

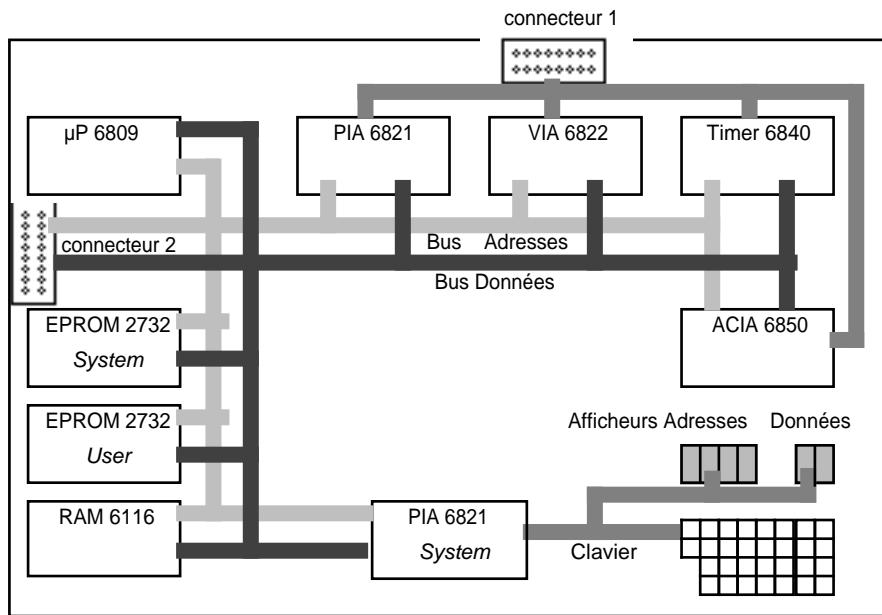
## B11&B17 - Étude d'un système numérique

But : exemples de codes binaires et principes de base du traitement numérique de l'information dans un système à microprocesseur. Réalisation d'un générateur de signaux programmable.

### 1ère partie : étude générale du système

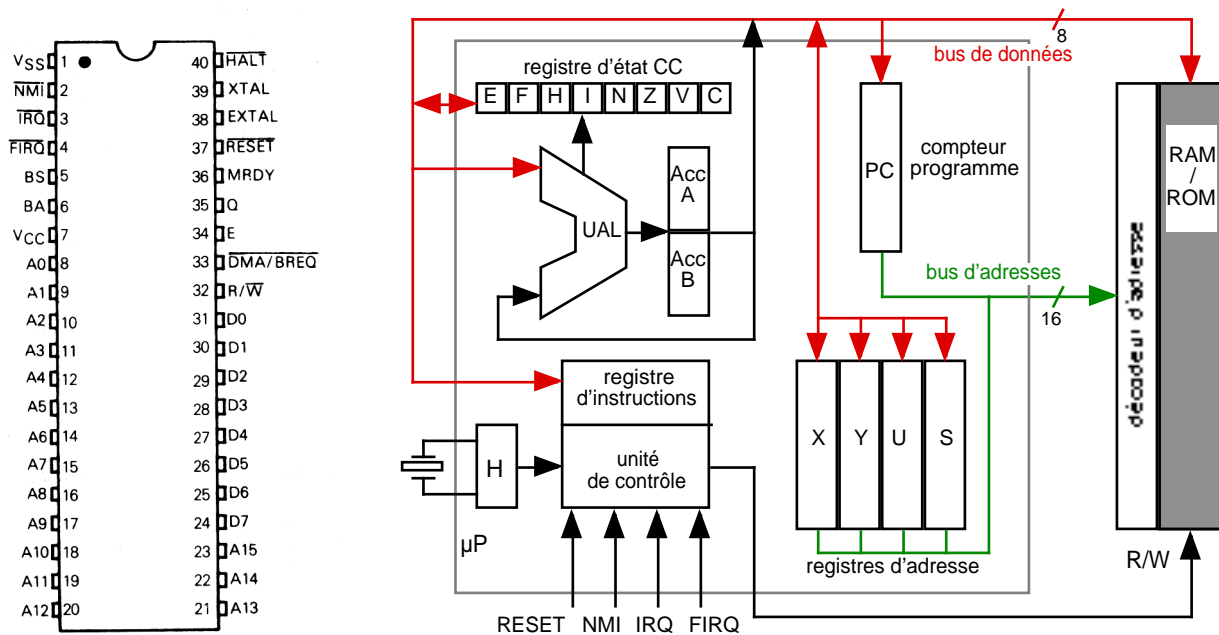
On utilise une carte MC09 (fabricant : SERIElectronique) à microprocesseur type 6809, associé à une carte d'entrée/sortie (ref 003) branchée sur le connecteur 1 à l'aide d'un câble plat en nappe.

