟵ R·M	`anda, andb`
OU bit à bit	R ⟵ R+M	`oraa, orab`
OU exclusif bit à bit	R ⟵ R⊕M	`eora, eorb`
test bits	R·M	`bita, bitb`

Opérations sur 2 opérandes dont 1 en mémoire

Opérations de la forme: registre ⟵ f(registre,mémoire)

Modes d'adressage possibles (pour l'opérande en mémoire):

- immédiat,
- direct,
- étendu,
- indexé.

Registres: accumulateurs A,B et D et registres IX et IY pour l'instruction de comparaison.

Exemples:

- `adda #10`
  Additionne le contenu de A avec la valeur décimale 10. Le résultat est stocké dans A.
- `subb $F0`
  Soustrait au contenu de B la valeur stockée à l'adresse $00F0. Le résultat est stocké dans B.
- `cpx #$1000`
  Compare la valeur contenue dans IX avec la valeur $1000. Seul les indicateurs sont mis à jour (N, Z, V et C). Le résultat n'est pas stocké.
- `anda 0,X`
  Effectue un ET bit à bit entre le contenu de A et celui de l'adresse mémoire donnée par X. Le résultat est stocké dans A.

Opérations à 2 opérandes entre registres:

Instruction	Effet	fonction
`aba`	A ⟵ A+B	additionne A et B
`sba`	A ⟵ A−B	soustrait B de A
`cba`	A−B	compare A à B
`mul`	D⟵ A×B	multiplie A par B
`abx`	IX ⟵ IX + 0:B	additionne les poids faibles de IX avec le contenu de B
`aby`	IY ⟵ IY + 0:B	additionne les poids faibles de IY avec le contenu de B

Remarque: Ces instructions utilisent un adressage implicite.

Instructions à un seul opérande

Opérations de la forme: registre ⟵ f(registre) où mémoire⟵f(mémoire)

Modes d'adressage possibles (pour l'opérande en mémoire):

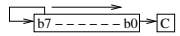
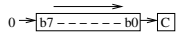
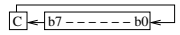
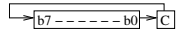
- étendu,
- indexé.

Registres: accumulateurs A,B et D et pour certaines instructions registres IX, IY et SP.

Dans le tableau suivant, on note M le contenu d'un registre où d'un emplacement mémoire.

Exemples:

- `aba`
  Additionne A et B. Le résultat est stocké dans A.
- `mul`
  Multiplie A par B. Le résultat sur 16 bits est stocké dans D. Soit les poids forts dans A et les poids faibles dans B.
  Exemple: si A contient $14 et B $03, après l'instruction `mul` D contiendra $003C soit $00 dans A et $3C dans B.
- `abx`
  Additionne à IX (registre 16 bits) le contenu de B (registre 8 bits). Le résultat est stocké dans IX.

Fonction	Effet	Instructions
Déc. vers la gauche		`asla, aslb, asl, asld`
		`lsla, lslb, lsl, lsld`
Déc. arith. vers la droite		`asra, asrb, asr`
Déc. log. vers la droite		`lsra, lsrb, lsr, lsrd`
Rotation vers la gauche		`rola, rolb, rol`
Rotation vers la droite		`rora, rorb, ror`
Mise à zéro	M⟵0	`clra, clrb, clr`
Décrémentation	M⟵M−1	`deca, decb, dec, des, dex,`
Incrémentation	M⟵M+1	`inca, incb, inc, ins, inx,`
Négation	M⟵0−M	`nega, negb, neg`
Complément à 1	M⟵$FF−M	`coma, comb, com`
Comparaison à 0	M−0	`tsta, tstb, tst`

Exemples:

- `clra`
  Met 0 dans le registre A.
- `inx`
  Ajoute 1 au contenu de IX.
- `nega`
  Remplace A par son complément à 2.
- `tst $1000`
  Compare le contenu de l'adresse $1000 avec 0. Seuls les indicateurs Z et N sont affectés.

Exemples:

- `bset $FD %00001000`
  Met à 1 le bit 3 de l'adresse mémoire $00FD.
- `bclr 0.X %00001000`
  Met à 0 le bit 3 de l'adresse mémoire contenue dans le registre IX.

Remarque: la notation %xxxxxxxx permet de représenter un nombre sous forme binaire.

Manipulations de bits

Deux instructions du 68HC11 permettent de modifier la valeur d'un ou plusieurs bits d'une donnée (en plus des fonctions logiques).

⇨ `bset` Cette instruction permet de mettre à 1 les bits d'une donnée d correspondants aux bits à 1 d'un masque m. Elle supporte l'adressage direct (limité aux adresse $00 à $FF) et indexé. Elle réalise la fonction $d + m$. Elle nécessite deux opérandes: l'adresse de la donnée et le masque (sur 8 bits).

Syntaxe: `bset adresse masque` ou `bset offset,X masque`

⇨ `bclr` Cette instruction permet de mettre à 0 les bits d'une donnée d correspondants aux bits à 1 d'un masque m. Comme l'instruction précédente, elle supporte l'adressage direct et indexé et nécessite deux opérandes. Elle réalise la fonction $d.\overline{m}$.

Syntaxe: `bclr adresse masque` ou `bclr offset,X masque`

Division

⇨ L'instruction `idiv` réalise la division entière (non-signé) sur 16 bits du contenu de D (numérateur) par le contenu de IX (dénominateur).

Rappel: la division entière de n par d produit un quotient q et un reste r tels que: $n/d = q \times d + r$.

Le résultat de la division (quotient q) est mis dans IX et le reste r dans D.

⇨ L'instruction `fdiv` permet de déterminer la valeur fractionnaire de D/IX.

D doit être inférieur à IX. Après l'opération, le résultat dans IX est interprétable comme un réel inférieur à 1 codé avec 16 bits après la virgule

Ces deux instructions utilisent un adressage implicite.

