

**NF 02**  
**2009 / 2010**

**Catherine MARQUE**

**SUJETS DE TP**

**Université de Technologie de Compiègne**



## **TP NF02**

<b>TP 1 - Simulation de portes logiques .....</b>	<b>1</b>
<b>TP 2 - Analyse de circuits séquentiels.....</b>	<b>11</b>
<b>TP 3 – Outil de développement pour microcontrôleur HCS12 .....</b>	<b>17</b>
<b>TP 4 – Carte à microcontrôleur HCS12, modes d’adressage.....</b>	<b>43</b>
<b>TP 5 – Sous-programmes.....</b>	<b>49</b>
<b>TP 6 – Interruption .....</b>	<b>53</b>



# TP 1 – Câblage et test de portes logiques

## Partie 1 : Câblage sur simulateur logique

### Exercice n°1

- Écrivez la table de vérité de la fonction NAND et de la fonction NOR :

E1	E2	NAND	NOR
0	0		
0	1		
1	0		
1	1		

- Déterminez le schéma de câblage d'une fonction NOR à partir de 4 portes NAND.
- Effectuez le câblage sur les Simulateurs logiques à partir du composant MM74C00N.

### Exercice n°2

#### *1- Câblage d'une bascule D*

- En câblant le composant MM74C74N et en vous appuyant sur les 2 tables de vérité (Annexe 5 et Annexe 2) de la bascule D, vérifiez le fonctionnement du Preset et du Clear. Quel état faut-il imposer sur Preset et Clear afin de pouvoir utiliser l'horloge et l'entrée Data ?
- Vérifiez également la recopie du signal de D vers Q sur front d'horloge.

Note : pour la génération de l'horloge, on utilisera un interrupteur manuel.

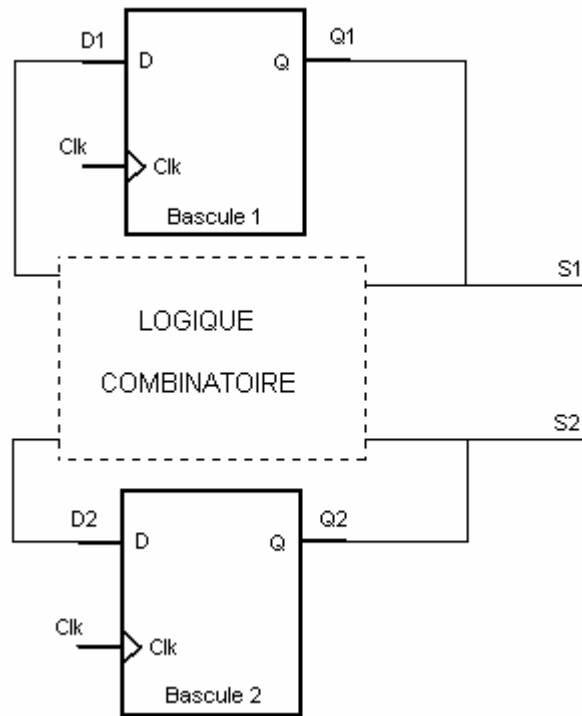
#### *2- Réalisation d'un compteur 2 bits*

On souhaite réaliser un compteur 2 bits, c'est à dire un compteur ayant comme entrée une horloge et un reset et comme sortie un compteur qui suit la séquence suivante à chaque front montant d'horloge :

S1-S2 :      00    01    10    11    00    01    10    11    ...

Pour cela, on utilise 2 bascules D, chaque bascule générant 1 bit du compteur. Voici le schéma sur lequel on se base (page suivante) :

.../...



Pour que S1 et S2 suivent la séquence de comptage décrite ci-dessus, il reste à déterminer le bloc de logique combinatoire qui va relier les sorties S1 et S2 (c'est à dire Q1 et Q2) aux entrées D1 et D2.

Complétez la table de vérité suivante en se basant sur les propriétés de la bascule D (la bascule D a une fonction mémoire puisque l'information en entrée se retrouve en sortie après un coup d'horloge).

Temps	Q1	Q2	D1	D2
t	0	0		
t+1	0	1		
t+2	1	0		
t+3	1	1		

Ecrivez l'équation logiques reliant D1 à Q1 et Q2.

D1 = ?

Faites de même avec celle reliant D2 à Q1 et Q2.

D2 = ?

Réalisez le circuit sur la plaquette en utilisant les composants indiqués en annexe (ex : MM74C74N, 74C86N, CD4069CN) et vérifiez son fonctionnement.

Remarque : pour simplifier le câblage on connectera directement les pattes clear et preset sur un niveau haut.

# ANNEXES

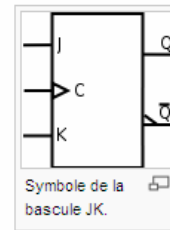
## Annexe1

### Bascule JK

- Pour  $J = K = 0$ , le signal d'horloge est sans effet, il y a conservation du dernier état logique pris par  $Q$  et  $\bar{Q}$  : il n'y a jamais de basculement.
- Pour  $J = K = 1$ , le système bascule à chaque front d'horloge (montant ou descendant selon les modèles).
- Pour  $J$  différent de  $K$ , la sortie  $Q$  recopie l'entrée  $J$  et la sortie  $\bar{Q}$  recopie l'entrée  $K$  à chaque front d'horloge.
- On utilise cette bascule pour faire des compteurs. On compte jusqu'à  $2^n$  avec  $n$  bascules à la suite et on compte dans l'ordre croissant avec des bascules à front descendant et dans l'ordre décroissant avec des bascules à front montant.

Table de vérité :

J	K	$Q_{n+1}$
0	0	$Q_n$
0	1	0
1	0	1
1	1	$\bar{Q}_n$



## Annexe2

### Bascule D (Delay)

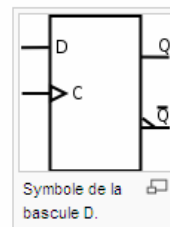
La bascule D (pour Delay) est une bascule JK à laquelle on a ajouté un inverseur entre les entrées  $J$  et  $K$ . Il y a donc une seule entrée, qui est notée  $D$  (pour Donnée ou *Data*). La table de vérité est la table de vérité d'une JK, limitée aux deux lignes  $J = 0, K = 1$  et  $J = 1, K = 0$ .

- La bascule D la plus simple possède 2 entrées (entrée  $D$  et l'horloge) et une sortie  $Q$ . À chaque front (ici montant) d'horloge,  $Q$  recopie l'entrée  $D$ .
- Parfois, un signal reset existe afin de pouvoir initialiser la valeur initiale de la bascule lors de la mise sous tension.
- Il existe des versions où les changements d'état ont lieu au moment des fronts descendants de l'horloge. Elles se signalent par une barre supplémentaire (en-dessous à 45°) sous l'entrée Clock.

Sa fonction est donc "mémoire" puisque l'information en entrée se retrouve en sortie après un "coup d'horloge" (un front).

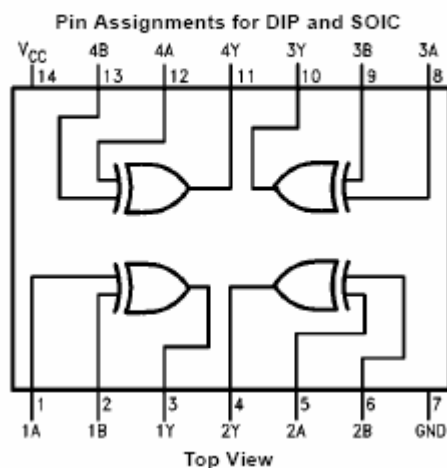
Table de vérité :

D	Ck	$Q_{n+1}$	$\bar{Q}_{n+1}$
0	↗	0	1
1	↗	1	0
X	0	$Q_n$	$\bar{Q}_n$



## Annexe 3 – Composant 74C86N

### Connection Diagram



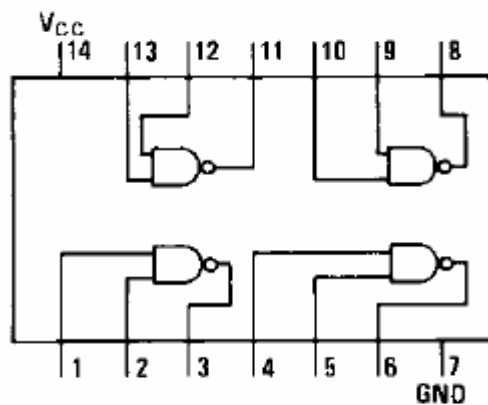
### Truth Table

Inputs		Output
A	B	Y
L	L	L
L	H	H
H	L	H
H	H	L

H = HIGH Level  
L = LOW Level

## Annexe 4 – Composant MM74C00N

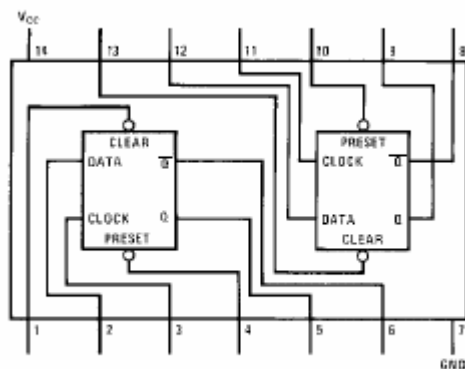
### MM74C00



Top View

## Annexe 5- Composant MM74C74M

### Connection Diagram



Note: A logic "0" on clear sets Q to logic "0".  
A logic "0" on preset sets Q to logic "1".

Top View

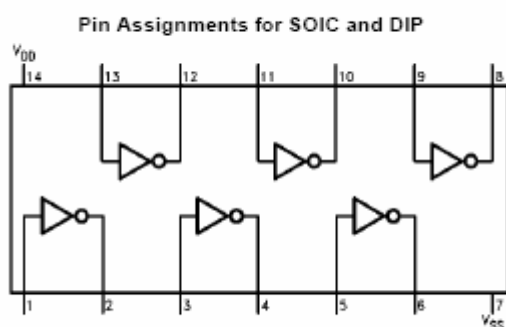
### Truth Table

Preset	Clear	$Q_n$	$\overline{Q}_n$
0	0	0	0
0	1	1	0
1	0	0	1
1	1	$Q_n$ (Note 1)	$\overline{Q}_n$ (Note 1)

Note 1: No change in output from previous state.

## Annexe 6 – Composant CD4069CN

### Connection Diagram





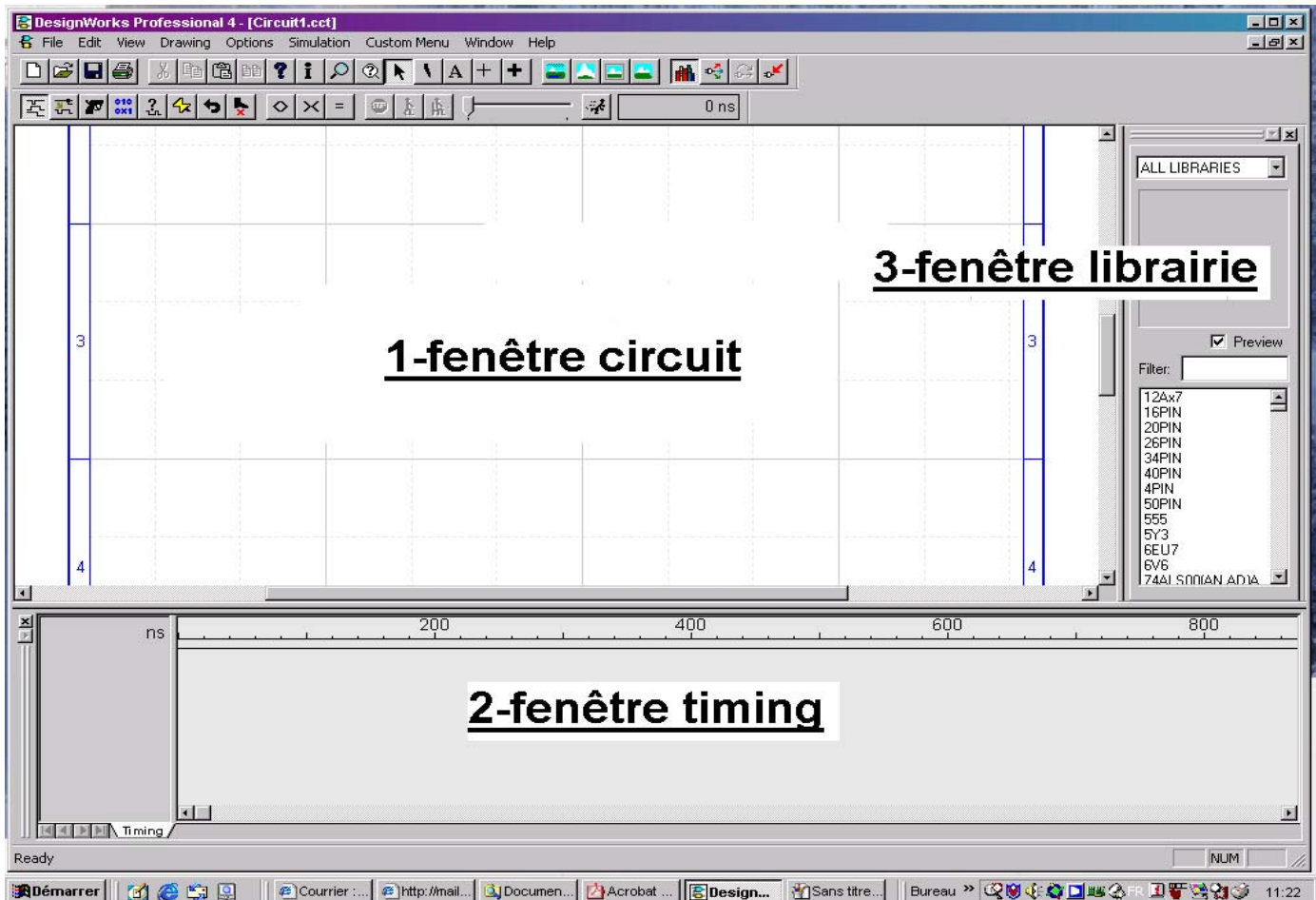
# PARTIE 2 : Simulation sur ordinateur

## I - Test d'une porte NAND

### 1. Activer le programme

Lancez « Designworks Professional 4 »

Validez **simulationNF02** → puis cliquez sur bouton **create**.



### 2. Les fenêtres de travail

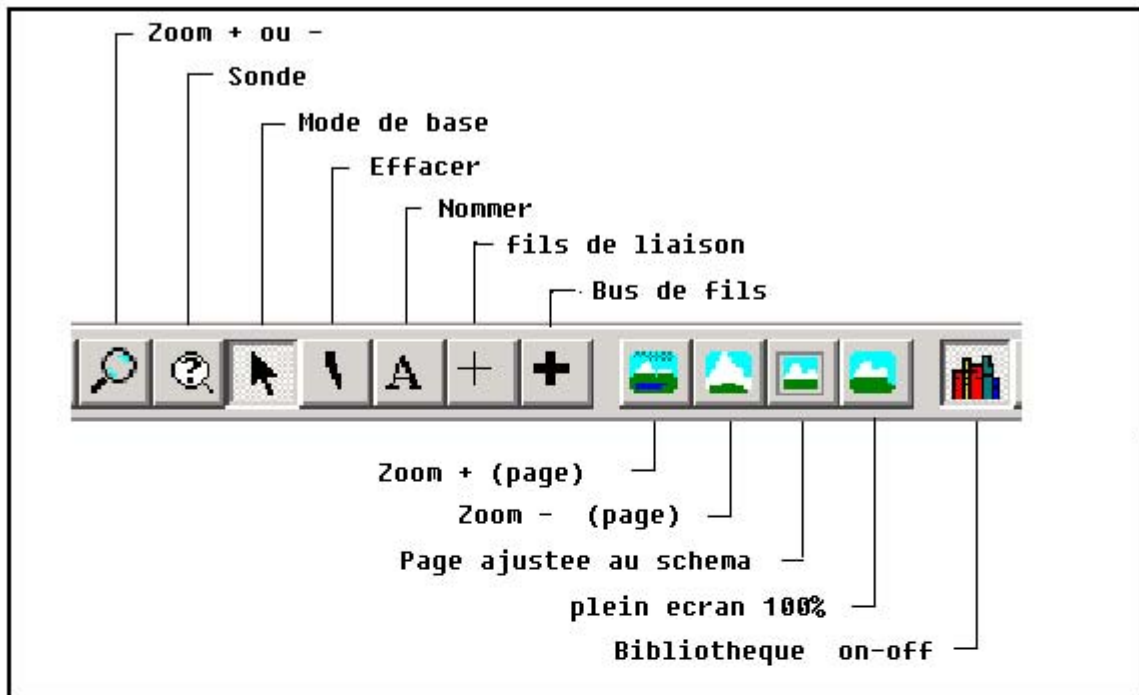
Fenêtre **circuit** (1): réalisation du schéma.

Fenêtre **timing** (2): dans laquelle apparaîtront les **chronogrammes** correspondant aux fils nommés.

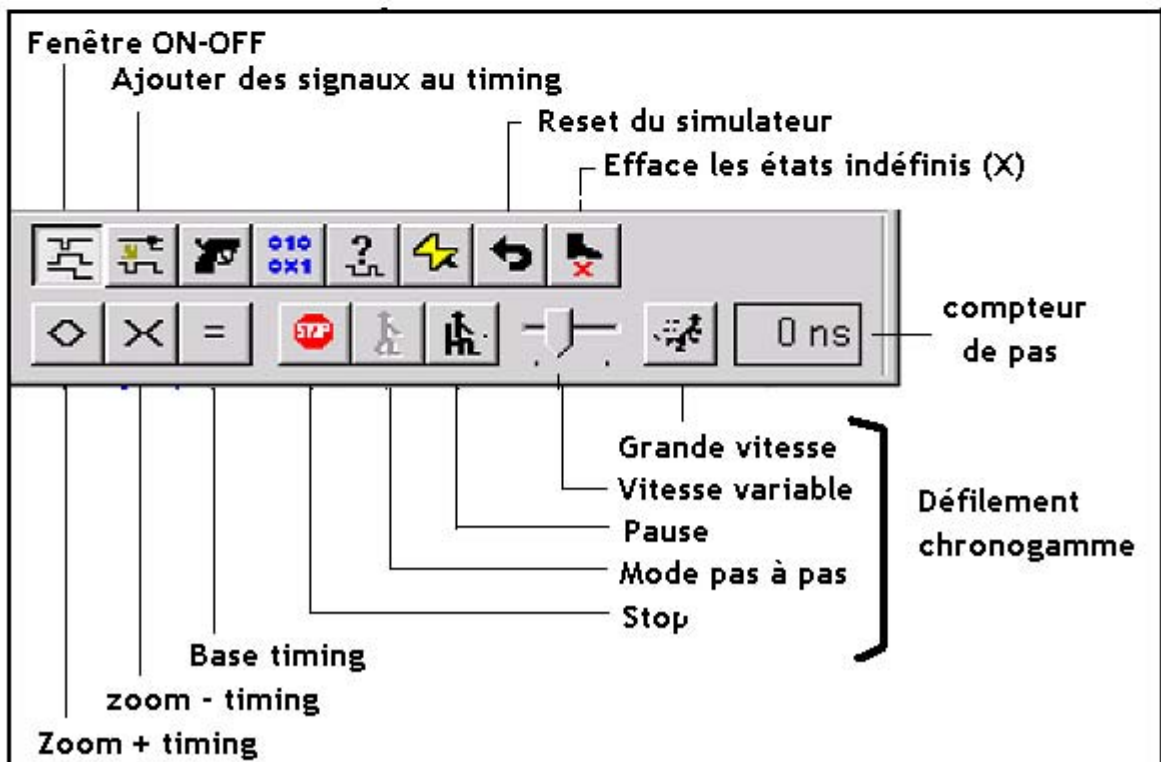
Fenêtre **librairie**(3): dans laquelle se situent les **composants**.

## FONCTION DES ICONES

### Icones schéma



### Icones timing



### 3. Aller chercher un composant

Dans la fenêtre librairie:

**ALL LIBRAIRIES → Simulation gates → NAND-3**

double-clic sur NAND-3

Déplacez le composant à l'endroit désiré dans la fenêtre circuit

Cliquez pour le fixer

Barre d'espace pour revenir au mode de base.(flèche)

Refaites de la même façon pour les autres composants.

All Libraries → simulation I/O → binary switch

" → " → Clock

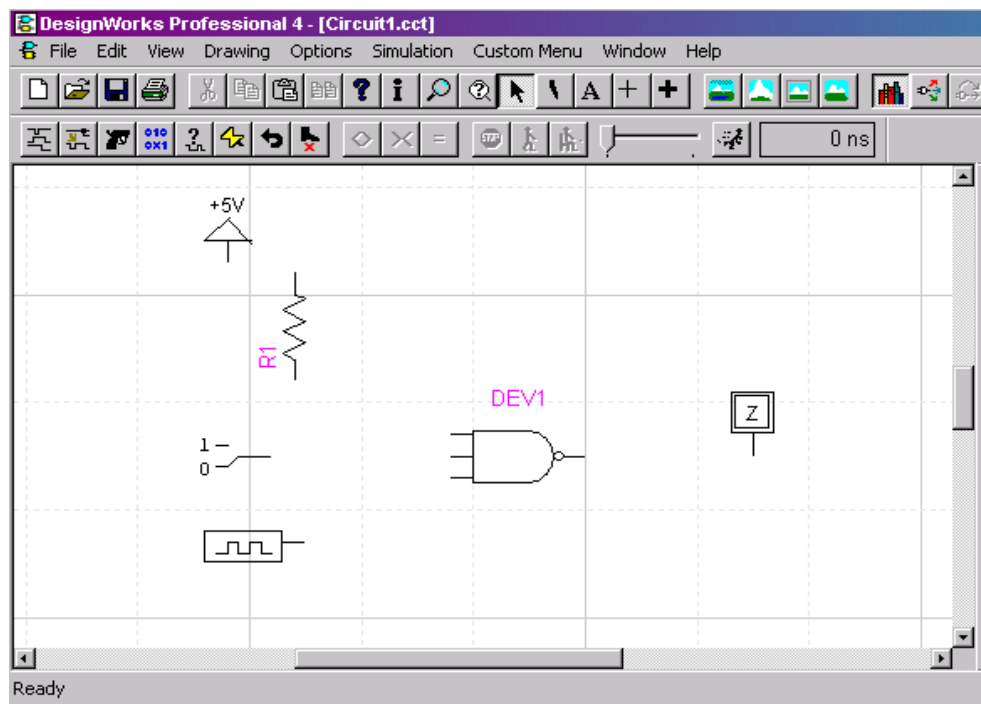
" → " → binary probe

" → discretres → res (résistance)

" → pseudo-devices → plus 5V

Pour modifier l'**orientation** d'un composant :

cliquez sur bouton gauche de la souris en pointant sur le composant → choix.



Réalisez la disposition indiquée sur la figure ci-dessus.

Pour certains composants comme switches, claviers etc... Il faut maintenir la touche **shift** enfoncée, pointer sur le composant, puis cliquer et maintenir pour sélectionner et déplacer.

#### 4. Ajouter des fils

Soit : . cliquez sur l'icône +. (icône schéma : fils de liaison)

. cliquez une fois au point de départ.

. cliquez une fois pour changer de direction.

. cliquez une fois pour arrêter .

. actionnez barre d'espace pour revenir au mode de base.

Soit : . directement avec la flèche placée sur l'entrée ou la sortie du composant, cliquez, maintenez et déplacez.