**⚠ Remarque :** un microcontrôleur intègre en un seul boîtier l'ensemble des éléments de cette carte plus quelques autres (CAN, CNA,...). Le microcontrôleur correspondant au  $\mu P$  6809 a pour référence  $\mu C$  6811. Toutefois, dans une première approche des systèmes numériques, il est plus commode d'étudier un système "éclaté" faisant apparaître chaque sous-ensemble de façon distincte.



Brochage du 6809 :

Schéma fonctionnel du 6809 :



1) Repérer sur la carte (double face) les différents composants du système :

- le clavier constitué d'un pavé numérique matriciel de 16 touches hexadécimales, de 12 touches de fonction et de deux touches spéciales (RST : remise à zéro ; NMI : interruption).
- les afficheurs 7-segments hexadécimaux répartis en deux groupes : à gauche, 4 digits pour les adresses ; à droite 2 digits pour les données.
- le microprocesseur (abrégé :  $\mu P$ ) et le quartz 4MHz qui lui est associé (fréquence horloge système :  $f_{\mu P} = 1 \text{ MHz} \Leftrightarrow T_{\mu P} = 1 \mu\text{s}$ ).
- le bus de données et le bus d'adresse.
- la mémoire, composée de :
  - 4 ko de RAM répartie en deux circuits intégrés type 6116 de 2 ko chacun
  - plusieurs EPROM type 2732 de 8 ko chacune, dont :
    - une EPROM marquée "MC09" contenant le "système d'exploitation", qui est un ensemble de programmes gérant le clavier, l'affichage, etc.
    - une EPROM marquée "TS2" contenant certains programmes utilisés en TP.
- les interfaces d'entrée/sortie (abrégé : E/S ou IO : *Input/Output*) : parallèle (PIA 6821 : *Parallel Interface Adapter*), série (ACIA 6850 : *Asynchronous Interface Adapter*), temporisateur programmable (PTM : *Programmable Timer* 6840), etc.
- les connecteurs, dont celui de l'alimentation (0/5V).

2) Du point de vue matériel, le  $\mu P$  6809 comprend notamment :

- une Unité Arithmétique et Logique (ALU : *Arithmetic and Logic Unit*) qui exécutent les opérations booléennes et les calculs numériques.
- deux registres de travail ou "accumulateurs" A et B
- un compteur programme (PC : *Program Counter*)
- quatre registres d'adresse (X, Y, S, U)
- un registre d'état (CC : *Code Condition register*) contenant un ensemble d'indicateurs binaires ou "drapeaux". Les quatre bits de poids faibles sont :
  - ( $b_3$ ) N : indicateur de résultat négatif
  - ( $b_2$ ) Z : indicateur de résultat nul
  - ( $b_1$ ) V : indicateur de dépassement ( $V$  : *overflow*)
  - ( $b_0$ ) C : retenue de calcul ( $C$  : *carry*)
- quatre entrées d'interruption : RES (*RESet*), NMI (*Non Masquable Interrupt*), IRQ (*Interrupt Request*), FIRQ (*Fast IRQ*).

3) Utilisation du clavier :

NB : notation des nombres :

nb entiers	notation "Motorola"	notation "Intel"	(valeur en base 10)
base 10	1100	1100	1100
base 2	%1100	b1100	12
base 16	\$1100	h1100	4352

- Brancher l'alimentation de la carte. Faire un *RESET*.
- Examen des registres : observer le contenu des registres (*clavier* : *REG* puis *INC* ou *DEC*).
- Examen de la mémoire : lire la mémoire (*clavier* : *MEM*) à l'adresse \$8000 (*clavier* : taper 0 0 0 0). Ecrire par exemple \$11 à cette adresse (les DELs vertes correspondantes s'allument sur la carte d'E/S). Puis incrémenter (*INC*) jusqu'à l'adresse \$8002 et écrire \$48 (DELs rouges). Terminer par *clavier* : *FIN* puis *RESET*.

