SIMULATION DE MICROPROCESSEURS UTILISANT LA TECHNIQUE DE MACRO-GENERATION

R. VERMOT-GAUCHY, D. AUBERT, C. GALAND

CER IBM, 06610 LA GAUDE , FRANCE

RESUME

Pour simuler un microprocesseur sur un ordinateur central, la voie classique utilise une méthode interprétative en analysant, instruction par instruction, le code machine. En raison de sa lenteur et de son manque de souplesse, ce type de simulateur est mal adapté au développement complet d'algorithmes complexes de traitement du signal.

Nous proposons une méthode qui permet d'accélérer par un facteur 1000 les temps de simulation. Cette méthode, qui est basée sur la technique de la génération par macro-instructions, consiste à compiler directement le langage symbolique du microprocesseur et à le traduire en instructions de l'ordinateur hôte.

Outre sa vitesse, ce type de simulateur permet d'accéder à toutes les facilités du système d'exploitation. On pourra, par exemple, utiliser des sous-programmes traduits par un compilateur. On simulera donc facilement en Fortran les opérations d'entrée-sortie, le monde extérieur, le mécanisme des interruptions, etc.

ABSTRACT

To simulate a microprocessor on a general purpose computer, the classical way uses an interpretative method by analysing instruction by instruction the machine code. This type of simulator is not well adapted to the development of complex signal processing algorithms.

This paper deals with an alternate method which consists in compiling directly the microprocessor symbolic assembler language by translating it into host assembler instructions. The development effort of this compiler is minimized with the help of macro generation techniques. This method allows to reduce the execution time by a ratio of 1000, when compared to the classical simulation method.

In addition, this method allows to take advantage of the operating system facilities. For example, one can integrate calls to Fortran sub-routines in the assembler micro-code, allowing to simulate easily functions like input/output, interrupts, etc.

1.0 INTRODUCTION

L'étude, le développement et la mise au point des algorithmes de traitement du signal sont souvent réalisés dans un langage de haut-niveau comme Fortran ou Pascal. Ensuite, en vue d'une utilisation sur un matériel donné il faut récrire les programmes dans le langage symbolique du microprocesseur de signal (MPS) choisi.

Si l'on dispose d'un ordinateur central, possédant un simulateur du microprocesseur le développement de ces programmes se simplifie considérablement. Au lieu des outils rudimentaires et peu performants, souvent associés au MPS, on utilise les nombreuses facilités offertes par l'ordinateur; en outre, on sépare les mises au point du matériel et du logiciel.

Il faut adopter une technique de simulation fournissant une très grande souplesse si l'on souhaite relier les programmes initiaux écrits en langage de haut niveau avec leur version en langage du MPS. D'autre part, elle doit être très performante dans le cas d'applications nécessitant des temps réels de l'ordre de 10 secondes et des puissances de calcul de l'ordre de 10 MIPS (Millions d'instructions par seconde). C'est le cas, par exemple, des algorithmes de traitement du signal de parole, des algorithmes modem, etc /1/.

Pour simuler un microprocesseur sur un ordinateur central, la voie classique utilise une méthode interprétative en analysant, instruction par instruction, le code machine. Pour être facilement transportable ce simulateur est souvent écrit dans un langage de haut niveau comme Fortran ou Pascal. On peut montrer que ce type de simulateur (en raison de sa lenteur et de son manque de souplesse) présente un certain nombre d'inconvénients et qu'il est mal adapté au problème posé.

Il faut en effet des simulateurs très rapides; on estime que leurs temps d'exécution doivent être divisés par un facteur 1000 par rapport aux simulateurs précédents.

La méthode proposée /2/ consiste à compiler directement le langage symbolique du microprocesseur et à le traduire en instructions de l'ordinateur hôte, en l'occurence le système S/370. Grâce à la technique de la génération par macro-instruction, le jeu d'instructions du MPS est considéré comme une simple extension de l'architecture de l'ordinateur hôte. Un programme écrit en langage MPS est donc traité par l'assembleur de l'IBM/370 qui fournit un fichier TEXT ayant la même structure que ceux produits par d'autres compilateurs tels que Fortran.

Par suite, ce type de simulateur permet d'accéder à toutes les facilités du système d'exploitation. On pourra, par exemple, utiliser des sous-programmes traduits par un compilateur comme Fortran. On simulera donc facilement en Fortran les opérations d'entrée-sortie, le monde extérieur, le mécanisme des interruptions, etc.

