Lógica e Sistemas Digitais - 6

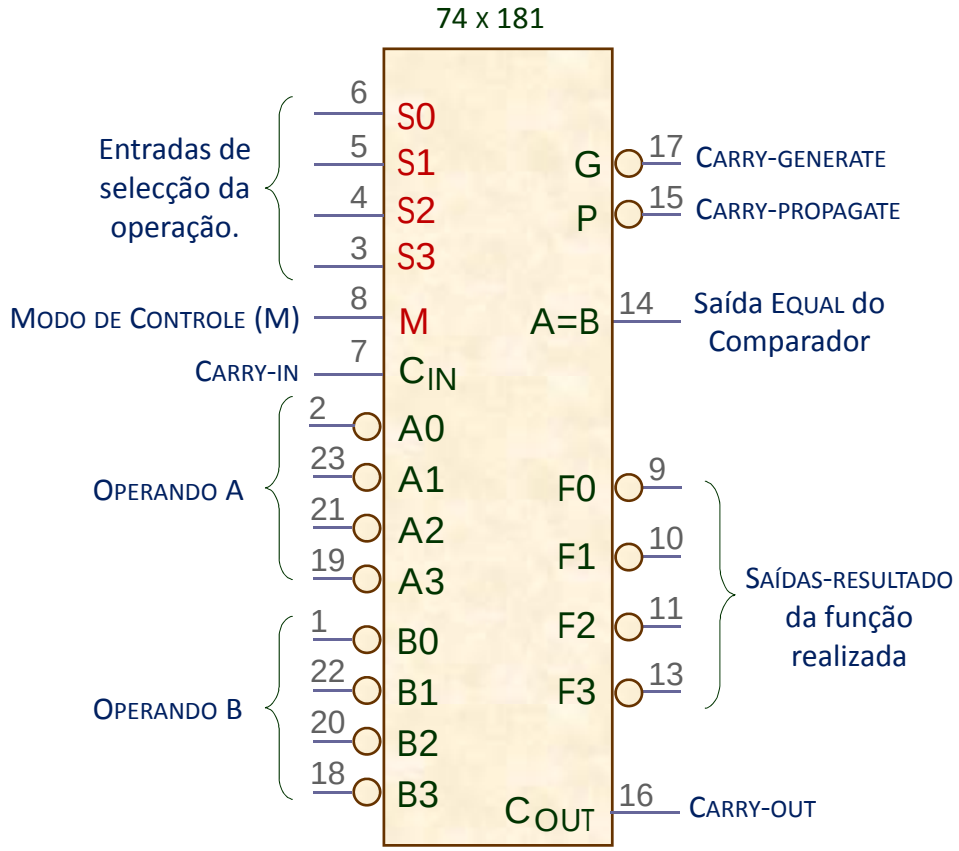
ALUs
e
Exemplo de Aplicação

ISEL
Departamento de Engenharia de Electrónica
e Telecomunicações e de Computadores
Lisboa

Mário Araújo

2016-1

ALU a 4 Bits em Circuito MSI



Símbolo lógico e configuração dos pinos de um circuito 74x181 4-bit Arithmetic Logic Unit (ALU de 4-bits).

- Os processadores dispõem de uma unidade capaz de realizar as operações básicas da aritmética e da lógica, designada **ALU** (Arithmetic Logic Unit).
- Uma ALU é um circuito combinatório que realiza operações aritméticas básicas (como adição e subtracção) e operações lógicas num par de operandos de n bits.
- A operação a ser realizada é especificada pela combinação de um conjunto de sinais selectores de entrada que a codificam.
- As ALUs MSI disponíveis usam operandos de 4-bits e entradas de selecção (S_i) com um conjunto de bits que varia entre 3 e 5, o que permite a realização de um número de funções até 32 .
- O circuito 74x181 é um exemplo de uma ALU de 4 bits realizada num circuito integrado MSI da família TTL.
- As funções a realizar são seleccionadas pelas entradas M (Modo) e S_0 a S_3 (Selecção).

Tabela Funcional de uma ALU a 4 Bits

Entradas Selectoras				Função Seleccionada (Entradas e Saídas Active-low)	
S3	S2	S1	S0	**M=0** Operações Aritméticas em Complemento para 2	**M=1** Operações Lógicas
0	0	0	0	F = A − 1 + CIN	F = A’
0	0	0	1	F = (A · B) − 1 + CIN	F = A’ + B’
0	0	1	0	F = (A · B’) − 1 + CIN	F = A’ + B
0	0	1	1	F = 1111 + CIN	F = 1111
0	1	0	0	F = A + (A + B’) + CIN	F = A’ · B’
0	1	0	1	F = (A · B) + (A + B’) + CIN	F = B’
0	1	1	0	F = A − B − 1 + CIN	F = A ⊕ B’
0	1	1	1	F = (A + B’) + CIN	F = A + B’
1	0	0	0	F = A + (A + B) + CIN	F = A’ · B
1	0	0	1	F = A + B + CIN	F = A ⊕ B
1	0	1	0	F = (A · B’) + (A + B) + CIN	F = B
1	0	1	1	F = (A + B) + CIN	F = A + B
1	1	0	0	F = A + A + CIN	F = 0000
1	1	0	1	F = (A · B) + A + CIN	F = A · B’
1	1	1	0	F = (A · B’) + A + CIN	F = A · B
1	1	1	1	F = A + CIN	F = A

O sinal **+** nas funções lógicas, e nas funções aritméticas quando entre parêntesis (e neste caso a vermelho), significa um OR. Nos restantes casos representa a adição aritmética. O sinal · tem um comportamento idêntico mas dual.

- As saídas P_L (Carry Propagate) e G_L (Carry Generate) da ALU no slide anterior são usadas na concatenação de várias ALUs para a propagação rápida Carry-lookahead.
- Na adição em complemento para 2, selecciona-se a operação A+B+CIN. Na subtracção em complemento para 2, selecciona-se A–B–CIN. Neste caso CIN funciona como BIN (Borrow-in) e terá de ser invertido na ALU menos significativa da cadeia de concatenação.
- Os operandos A e B e o resultado F são do tipo Active-low. A tabela da ALU seria diferente se fossem todos interpretados como Active-high.
- A entrada M seleciona o cálculo de funções lógicas (M=1) ou de funções aritméticas (M=0) para cada combinação das entradas de selecção S_0-S_3 de acordo com a tabela.

FLAGS RELACIONAIS

As FLAGS, também designadas CONDITION CODES ou CONDITION BITS, correspondem a bits que são alterados de forma consistente de modo a darem uma informação útil sobre o resultado de uma operação. As instruções de ciclos condicionados (ou saltos) dos processadores testam Flags.

As FLAGS relacionais indicam a relação de grandeza entre dois números A e B. Como saber se A > B ?

- Verificando, através da subtracção (com B_{in} = 0), se o resultado de A-B > 0, sendo este o método utilizado na ALU hipotética analisada neste capítulo;
- Por comparação dígito a dígito no sentido dos algarismos de maior peso para os de menor peso, como já verificado no estudo do comparador.

O número, a nomenclatura e o significado das FLAGS varia com o tipo de processador.

Utilizam-se, no que se segue, as designações e as regras utilizadas nos processadores do tipo 80x86 Intel de acordo com a tabela adiante que referencia o nome, o domínio de aplicação $\mathbb{N}_0$ (quando na cor verde), ou $\mathbb{Z}$ (quando na cor vermelha), o teste aplicável a cada FLAG, e a condição equivalente tal como usualmente aparecem descritas na literatura disponível.

$\mathbb{N}_0 = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 \ldots\} = \mathbb{Z}_0^+$

$\mathbb{N}_0$ é o conjunto dos números inteiros naturais (sem sinal, UNSIGNED).

$\mathbb{Z} = \{\ldots -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 \ldots\}$

$\mathbb{Z}$ é o conjunto dos números inteiros relativos (com sinal, SIGNED) – para cada elemento a de $\mathbb{Z}$ existe o seu simétrico $-a$ que somado com a tem resultado 0.

FLAGS RELACIONAIS NA FAMÍLIA 80X86

Nomenclatura		Domínio	Equivalência	Activação
A	above	unsigned	NBE	not carry flag and not zero flag
AE	above or equal	unsigned	NB	not carry flag
B	below	unsigned	NAE	carry flag
BE	below or equal	unsigned	NA	carry flag or zero flag
C	carry	unsigned		carry flag
E(Q)	equal	unsigned	Z	zero flag
G	greater	signed	NLE	not ((sign flag xor overflow flag) or zero flag)
GE	greater or equal	signed	NL	not (sign flag xor overflow flag)
L	less	signed	NGE	sign flag xor overflow flag
LE	less or equal	signed	NG	(sign flag xor overflow flag) or zero flag
NA	not above	unsigned	BE	carry flag or zero flag
NAE	not above nor equal	unsigned	B	carry flag
NB	not below	unsigned	AE	not carry flag
NBE	not below nor equal	unsigned	A	not carry flag and not zero flag
NC	not carry	unsigned		not carry flag
NE	not equal	unsigned	NZ	not zero flag

Nomenclatura e Significado das Flags Relacionais no contexto da Família de processadores Intel 80x86 (a palavra Carry é usada em sentido lato, significando Borrow quando está implícita uma subtracção).

FLAGS RELACIONAIS NA FAMÍLIA 80X86

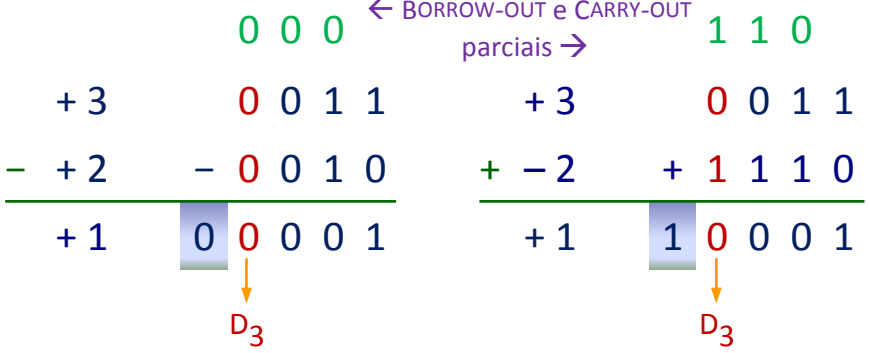
Nomenclatura		Domínio	Equivalência	Activação
NG	not greater	signed	LE	(sign flag xor overflow flag) or zero flag
NGE	not greater nor equal	signed	L	sign flag xor overflow flag
NL	not less	signed	GE	not (sign flag xor overflow flag)
NLE	not less nor equal	signed	G	not ((sign flag xor overflow flag) or zero flag)
NO	not overflow	signed		not overflow flag
NP	not parity	unsigned	PO	not parity flag
NS	not sign	signed		not sign flag
NZ	not zero	unsigned	NE	not zero flag
O(V)	overflow	signed		overflow flag
P	parity	unsigned	PE	parity flag
PE	parity even	unsigned	P	parity flag
PO	parity odd	unsigned	NP	not parity flag
S	sign (negative)	signed		sign flag
Z	zero	unsigned	E	zero flag

Nomenclatura e Significado das Flags Relacionais no contexto da Família de processadores Intel 80x86.

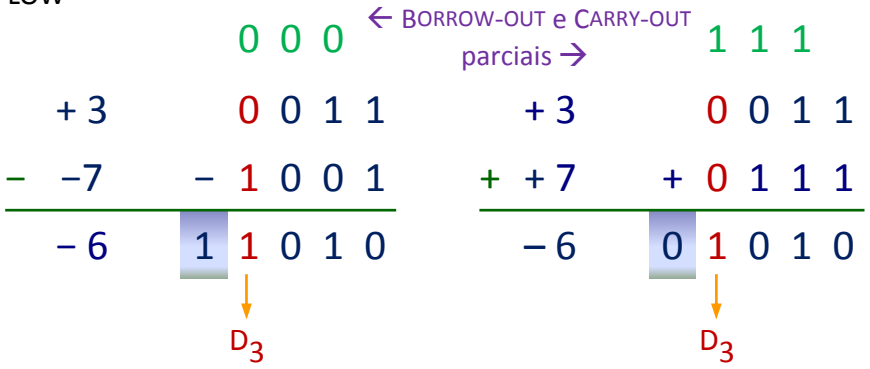
FLAGS RELACIONAIS E OVERFLOW (EX. 6-1)

Exemplo 6-1

SEM OVERFLOW



COM OVERFLOW



Dois exemplos de subtracção A-B a 4 bits: à esquerda na forma directa (papel e lápis), e à direita sob a forma de adição em código dos complementos; em cima num caso A>B com inexistência de OVERFLOW (e D_3=0), em baixo também num caso A>B mas com geração de OVERFLOW (e D_3=1).

Quando não há OVERFLOW (O_V), o bit de sinal D_3 da diferença entre dois operandos A e B a 4 bits indica directamente a relação entre A e B:

- $O_V = 0$ e $D_3 = 0 \Rightarrow$ A>=B.
- $O_V = 0$ e $D_3 = 1 \Rightarrow$ A<B.

Quando existe OVERFLOW, o bit de sinal D4 dessa diferença terá de ser invertido para se reverter para o caso anterior:

- $O_V = 1$ e $D_3 = 1 \Rightarrow$ A>=B.
- $O_V = 1$ e $D_3 = 0 \Rightarrow$ A<B.

D_3	O_V	CONDIÇÃO	GE	L
0	0	A≥B	1	0
0	1	A<B	0	1
1	0	A<B	0	1
1	1	A>B	1	0

Quadro representativo da afectação das flags GREATER OR EQUAL (GE) e LESS (L) pela existência de OVERFLOW.

FLAGS RELACIONAIS (EX. 6-2)

Exemplo 6-2

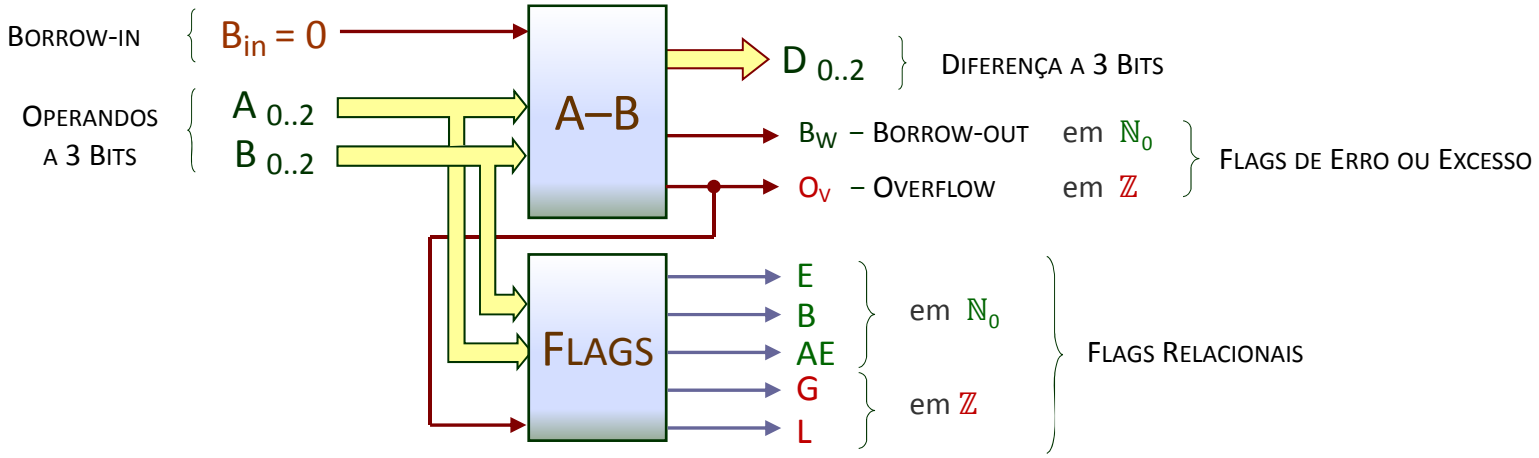
OPERAÇÃO	FLAGS G	L	AE	B	E	O_v	B_w	INTERPRETAÇÃO EM $\mathbb{N}_0$ (UNSIGNED)	INTERPRETAÇÃO EM $\mathbb{Z}$ (SIGNED)
0 0 1 1								3	+ 3
– 0 0 1 0								– 2	– + 2
0 0 0 0 1	1	0	1	0	0	0	0	1	+ 1
0 0 1 0								2	+ 2
– 0 0 1 1								– 3	– + 3
1 1 1 1 1	0	1	0	1	0	0	1	15	– 1
0 0 1 1								3	+ 3
– 1 0 0 1								– 9	– – 7
1 1 0 1 0	1	0	0	1	0	1	1	10	– 6
1 0 0 1								9	– 7
– 0 0 1 1								– 3	– + 3
0 0 1 1 0	0	1	1	0	0	1	0	6	+ 6
1 1 0 1								13	– 3
– 1 1 0 1								– 13	– – 3
0 0 0 0 0	0	0	1	0	1	0	0	0	+ 0

Exemplos de activação de Flags Relacionais na subtracção em $\mathbb{N}_0$ e em $\mathbb{Z}$.

Geração de Flags Relacionais

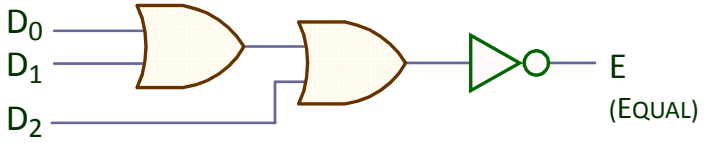
Os exemplos seguintes mostram algumas Flags Relacionais e o modo de as gerar numa ALU a 3 bits.

