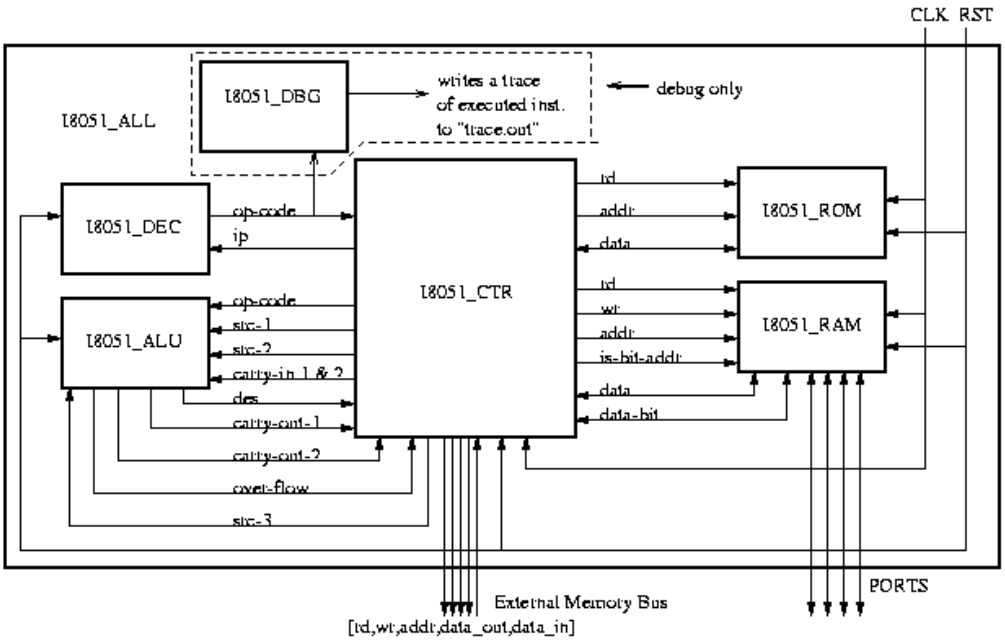
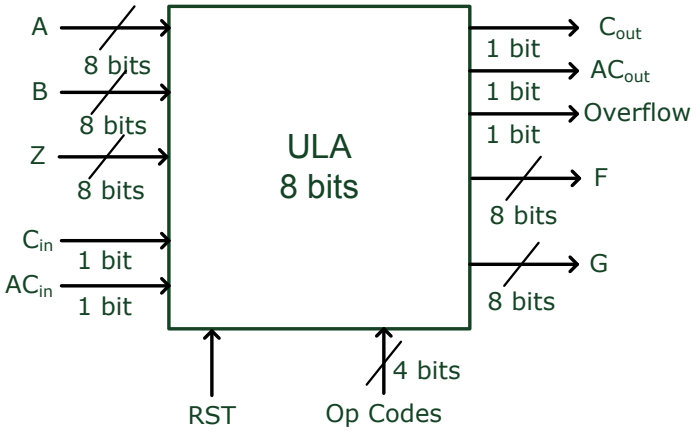
Projeto 2 de Sistemas Digitais 1
Unidade Lógico-Aritmética
do Microcontrolador 8051

I. Projeto

O Intel MCS-51 (popularmente conhecido como 8051) é um microcontrolador de 8 bits que segue a arquitetura de Harvard projetado para ser utilizado em sistemas embarcados. Hoje em dia diversos outros fabricantes vendem chips com essa arquitetura, como a Atmel, a Maxim, a Texas Instruments e muitas outras. Embora ele tenha um baixo poder de processamento, O 8051 possui um conjunto de instruções complexo (CISC), o que significa que uma única instrução pode executar várias operações de baixo nível (como carregar da memória, realizar uma operação aritmética e gravar na memória). Embora este seja um sistema simples, este sistema de instruções é capaz de executar qualquer algoritmo. O diagrama de blocos deste microcontrolador pode ser visto na figura abaixo.



A ideia principal do projeto é implementar, em VHDL, a unidade lógico-aritmética (ULA) do 8051. A ULA é um circuito puramente combinacional, cuja interface é mostrada abaixo:



O circuito mostra uma unidade lógico-aritmética (ULA) com as seguintes entradas / saídas:

Nome	Tipo	Tamanho	Descrição
A	Entrada	8 bits	Primeiro Operando
B	Entrada	8 bits	Segundo Operando
Z	Entrada	8 bits	Terceiro Operando
Cin	Entrada	1 bit	Carry para o BIT 7
ACin	Entrada	1 bit	Carry para o BIT 4
RST	Entrada	1 bit	Reset
S	Entrada	4 bits	Seletores (opcodes)
F	Saída	8 bits	Saída Principal
G	Saída	8 bits	Saída Secundária
Cout	Saída	1 bit	Carry out para o BIT 7
ACout	Saída	1 bit	Carry out para o BIT 4
OV	Saída	1 bit	Overflow

II. Definição dos Opcodes

A ULA tem 4 bits de seleção e portanto pode realizar $2^4 = 16$ operações distintas. Note que nem todas as opções utilizam todas as saídas - caso um determinado opcode não utilize uma das saidas, esta saída deve ser assinalada como irrelevantes (don't cares, ou '-'). Além disso, quando a entrada RESET estiver ativa, todas as saídas devem ser assinaladas como don't care.

