|  |  |
| --- | --- |
| Resultado de imagem para logo dcc ufmg | Introdução à Arquitetura de Computadores |
| Trabalho Prático 2 – Implementação de um processador de calculadora |
| Autor: Jhonatan Vinícius Mota Corrêa  Pedro Tavares de Carvalho  Pedro Henrique Fialho de Alcantara  Filipe Rodrigues Batista de Oliveira |
| Data: 23/06/2019 |

**Descrição**

Este trabalho consiste na implementação de um processador em Verilog que emula uma calculadora (o CALC1). Nessa calculadora, teremos um conjunto bem restrito de registradores, organizada em um banco de registradores, bem como uma ULA (unidade lógica e aritmética) e uma unidade de memória. O CALC1 é um processador monociclo.

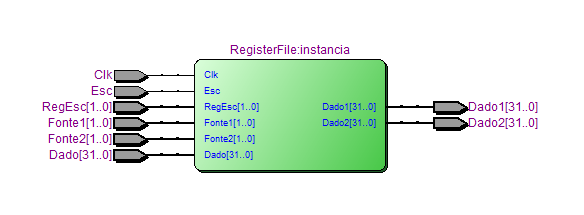
**Desenvolvimento**

As unidades funcionais a serem criadas são um banco de registradores, uma memória, uma ULA simples e uma unidade de controle. Todas as unidades, bem como o processador, serão especificadas usando little endian.

**Banco de Registradores:**

O banco de registradores deve ser composto por 3 registradores de 32 bits cada, que são os seguintes:

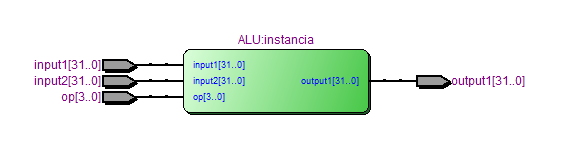
* Registrador fonte A (registrador número 0)
* Registrador fonte B (registrador número 1)
* Acumulador (registrador número 2)

  
Figura 1: Esquema do banco de registradores.

O banco vai ter leitura simultânea de dois registradores (portanto duas saídas), cada uma de 32 bits (tamanho dos registradores). Vai ter também entrada de clock, entrada de ativação da escrita de registradores, fonte de escrita, fontes de leituras dos registradores e entrada de dados para a escrita nos mesmos. Como há leitura e gravação de dados no mesmo período de clock, a leitura ocorre na borda de subida do clock e a gravação na borda de subida.

**ULA:**

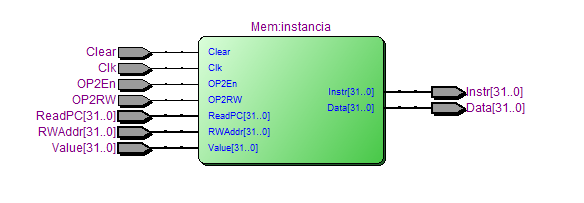
A unidade lógica e aritmética deve trabalhar com valores inteiros de 32 bits, representados em complemento de dois, e realizar as operações de adição, subtração, multiplicação e divisão. Vocês podem utilizar os operadores de adição e subtração do verilog.

  
Figura 2: Esquema da ULA.

Como a ULA é um componente assíncrono, ele basicamente tem suas entradas e saída de dados e a entrada de controle que define a operação a ser realizada.

**Memória**:

A memória do CALC1 é usada para guardamos resultados intermediários e instruções. Assim, ela é organizada em palavras de 32 bits, e a memória é indexada palavra a palavra. Assim, o endereço 0 lê a primeira palavra, o endereço 1 lê a segunda, etc. Isso é diferente do MIPS, onde a memória é indexada por byte, e assim a segunda palavra da memória seria acessada a partir do byte da memória. O CALC1 possui 1024 palavras na memória.

