14. Estrutura interna do PDS16 14-2
14.1.1 Banco de registos (*Register File*) 14-3
14.1.2 Data Processor 14-5
14.1.3 BIU (*Bus Interface Unit*) 14-7
14.2 Especificação dos sinais e organização do BUS 14-8
14.2.1 MCLK (*Master Clock*) 14-8
14.2.2 RESET e RESOUT 14-8
14.2.3 BUS de endereços e dados ($AD_{0\text{-}15}$) 14-8
14.2.4 Ciclos de leitura e escrita 14-9
14.3 Ciclos Máquina 14-10
14.4 Sincronização e interacção com dispositivos externos 14-11
14.5 RDY 14-11
14.6 Partilha de BUS 14-12
14.7 Informação de estado (S1, S0) 14-14

14. ESTRUTURA INTERNA DO PDS16

Na Figura 14-1 é apresentado o diagrama de blocos de uma estrutura para dar resposta à especificação estabelecida para o PDS16. Para realizar a leitura de um *byte*, foi inserido um multiplexer, que função do bit de endereço A_0, selecciona a parte alta ou a parte baixa da *word* lida da memória. Para permitir a escrita de um *byte* na memória, o módulo de controlo toma como entrada o bit de endereço A_0, e caso este tenha o valor lógico zero (endereço par), activa o sinal de a saída WRH, caso contrário activa WRL.