(soit, au clavier : *MEM*, 8, 0, 0, 0, 1, 1, *INC*, *INC*, 4, 8, *FIN*, *RESET*)

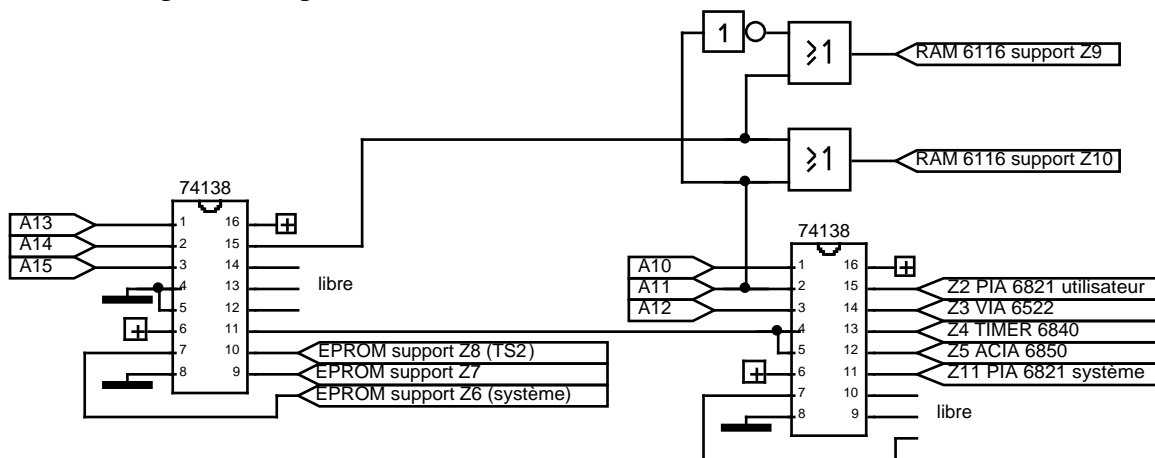
**Quelle est la valeur de \$11 et de \$48 en binaire ?**

4) Carte mémoire :

- a) Convertir sous forme hexadécimale les nombres  $16^3, 2^{10}, 2^{12} - 1$ .
- b) Convertir sous forme décimale les nombres \$A000, \$0FFF
- c) Quelle est la "largeur" (nb bits) du bus de données ? Du bus d'adresse ?
- d) Quelle est la taille de l'espace adressable total ?
- e) La RAM débute à l'adresse \$0000 . Quelle est l'adresse la plus haute de cette mémoire ?
- f) L'EPROM système finit à l'adresse \$FFFF. Quelle est son adresse la plus basse ?
- g) Le long d'un axe gradué de \$0000 à \$FFFF (document réponse), dessiner une carte simplifiée de l'espace adressable (memory map) en précisant l'endroit où se trouvent les différents composants, sachant que :

- l'EPROM "TS2" commence à l'adresse \$A000
- les I/O (entrées/sorties) sont entre les adresses \$8000 et \$9FFF

h) Schéma électrique (partiel) de l'adressage : celui-ci comprend notamment deux CI 74138 décodeur/démultiplexeur "1 parmi 8" :



D'après ce schéma, dresser la carte mémoire du système (cf doc réponse), sachant que :

- les différents circuits sont sélectionnés par un 0 sur leur entrée de sélection (CS : Chip Select").
- la table de vérité du 74138 est la suivante :

broche	6	4	5	3	2	1	15	14	13	12	11	10	9	7
fonction	CS1	CS2	CS3	A2	A1	A0	Out 0	Out 1	Out 2	Out 3	Out 4	Out 5	Out 6	Out 7
	X	X	1	X	X	X	1	1	1	1	1	1	1	1
	X	1	X	X	X	X	1	1	1	1	1	1	1	1
	0	X	X	X	X	X	1	1	1	1	1	1	1	1
	1	0	0	1	1	1	1	1	1	1	1	1	1	0
	1	0	0	1	1	0	1	1	1	1	1	1	0	1
	1	0	0	1	0	1	1	1	1	1	0	1	1	1
	1	0	0	1	0	0	1	1	1	0	1	1	1	1
	1	0	0	0	1	1	1	1	0	1	1	1	1	1
	1	0	0	0	0	1	1	0	1	1	1	1	1	1
	1	0	0	0	0	0	0	1	1	1	1	1	1	1

5) Interruptions :

Exécuter le programme "chronomètre", adresse \$AE97 (clavier : EXC, A, E, 9, 7, EXC).

Ce programme fait appel à une technique de programmation dite "par interruptions" matérialisée par une ligne de commande reliant le timer à une entrée d'interruption du µP (FIRQ).

Relever à l'oscilloscope ce signal (broches 4 du µP ou 9 du timer). Quelle est sa période ?

Puis, en agissant sur les touches RST et NMI, indiquer dans quel ordre de priorité sont exécutées les interruptions FIRQ, RESET et NMI.

