UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

JULHO DE 1981

IMPLEMENTAÇÃO DE ALGORITMOS DE CONTROLE ADAPTATIVO EM MICROPROCESSADORES

por : Roberto de Alencar Lotufo

Orientador: Prof.Dr. Luis Gimeno Latre

Tese de Mestrado apresentada à Faculdade de Engenharia FEC-UNICAMP.

Apoiada pela FAPESP sob processo 78/1174.

AGRADECIMENTOS

Agradeço de forma especial ao meu orientador Prof. Dr. Luis Gimeno Latre pela ajuda no desenvolvimento desse trabalho, ao Prof. Dr. Márcio Luis Andrade Netto pelas discussões elucidativas, ao colega Rubens de Campos Machado pela cooperação nos testes e simulações realizadas e ao Márcio Luis Varani pela montagem dos circuitos implementados.

Finalmente, um agradecimento a todos que de alguma forma colaboraram para o êxito desse trabalho.

À meus pais, Nelson e Iracy,
minha esposa, Valéria,
meu filho, Rafael,
meu afilhado, Thiago.

RESUMO

Este trabalho apresenta uma metodologia para implementação de controladores adaptativos auto-ajustáveis em microprocessadores. O controlador auto-ajustável se presta ao controle de sistemas com parâmetros constantes (ou que variem lentamente), mas desconhecidos. No sistema de desenvolvimento utilizado para essa implementação fazem parte, o microcomputador baseado no Intel 8086, o computador PDP-DEC-10 e o computador analógico EAI-680. O desenvolvimento de "software" básico para o microcomputador é feito no PDP-10 usando-se uma linguagem de alto nível e um montador cruzado generalizado. A linguagem de alto nível desenvolvida é implementada usando-se o processador de macros do MACRO-10. O desempenho de dois exemplos simulados digital e analogicamente são apresentados, ilustrando a implementação e comprovando a viabilidade de controladores auto-ajustáveis baseado em microprocessadores.

ÍNDICE

1.1. INTRODUÇÃO

Com a crescente evolução da tecnologia, a utilização de computadores digitais em controle de processos tem se expandido rapidamente nos últimos dez anos. Mais recentemente com o aparecimento dos microprocessadores, houve uma explosão no número de processos em que esses componentes podem ser aplicados, devido a sua grande versatilidade para serem configurados às diversas aplicações. Deste modo houve um aumento também explosivo na demanda de recursos humanos e tópicos de pesquisa neste campo multidisciplinar que envolve as áreas de controle, "software" e "hardware" tanto em técnicas digitais como analógicas.

A grande diferença entre as realizações convencionais e as modernas técnicas em computadores digitais é a existência de programas, os quais constituem um dos elementos principais no sistema de controle. Como esses programas estão se tornando maiores e complexos envolvendo a participação fundamental de pessoas especializadas, a relação de custos "software/hardware" está crescendo de maneira significativa, incrementado ainda pelo custo decrescente da tecnologia de microeletrônica.

Sistemas de controle de processos com microprocessadores são, na grande maioria dedicados, contendo apenas os programas que realizam o controle de um determinado processo. Para a elaboração desses programas usam-se os chamados sistemas de desenvolvimento. O "software" depurado é transportado para o sistema definitivo ou para um protótipo de testes finais.

Os sistemas de desenvolvimento se dividem em duas categorias: residentes e cruzados |1| . Ferramentas de desenvolvimento cruzado são programas que rodam em outro computador em vez do microcomputador para qual o "software" está sendo desenvolvido. Ferramentas de desenvolvimento residente são aquelas que rodam no próprio microprocessador objetivo.

A técnica adotada neste trabalho é a do desenvolvimento cruzado. Dentre as razões que influenciaram nesta decisão podemos citar:

- Disponibilidade do computador PDP-10 através de terminal em tempo compartilhado.

- Existência de trabalhos anteriores, envolvendo montadores |2| e simuladores |3| .

- Custo inicial alto para a opção de se usar desenvolvimento residente.

Devido ao aparecimento constante de novos microprocessadores no mercado, existe o problema sempre presente da necessidade da disponibilidade de novas ferramentas de "software". Seus custos, quer sejam por desenvolvimento próprio ou por aquisição, podem ser altos. Deste modo os trabalhos realizados na implantação do sistema de desenvolvimento cruzado procuram minimizar os custos necessários para a sua adequação a novos microcomputadores. A solução usada consiste em projetar ferramentas de "software" que possam ser generalizáveis, isto é, sem muito esforço serem ampliadas para atenderem a novos microprocessadores.

O desenvolvimento de "software" básico para microcomputadores é tratado no capítulo 2. As ferramentas nele descritas consistem de um tradutor generalizado, baseado em processador de macros |4|, que permite a implementação de algoritmos de controle adaptativo usando-se uma linguagem de alto nível, um montador generalizado, e um conjunto de subrotinas responsáveis para o tratamento dos cálculos em ponto flutuante necessários aos algoritmos de controle.

O tradutor implementado utiliza a técnica baseada em expansões de macros descrita detalhadamente na seção 2.2. Para que esse tradutor seja generalizável, utiliza-se tradução via código intermediário |5|comum a diversos microprocessadores.

A linguagem de alto nível desenvolvida é baseada principalmente na linguagem Fortran devido aos algoritmos de controle adaptativo já estarem testados nesta linguagem.

O algoritmo de controle implementado neste trabalho pertence à classe de controladores adaptativos |6| . Esse tipo de controle é interessante quando aplicado a processos que possuem uma incerteza associada ao modelo do sistema. Como exemplos de razões dessa incerteza pode-se citar as mudanças imprevisíveis no meio ambiente que afetam o desempenho do sistema, envelhecimento de certos componentes ou mesmo mudanças no ponto de funcionamento inviabilizando o modelo utilizado. Em todos esses casos torna-se necessária uma adaptação do parâmetros

do controlador para atuar devidamente no sistema sob as novas condições. O problema de controle adaptativo envolve necessariamente uma estimação "on-line" das características variáveis do processo.

O controlador auto-ajustável ("self-tuning") [7], escolhido para implementação em microprocessador, é o controlador adaptativo mais promissor hoje em dia do ponto de vista da utilização em aplicação industriais. Descrevem-se na literatura diversas aplicações na fabricação de papel, controle de reatores químicos, pilotos automáticos para navios, etc. [8-10].

O algoritmo utilizado foi desenvolvido por [11] e atende a processos com múltiplas entradas e múltiplas saídas. Na seção 3.2. ele é apresentado de forma resumida.

Na seção 3.3. são descritos os componentes "hardware" que fazem parte do sistema de desenvolvimento de algoritmos de controle adaptativo utilizado para a preparação, depuração e testes de programas aplicativos. O sistema envolve o computador PDP-10 atuando como hospedeiro, um microcomputador baseado no 8086 da Intel e o computador analógico EAI-680. Com essa configuração é possível a realização de simulações digitais e analógicas do processo a controlar. Na seção 3.3.2. é apresentada a configuração do microcomputador e na seção 3.3.3 descreve-se a interface implementada para a comunicação entre o microcomputador e o computador analógico.

A seguir, na seção 3.4 é indicado o procedimento de programação do algoritmo auto-ajustável utilizando a linguagem de alto nível desenvolvida no capítulo 2. Finalizando, dois exemplos são simulados: o primeiro digitalmente, mostrando que a implementação do algorítmo está correta e o segundo utilizando-se o computador analógico para simular o processo a ser controlado, ilustrando assim o comportamento do algoritmo em situações de tempo real.

CAPÍTULO 2

"SOFTWARE" BÁSICO

2.1. INTRODUÇÃO

O custo elevado da programação dos microprocessadores é, em boa parte, devido à utilização de linguagens de baixo nível. Faz-se necessário, portanto, a existência de uma linguagem de alto nível, de fácil tradução para a máquina objeto, para a redução deste custo. Nota-se entretanto que o processo de tradução por interpretação não é adequado para controle em tempo real, o que exclui de consideração a utilização de linguagens do tipo BASIC. Por outro lado, a implantação de compiladores específicos para uma dada linguagem e uma dada máquina inviabiliza o aproveitamento dos constantes avanços tecnológicos na área dos microprocessadores. A abordagem adotada neste trabalho procura contornar esse problema pela adoção de sistemas de desenvolvimento de "software" cruzado que sejam generalizáveis para várias máquinas e linguagens. Esses sistemas compõem-se de um tradutor generalizado baseado no processador de macros do MACRO-10, que é um macro-montador para o processador do computador PDP-DEC-10, e de um montador cruzado generalizado programado em FORTRAN.

Neste capítulo é descrito o processo de tradução da linguagem específica desenvolvida para implementação de algoritmos de controle adaptativo em linguagem simbólica do microcomputador 8086, escolhido para esse trabalho, assim como é mostrada a organização desse processo de forma a permitir a sua generalização para tradução de outras linguagens ou geração de código para outras máquinas. A seguir é descrita a geração do código de máquina pelo montador cruzado generalizado e o seu carregamento no microcomputador em questão. Por último é mostrado o pacote de rotinas aritméticas em ponto flutuante implementado no 8086.

2.2. TRADUÇÃO E GERAÇÃO DE CÓDIGO INTERMEDIÁRIO

2.2.1. Introdução

É bem conhecida, em engenharia de "software", a necessidade de se programar em uma linguagem de alto-nível dada a redução do esforço de programação, a redução do número de erros, a portabilidade de programas, etc.. Na ausência de um sistema de desenvolvimento de "software" para o microprocessador Intel 8086 dispondo de uma linguagem de alto-nível, optou-se pelo desenvolvimento de uma linguagem orientada para o tipo de algoritmos utilizados em con

trole de processos. As condições definidas à priori para tal linguagem foram as seguintes:

- simplicidade do compilador;
- pode ser implementada rapidamente por programadores não peritos em teoria de compilação;
- ser parecida com o Fortran, devido à quantidade de programas já desenvolvidos para controle adaptativo, no laboratório;
- possuir tratamento de números em ponto flutuante;
- ter correspondência clara entre código fonte e objeto;
- possuir tradução generalizável para vários microprocessadores.

Diversas técnicas de se construir um compilador são conhecidas. A mais usada consiste em escrever completamente um programa compilador numa linguagem de alto nível apropriada, ou mesmo em linguagem simbólica. Essa técnica exige um esforço considerável de programação, além da exigência de conhecimentos de teoria de compilação. Um outro método consiste em usar um compilador de compiladores, isto é, um programa que aceita a descrição sintática e semântica da linguagem de programação a desenvolver e produz como saída um programa compilador para essa linguagem. Esse método exige menos trabalho por parte do programador, porém requer a disponibilidade de um bom compilador de compiladores. Usando-se um processador de macros de uso geral, [4] mostrou que se obtém várias das características de um compilador de compiladores. Essa técnica consiste em se definir um conjunto de macros cujos nomes identificam os comandos da linguagem. Quando um comando é encontrado, sua tradução é feita pela expansão final da macro a ele associada.

Os processadores de macros foram desenvolvidos primeiramente para possibilitar facilidades na substituição de textos na programação em linguagem de montagem e são chamados macro-montadores. Posteriormente diversos processadores de macros diferentes foram desenvolvidos. Uns com a intenção de serem específicos ao problema e outros de uso geral, mas todos constituindo um sistema independente. Dentre eles podemos citar: TRAC, desenvolvido em 1965 por [12], GPM, projetado por [13] também em 1965. Dois anos após [14,15] implementou ML/I e em 1969, [16] desenvolveu STAGE 2 para a construção de um sistema móvel de programação. Em 1972, no ITA, [17] desenvolveu um processador de macros baseado no GPM .

A maioria desses desenvolvimentos tinham como objetivo a produção de "software" básico, pois na época não se dispunha de uma linguagem de programação de alto-nível independente da máquina que fosse eficiente na programação desse "software" básico.

Recentemente, |18| utilizaram a técnica de tradução por processadores de macros, porém com algumas modificações. No lugar de usarem um processador de macros de uso geral, como na maioria dos trabalhos anteriores, aproveitaram o processador de macros específico de um macro-montador. Na sua publicação, é mostrado o uso do macro-montador como macro-montador cruzado de diversos microprocessadores e a partir desses macros - montadores cruzados desenvolveram um sistema de tradução de uma linguagem de alto-nível para código de máquina de diversos microcomputadores.

A técnica utilizada aqui e descrita neste capítulo, foi baseada principalmente no trabalho |18| . Devido a disponibilidade de um montador cruzado generalizado para diversos microprocessadores desenvolvido por |2| ,em 1979, o processador de macros do macro-montador MACRO-lo disponível é usado na tradução da linguagem desenvolvida em linguagem de montagem do microcomputador desejado. O processo completo de tradução é portanto cruzado e feito em duas etapas como mostra a figura 2.1. Primeiro traduz a linguagem de alto nível em linguagem simbólica do microprocessador desejado com o uso de um conjunto de macros do MACRO-10 e posteriormente, gera-se o código de máquina a partir da linguagem simbólica, com o uso do montador cruzado generalizado.

A tradução da linguagem desenvolvida em linguagem simbólica do microprocessador 8086 é feita por tres conjuntos de macros distribuídos de forma a possibilitar um sistema tradutor modular e flexível,podendo ser generalizável para a geração de código de outros microprocessadores, como também ser utilizado na tradução de outra linguagem, sem grandes esforços de implementação. Os tres conjuntos de macros são: macros que definem a linguagem,macros que geram código intermediário e macros que geram linguagem simbólica de cada máquina específica. Na figura 2.2 esses conjuntos podem ser visualizados.

Além desses tres conjuntos de macros, existe um quarto responsável pela extensão do processador de macros do MACRO-10 em operações básicas úteis no processo de compilação.

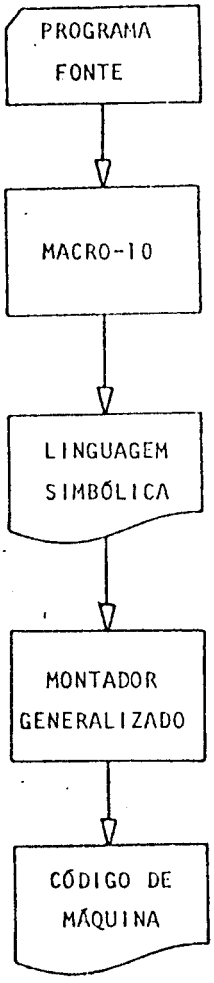


FIG 2.1 - Processo Completo de Tradução

Nesta seção são descritos o processador de macros do MACRO-10, o processo de tradução utilizando esse processador de macros, os comandos da linguagem desenvolvida usada na implementação dos algoritmos de controle de processos, a formação modular dos arquivos de macros que constituem o compilador, o código intermediário usado na compilação, e por último são apresentados os auxílios à depuração no processo de tradução.

2.2.2. Descrição do Processador de Macros do MACRO-10

A técnica de compilação usada neste trabalho utiliza como ferramenta um processador de macros. Devido a indisponibilidade de um processador de uso geral próprio para compilação, foi usado o processador de macros do MACRO-10.

O MACRO-10 é um macro-montador destinado a montar código de máquina para o computador PDP-DEC-10 a partir da sua linguagem simbólica.

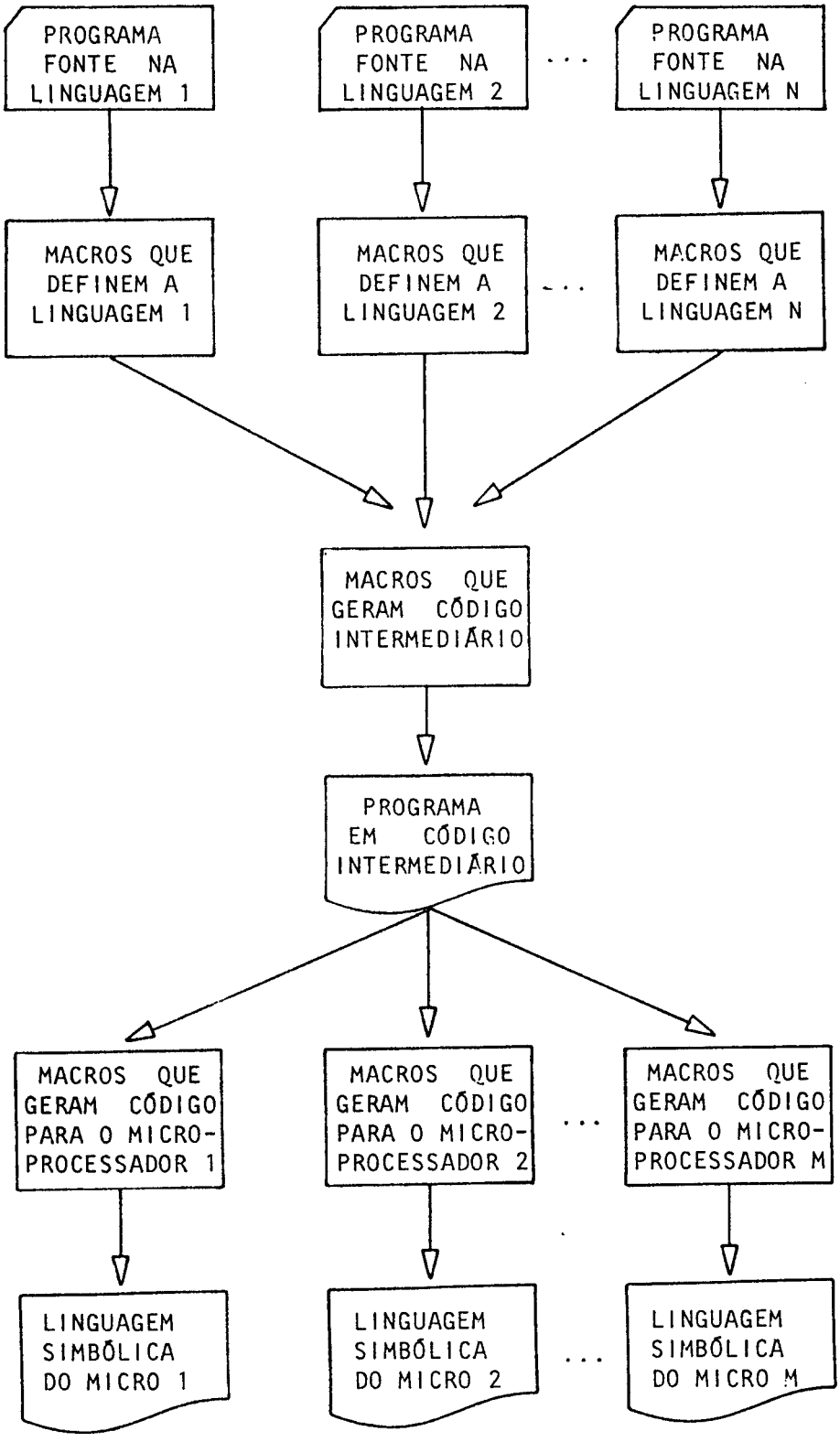


FIG 2.2 - Estrutura Modular das Macros que Constituem o Tradutor

- Repetição Indefinida: As pseudo-operações IRP e IRPC possibilitam uma maneira eficiente de se repetir toda ou uma parte do corpo de uma macro. A cada repetição é substituído um sub-argumento do argumento do IRP ou com cada caracter do argumento do IRPC no corpo da macro assinalado pelas pseudos IRP e IRPC. Desse modo, o número de repetições dependerá dos argumentos reais de uma dada chamada. Esses comandos são utilizados nas operações feitas com um número de operandos variável, definido durante a compilação, como por exemplo os comandos BYTE, WORD, FLOAT descritos no item 2.2.3 deste capítulo.