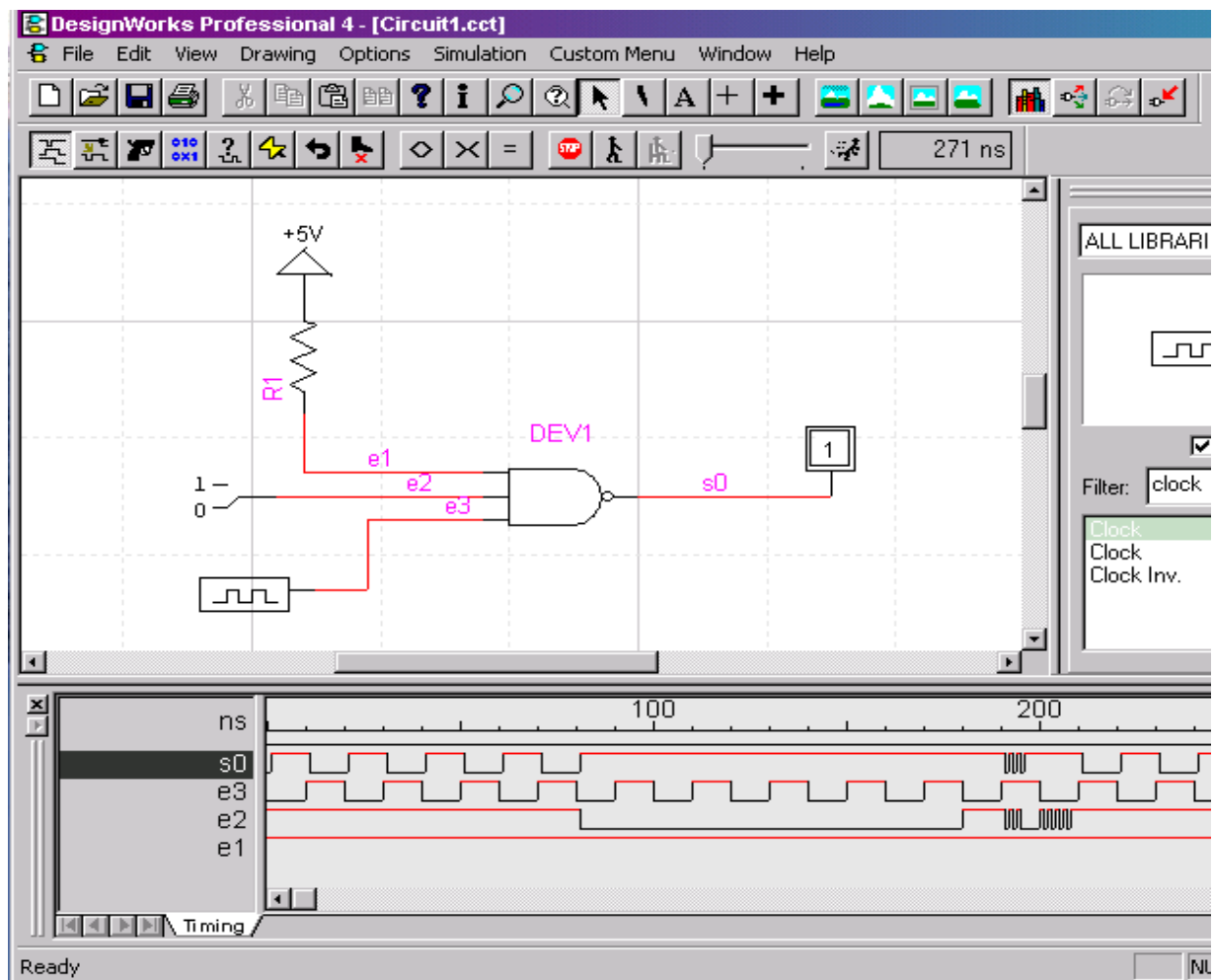
#### 5. Nommer les liaisons

Une liaison nommée pourra être visualisée sur le chronogramme.

. cliquez sur l'icône A → un crayon apparaît .

. pointez le fils et cliquez → SIG x apparaît.

. donnez un nom puis <return> (il doit apparaître dans la fenêtre timing).



Nommez les fils comme indiqué ci-dessus.(e1,e2,e3,s0).

## **6. Démarrer et stopper le défilement du chronogramme**

cliquez sur les icônes de défilement (ou voir le menu Simulation).  
stop.....pas à pas.....pause.....vitesse variable.....rapide.

Nota : si le montage présente des états indéfinis (X), activez l'icône d'effacement.  
S'ils persistent revoir le schéma : fils mal connectés par exemple ...

## **II – SIMULATION DES FONCTIONS CÂBLÉES EN PARTIE 1**

### **A) La porte NOR**

- . Dans File → close design (sans sauver les modifications).
- . Dans File → New
- Design → OK
- NF02 Simulation → create.

Réalisez le circuit et testez-le avec deux switches (binary switches) aux entrées et une sonde (binary probe) à la sortie.

### **B) Le compteur 2 bits**

Réalisez le compteur 2 bits à l'aide de 2 bascules D, une porte logique xor, une horloge et des + 5 V sur les Presets et Clears.

Bascules D :

All Libraries → 74LS\_C → 74LS74(AN,AD)A

Porte Xor :

All Libraries → simulation Logic → XOR-2



# TP 2 - Analyse de circuits séquentiels

## Partie 1 : Analyse d'une Unité Arithmétique et Logique

Le timing n'étant pas nécessaire dans cet exemple, réduisez sa fenêtre.

### 1. Pour trouver le composant

.soit : bibliothèque ----->74LS\_c ----->**74LS181(N,Dw)A**

.soit : choisissez **ALL LIBRAIRIES** : dans la fenêtre **Filter** , entrez 181, s'il existe une liste s'affiche.

### 2. Pour les entrées et les sorties

On dispose

. d'une simulation de **claviers hexadécimaux** (16 chiffres donc 4 sorties binaires).

Bibliothèque---->simulation I/O ---->**hex keyboard** à mettre sur A, B et S.

. d'**afficheurs hexadécimaux** (4 bits).

Simulation I/O-----> **hex display** à mettre sur F (un point sur l'afficheur indique le LSB).

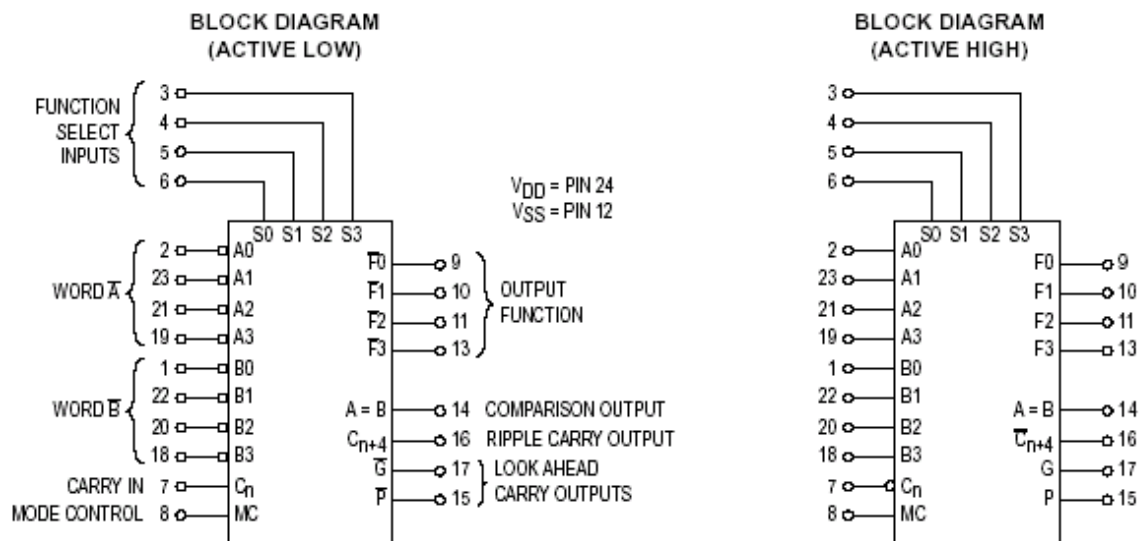
. de **binary switches (interrupteurs)** pour les entrées de contrôle Cn et M ,  
de **binary probe (points de test)** pour les sorties A=B, Cn +4, G et P.

### 3. Travail à effectuer

Faire les connexions et tester tout d'abord la fonction 1001 sur S0 -S3  
avec M=0 et Cn=1 puis 0.

Suivant le temps, tester d'autres fonctions de l'**UAL**, en particulier des fonctions logiques.

# SN54/74LS181



**FUNCTION TABLE**

MODE SELECT INPUTS				ACTIVE LOW INPUTS & OUTPUTS		ACTIVE HIGH INPUTS & OUTPUTS	
S <sub>3</sub>	S <sub>2</sub>	S <sub>1</sub>	S <sub>0</sub>	LOGIC (M = H)	ARITHMETIC** (M = L) (C <sub>n</sub> = L)	LOGIC (M = H)	ARITHMETIC** (M = L) (C <sub>n</sub> = H)
L	L	L	L	$\overline{A}$	A minus 1	$\overline{A}$	A
L	L	L	H	$\overline{AB}$	$\overline{AB}$ minus 1	$\overline{A + B}$	$A + \overline{B}$
L	L	H	L	$A + B$	$\overline{AB}$ minus 1	$\overline{AB}$	$A + B$
L	L	H	H	Logical 1 minus 1		Logical 0 minus 1	
L	H	L	L	$\overline{A + B}$	A plus $(A + \overline{B})$	$\overline{AB}$	A plus $\overline{AB}$
L	H	L	H	$\overline{B}$	$\overline{AB}$ plus $(A + B)$	$\overline{B}$	$(A + B)$ plus $\overline{AB}$
L	H	H	L	$\overline{A \oplus B}$	A minus B minus 1	$\overline{A \oplus B}$	A minus B minus 1
L	H	H	H	$\overline{A + B}$	A + B	$\overline{AB}$	$\overline{AB}$ minus 1
H	L	L	L	$\overline{AB}$	A plus $(A + B)$	$\overline{A + B}$	A plus $\overline{AB}$
H	L	L	H	$\overline{A \oplus B}$	$\overline{A}$ plus B	$\overline{A \oplus B}$	A plus $\overline{B}$
H	L	H	L	$\overline{B}$	$\overline{AB}$ plus $(A + B)$	$\overline{B}$	$(A + B)$ plus $\overline{AB}$
H	L	H	H	$\overline{A + B}$	A + B	$\overline{AB}$	$\overline{AB}$ minus 1
H	H	L	L	Logical 0	A plus A*	Logical 1	A plus A*
H	H	L	H	$\overline{AB}$	$\overline{AB}$ plus A	$\overline{A + B}$	$(A + \overline{B})$ plus A
H	H	H	L	$\overline{AB}$	$\overline{AB}$ plus A	$\overline{A + B}$	$(A + B)$ Plus A
H	H	H	H	A	A	A	A minus 1

L = LOW Voltage Level

H = HIGH Voltage Level

\*Each bit is shifted to the next more significant position

\*\*Arithmetic operations expressed in 2s complement notation



## TP 2

### Partie 2 : Analyse de circuits séquentiels


Pour ce qui suit, s'il y a un problème, activer l'icône timing qui efface les états indéfinis (X).

#### 1. Étude d'une bascule JK : 7476

- . Placer le composant, des switches sur les entrées J, K, CLR, PR , des points tests sur les sorties Q et Q', et une horloge sur CLK .
- . Nommer les lignes CLK , CLR et Q pour les faire apparaître sur le timing.
- . Vérifier la table de vérité et observer les valeurs de Q et Q' pour PR=CLR=0.

#### 2. Étude des compteurs 74LS160-74LS161

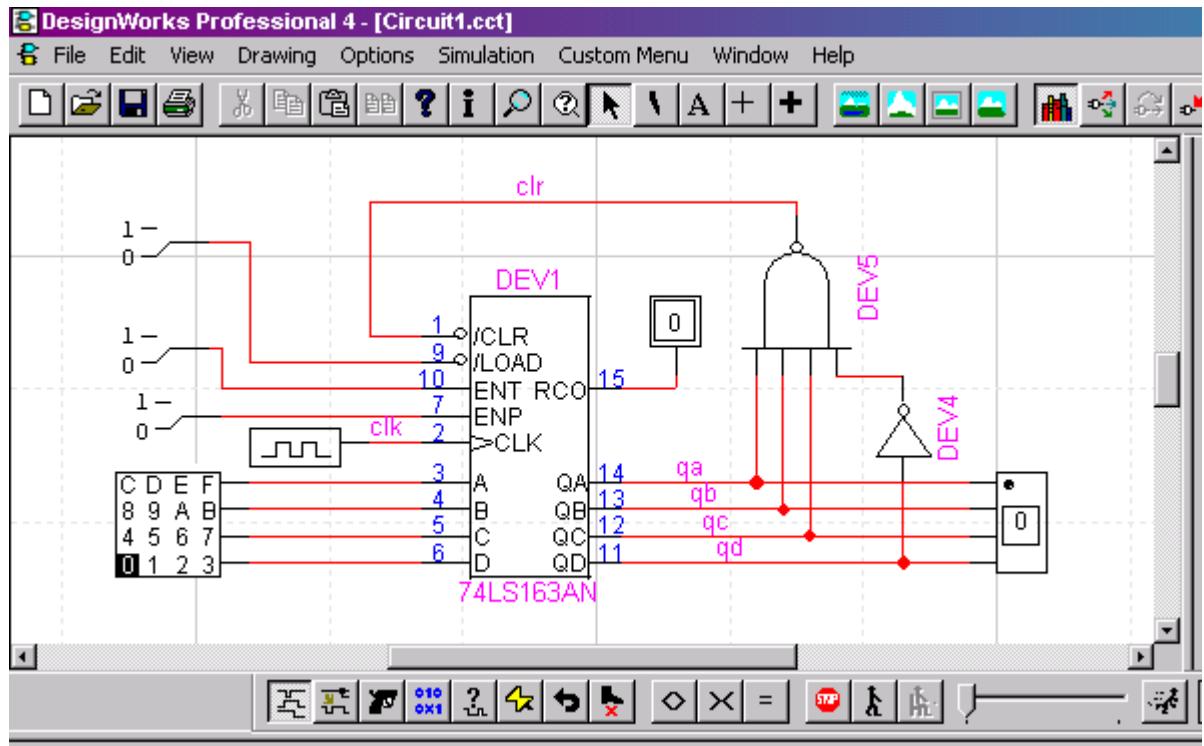
Fermer l'ancien circuit sans sauvegarde et ouvrir un nouveau circuit. Conserver la fenêtre timing.

- . Placer le composant 74160, une horloge sur l'entrée CLK, des switches sur P, T, LOAD et CLR, un clavier hexadécimal sur les entrées A-D, un afficheur hexa sur les sorties QA et QD, et un point test sur RCO.
- . Tester le fonctionnement global du circuit. Indiquer le rôle de RCO. Vérifier du LOAD et du CLR, quelle est la fonction prioritaire si on les active simultanément.
- . Remplacer le 74160 par un 74161
  - supprimer le circuit 74160
  - aller chercher le circuit 161
  - le replacer au même endroit que précédemment
  - actionner, si nécessaire, l'icône  qui efface les états indéfinis X.
- . Tester le fonctionnement de ce nouveau circuit.
- . Indiquer les différences de fonctionnement observées par rapport au circuit précédent.

### 3 . Test du synchronisme de CLR : 74LS161 et 74LS163

. Réaliser le schéma suivant, en conservant pour l'instant le 74LS161.

Rappel : pour l'orientation des composants : menu options--->orientation.



. Faire l'analyse du circuit et le faire fonctionner . Le CLR est-il synchrone dans ce cas ?

. Remplacer le 74LS161 par un **74LS163** (penser à initialiser les états indéfinis).

Quel est le nouveau cycle du compteur ?

Qu'en déduire quant à l'intérêt d' un CLR synchrone dans un circuit de ce type.

### 4 . Étude d'un registre à décalage : le circuit 74LS194.

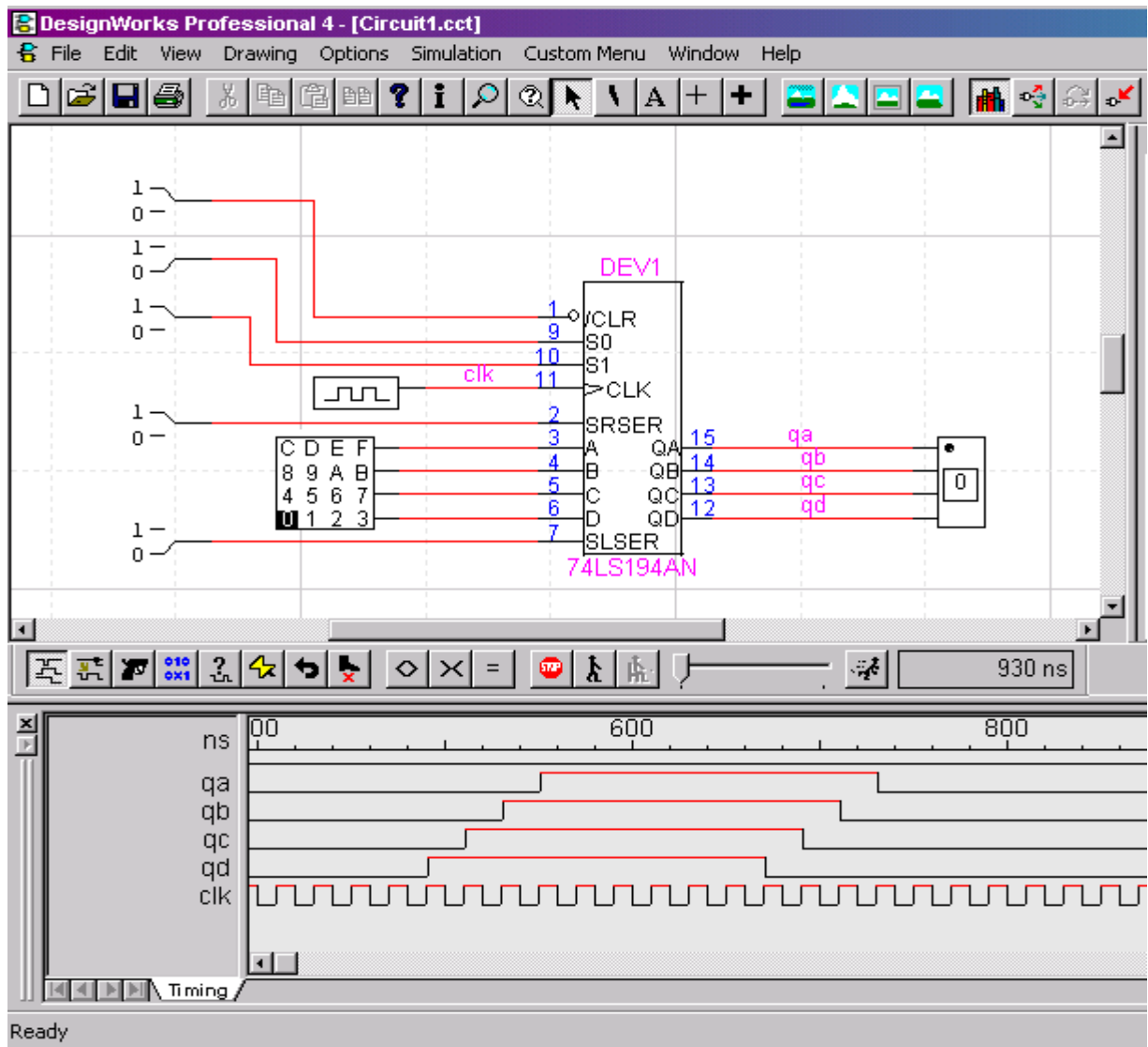
. Fermer l'ancien circuit sans sauvegarde et ouvrir un nouveau circuit. Conserver la fenêtre timing.

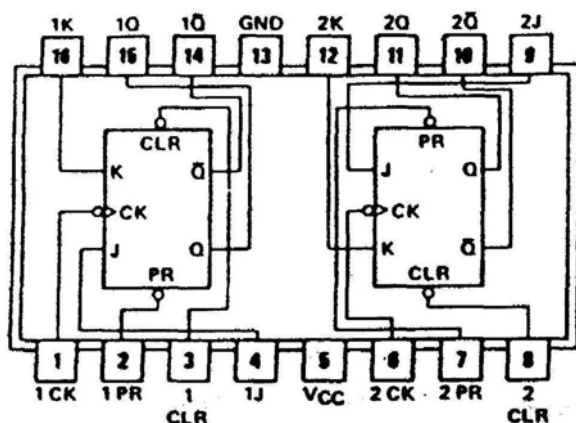
. Placer le composant 74LS194 ,une horloge sur l'entrée CLK, des switches sur S1, S0, SL (SLSER sur le schéma), SR (SRSER sur le schéma) et CLR, un clavier hexadécimal sur les entrées A ... D et un afficheur hexa sur les sorties QA ... QD.

. Nommer les lignes CLK, QA ... QD.

. Ajuster la taille de la fenêtre timing si nécessaire, pour ne voir que les cinq lignes du chronogramme.

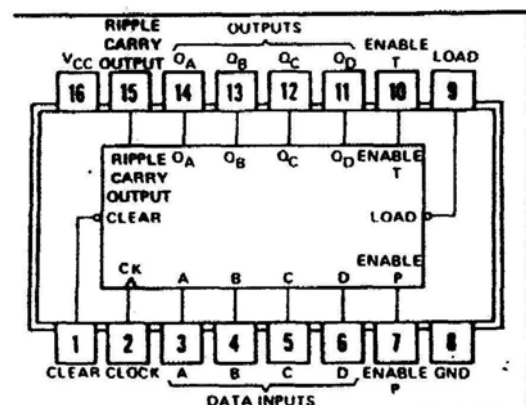
. Tester l'ensemble des fonctions de la table de vérité. Visualiser en particulier les décalages en ne changeant l'état des entrées série que pendant une ou deux périodes d'horloge.





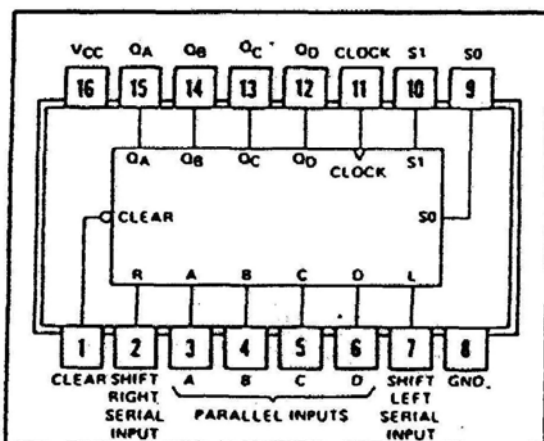
INPUTS					OUTPUTS	
PRESET	CLEAR	CLOCK	J	K	Q	$\bar{Q}$
L	H	X	X	X	H	L
H	L	X	X	X	L	H
L	L	X	X	X	H*	H*
H	H	↓	L	L	Q <sub>0</sub>	$\bar{Q}_0$
H	H	↓	H	L	H	L
H	H	↓	L	H	L	H
H	H	↓	H	H	TOGGLE	
H	H	H	X	X	Q <sub>0</sub>	$\bar{Q}_0$

CIRCUIT JK 7476



CLR	LOAD	T	P	Action on the Rising Clock Edge (⌋)
L	X	X	X	RESET (Clear)
H	L	X	X	LOAD (P <sub>n</sub> → Q <sub>n</sub> )
H	H	H	H	COUNT (Increment)
H	H	L	X	NO CHANGE (Hold)
H	H	X	L	NO CHANGE (Hold)

CIRCUITS COMPTEURS 74160-74163



INPUTS								OUTPUTS					
CLEAR	MODE		CLOCK	SERIAL		PARALLEL				Q <sub>A</sub>	Q <sub>B</sub>	Q <sub>C</sub>	Q <sub>D</sub>
	S <sub>1</sub>	S <sub>0</sub>		LEFT	RIGHT	A	B	C	D				
L	X	X	X	X	X	X	X	X	X	L	L	L	L
H	See note		L	X	X	X	X	X	X	Q <sub>A0</sub>	Q <sub>B0</sub>	Q <sub>C0</sub>	Q <sub>D0</sub>
H	H	H	↑	X	X	a	b	c	d	a	b	c	d
H	L	H	↑	X	H	X	X	X	X	H	Q <sub>An</sub>	Q <sub>Bn</sub>	Q <sub>Cn</sub>
H	L	H	↑	X	L	X	X	X	X	L	Q <sub>An</sub>	Q <sub>Bn</sub>	Q <sub>Cn</sub>
H	H	L	↑	H	X	X	X	X	X	Q <sub>Bn</sub>	Q <sub>Cn</sub>	Q <sub>Dn</sub>	H
H	H	L	↑	L	X	X	X	X	X	Q <sub>Bn</sub>	Q <sub>Cn</sub>	Q <sub>Dn</sub>	L
H	L	L	X	X	X	X	X	X	X	Q <sub>A0</sub>	Q <sub>B0</sub>	Q <sub>C0</sub>	Q <sub>D0</sub>

H = high level (steady state)  
 L = low level (steady state)  
 X = irrelevant (any input, including transitions)  
 ↑ = transition from low to high level  
 a, b, c, d = the level of steady-state input at inputs A, B, C, or D, respectively.  
 Q<sub>A0</sub>, Q<sub>B0</sub>, Q<sub>C0</sub>, Q<sub>D0</sub> = the level of Q<sub>A</sub>, Q<sub>B</sub>, Q<sub>C</sub>, or Q<sub>D</sub>, respectively, before the indicated steady-state input conditions were established.  
 Q<sub>An</sub>, Q<sub>Bn</sub>, Q<sub>Cn</sub>, Q<sub>Dn</sub> = the level of Q<sub>A</sub>, Q<sub>B</sub>, Q<sub>C</sub>, or Q<sub>D</sub>, respectively, before the most recent ↑ transition of the clock.

