LSDI 2013/14 - Laboratório 3

Construção de uma Unidade Lógica e Aritmética

1 - Introdução

Neste trabalho laboratorial será construído e ensaiado um circuito designado por Unidade Lógica e Aritmética ou ALU (*Arithmetic Logic Unit*). Este circuito é um componente importante de um microprocessador, sendo responsável pela realização das operações aritméticas e lógicas. O sistema digital que foi ensaiado no trabalho laboratorial anterior incluía uma unidade deste tipo, embora suportando apenas as operações de adição e subtração (circuito `addsub4b`). A escolha da operação a realizar era identificada pela entrada `add_sub`.

Neste trabalho pretende-se desenvolver uma ALU mais completa que realize operações entre números de 8 bits (A e B) e suporte um total de 8 operações. O símbolo lógico que representa esse circuito é mostrado na figura 1. Cada função a realizar é identificada por um código de 3 bits, aplicado na entrada OPR da ALU, conforme apresentado na tabela 1.

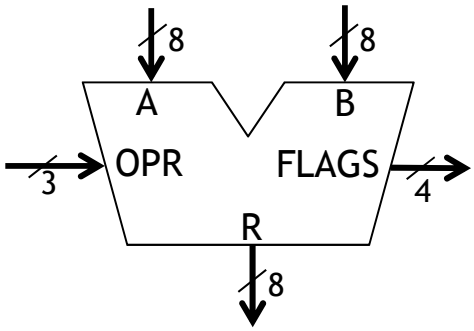

Figura 1 – Símbolo lógico representando a ALU a desenvolver.

Tabela 1 – Operações a realizar pela ALU determinadas pelo valor na entrada OPR.

OPR	Valor a produzir em R	Significado
000	B	R é igual ao valor de B
001	A - B	R é igual à diferença A – B
010	A + B	R é igual à adição A + B
011	A ^ B	R é igual ao valor do OU-Exclusivo entre A e B
100	A >> 1	R é igual ao valor A dividido por 2 (ver nota)
101	A << 1	R é igual ao valor A multiplicado por 2
110	A & B	R é igual à operação AND entre os bits de A e B
111	A | B	R é igual à operação OR entre os bits de A e B

Nota: esta operação deve considerar A como representando um valor com sinal em complemento para 2. Assim, ao deslocar os bits de A uma posição para a direita deverá ser feita a extensão de sinal de forma a manter o sinal do valor representado em A.

A figura 2 mostra com detalhe a constituição do circuito, salientando-se dois blocos: o que calcula o resultado R para as várias funções, ou seja, a ALU propriamente dita, e o bloco FLAGS. A ALU é composta pelos circuitos que realizam as funções definidas (aritméticas e lógicas) e por um multiplexador que, em função do código em OPR, escolhe o resultado a apresentar na saída R. Quanto ao bloco FLAGS, tem como função determinar certas características do resultado de uma operação realizada, assinalando-as à saída na forma de *flags*. A designação *flag* é correntemente atribuída a um bit que, num sistema digital, serve para assinalar o estado verdadeiro ou falso de uma condição associada ao resultado de uma operação aritmética ou lógica. Por exemplo, quando o resultado de uma operação realizada é negativo, a *flag* de sinal é colocada a 1.

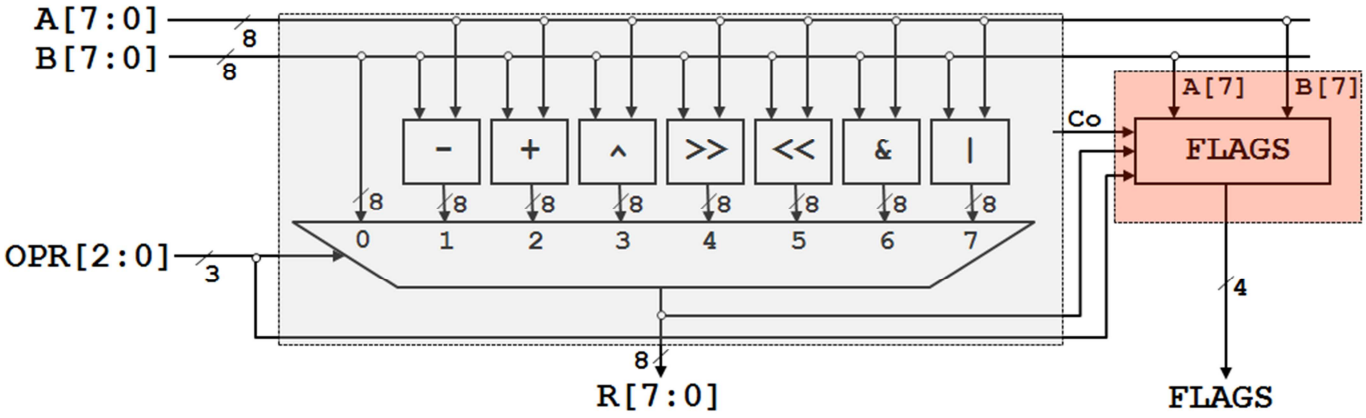

Figura 2 – Constituição da ALU a desenvolver.