  
Figura 3: Esquema da memória

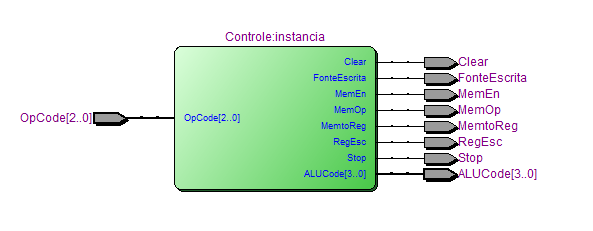
A memória vai contar além do Clock, com os sinais OP2En (indica que irá haver uma operação de memória) e OP2RW (indica qual o tipo de operação, se ‘1’ haverá escrita). Como haverá uma instrução de Memory Clear e o processador é uniciclo, foi optado por realizar esse processo de limpeza da memória internamente, logo há uma entrada “Clear”.

Como só haverá essa memória no processador, e ela deverá armazenar dados e instruções e elas deverão ser carregadas no mesmo ciclo, essa memória conta com duas saídas, uma para cada. Contam também com entradas de endereço distintas. Por fim, há uma entrada de dados para gravação na memória.

Vale ressaltar também que assim como no registrador, a gravação de um dado se dá na borda de descida do clock e a leitura na borda de subida. Para o correto funcionamento das instruções, uma nova instrução é carregada apenas na borda de descida do clock também.

**Unidade de Controle.**

A unidade de controle controla o fluxo de dados no caminho destes, determinando quais e quando os componentes serão ativados. Ela basicamente determina sinais de controle a partir do código de instrução (OPCODE, primeiros bits da instrução).

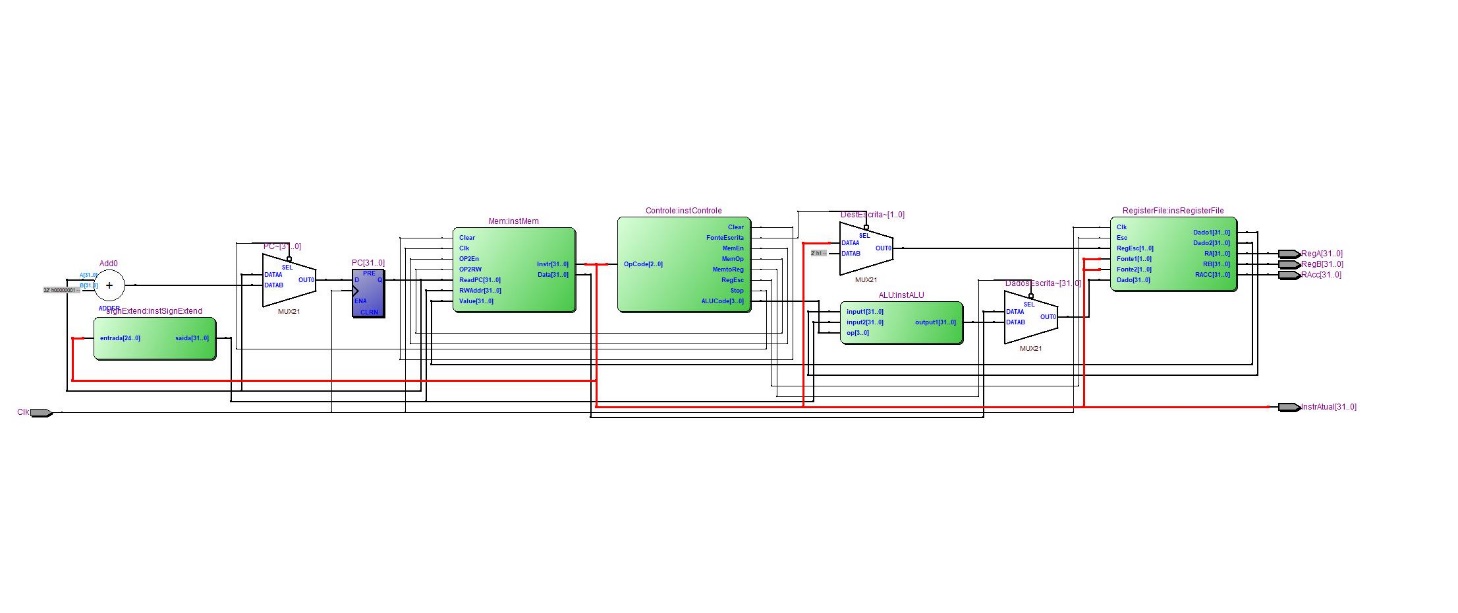
  
Figura 4: Controle do processador.