Exemples:

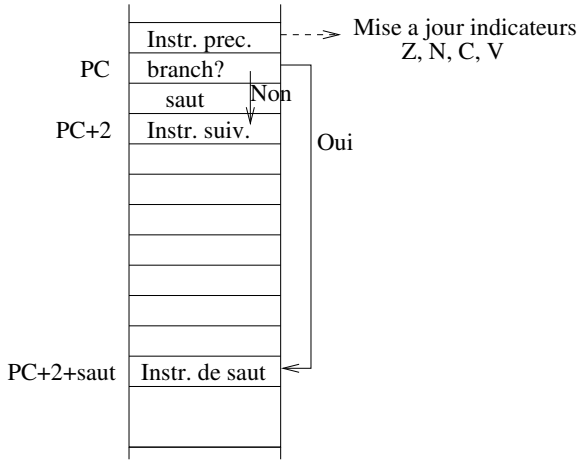
- `idiv`

  Soit 430 le contenu de D (soit $01AE) et soit 40 le contenu de IX (soit $0028). Après `idiv`, IX contiendra 10 (soit $000A) qui est le quotient de 430/40 et D contiendra la valeur 30 ($1E) reste de la division entière (430=10x40+30)

- `fdiv`

  Avec D=30 (reste de la division précédente) et IX=40 après l'instruction `fdiv`, D contiendra la valeur 0 et IX la valeur $C000 soit en binaire %1100 0000 0000 0000 qu'il faut interpréter comme un nombre en virgule fixe égal à $2^{-1} + 2^{-2} = 0.5 + 0.25 = 0.75$. Celà correspond à la partie fractionnaire de la division de 30 par 40 (30/40=0.75).

Branchement conditionnel



IV-3 Instructions de branchement

Les instructions de branchement permettent de réaliser des traitements conditionnels (du type “si condition alors faire”) et d'implémenter des structures de boucles (“faire tant que...”).

Elles sont codées sur deux octets et utilisent un mode d'adressage dit relatif. Le deuxième octet de l'instruction représente un déplacement (saut) sous forme d'un entier signé sur 8 bits (-128 à +127).

Principe: une condition, fonction de l'instruction de branchement, est calculée à partir des indicateurs du CCR: N, V, C et Z. Il s'agit d'une expression booléenne. Si celle-ci est vrai alors le branchement a lieu et le programme continue à une adresse calculée à partir de l'octet qui suit le code de l'instruction de branchement. Si le résulat est faux, alors le programmme continue en séquence à l'instruction suivante.

Calcul du saut

- Le saut représente le nombre, codé en complément à 2, à rajouter au pointeur de programme PC pour que celui-ci contienne l'adresse de la prochaine instruction à exécuter.
- L'instruction de branchement est codée sur 2 octets et au moment de son exécution PC contient l'adresse de l'instruction suivante en mémoire. Un saut de 0 revient donc à exécuter l'instruction suivante (donc à ne pas faire de saut!)
- En pratique le langage d'assemblage permet d'éviter tout calcul, grâce à l'utilisation d'étiquettes. Il suufit de mettre une étiquette devant l'instruction ou l'on souhaite se brancher et d'indiquer cette étiquette dans l'instruction de branchement. Exemple:

```
       bra suite
       ...
suite  ...
       ...
```

Instruction fonction de la condition de branchement

condition	instruction	condition	instruction
C=1	`bcs, blo`	C=0	`bcc, bhs`
V=1	`bvs`	V=0	`bvc`
N=1	`bmi`	N=0	`bpl`
Z=1	`beq`	Z=0	`bne`
C+Z=1	`bls`	C+Z=0	`bhi`
N⊕V=1	`blt`	N⊕V=0	`bge`
Z+(N⊕V) =1	`ble`	Z+(N⊕V) =0	`bgt`
1	`bra`	0	`bnr`

Autres instructions de branchement

Deux autres instructions de branchement conditionnel existent. Elles utilisent les modes d'adressage direct et indexé.

Elles sont codées sur 4 octets: code opératoire, adresse ($adresse en adressage direct et offset,X en adressage indexé), masque, saut (adressage relatif).

➵ `brset` Effectue l'opération M·masque et le saut si le résultat est différent de 0.

➵ `brclr` Effectue l'opération M·masque et le saut si le résultat vaut 0.

Ces deux insructions permettent de tester un bit et d'effectuer un branchement si celui-ci vaut 1 (`brset`) où 0 (`brclr`).

Exemple: `brset $10 %00001000 saut` Effectue le branchement si le bit 3 de la valeur contenue à l'adresse $0010 vaut 1.

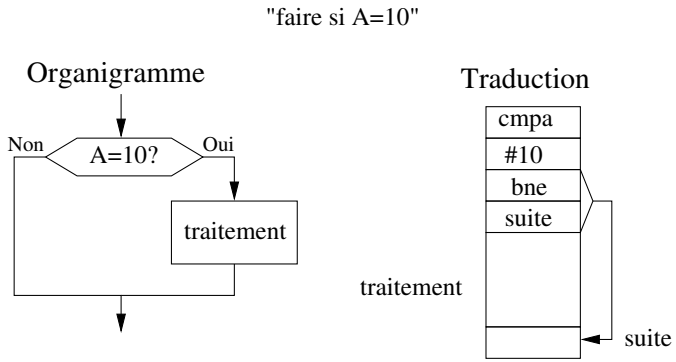
➵ `jmp` Saut inconditionnel en adressage étendu ou indexé.

Comparaisons et branchements

Si l'instruction qui précède l'instruction de branchement est une comparaison (`cmpa, cmpb, cpd, cpx, cpy`) ou une soustraction (`suba, subb, subd, sba`) de type R−M, on peut utiliser les instructions suivantes en fonction du contexte, signé ou non-signé:

test	instruction nombre signés	instruction nombres non-signés
R=M	`beq`	`beq`
R≠M	`bne`	`bne`
R>M	`bgt`	`bhi`
R≥M	`bge`	`bhs`
R<M	`blt`	`blo`
R≤M	`ble`	`bls`

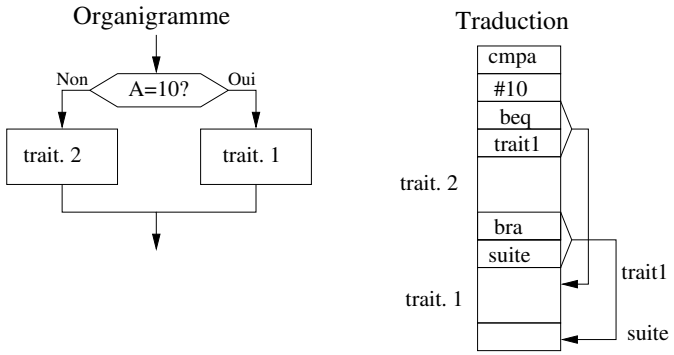
Implémentation de l'alternative simple



Remarque: Il est préférable d'inverser la condition, celà évite un branchement supplémentaire.

