Universidade Federal de São Carlos

Departamento de Computação

Disciplina:

Laboratório de Arquitetura e Organização de Computadores 1

Relatório Final

Projeto de arquitetura de microprocessador de 16 bits

Projeto de sistema de entrada e saída de dados usando a arquitetura Nios II.

Daniel Chaves Macedo RA: 280844

Pedro Gabriel Artiga RA: 351180

Felipe Fiori Campos Martins RA: 316660

São Carlos, 2011.

Introdução

Visando estudar de forma eficiente as arquiteturas existentes e como construi-las utilizamos a ferramenta Quartus II da Altera. No primeiro momento, através da linguagem de descrição de hardware Verilog nós desenvolvemos uma arquitetura funcional para um processador de 16 bits.

Usando a ferramentas SOPC Builder, presente no software Altera Quartus II, geramos uma arquitetura softcore Nios II, capaz de receber e retornar dados, além disso é possível programar as instruções que se deseja executar em linguagem C ou Assembly.

Objetivo

Este relatório apresentará os resultados e conclusões obtidos durante o desenvolvimento destes estudos.

CPU 16 bits

O processador contém duas memórias distintas: para dados e para instruções. Como o nome sugere a memória de dados armazena qualquer dado que o programa necessite, assim como os termos empilhados na estrutura de pilha criada. Já a memória de instruções possui somente as instruções a serem executadas. As mesmas têm comprimento fixo de 16 bits. Possui também quatro registradores de uso geral.

Para entrada e saída de dados existem portas que usadas as instruções IN ou OUT salvam ou recuperam dados de um determinado registrador.

Estrutura de pilha

A pilha é armazenada na memória de dados. Um registrador especial é usado para guardar o endereço do topo da pilha. No momento em que o componente é 'resetado' o ponteiro aponta para o último endereço da memória, como um endereço é composto por 8 bits, este é FFh. Dessa forma quando o primeiro PUSH é dado essa posição tem seu valor alterado e o ponteiro de pilha decrementado. Na instrução de POP o ponteiro é incrementado.

Ciclo busca, decodifica, executa

O processador possui uma máquina de estados que determina o que deve ser executado em cada ciclo de clock. Após o reset, a máquina está no estado de busca, onde recupera a instrução atual na memória de instruções, o próximo estado é a decodificação dessa instrução e em seguida o estado ou estados específicos da instrução, que deve ao seu fim recolocar a máquina no ciclo de execução voltando seu estado para busca.

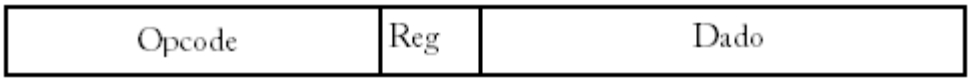
A cada mudança de estado, uma configuração diferente deve estar disposta no circuito, como por exemplo, o barramento de endereçamento da memória ou a habilitação ou não da escrita, endereço da próxima instrução a ser buscada, entre outros. Portanto foram usados dois blocos 'always' no código Verilog, assim um é executado quando o estado muda, alterando os dados nos barramentos, e outro que é executado quando o pulso de clock é efetivamente dado. Quando a máquina entra no estado de decodificação o campo da instrução referente ao endereço de memória já é colocado no barramento para endereçamento, no caso da próxima instrução necessitar de operações com dados da memória, o que é muito comum. Assim quando o pulso de clock é dado o estado passa a ser o de execução da instrução e o dado da memória no registrador de saída, economizando-se um ciclo.

No caso de instruções que lidam com escrita de memória, como por exemplo a instrução STORE, é necessário um estado adicional para garantir que o endereço no barramento e o valor no registrador de entrada continuem corretos até que a habilitação da escrita esteja desativada.

Instruções

Como já foi especificado, as instruções possuem comprimento fixo de 16 bits, a organização

da informação se dá da seguinte forma:



Campos:

Opcode[15..10]:	identifica a instrução (Ex.: add, sub, and, etc);
Reg[9..8]:	seleciona qual dos quatro registradores será usado.
Dado[7..0]:	pode ser tanto um imediato como um parâmetro adicional.