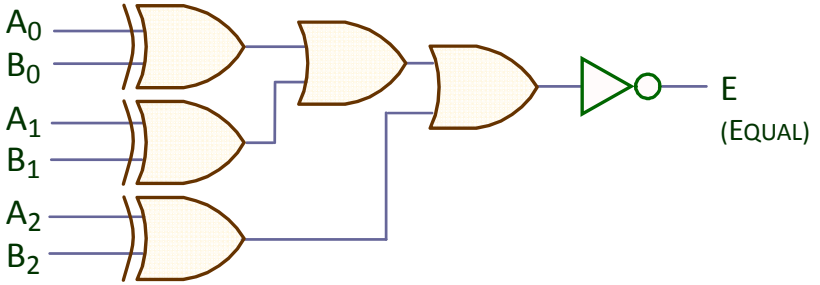
- **B** (Below), **E** (Equal) e **AE** (Above or Equal) para operações em $\mathbb{N}_0$, e
- **L** (Less) e **G** (Greater) para as operações em $\mathbb{Z}$.



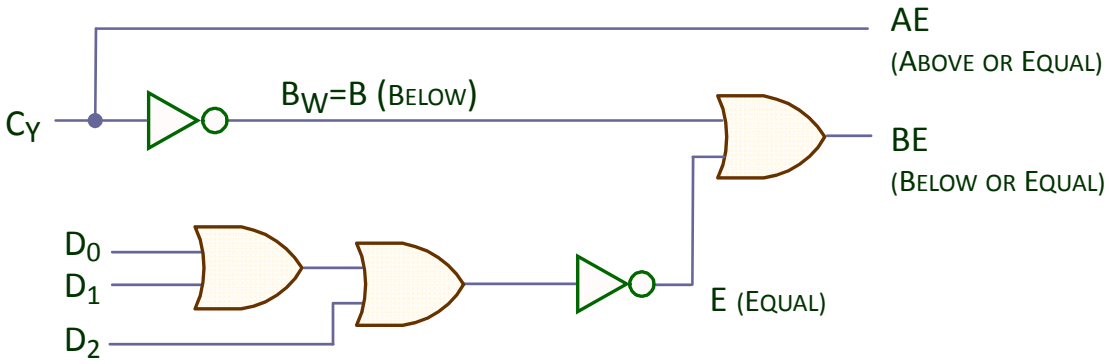
Bloco subtractor a 3 bits constituinte da ALU e lógica auxiliar para síntese das Flags Relacionais.

Flags Relacionais para Números sem Sinal





Flag EQUAL (ou ZERO), implementação para operandos a 3 bits no domínio $\mathbb{N}_0$: versão simples que utiliza os 3 bits do resultado DIFERENÇA (à esquerda), e versão que utiliza os 3 bits de cada operando (à direita).



Flags AE (ABOVE OR EQUAL) e BE (BELOW or EQUAL), implementação para números de 3 bits no domínio $\mathbb{N}_0$.

C_Y é o CARRY-OUT do somador e B_W o BORROW-OUT do subtractor que gera a diferença D = A–B (com B_{in} = 0).

A inexistência de B_W indica que A é maior que B entendendo-se A e B como números naturais.

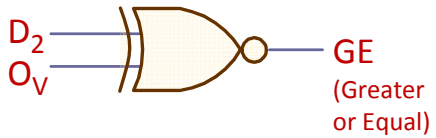
$A \geq B \Rightarrow B_w=0 \quad B=0 \quad AE=1$

$A < B \Rightarrow B_w=1 \quad B=1 \quad AE=0$

AE: ABOVE OR EQUAL

B: BELOW (B = AE')

Flags Relacionais para Números com Sinal

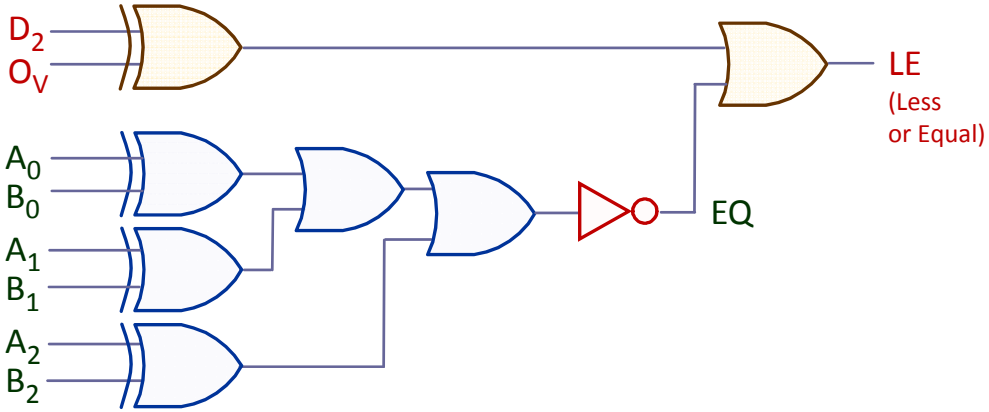


Flag GE (Greater or Equal), implementação para operandos a 3 bits no domínio $\mathbb{Z}$.

A Flag Greater or Equal (GE) não pode depender exclusivamente do bit de sinal do resultado, porque a operação de subtracção que está a ser realizada pode exceder a representação.

Torna-se pois necessário tomar em consideração o Overflow.

Idem para a Flag Less (L).



Flag LE (Less or Equal), uma implementação possível para operandos a 3 bits no domínio $\mathbb{Z}$.

$A \geq B \Rightarrow L=0 \quad GE=1$

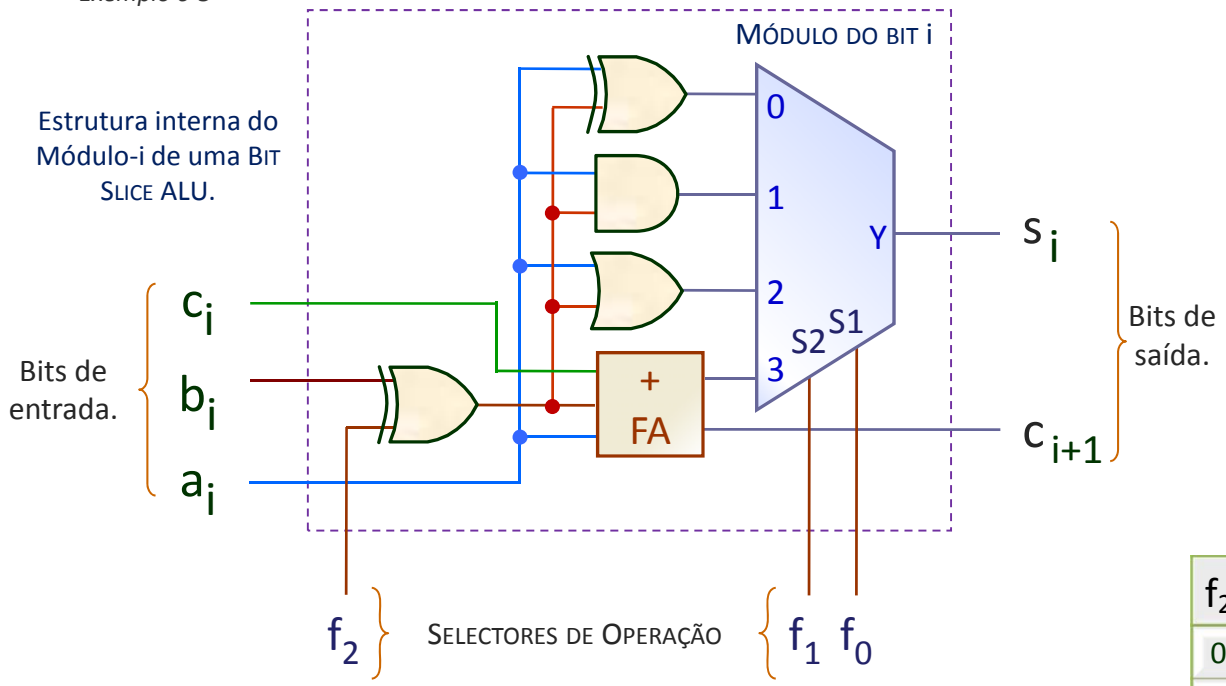
$A < B \Rightarrow L=1 \quad GE=0$

GE: Greater or Equal

L: Less (L = GE')

ANDAR BIT SLICE DE UMA ALU (EX. 6-3)

Exemplo 6-3



Uma ALU (ARITHMETIC AND LOGIC UNIT) de n-bits é obtida a partir de n-circuitos idênticos de 1-bit (BIT SLICES).

O circuito ao lado é constituído por um FULL-ADDER (+), portas lógicas simples e um Multiplexer. Representa o módulo **i** de um circuito iterativo de n bits constituído por n módulos idênticos. A ALU executa as operações básicas da aritmética e lógica representadas na tabela.

f_2	f_1	f_0	S_i	C_{i+1}	OPERAÇÃO
0	0	0	$a_i \oplus b_i$	$M(a_i,b_i,c_i)$	XOR
0	0	1	$a_i \cdot b_i$		AND
0	1	0	$a_i + b_i$		OR
0	1	1	$a_i \oplus b_i \oplus c_i$		ADC
1	0	0	$a_i \oplus b'_i$	$M(a_i,b'_i,c_i)$	XOR'
1	0	1	$a_i \cdot b'_i$		–
1	1	0	$a_i + b'_i$		–
1	1	1	$a_i \oplus b_i' \oplus c_i$		SBB (*)

Tabela funcional representativa das operações executadas pela ALU.

Há duas optimizações que normalmente se usam face ao desenho em cima:

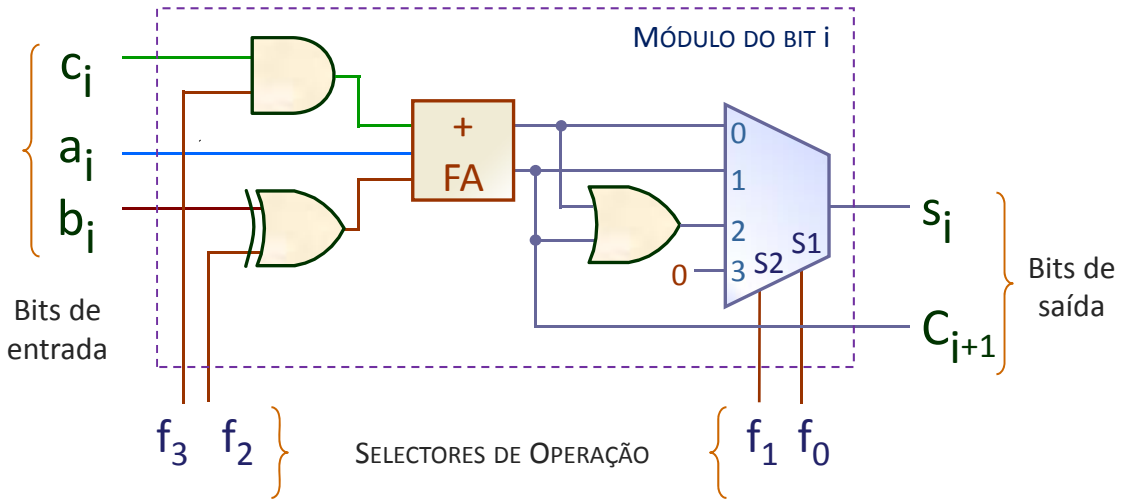
- uma relativa à propagação do CARRY (adopção de uma arquitectura CARRY LOOK AHEAD como se viu no capítulo anterior) ;
- outra que tira partido dos XORs intrínsecos ao FULL-ADDER e permite eliminar o XOR externo representado.

(*) esta operação requer a negação do CARRY-IN do primeiro andar e do CARRY-OUT do último.

M indica a função de Maioria.

ANDAR BIT SLICE DE UMA ALU (EX. 6-4)

Exemplo 6-4



Estrutura interna do Módulo-i de uma BIT SLICE ALU.

f_3	f_2	f_1	f_0	S_i	C_{i+1}	OPERAÇÃO
0	0	0	0	$a_i \oplus b_i$	$a_i \cdot b_i$	XOR
0	0	0	1	$a_i \cdot b_i$		AND
0	0	1	0	$a_i + b_i$		OR
0	0	1	1	0		0
0	1	0	0	$a_i \oplus b'_i$	$a_i \cdot b'_i$	XOR'
0	1	0	1	$a_i \cdot b'_i$		–
0	1	1	0	$a_i + b'_i$		–
0	1	1	1	0		0
1	0	0	0	$a_i \oplus b_i \oplus c_i$	$M(a_i,b_i,c_i)$	ADC
1	0	0	1	c_i		–
1	0	1	0	a + b + c		–
1	0	1	1	0		–
1	1	0	0	$ai \oplus b_i' \oplus c_i$	$M(a_i,b_i',c_i)$	SBB (*)
1	1	0	1	c_i		–
1	1	1	0	a + b'+ c		–
1	1	1	1	0		–

Tabela funcional representativa das operações executadas pela ALU – M indica a função de Maioria e a operação assinalada com * requer a negação do CARRY-IN do primeiro andar e do CARRY-OUT do último.

ALU a 3 Bits – Diagrama Genérico e Tabela de Operações

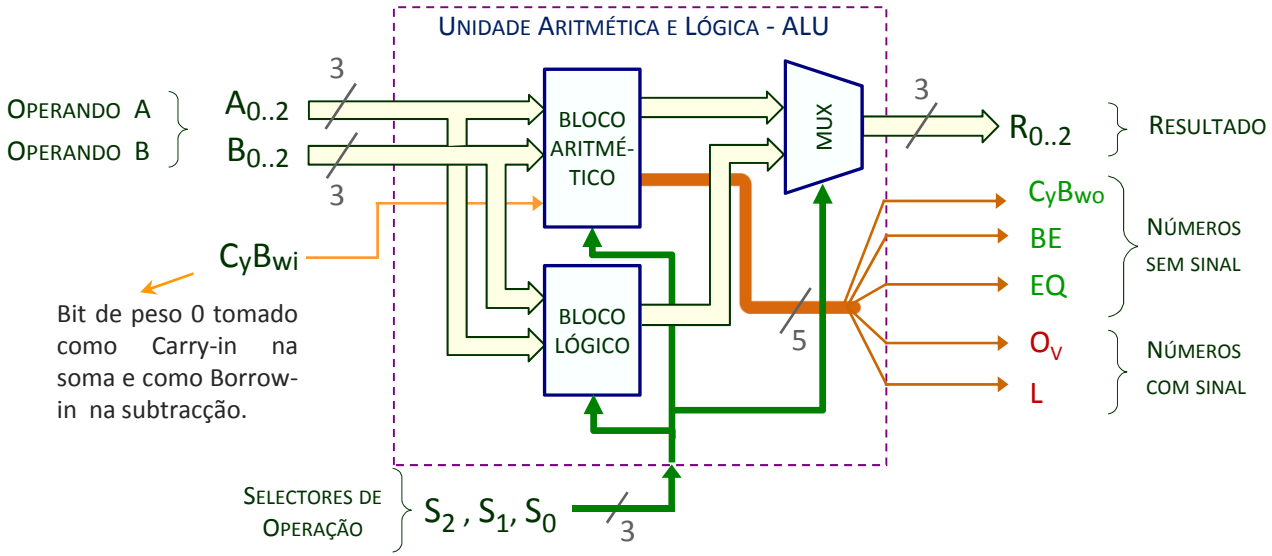


Diagrama genérico da ALU a 3 bits evidenciando os blocos Aritmético e Lógico .

Tabela da ALU			Operações			Flags (Indicadores)				
						de Excesso		Relacionais		
S_2	S_1	S_0	Operação e Sigla			C_yB_{wo}	Ov	BE	EQ	L
0	0	0	ADD	A + B	Adição	Cy	Ov	–	–	–
0	0	1	SUB	A – B	Subtracção	Bw	Ov	BE	EQ	L
0	1	0	ADC	A + B + C_yB_{wi}	Adição com Carry	Cy	Ov	–	–	–
0	1	1	SBB	A – B – C_yB_{wi}	Subtracção com Borrow	Bw	Ov	BE	EQ	L
1	0	0	DBL	A + A + C_yB_{wi}	Dobro de A com Carry	Cy	Ov	–	–	–
1	0	1								
1	1	0	ORL	A OR B	OR lógico	–	–	–	–	–
1	1	1	XRL	A XOR B	XOR lógico	–	–	–	–	–

Tabela detalhada das operações da ALU e da correspondente activação das Flags.

Pretende-se realizar com uma PAL 22V10 uma ALU que execute as operações descritas na tabela em baixo.

