Reconfigurando e selecionando conjuntos de instruções em uma arquitetura microprogramada

Leonardo Augusto Casillo
Depto. de Ciências Exatas e Naturais
Universidade Federal Rural do Semi-Árido
Mossoró – RN - Brasil
casillo@ufersa.edu.br

Ivan Saraiva Silva
Depto. de Informática e Estatística
Universidade Federal do Piauí
Teresina – PI - Brasil
ivan@ufpi.edu.br

***Resumo* — Os processadores RISP (*Reconfigurable Instruction Set Processors*) são um subconjunto dos processadores reconfiguráveis, que executam instruções fixas, assim como processadores de propósito gerais, e instruções personalizadas implementadas sob uma lógica reconfigurável. Este trabalho mostra o desenvolvimento de um RISP microprogramado que suporta diferentes conjuntos de instruções em uma mesma unidade de controle e a capacidade de receber e executar novas instruções em tempo de execução utilizando uma unidade de reconfiguração acoplada. Esta reconfiguração dinâmica e a seleção de diferentes padrões de instruções fornecem flexibilidade na realização de aplicações e permitem uma expansão do conjunto de instruções, sem a necessidade de reconfigurar o o caminho de dados da microarquitetura. Este trabalho também apresenta resultados de desempenho relacionados com a quantidade de ciclos de relógio na execução de uma aplicação simples contendo instruções reconfiguradas.**

Palavras-chave — RISP; arquiteturas microprogramadas; conjuntos de instruções; reconfiguração de instruções

I. INTRODUÇÃO

A tecnologia da computação reconfigurável consiste na habilidade de se modificar o *hardware* da arquitetura do sistema em tempo real para se adequar à aplicação em tempo de execução [1]. A alteração de um *hardware* reconfigurável permite uma adequação da arquitetura à aplicação corrente, possibilitando ao sistema obter desempenho semelhante ao de um processador ASIC (*Application Specific Integrated Circuit*) e evitando custos de processamento associado às arquiteturas de uso geral.

Um dos principais segmentos das arquiteturas reconfiguráveis são os processadores reconfiguráveis, os quais possuem a vantagem de combinar as funções de um microprocessador com uma lógica reconfigurável. Estes processadores podem ser adaptados a uma nova aplicação ou funcionalidade depois do processo de desenvolvimento do mesmo modo que os processadores programáveis podem se adaptar a mudanças na aplicação [2].

Os processadores RISP (*Reconfigurable Instruction Set Processors* - processadores com conjunto de instruções reconfiguráveis) são um subconjunto dos processadores reconfiguráveis, que tem como característica permitir ao projetista criar novas instruções, possibilitando a execução de determinadas tarefas em menos ciclos em relação aos processadores de uso geral, aumentando assim o seu desempenho [3].

Esse artigo apresenta uma arquitetura projetada para uso de processadores RISP que propicia o aproveitamento das técnicas de configuração do conjunto de instruções de um processador em tempo de desenvolvimento, bem como prover a reconfiguração de instruções em tempo de execução. Para tanto, foi desenvolvida uma unidade de reconfiguração acoplada a um processador RISC projetada para trabalhar com múltiplos conjuntos de instruções, permitindo ao usuário selecionar, em função da aplicação, qual conjunto oferece melhores condições de funcionamento. A unidade de reconfiguração também possui a capacidade de criar e configurar novas instruções em tempo de execução.

II. TRABALHOS RELACIONADOS

A. Processadores Reconfiguráveis

O primeiro processador de propósito geral acoplado a um FPGA foi denominado PRISM (*Processor Reconfigurable through Instruction-Set Metamorphosis*), em 1993 por [4]. Devido à limitação da tecnologia dos FPGAs da época, somente era possível desenvolver um sistema fracamente acoplado, no qual um FPGA comunicava-se com um processador por meio de um barramento.

O primeiro processador dito reconfigurável foi projetado em 1994 por [5] com o nome PRISC (*PRogrammable Instruction Set Computer*). Este processador foi o primeiro a explorar a existência de uma RFU (*Reconfigurable Functional Unit*) contida em um caminho de dados de um processador MIPS. Neste processador, verificou-se a eficiência da utilização de FPGAs para aplicações específicas, aumentando o desempenho com relação a soluções em *software*.

Outras arquiteturas clássicas com características reconfiguráveis podem ser vistas em [6], [7], [8], [9] e [10]. Atualmente, com a disponibilização de kits de desenvolvimento em FPGAs e o fácil acesso às ferramentas de prototipação e síntese de tais dispositivos, além da popularização do ensino de linguagens de descrição de *hardware* como VHDL, muitas instituições de ensino desenvolveram experimentos de microarquiteturas acadêmicas com características reconfiguráveis, dentre estas podem ser citadas [11], [12], [13], [14] e [15].