Double alternative

"si A=10, faire, sinon faire"

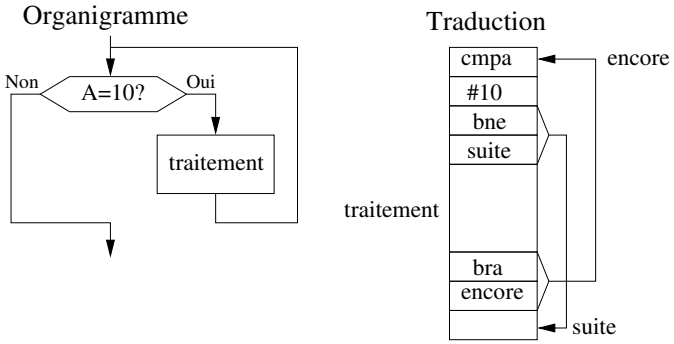


Remarque: Dans ce cas, on peut conserver la condition originale.

-

Structure de boucle

"faire tant que A=10"



Remarque: comme pour l'alternative simple, il est mieux d'inverser la condition.

-

IV-4 Sous-programmes, Pile, IT, ...

Les autres instructions du micro-contrôleur intégrent les instructions permettant la programmation modulaire:

- sous-programmes
- pile
- interruptions

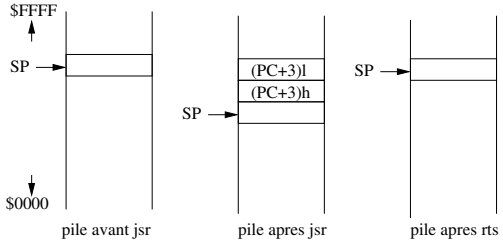
ainsi que diverses instructions.

Pile

Pour qu'un sous-programme revienne, quand il se termine, à l'instruction suivant son appel il faut conserver l'adresse de retour. ⇨ C'est le rôle de la *pile*.

La pile est gérée par le registre SP. Elle est stockée n'importe ou en mémoire et SP contient l'adresse de son sommet. Elle croit vers les adresses basses.

Exemple pour un sous-programme (instruction `jsr`)



Rem: l'adresse de la pile doit être fixée au début du programme (instruction `lds`).

Sous-programmes

Deux instructions pour exécuter un sous-programme:

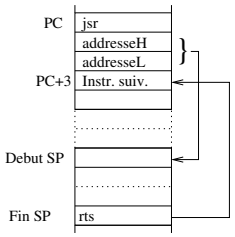
➳ `bsr` (adressage relatif)

➳ `jsr` (adressage direct, étendu ou indexé)

Et une instruction pour le terminer:

➳ `rts` (adressage implicite)

Exemple: `jsr $adresse`



Manipulation explicite de la pile

Deux instructions permettent de mettre des données dans la pile (empiler) où d'en récupérer (dépiler).

⇨ **empilage:**

Registres concernés: `A, B, IX, IY`

Adressage implicite (codage sur 1 octet de l'instruction)

Instructions: `psha, pshb, pshx, pshy`

⇨ **dépilage:**

Registres concernés: `A, B, IX, IY`

Adressage implicite (codage sur 1 octet de l'instruction)

Instructions: `pula, pulb, pulx, puly`

Interruptions, principe

Une interruption consiste à interrompre un programme en cours d'exécution pour exécuter un sous-programme traitant l'interruption puis à reprendre l'exécution du programme là ou elle avait été interrompue.

Une interruption est provoquée par un événement interne ou externe.

Elle possède deux différences avec un sous-programme:

- elle n'est pas appelée de manière explicite
- elle peut interrompre le programme n'importe ou et doit donc provoquer la sauvegarde des registres dans la pile et leur restauration au retour de l'interruption.
- son adresse est stockée dans un vecteur d'interruption (tableau à un emplacement prédéfini de la mémoire).

Sources d'interruptions

⇨ externes: entrées $\overline{\text{IRQ}}$, $\overline{\text{XIRQ}}$ et $\overline{\text{RESET}}$ (non masquables sauf $\overline{\text{IRQ}}$).

⇨ logicielles: instruction `swi` et "code opération illégal" (non masquables)

⇨ internes (unités périphériques intégrées): interruption temps réel, débordement timer, entrées capture, sorties comparaison, accumulateur d'impulsions, lignes séries (SCI et SPI). Toutes ces IT sont masquables.

Chaque source d'interruption possède une entrée spécifique dans le vecteur d'interruptions.

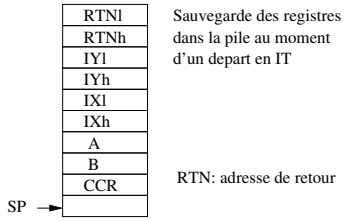
adresse	source	adresse	source	adresse	source
$FFD6-D7	sci	$FFD8-D9	spi	$FFDA-DB	accu. entrée
$FFDC-DE	accu. deb.	$FFDE-DF	deb. timer	$FFE0-E1	IC4/TOC5
$FFE2-E3	TOC4	$FFE4-E5	TOC3	$FFE6-E7	TOC2
$FFE8-E9	TOC1	$FFEA-EB	TIC3	$FFEB-EC	TIC2
$FFEE-EF	TIC1	$FFF0-F1	Int. T.R.	$FFF2-F3	$\overline{\text{IRQ}}$
$FFF4-F5	$\overline{\text{XIRQ}}$	$FFF6-F7	swi	$FFF8-F9	code op. ill.

