APÊNDICE A3: Linguagem *Assembly* do GLiTCH

Notas

1. FB representa a *flag de breakpoint* que assume, ou o valor **1** em caso de se desejar parar a execução após executar a instrução, ou o valor **0** no caso de não se desejar parar a execução.

2. Os campos **4** e **6** do microcódigo são padrões com 8 caracteres (0, 1 ou x). Por isso quando se representa um valor do tipo "x....x" significa "xxxxxxxx".

3. Quando se usarem as siglas "**ls**" e "**ms**" quer significar-se "menos significativo" e "mais significativo". Outras formas análogas são "**lsb/lsw**" e "**msb/msw**", para designar "byte/palavra menos significativo" e "byte/palavra mais significativo".

Instruções Para Manipular Dados do VSR

shift_vsr( )

- **Operação:** Lê a linha menos significativa do VSR para uma variável global, faz o deslocamento de uma posição das linhas do VSR e carrega a linha mais significativa do VSR com um novo valor.
- **Parâmetros:** não tem.
- **Operação do PBL:** pbl_nop( )
- **Operação dos PEs:** vsrShift( )
- **Resultados:**
  ◊ VOUT[num_chip] = linha ls do VSR
  ◊ Deslocamento de linhas do VSR
  ◊ Linha ms do VSR = VIN[num_chip]
- **Microcódigo:**
  64 0 0 x...x 0 x...x xxxx 222 FB VsrShift()

- **Notas:** Esta instrução foi incluída no emulador da forma mais simples, não funcionando actualmente de forma independente do resto do *chip* como seria de desejar.

rotate_image( )

- **Operação:** Copia-se a coluna menos significativa do VSR para as TAGs dos PEs, efectua-se a rotação de uma posição de todas as colunas do VSR e escreve-se o antigo valor das TAGs na coluna mais significativa do VSR.
- **Parâmetros:** Não tem
- **Operação do PBL:** pbl_nop( )
- **Operação dos PEs:** rotate_imag( )

- **Resultados:**
  - ◊ TEMP = VSR[coluna ls]
  - ◊ Rotação das colunas do VSR
  - ◊ VSR[coluna ms] = TAG
  - ◊ TAG = TEMP
- **Microcódigo:**

57 0 0 x...x 0 x...x xxxx 222 FB RotateImag()

extract_image(pos)

- **Operação:** Copia a coluna **pos** da CAM para as TAGs dos PEs, efectua a rotação de uma posição das colunas do VSR e escreve na coluna mais significativa do VSR o conteúdo das TAGs.
- **Parâmetros:**
  - ◊ **pos** → valor no intervalo [0:63] , que define a coluna da CAM a copiar para as TAGs.
- **Operação do PBL:** (i) pbl_match( ) e (ii) pbl_nop( )
- **Operação dos PEs:** (i) tag_match( ) e (ii) extract_imag( )
- **Resultados:**
  - ◊ TAG = CAM[coluna pos]
  - ◊ TEMP = VSR[coluna ls]
  - ◊ Rotação das colunas do VSR
  - ◊ VSR[coluna ms] = TAG
- **Microcódigo:**

2 1 pos x...x1 0 x...x xxxx 112 FB TagMatch()
65 0 0 x...x 0 x...x xxxx 222 FB ExtractImag()

- **Notas:** A microinstrução tag_match() copia uma coluna da CAM paras TAGs.

restore_image(pos)

- **Operação:** Copia as TAGs para a coluna mais significativa do VSR, faz a rotação das colunas do VSR, lê a coluna menos significativa do VSR para as TAGs e depois escreve as TAGs na coluna **pos** da CAM.
- **Parâmetros:**
  - ◊ **pos** → valor no intervalo [0:63] , que define a coluna da CAM onde se escrevem as TAGs.
- **Operação do PBL:** pbl_write()
- **Operação dos PEs:** write_image()
- **Resultados:**
  - ◊ TEMP = VSR[coluna ls]
  - ◊ Rotação das colunas do VSR
  - ◊ VSR[coluna ms] = TAG
  - ◊ TAG = TEMP
  - ◊ CAM[coluna pos] = TAG
- **Microcódigo:**

56 2 pos x...x1 0 x...x xxxx 001 FB WriteImage()

- **Notas:** Com o microcódigo indicado escreve-se em todos os bits da coluna **pos** da CAM a coluna lida do VSR. Se se desejar escrever apenas em alguns bits da CAM, devem incluir-se 2 microinstruções:
  < uma microinstrução que afecta o registo MS da forma desejada >
  56 2 pos x...x1 0 x...x xxxx 101 FB WriteImage()

Instruções de *Match*

match(pos1, patt1, pos2, patt2, sub)

- **Operação:** Os PEs que fizerem a correspondência (*match*) entre os 2 padrões (**patt1**, **patt2**) e o conteúdo da CAM nas respectivas posições (**pos1**, **pos2**), bem como a correspondência entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET, colocam a "1" a TAG e o registo de *carry* (CY). Os outros PEs colocam a TAG e o CY a "0". O registo MS de cada PE fica com o valor "1" se houver correspondência entre o padrão de subset e o conteúdo da CAM-de-SUBSET, e fica com "0" se não houver correspondência.
- **Parâmetros:**
  ◊ **pos1, pos2** → valores no intervalo [0:63] , que representam posições (colunas) na CAM
  ◊ **patt1, patt2** → padrões com 8 dígitos ternários (0|1|x)
  ◊ **sub** → padrão de subset com 4 dígitos ternários (0|1|x)
- **Operação do PBL:** pbl_match( )
- **Operação dos PEs:** match( )
- **Resultados:**
  ◊ CY = TAG = MS & MD0 & MD1
  ◊ registo MS = 1/0, se houver/não_houver correspondência entre **sub** e a CAM-de-SUBSET
  ◊ sinal MD0 = 1/0, se houver/não_houver correspondência entre os bits a "0" de **patt1,2** e a CAM
  ◊ sinal MD1 = 1/0, se houver/não_houver correspondência entre os bits a "1" de **patt1,2** e a CAM
- **Microcódigo:**
  1 1 pos1 patt1 pos2 patt2 sub 114 FB Match()

tag_match(pos1, patt1, pos2, patt2, sub)

- **Operação:** A funcionalidade é idêntica à instrução **match( )**, com a excepção de apenas afectar as TAGs. O registo de *carry* (CY) não é alterado por esta instrução.
- **Parâmetros:** Iguais aos de **match( )**
- **Operação do PBL:** pbl_match( )
- **Operação dos PEs:** tag_match( )
- **Resultados:**
  ◊ CY = CY
  ◊ TAG = MS & MD0 & MD1
  ◊ MS, MD0 e MD1 → são afectados do mesmo modo que em **match( )**

- **Microcódigo:**
  2 1 pos1 patt1 pos2 patt2 sub 112 FB TagMatch()

cy_match(pos1, patt1, pos2, patt2, sub)

- **Operação:** A funcionalidade é idêntica à instrução **match( )**, com a excepção de apenas afectar os registos de *carry* (CY). As TAGs não são alteradas por esta instrução.
- **Parâmetros:** Iguais aos de **match( )**
- **Operação do PBL:** pbl_match( )
- **Operação dos PEs:** cy_match( )
- **Resultados:**
  ◊ TAG = TAG
  ◊ CY = MS & MD0 & MD1
  ◊ MS, MD0 e MD1 → são afectados do mesmo modo que em **match( )**
- **Microcódigo:**
  3 1 pos1 patt1 pos2 patt2 sub 115 FB CyMatch()

or_match(pos1, patt1, pos2, patt2, sub)

- **Operação:** A funcionalidade é idêntica à instrução **match( )**, sendo as TAGs afectadas da mesma forma. A diferença está em que cada registo CY guardará o resultado da operação **OR** entre o transporte da presente operação de *match* e o conteúdo de CY.
- **Parâmetros:** Iguais aos de **match( )**
- **Operação do PBL:** pbl_match( )
- **Operação dos PEs:** match( )
- **Resultados:**
  ◊ TAG = MS & MD0 & MD1
  ◊ CY = CY | (MS & MD0 & MD1)
  ◊ MS, MD0 e MD1 → são afectados do mesmo modo que em **match( )**
- **Microcódigo:**
  1 1 pos1 patt1 pos2 patt2 sub 111 FB Match()

and_match(pos1, patt1, pos2, patt2, sub)

- **Operação:** A funcionalidade é idêntica à instrução **match( )**, sendo as TAGs afectadas da mesma forma. A diferença está em que cada registo CY guardará o resultado da operação **AND** entre o transporte da presente operação de *match* e o conteúdo de CY.
- **Parâmetros:** Iguais aos de **match( )**
- **Operação do PBL:** (i) pbl_match( ) e (ii) pbl_match( )
- **Operação dos PEs:** (i) tag_match( ) e (ii) and_match( )
- **Resultados:**
  ◊ TAG = MS & MD0 & MD1
  ◊ CY = CY & (MS & MD0 & MD1)
  ◊ MS, MD0 e MD1 → são afectados do mesmo modo que em **match( )**

- **Microcódigo:**

2	**1**	**pos1**	**patt1**	**pos2**	**patt2**	**sub**	**112**	**FB**	**TagMatch()**
4	**1**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**341**	**FB**	**AndMatch()**

tag_match_subset(sub)

- **Operação:** O registo TAG de cada PE fica com o valor "1" se houver correspondência entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET, e fica com "0" se não houver correspondência. Os registos MS e CY não são alterados por esta instrução.
- **Parâmetros:**
  - ◊ **sub** → padrão de subset com 4 dígitos ternários (0|1|x)
- **Operação do PBL:** pbl_match( )
- **Operação dos PEs:** tag_subset( )
- **Resultados:**
  - ◊ TAG = 1, se houver correspondência entre **sub** e a CAM-de-SUBSET
  - ◊ CY = CY
  - ◊ MS = MS
- **Microcódigo:**

60	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**222**	**FB**	**TagSubset()**

cy_match_subset(sub)

- **Operação:** O registo CY de cada PE fica com o valor "1" se houver correspondência entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET, e fica com "0" se não houver correspondência. Os registos MS e TAG não são alterados por esta instrução.
- **Parâmetros:**
  - ◊ **sub** → padrão de subset com 4 dígitos ternários (0|1|x)
- **Operação do PBL:** pbl_match( )
- **Operação dos PEs:** cy_subset( )
- **Resultados:**
  - ◊ CY = 1, se houver correspondência entre **sub** e a CAM-de-SUBSET
  - ◊ TAG = TAG
  - ◊ MS = MS
- **Microcódigo:**

61	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**114**	**FB**	**CySubset()**

Instruções Aritméticas

copy(op1, sub)

- **Operação:** Copia uma coluna da CAM (**op1**) para as TAGs e coloca os registos CY a "0". O registo MS guarda o resultado da operação de *match* entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET.

- **Parâmetros:**
  - ◊ **op1** → valor no intervalo [0:63] , que define a coluna da CAM a copiar para as TAGs
  - ◊ **sub** → padrão de subset com 4 dígitos ternários (0|1|x)
- **Operação do PBL:** pbl_match( )
- **Operação dos PEs:** arith_this( )
- **Resultados:**
  - ◊ TAG = CAM[coluna op1]
  - ◊ CY = 0
  - ◊ MS → é afectado do mesmo modo que em **match( )**
- **Microcódigo:**
  5 1 0 x...x op1 x...x1 sub 314 FB ArithThis()

add_nc(op1, op2, sub)

- **Operação:** Guarda nas TAGs o resultado da soma de 2 operandos especificados por **op1** e **op2**. Nos registos CY guarda o transporte desta soma e em MS o resultado da operação de *match* entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET.
- **Parâmetros:**
  - ◊ **op1, op2** :

"0"	←	operando = 0
"1"	←	operando = 1
posição na CAM [0:63]	←	operando = conteúdo da coluna da CAM
TAG	←	operando = TAGs dos
TAG_ABOVE	←	operando = TAGs após *shift* para cima **(*)**
TAG_BELOW	←	operando = TAGs após *shift* para baixo **(†)**
IMAGE	←	operando = coluna ls do VSR

  - ◊ **sub** → padrão de subset com 4 dígitos ternários (0|1|x)
- **Operação do PBL:** pbl_match( )
- **Operação dos PEs:**
  - ◊ arith_above( ) ← se **op1** ou **op2** for TAG_ABOVE
  - ◊ arith_below( ) ← se **op1** ou **op2** for TAG_BELOW
  - ◊ arith_image( ) ← se **op1** ou **op2** for IMAGE
  - ◊ arith_this( ) ← outros casos
- **Resultados: (‡)**
  - ◊ TAG = CY_IN ^ (OP0 ^ OP1)
  - ◊ CY = ((OP0 ^ OP1) & CY_IN) | (OP0 & OP1)

- **Notas:**
  (*) O operando é o conteúdo dos registos TAG após deslocar as TAGs dum PE para o PE vizinho menos significativo.
  (†) O operando é o conteúdo dos registos TAG após deslocar as TAGs dum PE para o PE vizinho mais significativo.
  (‡) O valor que CY_IN, OP0 ou OP1 assumem no cálculo de TAG e CY é definido pelo campo nº 8 do microcódigo ("*shared signals*").

- **Microcódigo** (conjunto completo de alternativas)**:**

add_nc("0", "0",sub)	**5**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**334**	**FB**	**ArithThis()**
add_nc("0", "1",sub)	**5**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**314**	**FB**	**ArithThis()**
add_nc("1", "0",sub)	**5**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**314**	**FB**	**ArithThis()**
add_nc("1", "1",sub)	**5**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**114**	**FB**	**ArithThis()**
add_nc("0",pos,sub)	**5**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**314**	**FB**	**ArithThis()**
add_nc(pos, "0",sub)	**5**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**314**	**FB**	**ArithThis()**
add_nc("1",pos,sub)	**5**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**114**	**FB**	**ArithThis()**
add_nc(pos, "1",sub)	**5**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**114**	**FB**	**ArithThis()**
add_nc(pos1,pos2,sub)	**5**	**1**	**pos1**	**x...x0**	**pos2**	**x...x1**	**sub**	**314**	**FB**	**ArithThis()**
add_nc(pos,TAG,sub)	**5**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**ArithThis()**
add_nc(TAG,pos,sub)	**5**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**ArithThis()**
add_nc(pos,TAG_ABOVE,sub)	**6**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**ArithAbove()**
add_nc(TAG_ABOVE,pos,sub)	**6**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**ArithAbove()**
add_nc(pos,TAG_BELOW,sub)	**7**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**ArithBelow()**
add_nc(TAG_BELOW,pos,sub)	**7**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**ArithBelow()**
add_nc(pos,IMAGE,sub)	**8**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**ArithImage()**
add_nc(IMAGE,pos,sub)	**8**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**ArithImage()**

add(op1, op2, sub)