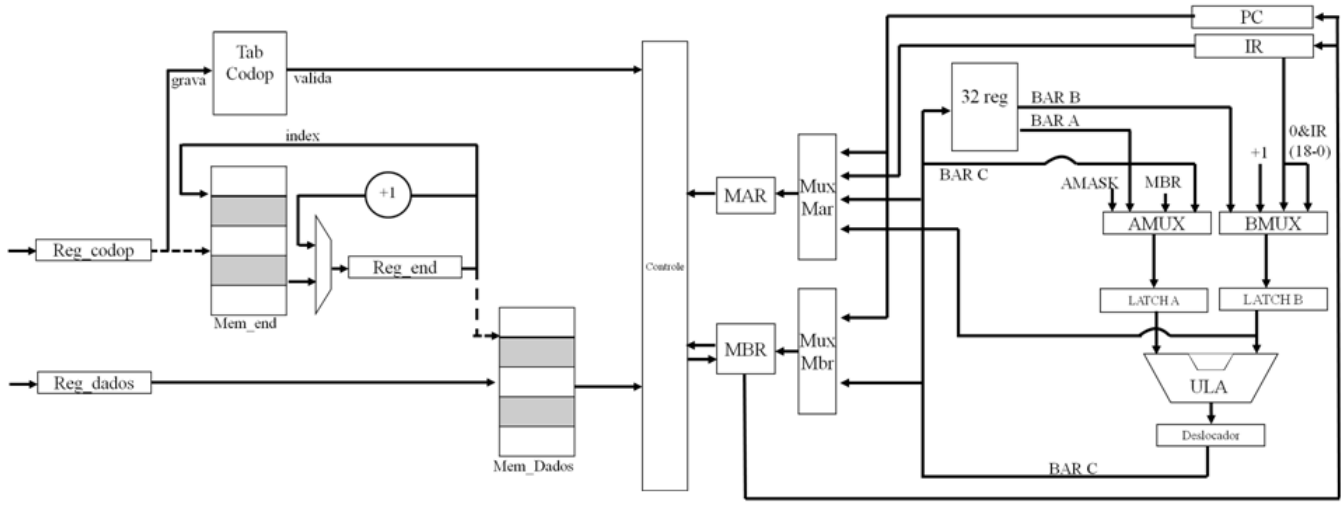

Fig. 1. Visão geral da microarquitetura CRASIS

B. Arquiteturas microprogramadas

A ideia de um controle microprogramado surgiu na década de 1950. O circuito lógico destas unidades é formado por uma sequência de microinstruções, contendo os sinais de controle para executar cada etapa do microprograma e uma lógica de sequenciamento para decisão da próxima microinstrução. Estes sinais de controle são enviados para a unidade operativa (caminho de dados) do dispositivo, indicando o comportamento de cada unidade funcional, como a operação da ULA, a codificação da seleção de multiplexadores ou a habilitação de escrita em registradores.

A vantagem de um controle microprogramado sobre uma implementação em *hardware*, além da redução da complexidade, é o fato de torná-la mais simples, compreensível e menos sujeita a erros. Em contrapartida, sua execução é mais lenta e o consumo de potência gerada pelo dispositivo é maior.

Das implementações publicadas na literatura relacionadas a processadores microprogramados, um trabalho de destaque é o de [16], no qual foi projetado um processador sub-RISC como uma classe de um processador ASIP com arquitetura programável.

Na literatura, até o presente momento não foram encontrados trabalhos, além dos experimentos desenvolvidos pelos autores deste trabalho, sobre uma arquitetura específica que reúna os conceitos e funcionalidades de processadores com conjuntos de instruções reconfiguráveis, arquiteturas microprogramadas e seleção de diferentes conjuntos de instruções.

III. ARQUITETURA PROPOSTA

O processador denominado CRASIS foi desenvolvido com sua organização do caminho de dados inspirada na arquitetura descrita por [17]. Trata-se de uma microarquitetura RISC simples de 32 *bits* de palavra descrita utilizando a linguagem VHDL, com sua unidade de controle de forma microprogramada, contendo diferentes conjuntos de instruções que podem ser alternados em tempo de execução, ao mesmo tempo em que podem ser inseridas novas instruções por meio de uma unidade de reconfiguração acoplada à sua estrutura física. A Fig. 1 ilustra as unidades funcionais do processador.

Sua unidade operativa é formada por blocos funcionais compostos por um conjunto de 32 registradores, sendo quatro destinados a funcionalidades específicas, como acumulador e apontador de pilha e os demais para uso geral, um deslocador de 32 *bits* para a esquerda ou direita, um contador de programa (PC), um registrador de instrução (IR) e uma ULA com capacidade para 8 operações distintas: transparência do primeiro operando, AND e OR lógicos, negação do primeiro operando, negação do segundo operando e operações aritméticas de adição, subtração e multiplicação. Para a comunicação com a memória, são utilizados dois registradores de interface, sendo um para endereços (MAR) e um para dados (MBR).

O funcionamento geral da arquitetura dá-se em um período de quatro subciclos de relógio para cada microinstrução realizada, que consistem nos seguintes eventos em sua unidade operativa:

- Obtenção dos sinais de controle das unidades funcionais;
- Transferência dos operandos;
- Obtenção do valor resultante da ULA;
- Armazenamento do resultado.

Todas as instruções possuem 32 *bits* de palavra. Existem 5 formatos diferentes de instruções, que utilizam de 0 a 4 operandos. Em todos os formatos de instrução, o primeiro *byte* é destinado ao Código da Operação (CodOp) e os demais utilizados para endereçamento ou para determinação do(s) registrador(es) selecionado(s) para a operação. Cada novo formato exclui 5 *bits* do campo de endereço/operando para incluir a seleção de mais um registrador. Dessa forma, o campo

de operando possui 24 *bits* no primeiro formato, 19 *bits* no segundo formato, e assim sucessivamente.

O modelo da microarquitetura contém três classes de conjuntos de instruções: utilizando apenas a pilha, contendo um registrador acumulador fixo como primeiro operando e utilizando três registradores. Cada um destes conjuntos contém 32 instruções, sendo 30 específicas para cada classe e duas comuns a todos os conjuntos para habilitar e selecionar o conjunto de instruções atual. A Tabela 1 contém a relação das instruções presentes em cada conjunto.