Instructions de gestion des IT

⇨ Un sous-programme d'interruption doit se terminer par une instruction spécifique: `rti` qui permet de restaurer les registres du programme interrompu.

⇨ L'instruction `swi` permet de provoquer une interruption dite logicielle. Elle sauvegarde les registres du 68HC11 puis exécute le code à partir de l'adresse contenue dans l'entrée correspondant du vecteur d'IT ($FFF6-$FFF7).

⇨ Il existe aussi une instruction d'attente d'une interruption: `wai` qui sauve les registres dans la pile puis attend qu'un interruption arrive.



Autorisation générale des IT

Le bit I du CCR permet de gérer les interruptions masquables.

Deux instructions permettent de manipuler ce bit:

instruction	effet	description
`cli`	I←0	mise à zéro de I, autorise les IT masquables
`sei`	I←1	mise à un de I, interdit les IT masquables

Mise en place d'une IT

- autoriser les interruptions de manière générale (`cli`)
- autoriser l'interruption souhaitée (bit xxxI d'un registre particulier à mettre à 1)
- mettre l'adresse du SP d'IT dans l'entrée correspondante du vecteur d'IT.
- dans le SP d'IT, forcer à 1 le drapeau de l'IT pour le réinitialiser.
- terminer le S.P. d'IT par l'instruction `rti`.

Instructions diverses

Manipulation directe des indicateurs C et V (adressage implicite):

instruction	effet	description
`clc`	C⟵0	mise à zéro de C
`clv`	V⟵0	mise à zéro de V
`sec`	C⟵1	mise à un de C
`sev`	V⟵1	mise à un de V

Autres instructions:

- `stop` Permet d'arréter l'horloge du micro-contrôleur si le bit S du CCR est à 0
- `test` Utilisée uniquement en mode test.
- `nop` Instruction sans action.
- `daa` Ajustement en décimal codé binaire.

Broches programmables:

- Pour programmer une broche en sortie, il faut mettre à 1 le bit correspondant du registre de contrôle. S'il est à 0,la broche est en entrée.

V-Entrées/Sorties

V-1 Entrées/Sorties logiques

Les entrées/sorties logiques sont aussi appelées E/S parallèles, digitales où numériques. Elles correspondent aux ports A à E du 68HC11E2 qui en posséde 38. Chaque broche est indépendante.

Broche en entrée:

- 5V sur une broche: bit correspondant du registre de données vaut 1.
- 0V sur une broche: bit correspondant du registre de données vaut 0.

Broche en sortie:

- bit du registre de données=0 ⟶ 0V sur la broche correspondante
- bit du registre de données=1 ⟶ 5V sur la broche correspondante

Port A

Le port A posséde 3 entrées (PA0 à PA2), 3 sorties (PA4 à PA6) et deux broches d'E/S programmables en entrée ou en sortie (PA3 et PA7).

Registre de données:

PORTA:	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0	$1000

Registre de contrôle:

PACTL:	DDRA7				DDRA3				$1026

Programmation en entrée: DDRx=0, en sortie: DDRx=1

Les broches du port A sont partagées avec les fonctions du timer.

Port B

Le port B est utilisable uniquement en sortie. Registre de données:

PORTB:	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0	$1004

Port C

Le port C est un port d'usage général. Chaque broche peut être programmée en entrée ou en sortie.

Registre de données:

PORTC:	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	$1003

Registre de contrôle:

DDRC:	DDRC7	DDRC6	DDRC5	DDRC4	DDRC3	DDRC2	DDRC1	DDRC0	$1007

V-2 Gestion des unités périphériques

Les unités périphériques (timer, convertisseur, liaisons séries, ...) fonctionnent de manière autonome vis à vis de l'unité centrale. La communication avec l'unité centrale (i.e. le programme) s'effectue par l'intermédiaire de registres situés dans l'espace adressable ($1000 à $103F).

Ces registres permettent de programmer ces unités, de lire ou d'écrire des données et de tester des drapeaux.

Un drapeau est un bit qui passe à 1 si un événement survient.

Il existe deux manières de gérer un événement:

- par scrutation: on attend le passage à 1 du drapeau correspondant à l'événement attendu. C'est un fonctionnement bloquant.
- par interruption: l'événement provoque une interruption du programme et l'exécution d'un sous-programme de traitement de l'interruption.

Port D

Le port D possède 6 broches programmables en entrée ou en sortie. Ces broches peuvent être utilisées par les unités de communication série.

Registre de données:

PORTD:			PD5	PD4	PD3	PD2	PD1	PD0	$1008

Registre de contrôle:

DDRD:			DDRD5	DDRD4	DDRD3	DDRD2	DDRD1	DDRD0	$1009

Port E

Le port E est utilisable uniquement en entrée. Il partage ses broches avec le convertisseur A/N. Registre de données:

PORTE:	PE7	PE6	PE5	PE4	PE3	PE2	PE1	PE0	$100A

Interruption v.s. scrutation

Les unités périphériques sont capables d'effectuer des tâches simples indépendamment de l'unité centrale, par exemple:

- fonction timer: attendre un certain temps (durée variable, quelques ms)
- fonction convertisseur: convertir une tension analogique (durée quelques μs)
- fonction capture: attendre le front d'un signal (durée imprévisible)

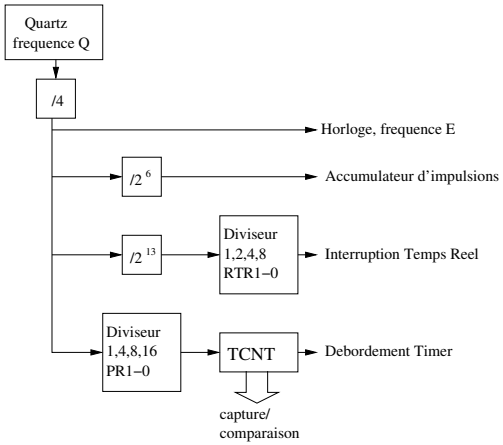
Une fois leur tâche effectuée, ces unités périphériques mettent à 1 un indicateur (lèvent un drapeau) et peuvent interrompre le programme en cours si on les y autorise.