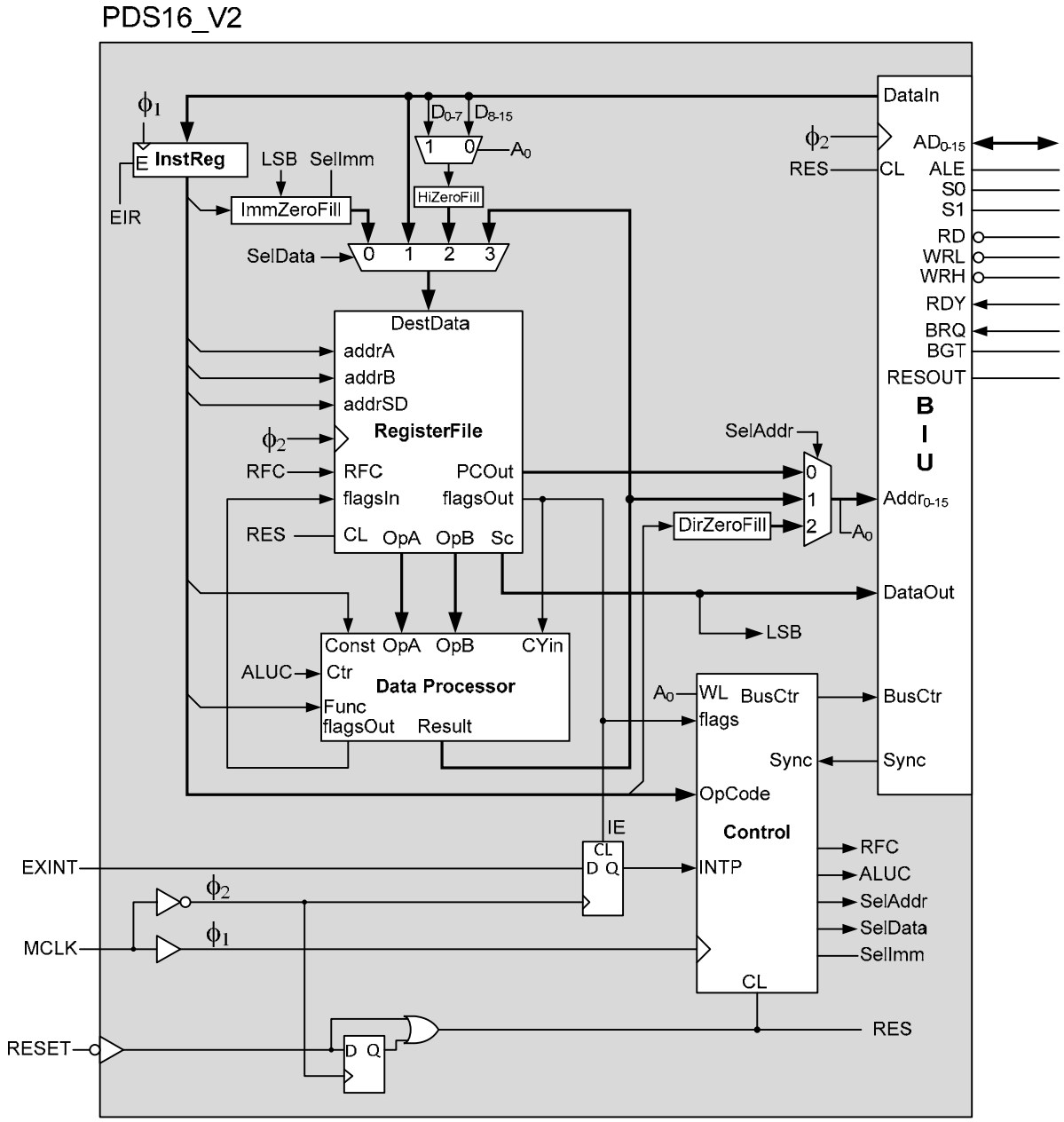

Figura 14-1 - Diagrama de blocos do CPU PDS16

14.1.1 Banco de registos (*Register File*)

O *Register File* é uma estrutura de dados constituída por vários registos de n bits, organizados sobe a forma de um vector, disponibilizando geralmente uma entrada de dados e duas ou mais saídas de dados, dependendo da funcionalidade pretendida. Independentemente do número de registos que constituem a estrutura, em geral, a implementação do *Register File* não permite o acesso simultâneo a todos os seus registos.

Se considerarmos um *Register File* como mostra a Figura 14-2, associado a uma ALU, capaz de em simultâneo fornecer os dois operandos A e B e registar o resultado produzida pela ALU, facilmente se conclui que o *Register File* tem que disponibilizar o acesso independente e simultâneo a três registos, dois para leitura e um para escrita.

