ARQUITETURA DE UM PROCESSADOR DE ARITMÉTICA INTERVALAR CONFIGURÁVEL COMO PROCESSADOR DE PRODUTO ESCALAR*

Carlos Marcelo Dias Pazos[1]
Cláudia Ma. Ribeiro Azevedo[2]
Márcia de Barros Correia[3]

SUMÁRIO

Este trabalho apresenta um Processador de Aritmética Intervalar que explora o paralelismo inerente às operações sobre intervalos. Este processador é reconfigurável para tratar operações de Produto Escalar com máxima exatidão usando técnicas da Aritmética Computacional Avançada. A implementação do processador é baseada em tecnologia Bit-Slice e microprogramação.

ABSTRACT

This paper presents an Interval Arithmetic Processor which explores the parallelism inherent to Interval Arithmetic operations. This processor can be configurated to implement the dot product operation with maximum accurancy by using Advanced Computer Arithmetic techniques. The processor implementation is based on Bit-Slice technology and microprogramming.

---

[1] Graduando em Engenharia Eletrônica na Universidade Federal de Pernambuco; áreas de interesse: Arquitetura de Computadores, Eletrônica Digital e Compiladores.

[2] Bacharel em Ciência da Computação (Universidade Federal de Pernambuco, 1989); Mestranda em Informática na Universidade Federal de Pernambuco; áreas de interesse: Arquitetura de Computadores, Sistemas Distribuídos e Processos Concorrentes.

[3] Professora Adjunta do Departamento de Informática da Universidade Federal de Pernambuco; Docteur Ingenieur (INP - Toulouse - France, 1983); áreas de interesse: Arquitetura de Computadores, Eletrônica Digital.

* Este projeto é parcialmente financiado pelo CNPq e FINEP.

1 - INTRODUÇÃO.

A Aritmética Intervalar desenvolveu-se com o intuito de permitir uma verificação automática do erro cometido em resultados obtidos através do computador. Nesta aritmética, um número real é representado por um intervalo que o contém.

O suporte ao tipo de dado intervalo pode ser feito tanto a nível de software quanto de hardware. Linguagens dedicadas ao desenvolvimento de programas numéricos, utilizando o conceito de intervalo, já são disponíveis . O PASCAL-SC [4], extensão do Pascal padrão dedicada à computação científica, é uma delas. Nele, o tipo intervalo é tratado como um registro de dois campos, o limite inferior e o limite superior. Operações elementares e funções sobre intervalos encontram-se definidas e fazem parte do repertório de instruções do PASCAL-SC.

O objetivo deste trabalho é propor um co-processador aritmético que, explorando o paralelismo inerente às operações com intervalos, dê suporte ao tipo de dado intervalo a nível de hardware.

Uma outra aritmética, a Aritmética Computacional Avançada, criou novas técnicas para atacar o problema da inexatidão em resultados fornecidos pelo computador. Dentre essas, destaca-se o cálculo do produto escalar com exatidão máxima.

A arquitetura aqui proposta permite também reconfigurar o co-processador de intervalos, transformando-o em um processador de produto escalar, que será implementado de acordo com as técnicas da Aritmética Computacional Avançada.

2 - O ASSEMBLY DO CO-PROCESSADOR

Para se fazer uso das instruções que o co-processador oferece, foi desenvolvida uma biblioteca especial no assembly do 80286.

A idéia básica é que se programe normalmente usando este assembly. Quando a instrução tiver que ser executada pelo co-processador, utiliza-se o seu nome precedido do caractere

diferenciador "_" Isso equivale à chamada de uma macro com esse nome. Essas macros se encontram na biblioteca implementada.

O co-processador é conectado ao sistema do 80286 como um periférico comum, ou seja, sua comunicação se dá através de portas. As macros que tratam as instruções do co-processador utilizam o endereço deste para se comunicar através das instruções IN e OUT.