6) Pour information : quelques instructions courantes du  $\mu P$  6809 (NB : pour plus de détails, cf la table de programmation fournie par le constructeur) :

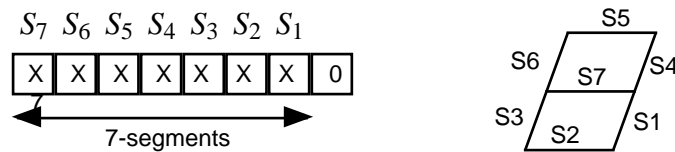
Mouvements de données		Opérateurs		Branchements et sauts		divers	
LD	lecture (LOAD)	CLR	RAZ	BEQ	Branchement si = 0	SWI	fin (END)
ST	écriture (STORE)	ADD	addition	BNE	Branchement si $\neq$ 0	NOP	tempo 2 $\mu s$
TFR	transfert entre registres	SUB	soustraction	BGT	branchement si >		
		MUL	multiplication	BGE	branchement si $\geq$		
		NEG	changement de signe	BLE	branchement si $\leq$		
		INC	incrémentatation	BLT	branchement si <		
		DEC	décrémentatation	JMP	saut inconditionnel (GOTO)		
		CMP	comparaison	JSR	saut à sous-programme		
		COM	NON	RTS	fin de sous-programme		
		AND	ET	RTI	fin de s-pg d'interruption		
		OR	OU				
		EOR	OU Exclusif				
		LSL	décalage à gauche				
		LSR	décalage à droite				
		ROL	rotation à gauche				
		ROR	rotation à droite				

### 2ème partie : codes binaires

#### I- Un exemple de code binaire alphanumérique : le code 7-segments

Lancer le programme "Code 7-segments" (adresse : \$AB40). Entrer un octet (qui apparaît sur les afficheurs de gauche), en appuyant successivement sur deux touches hexadécimales (poids fort puis poids faible). L'équivalent 7-segments apparaît à gauche. Entrer un deuxième octet : un deuxième afficheur est activé. Etc. Arrêt du programme par RST.

Exercice : à chaque bit correspond un segment affiché ("0" = segment éteint ; "1" = segment allumé). **En déduire le code 7-segments des chiffres 0 à 9 (cf document réponse).**



#### II- Un exemple de code binaire numérique : le code en complément à deux

1) Programmation : on considère le programme suivant :

code machine			assembleur			
adresse	code op.	donnée	label	mnémonique	donnée	commentaire
0000	86	17	pgm21	LDA	#\$17	chargement de l'acc. A par \$17
0002	8B	12		ADDA	#\$12	addition : A + \$12 → A
0004	3F			SWI		fin

a) Description : ce programme exécute l'addition hexadécimale  $\$17 + \$12 = \dots$

Les données sont insérées dans le programme lui-même. Le résultat est conservé dans l'accumulateur A en écrasant son contenu antérieur.

Ce programme est écrit sous deux formes complémentaires :

- Code machine : code hexadécimal mis en mémoire vive à l'aide du clavier :

MEM, 0, 0, 0, 0, 8, 6, INC, 1, 7, INC, 8, B, INC, 1, 2, INC, 3, F, FIN

Ce code comprend :

- l'adresse de chaque instruction. \$0000 est l'adresse de début du programme.
- le code de l'instruction, qui comprend :
  - le "code opératoire" (en abrégé : code op. ou *Op Code*), décodé par le  $\mu P$  dans son "registre d'instruction" (*IR : Instruction Register*)
  - la donnée sur laquelle porte l'instruction
- Assembleur : fichier texte traduisant "en clair" (?) le code machine, réparti en plusieurs "champs" :
  - label : nom du programme (pgm21)
  - mnémorique : nom abrégé de l'instruction : LD, ADD, SWI...
  - donnée
  - commentaire

b) *Exécution* : exécuter le programme. Relever le contenu de PC (touche *REG*) : **quelle est l'action de l'instruction SWI ?**

c) En modifiant les données, **remplir le tableau (cf document réponse)** en considérant que l'addition est exécutée en code binaire naturel... (le résultat \$r est à lire dans A par *REG*).

