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, operações de adição, subtração, deslocamentos lógicos, operações lógicas *and* e *or*. A Figura 1 abaixo mostra a forma geral de uma ULA.

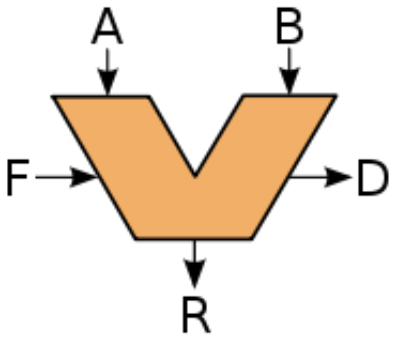

Figura 1: Unidade Lógica Aritmética.

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 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.

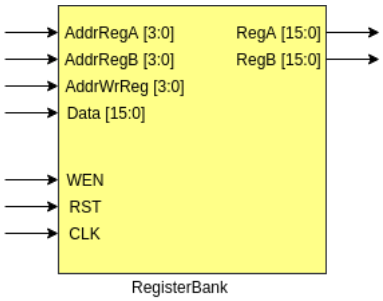

Figura 2. Bloco do Banco de Registradores.

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.

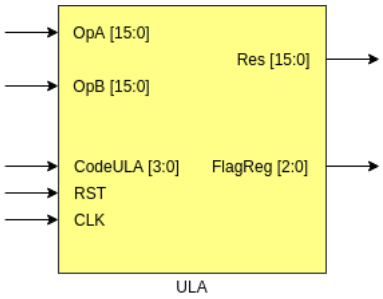

Figura 3. Bloco da ULA.

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 do resultado.

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 todos os campos do registro de estados *FlagReg*. O módulo trabalha com complemento de 2 para a descrição de números negativos.

3.3 Decodificador

O decodificador é responsável por receber a instrução e fazer a seleção correta dos operandos e da operação na ULA. Ele recebe a instrução de 16-bits na forma *INSTR DST, IMM|REGB, REGA*, onde

- *INSTR* (4-bit): Código da instrução, conforme a Tabela 1;
- *DST* (4-bit): Endereço do registrador de destino;
- *IMM/REGB* (4-bit): Endereço do registrador operando (RegB) ou imediato absoluto;
- *REGA* (4-bit): Endereço do segundo registrador operando (RegA).

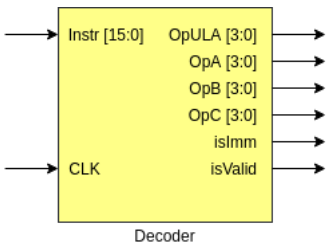

Figura 4. Bloco do Decodificador.

O módulo retorna, em suas saídas, os endereços dos operandos (*OpC*, *OpA* e *OpB*), a operação a ser executada pela ALU (*OpALU*), a indicação se o operando B é um registro ou um imediato (*isImm*) e indica se a instrução é válida ou não (*isValid*).

A sincronia é feita através do sinal de *clock*. Assim, a cada transição do sinal verifica-se o código da instrução, altera-se os sinais de *isImm* e *isValid*, ajusta os endereços dos operandos e a operação da ALU. Os códigos da *OpALU* seguem na Tabela 3.

4 Integração

Após a implementação dos módulos, faz-se a integração dos mesmos para testar o funcionamento da ULA e suas respostas as instruções, é criado o *Top-Level WrapperSim.v*. É necessário implementar uma pequena unidade de controle com uma máquina de estados para gerenciar o fluxo dos dados e a utilização de um multiplexador para fazer seleção entre imediato e registro, para o operando B. A Figura 5 mostra a integração do sistema, considere que os sinais de CLK e RST são globalmente interligados.

