

Universidade Católica Dom Bosco

Altera/NIOS

Renato Bezerra Herebia

UCDB - Campo Grande - MS - JUNHO/2008

Capítulo 1

Introdução

Este relatório apresenta informações sobre a implementação do processador Nios da Altera, que foi modelado pela linguagem ArchC [AC008]. A linguagem foi desenvolvida para criar um simulador de um processador a partir da definição das características da arquitetura desse processador, como tamanho da palavra, memória endereçada, banco de registradores, registradores e estágios de *pipeline*, forma de acesso aos bits (*endianess*), codificação e sintaxe do conjunto de instruções e o comportamento lógico dessas instruções. Essa linguagem permite criar um executável que simula as operações de um processador real, onde é possível submeter como entrada arquivos *assembly*, hexadecimais ou executáveis, e obter como saída o resultado da execução do arquivo de entrada.

O processador NIOS de trinta e dois bits, fabricado pela Altera, foi desenvolvido para ser utilizado em sistemas embarcados. O processador possui quatorze formatos de instruções e seu conjunto de instruções é composto por 72 instruções da arquitetura e 5 instruções que podem ser customizadas pelo desenvolvedor. As informações da arquitetura foram extraídas do manual disponibilizado pela Altera [AN008].

O relatório está dividido em capítulos. No Capítulo 2 serão apresentados as informações sobre o processador Nios, com as características da arquitetura. No Capítulo 3 serão apresentadas informações sobre a modelagem do processador, como recursos implementados e não implementados, bugs/falhas detectadas e não corrigidas, dificuldades na implementação e os experimentos realizados. No Capítulo 4 serão mostradas informações sobre o que foi percebido com o uso da linguagem ArchC e serão propostos trabalhos futuros para complementar esse trabalho. Por último, são apresentadas as referências para mais informações sobre o que foi abordado nesse trabalho.

Capítulo 2

O Processador Altera/NIOS

A implementação do processador foi feita com um *pipeline* de 5 estágios (ver Figura 2.1), onde sua arquitetura é parecida com a dos processadores MIPS [MP008]. É possível retirar algumas informações do diagrama de blocos do processador, onde existem sinais de controle para determinar como são processadas algumas instruções; sinais de entrada com os bits das instruções; e sinais de saída com os dados calculados e endereços para acesso externo. Na parte interna do processador existem registradores, unidade lógica-aritmética, unidade de decodificação e unidades de controle. É possível perceber que o dado de saída sempre estará armazenado no primeiro registrador e que o endereço será computado com o *program counter* por uma outra unidade aritmética.

As instruções são RISC e são emitidas uma a uma. O tamanho das instruções é de dezesseis bits, em contraste com os trinta e dois bits das palavras, que são representadas com os dezesseis bits mais significantes em zero. O acesso às palavras do processador NIOS é feito como big-endian.

Os registradores de propósito geral estão localizados no banco de registradores (BR), que pode ser definido com acesso a 128, 256 ou 512 registradores. Independente do tamanho do BR, são trinta e dois registradores acessíveis. Para definir qual o conjunto de registradores acessíveis, foi implementada uma janela deslizante, que é movimentada por instruções que salvam ou recuperam registradores (SAVE e RESTORE respectivamente). Essas instruções são utilizadas na chamada ou no retorno de um procedimento. Pode-se ver que a janela possui registradores de entrada, registradores locais e registradores de saída (Figura 2.2). Na movimentação da janela (Figura 2.3), os registradores de saída tornam-se os registradores de entrada da próxima janela. E os registradores globais sempre estão fixos por não dependerem de qual chamada está sendo executada.

Ainda no BR, existe o registrador K de onze bits, que recebe valores de instruções como PFX