**Quels sont les indicateurs de CC utiles ? quels sont les indicateurs inutiles ? Pourquoi ?**

d) ... ou en considérant que cette addition est exécutée en code signé complément à 2. **Remplir le deuxième tableau (cf document réponse)**

**Quels sont les indicateurs de CC utiles ? quels sont les indicateurs inutiles ? Pourquoi ?**

*Rappel* : le code d'un nombre négatif de valeur absolue *N* est obtenu en code complément à 2 en faisant :  $\overline{N} + 1$

**Annexe : code binaire naturel et code en complément à 2 en écriture hexadécimale**

0	0	0000	32	20	64	40	96	60	128	-128	80	160	-96	A0	192	-64	C0	224	-32	E0
1	1	0001	33	21	65	41	97	61	129	-127	81	161	-95	A1	193	-63	C1	225	-31	E1
2	2	0010	34	22	66	42	98	62	130	-126	82	162	-94	A2	194	-62	C2	226	-30	E2
3	3	0011	35	23	67	43	99	63	131	-125	83	163	-93	A3	195	-61	C3	227	-29	E3
4	4	0100	36	24	68	44	100	64	132	-124	84	164	-92	A4	196	-60	C4	228	-28	E4
5	5	0101	37	25	69	45	101	65	133	-123	85	165	-91	A5	197	-59	C5	229	-27	E5
6	6	0110	38	26	70	46	102	66	134	-122	86	166	-90	A6	198	-58	C6	230	-26	E6
7	7	0111	39	27	71	47	103	67	135	-121	87	167	-89	A7	199	-57	C7	231	-25	E7
8	8	1000	40	28	72	48	104	68	136	-120	88	168	-88	A8	200	-56	C8	232	-24	E8
9	9	1001	41	29	73	49	105	69	137	-119	89	169	-87	A9	201	-55	C9	233	-23	E9
10	A	1010	42	2A	74	4A	106	6A	138	-118	8A	170	-86	AA	202	-54	CA	234	-22	EA
11	B	1011	43	2B	75	4B	107	6B	139	-117	8B	171	-85	AB	203	-53	CB	235	-21	EB
12	C	1100	44	2C	76	4C	108	6C	140	-116	8C	172	-84	AC	204	-52	CC	236	-20	EC
13	D	1101	45	2D	77	4D	109	6D	141	-115	8D	173	-83	AD	205	-51	CD	237	-19	ED
14	E	1110	46	2E	78	4E	110	6E	142	-114	8E	174	-82	AE	206	-50	CE	238	-18	EE
15	F	1111	47	2F	79	4F	111	6F	143	-113	8F	175	-81	AF	207	-49	CF	239	-17	EF
16	10		48	30	80	50	112	70	144	-112	90	176	-80	B0	208	-48	D0	240	-16	F0
17	11		49	31	81	51	113	71	145	-111	91	177	-79	B1	209	-47	D1	241	-15	F1
18	12		50	32	82	52	114	72	146	-110	92	178	-78	B2	210	-46	D2	242	-14	F2
19	13		51	33	83	53	115	73	147	-109	93	179	-77	B3	211	-45	D3	243	-13	F3
20	14		52	34	84	54	116	74	148	-108	94	180	-76	B4	212	-44	D4	244	-12	F4
21	15		53	35	85	55	117	75	149	-107	95	181	-75	B5	213	-43	D5	245	-11	F5
22	16		54	36	86	56	118	76	150	-106	96	182	-74	B6	214	-42	D6	246	-10	F6
23	17		55	37	87	57	119	77	151	-105	97	183	-73	B7	215	-41	D7	247	-9	F7
24	18		56	38	88	58	120	78	152	-104	98	184	-72	B8	216	-40	D8	248	-8	F8
25	19		57	39	89	59	121	79	153	-103	99	185	-71	B9	217	-39	D9	249	-7	F9
26	1A		58	3A	90	5A	122	7A	154	-102	9A	186	-70	BA	218	-38	DA	250	-6	FA
27	1B		59	3B	91	5B	123	7B	155	-101	9B	187	-69	BB	219	-37	DB	251	-5	FB
28	1C		60	3C	92	5C	124	7C	156	-100	9C	188	-68	BC	220	-36	DC	252	-4	FC
29	1D		61	3D	93	5D	125	7D	157	-99	9D	189	-67	BD	221	-35	DD	253	-3	FD
30	1E		62	3E	94	5E	126	7E	158	-98	9E	190	-66	BE	222	-34	DE	254	-2	FE
31	1F		63	3F	95	5F	127	7F	159	-97	9F	191	-65	BF	223	-33	DF	255	-1	FF

### 3ème partie : mouvements de données

But : écrire un programme qui affiche la date du jour sous la forme de 6 chiffres au format jj/mm/aa. Les afficheurs sont aux adresses \$0F51, ..., \$0F56.

#### I- Adressages immédiat (type "constante") et étendu (type "variable")

En mode immédiat, la date est une **constante** située à l'intérieur du programme lui-même.