TABELA 1. CONJUNTOS DE INSTRUÇÕES DO PROCESSADOR

Código da Operação (Bin)	Conjunto 1	Conjunto 2	Conjunto 3
00000000	LODD X	LODD X	LODDp X
00000001	STOD X	STOD X	STODp X
00000010	LOCO X	LOCO X	LOCO X
00000011	LODL X	LODL Rx, Y	LODLp X
00000100	STOL X	STOL Y, Rx	STOLp X
00000101	ADDL X	LODR Ra, Rx, Ry	LODIp X
00000110	SUBL X	STOR Ra, Rx, Ry	STOIp X
00000111	CALL X	CALL X	CALL X
00001000	JUMP X	JUMP X	JUMP X
00001001	JNEG X	JNEG X	JNEG X
00001010	JPOS X	JPOS X	JPOS X
00001011	JNZE X	JNZE X	JNZE X
00001100	JZER X	JZER X	JZER X
00001101	ADDD X	ADDR Ra, Rx, Ry	ADDp X
00001110	SUB DX	SUBR Ra, Rx, Ry	SUBp X
00001111	MULL X	MULL Ra, Rx, Ry	MULp X
11110000	RETN	RETN	RETN
11110001	PSHI	SHL	SHLp
11110010	POPI	SHR	SHRp
11110011	PUSH	PUSH	PUSH
11110100	POP	POP	POP
11110101	SWAP	SWAP	SWAP
11110110	SETI	SETI	SETI
11110111	RETI	RETI	RETI
11111000	INSP	INSP	INSP
11111001	DESP	DESP	DESP
11111010	LDIO	LDIO	LDIO
11111011	STIO	STIO	STIO
11111100/01	-	-	-
11111110	muda_conj		
11111111	muda_modo		

O processador pode atuar sob três modos de funcionamento: reconfiguração de instruções, execução das instruções originais presentes no conjunto de instruções, denominadas "instruções fixas", e execução das instruções reconfiguradas. Estes dois últimos modos operam de maneira semelhante quanto ao ciclo de vida da instrução, composto pelos estágios de busca, decodificação, execução e armazenamento.

Caso seja uma instrução fixa, a unidade de controle envia os sinais de controle de cada microinstrução do conjunto selecionado para os blocos funcionais da unidade operativa. Caso seja uma instrução reconfigurada, a unidade de controle fará a busca das microinstruções armazenadas na unidade de reconfiguração para serem enviadas à unidade operativa. A Fig. 2 mostra um diagrama simplificado contendo a máquina de estados (*FSM – Finite State Machine*) destes modos de funcionamento.

A determinação do modo de funcionamento é feita através do estado denominado *Verifica_reconfiguração*, responsável por verificar se houve um pedido de reconfiguração de uma instrução através de um sinal de requisição externo, de forma análoga a um pedido de interrupção por *hardware*. Caso este sinal contenha uma solicitação de reconfiguração, será iniciada a sequência de reconfiguração de instruções; caso contrário será iniciado o processo de execução normal.

A. Super-estados

A unidade de controle do CRASIS contém uma máquina de estados que comporta diferentes conjuntos de instruções. Cada código de operação corresponde a uma série de microinstruções, os quais podem fazer parte do microprograma de uma ou mais instruções distintas, de acordo com o conjunto utilizado. Cada conjunto possui uma quantidade própria de estados formados pelas microinstruções necessárias para executar as instruções, sem interferir nos demais conjuntos.

Para gerar uma FSM menor e com menos recursos de *hardware*, foi introduzido o conceito de "super-estados", que são a reunião de microinstruções pertencentes a diferentes conjuntos em um mesmo estado. Para cada uma das 28 instruções, um CodOp foi associada a um super-estado, nomeados OP0 a OP27. Um mesmo código de operação pode corresponder às instruções com diferentes formatos e operandos.

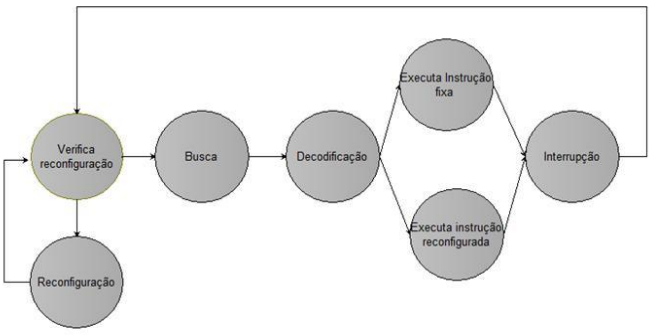

Fig. 2. Ciclo de vida de uma instrução

Form	AMUX	BMUX	ULA	SH	MBR	MAR	ENC	MUX MAR	MUX MBR	WR	RD	BIT_ END
6 bits	2 bits	2 bits	3 bits	6 bits	1 bit	1 bit	1 bit	2 bits	1 bit	1 bit	1 bit	1 bit

Fig. 3. Formato de uma instrução reconfigurada

Por exemplo, a instrução OP5 com CodOp "00000101" representa a instrução ADDL X para o conjunto 1, que utiliza apenas um operando. Para o conjunto 2, a instrução selecionada será LODR Ra, Rx, Ry, que utiliza 3 registradores. Para o conjunto 3, a instrução será LODI, que não requer nenhum operando adicional. A Fig. 4 ilustra o diagrama de estados desta operação.