- **Operação:** Guarda nas TAGs o resultado da soma de 2 operandos especificados por **op1** e **op2** com o registo CY. Nos registos CY guarda-se o transporte desta soma e em MS o resultado da operação de *match* entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET.
- **Parâmetros:** Iguais aos da instrução **add_nc( )**
- **Operação do PBL:** Igual à da instrução **add_nc( )**
- **Operação dos PEs:** Iguais às da instrução **add_nc( )**
- **Resultados:** Iguais aos da instrução **add_nc( )**
- **Microcódigo** (conjunto completo de alternativas)**:**

add("0", "0",sub)	**5**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**331**	**FB**	**ArithThis()**
add("0", "1",sub)	**5**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**311**	**FB**	**ArithThis()**
add("1", "0",sub)	**5**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**311**	**FB**	**ArithThis()**
add("1", "1",sub)	**5**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**111**	**FB**	**ArithThis()**
add("0",pos,sub)	**5**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**311**	**FB**	**ArithThis()**
add(pos, "0",sub)	**5**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**311**	**FB**	**ArithThis()**
add("1",pos,sub)	**5**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**111**	**FB**	**ArithThis()**
add(pos, "1",sub)	**5**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**111**	**FB**	**ArithThis()**
add(pos1,pos2,sub)	**5**	**1**	**pos1**	**x...x0**	**pos2**	**x...x1**	**sub**	**311**	**FB**	**ArithThis()**
add(pos,TAG,sub)	**5**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**341**	**FB**	**ArithThis()**
add(TAG,pos,sub)	**5**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**341**	**FB**	**ArithThis()**
add(pos,TAG_ABOVE,sub)	**6**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**341**	**FB**	**ArithAbove()**
add(TAG_ABOVE,pos,sub)	**6**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**341**	**FB**	**ArithAbove()**
add(pos,TAG_BELOW,sub)	**7**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**341**	**FB**	**ArithBelow()**
add(TAG_BELOW,pos,sub)	**7**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**341**	**FB**	**ArithBelow()**
add(pos,IMAGE,sub)	**8**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**341**	**FB**	**ArithImage()**
add(IMAGE,pos,sub)	**8**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**341**	**FB**	**ArithImage()**

sub_nc(op1, op2, sub)

- **Operação:** Guarda nas TAGs o resultado da subtracção entre 2 operandos especificados por **op1** e **op2**. Nos registos CY guarda-se o transporte desta subtracção e em MS o resultado da operação de *match* entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET. Para se obter um resultado correcto em complemento para 2, inverte-se **op2** e soma-se "1". Deste modo resulta que:
  TAG = <op1> ⊕ NOT(< op2>) ⊕ 1
- **Parâmetros:** Iguais aos da instrução **add_nc( )**
- **Operação do PBL:** Igual à da instrução **add_nc( )**
- **Operação dos PEs:** Iguais às da instrução **add_nc( )**
- **Resultados:** Usam as mesmas fórmulas da instrução **add_nc( )**
- **Microcódigo** (conjunto completo de alternativas)**:**

sub_nc("0", "0",sub)	**5**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**313**	**FB**	**ArithThis()**
sub_nc("0", "1",sub)	**5**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**333**	**FB**	**ArithThis()**
sub_nc("1", "0",sub)	**5**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**113**	**FB**	**ArithThis()**
sub_nc("1", "1",sub)	**5**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**133**	**FB**	**ArithThis()**
sub_nc("0",pos,sub)	**5**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**333**	**FB**	**ArithThis()**
sub_nc(pos, "0",sub)	**5**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**313**	**FB**	**ArithThis()**
sub_nc("1",pos,sub)	**5**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**133**	**FB**	**ArithThis()**
sub_nc(pos, "1",sub)	**5**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**333**	**FB**	**ArithThis()**
sub_nc(pos1,pos2,sub)	**5**	**1**	**pos1**	**x...x0**	**pos2**	**x...x1**	**sub**	**333**	**FB**	**ArithThis()**
sub_nc(pos,TAG,sub)	**5**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**353**	**FB**	**ArithThis()**
sub_nc(TAG,pos,sub)	**5**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**143**	**FB**	**ArithThis()**
sub_nc(pos,TAG_ABOVE,sub)	**6**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**353**	**FB**	**ArithAbove()**
sub_nc(TAG_ABOVE,pos,sub)	**6**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**143**	**FB**	**ArithAbove()**
sub_nc(pos,TAG_BELOW,sub)	**7**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**353**	**FB**	**ArithBelow()**
sub_nc(TAG_BELOW,pos,sub)	**7**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**143**	**FB**	**ArithBelow()**
sub_nc(pos,IMAGE,sub)	**8**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**353**	**FB**	**ArithImage()**
sub_nc(IMAGE,pos,sub)	**8**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**143**	**FB**	**ArithImage()**

sub(op1, op2, sub)

- **Operação:** As TAGs guardam o resultado da subtracção entre os 2 operandos especificados por **op1** e **op2**, tendo em conta o transporte existente em CY. Nos registos CY guarda-se o transporte desta subtracção e em MS o resultado da operação de *match* entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET. Utilizando notação em complemento para 2 resulta que:
  TAG = <op1> ⊕ NOT(< op2>) ⊕ CY
- **Parâmetros:** Iguais aos da instrução **add_nc( )**
- **Operação do PBL:** Igual à da instrução **add_nc( )**
- **Operação dos PEs:** Iguais às da instrução **add_nc( )**
- **Resultados:** Usam as mesmas fórmulas da instrução **add_nc( )**

- **Microcódigo** (conjunto completo de alternativas)**:**

sub("0", "0",sub)	**5**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**311**	**FB**	**ArithThis()**
sub("0", "1",sub)	**5**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**331**	**FB**	**ArithThis()**
sub("1", "0",sub)	**5**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**111**	**FB**	**ArithThis()**
sub("1", "1",sub)	**5**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**131**	**FB**	**ArithThis()**
sub("0",pos,sub)	**5**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**331**	**FB**	**ArithThis()**
sub(pos, "0",sub)	**5**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**311**	**FB**	**ArithThis()**
sub("1",pos,sub)	**5**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**131**	**FB**	**ArithThis()**
sub(pos, "1",sub)	**5**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**331**	**FB**	**ArithThis()**
sub(pos1,pos2,sub)	**5**	**1**	**pos1**	**x...x0**	**pos2**	**x...x1**	**sub**	**331**	**FB**	**ArithThis()**
sub(pos,TAG,sub)	**5**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**351**	**FB**	**ArithThis()**
sub(TAG,pos,sub)	**5**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**141**	**FB**	**ArithThis()**
sub(pos,TAG_ABOVE,sub)	**6**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**351**	**FB**	**ArithAbove()**
sub(TAG_ABOVE,pos,sub)	**6**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**141**	**FB**	**ArithAbove()**
sub(pos,TAG_BELOW,sub)	**7**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**351**	**FB**	**ArithBelow()**
sub(TAG_BELOW,pos,sub)	**7**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**141**	**FB**	**ArithBelow()**
sub(pos,IMAGE,sub)	**8**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**351**	**FB**	**ArithImage()**
sub(IMAGE,pos,sub)	**8**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**141**	**FB**	**ArithImage()**

swap_add_nc(op1, op2, sub)

- **Operação:** Guarda no registo CY a soma de 2 operandos especificados por **op1** e **op2**. No registo TAG guarda-se o transporte desta soma e em MS o resultado da operação de *match* entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET.
- **Parâmetros:** Iguais aos da instrução **add_nc( )**
- **Operação do PBL:** pbl_match( )
- **Operação dos PEs:**
  - ◊ swap_above( ) ← se **op1** ou **op2** for TAG_ABOVE
  - ◊ swap_below( ) ← se **op1** ou **op2** for TAG_BELOW
  - ◊ swap_image( ) ← se **op1** ou **op2** for IMAGE
  - ◊ swap_this( ) ← outros casos
- **Resultados:**
  - ◊ CY = CY_IN ^ (OP0 ^ OP1)
  - ◊ TAG = ((OP0 ^ OP1) & CY_IN) | (OP0 & OP1)

- **Microcódigo** (conjunto completo de alternativas)**:**

swap_add_nc("0", "0",sub)	**9**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**334**	**FB**	**SwapThis()**
swap_add_nc("0", "1",sub)	**9**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**314**	**FB**	**SwapThis()**
swap_add_nc("1", "0",sub)	**9**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**314**	**FB**	**SwapThis()**
swap_add_nc("1", "1",sub)	**9**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**114**	**FB**	**SwapThis()**
swap_add_nc("0",pos,sub)	**9**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**314**	**FB**	**SwapThis()**
swap_add_nc(pos, "0",sub)	**9**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**314**	**FB**	**SwapThis()**
swap_add_nc("1",pos,sub)	**9**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**114**	**FB**	**SwapThis()**
swap_add_nc(pos, "1",sub)	**9**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**114**	**FB**	**SwapThis()**
swap_add_nc(pos1,pos2,sub)	**9**	**1**	**pos1**	**x...x0**	**pos2**	**x...x1**	**sub**	**314**	**FB**	**SwapThis()**
swap_add_nc(pos,TAG,sub)	**9**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**SwapThis()**
swap_add_nc(TAG,pos,sub)	**9**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**SwapThis()**
swap_add_nc(pos,TAG_ABOVE,sub)	**10**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**SwapAbove()**
swap_add_nc(TAG_ABOVE,pos,sub)	**10**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**SwapAbove()**
swap_add_nc(pos,TAG_BELOW,sub)	**11**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**SwapBelow()**
swap_add_nc(TAG_BELOW,pos,sub)	**11**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**SwapBelow()**
swap_add_nc(pos,IMAGE,sub)	**12**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**SwapImage()**
swap_add_nc(IMAGE,pos,sub)	**12**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**SwapImage()**

swap_add(op1, op2, sub)

- **Operação:** A soma entre o valor especificado por **op1,** o valor especificado por **op2** e conteúdo de CY é guardado em CY. No registo TAG guarda-se o transporte desta soma e em MS o resultado da operação de *match* entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET.
- **Parâmetros:** Iguais aos da instrução **add_nc( )**
- **Operação do PBL:** Igual à da instrução **swap_add_nc( )**
- **Operação dos PEs:** Iguais às da instrução **swap_add_nc( )**
- **Resultados:** Usam as mesmas fórmulas da instrução **swap_add_nc( )**
- **Microcódigo** (conjunto completo de alternativas)**:**

swap_add("0", "0",sub)	**9**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**331**	**FB**	**SwapThis()**
swap_add("0", "1",sub)	**9**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**311**	**FB**	**SwapThis()**
swap_add("1", "0",sub)	**9**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**311**	**FB**	**SwapThis()**
swap_add("1", "1",sub)	**9**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**111**	**FB**	**SwapThis()**
swap_add("0",pos,sub)	**9**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**311**	**FB**	**SwapThis()**
swap_add(pos, "0",sub)	**9**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**311**	**FB**	**SwapThis()**
swap_add("1",pos,sub)	**9**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**111**	**FB**	**SwapThis()**
swap_add(pos, "1",sub)	**9**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**111**	**FB**	**SwapThis()**
swap_add(pos1,pos2,sub)	**9**	**1**	**pos1**	**x...x0**	**pos2**	**x...x1**	**sub**	**311**	**FB**	**SwapThis()**
swap_add(pos,TAG,sub)	**9**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**341**	**FB**	**SwapThis()**
swap_add(TAG,pos,sub)	**9**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**341**	**FB**	**SwapThis()**
swap_add(pos,TAG_ABOVE,sub)	**10**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**341**	**FB**	**SwapAbove()**
swap_add(TAG_ABOVE,pos,sub)	**10**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**341**	**FB**	**SwapAbove()**
swap_add(pos,TAG_BELOW,sub)	**11**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**341**	**FB**	**SwapBelow()**
swap_add(TAG_BELOW,pos,sub)	**11**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**341**	**FB**	**SwapBelow()**
swap_add(pos,IMAGE,sub)	**12**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**341**	**FB**	**SwapImage()**
swap_add(IMAGE,pos,sub)	**12**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**341**	**FB**	**SwapImage()**

swap_sub_nc(op1, op2, sub)

- **Operação:** Guarda nos registos CY o resultado da subtracção entre 2 operandos especificados por **op1** e **op2**. Nas TAGs guarda-se o transporte desta subtracção e em MS o resultado da operação de *match* entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET. Para se obter um resultado correcto em complemento para 2, inverte-se **op2** e soma-se "1". Deste modo resulta que:
  CY = <op1> ⊕ NOT(< op2>) ⊕ 1
- **Parâmetros:** Iguais aos da instrução **add_nc( )**
- **Operação do PBL:** Igual à da instrução **swap_add_nc( )**
- **Operação dos PEs:** Iguais às da instrução **swap_add_nc( )**
- **Resultados:** Usam as mesmas fórmulas da instrução **swap_add_nc( )**
- **Microcódigo** (conjunto completo de alternativas)**:**

swap_sub_nc("0", "0",sub)	**9**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**313**	**FB**	**SwapThis()**
swap_sub_nc("0", "1",sub)	**9**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**333**	**FB**	**SwapThis()**
swap_sub_nc("1", "0",sub)	**9**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**113**	**FB**	**SwapThis()**
swap_sub_nc("1", "1",sub)	**9**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**133**	**FB**	**SwapThis()**
swap_sub_nc("0",pos,sub)	**9**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**333**	**FB**	**SwapThis()**
swap_sub_nc(pos, "0",sub)	**9**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**313**	**FB**	**SwapThis()**
swap_sub_nc("1",pos,sub)	**9**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**133**	**FB**	**SwapThis()**
swap_sub_nc(pos, "1",sub)	**9**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**333**	**FB**	**SwapThis()**
swap_sub_nc(pos1,pos2,sub)	**9**	**1**	**pos1**	**x...x0**	**pos2**	**x...x1**	**sub**	**333**	**FB**	**SwapThis()**
swap_sub_nc(pos,TAG,sub)	**9**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**353**	**FB**	**SwapThis()**
swap_sub_nc(TAG,pos,sub)	**9**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**143**	**FB**	**SwapThis()**
swap_sub_nc(pos,TAG_ABOVE,sub)	**10**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**353**	**FB**	**SwapAbove()**
swap_sub_nc(TAG_ABOVE,pos,sub)	**10**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**143**	**FB**	**SwapAbove()**
swap_sub_nc(pos,TAG_BELOW,sub)	**11**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**353**	**FB**	**SwapBelow()**
swap_sub_nc(TAG_BELOW,pos,sub)	**11**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**143**	**FB**	**SwapBelow()**
swap_sub_nc(pos,IMAGE,sub)	**12**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**353**	**FB**	**SwapImage()**
swap_sub_nc(IMAGE,pos,sub)	**12**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**143**	**FB**	**SwapImage()**

swap_sub(op1, op2, sub)

- **Operação:** Os registos CY guardam o resultado da subtracção entre os 2 operandos especificados por **op1** e **op2**, tendo em conta o transporte existente em CY. Nas TAGs guarda-se o transporte desta subtracção e em MS o resultado da operação de *match* entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET. Utilizando notação em complemento para 2 resulta que:
  CY = <op1> ⊕ NOT(< op2>) ⊕ CY
- **Parâmetros:** Iguais aos da instrução **add_nc( )**
- **Operação do PBL:** Igual à da instrução **swap_add_nc( )**
- **Operação dos PEs:** Iguais às da instrução **swap_add_nc( )**
- **Resultados:** Usam as mesmas fórmulas da instrução **swap_add_nc( )**

- **Microcódigo** (conjunto completo de alternativas)**:**

swap_sub("0", "0",sub)	**9**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**311**	**FB**	**ArithThis()**
swap_sub("0", "1",sub)	**9**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**331**	**FB**	**ArithThis()**
swap_sub("1", "0",sub)	**9**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**111**	**FB**	**ArithThis()**
swap_sub("1", "1",sub)	**9**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**131**	**FB**	**ArithThis()**
swap_sub("0",pos,sub)	**9**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**331**	**FB**	**ArithThis()**
swap_sub(pos, "0",sub)	**9**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**311**	**FB**	**ArithThis()**
swap_sub("1",pos,sub)	**9**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**131**	**FB**	**ArithThis()**
swap_sub(pos, "1",sub)	**9**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**331**	**FB**	**ArithThis()**
swap_sub(pos1,pos2,sub)	**9**	**1**	**pos1**	**x...x0**	**pos2**	**x...x1**	**sub**	**331**	**FB**	**ArithThis()**
swap_sub(pos,TAG,sub)	**9**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**351**	**FB**	**ArithThis()**
swap_sub(TAG,pos,sub)	**9**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**141**	**FB**	**ArithThis()**
swap_sub(pos,TAG_ABOVE,sub)	**10**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**351**	**FB**	**ArithAbove()**
swap_sub(TAG_ABOVE,pos,sub)	**10**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**141**	**FB**	**ArithAbove()**
swap_sub(pos,TAG_BELOW,sub)	**11**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**351**	**FB**	**ArithBelow()**
swap_sub(TAG_BELOW,pos,sub)	**11**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**141**	**FB**	**ArithBelow()**
swap_sub(pos,IMAGE,sub)	**12**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**351**	**FB**	**ArithImage()**
swap_sub(IMAGE,pos,sub)	**12**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**141**	**FB**	**ArithImage()**

