Simulação de execução de instruções do processador de núcleo virtual Nios II

Willian dos Santos Lima[1], Renata Spolon Lobato[1], Antonio Carlos Fernandes da Silva[2] e Roberta Spolon Ulson[3]

[1] Departamento de Ciências de Computação e Estatística, Instituto de Biociências, Letras e Ciências Exatas, UNESP – Universidade Estadual Paulista, São José do Rio Preto, SP, Brasil.
willian.lima@sjrp.unesp.br, renata@ibilce.unesp.br

[2]Coordenação de Informática, UTFPR – Universidade Tecnológica Federal do Paraná, Cornélio Procópio, PR, Brasil.
antonio@utfpr.edu.br

[3]Departamento de Computação, Faculdade de Ciências, UNESP – Universidade Estadual Paulista, Bauru, SP, Brasil.
roberta@fc.unesp.br

Resumo. A computação reconfigurável tem sido alvo de diversas pesquisas no ramo das ciências de computação e, com isso, o estudo sobre dispositivos reconfiguráveis torna-se relevante. Tais dispositivos podem estar relacionados com processadores de núcleo virtual, como o processador Nios II da Altera®. Este processador possui todas as propriedades de um processador comum, o que inclui a especificação de um conjunto de instruções, diferindo no fato de que é projetado em software, e não em hardware, sendo necessária a configuração de um FPGA para sua efetiva implementação. Neste trabalho descrevemos um simulador capaz de ler um arquivo contendo um conjunto de instruções para o processador Nios II e de interpretá-lo de forma a simular sua execução.

Palavras-chave: computação reconfigurável, *reconfigware*, *soft-core*, Nios II, núcleo virtual, FPGA, simulador.

1 Introdução

O Nios II é um processador de núcleo virtual, desenvolvido pela Altera® [2], que possui a propriedade de poder ser configurado para determinadas aplicações. É possível retirar alguns de seus componentes e incluir outros, bem como excluir ou incluir instruções em seu conjunto de instruções. Por este motivo, o Nios II é um dispositivo interessante no contexto da computação reconfigurável [5, 7].

O fato de o Nios II ter núcleo virtual, ou *soft-core*, significa que ele é projetado em software, e não em hardware como os processadores comuns. Isto permite ao processador sofrer alterações antes de ser gravado em um dispositivo reconfigurável, como um FPGA (*Field Programmable Gate Array*), onde ele efetivamente funciona [6].

O processador Nios II é um processador RISC (*Reduced Instruction Set Computer*) de propósito geral, cujo sistema é equivalente a um microcontrolador com uma UCP (Unidade Central de Processamento) e uma combinação de periféricos e memória em um único *chip* [2]. Este processador possibilita a adição de instruções personalizadas ao seu conjunto de instruções quando se deseja aceleração para determinado trecho de um algoritmo, permitindo maior flexibilidade e adição de novas funcionalidades à ULA (Unidade Lógico Aritmética) do Nios II [1].

Devido às propriedades do processador Nios II, para que seja possível a execução de um programa, é necessário que haja a configuração da placa de FPGA na qual o hardware referente ao processador será gerado, de modo a permitir que um código escrito para o Nios II possa ser executado.

Para evitar a necessidade de realizar a configuração da placa de hardware para simplesmente testar um aplicativo escrito para o processador Nios II, conforme detalhado na seção 3, construiu-se um simulador do conjunto de instruções do mesmo, capaz de ler instruções no formato exigido pelo processador e emular os resultados.

O presente trabalho descreve este simulador. Na seção 2 é apresentado um detalhamento do conjunto de instruções do Nios II. Uma revisão dos trabalhos correlatos é feita na seção 3. Na seção 4 descreve-se a construção e o funcionamento do simulador. Alguns testes e seus respectivos resultados são mostrados na seção 5. Na seção 6 são apresentadas as conclusões do trabalho.

2 Instruções do processador Nios II

O processador Nios II possui 32 registradores de propósito geral, sendo que 10 deles são reservados ao controle do programa e não podem, por exemplo, armazenar valores de variáveis.