31 10 0

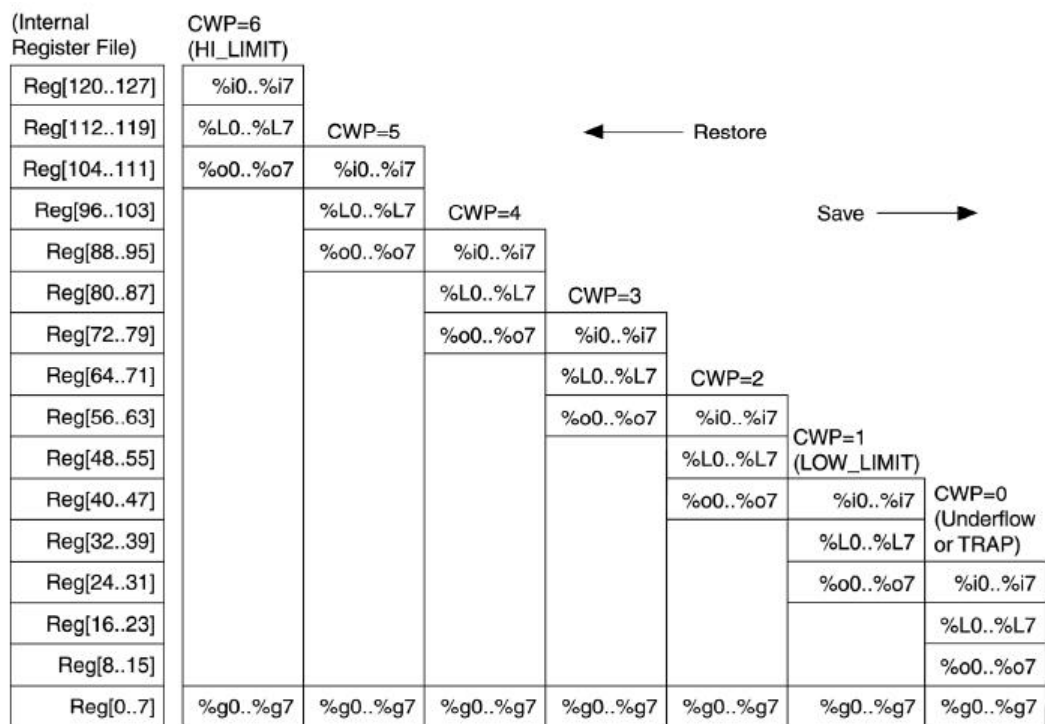


Figura 2.3: *Movimentação de uma janela deslizante.*

Formato	Instruções do Formato
RR	ADD, AND sem PFX, ANDN sem PFX, ASR, CMP, EXT16D, EXT8D, LD, LSL, LSR, MOV, NOP, OR sem PFX, ST, SUB, USR0, XOR sem PFX.
Ri5	ADDI, AND com PFX, ANDN com PFX, ASRI, BGEN, CMPI, IF0, IF1, LSLI, LSRI, MOVHI, MOVI, OR com PFX, SKP0, SKP1, SUBI, XOR com PFX.
Ri4	Não existem instruções desse tipo.
RPi5	LDP, STP.
Ri6	Não existem instruções desse tipo.
Ri8	LDS, STS.
i9	STS16S.
i10	STS8S.
i11	BR, BSR, PFX, PFXIO.
Ri1u	EXT16S, ST16S.
Ri2u	EXT8S, ST8S.
i8v	SAVE.
i6v	TRAP.
Rw	ABS, CALL, FILL16, FILL8, IFRNZ, IFRZ, JMP, LRET, MSTEP, MUL, NEG, NOT, RDCTL, RET, RLC, RRC, SEXT16, SEXT8, SKPRNZ, SKPRZ, ST16D, ST8D, SWAP, TRET, USR1, USR2, USR3, USR4, WRCTL.
i4w	IFS, SKPS.
w	RESTORE.

Tabela 2.1: Formatos das Instruções.

Capítulo 3

Modelagem do Processador Utilizando a Linguagem ArchC

A implementação da arquitetura do processador NIOS foi feita a partir da definição dos requisitos básicos, como tamanho de memória, tamanho da palavra, o tamanho do banco de registradores, os registradores de *pipeline*, os registradores de controle e o método de acesso aos bits (*big endian*). Depois de definida a arquitetura base para o processador, foi implementada a sintaxe das instruções e sua codificação. Nesta fase foram definidos os campos de cada formato de instrução. A última parte da implementação foram definidos os comportamentos das instruções, que inclui como o processador utiliza os registradores de *pipeline* e como realiza a lógica das instruções.

3.1 Recursos Implementados

A definição da arquitetura base para o processador foi feita, com registradores de controle, registradores de pipeline, a codificação das instruções no processador, e foram atestados vários reconhecimentos de instruções pelo processador. O que faltou para a implementação estar completa foi o comportamento de algumas instruções e verificar os comportamentos implementados.

3.2 Recursos Não-Implementados

Os recursos não implementados foram o `%hi(x)` que extrai os 11 bits mais significativos de `x` e o `%lo(x)` que extrai os 5 bits menos significativos de `x`, dentre essas instruções estão também o

$\%xlo(x)$, $\%xhi(x)$ e $x@h$. Também não foram reconhecidas instruções precedidas pela instrução PFX ou PFXIO, como AND, LD, LDP, OR, RDCLT, ST, SUB, WRCTL e XOR. E algumas outras instruções não foram reconhecidas por existir um erro na definição da sintaxe das instruções, pois um registrador sempre é precedido pelo símbolo $\%$. Essas instruções são: FILL16, FILL8, SAVE, ST16D, STS, STS16S, STS8S e todas as instruções que possuem têm um registrador definido para seu uso.