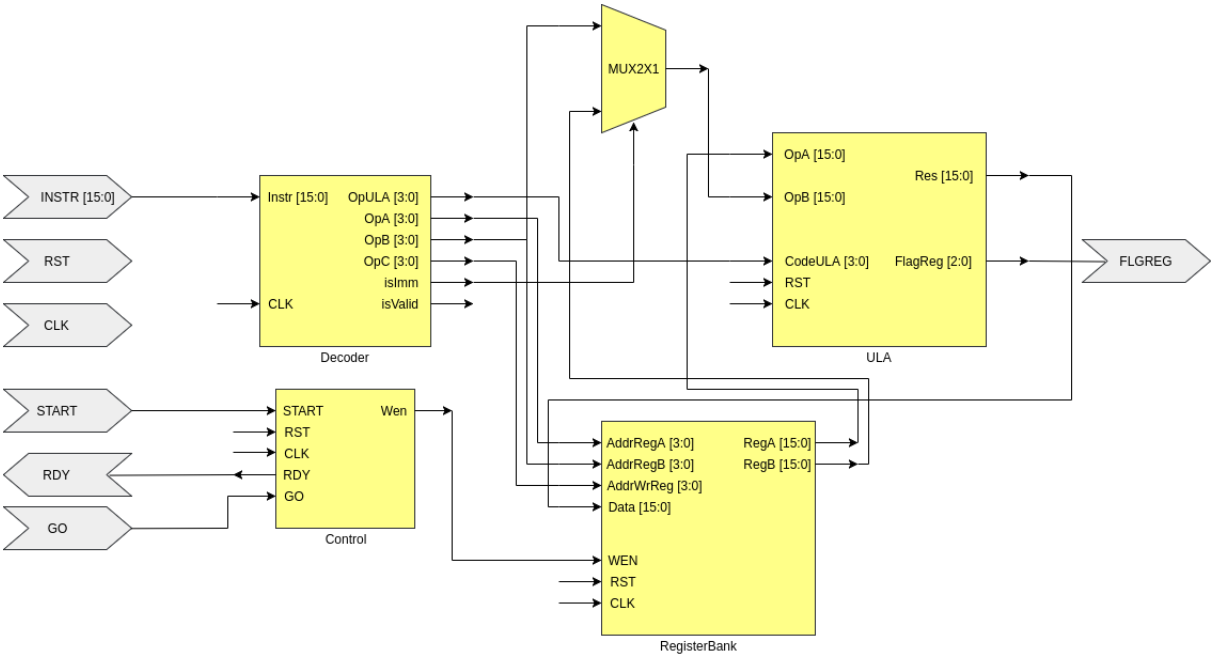

Figura 5. Integração.

O fluxo de dados segue o seguinte caminho: Decodificador, Banco de Registradores, ULA, Banco de Registradores. O sinal de *START* é utilizado para comandar o controle para iniciar a execução da instrução, e o sinal de *RDY* é utilizado pelo controle para indicar quando a máquina está pronta para executar/terminou execução da instrução passada. O sinal *GO* é utilizado para a implementação física, sendo necessário para fazer o processo de *debounce* do módulo Altera DE2. O sinal de *RST* reinicia a máquina, voltando para o estado inicial.

A progressão de estados é dado a cada transição positiva do sinal de *clock*. Inicialmente, a máquina começa no estado de *Halt* (S0) esperando alguma ação. O estado muda assim que o módulo recebe um sinal de *START*. Quando o sinal chega, a máquina faz a decodificação da instrução. O próximo passo é ir no banco 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. Para o próximo estágio (S5) a máquina aguarda o sinal de *GO* para poder retornar ao estado de *Halt*

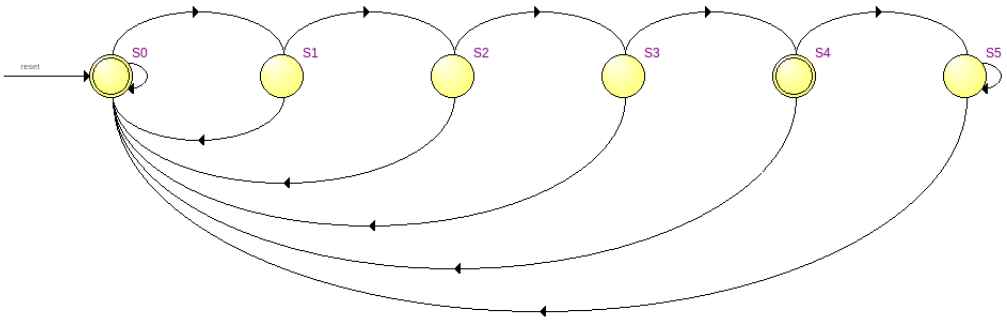

Figura 6. Máquina de estados do controle.

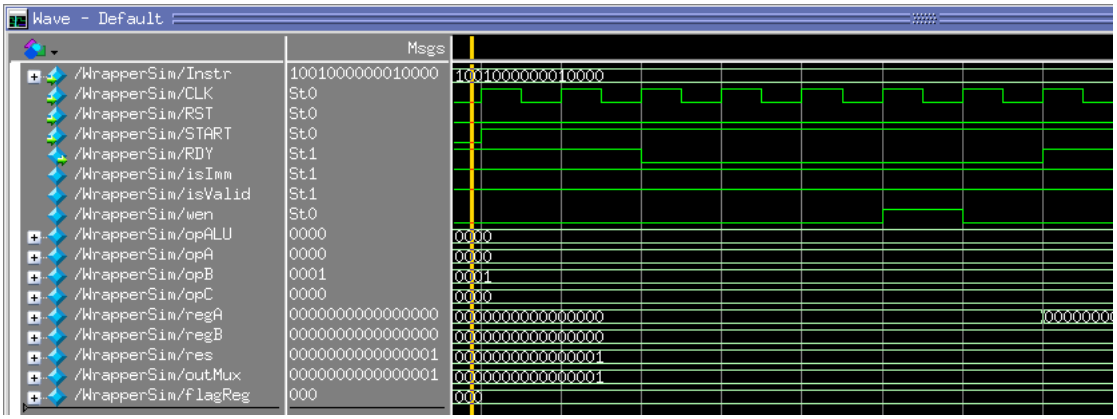

Figura 7. Formas de onda para execução da instrução ADDI.