Os registradores são manipulados por instruções do processador, as quais são divididas em 3 tipos:

- Tipo I: pertencem a essa classificação as instruções que basicamente manipulam endereços de memória ou constantes como valores imediatos, tal como a adição de uma constante a um registrador ou um salto para determinado endereço;
- Tipo R: esse tipo abrange, em geral, instruções de manipulação e operação entre registradores, tal como adição do valor de um registrador a outro ou salto para um endereço contido em um registrador;
- Tipo J: apenas uma instrução pertence ao tipo J, a instrução `call`, que é utilizada na chamada de sub-rotinas.

Todas as instruções do Nios II são representadas através de palavras de 32 bits. Entretanto, o formato da palavra, ou seja, a forma com que os bits são interpretados pelo processador, varia de acordo com o tipo da instrução. A Tabela 1 resume os formatos das palavras de acordo com seu tipo.

Cada instrução é identificada através de seu *opcode*, um número inteiro representado por 6 bits. No caso das instruções de tipo R, ele é capaz apenas de identificar este tipo, e não a instrução propriamente dita. Ou seja, toda instrução do

tipo R possui o mesmo *opcode*. Elas são diferenciadas através de uma extensão de *opcode*, denominada *opx* e representada por outros 6 bits, o que justifica o fato de que para instruções do tipo R são utilizados 12 bits de identificação.

Tabela 1. Formatos das palavras de instruções do Nios II.

Tipo de instrução	Campos na palavra
I	2 endereços para registrador (5 bits cada); 1 valor imediato (16 bits); Identificação da instrução (6 bits).
R	3 endereços para registrador (5 bits cada); 1 valor imediato (5 bits); Identificação da instrução (12 bits).
J	1 valor imediato (26 bits); Identificação da instrução (6 bits).

Algumas instruções do Nios II são ditas pseudo-instruções, pois não são efetivamente implementadas no projeto do processador, mas sim através da execução de outras instruções.

Um exemplo de pseudo-instrução é a instrução `movi`, a qual possui dois parâmetros, sendo que o primeiro é o endereço de um registrador e o segundo é um valor imediato. O valor imediato é movido para o registrador. Esta instrução existe apenas conceitualmente, sendo que não há a criação de uma palavra para ela, mesmo porque pseudo-instruções não têm *opcode* definido. Assim, ela é substituída pela instrução `addi`, que possui o *opcode* 0x04 (tipo I) e três parâmetros, sendo os dois primeiros endereços para registradores e o último um valor imediato. Esse valor imediato é somado com o valor do registrador no segundo parâmetro e armazenado no primeiro registrador. Desta forma, quando uma instrução `movi` é encontrada, a seguinte tradução é realizada pelo *assembler* antes da geração da palavra, em que `rX` é um registrador de propósito geral:

```
movi rX, 100   →   addi rX, r0, 100
```

3 Trabalhos correlatos

SimNios [8] e ModelSim [9] são exemplos de simuladores para o Nios II. Ambos possuem como entrada um programa cujo alvo são as instruções do processador Nios II, em conjunto com a configuração da placa de FPGA previamente selecionada no momento da criação do programa. Assim sendo, os simuladores simulam a execução do programa sobre o hardware selecionado.

Para que um programa possa ser simulado em qualquer um dos dois simuladores supracitados, é necessário que os códigos tenham sido gerados por meio de um aplicativo capaz de criar toda a configuração de uma placa de FPGA. O usuário de tal aplicativo precisa escolher o hardware destino e então compilar o programa.

Com os passos descritos no parágrafo anterior realizados, o usuário pode fazer uso dos simuladores, os quais simulam as instruções do Nios II sobre o hardware selecionado apresentando os estados dos registradores.

Desta forma, esses simuladores para o processador Nios II exigem que o usuário opte pelo hardware de destino, não sendo possível a simulação caso isto não aconteça. Assim, é interessante a criação de um simulador em que a configuração do hardware de destino não seja necessária.