Além das pseudo-operações próprias do processador de macros do MACRO-10, outras aqui descritas são também utilizadas para a tradução da linguagem.

- pseudo-operações condicionais : Permitem testar se um dado operando é numérico ou simbólico, se dois operandos são cadeias iguais de caracteres, se um argumento é uma expressão com valor maior, menor ou igual a zero, se um operando existe ou não. Todos esses testes são úteis para a escolha do código a ser gerado que depende fortemente dos tipos de operandos das macros.

- pseudo-operações de referência a arquivos externos: O par de pseudos UNIVERSAL e SEARCH é usado para que num dado programa se possa referenciar, pelo comando SEARCH, definições feitas em outros arquivos identificados pelo UNIVERSAL. Dessa forma é possível a divisão do conjunto de macros que formam o compilador em quatro arquivos independentes caracterizados como UNIVERSAL. O programa fonte referencia esses arquivos pelos SEARCH's escolhidos para uma dada linguagem e um dado microprocessador objetivo.

- pseudo-operações de controle da listagem do programa: XLIST e LALL servem respectivamente para inibir e listar o programa e todas as expansões de macros chamadas. Essas instruções são usadas para possibilitar a depuração das macros que formam o tradutor.

- pseudo-operação de saída de texto no aquivo binário de saída do MACRO-10: A pseudo .TEXT imprime no arquivo binário o texto indicado pe

Nesta seção é descrita a parte do MACRO-10 utilizada como ferramenta na tradução da linguagem: o processador de macros e algumas pseudo-operações. Maiores informações sobre o MACRO-10 poderão ser obtidas em [19] .

A definição de uma macro é composta de uma parte de declaração onde são apresentados seu nome e seus parâmetros formais optativos, e de um corpo composto de um texto. A chamada de uma macro já definida é feita invocando-se seu nome, seguido dos parâmetros reais. Como resultado da chamada, o corpo da macro é expandido, formando assim um texto com as ocorrências dos parâmetros formais substituídas pelos parâmetros reais dessa chamada.

Os comandos do processador de macros do MACRO-10 utilizados para o tratamento das macros que definem a linguagem de alto-nível desenvolvida são:

- Definição de Macro: Para se definir ou redefinir uma macro, usa-se a pseudo-operação DEFINE seguida do nome da macro, seus argumentos formais opcionais separados por vírgula e dentro de parêntesis e por último o corpo da macro iniciado pelo caracter < e terminado por >.

- Chamada de Macro: Chama-se uma macro, colocando-se seu nome seguido dos argumentos reais (quando houver) no programa. Os argumentos são separados por vírgulas somente se estas não estiverem dentro dos sinais < e >. Pode-se também passar o valor numérico ou valor ASCII do parâmetro real da chamada, precedendo-o pelos caracteres \ e \" respectivamente.

- Concatenação de Argumentos: Se se insere um apóstrofo imediatamente antes ou depois de um argumento formal do corpo de uma macro, o texto gerado por uma chamada de macro não conterá o apóstrofo e assim o parâmetro real substituído ficará concatenado a qualquer outro caracter vizinho do apóstrofo. Essa facilidade é usada por exemplo na formação de rótulos e montagem da linguagem simbólica do microprocessador.

lo seu argumento. Esse comando é responsável pela geração da linguagem simbólica do microprocessador desejado.

2.2.3. Descrição dos Comandos da Linguagem Desenvolvida

A linguagem utilizada neste trabalho é uma linguagem adaptada para programação de algoritmos de controle, pertencendo portanto à categoria de linguagens orientadas ao problema ("Problem Oriented Languages").

Utiliza tres tipos de variáveis: duas inteiras, "byte" - de 8 bits e "word" - de 16 bits e uma em ponto flutuante, necessária aos cálculos aritméticos com números reais. Essas variáveis podem ser estruturadas na forma de matrizes. Todo e qualquer tipo de variável referenciada por um programa deve ser antes declarada, podendo no entanto existir variáveis globais declaradas em outros programas.

Os seus comandos básicos são: transferência de dados, iteração, desvios e comandos condicionais. Permite declarações e chamadas de subrotinas, sendo a passagem de parâmetros feita por referência. Os comandos de entrada/saída podem ser via TTY, no envio de mensagens ou dados e busca de informações do operador, e via processo, através de macros especiais que se comunicam com os conversores analógicos-digitais e digitais-analógicos. Possui ainda um comando que busca endereços de subrotinas básicas contidas numa biblioteca de programas. E por fim, permite declarações de interrupções.

Um resumo dos comandos da linguagem é apresentado na tabela 2.1.

Para calcular expressões, buscar elementos matriciais, dimensionar variáveis e dar entrada/saída pelos conversores A/D e D/A, usa-se um conjunto de macros, mostradas na tabela 2.2. Essas macros são utilizadas como operandos dos comandos da linguagem.

Como ilustração, a tabela 2.3 contém a comparação do comando LET da linguagem desenvolvida e da instrução de atribuição do FORTRAN.

COMANDO	DESCRIÇÃO
BYTE < lista de nomes > WORD < lista de nomes > FLOAT < lista de nomes >	Reserva memória e faz referência na tabela de simbolos ao tipo de variavel usada: BYTE : 8 bits WORD : 16 bits FLOAT : 24 bits
BEGIN	Início do Programa
BIBLOF	Busca tabela de símbolos na biblioteca
COMPIL	Fim do Programa
LET variável, expressão	Transferência de Dados
FOR variável, expressão, expressão < comandos >	Iteração
IF condição, expressão , < comandos >	Comando condicional
LABEL número	Declaração de rótulos
GOTO número	Desvio do programa para o rótulo especificado
SUBROUTINE nome, < parâmetros >	Declaração de subrotina
RETURN	Retorno de subrotina
CALL nome, < parâmetros >	Chamada de subrotina
INTERR	Declaração de interrupção
RETINT	Retorno de interrupção
HALT	Parada do programa
BREARPOINT	Retorno ao monitor
TYPEF < lista de variáveis >	Impressão de variáveis no TTY
READF < lista de variáveis >	Leitura de variáveis pelo TTY
PRINT < sequência de caracteres >	Impressão de mensagens ASCII no TTY

TABELA 2.1 : Macros que Definem os Comandos da Linguagem

NOME	DESCRIÇÃO
DIM (nome, número 1, número 2)	Dimensiona a variável "nome" com "número 1" linhas e "número 2" colunas
MAT (nome,expressão 1, expressão 2)	Acessa o elemento da matriz "nome"
CDA (número)	Dá saída analógica via canal "número" do conversor D/A
CAD (número)	Dá entrada analógica via canal "número" do conversor A/D
OPER (expressão, tipo, expressão)	Calcula uma expressão binária com o operador "tipo".

onde:

variável - pode ser um número, um símbolo ou um elemento matricial

expressão - pode ser uma variável ou uma expressão calculada pelo OPER.

TABELA 2.2 : Macros Auxiliares que são Operandos de outras Macros

Linguagem Desenvolvida	Fortran
LET A,B	A = B
LET A,OPER(B,ADD,C)	A = B+C
LET A,MAT(D,I)	A = D(I)
LET A,MAT(D,OPER(I,ADD,I))	A = D(I+I)
LET MAT(E,1,4),5	E(1,4) = 5

TABELA 2.3 : Comparação entre o comando de Transferência de Dados da Linguagem Desenvolvida e do Fortran

2.2.4. Definição Modular das Macros que Constituem o Compilador

Conforme descrito no item 2.2.1., existem tres arquivos de definições de macros responsáveis pelo processo de tradução do programa fonte em código intermediário e por sua vez em linguagem simbólica, que podem ser visualizados na figura 2.2, e um arquivo auxiliar que capacita o processador de macros a tratar os problemas referentes ao processo de compilação.

1º ARQUIVO : Macros que definem a linguagem de programação

Neste arquivo são definidas todas as macros que possam aparecer no programa fonte. A esse nível da tradução, existem operandos de macros que são chamadas de novas macros. Esses operandos são testados e caso sejam nomes de outras ou da mesma macro, elas são expandidas.

Como exemplo, o comando de atribuição LET é feito através da seguinte definição da macro de nome LET pertencente a esse arquivo.

```
DEFINE  LET  (VAR1, VAR2)
     <  TESTE  VAR2
        IFE  FLAG, <  $LAE  VAR2 >
        IFN  FLAG,    LADO = 1
                      VAR2       >
        TESTE  VAR1
        IFE  FLAG, <  $SAE  VAR1 >
        IFN  FLAG, <  LADO = Ø
                      VAR1       >
     >
```

A macro TESTE, pertencente ao arquivo auxiliar do processador de macros verifica se o argumento VAR2 é uma variável simples ou é chamada de outra macro, por exemplo MAT, para variável indexada ou OPER, para cálculo de expressões. Como resultado, TESTE devolve a variável FLAG do MACRO-10 igual a zero ou 1 caso VAR2 seja simples ou não.

Se VAR2 for simples, é chamada a macro $LAE pertencente ao arquivo que gera código intermediário, carregando a posição de memória indicada por VAR2 no acumulador.

Se VAR2 for chamada de outra macro, coloca-se a variável LADO do MACRO-10 igual a 1 indicando que o valor calculado por VAR2 deve ser guardado no acumulador. A seguir, é feito o mesmo procedimento para VAR1. Agora, se VAR1 for simples, a macro $SAE do arquivo de geração de código intermediário, armazenará o conteúdo do acumulador na posição de memória dada por VAR1. Se VAR1 foi nova macro, a variável LADO agora indica que o conteúdo do acumulador deve ser armazenado na posição de memória calculada por VAR1.

2º ARQUIVO: Macros que geram código intermediário

As definições das macros pertencentes a esse arquivo são responsáveis pelos testes para identificar o tipo da geração e o tipo da variável a ser operada e assim gerar o código intermediário referente à operação e a variável selecionada. As expansões das macros deste arquivo só podem conter nomes de macros do arquivo de geração de linguagem simbólica de cada microprocessador específico. Desse modo este arquivo atua como interface entre as linguagens de programação desenvolvidas e as linguagens simbólicas geradas pelo terceiro arquivo.

Acompanhando a declaração LET já mencionada, sua expansão pode chamar a macro $LAE pertencente a esse arquivo. Vejamos então como exemplo a definição de $LAE, onde foram omitidas o tratamento das variáveis formais de subrotinas pra facilidade de explicação.

```
DEFINE   $LAE   (NOME)
     <   ANALVA  NOME
         IFE $S,    $LAREB  NOME >
         IFE $S-1,  $LAREW  NOME >
         IFE $S-2,  $LAREF  NOME >
     >
```

. A macro ANALVA pertencente ao arquivo auxiliar do processador de macros, verifica o tipo da variável a ser carregada no acumulador. ANALVA devolve na variável $S do Macro-10 o valor 0, 1 ou 2 se o tipo de NOME for BYTE (8 bits), WORD (16 bits) ou FLOAT (32 bits). As tres últimas linhas da definição de $LAE chamarão uma das macros $LAREB, $LAREW, ou $LAREF que é o código intermediário para cada tipo de variável passada como parâmetro por $LAE.

3º ARQUIVO: Macros que geram linguagem simbólica de cada microprocessador

Esse conjunto de macros é o mais simples porém o maior. Sua função é traduzir quase que diretamente uma operação em código intermediário para a linguagem simbólica do microprocessador desejado.

Na chamada da macro $LAE já vista, caso a variável referenciada for do tipo BYTE, $LAREB será chamada. Vejamos portanto o corpo da definição de $LAREB que pertence ao arquivo de geração da linguagem simbólica para o microprocessador 8086.

```
DEFINE      $LAREB   (ENDER)
        <   EXPAN  < LDA   ENDER > >
```

Sendo a macro EXPAN pertencente ao arquivo auxiliar ao processador de macros, responsável pela listagem do seu argumento no arquivo binário de saída do MACRO-10. A instrução LDA ENDER é a ilustração simbólica do 8086 que carrega o conteúdo do endereço ENDER no acumulador AL de 8 bits.

4º ARQUIVO: Macros auxiliares ao compilador

Além dos três conjuntos de macros descritos que são responsáveis pela tradução do texto fonte em linguagem simbólica, um outro conjunto de macros é usado para aumentar as facilidades do processador de macros necessárias ao processo de compilação. Esse conjunto de macros irá gerar a linguagem simbólica no arquivo de saída binário do MACRO-10, testar operandos, simular pilha em tempo de compilação, elaborar tabela de símbolos e criar rótulos diferentes automaticamente.

a) Geração da listagem em linguagem simbólica do microprocessador desejado

A macro responsável por essa função, utiliza na definição de seu corpo a pseudo operação .TEXT mencionada no item 2.2.2. Consegue-se deste modo que o arquivo de saída que gera código para o PDP-10 seja um texto formado da linguagem simbólica do microprocessador.

b) Teste de operando

Qualquer declaração de linguagem desenvolvida é formada por um operador, que é uma chamada de macro, e seus operandos opcionais. Esses por sua vez podem ser novas chamadas de macros. Existe portanto uma macro que verifica se um dado operando possui o caracter"(". Caso encontre, essa macro devolve o valor da variável interna FLAG

igual a 1 indicando que aquele argumento consiste de uma chamada de macro. Caso contrário, FLAG possuirá valor zero informando que o operando é simples.

c) Pilha em tempo de compilação

Uma ferramenta necessária a um compilador é a pilha e foi implementada por duas macros que empilham e desempilham um certo valor desejado. Esses valores são guardados nas variáveis internas do MACRO-10. Essa pilha reúne principalmente informações necessárias a recursividade usada no processo de compilação.

d) Tabela de símbolos

O armazenamento das informações pertinentes cada variável da linguagem desenvolvida é feito usando campos de conteúdo da variável do mesmo nome interna ao MACRO-10. As informações guardadas são: i) tipo de variável: BYTE, WORD ou FLOAT, ii) dimensão e iii) indicação se a variável é real ou formal, no tratamento de subrotinas.

Duas macros são responsáveis pela manipulação da tabela de símbolos internos: uma preenche a tabela de símbolos no instante da declaração de uma variável e outra busca nesta tabela informações de uma dada variável já declarada.

e) Criação automática de rótulos distintos

Um conjunto de macros que concatena uma letra ao valor de um contador inteiro do MACRO-10 é responsável pela formação de rótulos distintos usados no texto final da linguagem simbólica gerada.

A listagem das macros que constituem esses quatro arquivos se encontram em |20| .

2.2.5. Descrição do Código Intermediário

Para que o tradutor gere linguagem simbólica de vários microprocessadores, há necessidade de se usar um código comum que seja intermediário entre a linguagem de alto nível e a linguagem simbólica de diversos microcomputadores [5] . A tradução de código intermediário é feita pelo terceiro arquivo de macros do compilador.

Para facilitar a descrição do código intermediário, foi idealizado um microcomputador abstrato capaz de executar esse código. Cabe salientar aqui que o código intermediário usado neste trabalho não é geral, devido ao pouco tempo disponível a um estudo mais aprofundado desta questão. Contudo a idéia de se usar esse código é geral e os trabalhos que forem feitos neste sentido [21] devem estendê-lo, aproveitando a mesma estrutura do compilador usada neste trabalho.

A seguir é feita uma descrição do microprocessador abstrato e um detalhamento de suas instruções, que constituem o código intermediário. Nesta seção também é mostrada a correspondência de registros do 8086 usados para simular o microprocessador abstrato.

O microprocessador abstrato foi idealizado para o tratamento de cálculos em ponto flutuante necessários aos programas de controle adaptativos abordados neste trabalho, podendo operar com dados inteiros com 8 ou 16 bits ou ponto flutuante com 24 bits na sua quase totalidade de operações com dados.

A arquitetura desse microprocessador, mostrada na fig.2.3,consiste de uma memória organizada em "bytes" de 8 bits, uma bandeira indicando resultados de operação booleana e cinco registros: acumulador, podendo ser de 8 (AB), de 16 (AW) e 24 bits (AF), contador de programa de 16 bits (CP), ponteiro de pilha de 16 bits (PP), registro de índice de 16 bits (RI) e ponteiro de base (PB) também de 16 bits.

Possui quatro formas de endereçamento:

- direto: o endereço é dado diretamente pelo operando da instrução.
- indexado: o endereço é calculado somando-se o conteúdo do registro de

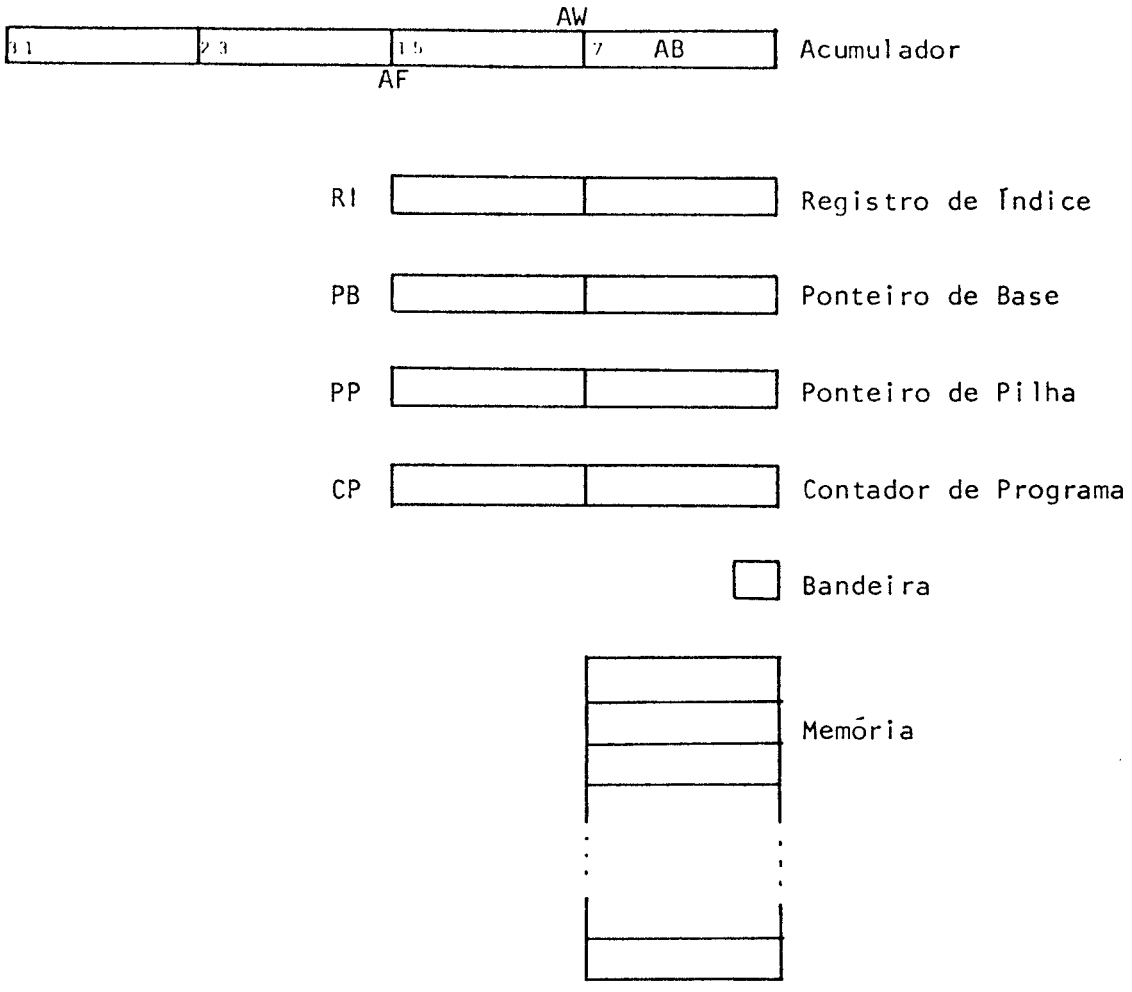


FIG 2.3 - Arquitetura do Microprocessador Abstrato

índice a um deslocamento de 16 bits dado pelo operando da instru - ção.