A seguir são mostradas as instruções do co-processador:

a)Operações sobre intervalos:

```
INEG reg       ; nega o intervalo (-reg)
IINV reg       ; inverte o intervalo (1 ÷ reg)
IADD reg, op   ; adiciona dois intervalos (reg + op)
ISUB reg, op   ; subtrai dois intervalos (reg - op)
ISUBR reg,op   ; subtrai dois intervalos (op - reg)
IMUL reg, op   ; multiplica dois intervalos (reg × op)
IDIV reg, op   ; divide dois intervalos (reg ÷ op)
IDIVR reg,op   ; divide dois intervalos (op ÷ reg)
IINT reg, op   ; intersecciona dois intervalos (reg ∩ op)
IUNI reg, op   ; une dois intervalos (reg U op)
IDIS reg, op   ; calcula a distância d(reg,op)
IDISR reg,op   ; calcula a distância d(op,reg)
IMOD reg       ; calcula o valor absoluto |reg|
IDIF reg       ; calcula o diâmetro D (reg)
IEQU reg, op   ; testa se igual (reg = op)
ILES reg, op   ; testa se menor (reg < op)
IGRE reg, op   ; testa se maior (reg > op)
ILEQ reg, op   ; testa se menor ou igual (reg ≤ op)
IGEQ reg, op   ; testa se maior ou igual (reg ≥ op)
ILOA reg, op   ; realiza a atribuição reg ← op
ISAV op, reg   ; realiza a atribuição op  ← reg
```

b)Operações de suporte ao produto escalar:

```
DPRO acc, memoi, memoj ;acc ← acc + memoi × memoj
DADD acc, memo         ;acc ← acc + memo
DCLE acc               ;acc ← 0
DROD memo, acc         ;memo ← ∇ acc
```

```
DROU memo, acc          ;memo ← Δ acc
DROP memo, acc          ;memo ← □ acc
DADS acci, accJ         ;acci ← acci + accJ
DSUS acci, accJ         ;acci ← acci - accJ
DCM2 acc                ;acc ← -acc
DSAV memo, acc          ;memo ← acc
DLOA acc, memo          ;acc ← memo
DTRA acci, accJ         ;acci ← accJ
```

c)Instruções administrativas do co-processador:

```
STCW memo               ;armazena CR em memo
LDCW memo               ;carrega CR com o conteúdo de
                         memo
STSW memo               ;armazena SR em memo
```

onde CR e SR são, respectivamente, a palavra de controle e a palavra de status do co-processador.

d)Operações que estendem a linguagem de montagem do hospedeiro:

```
ILIN reg,  op ; coloca em reg o limite inferior de op
ILSU reg,  op ; coloca em reg o limite superior de op
ICRI memo, op ; coloca em memo um intervalo degenerado
```

contendo reg.

Os modos de endereçamento permitidos pelas instruções acima correspondem aos do hospedeiro (80286). No entanto, eles são implementados pelas macros, pois as instruções do co-processador assumem que os dados estão disponíveis em registradores internos. Assim, a única restrição é a impossibilidade de transferências diretas entre a memória e os registradores do co-processador. Essas transferências são realizadas utilizando-se instruções do hospedeiro de acesso à memória e instruções de entrada e saída do hospedeiro e do co-processador. O cálculo do endereço dos operandos é feito pelo hospedeiro por questões de eficiência e preservação do sistema de segurança do mesmo.

A seguir são descritas as características do Assembly do co-processador e da comunicação entre ele e o hospedeiro.

2.1 - O Formato da Instrução:

O formato da instrução (figura 1) possui 4 campos: Opcode, Precisão (P), RAccDestino (RD) e RAccFonte (RF) (onde RAcc denota um registrador ou um acumulador).

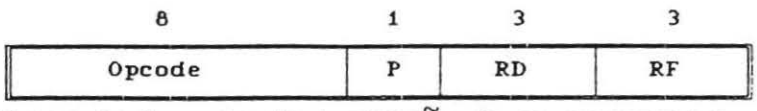


Figura 1: Formato da instrução do co-processador.

O co-processador possui 4 superacumuladores (para a operação de produto escalar com máxima exatidão) e 8 registradores numéricos (para as operações de Aritmética Intervalar) que podem ser usados como fonte ou destino pelas instruções codificadas pelo opcode.