Conjunto de instruções

Operações na memória de dados:

STORE

Salva o conteúdo do registrador selecionado na memória de dados no endereço do campo Dado.

LOAD

Move para o registrador selecionado o dado da memória de dados no endereço de Dado.

Operações de salto:

JUMP

Salto incondicional. Passa a executar a instrução no endereço de Dado na memória de instruções.

JNEG

Assim como o JUMP, porém é condicional, só salta se o valor contido no registrador selecionado é negativo, caso contrário continua o fluxo normal de execução.

JPOS

Salta se o valor contido no registrador selecionado é positivo.

JZERO

Salta se o registrador selecionado possui valor zero.

Operações aritméticas:

ADD

Soma o registrador selecionado com o valor contido na posição de memória Dado.

SUBT

Assim como ADD porém executa uma subtração.

XOR

Executa um xor entre o valor do registrador selecionado e o valor de Dado.

OR

Executa um Or entre o valor do registrador selecionado e o valor de Dado.

AND

Executa um And entre o valor do registrador selecionado e o valor de Dado.

ADDI

Soma o registrador selecionado com o valor do campo Dado e guarda no próprio registrador.

SHL

Executa ‘n’ shift’s para esquerda no valor do registrador selecionado. O valor de n deve estar em Dado.

SHR

Assim como SHL porém para a direita.

Operações de pilha:

PUSH

Empilha o valor do registrador selecionado.

POP

Move para o registrador selecionado o valor do topo da pilha.

Operações de E/S:

IN

Move para o registrador selecionado o valor que está na porta de entrada do módulo.

OUT

Move para a porta de saída do módulo o valor contido no registrador selecionado.

Outras operações:

WAIT

Espera um determinado número de Clock's até buscar a próxima instrução. Considerando-se os ciclos gastos com "fetch" e "decode", a instrução incrementa um contador iniciado em zero e muda para um estado de espera, no próximo clock o valor é incrementado novamente e checado se é maior ou igual ao passado como parâmetro em Dado, se sim, busca a próxima instrução.

Operações para uso de procedimentos:

CALL

Empilha o endereço de retorno da execução e salta para o endereço de Dado.

RETURN

Volta a executar a partir do endereço anteriormente empilhado por CALL. Portanto, não pode haver um PUSH sem um POP correspondente antes do RETURN.

Arquitetura NIOS 2

Parte 1 - Configurando as ferramentas

1. Execute a ferramente SOPC Builder

a. Crie um novo projeto usando atráves do New Project Wizard;

b. Aba Projeto >> Tools >> SOPC Builder;

c. O nome deve ser nios_system;

d. Selecione a Linguagem Verilog.

2. Adicionar os módulos do nosso sistema

a. Adicione o processador Nios II, versão ecônomica;

   i. Library >> Processors >> Nios II Processor;

   ii. Selecione a versão Nios II/e;

b. Módulo de memória on_chip_memory;

   i. Memories and Memory Controllers >> On-Chip >> On-Chip Memory;

c. Barramento de entrada e saída de dados;

   i. Peripherals >> Microcontroller Peripherals >> PIO;

   ii. Adicione um como input e outro como output;

d. Módulo Jtag UART;

   i. Interface Protocols >> Serial >> JTAG UART;

   ii. Este módulo é responsável por codificar os dados de entrada de modo a ser corretamente interpretado pelo Nios II.

3. Associe os módulos uns aos outros

a. Nios II com o módulo de memoria on_chip_memory.

   i. cpu_0 >> Reset Vector e Exception Vector com “onchip_memory2_0”;

b. System >> Auto-Assign Base;

c. System >> Auto-Assign IRQS.

4. Alterar nomes.

a. Altere o nome dos barramentos de entrada para “sw”;

   i. pio_1 >> Rename;

b. Altere o nome dos barramentos de saída para “led”;

   i. pio_0 >> Rename.