La scrutation (attente du passage à 1 d'un drapeau) oblige à attendre qu'un tâche soit terminée pour en exécuter une autre.

Le fonctionnement par interruption permet lui un pseudo-parallélisme, une tâche n'utilisant les ressources de l'unité centrale que quand elle se termine.

V-3 Timer

Présentation générale



Le compteur/timer TCNT

Aprés division par 4 et par une valeur réglable grace aux bits PR1 et PR0 de TMSK2 (1,4,8 ou 16), le signal résultant incrémente le compteur 16 bits TCNT à chaque période. (Remarque: PR0 et PR1 ne peuvent être modifiés que pendant les 64 cycles machine qui suivent le Reset).

Au débordement de TCNT (passage de 65535 à 0), le drapeau TOF (bit de TFLG2) passe à 1 et, si le masque d'IT TOI (bit de TMSK2) est à 1, le sous-programme d'IT dont l'adresse est stockée en $FFDE-DF est exécuté. Le drapeau TOF doit être réinitialisé dans ce sous-programme pour qu'une nouvelle IT soit possible.

Ce timer TCNT est utilisé pour les fonctions de capture et de comparaison.

Interruption Temps Réel

Permet de générer une interruption périodique.

La période se règle avec les bits RTR1 et RTR0 de PACTL Elle dépend de la fréquence d'horloge E=Q/4.

RTR1	RTR0	période	Q=4.9152 Mhz
0	0	$2^{13}/E$	6.7ms
0	1	$2^{14}/E$	13.3ms
1	0	$2^{15}/E$	26.7ms
1	1	$2^{16}/E$	53.3ms

Le bit RTII de TMASK2 doit être mis à 1 pour autoriser l'IT et le bit RTIF de TFLG2 doit être réinitialisé dans la fonction d'IT. L'adresse du sous-programme d'IT doit être stockée en: $FFF0-$FFF1 (vecteur d'IT).

PACTL:							RTR1	RTR0	$1026
TMASK2:		RTII							$1024
TFLG2:		RTIF							$1025

Registres du timer

TCNT(h):	15	14	13	12	11	10	9	8	$100E
TCNT(l):	7	6	5	4	3	2	1	0	$100F

Remarque: Le registre TCNT est à lecture seule.

TMSK2:	TOI						PR1	PR0	$1024
TFLG2:	TOF								$1025

Réglage de la période

			Q=4.9152 Mhz	
PR1	PR0	diviseur	résolution	débordement
0	0	1	0.813 μs	53.3ms
0	1	4	3.25 μs	213.3 ms
1	0	8	6.52 μs	426.6 ms
1	1	16	13 μs	853.2 ms

Fonction capture

Principe: Un événement (front d'un signal) sur une broche externe provoque la recopie du contenu du timer TCNT dans un registre. Ceci permet de "dater" des événements. Cet événement met par ailleurs un drapeau à 1 et peut générer une interruption.

Le 68HC11 possède 4 entrées de captures: IC1 (broche PA2), IC2 (PA1), IC3 (PA0), IC4 (PA3). A chaque entrée correspond un registre 16 bits TICx, x de 1 à 4.

L'événement provoquant la capture est programmable: front montant, front descendant ou front quelconque.

Programmation

Choix du type d'évènement:

TCTL2:	EDG4B	EDG4A	EDG3B	EDG3A	EDG2B	EDG2A	EDG1B	EDG1A	$1021

EDGxB	EDGxA	événement
0	0	inactif
0	1	front montant
1	0	front descendant
1	1	front quelconque

Registres de capture: TIC1: $1010-$1011, TIC2: $1012-$1013, TIC3: $1014-$1015, TIC4: $101E-$101F.

Drapeaux:

TFLG1:					IC4F	IC1F	IC2F	IC3F	$1023

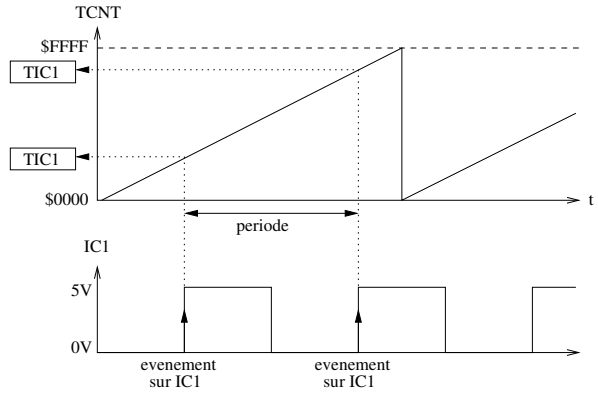
Masques d'interruption:

TMSK1:					IC4I	IC1I	IC2I	IC3I	$1022

Autres registres: bit 3 de PACTL à 0 pour utiliser IC4 et TMSK2 pour régler la résolution du timer TCNT.

Illustration de la fonction capture

Mesure de la période d'un signal. Capture sur front montant, broche PA2 (IC1).



Fonction comparaison

Principe: Quand la valeur du timer TCNT devient égale à la valeur d'un registre TOCx, une action est effectuée sur la broche OCx. Cette action peut-être une mise à 0, une mise à 1 ou un changement d'état.

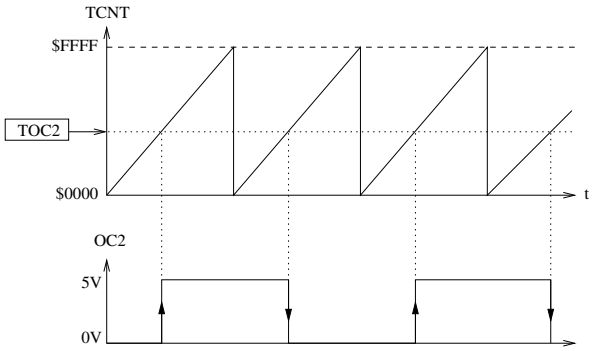
Ceci permet de générer des signaux périodiques sur certaines broches du 68HC11.