Saisir et exécuter le code du programme suivant en remplaçant les octets marqués \*\* par le code 7-segment du chiffre voulu (jj/mm/aa) :

```

0100 86 **          pgm31  LDA    #**      adr. immediat  (dizaine jour)
0102 B7 0F 51      STA    affich  adr. etendu
0105 86 **          LDA    #**              (unite jour)
0107 B7 0F 52      STA    affich+1
010A 86 **          LDA    #**              (diz mois)
010C B7 0F 53      STA    affich+2
010F 86 **          LDA    #**              (unite mois)
0111 B7 0F 54      STA    affich+3
0114 86 **          LDA    #**              (diz annee)
0116 B7 0F 55      STA    affich+4
0119 86 **          LDA    #**              (unit annee)
011B B7 0F 56      STA    affich+5
011E 8E 0F 51      LDX    #affich      appel s-p affichage
0121 BD A1 00      JSR    esclav      affichage
0124 3F            SWI                fin

```

*Description* : le programme exécute six fois le même déplacement de données :

**LDA #\*\*** Charge (*LOAD*) l'accumulateur A par la donnée qui suit le code opératoire (\$86) (le signe # indiquant le mode immédiat)

**STA affich** Sauvegarde (*STORE*) le contenu de l'accumulateur A à l'adresse qui suit le code opératoire (\$B7). Le contenu de cette adresse est donc **variable**.

Le programme fait ensuite appel à un sous-programme nommé "esclav" (*instruction JSR : Jump to Sub-Routine*) qui affiche les 6 codes 7-segments à partir de l'adresse chargée dans le registre d'adresse X.

**Compléter le croquis (cf doc réponse) en précisant l'endroit où se trouvent les données et le programme d'une part, l'affichage d'autre part. Indiquer par des flèches le mouvement d'une donnée**

#### II- Adressage indexé (type "pointeur")

Inscrire les codes 7-segments de la date (au clavier) dans un tableau à partir de l'adresse \$0000.

Le programme transfère ces données aux adresses utilisées par le sous-programme d'affichage.

Une première solution consiste à travailler uniquement en adressage étendu (*ne pas saisir ce programme*) :

```

0200 B6 00 00      pgm32a  LDA    >date      adr. etendu
0203 B7 0F 51      STA    affich      adr. etendu
0206 B6 00 01      LDA    >date+1
0209 B7 0F 52      STA    affich+1
020C B6 00 02      LDA    >date+2
020F B7 0F 53      STA    affich+2
0212 B6 00 03      LDA    >date+3
0215 B7 0F 54      STA    affich+3
0218 B6 00 04      LDA    >date+4
021B B7 0F 55      STA    affich+4
021E B6 00 05      LDA    >date+5
0221 B7 0F 56      STA    affich+5
0224 8E 0F 51      LDX    #affich      appel s-p affichage
0227 BD A1 00      JSR    esclav      affichage
022A 3F            SWI                fin

```

Une meilleure solution consiste à utiliser l'adressage indexé, qui est une forme d'adressage prévue

pour le traitement des tableaux (*ne pas saisir ce programme*) :

```

0300 8E 0F 51          pgm32b LDX  #affich
0303 CE 00 00          LDU  #date
0306 A6 C0            LDA  ,U+
0308 A7 80            STA  ,X+
030A A6 C0            LDA  ,U+
030C A7 80            STA  ,X+
030E A6 C0            LDA  ,U+
0310 A7 80            STA  ,X+
0312 A6 C0            LDA  ,U+
0314 A7 80            STA  ,X+
0316 A6 C0            LDA  ,U+
0318 A7 80            STA  ,X+
031A A6 C0            LDA  ,U+
031C A7 80            STA  ,X+
031E 8E 0F 51        LDX  #affich          appel s-p affichage
0321 BD A1 00        JSR  esclav           affichage
0324 3F              SWI  fin
    
```

*Description* : on commence par charger le registre X par l'adresse du début du tableau d'affichage et le registre U par l'adresse du début du tableau de données. Le programme exécute ensuite six fois le même déplacement de données :

*LDA ,U+* Charge l'accumulateur A par la donnée contenue à l'adresse **pointée** par U puis incrémente U d'une unité (pour la prochaine exécution)

*STA ,X+* Sauvegarde le contenu de A à l'adresse pointée par X puis incrémente X.

**Compléter le croquis (cf doc réponse) en précisant l'endroit où se trouvent les données, le programme et l'affichage. Indiquer par des flèches le mouvement d'une donnée. Indiquer par des flèches de couleur différente les adresses pointées par les registres X et U au début de l'exécution du programme.**

### III- Adressage relatif

Le programme précédent peut aisément être compressé en remplaçant la suite des six groupes d'instructions identiques par une boucle de type IF... THEN... ELSE... exécutée six fois.

