Annexe 13	**ADSP 2100 –Caractéristiques – Architecture – Notice de Programmation**	*19 pages*

Tournez la page S.V.P.

Table des Matières

Les DSPs sont optimisés (en temps de calcul et en taille du silicium) pour les applications du traitement du signal. Les applications temps réel des DSPs conduisent l'utilisateur à optimiser sa programmation aussi bien en taille qu'en vitesse. Ceci impose donc de programmer en assembleur pour que les performances de l'application soient atteintes.

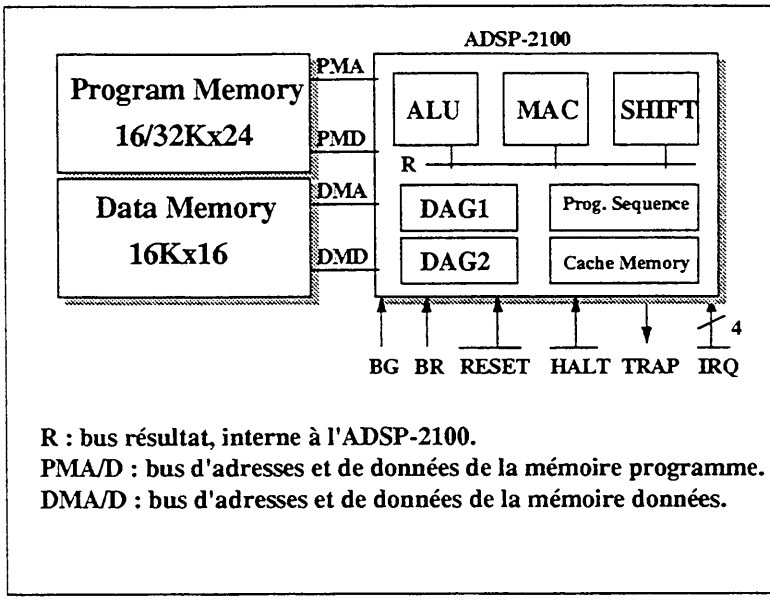
1 Caractéristiques de l'ADSP-2100

- Format entier : mots de 16 bits virgule fixe.
- Opérations arithmétiques rapides et flexibles : multiplication avec accumulation, décalage de bits et opérations arithmétiques et logiques réalisés en 1 cycle d'horloge.
- Extension de la dynamique du résultat de la multiplication et accumulation. Protection contre les débordements dans les accumulations répétées.
- Acquisition de 2 opérandes en 1 seul cycle d'horloge.
- Mémoire circulaire gérée par le hard du DSP.
- Bouclage et branchement sans suspendre l'exécution du programme. Le saut à la nouvelle adresse est calculé pendant l'exécution de l'instruction précédente.
- Le jeu d'instructions et la facilité à écrire les programmes.
- Mémoire cache interne 16×24 bits.

2 Architecture de l'ADSP-2100

L'ADSP-2100 est conçu selon une architecture dite de Harvard, c'est-à-dire que les bus d'adresses et de données (et par extension les mémoires programme et données) sont séparés. Sa partie calcul est formée de 3 unités en parallèle reliées par un bus R dit de Résultat. La partie contrôle comporte un séquenceur de programme et deux générateurs d'adresses. On note la présence d'une mémoire cache. La figure suivante représente cette architecture :

Tournez la page S.V.P.



R : bus résultat, interne à l'ADSP-2100.
PMA/D : bus d'adresses et de données de la mémoire programme.
DMA/D : bus d'adresses et de données de la mémoire données.

2.1 L'Unité Arithmétique et Logique (ALU)

- L'ALU effectue les opérations arithmétiques et logiques standards sur des mots de 16 bits : addition, soustraction, changement de signe, incrémentation, décrémentation, valeur absolue, ET logique, OU logique, OU exclusif, complément à 1 et exécute des divisions de nombres de 32 bits par des nombres de 16 bits. Dans tous les cas le résultat est un registre de 16 bits.

- Les opérandes sont 2 registres notés X et Y de 16 bits ou un des registres résultat du bus R.

- En plus du résultat sur 16 bits, un registre de statut de 6 bits (**ASTAT**) renseigne sur la nature du résultat. Les 6 bits de ce registre, du poids faible au poids fort, représentent : résultat nul (AZ), résultat négatif (AN), débordement (AV), retenue générée (AC), signe de l'entrée X (AS) et résultat d'une division (AQ).

- Les syntaxes des fonctions standards de l'ALU sont les suivantes :

$R = X + Y$	Addition de X et Y
$R = X + Y + CI$	Addition avec retenue
$R = X - Y$	Soustraction de Y à X
$R = X - Y + CI - 1$	Soustraction de Y à X avec retenue
$R = Y - X$	Soustraction de X à Y
$R = Y - X + CI - 1$	Soustraction de X à Y avec retenue
$R = -X$	Changement de signe de X
$R = -Y$	Changement de signe de Y
$R = Y + 1$	Incrémentation de Y
$R = Y - 1$	Décrémentation de Y
$R = PASS\ X$	$R = X$ et affecte ASTAT
$R = PASS\ Y$	$R = Y$ et affecte ASTAT
$R = ABS\ X$	Valeur absolue de X
$R = X\ AND\ Y$	ET logique de X et Y
$R = X\ OR\ Y$	OU logique de X et Y
$R = X\ XOR\ Y$	OU Exclusif de X et Y
$R = NOT\ X$	Complément à 1 de X
$R = NOT\ Y$	Complément à 1 de Y
$R = 0$	Mise à zéro du résultat

L'opérande X peut être l'un de ces registres : AX0, AX1, AR, MR0, MR1, MR2, SR0 ou SR1.

L'opérande Y : AY0, AY1 ou AF.

Le registre résultat R : AR ou AF.

Toutes ces opérations sont réalisées en 1 cycle d'horloge.

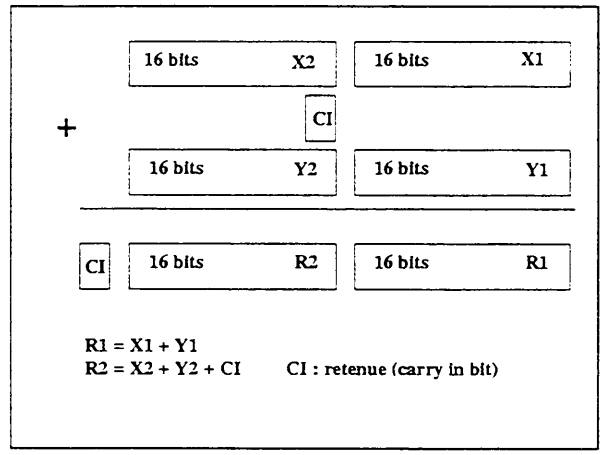
- Le registre résultat AR a la capacité de "saturation" permettant de le bloquer à la valeur maximale $(2^{15} - 1)$ ou minimale (-2^{15}) en cas de débordement. Ce mode est permis par un bit du registre **MSTAT.**

- L'ALU contient un second banc de registres permettant de faire une sauvegarde du contexte.

- L'ALU supporte les opérations en multiprécision grâce à la retenue générée automatiquement.

  L'exemple suivant montre l'addition de 2 nombres de 32 bits :

Tournez la page S.V.P.



- La division, exécutée en 16 cycles d'horloge, s'obtient de la façon suivante :

  - Le diviseur doit être stocké dans AX0, AX1 ou 1 des registres résultat.
  - Les poids forts du dividende signé doivent être dans AY1 ou AF, uniquement dans AF pour un dividende non signé.
  - Les poids faibles du dividende doivent être dans AY0.
  - A la fin de la division le quotient se trouve dans AY0.
  - En tout premier on exécute l'instruction DIVS qui calcule le signe du quotient. Si cette instruction n'est pas utilisée, le quotient est supposé positif.
  - Ensuite on exécute 15 fois l'instruction DIVQ qui calcule bit après bit le quotient. Dans le cas d'une division non signée cette instruction est exécutée 16 fois.
  - Le format du quotient se déduit des formats du diviseur et du dividende de la façon suivante :

$$\begin{array}{lcc} \text{Dividende} & \underbrace{bbbbb}_{NE \text{ bits}} . & \underbrace{bbbbbbbbbbbbbbbbbbbbbbbbbb}_{NF \text{ bits}} \\ \text{Diviseur} & \underbrace{bb}_{DE \text{ bits}} . & \underbrace{bbbbbbbbbbbbbb}_{DF \text{ bits}} \\ \text{Quotient} & \underbrace{bbbb}_{(NE-DE+1) \text{ bits}} . & \underbrace{bbbbbbbbbbbb}_{(NF-DF-1) \text{ bits}} \end{array}$$

    Un dépassement se produit si le résultat ne peut pas être représenté dans le format ci-dessus ou lorsque le diviseur est nul.

2.2 Le Multiplieur / Accumulateur (MAC)

- Le MAC réalise une multiplication, une multiplication avec addition ou une multiplication avec soustraction en 1 cycle d'horloge.

- Les opérandes sont 2 registres de 16 bits notés X et Y ou un des registres résultat du bus R.

- Le résultat est un registre de 40 bits $(32 + 8)$ pour permettre les accumulations successives. Le bit MV du registre ASTAT signale un dépassement lors de la multiplication.

- Les syntaxes des fonctions du MAC sont les suivantes :

$R = X * Y$	Multiplication de X et Y
$R = MR + X * Y$	Multiplie X et Y et l'ajoute au registre MR
$R = MR - X * Y$	Multiplie X et Y et soustrait le résultat de MR
$R = 0$	Mise à zéro du registre résultat

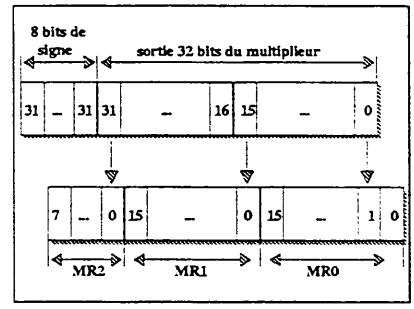
  L'opérande X peut être l'un des registres suivants : MX0, MX1, AR, MR0, MR1, MR2, SR0 ou SR1.

  L'opérande Y : MY0, MY1 ou MF.

  Le registre résultat R : MR ou MF.

- Pour faciliter les multiplications en multiprécision, le MAC accepte toutes combinaisons de nombres X et Y signés et non signés. Le format de chaque opérande est spécifié dans l'instruction.

- Le registre MR est composé de 3 registres : MR0 (bits 0 à 15), MR1 (bits 16 à 31) et MR2 (bits 32 à 39). Pour éviter la redondance de signe lors d'une multiplication signée (16 bits signés $\times$16 bits signés = 31 bits signés $\Longrightarrow$ 2 bits de signe sur 32 bits), le résultat est décalé de 1 bit vers la gauche (équivalent à une multiplication par 2). Le schéma suivant explicite cet ajustement :



Tournez la page S.V.P.

Quand le registre MR1 est chargé depuis le bus de données DMD, chaque bit de MR2 devient égal au bit de signe de MR1. MR2 est donc une extension du bit de signe de MR1. Pour charger le registre MR2 avec une autre valeur que le bit de signe de MR1, il faut charger MR2 après avoir chargé MR1.

- Une instruction de saturation permet de changer le contenu du registre MR par la valeur 32 bits la plus positive ou la plus négative, si le bit MV indiquant un dépassement, est positionné.

- Comme pour l'ALU, le MAC possède un second banc de registres.

2.3 L'unité de décalage (SHIFTER)

- Le bloc de décalage est l'unique moyen de réaliser une mise à l'échelle des données ou un formatage. Les fonctions de base sont : les décalages arithmétiques (extension du signe) et logiques, les normalisations et dénormalisations, les formatages en exposant et en bloc d'exposant.

- Le registre de décalage SR est un registre 32 bits accessible que par une entrée sur 16 bits. Cette entrée peut être placée sur les 16 bits poids faibles ou poids forts. Le registre résultat 32 bits SR est composé de 2 registres 16 bits SR1 (poids forts) et SR0 (poids faibles). L'opérande peut être l'un de ces registres : SI, AR, MR0, MR1, MR2, SR0 ou SR1.

- Le décalage immédiat permet de décaler une donnée d'un nombre de bits précisé dans l'instruction. Les 2 exemples suivants permettent de voir le principe du décalage immédiat :

  1. Donnée :

     $$AR = b\#1000000011111111;$$

     b# ⇒ format binaire,
     format décimal : AR = -32513,
     h# ⇒ format hexadécimal : AR = h#80FF,
     Décalage arithmétique (⇒ extension du signe) de 4 bits vers la droite :

     $$SR = ASHIFT\ AR\ BY - 4\ (HI);$$

     (HI) ⇒ Donnée dans SR1 (high),
     (LO) ⇒ Donnée dans SR0 (low),

Résultat :

$$SR \begin{cases} SR1 = b\#1111100000001111 \\ SR0 = b\#1111000000000000 \end{cases}$$

2. Donnée :

$$SI = b\#1000000011111111;$$

Décalage logique de 3 bits vers la gauche :

$$SR = LSHIFT\ SI\ BY\ 3\ (LO);$$

Résultat :

$$SR \begin{cases} SR1 = b\#0000000000000100 \\ SR0 = b\#0000011111111000 \end{cases}$$

- Le résultat du décalage peut subir un OU logique avec la valeur précédente du registre SR. L'exemple suivant montre cette possibilité :

  Affectation du registre SR :

$$\begin{aligned} SR0 &= b\#0000000000000000; \\ SR1 &= b\#1111111111111111; \end{aligned}$$

  Donnée :

$$SI = b\#1010101010101010;$$

  Décalage logique de 4 bits vers la droite suivi d'un OU logique :

$$SR = SR\ OR\ LSHIFT\ SI\ BY\ -4\ (HI);$$

  Résultat :

$$SR \begin{cases} SR1 = b\#1111111111111111 \\ SR0 = b\#1010000000000000 \end{cases}$$

- Au même titre que l'ALU et le MAC, le SHIFTER possède un second banc de registres.

Tournez la page S.V.P.

2.4 Le générateur d'adresses (DAG)

- Le DAG est composé de 2 générateurs d'adresses indépendants DAG1 et DAG2 permettant d'adresser simultanément les mémoires de données et programme. Le DAG1 ne peut adresser que la mémoire de données. Le DAG2 peut, par contre, adresser les 2 mémoires. Le DAG1 possède en plus l'option de réaliser un renversement de bits sur l'adresse de sortie (une valeur d'adresse d'entrée 10110 deviendra 01101 après renversement → "bit reverse" utilisé pour la FFT). Ce mode d'adressage est choisi grâce à un bit du registre MSTAT.

- Chacun des générateurs d'adresses possède 4 jeux de 3 registres de 14 bits : les registres d'index I, les registres de modification M et les registres de longueur L.

  I0 à I3 (DAG1) et I4 à I7 (DAG2) contiennent l'adresse courante de la mémoire à consulter.

  M0 à M3 (DAG1) et M4 à M7 (DAG2) contiennent la valeur du post-incrément du registre d'index.

  L0 à L3 (DAG1) et L4 à L7 (DAG2) permettent un adressage circulaire.

  La prochaine adresse est calculée automatiquement par :

  $$I \leftarrow (I + M) \text{ modulo } L$$

  Pour inhiber l'effet modulo il suffit de mettre L à zéro.

2.5 Le séquenceur de programme

- Le séquenceur de programme contrôle le déroulement du programme. En particulier il permet des bouclages sans cycle perdu, des branchements conditionnels ou inconditionnels en 1 seul cycle et gère les interruptions.

- Les adresses des instructions du programme sont gérées par le compteur programme PC de 14 bits. Pendant l'exécution de l'instruction le PC incrémente l'adresse de l'instruction en cours pour générer l'adresse suivante. Lors d'un appel à un sous-programme par l'instruction CALL ou par une interruption, la valeur du PC est sauvegardée dans la pile PC de 16 mots de 14 bits.

- Le séquenceur de programme contient 5 registres renseignant sur l'état du bloc arithmétique (**ASTAT**), sur la pile (**SSTAT**), sur les modes de

fonctionnement (**MSTAT**), sur le contrôleur d'interruptions (**ICNTL**), sur les masques d'interruptions (**IMASK**). La description de ces 5 registres est la suivante :

- **ASTAT** : Renseigne sur l'état arithmétique (accessible uniquement en lecture).

  bit 0 : AZ résultat nul de l'ALU
  bit 1 : AN résultat négatif de l'ALU
  bit 2 : AV dépassement dans l'ALU
  bit 3 : AC retenue générée dans l'ALU
  bit 4 : AS entrée X de l'ALU signée
  bit 5 : AQ quotient de l'ALU
  bit 6 : MV débordement dans le MAC
  bit 7 : SS entrée du SHIFTER signée

  Un bit à 1 correspond à vrai, un bit à 0 correspond à faux (logique positive).

  Exécuter l'instruction PASS X (ou PASS Y) de l'ALU positionne les bits AZ et AN.

  Les bits AZ, AN, AV, AC sont affectés par toutes les opérations de l'ALU sauf DIVS et DIVQ.

  Le bit AS est affecté par l'opération ABS de l'ALU.

  Le bit AQ est affecté par les opérations de division DIVS et DIVQ.

  Le bit MV est affecté par toutes les opérations du MAC sauf pour le mode saturation.

  Le bit SS est affecté par l'opération EXP du SHIFTER.

- **SSTAT** : Renseigne sur l'état des piles (accessible uniquement en lecture). La première pile (16×14 bits) sert à stocker l'adresse de l'instruction suivante du registre PC lors d'un saut à un sous-programme. La seconde (4×16 bits) stocke le registre d'état courant composé des registres ASTAT, MSTAT et IMASK, lors d'un saut à un sous-programme d'interruption. La troisième (4×14 bits) sauve la valeur du compteur de boucle CNTR. La quatrième (4×18 bits) sauve la dernière adresse et la condition de la boucle. Ces 2 dernières piles sont sollicitées lors de l'imbrication de plusieurs boucles (4 niveaux d'imbrication au maximum).

Tournez la page S.V.P.

bit 0 : pile PC vide
bit 1 : débordement de la pile PC
bit 2 : pile compteur vide
bit 3 : débordement de la pile compteur
bit 4 : pile d'état vide
bit 5 : dépassement de la pile d'état
bit 6 : pile de boucle vide
bit 7 : dépassement de la pile de boucle

Les bits sont affectés selon la logique positive. Un bit de débordement qui passe à 1 reste dans cet état quelles que soient les opérations futures. Seul un RESET du processeur peut remettre à zéro les bits de SSTAT.

– **MSTAT** : Définit les modes de fonctionnement.
bit 0 : choix du banc de registres (0 primaire, 1 secondaire)
bit 1 : renversement des bits d'adresse (DAG1)
bit 2 : mode "dépassement vérouillé" de l'ALU
bit 3 : mode saturation du registre AR

Le second banc de registres comprend tous les registres résultat et d'entrées de l'ALU, du MAC et du SHIFTER.

Le mode "dépassement verrouillé" permet de bloquer le bit AV à 1 après d'un dépassement dans l'ALU quelles que soient les opérations suivantes.

Le mode saturation du registre AR permet de saturer AR au maximum (h#7FFF) ou au minimum (h#8000) quand un dépassement survient.

– **IMASK** : Les demandes d'interruptions sont comparées (ET logique) au registre IMASK puis envoyées vers l'encodeur de priorité qui choisit la plus prioritaire. L'interruption $IRQ3$ étant la plus prioritaire, $IRQ0$ la moins prioritaire. Lorsque la sortie de l'encodeur de priorité devient active, il se produit un saut (JUMP) à l'adresse de la mémoire programme correspondant à l'interruption choisie. Les adresses sont h#0000 ($IRQ0$) à h#0003 ($IRQ3$). Cette adresse contient l'instruction de saut au sous-programme d'interruption correspondant.
bit 0 : permet IRQ0
bit 1 : permet IRQ1
bit 2 : permet IRQ2
bit 3 : permet IRQ3

Le bit à 1 permet l'interruption, le bit à 0 inhibe l'interruption.

Un RESET du processeur remet à zéro tous les bits de IMASK.

– **ICNTL** : Configure les modes de déclenchement et de prise en compte des interruptions.

bit 0 : mode de déclenchement de IRQ0
bit 1 : mode de déclenchement de IRQ1
bit 2 : mode de déclenchement de IRQ2
bit 3 : mode de déclenchement de IRQ3
bit 4 : mode de traitement des interruptions

Le mode de déclenchement détermine le déclenchement d'une interruption sur niveau, bit à zéro ou sur front, bit à 1.