2.2 Comunicação e Sincronização.

A comunicação entre o hospedeiro e o co-processador se dá desde o momento do envio da instrução até a entrega do último byte do operando. Os bits de recebimento RB1 e RB2, respectivamente das palavras de status e de controle do co-processador, servem para sincronizar a comunicação deste com o hospedeiro. O bit RB1 é iniciado com "1" e o hospedeiro fica executando uma rotina de teste sobre ele. O bit RB2 é iniciado com "0" e, quando marcado com "1", sinaliza ao co-processador a disponibilidade de dados no barramento.

Sempre que o co-processador necessite de uma palavra de memória, RB1 é marcado com "0". Quando este valor for lido pelo hospedeiro, a próxima palavra de memória (cujo endereço é fornecido pela instrução sendo executada) será enviada ao co-processador, o hospedeiro atualizará o endereço corrente e marcará RB2 com "1".

O co-processador aguarda o dado verificando se RB2 já mudou o valor para "1". Quando isso acontece, o co-processador lê o dado, marca RB1 com "1" e prossegue o seu processamento.

2.3 Funcionamento das Macros.

As macros fazem parte da biblioteca citada anteriormente e devem executar os seguintes passos:

* Em tempo de Compilação:

-Identificar o tipo da instrução (conforme o modo de endereçamento), para decidir que trechos da macro devem ser expandidos;

* Em tempo de Execução:

-Enviar o código da instrução ao co-processador;

-Se necessário, buscar sucessivamente cada palavra do operando na memória (ou entregar cada palavra do operando à memória, dependendo do tipo da instrução),

-Esperar o co-processador acabar a execução da instrução.

O último passo constitui uma restrição quanto ao paralelismo, justificada pela simplicidade e controle de segurança.

3 - Descrição Geral do Projeto.

O hardware aqui descrito funciona como um Processador de Aritmética Intervalar (PAI) ou como um Processador de Produto Escalar (PPE). Usa-se tecnologia Bit-Slice e microprogramação, onde campos da microinstrução configuram o sistema para operar como PAI ou como PPE.

O PAI consiste de quatro Processadores de Ponto Flutuante (PPF) interligados entre si segundo [1]. Cada PPF possui 14 ALU-slices e uma unidade de controle microprogramada e independente. É dada ao usuário a opção de tratar intervalos com limites em precisão simples ou dupla. Instruções distintas são disponíveis para tratar operações em ambos os casos. As micro-rotinas que implementam essas instruções têm, em campos de suas microinstruções, a definição sobre a precisão adotada. Essa informação é utilizada para definir a configuração de cada PPF. Ao operar em precisão simples, cada PPF utiliza apenas 7 ALU-slices, ao passo que em precisão dupla, faz uso de todo o conjunto das 14 ALU-slices.

A arquitetura usada para implementar o produto escalar foi proposta por [2], onde se utiliza o método da caixa preta [5]. Nesse modo de operação, os quatro PPF's individuais, que constituiam o PAI, são reconfigurados e todo o sistema passa a ter uma única unidade de controle microprogramada: a unidade de controle do PPF1. As ALU-slices de cada PPF são interligadas de modo a formar uma única ALU composta de 56 ALU-slices. Esta ALU funciona como o circuito somador proposto por [2]. Um conjunto de superacumuladores são utilizados para armazenar resultados intermediários no cálculo do produto escalar.