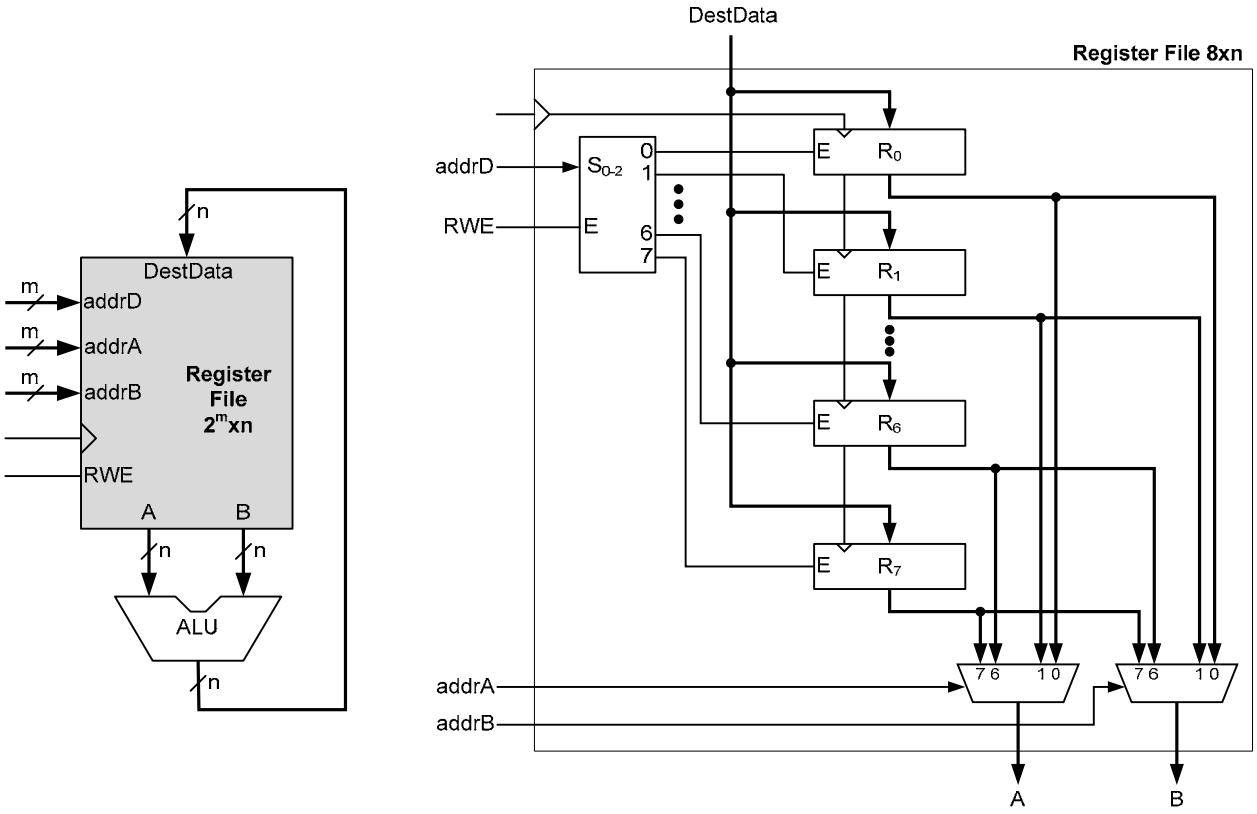

Figura 14-2 – *Register File*

Tratando-se de uma estrutura organizada sob a forma de vector, o acesso simultâneo e independente a três registos implica a existência de três *buses* de endereços. Um dos endereços é denominado por addrD (*Destination Data Address*), e serve para designar o registo que vai ser escrito. Os outros dois *buses* de endereços são denominados por addrA (A *address*) e addrB (B *address*) e servem para designar respectivamente os registos que põem a informação neles contida, disponível nas saídas de dados A e B. Os registos que constituem o *Register File* são de natureza *edge-trigger* com controlo de *Enable*, razão pela qual o *Register File* para além de uma entrada de *clock*, disponibiliza uma entrada para controlo de escrita, que denominaremos por RWE.

PDS16 Register File

O *Register File* do PDS16, cujo diagrama de blocos é mostrada na **Figura 14-3**, é composto por 8 registos (R0 a R7), contendo cada um 16 bit, permitindo não só dispor de mais registos que a versão PDS8, como permite que as instruções possam especificar até três registos (três fontes ou duas fontes e um destino), aumentando desta forma a versatilidade do ISA.

O bus de endereço addrD passou a denominar-se addrDS (*Destination and Source data Address*), pois também especifica qual o registo que serve de fonte ao bus de dados Sc.

Outra funcionalidade que o PDS16 apresenta, com o objectivo de flexibilizar e uniformizar o ISA, é que todos os registos funcionais, tais como o PC, o PSW e o LINK, foram inseridos no *Register File*, correspondendo R5 ao LINK, R6 ao PSW e R7 ao PC. Esta funcionalidade permite realizar operações sobre estes registos específicos com as mesmas instruções com que se opera sobre os registos genéricos.

O acção de incremento do valor do registo PC, é realizado por um somador específico existente no *Register File*, enquanto que o cálculo do valor do endereço na instrução de salto, é realizado pela ALU. Esta funcionalidade implica que em certas circunstâncias, o acesso aos registos funcionais (PC, PSW e LINK) se faça fora da estrutura de vector, tornando a estrutura do *Register File* ligeiramente mais complexa. É também por esta razão que em vez do sinal RWE passaremos a ter um bus RFC (*Register File Control*).

A entrada CL (*Clear*) é activada quando o CPU entra em estado de reset, e tem como objectivo colocar os registos PC e PSW ao valor zero.