5. Gere o projeto.

a. Clique em Generate.

Parte 2 - Criando o Diagrama Esquemático

1. Gere o Diagrama Esquemático utilizando o Quartus II

a. New >> Block Diagram/Schematic File;

b. Associe os pinos a placa Cyclone;

i. Symbol >> Project >> nios_system;

ii. Adicione Inputs e Outputs;

iii. Troque os nomes para sw[7..0] e LEDR[7..0]

iv. Troque os nomes pra CLOCK_50 e KEY[0];

v. Assignments >> Import Assignments >> DE1_Default.qsf;

vi. Salve e use “Set As Top_Level Entity”;

c. Compile.

2. Visualizando código gerado

a. Diretório >> cpu_0 (arquivo do tipo v).

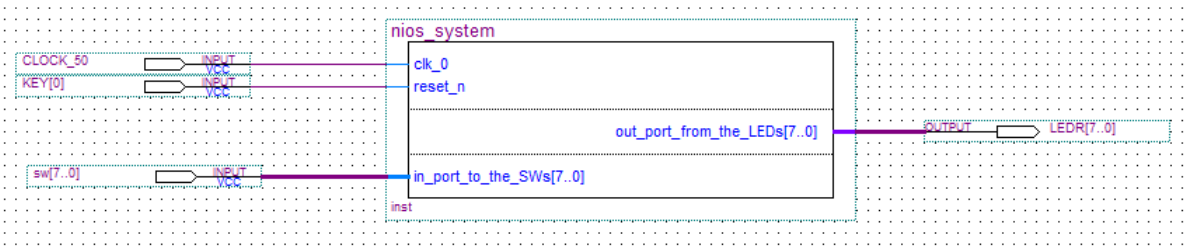


Figura 1.: Demonstração da pinagem implementada para a utilização do processador NIOS 2

Parte 3 - Criando e compilando o código a ser executado.

1. **Abra algum editor de texto ou um programa propício para programação**

a. Digite o seguinte código:

```
.include "nios_macros.s"
.equ sws, 0x00003000
.equ leds, 0x00003010
.global _start

_start: movia r2,sws
        movia r3,leds
loop:   ldbio r4,0(r2)
        stbio r4,0(r3)
        br loop
```

b. Salve o código escrito no diretório do projeto, com o nome de nios.s.

2. **Descarregue o projeto do Nios II na placa Cyclone**

   a. Tools >> Programer >> Hardware Setup >> USB Blaster;

   b. Clique em Start.

3. **Abra o programa Altera Monitor Program**

   a. Gere um novo projeto;

      i. Selecione o arquvio nios.s;

      ii. Custom System e Assembly Program (opções a serem selecionadas).

4. **Compile**

   a. Clique em Compile & Load;

   b. Botão Continue ativa o programa na placa Cyclone.

5. **Agora repita o processo para o código em C**

   a. Digite o seguinte código:

```
#define sws (volatile char*) 0x00003000
#define LEDs (char*) 0x00003010

void main()
{
        while(1)
                *LEDs = *sws;

}
```

   b. Salve o código escrito no diretório do projeto, com o nome de nios.c.

6. **Repita agora os passos 3 e 4.**

   a. **Nota:** Marcar a box “Use small C library”.

Resultados -

Ambos os testes forem um sucesso, a placa Cyclone II DE1 executou os códigos corretamente.

Parte 4 - Criando e Testando o código de uma fila.

1. **Abra o programa desejado para programação**
    a. Digite o seguinte código em Assembly;
    b. Repita os passos anteriores para testar o código;
    c. Digite o seguinte código em C;
2. **Resultado Obtidos**

As imagens abaixo demonstram os estados de execução do programa que gerência uma fila, implementado em assembly e executado utilizando a placa FPGA Altera Cyclone II disponível em sala. Na figura 2 podemos ver o estado inicial dos registradores do lado direito e um trecho do código exportado para a placa que está aguardando o comando para prosseguir com a execução.