Il est même possible, de faire coexister plusieurs simulateurs de MPS différents, de construire des systèmes multiprocesseurs avec ou sans partage de la mémoire, d'ajouter des instructions fictives afin d'étudier les performances de nouvelles architectures.

Dans les trois premières parties de cet article, nous examinons successivement: les fonctions, l'organisation et l'utilisation de ce type de simulateur. La dernière partie concerne la structure interne et s'adresse plutôt aux informaticiens qui ont à développer ce genre de logiciel. La méthode que nous proposons s'applique, en effet à la plupart des microprocessseurs /3,4/. Cette dernière partie permet, en outre, de justifier tous les avantages que nous avons énumérés.

2.0 FONCTIONS OFFERTES PAR LE SIMULATEUR

Le développement de programmes en simulation implique un certain nombre d'opérations: écriture des programmes avec un éditeur, compilation, création et exécution du module de chargement, enfin examen des sorties standards comme les images de la mémoire de données (en décimal ou en hexadécimal) et la trace en symbolique (que l'on parcourt avec un éditeur pour rechercher une instruction ou une variable particulière).

Aussi, un simulateur performant doit fournir:

- des outils de mise au point assez élaborés pour éviter à l'utilisateur de travailler avec le code machine.

- des outils d'évaluation de performances donnant le nombre de cycles ou d'instructions, ainsi que l'encombrement des mémoires de programmes ou de données, pour l'algorithme étudié.

- une méthode permettant de modifier le "data flow" du microprocesseur, en particulier la taille des bus ou des registres.

- une interface très souple avec les langages de haut-niveau ayant la faculté de se relier à de nombreuses facilités fournies par l'ordinateur central. On donnera un exemple simple de sous-programmes Fortran montrant le principe de l'interface.

- des procédures d'exécution pour appeler le compilateur, l'éditeur de liens du système d'exploitation, puis le programme MPS à exécuter. Il est initialisé par un mini-éditeur de liens, qui prépare les tables nécessaires pour les calculs d'adresses des mémoires du MPS.

- des services pour les tracés de courbes, les analyses statistiques, les analyses de signaux dans le domaine du temps ou des fréquences (évaluation du rapport signal/bruit, transformée de Fourier), les manipulations de données entre Fortran et le MPS (chargement de la mémoire, comparaison, etc), les lectures et écritures de fichiers (contenant par exemple, des signaux de parole avec la possibilité d'écoute).

3.0 ORGANISATION DU SYSTEME DE DEVELOPPEMENT

La figure 1 montre les chemins possibles pour développer un algorithme de traitement du signal. La partie gauche représente le domaine relatif à la simulation (on y trouve les compilateurs, l'éditeur, l'assembleur et les programmes de services), tandis que la partie droite donne le chemin à emprunter pour une exécution en temps réel. Pour cela on effectue un transfert de fichier à travers un ordinateur personnel IBM PC qui pilote le microprocesseur.

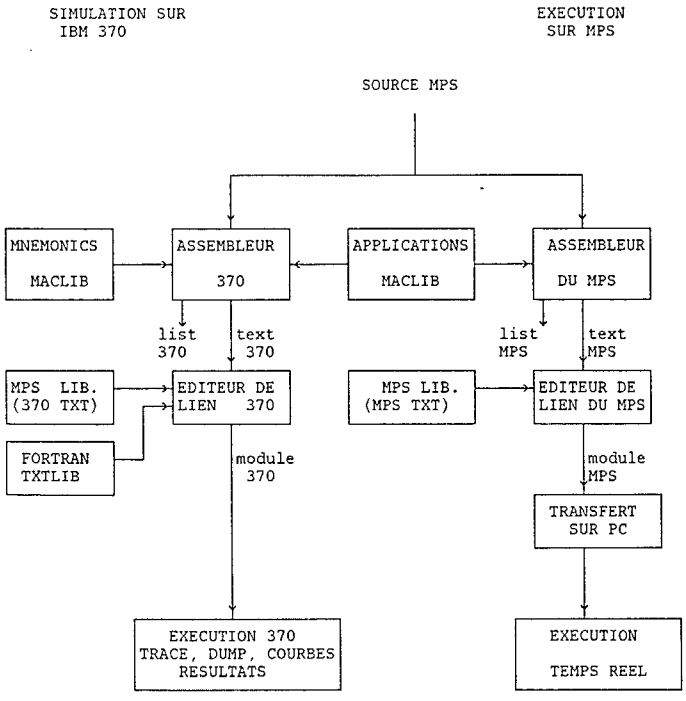

Figure 1: Organisation du simulateur et du système temps réel

Pour tester les algorithmes d'un vocodeur, l'utilisateur a, en outre, la possibilité de transférer des fichiers de parole entre l'IBM PC et le S/370 dans les deux sens.

