Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação

DCC819 - Arquitetura de Computadores

Relatório I - Unidade Lógica Aritmética

Guilherme Batista Santos
Iuri Silva Castro
João Mateus de Freitas Veneroso
Ricardo Pagoto Marinho

Belo Horizonte - MG
16 de outubro de 2017

1 Introdução

A Unidade Lógica Aritmética, ou *ULA*, é um circuito digital responsável por realizar operações lógicas e aritméticas no microprocessador, como por exemplo, executando operações de adição, subtração, deslocamentos lógicos, operações lógicas *and* e *or*. A Figura 1 abaixo mostra a forma geral de uma ULA.

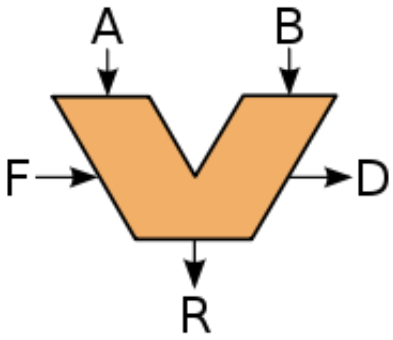

Figura 1: Unidade Lógica Aritmética.

A unidade possui duas entradas e uma saída de dados, A, B e R respectivamente, além de entradas e saídas de sinais de controle, F e D. As entradas A e B são conhecidas como *operandos*, valores em que a unidade executará a operação designada. O resultado da operação será dado em R. Os sinais entrada de controle F indicam a unidade, por exemplo, qual operação deverá ser feita, e os sinais de saída D são utilizados, por exemplo, para indicar o estado do resultado da operação como indicação de valor zero, negativo ou se houve *overflow* na operação.

Propõe-se, para esse trabalho, implementar uma Unidade Lógica Aritmética capaz de realizar operações específicas. Será utilizada a linguagem de descrição de hardware *Verilog HDL* e a plataforma *Quartus II* para a implementação. O simulador *ModelSim* será utilizado para auxiliar na verificação e testes da implementação. Testes físicos serão feitos no módulo de prototipação Altera DE2.

Futuramente, a ULA descrita neste trabalho será integrada como parte de um microprocessador.

2 Descrição

Deseja-se que o microprocessador seja capaz de executar as instruções conforme a tabela 1. As instruções são de 16-bits e trabalham com registradores de 16-bits e imediatos absolutos de 4-bits. Cada instrução possui um código de operação de 4-bits, código do registrador de destino $s4 de 4-bits, código do registrador $s3 ou o imediato absoluto de 4-bits e o código do registrador operando $s2 de 4-bits.

Observa-se que há instruções que trabalham com operandos registro-registro e operandos registro-imediato. Tais operações não são distinguidas pela ULA, indicando apenas se um dos operandos virá do banco de registradores ou da própria instrução. A unidade precisa ser capaz de apenas executar as operações de adição, subtração, comparação, E, OU e OU-EXCLUSIVO, deixando a parte de decisão de operandos para a unidade de decodificação.

Observa-se também que para a instrução de subtração, em que a posição dos operandos altera o resultado, a subtração entre registros e a subtração entre registro-imediato, tem a forma $s4 = $s3 - $s2 e $s4 = $s2 - imm. Para tornar a ULA genérica e diminuir a complexidade, optou-se pela modificação da instrução de subtração entre registros, ficando da forma $s4 = $s2 - $s3.