Instruções para Cálculo do Bit de Sinal

sign_add(op1, op2, sub)

- **Operação:** Efectua a soma em complemento para 2 entre 3 parcelas: os 2 operandos com sinal especificados por **op1** e **op2** e o registo CY. Guarda-se o resultado da soma nas TAGs e nos registos CY (com MS=1) o *overflow* desta soma. CY fica a 0 nos casos em que MS é 0. MS tem o resultado da operação de *match* entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET.
- **Parâmetros:** Iguais aos da instrução **add_nc( )**
- **Operação do PBL:** pbl_match( )
- **Operação dos PEs:**
  - ◊ overf_above( ) ← se **op1** ou **op2** for TAG_ABOVE
  - ◊ overf_below( ) ← se **op1** ou **op2** for TAG_BELOW
  - ◊ overf_image( ) ← se **op1** ou **op2** for IMAGE
  - ◊ overf_this( ) ← outros casos
- **Resultados:**
  - ◊ TAG = CY_IN ^ (OP0 ^ OP1)
  - ◊ CY = NOT(OP0 ^ OP1) & (CY_IN ^ OP0) & MS

- **Microcódigo** (conjunto completo de alternativas)**:**

sign_add("0", "0",sub)	**15**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**311**	**FB**	**OverfThis()**
sign_add("0", "1",sub)	**15**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**311**	**FB**	**OverfThis()**
sign_add("1", "0",sub)	**15**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**311**	**FB**	**OverfThis()**
sign_add("1", "1",sub)	**15**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**111**	**FB**	**OverfThis()**
sign_add("0",pos,sub)	**15**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**311**	**FB**	**OverfThis()**
sign_add(pos, "0",sub)	**15**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**311**	**FB**	**OverfThis()**
sign_add("1",pos,sub)	**15**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**111**	**FB**	**OverfThis()**
sign_add(pos, "1",sub)	**15**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**111**	**FB**	**OverfThis()**
sign_add(pos1,pos2,sub)	**15**	**1**	**pos1**	**x...x0**	**pos2**	**x...x1**	**sub**	**311**	**FB**	**OverfThis()**
sign_add(pos,TAG,sub)	**15**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**341**	**FB**	**OverfThis()**
sign_add(TAG,pos,sub)	**15**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**341**	**FB**	**OverfThis()**
sign_add(pos,TAG_ABOVE,sub)	**16**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**341**	**FB**	**OverfAbove()**
sign_add(TAG_ABOVE,pos,sub)	**16**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**341**	**FB**	**OverfAbove()**
sign_add(pos,TAG_BELOW,sub)	**17**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**341**	**FB**	**OverfBelow()**
sign_add(TAG_BELOW,pos,sub)	**17**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**341**	**FB**	**OverfBelow()**
sign_add(pos,IMAGE,sub)	**18**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**341**	**FB**	**OverfImage()**
sign_add(IMAGE,pos,sub)	**18**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**341**	**FB**	**OverfImage()**

sign_sub(op1, op2, sub)

- **Operação:** Efectua a subtracção em complemento para 2 entre os 2 operandos com sinal especificados por **op1** e **op2** e tendo em atenção o conteúdo do registo CY. Guarda-se o resultado da subtracção nas TAGs. Nos PEs em que MS está a 1, o *overflow* da subtracção é colocado em CY e nos PEs em que MS está a 0, CY fica com 0. MS tem o resultado da operação de *match* entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET. Desta instrução resulta.
  TAG = <op1> ⊕ NOT(< op2>) ⊕ CY
- **Parâmetros:** Iguais aos da instrução **add_nc( )**
- **Operação do PBL:** Igual à da instrução **sign_add( )**
- **Operação dos PEs:** Iguais às da instrução **sign_add( )**
- **Resultados:** Usam as mesmas fórmulas da instrução **sign_add( )**
- **Microcódigo** (conjunto completo de alternativas)**:**

sign_sub("0", "0",sub)	**15**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**311**	**FB**	**OverfThis()**
sign_sub("0", "1",sub)	**15**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**331**	**FB**	**OverfThis()**
sign_sub("1", "0",sub)	**15**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**111**	**FB**	**OverfThis()**
sign_sub("1", "1",sub)	**15**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**131**	**FB**	**OverfThis()**
sign_sub("0",pos,sub)	**15**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**331**	**FB**	**OverfThis()**
sign_sub(pos, "0",sub)	**15**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**311**	**FB**	**OverfThis()**
sign_sub("1",pos,sub)	**15**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**131**	**FB**	**OverfThis()**
sign_sub(pos, "1",sub)	**15**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**331**	**FB**	**OverfThis()**
sign_sub(pos1,pos2,sub)	**15**	**1**	**pos1**	**x...x0**	**pos2**	**x...x1**	**sub**	**331**	**FB**	**OverfThis()**
sign_sub(pos,TAG,sub)	**15**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**351**	**FB**	**OverfThis()**
sign_sub(TAG,pos,sub)	**15**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**141**	**FB**	**OverfThis()**
sign_sub(pos,TAG_ABOVE,sub)	**16**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**351**	**FB**	**OverfAbove()**
sign_sub(TAG_ABOVE,pos,sub)	**16**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**141**	**FB**	**OverfAbove()**
sign_sub(pos,TAG_BELOW,sub)	**17**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**351**	**FB**	**OverfBelow()**
sign_sub(TAG_BELOW,pos,sub)	**17**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**141**	**FB**	**OverfBelow()**
sign_sub(pos,IMAGE,sub)	**18**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**351**	**FB**	**OverfImage()**
sign_sub(IMAGE,pos,sub)	**18**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**141**	**FB**	**OverfImage()**

Instruções para Comparação de Magnitude

compare_data(op1, op2, sub)

- **Operação:** Efectua a comparação entre 2 operando especificados por **op1** e **op2**, e posteriormente faz o OR do resultado da comparação com o conteúdo das TAGs. Nos registos CY guarda-se o transporte da comparação (subtracção) e em MS o resultado da operação de *match* entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET. Após execução da comparação:
  - ◊ TAG=0 indica que <op1> = <op2>
  - ◊ se TAG=1, CY=0 indica que <op1> < <op2>
- **Parâmetros:**
  - ◊ **op1, op2** :
    - "0" ← operando = 0
    - "1" ← operando = 1
    - posição na CAM [0:63] ← operando = conteúdo da coluna da CAM
    - TAG ← operando = TAGs dos
    - TAG_ABOVE ← operando = TAGs após *shift* para cima **(*)**
    - TAG_BELOW ← operando = TAGs após *shift* para baixo **(*)**
    - IMAGE ← operando = coluna ls do VSR (*)
  - ◊ **sub** → padrão de subset com 4 dígitos ternários (0|1|x)
- **Operação do PBL:** pbl_match( )
- **Operação dos PEs:**
  - ◊ cpdata_above( ) ← se **op1** ou **op2** for TAG_ABOVE (†)
  - ◊ cpdata_below( ) ← se **op1** ou **op2** for TAG_BELOW (†)
  - ◊ cpdata_image( ) ← se **op1** ou **op2** for IMAGE (†)
  - ◊ cpdata_this( ) ← outros casos
- **Resultados:**
  - ◊ TAG = TAG | (CY_IN ^ OP0 ^ OP1)
  - ◊ CY = ((OP0 ^ OP1) & CY_IN) | (OP0 ^ OP1)
- **Microcódigo** (conjunto completo de alternativas)**:**

compare_data("0", "0",sub)	**20**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**311**	**FB**	**CpdataThis()**
compare_data("0", "1",sub)	**20**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**331**	**FB**	**CpdataThis()**
compare_data("1", "0",sub)	**20**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**111**	**FB**	**CpdataThis()**
compare_data("1", "1",sub)	**20**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**131**	**FB**	**CpdataThis()**
compare_data("0",pos,sub)	**20**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**331**	**FB**	**CpdataThis()**
compare_data(pos, "0",sub)	**20**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**311**	**FB**	**CpdataThis()**
compare_data("1",pos,sub)	**20**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**131**	**FB**	**CpdataThis()**
compare_data(pos, "1",sub)	**20**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**331**	**FB**	**CpdataThis()**
compare_data(pos1,pos2,sub)	**20**	**1**	**pos1**	**x...x0**	**pos2**	**x...x1**	**sub**	**331**	**FB**	**CpdataThis()**
compare_data(pos,TAG,sub)	**20**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**351**	**FB**	**CpdataThis()**
compare_data(TAG,pos,sub)	**20**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**141**	**FB**	**CpdataThis()**
compare_data(pos,TAG_ABOVE,sub) (‡)	**21**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**351**	**FB**	**CpdataAbove()**
compare_data(TAG_ABOVE,pos,sub) (‡)	**21**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**141**	**FB**	**CpdataAbove()**
compare_data(pos,TAG_BELOW,sub)(‡)	**22**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**351**	**FB**	**CpdataBelow()**
compare_data(TAG_BELOW,pos,sub)(‡)	**22**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**141**	**FB**	**CpdataBelow()**
compare_data(pos,IMAGE,sub) (‡)	**23**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**351**	**FB**	**CpdataImage()**
compare_data(IMAGE,pos,sub) (‡)	**23**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**141**	**FB**	**CpdataImage()**

- **Notas:**
  (*) Estes parâmetros não podem ser utilizados na versão actual do emulador.
  (†) Estas funções não funcionam na versão actual do emulador.
  (‡) Estas instruções não estão disponíveis na versão actual do emulador.

compare_sign(op1, op2, sub)

- **Operação:** Esta instrução é utilizada para comparar o bit de sinal de 2 grupos de bits (previamente comparados com várias instruções compare_data()), permitindo concluir qual dos operandos é maior/menor. Na comparação de um valor sem sinal com outro com sinal, quando se faz a última comparação de bits com compare_sign(), o operando sem sinal a passar a esta instrução é "0". Após executar compare_sign():
  ◊ TAG=0 indica que <op1> = <op2>
  ◊ se TAG=1, CY=1 indica que <op1> < <op2>
- **Parâmetros:** Iguais aos da instrução **compare_data( )**
- **Operação do PBL:** pbl_match( )
- **Operação dos PEs:**
  ◊ cpsign_above( ) ← se **op1** ou **op2** for TAG_ABOVE (*)
  ◊ cpsign_below( ) ← se **op1** ou **op2** for TAG_BELOW (*)
  ◊ cpsign_image( ) ← se **op1** ou **op2** for IMAGE (*)
  ◊ cpsign_this( ) ← outros casos
- **Resultados:**
  ◊ TAG = TAG | (CY_IN ^ OP0 ^ OP1)
  ◊ CY = CY ^ (OP0 ^ OP1)
- **Microcódigo** (conjunto completo de alternativas)**:**

compare_sign("0", "0",sub)	**24**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**311**	**FB**	**CpsignThis()**
compare_sign("0", "1",sub)	**24**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**331**	**FB**	**CpsignThis()**
compare_sign("1", "0",sub)	**24**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**111**	**FB**	**CpsignThis()**
compare_sign("1", "1",sub)	**24**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**131**	**FB**	**CpsignThis()**
compare_sign("0",pos,sub)	**24**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**331**	**FB**	**CpsignThis()**
compare_sign(pos, "0",sub)	**24**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**311**	**FB**	**CpsignThis()**
compare_sign("1",pos,sub)	**24**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**131**	**FB**	**CpsignThis()**
compare_sign(pos, "1",sub)	**24**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**331**	**FB**	**CpsignThis()**
compare_sign(pos1,pos2,sub)	**24**	**1**	**pos1**	**x...x0**	**pos2**	**x...x1**	**sub**	**331**	**FB**	**CpsignThis()**
compare_sign(pos,TAG,sub)	**24**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**351**	**FB**	**CpsignThis()**
compare_sign(TAG,pos,sub)	**24**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**141**	**FB**	**CpsignThis()**
compare_sign(pos,TAG_ABOVE,sub) (†)	**25**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**351**	**FB**	**CpsignAbove()**
compare_sign(TAG_ABOVE,pos,sub) (†)	**25**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**141**	**FB**	**CpsignAbove()**
compare_sign(pos,TAG_BELOW,sub)(†)	**26**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**351**	**FB**	**CpsignBelow()**
compare_sign(TAG_BELOW,pos,sub)(†)	**26**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**141**	**FB**	**CpsignBelow()**
compare_sign(pos,IMAGE,sub) (†)	**27**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**351**	**FB**	**CpsignImage()**
compare_sign(IMAGE,pos,sub) (†)	**27**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**141**	**FB**	**CpsignImage()**

- **Exemplo:** Mostra como se utiliza compare_data() e compare_sign() para comparar 1 campo (sem sinal) guardado nos bits 0:3 da CAM com outro campo (com sinal) guardado nos bits 5:9 da CAM.

```
sub_nc(0,5,"xxxx");
compare_data(1,6, "xxxx");
compare_data(2,7, "xxxx");
compare_data(3,8, "xxxx");
compare_sign("0",9, "xxxx");
```

- **Notas:**
  (*) Estas funções não funcionam na versão actual do emulador.
  (†) Estas instruções não estão disponíveis na versão actual do emulador.

Instruções com Operações Lógicas

tag_copy(op1,sub)

- **Operação:** Copia o operando especificado por **op1** para as TAGs. Os registos CY ficam inalterados e MS guarda o resultado da operação de *match* entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET.
- **Parâmetros:**
  ◊ **op1**:
  ⎧ "0" ← operando1 = 0
  ⎨ "1" ← operando1 = 1
  ⎩ posição na CAM [0:63] ← operando1 = conteúdo da coluna da CAM
  ◊ **sub** → padrão de subset com 4 dígitos ternários (0|1|x)
- **Operação do PBL:** pbl_match( )
- **Operação dos PEs:** logic_this( )
- **Resultados:**
  ◊ TAG = <op1>
  ◊ CY = CY
  ◊ MS → é afectado do mesmo modo que em **match( )**
- **Microcódigo:**

tag_copy("0",sub)	**28**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**134**	**FB**	**LogicThis()**
tag_copy("1",sub)	**28**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**114**	**FB**	**LogicThis()**
tag_copy(pos,sub)	**28**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**114**	**FB**	**LogicThis()**

invert(op1,sub)

- **Operação:** Inverte o operando especificado por **op1** e guarda o resultado nas TAGs. Os registos CY ficam inalterados e MS guarda o resultado da operação de *match* entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET.