Outros recursos que não foram implementados foram as manipulações dos registradores de controle. Alguns comportamentos de instruções possuem essa manipulação, como ADD, SUB, onde os registradores são usados a partir do resultado da instrução. O uso dos registradores de controle está incompleto. O acesso à memória e à memória *cache* também não foi implementado pelos motivos da Seção 3.4.

3.3 Bugs/Falhas Detectados e Não-Corrigidos

Alguns detalhes da arquitetura demandaram manipulações diferentes para as instruções, como as instruções precedidas pela instrução PFX e as instruções não-precedidas por PFX. Dependendo da precedência, as instruções assumem formatos diferentes, o que dificulta o reconhecimento de uma instrução. A solução foi criar uma instrução para cada formato, onde a sintaxe é diferente e por consequência o comportamento.

A implementação possui várias falhas no reconhecimento das instruções pelo processador. Além de algumas instruções não possuírem a implementação de seus comportamentos. Esses resultados são devido aos motivos citados na Seção 3.4.

3.4 Dificuldades Encontradas Durante o Desenvolvimento do Processador

No início, a maior dificuldade encontrada foi para definir o *pipeline* da arquitetura. O trabalho ficou parado por um tempo por não conseguir definir os estágios e os registradores do *pipeline*, pois a implementação dos comportamentos dependem diretamente dessa definição. Outro problema aconteceu na compilação dos arquivos da arquitetura, onde algumas características da linguagem ArchC 2.0 faltavam ser configuradas para gerar o simulador do processador. Com isso, os testes dos comportamentos foram adiados.

3.5 Experimentos com o Modelo em ArchC

Os experimentos foram divididos em etapas. Depois de serem geradas as codificações das instruções, a primeira etapa foi testar a identificação das instruções. Depois, testar se as instruções são reconhecidas pelo processador. E a seguir, testar os comportamentos lógicos das instruções. Os testes foram programados para seguir esse roteiro:

- Testar a codificação das instruções: depois de criadas as ferramentas do *binutils* para a arquitetura, para cada instrução foi criado um arquivo *.s* com sua sintaxe. Com a ferramenta *elf - as* foi gerado um arquivo *.o* a partir do arquivo *.s*. E com a ferramenta *elf - read* foi lida a seção de texto, onde é apresentada a codificação hexadecimal das instruções. A partir da codificação esperada foi feita uma comparação com a codificação obtida;
- Testar o reconhecimento das instruções pelo processador: depois de compilada a ferramenta, é gerado um arquivo *.x* que é o simulador do processador. O simulador recebe como entrada arquivos com extensão *.o* ou *.hex* para a execução e reconhecimento. Aproveitando os arquivos gerados no item anterior, foram usados para o processador, onde a cada arquivo passado, o processador mostrava instruções que conseguiu reconhecer ou erros na decodificação;
- Testar os comportamentos das instruções: aqui é a fase onde a lógica das instruções é testada. Essa etapa não foi realizada por problemas com a implementação na codificação das instruções e no reconhecimento pelo processador. Além dos motivos citados na Seção 3.4.

Capítulo 4

Conclusões e Trabalhos Futuros

Com a linguagem ArchC é possível modelar qualquer tipo de processador, desde que sejam conhecidos sua arquitetura e seu conjunto de instruções. Para um projeto de um novo processador, a modelagem contribui como uma forma de aplicar os requisitos desse processador em nível de software, antes mesmo de criar o seu hardware. Essa característica é importante visto que podem ocorrer erros na implementação do hardware de um processador, e esses erros geram custos na correção do hardware. Com a modelagem em software vários testes podem ser feitos para verificar o comportamento do processador, e correções podem ser feitas sem custos de componentes.

O processador Nios foi construído para ter suporte a programas desenvolvidos em C ou C++, e isso explica as características de sua arquitetura, com o recurso da janela deslizante para chamadas de procedimentos, e instruções que manipulam bits e operações aritméticas.

Para trabalhos futuros estão a implementação de todos os comportamentos das instruções; a correção de alguns detalhes não implementados na arquitetura; os testes dos comportamentos implementados; os testes utilizando os *benchmarks* sugeridos no *road map* do site da linguagem ArchC.

Referências Bibliográficas

- [AC008] 2008. ArchC Site. Página acessada em 20/05/2008. <http://archc.sourceforge.net/>.
- [AN008] 2008. Literature: Nios Processor. Página acessada em 20/05/2008. www.altera.com/literature/manual/mnl_nios_programmers32.pdf.
- [MP008] 2008. Processors - MIPS Technologies -MIPS Everywhere - MIPS Technologies. Página acessada em 20/05/2008. <http://www.mips.com/products/processors/>.