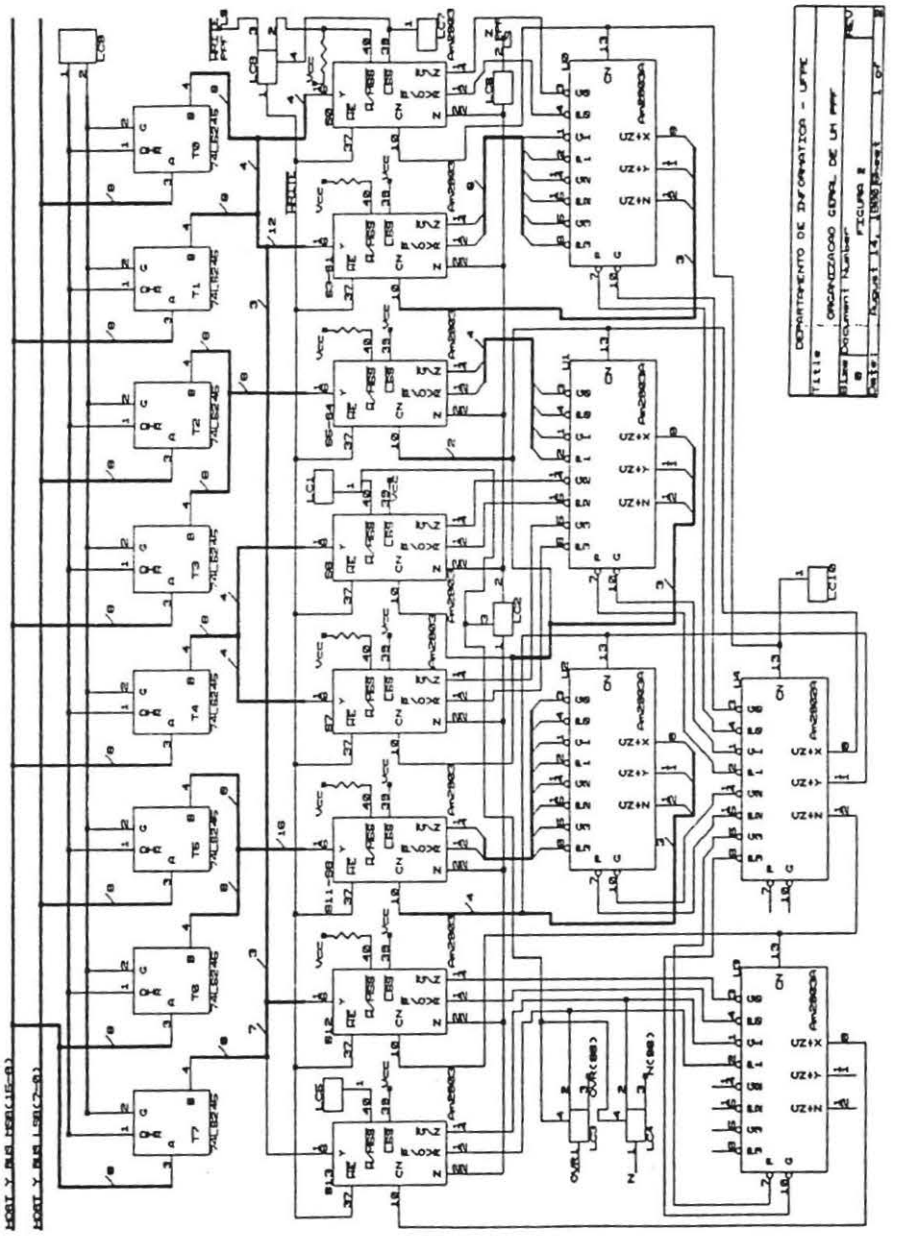
4 - As Reconfigurações do Hardware.

A configuração do co-processador para que opere como PAI ou PPE é um ponto fundamental deste trabalho, sendo este o assunto agora abordado.

4.1 - O PPF.

O PPF é um processador microprogramado cuja unidade de controle foi explorada em [1]. As operações de aritmética e lógica são executadas em um conjunto de 14 chips Am2903 [3], ALU-slices de 4 bits. Com o intuito de acelerar a propagação do bit de carry através da cadeia de ALU-slices, são usados mais 5 chips Am2902 em dois níveis, segundo [3], que implementam a lógica do "carry-look-ahead". A organização geral de um PPF é ilustrada na figura 2, onde os blocos marcados por LC_i são as lógicas de controle que configuram o hardware do PPF para o modo de operação desejado.