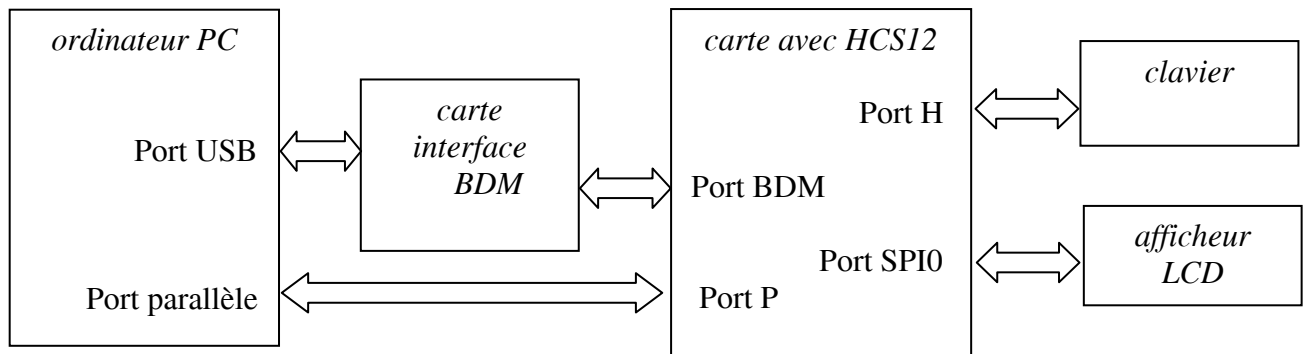
CIRCUIT REGISTRE A DECALAGE 74194

## TP 3 – Outil de développement pour microcontrôleur HCS12

### Objectifs :

- maîtriser l'environnement de développement CodeWarrior
- charger un projet, compiler les sources et utiliser le débogueur

Les TPs 3 à 6 s'inscrivent dans l'étude du microcontrôleur HCS12. Ils permettront d'étudier le système suivant :



TP3 : CodeWarrior

TP6 : interruption via le port parallèle

TP5 : clavier et afficheur

TP4 : sous-programme

### I. Outil CodeWarrior

L'outil CodeWarrior est un outil complet qui permet l'édition de texte, l'assemblage et le test (débogage).

#### 1.1 Lancement de l'outil

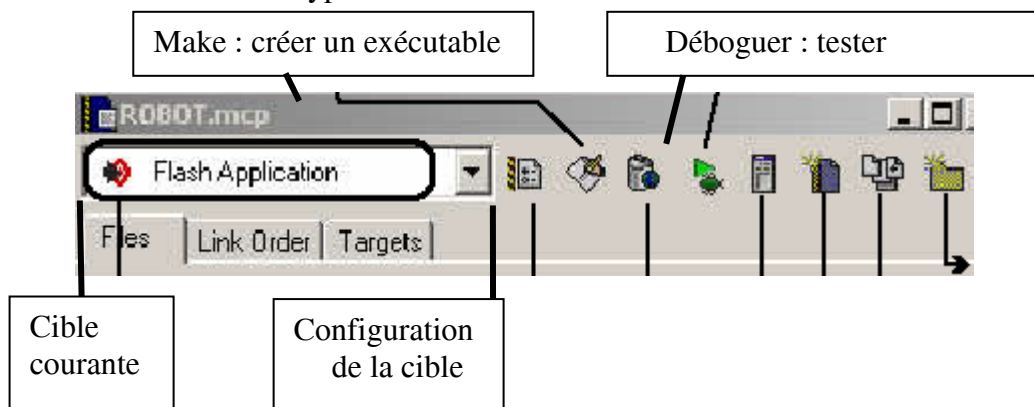
- Double-cliquez sur l'icône CodeWarrior du bureau.
- Une fenêtre s'ouvre avec une barre principale de menus suivante :



#### 1.2 Ouverture d'un projet de base

- Cliquez sur menu File → Open et choisissez le fichier projet *Projet\_asm.mcp*.

Une autre fenêtre s'ouvre de type suivant :



La cible (*target*) est ici une carte inDART-HCS12 du constructeur Softec. Elle permet une interface d'émulation et de contrôle de la carte à microcontrôleur HCS12 : on parle alors de mode BDM (*background debug mode*).

### 1.3 Édition du fichier source *main.asm*

- Dans la fenêtre Projet et l'onglet Files, développez les dossiers.
- Repérez le fichier *main.asm* et double-cliquez dessus.

- La déclaration INCLUDE 'mc9s12dp256.inc' permet d'inclure le fichier « *mc9s12dp256.inc* » où sont reportées les équivalences entre symboles et adresses physiques de mémoire. La valeur de registres, d'emplacement des ports sont donc spécifiés dans ce fichier « include »

- Une allocation mémoire des données est faite par la ligne : temp\_byte DS.B 1  
La directive DS.B 1 réserve un octet en mémoire.

- Toutes le code qui suit l'instruction ORG XXXX sera placé en XXXX dans la mémoire du microcontrôleur

- La variable temp\_byte sera donc placée en \$2000 dans la mémoire
- L'instruction MOVB #1, temp\_byte sera placée en \$1000 dans la mémoire et les instructions suivantes (NOP et BRA) seront placées à la suite dans la mémoire
- Ici le point de démarrage du programme est indiqué par *Entry*
  - Donc le programme *main.asm* débute à l'adresse \$1000

Pour écrire des commentaires dans le programme, il faut commencer le texte de commentaire par le caractère « ; ».

#### Description du programme :

Instructions :

MOVB #1,temp\_byte :

La valeur 1 va être écrite dans la variable temp\_byte (en \$2000).

NOP

L'instruction NOP n'a pas d'effet ; elle dure un cycle d'horloge.

BRA Entry

C'est un branchement non conditionnel vers l'étiquette « Entry ».

En raison du branchement « BRA Entry », le programme va exécuter une boucle infinie. À l'intérieur de cette boucle, on va écrire la valeur 1 dans temp\_byte.

### 1.4 Génération de l'exécutable

- Dans le menu de la fenêtre principale, cliquez sur Project → Make, ou tapez sur la touche F7 du clavier.
- Si rien de particulier ne se passe à l'écran, c'est que le programme a été compilé sans erreur. Dans le cas contraire une liste d'erreur est affichée.

## 1.5 Lancement du débogueur








- Dans le menu de la fenêtre principale, cliquez sur Project → Debug, ou tapez sur la touche F5 du clavier. Pour la configuration MCU, choisissez le modèle inDART-HCS12 et le « device » MC9S12DP256B.

Le débogueur se met en route, se connecte à la carte Softec puis à la carte HCS12 pour charger le programme en mémoire. Le processus est relativement long : attendez la fin du processus avant de poursuivre.

## 1.6 Contrôle de l'exécution

L'exécution peut être contrôlée par l'emploi de la barre d'outils :



	[F10]	Step Over	Avance d'une ligne de programme sans rentrer dans les fonctions
	[F11]	Single Step	Avance d'une ligne de programme en rentrant dans les fonctions
	[MAJ+F11]	Step Out	Finir la fonction en cours
	[CTRL+F11]	Assembly Step	Exécuter une instruction assembleur
	[F5]	Start / Continue	Exécuter le programme
	[F6]	Halt	Arrêter l'exécution
	[CTRL+R]	Reset Target	Redémarrer le programme

## 1.7 Test du programme, vérification de l'état de la mémoire

Les fenêtres Memory et Register représentent l'état respectif de la mémoire et des registres. Pour la mémoire, la colonne de gauche représente les adresses. Les autres colonnes représentent des données. Par défaut, le contenu de la mémoire est présenté en hexadécimal, mais un menu contextuel (bouton droit) permet de choisir d'autres formats d'affichage.

Pour effectuer un test du programme :

Etape 1 : on effectue un RESET pour se placer au début du programme :



Etape 2 : on se place en \$2000 dans la mémoire, c'est là où a été placé la variable temp\_byte :

- Sélectionnez la fenêtre Memory
- Bouton droit puis « Adress... »
- Choisissez la valeur 2000.

Etape 3 : on écrit la valeur « FF » dans temp\_byte :

- Dans la fenêtre Memory, double cliquez sur le premier octet de données (les données de la mémoire sont affichées en valeurs hexadécimales)
- Indiquez la valeur « FF » et validez.

Etape 4 : On exécute le programme en mode pas à pas  :


- Vérifiez que la valeur en \$2000 passe de « FF » à « 01 »
- Lancez plusieurs fois le mode pas à pas
- Vérifiez que le BRA Entry effectue le bon branchement
- Observez l'évolution du registre PC

### 1.8 Placement de point d'arrêt

- Pour utiliser un point d'arrêt :

Sélectionnez la fenêtre « Source ». Avec le menu contextuel (bouton droit), placez un point d'arrêt (Set Breakpoint) sur la ligne souhaitée.

Le point d'arrêt se matérialise par une flèche.

Lancez le programme  et vérifiez qu'il s'arrête uniquement sur le point d'arrêt (par exemple en observant le contenu du registre PC).

- Pour retirer un point d'arrêt :

Sélectionnez la ligne où se trouve le point d'arrêt, puis Bouton droit et delete Breakpoint

## II. Création et modification de programmes

### Programme n°1 :

Remplacez dans *main.asm* le programme actuel (placé après l'étiquette Entry) par le programme suivant :

Charger le registre A avec la valeur hexadécimale 1F  
Ecrire la valeur hexadécimale 2F à l'adresse 2000  
Charger le registre B avec le contenu à l'adresse 2000  
Effectuer un 'MUL'

Note : Avant une valeur numérique il est possible de placer un # ou un \$

- Si vous avez un #, le nombre indiqué représente une valeur
- Si vous n'avez pas de #, le nombre indiqué représente une adresse
- Si vous avez un \$, le nombre indiqué est en hexadécimal
- Si vous n'avez pas de \$, le nombre indiqué est en décimal

Compilez le programme, puis exécutez-le en mode pas à pas.

- Quelle est la valeur de D à chaque nouvelle ligne exécutée ?
- Quelle est la relation entre A,B et D ?

Utilisez la calculatrice Windows en mode scientifique pour vérifier MUL.

- Comment fonctionne l'instruction MUL ?

### Programme n°2 :

Ecrivez maintenant le programme suivant (attention à respecter les tabulations).

```
    ORG $1500
NumX    ds.b 1
    ORG $1000
Entry:
    INC NumX
    LDAB #1
```



Boucle:

```
DEC NumX
BEQ Fin
LDAA NumX
MUL
BRA Boucle
```

Fin:

```
STAB NumX
BGND
```

Compilez-le. Chargez le programme dans le microcontrôleur, puis écrivez la valeur 5 sur l'octet situé en 1500 (donc dans NumX). Exécutez le programme en mode pas à pas en observant l'évolution des registres A et B (vous pourrez notamment noter la valeur de B à chaque passage sur l'instruction BRA Boucle).

- Quelle instruction modifie le flag 'Z' testé à l'instruction **BEQ Fin** ? (voir note ci-dessous)
- Quel est le but de ce programme ?

Question : Cette fonction peut être optimisée. Réorganisez le programme notamment en passant de 2 branchements (BRA et BEQ) à un seul branchement.

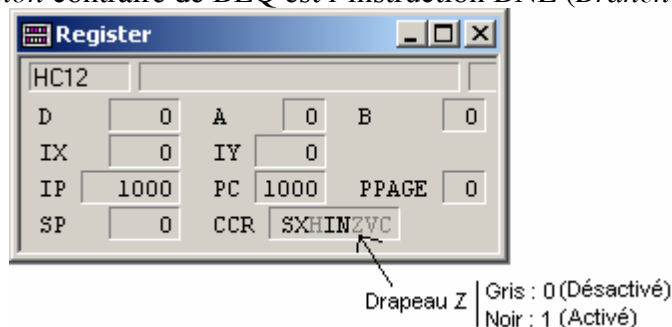
Il est ainsi possible de réduire la taille du programme de 2 lignes. Vérifiez que le résultat obtenu est identique à celui de la fonction initiale.

Note : Fonctionnement du branchement conditionnel BEQ (Branch if equal)

- L'instruction BEQ effectue un branchement uniquement lorsque le drapeau 'Z' vaud 1.
- Le drapeau 'Z' est le drapeau 'Zéro' qui est activé par certaines instructions.

Exemple :

- Si le registre A vaut 10, **CMPA #\$10** active le drapeau 'Z'
- Si le registre A vaut 1, **DECA** active le drapeau Z car A passe à 0
- L'instruction contraire de BEQ est l'instruction BNE (Branch if not equal)



Question supplémentaire :

On ne souhaite plus utiliser NumX, mais uniquement des registres. En utilisant A, B et D refaites la même fonction.

Vous pourrez sauvegarder A dans la pile avec PSHA et le récupérer de la pile avec PULA.

Note : Pour initialiser la pile, effectuez l'instruction LDS #\$3000 en début de programme.

## Annexe

### Directive assembleur

Il existe des directives qui permettent à l'assembleur certaines opérations autres que les instructions du HCS12. Les principales directives sont listées ci-après (pour plus d'informations, consultez le guide assembleur disponible sur la rubrique TP du site).

Directive	Description
ORG	Définit le début d'une section en mémoire
EQU	Assigne un nom à une expression (adresse)
DC.B ou FCB	Définit une variable constante en octet (byte)
DCB	Définit un bloc constant
DS.B ou RMB	Définit un espace mémoire en octet (byte)
ABSENTRY	Définit un point d'entrée de début de programme
INCLUDE	Inclut du texte à partir d'un fichier
END	Fin de programme utilisateur

# Appendix A. Instruction Reference

## A.1 Introduction

This appendix provides quick references for the instruction set, opcode map, and encoding.

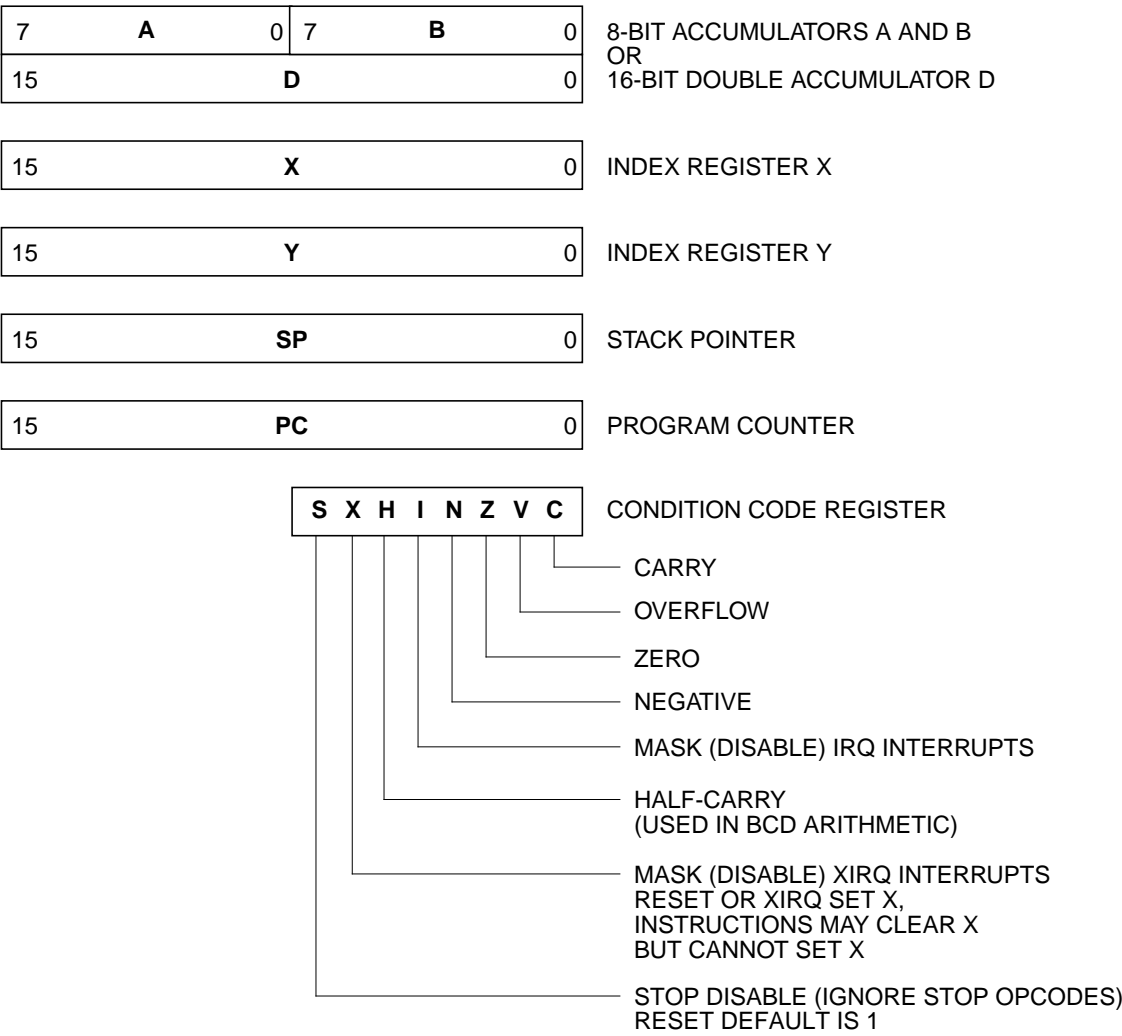
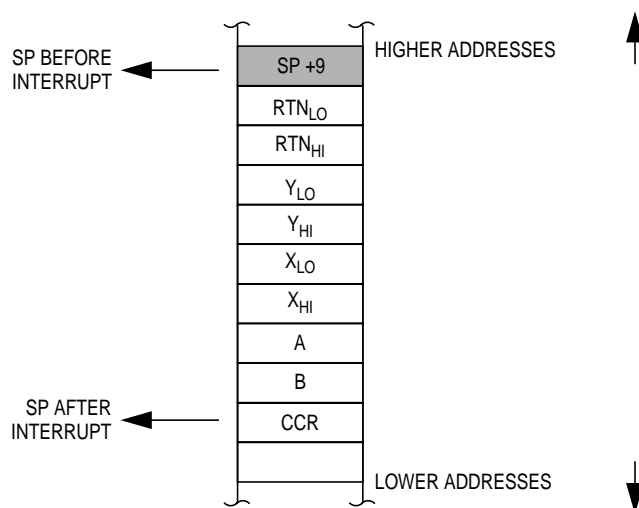


Figure A-1. Programming Model

## A.2 Stack and Memory Layout



STACK UPON ENTRY TO SERVICE ROUTINE  
IF SP WAS ODD BEFORE INTERRUPT

SP +8	RTN <sub>LO</sub>		SP +9
SP +6	Y <sub>LO</sub>	RTN <sub>HI</sub>	SP +7
SP +4	X <sub>LO</sub>	Y <sub>HI</sub>	SP +5
SP +2	A	X <sub>HI</sub>	SP +3
SP	CCR	B	SP +1
SP -2			SP -1

STACK UPON ENTRY TO SERVICE ROUTINE  
IF SP WAS EVEN BEFORE INTERRUPT

SP +9			SP +10
SP +7	RTN <sub>HI</sub>	RTN <sub>LO</sub>	SP +8
SP +5	Y <sub>HI</sub>	Y <sub>LO</sub>	SP +6
SP +4	X <sub>HI</sub>	X <sub>LO</sub>	SP +4
SP +1	B	A	SP +2
SP -1		CCR	SP

## A.3 Interrupt Vector Locations

\$FFFE, \$FFFF	Power-On (POR) or External Reset
\$FFFC, \$FFFD	Clock Monitor Reset
\$FFFA, \$FFFB	Computer Operating Properly (COP Watchdog Reset
\$FFF8, \$FFF9	Unimplemented Opcode Trap
\$FFF6, \$FFF7	Software Interrupt Instruction (SWI)
\$FFF4, \$FFF5	XIRQ
\$FFF2, \$FFF3	IRQ
\$FFC0-\$FFF1 (M68HC12)	Device-Specific Interrupt Sources
\$FF00-\$FFF1 (HCS12)	Device-Specific Interrupt Sources

## A.4 Notation Used in Instruction Set Summary

### CPU Register Notation

Accumulator A — A or a	Index Register Y — Y or y
Accumulator B — B or b	Stack Pointer — SP, sp, or s
Accumulator D — D or d	Program Counter — PC, pc, or p
Index Register X — X or x	Condition Code Register — CCR or c

### Explanation of Italic Expressions in Source Form Column

<i>abc</i>	— A or B or CCR
<i>abcdxys</i>	— A or B or CCR or D or X or Y or SP. Some assemblers also allow T2 or T3.
<i>abd</i>	— A or B or D
<i>abdxys</i>	— A or B or D or X or Y or SP
<i>dxys</i>	— D or X or Y or SP
<i>msk8</i>	— 8-bit mask, some assemblers require # symbol before value
<i>opr8i</i>	— 8-bit immediate value
<i>opr16i</i>	— 16-bit immediate value
<i>opr8a</i>	— 8-bit address used with direct address mode
<i>opr16a</i>	— 16-bit address value
<i>opr0_xysp</i>	— Indexed addressing postbyte code:
<i>opr3,-xys</i>	— Predecrement X or Y or SP by 1 . . . 8
<i>opr3,+xys</i>	— Preincrement X or Y or SP by 1 . . . 8
<i>opr3,xys-</i>	— Postdecrement X or Y or SP by 1 . . . 8
<i>opr3,xys+</i>	— Postincrement X or Y or SP by 1 . . . 8
<i>opr5,xysp</i>	— 5-bit constant offset from X or Y or SP or PC
<i>abd,xysp</i>	— Accumulator A or B or D offset from X or Y or SP or PC
<i>opr3</i>	— Any positive integer 1 . . . 8 for pre/post increment/decrement
<i>opr5</i>	— Any integer in the range -16 . . . +15
<i>opr9</i>	— Any integer in the range -256 . . . +255
<i>opr16</i>	— Any integer in the range -32,768 . . . 65,535
<i>page</i>	— 8-bit value for PPAGE, some assemblers require # symbol before this value
<i>rel8</i>	— Label of branch destination within -128 to +127 locations
<i>rel9</i>	— Label of branch destination within -256 to +255 locations
<i>rel16</i>	— Any label within 64K memory space
<i>trapnum</i>	— Any 8-bit integer in the range \$30-\$39 or \$40-\$FF
<i>xys</i>	— X or Y or SP
<i>xysp</i>	— X or Y or SP or PC

### Operators

+	— Addition
-	— Subtraction
•	— Logical AND
+	— Logical OR (inclusive)

Continued on next page

## Operators (continued)

- ⊕ — Logical exclusive OR
- × — Multiplication
- ÷ — Division
- $\overline{M}$  — Negation. One's complement (invert each bit of M)
- : — Concatenate  
Example: A : B means the 16-bit value formed by concatenating 8-bit accumulator A with 8-bit accumulator B.  
A is in the high-order position.
- ⇒ — Transfer  
Example: (A) ⇒ M means the content of accumulator A is transferred to memory location M.
- ↔ — Exchange  
Example: D ↔ X means exchange the contents of D with those of X.