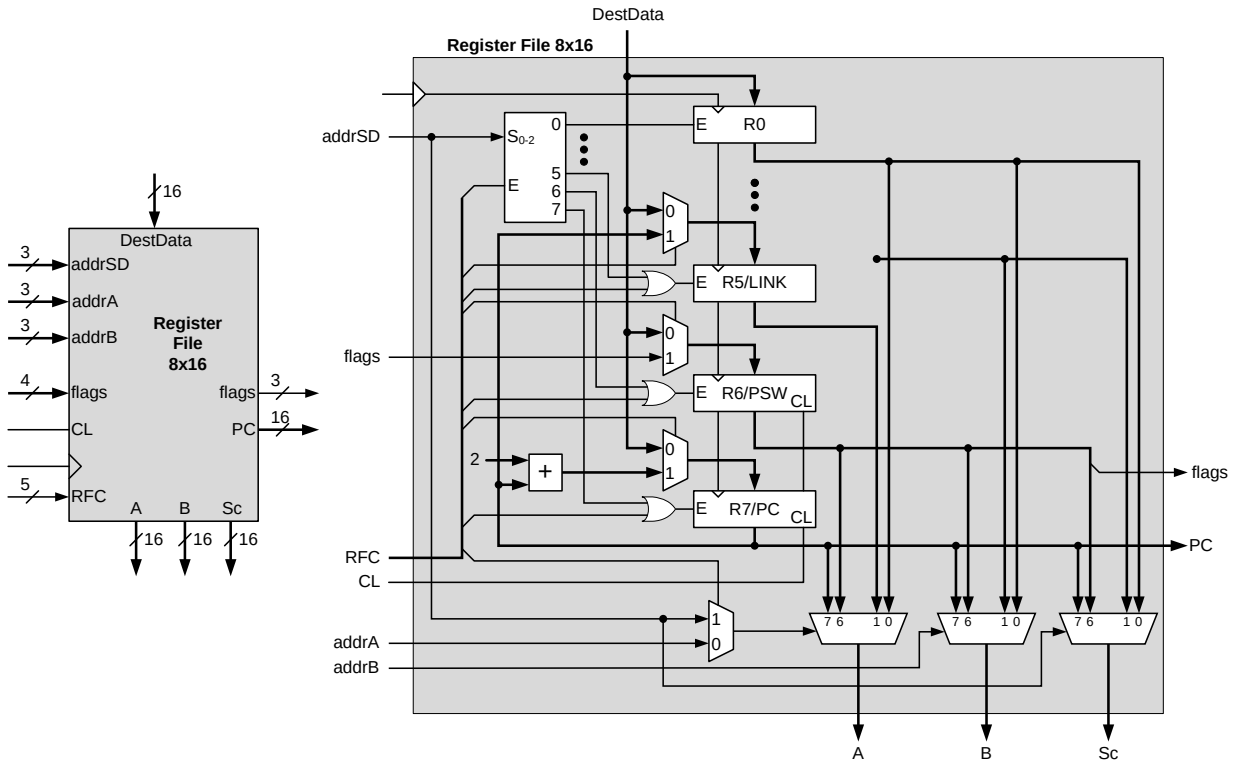

Figura 14-3 – *Register File* do PDS16

14.1.2 Data Processor

O projecto de unidades aritméticas e lógicas constitui desde à muito uma área de estudo muito importante, pois são normalmente estruturas muito complexas e onde a velocidade de cálculo é determinante no desempenho da estrutura de computação. Neste documento esta problemática não é abordada, sendo as soluções aqui apresentadas, simples exemplos, cujo objectivo é a fácil compreensão do ponto de vista funcional.

Na Figura 14-4 é apresentado o diagrama de blocos da unidade *Data Processor* do PDS16_V2. A unidade *Data Processor* do PDS16_V2, inclui para além da ALU, um elemento para multiplexagem e transformação do operando B, ou seja, extensão de sinal, preenchimento à direita com zeros e produto por dois. Os blocos que realizam a operação de produto por 2, não envolvem adição de elementos lógica, pois trata-se de simples deslocação do operando um bit para a esquerda.

Dado que a unidade *Data Processor* também é utilizada no cálculo do endereço das instruções que envolvem acesso à memória no modo indexado e baseado indexado, bem como no cálculo do valor do registo PC nas instruções de salto, foi inserida uma lógica de intercepção que força a ALU a realizar uma adição entre A e B, em todas as instruções que não pertençam ao grupo de *Data Process*. A razão de Opb e Const4, poderem ter que ser multiplicado por dois prende-se com o acesso poder ser ao byte ou à *word*.

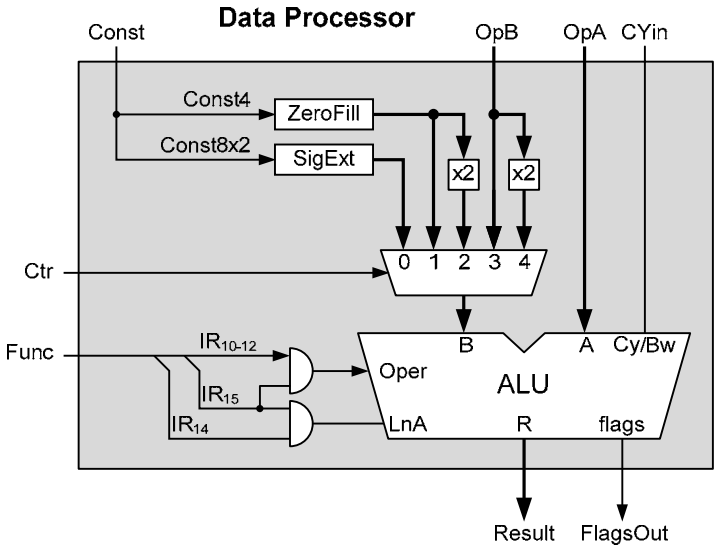

Figura 14-4 – Diagrama de blocos do *Data Processor*

A ALU está dividida em três blocos funcionais conforme a Figura 14-5. O bloco Aritmético que é responsável pelas somas e subtracções, o bloco Lógico responsável pelas operações AND, OR, XOR e NOT, e o bloco de *Shift* que executa as operações de SHL, SHR, RRL e RRM.

As operações deslocamento são realizadas por um *barrel shift*. Para mais fácil compreensão é apresentado na Figura 14-6 um *barrel shift* de quatro bits cuja solução é facilmente extrapolável para 16 bits.