Sendo o PPF uma cadeia de Am2903 ligados em série, a slice S0 é programada como "Least Significant Slice" (LSS), a slice S13 como "Most Significant Slice" (MSS), enquanto que as demais são programadas como "Intermediary Slices" (IS) [3]. A questão da reconfiguração consiste basicamente em se modificar a posição relativa de algumas slices.


DEPARTAMENTO DE INFORMATICA - UFPE
ORGANIZACION GENERAL DE UN PMP
FIGURA 2

4.2 - Campos da Microinstrução que Determinam as Configurações.

As informações para a configuração dos PPF's estão presentes em campos específicos das microinstruções. É interessante ressaltar que esses campos são utilizados para coordenar outras operações, além da reconfiguração do hardware.

O campo PE (1 bit), quando igual a "1", indica que a microinstrução deve ser executada com o co-processador configurado como PPE, caso contrário, como PAI.

O campo PD também ocupa um bit nas microinstruções. Quando PD for igual a "1", os operandos tratados pelo PAI estão em precisão dupla, caso contrário, em precisão simples. Esta informação só tem sentido quando o campo PE for igual a "0", o que indica operação executada pelo PAI.

O co-processador, operando como PPE, concatena os quatro PPF's como um única cadeia de 56 ALU-slices. Portanto, as slices S0 e S13 que estavam configuradas, respectivamente, como LSS e MSS nos PPF's, passam a ser IS em alguns PPF's, dependendo da posição significativa destes. Essa posição é codificada pelos campos de um bit ppf_1 e ppf_2. A combinação dos valores desses campos indica em qual PPF a micro-rotina está sendo executada e determina a posição significativa dos PPF's (tabela 1).

ppf_1	ppf_2	PPF Codificado	Posição Significativa
0	0	PPF4	Menos Significativo
0	1	PPF3	Intermediário
1	0	PPF2	Intermediário
1	1	PPF1	Mais Significativo

Tabela 1: Codificação dos campos ppf_1 e ppf_2 e posição significativa de cada PPF.

4.3 - Configurando o Co-processador para Operar como Processador de Aritmética Intervalar (PAI).

Ao executar uma instrução do PAI, os PPF's podem

utilizar apenas 7 ou as 14 slices Am2903 que os constituem. Esta escolha é baseada no valor do campo PD (precisão) da microinstrução sendo executada.

Inicialmente, observa-se a necessidade de se definir qual das slices, S6 ou S13, será configurada como MSS. Esta determinação é importante porque a MSS gera sinais, tais como overflow (OVR) e negativo (N), que fazem parte da palavra de status de cada PPF.