4.0 REGLES D'UTILISATION DU SIMULATEUR

4.1 Programmes de services

Le simulateur doit fournir des outils de mise au point pour que l'utilisateur n'ait pas besoin de travailler avec le code machine. Les modifications de programme se feront au niveau du langage symbolique, de même l'utilisateur doit retrouver dans la trace les instructions telles qu'il les a écrites, en particulier le nom de ses variables.

Sur la figure 2 on remarque, à gauche deux nombres permettant de repérer facilement les instructions, suivis par le code symbolique et son étiquette entre parenthèses enfin, à droite, le contenu des principaux registres.

```
11 1 LDY 2           A=00000000 B=00000000 I=000 J=000 X=0000 Y=0020
11 2 LDX 1 (MONACO)  A=00000000 B=00000000 I=000 J=000 X=0010 Y=0020
11 3 LDY 4 ADD XY SA A=00300000 B=00000000 I=000 J=000 X=0010 Y=0040
11 4 BDZ CER OR XY   A=00300000 B=00000000 I=000 J=000 X=0010 Y=0040
11 5 SMAH 34         A=00300000 B=00000000 I=000 J=000 X=0010 Y=0040
11 6 LMX 34          A=00300000 B=00000000 I=000 J=000 X=0030 Y=0040
11 7 SMX 36 (CER)    A=00300000 B=00000000 I=000 J=000 X=0030 Y=0040
```

Figure 2: Exemple de trace

Ceci rappelle une sortie d'analyseur logique, mais cette trace en symbolique placée dans un fichier peut être analysée par des logiciels de manipulation de fichiers. Par exemple, comparaison de trace, balayage par un éditeur, permettant la recherche d'une instruction particulière (comme le stockage SMX du registre X dans la mémoire 36).

L'utilisateur désire également retrouver ses résultats sous une forme facilement exploitable. L'instruction DUMP fournit conditionnellement l'image de la mémoire de données (figure 3) en décimal ou hexadecimal à raison de 8 mots par ligne.

```
MAIN STORAGE: Z8
 0020   0000 0000 0030 0000   0030 0000 0000 0000
 0028   0000 0000 0000 0000   0000 0000 0000 0000
```

Figure 3: Image de la mémoire après exécution du programme précédent (36 = X'24')

Le simulateur peut aussi générer (avec l'ordre MAP) une table des symboles externes, trés utile pour l'analyse de la mémoire de données.

Le simulateur offre également la possibilité de modifier, dans une certaine mesure, le "data flow" du microprocesseur. Ainsi des ordres comme MPR ou MCR agissent sur le nombre de bits de certains registres.

Enfin, l'utilisateur souhaite des aides au développement permettant, par exemple, l'évaluation du nombre de cycles ou d'instructions, ainsi que celle de l'encombrement des mémoires de programmes ou de données.

L'évaluation de la complexité d'un algorithme est possible grâce à des ordres tels que CYCLE ou COUNT qui permettent de faire une statistique de l'utilisation de chaque instruction au cours de l'exécution du programme.

4.2 Interface avec les langages de haut-niveau

La souplesse de la méthode de simulation réside dans sa faculté de se relier à de nombreuses facilités fournies par l'ordinateur central, grâce à l'interface existant entre le simulateur et les compilateurs de langages de haut-niveau. Le principe est simple: la mémoire de données du MPS est représentée par une variable externe ZMSTOR. L'échange de données entre Fortran et MPS s'effectue par l'intermédiaire du commun ZMSTOR, mais également à partir d'arguments, comme on peut le voir dans l'exemple ci-après:

```
      SUBROUTINE HPRINT(NI)
      COMMON /ZMSTOR/RAM(1000)
      INTEGER *2 RAM,N1,N2,NI(2)
      N1=NI(1)+1
      N2=NI(2)+1
      WRITE(6,1) (RAM(I),I=N1,N2)
    1 FORMAT(8I9)
      RETURN
      END
```

Figure 4: Sous-programme d'impression de zone mémoire

Ce passage des données caractérise le mode dit "par valeur", Fortran et le micro partagent la mémoire. Un

autre mode (par adresse) permet d'échanger des informations sans utiliser la mémoire ZMSTOR.