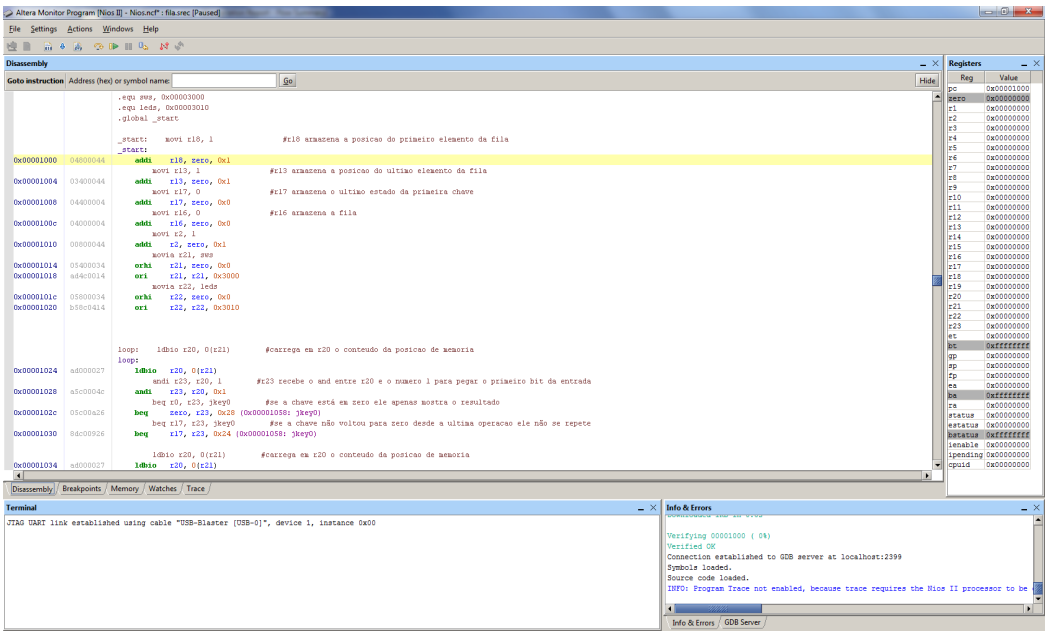


Figura 2.: Estado inicial da tela de depuração remota da plataforma Altera.

Assim que a execução retomou seu curso normal nós usamos a primeira chave mecânica disponível na placa para indicar que uma operação deveria ser realizada. A segunda chave foi utilizada para indicar qual operação deveria ser realizada na fila. Quando a segunda chave está em estado alto a fila faz uma inserção, e quando a chave está em estado baixo ela faz uma remoção.

Na figura 3 podemos observar o estado do processador após quatro inserções na fila.