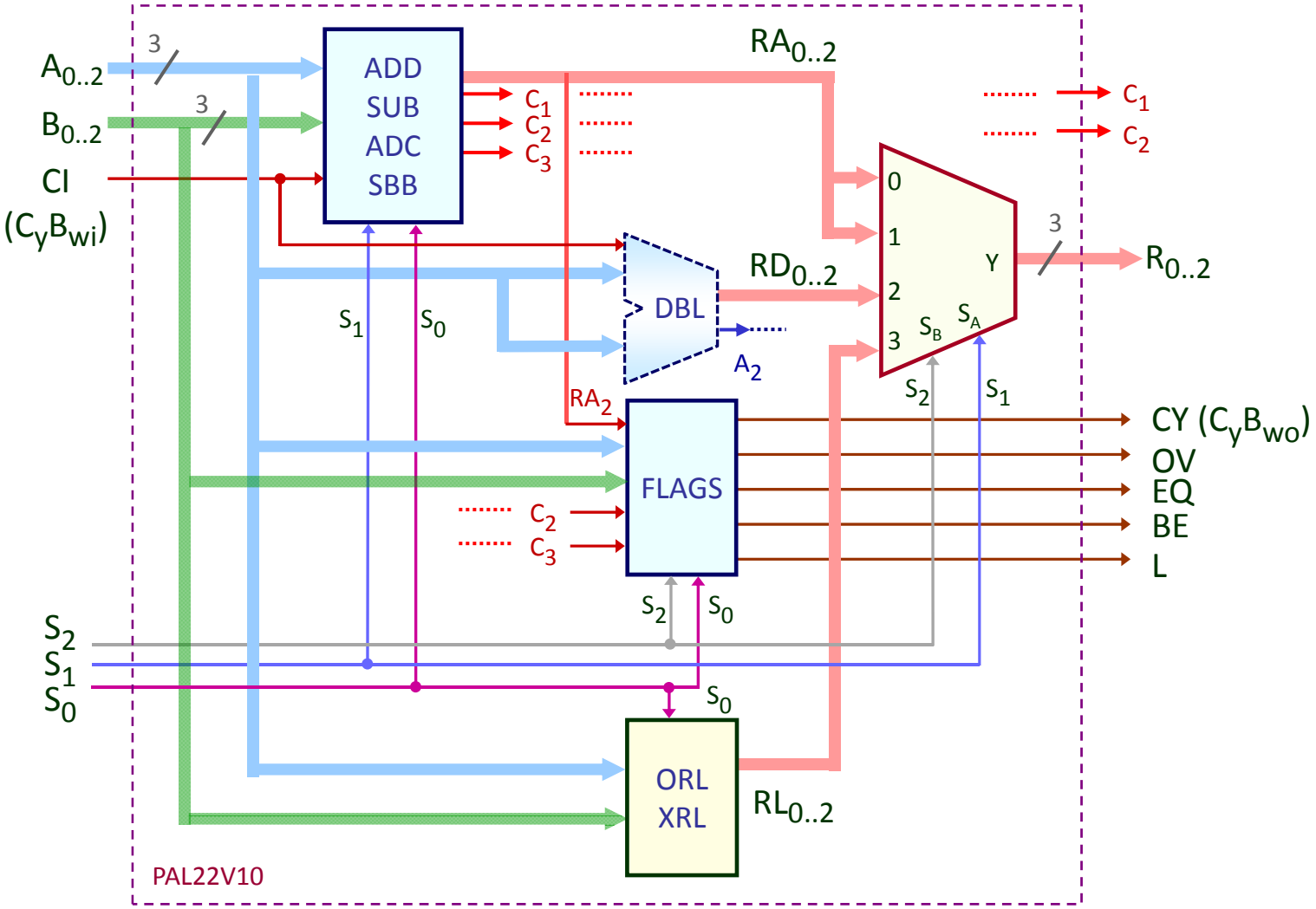
A ALU realiza as 7 operações pelo que é necessário que existam 3 bits de selecção $S_{0..2}$ para codificar a operação pretendida.

A estrutura interna divide-se em dois grandes blocos: Aritmético e Lógico. A saída da ALU é escolhida através de um multiplexer cuja entrada de selecção é controlada por um ou mais bits selectores da operação da ALU.

O bloco Aritmético tem como elemento principal um somador de números de 3 bits que realiza as operações de soma e subtracção.

Para além do resultado R a 3 bits a ALU põe disponíveis os indicadores (Flags) de excesso (erro) e relacionais indicados.

ALU A 3 BITS – ARQUITECTURA INTERNA

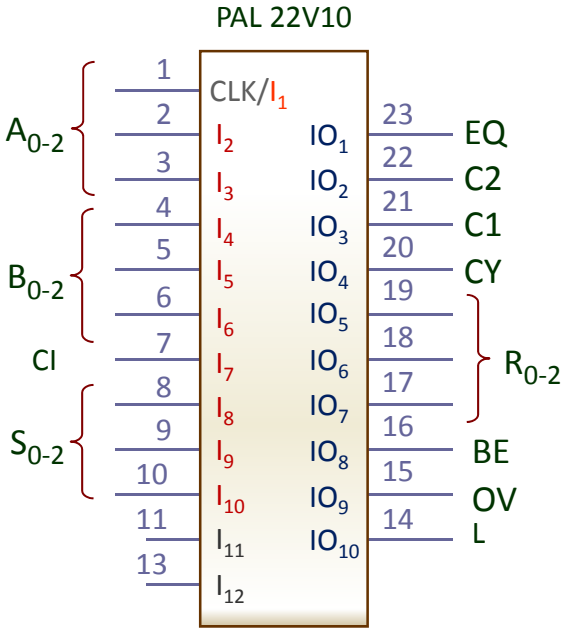


Arquitectura interna da ALU a 3 bits evidenciando os blocos principais.

S_2	S_1	S_0	R
0	0	0	ADD
0	0	1	SUB
0	1	0	ADC
0	1	1	SBB
1	0	0	DBL
1	0	1	
1	1	0	ORL
1	1	1	XRL

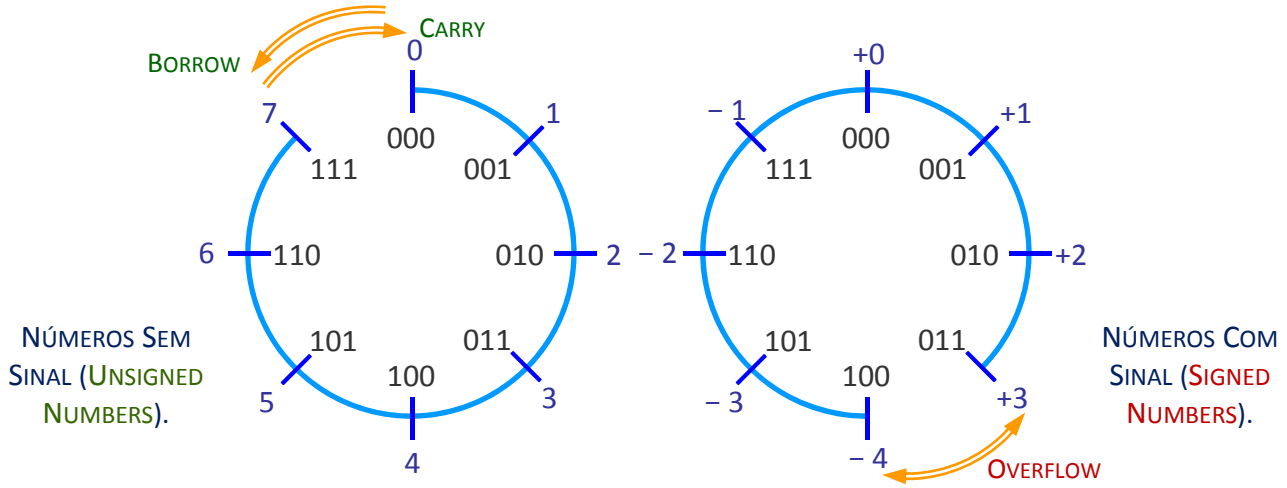
Tabela de Operações da ALU evidenciando a configuração de código escolhida para cada uma.

ALU a 3 Bits – Implementação em PAL



Símbolo lógico da PAL assinalando os 10 pinos de entrada e os 10 pinos de saída da ALU a 3 bits a implementar.

VCC (pino 24) e GND (pino 12) não estão representados. Os pinos 11 e 13 não são utilizados.

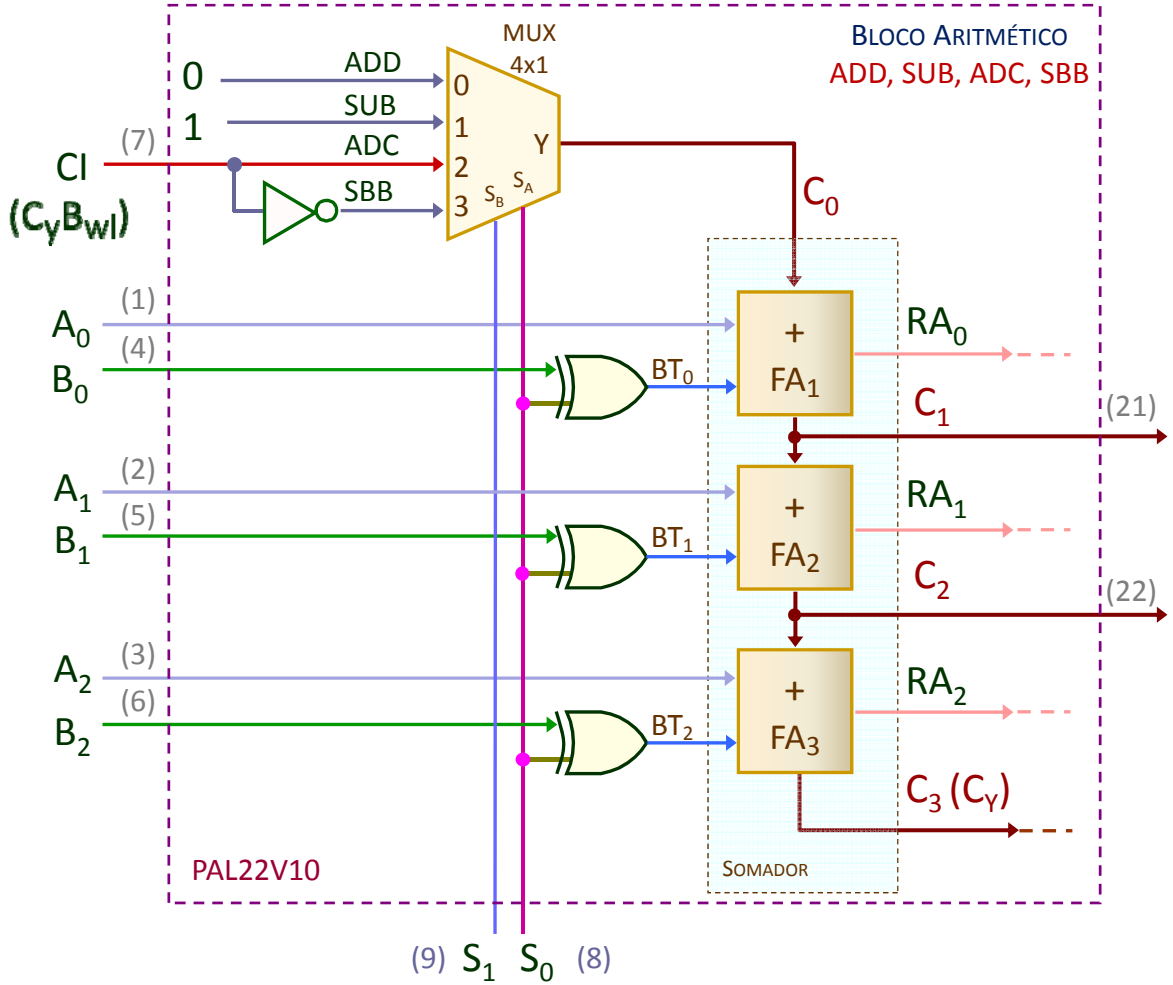


Representação gráfica para o sistema de numeração a 3-bits a utilizar na ALU.

```
/***************** INPUT PINS ************************/
PIN [1..3]   =  [A0..2] ; /* Operando A               */
PIN [4..6]   =  [B0..2] ; /* Operando B               */
PIN  7       =   CI     ; /* Carry-in/Borrow-in       */
PIN [8..10]  =  [S0..2] ; /* Selectores de operação   */

/***************** OUTPUT PINS ***********************/
PIN  14      =  L       ; /* Flag Less                */
PIN  15      =  OV      ; /* Flag Overflow            */
PIN  16      =  BE      ; /* Flag Below or Equal      */
PIN  20      =  CY      ; /* Carry-out/Borrow-out     */
PIN [17..19] =  [R2..0] ; /* Resultado final          */
PIN [21..22] =  [C1..2] ; /* Carrys parciais          */
PIN  23      =  EQ      ; /* Flag Equal(caso de A=B)  */
```

Troço do código CUPL com a atribuição dos pinos de entrada e saída.

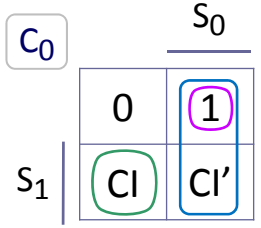
Estrutura Funcional correspondente às Operações ADD, SUB, ADC e SBB



Estrutura interna de uma parte do Bloco Aritmético evidenciando o somador-subtractor iterativo de 3 bits da PAL.

S_2	S_1	S_0	R	Operação
0	0	0	ADD	A + B
0	0	1	SUB	A – B
0	1	0	ADC	A + B + CyBwi
0	1	1	SBB	A – B – CyBwi
1	0	0	DBL	
1	0	1		
1	1	0	ORL	
1	1	1	XRL	

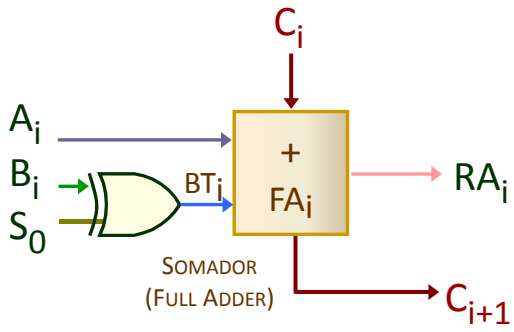
Tabela de Operações da ALU realçando as realizadas pelo bloco aritmético ao lado.



$$C_0 = S_0 S_1' + S_0 CI' + S_0' S_1 CI$$

M-K e equação do sinal C_O de Carry-in do somador-subtractor iterativo da PAL.

Estrutura Funcional correspondente às Operações ADD, SUB, ADC e SBB



Full-adder (FA) e lógica associada do módulo i do somador-subtractor iterativo da PAL.

$$BT_i = B_i \oplus S_0$$

$$RA_i = A_i \oplus BT_i \oplus C_i$$

$$C_{i+1} = A_i\, BT_i + C_i\,(A_i \oplus BT_i)$$

Equações lógicas dos bits intervenientes em cada módulo **i** do somador iterativo do bloco aritmético.

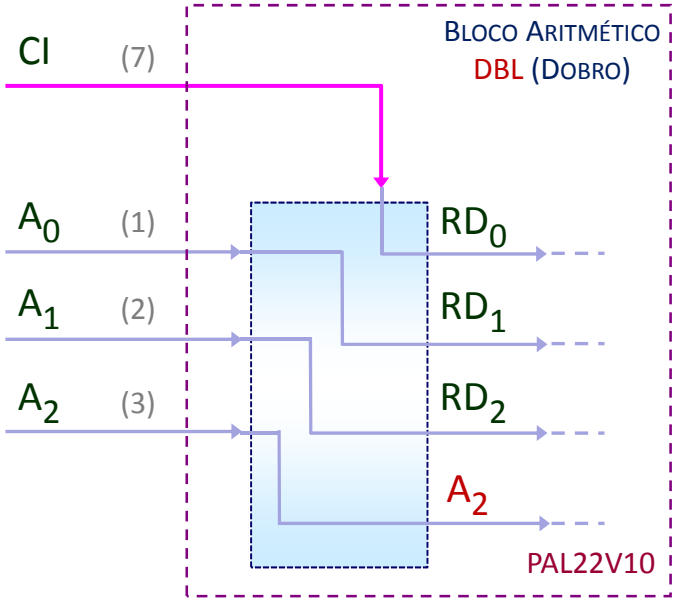
```
/* Expressão de C0: Saída do Mux 4x1 */
C0 = CI & !S0 & S1 # !CI & S0 # S0 & !S1
/* Expressão de BTi: Operando B 'invertido'  */
$REPEAT i = [ 0..2 ]
BT{i} = B{i} $ S0;
$REPEND
/* Expressões de Ci (Carry-out parcial) e RAi
resultado parcial da Adição/Subtracção) */
$REPEAT i = [ 0..2 ]
RA{i}  = A{i} $ BT{i} $ C{i}
C{i+1} = A{i} & BT{i} # C{i} & (A{i} $ BT{i})
$REPEND
```

- A_i – Bit i do Operando A
- B_i – Bit i do Operando B
- C_i – Carry-in do módulo i do Full-adder
- C_{i+1} – Carry-out do módulo i do Full-adder
- Bt_i – Bit i do Operando B invertido ('transformado')
- RA_i – Bit i do Resultado Aritmético

Troço do código CUPL correspondente às diversas operações de Soma e Subtracção.

Estrutura Funcional correspondente à Operação DBL

$$DBL \Rightarrow A + A + CI$$



Funcionalidade associada à operação Dobro denotando a inutilidade de quaisquer circuitos adicionais.

```
/*Expressão de RDi (Dobro de A mais Carry-in)*/
RD0 = CI
RD1 = A0
RD2 = A1
```

Troço do código CUPL correspondentes à operação Dobro (DBL).