- indireto: o cálculo do endereço é feito com o conteúdo da posição de memória endereçada pelo registro de base somado a um deslocamento de 16 bits dado pelo operando da instrução.

- pilha: o endereço é dado pelo ponteiro de pilha, e o acesso ao topo da pilha segue a regra: primeiro a entrar, último a sair.

A equivalência de registros usada para a tradução do código intermediário em linguagem simbólica do 8086 está mostrada na tabela 2.4.

O jogo de instruções do microprocessador idealizado é dividido em cinco partes: movimento de dados, operações aritméticas e lógicas, controle do programa, controle do processador e operações de entrada e saída. Na tabela 2.5 é apresentado o mnemônico e uma breve descrição de cada operação. Os mnemônicos utilizados iniciam sempre com o caracter "$" para distinguí-los dos mnemônicos da linguagem de alto nível desenvolvida da linguagem simbólica, do microprocessador objetivo e das instruções do MACRO-10. Após o "$", podem vir até cinco letras indicando o tipo da instrução, "load", "store", "call", etc., e o tipo de operando, se é real (R) ou formal (D), se é BYTE (B), WORD (W)ou FLOAT (F). Usa-se essa formação do mnemônico para facilitar a sua tradução,visto que deste modo as instruções do microprocessador abstrato possuem no máximo um operando sempre de um mesmo tipo, não havendo portanto necessidade de testes durante a tradução das instruções do microprocessador idealizado para as do microprocessador objetivo.

A passagem de parâmetros para a subrotina é feita por referência . Os endereços dos parâmetros reais são guardados num vetor apontado pelo ponteiro de base (PB) e a cada parâmetro formal é associado um deslocamento em relação ao PB, podendo-se deste modo obter o endereço da variável real usando-se o modo de endereçamento indireto.

As operações aritméticas e lógicas são efetuadas entre o acumulador e o dado contido no topo da pilha, retornando o resultado no acumulador, e a pilha com um elemento a menos.

Microprocessador Abstrato	Microprocessador 8086
AB	AL
AW	AX
AF	DX:AX
RI	SI
PB	BP
PP	SP
CP	PC
BANDEIRA	AX

TABELA 2.4 : Mapeamento dos Registros do 8086 Usados para Simulação do Microprocessador Abstrato.

INSTRUÇÃO	DESCRIÇÃO
$SAvts	; armazena acumulador na memória
$LAvts	; carrega acumulador da memória
$LIvts	; carrega registro de índice da memória
$IMvts	; incrementa memória
$DMvts	; decrementa memória
$OADDs	; operações aritméticas
$OSUBs	
$OMULs	
$ODIVs	
$OANDs	; operações lógicas
$OORs	
$OXORs	
$OEQs	; operações que afetam a bandeira
$ONEs	
$OGTs	
$ONGs	
$OGEs	
$OLTs	
$ONLs	
$OLEs	
$PUSHs	; coloca o acumulador no topo da pilha
$POPs	; retira o topo da pilha e coloca no acumulador

TABELA 2.5 : Jogo de Instruções do Microprocessador Abstrato (continua)

INSTRUÇÃO	DESCRIÇÃO
$CACIJ	; calcula endereço de elemento matricial
$JTRUE	; desvio do programa se a bandeira for verdadeira
$JFALS	; desvio do programa se a bandeira for falsa
$JUMP	; desvio incondicional do programa
$CALL	; chamada de subrotina
$SUB	; declaração de subrotina
$RET	; retorno de subrotina
$HALT	; pára o processador
$NOP	; operação nula
$BREAR	; inserção de "breakpoint"
$READS	; leitura de dados
$TYPES	; impressão de dados
$PRINT	; impressão de texto
$CDA	; saída via conversor D/A
$CAD	; entrada via conversor A/D

onde:

v -
- R; indica variável real
- D; indica parâmetro formal de subrotina

t -
- E; indica variável escalar
- V; indica variável vetorial, matricial

s -
- B; indica variável de 8 bits
- W; indica variável de 16 bits
- F; indica variável de 24 bits em ponto flutuante

TABELA 2.5 : Jogo de Instruções do Microprocessador Abstrato (continuação)

O registro de um bit chamado bandeira é usado para desvios condicionais do programa pelas instruções $JTRUE e $JFALS. As instruções que modificam a bandeira são as instruções de relação, que operam com os dados do acumulador e o topo da pilha de maneira análoga às operações aritméticas e lógicas, retornando a bandeira com o valor 1 ou 0 se a relação for verdadeira ou falsa respectivamente.

As operações de entrada e saída podem ser feitas pelo teletipo, comunicando-se com o operador e feitas pelos conversores digitais-analógicos e analógicos-digitais para interação com o processo a ser controlado pelo microprocessador.

2.2.6. Auxílios à Depuração no Processo de Compilação por Macros

A fim de prestar ajuda ao usuário durante a fase de tradução da linguagem fonte à linguagem simbólica do microprocessador desejado, foram implementadas três variáveis que modificam o modo de operação do compilador conforme os valores que elas possam assumir:

$TRACE: É inicialmente colocada com o valor 1 e permite, no início e término da tradução de cada macro da linguagem fonte, uma mensagem via terminal indicando qual macro que está sendo processada. Se seu valor for colocado em zero, essa mensagem é inibida.

$SAIDA: Inicializada em 1, o MACRO-10 gerará um texto objeto final com comentários indicando o comando da linguagem fonte que expandiu o código simbólico assinalado. Caso a variável possua valor zero, esse processo de gerar comentários no programa simbólico é interrompido.

$COMPI: Com valor 0, o processo de compilação é normal;
com valor 1, não há geração de linguagem simbólica, sendo usado esse método apenas na deteção de erros;
e com valor 2, o MACRO-10 fornece a listagem da expansão de todas as macros envolvidas no processo de compilação.

Essas três variáveis podem ser inseridas no programa fonte e selecionar partes do programa a serem analisadas conforme as necessidades do usuário.

2.3. GERAÇÃO E CARREGAMENTO DO CÓDIGO DE MÁQUINA

Conforme descrito na introdução desse capítulo, o desenvolvimento de software básico de microprocessadores é feito no computador PDP-10. O tradutor baseado no processador de macros do MACRO-10 descrito no item 2.2 dá como saída a linguagem simbólica do microprocessador desejado. Nesta seção é descrito o processo de geração do código de máquina do microprocessador, a partir de sua linguagem simbólica e a forma de carregamento desse código no microcomputador.

2.3.1. Geração do Código de Máquina

A montagem do código de máquina a partir da linguagem simbólica é feita pelo programa montador. Se esse programa reside em uma outra máquina, seu nome é montador cruzado e se este permite a montagem de código de várias máquinas, o programa é chamado de montador cruzado generalizado.

Neste trabalho utilizou-se o montador cruzado generalizado desenvolvido no laboratório de Micro e Minicomputadores da FEC [2] com a extensão para a geração de código para o microprocessador 8086. A seguir é descrita a estrutura desse programa e alguns detalhes específicos da implementação da geração de código para o 8086.

a) Estrutura do Montador Cruzado Generalizado

O montador cruzado foi implementado em Fortran para dar o caráter portátil ao programa, podendo ser implantado nos computadores que possuam um compilador Fortran. Devido ao aparecimento de um número crescente de microprocessadores a todo ano, procurou-se dar ao montador um aspecto geral, isto é, que pudesse gerar código objeto para vários microcomputadores diferentes. Isso foi possível pela padronização, entre os diversos microprocessadores, das funções

realizadas através de um montador comum. Como resultado foi implementado um único programa que, em função do micro para o qual será gerado código objeto, buscará as informações necessárias numa tabela específica do microcomputador para o qual o programa fonte será traduzido em programa objeto. A implementação de um novo microcomputador na estrutura é conseguida através da criação desta tabela. Isto reduz significativamente o tempo de implementação se compararmos com aquele correspondente ao desenvolvimento de um montador para um único microcomputador. Como vantagem adicional, esse programa necessita de um espaço menor de armazenamento comparado ao espaço ocupado por vários montadores específicos de cada microprocessador. Maior detalhe pode ser encontrado em |2|.

b) Extensão do Montador Cruzado para inclusão do Intel 8086

Conforme descrito em |2|, os passos necessários para a implementação de um microprocessador na estrutura do Montador Cruzado Generalizado é a seguinte:

- Construção da Tabela de Instruções
    - Essa tabela possui o mnemônico, o código objeto básico, o seu comprimento (número de "bytes") o número de operandos, e informações quanto ao tipo de operando (quando houver), referentes a cada instrução do microprocessador a ser implementado.

- Criação de uma subrotina para leitura das informações contidas na tabela de instruções.

- Criação de uma subrotina para definição de símbolos utilizados pelo fabricante para referenciar registros.

- Implementação dos tipos de operandos usados no novo microprocessador ainda não pertencentes ao montador.

Na implementação do montador cruzado para o 8086 apareceram algumas dificuldades devido ao fato desse microprocessador possuir uma montagem de

código de máquina mais complexa que os micros anteriores, apresentando operandos tanto de 8 como 16 bits e várias formas de endereçamento numa mesma instrução. O seu número de instruções é grande, possuindo da ordem de 130 tipos diferentes de códigos de máquina básicos.

Essas características aliadas ao fato do montador cruzado generalizado possuir uma estrutura que não permite para um mesmo mnemônico, vários comprimentos de código de máquina, levaram a implementação de um número grande de mnemônicos tornando a tabela de instruções bastante extensa.

2.3.2. Carregamento do Código de Máquina criado no computador hospedeiro

O carregamento do código objeto do microcomputador, gerado pelo montador cruzado no computador hospedeiro é feito diretamente pelo "link" de comunicação seriada existente entre o centro de computação e o laboratório de microcomputadores da FEC.

O formato usado no código objeto é o arquivo objeto hexadecimal consistindo de várias linhas as quais apresentam a seguinte forma:

```
:NN AAAA TT DD DD ... DD CC
```

onde : (dois pontos) indica o início da linha;

NN é um número hexadecimal informando o número de "bytes" a ser carregado na linha;

AAAA é o endereço em hexadecimal a partir do qual os dados são carregados;

TT é um código para o carregador: 00 indica carregamento de dados e
01 indica fim de carregamento;

DDDD ... DD são dados hexadecimais, onde cada 2 caracteres formam um dado de 8 bits; e

CC é um número hexadecimal que informa o "check-sum" , o qual adicionado a todos os números hexadecimais de 8 bits desta linha dá como resultado 00 , considerando apenas os seus 8 bits menos significativos. Esse campo de caracter é usado na deteção de erros de transmissão.

2.4. ARITMÉTICA EM PONTO FLUTUANTE

Qualquer algoritmo de controle adaptativo supõe um volume de cálculo não desprezível que implicará, numa aplicação industrial, o uso de um processador "hardware" para as operações aritméticas em ponto flutuante.Tais processadores estão hoje disponíveis no mercado, cabendo citar como exemplo o Intel 8232 , entre outros. Na ausência de um processador "hardware" no laboratório, foi desenvolvido um conjunto de subrotinas responsáveis pela aritmética em ponto flutuante.

Descreve-se a seguir, o formato utilizado e os algoritmos implementados para as quatro operações básicas. Esta seção finaliza com uma comparação dos tempos de execução desta aritmética com os tempos do processador Intel 8232 e com a descrição das subrotinas de conversão de base e os formatos de entrada e saída de dados necessários para a comunicação com o operador e com o processo.

Utilizam-se 16 bits com sinal, em complemento de 2 para a mantissa normalizada entre $\frac{1}{2}$ e 2 , e 8 bits com sinal, também em complemento de 2 para o expoente. Com esse formato, pode-se representar valores numa variação de aproximadamente $\pm 10^{38}$ com a precisão de $4\frac{1}{2}$ dígitos decimais (aprox. 0,003%).

Este formato minimiza o tempo de execução das operações básicas implementadas no microprocessador 8086, pois a sua arquitetura permite instruções de multiplicação e divisão de números em complemento de 2 com até 16 bits, e instruções de tratamento de "overflow" de números nesta representação.

Não se usou o formato de aritmética binária em ponto flutuante para mini e micro-computadores recomendados pelo IEEE |22|, pois como esse formato utiliza mantissa de 23 bits, representada em sinal e magnitude, e o 8086 trabalha com números de no máximo 16 bits, a execução dos algoritmos ponto flutuante seria mais lenta, pois as operações com 23 bits deveriam ser feitas por duas ou mais operações de 16 bits do 8086. A representação do número ponto flutuante em sinal e magnitude também retardaria o processamento das operações básicas, pois seria necessário o desempacotamento do número para se trabalhar separadamente com o sinal e o valor absoluto da mantissa.

Como o fator tempo é importante para controle em tempo real e o comportamento do algoritmo de controle adaptativo auto-ajustável é bom com a precisão de $4\frac{1}{2}$ dígitos |7|, pode-se dizer que o formato utilizado maximiza a relação [precisão/tempo de execução], para a implementação no Intel 8086.

O tratamento de exceções ("overflow" e "underflow"), foi implementado como se descreve a seguir.

Em caso de "overflow", é deixado o maior valor possível de ser representado e em "underflow" é colocado o menor valor diferente de zero. Com essa tática tentou-se fazer com que a matriz de covariança no algoritmo de estimação tratado no capítulo 3 não tenha seus elementos zerados por imprecisão nos cálculos, o que levaria à não estimação dos parâmetros que possuissem variância zero. São usadas duas posições de memória para assinalarem as ocorrências de "overflow" e "underflow".

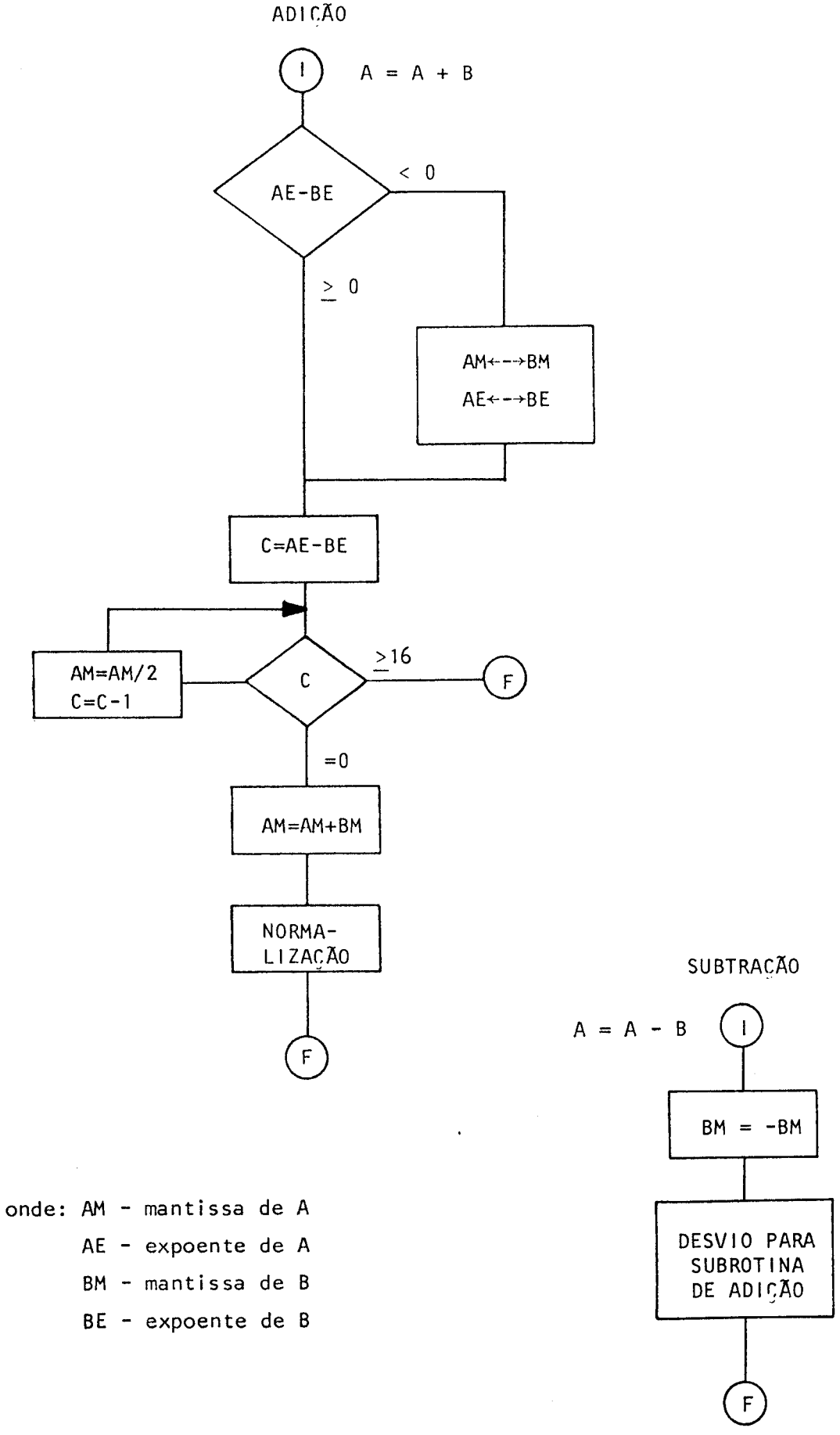
Os algoritmos usados para as quatro operações básicas assim como também o de normalização são mostrados nas figuras 2.4. e 2.5.. São todos reentrantes utilizando apenas os registros da CPU como área de trabalho. As listagens dessas subrotinas se encontram em |20| .

Os tempos de execução e memória utilizada pelas quatro operações básicas são apresentados na tabela 2.6. É mostrado também o tempo requerido pelo processador ponto flutuante 8232 para operar com os números no formato padronizado pelo IEEE já citado.