Pour cela, le simulateur introduit des COMMONs supplémentaires comme ZLSTOR pour les registres du micro, mais aussi accepte tous les symboles commençant par les lettres ZX. La figure 5 illustre ce principe:

```
COMMON /ZXNICE/NICE
COMMON /ZMSTOR/RAM(1000)
INTEGER *2 NICE,RAM

NICE=100
RAM(NICE)=64
```

Figure 5: Adressage dynamique

Ainsi, on change dynamiquement les adresses; ici NICE correspond au symbole XNICE dans le langage du micro et correspond à l'adresse 99. Il faut se souvenir que l'adressage du micro et l'indexage Fortran commencent respectivement en zéro et en un.

Par suite, toutes les instructions qui utiliseront le symbole XNICE seront modifiées; par exemple, les registres Y et I contiendront respectivement les valeurs 64 et 100 après exécution des instructions:

```
LMY   XNICE-1   chargement d'un mot mémoire
LDI   XNICE     chargement d'une adresse
```

Des ordres CALL, FCALL permettent de passer d'un sous-programme à un autre, quelque soit le langage d'écriture (Fortran, PL/I ou langage d'assembleur du micro).

Enfin, les modules de service du simulateur, vus précédemment et appelés par les ordres TRACE ou DUMP, sont écrits en Fortran, ils utilisent la même interface et peuvent servir d'exemples. On les modifie facilement (moins de 20 instructions) pour introduire toute une famille de sous-programmes donnant, par exemple des traces conditionnelles sur le code instruction, le contenu d'un registre, l'adresse mémoire, etc.

4.3 Le mini-éditeur de liens

La première section de contrôle exécutée est le mini-éditeur de liens du simulateur, il prépare les tables nécessaires pour les calculs d'adresses entre sections de contrôle. Ce mini-éditeur de liens est produit par les macro-instructions INCLUDE et GEN.

Soit un programme SPR0 et deux sous-programmes SPR1 et SPR2 écrits en langage symbolique du microprocesseur, le mini-éditeur de liens est produit par les macro-instructions de la figure 5. Il peut être appelé par Fortran (CALL SIM) comme un sous-programme ordinaire. Les opérations se déroulent selon la séquence suivante:

```
Fortran
CALL SIM -----> SIM ----> SPR0---> SPR2 et SPR1

          INCL SPR0
          INCL SPR2,SPR1
          GEN  SPR0          point d'entrée
```

Figure 6: Sous-programme mini-éditeur

5.4 Procédures d'exécution

Ces procédures sont proposées pour appeler le compilateur, l'éditeur de liens, puis le programme MPS à exécuter. Elles dépendent essentiellement des besoins de l'utilisateur.

Un programme MPS est, en général, constitué d'un ensemble de sous-programmes. La commande de chargement du système d'exploitation relie les différents modules produits par Fortran, ou par l'assembleur du simulateur MPS.

5.0 STRUCTURE DU SIMULATEUR MPS

5.1 Principes de mise en oeuvre

A partir du programme écrit en langage symbolique du MPS, l'assembleur du S/370 fournit une liste qui ressemble beaucoup à celle du compilateur Fortran. Son examen permet de comprendre la structure du simulateur à condition, toutefois, de connaître le langage du S/370. Nous donnons ici les principes généraux permettant au spécialiste informaticien de mettre en oeuvre cette méthode. Nous considérerons dans la suite l'exemple du processeur de signal IBM /5/.

Pour obtenir les meilleures performances l'utilisation des registres généraux du S/370 doit être judicieuse.

Certains sont réservés aux liens entre sous-programmes avec les mêmes conventions que celles du système d'exploitation et obéissent de ce fait aux règles classiques du système. D'autres représentent ceux du microprocesseur, par exemple, X et Y bien qu'ils aient moins de 32 bits, comme on peut le voir sur la figure 2. D'autres enfin servent de registres de travail ou contiennent des pointeurs.