## Address Mode Notation

- INH — Inherent; no operands in object code
- IMM — Immediate; operand in object code
- DIR — Direct; operand is the lower byte of an address from \$0000 to \$00FF
- EXT — Operand is a 16-bit address
- REL — Two's complement relative offset; for branch instructions
- IDX — Indexed (no extension bytes); includes:  
5-bit constant offset from X, Y, SP, or PC  
Pre/post increment/decrement by 1 . . . 8  
Accumulator A, B, or D offset
- IDX1 — 9-bit signed offset from X, Y, SP, or PC; 1 extension byte
- IDX2 — 16-bit signed offset from X, Y, SP, or PC; 2 extension bytes
- [IDX2] — Indexed-indirect; 16-bit offset from X, Y, SP, or PC
- [D, IDX] — Indexed-indirect; accumulator D offset from X, Y, SP, or PC

## Machine Coding

- dd — 8-bit direct address \$0000 to \$00FF. (High byte assumed to be \$00).
- ee — High-order byte of a 16-bit constant offset for indexed addressing.
- eb — Exchange/Transfer post-byte. See [Table A-5](#) on page 405.
- ff — Low-order eight bits of a 9-bit signed constant offset for indexed addressing, or low-order byte of a 16-bit constant offset for indexed addressing.
- hh — High-order byte of a 16-bit extended address.
- ii — 8-bit immediate data value.
- jj — High-order byte of a 16-bit immediate data value.
- kk — Low-order byte of a 16-bit immediate data value.
- lb — Loop primitive (DBNE) post-byte. See [Table A-6](#) on page 406.
- ll — Low-order byte of a 16-bit extended address.

- mm — 8-bit immediate mask value for bit manipulation instructions.  
Set bits indicate bits to be affected.
- pg — Program page (bank) number used in CALL instruction.
- qq — High-order byte of a 16-bit relative offset for long branches.
- tn — Trap number \$30–\$39 or \$40–\$FF.
- rr — Signed relative offset \$80 (–128) to \$7F (+127).  
Offset relative to the byte following the relative offset byte, or  
low-order byte of a 16-bit relative offset for long branches.
- xb — Indexed addressing post-byte. See [Table A-3](#) on page 403  
and [Table A-4](#) on page 404.

#### Access Detail

Each code letter except (,), and comma equals one CPU cycle. Uppercase = 16-bit operation and lowercase = 8-bit operation. For complex sequences see the *CPU12 Reference Manual* (CPU12RM/AD) for more detailed information.

- f — Free cycle, CPU doesn't use bus
- g — Read PPAGE internally
- I — Read indirect pointer (indexed indirect)
- i — Read indirect PPAGE value (CALL indirect only)
- n — Write PPAGE internally
- O — Optional program word fetch (P) if instruction is misaligned and has  
an odd number of bytes of object code — otherwise, appears as  
a free cycle (f); Page 2 prebyte treated as a separate 1-byte instruction
- P — Program word fetch (always an aligned-word read)
- r — 8-bit data read
- R — 16-bit data read
- s — 8-bit stack write
- S — 16-bit stack write
- w — 8-bit data write
- W — 16-bit data write
- u — 8-bit stack read
- U — 16-bit stack read
- V — 16-bit vector fetch (always an aligned-word read)
- t — 8-bit conditional read (or free cycle)
- T — 16-bit conditional read (or free cycle)
- x — 8-bit conditional write (or free cycle)
- ( ) — Indicate a microcode loop
- , — Indicates where an interrupt could be honored

#### Special Cases

- PPP/P — Short branch, PPP if branch taken, P if not
- OPPP/OPO — Long branch, OPPP if branch taken, OPO if not

### Condition Codes Columns

- — Status bit not affected by operation.
- 0 — Status bit cleared by operation.
- 1 — Status bit set by operation.
- Δ — Status bit affected by operation.
- fl — Status bit may be cleared or remain set, but is not set by operation.
- ↑ — Status bit may be set or remain cleared, but is not cleared by operation.
- ? — Status bit may be changed by operation but the final state is not defined.
- ! — Status bit used for a special purpose.

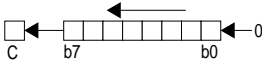
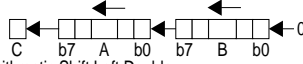



Table A-1. Instruction Set Summary (Sheet 1 of 14)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail		S X H I	N Z V C
				HCS12	M68HC12		
ABA	$(A) + (B) \Rightarrow A$ Add Accumulators A and B	INH	18 06	00	00	--Δ-	ΔΔΔΔ
ABX	$(B) + (X) \Rightarrow X$ <i>Translates to LEAX B,X</i>	IDX	1A E5	Pf	PP <sup>1</sup>	----	----
ABY	$(B) + (Y) \Rightarrow Y$ <i>Translates to LEAY B,Y</i>	IDX	19 ED	Pf	PP <sup>1</sup>	----	----
ADCA #opr8i ADCA opr8a ADCA opr16a ADCA oprx0_xysp ADCA oprx9_xysp ADCA oprx16_xysp ADCA [D,xysp] ADCA [oprx16,xysp]	$(A) + (M) + C \Rightarrow A$ Add with Carry to A	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	89 ii 99 dd B9 hh 11 A9 xb A9 xb ff A9 xb ee ff A9 xb A9 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rPf rOP rPf rPO frPP fIfrPf fIPrPf	--Δ-	ΔΔΔΔ
ADCB #opr8i ADCB opr8a ADCB opr16a ADCB oprx0_xysp ADCB oprx9_xysp ADCB oprx16_xysp ADCB [D,xysp] ADCB [oprx16,xysp]	$(B) + (M) + C \Rightarrow B$ Add with Carry to B	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C9 ii D9 dd F9 hh 11 E9 xb E9 xb ff E9 xb ee ff E9 xb E9 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rPf rOP rPf rPO frPP fIfrPf fIPrPf	--Δ-	ΔΔΔΔ
ADDA #opr8i ADDA opr8a ADDA opr16a ADDA oprx0_xysp ADDA oprx9_xysp ADDA oprx16_xysp ADDA [D,xysp] ADDA [oprx16,xysp]	$(A) + (M) \Rightarrow A$ Add without Carry to A	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	8B ii 9B dd BB hh 11 AB xb AB xb ff AB xb ee ff AB xb AB xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rPf rOP rPf rPO frPP fIfrPf fIPrPf	--Δ-	ΔΔΔΔ
ADDB #opr8i ADDB opr8a ADDB opr16a ADDB oprx0_xysp ADDB oprx9_xysp ADDB oprx16_xysp ADDB [D,xysp] ADDB [oprx16,xysp]	$(B) + (M) \Rightarrow B$ Add without Carry to B	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	CB ii DB dd FB hh 11 EB xb EB xb ff EB xb ee ff EB xb EB xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rPf rOP rPf rPO frPP fIfrPf fIPrPf	--Δ-	ΔΔΔΔ
ADDD #opr16i ADDD opr8a ADDD opr16a ADDD oprx0_xysp ADDD oprx9_xysp ADDD oprx16_xysp ADDD [D,xysp] ADDD [oprx16,xysp]	$(A:B) + (M:M+1) \Rightarrow A:B$ Add 16-Bit to D (A:B)	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C3 jj kk D3 dd F3 hh 11 E3 xb E3 xb ff E3 xb ee ff E3 xb E3 xb ee ff	PO RPf RPO RPf RPO frPP fIfrPf fIPrPf	OP RfP ROP RfP RPO frPP fIfrPf fIPrPf	----	ΔΔΔΔ
ANDA #opr8i ANDA opr8a ANDA opr16a ANDA oprx0_xysp ANDA oprx9_xysp ANDA oprx16_xysp ANDA [D,xysp] ANDA [oprx16,xysp]	$(A) \bullet (M) \Rightarrow A$ Logical AND A with Memory	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	84 ii 94 dd B4 hh 11 A4 xb A4 xb ff A4 xb ee ff A4 xb A4 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rPf rOP rPf rPO frPP fIfrPf fIPrPf	----	ΔΔ0-
ANDB #opr8i ANDB opr8a ANDB opr16a ANDB oprx0_xysp ANDB oprx9_xysp ANDB oprx16_xysp ANDB [D,xysp] ANDB [oprx16,xysp]	$(B) \bullet (M) \Rightarrow B$ Logical AND B with Memory	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C4 ii D4 dd F4 hh 11 E4 xb E4 xb ff E4 xb ee ff E4 xb E4 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rPf rOP rPf rPO frPP fIfrPf fIPrPf	----	ΔΔ0-
ANDCC #opr8i	$(CCR) \bullet (M) \Rightarrow CCR$ Logical AND CCR with Memory	IMM	10 ii	P	P	↓↓↓↓	↓↓↓↓

Note 1. Due to internal CPU requirements, the program word fetch is performed twice to the same address during this instruction.

**Table A-1. Instruction Set Summary (Sheet 2 of 14)**

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail		S X H I	N Z V C
				HCS12	M68HC12		
ASL <i>opr16a</i> ASL <i>opr0_xysp</i> ASL <i>opr9_xysp</i> ASL <i>opr16_xysp</i> ASL [D, <i>xysp</i> ] ASL [ <i>opr16_xysp</i> ] ASLA ASLB	 Arithmetic Shift Left  Arithmetic Shift Left Accumulator A Arithmetic Shift Left Accumulator B	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	78 hh 11 68 xb 68 xb ff 68 xb ee ff 68 xb 68 xb ee ff 48 58	rPwO rPw rPwO frPwP fIfPrPw fIPrPw O O	rOPw rPw rPOw frPPw fIfPrPw fIPrPw O O	----	Δ Δ Δ Δ
ASLD	 Arithmetic Shift Left Double	INH	59	O	O	----	Δ Δ Δ Δ
ASR <i>opr16a</i> ASR <i>opr0_xysp</i> ASR <i>opr9_xysp</i> ASR <i>opr16_xysp</i> ASR [D, <i>xysp</i> ] ASR [ <i>opr16_xysp</i> ] ASRA ASRB	 Arithmetic Shift Right  Arithmetic Shift Right Accumulator A Arithmetic Shift Right Accumulator B	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	77 hh 11 67 xb 67 xb ff 67 xb ee ff 67 xb 67 xb ee ff 47 57	rPwO rPw rPwO frPwP fIfPrPw fIPrPw O O	rOPw rPw rPOw frPPw fIfPrPw fIPrPw O O	----	Δ Δ Δ Δ
BCC <i>rel8</i>	Branch if Carry Clear (if C = 0)	REL	24 rr	PPP/P <sup>1</sup>	PPP/P <sup>1</sup>	----	----
BCLR <i>opr8a, msk8</i> BCLR <i>opr16a, msk8</i> BCLR <i>opr0_xysp, msk8</i> BCLR <i>opr9_xysp, msk8</i> BCLR <i>opr16_xysp, msk8</i>	(M) • (mm) ⇒ M Clear Bit(s) in Memory	DIR EXT IDX IDX1 IDX2	4D dd mm 1D hh 11 mm 0D xb mm 0D xb ff mm 0D xb ee ff mm	rPwO rPwP rPwO rPwP frPwPO	rPOw rPPw rPOw rPwP frPwOP	----	Δ Δ 0 -
BCS <i>rel8</i>	Branch if Carry Set (if C = 1)	REL	25 rr	PPP/P <sup>1</sup>	PPP/P <sup>1</sup>	----	----
BEQ <i>rel8</i>	Branch if Equal (if Z = 1)	REL	27 rr	PPP/P <sup>1</sup>	PPP/P <sup>1</sup>	----	----
BGE <i>rel8</i>	Branch if Greater Than or Equal (if N ⊕ V = 0) (signed)	REL	2C rr	PPP/P <sup>1</sup>	PPP/P <sup>1</sup>	----	----
BGND	Place CPU in Background Mode see <i>CPU12 Reference Manual</i>	INH	00	VfPPP	VfPPP	----	----
BGT <i>rel8</i>	Branch if Greater Than (if Z + (N ⊕ V) = 0) (signed)	REL	2E rr	PPP/P <sup>1</sup>	PPP/P <sup>1</sup>	----	----
BHI <i>rel8</i>	Branch if Higher (if C + Z = 0) (unsigned)	REL	22 rr	PPP/P <sup>1</sup>	PPP/P <sup>1</sup>	----	----
BHS <i>rel8</i>	Branch if Higher or Same (if C = 0) (unsigned) same function as BCC	REL	24 rr	PPP/P <sup>1</sup>	PPP/P <sup>1</sup>	----	----
BITA <i>#opr8i</i> BITA <i>opr8a</i> BITA <i>opr16a</i> BITA <i>opr0_xysp</i> BITA <i>opr9_xysp</i> BITA <i>opr16_xysp</i> BITA [D, <i>xysp</i> ] BITA [ <i>opr16_xysp</i> ]	(A) • (M) Logical AND A with Memory Does not change Accumulator or Memory	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	85 ii 95 dd B5 hh 11 A5 xb A5 xb ff A5 xb ee ff A5 xb A5 xb ee ff	P rPf rPO rPf rPO frPP fIfPrPf fIPrPf	P rfP rOP rfP rPO frPP fIfPrP fIPrP	----	Δ Δ 0 -
BITB <i>#opr8i</i> BITB <i>opr8a</i> BITB <i>opr16a</i> BITB <i>opr0_xysp</i> BITB <i>opr9_xysp</i> BITB <i>opr16_xysp</i> BITB [D, <i>xysp</i> ] BITB [ <i>opr16_xysp</i> ]	(B) • (M) Logical AND B with Memory Does not change Accumulator or Memory	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C5 ii D5 dd F5 hh 11 E5 xb E5 xb ff E5 xb ee ff E5 xb E5 xb ee ff	P rPf rPO rPf rPO frPP fIfPrPf fIPrPf	P rfP rOP rfP rPO frPP fIfPrP fIPrP	----	Δ Δ 0 -
BLE <i>rel8</i>	Branch if Less Than or Equal (if Z + (N ⊕ V) = 1) (signed)	REL	2F rr	PPP/P <sup>1</sup>	PPP/P <sup>1</sup>	----	----
BLO <i>rel8</i>	Branch if Lower (if C = 1) (unsigned) same function as BCS	REL	25 rr	PPP/P <sup>1</sup>	PPP/P <sup>1</sup>	----	----

Note 1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

Table A-1. Instruction Set Summary (Sheet 3 of 14)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail		S X H I	N Z V C
				HCS12	M68HC12		
BLS <i>rel8</i>	Branch if Lower or Same (if C + Z = 1) (unsigned)	REL	23 rr	PPP/P <sup>1</sup>	PPP/P <sup>1</sup>	----	----
BLT <i>rel8</i>	Branch if Less Than (if N ⊕ V = 1) (signed)	REL	2D rr	PPP/P <sup>1</sup>	PPP/P <sup>1</sup>	----	----
BMI <i>rel8</i>	Branch if Minus (if N = 1)	REL	2B rr	PPP/P <sup>1</sup>	PPP/P <sup>1</sup>	----	----
BNE <i>rel8</i>	Branch if Not Equal (if Z = 0)	REL	26 rr	PPP/P <sup>1</sup>	PPP/P <sup>1</sup>	----	----
BPL <i>rel8</i>	Branch if Plus (if N = 0)	REL	2A rr	PPP/P <sup>1</sup>	PPP/P <sup>1</sup>	----	----
BRA <i>rel8</i>	Branch Always (if 1 = 1)	REL	20 rr	PPP	PPP	----	----
BRCLR <i>opr8a, msk8, rel8</i> BRCLR <i>opr16a, msk8, rel8</i> BRCLR <i>opr0_xysp, msk8, rel8</i> BRCLR <i>opr9_xysp, msk8, rel8</i> BRCLR <i>opr16_xysp, msk8, rel8</i>	Branch if (M) • (mm) = 0 (if All Selected Bit(s) Clear)	DIR EXT IDX IDX1 IDX2	4F dd mm rr 1F hh ll mm rr 0F xb mm rr 0F xb ff mm rr 0F xb ee ff mm rr	rPPP rfPPP rPPP rfPPP PrfPPP	rPPP rfPPP rPPP rffPPP frPfPPP	----	----
BRN <i>rel8</i>	Branch Never (if 1 = 0)	REL	21 rr	P	P	----	----
BRSET <i>opr8, msk8, rel8</i> BRSET <i>opr16a, msk8, rel8</i> BRSET <i>opr0_xysp, msk8, rel8</i> BRSET <i>opr9_xysp, msk8, rel8</i> BRSET <i>opr16_xysp, msk8, rel8</i>	Branch if (M) • (mm) = 0 (if All Selected Bit(s) Set)	DIR EXT IDX IDX1 IDX2	4E dd mm rr 1E hh ll mm rr 0E xb mm rr 0E xb ff mm rr 0E xb ee ff mm rr	rPPP rfPPP rPPP rfPPP PrfPPP	rPPP rfPPP rPPP rffPPP frPfPPP	----	----
BSET <i>opr8, msk8</i> BSET <i>opr16a, msk8</i> BSET <i>opr0_xysp, msk8</i> BSET <i>opr9_xysp, msk8</i> BSET <i>opr16_xysp, msk8</i>	(M) + (mm) ⇒ M Set Bit(s) in Memory	DIR EXT IDX IDX1 IDX2	4C dd mm 1C hh ll mm 0C xb mm 0C xb ff mm 0C xb ee ff mm	rPwO rPwP rPwO rPwP frPwPO	rPOw rPPw rPOw rPPw frPPwP	----	Δ Δ 0 -
BSR <i>rel8</i>	(SP) - 2 ⇒ SP; RTN <sub>H</sub> :RTN <sub>L</sub> ⇒ M <sub>(SP)</sub> :M <sub>(SP+1)</sub> Subroutine address ⇒ PC Branch to Subroutine	REL	07 rr	SPPP	PPPS	----	----
BVC <i>rel8</i>	Branch if Overflow Bit Clear (if V = 0)	REL	28 rr	PPP/P <sup>1</sup>	PPP/P <sup>1</sup>	----	----
BVS <i>rel8</i>	Branch if Overflow Bit Set (if V = 1)	REL	29 rr	PPP/P <sup>1</sup>	PPP/P <sup>1</sup>	----	----
CALL <i>opr16a, page</i> CALL <i>opr0_xysp, page</i> CALL <i>opr9_xysp, page</i> CALL <i>opr16_xysp, page</i> CALL [D, <i>xysp</i> ] CALL [ <i>opr16, xysp</i> ]	(SP) - 2 ⇒ SP; RTN <sub>H</sub> :RTN <sub>L</sub> ⇒ M <sub>(SP)</sub> :M <sub>(SP+1)</sub> (SP) - 1 ⇒ SP; (PPG) ⇒ M <sub>(SP)</sub> ; pg ⇒ PPAGE register; Program address ⇒ PC  Call subroutine in extended memory (Program may be located on another expansion memory page.)  Indirect modes get program address and new pg value based on pointer.	EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	4A hh ll pg 4B xb pg 4B xb ff pg 4B xb ee ff pg 4B xb 4B xb ee ff	gnSsPPP gnSsPPP gnSsPPP fgnSsPPP fIignSsPPP fIignSsPPP	gnfSsPPP gnfSsPPP gnfSsPPP fgnfSsPPP fIignSsPPP fIignSsPPP	----	----
CBA	(A) - (B) Compare 8-Bit Accumulators	INH	18 17	OO	OO	----	Δ Δ Δ Δ
CLC	0 ⇒ C Translates to ANDCC #\$FE	IMM	10 FE	P	P	----	---0
CLI	0 ⇒ I Translates to ANDCC #\$EF (enables I-bit interrupts)	IMM	10 EF	P	P	----0	----
CLR <i>opr16a</i> CLR <i>opr0_xysp</i> CLR <i>opr9_xysp</i> CLR <i>opr16_xysp</i> CLR [D, <i>xysp</i> ] CLR [ <i>opr16, xysp</i> ] CLRA CLRB	0 ⇒ M Clear Memory Location       0 ⇒ A Clear Accumulator A 0 ⇒ B Clear Accumulator B	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	79 hh ll 69 xb 69 xb ff 69 xb ee ff 69 xb 69 xb ee ff 87 C7	PwO Pw PwO PwP PIfW PIPW O O	wOP Pw PwO PwP PIfPw PIPPw O O	----	0100
CLV	0 ⇒ V Translates to ANDCC #\$FD	IMM	10 FD	P	P	----	--0-

Note 1. PPP/P indicates this instruction takes three cycles to refill the instruction queue if the branch is taken and one program fetch cycle if the branch is not taken.