Par ailleurs, l'action sur la sortie s'accompagne de la levée d'un drapeau (OCxF) qui peut générer une interruption si celle-ci est autorisée (bit OCxI à 1).

Le 68HC11 possède 5 sorties de comparaisons: OC1 (PA7), OC2 (PA6), OC3 (PA5), OC4 (PA4) et OC5(PA3). Le fonctionnement de TOC1 est un peu particulier. Il permet de d'affecter les 5 broches de sortie (PA3 à PA7) à chaque comparaison.

Illustration de la fonction comparaison

Génération d'un signal périodique sur la broche PA6 (OC2). Changement d'état de la sortie à chaque comparaison.



Programmation de OC1

Choix des broches affectées: pour affecter la broche PAx, il faut mettre à 1 le bit OC1Mx.

OC1M:	OC1M7	OC1M6	OC1M5	OC1M4	OC1M3				$100C

Choix de la valeux mise sur PAx (si OC1Mx=1) à chaque comparaison valide de TOC1 avec TCNT:

OC1D:	OC1D7	OC1D6	OC1D5	OC1D4	OC1D3				$100D

Autres registres:

bit 3 de PACTL à 1 pour utiliser OC5 (PA3 en sortie)

TMSK2 pour régler la résolution du timer TCNT.

Programmation

Registres de comparaison (16 bits): TOC1: $1016-17, TOC2: $1018-19, TOC3: $101A-1B, TOC4: $101C-1D et TOC5: $101E-1F.

Choix de l'action sur la sortie (TOC2 à TOC5):

TCLT1:	OM2	OL2	OM3	OL3	OM4	OL4	OM5	OL5	$1020

OMx	OLx	événement
0	0	non connecté
0	1	changement d'état de OCx
1	0	mise à 0 de OCx
1	1	mise à 1 de OCx

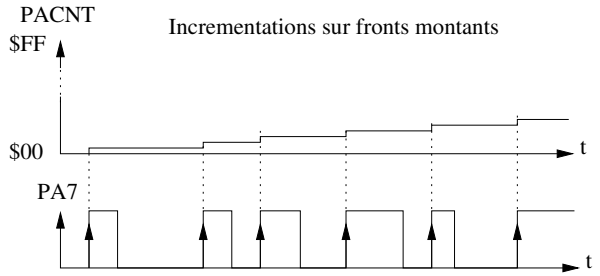
Drapeaux et masques d'interruption:

TFLG1:	OC1F	OC2F	OC3F	OC4F	OC5F				$1023
TMSK1:	OC1I	OC2I	OC3I	OC4I	OC5I				$1022

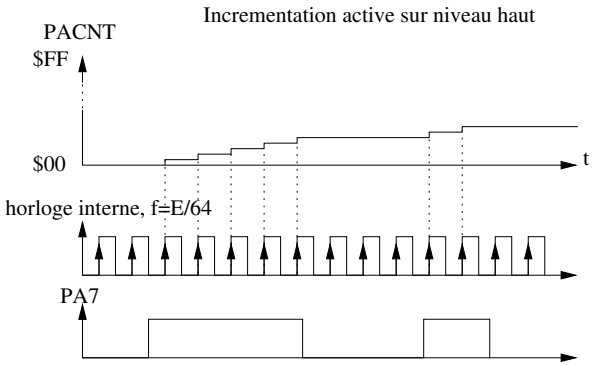
Accumulateur d'impulsions

L'accumulateur d'impulsions est un registre 8 bits qui possède deux fonctions:

⇨ Compteur d'événements: l'accumulateur est incrémenté à chaque évènement du signal relié à la broche PAI (PA7). Une interruption peut être générée lorsque l'accumulateur est plein. La fréquence maximale du signal externe est de E/2.



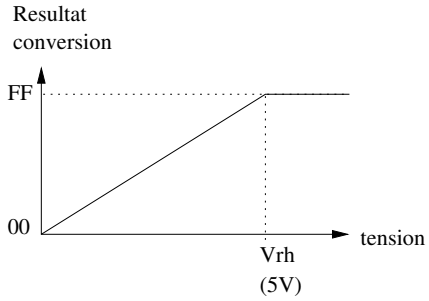
⇨ Accumulation d'impulsions: PACNT est incrémenté, tant que le signal d'entrée sur la broche PAI est à un niveau donné, par les impulsions de l'horloge interne E divisée par 64.



V-4 Convertisseur Analogique/Numérique

Principe

Convertir une tension analogique en une valeur numérique. Un convertisseur est caractérisé par sa résolution en nombre de bits codant le résultat de la conversion.



Registres de programmation

Registre de données: PACNT en $1027

PACTL:	DDRA7	PAEN	PAMOD	PEDGE				$1026

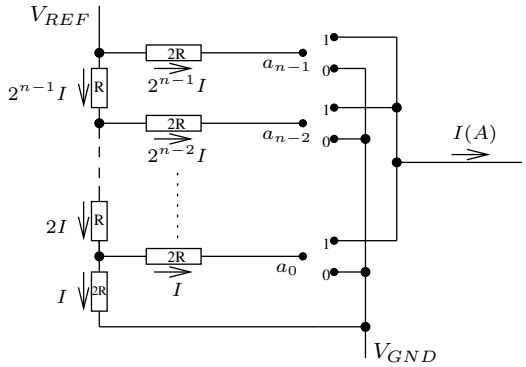
DDRA7: direction de PA7, a mettre à 0 (en entrée). PAEN: mettre à 1 pour activer les fonctions de l'accumulateur. PAMOD: 0 en compteur d'événements, 1 en accumulateur d'impulsions. PEDGE: en compteur d'événements: 0: compte les fronts descendants, 1 les fronts montants. En accu. d'impulsions: 0 compteur actif sur niveau haut, 1 actif sur niveau bas.

TMSK2:			PAOVI	PAII				$1024
TFLG2:			PAOVF	PAIF				$1025

PAOVI et PAOVF: masque et drapeau d'interruption sur le débordement de PACNT. Vecteur d'IT: $FFDC-DD.