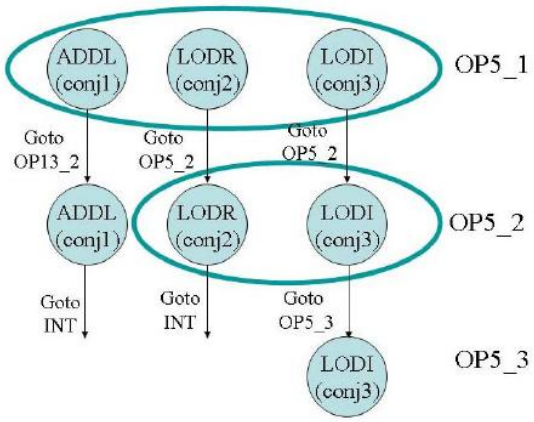

Fig. 3. Exemplo de super-estados

O primeiro super-estado da instrução OP5 (OP5_1) contém a microinstrução referente ao início da instrução ADDL X, presente no conjunto 1, a microinstrução referente ao início da instrução LODR Ra, Rx, Ry, presente no conjunto 2 e a microinstrução referente ao início da instrução LODI, presente no conjunto 3. A sequência da primeira instrução contém o mesmo comportamento de outro super-estado, portanto sua rotina é desviada e não necessita de conteúdo no segundo super-estado (OP5_2), que contém as microinstruções pertencentes às instruções dos conjuntos 2 e 3. Neste ponto, a instrução do conjunto 2 é finalizada, mas a instrução do conjunto 3 ainda necessita de mais uma microinstrução. Portanto, existe um terceiro super-estado (OP5_3) que contém apenas a microinstrução referente ao conjunto 3.

A implementação de super-estados é realizada utilizando estruturas de decisão dentro da unidade de controle, após a decodificação de uma instrução. Dessa forma, um mesmo super-estado pode conter microinstruções pertencentes a 1, 2 ou mesmo todos os conjuntos. Para um mesmo CodOp em particular, podem existir diferentes instruções com quantidade de microinstruções distintas, bem como existe a possibilidade de unir instruções de diferentes conjuntos em um mesmo estado.

B. Execução de instruções fixas

Durante a decodificação, se os quatro primeiros *bits* do CodOp forem "0000" ou "1111", a instrução pertence a algum dos conjuntos fixos. Caso contrário, se a instrução a ser executada for reconfigurável, será iniciado processo de execução de instruções reconfiguradas. Uma instrução reconfigurada é composta por uma ou mais microinstruções que formam um microprograma que modela sua execução, formada por 28 *bits* de controle, dispostos na Fig. 3.

As instruções podem conter 1 a 4 registradores como operandos de entrada ou saída, podendo assumir os formatos "OP A" a "OP A, B, C, D". Dependendo do formato utilizado e da quantidade de operações que compõem a instrução, um registrador utilizado como destino na microinstrução '1' pode ser utilizado como entrada na microinstrução '2', permitindo uma maior integração entre diferentes etapas de uma mesma instrução, como uma realimentação de dados. As configurações de cada um dos demais *bits* de controle devem permitir todas as possibilidades de funcionamento de cada um dos blocos funcionais da unidade operativa, como apresenta a Tabela 2.

TABELA 2. BITS DE CONTROLE DO CRASIS

Nome	Função	Valores
Form	Define formato da instrução	Vide consulta tabela de formatos
AMUX	Operando esquerdo da ULA	00 a 11 – Barramento A, Barramento C, AMASK, MBR
BMUX	Operando direito da ULA	00 a 11 – Barramento B, Constante '1', IR, 0x00 & IR(18-0)
ULA	Operação lógica/aritmética	000 a 111 - A (transparência), A AND B, NOT(A), NOT(B), A + B , A – B, A * B, A OR B
SH	Tipo/Quant. de deslocamentos	*Bit* 1 – Direção (esq/dir) *Bits* 2-6 – quantidade
MBR	Carrega buffer de dados	0 – desativado 1 - ativado
MAR	Carrega buffer de endereço	0 – desativado 1 – ativado
ENC	Habilita escrita no banco de registradores	0 – desativado 1 – ativado
MUX_MAR	Seleciona entrada de MAR	00 a 11 – PC, IR, Latch B, Barramento C
MUX_MBR	Seleciona entrada de MBR	0 – PC 1 – Barramento C
WR	Escrita na memória	0 – desativado 1 – ativado
RD	Leitura na memória	0 – desativado 1 – ativado
BIT_END	Fim de instrução	0 – não finalizada 1 – finalizada

C. *Unidade de reconfiguração*

A unidade de reconfiguração foi projetada para o tratamento de instruções customizadas. Esta unidade possibilita criar uma instrução que não se encontra nos conjuntos de instruções já existentes ou combinar duas ou mais instruções reconfiguradas. A Fig. 5 ilustra os blocos funcionais constituintes.

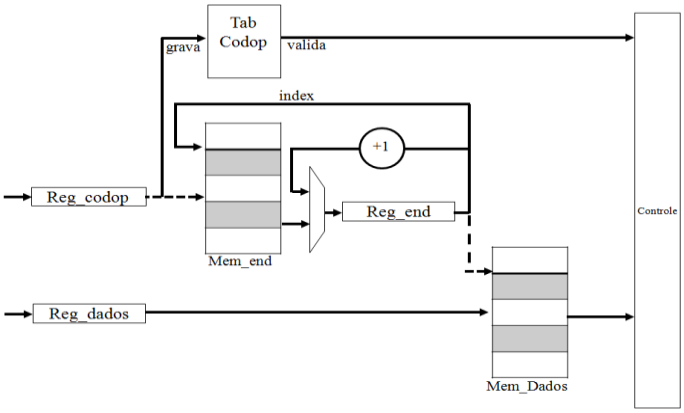

Fig. 5. Unidade de Reconfiguração do CRASIS

A reconfiguração de uma instrução ocorre quando o processador recebe um sinal externo, indicando um pedido de reconfiguração. Neste processo, o processador deve receber um CodOp identificando a instrução customizada e as microinstruções para a configuração da instrução. Durante todo o processo de reconfiguração, o Contador de Programa (PC) permanece com seu valor atual.