**Table A-1. Instruction Set Summary (Sheet 4 of 14)**

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail		S X H I	N Z V C
				HCS12	M68HC12		
CMPA #opr8i CMPA opr8a CMPA opr16a CMPA oprx0_xysp CMPA oprx9_xysp CMPA oprx16_xysp CMPA [D,xysp] CMPA [oprx16,xysp]	(A) – (M) Compare Accumulator A with Memory	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	81 ii 91 dd B1 hh 11 A1 xb A1 xb ff A1 xb ee ff A1 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rfP rOP rfP rPO frPP fIfrfP fIPrfP	----	Δ Δ Δ Δ
CMPB #opr8i CMPB opr8a CMPB opr16a CMPB oprx0_xysp CMPB oprx9_xysp CMPB oprx16_xysp CMPB [D,xysp] CMPB [oprx16,xysp]	(B) – (M) Compare Accumulator B with Memory	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C1 ii D1 dd F1 hh 11 E1 xb E1 xb ff E1 xb ee ff E1 xb E1 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rfP rOP rfP rPO frPP fIfrfP fIPrfP	----	Δ Δ Δ Δ
COM opr16a COM oprx0_xysp COM oprx9_xysp COM oprx16_xysp COM [D,xysp] COM [oprx16,xysp] COMA COMB	(M) ⇒ M equivalent to \$FF – (M) ⇒ M 1's Complement Memory Location  (A) ⇒ A Complement Accumulator A (B) ⇒ B Complement Accumulator B	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	71 hh 11 61 xb 61 xb ff 61 xb ee ff 61 xb 61 xb ee ff 41 51	rPwO rPw rPwO frPwP fIfrPw fIPrPw O O	rOPw rPw rPOw frPPw fIfrfPw fIPrfPw O O	----	Δ Δ 0 1
CPD #opr16i CPD opr8a CPD opr16a CPD oprx0_xysp CPD oprx9_xysp CPD oprx16_xysp CPD [D,xysp] CPD [oprx16,xysp]	(A:B) – (M:M+1) Compare D to Memory (16-Bit)	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	8C jj kk 9C dd BC hh 11 AC xb AC xb ff AC xb ee ff AC xb AC xb ee ff	PO RPf RPO RPf RPO frPP fIfRPf fIPRPf	OP rfP ROP rfP RPO frPP fIfrfP fIPrfP	----	Δ Δ Δ Δ
CPS #opr16i CPS opr8a CPS opr16a CPS oprx0_xysp CPS oprx9_xysp CPS oprx16_xysp CPS [D,xysp] CPS [oprx16,xysp]	(SP) – (M:M+1) Compare SP to Memory (16-Bit)	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	8F jj kk 9F dd BF hh 11 AF xb AF xb ff AF xb ee ff AF xb AF xb ee ff	PO RPf RPO RPf RPO frPP fIfRPf fIPRPf	OP rfP ROP rfP RPO frPP fIfrfP fIPrfP	----	Δ Δ Δ Δ
CPX #opr16i CPX opr8a CPX opr16a CPX oprx0_xysp CPX oprx9_xysp CPX oprx16_xysp CPX [D,xysp] CPX [oprx16,xysp]	(X) – (M:M+1) Compare X to Memory (16-Bit)	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	8E jj kk 9E dd BE hh 11 AE xb AE xb ff AE xb ee ff AE xb AE xb ee ff	PO RPf RPO RPf RPO frPP fIfRPf fIPRPf	OP rfP ROP rfP RPO frPP fIfrfP fIPrfP	----	Δ Δ Δ Δ
CPY #opr16i CPY opr8a CPY opr16a CPY oprx0_xysp CPY oprx9_xysp CPY oprx16_xysp CPY [D,xysp] CPY [oprx16,xysp]	(Y) – (M:M+1) Compare Y to Memory (16-Bit)	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	8D jj kk 9D dd BD hh 11 AD xb AD xb ff AD xb ee ff AD xb AD xb ee ff	PO RPf RPO RPf RPO frPP fIfRPf fIPRPf	OP rfP ROP rfP RPO frPP fIfrfP fIPrfP	----	Δ Δ Δ Δ
DAA	Adjust Sum to BCD Decimal Adjust Accumulator A	INH	18 07	OfO	OfO	----	Δ Δ ? Δ
DBEQ abdxys, rel9	(cntr) – 1 ⇒ cntr if (cntr) = 0, then Branch else Continue to next instruction  Decrement Counter and Branch if = 0 (cntr = A, B, D, X, Y, or SP)	REL (9-bit)	04 1b rr	PPP (branch) PPO (no branch)	PPP	----	----

Table A-1. Instruction Set Summary (Sheet 5 of 14)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail		S X H I	N Z V C
				HCS12	M68HC12		
DBNE <i>abdxys, rel9</i>	(cntr) – 1 $\Rightarrow$ cntr If (cntr) not = 0, then Branch; else Continue to next instruction  Decrement Counter and Branch if $\neq$ 0 (cntr = A, B, D, X, Y, or SP)	REL (9-bit)	04 1b rr	PPP (branch) PPO (no branch)	PPP	----	----
DEC <i>opr16a</i> DEC <i>opr0_xysp</i> DEC <i>opr9_xysp</i> DEC <i>opr16_xysp</i> DEC [D, <i>xysp</i> ] DEC [ <i>opr16_xysp</i> ] DECA DECB	(M) – \$01 $\Rightarrow$ M Decrement Memory Location  (A) – \$01 $\Rightarrow$ A      Decrement A (B) – \$01 $\Rightarrow$ B      Decrement B	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	73 hh 1l 63 xb 63 xb ff 63 xb ee ff 63 xb 63 xb ee ff 43 53	rPwO rPw rPwO frPwP fIfrPw fIPrPw O O	rOPw rPw rPOw frPPw fIfrPPw fIPrPPw O O	----	$\Delta \Delta \Delta -$
DES	(SP) – \$0001 $\Rightarrow$ SP <i>Translates to LEAS –1, SP</i>	IDX	1B 9F	Pf	pp <sup>1</sup>	----	----
DEX	(X) – \$0001 $\Rightarrow$ X Decrement Index Register X	INH	09	O	O	----	$- \Delta - -$
DEY	(Y) – \$0001 $\Rightarrow$ Y Decrement Index Register Y	INH	03	O	O	----	$- \Delta - -$
EDIV	(Y:D) $\div$ (X) $\Rightarrow$ Y Remainder $\Rightarrow$ D 32 by 16 Bit $\Rightarrow$ 16 Bit Divide (unsigned)	INH	11	fffffffffO	fffffffffO	----	$\Delta \Delta \Delta \Delta$
EDIVS	(Y:D) $\div$ (X) $\Rightarrow$ Y Remainder $\Rightarrow$ D 32 by 16 Bit $\Rightarrow$ 16 Bit Divide (signed)	INH	18 14	OffffffffffO	OffffffffffO	----	$\Delta \Delta \Delta \Delta$
EMACS <i>opr16a</i> <sup>2</sup>	(M <sub>(X)</sub> :M <sub>(X+1)</sub> ) $\times$ (M <sub>(Y)</sub> :M <sub>(Y+1)</sub> ) + (M – M+3) $\Rightarrow$ M – M+3  16 by 16 Bit $\Rightarrow$ 32 Bit Multiply and Accumulate (signed)	Special	18 12 hh 1l	ORROffRRfWWP	ORROffRRfWWP	----	$\Delta \Delta \Delta \Delta$
EMAXD <i>opr0_xysp</i> EMAXD <i>opr9_xysp</i> EMAXD <i>opr16_xysp</i> EMAXD [D, <i>xysp</i> ] EMAXD [ <i>opr16_xysp</i> ]	MAX((D), (M:M+1)) $\Rightarrow$ D MAX of 2 Unsigned 16-Bit Values  N, Z, V and C status bits reflect result of internal compare ((D) – (M:M+1))	IDX IDX1 IDX2 [D,IDX] [IDX2]	18 1A xb 18 1A xb ff 18 1A xb ee ff 18 1A xb 18 1A xb ee ff	ORPf ORPO OfRRPf OfIfrPf OfIPRPf	ORfP ORPO OfRRP OfIfrP OfIPRP	----	$\Delta \Delta \Delta \Delta$
EMAXM <i>opr0_xysp</i> EMAXM <i>opr9_xysp</i> EMAXM <i>opr16_xysp</i> EMAXM [D, <i>xysp</i> ] EMAXM [ <i>opr16_xysp</i> ]	MAX((D), (M:M+1)) $\Rightarrow$ M:M+1 MAX of 2 Unsigned 16-Bit Values  N, Z, V and C status bits reflect result of internal compare ((D) – (M:M+1))	IDX IDX1 IDX2 [D,IDX] [IDX2]	18 1E xb 18 1E xb ff 18 1E xb ee ff 18 1E xb 18 1E xb ee ff	ORPW ORPWO OfRPWP OfIfrPW OfIPRPW	ORPW ORPWO OfRPWP OfIfrPW OfIPRPW	----	$\Delta \Delta \Delta \Delta$
EMIND <i>opr0_xysp</i> EMIND <i>opr9_xysp</i> EMIND <i>opr16_xysp</i> EMIND [D, <i>xysp</i> ] EMIND [ <i>opr16_xysp</i> ]	MIN((D), (M:M+1)) $\Rightarrow$ D MIN of 2 Unsigned 16-Bit Values  N, Z, V and C status bits reflect result of internal compare ((D) – (M:M+1))	IDX IDX1 IDX2 [D,IDX] [IDX2]	18 1B xb 18 1B xb ff 18 1B xb ee ff 18 1B xb 18 1B xb ee ff	ORPf ORPO OfRRPf OfIfrPf OfIPRPf	ORfP ORPO OfRRP OfIfrP OfIPRP	----	$\Delta \Delta \Delta \Delta$
EMINM <i>opr0_xysp</i> EMINM <i>opr9_xysp</i> EMINM <i>opr16_xysp</i> EMINM [D, <i>xysp</i> ] EMINM [ <i>opr16_xysp</i> ]	MIN((D), (M:M+1)) $\Rightarrow$ M:M+1 MIN of 2 Unsigned 16-Bit Values  N, Z, V and C status bits reflect result of internal compare ((D) – (M:M+1))	IDX IDX1 IDX2 [D,IDX] [IDX2]	18 1F xb 18 1F xb ff 18 1F xb ee ff 18 1F xb 18 1F xb ee ff	ORPW ORPWO OfRPWP OfIfrPW OfIPRPW	ORPW ORPWO OfRPWP OfIfrPW OfIPRPW	----	$\Delta \Delta \Delta \Delta$
EMUL	(D) $\times$ (Y) $\Rightarrow$ Y:D 16 by 16 Bit Multiply (unsigned)	INH	13	ffO	ffO	----	$\Delta \Delta - \Delta$
EMULS	(D) $\times$ (Y) $\Rightarrow$ Y:D 16 by 16 Bit Multiply (signed)	INH	18 13	OfO (if followed by page 2 instruction) OffO	OfO OfO	----	$\Delta \Delta - \Delta$
EORA # <i>opr8i</i> EORA <i>opr8a</i> EORA <i>opr16a</i> EORA <i>opr0_xysp</i> EORA <i>opr9_xysp</i> EORA <i>opr16_xysp</i> EORA [D, <i>xysp</i> ] EORA [ <i>opr16_xysp</i> ]	(A) $\oplus$ (M) $\Rightarrow$ A Exclusive-OR A with Memory	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	88 ii 98 dd B8 hh 1l A8 xb A8 xb ff A8 xb ee ff A8 xb A8 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rFP rOP rFP rPO frPP fIfrPP fIPrPP	----	$\Delta \Delta 0 -$

Notes:

- Due to internal CPU requirements, the program word fetch is performed twice to the same address during this instruction.
- opr16a* is an extended address specification. Both X and Y point to source operands.

## Table A-1. Instruction Set Summary (Sheet 6 of 14)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail		S X H I	N Z V C
				HCS12	M68HC12		
EORB <i>#opr8i</i> EORB <i>opr8a</i> EORB <i>opr16a</i> EORB <i>opr0_xysp</i> EORB <i>opr9_xysp</i> EORB <i>opr16_xysp</i> EORB [ <i>D,xysp</i> ] EORB [ <i>opr16_xysp</i> ]	$(B) \oplus (M) \Rightarrow B$ Exclusive-OR B with Memory	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C8 ii D8 dd F8 hh ll E8 xb E8 xb ff E8 xb ee ff E8 xb E8 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rPf rOP rPf rPO frPP fIfrPf fIPrPf	----	$\Delta \Delta 0 -$
ETBL <i>opr0_xysp</i>	$(M:M+1) + [(B) \times ((M+2:M+3) - (M:M+1))] \Rightarrow D$ 16-Bit Table Lookup and Interpolate  Initialize B, and index before ETBL. <ea> points at first table entry (M:M+1) and B is fractional part of lookup value  (no indirect addr. modes or extensions allowed)	IDX	18 3F xb	ORRfffffP ORRfffffP	ORRfffffP ORRfffffP	----	$\Delta \Delta - \Delta$ ?  C Bit is undefined in HC12
EXG <i>abcdxys,abcdxys</i>	$(r1) \Leftrightarrow (r2)$ (if r1 and r2 same size) or \$00:(r1) $\Rightarrow$ r2 (if r1=8-bit; r2=16-bit) or $(r1_{low}) \Leftrightarrow (r2)$ (if r1=16-bit; r2=8-bit)  r1 and r2 may be A, B, CCR, D, X, Y, or SP	INH	B7 eb	P	P	----	----
FDIV	$(D) \div (X) \Rightarrow X$ ; Remainder $\Rightarrow$ D 16 by 16 Bit Fractional Divide	INH	18 11	OfrrrrrrrrfO OfrrrrrrrrfO	OfrrrrrrrrfO OfrrrrrrrrfO	----	$-\Delta \Delta \Delta$
IBEQ <i>abdxys,rel9</i>	$(cntr) + 1 \Rightarrow cntr$ If $(cntr) = 0$ , then Branch else Continue to next instruction  Increment Counter and Branch if = 0 (cntr = A, B, D, X, Y, or SP)	REL (9-bit)	04 1b rr	PPP (branch) PPO (no branch)	PPP PPP	----	----
IBNE <i>abdxys,rel9</i>	$(cntr) + 1 \Rightarrow cntr$ if $(cntr) \neq 0$ , then Branch; else Continue to next instruction  Increment Counter and Branch if $\neq 0$ (cntr = A, B, D, X, Y, or SP)	REL (9-bit)	04 1b rr	PPP (branch) PPO (no branch)	PPP PPP	----	----
IDIV	$(D) \div (X) \Rightarrow X$ ; Remainder $\Rightarrow$ D 16 by 16 Bit Integer Divide (unsigned)	INH	18 10	OfrrrrrrrrfO OfrrrrrrrrfO	OfrrrrrrrrfO OfrrrrrrrrfO	----	$-\Delta 0 \Delta$
IDIVS	$(D) \div (X) \Rightarrow X$ ; Remainder $\Rightarrow$ D 16 by 16 Bit Integer Divide (signed)	INH	18 15	OfrrrrrrrrfO OfrrrrrrrrfO	OfrrrrrrrrfO OfrrrrrrrrfO	----	$\Delta \Delta \Delta \Delta$
INC <i>opr16a</i> INC <i>opr0_xysp</i> INC <i>opr9_xysp</i> INC <i>opr16_xysp</i> INC [ <i>D,xysp</i> ] INC [ <i>opr16_xysp</i> ] INCA INCB	$(M) + \$01 \Rightarrow M$ Increment Memory Byte      $(A) + \$01 \Rightarrow A$ Increment Acc. A $(B) + \$01 \Rightarrow B$ Increment Acc. B	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	72 hh ll 62 xb 62 xb ff 62 xb ee ff 62 xb 62 xb ee ff 42 52	rPwO rPw rPwO frPwP fIfrPw fIPrPw O O	rOPw rPw rPOw frPPw fIfrPPw fIPrPPw O O	----	$\Delta \Delta \Delta -$
INS	$(SP) + \$0001 \Rightarrow SP$ Translates to LEAS 1,SP	IDX	1B 81	Pf	pp <sup>1</sup>	----	----
INX	$(X) + \$0001 \Rightarrow X$ Increment Index Register X	INH	08	O	O	----	$-\Delta --$
INY	$(Y) + \$0001 \Rightarrow Y$ Increment Index Register Y	INH	02	O	O	----	$-\Delta --$
JMP <i>opr16a</i> JMP <i>opr0_xysp</i> JMP <i>opr9_xysp</i> JMP <i>opr16_xysp</i> JMP [ <i>D,xysp</i> ] JMP [ <i>opr16_xysp</i> ]	Routine address $\Rightarrow$ PC  Jump	EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	06 hh ll 05 xb 05 xb ff 05 xb ee ff 05 xb 05 xb ee ff	PPP PPP PPP fPPP fIfPPP fIfPPP	PPP PPP PPP fPPP fIfPPP fIfPPP	----	----


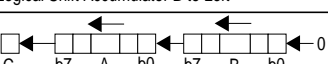
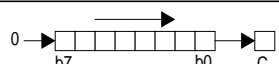
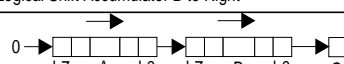
Note 1. Due to internal CPU requirements, the program word fetch is performed twice to the same address during this instruction.

Table A-1. Instruction Set Summary (Sheet 7 of 14)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail		S X H I	N Z V C
				HCS12	M68HC12		
JSR <i>opr8a</i> JSR <i>opr16a</i> JSR <i>opr0_xysp</i> JSR <i>opr9_xysp</i> JSR <i>opr16_xysp</i> JSR [D, <i>xysp</i> ] JSR [ <i>opr16_xysp</i> ]	(SP) – 2 ⇒ SP; RTN <sub>H</sub> :RTN <sub>L</sub> ⇒ M <sub>(SP)</sub> :M <sub>(SP+1)</sub> ; Subroutine address ⇒ PC  Jump to Subroutine	DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	17 dd 16 hh ll 15 xb 15 xb ff 15 xb ee ff 15 xb 15 xb ee ff	SPPP SPPP PPPS PPPS fPPPS fIfPPPS fIfPPPS	PPPS PPPS PPPS PPPS fPPPS fIfPPPS fIfPPPS	----	----
LBCC <i>rel16</i>	Long Branch if Carry Clear (if C = 0)	REL	18 24 qq rr	OPPP/OPO <sup>1</sup>	OPPP/OPO <sup>1</sup>	----	----
LBCS <i>rel16</i>	Long Branch if Carry Set (if C = 1)	REL	18 25 qq rr	OPPP/OPO <sup>1</sup>	OPPP/OPO <sup>1</sup>	----	----
LBEQ <i>rel16</i>	Long Branch if Equal (if Z = 1)	REL	18 27 qq rr	OPPP/OPO <sup>1</sup>	OPPP/OPO <sup>1</sup>	----	----
LBGE <i>rel16</i>	Long Branch Greater Than or Equal (if N ⊕ V = 0) (signed)	REL	18 2C qq rr	OPPP/OPO <sup>1</sup>	OPPP/OPO <sup>1</sup>	----	----
LBGT <i>rel16</i>	Long Branch if Greater Than (if Z + (N ⊕ V) = 0) (signed)	REL	18 2E qq rr	OPPP/OPO <sup>1</sup>	OPPP/OPO <sup>1</sup>	----	----
LBHI <i>rel16</i>	Long Branch if Higher (if C + Z = 0) (unsigned)	REL	18 22 qq rr	OPPP/OPO <sup>1</sup>	OPPP/OPO <sup>1</sup>	----	----
LBHS <i>rel16</i>	Long Branch if Higher or Same (if C = 0) (unsigned) same function as LBCC	REL	18 24 qq rr	OPPP/OPO <sup>1</sup>	OPPP/OPO <sup>1</sup>	----	----
LBLE <i>rel16</i>	Long Branch if Less Than or Equal (if Z + (N ⊕ V) = 1) (signed)	REL	18 2F qq rr	OPPP/OPO <sup>1</sup>	OPPP/OPO <sup>1</sup>	----	----
LBLO <i>rel16</i>	Long Branch if Lower (if C = 1) (unsigned) same function as LBCS	REL	18 25 qq rr	OPPP/OPO <sup>1</sup>	OPPP/OPO <sup>1</sup>	----	----
LBLS <i>rel16</i>	Long Branch if Lower or Same (if C + Z = 1) (unsigned)	REL	18 23 qq rr	OPPP/OPO <sup>1</sup>	OPPP/OPO <sup>1</sup>	----	----
LBLT <i>rel16</i>	Long Branch if Less Than (if N ⊕ V = 1) (signed)	REL	18 2D qq rr	OPPP/OPO <sup>1</sup>	OPPP/OPO <sup>1</sup>	----	----
LBMI <i>rel16</i>	Long Branch if Minus (if N = 1)	REL	18 2B qq rr	OPPP/OPO <sup>1</sup>	OPPP/OPO <sup>1</sup>	----	----
LBNE <i>rel16</i>	Long Branch if Not Equal (if Z = 0)	REL	18 26 qq rr	OPPP/OPO <sup>1</sup>	OPPP/OPO <sup>1</sup>	----	----
LBPL <i>rel16</i>	Long Branch if Plus (if N = 0)	REL	18 2A qq rr	OPPP/OPO <sup>1</sup>	OPPP/OPO <sup>1</sup>	----	----
LBRA <i>rel16</i>	Long Branch Always (if 1=1)	REL	18 20 qq rr	OPPP	OPPP	----	----
LBRN <i>rel16</i>	Long Branch Never (if 1 = 0)	REL	18 21 qq rr	OPO	OPO	----	----
LBVC <i>rel16</i>	Long Branch if Overflow Bit Clear (if V=0)	REL	18 28 qq rr	OPPP/OPO <sup>1</sup>	OPPP/OPO <sup>1</sup>	----	----
LBVS <i>rel16</i>	Long Branch if Overflow Bit Set (if V = 1)	REL	18 29 qq rr	OPPP/OPO <sup>1</sup>	OPPP/OPO <sup>1</sup>	----	----
LDAA # <i>opr8i</i> LDAA <i>opr8a</i> LDAA <i>opr16a</i> LDAA <i>opr0_xysp</i> LDAA <i>opr9_xysp</i> LDAA <i>opr16_xysp</i> LDAA [D, <i>xysp</i> ] LDAA [ <i>opr16_xysp</i> ]	(M) ⇒ A Load Accumulator A	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	86 ii 96 dd B6 hh ll A6 xb A6 xb ff A6 xb ee ff A6 xb A6 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rPf rOP rPf rPO frPP fIfrPf fIPrPf	----	Δ Δ 0 –
LDAB # <i>opr8i</i> LDAB <i>opr8a</i> LDAB <i>opr16a</i> LDAB <i>opr0_xysp</i> LDAB <i>opr9_xysp</i> LDAB <i>opr16_xysp</i> LDAB [D, <i>xysp</i> ] LDAB [ <i>opr16_xysp</i> ]	(M) ⇒ B Load Accumulator B	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C6 ii D6 dd F6 hh ll E6 xb E6 xb ff E6 xb ee ff E6 xb E6 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rPf rOP rPf rPO frPP fIfrPf fIPrPf	----	Δ Δ 0 –
LDD # <i>opr16i</i> LDD <i>opr8a</i> LDD <i>opr16a</i> LDD <i>opr0_xysp</i> LDD <i>opr9_xysp</i> LDD <i>opr16_xysp</i> LDD [D, <i>xysp</i> ] LDD [ <i>opr16_xysp</i> ]	(M:M+1) ⇒ A:B Load Double Accumulator D (A:B)	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	CC jj kk DC dd FC hh ll EC xb EC xb ff EC xb ee ff EC xb EC xb ee ff	PO RPf RPO RPf RPO frPP fIfrPf fIPrPf	OP RfP ROP RfP RPO frPP fIfrPf fIPrPf	----	Δ Δ 0 –

Note 1. OPPP/OPO indicates this instruction takes four cycles to refill the instruction queue if the branch is taken and three cycles if the branch is not taken.

**Table A-1. Instruction Set Summary (Sheet 8 of 14)**

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail		S X H I	N Z V C
				HCS12	M68HC12		
LDS #opr16i LDS opr8a LDS opr16a LDS oprx0_xysp LDS oprx9_xysp LDS oprx16_xysp LDS [D,xysp] LDS [oprx16,xysp]	(M:M+1) ⇒ SP Load Stack Pointer	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	CF jj kk DF dd FF hh ll EF xb EF xb ff EF xb ee ff EF xb EF xb ee ff	PO Rpf RPO Rpf RPO fRPP fIfRPF fIPRPF	OP RfP ROP RfP RPO fRPP fIfRfP fIPRfP	----	Δ Δ 0 -
LDX #opr16i LDX opr8a LDX opr16a LDX oprx0_xysp LDX oprx9_xysp LDX oprx16_xysp LDX [D,xysp] LDX [oprx16,xysp]	(M:M+1) ⇒ X Load Index Register X	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	CE jj kk DE dd FE hh ll EE xb EE xb ff EE xb ee ff EE xb EE xb ee ff	PO Rpf RPO Rpf RPO fRPP fIfRPF fIPRPF	OP RfP ROP RfP RPO fRPP fIfRfP fIPRfP	----	Δ Δ 0 -
LDY #opr16i LDY opr8a LDY opr16a LDY oprx0_xysp LDY oprx9_xysp LDY oprx16_xysp LDY [D,xysp] LDY [oprx16,xysp]	(M:M+1) ⇒ Y Load Index Register Y	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	CD jj kk DD dd FD hh ll ED xb ED xb ff ED xb ee ff ED xb ED xb ee ff	PO Rpf RPO Rpf RPO fRPP fIfRPF fIPRPF	OP RfP ROP RfP RPO fRPP fIfRfP fIPRfP	----	Δ Δ 0 -
LEAS oprx0_xysp LEAS oprx9_xysp LEAS oprx16_xysp	Effective Address ⇒ SP Load Effective Address into SP	IDX IDX1 IDX2	1B xb 1B xb ff 1B xb ee ff	Pf PO PP	pp <sup>1</sup> PO PP	----	----
LEAX oprx0_xysp LEAX oprx9_xysp LEAX oprx16_xysp	Effective Address ⇒ X Load Effective Address into X	IDX IDX1 IDX2	1A xb 1A xb ff 1A xb ee ff	Pf PO PP	pp <sup>1</sup> PO PP	----	----
LEAY oprx0_xysp LEAY oprx9_xysp LEAY oprx16_xysp	Effective Address ⇒ Y Load Effective Address into Y	IDX IDX1 IDX2	19 xb 19 xb ff 19 xb ee ff	Pf PO PP	pp <sup>1</sup> PO PP	----	----
LSL opr16a LSL oprx0_xysp LSL oprx9_xysp LSL oprx16_xysp LSL [D,xysp] LSL [oprx16,xysp] LSLA LSLB	 Logical Shift Left same function as ASL	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	78 hh ll 68 xb 68 xb ff 68 xb ee ff 68 xb 68 xb ee ff 48 58	rPwO rPw rPwO frPPw fIfrPw fIPrPw O O	rOPw rPw rPOw frPPw fIfrPw fIPrPw O O	----	Δ Δ Δ Δ
LSLD	 Logical Shift Left D Accumulator same function as ASLD	INH	59	O	O	----	Δ Δ Δ Δ
LSR opr16a LSR oprx0_xysp LSR oprx9_xysp LSR oprx16_xysp LSR [D,xysp] LSR [oprx16,xysp] LSRA LSRB	 Logical Shift Right	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	74 hh ll 64 xb 64 xb ff 64 xb ee ff 64 xb 64 xb ee ff 44 54	rPwO rPw rPwO frPPw fIfrPw fIPrPw O O	rOPw rPw rPOw frPPw fIfrPw fIPrPw O O	----	0 Δ Δ Δ
LSRD	 Logical Shift Right D Accumulator	INH	49	O	O	----	0 Δ Δ Δ
MAXA oprx0_xysp MAXA oprx9_xysp MAXA oprx16_xysp MAXA [D,xysp] MAXA [oprx16,xysp]	MAX((A), (M)) ⇒ A MAX of 2 Unsigned 8-Bit Values  N, Z, V and C status bits reflect result of internal compare ((A) - (M)).	IDX IDX1 IDX2 [D,IDX] [IDX2]	18 18 xb 18 18 xb ff 18 18 xb ee ff 18 18 xb 18 18 xb ee ff	OrPf OrPO OfrrPP OfIfrrPf OfIPrrPf	OrfP OrfPO OfrrPP OfIfrrfP OfIPrrfP	----	Δ Δ Δ Δ

Note 1. Due to internal CPU requirements, the program word fetch is performed twice to the same address during this instruction.