Código	Instrução	Operação	Descrição
0	ADD $s4,$s3,$s2	$s4 = $s2 + $s3	Adição entre registros
1	SUB $s4,$s3,$s2	$s4 = $s3 - $s2	Subtração entre registros
2	SLTI $s4,imm,$s2	$s2 > imm ? $s4 = 1 : $s4 = 0	Comparação entre registro e imediato
3	AND $s4,$s3,$s2	$s4 = $s2 and $s3	*AND* lógico com dois registros
4	OR $s4,$s3,$s2	$s4 = $s2 or $s3	*OR* lógico com dois registros
5	XOR $s4,$s3,$s2	$s4 = $s2 xor $s3	*XOR* lógico com dois registros
6	ANDI $s4,imm,$s2	$s4 = $s2 and imm	*AND* lógico com um registro e um imediato
7	ORI $s4,imm,$s2	$s4 = $s2 or imm	*OR* lógico com um registro e um imediato
8	XORI $s4,imm,$s2	$s4 = $s2 xor imm	*XOR* lógico com um registro e um imediato
9	ADDI $s4,imm,$s2	$s4 = $s2 + imm	Adição entre registro e um imediato
10	SUBI $s4,imm,$s2	$s4 = $s2 - imm	Subtração entre registro e um imediato

Tabela 1: Descrição das instruções requisitadas.

Sinais de indicação de características do resultado da operação são desejáveis, assim opta-se por um registro de indicadores de *zero*, *negativo* e *overflow*, conforme a Tabela 2 abaixo, sendo esse registro atualizado a cada operação da unidade.

Índice	Sinal
0	*Overflow*
1	Negativo
2	Zero

Tabela 2. Sinais de saída da ULA.

Definido as instruções, parte-se para a implementação.

3 Implementação

Necessita-se, além da ULA, de implementar unidades auxiliares, como o banco de registradores e a unidade de decodificação, que dão suporte ao funcionamento da ULA. Implementa-se também uma unidade de controle para comandar a execução das instruções propostas. As seções seguintes descrevem o funcionamento e a implementação das unidades.

3.1 Banco de registradores

O banco de registradores é utilizado para fazer o armazenamento temporário de valores que serão utilizados e escritos pela ULA. Implementou-se um banco de registradores com em 16 registradores de 16-bits cada. A figura 2 mostra os sinais de entrada e saída do módulo.

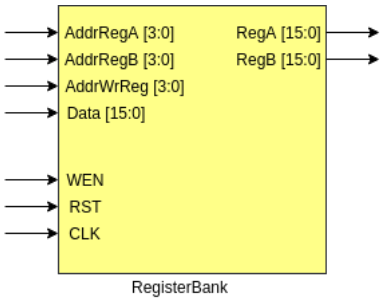

Figura 2. Bloco do Banco de Registradores.

O módulo possui três entradas de 4-bits para endereços de registros, sendo dois endereços para leitura dos registros A e B (*AddrRegA* e *AddrRegB*), com os dados de leitura apresentados em *RegA* e *RegB*, e um endereço (*AddrWriteReg*) para escrita do valor de entrada (*Data*). Há três sinais de controle, sendo:

- WEN (*Write Enable*): Indica a operação do banco (leitura ou escrita). Ele será escrito, caso WEN=1 e lido caso WEN=0;
- RST (*Reset*): Faz o *reset* do banco, zerando os valores de todos os registros;
- CLK (*Clock*): Utilizado para sincronização e execução.

Para a utilização correta do módulo primeiro deve-se definir qual será a operação a ser realizada (leitura ou escrita) e o devido sinal deve ser colocado em WEN. Os endereços dos registros devem ser definidos e, em seguida, deve-se fazer uma transição do sinal de relógio (CLK) para que a operação seja executada.

O sinal de *reset* (RST) deve ser mantido em nível lógico baixo durante toda a execução. Ele é utilizado apenas para a inicialização do banco de registros.

3.2 ULA

O módulo desenvolvido possui 5 entradas e duas saídas: *OpA*, *OpB*, *Op*, *RST*, *CLK*, *Res* e *FlagReg* respectivamente. A figura 3 mostra os sinais de entrada e saída do módulo.

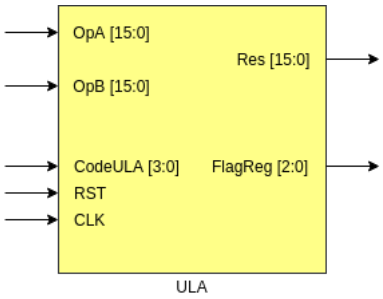

Figura 3. Bloco da ULA.