Quanto aos algoritmos implementados no 8086 observa-se pouca diferença , de tempo de execução entre as operações de soma/subtração e multiplicação/divisão. Isto se deve ao fato do 8086 apresentar operações de multiplicação e divisão inteira de 16 bits no seu conjuntos de instruções.

Pela tabela, observa-se também que um processador ponto flutuante por "hardware" possui um desempenho bastante superior quando comparado aos algoritmos implementados por "software".

Os algoritmos de controle adaptativo, de um modo geral necessitam de uma inicialização nos parâmetros do controlador. É necessário então uma comunicação



onde: AM - mantissa de A
AE - expoente de A
BM - mantissa de B
BE - expoente de B

FIG 2.4 - Algoritmos de Adição e Subtração em Ponto Flutuante

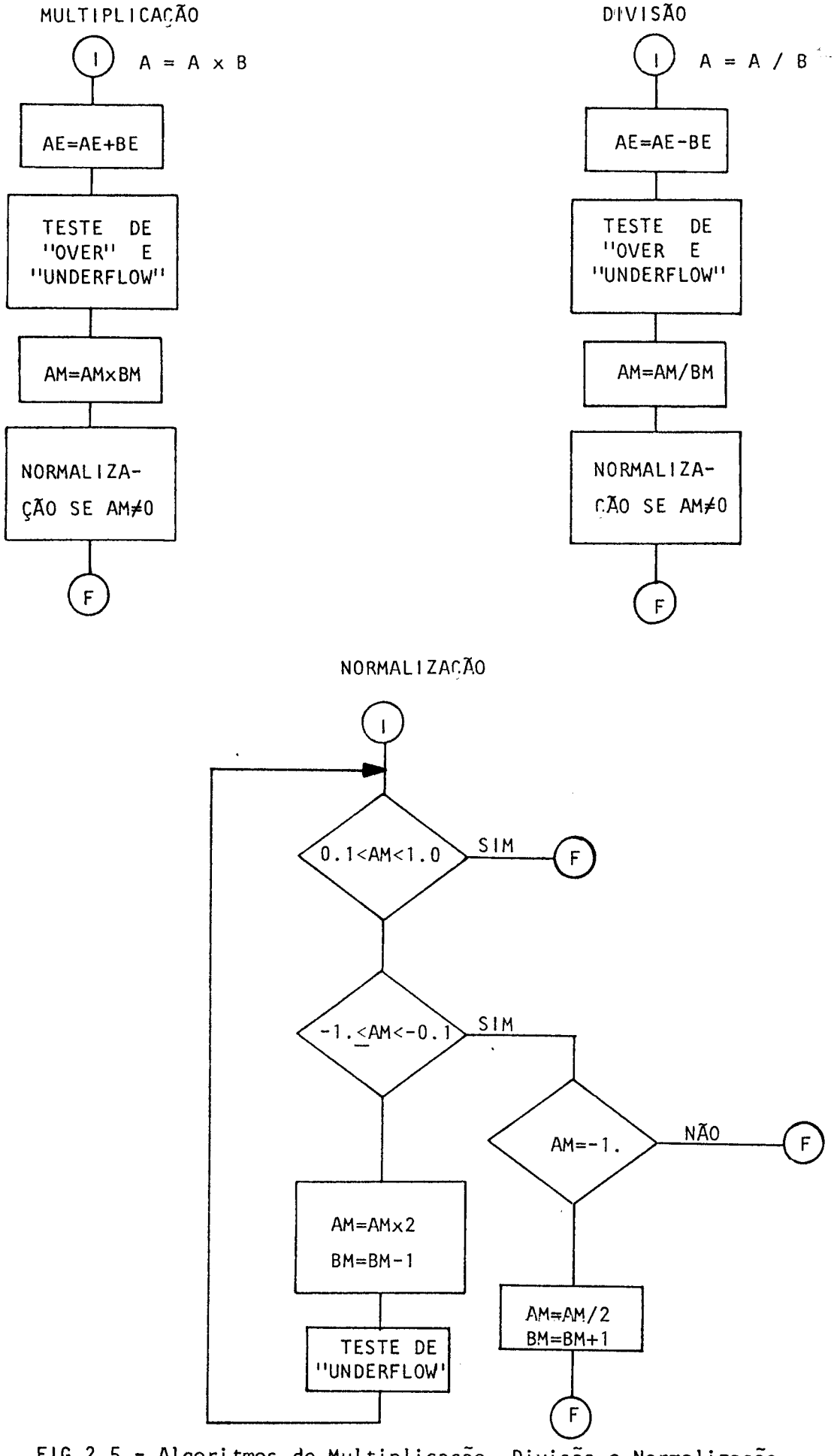


FIG 2.5 - Algoritmos de Multiplicação, Divisão e Normalização

operação	rotinas ponto flutuante Intel 8086 precisão de 15 bits		processador ponto flutuante 8232 precisão de 23 bits
	ciclos (média)	memória (bytes)	ciclos
soma / subtração	350	68	57
multiplicação	450	65	198
divisão	550	84	228

ciclo do 8086 - 125μs a 2000 μs

TABELA 2.6 : Tempos de Execução das Operações Aritméticas Básicas em Ponto Flutuante

entre este e o operador. As subrotinas de conversão da base dois para a base dez e vice-versa de números em ponto flutuante se tornam portanto necessárias apenas nos diálogos com o homem. Podemos notar que não existe necessidade de rapidez nem alta precisão nas rotinas destinados a essa comunicação. Sendo assim, utilizam-se algoritmos descritos na figura 2.6. que minimizam memória por não necessitarem tabelas de conversão e sim usarem multiplicação e divi - são sucessivas em ponto flutuante. A precisão do número convertido por este processo é um pouco menor que a precisão que o formato permite, mas para o objetivo que se pretende, é completamente satisfatória.

Dois formatos de entrada e saída de dados são requeridos: um para a comunicação com o operador, e outro na comunicação com o processo via conversores digitais-analógicos e analógicos-digitais.

O formato de saída via terminal é da forma

±D.DDDDE±DD

onde D é um algarismo decimal.

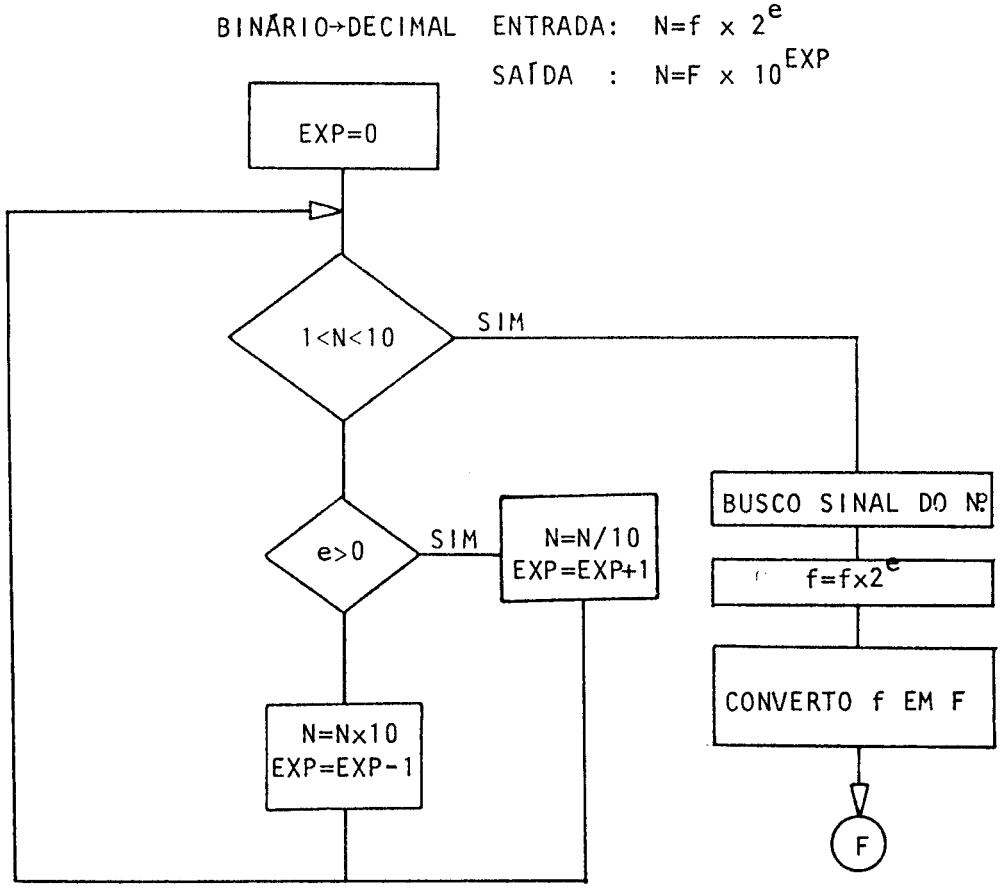
E o formato de entrada pode ser descrito pelo diagrama sintático da figura 2.7 onde D é um algarismo decimal.

As conversões A/D e D/A utilizam doze bits na representação binária complementar de 2, com a limitação da faixa de atuação entre -1 e +1. Assim os valores de saída são truncados dentro dessa faixa.

2.5. CONCLUSÃO

A relação entre o custo de desenvolvimento de "software" e o de"hardware" está crescendo fortemente e são necessárias técnicas de desenvolvimento de "software" que viabilize o uso de controladores baseados em microprocessadores na indústria.

A técnica de desenvolvimento de software descrita neste capítulo procura minimizar a necessidade de aquisição de novos sistemas, fazendo uso de um computador hospedeiro para o desenvolvimento de "software" cruzado. Existe uma



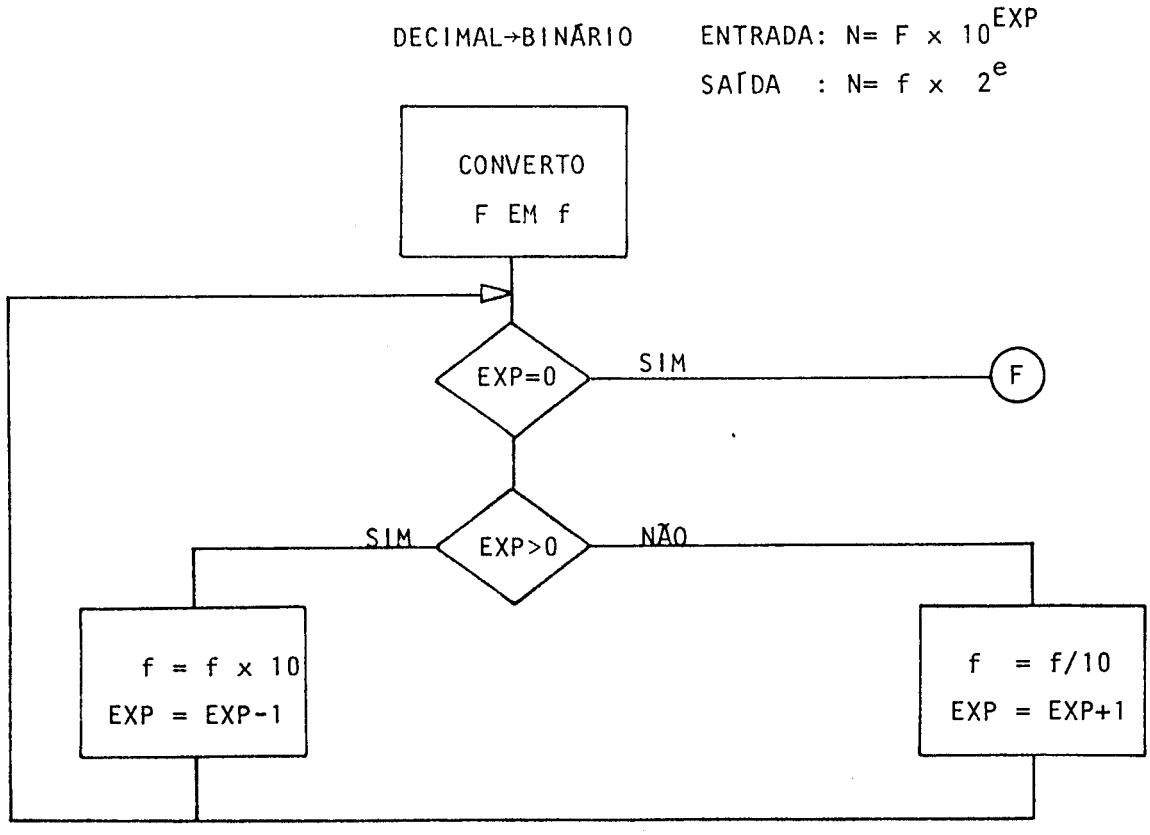


FIG 2.6 - Rotinas de Conversão Binário-Decimal e Decimal-Binário

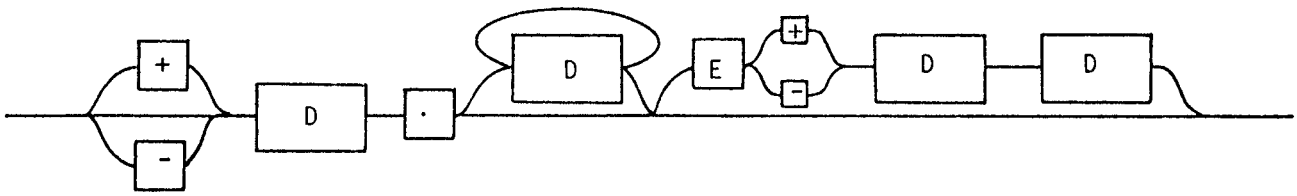


FIG 2.7 - Diagrama Sintático do Número Decimal em Ponto Flutuante

preocupação de que esse conjunto de programas de "software" básico se adapte sem grandes esforços para atender às novas máquinas computacionais que possam ser lançadas comercialmente.

Esse "software" básico foi conseguido por dois conjuntos de programas utilizando técnicas diferentes de programação. Um deles, mais básico é o montador cruzado implementado em FORTRAN e detalhadamente descrito em |2| . O segundo conjunto consiste de um compilador não convencional, utilizando os recursos de um processador de macros existente no macro montador do PDP-10. Em relação a essa técnica, algumas vantagens e desvantagens principais podem ser assinaladas. Como vantagem, ela permite a construção de um compilador em curto espaço de tempo e bastante compacto, podendo facilmente ser entendido por outras pessoas. Por outro lado, suas principais desvantagens são: o tempo de compilação, que chega a ser de uma ordem de grandeza maior que os compiladores convencionais e a sintaxe que obriga a linguagem projetada atender à sintaxe do macro montador do PDP-10. Esse último inconveniente pode ser eliminado utilizando-se um processador de macros de uso geral como ML/I e STAGE2 por exemplo. Conclui-se portanto que essa técnica não deve ser usada para construção de compiladores que serão usados na produção constante de "software", mas sim em projetos especiais e de pesquisa.

Em relação à aritmética em ponto flutuante desenvolvida, o avanço tecnológico em microeletrônica permitirá em poucos anos que as operações em ponto flutuante façam parte do jogo de instruções dos futuros microprocessadores. Assim, o problema dos cálculos em ponto flutuante deixarão de ser analisados na implementação de algoritmos de controle em microprocessadores.

CAPÍTULO 3

IMPLEMENTAÇÃO DO CONTROLADOR AUTO-AJUSTÁVEL

3.1. INTRODUÇÃO

A utilização de algoritmos eficientes para o controle de processos industriais impõe invariavelmente o conhecimento do modelo dinâmico do sistema a ser controlado. Grande parte dos processos industriais básicos de transformação de matéria prima são pouco conhecidos de um ponto de vista dinâmico, dada a complexidade dos processos físico-químicos que ocorrem. A técnica normalmente usada consiste na construção de um modelo experimental de baixa ordem com o menor número possível de parâmetros e que modele convenientemente o processo real para pequenas variações em torno do regime normal de funcionamento. Tal modelo poderá não conter parâmetros com significado físico, e seus parâmetros deverão ser determinados por identificação experimental a partir das medidas das entradas e saídas do processo real. Perturbações e mudanças no ponto de funcionamento do sistema real fazem com que os parâmetros do modelo reduzido não sejam constantes e devem ser adaptados "on-line" originando-se o problema característico de controle adaptativo.

Neste trabalho é implementado um algoritmo auto-ajustável multivariável |11|, que é uma generalização do algoritmo monovariável proposto em |7|. O controlador auto-ajustável é o controlador adaptativo que está sendo mais usado em aplicações industriais. Trata-se de um controlador do tipo CE ("certainty equivalence") |6| com necessidades de cálculo relativamente pequenas que já vem permitindo a sua implementação em microprocessadores desde 1975. |7|

O algoritmo de controle apresentado em |11| é resumido na seção 3.2.

Para a implementação de um algoritmo num microprocessador, é necessário o uso de um sistema de desenvolvimento que permita o projeto, a programação, os testes e a depuração do controlador. Esse sistema pode ser simples, contendo poucas ferramentas de desenvolvimento, requerendo assim um trabalho maior por parte do projetista ou pode ser bastante sofisticado, com diversas facilidades que permitam ao engenheiro de controle de processos o projeto de um controlador em curto espaço de tempo.

O sistema de desenvolvimento usado neste trabalho consiste das ferramentas de software mencionadas no capítulo 2, que se encontram no computador de médio porte PDP-10, e da parte de "hardware" formada pelo microprocessador

Intel 8086 de 16 bits e pelo computador analógico EAI-680. Na seção 3.3. é descrita a estrutura do sistema de desenvolvimento destacando-se o microcomputador e a interface analógica entre este e o computador analógico.

A seguir são mostrados os passos necessários à implementação do algoritmo auto-ajustável, utilizando o sistema de desenvolvimento, e por último, são feitas algumas simulações para mostrar o funcionamento do controlador implementado, comparando-o com os resultados obtidos pela simulação feita exclusivamente no computador PDP-10.

3.2. ALGORITMO AUTO-AJUSTÁVEL MULTIVARIÁVEL

O algoritmo apresentado a seguir é destinado ao controle de processos de múltiplas entradas e múltiplas saídas cujos parâmetros são constantes, (ou variáveis lentamente no tempo) mas desconhecidos.

O processo a ser controlado é descrito por uma equação vetorial estocástica às diferenças:

$$\begin{aligned} \underline{y}(t) + A_1 \underline{y}(t-1) + \ldots A_n \underline{y}(t-n) = \\ = B_0 \underline{u}(t-k) + B_1 \underline{u}(t-k-1) + \ldots + B_n \underline{u}(t-k-n) + \\ + \underline{e}(t) + C_1 \underline{e}(t-1) + \ldots + C_n \underline{e}(t-n) + \underline{d}, \end{aligned} \qquad (1)$$

onde:

$\underline{y}(t)$ - vetor de saída: $p \times 1$

$\underline{u}(t)$ - vetor de controle: $p \times 1$

$\underline{d}$ - vetor de entrada constante $p \times 1$

A_i, B_i e C_i são matrizes $p \times p$;

k - atraso de transporte

Supõe-se que B_0, sem perda de generalidade é não singular e que $\{\underline{e}(t), t = 1,2,\ldots\}$ é uma sequência de vetores aleatórios $p \times 1$ normais, independentes, igualmente distribuídos, com média zero e matriz de covariância R.