Os sinais de controle pensados foram:

* Clear: Limpa a Memória.
* FonteEscrita: Define qual o destino de escrita no Banco de Registradores (Se será o destino na instrução ou o acumulador).
* MemEn: Define se irá ativar ou não a memória.
* MemOp: Operação da memória (leitura ou escrita).
* MemtoReg: Define se o valor a ser escrito no banco de registradores virá da memória ou não.
* RegEsc: Define se a escrita no banco de registradores será ativada ou não.
* Stop: Para do processador (não carrega mais o PC).
* ALUCode: Código de operação da ULA.

**Caminho de Dados:**

O caminho de dados consiste da interligação dos componentes tais como necessários para funcionar:

  
Figura 5: Caminho de dados.

Além dos componentes já citados, foram necessários multiplexadores cara controlar o fluxo de dados (fonte de escrita dos registradores e atualizar o Program Counter), o registrador de programa (Program Counter), um somador para atualizar o PC e como as instruções do processador são baseadas em imediatos, foi necessário um sign extend para estender o sinal de 25 bits para 32 (replicando o ultimo bit para garantir o sinal do imediato).

Todos os componentes foram desenhados para obedecer à descrição disponibilizada do trabalho, tal como os endereços de registradores e organização de instruções, opcode, etc.

**Implementação/Simulação no Quartus/ModelSim**

A primeira implementação feita deste processador realizada foi feita no software Altera Quartus 11.1SP2 (Os diagramas acima foram gerados por ele). Com suporte à sintetização de circuitos lógicos, todos os códigos foram criados, compilados e sintetizados no Quartus, tomando por base o kit de desenvolvimento AlteraDE2 (Que contém um FPGA Altera Ciclone II EP2C35F672C6).

Como a síntese de memória demanda altos custos computacionais e lógicos, a memória foi desenhada para contar aqui apenas com 30 posições (as 20 primeiras com instruções e as 10 últimas para dados). Para teste das instruções, foi elaborado o pequeno programinha que testa todas as instruções:

MR X, A, 0H00000013 //Carrega no registrador A o valor da posição 20 de memória  
SUB A, X, 0H00000001 //Subtrai 1 do registrador A e coloca no ACC  
DIV ACC, X, 0H00000002 //divide o valor de ACC por 2  
MUL ACC, X, 0H00000005 //multiplica o valor de ACC por 5  
ADD ACC, X, 0H0000000A //adiciona 10 ao ACC  
MW ACC, X, 0H00000014 //Guarda o valor do ACC na posição 21 da memória  
MR X, B, 0H00000014 //Lê o valor da posição 21 em B (Copia de ACC para B)  
MC X, X, 0H00000014 //Limpa a memória na posição 21 da memória  
ADD ZERO, X, 0H00000000 //Zera o ACC  
MR X, A, 0H00000014 //Carrega a posição 21 em A (Zera A)  
HLT //Para o processador

Esse instruções codificadas em hexadecimal seguindo as orientações dadas na especificação fica (Os valores don’t care foram substituídos por 0):

C0000013

20000001

50000002

70000005

1000000A

F0000014

C2000014

80000014

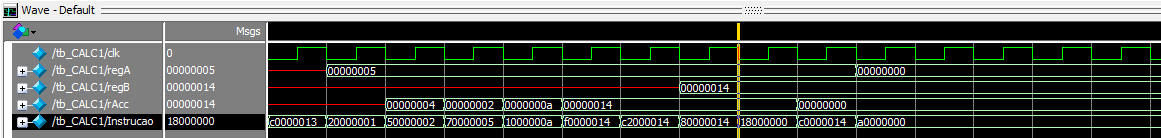
18000000

C0000014

A0000000

Essas instruções e dados foram colocados num arquivo de texto “memoria.txt” e carregadas via comando “$readmemh("memoria.txt", memory);” no arquivo verilog da memória.