	A_1	A_0	C_I
	A_2	A_1	A_0
+	A_2	A_1	A_0
A_2	A_1	A_0	C_I
↓	↓	↓	↓
CY	RD_2	RD_1	RD_0

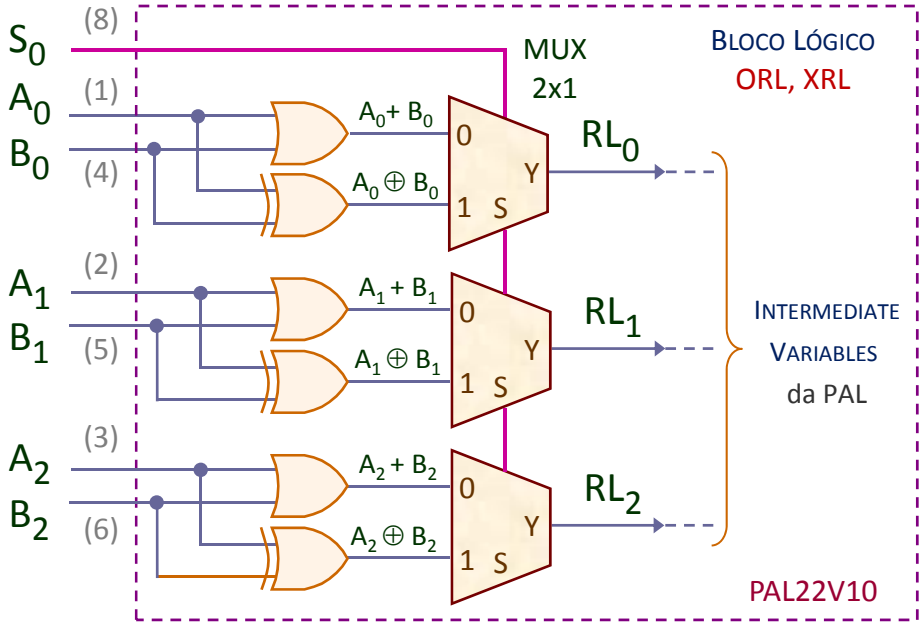
Carry-out da operação Dobro — Resultado da operação Dobro

Soma de A+A evidenciando o deslocamento para a esquerda dos bits do operando A para a obtenção dos bits do **R**esultado da operação **D**obro RD, sem necessidade de circuitos adicionais.

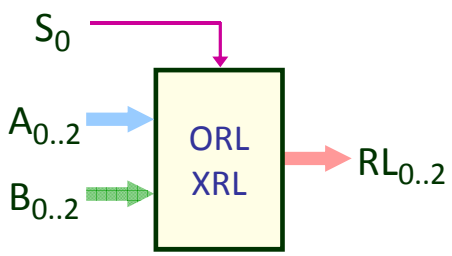
$$RD_0 = CI$$
$$RD_1 = A_0$$
$$RD_2 = A_1$$

Equações dos bits do resultado da operação Dobro.

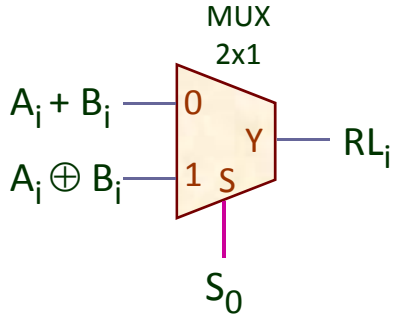
Estrutura Funcional correspondente às Operações ORL e XRL

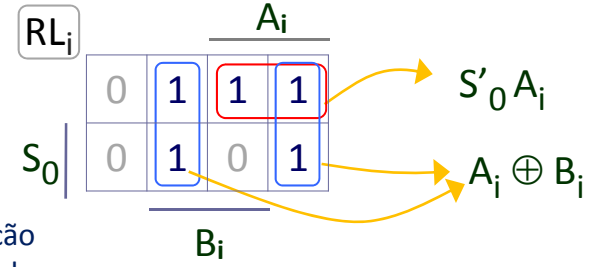


Estrutura interna do bloco lógico da ALU.



Bloco lógico da ALU.





Circuito, m-K e equação de cada bit RL_i (resultado lógico para o bit i).

$$RL_i = S_0' A_i + (A_i \oplus B_i)$$

S_2	S_1	S_0	R	Operação
0	0	0	ADD	A + B
0	0	1	SUB	A – B
0	1	0	ADC	A + B + CyBwi
0	1	1	SBB	A – B – CyBwi
1	0	0	DBL	A + A + Cin
1	0	1		
1	1	0	ORL	A OR B
1	1	1	XRL	A XOR B

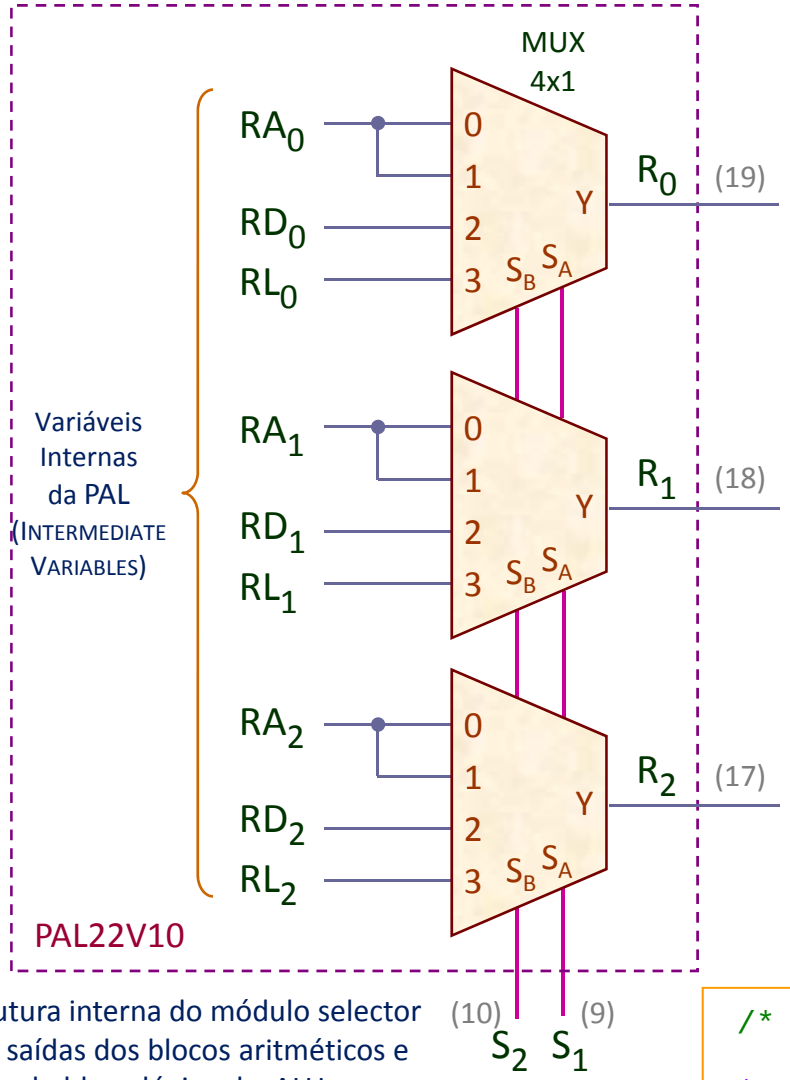
Tabela de Operações da ALU realçando as realizadas pelo bloco lógico em cima.

As Intermediate Variables são variáveis internas que definem equações mas que não são enviadas para um pino de saída.

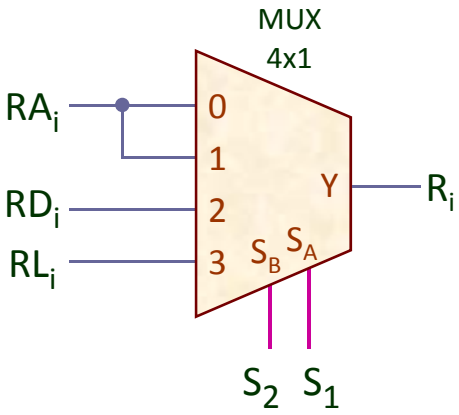
```
/* Expressão de RLi: resultado da Operação Lógica */

$REPEAT i = [ 0..2 ]
RL{i} = (A{i} $ B{i}) # !S0 & A{i};
$REPEND
```

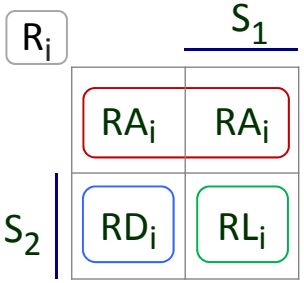
Troço do código CUPL correspondentes às operações ORL e XRL.

MULTIPLEXER DE SAÍDA



Estrutura interna do módulo selector das saídas dos blocos aritméticos e do bloco lógico da ALU .





	S_2	S_1	S_0	R
0	0	0	0	ADD
	0	0	1	SUB
1	0	1	0	ADC
	0	1	1	SBB
2	1	0	0	DBL
	1	0	1	
3	1	1	0	ORL
	1	1	1	XRL

Tabela de Operações da ALU.

RA_i – resultado das operações de Adição e Subtracção

RD_i – resultado da operação Dobro

RL_i – resultado das operações Lógicas

$$R_i = S_2' RA_i + S_2 (RD_i S_1' + RL_i S_1)$$

Circuito lógico, m-K e equação de cada bit R_i de saída.

Troço do código CUPL correspondente a uma saída R_i.

```
/* Expressão de Ri: saída do MUX final da ALU */

$REPEAT i = [ 0..2 ]
R{i} = ( RA{i} & !S2 ) # (RD{i} & !S1 # RL{i} & S1) & S2
$REPEND
```

GERAÇÃO DAS FLAGS NA ALU

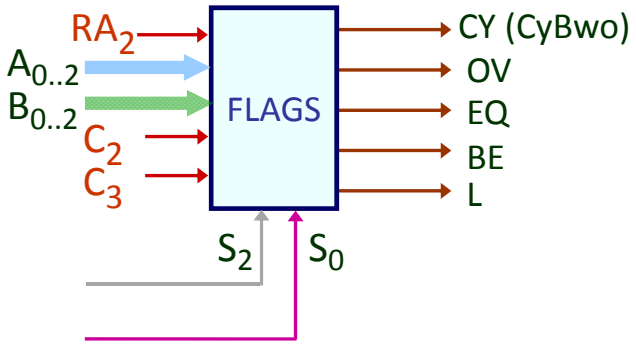
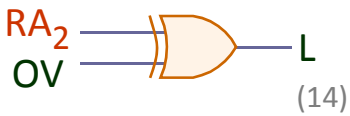


Diagrama genérico do bloco gerador das FLAGS.

	S_2	S_1	S_0	R
0	0	0	0	ADD
	0	0	1	SUB
1	0	1	0	ADC
	0	1	1	SBB
2	1	0	0	DBL
	1	0	1	
3	1	1	0	ORL
	1	1	1	XRL

Tabela de Operações da ALU realçando os selectores intervenientes na geração das FLAGS.



$$L = RA_2 \oplus OV$$

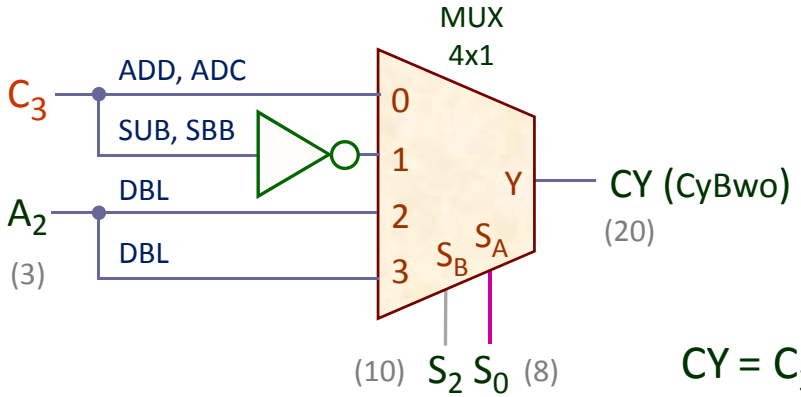
Equação e circuito gerador da FLAG L (LESS) da ALU.

A lógica geradora da FLAG L (LESS) faz uso da FLAG OVERFLOW (OV), gerada separadamente.

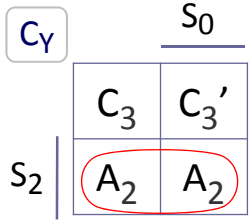
O OVERFLOW corresponde à saída colocada no pino 15 da PAL.

Como na PAL é possível realimentar o valor das funções de saída, o OVERFLOW é realimentado e colocado na coluna 34 da matriz da PAL.

A FLAG LESS é gerada a partir do OVERFLOW e do termo RA_2, e colocada na saída a que corresponde o pino 14, como patente nos slides mais à frente.



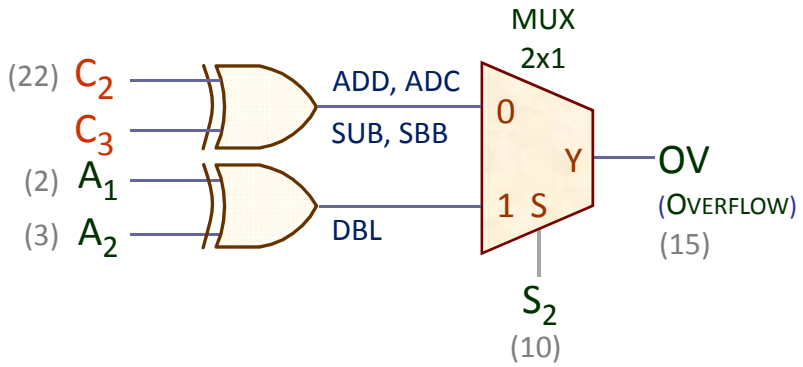
Circuito gerador da FLAG CARRY-OUT/BORROW-OUT (designada CY para simplificação).



$$CY = C_3 S_2' S_0' + C_3' S_2' S_0 + A_2 S_2$$

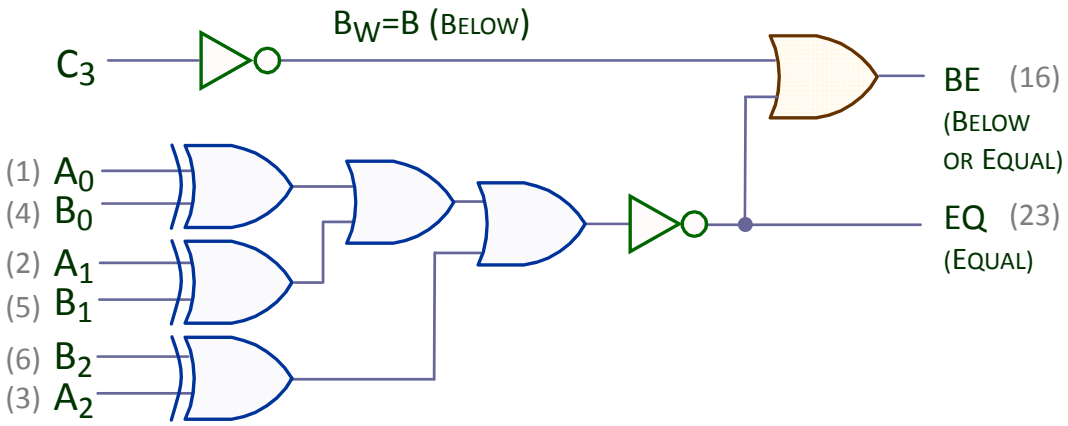
Mapa de Karnaugh e equação da FLAG CY (CARRY-OUT/BORROW-OUT) da ALU.

Expressões Algébricas e Código CUPL das Flags Overflow, Less e Below or Equal



$$OV = S_2' (C_2 \oplus C_3) + S_2 (A_1 \oplus A_2)$$

Circuito e equação da Flag OV (Overflow) da ALU.



$$EQ = [(A_0 \oplus B_0) + (A_1 \oplus B_1) + (A_2 \oplus B_2)]'$$

$$BE = C_3' + EQ$$

Circuito e equações algébricas das Flags EQ (Equal) e BE (Below or Equal) da ALU.

```
/* Expressões das Flags da ALU */
CY = C3 & !S2 & !S0 # !C3 & !S2 & S0 # A2 & S2
EQ = !((A0 $ B0) # (A1 $ B1) # (A2 $ B2))          ;
BE = !C3 # EQ                                      ;
OV = (C3 $ C2) & !S2 # (A2 $ A1) & S2              ;
L  = (R2 $ OV)                                     ;
```

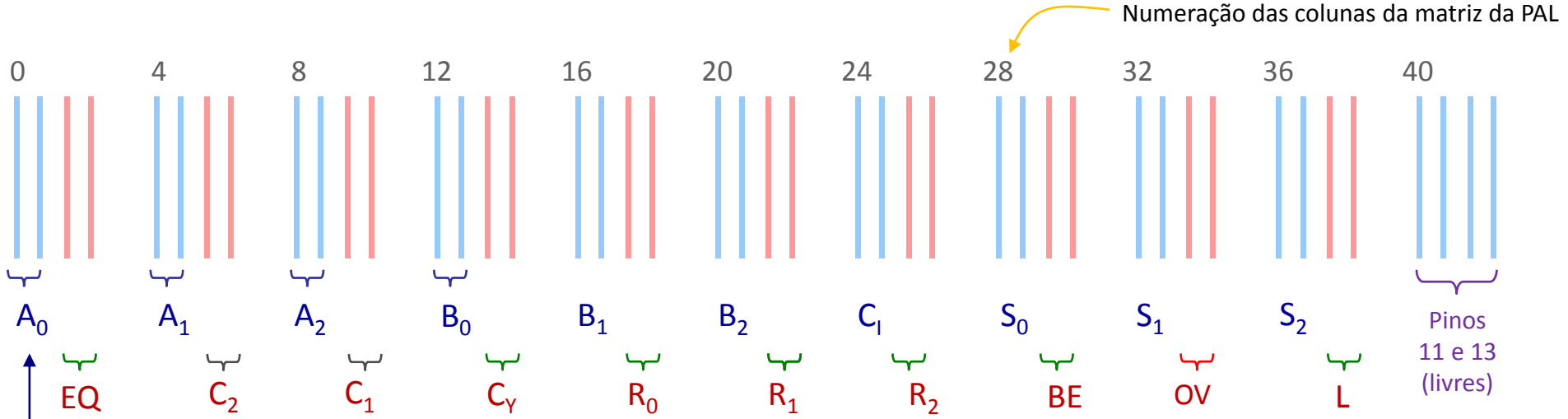
Troço do código CUPL correspondente à geração das Flags.