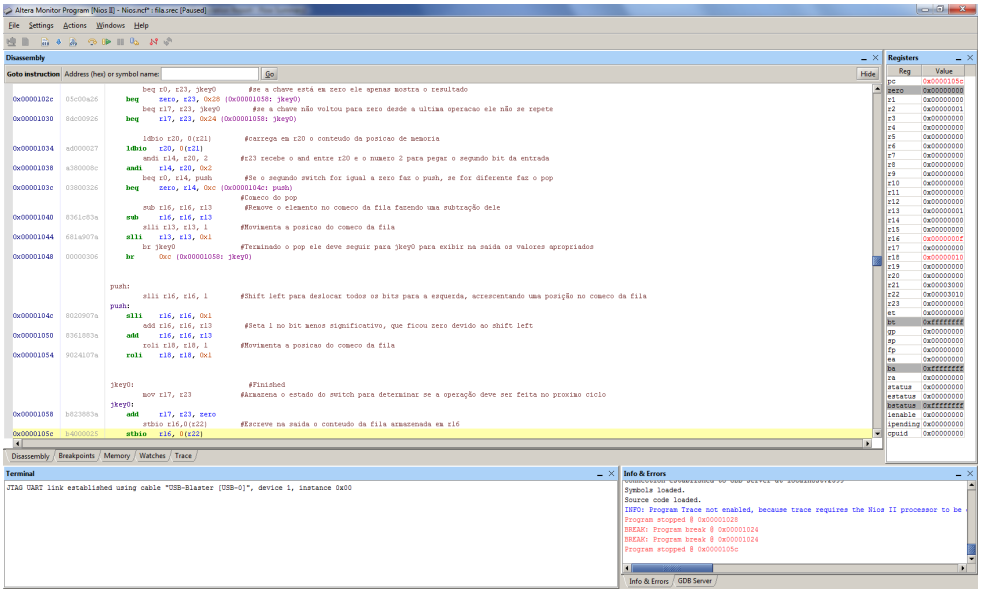


Figura 3.: Estado da plataforma de depuração após 4 ‘pushs’ da fila implementada

Após a realização das inserções a fila pode ter alguns elementos removidos. A figura 4 demonstra o estado do sistema após algumas remoções.

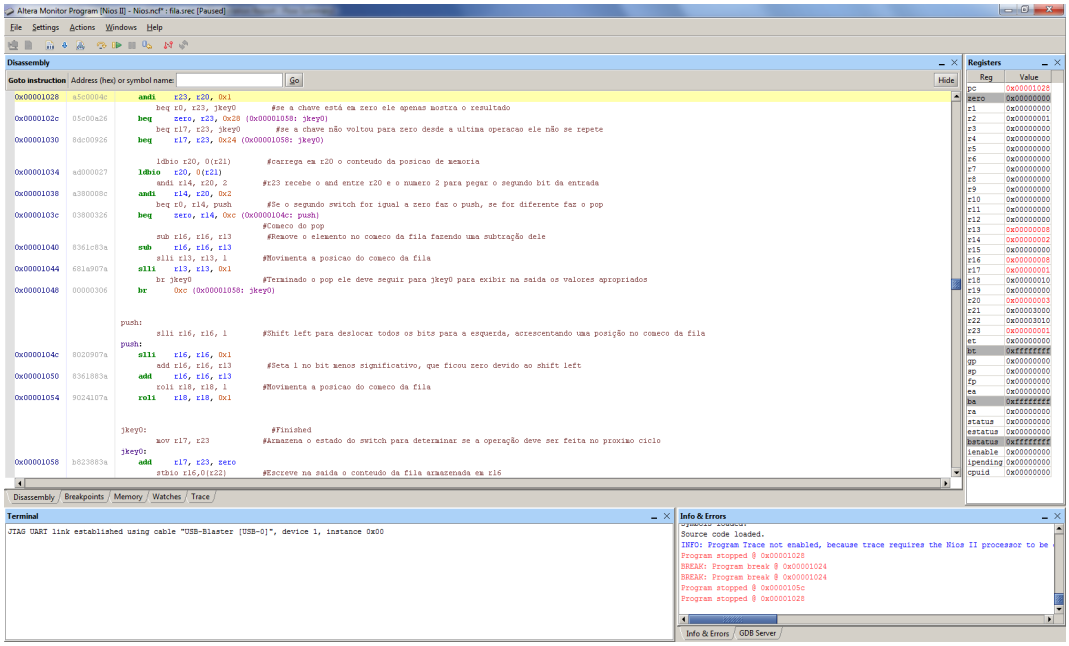


Figura 4.: Estado da plataforma de depuração após 3 ‘pops’ da fila implementada

Conclusão

As diversas ferramentas disponíveis hoje em dia tornam praticamente trivial o desenvolvimento de hardware, desde que o conhecimento necessário esteja acessível. Vimos aqui que seguindo alguns passos é possível acessar uma arquitetura customizável e portável a qualquer momento, sem falar o desenvolvimento relativamente simples de arquiteturas dedicadas a tarefas específicas, o que as tornam mais eficientes e baratas.