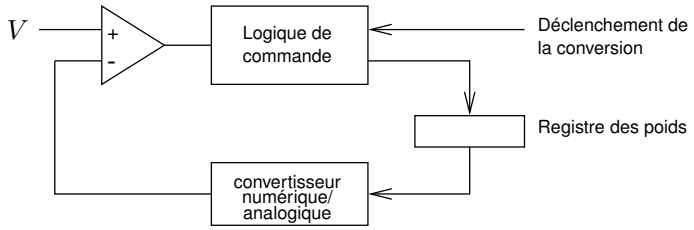
PAII et PAIF: masque et drapeau d'IT déclenchée par les fronts de PA7 (fonctionnement en accumulateur d'impulsions). Vecteur d'IT: $FFDA-DB.

Conversion Numérique/Analogique



Convertion Analogique/Numérique

Conversion par approximations successives.



Registres liés au convertisseur

OPTION:	ADPU	CSEL							$1039

- ADPU:
  - convertisseur inactif
  - convertisseur actif.
- CSEL: choix de l'horloge pour la conversion:
  - 0: horloge E,
  - 1: circuit RC interne.

Fonctionnement du convertisseur

Le 68HC11 possède 8 entrées de conversion (PE0 à PE7).

Les conversions s'effectuent par groupe de quatre:

- soit sur une seule entrée, soit sur un groupe de 4 entrées
- dès qu'une série de conversion est terminée, le drapeau de fin de conversion passe à 1 (bit CCF de ADCTL)
- les résultats des 4 conversions sont stockés dans les registres 8 bits ADR1 à ADR4 (adresses: $1031 à $1034).
- le convertisseur doit être activé par mise à 1 du bit ADPU du registre OPTION avant toute conversion
- les séries de conversions peuvent se faire en continu ou pas
- l'écriture d'une valeur dans le registre ADCTL initialise le drapeau de fin de conversion (CCF) à 0 et redémarre une nouvelle série de conversion.

ADCTL:	CCF	-	SCAN	MULT	CD	CC	CB	CA	$1030

- CCF: drapeau fin de conversion, mis à 1 lorsque les 4 registres de conversion ADR1 à ADR4 contiennent des résultats valides.
- SCAN:
  - 0: 4 conversions une seule fois. Les résultats sont rangés dans ADR1 à ADR4.
  - 1: Conversions effectuées en permanence. ADR1 à ADR4 sont mis à jour après chaque conversion.
- MULT:
  - 0: les 4 conversions sont faites sur un seul canal (choisi par CC-CA).
  - 1: une conversion sur chacun des 4 canaux d'un groupe. Le groupe est fixé par CC.
- CD, CC, CB, CA: choix de l'entrée ou du groupe d'entrées.

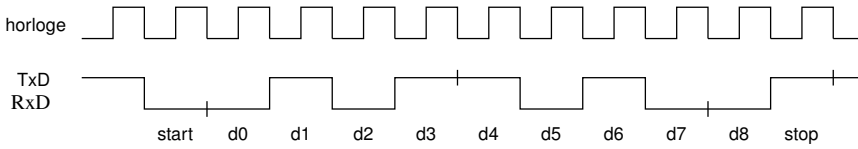
⇨ Pour une seule entrée (MUL=0), CD=0 et CC.CB.CA représente le numéro de l'entrée (Ex: CC=1, CB=0, CA=1 pour PE5)

⇨ Pour 4 entrées (MUL=1): CC=0 pour PE0 à PE3, CC=1 pour PE4 à PE7.

CC	CB	CA	Entrée	Résultat (MULT=1)
0	0	0	AN0 (PE0)	ADR1
0	0	1	AN1 (PE1)	ADR2
0	1	0	AN2 (PE2)	ADR3
0	1	1	AN3 (PE3)	ADR4
1	0	0	AN4 (PE4)	ADR1
1	0	1	AN5 (PE5)	ADR2
1	1	0	AN6 (PE6)	ADR3
1	1	1	AN7 (PE7)	ADR4

SCI - UART

Format des données: 8 bits ou 9 bits.



Broches utilisées: PD0 (RxD) et PD1 (TxD)

V-5 Communication série et 68HC11

Deux unités de communication série:

⇨ **SCI** Serial Comunication Interface.
Communication série de type RS232. Utilisée pour le chargement des programmes.

⇨ **SPI** Serial Peripheral Interface. Communication synchrone entres des 68HC11 ou des périphériques synchrones (controleurs LCD, convertisseurs N/A série, horloges...)

Registres:

Données: SCDR ($102F)

Contient les 8 premiers bits de la donnée reçue ou de la donnée à transmettre.

Contrôle:

SCCR1 ($102C)

Fixe le mode de tranmission (nombre de bits), le mode de réveil en réception et contient le 9ième bit éventuel.

SCCR2 ($102D)

Activation du récepteur ou de l'émetteur. Masques d'IT.

SCSR ($102E)

Registre de statut (drapeaux).

BAUD ($102B)

Fixe la cadence de transmission en bauds.

V-6 Système de développement

V-6.1 Présentation générale

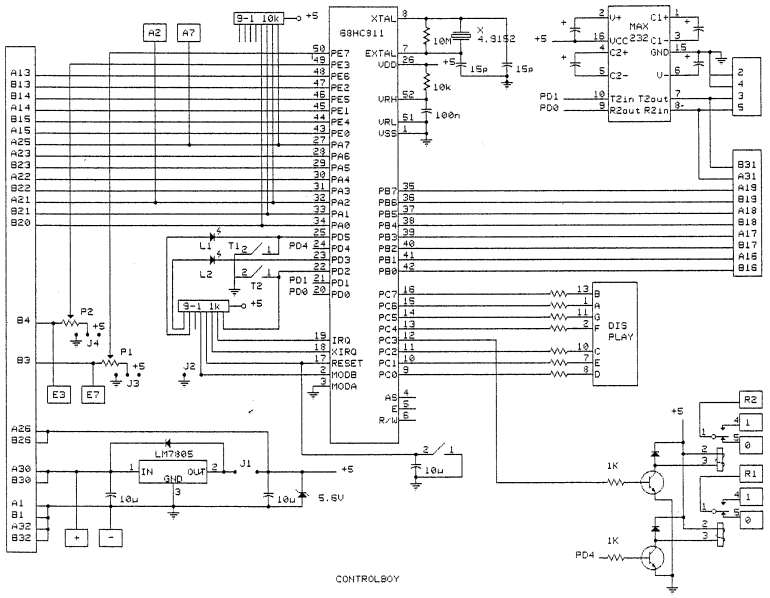
Le système de développement utilisé en TP est une carte controlboy 1 commercialisé par la société Controlord et basée sur un 68HC11E2.