Se definirmos as matrizes polinominais

$$A(q^{-1}) = I + A_1 q^{-1} + A_2 q^{-2} + \ldots A_n q^{-n}, \qquad (2a)$$

$$B(q^{-1}) = B_0 + B_1 q^{-1} + B_2 q^{-2} + \ldots + B_n q^{-n}, \qquad (2b)$$

$$C(q^{-1}) = I + C_1 q^{-1} + C_2 q^{-2} + \ldots C_n q^{-n}, \qquad (2c)$$

o processo pode ser representado por:

$$A(q^{-1})\ \underline{y}(t) = B(q^{-1})\ \underline{u}(t-k) + C(q^{-1})\ \underline{e}(t) + \underline{d} \qquad (3)$$

O critério do custo a ser minimizado pelo controlador é dado por:

$$J = E\{[(\underline{y}(t+k) - \underline{w}(t))^T (\underline{y}(t+k) - \underline{w}(t)) + \underline{u}^T(t)\, Q\underline{u}(t)]/\underline{y}(t), \underline{u}(t), \underline{y}(t-1) \ldots\} \qquad (4)$$

onde:

$E\{.\}$ é o operador esperança matemática

$\underline{w}(t)$ é o vetor de referência $p \times 1$ ("set point")

Q é uma matriz $p \times p$ definida positiva que representa a ponderação no sinal de controle

A obtenção do algoritmo de controle implica na construção de um predidor do vetor de saída k passos à frente.

A equação que fornece esta previsão [11] é dada por:

$$\tilde{C}(q^{-1})\ \hat{\underline{y}}(t+k/t) = \tilde{F}(q^{-1})\ \underline{y}(t) + \tilde{E}(q^{-1})\ B(q^{-1})\underline{u}(t) + \tilde{E}(q^{-1})\ \underline{d} \qquad (5a)$$

$$\underline{y}(t+k) = \hat{\underline{y}}(t+k/t) + E\ \underline{e}(t+k) \qquad (5b)$$

onde as matrizes polinominais envolvidas são dadas por:

$$\tilde{E}\,(q^{-1})\;F\,(q^{-1}) = \tilde{F}\,(q^{-1})\;E\,(q^{-1}) \tag{6a}$$

$$\det\;\tilde{E}\,(q^{-1}) = \det\;E\,(q^{-1}) \tag{6b}$$

$$\tilde{E}\,(o) = E\,(o) = I \tag{6c}$$

$$\tilde{E}\,(q^{-1})\;C(q^{-1}) = \tilde{C}(q^{-1})\;\;E(q^{-1}) \tag{7a}$$

$$\det\;\tilde{C}\,(q^{-1}) = \det\;C(q^{-1}) \tag{7b}$$

$$C\,(q^{-1}) = A(q^{-1})\;E\,(q^{-1}) + q^{-k}\;F\,(q^{-1}) \tag{8}$$

com:

$$E\,(q^{-1}) = I + E_1\;q^{-1} + E_2\;q^{-2} + \ldots + E_{k-1}\;q^{-k+1} \tag{9}$$

$$F\,(q^{-1}) = F_0 + F_1\,q^{-1} + F_2 q^{-2} + \ldots + F_{n-1}\,q^{-n+1} \tag{10}$$

Utilizando o valor previsto da saída na função de custo (4)e procedendo à minimização em relação a $\underline{u}\,(t)$ obtém-se:

$$\hat{\underline{y}}\,(t + k/t) - \underline{w}\,(t) + \tilde{Q}\,\underline{u}\,(t) = 0 \tag{11a}$$

onde

$$\tilde{Q} = B_0^{-1}\,Q \tag{11b}$$

ou seja, o controle $\underline{u}\,(t)$ é determinado de forma a anular a previsão k passos a frente de uma saída auxiliar $\Phi(t+k)$ dada por

$$\Phi\,(t+k) = \underline{y}\,(t + k) - \underline{w}\,(t) + \tilde{Q}\,\underline{u}\,(t) \tag{12}$$

Chamando $$\tilde{G}\,(q^{-1}) = \tilde{E}\,(q^{-1})\;B\,(q^{-1}) + \tilde{C}\,(q^{-1})\;\tilde{Q} \tag{13}$$

temos: $$\tilde{G}\,(q^{-1})\;\underline{u}\,(t) = -\,\tilde{F}(q^{-1})\;\underline{y}\,(t) + \tilde{C}\,(q^{-1})\;\underline{w}\,(t) - \underline{d}^{\,*} \tag{14a}$$

onde: $$\underline{d}^{*} = \tilde{E}\;\;(q^{-1})\;\underline{d} = \tilde{E}(1)\;\underline{d} \tag{14b}$$

A equação anterior mostra que o vetor de controle $\underline{u}\,(t)$ é determinado como uma função de : i) vetor de controle nos instantes anteriores, ii) medidas do vetor da saída nos instantes anteriores e atual, iii) valor do vetor de referência nos instantes anteriores e atual.

e explicitamente é dada por:

$$\underline{u}(t) = -\tilde{G}_0^{-1} \{ \sum_{i=0}^{n-1} \tilde{F}_i \underline{y}(t-i) + \sum_{i=1}^{n+k-1} \tilde{G}_i \underline{u}(t-i) - \sum_{i=0}^{n} \tilde{C}_i \underline{w}(t-i) + \underline{d}^* \} \quad (15)$$

Como os coeficientes das matrizes polinominais $\tilde{G}(q^{-1})$, $\tilde{F}(q^{-1})$ e $\tilde{C}(q^{-1})$ são desconhecidos e variáveis no tempo, eles devem ser estimados "on line" para que o controle seja calculado.

Baseado nas equações (5a), (5b) e (12), podemos escrever

$$\Phi(t) = \tilde{F}(q^{-1})\, \underline{y}(t-k) + \tilde{G}(q^{-1})\, \underline{u}(t-k) - \tilde{C}(q^{-1})\, \underline{w}(t-k) + \underline{d}^* - \tilde{C}'(q^{-1})\, \hat{\underline{\Phi}}(t-1/t-k-1) + E(q^{-1})\, \underline{e}(t) \quad (16a)$$

onde:

$$\tilde{C}'(q^{-1}) = q\,(\tilde{C}(q^{-1}) - 1) \quad (16b)$$

Definindo

$$\underline{\Psi}(t) = \left| \underline{u}^T(t), \underline{u}^T(t-1), \ldots, \underline{y}^T(t), \underline{y}^T(t-1), \ldots; \underline{w}^T(t), \underline{w}^T(t-1), \ldots; \underline{o} \right|^T \quad (17a)$$

$$\Theta = \left| \tilde{G}_0 \;\; \tilde{G}_1 \ldots \;\; \tilde{F}_0 \;\; \tilde{F}_1 \ldots \;\; -\tilde{C}_0 \;\; -\tilde{C}_1 \;\; \ldots \;\; 1 \right| \quad (17b)$$

$$\underline{\upsilon}(t) = E(q^{-1})\, \underline{e}(t) - \tilde{C}'(q^{-1})\, \hat{\underline{\Phi}}(t-1/t-k-1) \quad (17c)$$

temos

$$\underline{\Phi}(t) = \Theta\, \underline{\Psi}(t-k) + \underline{\upsilon}(t) \quad (18)$$

Um algoritmo de estimação de mínimos quadrados ("least squares") é, então, utilizado para estimar a matriz Θ a partir da equação (18). Para que as estimações sejam assintáticamente não polarizadas, é necessário que a perturbação $\underline{\upsilon}(t)$ seja não correlata com o vetor de medidas $\underline{\Psi}(t-k)$. A componente $E(q^{-1})\, \underline{e}(t)$ de $\underline{\upsilon}(t)$ é não correlata com o vetor $\underline{\Psi}(t-k)$ pois é função apenas de valores do ruído $\underline{e}(j)$, $j = t, t-1, \ldots, t-k+1$. Entretanto as parcelas $\tilde{C}_i \hat{\Phi}(t-i/t-k-i)$, $i = 1, 2 \ldots n$ são correlatas com $\underline{\Psi}(t-k)$.

Ocorre porém que como o estimador opera em cascata com a lei de controle (11) , que anula a cada instante a previsão de Φ k passos à frente, as parcelas $\tilde{C}_i \hat{\Phi}(t-i/t-k-i)$ tendem a zero se os parâmetros estimados se aproximam de seus valores reais.

Assim, se a matriz de parâmetro Θ (t) convergir para os valores reais, a perturbação $\upsilon(t)$ será dada por:

$$\underline{\upsilon}(t) = E(q^{-1})\ \underline{e}\ (t) \qquad (19)$$

que não é correlata com $\underline{\Psi}$ (t-k). O estimador é assim, não polarizado assintoticamente.

A equação (18) pode ser escrita como um conjunto de equações

$$\Phi_i(t) = \underline{\Psi}^T(t-k)\ \underline{\Theta}_i + \underline{\upsilon}_i\ (t) \qquad i = 1,\ldots,\ p \qquad (20)$$

onde:

Φ_i e $\underline{\upsilon}_i$ sãos os i-ésimos componentes dos vetores Φ e $\underline{\upsilon}$

$\underline{\Theta}_i$ é a i-ésima linha da matriz Θ

As equações (20) permitem que se estime separadamente cada linha da matriz Θ , utilizando o seguinte estimador mínimos quadrados recursivo:

$$\hat{\underline{\Theta}}_i\ (t) = \hat{\underline{\Theta}}_i\ (t-1) + \underline{k}\ (t)\{\ \Phi_i(t) - \underline{\Psi}^T(t-k)\ \hat{\underline{\Theta}}_i\ (t-1)\ \} \qquad (21)$$

$$\underline{k}(t) = \frac{P(t-1)\ \underline{\Psi}\ (t-k)}{\alpha + \underline{\Psi}^T\ (t-k)\ P\ (t-1)\ \underline{\Psi}\ (t-k)} \qquad (22)$$

$$P(t) = \frac{1}{\alpha}\ \left|\ I - \underline{k}\ (t)\ \underline{\Psi}^T\ (t-k)\ \right|\ P(t-1) \qquad (23)$$

Esse algoritmo de identificação utiliza o mesmo vetor de ganho $\underline{k}$ (t)e a mesma matriz P(t) para a estimação de todas as linhas de Θ, com a consequente redução nas respectivas dimensões, diminuindo assim a memória de dados e tempo de execução requeridos pelo algoritmo. Obviamente isto requer uma mesma inicialização para a matriz de covariância dos parâmetros de cada linha |23|.

Nas equações (22) e (23) do estimador, α é um fator de esquecimento que permite o algoritmo ponderar mais as últimas medidas em relação às mais antigas. O fator de esquecimento é importante para a eliminação mais rápida da polarização introduzida por valores iniciais ruins. Além disso, permite que o controlador possa ser usado no controle de processos com parâmetros lentamente variáveis no tempo.

O algoritmo de identificação dado pelas equações (23) a (25) é sujeito a erros de arredondamento. Por exemplo, se $P(t-1)\ \underline{\Psi}\ (t-k)$ crescer muito, na equação (24), $\underline{k}\ (t)\ \underline{\Psi}^T\ (t-k)$ se tornará muito próximo a 1. A subtração em (25) estará portanto sujeita a erros de precisão de cálculos.

Uma solução para esse problema é utilizar o algoritmo da raiz quadrada [24] o qual atualiza, simultaneamente $S(t)$ definida em (26) e o vetor de ganho $\underline{k}(t)$ (22). Este método dobra a precisão dos cálculos e reduz o tempo de execução em relação ao algoritmo anterior.

Definindo $S(t)$ como

$$S(t)\ S^T(t) = P(t) \tag{26}$$

Sendo $S(t)$ uma matriz triangular superior, o

algoritmo é descrito pelas equações:

$$\underline{f} = S^T(t)\ \underline{\Psi}(t) \tag{27}$$

$$\mu_0 = \sqrt{\alpha} \tag{28}$$

$$\mu_i = \sqrt{\alpha + |f|_i^2}\ , \quad 1 \leq i \leq m \tag{29}$$

$$g_i^n = \begin{cases} \sum_{k=1}^{n} |S(t)|_{ik}\ |\underline{f}|_k\ , & i \leq i \leq m \\ 0\ , & \text{outros casos} \end{cases} \tag{30}$$

$$\left|\underline{k}(t)\right|_i = \frac{g_i^n}{\mu_n^2} \tag{31}$$

$$|S(t+1)|_{ij} = \frac{1}{\sqrt{\alpha}}\ \frac{\mu_{j-1}}{\mu_j}\ \left\{ |S(t)|_{ij} - \frac{|\underline{f}|_i\ g_i^{j-1}}{\mu_{j-1}^2} \right\} \tag{32}$$

onde $|\underline{x}|_i$ denota o i-ésimo elemento do vetor $\underline{x}$.

Por outro lado, este algoritmo permite uma economia de espaço de memória utilizado para armazenamento da matriz S(t).

Resumindo, o algoritmo envolve os seguintes passos:

1. Lê o valor dos vetores $\underline{y}$ (t) e $\underline{w}$ (t)

2. Salva os vetores $\underline{y}$(t), $\underline{w}$(t) e $\underline{u}$(t)

3. Forma o vetor de medidas $\underline{\Psi}$(t-k). Equação (17a)

4. Atualiza o ganho $\underline{k}$(t) e S(t)
   (Equações (27) a (32))

5. Repete para i de 1 a p:

   5a. Calcula a saída auxiliar Φ_i (t)

   $$\Phi_i(t) = y_i(t) - w_i\ (t-k) + \lambda_i\ u_i\ (t-k) \qquad (33)$$

   A matriz de ponderação $\tilde{Q}$ da equação (12) foi suposta diagonal.

   5b. Atualiza o vetor de parâmetros $\underline{\Theta}_i$(t) (Equação 21)

   5c. Calcula controle auxiliar $\bar{u}_i(t) = \hat{\underline{G}}_0^i\ \underline{u}(t)$ (34)

$$\bar{u}_i(t) = -\sum_{j=0}^{n-1} \hat{\underline{F}}_j^i\ \underline{y}(t-j) - \sum_{j=1}^{n+k-1} \hat{\underline{G}}_j^i\ \underline{u}(t-j) + \sum_{j=0}^{n} \hat{\underline{C}}_j^i\ \underline{w}(t-j) - \underline{d}_i^* \qquad (35)$$

6. Calcula controle $\underline{u}$ (t)

$$\underline{u}(t) = \hat{G}_0^{-1}\ .\ \bar{\underline{u}}(t) \qquad (36)$$

7. No próximo instante de amostragem, volta a 1.

3.3. "HARDWARE" UTILIZADO

3.3.1. Configuração do Sistema de Desenvolvimento

A estrutura utilizada no sistema de desenvolvimento é mostrada na figura 3.1. Consiste de três unidades básicas:

a) Computador PDP-10 de médio porte, onde é feito o desenvolvimento do "software" básico para o microprocessador e o monitoramento e acompanhamento do comportamento do controlador. No PDP-10 pode também ser simulado o processo a ser controlado pelo microcomputador.

b) Microcomputador SDK-86 onde é implantado o controlador interagindo com o processo simulado digitalmente no PDP-10 ou simulado analogicamente no computador EAI-680.

c) Computador Analógico EAI-680 que atua na simulação analógica do processo em tempo real a ser controlado.

Os periféricos mais usados do PDP-10 são: disco magnético onde são armazenados os programas e dados necessários ao desenvolvimento dos algoritmos, impressora de linha para listar programas e dados oriundos do 8086 ou do PDP-10, e "plotter" para desenhar gráficos do comportamento dos algoritmos em desenvolvimento.

Ao SDK-86 foram adicionados 32kbytes de memória RAM para a implantação e acompanhamento dos algoritmos de controle adaptativo. A sua interface seriada é utilizada para a comunicação com o PDP-10 ou com o TTY do operador, e a interface paralela é usada para a comunicação com o computador analógico. Pode-se também colocar um processador de cálculos em ponto flutuante por "hardware" junto ao microprocessador, para melhorar o desempenho da execução dos cálculos efetuados pelas subrotinas aritméticas em ponto flutuante.

A comunicação entre o microprocessador e as variáveis analógicas do computador EAI-680 é feita por intermédio da interface EAI-693 do sistema híbrido EAI-690, contendo vários conversores analógicos-digitais.

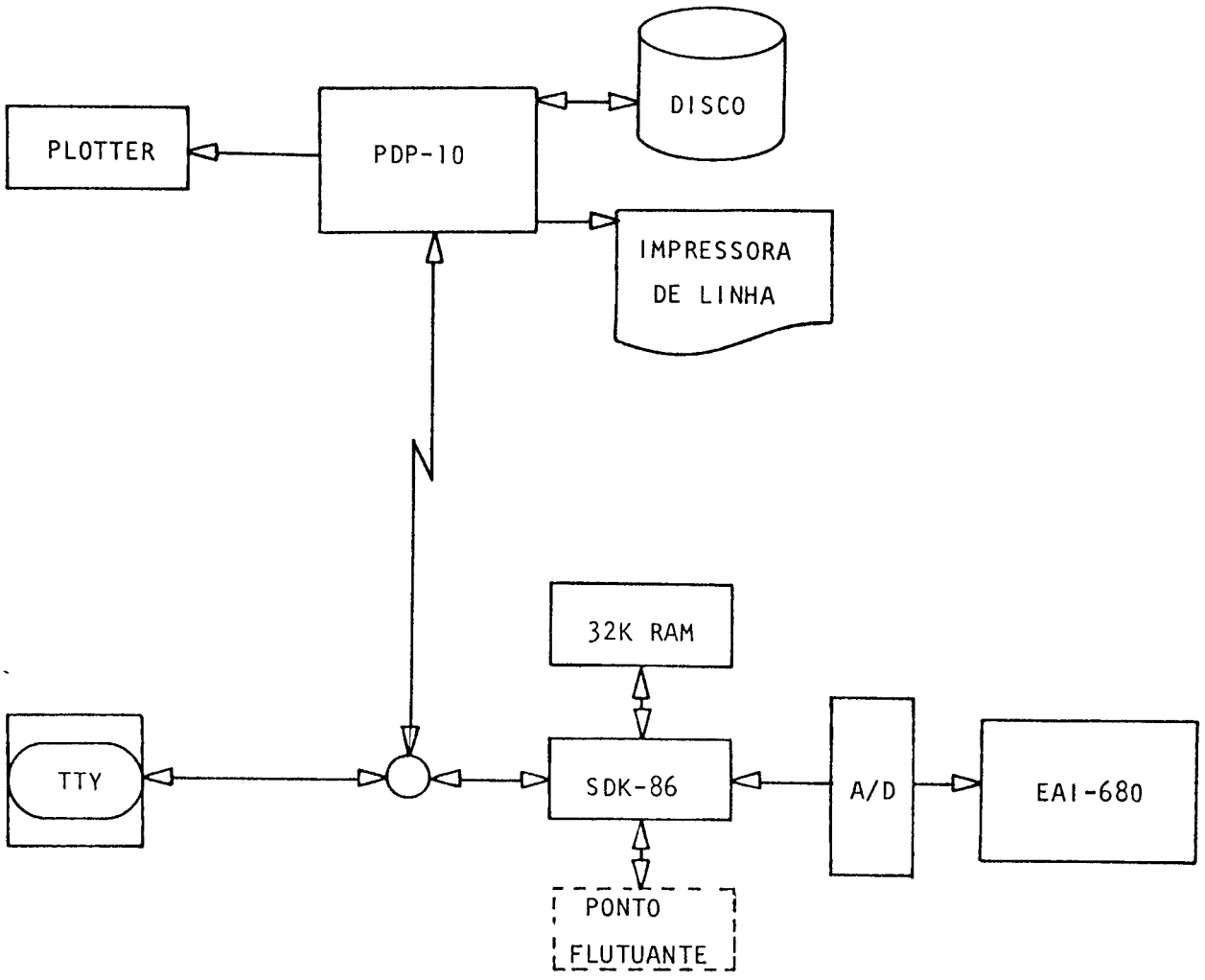


FIG 3.1 - Configuração do Sistema de Desenvolvimento

3.3.2. Configuração do Microcomputador

O microcomputador utilizado neste trabalho é baseado no kit de desenvolvimento SDK-86 |25| produzido pela Intel Corp.. Utiliza o microprocessador de 16 bits 8086 trabalhando a 2.5 MHz como unidade central de processamento selecionada na configuração de "hardware" mínima. Um diagrama de blocos funcional do kit é apresentado na figura 3.2. Possui 8 k bytes de memória ROM pré-programada com programas monitores fornecendo ferramentas básicas de software. A sua memória RAM pode ser expandida até 4 k bytes, utilizando soquetes da placa do SDK-86. Para a configuração do sistema de desenvolvimento, foram adicionadas duas placas de 16 bytes cada, totalizando 32kbytes de memória RAM. O SDK-86 possui três possibilidades de E/S: i) via interface seriada, ii) via interface paralela e iii) via conjunto teclado/"display" fixado à placa do kit de desenvolvimento.