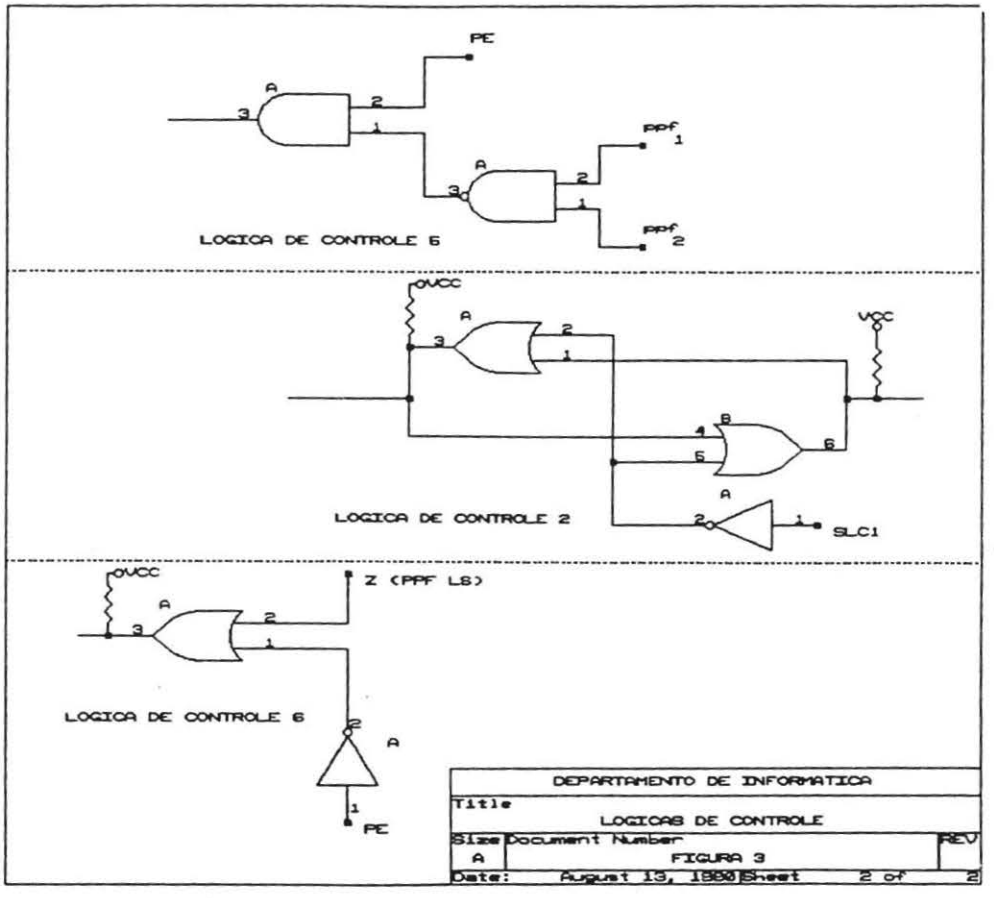
A slice S6 será configurada como MSS quando a precisão simples for adotada em operações com o PAI, ou seja, quando PD for igual a "0". Operações com precisão dupla ou de produto escalar configuram esta slice como IS. Portanto, a configuração apropriada para a slice S6 é obtida através do bloco LC1 (figura 2). Esse bloco é simplesmente uma porta "OR" de duas entradas, que correspondem aos campos PD e PE e cuja saída é ligada ao pino $\overline{\text{WRITE/MSS}}$ da slice S6.

A slice S13 será sempre configurada como MSS quando operações do PAI estiverem sendo executadas. A possibilidade de se configurar esta slice como IS será discutida na subseção 4.4.

Observa-se então que as possíveis fontes dos sinais de estado OVR e N são duas (S6 ou S13). Para que as informações apropriadas sejam armazenadas na palavra de status de cada PPF, são utilizados os blocos LC3 e LC4, responsáveis, respectivamente, pela geração de um dos sinais de OVR e N. Esses blocos idênticos e elementares são constituídos de um multiplexador 2x1, onde a entrada de seleção corresponde à saída da LC1 e as demais entradas correspondem às possíveis fontes dos respectivos sinais.

Um outro fato relevante é que o pino Z de todas as slices Am2903 de um PPF têm que ser conectados entre si e também ao Vcc através de um resistor de "pull-up" conforme [3]. Contudo, a informação que provém da microinstrução, de qual operação deva ser executada pelo PPF, é enviada simultaneamente a todas as 14 ALU-slices, independente da precisão adotada. No entanto, quando a operação envolve precisão simples, o sinal gerado pelo pino Z das slices S7 a S13 não têm significado e, por isso, não devem alterar o sinal obtido pelas slices S0 a S6. Para tal, o bloco LC2 da figura 2, uma lógica em coletor aberto,

isola eletricamente o efeito da saída Z das slices S7 a S13 na saída Z das S0 a S6. Na figura 3, encontra-se uma ilustração deste bloco.



4.4 - Reconfigurando o Co-processador para Operar como Processador de Produto Escalar (PPE).

A fim de executar uma operação do PPE, todo o conjunto das 56 Am2903 deve estar conectado como uma única ALU de 224 bits, que funciona como o circuito somador proposto por [2].

Como visto na subseção 4.2, as slices S0 e S13, em alguns PPF's, deixam de ser LSS e MSS respectivamente e passam a

ser IS. Os campos ppf_1 e ppf_2 definem em quais PPF's essa reconfiguração vai se processar. Segundo a tabela 1, a slice S13 do PPF1 será a MSS do circuito somador, enquanto que a slice S0 do PPF4 será a LSS. Todas as demais têm que ser configuradas como IS.

