Relatório Projeto Nibble **07/11/2016**

Alunos: Bruno Gabriel C. Lima e César Augusto dos S. Caetano

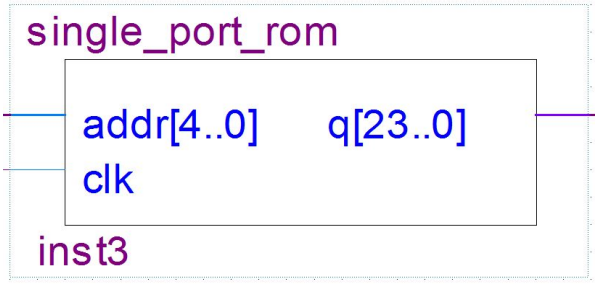
Descrição do projeto

Processador de conjunto reduzido de instruções de 24 bits; com 5 bits de endereçamento para instrução e dados; trabalhando com variáveis (palavras) de 8 bits.

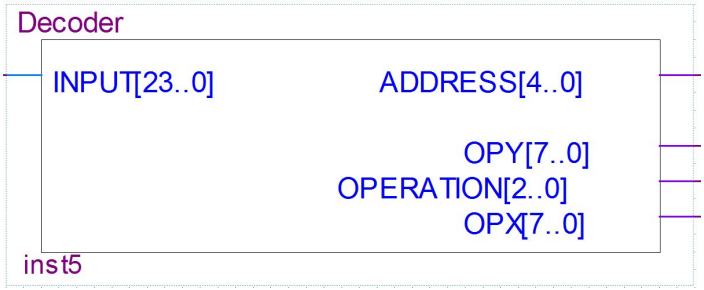
Implementado em arquitetura Harvard, com 4 estágio de pipeline, compreende 4 funções: ADD, ADDP, SUB e SUBP. Seu funcionamento divide-se em 4 fases distintas e independentes: FETCH, DECODE, EXECUTE, STORE.

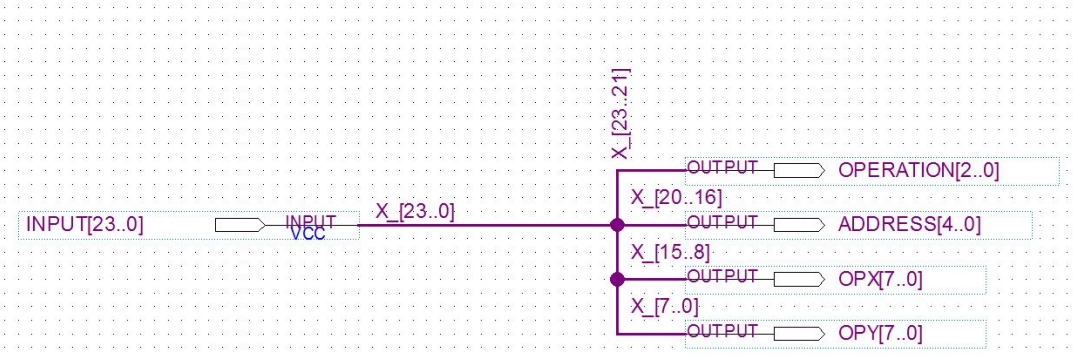
Implementação dos Módulos

1. Memória ROM(programa): Utilizamos uma ROM simples, somente leitura, cujos dados estavam armazenados num documento de texto externo (single_port_rom_init.txt). Implementação já existente em verilog no Quartus como single_port_rom.