4 O simulador

O simulador foi implementado em linguagem C-ANSI [3], o que permite que ele possa ser compilado por qualquer compilador que suporte os padrões da linguagem C para qualquer sistema operacional ou arquitetura.

A entrada do simulador deve conter uma seqüência de palavras representando instruções do Nios II. O simulador então executa as instruções e ao final exibe o estado de todos os registradores e o tempo gasto na execução. As seções subseqüentes detalham o formato de entrada para o simulador, como o processamento é realizado e quais dados ele proporciona como saída.

4.1 Entrada de dados

O simulador deve ser acionado tendo como parâmetro o nome do arquivo binário que contém a seqüência de instruções do Nios II a serem processadas. Este arquivo deve conter uma seqüência de palavras de 4 bytes cada uma de acordo com o especificado na referência do processador Nios II [2] e mencionado na seção 2. As palavras deverão estar no formato *little-endian* [4].

Nenhuma informação sobre configuração de placas de FPGA é necessária para o simulador. Apenas a seqüência de instruções é requerida, pois o intuito é simular o programa em si de maneira a verificar se as instruções do Nios II estão corretamente elaboradas e se não há erros na formação das palavras.

Como o simulador foi desenvolvido para não depender da configuração de hardware, comandos de entrada e saída de dados através de periféricos não são simulados.

4.2 Processamento do arquivo binário

As palavras contidas no arquivo de entrada são lidas uma a uma e interpretadas de forma a identificar primeiramente a que tipo de instrução pertence. Feito isto, os bits da palavra são interpretados adequadamente obtendo-se, assim, os parâmetros para a instrução identificada.

Com a instrução e seus parâmetros identificados, há uma checagem em relação à validade daquela instrução, ou seja, se aquela instrução não tenta escrever em registrador somente de leitura ou se os parâmetros não condizem com a operação a ser realizada.

No caso da instrução estar corretamente especificada na palavra, simula-se o efeito de sua execução. Cada instrução poderá alterar algum dos elementos constituintes do simulador, os quais são listados a seguir:

- Lista de registradores de propósito geral: arranjo de 32 *containers* para números inteiros de 32 bits que simula o conjunto de registradores de propósito geral do Nios II;
- Lista de registradores de controle: arranjo de 6 elementos inteiros de 32 bits que simula o conjunto de registradores de controle [2] do Nios II;
- Memória interna: estrutura de dados do tipo lista que simula a porção de memória utilizada pelo programa em execução e por dados manipulados pelo programa;
- Contador de programa (PC – *Program Counter*): estrutura cujo valor significa a posição do controle do programa dentro da memória interna.

A cada instrução executada, exceto as de desvio, o contador de programa é incrementado de 4 unidades, para que seja posicionado no início da próxima palavra a ser interpretada.

Quando o contador de programa aponta para uma posição de memória que não pertence ao código o programa termina. Neste caso, o simulador pára a interpretação e exibe os resultados.

4.3 Saída de dados

Como supracitado, os principais dados de saída do simulador são o estado do conjunto de registradores de propósito geral e o tempo gasto na simulação. Cada um dos 32 registradores de propósito geral tem seu valor mostrado, sendo possível a análise de quais registradores efetivamente foram modificados e manipulados pelo programa simulado. A medida de tempo poderá ser utilizada na análise de complexidade de um programa em relação a outro.

Além disso, se um erro de interpretação de instrução é encontrado ou se uma instrução manipula indevidamente seus parâmetros, uma mensagem de erro de execução será exibida e a simulação não prossegue.

5 Testes

Para testar o simulador, foi criado um *assembler* (ou montador) do Nios II, o qual gera as palavras correspondentes a cada uma das instruções através da especificação de seus mnemônicos.

Inicialmente, foram criados dois programas, sendo um para cálculo do fatorial do número 10 (Figura 1) e outro para o cálculo dos 20 primeiros números da seqüência de Fibonacci (Figura 2), iniciando-se a partir do segundo elemento da seqüência.

No primeiro programa são utilizados 4 registradores de propósito geral, sendo que 2 servem apenas como auxiliares (r2 e r9), com o resultado do cálculo sendo colocado no registrador r3. Uma atenção especial merece ser dada às instruções de salto, `beq` e `br`, em relação ao valor imediato que é passado como parâmetro. Este valor significa o *offset* a ser levado em conta no desvio, e é sempre especificado em número de bytes.

```
addi r4, r0, 10     ;coloca em r4 o valor 10
add  r3, r0, r4     ;coloca em r3 o valor de r4
addi r2, r0, 1      ;coloca em r2 o valor 1
beq  r4, r2, 16     ;salta 16 bytes à frente se r4 == r2
subi r9, r4, 1      ;faz r9 conter o valor de r4 - 1
mul  r3, r9, r3     ;faz r3 conter o valor de r9 * r3
subi r4, r4, 1      ;decrementa o valor contido em r4
br   -20            ;salta 20 bytes para trás
```

Figura 1. Programa para cálculo do fatorial do número 10.

Quando uma instrução está sendo executada, o contador de programa já estará posicionado na instrução imediatamente posterior da ordem linear de especificação de instruções. Por este motivo, saltos para posições de programa anteriores à instrução do desvio devem possuir o *offset* que leva em conta também a quantidade de bytes da própria instrução de salto, como é o caso da última instrução do programa apresentado na Figura 1.

O registrador r0, utilizado em ambos os programas, é um registrador apenas de leitura e sempre contém o valor zero, por isso é utilizado geralmente como elemento neutro em adições.

No programa da Figura 2 não há repetição de bloco de código como ocorre no programa da Figura 1, contudo 18 registradores de propósito geral foram utilizados, sendo que cada um deles armazena o valor do elemento correspondente da seqüência de Fibonacci.

```
addi r2,  r0,  1
addi r3,  r0,  2
add  r4,  r2,  r3
add  r5,  r3,  r4
add  r6,  r4,  r5
add  r7,  r5,  r6
add  r8,  r6,  r7
add  r9,  r7,  r8
add  r10, r8,  r9
add  r11, r9,  r10
add  r12, r10, r11
add  r13, r11, r12
add  r14, r12, r13
add  r15, r13, r14
add  r16, r14, r15
add  r17, r15, r16
add  r18, r16, r17
add  r19, r17, r18
add  r20, r18, r19
```

Figura 2. Programa para cálculo dos 20 primeiros números da seqüência de Fibonacci.

A Figura 3 apresenta o estado dos registradores manipulados por cada um dos programas testados.

Por padrão, os registradores inicialmente possuem valor zero, apesar disto não estar definido na referência do processador Nios II [2], que não especifica valores

iniciais para os registradores. Devido a uma questão de simplicidade, os registradores não utilizados pelos programas não são mostrados; para todos os outros, seu valor final é exibido.

Conforme pode ser confirmado através da Figura 3, o simulador interpretou corretamente os dois programas testados, fornecendo os resultados esperados.

Foram realizados outros dois testes, sendo criados, para tanto, os programas apresentados na Figura 4 e na Figura 6, que possuem a propriedade de, respectivamente, utilizar instruções variadas do Nios II e fazer com que o simulador entre em *loop* infinito. Nesses programas, algumas pseudo-instruções, grafadas em negrito, são utilizadas.

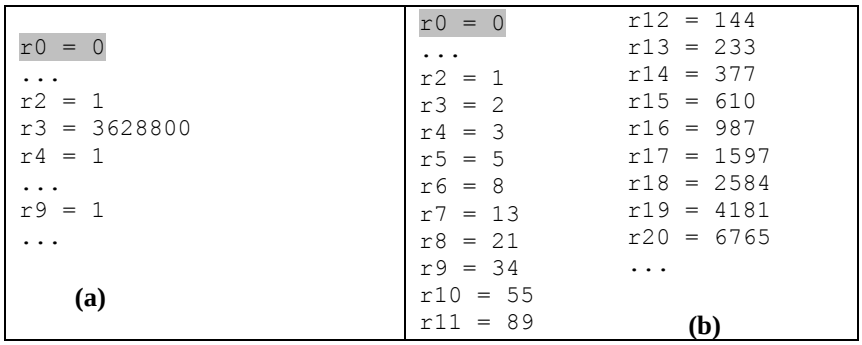

Figura 3. Estado final dos registradores **(a)** para o programa da Figura 1 e **(b)** para o programa da Figura 2.