CÓDIGO CUPL DA ALU

```
...
Company  CCISEL ;
Assembly None ;
Location  ;
Device   p22v10 ;

/* *************** INPUT PINS ************ */
PIN [1..3] = [A0..2]     ; /* Operando A            */
PIN [4..6] = [B0..2]     ; /* Operando B            */
PIN 7 = CI               ; /* Carry-in/Borrow-in    */
PIN [8..10] = [S0..2]    ; /* Selectores de operação*/

/* *************** OUTPUT PINS *********** */
PIN  14 = L              ; /* Flag Less             */
PIN  15 = OV             ; /* Flag Overflow         */
PIN  16 = BE             ; /* Flag Below or Equal   */
PIN  20 = CY             ; /* Carry-out/Borrow-out  */
PIN  [17..19] = [R2..0]  ; /* Resultado final       */
PIN  [21..22] = [C1..2]  ; /* Carrys parciais       */
PIN  23 = EQ             ; /* Flag Equal            */

/* *************** BODY ***************** */
/* Expressão de C0: Carry-out do Mux 4x1 de entrada */

C0 = CI & !S0 & S1 # !CI & S0 # S0 & !S1;

/* Expressão de BTi: Operando B 'invertido'  */

$REPEAT i=[0..2]
BT{i} = B{i} $ S0;
$REPEND
```

```
/* Expressões de Ci (Carry-out parcial) e RAi
(resultado da Adição/Subtracção) */

$REPEAT i=[0..2]
RA{i} = A{i} $ BT{i} $ C{i};
C{i+1}  = A{i} & BT{i} # C{i} & (A{i} $ BT{i});
$REPEND

/* Expressão de RDi (Dobro de A mais Carry) */

RD0 = CI;
RD1 = A0;
RD2 = A1;

/*Expressão de RLi: Resultado das Operações Lógicas */

$REPEAT i=[0..2]
RL{i} = (A{i} $ B{i}) # !S0 & A{i};
$REPEND

/* Expressão de Ri: Saída do MUX selector da ALU */

$REPEAT i=[0..2]
R{i} = (RA{i} & !S2) # (RD{i}& !S1 # RL{i}& S1) & S2;
$REPEND

/* Expressão das Flags da ALU */

EQ = !((A0 $ B0) # (A1 $ B1) # (A2 $ B2));
OV = (C3 $ C2) & !S2 # (A2 $ A1) & S2;
CY = C3 & !S2 & !S0 # !C3 & !S2 & S0 # A2 & S2;
BE = (!C3 # EQ);
L  = (R2 $ OV)& !EQ;
```

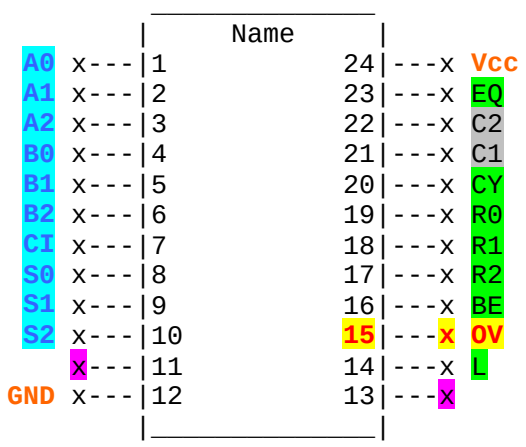
Configuração da Matriz Programável da PAL



Esquema representativo da distribuição dos sinais de entrada e de saída da PAL pelas 44 colunas da matriz 44x132 da PAL.

Entradas externas (linhas a azul).

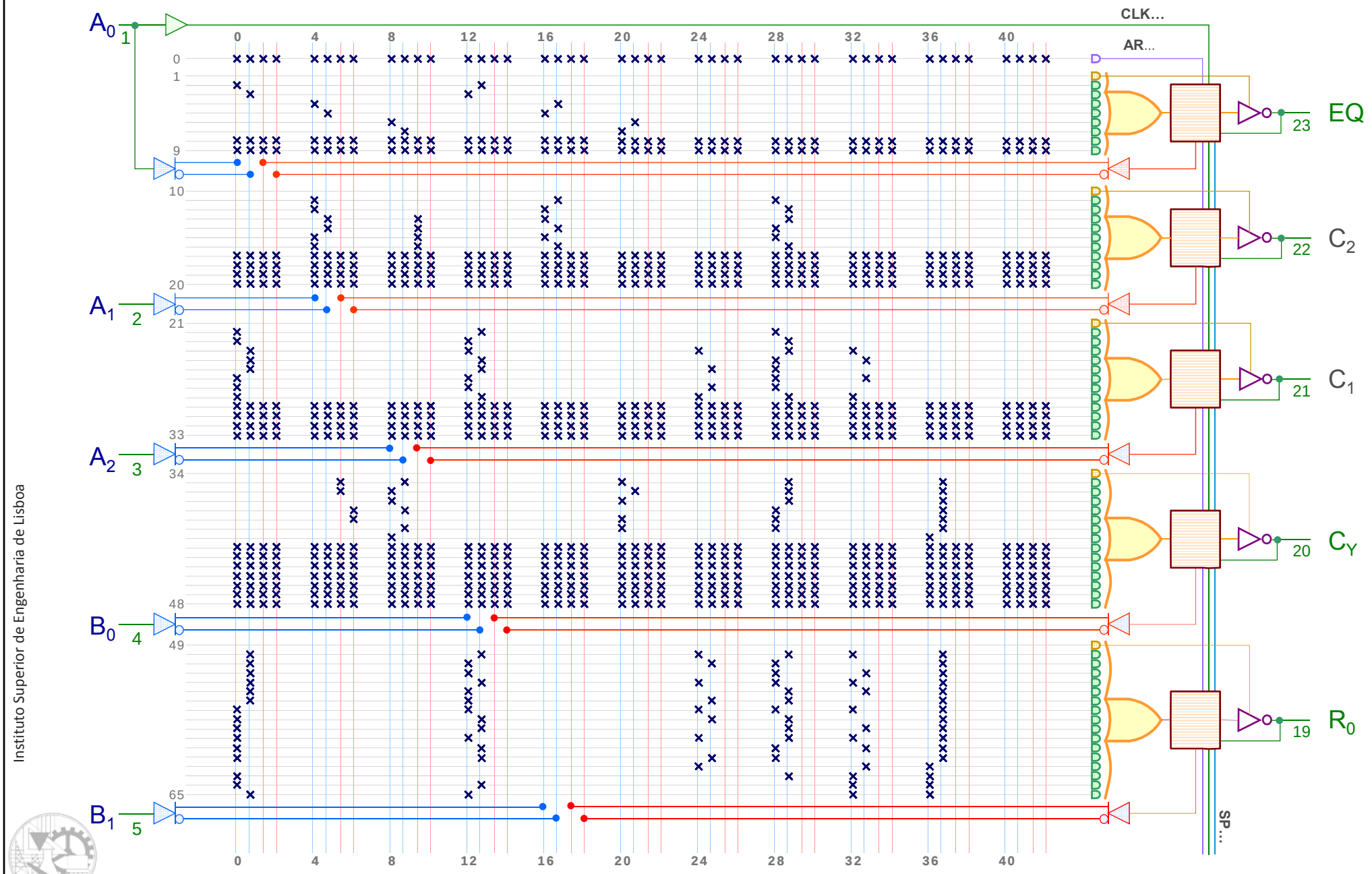
Saídas (linhas a vermelho) realimentadas através de buffers.



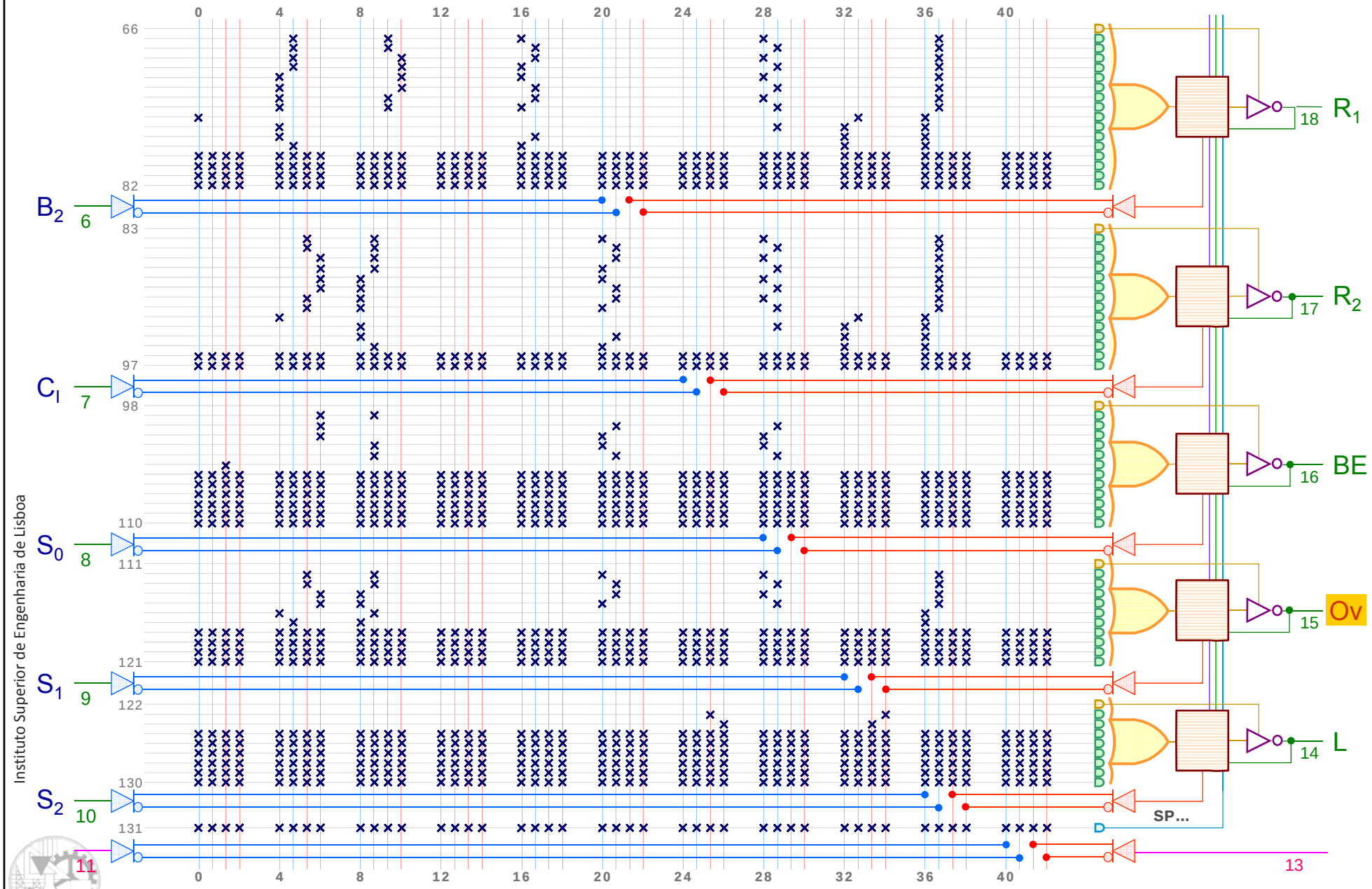
Chip Diagram (tal como gerado pelo ficheiro de extensão .DOC).

As linhas verticais em cima assinalam as 44 colunas da 'matriz' programável da PAL. Os sinais provêm dos doze pinos de entrada (complementados e não complementados, linhas a azul), e dos dez pinos bidireccionais usados como saídas (complementados e não complementados, linhas a vermelho). Estas saídas são reintroduzidas na matriz de entrada através de buffers como se viu nos esquemas de arquitectura da PAL.

FUSE PLOT DA ALU NA PAL – PARTE 1



FUSE PLOT DA ALU NA PAL – PARTE 2



Instituto Superior de Engenharia de Lisboa

TABELA DE SÍMBOLOS DO FICHEIRO .DOC

```
=============================================================
                        Symbol Table
=============================================================
Pin   Variable                                  Pterms   Max     Min
Pol   Name          Ext          Pin    Type    Used     Pterms  Level
---   ------        - --         ---    ----    ------   ------  -----

      A0                         1      V       -        -       -
      A1                         2      V       -        -       -
      A2                         3      V       -        -       -
      B0                         4      V       -        -       -
      B1                         5      V       -        -       -
      B2                         6      V       -        -       -
      BE                         16     V       6        12      1
      BT0                        0      I       2        -       -
      BT1                        0      I       2        -       -
      BT2                        0      I       2        -       -
      C0                         0      I       3        -       -
      C1                         21     V       8        12      1
      C2                         22     V       6        10      1
      C3                         0      I       6        -       -
      CI                         7      V       -        -       -
      CY                         20     V       7        14      1
      EQ                         23     V       6        8       1
      L                          14     V       2        8       1
      OV                         15     V       6        10      1
      R0                         19     V       10       16      1
      R1                         18     V       12       16      1
      R2                         17     V       12       14      1
      RA0                        0      I       6        -       -
      RA1                        0      I       8        -       -
      RA2                        0      I       8        -       -
      RD0                        0      I       1        -       -
      RD1                        0      I       1        -       -
      RD2                        0      I       1        -       -
      RL0                        0      I       3        -       -
      RL1                        0      I       3        -       -
      RL2                        0      I       3        -       -
      S0                         8      V       -        -       -
      S1                         9      V       -        -       -
      S2                         10     V       -        -       -
      BE           oe            16     D       1    1   0
      C1           oe            21     D       1    1   0
      C2           oe            22     D       1    1   0
      CY           oe            20     D       1    1   0
      EQ           oe            23     D       1    1   0
      L            oe            14     D       1    1   0
      OV           oe            15     D       1    1   0
      R0           oe            19     D       1    1   0
      R1           oe            18     D       1    1   0
      R2           oe            17     D       1    1   0
```

```
LEGEND

D : default variable
I : intermediate variable
U : undefined
T : function

F : field
N : node
V : variable

G : group
M : extended node
X : extended variable
```

O ficheiro **.doc** é sempre gerado, mesmo em caso de erro de compilação.

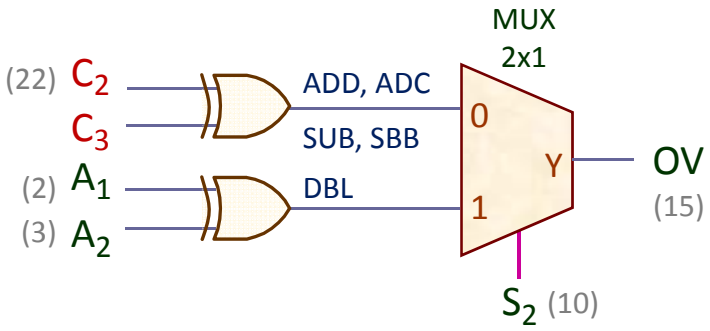
Contém todas as equações lógicas geradas pelo compilador.
Contém também a tabela de símbolos (ao lado), que revela a acomodação da lógica na PAL:

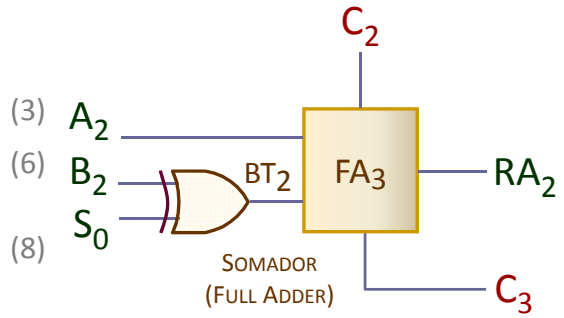
- a azul estão indicadas entradas.
- a vermelho estão indicadas saídas.
- a verde estão indicadas variáveis intermédias.

Como os 10 pinos I/O bidireccionais são utilizados como saídas, é necessário activar as portas tri-state correspondentes. Isto está patente nas últimas 10 linhas da tabela de símbolos com a indicação explícita dos sinais OE (OUTPUT ENABLE) de controlo das portas tri-state de saída . Para isto acontecer, é necessário não ligar nenhuma entrada às portas AND de controlo respectivas.

Tabela de símbolos da ALU gerada pelo ficheiro '.doc'.