O "software" básico que o acompanha permite a execução das seguintes operações no microcomputador, tanto por comando da interface seriada como pelo conjunto teclado/"display":

- examinar e modificar registros do 8086,
- examinar e modificar posições de memória,
- executar programas ou subrotinas do usuário,
- depurar programas utilizando recursos de execução passo-a-passo ou com "breakpoint",
- mover blocos de dados na memória,
- ler ou escrever dados pelas portas de E/S,
- carregar na memória arquivos objeto hexadecimais utilizando a E/S seriada, e
- dar saída a blocos de memória no formato objeto hexadecimal via E/S seriada.

As especificações gerais do microprocessador 8086 podem ser encontradas em |26|

As suas principais características relevantes para o bom desempenho dos algoritmos de controle adaptativos são: velocidade, modos de endereçamento e instruções aritméticas, descritas a seguir.

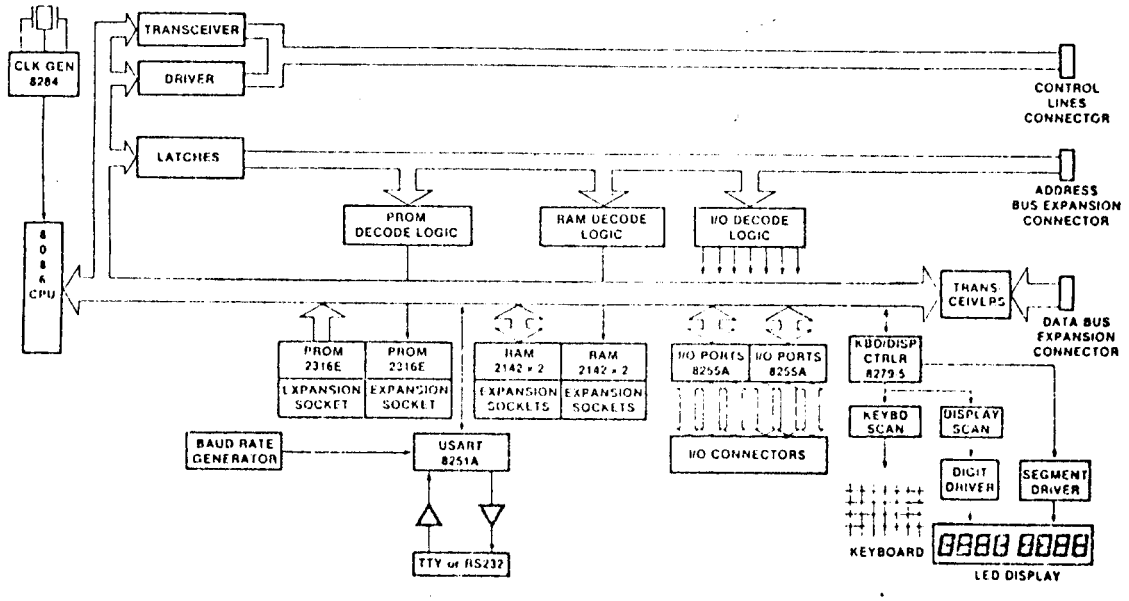


FIG 3.2 - Diagrama de Blocos do SDK-86
(extraído de |25|)

a) velocidade: o 8086 usa tecnologia H-MOS, permitindo trabalhar com ciclos de relógio de até 125 μs. Além da velocidade de funcionamento, sua arquitetura possui uma unidade de interface com o barramento que opera em paralelo com a unidade de execução, fazendo inclusive buscas de instruções antes de requerê-las. Com isso, o 8086 maximiza o tempo de utilização do barramento.

b) modos de endereçamento: o conjunto de instruções do 8086 permite várias maneiras diferentes de acessar operandos. A maioria das instruções com dois operandos permitem um registro ou uma posição de memória servindo como um operando, e um registro ou uma constante imediata servindo como o outro operando. Os operandos na memória podem ser acessados diretamente com um deslocamento de 16 bits ou indiretamente através de registros base (BX ou BP) e/ou registros índice (SI ou DI) adicionados a deslocamentos constantes de 8 ou 16 bits. O resultado das operações com dois operandos podem residir em qualquer um deles à exceção, é claro da constante imediata. As operações com um único operando são igualmente aplicáveis à memória e aos registros. Praticamente todas as instruções podem ser especificadas para operandos de 8 ou 16 bits.

c) instruções aritméticas: O 8086 permite diversas variações das quatro operações aritméticas básicas (soma, subtração, multiplicação e divisão). Pode operar em aritmética com ou sem sinal. Desta forma, adição e subtração podem ser operações com números naturais ou inteiros, dependendo do "flag" usado para testar os resultados das operações. O teste de "overflow" também faz parte de seu jogo de instruções, permitindo assim subrotinas em ponto flutuante simples e rápidas.

3.3.3. Interface com Computador Analógico do Sistema EAI

O computador analógico é usado para simular um processo a ser controlado pelo microcomputador na verificação do comportamento em tempo real do algoritmo neste implementado. A seguir é descrita a configuração do sistema EAI e a interface entre o microcomputador e o conjunto analógico.

a) Configuração do Sistema EAI-690

O sistema híbrido EAI constitui de três unidades distintas:

- EAI-680 - forma o computador analógico
- EAI-640 - forma o computador digital
- EAI-693 - constitue a interface entre o computador analógico e o digital

As operações que podem ser executadas pelo computador digital sobre o analógico estão listados na figura 3.3. Essas instruções são chamadas de híbridas e são responsáveis pela interação entre o computador digital e o analógico.

Maiores informações em relação ao sistema EAI-690 podem ser encontradas em |27| e |28|.

b) Interface entre o microcomputador 8086 e o sistema EAI

O 8086 está colocado de forma a substituir o computador digital EAI-640. Assim todas as suas operações híbridas podem ser executadas pelo microprocessador.

Para a ligação deste com o EAI-693 (interface analógica-digital), foram usadas as duas portas paralelas programáveis 8255 do kit SDK-86 e a sua configuração está mostrada na figura 3.4.

Existem um canal de dados bidirecional de 16 bits e uma via de endereço de 8 bits. Os sinais de controle são assim definidos:

- DOL, DIL, DFL e SIL: servem para indicar um dos quatro tipos de operações: DO, DI, DF ou SI da figura 3.3.
- DRL: é um sinal de reconhecimento de uma das quatro operações executadas acima.
- CLEAR: para inicializar a interface.

Para tratamento das operações de E/S analógica utilizadas neste trabalho foi desenvolvido um conjunto de subrotinas específicas à comunicação entre o microprocessador e o computador analógica. A função dessas subrotinas se encontram na tabela 3.1., e suas listagens estão em |20|.

Octal	Instruction	603 Steering Code	Function
002040	DI,'40	0	Read Digital Timer
002041	DI,'41	1	Read DVM
002042	DI,'42	2	Read Sense Lines
002043	DI,'43	3	Read Fault Word
002044	DI,'44	4	Read Analog Component Address
002045	DI,'45	5	Read Comparator
002046	DI,'46	6	Read Status Word
002053	DI,'53	11	Read Console Selection
003040	DO,'40	0	Suspend Interrupts
003044	DO,'44	4	Select Analog Component
003045	DO,'45	5	Transfer Analog Value
003046	DO,'46	6	Restore Interrupts
003047	DO,'47	7	Analog Mode Control
003050	DO,'50	8	Set Time Constant - Fine
003051	DO,'51	9	Set Time Constant - Coarse
003052	DO,'52	10	Logic Mode Control
003053	DO,'53	11	Set Console and Disable Interrupts
003054	DO,'54	12	Reset Console and Enable Interrupts
003055	DO,'55	13	Set Potentiometer
003056	DO,'56	14	DVM Convert
004040	SI,'40	0	Test Sense Line 0
004041	SI,'41	1	Test Sense Line 1
004042	SI,'42	2	Test Sense Line 2
004043	SI,'43	3	Test Sense Line 3
004044	SI,'44	4	Test Sense Line 4
004045	SI,'45	5	Test Sense Line 5
004046	SI,'46	6	Test Sense Line 6
004047	SI,'47	7	Test Sense Line 7
005040	DF,'40	0	Preset Digital Timer
005041	DF,'41	1	Set Operation Control Lines
005042	DF,'42	2	Jam Operation Control Lines
005043	DF,'43	3	Reset Operation Control Lines
005044	DF,'44	4	Set Function Relay
005045	DF,'45	5	Reset Function Relay
004050	SI,'50	8	Test GP Interrupt 0
004051	SI,'51	9	Test GP Interrupt 1
004052	SI,'52	10	Test GP Interrupt 2
004053	SI,'53	11	Test GP Interrupt 3
004054	SI,'54	12	Test GP Interrupt 4
004055	SI,'55	13	Test GP Interrupt 5
004056	SI,'56	14	Test GP Interrupt 6
004057	SI,'57	15	Test GP Interrupt 7

FIG 3.3 - Operações Híbridas (continua)

Octal	Instructions	693 Steering Code	Function
002064	DI,'64	4	Read ADC, Increment to Successive Channel and Convert
002065	DI,'65	5	Read ADC
003040	DO,'40	0	Suspend Interrupts
003046	DO,'46	6	Restore Interrupts
003060	DO,'60	0	Load DAC/DAM and Increment to Successive Channel
003140 thru 003167	DO,'140 thru DO,'167	0 thru 15	Select and Load a Specific DAC/DAM channel.
004064	SI,'64	4	Read MUX Address
005060	DF,'60	0	Select Initial DAM/DAC Channel
005061	DF,'61	1	Set *Jam* Mode
005062	DF,'62	2	Reset *Jam* Mode
005063	DF,'63	3	Transfer DAC/DAM Channels
005064	DF,'64	4	Select MUX Channel and Convert
005065	DF,'65	5	ADC Convert

FIG 3.3 - Operações Híbridas (continuação)
(extraído de |28|)

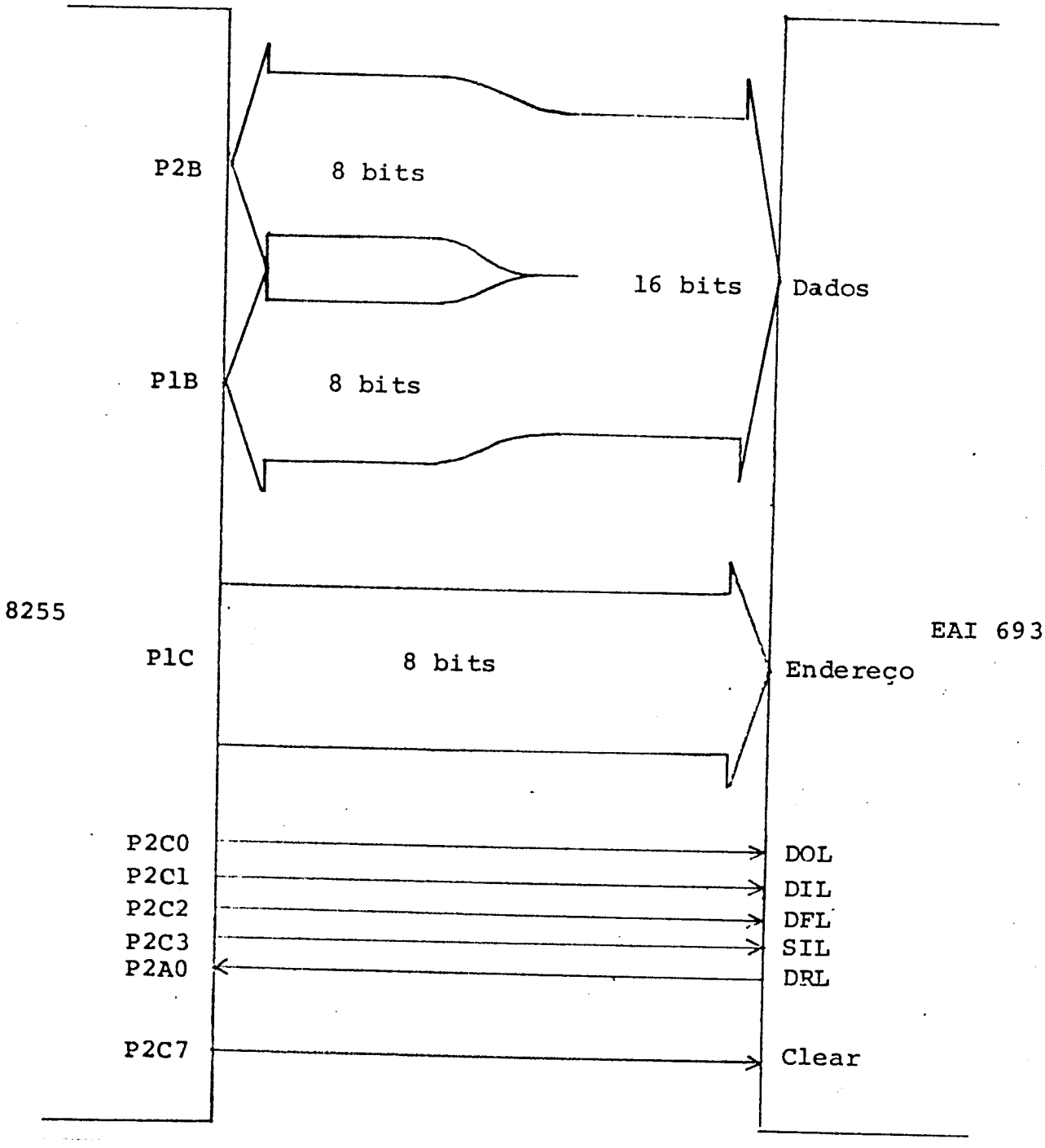


FIG 3.4 - Ligação entre interface paralela do SDK-86 (8255) com a interface analógica EAI-693

Nome	Função
EIN	Simula a instrução SI ou DI do Computador EAI-640, isto é, simula instruções de entrada de dados do computador analógico
EOUT	Simula a instrução DF ou DO do Computador EAI-640, isto é, simula as instruções de saída de dados para o computador analógico
INIC	Inicializa interface analógica e conversores analógicos-digitais e digitais-analógicos
CONVE	Converte e lê dados dos conversores analógicos-digitais
OUTP	Dá saída de dados nos conversores digitais-analógicos

Tabela 3.1 - Função das subrotinas responsáveis pela interface do microcomputador e computador analógico.

3.4. IMPLEMENTAÇÃO DO ALGORITMO AUTO-AJUSTÁVEL MULTIVARIÁVEL NO MICROPROCESSADOR

O procedimento para o desenvolvimento do algoritmo auto-ajustável multivariável no microcomputador consiste das seguintes etapas:

- Desenvolvimento do algoritmo no PDP-10, usando a linguagem FORTRAN
- Teste sobre uma simulação digital do processo
- Implementação do algoritmo no microcomputador, usando a linguagem desenvolvida para este fim. (Capítulo 2).
- Teste da implementação sobre a simulação digital e comparação dos resultados
- Teste e verificação do comportamento do algoritmo nas condições de tempo real, controlando o processo analógico simulado no EAI-680.

As listagens em Fortran e na linguagem de alto nível desenvolvida, da subrotina do controlador são mostradas nas figuras 3.5 e 3.6 respectivamente. Através delas é possível tecer algumas considerações:

- o número de linhas dos programas é aproximadamente o mesmo, sendo seu comprimento maior na linguagem desenvolvida.
- o comando DO do Fortran é relacionado ao comando FOR sendo o seu campo de atuação delimitado pelos caracteres < e >.
- a declaração de variáveis em COMMON feita em Fortran tem correspondência com a declaração de variáveis globais feitas no arquivo UNIVERSAL MIMO (fig. 3.7) e referenciadas na subrotina pelo comando SEARCH MIMO
- às variáveis formais da subrotina na linguagem de alto nível utilizada foi acrescentado o caracter $ para diferenciá-las visualmente das outras variáveis.