A. Opcode ***0000*** - *None*

Nenhuma ação (isto é, todas as saídas devem ser assinaladas como don't care).

B. Opcode ***0001*** - *ADD*

Operação de adição de 8 bits:

Saída	Descrição	Equação
F	Soma dos operandos	$\mathbf{A + B + Cin}$
Cout	Carry out do BIT 7	
ACout	Carry out do BIT 4	
OV	Overflow (signed carry out)	

C. Opcode ***0010*** - *SUB*

Operação de subtração de 8 bits:

Saída	Descrição	Equação
F	Subtração dos operandos	$\mathbf{A - B - Cin}$
Cout	Borrow out do BIT 7	
ACout	Borrow out do BIT 4	
OV	Overflow (signed borrow out)	

D. Opcode ***0011*** - *MUL*

Operação de multiplicação de 8 bits. A multiplicação é realizada entre os dois operandos de 8 bits ($\mathbf{A}$ e $\mathbf{B}$) e a saída é colocada em $\mathbf{F}$ (8 bits menos significativos, LSB) e em $\mathbf{G}$ (8 bits mais significativos, MSB). A multiplicação é realizada *sem sinal*.

Saída	Descrição	Equação
F	LSB da Multiplicação dos operandos	$\mathbf{A \times B}$
G	MSB da Multiplicação dos operandos	$\mathbf{A \times B}$
OV	Overflow (se o resultado tem mais de 8 bits)	

*E. Opcode **0100** - DIV*

Operação de divisão de 8 bits. A divisão é realizada entre os dois operandos de 8 bits ($\mathbf{A}$ e $\mathbf{B}$) e a saída é colocada em $\mathbf{F}$ (quociente) e em $\mathbf{G}$ (resto da divisão). Na tentativa de divisão por zero, a saída deverá ser irrelevante.

Saída	Descrição	Equação
$\mathbf{F}$	Quociente da divisão	$\mathbf{A} \div \mathbf{B}$
$\mathbf{G}$	Resto da divisão	$\mathbf{A} \% \mathbf{B}$
$\mathbf{OV}$	Overflow (indica divisão por zero)	

*F. Opcode **0101** - DA*

A operação de Ajuste Decimal (Decimal Adjust) ajusta o conteúdo do operando para que ele corresponda a um número BCD (Binary Coded Decimal) após dois números BCD terem sido somados pela operação de adição. Um número BCD coloca cada dígito em um *nibble* (4 bits) do operando (e, portanto, o operando pode guardar 100 números, de 00 a 99). O carry do BIT 7 ($\mathbf{Cin}$) funciona como o carry do segundo dígito enquanto o carry do BIT 3 ($\mathbf{ACin}$) funciona como o carry do primeiro bit. Logo, se $\mathbf{ACin}$ estiver setado ou se o valor do *nibble* LSB for maior do que 0x09 (isto é, se o o dígito das unidades for maior que 9), então 0x06 é somado ao operando (para passar este carry para o segundo dígito, das dezenas). Se $\mathbf{Cin}$ estiver setado, ou se 0x06 foi adicionado ao operando no primeiro passo, então 0x60 é adicionado ao operando. O bit de Carry out ($\mathbf{Cout}$) é setado se o valor resultante for maior do que 0x99.

Saída	Descrição	Equação
$\mathbf{F}$	Operador ajustado	$\mathbf{A}$
$\mathbf{Cout}$	Indica se o operador ajustado for maior que 0x99	

*G. Opcode **0110** - NOT*

Operação **not** bit a bit:

$$\mathbf{F} = \overline{\mathbf{A}}$$

*H. Opcode **0111** - AND*

Operação **and** bit a bit:

$$\mathbf{F} = \mathbf{A} \cdot \mathbf{B}$$

*I. Opcode **1000** - XOR*

Operação **xor** bit a bit:

$$\mathbf{F} = \mathbf{A} \oplus \mathbf{B}$$

*J. Opcode **1001** - OR*

Operação **or** bit a bit:

$$\mathbf{F} = \mathbf{A} + \mathbf{B}$$

*K. Opcode **1010** - RL*

Operação de rotacionar à esquerda (rotate left).

$$\mathbf{F} = [\mathbf{F_7}\ \mathbf{F_6}\ \mathbf{F_5}\ \mathbf{F_4}\ \mathbf{F_3}\ \mathbf{F_2}\ \mathbf{F_1}\ \mathbf{F_0}] = [\mathbf{A_6}\ \mathbf{A_5}\ \mathbf{A_4}\ \mathbf{A_3}\ \mathbf{A_2}\ \mathbf{A_1}\ \mathbf{A_0}\ \mathbf{A_7}]$$