- **Parâmetros:**
  - ◊ **op1**:
    - "0" ← operando1 = 0
    - "1" ← operando1 = 1
    - posição na CAM [0:63] ← operando1 = conteúdo da coluna da CAM
    - TAG ← operando1 = TAGs dos
    - TAG_ABOVE ← operando1 = TAGs após *shift* para cima
    - TAG_BELOW ← operando1 = TAGs após *shift* para baixo
    - IMAGE ← operando1 = coluna ls do VSR
  - ◊ **sub** → padrão de subset com 4 dígitos ternários (0|1|x)
- **Operação do PBL:** pbl_match( )
- **Operação dos PEs:**
  - ◊ logic_above( ) ← se **op1** for TAG_ABOVE
  - ◊ logic_below( ) ← se **op1** for TAG_BELOW
  - ◊ logic_image( ) ← se **op1** for IMAGE
  - ◊ logic_this( ) ← outros casos
- **Resultados:**
  - ◊ TAG = NOT(<op1>)
  - ◊ CY = CY
  - ◊ MS → é afectado do mesmo modo que em **match( )**
- **Microcódigo:**

invert("0",sub)	**28**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**114**	**FB**	**LogicThis()**
invert("1",sub)	**28**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**134**	**FB**	**LogicThis()**
invert(pos,sub)	**28**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**134**	**FB**	**LogicThis()**
invert(TAG,sub)	**28**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**154**	**FB**	**LogicThis()**
invert(TAG_ABOVE,sub)	**29**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**154**	**FB**	**LogicAbove()**
invert(TAG_BELOW,sub)	**30**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**154**	**FB**	**LogicBelow()**
invert(IMAGE,sub)	**31**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**154**	**FB**	**LogicImage()**

and(op1,op2,sub)

- **Operação:** Calcula a operação lógica AND entre os operandos especificados por **op1** e **op2,** guardando o resultado nas TAGs. Os registos CY ficam inalterados e MS guarda o resultado da operação de *match* entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET.
- **Parâmetros:** Iguais aos da instrução **add_nc( )**
- **Operação do PBL:** Igual à da instrução **invert( )**
- **Operação dos PEs:** Iguais às da instrução **invert( )**
- **Resultados:**
  - ◊ TAG = <op1> & <op2>
  - ◊ CY = CY
  - ◊ MS → é afectado do mesmo modo que em **match( )**

- **Microcódigo** (conjunto completo de alternativas)**:**

and("0", "0",sub)	**28**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**334**	**FB**	**LogicThis()**
and("0", "1",sub)	**28**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**314**	**FB**	**LogicThis()**
and("1", "0",sub)	**28**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**314**	**FB**	**LogicThis()**
and("1", "1",sub)	**28**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**114**	**FB**	**LogicThis()**
and("0",pos,sub)	**28**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**314**	**FB**	**LogicThis()**
and(pos, "0",sub)	**28**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**314**	**FB**	**LogicThis()**
and("1",pos,sub)	**28**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**114**	**FB**	**LogicThis()**
and(pos, "1",sub)	**28**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**114**	**FB**	**LogicThis()**
and(pos1,pos2,sub)	**28**	**1**	**pos1**	**x...x0**	**pos2**	**x...x1**	**sub**	**314**	**FB**	**LogicThis()**
and(pos,TAG,sub)	**28**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**LogicThis()**
and(TAG,pos,sub)	**28**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**LogicThis()**
and(pos,TAG_ABOVE,sub)	**29**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**LogicAbove()**
and(TAG_ABOVE,pos,sub)	**29**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**LogicAbove()**
and(pos,TAG_BELOW,sub)	**30**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**LogicBelow()**
and(TAG_BELOW,pos,sub)	**30**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**LogicBelow()**
and(pos,IMAGE,sub)	**31**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**LogicImage()**
and(IMAGE,pos,sub)	**31**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**LogicImage()**

or(op1,op2,sub)

- **Operação:** Calcula a operação lógica OR entre os operandos especificados por **op1** e **op2,** guardando o resultado nas TAGs. Os registos CY ficam inalterados e MS guarda o resultado da operação de *match* entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET.
- **Parâmetros:** Iguais aos da instrução **add_nc( )**
- **Operação do PBL:** Igual à da instrução **invert( )**
- **Operação dos PEs:** Iguais às da instrução **invert( )**
- **Resultados:**
  - ◊ TAG = <op1> OR <op2>
  - ◊ CY = CY
  - ◊ MS → é afectado do mesmo modo que em **match( )**
- **Microcódigo** (conjunto completo de alternativas)**:**

or("0", "0",sub)	**28**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**333**	**FB**	**LogicThis()**
or("0", "1",sub)	**28**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**313**	**FB**	**LogicThis()**
or("1", "0",sub)	**28**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**313**	**FB**	**LogicThis()**
or("1", "1",sub)	**28**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**113**	**FB**	**LogicThis()**
or("0",pos,sub)	**28**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**313**	**FB**	**LogicThis()**
or(pos, "0",sub)	**28**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**313**	**FB**	**LogicThis()**
or("1",pos,sub)	**28**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**113**	**FB**	**LogicThis()**
or(pos, "1",sub)	**28**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**113**	**FB**	**LogicThis()**
or(pos1,pos2,sub)	**28**	**1**	**pos1**	**x...x0**	**pos2**	**x...x1**	**sub**	**313**	**FB**	**LogicThis()**
or(pos,TAG,sub)	**28**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**343**	**FB**	**LogicThis()**
or(TAG,pos,sub)	**28**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**343**	**FB**	**LogicThis()**
or(pos,TAG_ABOVE,sub)	**29**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**343**	**FB**	**LogicAbove()**
or(TAG_ABOVE,pos,sub)	**29**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**343**	**FB**	**LogicAbove()**
or(pos,TAG_BELOW,sub)	**30**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**343**	**FB**	**LogicBelow()**
or(TAG_BELOW,pos,sub)	**30**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**343**	**FB**	**LogicBelow()**
or(pos,IMAGE,sub)	**31**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**343**	**FB**	**LogicImage()**
or(IMAGE,pos,sub)	**31**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**343**	**FB**	**LogicImage()**

nand(op1,op2,sub)

- **Operação:** Calcula a operação lógica NAND entre os operandos especificados por **op1** e **op2,** guardando o resultado nas TAGs. Os registos CY ficam inalterados e MS guarda o resultado da operação de *match* entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET.
- **Parâmetros:** Iguais aos da instrução **add_nc( )**
- **Operação do PBL:** Igual à da instrução **invert( )**
- **Operação dos PEs:** Iguais às da instrução **invert( )**
- **Resultados:**
  - ◊ TAG = NOT(<op1> & <op2>)
  - ◊ CY = CY
  - ◊ MS → é afectado do mesmo modo que em **match( )**
- **Microcódigo** (conjunto completo de alternativas)**:**

nand("0", "0",sub)	**28**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**113**	**FB**	**LogicThis()**
nand("0", "1",sub)	**28**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**133**	**FB**	**LogicThis()**
nand("1", "0",sub)	**28**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**133**	**FB**	**LogicThis()**
nand("1", "1",sub)	**28**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**333**	**FB**	**LogicThis()**
nand("0",pos,sub)	**28**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**133**	**FB**	**LogicThis()**
nand(pos, "0",sub)	**28**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**133**	**FB**	**LogicThis()**
nand("1",pos,sub)	**28**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**333**	**FB**	**LogicThis()**
nand(pos, "1",sub)	**28**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**333**	**FB**	**LogicThis()**
nand(pos1,pos2,sub)	**28**	**1**	**pos1**	**x...x0**	**pos2**	**x...x1**	**sub**	**133**	**FB**	**LogicThis()**
nand(pos,TAG,sub)	**28**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**153**	**FB**	**LogicThis()**
nand(TAG,pos,sub)	**28**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**153**	**FB**	**LogicThis()**
nand(pos,TAG_ABOVE,sub)	**29**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**153**	**FB**	**LogicAbove()**
nand(TAG_ABOVE,pos,sub)	**29**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**153**	**FB**	**LogicAbove()**
nand(pos,TAG_BELOW,sub)	**30**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**153**	**FB**	**LogicBelow()**
nand(TAG_BELOW,pos,sub)	**30**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**153**	**FB**	**LogicBelow()**
nand(pos,IMAGE,sub)	**31**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**153**	**FB**	**LogicImage()**
nand(IMAGE,pos,sub)	**31**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**153**	**FB**	**LogicImage()**

nor(op1,op2,sub)

- **Operação:** Calcula a operação lógica NOR entre os operandos especificados por **op1** e **op2,** guardando o resultado nas TAGs. Os registos CY ficam inalterados e MS guarda o resultado da operação de *match* entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET.
- **Parâmetros:** Iguais aos da instrução **add_nc( )**
- **Operação do PBL:** Igual à da instrução **invert( )**
- **Operação dos PEs:** Iguais às da instrução **invert( )**
- **Resultados:**
  - ◊ TAG = NOT(<op1> | <op2>)
  - ◊ CY = CY
  - ◊ MS → é afectado do mesmo modo que em **match( )**

- **Microcódigo** (conjunto completo de alternativas)**:**

nor("0", "0",sub)	**28**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**114**	**FB**	**LogicThis()**
nor("0", "1",sub)	**28**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**134**	**FB**	**LogicThis()**
nor("1", "0",sub)	**28**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**134**	**FB**	**LogicThis()**
nor("1", "1",sub)	**28**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**334**	**FB**	**LogicThis()**
nor("0",pos,sub)	**28**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**134**	**FB**	**LogicThis()**
nor(pos, "0",sub)	**28**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**134**	**FB**	**LogicThis()**
nor("1",pos,sub)	**28**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**334**	**FB**	**LogicThis()**
nor(pos, "1",sub)	**28**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**334**	**FB**	**LogicThis()**
nor(pos1,pos2,sub)	**28**	**1**	**pos1**	**x...x0**	**pos2**	**x...x1**	**sub**	**134**	**FB**	**LogicThis()**
nor(pos,TAG,sub)	**28**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**154**	**FB**	**LogicThis()**
nor(TAG,pos,sub)	**28**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**154**	**FB**	**LogicThis()**
nor(pos,TAG_ABOVE,sub)	**29**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**154**	**FB**	**LogicAbove()**
nor(TAG_ABOVE,pos,sub)	**29**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**154**	**FB**	**LogicAbove()**
nor(pos,TAG_BELOW,sub)	**30**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**154**	**FB**	**LogicBelow()**
nor(TAG_BELOW,pos,sub)	**30**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**154**	**FB**	**LogicBelow()**
nor(pos,IMAGE,sub)	**31**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**154**	**FB**	**LogicImage()**
nor(IMAGE,pos,sub)	**31**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**154**	**FB**	**LogicImage()**

xor(op1,op2,sub)

- **Operação:** Calcula a operação lógica OU-EXCLUSIVO entre os operandos especificados por **op1** e **op2,** guardando o resultado nas TAGs. Os registos CY ficam inalterados e MS guarda o resultado da operação de *match* entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET.
- **Parâmetros:** Iguais aos da instrução **add_nc( )**
- **Operação do PBL:** pbl_match( )
- **Operação dos PEs:**
  - ◊ xor_above( ) ← se **op1** ou **op2** for TAG_ABOVE
  - ◊ xor_below( ) ← se **op1** ou **op2** for TAG_BELOW
  - ◊ xor_image( ) ← se **op1** ou **op2** for IMAGE
  - ◊ xor_this( ) ← outros casos
- **Resultados:**
  - ◊ TAG = <op1> ^ <op2>
  - ◊ CY = CY
  - ◊ MS → é afectado do mesmo modo que em **match( )**

- **Microcódigo** (conjunto completo de alternativas)**:**

xor("0", "0",sub)	**32**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**334**	**FB**	**XorThis()**
xor("0", "1",sub)	**32**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**314**	**FB**	**XorThis()**
xor("1", "0",sub)	**32**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**314**	**FB**	**XorThis()**
xor("1", "1",sub)	**32**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**114**	**FB**	**XorThis()**
xor("0",pos,sub)	**32**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**314**	**FB**	**XorThis()**
xor(pos, "0",sub)	**32**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**314**	**FB**	**XorThis()**
xor("1",pos,sub)	**32**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**114**	**FB**	**XorThis()**
xor(pos, "1",sub)	**32**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**114**	**FB**	**XorThis()**
xor(pos1,pos2,sub)	**32**	**1**	**pos1**	**x...x0**	**pos2**	**x...x1**	**sub**	**314**	**FB**	**XorThis()**
xor(pos,TAG,sub)	**32**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**XorThis()**
xor(TAG,pos,sub)	**32**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**XorThis()**
xor(pos,TAG_ABOVE,sub)	**33**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**XorAbove()**
xor(TAG_ABOVE,pos,sub)	**33**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**XorAbove()**
xor(pos,TAG_BELOW,sub)	**34**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**XorBelow()**
xor(TAG_BELOW,pos,sub)	**34**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**XorBelow()**
xor(pos,IMAGE,sub)	**35**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**XorImage()**
xor(IMAGE,pos,sub)	**35**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**344**	**FB**	**XorImage()**

equal(op1,op2,sub)

- **Operação:** Guarda na TAG de cada PE o resultado da determinação de igualdade entre o bit correspondente dos operandos especificados por **op1** e **op2**. Para cada PE, se o bit correspondente de <op1> for igual ao bit de <op2> TAG=1, senão TAG=0. Os registos CY ficam inalterados e MS guarda o resultado da operação de *match* entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET.
- **Parâmetros:** Iguais aos da instrução **add_nc( )**
- **Operação do PBL:** Igual à da instrução **xor( )**
- **Operação dos PEs:** Iguais às da instrução **xor( )**
- **Resultados:**
  - ◊ TAG = NOT(<op1> ^ <op2>)
  - ◊ CY = CY
  - ◊ MS → é afectado do mesmo modo que em **match( )**

Microcódigo (conjunto completo de alternativas)**:**

equal("0", "0",sub)	**32**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**333**	**FB**	**XorThis()**
equal("0", "1",sub)	**32**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**313**	**FB**	**XorThis()**
equal("1", "0",sub)	**32**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**313**	**FB**	**XorThis()**
equal("1", "1",sub)	**32**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**113**	**FB**	**XorThis()**
equal("0",pos,sub)	**32**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**313**	**FB**	**XorThis()**
equal(pos, "0",sub)	**32**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**313**	**FB**	**XorThis()**
equal("1",pos,sub)	**32**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**113**	**FB**	**XorThis()**
equal(pos, "1",sub)	**32**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**113**	**FB**	**XorThis()**
equal(pos1,pos2,sub)	**32**	**1**	**pos1**	**x...x0**	**pos2**	**x...x1**	**sub**	**313**	**FB**	**XorThis()**
equal(pos,TAG,sub)	**32**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**343**	**FB**	**XorThis()**
equal(TAG,pos,sub)	**32**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**343**	**FB**	**XorThis()**
equal(pos,TAG_ABOVE,sub)	**33**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**343**	**FB**	**XorAbove()**
equal(TAG_ABOVE,pos,sub)	**33**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**343**	**FB**	**XorAbove()**
equal(pos,TAG_BELOW,sub)	**34**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**343**	**FB**	**XorBelow()**
equal(TAG_BELOW,pos,sub)	**34**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**343**	**FB**	**XorBelow()**
equal(pos,IMAGE,sub)	**35**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**343**	**FB**	**XorImage()**
equal(IMAGE,pos,sub)	**35**	**1**	**pos**	**x...x0**	**0**	**x...x**	**sub**	**343**	**FB**	**XorImage()**

cy_and_equal(op1,sub)