Foram criadas saídas auxiliares ao circuito, para analisar os valores armazenados nos registradores e a instrução corrente. Foi armazenado o valor 5 na posição 20 da memória. O resultado final é visto na simulação realizada via o software ModelSim:

  
Figura 6: Resultado da simulação no ModelSim.

Podemos ver pela simulação que os valores foram os esperados, atestando o funcionamento das instruções e a comunicação entre memória/banco de registradores/ULA/controle.

**Implementação no iverilog e gtkwave**

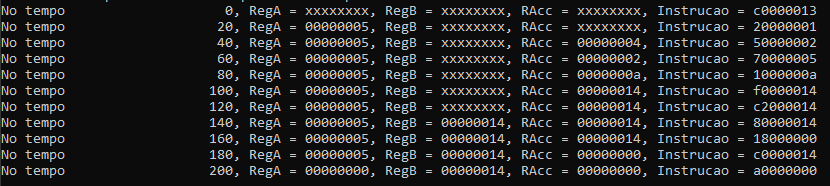
Como era necessário, o projeto foi compilado também no iverilog a fim de analisar os resultados também nesse software. O projeto foi todo compilado em Windows. Foi gerado uma nova compilação, agora com as 1023 posições de memória.

Foi gerado um testbench alternativo para o iverilog, mas com o mesmo programa (salvo o fato de que agora os dados começam a ser guardados na posição 100 da memória e o arquivo base é o mem.txt).

Para a simulação os seguintes comandos devem ser executados, no diretório dos arquivos:

iverilog -o simulação tb\_CALC1.v CALC1.v signExtend.v RegisterFile.v Controle.v ALU.v Mem.v  
vvp simulacao

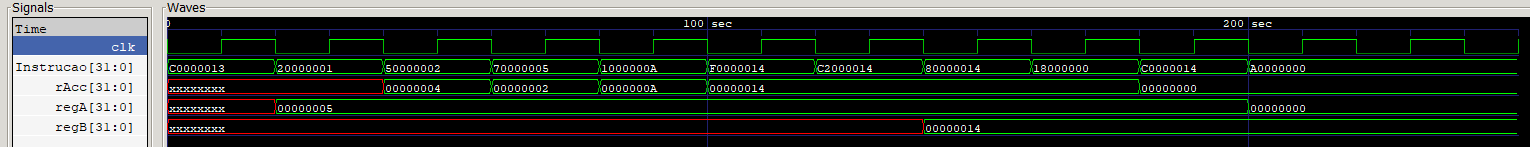
O resultado recebido é o seguinte:

  
Figura 7: Resultado da simulação com o iverilog.

Com o mesmo testbench foi gerada uma saída para o gtkwave. Para rodar a saída no gtk wave basta executar o comando:

gtkwave saida.vcd

E o resultado é o seguinte:

  
Figura 8: Resultado da simulação com o GTKWave.

Conclusão

Apesar de ser um circuito relativamente simples, esse pequeno processador foi um grande exercício, principalmente no que toca ao exercício das lógicas de um circuito digital. Com poucas ligações e unidades funcionais, foi possível criar um processador que faz todas as atividades básicas de uma calculadora. Um dos maiores (senão o maior) desafios do projeto foi conciliar todos os tempos dos componentes, a fim de garantir que os dados estariam nos componentes no tempo correto, evitando assim gravação de lixos na memória. Mas apesar de todos os problemas, o projeto foi um sucesso e funcionou como o esperado com todos os testes.

Link github do trabalho: