

Relatório - Modelagem do processador Nios 32bits utilizando Simulador Archc

Renato da Costa Rech¹, Thiago William Machado¹

¹Universidade Católica Dom Bosco - UCDB

Avenida Tamandaré - 6000 – Jd. Seminário – Campo Grande - MS - CEP: 79117-900

`rena.rech@gmail, thiagowilliamm@yahoo.com.br`

1. Introdução

Este relatório descreve a continuação do trabalho iniciado pelo acadêmico Renato Bezerra Herebia, que implementa o processador Altera/Nios de 32 bits, utilizando o simulador Archc. O Archc simula o processador de acordo com a implementação da arquitetura do mesmo. O Archc cria um executável para simulação de programas escritos em hexadecimal.

O acadêmico Renato Bezerra Herebia implementou toda a base do trabalho, os códigos fontes (Os arquivos: `nios32.ac`, `nios32isa.ac` e `nios32isa.cpp`) e algumas instruções. Porém não foram efetuados testes.

Este relatório é composto por 3 capítulos. No capítulo 1 é apresentado uma visão geral sobre a arquitetura do Altera/Nios, no capítulo 2 há 3 partes que detalham o desenvolvimento do trabalho e no capítulo 3 apresentamos a conclusão e possíveis melhoras a serem feitas.

2. Sobre o processador Altera/Nios de 32 bits

O Processador Nios foi desenvolvido pela fabricante Altera e possui um campo de endereçamento de 32 bits. Nosso trabalho foi implementado com pipeline de 5 estágios (ID, IF, EX, MEM, WB) que assemelha-se com a arquitetura do MIPS. O tamanho das instruções é de 16 bits, com uma word de tamanho de 32 bits. É baseado na arquitetura RISC.

Os registradores de propósito geral se encontram no B.R. ¹, que pode ser definido com um tamanho de 128, 256 ou 512 registradores. Apesar do número de registradores, são somente 32 registradores acessíveis. No BR há também o registrador K de 11 bits, que é setado em 0 por todas as instruções, exceto pelas instruções precedidas por PFX e PFXIO, essas instruções desabilitam interrupções para um ciclo, o conteúdo do registrador K é utilizado para a concatenação de bits para serem usados na próxima instruções. O registrador PC ² contém o endereço da instrução que está sendo executada e é sempre incrementado por 2 após a execução de cada instruções, exceto pelas instruções BR, BSR, CALL, JMP, LRET, RET e TRET que modificam o seu valor diretamente. Existem também no processador Nios um conjunto de registradores chamado de Registradores de Controle, que são: STATUS - `ctl0`, ISTATUS - `ctl1`, WVALID - `ctl2`, ICACHE - `ctl5`, CPUID - `ctl6`, DCACHE - `ctl7`, CLRIE - `ctl8` e SETIE - `ctl9`, onde seus bits representam informações que auxiliam no processo de execução no processador.

¹Banco de Registradores

²Program Counter

O acesso a memória feita pelo processador Nios é muito parecida com a do processador MIPS, tendo uma instrução principal de Load (LD) e uma de Store (ST). A memória Cache do Nios trabalha de forma parecida com a Cache do processador MIPS. Cada linha da cache do Nios possui os campos Tag (endereço da dado na cache), Valid (se contém ou não o dado), Cache Date (dados do referente endereço). Quando se executa uma instrução de Load(LD, LDP, LDS) o Nios compara os Bits mais significativos do endereço da instrução com o campo Tag na Cache, se o for igual e se houver esse dado na cache acontece um "hit", caso contrário ocorre um "miss" e ele busca na memória.

3. Objetivos Iniciais

Como toda a base do processador foi implementado, sua arquitetura, os registradores de pipeline, os registradores de controle, o nosso objetivo era implementar as principais instruções que não foram terminadas e observar o comportamento das mesmas. Principalmente aquelas de memória, para observar o comportamento do processador.

As instruções não terminadas no trabalho que nos foi passado são todas precedidas pelas instruções PFX ou PFXIO, como AND, OR, XOR, SUB e ADD. Outras instruções importantes a serem terminadas são as que utilizam memórias como LD e ST que também são precedidas por PFX ou PFXIO.

3.1. Objetivos Alcançados

Conseguimos desenvolver algumas dessas instruções consideradas importantes para o processador e que não foram implementadas, como as instruções de memória LD e ST e as instruções precedidas por PFXIO ou PFX como, XOR, OR, ADD e SUB. Porém, nas instruções de acesso a memória pouca coisa conseguimos avançar, implementando apenas as instruções LD e ST, mencionadas acima.

A parte de testes foram feitas testando uma instrução de cada vez, depois desenvolvemos um exemplo em hexadecimal de um programa mais completo.

3.2. Dificuldades Encontradas e recursos não implementados

As principais dificuldades encontradas no desenvolvimento foram as instruções que utilizavam memória, principalmente quando tinha que se alinhar o endereço de memória com algum operando, como nas instruções LD e ST, as instruções que manipulavam registradores de controle também trouxeram uma certa dificuldade, outro ponto onde encontramos certas complicações foram as instruções que implementavam acesso à cache ou algo relacionado como RDCTL e WDCTL, onde dessa última não conseguimos avançar em praticamente nada no trabalho. Perdemos um pouco de tempo com as instruções que usavam os bits de flag, principalmente o bit C, que representa o carry de uma operação aritmética.

Muitas das instruções não implementadas, foram devido a falta de detalhamento no datasheet. Principalmente as instruções mais complexas que envolviam memórias e memória cache, como STD16, ST16S, ST8D e ST8S, STS, STP, LDP, LDS dentre outras.

Os testes não foram feitos de forma mais detalhada por termos dificuldades com o funcionamento do montador.

4. Conclusões e Trabalhos Futuros

Visto que, a linguagem Archc proporciona a implementação de qualquer processador, basta ter o seu conjunto de instruções e sua arquitetura. Essa implementação do Nios 32 já se pode testar o funcionamento de alguns algoritmos e o desempenho do processador. Como trabalhos futuros devem ser implementadas instruções relacionadas a memória principal e memória cache, assim como a utilização do montador que transcreve códigos da linguagem assembler para a codificação hexadecimal, fazendo com que os testes das instruções sejam mais fáceis e eficientes.