- **Operação:** Determina a igualdade, tal como definido em **equal( )**, entre o operando especificado por **op1** e o conteúdo de CY, guardando o resultado nas TAGs. Nos registos CY fica o transporte da soma entre CY e o operando especificado por **op1**. Os registos MS guardam o resultado da operação de *match* entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET.
- **Parâmetros:**
  - ◊ **op1**:
    - "0" ← operando1 = 0
    - "1" ← operando1 = 1
    - posição na CAM [0:63] ← operando1 = conteúdo da coluna da CAM
    - TAG ← operando1 = TAGs dos
    - TAG_ABOVE ← operando1 = TAGs após *shift* para cima
    - TAG_BELOW ← operando1 = TAGs após *shift* para baixo
    - IMAGE ← operando1 = coluna ls do VSR
  - ◊ **sub** → padrão de subset com 4 dígitos ternários (0|1|x)
- **Operação do PBL:** Igual à da instrução **sign_add( )**
- **Operação dos PEs:** Iguais às da instrução **sign_add( )**
- **Resultados:**
  - ◊ TAG = NOT(<op1>) ⊕ CY
  - ◊ CY = <op1> & CY
  - ◊ MS → é afectado do mesmo modo que em **match( )**
- **Microcódigo:**

cy_and_equal("0",sub)	**15**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**311**	**FB**	**OverfThis()**
cy_and_equal("1",sub)	**15**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**331**	**FB**	**OverfThis()**
cy_and_equal(pos,sub)	**15**	**1**	**0**	**x...x**	**pos**	**x...x1**	**sub**	**331**	**FB**	**OverfThis()**
cy_and_equal(TAG,sub)	**15**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**351**	**FB**	**OverfThis()**
cy_and_equal(TAG_ABOVE,sub)	**16**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**351**	**FB**	**OverfAbove()**
cy_and_equal(TAG_BELOW,sub)	**17**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**351**	**FB**	**OverfBelow()**
cy_and_equal(IMAGE,sub)	**18**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**351**	**FB**	**OverfImage()**

Instruções para Deslocamento de TAGs

shift(dist,tipo)

- **Operação:** Desloca as TAGs entre PEs de uma distância dada por **dist**. O parâmetro **tipo** indica o que é inserido no início da cadeia de PEs. Se **dist** for **positivo** o deslocamento das TAGs é no sentido do PE mais significativo para o menos significativo, e se for negativo TAGs é no sentido do PE menos significativo para o mais significativo. Se **tipo** for **ZERO/UM** insere-se "0/1" e ser for **CICLICO** insere-se a *flag* TAG_ABOVE ou TAG_BELOW (consoante o sinal de **dist**).
- **Parâmetros:**
  - ◊ **dist** →valor no intervalo ±[0:63] (tamanho do deslocamento das TAGs)
  - ◊ **tipo** → {**ZERO, UM**, **CICLICO}** (insere-se "0"/ insere-se"1"/é cíclico)

- **Operação do PBL:** pbl_nop( )
- **Operação dos PEs:**
  - ◊ shift_up( ) ← se **dist** < 0
  - ◊ shift_down( ) ← se **dist** ≥ 0
- **Microcódigo:**

shift(dist,tipo)	**(dist < 0)**	**36**	**0**	**dist**	**x...x**	**tipo**	**x...x**	**xxxx**	**222**	**FB**	**ShiftUp()**
shift(dist,tipo)	**(dist ≥ 0)**	**37**	**0**	**dist**	**x...x**	**tipo**	**x...x**	**xxxx**	**222**	**FB**	**ShiftDown()**

- **Notas:** A passagem dos parâmetros **dist** e **tipo** de shift() no microcódigo de shift_up() e shift_down() não está de acordo com o objectivo dos campos onde são colocados. Optou-se por esta estratégia por facilitar a implementação destas microinstruções e permitir testá-las sem dispor de um controlador exterior ao GLiTCH. Uma estratégia para contornar esta opção, a qual exige controlo externo ao GLiTCH é a seguinte:
  - ◊ **dist** → o problema é resolvido gerando em vez de uma microinstrução que faz o deslocamento de **dist** posições, **dist** microinstruções em que cada uma faz o deslocamento de uma posição.
  - ◊ **tipo** → o problema é resolvido se, antes de executar a instrução, se alterar a TAG_ABOVE ou TAG_BELOW (consoante o sinal de **dist**) de acordo com o valor de **tipo**: se tipo=**ZERO** coloca-se a *flag* a 0**,** se tipo=**UM** coloca-se a *flag* a 1, se tipo=**CICLICO** não se faz nada (usa-se o conteúdo da *flag*).

barrel_shift(dist)

- **Operação:** Utiliza-se o mecanismo de *barrel* para deslocar as TAGs, dentro do mesmo *chip*, de uma valor que é múltiplo de 4 (**dist**).
- **Parâmetros:**
  - ◊ **dist** → {-60, -56, -52, ... , -8, -4, 0, +4, +8, ... , +52, +56, +60}
    (tamanho do deslocamento das TAGs)
- **Operação do PBL:** (i) pbl_nop( ) (ii) pbl_read2( ) (iii) pbl_nop( )
- **Operação dos PEs:** (i) extract_tag( ) (ii) nop( ) (iii) restore_tag( )
- **Microcódigo:**

38	**0**	**dist**	**x...x**	**0**	**x...x**	**xxxx**	**222**	**FB ExtractTag()**
0	**3**	**dist**	**x...x**	**0**	**x...x**	**xxxx**	**222**	**FB PblRead2()**
39	**0**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**222**	**FB RestoreTag()**

- **Notas:** Ver a Nota da instrução **shift( )**.

extract_tags(dist)

- **Operação:** Copia as 64 TAGs para o registo PBLR, aplicando uma rotação de TAGs de um valor dado por **dist**. Por exemplo, com uma rotação R obtém-se: PBLR[63]=TAG[R-1], PBLR[62]=TAG[R-2], ... , PBLR[63-R+1]=TAG[0], PBLR[63-R]=TAG[63], ... , PBLR[1]=TAG[R+1], PBLR[0]=TAG[R]
- **Parâmetros:**
  - ◊ **dist** → {-60, -56, -52, ... , -8, -4, 0, +4, +8, ... , +52, +56, +60}
    (tamanho da rotação das TAGs)

- **Operação do PBL:** (i) pbl_nop( ) (ii) pbl_read2( )
- **Operação dos PEs:** (i) extract_tag( ) (ii) nop( )
- **Microcódigo:**

  38 0 dist x...x 0 x...x xxxx 222 FB ExtractTag()
  0 3 dist x...x 0 x...x xxxx 222 FB PblRead2()

restore_tags( )

- **Operação:** Copia 64 bits do PBLR para as TAGs, de acordo com: TAG[0]=PBLR[0], TAG[1]=PBLR[1], ... , TAG[63]=PBLR[63]
- **Parâmetros:** Não tem.
- **Operação do PBL:** pbl_nop( )
- **Operação dos PEs:** restore_tag( )
- **Microcódigo:**

  39 0 0 x...x 0 x...x xxxx 222 FB RestoreTag()

merge_tags(pos_no_pbl,mascara)

- **Operação:** Combina no PBLR grupos de 4 bits a partir do caminho de 32 bits que liga os vários GLiTCHs com os bits do PBLR. O 1º parâmetro controla se se escreve nos 32 bits mais/menos significativos do PBLR e cada caractere a "0"/"1" no 2º parâmetro define se escreve/não_escreve cada grupo de 4 bits no PBLR.
- **Parâmetros:**
  - ◊ **pos_no_pbl** → {LEFT, RIGHT} (metade mais/menos significativa do PBLR)
  - ◊ **mascara** → *string* com 8 caracteres 0/1 (1 - escreve no PBLR, 0 - não escreve no PBLR)
- **Operação do PBL:**
  - ◊ pbl_mergeR( ) ← se **pos_no_pbl** = RIGHT
  - ◊ pbl_mergeL( ) ← se **pos_no_pbl** = LEFT
- **Operação dos PEs:** nop( )
- **Microcódigo:**

merge_tags(RIGHT,mask)	**0**	**5**	**0**	**mask**	**0**	**x...x**	**xxxx**	**222**	**FB**	**PblMergeR()**
merge_tags(LEFT,mask)	**0**	**6**	**0**	**mask**	**0**	**x...x**	**xxxx**	**222**	**FB**	**PblMergeL()**

- **Notas:** A correspondência entre cada caractere do parâmetro **mascara** e o grupo de bits que controla no PBLR é a seguinte:

  mascara[0] controla o grupo de bits [0:3]
  mascara[1] controla o grupo de bits [4:7]
  ...
  mascara[7] controla o grupo de bits [28:31]

bounce_tags(pos_no_pbl,num_chips)

- **Operação:** Desloca metade do PBLR para um *chip* distante **num_chips** *chips* do actual. Do ponto de vista de cada GLiTCH, o conteúdo do PBLR só é colocado no exterior desse *chip*, o resto deverá ser feito pelo controlador do *chip*. Esta informação será depois lida por um dos 2 *chips* vizinhos e enviada por sua vez para outro *chip* até chegar ao *chip* destino. Se o mecanismo exterior de encaminhamento permitir que 2 *chips* não vizinhos comuniquem directamente, então não é necessário passar a informação por todos os *chips* que medeiam entre a origem e o destino.
- **Parâmetros:**
  - ◊ **pos_no_pbl** → {LEFT, RIGHT} (metade mais/menos significativa do PBLR)
  - ◊ **num_chips** → nº de *chips* de distância ao *chip* destino
- **Operação do PBL:** (i) pbl_read3( ) (ii) pbl_nop( )
- **Operação dos PEs:** (i) nop( ) (ii) nop( )
- **Microcódigo:**
  - ♦ **bounce_tags(RIGHT,num_chips) :**

num_chips vezes
```
0  4  0  x...x  0   x...x  xxxx   222  FB  PblRead3()
0  0  0  x...x  0   x...x  xxxx   222  FB  Nop()
```

  - ♦ **bounce_tags(LEFT,num_chips) :**

num_chips vezes
```
0  4  32  x...x   0  x...x   xxxx  222  FB  PblRead3()
0  0  0   x...x   0  x...x   xxxx  222  FB  Nop()
```

- **Notas:** Permanecem bastantes dúvidas sobre esta instrução, pois depende do modo como os vários *chips* GLiTCH se ligam.

long_shift(dist)

- **Operação:** Desloca as TAGs para cima/baixo de uma distância em nº de PEs que é múltipla de 4, e utilizando para isso o mecanismo de *barrel* e o caminho de 32 bits entre *chips*.
- **Parâmetros:**
  - ◊ **dist** → qualquer valor positivo/negativo que seja múltiplo de 4 e esteja dentro do limite de PEs existentes para o lado menos/mais significativo do *chip* (com 64 PEs por *chip*).
- **Operação do PBL:** pbl_nop(), pbl_read3(), pbl_mergeR() e pbl_mergeL()
- **Operação dos PEs:** extract_tag(), restore_tag(), shift_up(), shift_down(), nop()
- **Microcódigo: A** função **long_shift_uc()** gera o microcódigo da instrução **long_shift()**, na qual o código envolvido por janelas representa funções que geram o microcódigo de instruções já referidas. Por exemplo, a função extract_tags_uc(rotacao) representa uma função que gera o microcódigo da instrução extract_tags(rotacao), tal como é descrito no manual dessa instrução.

```
void long_shift_uc(WORD dist)
    {
    char mascaraR[9]="00000000";
    char mascaraL[9]="00000000";
    WORD barrel, simples, rotacao, xs_rotacao, chips, i, r, l;

    barrel = dist - (dist%4);
    simples = dist - barrel;
    rotacao = barrel % PES_CHIP;

    if(dist >=0)  /* PE_ms >> SHIFT >> PE_ls */
        {
        chips = 1 + (barrel / PES_CHIP);
        xs_rotacao = rotacao - 32;
        for(r=4,l=4,i=7;i>=0;--i,r+=4,l+=4)
            {
            if(r <= rotacao) mascaraR[i] = '1';
            if(l <= xs_rotacao) mascaraL[i] = '1';
            }
        if(rotacao > 0)
            extract_tags_uc(rotacao);
        for(i=0;i<chips;++i)
            {
            bounce_tags_uc(RIGHT,1);
            merge_tags_uc(RIGHT,"11111111");
            bounce_tags_uc(LEFT,1);
            merge_tags_uc(LEFT,"11111111");
            }
        if(rotacao > 0)
            {
            bounce_tags_uc(RIGHT,1);
            merge_tags_uc(RIGHT,mascaraR);
            }
        if(xs_rotacao > 0)
            {
            bounce_tags_uc(LEFT,1);
            merge_tags_uc(LEFT,mascaraL);
            }
        if(rotacao > 0)
            restore_tags_uc();
        if(simples != 0)
            shift_uc(simples,CICLICO);
        }
```

```
else /* PE_ms << SHIFT << PE_ls */
    {
    chips = -1 + (barrel / PES_CHIP);
    xs_rotacao = rotacao + 32;
    for(r=-4,l=-4,i=0;i<8;++i,r-=4,l-=4)
        {
        if(l >= rotacao) mascaraL[i] = '1';
        if(r >= xs_rotacao) mascaraR[i] = '1';
        }
    if(rotacao < 0)
        extract_tags_uc(rotacao);
    for(i=chips+1;i<0;++i)
        {
        bounce_tags_uc(RIGHT,-1);
        merge_tags_uc(RIGHT,"11111111");
        bounce_tags_uc(LEFT,-1);
        merge_tags_uc(LEFT,"11111111");
        }
    if(rotacao < 0)
        {
        bounce_tags_uc(LEFT,-1);
        merge_tags_uc(LEFT,mascaraL);
        }
    if(xs_rotacao < 0)
        {
        bounce_tags_uc(RIGHT,-1);
        merge_tags_uc(RIGHT,mascaraR);
        }
    if(rotacao < 0)
        restore_tags_uc();

    if(simples != 0)
        shift_uc(simples,CICLICO);
    }
}
```

Instruções para Manipulação de TAGs de Vários *Chips*

alt(inicio,tipo_bloco,num_alts)

- **Operação:** Dividem-se as TAGs que estão a "1" ao meio e de forma alternada, ou seja, percorrem-se sequencialmente todos os PEs mudando uma TAG antes a "1" para "0" e mantendo a TAG seguinte antes a "1" com "1". O sinal de controlo ENABLE, utilizado nesta instrução, é guardado a intervalos de 16 ciclos sendo igual para cada 16PEs. Deste modo, a operação pode funcionar em grupos de 16 PEs, num *chip* inteiro ou em vários *chips*.
- **Parâmetros:**
  - ◊ **inicio:**
    - **0** → coloca a "0" a 1ª TAG que contenha "1", mantém a "1" a 2ª TAG que contenha "1", ...
    - **1** → mantém a "1" a 1ª TAG que contenha "1", coloca a "0" a 2ª TAG que contenha "1", ...

    ◊ **tipo_bloco:** {BLOCO, CHIP, CHIP2, CHIP4, CHIP8, CHIP64, ARRAY} → Indica que o bit definido por **inicio** é colocado no início de cada: grupo de 16 PEs, *chip*, 2 *chips*, 4 *chips*, 8 *chips*, 64 *chips* ou cadeia de todos os *chips*. O tipo de bloco define o conjunto de PEs em que a instrução alt() funciona de forma continuada.