As entradas *OpA* e *OpB* representam os operandos de 16-bits *A* e *B* a serem utilizados na operação. A entrada *CodeULA* de 4-btis indica qual operação a ser executada, como adição, subtração, etc., conforme a tabela 3 abaixo. As saídas *Res* de 16-bits e *FlagReg* de 3-bits indicam, respectivamente, o resultado da operação e o estado da operação.

Conforme descrito anteriormente, não há destinção, por parta da ULA, se o operando é um imediato ou um registro. Assim, cria-se a partir das instruções desejadas na tabela 1, uma ULA com as seguintes instruções genéricas:

Código	Operação
0000	Adição
0001	Subtração
0010	Comparação
0011	*AND* lógico
0100	*OR* lógico
0101	*XOR* lógico

Tabela 3. Descrição das operações implementadas.

Optou-se por implementar uma ALU síncrona, utilizando o passo do relógio (CLK) para orientar sua execução. Assim, depois de selecionados os operandos e a operação a ser executada, deve-se fazer uma transição no sinal de clock para que a operação seja realizada. O sinal de reset (RST) é utilizado apenas para limpar o registro de estados *FlagReg* e deve ser mantido em nível lógico baixo durante a execução.

Todas as operações da ALU modificam todo o registro de estados *FlagReg*.

3.3 Decodificador

O decodificador é responsável por receber a instrução completa, *i.e.*, todos os 16 bits contendo o código da operação e os endereços de registrador de entrada e saída e manipular a instrução para a ALU realizar a operação correta.

A implementação deste módulo possui uma entrada que indica a instrução que será executada(*Instr*) e uma de *clock* (*CLK*). A entrada *Instr* possui 16 bits. Este módulo possui como saída os operadores entregues pela instrução (dois de entrada - *OpA* e *OpB*, e um de saída - *OpC*), além da operação a ser realizada, *OpALU*, junto com duas *flags*: *isImm* e *isValid*. As duas *flags* indicam se a operação será realizada com um imediato e se ela é válida, ou seja, foi implementada pela ALU. A *flag isImm* que indica para o multiplexador qual entrada a ALU deverá receber, se do banco de registradores ou se é um imediato, logo, essa *flag* serve como entrada de seleção do multiplexador, descrito na próxima subseção.

A cada subida do *clock*, verificamos qual operação a instrução recebida realizará verificando os 4 bits mais significativos dela. Caso ela seja uma das instruções implementadas, a *flag isValid* recebe o valor 1, caso contrário, ela recebe o valor 0. Além disso, quando a instrução utilizar um imediato, seguindo os códigos de operação da Tabela 1, a *flag isImm* passa ter o valor 1. Se for uma instrução com dois registradores de entrada, ela tem o valor 0. Já a saída *OpALU* recebe o código da operação que a ALU deverá executar. As saídas *OpA*, *OpB* e *OpC* recebem os endereços de registradores dentro da instrução quando lemos os 12 bits menos significativos, sendo 4 bits para cada saída: 0 a 3, *OpA*, 4 a 7, *OpB* e 8 a 11, *OpC*. Ressaltamos que caso a *flag isImm* possua o valor 1, o valor em *OpB* será avaliado como um imediato de 4 bits e não como um endereço de registrador.

3.4 Multiplexador

O multiplexador implementado tem como entrada dois valores A e B além de um seletor *SEL*. A saída é o resultado da seleção do multiplexador: *S*.

Caso o valor em *SEL* seja 0, a saída é o valor de um registrador, indicado pela entrada A. Caso o valor seja 1, então o valor é um imediato estendido em 12 bits, vindo da entrada B. Com isso, a ALU fica genérica o suficiente para sua lógica não se preocupar com operações que recebam imediato ou valor de registrador. É importante frisar que o multiplexador é responsável apenas pela entrada B da Figura 1. Como explicado anteriormente, essa entrada que, caso a instrução requisite, recebe um imediato ao invés de um registrador.