O circuito FLAGS possui 4 saídas, correspondendo cada uma a uma *flag*, de acordo com o significado dado pela tabela 2. Este circuito é o único que precisa de ser construído, pois todos os restantes são fornecidos no projeto, em Verilog ou em forma esquemática.

Tabela 2 – Significado dos 4 bits da saída FLAGS.

Bit da saída FLAGS	Significado
FLAGS[0]	*flag* ZERO: é igual a 1 se o resultado R for igual a zero
FLAGS[1]	*flag* SINAL: é igual a 1 se R for negativo (bit de sinal igual a 1)
FLAGS[2]	*flag* CARRY: é igual a 1 se ocorrer *carry* na adição ou subtração
FLAGS[3]	*flag* OVFL: é igual a 1 se ocorrer *overflow* na adição ou subtração

Nota: nos casos contrários aos indicados o bit correspondente deverá ser igual a 0.

Este trabalho é composto por duas partes: projeto, a realizar em casa, e a simulação, implementação e ensaio a realizar na aula de laboratório. O trabalho de preparação em casa consiste no projeto de um dos circuitos constituintes da ALU (secção 2) e no estudo dos conteúdos da secção 4 deste guião, em particular as secções 4.3 e 4.4.

2 - Projeto

O circuito a projetar, FLAGS, determina o estado das 4 *flags*, como funções das respetivas entradas, tendo em consideração que:

- a *flag* ZERO apenas depende da entrada de 8 bits R, sendo ZERO=1 quando todos os bits de R forem 0 e ZERO=0 no caso contrário;
- a *flag* SINAL só depende do sinal de R, assumindo o valor do bit mais significativo de R;
- a *flag* CARRY depende da operação realizada (adição ou subtração) e da ocorrência de *carry-out* nessa operação, ou seja, depende das entradas OPR e Co. Deverá ser CARRY=1 apenas quando a operação realizada é uma adição (OPR=010) ou uma subtração (OPR=001) e caso o *carry* resultante seja 1. Caso contrário, CARRY=0;
- a *flag* OVFL depende da operação realizada (adição ou subtração), do sinal dos operandos A e B, e do sinal do resultado R. Recorde o que aprendeu sobre as condições de ocorrência de *overflow*, na adição e na subtração, em função do sinal dos operandos e do resultado.

Deverá utilizar as seguintes designações para as entradas e saídas do circuito a desenvolver:

- Entradas: `sA` (sinal de A), `sB` (sinal de B), `R` (resultado de uma operação), Co (*carry-out*) e OPR (código da operação);
- Saídas: `ZERO`, `SINAL`, `CARRY` e `OVFL`, com o significado dado na tabela 2.

a) Desenhe o circuito lógico que realiza a função das saídas ZERO e SINAL.

b) As saídas CARRY e OVFL dependem de a operação realizada ser a adição ou a subtração. Comece por definir a função AS (Adição/Subtração), que é 1 se OPR=010 ou OPR=001 e 0 no caso contrário, e desenhe o respetivo circuito lógico.

c) Desenhe o circuito que gera a saída CARRY como função de AS e Co.

d) Obtenha o circuito que realiza a saída OVFL. Para tal, comece por definir a condição de ocorrência de *overflow* numa tabela de verdade, tendo como entradas OPR[0] (o LSB do código da operação permite distinguir se é uma adição ou uma subtração), R[7] (sinal do resultado), sA (sinal de A) e sB (sinal de B). Seguidamente, escreva a expressão de OVFL através de uma soma de produtos e desenhe o circuito lógico.

e) Obtenha a saída OVFL, recorrendo à função AS (alínea b)) e ao circuito anterior (alínea d)) que verifica a condição de *overflow*. Note que OVFL só pode ser 1 se AS for 1 e a condição de *overflow* for verdadeira.