Ao ser recebido o pedido de reconfiguração, deve ser confirmado se a instrução recebida é válida, ou seja, se o CodOp recebido pode ser utilizado por uma instrução customizada. Ao ser confirmado se tratar de um CodOp válido, o processador deverá armazenar o código recebido, associando-o com os valores das microinstruções da instrução customizada, que devem ser armazenadas sequencialmente até que seja indicado o final da instrução. O diagrama de estados da Fig. 6 representa os estados correspondentes ao processo de reconfiguração de uma instrução.

A verificação do CodOp recebido é realizada da seguinte forma: os quatro primeiros *bits* do CodOp são comparados com os valores "0000" e "1111". Caso o resultado dessa comparação seja positivo, o CodOp não será considerado válido, e consequentemente a instrução não poderá ser configurada. Como resultado, é gerado um sinal de erro indicando que não é possível realizar esta operação. Caso o resultado da comparação seja negativo, o CodOp recebido encontra-se na faixa de valores reservados para instruções reconfiguráveis, ou seja, entre "0001" e "1110", continuando a sequência de execução.

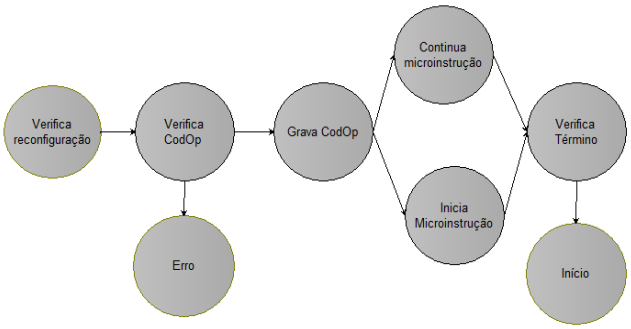

Fig. 6. Diagrama de estados do processo de reconfiguração

Para esta etapa, foi utilizada uma tabela de 64 posições contendo um *bit* de validação de cada endereço, convencionado como '0' para endereço livre e '1' para endereço reconfigurado. A cada processo de escrita, ao verificar que o endereço está disponível, o *bit* de validação é alterado para '1' até que a unidade de reconfiguração seja reiniciada. Esta atualização da tabela é necessária para o modo de execução de instruções, quando, durante o estágio de decodificação, é verificado se o CodOp recebido está realmente associado a uma instrução reconfigurada existente na unidade.

Em seguida, o valor do CodOp é enviado como endereço de um módulo de memória LPM RAM (*Library of Parameterized Modules RAM*) de 64 posições de palavras de 5 *bits*, contendo um índice que marca a posição inicial da primeira microinstrução. A cada final de gravação de uma instrução, a posição da última microinstrução armazenada é salva em um registrador de índices, que será utilizado na escrita da próxima instrução.

A primeira microinstrução de cada CodOp reconfigurado é armazenada em outro módulo de memória RAM com *n* palavras de 28 *bits*, utilizando como endereço o índice especificado na gravação do CodOp. O processo de armazenamento de microinstruções na memória RAM de dados é realizado sequencialmente até que seja identificado um *bit* de finalização, informando o término do processo de gravação da instrução. Como única exceção, a unidade não reconfigura instruções de desvio condicional por não utilizar os *flags* da ULA como componentes das microinstruções.

Para armazenar uma instrução reconfigurável MAC D, A, B, C (multiplica e acumula), em que D = A + (B*C), o processo consiste em armazenar o CodOp da instrução, caso este seja válido, e em seguida todas as microinstruções. Esta instrução possui o Código de Operação "0001", armazenado na RAM de endereços, e 2 microinstruções armazenadas na RAM de dados, com os seguintes valores:

- 0xA830040 / D = B * C / Bar C = Bar B op Acc
- 0x4C20041 / D = D + A / Bar C = Bar C op Bar A

Sendo esta a primeira instrução reconfigurada, o valor 0x00 será armazenado na memória de endereços, na posição "0001" dada pelo CodOp, e as microinstruções armazenadas nas posições 0x00 e 0x01 da memória de dados. Após o término do processo de escrita, o registrador de índice é atualizado com o

valor 0x02, que será o conteúdo armazenado na posição identificada pelo próximo Código de Operação reconfigurável.

Durante o modo de execução de instruções, caso o processador receba da memória a instrução 0x11390A60, a consulta aos quatro primeiros *bits* de CodOp "0001" resultará em uma busca na unidade de reconfiguração. Observando a tabela de verificação, utilizando o valor dos quatro primeiros *bits* do CodOp como endereço, será informado que se trata de uma instrução reconfigurada por meio do *bit* de validação ativado ao cadastrar a instrução.

A instrução possui o valor "00010001 00111 00100 00101 00110 0000", o que resulta em (MAC R7, R4, R5, R6, 0000), com significado R7 ←R4 + (R5 * R6). Isto mostra que, mesmo os conjuntos de instruções contendo instruções com apenas 1 ou 3 operandos, é possível definir diferentes formatos de instruções e criar instruções com até 4 registradores distintos, variando a posição dos operandos em cada microinstrução. Para isto, foi utilizado um codificador no estágio de decodificação de instruções reconfiguráveis, nos quais os 6 primeiros *bits* de uma microinstrução podem oferecer até 64 formatos diferentes com relação a quantidade de operandos, podendo também oferecer formatos quanto a modos de desvio condicional, incondicional e outras funcionalidades.

IV. VALIDAÇÃO DA MICROARQUITETURA

Esta aplicação envolve a execução de um algoritmo simples para a obtenção do fatorial de um número de diferentes formas, utilizando instruções fixas e instruções reconfiguradas, analisando para cada caso o tempo de execução. O algoritmo que ilustra a aplicação é apresentado na Fig. 7.