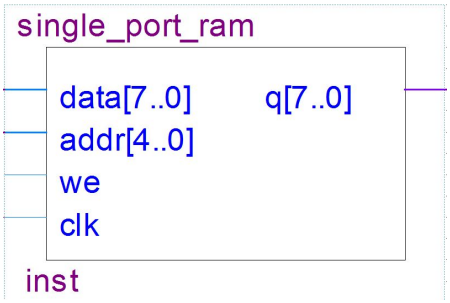


2. Decodificador de instrução: Possui como entrada a instrução completa (24 bits). Realiza a extração das partes da instrução, tendo como saída, respectivamente: o identificador da operação (3 bits); o endereço de dados que será utilizado na RAM (5 bits); e os operadores X e Y (8 bits cada). Essa separação foi feita utilizando buses. Segue o diagrama em bloco e o esquemático em seu interior:

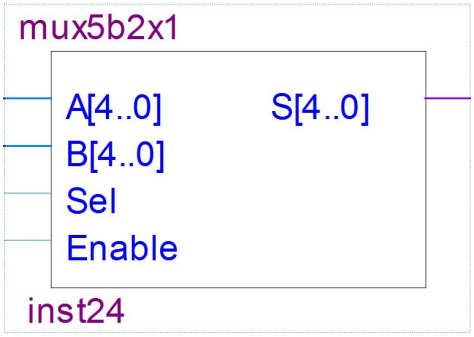




3. Memória RAM(dados): Também disponível pronta em Verilog no Quartus. Possui a finalidade de leitura/escrita de dados na memória, e opera na borda de descida do clock. A variável de controle W/E é a responsável por setar a memória para escrita ou leitura. No caso de escrita, o dado a ser escrito deve estar presente da entrada DATA.

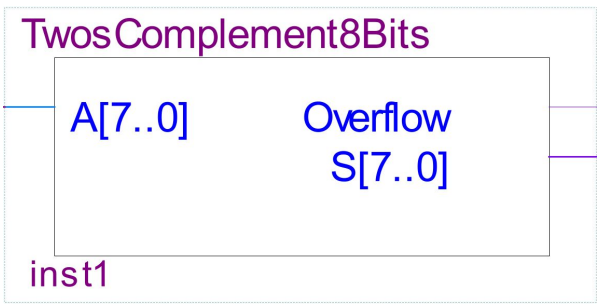


4. Multiplexadores: Responsáveis por configurar o processador adequadamente para cada tipo de operação a ser executada. Optamos por implementá-los em Verilog, para conhecer mais sobre a linguagem e para praticar melhor o aprendizado. Segue o diagrama em bloco de um MUX 2x1 de dados de 5 bits, como exemplo, e seu código em Verilog:

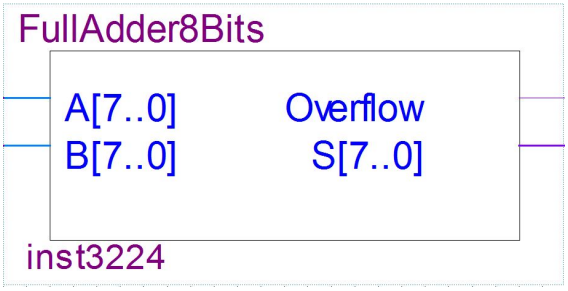



```
// Mux 2x1 de 5 bits. Código em Verilog
module mux5b2x1( input [4:0] A, B, input Sel, input Enable, output reg [4:0] S );
always begin
        case (Sel)
                1'b0: S = (Enable) ? A : 5'bz;
                1'b1: S = (Enable) ? B : 5'bz;
        Endcase
 end
endmodule
```

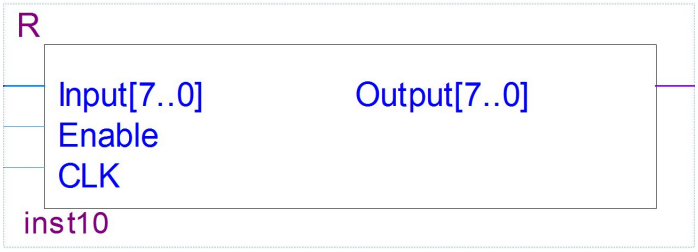
5. Módulo Complemento de 2: O mesmo módulo implementado na atividade passada de Microeletrônica, utilizando Flip-Flops. Possui como o números de 8 bits a ser complementado, e como saída o resultado (também 8 bits) e um bit indicador de overflow.