3 – Simulação do circuito

Com o objetivo de verificar o funcionamento da ALU antes da sua implementação na placa de prototipagem, vai proceder-se à simulação do módulo `lsdalu` (ilustrado na figura 2), que implementa a ALU propriamente dita e o circuito FLAGS que obteve. Para tal:

a) Edite o ficheiro `detFlags.sch` incluído no projeto e complete-o com o circuito FLAGS que preparou em casa, tendo o cuidado de não alterar o nome de nenhum sinal. Utilize as portas lógicas que entender necessárias, disponíveis na biblioteca de símbolos "*Categories* -> *Logic*". Note que, ao contrário do que aconteceu no trabalho laboratorial 2, não vai ser modelado o atraso das portas lógicas. Por conseguinte, nas simulações que realizar observará as saídas sem atraso face às entradas.

b) Analise o conteúdo do ficheiro `lsdalu.v` e simule-o usando o modelo de teste dado no ficheiro `lsdalu_testbench.v`, verificando atentamente que o circuito funciona como esperado. Note que o circuito a simular tem um total de 19 entradas (3+8+8) e 12 (4+8) saídas, e por isso uma simulação exaustiva (i.e. aplicando nas entradas todos os valores possíveis) obrigaria a testar todas as 2^{19}=524288 situações diferentes para cada saída. O modelo de teste não faz esta verificação exaustiva, aplicando apenas alguns valores em `A` e `B` para cada uma das operações (`OPR`), mostrando o valor dos resultados (`R` e `FLAGS`).

4 – Implementação e ensaio

4.1 – Projeto ISE

Para ensaiar a ALU desenvolvida, esta é integrada num projeto do *software* ISE para a placa S3board, que implementa uma unidade de cálculo elementar. O circuito lógico que está construído neste projeto é ilustrado na figura 3 onde se mostra a ligação das entradas e saídas aos botões de pressão, interruptores, LEDs e mostradores.

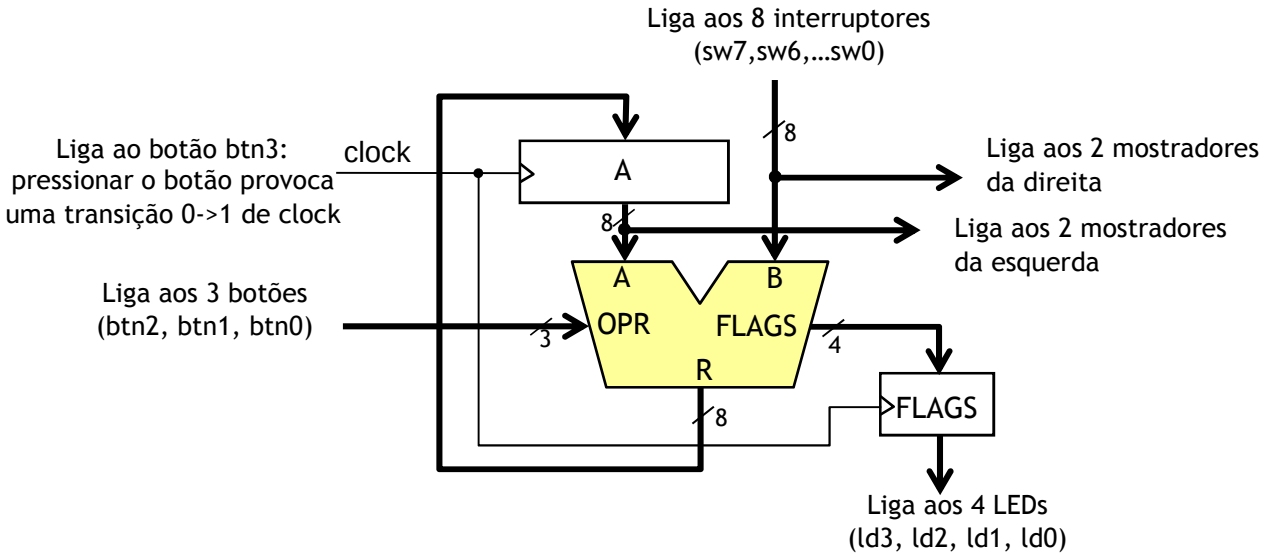

Figura 3 – Circuito lógico para ensaio da ALU.

As entradas e saídas do circuito definem-se da seguinte forma:

- Os 8 interruptores (sob os mostradores) permitem definir, em binário, o valor do operando B da ALU (colocando um interruptor para cima define um 1 e para baixo representa um 0). O valor colocado nestes interruptores é afixado nos 2 mostradores da direita em hexadecimal.

- O valor da entrada OPR da ALU é definido com os 3 botões de pressão do lado direito (btn2, btn1, btn0). Pressionando um botão coloca o bit correspondente a 1; caso contrário é 0. Por exemplo, para definir a operação OU-exclusivo deve ser aplicado na entrada OPR o valor 011 o que é feito pressionando os botões btn1 e btn0.