    ◊ **num_alts** → valor inteiro (define o nº de vezes que se repete a operação)
- **Operação do PBL:** pbl_nop( )
- **Operação dos PEs:** init_alt_b0( ) / init_alt_b1( ) / init_alt_ch( ) e alt( )
- **Resultados:**

  ◊ $TAG_i = (TAG_i \,\&\, ALT_{i-1} \,\&\, ENABLE_i) \mid (TAG_i \,\&\, NOT(ENABLE_i))$

  ◊ $CY_i = (ALT_{i-1} \,\&\, ENABLE_i) \mid (CY_i \,\&\, NOT(ENABLE_i))$

  ◊ $ALT_i = ALT_{i-1} \,\hat{}\, TAG_i$

  (**i** define a ordem do PE e $ENABLE_i$ é igual para cada bloco de 16 PEs)
- **Microcódigo: A** função em linguagem C **alt_asm_uc()** apresentada a seguir, gera o microcódigo da instrução **alt()**. Nesta função o código envolvido por janelas representa o microcódigo de microinstruções. Por exemplo, **init_alt_b0_uc()** representa o microcódigo de **init_alt_b0().** Esquecendo o aspecto das *flags* indicado no algoritmo de **alt_asm_uc()**, o microcódigo da instrução **alt()** resulta em:

♦ **alt(0,BLOCO,num) :**

	42	**0**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**222**	**FB**	**InitAltB0()**
num vezes	43	0	0	x...x	0	x...x	xxxx	222	FB	Alt()

♦ **alt(1,BLOCO,num) :**

	41	**0**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**222**	**FB**	**InitAltB1()**
num vezes	**43**	**0**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**222**	**FB**	**Alt()**

♦ **alt(0,chip,num) : // chip={**CHIP, CHIP2, CHIP4, CHIP8, CHIP64, ARRAY}

	40	**0**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**222**	**FB**	**InitAltCh()**
num vezes	**43**	**0**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**222**	**FB**	**Alt()**

```
/* Função que gera o microcódigo de alt(inicio, tipo_bloco, num_alts)
para o chip num_chip, supondo que existem NCHIPS chips. */
int alt_asm_uc(int inicio, int tipo_bloco, int num_alts, int num_chip)
    {
    int i, j;
    for(j=0; j<num_alts; ++j)
        {
        switch(tipo_bloco)
            {
            case BLOCO:
                if(inicio == 0)
                    init_alt_b0_uc();
                else if(inicio == 1)
                    init_alt_b1_uc();
                break;

            case CHIP:
                init_alt_ch_uc();
                EFLAGS_IN[num_chip]=(inicio) ? (EFLAGS_IN[num_chip]
                     ALTINOUT) : (EFLAGS_IN[num_chip] & (!ALTINOUT));
                EFLAGS_IN[num_chip] = (EFLAGS_IN[num_chip]
                    | ENABLEINOUT);
                break;

            case CHIP2:
                init_alt_ch_uc();
                if((((NCHIPS-1)-num_chip)%2) == 0)
                    {
                    EFLAGS_IN[num_chip]=(inicio)?(EFLAGS_IN[num_chip] |
                        ALTINOUT): (EFLAGS_IN[num_chip] & (!ALTINOUT));
                    EFLAGS_IN[num_chip] = (EFLAGS_IN[num_chip] |
                        ENABLEINOUT);
                    }
                break;

            case CHIP4:
                init_alt_ch_uc();
                if((((NCHIPS-1)-num_chip)%4) == 0)
                    {
                    EFLAGS_IN[num_chip]=(inicio) ? (EFLAGS_IN[num_chip] |
                        ALTINOUT): (EFLAGS_IN[num_chip] & (!ALTINOUT));
                    EFLAGS_IN[num_chip] = (EFLAGS_IN[num_chip] |
                        ENABLEINOUT);
                    }
                break;
```

```
            case CHIP8:
                init_alt_ch_uc();
                if((((NCHIPS-1)-num_chip)%8) == 0)
                    {
                    EFLAGS_IN[num_chip]=(inicio) ? (EFLAGS_IN[num_chip] |
                        ALTINOUT) : (EFLAGS_IN[num_chip] & (!ALTINOUT));
                    EFLAGS_IN[num_chip] = (EFLAGS_IN[num_chip] |
                        ENABLEINOUT);
                    }
                break;

            case CHIP64:
                init_alt_ch_uc();
                if((((NCHIPS-1)-num_chip)%64) == 0)
                    {
                    EFLAGS_IN[num_chip]=(inicio) ? (EFLAGS_IN[num_chip] |
                        ALTINOUT) : (EFLAGS_IN[num_chip]&(!ALTINOUT));
                    EFLAGS_IN[num_chip] = (EFLAGS_IN[num_chip] |
                        ENABLEINOUT);
                    }
                break;

            case ARRAY:
                init_alt_ch_uc();
                if(num_chip==(NCHIPS-1))
                    {
                    EFLAGS_IN[num_chip]=(inicio) ? (EFLAGS_IN[num_chip] |
                        ALTINOUT) : (EFLAGS_IN[num_chip] & (!ALTINOUT));
                    EFLAGS_IN[num_chip] = (EFLAGS_IN[num_chip] |
                        ENABLEINOUT);
                    }
                break;
            }
          alt_uc();
        }
    }
```

tag_alt(inicio,tipo_bloco,num_alts)

- **Operação:** Funcionalidade idêntica à da instrução **alt()**, afectando apenas as TAGs. Os registos CY ficam inalterados.
- **Parâmetros:** Iguais aos da instrução **alt( )**
- **Operação do PBL:** pbl_nop( )
- **Operação dos PEs:** init_alt_b0( ) / init_alt_b1( ) / init_alt_ch( ) e tag_alt( )

- **Resultados:**
  - ◊ $TAG_i = (TAG_i \ \& \ ALT_{i-1} \ \& \ ENABLE_i) \mid (TAG_i \ \& \ NOT(ENABLE_i))$
  - ◊ $CY_i = CY_i$
  - ◊ $ALT_i = ALT_{i-1} \ \hat{} \ TAG_i$

  (**i** define a ordem do PE e $ENABLE_i$ é igual para cada bloco de 16 PEs)
- **Microcódigo: A** função para obter o microcódigo da instrução **tag_alt()** é idêntica à função que gera o microcódigo da instrução **alt()** - alt_asm_uc(), substituindo alt_uc() por **tag_alt_uc()**. Esquecendo o aspecto das *flags* indicado no algoritmo de **alt_asm_uc()**, o microcódigo de **tag_alt()** resulta em:

♦ **tag_alt(0,BLOCO,num) :**

	42	**0**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**222**	**FB**	**InitAltB0()**
num vezes	**44**	**0**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**222**	**FB**	**TagAlt()**

♦ **tag_alt(1,BLOCO,num) :**

	41	0	0	x...x	0	x...x	xxxx	222	FB	InitAltB1()
num vezes	**44**	**0**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**222**	**FB**	**TagAlt()**

♦ **tag_alt(0,chip,num) : // chip={**CHIP, CHIP2, CHIP4, CHIP8, CHIP64, ARRAY**}**

	40	**0**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**222**	**FB**	**InitAltCh()**
num vezes	**44**	**0**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**222**	**FB**	**TagAlt()**

cy_alt(inicio,tipo_bloco,num_alts)

- **Operação:** Funcionalidade idêntica à da instrução **alt()**, afectando apenas os registos CY. As TAGs ficam inalteradas.
- **Parâmetros:** Iguais aos da instrução **alt( )**
- **Operação do PBL:** pbl_nop( )
- **Operação dos PEs:** init_alt_b0( ) / init_alt_b1( ) / init_alt_ch( ) e carry_alt( )
- **Resultados:**
  - ◊ $CY_i = (ALT_{i-1} \ \& \ ENABLE_i) \mid (CY_i \ \& \ NOT(ENABLE_i))$
  - ◊ $TAG_i = TAG_i$
  - ◊ $ALT_i = ALT_{i-1} \ \hat{} \ TAG_i$

  (**i** define a ordem do PE e $ENABLE_i$ é igual para cada bloco de 16 PEs)
- **Microcódigo: A** função para obter o microcódigo da instrução **cy_alt()** é idêntica à função que gera o microcódigo da instrução **alt()** - alt_asm_uc(), substituindo alt_uc() por **carry_alt_uc()**. Esquecendo o aspecto das *flags* indicado no algoritmo de **alt_asm_uc()**, o microcódigo de **cy_alt()** resulta em:

♦ **cy_alt(0,BLOCO,num) :**

	42	**0**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**222**	**FB**	**InitAltB0()**
num vezes	**45**	**0**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**222**	**FB**	**CarryAlt()**

♦ **cy_alt(1,BLOCO,num) :**

num vezes (**41 0 0 x...x 0 x...x xxxx 222 FB InitAltB1()**
num vezes (**45 0 0 x...x 0 x...x xxxx 222 FB CarryAlt()**

♦ **cy_alt(0,chip,num) : // chip={**CHIP, CHIP2, CHIP4, CHIP8, CHIP64, ARRAY**}**

num vezes (40 0 0 x...x 0 x...x xxxx 222 FB InitAltCh()
num vezes (**45 0 0 x...x 0 x...x xxxx 222 FB CarryAlt()**

first(tipo_bloco,tipo_op)

- **Operação:** Marca o 1º PE que tiver a TAG a "1", colocando o registo CY a "1". Todos os outros PEs ficam com CY a "0". De acordo com o parâmetro **tipo_bloco,** marca-se 1 PE em cada bloco de 16 PEs, 1 PE em cada *chip* ou 1 PE na cadeia de *chips*. O parâmetro **tipo_op** indica onde se começa a busca de TAGs a "1" na cadeia de *chips*, permitindo assim continuar uma busca onde a instrução prévia acabou. Para efectuar várias buscas seguidas com first(), em que cada busca começa na posição onde a anterior acabou, a 1ª busca usa a instrução first() com **tipo_op**=RESET e as seguintes utilizam **tipo_op**=CONTINUE.
- **Parâmetros:**
  ◊ **tipo_bloco:** Define o conjunto de PEs em que a instrução first() funciona de forma continuada.
  (BLOCO → Indica que se marca-se 1 PE em cada bloco de 16 PEs
  (CHIP → Indica que se marca-se 1 PE em cada *chip*
  (ARRAY → Indica que se marca-se 1 PE na cadeia de *chips*
  ◊ **tipo_op:**
  (RESET → Inicia a busca da 1ª TAGs "1" no início do bloco (ou *chip*, ou cadeia de *chips*)
  (CONTINUE → Continua a busca de TAGs a "1" na posição onde terminou a busca anterior (definida por CY=1)
- **Operação do PBL:** pbl_nop( )
- **Operação dos PEs:** init_first0( ) / init_first1( ) e first( )
- **Resultados:**
  ◊ $CY_i = TAG_i$ & $FIRST_i$
  ◊ $TAG_i = TAG_i$
  ◊ $FIRST_i = FIRST_{i-1}$ & NOT (TAG_i)
  (**i** define a ordem do PE)
- **Microcódigo: A** função em linguagem C **first_asm_uc()** apresentada a seguir, gera o microcódigo da instrução **first()**. Nesta função o código envolvido por janelas representa o microcódigo de microinstruções. Por exemplo, **init_first0_uc()** representa o microcódigo de **init_first0().** Esquecendo o aspecto das *flags* indicado no algoritmo de **first_asm_uc()**, o microcódigo da instrução **first()** resulta em:

first(BLOCO,RESET)	**47**	**0**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**222**	**FB**	**InitFirst1()**
	48	**0**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**222**	**FB**	**First()**
first(BLOCO,CONTINUE)	**48**	**0**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**222**	**FB**	**First()**
first(CHIP,RESET)	**46**	**0**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**222**	**FB**	**InitFirst0()**
	48	**0**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**222**	**FB**	**First()**
first(CHIP,CONTINUE)	**48**	**0**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**222**	**FB**	**First()**
first(ARRAY,RESET)	**46**	**0**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**222**	**FB**	**InitFirst0()**
	48	**0**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**222**	**FB**	**First()**
first(ARRAY,CONTINUE)	**48**	**0**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**222**	**FB**	**First()**

/* Função que gera o microcódigo de **first(tipo_bloco, tipo_op)**
para o *chip* **num_chip**, supondo que existem NCHIPS *chips*. */

```
int first_asm_uc(int tipo_bloco, int tipo_op, int num_chip)
    {
    switch(tipo_bloco)
        {
        case BLOCO:
            if(tipo_op == RESET)
                init_first1_uc();
            else if(tipo_op == CONTINUE)
                ;
            break;

        case CHIP:
            EFLAGS_IN[num_chip] = (EFLAGS_IN[num_chip] | FIRSTINOUT);
            if(tipo_op == RESET)
                init_first0_uc();
            else if(tipo_op == CONTINUE)
                ;
            break;

        case ARRAY:
            if(num_chip==(NCHIPS-1))
                EFLAGS_IN[num_chip] = (EFLAGS_IN[num_chip] | FIRSTINOUT);
            else if((num_chip>=0) && (num_chip<NCHIPS))
                EFLAGS_IN[num_chip] = EFLAGS_IN[num_chip] & (!FIRSTINOUT);
            if(tipo_op == RESET)
                init_first0_uc();
            else if(tipo_op == CONTINUE)
                ;
            break;
        }
    first_uc();
    }
```

clear_first( )

- **Operação:** Para todos os PEs com o registo CY a "1", a TAG e o CY são colocados a "0". Esta instrução é adequada para os casos em que só existe um registo CY a "1", ou seja, para usar após a execução duma instrução first().
- **Parâmetros:** Não tem
- **Operação do PBL:** pbl_match( )
- **Operação dos PEs:** clear_first( )
- **Resultados:**
  - ◊ TAG = TAG & NOT (CY)
  - ◊ CY = NOT (OP0 ^ OP1) & (OP0 ^ CY) & MS
- **Microcódigo:**

49 1 0 x...x 0 x...x xxxx 312 FB ClearFirst()

Instruções para Leitura da CAM

read_cam(PE_select,pos)

- **Operação:** Lê 32 bits da linha da CAM seleccionada pelo parâmetro **PE_select**, a partir da posição **pos**. O resultado da leitura aparece no exterior do *chip* através do *read barrel routing shifter* e passando antes pelo PBLR. A linha que se lê da CAM corresponde ao PE que tiver a TAG com valor "1", ou ao PE com CY a "1" (se antes se executou uma instrução de first()). Se houver mais do que um PE com a TAG a "1" (ou CY a "1"), a instrução não funciona pois só pode ler uma linha de cada vez.
- **Parâmetros:**
  - ◊ **PE_select**:
    - TAG ← selecciona-se a linha em que a TAG é "1"
    - TAG_ABOVE ← faz o *shift* das TAGs para cima (*) e depois selecciona-se a linha em que a TAG é "1"
    - TAG_BELOW ← faz o *shift* das TAGs para baixo (†) e depois selecciona-se a linha em que a TAG é "1"
    - CARRY ← selecciona-se a linha em que o CY é "1"
  - ◊ **pos** → posição na linha da CAM [0:63]
- **Operação do PBL:** (i) pbl_nop(), (ii) pbl_read2() e (iii) pbl_read3()
- **Operação dos PEs:** (i)read_this()/read_above()/read_below(), (ii)refresh() e (ii)nop()
- **Microcódigo:**

read_cam(TAG,pos)	**50**	**0**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**010**	**FB**	**ReadThis()**
	63	**3**	**pos**	**x...x**	**0**	**x...x**	**xxxx**	**000**	**FB**	**Refresh()**
	0	**4**	**pos**	**x...x**	**0**	**x...x**	**xxxx**	**000**	**FB**	**PblRead3()**
read_cam(TAG_ABOVE,pos)	**51**	**0**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**010**	**FB**	**ReadAbove()**
	63	**3**	**pos**	**x...x**	**0**	**x...x**	**xxxx**	**000**	**FB**	**Refresh()**
	0	**4**	**pos**	**x...x**	**0**	**x...x**	**xxxx**	**000**	**FB**	**PblRead3()**
read_cam(TAG_BELOW,pos)	**52**	**0**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**010**	**FB**	**ReadBelow()**
	63	**3**	**pos**	**x...x**	**0**	**x...x**	**xxxx**	**000**	**FB**	**Refresh()**
	0	**4**	**pos**	**x...x**	**0**	**x...x**	**xxxx**	**000**	**FB**	**PblRead3()**
read_cam(CARRY,pos)	**50**	**0**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**020**	**FB**	**ReadThis()**
	63	**3**	**pos**	**x...x**	**0**	**x...x**	**xxxx**	**000**	**FB**	**Refresh()**
	0	**4**	**pos**	**x...x**	**0**	**x...x**	**xxxx**	**000**	**FB**	**PblRead3()**

- **Notas:**
  (*) "*Shift* das TAGs para cima" significa deslocar as TAGs dum PE para o PE vizinho menos significativo.
  (†) "*Shift* das TAGs para baixo" significa deslocar as TAGs dum PE para o PE vizinho mais significativo.