A respeito da listagem do programa nessa linguagem, alguns comentários podem ainda ser feitos:

O programa começa com o comando de SEARCH, que busca o arquivo auxiliar ao processador de macros e também faz referência ao arquivo MIMO onde são declaradas as variáveis globais usadas. A seguir o comando BEGIN busca todos os outros arquivos que constituem o compilador: o arquivo que define a linguagem, o que traduz em código intermediário e o que traduz em linguagem sim-

```
C
C***   ATUALIZACAO DO GANHO E BAIXA ORDENADA DA MATRIZ DE COV
C
1.      SIG=FORGET
        SIGSQ=FORGET*FORGET
        IJ=0
        JI=0

        DO 100 J=1,NPAR
        J1=J-1
        FJ=0.0
        DO 20 I=1,J
        JI=J1+1
2.      FJ=FJ+PCOV(JI)*DATA(I)
        A=SIG/FORGET
        B=FJ/SIGSQ
        SIGSQ=SIGSQ+FJ*FJ
        SIG=SQRT(SIGSQ)
        A=A/SIG
        GANHO(J)=PCOV(JI)*FJ
        PCOV(JI)=A*PCOV(JI)
        IF(J1.EQ.0)GOTO 31
        DO 30 I=1,J1
        IJ=IJ+1
        SQP=PCOV(IJ)
        PCOV(IJ)=A*(SQP-B*GANHO(I))
        GANHO(I)=GANHO(I)+SQP*FJ
3.      CONTINUE
31      CONTINUE
        IJ=IJ+1
100     CONTINUE
C
```

FIG 3.5 - Listagem Fortran de um Trecho do Algoritmo Auto-Ajustável

```
; * * *   ATUALIZA O GANHO E RAIZ QUADRADA DA MATRIZ DE COV
L T SIG,FORGET
L T SIGSQ,OPER(FORGET,MUL,FORGET)

L T J1,0
L T J1,0

F R 0,1,NPAR,<
  L T J1,OPER(J1,ADD,1)
  L T FJ,1
  F R I,1,J,<
    INCR J1
    L T FJ,OPER(FJ,ADD,OPER(MAT(PCOV,J1),MUL,MAT(DATA,I)))>
  L T A,OPER(SIG,DIV,FORGET)
  L T B,OPER(FJ,DIV,SIGSQ)
  L T SIGSQ,OPER(SIGSQ,ADD,OPER(FJ,MUL,FJ))
  C LL AUXT,<SIG,SIGSQ>
  L T A,OPER(A,DIV,SIG)
  L T MAT(GANHO,J),OPER(MAT(PCOV,J1),MUL,FJ)
  L T MAT(PCOV,J1),OPER(A,MUL,MAT(PCOV,J1))

  IF UZ,OPER(J1,SUB,0),<
    FOR I,1,J1,<
      INCR IJ
      LET SOP,MAT(PCOV,IJ)
      LET MAT(PCOV,IJ),OPER(A,MUL,OPER(SOP,SUB,OPER(B,MUL,MAT(GANHO,I))))
      LET MAT(GANHO,I),OPER(MAT(GANHO,I),ADD,OPER(SOP,MUL,FJ))
            >
          >
  INCR IJ
        >
```

FIG 3.6 - Listagem na Linguagem de Alto Nível Desenvolvida de um Trecho do Algoritmo Auto-Ajustável

```
;ARQUIVO MIMO4.MAC: DECLARACAO DAS VARIAVEIS GLOBAIS

UNIVERSAL MIMO
SEARCH CONCAT
BEGIN
ORIGEM #1000

FLOAT <F1,F2,SIG,SIGSQ,PORCENT,A,B,SOP,PLIP,[illegible],FJ,Y,U,W,PHI>

FLOAT <<DIM(YPAST,3,7)>,<DIM(UPAST,3,7)>,<DIM(WPAST,3,7)>>
FLOAT <<DIM(DATA,45)>,<DIM(PORTY,1[illegible]>,<DIM(GAMMA,45)>>
FLOAT <<DIM(THETA,45,3)>,<DIM(XDMODA,3)>>

BYTE <B1,B2,B5,B6,F7,FS1,BAUX1,BAUX2,BAUX3,BAUX4,BAUX5,BYP2,BWP2,BDIM,IDC>
BYTE <ND,KK,NU,NY,NN,SPAR,I,J,IJ,JI,JJ,IA,K>

COMMON
END
```

FIG 3.7 - Declaração das Variáveis Globais

bólica. Os próximos comandos já tem a sua equivalência na listagem Fortran . No final, o comando BIBLOF chama endereços das subrotinas utilitárias usadas neste programas. O comando COMPIL informa ao compilador o final do programa fonte e o END finaliza o arquivo para o MACRO-10.

O teste e a depuração do programa é feito no próprio microcomputador utilizando as facilidades do seu monitor, descritas na seção 3.3.2.

As listagens completas do algoritmo se encontram em |20|.

O algoritmo auto-ajustável multivariável implementado pode controlar um sistema com até tres entradas, tres saídas, até quarta ordem.

Nesta configuração, a distribuição de memória é:

- sistema operacional do controlador - 2.5 k bytes
  incluindo: operações ponto flutuante
  conversões - decimal - binária
  subrotinas de E/S com processo e operador
- controlador - 2.5 k bytes

totalizando 5 k bytes que podem ser colocados em memórias EPROM.

Para a aplicação em um determinado processo industrial, a memória de dados, assim como o tempo de execução desse algoritmo dependerá praticamente da ordem da matriz de parâmetros a estimar. Na próxima seção são mostrados os valores de memória de dados e tempo de execução para os exemplos simulados.

3.5. EXEMPLOS SIMULADOS

Com a finalidade de ilustração da implementação do algoritmo no microcomputador, dois exemplos são simulados. O primeiro, realizado digitalmente, é simulado no PDP-10 utilizando um pacote de subrotinas Fortran desenvolvido por |11|. A comunicação entre o processo simulado no PDP-10 e o controlador implementado se dá pelo "link" descrito na seção 3.3.1. Deste modo é possível a comparação dos desempenhos dos controladores implantados no 8086 e no PDP-10, pois o processo pode ser simulado várias vezes com as mesmas sequências de ruído.

A segunda simulação descrita nesta seção é feita no computador analógico, verificando-se assim o desempenho do controlador em situações de tempo real.

Finalizando, são apresentados os tempos de execução e memória de dados utilizada em cada exemplo.

3.5.1. Exemplo 1

O processo simulado no PDP-10 tem a estrutura indicada pela equação (1) onde:

$$A_1 = \begin{vmatrix} 0.8 & -2.9 \\ 0.4 & -1.4 \end{vmatrix} \qquad A_2 = \begin{vmatrix} 0.2 & -1.75 \\ 0.2 & -0.95 \end{vmatrix}$$

$$B_0 = \begin{vmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{vmatrix} \qquad B_1 = \begin{vmatrix} -0.4 & 0.6 \\ 0.4 & 0.6 \end{vmatrix}$$

$$C_1 = \begin{vmatrix} 1.4 & 0.0 \\ 1.0 & -1.2 \end{vmatrix} \qquad C_2 = \begin{vmatrix} 0.49 & 0.0 \\ 1.0 & 0.36 \end{vmatrix}$$

$$\underline{d} = \begin{vmatrix} 0.0 \\ 0.0 \end{vmatrix} \qquad R = \begin{vmatrix} 0.1 & 0.1 \\ 0.1 & 0.2 \end{vmatrix} \qquad k = 1$$

O sistema é de segunda ordem, estável e de fase mínima. O processo deve ser controlado de forma a seguir um "set-point" $\underline{w}(t)$ dado por:

$$\underline{w}(t) = 10.0 \begin{vmatrix} \text{sen } \frac{\pi t}{50} \\ \text{sen } \frac{\pi t}{40} \end{vmatrix}$$

O controlador usa o valor correto do atraso e estima duas linhas de parâmetros, com 14 parâmetros cada. O fator de esquecimento é igual a 0.995 e não se utiliza ponderação no controle.

A matriz de parâmetros é inicializada com:

$$\theta = \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -1.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

e a matriz raiz quadrada da matriz de covariança com

$$S = \text{diag.} \begin{bmatrix} 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 0.0 & 0.0 & 1.0 & 1.0 & 1.0 \end{bmatrix}$$

O comportamento das saídas y_1 (t) e y_2 (t) é mostrado nos gráficos das figuras 3.8 e 3.9 respectivamente. Eles indicam os 100 primeiros passos da realização. As senóides apresentadas são relativas aos "set-points".

As curvas de y_1 (t) e y_2 (t) referentes aos resultados obtidos controlando-se o processo pelo computador PDP-10 e pelo microcomputador SDK - 86 são praticamente coincidentes, não podendo ser possível distinguí-las nessas figuras.

Na figura 3.10 são plotados os custos acumulados dados pela e - quação

$$C(t) \sum_{i=1}^{t} \left[\underline{y} \ (t) - \underline{w}(t-k) \right]^T \left[\underline{y} \ (t) - w \ (t-k) \right]$$

referentes as realizações feitas no PDP-10 e no microcomputador e o custo acumulado se o sistema fosse controlado com seus parâmetros conhecidos. Neste gráfico são apresentados os primeiros 1.000 passos da simulação. Verifica-se um comportamento idêntico para o processo controlado pelo algoritmo programado no PDP-10 ou no microcomputador SDK-86. Por outro lado o algoritmo auto - ajustável apresenta, após a fase transitória devida à inicialização, um custo mínimo estatisticamente não diferente do custo mínimo teórico que poderia ser obtido se os coeficientes fossem conhecidos.

As curvas obtidas com o controlador no PDP-10 e no microcomputador começam a se destacar em torno do passo 700, mostrando assim que os erros de cálculo numérico apresentados pelo 8086 são desprezíveis em relação aos er - ros apresentados pelo PDP-10.

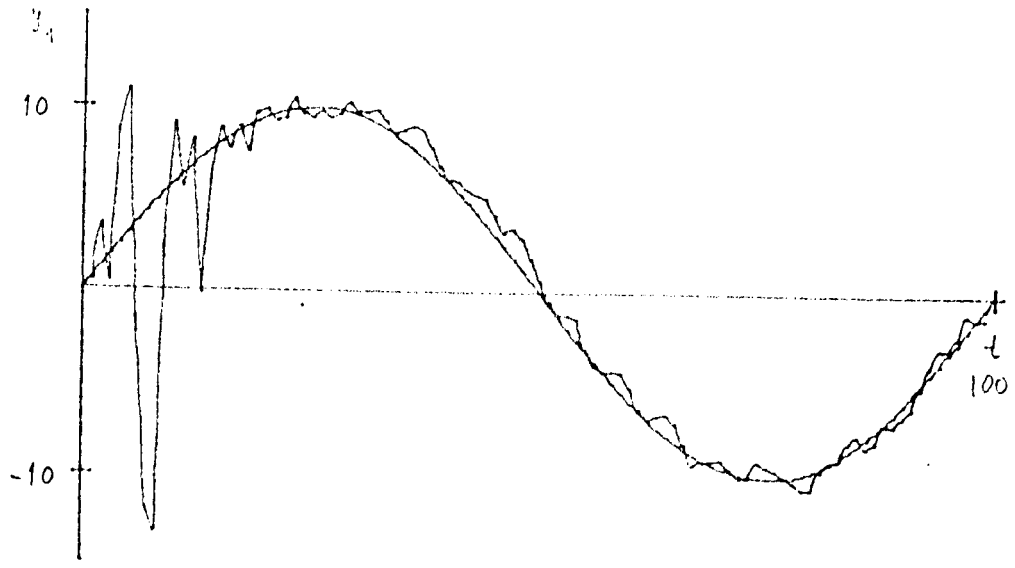


FIG 3.8 - Gráfico da Saída $y_1(t)$ do exemplo 1

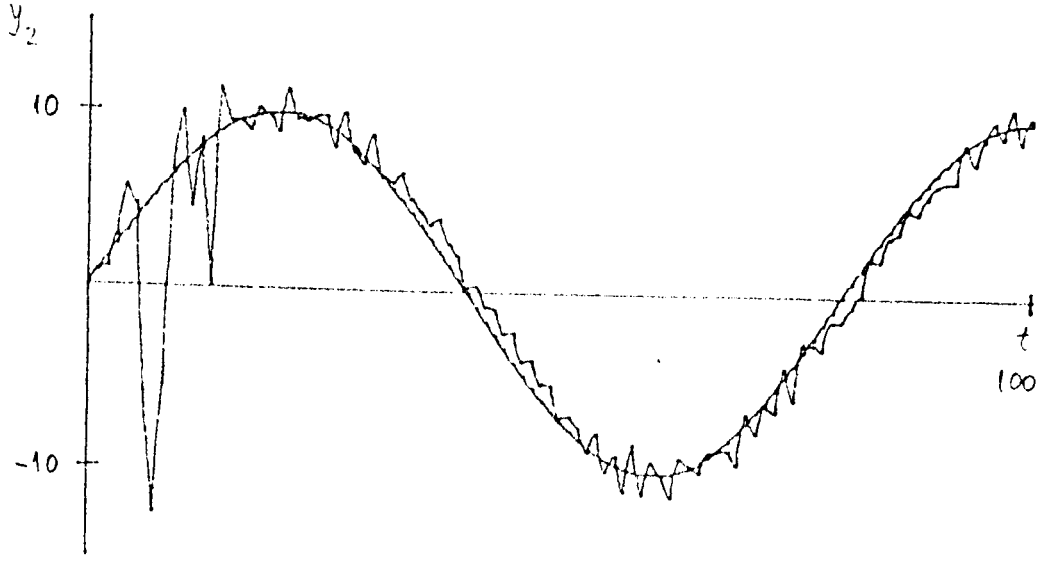


FIG 3.9 - Gráfico da Saída $y_2(t)$ do exemplo 1

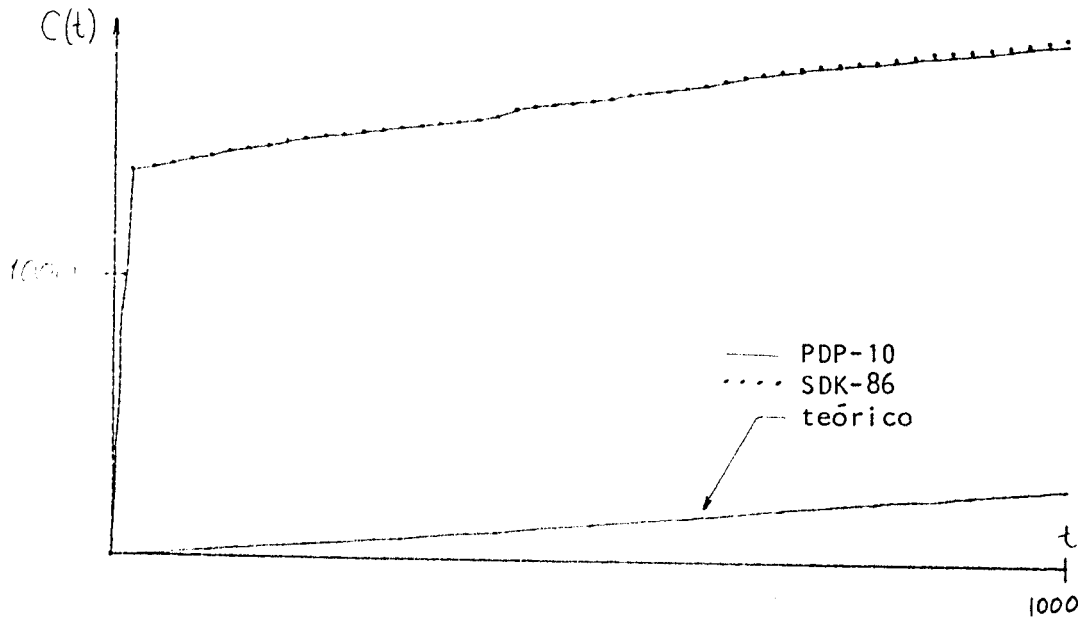


FIG 3.10 - Custo Acumulado. Exemplo 1

3.5.2. Exemplo 2

Com finalidade ilustrativa, neste exemplo o processo é simulado no computador analógico e controlado pelo microcomputador em tempo real.

A simulação do processo está apresentada na figura 3.11. Como fonte de ruído é usado o gerador de ruídos HP modelo 3722A que produz uma saída gaussiana com densidade espectral constante até 5 H_Z. A partir do ruído r, conseguem-se os sinais r_1 e r_2 passando-os pelos filtros de primeira ordem $H_1(s)$ e $H_2(s)$ respectivamente. Esses dois sinais são apresentados nas figs. 3.12 e 3.13. Eles são coincidentes com os sinais de saída y_1 e y_2 fazendo-se as entradas u_1 e u_2 iguais a zero.

O controlador é utilizado como regulador, isto é, procura manter as saídas do processo o mais próximo possível de zero. O intervalo de amostragem utilizado é de um segundo.

A estrutura do controlador foi colocada de forma a controlar um sistema com atraso dois e ordem susposta também igual a dois, estimando-se uma matriz de parâmetro 2 x 14. Cabe observar que a estrutura escolhida não é a estrutura do processo real, pois como ruído r não é branco, a ordem do sistema é desconhecida.

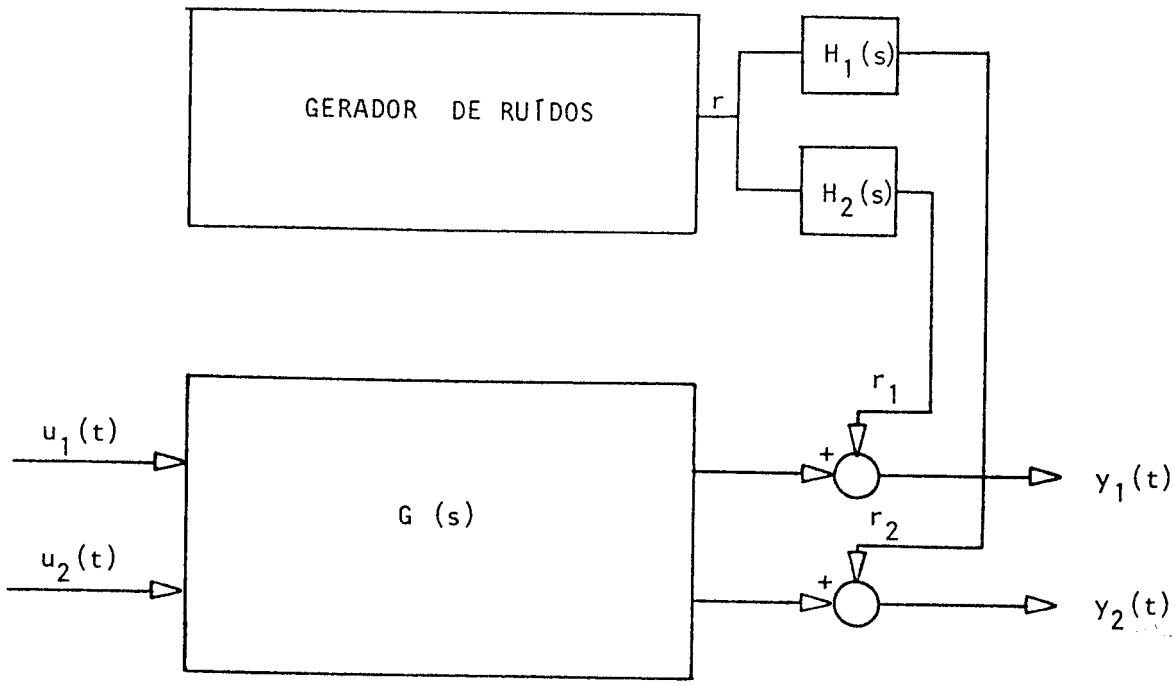
A inicialização desse controlador, quando atua como regulador, exige uma escolha da matriz B_0 segundo certos critérios para não levar o sistema à instabilidade. Esses critérios são descritos em |11|. Neste exemplo, B_0 foi colocado com:

$$B_0 = \begin{vmatrix} 5.0 & 0.0 \\ 0.0 & 10.0 \end{vmatrix} .$$

o qual é mantido constante. Os demais parâmetros são inicializados com zero e variância 1.0.

O fator de esquecimento é colocado com 0.995 e a matriz de ponderação no controle é deixada em zero.