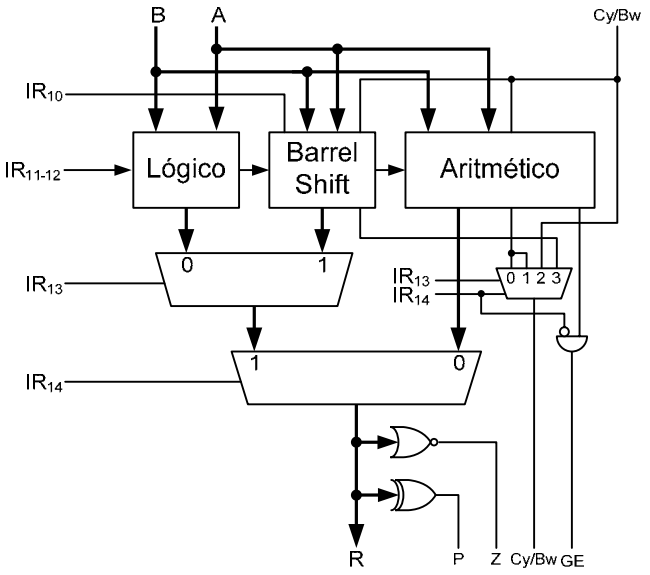

Figura 14-5 – Diagrama de blocos da ALU

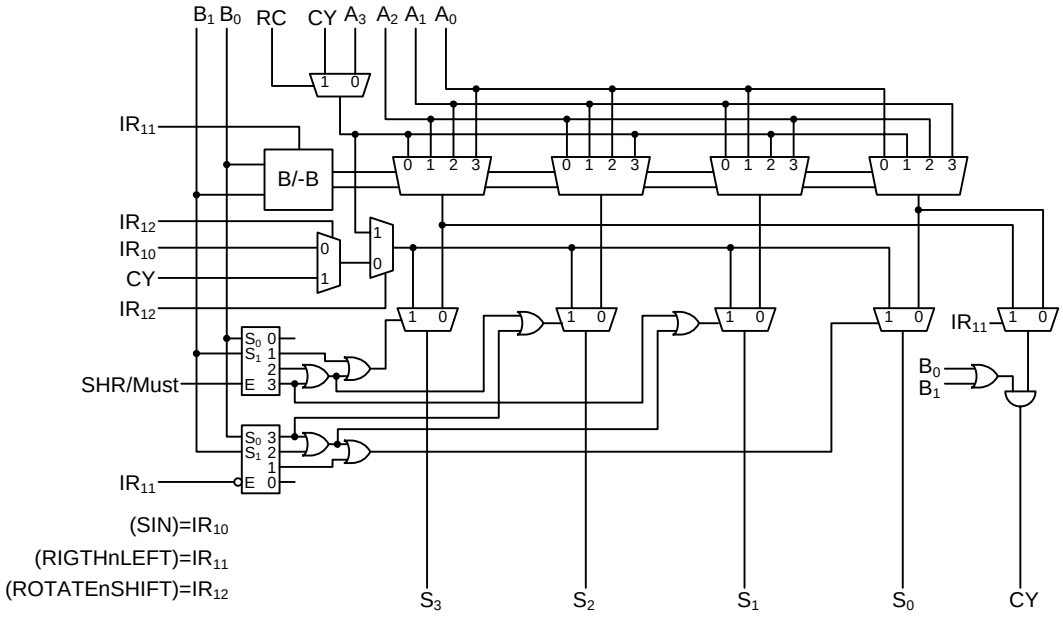

Figura 14-6 – *Barrel shift* de 4 bits

O conjunto dos quatro multiplexers de 4x1 realizam a rotação do conjunto de todos os bits, ou seja, os bits que saem pela esquerda entram pela direita e vice-versa, pelo que rodar por exemplo para a esquerda 3 bits é o mesmo que rodar para a direita 1 bit. Esta é a razão da existência do bloco que realiza a operação B ou –B, função do deslocamento ser para a esquerda ou para a direita.

14.1.3 BIU (*Bus Interface Unit*)

No sentido de tornar independente a arquitectura interna do bus de dados, endereços e controlo, bem como de outros sinais associados à comunicação com o exterior foi criada uma interface denominada BIU.

Este módulo tem a estrutura apresentada na Figura 14-7.