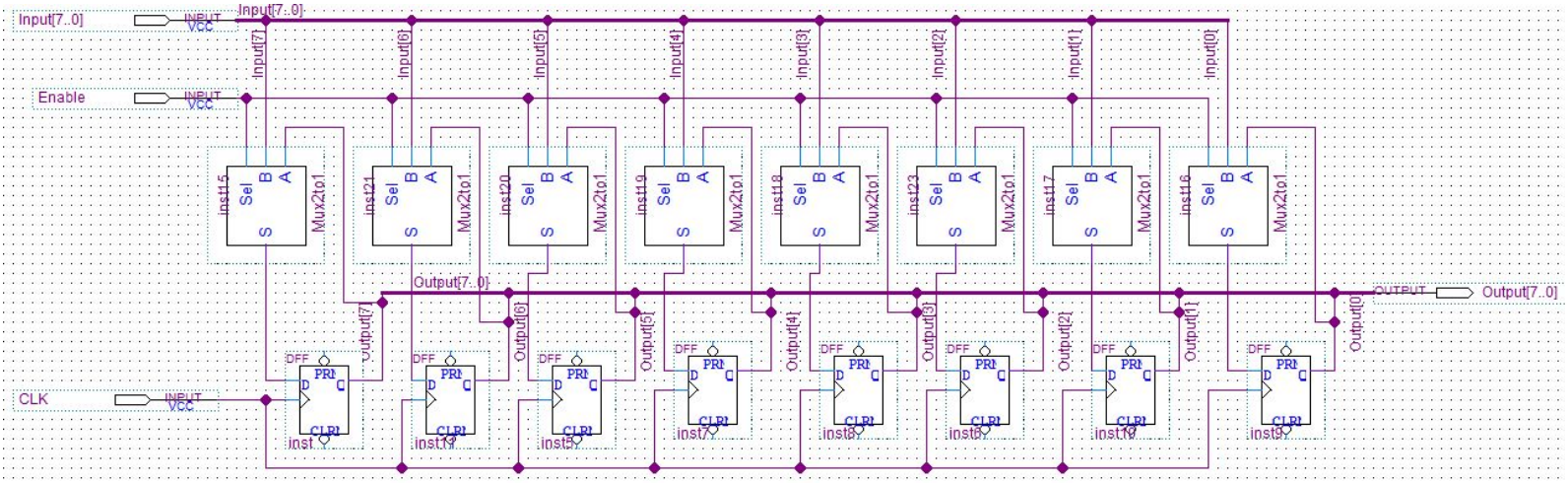


6. Módulo Somador Completo: Também reaproveitado da atividade de Microeletrônica. Diagrama em bloco mostrado a seguir:



7. Registradores de Carga Paralela: Implementados utilizando Mux2x1 e Flip-Flops, como mostram o bloco e o esquemático a seguir:





8. Módulo de Controle Central: É o responsável pelo comando do processador como um todo, habilitando os módulos responsáveis pela execução da instrução atual. Também garante que não haja interferência entre os estágios do Pipeline.

Esquemático Geral

Simulação e Resultados

Conclusão

Com tal atividade pudemos adquirir conhecimentos práticos sobre o processo de implementação de um processador. Mesmo um processador simples, com apenas 4 instruções, foi o suficiente para que pudéssemos projetar, modularizar e sincronizar seu funcionamento. Passamos por diversas fases na implementação, indo desde o projeto de um simples registrador até o projeto de uma central de controle completa.

Notamos que ao decorrer do projeto tivemos uma maior inclinação ao uso do Verilog. Inicialmente, a linguagem de blocos parecia suprir nossas necessidades. Entretanto, á medida em que o projeto cresca, foram surgindo necessidades que eram supridas mais facilmente pela linguaguem de descrição de hardware.

Por fim, adquirimos maior familiaridade com a ferramenta Quartus, incluindo suas técnicas de simulação. Pode-se dizer que as expectativas quanto ao projeto foram atendidas.