Na figura 3, **A** e **FLAGS** são registos (conjuntos de *flip-flops* do tipo D) que são atualizados sempre que o sinal **clock** passa de 0 para 1. Esta ação é realizada pressionando o botão de pressão btn3 (botão mais à esquerda), o que faz carregar o registo **A** com o resultado atual da saída **R** da ALU e o registo **FLAGS** com o valor que a ALU produz na saída **FLAGS**. O valor presente no registo **A** é afixado nos 2 mostradores do lado esquerdo e o estado das 4 *flags* é mostrado nos 4 LEDs da direita: (ld3, ld2, ld1, ld0) = (OVFL, CARRY, SINAL, ZERO).

4.2 – Implementação do projeto

Para implementar o projeto fornecido, com a sua ALU, basta selecionar o módulo `lsdi_lab3`, e fazer duplo clique em "*Generate Programming File*" ou em "*Configure Target Device*" (disponíveis em "*Processes*"). Caso não ocorram erros na implementação, programe a FPGA da placa e experimente o circuito.

4.3 - Como utilizar o circuito?

O circuito permite efetuar operações elementares entre um número de 8 bits e o valor presente no registo **A**. Para realizar uma operação basta definir o valor do operando **B** atuando nos 8 interruptores, pressionar os botões que definem a operação a realizar e (mantendo estes premidos) pressionar e largar o botão btn3 (esta ação provoca uma transição de 0 para 1 no sinal **clock**). O resultado produzido pela ALU é carregado no registo **A** e mostrado em hexadecimal nos 2 mostradores da esquerda.

Por exemplo, para efetuar a adição 5 + 23 devem ser efetuados os seguintes passos:

1º para carregar o valor 5 no registo **A**: colocar os interruptores com 00000101, não pressionar nenhum dos botões que definem **OPR** (ou seja, **OPR**=000 o que coloca na saída da ALU o valor de **B**) e atuar a entrada *clock* para carregar o registo **A** com o valor presente na saída da ALU que é o valor de **B**;

2º para adicionar 23 ao valor de **A** e colocar o resultado no registo **A**: colocar os interruptores com 00010111 (**B**=23), colocar **OPR**=010 (código da adição) e atuar a entrada *clock*.

Combinando várias destas operações simples é possível efetuar cálculos com alguma complexidade, embora restritos a valores numéricos representados com 8 bits. Cada operação elementar, como as envolvidas no exemplo anterior, pode ser representada com uma notação simbólica que torna mais fácil especificar uma sequência de operações necessárias para efetuar um determinado cálculo (tabela 3).

Tabela 3 – Operações elementares realizadas pelo circuito dado e representação simbólica de cada uma.

OPR	Valor colocado no registo A	operação simbólica	Significado
000	B	A ← B	A fica com o valor de B
001	A - B	A ← A - B	A fica com o resultado de A – B
010	A + B	A ← A + B	A fica com o resultado de A – B
011	A ^ B	A ← A ^ B	A fica com o resultado do XOR entre A e B (A ^ B)
100	A >> 1	A ← A >> 1	A fica com o valor de A deslocado para a direita 1 bit
101	A << 1	A ← A << 1	A fica com o valor de A deslocado para a esquerda 1 bit
110	A & B	A ← A & B	A fica com o resultado do E entre A e B (A & B)
111	A | B	A ← A | B	A fica com o resultado do OU entre A e B (A | B)

Por exemplo, para calcular o valor da expressão (((23 − 56) / 2) + 34) × 2, pode formar-se uma lista de 5 operações simbólicas que podem depois ser executadas na ALU (uma em cada ciclo do sinal de relógio):

Operação simbólica	**Como realizar**
A ← 23	Colocar B com 23 e OPR com 000
A ← A − 56	Colocar B com 56 e OPR com 001
A ← A >> 1	Colocar OPR com 100 (o valor de B não interessa)
A ← A + 34	Colocar B com 34 e OPR com 010
A ← A << 1	Colocar OPR com 101 (o valor de B não interessa)

4.4 – Ensaio do circuito

a) Defina um conjunto de dados e de operações que lhe permitam verificar todas as operações que a ALU consegue realizar, verificando o valor das saídas FLAGS para cada caso. Relembre-se que está a operar em complemento para 2.

b) Construa conjuntos de operações elementares, usando a notação simbólica introduzida acima, que permitam calcular as expressões aritméticas seguintes:

1: 4 + 4 + 5 + 6
2: −89 − (9 × 4) − 45
3: (13 × 9) / 2
4: −3 × (15 + 8) − 99
5: 127 + (−128)

c) Execute as sequências de operações elementares que construiu na alínea anterior e confirme os resultados obtidos.