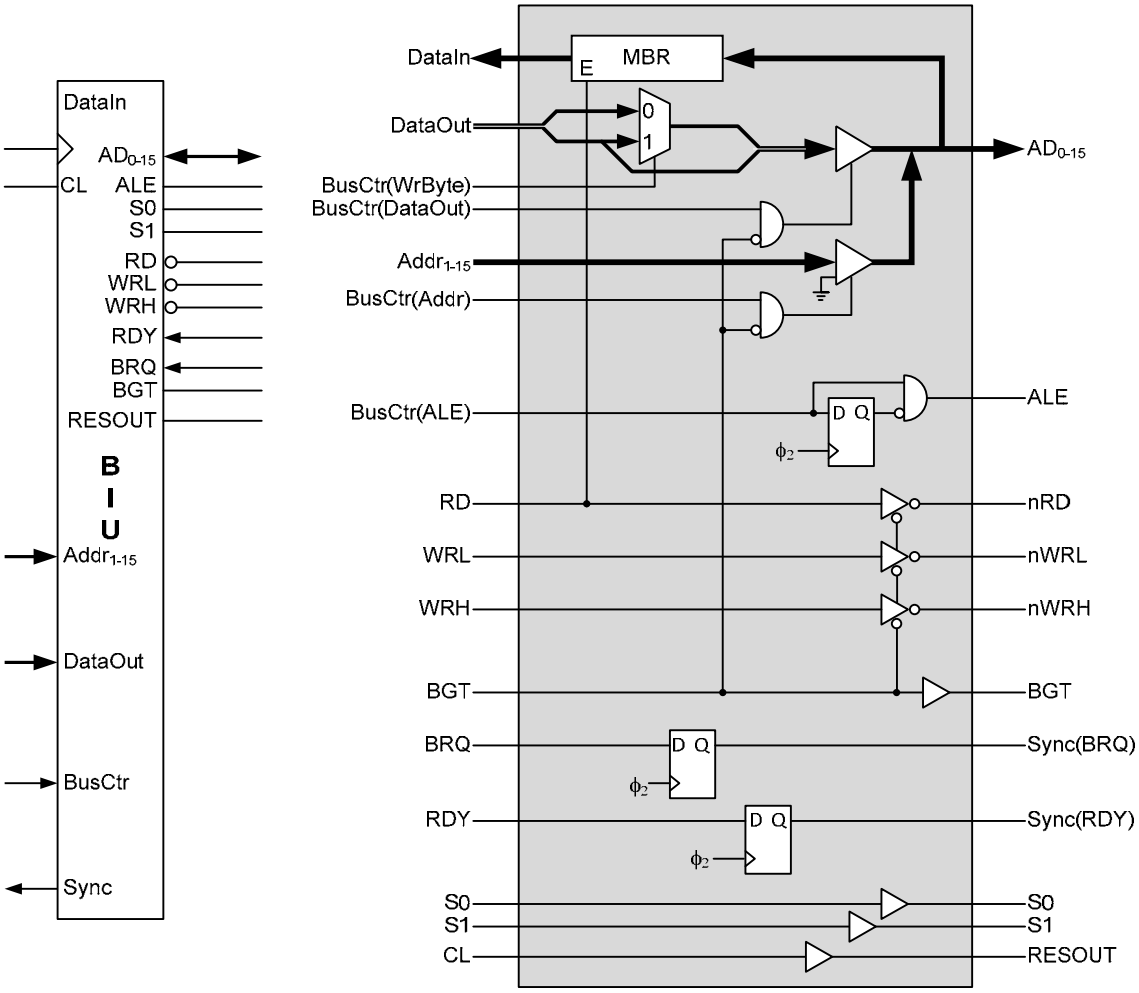

Figura 14-7 – Diagrama de blocos do *BIU*

O multiplexer associado ao **DataOut** tem como objectivo colocar na parte alta do bus de dados o mesmo valor que existe na parte baixa, quando se trata de um ciclo de escrita de um byte, pois o byte a ser escrito na memória é sempre o byte de menor peso do registo, independentemente do valor do endereço onde vais ser escrito ser par ou impar.

O *flip-flop* associado ao sinal **ALE**, tem por objectivo gerar um sinal com a duração de meio período do sinal MCLK. O *flip-flop* encontra-se a zero no momento de activação do sinal **BusCtr/ALE** (transição ascendente de ϕ_1) o que leva á imediata activação do pino **ALE**. A desactivação do pino ALE dá-se com a transição ascendente de ϕ_2.

14.2 Especificação dos sinais e organização do BUS

14.2.1 MCLK (*Master Clock*)

O PDS16 é uma estrutura síncrona, ou seja, as várias acções são-lhe determinadas por um sinal de *clock* ligado ao pino MCLK e que é proveniente de uma fonte externo de *clock*. É este sinal que define a sequência de estados que leva o PDS16 a executar as instruções de um programa, bem como o ritmo a que as executa.

O MCLK é um sinal com *duty cycle* de 50% e frequência entre DC e 500Hz cujo diagrama temporal é mostrado na Figura 14-8.