As saídas y_1 e y_2 do processo regulado são mostradas nas figuras 3.14 e 3.15. Pelas figuras nota-se claramente um período transitório, seguido



$$G(s) = \begin{vmatrix} \frac{0.1}{s + 0.1} & \frac{0.1}{s + 0.1} \\ -\frac{0.25}{s + 0.2} & \frac{0.75}{s + 0.2} \end{vmatrix}$$

$$H_1(s) = \frac{1}{s + 0.5}$$

$$H_2(s) = \frac{1}{s + 1}$$

FIG 3.11 - Simulação do Processo do exemplo 2

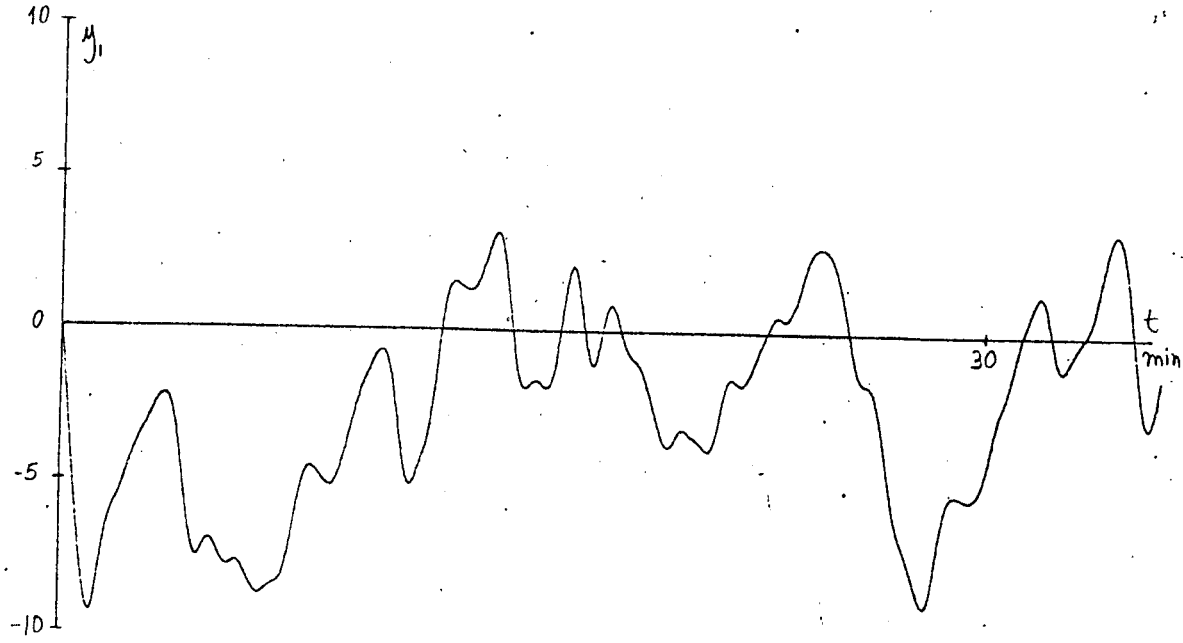


FIG 3.12 - Saída $y_1(t)$ sem controle ($\underline{u}=\underline{0}$). Exemplo 2

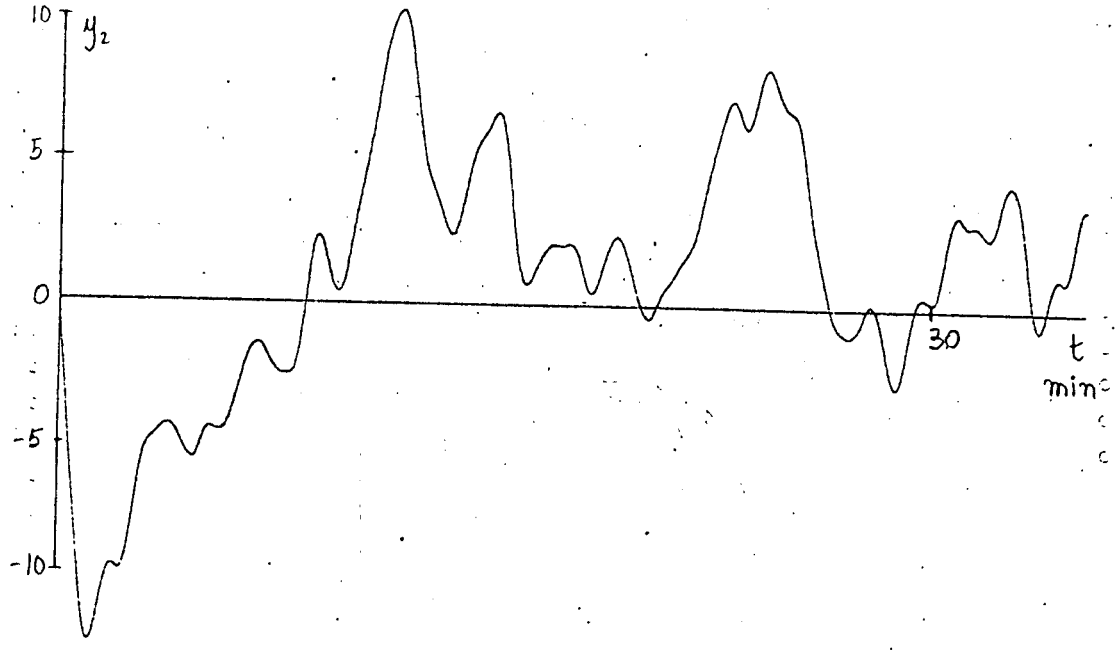


FIG 3.13 - Saída $y_2(t)$ sem controle ($\underline{u}=\underline{0}$). Exemplo 2

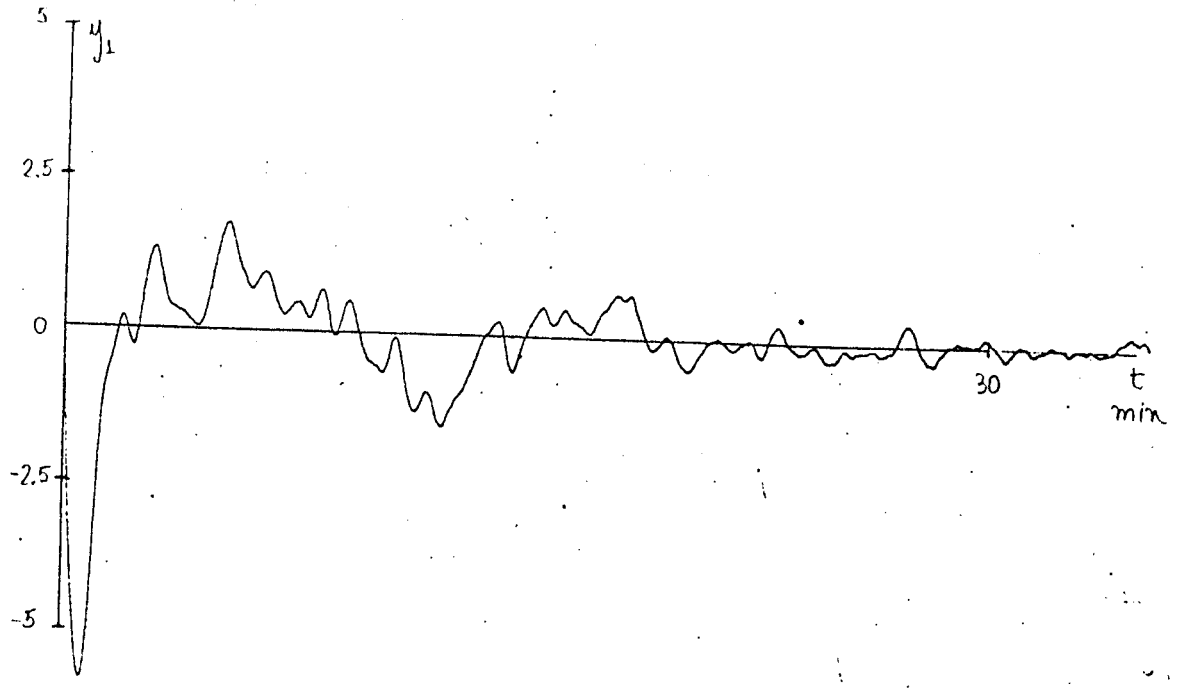


FIG 3.14 - Saída $y_1(t)$ com controlador auto-ajustável. Exemplo 2

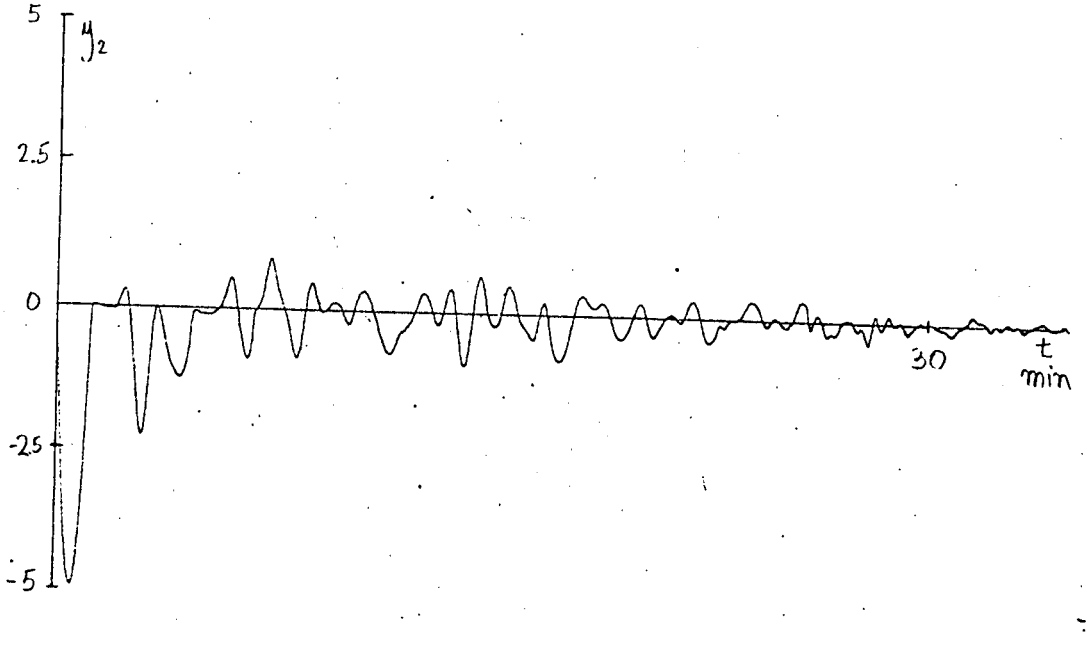


FIG 3.15 - Saída $y_2(t)$ com controlador auto-ajustável. Exemplo 2

por uma fase de regulação satisfatória do processo.

A memória de dados e o tempo de execução dos dois exemplos simulados são os mesmos, pois para ambos é estimada uma matriz de parâmetros de ordem 2 x 14. A memória de dados utilizada é de 822 bytes e o tempo de execução de um ciclo do algoritmo é de 0.4 segundos.

3.6. CONCLUSÃO

O sistema de desenvolvimento utilizado para a implementação do algoritmo auto-ajustável multivariável mostrou-se satisfatório, permitindo sem grandes dificuldades a colocação no microcomputador SDK-86 de um programa para controle de processos já testado em Fortran no computador PDP-10.

Dos exemplos simulados podemos tirar algumas conclusões. Com o primeiro exemplo comprovou-se que o algoritmo implementado no SDK-86 se comporta como o implementado no PDP-10 e que a representação dos números em ponto flutuante é viável para a estimação de parâmetros utilizada neste algoritmo. Com o segundo exemplo, procurou-se controlar um processo mais real, no qual o seu comportamento não é totalmente conhecido, o que é comum nos processos industriais. Como resultado, o processo foi facilmente controlado após terem sido feitas algumas realizações com inicializações diferentes no controlador. Com esse exemplo, testou-se também a ligação feita entre o microcomputador e o computador analógico.

A quantidade de memória necessária para os exemplos, assim como o tempo de execução do algoritmo comprova a viabilidade de uso desse controlador em processos industriais. Certamente deverão ser adicionados dispositivos de segurança para casos de falha ou mesmo durante a inicialização do algoritmo , possibilitando chaveamento para controle manual ou para algoritmo P.I.D. , etc..

CAPÍTULO 4

CONCLUSÕES FINAIS E

SUGESTÕES PARA TRABALHOS FUTUROS

4. CONCLUSÕES E SUGESTÕES PARA TRABALHOS FUTUROS

A realização de controladores auto-ajustáveis multivariáveis usando-se a tecnologia de microprocessadores é prática e economicamente viável. Por exemplo, um microcomputador baseado no 8086, com 5k bytes de memória EPROM, 5 K bytes de memória RAM, unidade de entrada e saída analógica e opcionalmente um processador ponto flutuante por "hardware", permite implementar um controlador adaptativo suficientemente desenvolvido para atuar numa grande diversidade de processos.

O sistema de desenvolvimento apresentado consiste de: i) o computador PDP-10 para o desenvolvimento de "software" básico do microcomputador e para a simulação digital do processo a ser controlado, ii) o computador analógico EAI-680 para a simulação analógica em tempo real, e iii) o microcomputador baseado no 8086 da Intel.

O sistema permite o desenvolvimento e testes de algoritmos de controle adaptativo de forma satisfatória, possibilitando também a verificação do comportamento do controlador em situações tempo real.

O método usado no desenvolvimento de "software" é perfeitamente viável para o algoritmo implementado, e apesar do sistema de desenvolvimento ter sido feito para a realização de uma aplicação específica, ele pode ser utilizado para a implementação de outros algoritmos de controle e no desenvolvimento de programas aplicativos em outras áreas que utilizam microprocessadores.

A estrutura modular utilizada no processo de compilação permite incluir a tradução de novas linguagens de alto nível de uso específico, utilizando-se o código intermediário na geração da linguagem simbólica de vários microprocessadores. A partir dela, o montador generalizado cria o código de máquina referente a cada microcomputador.

O tradutor baseado em processador de macros possui um tempo de compilação da ordem de uma linha por segundo, o que chega a ser mais de uma ordem de grandeza maior que o tempo de compilação usando-se os métodos convencionais. O uso dessa técnica portanto, não deve ser empregado em compiladores usados para produção extensiva de programas.

Como sugestões para próximos trabalhos pode-se citar:

- escolha da estrutura de um controlador auto-ajustável para uso industrial visando: - comunicação homem-máquina
  - inclusão de algoritmos convencionais como P.I.D.,etc.;
- especificação de uma linguagem de controle com diversos aspectos de tempo real;
- otimização do código gerado pelo tradutor baseado em processador de macros;
- estudo do código intermediário aplicável na tradução de diversas linguagens para diversos microprocessadores;
- extensão do montador generalizado, permitindo:
  - o uso de mnemônicos sugeridos pelo fabricante
  - geração de código relocável
  - declaração de variáveis externas.

Alguns desses temas estão sendo atualmente estudados |21|e|29| dando seassim à linha de trabalho adotada pelo laboratório de micro e minicomputadores da FEC.

CAPÍTULO 5

BIBLIOGRAFIA

5. BIBLIOGRAFIA

| 1 | I. M. Watson - "Comparison of Commercially Available Software Tools for Microprocessors Programming". Proceedings of the IEEE, vol. 64, nº 6, p.910-920, 1976.

| 2 | M. F. Magalhães - Cross-Assembler Generalizado para Microcomputadores de 4,8,12 e 16 bits. Tese de Mestrado. FEC-UNICAMP, 1979.

| 3 | J. R. Oliveira - SIMLA- Um Sistema Generalizado de Simuladores de Microcomputadores. Tese de Mestrado. FEC-UNICAMP, 1979.

| 4 | A. S. Tanenbaum - "A General Purpose Macroprocessor as a Poor Man's Compiler-Compiler". Software Engineering, vol. SE-2, nº 2 , p.121-125, 1976.

| 5 | E. F. Elsworth - "Compilation via an Intermediate Language". The Computer Journal, vol 22, nº 3, p.226-233, 1979.

| 6 | K. J. Astrom - Introduction to Stochastic Control Theory. Academic Press, New York, 1970.

| 7 | D. W. Clarke,S. N. Cope e P. J. Gawthrop - "Feasibility Study of the Application of Microprocessors to Self-Tuning Controllers". OUEL Report 1135/75. Oxford, 1975.

| 8 | T. Cegrell e T. Hedovist - "Succesful Adaptive Control of Paper Machines" IFAC Symposium on Identification and System Parameter Estimation. Hague, 1973.

| 9 | H. E. Horn - "Feasibility Study of The Application of Self Tuning Controllers to Chemical Batch Reactors". Report nº 1248/78. University of Oxford, 1978.

| 10 | C. G. Kallstrom, K. J. Astrom et al. - "Adaptive Autopilots for Large Tankers". 7th IFAC Congress. Helsinki, 1978.

|11| R. C. Machado - Desenvolvimento de Algoritmos de Controle Adaptativo para Implementação em Microcomputadores. Tese de Mestrado a ser publicada. FEC-UNICAMP, 1981.

|12| C. N. Mooers - "TRAC, A Procedure Describing Language for the Reactive Typewriter". Comm. ACM, vol. 9, nº 3, p.215-219, 1966.

|13| C. Strachey - "A General Purpose Macrogenerator". Computer Journal,vol 8 nº 3, p.225-241, 1965.

|14| P. J. Brown - "The ML/I Macro Processor". Comm. ACM, vol. 10, p.618-623, 1979.

|15| P. J. Brown - "Using a Macroprocessor to Aid Software Implementation". Computer Journal, vol. 12, nº 4, p.327-331, 1968.

|16| W. M. Waite - Implementing Software for Non-Numeric Applications. Prentice Hall, 1974.

|17| R. N. Melo - PM- Um Processador de Macros. Tese de Mestrado. ITA, 1972.

|18| H. A. Cohen e R. S. Francis - "Macro-Assemblers and Macro Based Languages in Microprocessors Software Development". Computer, vol.12 nº 2, p.53-64, 1979.

|19| Digital Equipment Corp. - MACRO-10, Macro Assembler Reference Manual. Version 53. Digital Equipment Corp., 1978.

|20| R. A. Lotufo - Relatório Técnico nº 06/81. FEC-UNICAMP, 1981.

|21| A. A. A. C. P. Oliveira - Uma Linguagem de Alto Nível para Aplicação em Microprocessadores. Tese de Mestrado a ser Publicada. FEC - UNICAMP, 1981.

|22| J. T. Coonen - "An Implementation Guide to a Proposed Standard for Floating Point Arithmetic". Computer, vol. 13, nº 1, p.68-79 , 1980.

|23| W. C. Amaral - Identificação de Sistemas Multivariáveis. Tese de Doutoramento. FEC-UNICAMP, 1981.

|24| V. Strejc - "Least Square Parameter Estimation". 5th IFAC Symposium on Identification and System Parameter Estimation. Darmstadt,1979.

|25| Intel Corporation - SDK-86, MCS-86 System Design Kit, User's Guide. Intel Corp., 1978.

|26| Intel Corporation - MCS-86 User's Manual. Intel Corp., 1979.

|27| Electronics Associates Inc. - 690 Hybrid System. EAI, 1969.

|28| Electronics Associates Inc. - 693 Hybrid System Interface. EAI, 1969.

|29| N. Endo - Tese de Mestrado em Andamento. FEC-UNICAMP.