Algoritmo Fatorial:

```
            Início: Leia Acc;          (01)
            A <= 1;                    (02)
            B <= 1;                    (03)
            C <= 1;                    (04)
            Acc <= Acc - C;            (05)
            Se Acc = 0 então           (06)
                 vá para Fim;
            Fim Se;
(Loop:)     B <= B + C;                (07)
            A <= A * B;                (08)
            Acc <= Acc - C;            (09)
            Se Acc != 0 então          (10)
                 vá para Loop;
            Fim Se;
(Fim:)      Acc <= A;                  (11)
```

Fig. 7. Algoritmo da aplicação

Inicialmente, esta aplicação foi implementada utilizando apenas instruções fixas utilizando o conjunto de instruções 2. A execução de uma instrução fixa demora no mínimo 9 estágios para ser finalizada, incluindo busca, decodificação, execução, armazenamento e verificação de pedido de interrupção, com exceção das instruções de desvio condicional, que necessitam de 1 estágio a mais para a estrutura de decisão. A Tabela 3 mostra a sequência de execução da aplicação para o fatorial do número 5, em que cada linha da tabela representa a execução de uma instrução distinta.

TABELA 3. PROGRAMA COM INSTRUÇÕES FIXAS

PC	IR	Mnemônico	Efeito	Estados
0X00	0XFF00000F	Muda_modo	modo supervisor	9
0X01	0XFE000001	Muda_conjunto	conjunto 2	9
0X02	0X02000001	LOCO 01	Acc = 1	9
0X03	0X0D204000	ADDR A, Acc, 0	A = Acc + 0	9
0X04	0X0D284000	ADDR B, Acc, 0	B = Acc + 0	9
0X05	0X0D304000	ADDR C, Acc, 0	C = Acc + 0	9
0X06	0X02000005	LOCO 05	Acc = 0X05	9
0X07	0X0E084C00	SUBR Acc, Acc, C	Acc = Acc - C	9
0X08	0X0C00000D	JZER 0D	IF Acc = 0 PC = 0X0D	9
0X09	0X0D294C00	ADDR B, B, C	B = B + C	9
0X0A	0X0F210A00	MULR A, A, B	A = A * B	9
0X0B	0X0E084C00	SUBR Acc, Acc, C	Acc = Acc - C	9
0X0C	0X0B000009	JNZE 09	IF *Acc* ≠ 0 PC = 0X09	10
0X0D	0X0D090000	ADDR Acc, A, 0	Acc = A + 0	9

Para se executar este programa, foram necessários 237 estágios. Como cada microinstrução necessita de 4 subciclos de relógio com período de 20ns, resultando em 80ns por estágio, o tempo de execução desta aplicação é de 18.960ns, embora o tempo real de execução da aplicação dependa do tempo de resposta da memória RAM para cada solicitação de leitura. A aplicação foi finalizada aos 19,46µs.

A segunda implementação utilizou instruções customizadas criando as instruções INC R e DEC R, não existentes em nenhum conjunto de instruções pré-existente. A Tabela 4 mostra o programa do fatorial utilizando as novas instruções e a configuração das microinstruções de cada CodOp reconfigurado.

TABELA 4. PROGRAMA COM INSTRUÇÕES RECONFIGURADAS

PC	IR	Mnemônico	Efeito	Estados
0X00	-	Criar INC R	R = R + 1	6
0X00	-	Criar DEC R	R = R - 1	6
0X00	0XFF00000F	Muda_modo	modo supervisor	9
0X01	0XFE00001F	Muda_conjunto	conjunto 2	9
0X02	0X02000005	LOCO 05	Acc = 0X05	9
0X03	0X1A084000	DEC Acc	Acc = Acc - 1	11
0X04	0X0C00000B	JZER 0B	IF Acc = 0 PC = 0X0B	9
0X05	0X10210000	INC A	A = A + 1	11
0X06	0X10294000	INC B	B = B + 1	11
0X07	0X10294000	INC B	B = B + 1	11
0X08	0X0F210A00	MULR A, A, B	A = A * B	9
0X09	0X1A084000	DEC Acc	Acc = Acc - 1	11
0X0A	0X0B000007	JNZE 07	IF *Acc* ≠ 0 PC = 0X07	9/10
0X0B	0X0D090000	ADDR Acc, A, 0	Acc = A + 0	9

INC R	INC R	0X6460041	0X1A	00
DEC R	DEC R	0X6468041	0X10	01

Todas as instruções que possuem apenas uma microinstrução necessitam de 11 estágios para serem executadas. Uma instrução customizada necessita de 2 ciclos a mais do que uma instrução fixa devido aos ciclos extras que cada componente RAM da unidade de reconfiguração necessita para disponibilizar o dado solicitado após a solicitação de leitura.

Esta aplicação foi executada em um total de 253 estágios, incluindo os estágios necessários para a configuração das instruções. Multiplicando este valor pelo período de relógio total (80ns), o tempo mínimo de execução desta aplicação é de 20.240ns e o tempo real desta simulação, considerando a variação de tempo durante as operações de leitura, é de 20,765µs.

A partir da segunda vez que esta aplicação é executada, não se faz necessário repetir o processo de reconfiguração das instruções. Desse modo, o total de estados necessários para executar o fatorial é reduzido para 241, resultando em um tempo mínimo de simulação de 19.517ns.

A última implementação consiste na criação de uma instrução reconfigurável FAT Acc, A, B. Esta instrução necessita de 3 microinstruções, que substituem o comportamento das instruções INC B, MULR A, A, B e DEC Acc, combinando estas instruções em apenas um código de operação, reduzindo a quantidade de buscas na memória e consequentemente aumentando o desempenho de sua aplicação com relação a quantidade de ciclos de relógio.