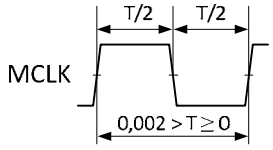

Figura 14-8 – *Master Clock*

14.2.2 RESET e RESOUT

RESET é o sinal de que leva o CPU ao estado inicial e que corresponde a colocar a máquina de estados no estado zero, os registos PC e PSW com o valor zero. Esta entrada é activada com o valor lógico zero e tem característica assíncrona, o que implica que enquanto estiver activa o CPU permanece no estado de reset. A activação do sinal RESET leva a que o PDS16 active a saída RESOUT. Após a desactivação do sinal RESET o PDS16 permanece ainda no estado de reset até que surja uma transição descendente do sinal MCLK, conforme é mostrado no diagrama temporal da Figura 14-9. Este sinal tem como objectivo informar os dispositivos externos ao CPU que este sofreu uma acção de reset.

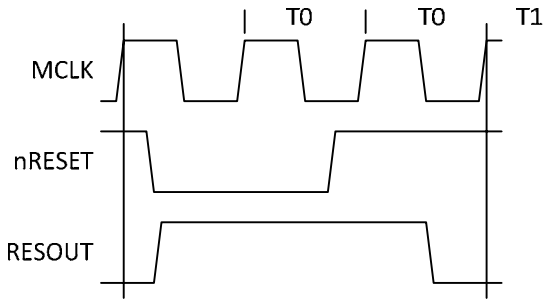

Figura 14-9 – RESET e RESOUT

14.2.3 BUS de endereços e dados (AD_{0-15})

No sentido de diminuir o número de pinos do CPU, um mesmo bus de 16 bits será utilizado para transferir dados e endereços. Para tal o CPU disponibiliza um sinal denominado ALE (*Address Latch Enable*) para validar como endereço a informação presente no bus AD_{0-15}. Neste mesmo bus circula informação de dados que o CPU valida com os sinais nRD, nWRL e nWRH.

Este facto vai obrigar a que a arquitectura do sistema inclua obrigatoriamente um LATCH de 16 bits para registar o endereço, garantindo-se desta forma a estabilidade do bus de endereços durante

todo o ciclo de leitura ou escrita. Na Figura 14-10 é mostrado o diagrama bloco de um sistema mínimo baseado no PDS16_V2.

Como se pode observar na Figura 14-10, embora o espaço de memória do PDS16 seja de 32K x 16, este está organizada em dois espaços de 32K x 8 bits. Esta organização, deve-se ao facto das instruções de transferência de dados entre o CPU e a memória, poderem especificar se uma transferência é realizada a 8 bits ou a 16 bits.

No caso da leitura de um byte, o CPU activa o sinal de RD, lê os 16 bits correspondentes ao endereço estabelecido pelo LATCH e, internamente ao CPU, selecciona a parte alta ou a parte baixa função do valor lógico do bit de endereço A_0 ser respectivamente 0 ou 1 (ver o diagrama de blocos do PDS16_V2 Figura 14-1).

No caso da escrita de um byte, o CPU coloca nos 8 bits de maior e menor peso do bus de dados, os 8 bits de menor peso do registo fonte **RS,** e activa o sinal WRH ou WRL função do valor lógico do bit de endereço A_0. Um endereço par corresponde ao byte de maior peso (ver BIU Figura 14-7). Se a escrita for de 16 bits activa em simultâneo os sinais WRH e WRL.

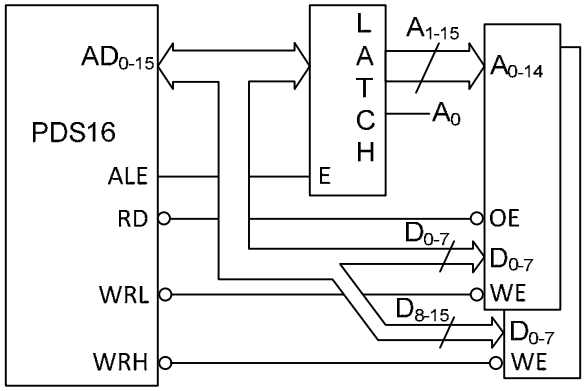

Figura 14-10 – Diagrama de blocos da multiplexagem do bus

14.2.4 Ciclos de leitura e escrita

Os ciclos de leitura e escrita do PDS16 têm o diagrama temporal mostrado na Figura 14-11.