Instruções para Escrita na CAM

write_local(PE_select, DATA_select,pos,patt16,sub)

- **Operação:** De acordo com os PEs seleccionados pelo parâmetro **PE_select**, escreve-se a partir da posição **pos** da CAM (e na CAM-de-SUBSET) o bit seleccionado por **DATA_select** (se o bit correspondente dos padrões **patt16/sub** for "1") , ou então escreve-se nessas posições a negação do bit seleccionado por **DATA_select** (se o bit correspondente dos padrões **patt16/sub** for "0"). Apresenta-se a seguir uma explicação mais detalhada desta instrução:
  [**tag** = {TAG, TAG_ABOVE, TAG_BELOW, IMAGE}]

  write_local(ALL,tag,pos1,patt1,sub) → Em todos os PEs: escreve na posição **pos1** e seguintes da CAM (na CAM-de-SUBSET) o valor de **tag** se o bit do padrão **patt1** (**sub)** for "1", ou escreve **not(tag)** se o bit do padrão **patt1/sub** for "0".

  write_local(ALL,CARRY,pos1,patt1,sub) → Em todos os PEs: escreve na posição **pos1** e seguintes da CAM (na CAM-de-SUBSET) o valor de **CY** se o bit do padrão **patt1** (**sub)** for "1", ou escreve **not(CY)** se o bit do padrão **patt1** (**sub)** for "0".

  write_local(SUBSET,tag,pos1,patt1,sub):
  ◊ se (MS="0") → mantêm-se os valores da CAM/CAM-de-SUBSET;
  ◊ se (MS="1" & Bit_padrão_patt1/sub="0") → escreve na posição **pos1** e seguintes da CAM/na CAM-de-SUBSET o valor de **not(tag)**;
  ◊ se (MS="1" & Bit_padrão_patt1/sub="1") → escreve na posição **pos1** e seguintes da CAM/na CAM-de-SUBSET o valor de **tag**.

  write_local(SUBSET,CARRY,pos1,patt1,sub):
  ◊ se (MS="0") → mantêm-se os valores da CAM/CAM-de-SUBSET;
  ◊ se (MS="1" & Bit_padrão_patt1/sub="0") → escreve na posição **pos1** e seguintes da CAM/na CAM-de-SUBSET o valor de **not(CY)**;
  ◊ se (MS="1" & Bit_padrão_patt1/sub="1") → escreve na posição **pos1** e seguintes da CAM/na CAM-de-SUBSET o valor de **CY**.

  write_local(NOT_SUBSET,tag,pos1,patt1,sub):
  ◊ se (MS="1") → mantêm-se os valores da CAM/CAM-de-SUBSET;
  ◊ se (MS="0" & Bit_padrão_patt1/sub="0") → escreve na posição **pos1** e seguintes da CAM/na CAM-de-SUBSET o valor de **not(tag)**;

◊ se (MS="0" & Bit_padrão_patt1/sub="1") → escreve na posição **pos1** e seguintes da CAM/na CAM-de-SUBSET o valor de **tag**.

write_local(NOT_SUBSET,CARRY,pos1,patt1,sub):
◊ se (MS="1") → mantêm-se os valores da CAM/CAM-de-SUBSET;
◊ se (MS="0" & Bit_padrão_patt1/sub="0") → escreve na posição **pos1** e seguintes da CAM/na CAM-de-SUBSET o valor de **not(CY)**;
◊ se (MS="0" & Bit_padrão_patt1/sub="1") → escreve na posição **pos1** e seguintes da CAM/na CAM-de-SUBSET o valor de **CY**.

write_local(tag,CARRY,pos1,patt1,sub):
◊ se ((MS & TAG) ="0") → mantêm-se os valores da CAM/CAM-de-SUBSET;
◊ se ((MS & TAG) ="1") & (Bit_padrão_patt1/sub="0") → escreve na posição **pos1** e seguintes da CAM/na CAM-de-SUBSET o valor de **not(CY)**;
◊ se ((MS & TAG) ="1") & (Bit_padrão_patt1/sub="1") → escreve na posição **pos1** e seguintes da CAM/na CAM-de-SUBSET o valor de **CY**.

write_local(CARRY,tag,pos1,patt1,sub):
◊ se ((MS & CY) ="0") → mantêm-se os valores da CAM/CAM-de-SUBSET;
◊ se ((MS & CY) ="1") & (Bit_padrão_patt1/sub="0") → escreve na posição **pos1** e seguintes da CAM/na CAM-de-SUBSET o valor de **not(tag)**;
◊ se ((MS & CY) ="1") & (Bit_padrão_patt1/sub="1") → escreve na posição **pos1** e seguintes da CAM/na CAM-de-SUBSET o valor de **tag**.

- **Parâmetros:**
  - ◊ **PE_select**:
    - ALL ← Escreve na CAM/CAM-de-SUBSET de todos os PEs
    - SUBSET ← Escreve na CAM/CAM-de-SUBSET dos PEs com MS=1
    - NOT_SUBSET ← Escreve na CAM/CAM-de-SUBSET dos PEs com MS=0
    - TAG ← " " " " dos PEs com (TAG&MS)=1
    - TAG_ABOVE ← " " " " dos PEs com (TAG_ABOVE&MS)=1
    - TAG_BELOW ← " " " " dos PEs com (TAG_BELOW & MS)=1
    - IMAGE ← " " " " dos PEs com (LSBit_VSR & MS)=1
    - CARRY ← " " " " dos PEs com (CY & MS)=1
  - ◊ **DATA_select**:
    - TAG ← Selecciona a TAG para escrever
    - TAG_ABOVE ← Selecciona a TAG_ABOVE para escrever
    - TAG_BELOW ← Selecciona a TAG_BELOW para escrever
    - IMAGE ← Selecciona o LSBit_VSR para escrever
  - ◊ **pos** → posição inicial a partir da qual se escreve na CAM, valor em [0:63]
  - ◊ **patt16** → padrão ternário com 16 caracteres:
    - "0" ← Escreve na CAM a negação do que é seleccionado por DATA_select
    - "1" ← Escreve na CAM o que é seleccionado por DATA_select
    - "x" ← Não escreve na CAM

◊ **sub** → padrão ternário com 4 caracteres:
- "0" ← Escreve na CAM-de-SUBSET a negação do que é seleccionado por DATA_select
- "1" ← Escreve na CAM-de-SUBSET o que é seleccionado por DATA_select
- "x" ← Não escreve na CAM-de-SUBSET

- **Operação do PBL:** pbl_write( )
- **Operação dos PEs:** write_this( ) / write_above( ) / write_below( ) / write_image( )
- **Microcódigo:**

PARÂMETROS DE write_local()	MICROCÓDIGO
(ALL,TAG,pos,patt,sub)	53 2 pos patt 0 x...x sub 001 FB WriteThis()
(ALL,TAG_ABOVE,pos,patt,sub)	54 2 pos patt 0 x...x sub 001 FB WriteAbove()
(ALL,TAG_BELOW,pos,patt,sub)	55 2 pos patt 0 x...x sub 001 FB WriteBelow()
(ALL,CARRY,pos,patt,sub)	53 2 pos patt 0 x...x sub 002 FB WriteThis()
(ALL,IMAGE,pos,patt,sub)	56 2 pos patt 0 x...x sub 001 FB WriteImage()
(SUBSET,TAG,pos,patt,sub)	53 2 pos patt 0 x...x sub 101 FB WriteThis()
(SUBSET,TAG_ABOVE,pos,patt,sub)	54 2 pos patt 0 x...x sub 101 FB WriteAbove()
(SUBSET,TAG_BELOW,pos,patt,sub)	55 2 pos patt 0 x...x sub 101 FB WriteBelow()
(SUBSET,CARRY,pos,patt,sub)	53 2 pos patt 0 x...x sub 102 FB WriteThis()
(SUBSET,IMAGE,pos,patt,sub)	56 2 pos patt 0 x...x sub 101 FB WriteImage()
(NOT_SUBSET,TAG,pos,patt,sub)	53 2 pos patt 0 x...x sub 201 FB WriteThis()
(NOT_SUBSET,TAG_ABOVE,pos, patt,sub)	54 2 pos patt 0 x...x sub 201 FB WriteAbove()
(NOT_SUBSET,TAG_BELOW,pos, patt,sub)	55 2 pos patt 0 x...x sub 201 FB WriteBelow()
(NOT_SUBSET,CARRY,pos,patt,sub)	53 2 pos patt 0 x...x sub 202 FB WriteThis()
(NOT_SUBSET,IMAGE,pos,patt,sub)	56 2 pos patt 0 x...x sub 201 FB WriteImage()
(TAG,CARRY,pos,patt,sub)	53 2 pos patt 0 x...x sub 112 FB WriteThis()
(TAG_ABOVE,CARRY,pos,patt,sub)	54 2 pos patt 0 x...x sub 112 FB WriteAbove()
(TAG_BELOW,CARRY,pos,patt,sub)	55 2 pos patt 0 x...x sub 112 FB WriteBelow()
(IMAGE,CARRY,pos,patt,sub)	56 2 pos patt 0 x...x sub 112 FB WriteImage()
(CARRY,TAG,pos,patt,sub)	53 2 pos patt 0 x...x sub 121 FB WriteThis()
(CARRY,TAG_ABOVE, pos,patt,sub)	54 2 pos patt 0 x...x sub 121 FB WriteAbove()
(CARRY,TAG_BELOW, pos,patt,sub)	55 2 pos patt 0 x...x sub 121 FB WriteBelow()
(CARRY,IMAGE, pos,patt,sub)	56 2 pos patt 0 x...x sub 121 FB WriteImage()

write_all(PE_select, pos1,patt1,pos2,patt2,sub)

- **Operação:** Nos PEs seleccionados pelo parâmetro **PE_select**, escreve-se o padrão **patt1** a partir da posição **pos1** da CAM, escreve-se o padrão **patt2** a partir da posição **pos2** da CAM e escreve-se o padrão **sub** na CAM-de-SUBSET). Nos outros PEs não se escreve nada. Apresenta-se a seguir uma explicação mais detalhada desta instrução:
  [**tag** = {TAG, TAG_ABOVE, TAG_BELOW, IMAGE}]
  [not_**tag** = {NOT_TAG, NOT_TAG_ABOVE, NOT_TAG_BELOW}]

  write_all(ALL,pos1,patt1,pos2,patt2,sub) → Escreve em **todos** os PEs: **patt1** a partir da posição **pos1** da CAM, **patt2** a partir da posição **pos2** da CAM e **sub** na CAM-de-SUBSET).

write_all(SUBSET,pos1,patt1,pos2,patt2,sub) → Nos PEs que tiverem **MS=1** escreve: **patt1** a partir da posição **pos1** da CAM, **patt2** a partir da posição **pos2** da CAM e **sub** na CAM-de-SUBSET). Nos PEs com **MS=0** não se escreve nada.

write_all(NOT_SUBSET,pos1,patt1,pos2,patt2,sub) → Nos PEs que tiverem **MS=0** escreve: **patt1** a partir da posição **pos1** da CAM, **patt2** a partir da posição **pos2** da CAM e **sub** na CAM-de-SUBSET). Nos PEs com **MS=1** não se escreve nada.

write_all(tag,pos1,patt1,pos2,patt2,sub) → Nos PEs que tiverem **tag=1** escreve: **patt1** a partir da posição **pos1** da CAM, **patt2** a partir da posição **pos2** da CAM e **sub** na CAM-de-SUBSET). Nos PEs com **tag=0** não se escreve nada.

write_all(not_tag,pos1,patt1,pos2,patt2,sub) → Nos PEs que tiverem **not_tag=1** escreve: **patt1** a partir da posição **pos1** da CAM, **patt2** a partir da posição **pos2** da CAM e **sub** na CAM-de-SUBSET). Nos PEs com **not_tag=0** não se escreve nada.

write_all(CARRY,pos1,patt1,pos2,patt2,sub) → Nos PEs que tiverem **CY=1** escreve: **patt1** a partir da posição **pos1** da CAM, **patt2** a partir da posição **pos2** da CAM e **sub** na CAM-de-SUBSET). Nos PEs com **CY=0** não se escreve nada.

write_all(NOT_CARRY,pos1,patt1,pos2,patt2,sub) → Nos PEs que tiverem **CY=0** escreve: **patt1** a partir da posição **pos1** da CAM, **patt2** a partir da posição **pos2** da CAM e **sub** na CAM-de-SUBSET). Nos PEs com **CY=1** não se escreve nada.

- **Parâmetros:**
  - ◊ **PE_select**:

    ⎡ALL ← Escreve na CAM/CAM-de-SUBSET de todos os PEs
    ⎢SUBSET ← Escreve na CAM/CAM-de-SUBSET dos PEs com MS=1
    ⎢NOT_SUBSET ← Escreve na CAM/CAM-de-SUBSET dos PEs com MS=0
    ⎢TAG ← ″ ″ ″ ″ dos PEs com TAG=1
    ⎢TAG_ABOVE ← ″ ″ ″ ″ dos PEs com TAG_ABOVE=1
    ⎢TAG_BELOW ← ″ ″ ″ ″ dos PEs com TAG_BELOW =1
    ⎢NOT_TAG ← ″ ″ ″ ″ dos PEs com TAG=0
    ⎢NOT_TAG_ABOVE ← ″ ″ ″ ″ dos PEs com TAG_ABOVE=0
    ⎢NOT_TAG_BELOW ← ″ ″ ″ ″ dos PEs com TAG_BELOW=0
    ⎢IMAGE ← ″ ″ ″ ″ dos PEs com LSBit_VSR=1
    ⎢CARRY ← ″ ″ ″ ″ dos PEs com CY=1
    ⎣NOT_CARRY ← ″ ″ ″ ″ dos PEs com CY=0

  - ◊ **pos1, pos2** → posição inicial a partir da qual se escreve na CAM o padrão patt1/patt2, valor em [0:63]
  - ◊ **patt1, patt2** → padrão ternário com 8 caracteres a escrever na CAM
  - ◊ **sub** → padrão ternário com 4 caracteres a escrever na CAM-de-SUBSET
- **Operação do PBL:** pbl_write( )
- **Operação dos PEs:** write_this( ) / write_above( ) / write_below( ) / write_image( )