3.5 Estágio do caminho de dados

Por último, descreveremos como implementamos o estágio do caminho de dados para saber o que fazer em cada *clock*.

Os estágios foram implementados como uma máquina de estados, ou seja, a cada subida de *clock*, mudamos de estado e fazemos um estágio do caminho de dados. Inicialmente, a máquina começa no estado de *Halt* esperando alguma ação. O estado muda assim que o módulo recebe um sinal de *START*. Quando o sinal chega, a máquina decodifica a instrução no decodificador. O próximo passo é ir no bando de registradores e buscar os valores requisitados pela instrução. Após a busca, a instrução é executada na ALU e então, o resultado é escrito no banco de registradores. Depois de escrever o resultado, a máquina volta ao estado inicial de *HALT* esperando a próxima instrução.

4 Integração

Foi especificado que a ULA deve possuir entradas para dois operandos de 16 bits e um entrada para o código da operação a ser executada, que possui 4 bits. Para este trabalho, utilizamos apenas 16 registradores, ou seja, apenas 4 bits são necessários para fazer o endereçamento no banco de registradores.

Como dito anteriormente, a ULA implementada deve ser capaz de realizar 11 operações diferentes. Observe que é possível, além de fazer operações com registradores, realizar operações com imediatos, *i.e.*, números absolutos. Esses números devem possuir 4 bits de largura.

A saída *RegA* do banco de registradores se conecta à entrada *OpA* do módulo da ULA, enquanto a saída *RegB*, se conecta à entrada *OpB* do módulo da ULA. Lembrando que essa entrada da ULA pode ser um imediato, a decisão de pegar a saída do banco de registradores ou o imediato segue como descrito na Seção 3.4.

4.1 Prototipação

A FPGA utilizada para a prototipação possui 4 botões para que possamos inserir dados e realizar as operações, 16 *switches* para informar o valor dos dados inseridos e 8 *displays* que mostram o valor dos dados. Cada *switch* possui dois estados: *cima* e *baixo*. Quando um *switch* está no estado *cima*, ele possui valor 1, e quando está no estado *baixo*, ele possui valor 0. Desta forma, cada *switch* se comporta com 1 bit do dado.

Quando o botão de nome *KEY0* for apertado, os *displays HEX7* e *HEX6* mostram o valor no conjunto de *switches SW8* a *SW11* (4 bits) e os *displays HEX5* e *HEX4* mostram o valor no conjunto de *switches SW4* a *SW7* (4 bits). Desta forma, é possível visualizar os valores passados para a placa.

Se apertarmos o botão *KEY3*, os *switches* serão interpretados como uma instrução completa, ou seja, com código da operação e operandos de entrada e saída, sendo que o *switch SW0* é o menos significativo enquanto o *SW15* o mais significativo. Assim, cada instrução possui 16 bits, como especificado no documento. O conjunto de *switches SW0* a *SW3* indica a entrada A e do *switch*

SW4 ao *SW7*, a entrada *B* da Figura 1. Lembrando que a entrada *B* pode ser um imediato. Já o conjunto de *switches SW8* a *SW11* indica a saída do resultado (saída R na Figura 1), enquanto os *switches SW12* a *SW15* indicam a operação a ser realizada, *i.e.*, o código da operação.

A Figura 4 mostra a divisão na placa.

5 Experimentos

Alguns experimentos foram realizados com o intuito de testar as funcionalidades da Unida Lógica Aritmética implementada em Verilog HDL. Observe que a ULA não distingue operações com registradores e imediatos, uma vez que os operandos A e B são tratados pelo módulo *Decoder* e transformados em sua representação númerica de 16 bits antes de serem encaminhados à ULA. Durante o estágio de decodificação, os valores dos registradores enderaçados pela instrução são lidos do banco de registradores e um *padding* de 12 bits é concatenado ao valor dos operandos imediatos. Dessa forma, a ULA precisa executar apenas seis tipos de instruções: ADD, SUB, SLT, AND, OR e XOR.