Table A-1. Instruction Set Summary (Sheet 9 of 14)

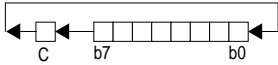
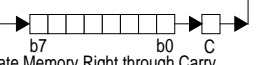
Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail		S X H I	N Z V C
				HCS12	M68HC12		
MAXM <i>opr</i> x0, <i>xy</i> sp MAXM <i>opr</i> x9, <i>xy</i> sp MAXM <i>opr</i> x16, <i>xy</i> sp MAXM [D, <i>xy</i> sp] MAXM [ <i>opr</i> x16, <i>xy</i> sp]	MAX((A), (M)) $\Rightarrow$ M MAX of 2 Unsigned 8-Bit Values  N, Z, V and C status bits reflect result of internal compare ((A) – (M)).	IDX IDX1 IDX2 [D, IDX] [IDX2]	18 1C xb 18 1C xb ff 18 1C xb ee ff 18 1C xb 18 1C xb ee ff	OrPw OrPwO OfIfrPwP OfIfrPw OfIPrPw	OrPw OrPwO OfIfrPwP OfIfrPw OfIPrPw	----	Δ Δ Δ Δ
MEM	$\mu$ (grade) $\Rightarrow M_{\lceil Y \rceil}$ ; (X) + 4 $\Rightarrow$ X; (Y) + 1 $\Rightarrow$ Y; A unchanged  if (A) < P1 or (A) > P2 then $\mu = 0$ , else $\mu = \text{MIN}(((A) - P1) \times S1, (P2 - (A)) \times S2, \$FF)$ where: A = current crisp input value; X points at 4-byte data structure that describes a trapezoidal membership function (P1, P2, S1, S2); Y points at fuzzy input (RAM location). See <i>CPU12 Reference Manual</i> for special cases.	Special	01	RRfOw	RRfOw	--?–	????
MINA <i>opr</i> x0, <i>xy</i> sp MINA <i>opr</i> x9, <i>xy</i> sp MINA <i>opr</i> x16, <i>xy</i> sp MINA [D, <i>xy</i> sp] MINA [ <i>opr</i> x16, <i>xy</i> sp]	MIN((A), (M)) $\Rightarrow$ A MIN of 2 Unsigned 8-Bit Values  N, Z, V and C status bits reflect result of internal compare ((A) – (M)).	IDX IDX1 IDX2 [D, IDX] [IDX2]	18 19 xb 18 19 xb ff 18 19 xb ee ff 18 19 xb 18 19 xb ee ff	OrPf OrPO OfIfrPP OfIfrPf OfIPrPf	OrFP OrPO OfIfrPP OfIfrPf OfIPrPf	----	Δ Δ Δ Δ
MINM <i>opr</i> x0, <i>xy</i> sp MINM <i>opr</i> x9, <i>xy</i> sp MINM <i>opr</i> x16, <i>xy</i> sp MINM [D, <i>xy</i> sp] MINM [ <i>opr</i> x16, <i>xy</i> sp]	MIN((A), (M)) $\Rightarrow$ M MIN of 2 Unsigned 8-Bit Values  N, Z, V and C status bits reflect result of internal compare ((A) – (M)).	IDX IDX1 IDX2 [D, IDX] [IDX2]	18 1D xb 18 1D xb ff 18 1D xb ee ff 18 1D xb 18 1D xb ee ff	OrPw OrPwO OfIfrPwP OfIfrPw OfIPrPw	OrPw OrPwO OfIfrPwP OfIfrPw OfIPrPw	----	Δ Δ Δ Δ
MOVB # <i>opr</i> 8, <i>opr</i> 16a <sup>1</sup> MOVB # <i>opr</i> 8i, <i>opr</i> x0, <i>xy</i> sp <sup>1</sup> MOVB <i>opr</i> 16a, <i>opr</i> 16a <sup>1</sup> MOVB <i>opr</i> 16a, <i>opr</i> x0, <i>xy</i> sp <sup>1</sup> MOVB <i>opr</i> x0, <i>xy</i> sp, <i>opr</i> 16a <sup>1</sup> MOVB <i>opr</i> x0, <i>xy</i> sp, <i>opr</i> x0, <i>xy</i> sp <sup>1</sup>	(M <sub>1</sub> ) $\Rightarrow$ M <sub>2</sub> Memory to Memory Byte-Move (8-Bit)	IMM-EXT IMM-IDX EXT-EXT EXT-IDX IDX-EXT IDX-IDX	18 0B ii hh ll 18 08 xb ii 18 0C hh ll hh ll 18 09 xb hh ll 18 0D xb hh ll 18 0A xb xb	OPwP OPwO OrPwPO OPrPw OrPwP OrPwO	OPwP OPwO OrPwPO OPrPw OrPwP OrPwO	----	----
MOVW # <i>opr</i> 16, <i>opr</i> 16a <sup>1</sup> MOVW # <i>opr</i> 16i, <i>opr</i> x0, <i>xy</i> sp <sup>1</sup> MOVW <i>opr</i> 16a, <i>opr</i> 16a <sup>1</sup> MOVW <i>opr</i> 16a, <i>opr</i> x0, <i>xy</i> sp <sup>1</sup> MOVW <i>opr</i> x0, <i>xy</i> sp, <i>opr</i> 16a <sup>1</sup> MOVW <i>opr</i> x0, <i>xy</i> sp, <i>opr</i> x0, <i>xy</i> sp <sup>1</sup>	(M:M+1) $\Rightarrow$ M:M+2 Memory to Memory Word-Move (16-Bit)	IMM-EXT IMM-IDX EXT-EXT EXT-IDX IDX-EXT IDX-IDX	18 03 jj kk hh ll 18 00 xb jj kk 18 04 hh ll hh ll 18 01 xb hh ll 18 05 xb hh ll 18 02 xb xb	OPWPO OPPW ORPWPO OPRPW ORPWP ORPWO	OPWPO OPPW ORPWPO OPRPW ORPWP ORPWO	----	----
MUL	(A) × (B) $\Rightarrow$ A:B 8 by 8 Unsigned Multiply	INH	12	O	ffO	----	---Δ
NEG <i>opr</i> 16a NEG <i>opr</i> x0, <i>xy</i> sp NEG <i>opr</i> x9, <i>xy</i> sp NEG <i>opr</i> x16, <i>xy</i> sp NEG [D, <i>xy</i> sp] NEG [ <i>opr</i> x16, <i>xy</i> sp] NEGA  NEGB	0 – (M) $\Rightarrow$ M equivalent to (M) + 1 $\Rightarrow$ M Two's Complement Negate  0 – (A) $\Rightarrow$ A equivalent to (A) + 1 $\Rightarrow$ A Negate Accumulator A 0 – (B) $\Rightarrow$ B equivalent to (B) + 1 $\Rightarrow$ B Negate Accumulator B	EXT IDX IDX1 IDX2 [D, IDX] [IDX2] INH  INH	70 hh ll 60 xb 60 xb ff 60 xb ee ff 60 xb 60 xb ee ff 40  50	rPwO rPw rPwO frPwP fIfrPw fIPrPw O  O	rOPw rPw rPOw frPPw fIfrPw fIPrPw O  O	----	Δ Δ Δ Δ
NOP	No Operation	INH	A7	O	O	----	----
ORAA # <i>opr</i> 8i ORAA <i>opr</i> 8a ORAA <i>opr</i> 16a ORAA <i>opr</i> x0, <i>xy</i> sp ORAA <i>opr</i> x9, <i>xy</i> sp ORAA <i>opr</i> x16, <i>xy</i> sp ORAA [D, <i>xy</i> sp] ORAA [ <i>opr</i> x16, <i>xy</i> sp]	(A) + (M) $\Rightarrow$ A Logical OR A with Memory	IMM DIR EXT IDX IDX1 IDX2 [D, IDX] [IDX2]	8A ii 9A dd BA hh ll AA xb AA xb ff AA xb ee ff AA xb AA xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rFP rOP rFP rPO frPP fIfrFP fIPrFP	----	Δ Δ 0 –

Note 1. The first operand in the source code statement specifies the source for the move.

**Table A-1. Instruction Set Summary (Sheet 10 of 14)**

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail		S X H I	N Z V C
				HCS12	M68HC12		
ORAB #opr8i ORAB opr8a ORAB opr16a ORAB oprx0_xysp ORAB oprx9_xysp ORAB oprx16_xysp ORAB [D,xysp] ORAB [opr16,xysp]	(B) + (M) ⇒ B Logical OR B with Memory	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	CA ii DA dd FA hh ll EA xb EA xb ff EA xb ee ff EA xb EA xb ee ff	P rPf rPO rPf rPO frPP fIfPrPf fIPrPf	P rfP rOP rfP rPO frPP fIfPrPf fIPrPf	----	Δ Δ 0 -
ORCC #opr8i	(CCR) + M ⇒ CCR Logical OR CCR with Memory	IMM	14 ii	P	P	↑↑↑↑	↑↑↑↑
PSHA	(SP) - 1 ⇒ SP; (A) ⇒ M <sub>(SP)</sub> Push Accumulator A onto Stack	INH	36	Os	Os	----	----
PSHB	(SP) - 1 ⇒ SP; (B) ⇒ M <sub>(SP)</sub> Push Accumulator B onto Stack	INH	37	Os	Os	----	----
PSHC	(SP) - 1 ⇒ SP; (CCR) ⇒ M <sub>(SP)</sub> Push CCR onto Stack	INH	39	Os	Os	----	----
PSHD	(SP) - 2 ⇒ SP; (A:B) ⇒ M <sub>(SP)</sub> :M <sub>(SP+1)</sub> Push D Accumulator onto Stack	INH	3B	OS	OS	----	----
PSHX	(SP) - 2 ⇒ SP; (X <sub>H</sub> :X <sub>L</sub> ) ⇒ M <sub>(SP)</sub> :M <sub>(SP+1)</sub> Push Index Register X onto Stack	INH	34	OS	OS	----	----
PSHY	(SP) - 2 ⇒ SP; (Y <sub>H</sub> :Y <sub>L</sub> ) ⇒ M <sub>(SP)</sub> :M <sub>(SP+1)</sub> Push Index Register Y onto Stack	INH	35	OS	OS	----	----
PULA	(M <sub>(SP)</sub> ) ⇒ A; (SP) + 1 ⇒ SP Pull Accumulator A from Stack	INH	32	ufo	ufo	----	----
PULB	(M <sub>(SP)</sub> ) ⇒ B; (SP) + 1 ⇒ SP Pull Accumulator B from Stack	INH	33	ufo	ufo	----	----
PULC	(M <sub>(SP)</sub> ) ⇒ CCR; (SP) + 1 ⇒ SP Pull CCR from Stack	INH	38	ufo	ufo	Δ Δ Δ Δ	Δ Δ Δ Δ
PULD	(M <sub>(SP)</sub> :M <sub>(SP+1)</sub> ) ⇒ A:B; (SP) + 2 ⇒ SP Pull D from Stack	INH	3A	Ufo	Ufo	----	----
PULX	(M <sub>(SP)</sub> :M <sub>(SP+1)</sub> ) ⇒ X <sub>H</sub> :X <sub>L</sub> ; (SP) + 2 ⇒ SP Pull Index Register X from Stack	INH	30	Ufo	Ufo	----	----
PULY	(M <sub>(SP)</sub> :M <sub>(SP+1)</sub> ) ⇒ Y <sub>H</sub> :Y <sub>L</sub> ; (SP) + 2 ⇒ SP Pull Index Register Y from Stack	INH	31	Ufo	Ufo	----	----
REV	MIN-MAX rule evaluation Find smallest rule input (MIN). Store to rule outputs unless fuzzy output is already larger (MAX).  For rule weights see REVW.  Each rule input is an 8-bit offset from the base address in Y. Each rule output is an 8-bit offset from the base address in Y. \$FE separates rule inputs from rule outputs. \$FF terminates the rule list.  REV may be interrupted.	Special	18 3A	Orf(t,tx)O (exit + re-entry replaces comma above if interrupted) ff + Orf(t,      ff + Orf(t,	Orf(t,tx)O (exit + re-entry replaces comma above if interrupted) ff + Orf(t,      ff + Orf(t,	--?--	??Δ?
REVV	MIN-MAX rule evaluation Find smallest rule input (MIN), Store to rule outputs unless fuzzy output is already larger (MAX).  Rule weights supported, optional.  Each rule input is the 16-bit address of a fuzzy input. Each rule output is the 16-bit address of a fuzzy output. The value \$FFF separates rule inputs from rule outputs. \$FFFF terminates the rule list.  REVV may be interrupted.	Special	18 3B	ORf(t,Tx)O (loop to read weight if enabled) (r,RfRf) (exit + re-entry replaces comma above if interrupted) ffff + ORf(t,      fff + ORf(t,	ORf(t,Tx)O (loop to read weight if enabled) (r,RfRf) (exit + re-entry replaces comma above if interrupted) ffff + ORf(t,      fff + ORf(t,	--?--	??Δ!

Table A-1. Instruction Set Summary (Sheet 11 of 14)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail		S X H I	N Z V C
				HCS12	M68HC12		
ROL <i>opr16a</i> ROL <i>opr0_xysp</i> ROL <i>opr9_xysp</i> ROL <i>opr16_xysp</i> ROL [D, <i>xysp</i> ] ROL [ <i>opr16_xysp</i> ] ROLA ROLB	 Rotate Memory Left through Carry  Rotate A Left through Carry Rotate B Left through Carry	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	75 hh 11 65 xb 65 xb ff 65 xb ee ff 65 xb 65 xb ee ff 45 55	rPwO rPw rPwO frPwP fIfPrPw fIPrPw O O	rOPw rPw rPOw frPPw fIfPrPw fIPrPw O O	---- ---- ---- ---- ---- ---- ---- ----	Δ Δ
ROR <i>opr16a</i> ROR <i>opr0_xysp</i> ROR <i>opr9_xysp</i> ROR <i>opr16_xysp</i> ROR [D, <i>xysp</i> ] ROR [ <i>opr16_xysp</i> ] RORA RORB	 Rotate Memory Right through Carry  Rotate A Right through Carry Rotate B Right through Carry	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	76 hh 11 66 xb 66 xb ff 66 xb ee ff 66 xb 66 xb ee ff 46 56	rPwO rPw rPwO frPwP fIfPrPw fIPrPw O O	rOPw rPw rPOw frPPw fIfPrPw fIPrPw O O	---- ---- ---- ---- ---- ---- ----	Δ Δ
RTC	$(M_{(SP)} \Rightarrow PPAGE; (SP) + 1 \Rightarrow SP;$ $(M_{(SP)}:M_{(SP+1)}) \Rightarrow PC_H:PC_L;$ $(SP) + 2 \Rightarrow SP$ Return from Call	INH	0A	uUnfPPP	uUnPPP	----	----
RTI	$(M_{(SP)} \Rightarrow CCR; (SP) + 1 \Rightarrow SP$ $(M_{(SP)}:M_{(SP+1)}) \Rightarrow B:A; (SP) + 2 \Rightarrow SP$ $(M_{(SP)}:M_{(SP+1)}) \Rightarrow X_H:X_L; (SP) + 4 \Rightarrow SP$ $(M_{(SP)}:M_{(SP+1)}) \Rightarrow PC_H:PC_L; (SP) - 2 \Rightarrow SP$ $(M_{(SP)}:M_{(SP+1)}) \Rightarrow Y_H:Y_L; (SP) + 4 \Rightarrow SP$ Return from Interrupt	INH	0B	uUUUUPPP (with interrupt pending) uUUUUvfPPP	uUUUUPPP uUUUUvfPPP	Δ Δ Δ Δ Δ Δ Δ Δ	Δ Δ Δ Δ Δ Δ Δ Δ
RTS	$(M_{(SP)}:M_{(SP+1)}) \Rightarrow PC_H:PC_L;$ $(SP) + 2 \Rightarrow SP$ Return from Subroutine	INH	3D	UfPPP	UfPPP	----	----
SBA	$(A) - (B) \Rightarrow A$ Subtract B from A	INH	18 16	OO	OO	----	Δ Δ Δ Δ
SBCA # <i>opr8i</i> SBCA <i>opr8a</i> SBCA <i>opr16a</i> SBCA <i>opr0_xysp</i> SBCA <i>opr9_xysp</i> SBCA <i>opr16_xysp</i> SBCA [D, <i>xysp</i> ] SBCA [ <i>opr16_xysp</i> ]	$(A) - (M) - C \Rightarrow A$ Subtract with Borrow from A	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	82 ii 92 dd B2 hh 11 A2 xb A2 xb ff A2 xb ee ff A2 xb A2 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rFP rOP rFP rPO frPP fIfrFP fIPrFP	---- ---- ---- ---- ---- ---- ----	Δ Δ
SBCB # <i>opr8i</i> SBCB <i>opr8a</i> SBCB <i>opr16a</i> SBCB <i>opr0_xysp</i> SBCB <i>opr9_xysp</i> SBCB <i>opr16_xysp</i> SBCB [D, <i>xysp</i> ] SBCB [ <i>opr16_xysp</i> ]	$(B) - (M) - C \Rightarrow B$ Subtract with Borrow from B	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C2 ii D2 dd F2 hh 11 E2 xb E2 xb ff E2 xb ee ff E2 xb E2 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rFP rOP rFP rPO frPP fIfrFP fIPrFP	---- ---- ---- ---- ---- ---- ----	Δ Δ
SEC	$1 \Rightarrow C$ Translates to ORCC #01	IMM	14 01	P	P	----	---1
SEI	$1 \Rightarrow I$ ; (inhibit I interrupts) Translates to ORCC #10	IMM	14 10	P	P	---1	----
SEV	$1 \Rightarrow V$ Translates to ORCC #02	IMM	14 02	P	P	----	--1-
SEX <i>abc,dxys</i>	$\$00:(r1) \Rightarrow r2$ if r1, bit 7 is 0 or $\$FF:(r1) \Rightarrow r2$ if r1, bit 7 is 1  Sign Extend 8-bit r1 to 16-bit r2 r1 may be A, B, or CCR r2 may be D, X, Y, or SP  Alternate mnemonic for TFR r1, r2	INH	B7 eb	P	P	----	----

## Table A-1. Instruction Set Summary (Sheet 12 of 14)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail HCS12 M68HC12	S X H I	N Z V C
STAA <i>opr8a</i> STAA <i>opr16a</i> STAA <i>opr0_xysp</i> STAA <i>opr9_xysp</i> STAA <i>opr16_xysp</i> STAA [D, <i>xysp</i> ] STAA [ <i>opr16_xysp</i> ]	(A) ⇒ M Store Accumulator A to Memory	DIR EXT IDX IDX1 IDX2 [D, IDX] [IDX2]	5A dd 7A hh ll 6A xb 6A xb ff 6A xb ee ff 6A xb 6A xb ee ff	Pw PwO Pw PwO PwP PIfPw PIfPw Pw wOP Pw PwO PwP PIfPw PIfPw	----	Δ Δ 0 -
STAB <i>opr8a</i> STAB <i>opr16a</i> STAB <i>opr0_xysp</i> STAB <i>opr9_xysp</i> STAB <i>opr16_xysp</i> STAB [D, <i>xysp</i> ] STAB [ <i>opr16_xysp</i> ]	(B) ⇒ M Store Accumulator B to Memory	DIR EXT IDX IDX1 IDX2 [D, IDX] [IDX2]	5B dd 7B hh ll 6B xb 6B xb ff 6B xb ee ff 6B xb 6B xb ee ff	Pw PwO Pw PwO PwP PIfPw PIfPw Pw wOP Pw PwO PwP PIfPw PIfPw	----	Δ Δ 0 -
STD <i>opr8a</i> STD <i>opr16a</i> STD <i>opr0_xysp</i> STD <i>opr9_xysp</i> STD <i>opr16_xysp</i> STD [D, <i>xysp</i> ] STD [ <i>opr16_xysp</i> ]	(A) ⇒ M <sub>L</sub> , (B) ⇒ M+1 Store Double Accumulator	DIR EXT IDX IDX1 IDX2 [D, IDX] [IDX2]	5C dd 7C hh ll 6C xb 6C xb ff 6C xb ee ff 6C xb 6C xb ee ff	PW PWO PW PWO PWP PIfPW PIfPW PW wOP PW PWO PWP PIfPW PIfPW	----	Δ Δ 0 -
STOP	(SP) - 2 ⇒ SP; RTN <sub>H</sub> :RTN <sub>L</sub> ⇒ M <sub>(SP)</sub> :M <sub>(SP+1)</sub> ; (SP) - 2 ⇒ SP; (Y <sub>H</sub> :Y <sub>L</sub> ) ⇒ M <sub>(SP)</sub> :M <sub>(SP+1)</sub> ; (SP) - 2 ⇒ SP; (X <sub>H</sub> :X <sub>L</sub> ) ⇒ M <sub>(SP)</sub> :M <sub>(SP+1)</sub> ; (SP) - 2 ⇒ SP; (B:A) ⇒ M <sub>(SP)</sub> :M <sub>(SP+1)</sub> ; (SP) - 1 ⇒ SP; (CCR) ⇒ M <sub>(SP)</sub> ; STOP All Clocks  Registers stacked to allow quicker recovery by interrupt.  If S control bit = 1, the STOP instruction is disabled and acts like a two-cycle NOP.	INH	18 3E	(entering STOP) OOSSSSf OOSSSfSs (exiting STOP) fVfPPP fVfPPP (continue) ff fO (if STOP disabled) OO OO	----	----
STS <i>opr8a</i> STS <i>opr16a</i> STS <i>opr0_xysp</i> STS <i>opr9_xysp</i> STS <i>opr16_xysp</i> STS [D, <i>xysp</i> ] STS [ <i>opr16_xysp</i> ]	(SP <sub>H</sub> :SP <sub>L</sub> ) ⇒ M:M+1 Store Stack Pointer	DIR EXT IDX IDX1 IDX2 [D, IDX] [IDX2]	5F dd 7F hh ll 6F xb 6F xb ff 6F xb ee ff 6F xb 6F xb ee ff	PW PWO PW PWO PWP PIfPW PIfPW PW wOP PW PWO PWP PIfPW PIfPW	----	Δ Δ 0 -
STX <i>opr8a</i> STX <i>opr16a</i> STX <i>opr0_xysp</i> STX <i>opr9_xysp</i> STX <i>opr16_xysp</i> STX [D, <i>xysp</i> ] STX [ <i>opr16_xysp</i> ]	(X <sub>H</sub> :X <sub>L</sub> ) ⇒ M:M+1 Store Index Register X	DIR EXT IDX IDX1 IDX2 [D, IDX] [IDX2]	5E dd 7E hh ll 6E xb 6E xb ff 6E xb ee ff 6E xb 6E xb ee ff	PW PWO PW PWO PWP PIfPW PIfPW PW wOP PW PWO PWP PIfPW PIfPW	----	Δ Δ 0 -
STY <i>opr8a</i> STY <i>opr16a</i> STY <i>opr0_xysp</i> STY <i>opr9_xysp</i> STY <i>opr16_xysp</i> STY [D, <i>xysp</i> ] STY [ <i>opr16_xysp</i> ]	(Y <sub>H</sub> :Y <sub>L</sub> ) ⇒ M:M+1 Store Index Register Y	DIR EXT IDX IDX1 IDX2 [D, IDX] [IDX2]	5D dd 7D hh ll 6D xb 6D xb ff 6D xb ee ff 6D xb 6D xb ee ff	PW PWO PW PWO PWP PIfPW PIfPW PW wOP PW PWO PWP PIfPW PIfPW	----	Δ Δ 0 -
SUBA # <i>opr8i</i> SUBA <i>opr8a</i> SUBA <i>opr16a</i> SUBA <i>opr0_xysp</i> SUBA <i>opr9_xysp</i> SUBA <i>opr16_xysp</i> SUBA [D, <i>xysp</i> ] SUBA [ <i>opr16_xysp</i> ]	(A) - (M) ⇒ A Subtract Memory from Accumulator A	IMM DIR EXT IDX IDX1 IDX2 [D, IDX] [IDX2]	80 ii 90 dd B0 hh ll A0 xb A0 xb ff A0 xb ee ff A0 xb A0 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf P rPf rPO rPf rPO frPP fIfrPf fIPrPf	----	Δ Δ Δ Δ