- **Microcódigo:**

PARÂMETROS DE write_all()	MICROCÓDIGO
(ALL,pos1, patt1,pos2,patt2,sub)	53 2 pos1 patt1 pos2 patt2 sub 000 FB WriteThis()
(SUBSET,pos1, patt1,pos2,patt2,sub)	53 2 pos1 patt1 pos2 patt2 sub 100 FB WriteThis()
(NOT_SUBSET,pos1, patt1,pos2, patt2,sub)	53 2 pos1 patt1 pos2 patt2 sub 200 FB WriteThis()
(TAG,pos1, patt1,pos2,patt2,sub)	53 2 pos1 patt1 pos2 patt2 sub 010 FB WriteThis()
(TAG_ABOVE,pos1, patt1,pos2, patt2,sub)	54 2 pos1 patt1 pos2 patt2 sub 010 FB WriteAbove()
(TAG_ BELOW,pos1, patt1,pos2, patt2,sub)	55 2 pos1 patt1 pos2 patt2 sub 010 FB WriteBelow ()
(NOT_TAG,pos1, patt1,pos2,patt2,sub)	53 2 pos1 patt1 pos2 patt2 sub 030 FB WriteThis()
(NOT_TAG_ABOVE,pos1, patt1,pos2, patt2,sub)	54 2 pos1 patt1 pos2 patt2 sub 030 FB WriteAbove()
(NOT_TAG_ BELOW,pos1, patt1, pos2,patt2,sub)	55 2 pos1 patt1 pos2 patt2 sub 030 FB WriteBelow()
(CARRY,pos1, patt1,pos2, patt2,sub)	53 2 pos1 patt1 pos2 patt2 sub 020 FB WriteThis()
(NOT_CARRY,pos1, patt1,pos2, patt2,sub)	53 2 pos1 patt1 pos2 patt2 sub 040 FB WriteThis()
(IMAGE,pos1, patt1,pos2, patt2,sub)	56 2 pos1 patt1 pos2 patt2 sub 010 FB WriteImage ()

write_subset(PE_select, pos1,patt1,pos2,patt2,sub)

- **Operação:** Nos PEs seleccionados pela conjunção AND entre **PE_select** e **MS**, escreve-se o padrão **patt1** a partir da posição **pos1** da CAM, escreve-se o padrão **patt2** a partir da posição **pos2** da CAM e escreve-se o padrão **sub** na CAM-de-SUBSET). Nos outros PEs não se escreve nada. Apresenta-se a seguir uma explicação mais detalhada desta instrução:
  [**tag** = {TAG, TAG_ABOVE, TAG_BELOW, IMAGE}]
  [not_**tag** = {NOT_TAG, NOT_TAG_ABOVE, NOT_TAG_BELOW}]

  ⌈**write_subset(ALL,pos1,patt1,pos2,patt2,sub)** ⌉
  ⌊**write_subset(SUBSET,pos1,patt1,pos2,patt2,sub)**⌋
  ⇑
  ⇓
  write_all(SUBSET,pos1,patt1,pos2,patt2,sub)

  write_subset(tag,pos1,patt1,pos2,patt2,sub) → Nos PEs em que (**tag=1 & MS=1**) escreve: **patt1** a partir da posição **pos1** da CAM, **patt2** a partir da posição **pos2** da CAM e **sub** na CAM-de-SUBSET). Nos outros PEs não se escreve nada.

  write_subset(not_tag,pos1,patt1,pos2,patt2,sub) → Nos PEs em que **(MS=1 & not_tag=1)** escreve: **patt1** a partir da posição **pos1** da CAM, **patt2** a partir da posição **pos2** da CAM e **sub** na CAM-de-SUBSET). Nos outros PEs não se escreve nada.

  write_subset(CARRY,pos1,patt1,pos2,patt2,sub) → Nos PEs em que **(MS=1 & CY=1)** escreve: **patt1** a partir da posição **pos1** da CAM, **patt2** a partir da posição **pos2** da CAM e **sub** na CAM-de-SUBSET). Nos outros PEs não se escreve nada.

write_subset(NOT_CARRY,pos1,patt1,pos2,patt2,sub) → Nos PEs em que (**MS=1 & CY=0**) escreve: **patt1** a partir da posição **pos1** da CAM, **patt2** a partir da posição **pos2** da CAM e **sub** na CAM-de-SUBSET). Nos outros PEs não se escreve nada.

- **Parâmetros:** Iguais aos da instrução **write_all(**), excepto a opção NOT_SUBSET para o parâmetro **PE_select.**
- **Operação do PBL:** pbl_write( )
- **Operação dos PEs:** write_this( ) / write_above( ) / write_below( ) / write_image( )
- **Microcódigo:**

PARÂMETROS DE write_subset( )	MICROCÓDIGO
(ALL,pos1, patt1,pos2,patt2,sub)	53 2 pos1 patt1 pos2 patt2 sub 100 FB WriteThis()
(SUBSET,pos1, patt1,pos2,patt2,sub)	53 2 pos1 patt1 pos2 patt2 sub 100 FB WriteThis()
(TAG,pos1, patt1,pos2,patt2,sub)	53 2 pos1 patt1 pos2 patt2 sub 110 FB WriteThis()
(TAG_ABOVE,pos1, patt1,pos2, patt2,sub)	54 2 pos1 patt1 pos2 patt2 sub 110 FB WriteAbove()
(TAG_ BELOW,pos1, patt1,pos2, patt2,sub)	55 2 pos1 patt1 pos2 patt2 sub 110 FB WriteBelow ()
(NOT_TAG,pos1, patt1,pos2,patt2,sub)	53 2 pos1 patt1 pos2 patt2 sub 130 FB WriteThis()
(NOT_TAG_ABOVE,pos1, patt1,pos2, patt2,sub)	54 2 pos1 patt1 pos2 patt2 sub 130 FB WriteAbove()
(NOT_TAG_ BELOW,pos1, patt1, pos2,patt2,sub)	55 2 pos1 patt1 pos2 patt2 sub 130 FB WriteBelow()
(CARRY,pos1, patt1,pos2, patt2,sub)	53 2 pos1 patt1 pos2 patt2 sub 120 FB WriteThis()
(NOT_CARRY,pos1, patt1,pos2, patt2,sub)	53 2 pos1 patt1 pos2 patt2 sub 140 FB WriteThis()
(IMAGE,pos1, patt1,pos2, patt2,sub)	56 2 pos1 patt1 pos2 patt2 sub 110 FB WriteImage ()

write_not_subset(PE_select, pos1,patt1,pos2,patt2,sub)

- **Operação:** Nos PEs seleccionados pela conjunção AND entre **PE_select** e **NOT(MS)**, escreve-se o padrão **patt1** a partir da posição **pos1** da CAM, escreve-se o padrão **patt2** a partir da posição **pos2** da CAM e escreve-se o padrão **sub** na CAM-de-SUBSET). Nos outros PEs não se escreve nada. Apresenta-se a seguir uma explicação mais detalhada desta instrução:
  [**tag** = {TAG, TAG_ABOVE, TAG_BELOW, IMAGE}]
  [not_**tag** = {NOT_TAG, NOT_TAG_ABOVE, NOT_TAG_BELOW}]

⌈ **write_not_subset(ALL,pos1,patt1,pos2,patt2,sub)** ⌉
⌊ **write_not_subset(NOT_SUBSET,pos1,patt1,pos2,patt2,sub)** ⌋

⇑
⇓

write_all(NOT_SUBSET,pos1,patt1,pos2,patt2,sub)

write_not_subset(tag,pos1,patt1,pos2,patt2,sub) → Nos PEs em que (**MS=0 & tag=1**) escreve: **patt1** a partir da posição **pos1** da CAM, **patt2** a partir da posição **pos2** da CAM e **sub** na CAM-de-SUBSET). Nos outros PEs não se escreve nada.

write_not_subset(not_tag,pos1,patt1,pos2,patt2,sub) → Nos PEs em que **(MS=0 & not_tag=1)** escreve: **patt1** a partir da posição **pos1** da CAM, **patt2** a partir da posição **pos2** da CAM e **sub** na CAM-de-SUBSET). Nos outros PEs não se escreve nada.

write_not_subset(CARRY,pos1,patt1,pos2,patt2,sub) → Nos PEs em que **(MS=0 & CY=1)** escreve: **patt1** a partir da posição **pos1** da CAM, **patt2** a partir da posição **pos2** da CAM e **sub** na CAM-de-SUBSET). Nos outros PEs não se escreve nada.

write_not_subset(NOT_CARRY,pos1,patt1,pos2,patt2,sub) → Nos PEs em que **(MS=0 & CY=0)** escreve: **patt1** a partir da posição **pos1** da CAM, **patt2** a partir da posição **pos2** da CAM e **sub** na CAM-de-SUBSET). Nos outros PEs não se escreve nada.

- **Parâmetros:** Iguais aos da instrução **write_all( )**, excepto a opção SUBSET para o parâmetro **PE_select.**
- **Operação do PBL:** pbl_write( )
- **Operação dos PEs:** write_this( ) / write_above( ) / write_below( ) / write_image( )
- **Microcódigo:**

PARÂMETROS DE write_not_subset( )	MICROCÓDIGO
(ALL,pos1, patt1,pos2,patt2,sub)	53 2 pos1 patt1 pos2 patt2 sub 200 FB WriteThis()
(NOT_SUBSET,pos1, patt1,pos2, patt2,sub)	53 2 pos1 patt1 pos2 patt2 sub 200 FB WriteThis()
(TAG,pos1, patt1,pos2,patt2,sub)	53 2 pos1 patt1 pos2 patt2 sub 210 FB WriteThis()
(TAG_ABOVE,pos1, patt1,pos2, patt2,sub)	54 2 pos1 patt1 pos2 patt2 sub 210 FB WriteAbove()
(TAG_ BELOW,pos1, patt1,pos2, patt2,sub)	55 2 pos1 patt1 pos2 patt2 sub 210 FB WriteBelow ()
(NOT_TAG,pos1, patt1,pos2,patt2,sub)	53 2 pos1 patt1 pos2 patt2 sub 230 FB WriteThis()
(NOT_TAG_ABOVE,pos1, patt1,pos2, patt2,sub)	54 2 pos1 patt1 pos2 patt2 sub 230 FB WriteAbove()
(NOT_TAG_ BELOW,pos1, patt1, pos2,patt2,sub)	55 2 pos1 patt1 pos2 patt2 sub 230 FB WriteBelow()
(CARRY,pos1, patt1,pos2, patt2,sub)	53 2 pos1 patt1 pos2 patt2 sub 220 FB WriteThis()
(NOT_CARRY,pos1, patt1,pos2, patt2,sub)	53 2 pos1 patt1 pos2 patt2 sub 240 FB WriteThis()
(IMAGE,pos1, patt1,pos2, patt2,sub)	56 2 pos1 patt1 pos2 patt2 sub 210 FB WriteImage ()

Instruções Diversas

nop( )

- **Operação:** Não tem associada qualquer acção.
- **Parâmetros:** Não tem.
- **Operação do PBL:** pbl_nop( )

- **Operação dos PEs:** nop( )
- **Microcódigo:**
  0 0 0 x...x 0 x...x xxxx 222 FB Nop()

refresh( )

- **Operação:** Faz o *refresh* da CAM.
- **Parâmetros:** Não tem.
- **Operação do PBL:** pbl_nop( )
- **Operação dos PEs:** refresh( )
- **Microcódigo:**
  63 0 0 x...x 0 x...x xxxx 222 FB Refresh()

tag_to_cy(sub)

- **Operação:** Copiam-se todas as TAGs para os registos CY. O registo MS guarda o resultado da operação de *match* entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET.
- **Parâmetros:**
  - ◊ **sub** → padrão de subset com 4 dígitos ternários (0|1|x)
- **Operação do PBL:** pbl_match( )
- **Operação dos PEs:** tag_to_cy( )
- **Resultados:**
  - ◊ TAG = TAG
  - ◊ CY = TAG
  - ◊ MS → é afectado do mesmo modo que em **match( )**
- **Microcódigo:**
  58 1 0 x...x 0 x...x sub 344 FB TagToCy()

cy_to_tag(sub)

- **Operação:** Copiam-se todos os registos CY para as TAGs. O registo MS guarda o resultado da operação de *match* entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET.
- **Parâmetros:**
  - ◊ **sub** → padrão de subset com 4 dígitos ternários (0|1|x)
- **Operação do PBL:** pbl_match( )
- **Operação dos PEs:** xor_this( )
- **Resultados:**
  - ◊ TAG = CY
  - ◊ CY = CY
  - ◊ MS → é afectado do mesmo modo que em **match( )**
- **Microcódigo:**
  32 1 0 x...x 0 x...x sub 331 FB XorThis()

cy_clear_to_tag(sub)

- **Operação:** Copiam-se todos os registos CY para as TAGs e depois colocam-se os registos CY a "0". O registo MS guarda o resultado da operação de *match* entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET.
- **Parâmetros:**
  - ◊ **sub** → padrão de subset com 4 dígitos ternários (0|1|x)
- **Operação do PBL:** pbl_match( )
- **Operação dos PEs:** arith_this( )
- **Resultados:**
  - ◊ TAG = CY
  - ◊ CY = 0
  - ◊ MS → é afectado do mesmo modo que em **match( )**
- **Microcódigo:**

5 1 0 x...x 0 x...x sub 331 FB ArithThis()

tag_clear_to_cy(sub)

- **Operação:** Copiam-se todas as TAGs para os registos CY e depois colocam-se as TAGs a "0". O registo MS guarda o resultado da operação de *match* entre o padrão de subset (**sub**) e o conteúdo da CAM-de-SUBSET.
- **Parâmetros:**
  - ◊ **sub** → padrão de subset com 4 dígitos ternários (0|1|x)
- **Operação do PBL:** pbl_match( )
- **Operação dos PEs:** swap_this( )
- **Resultados:**
  - ◊ TAG = 0
  - ◊ CY = TAG
  - ◊ MS → é afectado do mesmo modo que em **match( )**
- **Microcódigo:**

9 1 0 x...x 0 x...x sub 344 FB SwapThis()

tag_to_subset( )

- **Operação:** Copiam-se todas as TAGs para os registos de *subset* MS.
- **Parâmetros:** Não tem.
- **Operação do PBL:** pbl_nop( )
- **Operação dos PEs:** tag_to_subs( )
- **Resultados:**
  - ◊ TAG = TAG
  - ◊ CY = CY
  - ◊ MS = TAG
- **Microcódigo:**

59 0 0 x...x 0 x...x xxxx 222 FB TagToSubs()

clear(opc)

- **Operação:** Permite optar entre colocar a "0" todos os registos CY, todas as TAGs ou ambos.
- **Parâmetros:**
  - ◊ **opc:**
    - TAG ← Coloca as TAGs a "0".
    - CARRY ← Coloca os registos CY a "0".
    - TAG_CARRY ← Coloca tanto as TAGs como os registos CY a "0".
- **Operação do PBL:** pbl_match( )
- **Operação dos PEs:**
  - swap_holdms() ← opc=TAG
  - and_match() ← opc=CARRY
  - arith_holdms() ← opc=TAG_CARRY
- **Microcódigo:**

clear(TAG)	**14**	**1**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**331**	**FB**	**SwapHoldms()**
clear(CARRY)	**4**	**1**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**334**	**FB**	**AndMatch()**
clear(TAG_CARRY)	**13**	**1**	**0**	**x...x**	**0**	**x...x**	**xxxx**	**334**	**FB**	**ArithHoldms()**

clear_not_subset(opc,sub)

- **Operação:** Colocam-se a "0" os registos CY, ou as TAGs, de todos os PEs que não estejam dentro dum dado *subset*, ou seja, de todos os PE que tenham o registo MS a "0". Isto garante que haja apenas TAG_REPLY ou CARRY_REPLY activos quando houver CYs, ou TAGs, a "1" dentro dum dado *subset*.
- **Parâmetros:**
  - ◊ **opc:**
    - TAG ← Coloca TAGs a "0".
    - CARRY ← Coloca registos CY a "0".
  - ◊ **sub** → padrão de subset com 4 dígitos ternários (0|1|x), que define que PEs são afectados pela instrução.
- **Operação do PBL:** pbl_match( )
- **Operação dos PEs:**
  - tag_match() ← opc=TAG
  - clear_n_sub() ← opc=CARRY
- **Microcódigo:**

clear_not_subset(TAG,sub)	**2**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**142**	**FB**	**TagMatch()**
clear_not_subset (CARRY,sub)	**62**	**1**	**0**	**x...x**	**0**	**x...x**	**sub**	**331**	**FB**	**ClearNSub()**