*L. Opcode **1011** - RLC*

Operação de rotacionar à esquerda com carry (rotate left carry).

$$\mathbf{F} = [\mathbf{F_7}\ \mathbf{F_6}\ \mathbf{F_5}\ \mathbf{F_4}\ \mathbf{F_3}\ \mathbf{F_2}\ \mathbf{F_1}\ \mathbf{F_0}] = [\mathbf{A_6}\ \mathbf{A_5}\ \mathbf{A_4}\ \mathbf{A_3}\ \mathbf{A_2}\ \mathbf{A_1}\ \mathbf{A_0}\ \mathbf{Cin}]$$
$$\mathbf{Cout} = \mathbf{A_7}$$

*M. Opcode **1100** - RR*

Operação de rotacionar à direita (rotate right).

$$\mathbf{F} = [\mathbf{F_7}\ \mathbf{F_6}\ \mathbf{F_5}\ \mathbf{F_4}\ \mathbf{F_3}\ \mathbf{F_2}\ \mathbf{F_1}\ \mathbf{F_0}] = [\mathbf{A_0}\ \mathbf{A_7}\ \mathbf{A_6}\ \mathbf{A_5}\ \mathbf{A_4}\ \mathbf{A_3}\ \mathbf{A_2}\ \mathbf{A_1}]$$

*N. Opcode **1101** - RRC*

Operação de rotacionar à direita com carry (rotate right carry).

$$\mathbf{F} = [\mathbf{F_7}\ \mathbf{F_6}\ \mathbf{F_5}\ \mathbf{F_4}\ \mathbf{F_3}\ \mathbf{F_2}\ \mathbf{F_1}\ \mathbf{F_0}] = [\mathbf{Cin}\ \mathbf{A_7}\ \mathbf{A_6}\ \mathbf{A_5}\ \mathbf{A_4}\ \mathbf{A_3}\ \mathbf{A_2}\ \mathbf{A_1}]$$
$$\mathbf{Cout} = \mathbf{A_0}$$

*O. Opcode **1110** - PCSADD*

Operação de adição de 16 bits. Realiza a soma do número de 16 bits formado por $\mathbf{B}$ (MSB) e $\mathbf{A}$ (LSB) com o número de 8 bits $\mathbf{Z}$. O número $\mathbf{Z}$ é considerado como com sinal. A saída é colocada nos operandos $\mathbf{G}$ (MSB) e $\mathbf{F}$ (LSB). O carry out **não** é setado como parte dessa operação.

Saída	Descrição	Equação
$[\mathbf{GF}]$	Soma dos operandos	$[\mathbf{GF}] + bvZ$

*P. Opcode **1111** - PCUADD*

Operação de adição de 16 bits. Realiza a soma do número de 16 bits formado por $\mathbf{B}$ (MSB) e $\mathbf{A}$ (LSB) com o número de 8 bits $\mathbf{Z}$. O número $\mathbf{Z}$ é considerado como sem sinal. A saída é colocada nos operandos $\mathbf{G}$ (MSB) e $\mathbf{F}$ (LSB).

Saída	Descrição	Equação
$[\mathbf{GF}]$	Soma dos operandos	$[\mathbf{GF}] + bvZ$

III. Objetivos e Restrições

O projeto deve ser entregue em VHDL em um arquivo ZIP (zip, e não RAR!) com a matrícula do aluno.

No caso do código em VHDL, o circuito deve ser criado e simulado utilizando **apenas** a biblioteca IEEE.STD_LOGIC_1164, que inclui os tipos STD_LOGIC e STD_LOGIC_VECTOR (todos os barramentos devem ser declarados como vetores). **Não** podem ser utilizados os operadores relacionais e aritméticos (uma vez que eles se encontram em outra biblioteca).

Além disso, é **obrigatório** que a entidade tenha **exatamente** a formatação mostrada no Algoritmo 1 (incluindo os nomes das variáveis). Projetos com formatações diferentes **não** serão corrigidos!

Algoritmo 1 Entidade do Projeto

```
1:  library IEEE;
2:  use IEEE.STD_LOGIC_1164.ALL;
3:
4:  entity Projeto2 is
5:          port( RST   : in STD_LOGIC;
6:                S     : in STD_LOGIC_VECTOR (3 downto 0);
7:                A     : in STD_LOGIC_VECTOR (7 downto 0);
8:                B     : in STD_LOGIC_VECTOR (7 downto 0);
9:                Z     : in STD_LOGIC_VECTOR (7 downto 0);
10:               Cin   : in STD_LOGIC;
11:               ACin  : in STD_LOGIC;
12:               F     : out STD_LOGIC_VECTOR (7 downto 0);
13:               G     : out STD_LOGIC_VECTOR (7 downto 0);
14:               Cout  : out STD_LOGIC;
15:               ACout : out STD_LOGIC;
16:               OV    : out STD_LOGIC);
17: end Projeto2;
```