GERAÇÃO DE OVERFLOW NA PAL





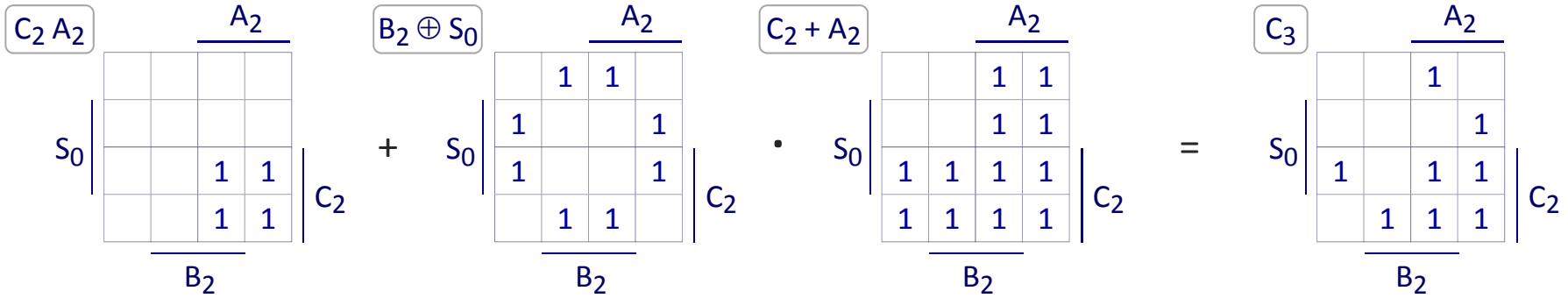
$$BT_2 = B_2 \oplus S_0$$

$$OV = (C_3 \oplus C_2) \cdot S'_2 + (A_2 \oplus A_1) \cdot S_2$$

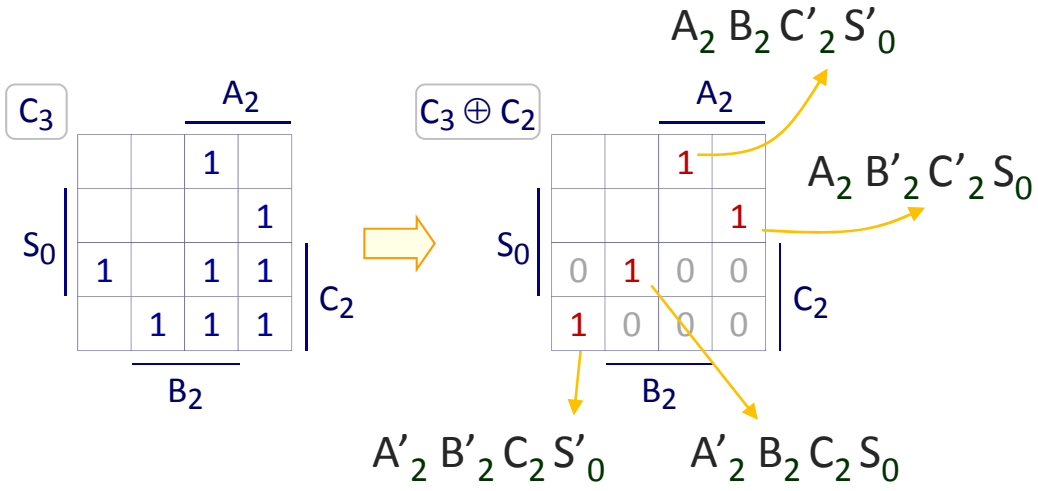
Circuito e equação da FLAG OV (OVERFLOW) da ALU.

$$C_3 = A_2\,BT_2 + C_2 \cdot A_2 + C_2 \cdot BT_2 =$$
$$= C_2 \cdot A_2 + BT_2\,(C_2 + A_2) =$$
$$= C_2 \cdot A_2 + (B_2 \oplus S_0)\,(C_2 + A_2)$$

Equação da variável intermédia C3 em função de C2 e das entradas S0, A2 e B2 por substituição de BT2.



Mapas de Karnaugh auxiliares utilizados na geração da equação da saída correspondente à FLAG OVERFLOW tal como documentada no ficheiro .DOC.



Expansão do termo C 2⊕ C3 em função de C2 e das entradas S0, A2 e B2.

$$OV = (C_3 \oplus C_2) \cdot S'_2 + (A_2 \oplus A_1) \cdot S_2$$

```
OV =>
    !A2 &  B2 &  C2 &  S0 & !S2
 #  !A2 & !B2 &  C2 & !S0 & !S2
 #   A2 & !B2 & !C2 &  S0 & !S2
 #   A2 &  B2 & !C2 & !S0 & !S2
 #   A1 & !A2 &  S2
 #  !A1 &  A2 &  S2
```

Equação da FLAG OVERFLOW (mais acima) e na forma expandida de soma de termos produtos documentada no ficheiro de extensão .DOC (imediatamente acima).

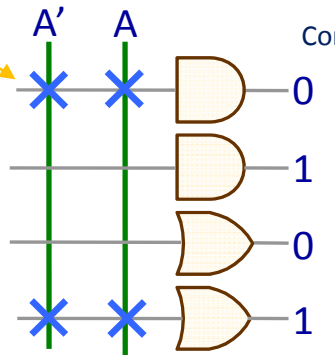
```
Pin #15  05824 Mode  --
 04884 ------------------------------------------
 04928 ------x--x----------x-------x--------x------ !A2 &  B2 &  C2 &  S0 & !S2
 04972 ------x--x-----------x-------x-------x------ !A2 & !B2 &  C2 & !S0 & !S2
 05016 -------xx------------x-------x--------x------  A2 & !B2 & !C2 &  S0 & !S2
 05060 -------xx-----------x--------x-------x------  A2 &  B2 & !C2 & !S0 & !S2
 05104 ----x----x-------------------------x-------  A1 & !A2 &  S2
 05148 -----x--x-------------------------x------- !A1 &  A2 &  S2
 05192 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 05236 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 05280 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 05324 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

LEGEND    X : fuse not blown
          - : fuse blown
```

Representação dos 6 termos produtos dos 10 disponíveis utilizados na geração da FLAG OVERFLOW na saída 10 da PAL22V10 tal como documentado no ficheiro de extensão .DOC gerado pelo compilador.

No ficheiro de extensão .DOC, o símbolo X significa que é feita uma ligação. O símbolo '–' significa que é eliminada uma ligação. No ficheiro de extensão .JED gerado pelo compilador para a programação da PAL, o mapa de fusíveis correspondente representa o símbolo X (do ficheiro .DOC) por um 0, e o símbolo '–' por um 1.

Análise do Fuse-plot do Overflow a partir do Ficheiro .doc

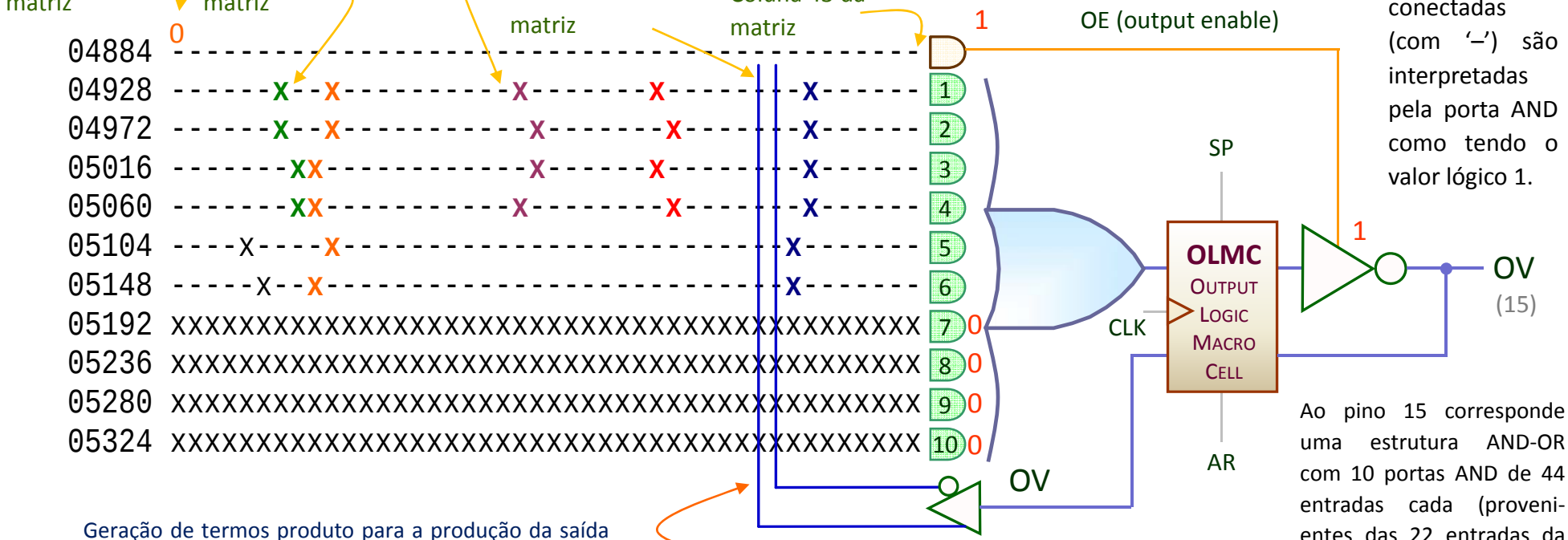
Uma cruz na intersecção indica que o sinal dessa coluna entra na porta AND da linha respectiva.



Convenções da notação PAL.

Todas as portas TRI-STATE associadas às saídas das funções têm de estar activas. Para tal, torna-se necessário não ligar nenhuma entrada às portas AND de controlo respectivas o que é indicado pelo símbolo '–' .

As entradas não conectadas (com '–') são interpretadas pela porta AND como tendo o valor lógico 1.

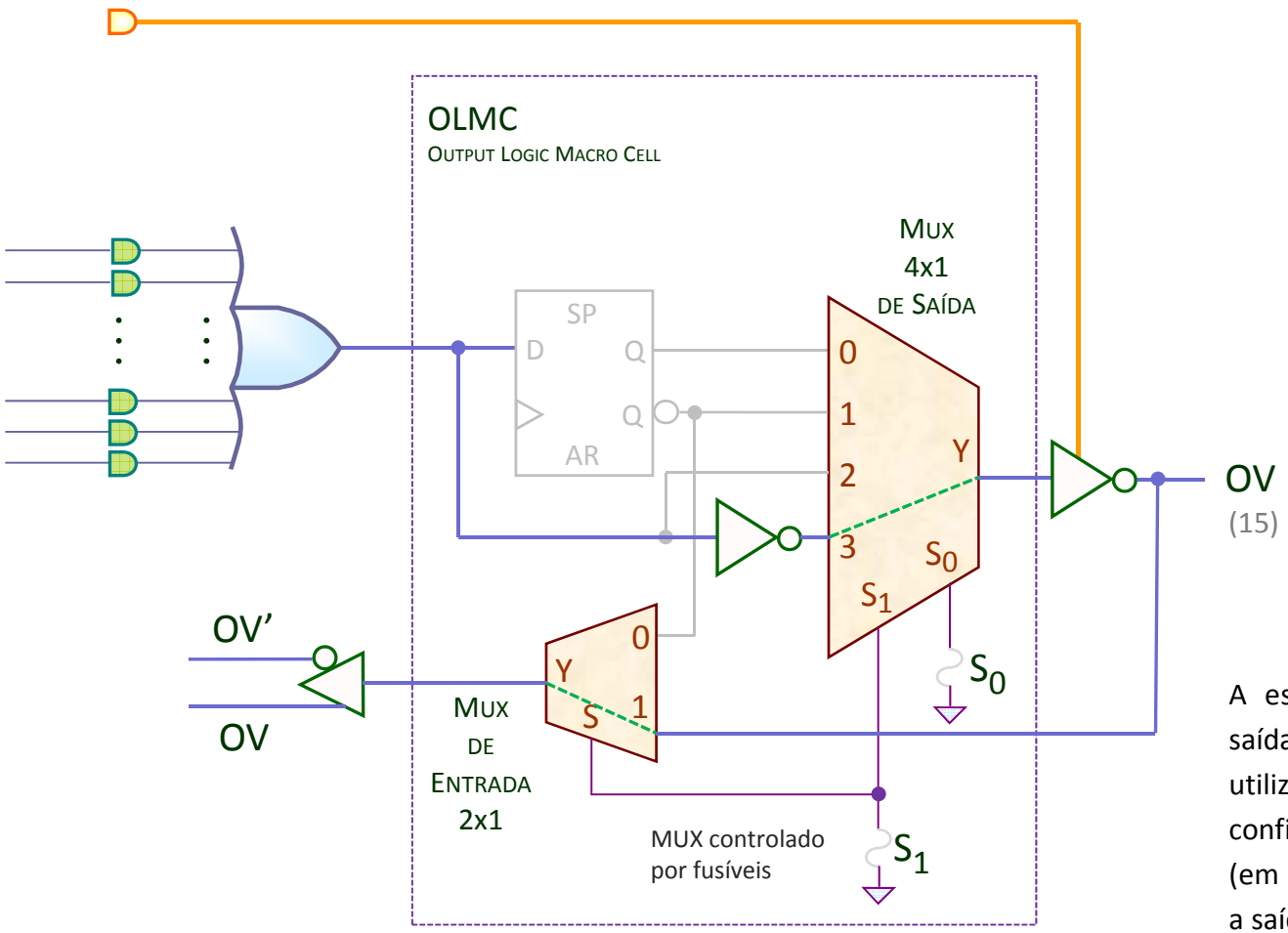


Geração de termos produto para a produção da saída OV no pino 15 de saída da PAL tal como documentado no ficheiro de extensão .DOC gerado pelo compilador.

OVERFLOW (OV) interveniente na geração da equação da FLAG **LESS**

Ao pino 15 corresponde uma estrutura AND-OR com 10 portas AND de 44 entradas cada (provenientes das 22 entradas da PAL). A décima primeira porta AND é dedicada ao Enable.

Configuração da Macrocélula da Saída Overflow



$$OV = (C_3 \oplus C_2) \cdot S'_2 + (A_2 \oplus A_1) \cdot S_2$$

Estrutura interna da Macrocélula da PAL configurada para a saída OV (Overflow).

As saídas combinatórias na PAL não passam pelo flip-flop tipo D presente em cada uma das 10 macrocélulas. A indicação de que se pretende usar o valor presente numa saída combinatória para realimentação da matriz é feita utilizando o nome dessa saída no lado direito de uma expressão (sem qualquer extensão).

Por exemplo, L = OV $ RA2 significa que a saída combinatória L (Flag Less) é igual ao XOR entre RA2 e a função lógica da saída OV (Flag Overflow) que será pois realimentada como exemplificado na figura.

A escolha da configuração da Macrocélula de saída é feita de acordo com a aplicação do utilizador e é conseguida através de dois bits de configuração: S0 e S1. O Multiplexer de entrada (em baixo), realimenta a matriz programável com a saída.

Para as saídas de natureza Active-high como o OV, é seleccionada a entrada 3 negada do Multiplexer de saída, e a entrada 1 do Multiplexer de entrada – combinação S0=S1=1.