Table A-1. Instruction Set Summary (Sheet 13 of 14)

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail		S X H I	N Z V C
				HCS12	M68HC12		
SUBB #opr8i SUBB opr8a SUBB opr16a SUBB oprx0_xysp SUBB oprx9_xysp SUBB oprx16_xysp SUBB [D,xysp] SUBB [opr16,xysp]	(B) – (M) ⇒ B Subtract Memory from Accumulator B	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	C0 ii D0 dd F0 hh ll E0 xb E0 xb ff E0 xb ee ff E0 xb E0 xb ee ff	P rPf rPO rPf rPO frPP fIfrPf fIPrPf	P rPf rOP rPf rPO frPP fIfrPf fIPrPf	----	Δ Δ Δ Δ
SUBD #opr16i SUBD opr8a SUBD opr16a SUBD oprx0_xysp SUBD oprx9_xysp SUBD oprx16_xysp SUBD [D,xysp] SUBD [opr16,xysp]	(D) – (M:M+1) ⇒ D Subtract Memory from D (A:B)	IMM DIR EXT IDX IDX1 IDX2 [D,IDX] [IDX2]	83 jj kk 93 dd B3 hh ll A3 xb A3 xb ff A3 xb ee ff A3 xb A3 xb ee ff	PO RPf RPO RPf RPO frPP fIfrPf fIPrPf	OP RfP ROP RfP RPO frPP fIfrPf fIPrPf	----	Δ Δ Δ Δ
SWI	(SP) – 2 ⇒ SP; RTN <sub>H</sub> :RTN <sub>L</sub> ⇒ M <sub>(SP)</sub> :M <sub>(SP+1)</sub> ; (SP) – 2 ⇒ SP; (Y <sub>H</sub> :Y <sub>L</sub> ) ⇒ M <sub>(SP)</sub> :M <sub>(SP+1)</sub> ; (SP) – 2 ⇒ SP; (X <sub>H</sub> :X <sub>L</sub> ) ⇒ M <sub>(SP)</sub> :M <sub>(SP+1)</sub> ; (SP) – 2 ⇒ SP; (B:A) ⇒ M <sub>(SP)</sub> :M <sub>(SP+1)</sub> ; (SP) – 1 ⇒ SP; (CCR) ⇒ M <sub>(SP)</sub> 1 ⇒ I; (SWI Vector) ⇒ PC Software Interrupt	INH	3F	VSPSSPSsP* (for Reset) VfPPP	VSPSSPSsP* VfPPP	---1 11-1	----
*The CPU also uses the SWI microcode sequence for hardware interrupts and unimplemented opcode traps. Reset uses the VfPPP variation of this sequence.							
TAB	(A) ⇒ B Transfer A to B	INH	18 0E	OO	OO	----	Δ Δ 0 –
TAP	(A) ⇒ CCR Translates to TFR A, CCR	INH	B7 02	P	P	Δ ↓ Δ Δ	Δ Δ Δ Δ
TBA	(B) ⇒ A Transfer B to A	INH	18 0F	OO	OO	----	Δ Δ 0 –
TBEQ abdxys,rel9	If (cntr) = 0, then Branch; else Continue to next instruction  Test Counter and Branch if Zero (cntr = A, B, D, X, Y, or SP)	REL (9-bit)	04 1b rr	PPP (branch) PPO (no branch)	PPP	----	----
TBL oprx0_xysp	(M) + [(B) × ((M+1) – (M))] ⇒ A 8-Bit Table Lookup and Interpolate  Initialize B, and index before TBL. <ea> points at first 8-bit table entry (M) and B is fractional part of lookup value.  (no indirect addressing modes or extensions allowed)	IDX	18 3D xb	ORffffP	OrrffffP	----	Δ Δ – Δ ?  C Bit is undefined in HC12
TBNE abdxys,rel9	If (cntr) not = 0, then Branch; else Continue to next instruction  Test Counter and Branch if Not Zero (cntr = A, B, D, X, Y, or SP)	REL (9-bit)	04 1b rr	PPP (branch) PPO (no branch)	PPP	----	----
TFR abcdxys,abcdxys	(r1) ⇒ r2 or \$00:(r1) ⇒ r2 or (r1[7:0]) ⇒ r2  Transfer Register to Register r1 and r2 may be A, B, CCR, D, X, Y, or SP	INH	B7 eb	P	P	---- or Δ ↓ Δ Δ	---- Δ Δ Δ Δ
TPA	(CCR) ⇒ A Translates to TFR CCR, A	INH	B7 20	P	P	----	----

**Table A-1. Instruction Set Summary (Sheet 14 of 14)**

Source Form	Operation	Addr. Mode	Machine Coding (hex)	Access Detail		S X H I	N Z V C
				HCS12	M68HC12		
TRAP <i>trapnum</i>	$(SP) - 2 \Rightarrow SP;$ $RTN_H:RTN_L \Rightarrow M_{(SP)}:M_{(SP+1)};$ $(SP) - 2 \Rightarrow SP; (Y_H:Y_L) \Rightarrow M_{(SP)}:M_{(SP+1)};$ $(SP) - 2 \Rightarrow SP; (X_H:X_L) \Rightarrow M_{(SP)}:M_{(SP+1)};$ $(SP) - 2 \Rightarrow SP; (B:A) \Rightarrow M_{(SP)}:M_{(SP+1)};$ $(SP) - 1 \Rightarrow SP; (CCR) \Rightarrow M_{(SP)}$ $1 \Rightarrow I; (TRAP\ Vector) \Rightarrow PC$  Unimplemented opcode trap	INH	18 <i>tn</i> $tn = \$30-\$39$ or $\$40-\$FF$	OVSPSSPSSp	OfVSPSSPSSp	--- 1	----
TST <i>opr16a</i> TST <i>opr0_xysp</i> TST <i>opr9_xysp</i> TST <i>opr16_xysp</i> TST <i>[D,xysp]</i> TST <i>[opr16,xysp]</i> TSTA TSTB	(M) - 0 Test Memory for Zero or Minus  (A) - 0      Test A for Zero or Minus (B) - 0      Test B for Zero or Minus	EXT IDX IDX1 IDX2 [D,IDX] [IDX2] INH INH	F7 hh ll E7 xb E7 xb ff E7 xb ee ff E7 xb E7 xb ee ff 97 D7	rPO rPf rPO frPP fIfrPf fIPrPf O O	rOP rFP rPO frPP fIfrFP fIPrFP O O	----	Δ Δ 0 0
TSX	$(SP) \Rightarrow X$ <i>Translates to TFR SP,X</i>	INH	B7 75	P	P	----	----
TSY	$(SP) \Rightarrow Y$ <i>Translates to TFR SP,Y</i>	INH	B7 76	P	P	----	----
TXS	$(X) \Rightarrow SP$ <i>Translates to TFR X,SP</i>	INH	B7 57	P	P	----	----
TYS	$(Y) \Rightarrow SP$ <i>Translates to TFR Y,SP</i>	INH	B7 67	P	P	----	----
WAI	$(SP) - 2 \Rightarrow SP;$ $RTN_H:RTN_L \Rightarrow M_{(SP)}:M_{(SP+1)};$ $(SP) - 2 \Rightarrow SP; (Y_H:Y_L) \Rightarrow M_{(SP)}:M_{(SP+1)};$ $(SP) - 2 \Rightarrow SP; (X_H:X_L) \Rightarrow M_{(SP)}:M_{(SP+1)};$ $(SP) - 2 \Rightarrow SP; (B:A) \Rightarrow M_{(SP)}:M_{(SP+1)};$ $(SP) - 1 \Rightarrow SP; (CCR) \Rightarrow M_{(SP)};$ WAIT for interrupt	INH	3E	OSSSSsf (after interrupt) fVfPPP	OSSsfSsf VfPPP	---- or --- 1 or - 1 - 1	---- ---- ----
WAV	$B \sum_{i=1} S_i F_i \Rightarrow Y:D \quad \text{and} \quad \sum_{i=1}^B F_i \Rightarrow X$  Calculate Sum of Products and Sum of Weights for Weighted Average Calculation  Initialize B, X, and Y before WAV. B specifies number of elements. X points at first element in $S_i$ list. Y points at first element in $F_i$ list.  All $S_i$ and $F_i$ elements are 8-bits.  If interrupted, six extra bytes of stack used for intermediate values	Special	18 3C	Of(frr,ffff)O Of(frr,ffff)O (add if interrupt) SSS + UUUrr,      SSSf + UUUrr		-- ? -	? Δ ? ?
wavr  pseudo-instruction	<i>see WAV</i>  Resume executing an interrupted WAV instruction (recover intermediate results from stack rather than initializing them to zero)	Special	3C	UUUrr,ffff      UUUrrfffff (frr,ffff)O      (frr,ffff)O (exit + re-entry replaces comma above if interrupted) SSS + UUUrr,      SSSf + UUUrr		-- ? -	? Δ ? ?
XGDX	$(D) \Leftrightarrow (X)$ <i>Translates to EXG D, X</i>	INH	B7 C5	P	P	----	----
XGDY	$(D) \Leftrightarrow (Y)$ <i>Translates to EXG D, Y</i>	INH	B7 C6	P	P	----	----

**Table A-4. Indexed Addressing Mode Summary**

Postbyte Code (xb)	Operand Syntax	Comments
rr0nnnnn	,r n,r -n,r	<b>5-bit constant offset</b> n = -16 to +15 rr can specify X, Y, SP, or PC
111rr0zs	n,r -n,r	<b>Constant offset</b> (9- or 16-bit signed) z- 0 = 9-bit with sign in LSB of postbyte (s) 1 = 16-bit if z = s = 1, 16-bit offset indexed-indirect (see below) rr can specify X, Y, SP, or PC
rr1pnnnn	n,-r n,+r n,r- n,r+	<b>Auto predecrement, preincrement, postdecrement, or postincrement;</b> p = pre-(0) or post-(1), n = -8 to -1, +1 to +8 rr can specify X, Y, or SP (PC not a valid choice)
111rr1aa	A,r B,r D,r	<b>Accumulator offset</b> (unsigned 8-bit or 16-bit) aa - 00 = A 01 = B 10 = D (16-bit) 11 = see accumulator D offset indexed-indirect rr can specify X, Y, SP, or PC
111rr011	[n,r]	<b>16-bit offset indexed-indirect</b> rr can specify X, Y, SP, or PC
111rr111	[D,r]	<b>Accumulator D offset indexed-indirect</b> rr can specify X, Y, SP, or PC

**Table A-8. Hexadecimal to ASCII Conversion**

Hex	ASCII	Hex	ASCII	Hex	ASCII	Hex	ASCII
\$00	NUL	\$20	SP <i>space</i>	\$40	@	\$60	` <i>grave</i>
\$01	SOH	\$21	!	\$41	A	\$61	a
\$02	STX	\$22	" <i>quote</i>	\$42	B	\$62	b
\$03	ETX	\$23	#	\$43	C	\$63	c
\$04	EOT	\$24	\$	\$44	D	\$64	d
\$05	ENQ	\$25	%	\$45	E	\$65	e
\$06	ACK	\$26	&	\$46	F	\$66	f
\$07	BEL <i>beep</i>	\$27	' <i>apost.</i>	\$47	G	\$67	g
\$08	BS <i>back sp</i>	\$28	(	\$48	H	\$68	h
\$09	HT <i>tab</i>	\$29	)	\$49	I	\$69	i
\$0A	LF <i>linefeed</i>	\$2A	*	\$4A	J	\$6A	j
\$0B	VT	\$2B	+	\$4B	K	\$6B	k
\$0C	FF	\$2C	, <i>comma</i>	\$4C	L	\$6C	l
\$0D	CR <i>return</i>	\$2D	- <i>dash</i>	\$4D	M	\$6D	m
\$0E	SO	\$2E	. <i>period</i>	\$4E	N	\$6E	n
\$0F	SI	\$2F	/	\$4F	O	\$6F	o
\$10	DLE	\$30	0	\$50	P	\$70	p
\$11	DC1	\$31	1	\$51	Q	\$71	q
\$12	DC2	\$32	2	\$52	R	\$72	r
\$13	DC3	\$33	3	\$53	S	\$73	s
\$14	DC4	\$34	4	\$54	T	\$74	t
\$15	NAK	\$35	5	\$55	U	\$75	u
\$16	SYN	\$36	6	\$56	V	\$76	v
\$17	ETB	\$37	7	\$57	W	\$77	w
\$18	CAN	\$38	8	\$58	X	\$78	x
\$19	EM	\$39	9	\$59	Y	\$79	y
\$1A	SUB	\$3A	:	\$5A	Z	\$7A	z
\$1B	ESCAPE	\$3B	;	\$5B	[	\$7B	{
\$1C	FS	\$3C	<	\$5C	\	\$7C	
\$1D	GS	\$3D	=	\$5D	]	\$7D	}
\$1E	RS	\$3E	>	\$5E	^	\$7E	~
\$1F	US	\$3F	?	\$5F	_ <i>under</i>	\$7F	DEL <i>delete</i>





# TP 4 – Carte à microcontrôleur HCS12, modes d'adressage

## Objectifs :

- utiliser une carte à microcontrôleur
- analyser une partie du schéma de la carte (cf. annexe)
- tester les différents modes d'adressage du microcontrôleur sous environnement CodeWarrior

## 1 Carte CML12SDP256

### 1.1 La carte

Les caractéristiques *on chip* (sur le microcontrôleur) principales sont les suivantes :

- 12 ko RAM,
- 256 ko flash EEPROM,
- 4 ko EEPROM,
- 89 canaux d'entrées-sorties numériques,
- bus externe multiplexé (adresses et données),
- 2 ports série asynchrones (SCI),
- 3 ports série synchrones (SPI),
- 2 convertisseurs analogiques-numériques 10 bits à 8 entrées,
- timer 16 bits,
- horloge réglable de 4 à 24 MHz.

Le microcontrôleur peut adresser en externe 256 Ko de RAM (circuits U4 et U5 du schéma).

Analysez le décodage d'adresse réalisé par le circuit U9 pour accéder aux parties hautes et basses des mots mémoire 16 bits (feuille 2/2, repère B6) : ses entrées sont A0, LSTRB (indicateur d'adresse impaire) et ECLK (horloge) ; ses sorties sont CS0 et CS1.

Le circuit U10 permet de générer la commande d'écriture ou lecture de la mémoire : ses entrées sont RW (read/write du HCS12) et ECLK (horloge) ; ses sorties sont WE (write enable) et OE (output enable).

### 1.2 Le plan mémoire

L'espace mémoire adressable par le microcontrôleur de 64 ko est organisé selon le tableau suivant.

Adresses	Type de mémoire	Application
\$0000 - \$0FFF	registres HCS12 / EEPROM 4 ko	accès aux registres du HCS12
\$1000 - \$3FFF	RAM 12 ko	mémoire utilisateur
\$4000 - \$FEFF	flash EEPROM ou RAM	mémoire flash
\$FF00 - \$FFFF	EEPROM	Vecteurs d'interruption, BDM

Cette organisation correspond au mode normal puce seule (normal single chip) et à la configuration du microcontrôleur au reset.

## 2. Modes d'adressage

### 2.1 Mode d'adressage inhérent (INH)

Ce mode se reconnaît par l'absence d'opérande. Il ne nécessite pas d'accéder à la mémoire externe par les bus d'adresses et de données.

Exemple : INX

Description : l'instruction incrémente le registre X de 1.

## 2.2 Mode d'adressage immédiat (IMM)

L'opérande de l'instruction utilise une donnée numérique (constante) : le nombre est écrit en le précédant du signe #.

Exemple : LDAA #\$22

Description : l'instruction charge le nombre hexadécimal \$22 dans le registre A.

## 2.3 Mode d'adressage direct (DIR)

Ce mode est utilisé pour accéder à une donnée dont l'adresse tient sur 1 octet.

Exemple : SUBA \$20

Description : l'instruction soustrait le contenu de l'adresse \$20 du contenu du registre A et place le résultat dans A.

## 2.4 Mode d'adressage étendu (EXT)

Ce mode est utilisé pour accéder à une donnée dont l'adresse tient sur 2 octets.

Exemple : ADDD \$1030

Description : l'instruction additionne le contenu des adresses \$1030 : \$1031 au contenu du registre D et place le résultat dans D : dans cette opération, le registre B (partie basse de D) est additionné au contenu de l'adresse \$1031 (partie basse du mot mémoire).

## 2.5 Mode d'adressage indexé (IDX)

Pour réaliser des programmes de boucles, les registres d'index (X ou Y), de programme PC ou de pile S sont souvent utilisés dans les instructions.

L'adresse effective est calculée en additionnant deux parties : le registre de base et le déplacement. Le déplacement est signé avec 3 formats possibles : 5, 9 ou 16 bits.

Avec les registres X et Y, on peut également utiliser la pré-décrémentation, la post-décrémentation, la pré-incrémentation ou la post-incrémentation. L'incrément ou le décrétement est un nombre de 1 à 8 ; le déplacement est alors nul.

Exemple : ADDA \$10,X                      avec X = \$3000

Description : l'instruction extrait le contenu de l'adresse \$3010, l'additionne au contenu du registre A et place le résultat dans A.

L'adresse effective peut aussi être calculée par indirection avec un registre D ou une adresse mémoire 16 bits : on additionne alors le registre de base avec le déplacement spécifié par le contenu 16 bits. L'utilisation des crochets est obligatoire pour l'écriture de l'opérande.

Exemple : LDAA [D,X]    avec D = \$2035 , X = \$1000 , (\$3035) = \$20 , (\$3036) = \$02

Description : l'instruction calcule l'adresse \$1000 + \$2035 soit \$3035, extrait les contenus des adresses \$3035 et \$3036 (soit \$2002), extrait le contenu de l'adresse \$2002 et le place dans le registre A.

## 2.6 Mode d'adressage relatif (REL)

Ce mode est utilisé dans les instructions de branchement conditionnel (Bcc) ou inconditionnel (JMP). Il permet le changement du registre de programme PC.

Exemple :

```
$2000    BNE SUITE
$2002    STAA $C000
...      ...      ...
$2012 SUITE: INX
```

Description : l'instruction BNE exécute le branchement si le bit Z du registre d'état est à 0. Le registre PC initialement à \$2002 est additionné d'un déplacement relatif qui est ici \$10 (il remplace le label SUITE) : sa nouvelle valeur est \$2012. Si le bit Z vaut 1, la valeur de PC est inchangée : l'instruction située en \$2002 est exécutée.

## 3. Travail à réaliser

a - Programme de copie de mémoire :

- précharger une mémoire d'adresse \$1500 avec la valeur \$0A
- effectuer une 1ère boucle qui charge une zone mémoire d'adresses \$1500 à \$1600 avec la valeur \$0A
- effectuer une deuxième boucle à la suite qui effectue la recopie de la zone mémoire d'adresse \$1500 à \$1600 dans la zone mémoire d'adresse \$1800 à \$1900

Indiquez en commentaires les modes d'adressage utilisés.

b - Test :

Une fois le programme de copie de mémoire effectué, demander à votre chargé de TP une feuille de test à compléter et à lui rendre en fin de séance concernant les différents modes d'adressages du HCS12 (polycopié de TP autorisé).







# TP 5 – Sous-programmes

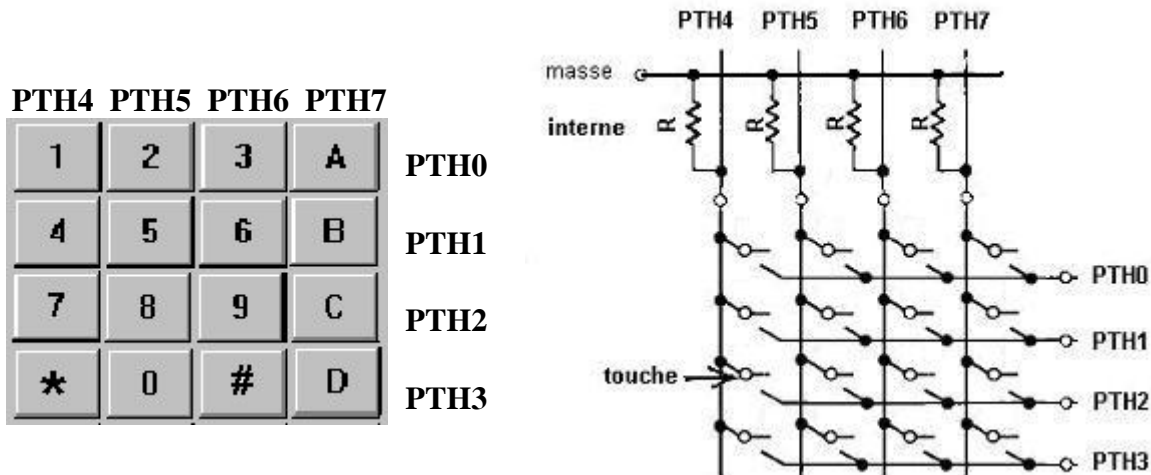
## Objectifs :

- utiliser un clavier connecté au port H du HCS12
- développer et tester des sous-programmes sous environnement CodeWarrior

## 1. Clavier

### 1.1 Description

Le clavier est constitué de 16 touches réparties en matrice de quatre colonnes (reliées aux broches PTH4 à PTH7) et quatre lignes (reliées aux broches PTH0 à PTH3).



*Aspect externe*

*Schéma électrique*

Des résistances R internes au port H permettent de fixer le niveau des lignes PTH4 à PTH7 à 0 V au repos.

### 1.2 Principe de décodage

L'activation d'une ligne à 1 provoque un niveau 1 sur une colonne si une touche de la colonne est appuyée. Dans le cas contraire, les colonnes ont un niveau 0 du fait de la connexion des résistances à la masse.