L'octet marqué \*\* situé à l'adresse \$040F indique le nombre d'octets dont doit être incrémenté ou décrémenté le compteur programme pour retourner au début de la boucle (adresse \$0406). **Calculer cet octet** (exprimé en code en complément à deux sous forme hexadécimale) en s'aidant de l'exemple ci-contre. Saisir et exécuter le code du programme.

adrs.	saut	saut
boucle	(dec)	hexa
0400 8E 0F 51		
0403 CE 00 00		
0406 A6 C0		
0408 A7 80		
040A 11 83 00 06		
040E 26 **		
0410 8E 0F 51		
0413 BD A1 00		
0416 3F		
....	....	....
\$040B	-5	\$FB
\$040C	-4	\$FC
\$040D	-3	\$FD
\$040E	-2	\$FE
\$040F	-1	\$FF
\$0410	0	\$00
\$0411	1	\$01
\$0412	2	\$02
\$0413	3	\$03
\$0414	4	\$04
....	....	....

*Description* :

*CMPU date + 1* Pour savoir s'il faut sortir de la boucle, on compare U à la quantité \$0006  
L'instruction *CMP (Compare)* positionne l'indicateur Z de CC :

- à 1 si U = \$0006

- à 0 si U ≠ \$0006

*BNE boucle* Si U ≠ \$0006, alors on recommence la boucle.

L'instruction *BNE (Branch if Not Equal)* teste l'indicateur Z.

**4ème partie : réalisation d'un générateur de signaux programmable**

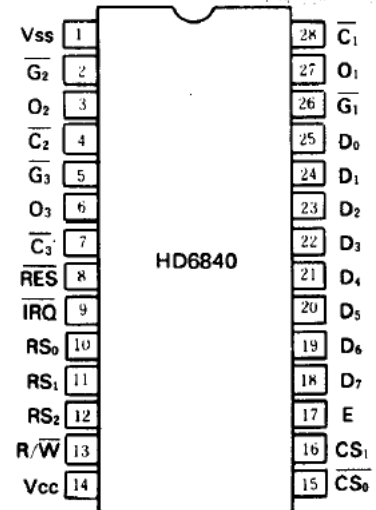
But : écrire un programme pour générer un signal de fréquence 5 kHz et de rapport cyclique 1/3 par le timer1 du PTM 6840.