O programa da Figura 4 serve para testar se ocorre *overflow* após duas operações de somas. Um *overflow* pode ser detectado quando a soma de dois números positivos resulta em negativo ou quando a soma de dois números negativos resulta em um número positivo [2].

Caso tenha ocorrido um *overflow* na primeira soma, o valor do registrador r8 será diferente de zero e, para a segunda soma, o valor de r18 será diferente de zero. Para tanto, são realizadas comparações de sinais através das operações de "ou exclusivo" seguidas de uma operação "e" entre os resultados, a qual escreverá em um registrador o resultado do teste.

Na segunda metade do programa (abaixo das 3 instruções `nop`), é feita a atribuição do número 2.147.450.879 (0x7FFF7FFF) ao registrador r10 e, em seguida, as mesmas operações anteriores sobre os operandos e o resultado.

A instrução `nop` é uma pseudo-instrução e proporciona um caso especial na especificação de sua palavra correspondente: `add r0, r0, r0`. Como já explicado anteriormente, o registrador r0 é apenas de leitura, porém, quando temos uma operação de adição em que todos os parâmetros são r0, isto significa que nenhuma operação será realizada, tornando-se um caso particular na interpretação dessa palavra.

```
nop                      ;instrução 'no operation'
movi r3, 7               ;r3 := 7
movi r4, 8               ;r4 := 8
add  r5, r3, r4         ;r5 := r3 + r4
xor  r6, r5, r3         ;r6 := r5 ^ r3 (comp. de sinais)
xor  r7, r5, r4         ;r7 := r5 ^ r4 (comp. de sinais)
and  r8, r6, r7         ;r8 := r6 & r7
nop
nop
nop
movi r10, 32767         ;r10 := 32767 (0x7FFF)
orhi r11, r10, 32767    ;r11 := r10 | 0x7FFF:0000
mov  r10, r11           ;r10 := r11
add  r15, r10, r11      ;r15 := r10 + r11
xor  r16, r15, r10      ;r16 := r15 ^ r10 (comp. sinais)
xor  r17, r15, r11      ;r17 := r15 ^ r11 (comp. sinais)
and  r18, r16, r17      ;r18 := r16 & r17
nop
```

Figura 4. Programa que faz uso de algumas instruções aritméticas, lógicas e de desvios, bem como pseudo-instruções, as quais estão grafadas em negrito.

A execução do programa da Figura 4 no simulador resultou no conjunto de valores de registradores apresentado na Figura 5.

```
r0 = 0
...
r3 = 7
r4 = 8
r5 = 15
r6 = 8
r7 = 7
r8 = 0
...
r10 = 2147450879
r11 = 2147450879
...
R15 = -65538
r16 = -2147385343
r17 = -2147385343
r18 = -2147385343
...
```

Figura 5. Estado dos registradores após simulação do programa da Figura 4.

No programa da Figura 6 o valor imediato 1.000 é atribuído ao registrador r2 e o valor 1 ao registrador r11. A seguir, é realizada uma operação de divisão entre os valores dos registradores e armazenamento em r2. A instrução `cmple` compara se o valor contido em r2 é menor ou igual ao que está contido em r0 e armazena em r3 o valor 1 caso seja verdade e 0 caso seja falso. A próxima instrução é uma instrução de salto condicional, que analisa se os valores em r3 e r11 são diferentes para saltar 12 bytes para trás.

Como a única operação que modifica o valor de r2 é a de divisão por r11, e como r11 sempre possui valor 1, o programa entrará em *loop*, fato que pôde ser comprovado através de sua execução sobre o simulador.