Par exemple, si PTH0 est mis à 1, la broche PTH5 sera à 1 si on appuie sur la touche « 2 ».

Un procédé de décodage consiste à :

- mettre à 1 la ligne PTH0,
- lire l'état des 4 colonnes,
- si aucune colonne n'est à 1, répéter les 2 actions précédentes avec la ligne suivante.

Le procédé s'arrête pour la colonne et la ligne visitées à 1. Un compteur peut être inséré dans la boucle pour représenter un numéro de touche de 0 à 15 : ce numéro servira d'index à un tableau \_CHAINE contenant les 16 symboles des touches (code ASCII).

## 2 Port d'entrée/sortie H

### 2.1 Registre de direction des données

Les broches du port H peuvent être configurés en entrée ou en sortie selon le registre de direction des données DDRH. Le fil PTHi sera en entrée si le bit  $b_i$  de DDRH est à 0 ; il sera en



sortie si  $b_i$  de DDRH est à 1.

## 2.2 Résistance de relèvement ou d'abaissement

Les résistances qui complètent le clavier sur la figure précédente peuvent être internes au microcontrôleur. Si la résistance est reliée à la masse, on parle de résistance d'abaissement (pull-down en anglais) ; si elle est reliée à l'alimentation  $V_{CC}$ , on parle de résistance de relèvement (pull-up en anglais). Deux registres sont à initialiser pour mettre en oeuvre ces résistances : PERH (autorisation) et PPSH (polarité). On donne ci-dessous les spécifications du constructeur.

Address Offset: \$__24		Port H Pull Device Enable Register (PERH)							
		Bit 7	6	5	4	3	2	1	Bit 0
Read:		PERH7	PERH6	PERH5	PERH4	PERH3	PERH2	PERH1	PERH0
Write:									
Reset:		0	0	0	0	0	0	0	0

This register configures whether a pull-up or a pull-down device is activated, if the port is used as input. This bit has no effect if the port is used as output. Out of reset no pull device is enabled.

PERH[7:0] — Pull Device Enable Port H

1 = Either a pull-up or pull-down device is enabled.

0 = Pull-up or pull-down device is disabled.

Address Offset: \$__25		Port H Polarity Select Register (PPSH)							
		Bit 7	6	5	4	3	2	1	Bit 0
Read:		PPSH7	PPSH6	PPSH5	PPSH4	PPSH3	PPSH2	PPSH1	PPSH0
Write:									
Reset:		0	0	0	0	0	0	0	0

This register serves a dual purpose by selecting the polarity of the active interrupt edge as well as selecting a pull-up or pull-down device if enabled.

PPSH[7:0] — Polarity Select Port H

1 = Rising edge on the associated port H pin sets the associated flag bit in the PIFH register.

A pull-down device is connected to the associated port H pin, if enabled by the associated bit in register PERH and if the port is used as input.

0 = Falling edge on the associated port H pin sets the associated flag bit in the PIFH register.

A pull-up device is connected to the associated port H pin, if enabled by the associated bit in register PERH and if the port is used as input.

## 3 Travail à réaliser

On ouvrira sous CodeWarrior un projet *Projet\_asm.mcp* (cf. TP n°3). On remplacera dans le projet le fichier *main.asm* par le fichier source *main\_clavier.asm*. Ce fichier est accessible dans le dossier FichiersAssembleur (on copiera ce dossier du lecteur Y: dans le lecteur Z: ).

On inclura dans *main\_clavier.asm* les sous-programmes suivants.

### 3.1 Programme principal

Le programme principal se présente comme suit :

```
jsr  INITKEY      ; initialisation du clavier
jsr  GETKEY       ; saisie d'une touche du clavier et stockage dans l'accumulateur A
jsr  LCD_OUT      ; affichage à l'écran LCD du caractère stocké dans l'accumulateur A
BGND              ; sortie du programme principal
```

Le sous-programme LCD\_OUT n'est pas à réaliser.

### 3.2 Sous-programme INITKEY d'initialisation

Le sous-programme INITKEY doit configurer le port H, avec PTH0 à PTH3 en sortie (ligne) et PTH4 à PTH7 en entrée (colonne). Les entrées seront configurées avec des résistances d'abaissement à zéro de manière à avoir 0 V au repos en entrée.

### 3.3 Sous-programme GETKEY de saisie de caractère

Le sous-programme GETKEY sert à indiquer l'appui d'une touche par l'utilisateur. Il envoie un niveau 1 sur les 4 lignes et teste l'appui d'une touche sur les 4 colonnes. S'il n'y a pas de touches appuyées, le sous programme reboucle au début de GETKEY. S'il y a une touche appuyée, le sous programme appelle le sous-programme FINDKEY, puis rend la main au programme principal.

Lorsque l'on écrit dans le port H, il faut effectuer une pause de 1 milliseconde avant de lire à nouveau les valeurs du port H modifiée par l'appui éventuel sur une touche.

On pourra appeler le sous-programme DELAY\_1MS existant pour cela.

Il est conseillé de tester le programme bloc par bloc au fur et à mesure de sa création. Pour tester le sous-programme GETKEY, on pourra dans une première étape remplacer l'appel au sous-programme FINDKEY par le stockage d'un caractère quelconque dans l'accumulateur A.

### 3.4 Sous-programme FINDKEY de décodage de la touche appuyée

La fonction FINDKEY a pour but de récupérer le caractère correspondant à la touche du clavier qui a été appuyée. Cette valeur de caractère est récupérée à partir de la variable \_CHaine et doit être stockée dans l'accumulateur A avant le retour au programme principal.

La variable \_CHaine contient l'adresse du début de la chaîne de caractère "\*0#D789C456B123A" qui est écrite dans la mémoire. Pour accéder à un caractère de la chaîne, il va falloir ajouter un « offset » à cette adresse de début de chaîne (exemple : si l'offset vaut 7, le caractère désigné dans la chaîne de caractère est « C », situé à l'adresse \_CHaine + 7).

Le stockage dans l'accumulateur A du caractère sélectionné permet son affichage à l'écran. En effet, la fonction LCD\_OUT affiche un caractère sauvegardé dans l'accumulateur A.

Pour calculer cet offset, il est possible de procéder ainsi :

- Mise à 0 de l'offset,
  - Si la touche appuyée est située :
    - en colonne n° 1 alors offset = offset + 0
    - en colonne n° 2 alors offset = offset + 1
    - en colonne n° 3 alors offset = offset + 2
    - en colonne n° 4 alors offset = offset + 3
- puis
- en ligne n° 1 alors offset = offset + 12
  - en ligne n° 2 alors offset = offset + 8
  - en ligne n° 3 alors offset = offset + 4
  - en ligne n° 4 alors offset = offset + 0

La fonction FINDKEY devra comporter une boucle permettant les opérations suivantes :

- o allumer séparément une à une les 4 lignes,
- o déterminer la colonne qui est passée à l'état 1 (s'il y en a une),
- o mettre à jour l'offset.

Après écriture du programme, compilez le puis lancez le débogueur.

Pour exécuter le programme, utilisez la touche F5 (Start/Continue). Au final, on doit obtenir le fonctionnement suivant : l'appui sur une touche doit provoquer l'affichage du caractère correspondant sur l'afficheur LCD. On pourra aussi vérifier le contenu ASCII de l'accumulateur A (fenêtre *register*).

# TP 6 – Interruption

## Objectifs :

- déclencher par interruption une action sur le port P du HCS12
- tester des programmes d'interruption logicielle ou matérielle

## **1. Test des interruptions logicielles**

Un programme peut être interrompu par des interruptions (*exceptions*) de type logiciel (*software interrupt*). Ces interruptions provoquent la fin de l'exécution d'un programme en cours : elles peuvent être dues à des erreurs d'exécution d'une instruction, ou bien déclenchées volontairement par des instructions de type TRAP ou SWI (*software interrupt*). Ces interruptions sont non masquables et ont lieu indépendamment de la valeur du bit X ou I du registre d'état. Elles sont moins prioritaires que les interruptions de type *reset*. Lors de l'entrée dans une interruption, le microcontrôleur sauvegarde les registres (X, Y, D, SP, CCR ...) dans la pile et les restaure en quittant l'interruption.

### *Travail à effectuer :*

On souhaite tester une interruption logicielle. Dans le programme principal :

- Ecrire #\$1234 dans X, #\$5678 dans Y et #\$9009 dans D
- Ecrire l'instruction SWI qui permettra de lancer une interruption logicielle

Dans la fonction SWI\_ISR :

- Ecrire un NOP (instruction qui fait attendre un cycle d'horloge au microcontrôleur)
- Ecrire l'instruction RTI (Quel est son rôle ?)

En fin de programme :

- Initialiser le vecteur d'interruption *Vswi* avec l'adresse de la fonction d'interruption que l'on souhaite exécuter (fonction *SWI\_ISR*). Prendre comme modèle le réglage du vecteur d'interruption *Vreset* qui exécute *Entry* comme modèle.

### *Test du programme*

Compilez et chargez le programme dans le microcontrôleur. Placez un point d'arrêt sur le NOP de la fonction SWI\_ISR. Lancez le programme.

Lorsque le programme s'arrête sur le point d'arrêt, quel est l'état de la pile (située en \$3FFE dans la mémoire) ? Écrivez en commentaire dans le programme les différents éléments dans la pile.

## **2. Réglages du port P**

Dans la suite du TP, on souhaite contrôler une roue (programme Nf02Prj.exe qui se trouve dans le répertoire TP6\_Executable). Le contrôle de la roue se fait par l'intermédiaire du port P du HCS12. Les caractéristiques du port P sont les suivantes :

PTP0 à PTP2 : valeur de la vitesse est disponible sur 3 bits

PTP3 reçoit un front descendant pour signifier le passage d'un quart de tour de roue,

- le HCS12 envoie une valeur de consigne (4 bits) en sortie PTP4 à PTP7 :

PTP4 : incrémentation de vitesse de 1 unité (PTP4 = 1) ou vitesse uniforme (PTP4 = 0),

PTP5 : décrémentation de vitesse de 1 unité (PTP5 = 1) ou vitesse uniforme (PTP5 = 0),

PTP6 : multiplication par deux de l'incrément ou décrémentation de vitesse (PTP6 = 1),

PTP7 : marche (PTP7 = 0) ou arrêt (PTP7 = 1) de la roue.

### **Travail à effectuer**

#### Réglage des registres

Initialisation du port P (port de communication avec la roue) (réglez DDRP, PERP et PPSP)

- PTP0 à PTP3 sont des entrées, PTP4 à PTP7 des sorties

- Front montant (donc résistance en « Pull Down ») pour PTP0 à PTP2, Front descendant pour PTP3

### 3. Interruption matérielle

#### 2.1 Test de l'interruption IRQ

Les interruptions matérielles sont provoquées par des événements extérieurs. On distingue les interruptions de type masquables (contrôlées par un bit d'un registre particulier) et celles de type non masquables. Ici, nous prenons l'exemple simulé d'un système à microcontrôleur HCS12 qui gère la vitesse de rotation d'une grande roue de fête foraine. On va démarrer la roue grâce à l'interrupteur.

Une entrée IRQ est reliée à un interrupteur à 2 positions (en face avant) : elle est à 1 au repos (position basse de l'interrupteur). Le basculement de l'interrupteur en position haute fait passer IRQ à 0 ; ce front descendant doit conduire à l'exécution d'un programme d'interruption IRQ\_ISR (adresse \$1600) avec le plus haut niveau de priorité. Le programme IRQ\_ISR doit démarrer la roue.

#### Travail à effectuer

##### Réglage du Vecteur d'interruption

On souhaite qu'une interruption générée par l'IRQ (Virq) conduise à l'exécution de la fonction IRQ\_ISR (même principe que le vecteur d'interruption Vswi fait en 1<sup>ère</sup> partie).

##### Réglage des registres (ne pas oublier CLI et SEI)

Attention, il faut désactiver les interruptions le temps de la modification des registres (SEI pour désactiver les interruptions - CLI pour les réactiver).

Initialisation de l'interruption par IRQ par utilisation de l'interrupteur :

- l'IRQ sera détectée sur front descendant ; la broche IRQ est connectée à un circuit logique qui est l'interrupteur de la face avant (réglez INTCR).

Réglages supplémentaires du microcontrôleur (notamment la priorité des interruptions) :

- #\$F2 dans HPRI0, #\$90 dans PUCR, #\$FC dans DDRE

##### Programmation de la mise en route de la roue sur l'interrupteur

Après le réglage des registres, activez le bit du port P qui permet de bloquer la roue.

Dans le programme qui gère l'interruption sur IRQ, modifiez PTP pour mettre en route la roue.

Exécutez le programme NF02Prj.exe qui affiche la roue.

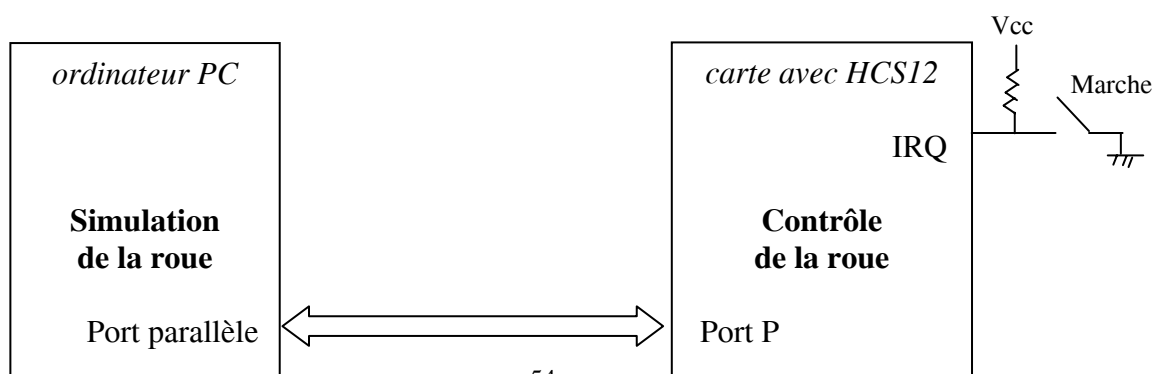
Compilez et exécutez le programme assembleur et vérifiez que la roue est à l'arrêt.

Déclenchez une interruption matérielle (par l'interrupteur) et vérifiez que la roue se met en route.

#### 2.2 Test de l'interruption sur le port P

##### Description

La communication entre le PC et le port P microcontrôleur est assurée via le port parallèle.

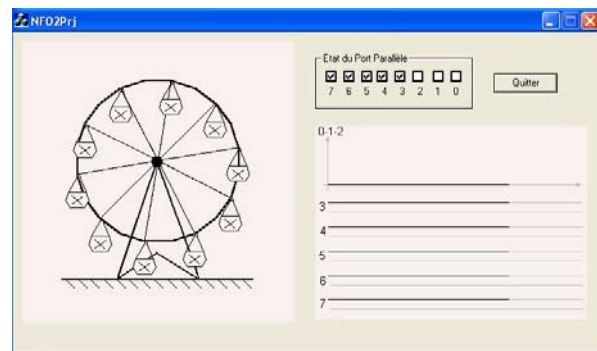
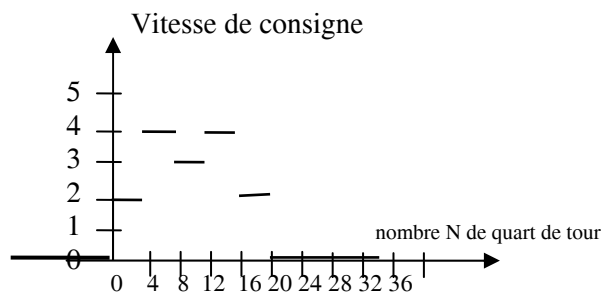


### Création d'un profil de vitesse

En tournant, la roue va générer des fronts descendants sur PTP3 tous les quarts de tour. Ces fronts permettent de contrôler la roue par les interruptions. La roue va effectuer le suivi d'une consigne de vitesse qui évolue de la manière suivante :

- Tour 1 à 2 : accélérer jusqu'à la vitesse 4 par pas de 2
- Tour 3 : ralentir d'un pas de 1
- Tour 4 : accélérer d'un pas de 1
- Tours 5 à 6 : ralentir jusqu'à l'arrêt par pas de 2

Le programme d'interruption PTP\_ISR (adresse \$1400) doit ainsi générer une consigne sur les sorties PTP4 à PTP6 comme indiquée sur la figure ci-dessous : l'envoi de consigne consiste à comparer le nombre N de quart de tour à des valeurs spécifiques et à mettre sur PTP4 à PTP6 les valeurs demandées.



### Travail à effectuer

#### Réglage du Vecteur d'interruption

Régalez le vecteur d'interruption pour qu'une interruption générée par le port P (*Vportp*) conduise à l'exécution de la fonction *PTP\_ISR* (même principe que le vecteur d'interruption *Vswi*).

#### Réglage des registres (ne pas oublier *CLI* et *SEI*)

Initialisation de l'interruption sur le bit 3 du port P (régler *PIEP*)

- Autorisez les interruptions sur PTP3.

#### Programmation du profil de roue

La rotation de la roue génère une interruptions sur PTP3 tout les  $\frac{1}{4}$  de tours, la fonction d'interruption du port P sera donc appelée automatiquement tous les  $\frac{1}{4}$  de tours.

Via la fonction d'interruption du port P, contrôlez la rotation de la roue pour qu'elle exécute le profil de vitesse indiqué dans l'énoncé (attention aux remarques qui suivent !!).

#### Remarques :

Les registres sont réinitialisés à chaque fin d'interruption. Dans la fonction d'interruption du port P, le compteur qui s'incrémente à chaque appel d'interruption doit donc être une variable mémoire.

Pour réinitialiser l'interruption sur le bit 3 du port P, activez le bit concerné sur *PIFP* en fin d'interruption. Ecrire 1 sur le bit 3 de *PIFP* permet de réinitialiser l'interruption sur le bit 3 du port P.

## Annexes

Address: \$001F Highest Priority I Interrupt Register (HPRIO)

	BIT 7	6	5	4	3	2	1	BIT 0
Read:	PSEL7	PSEL6	PSEL5	PSEL4	PSEL3	PSEL2	PSEL1	0
Write:								
Reset:	1	1	1	1	0	0	1	0

Read: anytime

Write: only if I mask in OCR = 1

PSEL7 - PSEL1 - Highest priority I interrupt select bits

The state of these bits determines which I bit maskable interrupt will be promoted to highest priority (of the I bit maskable interrupts). To promote an interrupt, the user writes the least significant byte of the associated interrupt vector address to this register. If an unimplemented vector address or a non I bit masked vector address (value higher than \$F2) is written, IRQ (\$FFF2) will be the default highest priority interrupt.

Address: Base + \$\_\_1E IRQ Control Register (INTCR)

	BIT 7	6	5	4	3	2	1	BIT 0
Read:	IRQE	IRQEN	0	0	0	0	0	0
Write:								
Reset:	0	1	0	0	0	0	0	0

Unimplemented

Read: see individual bit descriptions below

Write: see individual bit descriptions below

IRQE — IRQ Select Edge Sensitive Only

Special modes: read or write anytime

Normal and Emulation modes: read anytime, write once

1 – IRQ configured to respond only to falling edges. Falling edges on the IRQ pin will be detected anytime IRQE = 1 and will be cleared only upon a reset or the servicing of the IRQ interrupt.  
0 – IRQ configured for low level recognition.

IRQEN — External IRQ Enable

Normal, Emulation, and Special modes: read or write anytime

1 – External IRQ pin is connected to interrupt logic.  
0 – External IRQ pin is disconnected from interrupt logic.

Address: Base + \$\_\_0C Pullup Control Register (PUCR)

	BIT 7	6	5	4	3	2	1	BIT 0
Read:	PUPKE	0	0	PUPEE	0	0	PUPBE	PUPAE
Write:								

This register is used to select pull resistors for the pins associated with the core ports. Pull resistors are assigned on a per-port basis and apply to any pin in the corresponding port that is currently configured as an input. The polarity of these pull resistors is determined by chip integration. Please refer to the specific device User's Guide to determine the polarity of these resistors.

This register is not in the on-chip memory map in expanded and special peripheral modes. Therefore, these accesses will be echoed externally.

**NOTE:** These bits have no effect when the associated pin(s) are outputs. (The pull resistors are inactive.)

PUPKE — Pull-Up Port K Enable

1 – Enable pull resistors for Port K input pins.  
0 – Port K pull resistors are disabled.

PUPEE — Pull-Up Port E Enable

1 – Enable pull resistors for Port E input pins bits 7, 4–0.  
0 – Port E pull resistors on bits 7, 4–0 are disabled.

**NOTE:** Bits 5 and 6 of Port E have pull resistors which are only enabled during reset. This bit has no effect on these pins.

PUPBE — Pull-Up Port B Enable

1 – Enable pull resistors for all Port B input pins.  
0 – Port B pull resistors are disabled.

PUPAE — Pull-Up Port A Enable

1 – Enable pull resistors for all Port A input pins.

**Address Offset: \$\_\_1A Port P Data Direction Register (DDRP)**

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	DDRP7	DDRP6	DDRP5	DDRP4	DDRP3	DDRP2	DDRP1	DDRP0
Write:								
Reset:	0	0	0	0	0	0	0	0

DDRP[7:0] — Data Direction Port P

- 1 = Associated pin is configured as output.
- 0 = Associated pin is configured as input.

**Address Offset: \$\_\_1C Port P Pull Device Enable Register (PERP)**

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	PERP7	PERP6	PERP5	PERP4	PERP3	PERP2	PERP1	PERP0
Write:								
Reset:	0	0	0	0	0	0	0	0

This register configures whether a pull-up or a pull-down device is activated, if the port is used as input. This bit has no effect if the port is used as output. Out of reset no pull device is enabled.

PERP[7:0] — Pull Device Enable Port P

- 1 = Either a pull-up or pull-down device is enabled.
- 0 = Pull-up or pull-down device is disabled.

**Address Offset: \$\_\_1D Port P Polarity Select Register (PPSP)**

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	PPSP7	PPSP6	PPSP5	PPSP4	PPSP3	PPSP2	PPSP1	PPSP0
Write:								
Reset:	0	0	0	0	0	0	0	0

This register serves a dual purpose by selecting the polarity of the active interrupt edge as well as selecting a pull-up or pull-down device if enabled.

PPSP[7:0] — Polarity Select Port P

- 1 = Rising edge on the associated port P pin sets the associated flag bit in the PIFP register. A pull-down device is connected to the associated port P pin, if enabled by the associated bit in register PERP and if the port is used as input.
- 0 = Falling edge on the associated port P pin sets the associated flag bit in the PIFP register. A pull-up device is connected to the associated port P pin, if enabled by the associated bit in register PERP and if the port is used as input.

**Address Offset: \$\_\_1E Port P Interrupt Enable Register (PIEP)**

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	PIEP7	PIEP6	PIEP5	PIEP4	PIEP3	PIEP2	PIEP1	PIEP0
Write:								
Reset:	0	0	0	0	0	0	0	0

This register disables or enables on a per pin basis the edge sensitive external interrupt associated with port P.

PIEP[7:0] — Interrupt Enable Port P

- 1 = Interrupt is enabled.
- 0 = Interrupt is disabled (interrupt flag masked).

**Address Offset: \$\_\_1F Port P Interrupt Flag Register (PIFP)**

	Bit 7	6	5	4	3	2	1	Bit 0
Read:	PIFP7	PIFP6	PIFP5	PIFP4	PIFP3	PIFP2	PIFP1	PIFP0
Write:								
Reset:	0	0	0	0	0	0	0	0

Each flag is set by an active edge on the associated input pin. This could be a rising or a falling edge based on the state of the PPSP register. To clear this flag, write "1" to the corresponding bit in the PIFP register. Writing a "0" has no effect.

PIFP[7:0] — Interrupt Flags Port P

- 1 = Active edge on the associated bit has occurred (an interrupt will occur if the associated enable bit is set).  
Writing a "1" clears the associated flag.
- 0 = No active edge pending.  
Writing a "0" has no effect.