IMPLEMENTAÇÃO DE UMA ALU A 3 BITS EM LÓGICA DISCRETA

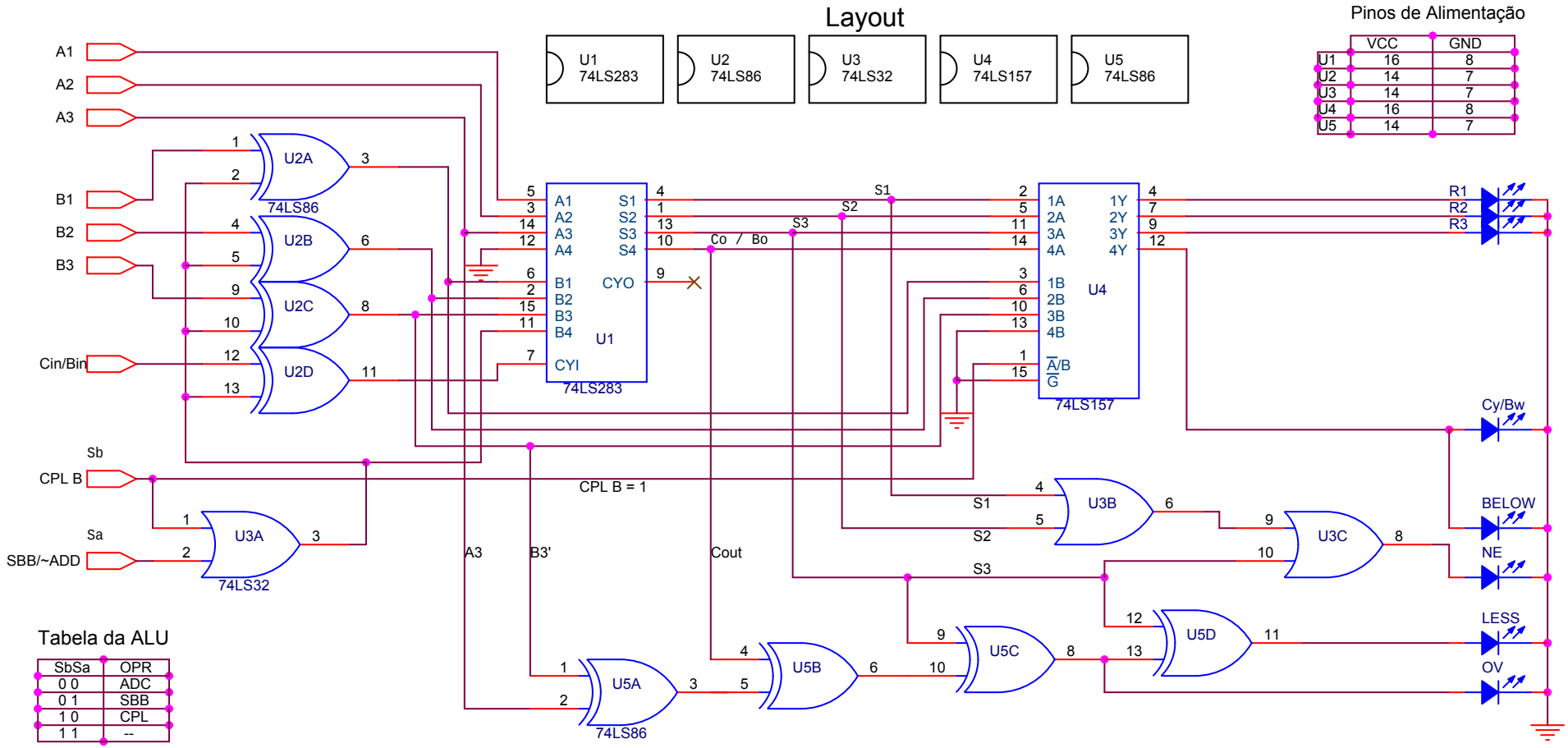


Tabela da ALU

SbSa	OPR
0 0	ADC
0 1	SBB
1 0	CPL
1 1	--

OPR	Flags			
	Sem Sinal		Com Sinal	
	Flag Erro	Flag Rel	Flag Erro	Flag Rel
ADC	Cy	- -	OV	- -
SBB	Bw	BELOW / NE	OV	LESS / NE
CPL	- -	- -	- -	- -

Flags Relacionais (Rel)	Flags Erro
BELOW (A < B): Cy	OVERFLOW - A3 XOR B3' XOR S3 XOR Co/Bo
LESS (A < B): OV XOR S3	Cy - Co
NE (Not Equal) (A <> B): S1 OR S2 OR S3	Bw - Bo

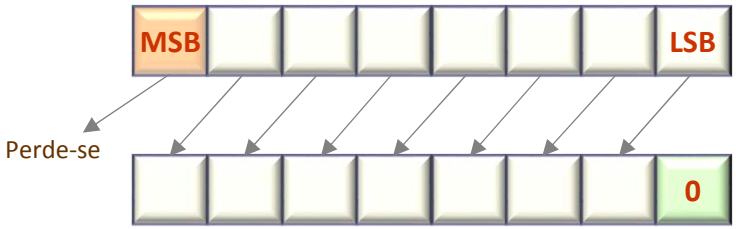
Diagrama Lógico da ALU revelando o SOMADOR-SUBTRACTOR, o MULTIPLEXER de saída e a lógica de geração das FLAGS.

Title: ALU com ADC, SBB, CPL e Flags Cy/Bw, Below, Overflow, Less e Not Equal		
Size: A	Document Number: ISEL, EIC, SD, Semestre Verão 2004	Rev: 2
Date: Thursday, May 27, 2004	Sheet 1 of 1	

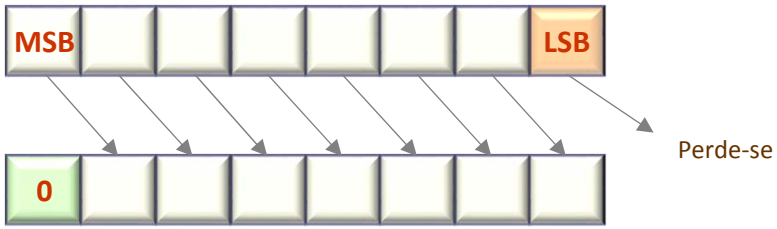
Instituto Superior de Engenharia de Lisboa

OUTRAS OPERAÇÕES DE UMA ALU – SHIFT

SHIFT LEFT LOGICAL (SLL)

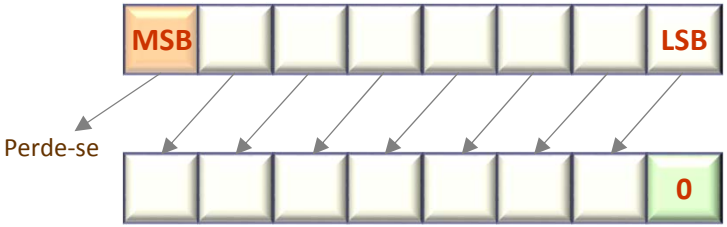


SHIFT RIGHT LOGICAL (SRL)

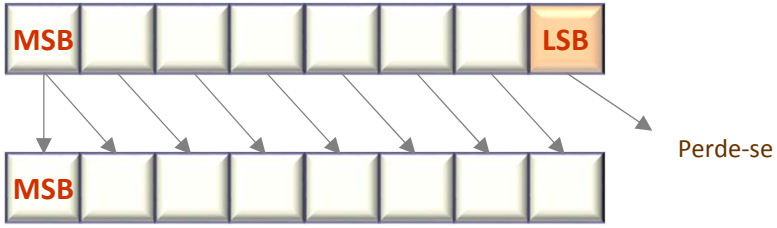


- Nas operações SLL e SRL é forçado um zero nas posições deixadas vagas.

SHIFT LEFT ARITHMETIC (SLA)



SHIFT RIGHT ARITHMETIC (SRA)

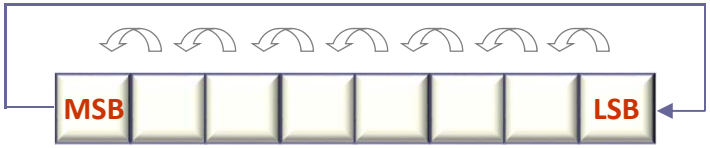


Variantes das operações SHIFT LEFT e SHIFT RIGHT mostrando o conteúdo de um registo de 8 bits antes e depois de cada operação.

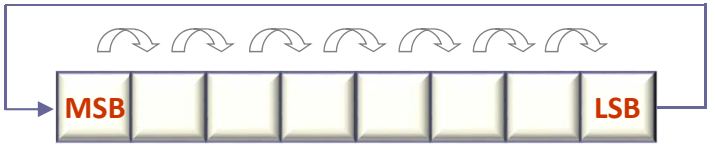
- As operações de deslocamento lógico SLL e SRL realizam um deslocamento sobre os bits de um número binário acrescentando zeros às posições que se tornarem vazias e desprezando os bits que ficarem em posições não existentes (função do tamanho de palavra).
- As operações SLA e SRA tratam os operandos na representação em complemento para 2 (números com sinal).
- Na operação SLA a mudança do MSB significa ocorrência de OVERFLOW. Na operação SRA não pode haver ocorrência de OVERFLOW.
- As operações SLL e SLA parecem equivalentes: a única diferença é a afectação da flag OVERFLOW na operação SLA.
- A operação SLA a um bit equivale à multiplicação por dois. A operação SRA a um bit equivale à divisão por dois, mas trunca os números no 'sentido errado' (o resultado de –1 ÷ 2 = –1 e não a 0).

Outras Operações de uma ALU – Rotate

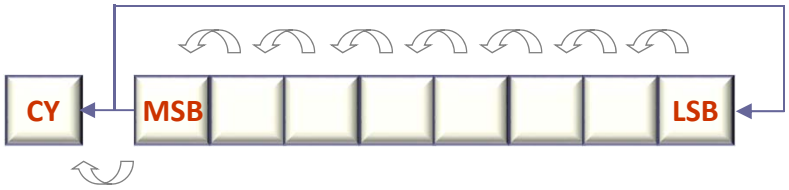
Rotate Left (Circular Shift Left)



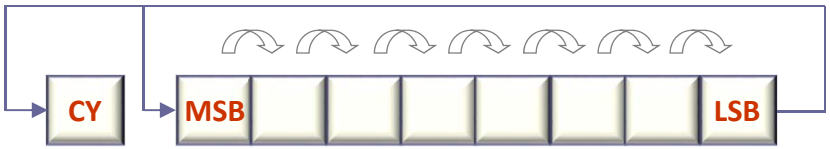
Rotate Right (Circular Shift Right)



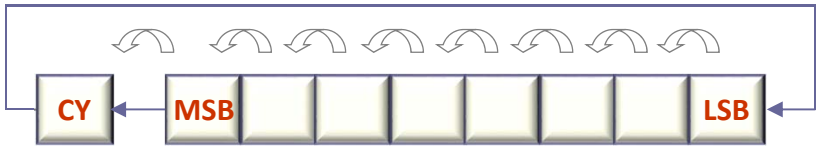
Rotate Left with Carry



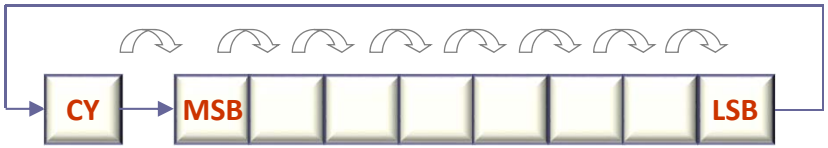
Rotate Right with Carry



Rotate Left through Carry

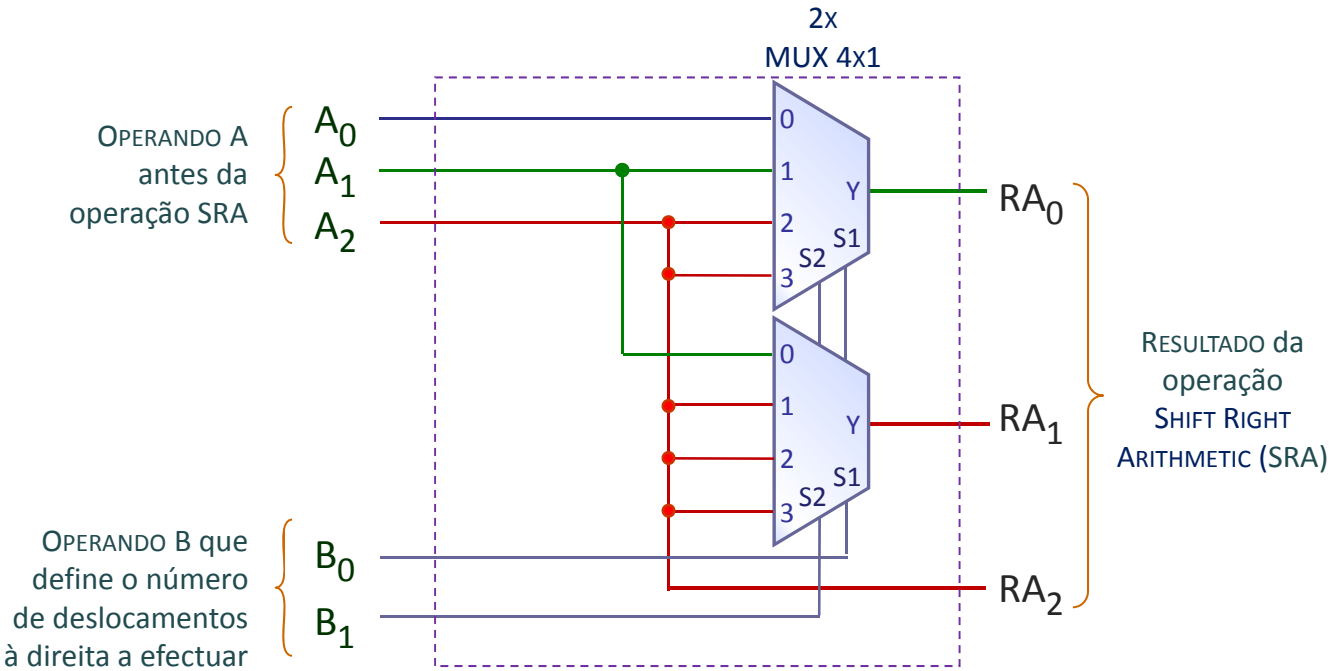


Rotate Right through Carry



Variantes das operações Rotate Left e Rotate Right mostrando o conteúdo de um registo de 8 bits antes e depois de cada operação.

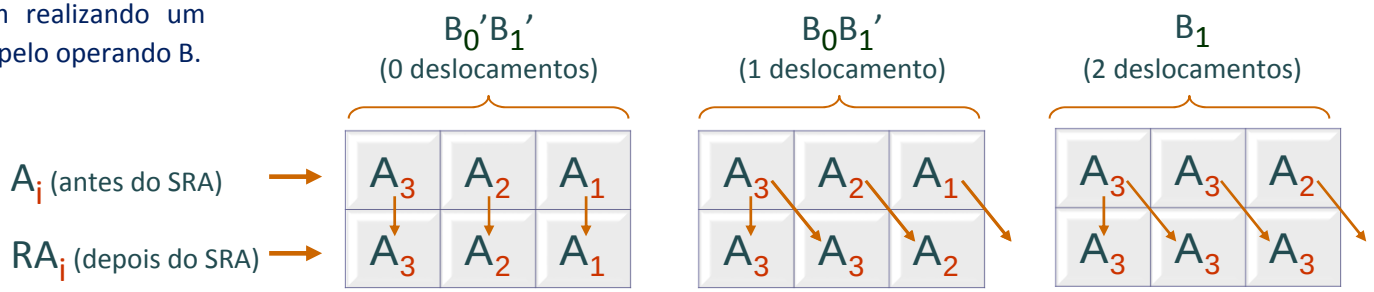
Operação Shift Right Arithmetic



B_1	B_0	Número de Deslocamentos
0	0	0
0	1	1
1	0	2
1	1	2

Tabela operacional relativa ao operando B que define o número de deslocamentos a efectuar pelo operando A.

Implementação do bloco de um Barrel Shift que implementa uma operação Shift Right Arithmetic (SRA) de 3 bits usando dois MUX 4x1, cada um realizando um número de deslocamentos que é definido pelo operando B.

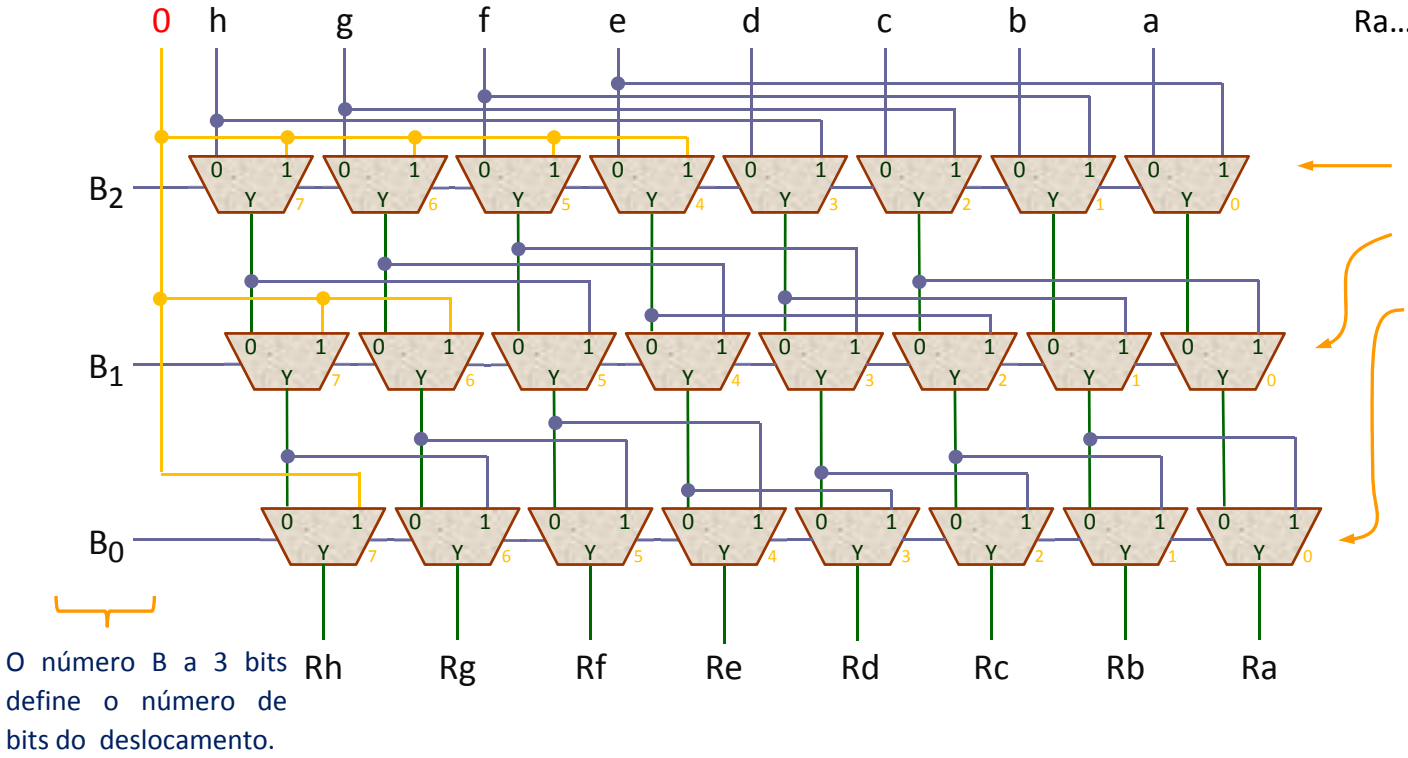


O Deslocamento Aritmético à direita (SRA) replica sempre o bit de sinal (MSB)

Número de deslocamentos da operação Shift Right Arithmetic mostrando o conteúdo do operando A e do Resultado RA depois de cada operação de deslocamento.