**I- Programmation du timer**

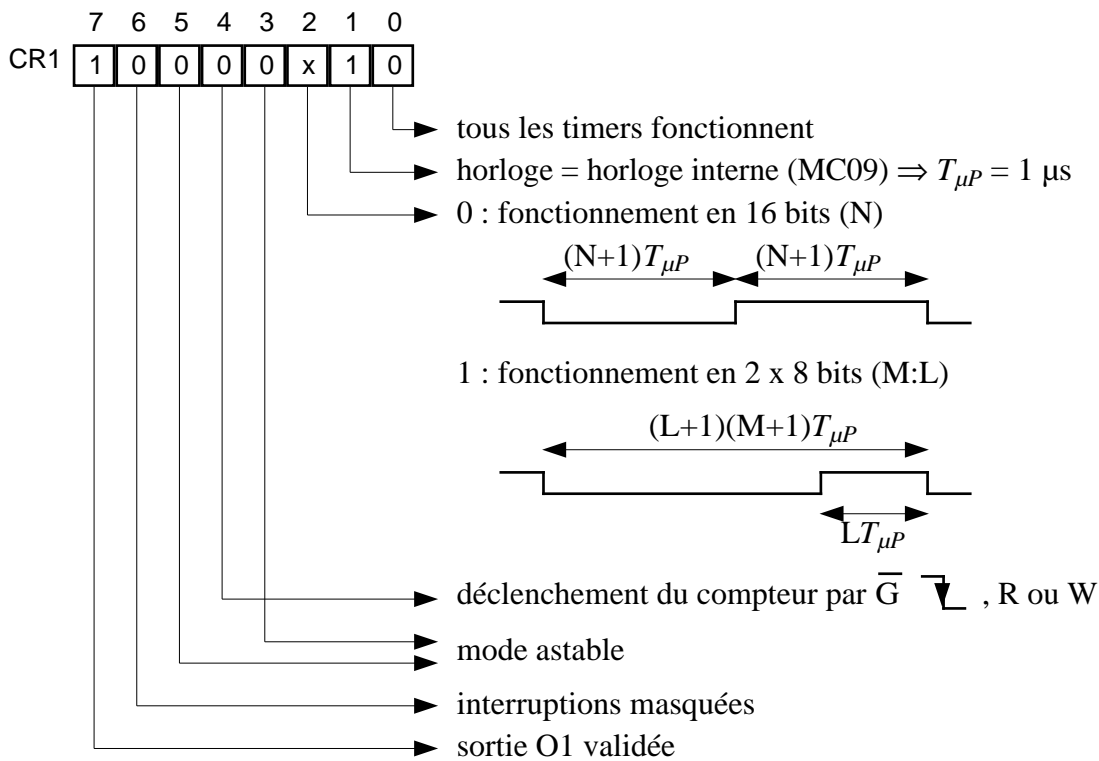
Documentation : datasheet du PTM 6840

Ce PTM comprend trois timers (x = 1, 2 ou 3), chacun muni :

- d'une sortie Ox,
- d'une entrée d'horloge externe C<sub>x</sub>,
- d'une entrée de déclenchement  $\overline{G_x}$  (il faut  $\overline{G_x}=0$  pour que le PTM fonctionne en compteur, ce qui est le cas ici).
- d'un registre de contrôle CR<sub>x</sub>



CR1 est l'octet de contrôle du timer 1 :



Les valeurs de comptage N, ou L et M, seront entrées dans le programme (variable LMN).

Calculer l'octet de programmation du timer 1 et la valeur (en hexa) de la variable LMN.

**II- Saisie et exécution du programme**



Au préalable, faire une RAZ matérielle en débranchant le kit MC09 pendant quelques secondes.

```

essai1 LDA  #$CR1      chargt de l'acc. A par octet contrôle timer1
        LDX  #$M:L      chargement du compteur par valeur de comptage
        JSR  $A880      début du comptage (saut sp init timer1)
        SWI              fin
    
```

Connecter une sonde d'oscilloscope sur la broche 27 du PTM. Relever le signal obtenu.

**1ère PARTIE**

4.g) carte mémoire simplifiée :

4.h) carte mémoire détaillée (extrait) :

circuits	bits d'adresse							adresse hexa haute	taille
	A15	A14	A13	A12	A11	A10	A9A8A7...A0	adresse hexa basse	segment
				1	1	1	1111111111		
Z6				0	0	0	0000000000		
				1	1	1	1111111111		
Z7				0	0	0	0000000000		
				1	1	1	1111111111		
Z8				0	0	0	0000000000		
							1111111111		
Z11							0000000000		
							1111111111		
Z5							0000000000		
							1111111111		
Z4							0000000000		
							1111111111		
Z3							0000000000		
							1111111111		
Z2							0000000000		
				X		1	1111111111		
Z9				X		0	0000000000		
				X		1	1111111111		
Z10				X		0	0000000000		

**2ème PARTIE**

I)

nb décimal	0	1	2	3	4	5	6	7	8	9
code 7-segments										

II.c)

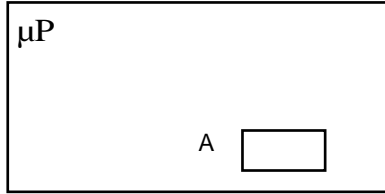
opération en code hexadécimal			interprétation de l'opération en code décimal			examen du registre CC				conclusion : résultat exact (V) ou faux (F) ?
\$x	\$y	\$r	x	y	r	%N	%Z	%V	%C	V / F ?
\$17	\$12									
\$7B	\$12									
\$7B	\$FE									
\$83	\$FE									
\$83	\$F6									

II.d)

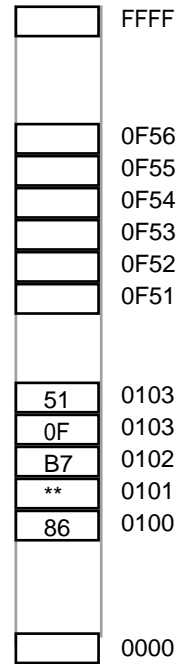
\$x	\$y	\$r	x	y	r	%N	%Z	%V	%C	V / F ?
\$17	\$12									
\$7B	\$12									
\$7B	\$FE									
\$83	\$FE									
\$83	\$F6									

**3ème PARTIE**

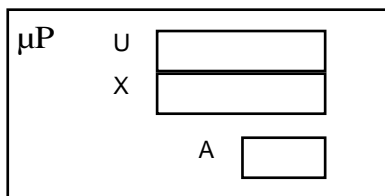
I)



*espace mémoire*



II)



*espace mémoire*