A tabela 4 descreve os resultados de onze instruções e o valor das *flags* da ULA após o término de cada operação. As colunas O, N e Z se referem respectivamente às flags *Overflow*, *Negative* e *Zero*. Os experimentos também estão descritos no formato de um *Test Bench* implementado no arquivo "*ULA.v*".

Referências

[1] John L. Hennessy I. and David A. Patterson. *Computer Architecture: a Quantitative Approach (Fifth Edition)*. 2012.

Instrução	Operando A	Operando B	Resultado	O	N	Z
ADD	0011 1111 1111 1111	0100 0000 0000 0000	0111 1111 1111 1111	0	0	0
ADD	0100 0000 0000 0000	0100 0000 0000 0000	1000 0000 0000 0000	1	1	0
ADD	0000 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0000	0	0	1
ADD	0010 0000 0000 0000	1100 0000 0000 0000	1110 0000 0000 0000	0	1	0
SUB	0100 0000 0000 0000	0011 1111 1111 1111	0000 0000 0000 0001	0	0	0
SUB	0011 1111 1111 1111	0100 0000 0000 0000	1111 1111 1111 1111	0	1	0
SLT	0100 0000 0000 0000	0000 0000 0000 0000	0000 0000 0000 0001	0	0	0
SLT	0000 0000 0000 0000	0100 0000 0000 0000	0000 0000 0000 0000	0	0	1
AND	0111 1111 1111 1111	0011 1111 1111 1111	0011 1111 1111 1111	0	0	0
OR	0000 0000 1111 1111	1111 1111 0000 0000	1111 1111 1111 1111	0	1	0
XOR	0101 0101 0101 0101	0010 1010 1010 1010	0111 1111 1111 1111	0	0	0

Tabela 4. Experimentos.

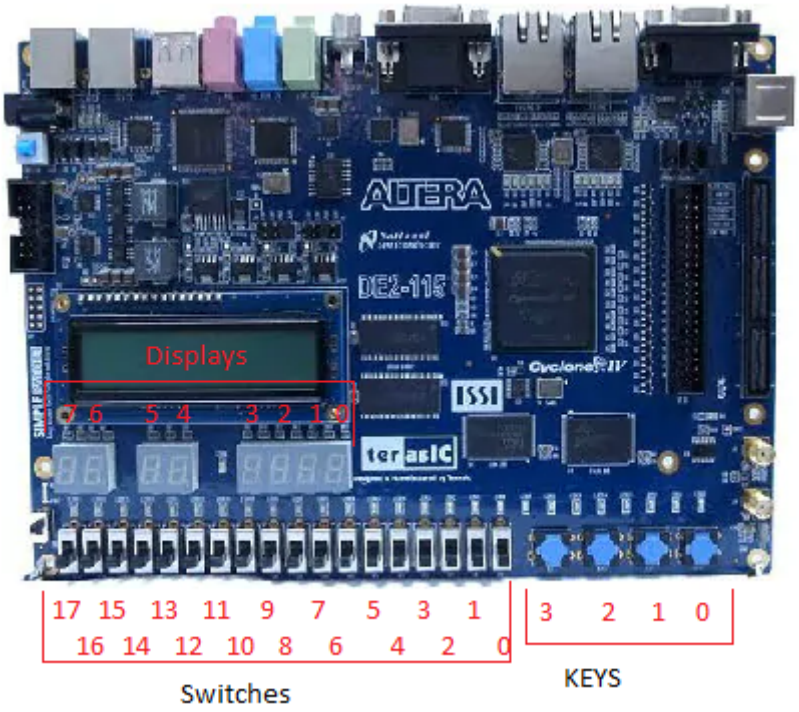

Figura 4. Módulo Altera DE2.