Le mode de traitement permet l'imbrication ou la non imbrication des interruptions. Si le bit 4 est à zéro, tous les bits de IMASK sont forcés à zéro lorsqu'une interruption est prise en compte. La valeur précédente de IMASK est sauvegardée dans la pile d'état. Si le bit 4 est à 1, IMASK est positionné de manière à masquer les interruptions les moins prioritaires.

- Le séquenceur de programme permet l'exécution conditionnelle de certaines instructions. Ces instructions sont :

  – les sauts (JUMP),
  – l'arrêt du processeur (TRAP),
  – l'appel d'un sous-programme (CALL),
  – le retour de sous-programme (RTS et RTI),
  – les opérations du MAC et de l'ALU,
  – les opérations du SHIFTER autres que les décalages immédiats.

  La syntaxe d'une instruction conditionnelle est :

  $$IF\ condition\ instruction\ ;$$

  Les conditions sont :

EQ	résultat ALU nul
NE	résultat ALU non nul
LT	résultat ALU < 0
GE	résultat ALU ≥ 0
LE	résultat ALU ≤ 0
GT	résultat ALU > 0
AC	retenue ALU $= 1$
$NOT\ AC$	retenue ALU $= 0$

Tournez la page S.V.P.

AV	dépassement dans l'ALU
NOT AV	pas de dépassement dans l'ALU
MV	dépassement dans le MAC
NOT MV	pas de dépassement dans le MAC
NEG	entrée X négative
POS	entrée X positive
NOT CE	compteur de boucle CNTR $\neq 0$
TRUE	toujours vrai

Le résultat d'une condition dépend de l'affectation du registre d'état du bloc arithmétique ASTAT à la fin du cycle précédent.

- Le séquenceur de programme est aussi capable de gérer les boucles. L'instruction qui permet d'exécuter un paquet d'instructions jusqu'à ce qu'une condition devienne vraie, est :

$$\begin{array}{rl} & \textit{DO}\ \text{fin\_de\_boucle}\ \textit{UNTIL condition}\ ; \\ & \textit{instruction}\ 1\ ; \\ & \vdots \\ \text{fin\_de\_boucle}: & \textit{instruction}\ N\ ; \end{array}$$

Lorsque le nombre de passages dans la boucle est déterminé on utilise le compteur de boucle CNTR dont la décrémentation est gérée par le séquenceur de programme. L'exemple suivant montre l'exécution 10 fois d'une boucle :

$$\begin{array}{rl} & \textit{CNTR} = 10\ ; \\ & \textit{DO}\ \text{fin\_de\_boucle}\ \textit{UNTIL CE}\ ; \\ & \textit{instruction}\ 1\ ; \\ & \vdots \\ \text{fin\_de\_boucle}: & \textit{instruction}\ N\ ; \end{array}$$

- Grâce à la présence d'une mémoire interne au DSP, nommée mémoire cache, le séquenceur de programme peut optimiser le temps d'exécution de certaines instructions. Cette mémoire de 16 mots stocke au fur et à mesure les instructions lues de la mémoire programme. Elle constitue un historique des 16 dernières instructions. Dans le cas d'une boucle de moins de 16 instructions, le séquenceur de programme va chercher les instructions dans la mémoire cache dont l'accès est très rapide.

2.6 L'échange de données entre bus PMD et DMD

Cette fonction permet d'effectuer des transferts de données entre les mémoires de données et de programme. Comme la taille du bus programme PMD, donc

la taille des mots de la mémoire programme, est de 24 bits alors que celle du bus de données DMD n'est que de 16 bits, seuls les bits de poids forts du bus PMD sont transférés directement. Un registre intermédiaire PX contient les 8 bits de poids faibles. Ce registre peut être préalablement chargé lors d'un transfert du bus 16 bits vers le bus 24 bits.

2.7 Les interruptions

Comme son nom l'indique, l'interruption interrompt l'exécution d'un programme pour permettre soit l'exécution d'un sous-programme soit l'accès du PC hôte aux mémoires de données et de programme. Les interruptions peuvent être de 2 types : logicielles ou matérielles.