Dessa forma, o bloco LC5 (figura 2) foi incluído no circuito para que, a partir dos valores dos campos PE, ppf_1 e ppf_2, envie o sinal apropriado ao pino $\overline{WRITE}/\overline{MSS}$ da slice S13. A figura 3 ilustra esta lógica simples. Observe-se que apenas o bloco LC5 do PPF1 gera um "0" lógico, programando a slice S13 deste como MSS. Em conseqüência, as demais slices S13 dos outros PPF's são automaticamente programadas como IS por este bloco.

O bloco LC7 (figura 2) é semelhante ao LC5, onde em lugar da porta "NAND" tem-se uma porta "OR". Aqui, no entanto, a saída da LC7 é conectada ao pino $\overline{LSS}$ da slice S0; garantindo assim que apenas a slice S0 do PPF4 seja configurada como LSS e as demais, dos outros PPF's, como IS.

Para acelerar a propagação do bit de carry através das 56 ALU-slices, são usadas as Am2902 de cada PPF e mais uma que acelera a propagação entre os PPF's. Assim sendo, o circuito somador usa 21 Am2902 em três níveis, implementando a lógica do "carry-look-ahead".

Um sinal muito importante, o $\overline{WRITE}$ gerado pelo pino $\overline{WRITE}/\overline{MSS}$ da LSS dos PPF's, é usado para habilitar a escrita de dados no conjunto de RAM das Am2903 [3]. Este sinal é enviado a todas as ALU-slices de um mesmo PPF quando o PAI está em operação. Por outro lado, operando como PPE, este sinal é gerado pela LSS do circuito somador, ou seja, a slice S0 do PPF4. Como cada PPF é montado em uma única placa de circuito, o bloco LC8, um multiplexador 2x1, seleciona a fonte do sinal de $\overline{WRITE}$ que pode ser a saída do pino $\overline{WRITE}/\overline{MSS}$ da LSS do mesmo PPF (operando o PAI, ou operando o PPE se for o PPF4) ou o sinal $\overline{WRITE}$ de um PPF menos significativo (operando o PPE). A entrada de seleção desse multiplexador é a saída do bloco LC7. Observe que quando LC7 configura uma slice S0 como LSS também seleciona a saída $\overline{WRITE}/\overline{MSS}$ da mesma para ser a fonte do sinal de $\overline{WRITE}$.

O circuito somador, sendo uma cadeia de 56 ALU-slices, também deve ter o pino Z de todas elas interligados como dito

anteriormente. Contudo, esta ligação entre os PPF's precisa ser desfeita quando o co-processador operar como PAI. Para que isso fosse implementado, incluiu-se o bloco LC9, um circuito em coletor aberto, mostrado na figura 3. Quando PE for igual a "0", LC9 desconecta, eletricamente, o pino Z das ALU-slices entre PPF's e quando igual a "1", os conecta.

5 - A Arquitetura.

Neste trabalho foi proposta uma arquitetura que dá suporte ao tipo de dado intervalo quando o co-processador opera como PAI. Os operandos podem ser tratados em precisão simples ou dupla. Cada limite de um resultado intervalar é arredondado de maneira distinta. O limite superior sempre sofre um arredondamento direcionado para cima, enquanto que o limite inferior, um arredondamento direcionado para baixo. Esses dois tipos de arredondamento são implementados pelo PAI.

Funcionando como PPE, o co-processador opera com números reais, onde apenas operandos em precisão dupla são usados. O método aqui adotado para implementar o produto escalar armazena resultados intermediários. Dessa forma, apenas ao final do cálculo é feito um único arredondamento que, de acordo com a instrução utilizada (seção 2), pode ser para baixo, para cima ou para o mais próximo.

Esta arquitetura dá suporte a um limitado conjunto de instruções, onde os operandos encontram-se disponíveis em registradores do co-processador. Assim, operandos são armazenados e recuperados desses registradores através de instruções do tipo "load" e "store".