(S0).

5 Simulação

Para o processo de simulação utiliza-se a ferramenta *ModelSim*.

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 endereçados 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*".

Para a simulação da máquina completa utilizando as unidades de controle, banco de registradores e decodificador, fez-se a execução de teste para a instrução ADDI Reg[0], 1, Reg[0]. As formas de ondas geradas podem ser visualizadas na Figura 7.

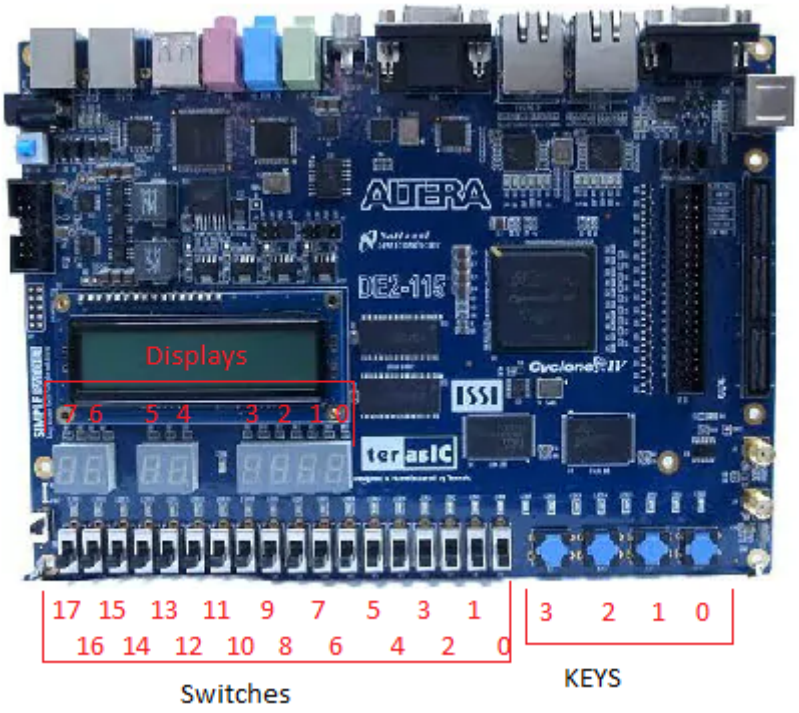

Figura 8. Módulo Altera DE2.

6 Prototipação

Para a prototipagem, utiliza-se a placa Altera DE2. Nela são utilizados três *pushbuttons* para executar comandos, 16 *switches* para inserir a instrução e dados, 1 *switche* para controle da máquina de estados e 8 *displays* de 7-segmentos para mostrar resultados de operações e registros. A Figura 8 mostra a divisão na placa.

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 *pushbuttom KEY0* é pressionado, os *displays HEX7* e *HEX6* mostram o valor atual do registrador endereçado pelo conjunto de *switches SW8* a *SW11* (4 bits) e os *displays HEX5* e *HEX4* mostram o valor atual do registrador endereçado pelo conjunto de *switches SW4* a *SW7* (4 bits). Desta forma, é possível visualizar os valores dos registradores no banco de registradores.

O *pushbuttom KEY1* é como sinal de *reset* para os módulos.

O botão *pushbuttom KEY3*, quando pressionado, faz com que os *switches* de 0 a 15 sejam interpretados como uma instrução completa, sendo que o *switch SW0* é o menos significativo enquanto o *SW15* o mais significativo. O conjunto de *switches SW0* a *SW3* indica o endereço *registrador A*, o conjunto do *switch SW4* ao *SW7* indicam o endereço do *registrador B* ou do imediado da instrução. O endereço do *registrador C* de destino é dado pelo conjunto de *switches SW8* a *SW11* e o *switches SW12* a *SW15* indicam a operação a ser realizada, ou seja, o código da operação conforme a Tabela 1.

Utilizou-se também a *switch 17* para fazer o controle da máquina de estados (sinal *GO* da máquina). Tal mecanismo é necessário devido o *clock* da placa ser sempre ativo e a falta de *debounce* nos *pushbuttons*, havendo efeitos de trepidação quando apertados fazendo com que a instrução seja executada várias vezes consecutivas. Assim, a cada execução, deve-se chavear a *switch 17* para

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.

retornar a máquina ao estado de *Halt*. Com isso, garante-se que a instrução é executada apenas uma vez.

Referências

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