L'interruption logicielle : L'instruction $TRAP$ suspend l'exécution du programme. Le processeur signale au PC hôte l'exécution du $TRAP$ en activant sa ligne de sortie $TRAP$. En réponse le PC active la ligne $\overline{HALT}$ du DSP pour le mettre en attente et provoque la désactivation de la ligne $TRAP$. Dès que les échanges entre PC et mémoires données et programme sont terminés, le PC désactive la ligne $\overline{HALT}$ et le DSP reprend l'exécution du programme.

Les interruptions matérielles : Les quatre lignes d'interruptions $\overline{IRQ0}$ à $\overline{IRQ3}$ permettent l'interruption d'un programme, par des évènements extérieurs tels que des signaux provenant de compteurs. La prise en compte et le mode de traitement des interruptions dépendent des registres IMASK et ICNTL décrits au paragraphe 2.5.

Le processeur peut être interrompu par le PC hôte si ce dernier active la ligne $\overline{DMBR}$ ou $\overline{PMBR}$ (Data / Program Memory Bus Request) de demande de bus. En réponse le DSP active la ligne $\overline{DMBG}$ ou $\overline{PMBG}$ (Data / Program Memory Bus Granted) signalant que le bus est accordé.

3 Notice de programmation

Tous les programmeurs en assembleur vous diront que l'assembleur de l'ADSP-2100 est très simple. En effet il ressemble aux langages évolués tels que le Turbo Pascal ou le C.

Un programme n'est en général pas consentré dans un unique fichier. Il est composé de plusieurs modules dans lesquels on trouve le programme principal, des déclarations de constantes et de variables et des sous-programmes.

Tournez la page S.V.P.

Pour obtenir le programme exécutable par le DSP il est nécessaire de passer par les 2 étapes suivantes :

- compiler (avec un compilateur) tous les modules,
- lier (avec un linkeur) tous les modules compilés pour obtenir le fichier exécutable.

3.1 Directives de l'assembleur

Les directives commencent toutes par un point. Elles permettent entre autres de définir le début et la fin d'un module, les déclarations de variables et de constantes. Ces directives sont les suivantes :

.MODULE : Début d'un module assembleur.

.ENDMOD : Fin du module assembleur.

Exemple :

$.MODULE$ MON_PROGRAMME;
⋮
$.ENDMOD$;

.VAR : Déclaration des variables.

/DM : Variable de la mémoire de données.

/PM : Variable de la mémoire programme.

/CIRC : Tableau circulaire.

/ABS : Adresse de la variable.

Exemple :

$.VAR/DM/ABS = 0$ variable1, variable2, tab1[10];
{variable1 en mémoire de données à l'adresse 0}
{variable2 en mémoire de données à l'adresse 1}
{tab1 est un tableau de 10 éléments}
$.VAR/PM/CIRC$ tab2[5];
{tab2 est un tableau circulaire de 5 éléments}

.CONST : Déclaration des constantes.

Exemple :

$.CONST$ N=10, M=1024;

.INIT : Permet l'initialisation des variables lorsque le programme est chargé sur la carte DX2100.

Exemple :

```
.INIT   tab2 : -1, 0, 1, 4, -45;
```

.GLOBAL : Rend une variable utilisable dans d'autres modules.

.ENTRY : Déclaration d'un sous-programme.

.EXTERNAL : Permet l'utilisation de variables et de sous-programmes définis dans d'autres modules.

Exemple :

```
.MODULE   module1;
.CONST    N=1024;
.VAR/DM   x, y, tab1[N];
.GLOBAL   x, tab1;
.ENTRY    sousprog1;
sousprog1 :
     ⋮
RTS; {fin du sous-programme}
.ENDMOD;

.MODULE   module2;
.CONST    M=10;
.VAR/DM   a, b, tab2[M];
.EXTERNAL   x, tab1, sousprog1;
     ⋮
CALL sousprog1; {appel du sous-programme}
AR = DM(x); {utilisation de la variable x}
     ⋮
.ENDMOD;
```

.INCLUDE : Permet d'inclure des fichiers.

Exemple :

```
.INCLUDE   <const.h>;
```

où const.h est un fichier qui contient par exemple une déclaration de constantes.