L'adressage des mémoires de données et d'instructions est un autre problème délicat. Les symboles du micro sont déclarés avec l'ordre S/370 EQU. On différencie éventuellement les données des instructions en leur affectant deux types différents, par exemple:

```
MONACO    EQU   54,,C'D'
CER       EQU   *,,C'I'
```

Si, pour les données, les mémoires du S/370 et du MPS se définissent respectivement au niveau de l'octet et du mot (2 octets), le calcul d'adresse demande alors 3 instructions S/370, il s'effectue en deux temps: chargement de l'adresse MPS dans un registre ZW, puis traduction en adresse S/370 sachant que le registre ZMS pointe le début de la mémoire de données (soit: ZMS + 2*ZW).

La figure 7 donne la traduction de quelques instructions; elles se terminent par l'appel d'un sous-programme ZNIF nécessaire pour traiter les problèmes dûs au pipe-line.

```
          LDX   5    donne:     LA   ZX,=AL2(5*16)

LMX  MONACO+2          LMY   XNICE          BDZ  CER,X

LH   ZW,=AL2(MONACO+2) L     ZW,=V(ZXNICE)  LTR  ZX,ZX
AR   ZW,ZW             LH    ZW,0(ZW)       BNZ  *+6
LH   ZX,0(ZW,ZMS)      AR    ZW,ZW          LA   ZBR,CER
BAL  ZN,ZNIF           LH    ZY,0(ZW,ZMS)   BAL  ZN,ZNIF
                       BAL   ZN,ZNIF
```

Figure 7: Modèle de traduction

L'adressage des instructions peut, dans la plupart des cas, se résoudre directement. Par exemple (fig 7), le branchement conditionnel à l'instruction étiquetée CER est traduit par un chargement conditionnel (BDZ: Branch on zéro) du registre ZBR qui est ensuite analysé par un sous-programme ZNIF. Ceci s'avère nécessaire quand l'instruction située après un branchement est exécutée, ce qui est le cas avec un dispositif de "pipe-line".

Notons, également, que le programmeur MPS a la possibilité de calculer ou d'incrémenter certaines adresses d'instructions. Dans ce cas, la correspondance avec celles du S/370 nécessite un mécanisme plus sophistiqué: les adresses MPS servent d'index pour consulter une table de pointeurs donnant celles équivalentes en S/370.

Enfin, signalons que pour garder le code symbolique, l'option trace introduit pour chaque instruction du MPS une instruction supplémentaire:

```
MVC      ZTAG(18),=CL18' BDZ CER,X '
```

Le dispositif de trace est conditionnel, il ne pénalise pas, au moment de l'exécution, les modules déjà mis au point. De mêmes, le simulateur n'introduit les différents compteurs d'évaluation qu'à la demande, avec les ordres CYCLE ou COUNT.

5.2 La bibliotheque de macro instructions

L'assembleur du système IBM/370 construit les programmes objets en utilisant une bibliothèque de macro-instructions. Cette bibliothèque contient les macros de description, entre autres celles:

- de l'interface entre modules (ex: FCALL,CALL,PLOT),
- du mini-éditeur de liens (ex: INCL, GEN, MAP),
- de la mise au point (ex: DUMP, TRACE),
- d'analyse des performances (CYCLE, COUNT),
- de changement du nombre de bits des registres (MCR, MPR),

ainsi que les macros de description des instructions du MPS et de son assembleur.

La plupart des macro-instructions sont écrites à l'aide de macro primitives ayant des fonctions bien spécifiques. Ainsi, le chargement d'un registre LMX doit traiter: les étiquettes (ZORG), les manipulations de données (ZLS), les ordres arithmétiques (ZARITH) et les ordres spéciaux de trace ou de test (ZNIF):

```
         MACRO
&LAB     LMX    &A,&B,&C,&D,&R=,&X=X
&LAB     ZORG   L&R.M&X,&A
         ZLS    L,&A,&R,&X
         ZARITH &B,&C,&D
         ZNIF
         MEND
```

Des paramètres ont été ajoutés par mot-clé à ces instructions pour qu'elles puissent servir de primitives à d'autres instructions MPS. Par exemple, LIMY le chargement du registre Y indexé par I utilise LMX où &R spécifie le registre index I et &X le registre Y:

```
         MACRO
&LAB     LIMY   &A,&B,&C,&D
&LAB     LMX    &A,&B,&C,&D,R=I,X=Y
         MEND
```

Le branchement inconditionnel BD est construit sur le le même modèle que LMX, il utilise des primitives fondamentales et sert de primitive pour la plupart des instructions de branchements de même format. Le paramètre &S spécifie la condition de branchement:

```
          MACRO
&LAB      BDZ      &A,&B,&C,&D
&LAB      BD       &A,&B,&C,&D,S=(Z,NZ)
          MEND
```