A instrução FAT necessita de 19 estágios para ser executada, sendo 10 para a primeira microinstrução, 4 estágios para a segunda e 5 estágios para a última. A partir da segunda microinstrução, os estágios de busca e decodificação não são utilizados, já que ainda está sendo executado o mesmo CodOp. A Tabela 5 mostra o programa completo para executar a aplicação, incluindo as microinstruções das instruções customizadas.

TABELA 5. PROGRAMA FINAL DA APLICAÇÃO TESTE

PC	IR	Mnemônico	Efeito	Estados
0X00	-	Criar FAT C, A, B	INC R	6
0X00	-	Continua FAT C, A, B	MULR A, A, B	6
0X00	-	Continua FAT C, A, B	Dec R	6
0X00	0XFF00000F	Muda_modo	modo supervisor	9
0X01	0XFE00001F	Muda_conjunto	conjunto 2	9
0X02	0X02000005	LOCO 05	Acc = 0X05	9
0X02	-	Criar DEC R	R = R - 1	6
0X03	0X15084001	DEC Acc	Acc = Acc - 1	11
0X04	0X0C000009	JZER 09	IF Acc = 0 PC = 0X09	9
0X04	-	Criar INC R	R = R + 1	6
0X05	0X18210001	INC A	A = A + 1	11
0X06	0X18294001	INC B	B = B + 1	11
0X07	0X10090A00	FAT Acc, A, B	Acc!	19
0X08	0X0B000007	JNZE 07	IF $Acc \neq 0$ PC = 0X07	9/10
0X09	0X0D090000	ADDR Acc, A, 0	Acc = A + 0	9

Instrução	Operação	MData	MAddr	Endereço RAM Reconf.
FAT C, A, B	INC R	0X2860040	0X10	00
FAT C, A, B	MULR A, A, B	0X0430040	0X10	01
FAT C, A, B	DEC R	0X4C68041	0X10	02
DEC R	DEC R	0X6468041	0X15	03
INC R	INC R	0X6460041	0X18	04

A primeira execução, considerando o tempo necessário para reconfigurar a instrução FAT, necessita de 18,498µs, em um total de 223 estados. A segunda execução, por não necessitar que o processo de reconfiguração seja realizado novamente, é realizada em 16,145µs.

A Tabela 6 contém um quadro comparativo das três simulações realizadas do mesmo algoritmo. O terceiro exemplo (a partir da 2ª execução) apresentou melhorias em todos os aspectos com relação ao primeiro exemplo. Com relação ao consumo de energia, por meio do aplicativo PowerPlay disponível na ferramenta Quartus II,verificou-se que houve uma redução de 0,58µJ, ou seja, houve um ganho de 22,5%.

TABELA 6. QUADRO COMPARATIVO DE CONJUNTOS DE INSTRUÇÕES

Exemplo	Tempo	Potência	Energia
Exemplo 1	$19,460\mu$ s	131,35mW	$2,54\mu J$
Exemplo 2 (1^a execução)	$20,765\mu$ s	135,84mW	$2,80\mu J$
Exemplo 2 (2^a execução)	$19,517\mu$ s	121,30mW	$2,36\mu J$
Exemplo 3 (1^a execução)	$18,498\mu$ s	135,45mW	$2,50\mu J$
Exemplo 3 (2^a execução)	$16,145\mu$ s	121,95mW	$1,96\mu J$

V. SÍNTESE DO PROCESSADOR

Para a síntese e validação do processador, foi utilizado um dispositivo da família Cyclone II e o ambiente de desenvolvimento Quartus II, da Altera. Os resultados da síntese mostram que foram utilizados 3.387 elementos lógicos disponíveis do dispositivo, o que corresponde a 11% de sua área (33.216 Elementos Lógicos). A unidade de reconfiguração consome em sua maior parte bits de memória disponíveis no dispositivo FPGA, de modo que o acréscimo na quantidade de elementos lógicos é dado apenas pelo uso dos registradores e lógica de seleção do endereço da memória RAM de dados.

Para verificar o impacto do acréscimo de cada conjunto de instruções, foram realizadas sínteses utilizando cada conjunto de instruções isolado e agrupando um ou mais conjuntos, analisando o total de elementos lógicos e a quantidade de estados necessários para a execução de instruções fixas, excluindo os estágios iniciais e de interrupção, apresentando os resultados na Tabela 7.

TABELA 7. SÍNTESE DOS CONJUNTOS DE INSTRUÇÕES

Conjunto de instruções	LE	Estados de execução	Total de estados	Área do dispositivo
Conjunto 1	822	59	90	3%
Conjunto 2	828	53	84	3%
Conjunto 3	909	87	118	4%
Conjuntos 1 e 2	1023	64	95	5%
Conjuntos 1 e 3	1081	87	118	6%
Conjuntos 2 e 3	1175	87	118	7%
Conjuntos 1, 2 e 3	1380	87	118	7%

Os valores obtidos mostram que a diferença entre o menor conjunto de instruções isolado e o conjunto de instruções completo é de 558 elementos lógicos, ou seja, um aumento de aproximadamente 4% da capacidade do dispositivo.

VI. CONCLUSÕES

A utilização de diversos conjuntos de instruções em uma mesma microarquitetura apresenta-se como uma prova de conceito de uma microarquitetura com características genéricas, híbridas e adaptativas, que mostra ser possível selecionar um conjunto de instruções que seja mais adequado a uma determinada aplicação.

Utilizar uma arquitetura microprogramada faz com que a complexidade de funcionamento do processador seja destinada à unidade de controle, a qual utiliza microinstruções para

prover os estágios do ciclo de vida de uma instrução. Os processadores RISP podem alterar formatos e estruturas de um microprograma, sem a necessidade de modificações do caminho de dados. Portanto, a reconfiguração é realizada nas instruções, e não no *hardware* do dispositivo.