La carte comporte, en complément d'un micro-contrôleur 68HC11E2:

- Une alimentation stabilisée (LM7805), un quartz à 4,9152 MHz et un circuit de mise en forme des signaux pour la communication série (Max 232).
- A disposition du programmeur, on trouve: un afficheur 7 segments relié aux port C, deux relais R1 (relié à PD4) et R2 (relié à PC3), une led L1 associée à un bouton poussoir T1 (relié à PD5), une led L2 reliée à PD3, un bouton poussoir T2 relié à PD2.
- Différents connecteurs et borniers complétent la carte permettant d'accéder à la plupart des E/S du micro-contrôleur.

V-6.2 Langage d'assemblage

- Le langage d'assemblage (ou assembleur) est le langage dans lequel vous écrirez vos programmes.
- Chaque instruction machine est représenté par un mémmonique qui sera traduit en code machine par l'assembleur.
- Les premiers caractères d'une ligne sont réservées aux étiquettes. Une instruction machine doit donc être précédée d'un espace ou d'une tabulation.
- Une étiquette placée en tête de ligne permet de faire référence à une adresse de manière symbolique.
- Commentaires: tout ce qui suit un point-virgule (jusqu'à la fin de la ligne) est considéré comme un commentaire.
- Représentation des constantes: On peut représenter des constantes sous différentes formes: décimal, hexadécimal, binaire, caractère...
  Exemples: décimal: 97, hexadécimal $61, binaire: %01100001, caractère: 'a'



V-6.3 Directives d'assemblage:

- `equ` définit une constante symbolique. Exemple: `porta equ $1000`
- `org` précise l'adresse à partir de laquelle les données qui suivent seront implémentées en mémoire. Exemple: `org $F800`: début du programme
- `fcb` initialise des octets en mémoire avec des valeurs. Exemple: `fcb 'a','b'` ou encore `fcb 'ab'`
- `fdb` initialise des mots de 16 bits avec des valeurs. Exemple: `fdb $1000`
- `rmb` réserve des octets en mémoire. Exemple: `rmb 10` réserve 10 octets en mémoire
- `end` indique la fin de programme pour l'assembleur

Remarque: les directives `fcb, fdb` et `rmb` initialisent ou réservent la mémoire à partir de la dernière directive `org` rencontrée. Elles peuvent être précédées d'une étiquette qui fera ensuite référence à l'adresse en mémoire correspondante.

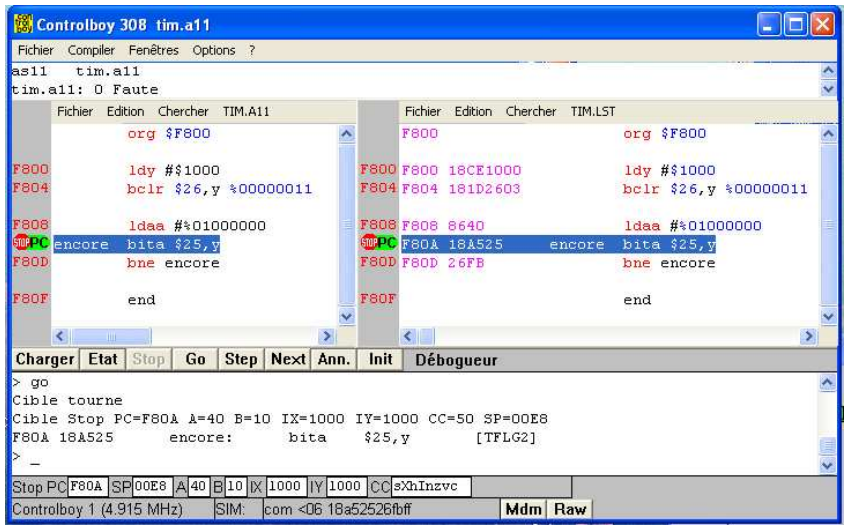
V-6.4 Exemple de programme (exemp.a11)

```
;----------------- constantes symboliques
ddrd    equ $09
portd   equ $08
bit3    equ %00001000
;----------------- variables ------------
        org 0
compt   fdb 42850
;----------------- programme ------------
        org $F800
        ldx #$1000
        bset ddrd,x bit3 ; PD3 en sortie
encore  ldy compt
tempo   dey              ; boucle 300ms
        bne tempo
        ldaa portd,x     ; changement
        eora bit3        ; etat PD3
        staa portd,x
        bra encore
        end
```


V-6.6 Fichiers

- Le programme est un fichier texte d'extension `.a11`.
- Il est traduit en langage machine ce qui fournit un fichier listing d'extension `.lst`.
- Il est enfin traduit sous un format qui permet son téléchargement sur la cible (le micro-contrôleur). Ce fichier a pour extension `.s19`.

V-6.5 Mise au point des programmes



V-6.7 Exemple de listing (exemp.lst)

```
                ;----------------- constantes symboliques
0009            ddrd    equ $09
0008            portd   equ $08
0008            bit3    equ %00001000
                ;----------------- variables ------------
0000                    org 0
0000 A762       compt   fdb 42850
                ;----------------- programme ------------
F800                    org $F800
F800 CE1000             ldx #$1000
F803 1C0908             bset ddrd,x bit3 ; PD3 en sortie
F806 18DE00     encore  ldy compt
F809 1809       tempo   dey              ; boucle 300ms
F80B 26FC               bne tempo
F80D A608               ldaa portd,x     ; changement
F80F 9808               eora bit3        ; etat PD3
F811 A708               staa portd,x
F813 20F1               bra encore
                        end
```