BARREL-SHIFTER DE 8 BITS E 3 ANDARES QUE EXECUTA OPERAÇÕES DE SHIFT RIGHT LOGICAL

a..h é o operando de entrada a 8 bits.
Ra…Rh é o resultado da operação de SHIFT RIGHT LOGICAL.



Estrutura de um BARREL-SHIFTER que realiza o deslocamento lógico de números binários de uma quantidade entre 0 e 7 bits para a direita (SRL) fazendo uso de blocos Multiplexer 2x1 de 1 bit. B define a quantidade de bits a deslocar, e cada bit de B controla um andar do BARREL-SHIFTER.

Na notação Java e C/C++ usa-se a simbologia >> para indicar SHIFT RIGHT e << para indicar SHIFT LEFT. OS BARREL-SHIFTERS são utilizados na manipulação de bits (extração e fixação de determinados bits em palavras), e na multiplicação e divisão por potências inteiras de 2.

O BARREL-SHIFTER é um circuito combinatório que executa uma operação de deslocamento para a esquerda ou para a direita.

Pode ser obtido usando multiplexers dispostos em andares sucessivos, cada um realizando uma quantidade de deslocamentos que é potência de dois (4 bits, 2 bits e 1 bit).

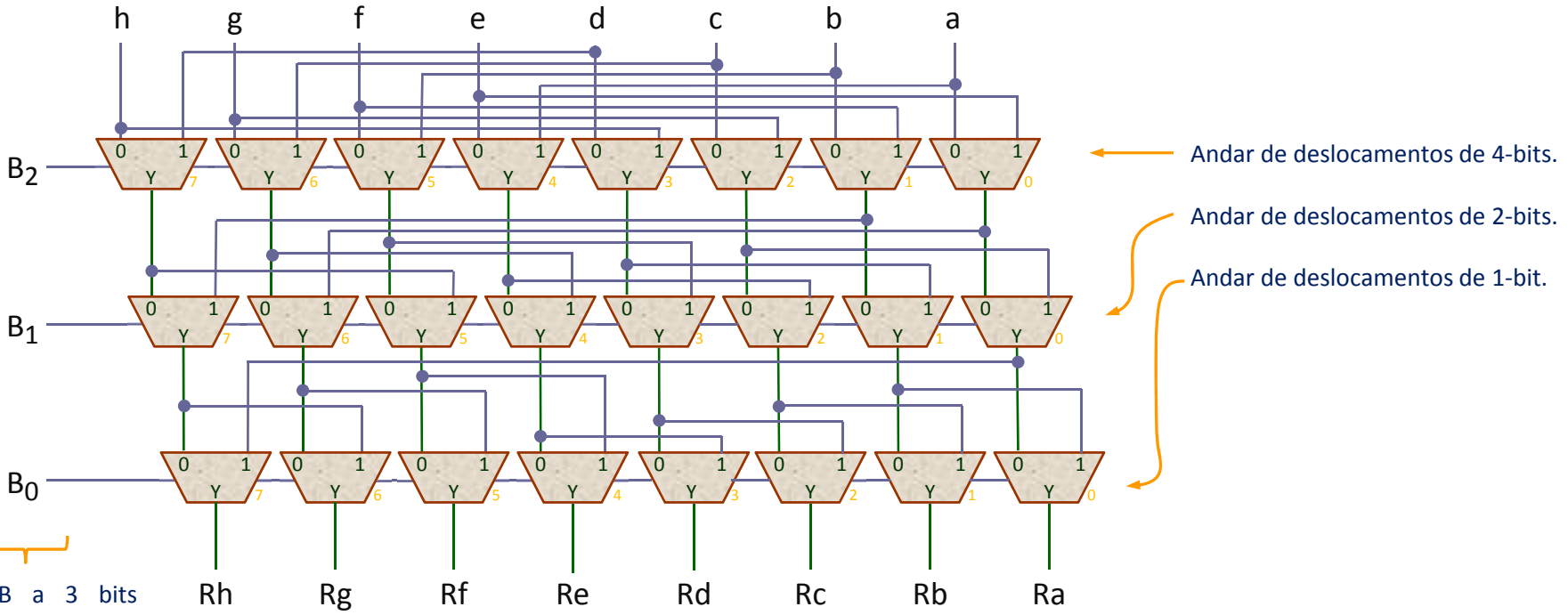
Através da ativação de cada um ou de vários andares, é possível obter um número diferente de posições a deslocar.

Barrel-shifter de 8 bits: Tabela de Operações de Shift e Rotate

Operação	B	Rh	Rg	Rf	Re	Rd	Rc	Rb	Ra
Shift Right Logical (SRL) 3 bits	011	0	0	0	h	g	f	e	d
Shift Left Logical (SLL) 3 bits	011	e	d	c	b	a	0	0	0
Shift Right Arithmetic (SRA) 2 bits	010	h	h	h	g	f	e	d	c
Shift Left Arithmetic (SLA) 2 bits	010	f	e	d	c	b	a	0	0
Rotate Right (RR) 7 bits	010	g	f	e	d	c	b	a	h
Rotate Left (RL) 2 bits	010	f	e	d	c	b	a	h	g

Exemplos de operações básicas de deslocamento em Shift e em Rotate.

BARREL-SHIFTER DE 8 BITS E 3 ANDARES QUE EXECUTA OPERAÇÕES DE ROTATE RIGHT



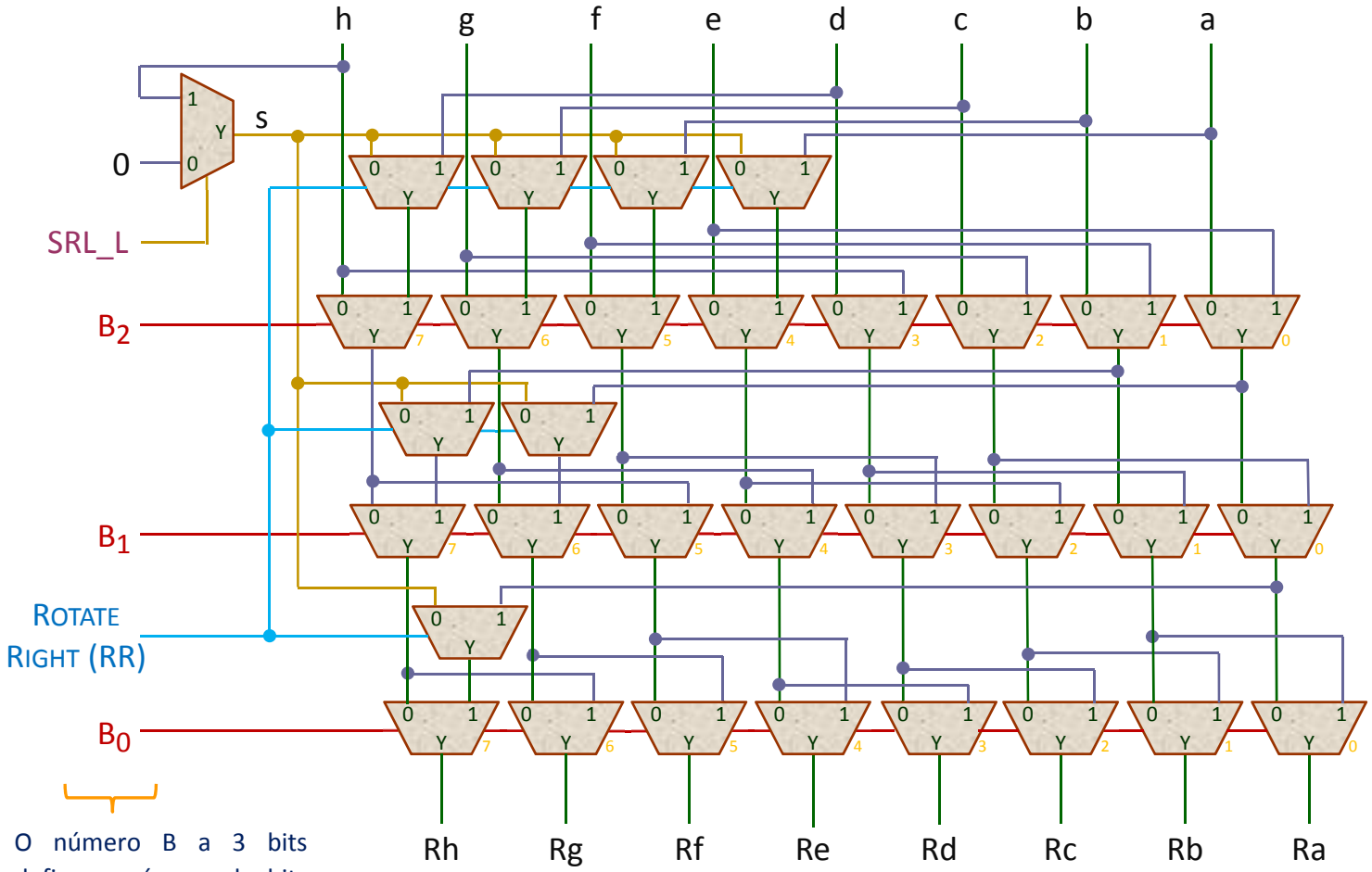
O número B a 3 bits define o número de bits de uma operação ROTATE .

Estrutura de um BARREL-SHIFTER que realiza o deslocamento rotativo (ROTATE) de números binários de uma quantidade entre 0 e 7 bits para a direita (RR).

Em cada andar controlado por B_k as linhas nas entradas de cada Multiplexer são ligadas de modo a permitir o encaminhamento dos 2^k bits de menor peso para os 2^k Multiplexers de maior peso.

Se B_k =0 não há deslocamento.
Se B_k =1 os dados de entrada são deslocados de 2^k bits.

BARREL-SHIFTER DE 8 BITS E 3 ANDARES QUE EXECUTA OPERAÇÕES DE SHIFT LOGICAL E ARITHMETIC E ROTATE RIGHT



O número B a 3 bits define o número de bits de uma operação SHIFT ou ROTATE .

Estrutura de um BARREL-SHIFTER com Multiplexers adicionais que realiza as operações de SHIFT RIGHT LOGICAL, SHIFT RIGHT ARITHMETIC OU ROTATE Right de números binários de uma quantidade entre 0 e 7 bits.

O MULTIPLEXER em cima à esquerda (com a saída 'S') selecciona entre:

- 0 para operações de deslocamento lógico à direita (SHIFT RIGHT LOGICAL, SRL), e
- h para operações de deslocamento aritmético à direita (SHIFT RIGHT ARITHMETIC, SRA).

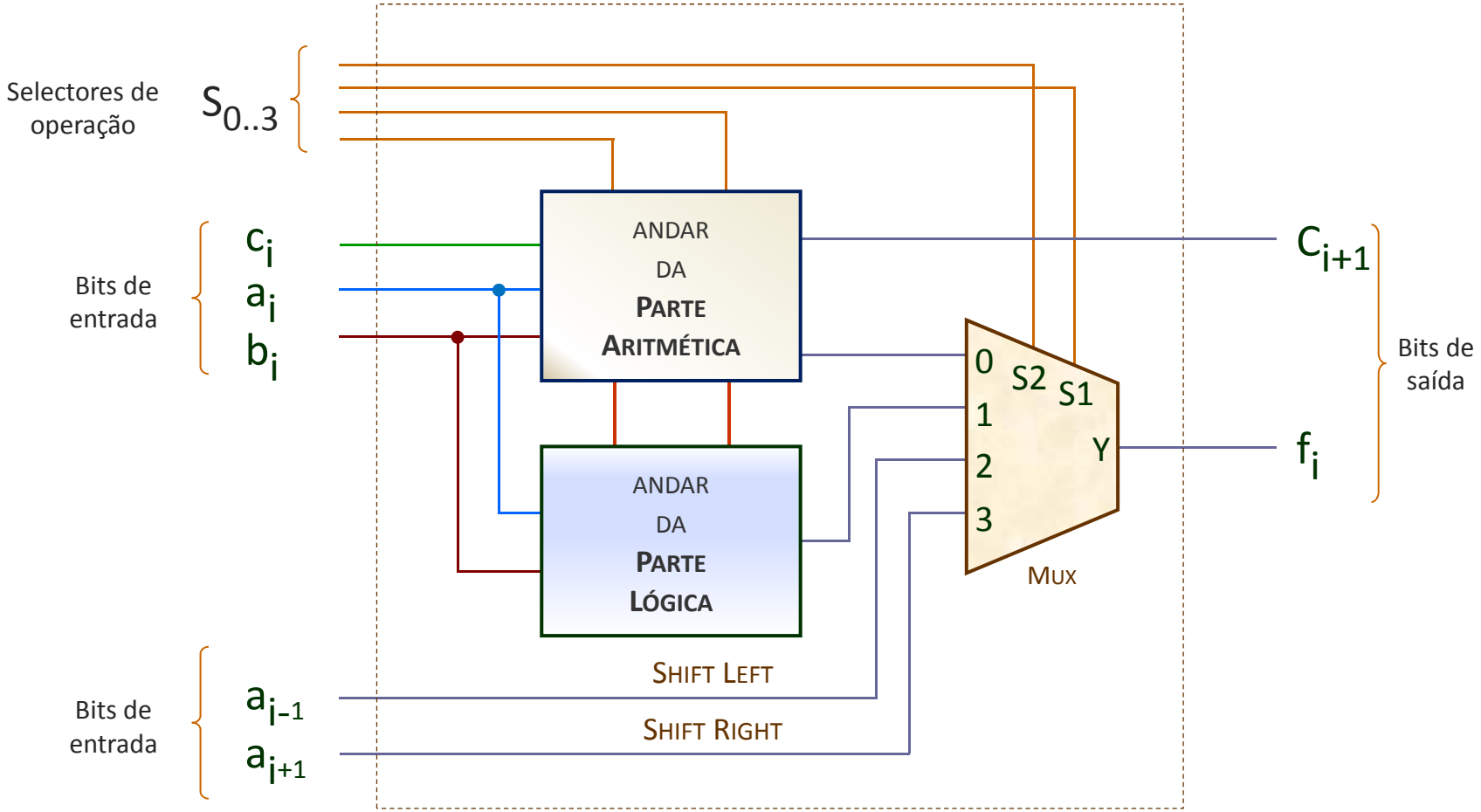
Em cada andar controlado por B_k existem 2^k Multiplexers 2x1 que selecionam entre a entrada S para as operações de SHIFT, e os 2^k bits de menor peso do operando de entrada para as operações de ROTATE.

Tabela de Operações do BARREL-SHIFTER.

RR	SRA_L	Operação
0	0	SRL
0	1	SRA
1	0	RR
1	1	RR

ANDAR BIT SLICE DE UMA ALU INCLUINDO OPERAÇÕES DE SHIFT (EX. 6-5)

Exemplo 6-5



Andar BIT SLICE de uma ALU.

OPERANDOS, RESULTADO E FLAGS NUMA ALU (EX. 6-6-1)

Exemplo 6-6

OBJECTIVO

a) Representar o número $-(624)_8$ em código dos complementos na base 2, com o menor número de bits.

b) Na subtração indicada ao lado em cima, obter as expressões booleanas dos bits do resultado R e do indicador de arrasto (Bw), entendendo A e B como variáveis binárias.

c) Completar os campos da tabela ao lado, assumindo que numa ALU de 4 bits está selecionada a operação SBB (R = A – B – Bin).

		A'B	1	A
	–	B	A'B	B
Bw	R_2	R_1	R_0	

		R	A	B	CyBwi	CyBwo	OV	AE	L
Base 2					0		–		–
Base 10	Natural	10							
	Relativo			–7		–		–	

RESOLUÇÃO

a) $+624_8 = 0\,110\;010\;100$ ⇨ $-624_8 = 1\,001\;101\;100$ Há que não esquecer o bit de sinal.

b)

			0 ←	Borrow-in		
	A'B	1	A ←	A_2	A_1	A_0
–	B	A'B	B ←	B_2	B_1	B_0
Bw	R_2	R_1	R_0 ←	Resultado		

⇨

			A+B' A+B' 1 ←	Borrow-in		
	A'B	1	A			
+	B'	A+B'	B'			
B'	B	1	A⊕B ←	Resultado		

Bw = Cy' = B

OPERANDOS, RESULTADO E FLAGS NUMA ALU (EX. 6-6-2)

Exemplo 6-6

c) R = A – B – Bin

				0	←	Borrow-in
					←	$A_3\ A_2\ A_1\ A_0$
–	1	0	0	1	←	$B_3\ B_2\ B_1\ B_0$
	1	0	1	0	←	Resultado

⇨

	1	1	1	1	←	Borrow-in
	0	0	1	1	←	$A_3\ A_2\ A_1\ A_0$
+	0	1	1	0	←	$B_3\ B_2\ B_1\ B_0$
0	1	0	1	0	←	Resultado

Bw = Cy'= 1

	A	B	R	C_yB_{wi}	C_yB_{wo}	Ov	AE	L
$\mathbb{N}_0$	3	9	10	0	1	–	0	–
$\mathbb{Z}$	+ 3	– 7	– 6		–	1	–	0

LSD – 6 ÍNDICE 2