Este conceito pode ser estendido para o uso de diferentes arquiteturas de conjuntos de instruções (ISA's) por um processador, desde que o caminho de dados possa executar as instruções e os formatos específicos de cada uma das arquiteturas. Poder alternar entre diferentes arquiteturas torna um processador adaptativo, que pode ser integrado a outros componentes ou processadores em um ambiente de multiprocessamento homogêneo sem alterar suas características particulares de organização e estrutura física.

A possibilidade de criar instruções reconfiguradas em tempo de execução da máquina permite a possibilidade de expansão do conjunto de instruções original ou ainda a união de instruções já existentes em um único código de operação, otimizando o desempenho da arquitetura com relação a quantidade de ciclos de relógio. A unidade de reconfiguração pode, portanto, criar novas instruções para a arquitetura, desde que o caminho de dados do processador possa prover sua execução e respeitando os formatos pré-determinados na especificação da arquitetura. Estas instruções, depois de criadas, serão executadas de forma análoga às instruções fixas do processador.

Tais características fazem com que esta arquitetura possa se tornar uma ferramenta de auxílio acadêmico em disciplinas que envolvam arquitetura e organização de computadores, visto que é necessário ao projetista e ao programador o conhecimento acerca de cada ISA com relação a seu formato, tipos de instruções e endereçamento, bem como o funcionamento de todos os blocos funcionais da microarquitetura, como ULA, deslocadores, registradores, multiplexadores e demais componentes de baixa e média complexidade de *hardware*. Ao utilizar o processador academicamente, faz-se necessário o desenvolvimento de *softwares* básicos como montadores, compiladores e simuladores caso sejam utilizados conjuntos de instruções genéricos ou experimentais, ou utilizar os aplicativos e ferramentas já existentes e disponibilizados para a comunidade científica e acadêmica caso seja inserida ou utilizada uma ISA existente, como o padrão MIPS.

REFERÊNCIAS

[1] A. M. Adário, S. Bampi, R. P. Jacobi, "Reconfigurable architectures". In: UFRGS Microelectronics Seminar. Porto Alegre, RS, Brasil.1997. p. 133–136. ISBN T.I. - 650 - CPGCC.

[2] F. Barat, P. Lauwereins, "Reconfigurable Instruction Set Processors: A Survey", in: RSP '00: Proceedings of the 11th IEEE International Workshop on Rapid System Prototyping (RSP 2000). Washington, DC, USA: IEEE Computer Society, 2000. p. 168. ISBN 0-7695-0668-2.

[3] L. H. Moller, F. G. Moraes, N. L. Calazans, "Processadores Reconfiguráveis: estado da arte". In: XI Workshop Iberchip, 2005, Salvador. Brasil.

[4] P. M. Athanas,, H. F. Silverman, "Processor Reconfiguration Through Instruction-Set Metamorphosis". IEEE Computer, Vol. 26, no. 3 pp.11-18. 1993

[5] R. Razdan, M. D. Smith, "A high-performance microarchitecture with hardware programmable functional units". In: MICRO 27: Proceedings of the 27th annual international symposium on Microarchitecture. New York, NY, USA: ACM Press, 1994. p. 172–180. ISBN 0-89791-707-3

[6] M. Wirthlin, B. Hutchings,. "A Dynamic Instruction Set Computer". In: FPGAs for Custom Computing Machines (FCCM), 1995, pp. 99-107.

[7] R. Wittig, P. Chow, "OneChip: an FPGA Processor with Reconfigurable Logic". In: FPGAs for Custom Computing Machines (FCCM), 1996, pp. 126-135.

[8] S. Hauck, T. Fry, M. Hosler, J. Kao. "The Chimaera Reconfigurable Functional Unit". In: FPGAs for Custom Computing Machines (FCCM), 1997, pp. 87-96.

[9] J. Hauser, J. Wawrzynek, "Garp: a MIPS Processor with a Reconfigurable Coprocessor". In: FPGAs for Custom Computing Machines (FCCM), 1997, pp. 12-21.

[10] H. Singh, G. Lu, E. Filho, R. Maestre, M. Lee, F. Kurdahi, and N. Bagherzadeh. "MorphoSys: case study of a reconfigurable computing system targeting multimedia applications". 2000. In *Proceedings of the 37th Annual Design Automation Conference* (DAC '00). ACM, New York, NY, USA, 573-578.

[11] R. Hartenstein, R. Kress, "A Datapath Synthesis System for the Reconfigurable Datapath Architecture". In: Asia and South Pacific Design Automation Conference (ASP-DAC), 1995, pp. 479-484.

[12] E. Mirsky, A. DeHon, "MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources". In: FPGAs for Custom Computing Machines (FCCM), 1996, pp. 157-166.

[13] J. Becker, T. Piontek, M. Glesner, "DReAM: A Dynamically Reconfigurable Architecture for Future Mobile Communications Applications". In: Field Programable Logic and Applications (FPL), 2000, pp. 312-321.

[14] F.G. Moraes and N. L. V. Calazans. "R8 Processor Architecture and Organization Soecification and Desine Guidelines". 2003. disp. em http://www.inf.pucrs.br/~gaph/Projects/R8/public/R8_arq_spec_eng.pdf

[15] A. Azevedo, R. Soares and I. S. Silva. "A New Hybrid Parallel/Reconfigurable Architecture: The X4CP32". SBCCI, page 225-230. IEEE Computer Society, 2003.

[16] A. Mihal, S. Weber, K., Keutzer." Sub-RISC processors". In P. Ienne and R. Leupers, editors, Customizable Embedded Processors: Design Technologies and Applications, chapter 13. Elsevier, 2006

[17] A. S. Tanenbaum, "Organização Estruturada de Computadores", 5ª Edição, Pearson Prentice Hall, São Paulo, Brasil.