La bibliothèque contient aussi quelques macro-instructions d'intérêt général. Dans le monde réel, elles sont écrites en langage MPS, c'est le cas de MOVEIJ, qui, en fait, appelle un sous-programme MOVEX: le mouvement se fait mot par mot dans une boucle, &AA donne le nombre de mots.

```
          MACRO
          MOVEIJ   &AA
          LDX      &AA
          CALL     MOVEX
          MEND
```

Pour améliorer les performances, en simulation, MOVEIJ a été traduit en langage S/370: en 7 instructions S/370 (dont un MVC) le simulateur déplace 100 mots MPS alors que dans le monde réel, le programme MPS prend plus de 220 instructions.

De nombreuses macros utilitaires ont été ainsi transcrites en langage symbolique S/370 réduisant considérablement le temps d'exécution. Aussi, dans certains cas, le simulateur déroule moins d'instructions que le microprocesseur dans le monde réel. Il va s'en dire que ces traductions doivent être fidèles au bit près et que l'utilisateur, en changeant les bibliothèques de macros, a la possibilité d'employer la version qu'il désire.

6.0 PERFORMANCES DU SIMULATEUR

L'examen de la figure 7 permet d'évaluer les performances du simulateur sachant que le sous-programme ZNIF nécessite 5 instructions S/370 en dehors du mode trace et lorsque les macros CYCLE ou COUNT ne sont pas employées. Une analyse plus poussée montre qu'une instruction MPS simulée demande en moyenne:

- 10 instructions sur le système S/370.
- 16 octets de mémoire principale du S/370.

Pour l'occupation mémoire, il faut évidemment tenir compte des sous-programmes de service du simulateur.

Nous avons effectué des comparaisons de performances entre notre simulateur et celui, classique, qui consiste à utiliser le programme binaire du microprocesseur lui-même. Le gain de temps pour la simulation d'algorithmes complexes permet d'apprécier l'utilité de notre méthode: par exemple, un algorithme de compression numérique de la parole /1/ nécessitant une puissance de calcul de 5 MIPS demande, selon le cas, 7 secondes ou 3 heures, pour simuler une seconde de parole sur le même IBM/370.

Ceci nous a permis de mettre au point l'algorithme complet du vocodeur en intégrant tous les modules, y compris les routines d'interruption, en simulation. Aussi la mise en oeuvre en temps réel ne demanda-t-elle que quelques jours, la plupart des problèmes venant essentiellement du hardware.

7.0 CONCLUSION

Le problème posé au début de cette étude était le manque de souplesse et la lenteur des simulateurs classiques, qui ne permettent pas un développement rapide d'algorithmes complexes de traitement du signal.

Dans cet article, nous avons proposé et expérimenté une méthode qui permet d'accélérer par un facteur 1000 les temps de simulation par rapport aux simulateurs classiques de processeurs.

Outre sa vitesse, ce type de simulateur permet d'accéder à toutes les facilités du système d'exploitation. On a pu par exemple simuler tout l'environnement de l'algorithme de traitement du signal en utilisant des sous-programmes Fortran. Ceci a permis d'accélérer et de simplifier le développement, tout en soumettant l'algorithme à des contraintes difficilement reproductibles en temps réel.

BIBLIOGRAPHIE

/1/ C.Galand, C.Couturier, G.Platel, R.Vermot-Gauchy, "Voice exited predictive coder VEPC implementation on high performance signal processor", IBM Journal of Research and Development, Vol 29, No 2, pp 147- 157, March 1985.

/2/ Vermot-Gauchy R., Simulation de microprocesseur en utilisant la technique de la macro génération,1984 documention interne IBM.

/3/ Aubert D., Etude, mise en oeuvre et tests d'un simulateur de processeur de traitement du signal thèse de 3ème cycle, Nice, 1987

/4/ Vermot-Gauchy R., Utilisation du macro-traitement dans différents domaines du logiciel (application à un système type S/370), en préparation: thèse PH. D., Nice, 1987.

/5/ Ungerboeck G. Maiwald D. Kaeser H.P. Chevillat P.R. Beraud J.P.,"Architecture of a Digital Signal Processor",IBM Journal of Research and Development, Vol.29, No.2, pp.132-139, March 1985.