Tournez la page S.V.P.

.PORT : Déclaration des adresses mémoires donnant accès aux périphériques.

Exemple :

.PORT CNTL_IRQ, CNTL_TMR, TIMER0, DXM2106_1;

La déclaration proprement dite des adresses des périphériques est faite dans le fichier architecture DX2100.ACH utilisé lors de l'édition de liens (linker).

.MACRO : Début d'une macro-instruction.

.ENDMACRO : Fin d'une macro-instruction.

Exemple :

```
.MACRO division(%0, %1);
divs %1, %0;
divq %0; divq %0; divq %0; divq %0;
divq %0; divq %0; divq %0; divq %0;
divq %0; divq %0; divq %0; divq %0;
divq %0; divq %0; divq %0;
.ENDMACRO;
```

Cette macro s'utilise dans un programme de la façon suivante :

```
    ⋮
AX1 = 5; {diviseur)
AY1 = 0; {poids forts du dividende}
AY0 = 103; {poids faibles du dividende}
division(AX1,AY1);
DM(resultat) = AY0;
    ⋮
```

3.2 Conventions de l'assembleur

Les nombres peuvent être écrits en décimal, en binaire, en hexadécimal ou en octal. La base est précisée juste avant le nombre :

d# ou rien pour un nombre en base 10,

b# pour un nombre en base 2,

h# pour un nombre en base 16,

o# pour un nombre en base 8.

Les commentaires sont entre accolades {...}.

3.3 Les 4 premières lignes d'un programme

Les 4 premières lignes du programme principal sont réservées aux instructions associées aux interruptions. En effet lorsque l'interruption $\overline{IRQx}$ est déclenchée, le séquenceur de programme vient exécuter l'instruction se trouvant à l'adresse h#000x. Cette instruction peut être soit un saut à un sous-programme d'interruption (JUMP), soit la fin de l'interruption (RTI).

3.4 Lecture et écriture des variables

Dans le cas d'une variable simple en mémoire de données l'accès se fait par :

DM(Nom_de_la_variable) = *registre*; pour une écriture,

registre = DM(Nom_de_la_variable); pour une lecture.

Pour des variables en mémoire programme ou pour des tableaux, l'accès doit se faire par les registres d'adressage des générateurs d'adresses DAG1 et DAG2. DAG1 n'adresse que la mémoire de données.

Pour mettre l'adresse d'une variable dans un registre d'adressage ou pour mettre l'adresse de début d'un tableau on écrit :

$I0 =$ ^Nom_de_la_variable; ou $I0 =$ ^Nom_du_tableau;

L'exemple qui suit montre l'accès séquentiel aux éléments d'un tableau en mémoire programme :

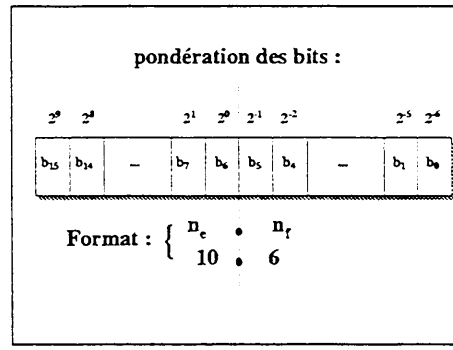
```
            ⋮
        .CONST   N=10;
        .VAR/PM  tab1[N];
            ⋮
        I4 = ^tab1;
        M5 = 1; {incrément}
        L4 = 0; {pour un tableau non circulaire}
        CNTR =N;
        DO boucle UNTIL CE;
boucle: PM(I4,M5) = CNTR;
        {I4 s'incrémente après chaque accès à la mémoire}
            ⋮
```

Tournez la page S.V.P.

3.5 Formats des nombres

L'ADSP-2100 ne traite que des nombres de 16 bits virgule fixe. C'est-à-dire qu'un nombre est représenté dans le DSP par n_f bits poids faibles pour sa partie fractionnaire et par n_e bits poids forts pour sa partie entière avec $n_e + n_f = 16$. On nomme ce format $n_e \bullet n_f$.

Exemple :



Par exemple le nombre $b\#0010011101101100$ (10092 en décimal) codé en $10 \bullet 6$ correspond à $\frac{10092}{2^6} = 157.6875$.