As transferências de dados entre o co-processador e o hospedeiro são realizadas através de um barramento de 16 bits. Assim, a entrega de operandos e resultados se dá em mais de uma fase (etapa). A quantidade de fases depende da precisão adotada. Na precisão simples, os dados são números reais representados com 23 bits de mantissa, 8 bits de expoente e um de sinal. O carregamento (entrega) de dados é executado em duas fases. Na primeira etapa, os 16 LSB da mantissa são transferidos para as

slices S0 a S3, onde apenas os transceivers T0 e T1 estão habilitados pelo bloco LC9 para comunicação hospedeiro-PAI. Na segunda etapa, os 8 bits mais significativos da mantissa são transferidos para as slices S4 e S5, e os 8 bits do expoente para as S0 e S1. Aqui, apenas os transceivers T0 e T3 são habilitados. Pode-se observar, na figura 1, como estão dispostos os transceivers em relação ao barramento Y do hospedeiro e como estão conectados às slices Am2903.

Dados em precisão dupla são números reais representados com 52 bits de mantissa, 11 de expoente e 1 de sinal. O carregamento (entrega) é executado aqui em quatro fases. A primeira, segunda e terceira fases são idênticas à primeira fase do carregamento em precisão simples descrita acima, onde os bits m_{15} a m_{0}, m_{31} a m_{16} e m_{47} a m_{32} da mantissa são transferidos para as slices S0 a S3, S4 a S7 e S8 a S11, através dos transceivers T0 e T1, T2 e T4 e T5 e T6 nas respectivas fases. Na quarta etapa, os bits m_{48} a m_{51} da mantissa, juntamente com o bit de sinal são transferidos para as slices S12 e S13, através do transceiver T7. Paralelamente, os 11 bits do expoente são transferidos para as slices S0 a S2, através dos transceivers T0 e T7.

O carregamento de operandos e a entrega de resultados pelo PPE são implementados através de transferências de blocos de 8 bits por vez. A transferência em blocos de 16 bits necessitaria de 7 transceivers adicionais em cada PPF par (2 e 4), encarecendo o projeto e comprometendo a modularidade dos PPF's.

O hospedeiro fica monitorando o bit de recebimento (RB1) da palavra de status do co-processador, num processo de "pooling", para determinar se um novo bloco de 8 bits pode ser enviado ao PPE.

6 - Considerações Finais.

Este trabalho detalha o projeto de construção de um co-processador dedicado à Aritmética Intervalar e ao cálculo do Produto Escalar com máxima exatidão que está sendo desenvolvido

pelo Departamento de Informática da UFPE.

Foi mostrado o estágio atual das pesquisas, estando ainda previstas as fases de montagem do processador e o desenvolvimento de um microassembly para auxiliar na codificação dos microprogramas.

A fim de melhorar a interação com o usuário, será desenvolvida posteriormente uma extensão de uma linguagem de alto nível que opere o tipo de dado intervalo usando o co-processador.

REFERÊNCIAS.

[1] S.S.A. Montenegro, G.C. Vasconcelos & N.C.L Vieira. A Arquitetura de um Processador de Aritmética de Intervalos. Anais do IX Congresso da SBC, Uberlândia, 1989.

[2] D. Florissi & M.B. Correia. Métodos Existentes para a Implementação do Produto Escalar com Máxima Exatidão e uma Arquitetura para Microcomputadores. Anais do IX Congresso da SBC, Uberlândia, 1989.

[3] Am2900 Family Data Book. Advanced Micro Devices, 1983.

[4] PASCAL-SC, Information Manual and Floppy Disks.

[5] U.W. Kulish & W.L. Miranker. A New Approach to Scientific Computation. Proceedings of the Symposium on A New Approach to Scientific Computation. Academic Press, New York, 1982.

[6] G.J. Myers. Digital System Design with LSI Bit-Slice Logic. Wiley - Interscience, New York, 1980.

[7] D. Florissi. Implementação de um Co-processador de Suporte à Aritmética Computacional Avançada e à Aritmética de Intervalos. Dissertação de Mestrado. Universidade Federal de Pernambuco, 1989.