A pseudo-instrução `cmple` pode ser implementada através da instrução `cmpge` (*opcode* 0x3A e *opx* 0x08 – tipo R) com uma alteração na ordem dos operandos:

```
cmple rX, rA, rB   →   cmpge rX, rB, rA
```

```
movi  r2,  1000      ;r2 := 1000
movi  r11, 1         ;r11 := 1
div   r2,  r2,  r11  ;r2 := r2 / r11
cmple r3,  r2,  r0   ;r3 := r2 <= r0
bne   r3,  r11, -12  ;salta -12 bytes de r3 != r11
```

Figura 6. Programa que força o simulador a entrar em loop infinito.

Outros dois programas foram submetidos ao simulador, sendo que um deles envolvia uma instrução de adição sobre o registrador r0 com outros registradores como operandos, ocasionando o erro de "tentativa de escrita em registrador somente leitura". No outro, uma palavra foi modificada no arquivo binário de modo a gerar o erro "impossível decodificar instrução" no simulador. Em ambos os casos, a execução foi terminada, como esperado.

6 Conclusões

O simulador construído tem como objetivo a simulação de instruções pertencentes ao conjunto de instruções do processador de núcleo virtual Nios II da Altera®. O trabalho mostrou que o simulador não está associado a qualquer hardware, o que o limita a programas que não realizam entrada e saída de dados através de periféricos. Entretanto, esta propriedade é relevante quando há a necessidade de testar implementações em baixo nível de programas para o Nios II, bem como compiladores e/ou montadores para esta arquitetura, permitindo que as instruções possam ser validadas antes da configuração de um dispositivo reconfigurável para implementação do Nios II.

A principal contribuição deste trabalho está no fato de que foi desenvolvida uma ferramenta de validação de seqüências de instruções para o Nios II sem a necessidade de configuração de hardware. Dessa maneira, proporciona-se economia de tempo e esforço para testes iniciais de programas para o processador ou tradutores cujo código alvo é o conjunto de instruções do Nios II.

O simulador, em seu atual estado de implementação, exibe o estado final do conjunto de registradores, ou seja, após o programa ter terminado sua execução. Como trabalho futuro, propõe-se uma modificação sobre o simulador de forma que ele exiba o estado de registradores conforme as instruções são simuladas, bem como o valor do contador de programa.

Agradecimentos. Os autores agradecem às entidades brasileiras CNPq (pelo financiamento ao autor Willian dos Santos Lima), Fundunesp (pelo apoio às autoras Renata Spolon Lobato e Roberta Spolon Ulson) e UTFPR (pelo apoio ao autor Antonio Carlos Fernandes da Silva).

Referências

1. Altera, http://www.altera.com/ literature/ug/ug_nios2_custom_instruction.pdf.
2. Altera, http://www.altera.com/literature/lit-nio2.jsp.
3. Schildt, H.: C Completo e Total. Makron Books, São Paulo (1997).
4. Henessy, J. L., Patterson, D. A.: Arquitetura de Computadores: Uma Abordagem Quantitativa. Editora Campus, São Paulo (2003).
5. Compton, K,. Hauck, S.: Reconfigurable Computing: A Survey of Systems and Software. ACM Computing Surveys. 34, 171—210 (2002).
6. Cardoso, J. M. P., Neto, H. C.: Compilation for FPGA-based reconfigurable hardware. IEEE Design and Test of Computers. 20, 65—75 (2003).
7. Cheung, P. Y. K.; Constantinides, G. A.; Luk W.; Mencer, O.; Todman, T. J.; Wilton, S. J. E.: Reconfigurable Computing: architectures and design methods. IEE Proceedings – Computer and Digital Techniques. 152, 193—207 (2005).
8. Lauterbach GmbH – Trace32 Microprocessor Development Tools, Emulators, Debuggers, Simulators, http://www.lauterbach.co.uk/frames.html?simnios.html.
9. ModelSim – Downloads, http://www.model.com/downloads/default.asp.