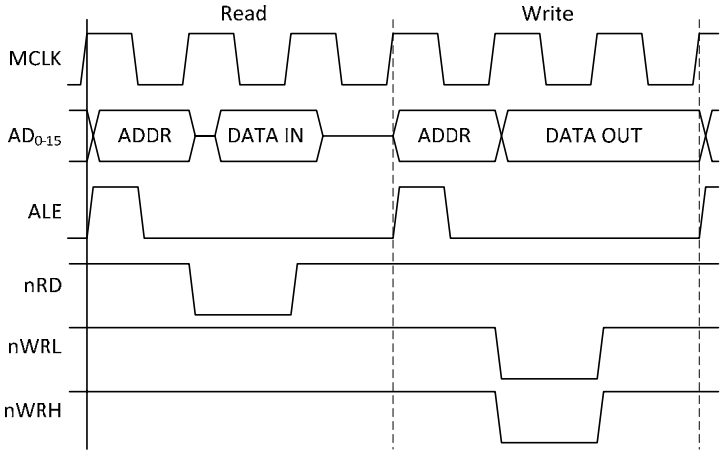

Figura 14-11 – Diagrama temporal de um ciclo de leitura e de escrita

Como se pode observar, no primeiro ciclo de MCLK prepara endereço, no segundo ciclo de MCLK realiza a leitura ou a escrita, o terceiro ciclo serve, na leitura, para esperar a libertação do bus por parte do dispositivo lido e, na escrita, para garantir T_{DH} ao dispositivo que foi escrito. Estes ciclos são normalmente denominados por ciclos máquina.

14.3 Ciclos Máquina

Entende-se por ciclo máquina, o ciclo de leitura ou escrita na memória. O PDS16 tem dois tipos de ciclo máquina o ciclo OP CODE FETCH em que o CPU lê da memória o código da instrução a ser executada e o ciclo EXECUTE associado às instruções LD e ST, para leitura ou escrita em memória de um valor associado a uma variável. Na Figura 14-12 estão representados o *op code fetch* e o *execute* de duas instruções; uma só com processamento interno e outra com acesso de escrita à memória.

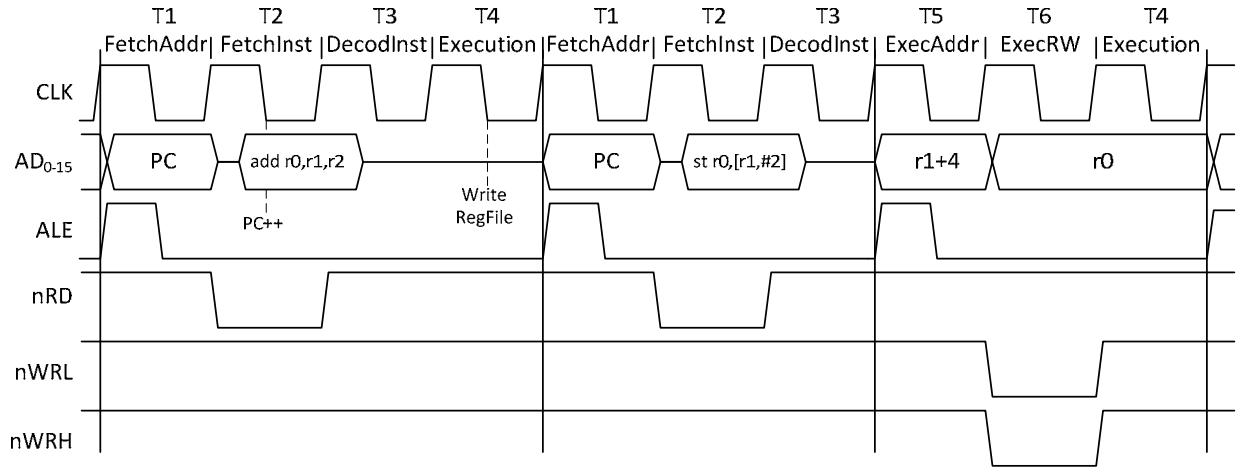

Figura 14-12 – Ciclos FETCH e EXECUTE

O ciclo OP CODE FETCH consiste numa sequência de 3 estados de T1 a T3, sendo T1 para preparação de endereço, T2 para leitura da instrução e T3 para descodificação da instrução. O ciclo de EXECUTE consiste num único estado T4, antecedido por um ciclo de leitura, caso a instrução em execução seja LD, ou por um ciclo de escrita, caso a instrução seja ST. No caso da instrução ST, o estado de execução não realiza nenhuma acção internamente ao processador. No caso da instrução LD, a meio do estado de execução, dá-se a escrita no *Register File*. É de notar que o registo PC é incrementado a meio de T2, pelo que a execução de uma instrução que utilize o conteúdo do registo PC (R7) tem que ter em atenção que o seu valor é o do endereço da instrução seguinte.

Em qualquer das arquitecturas até aqui estudadas, a leitura de uma nova instrução só tem início após a descodificação e execução da instrução anterior. Este tipo de implementação há muito que foi abandonada pela ineficiência que apresenta, relativamente à frequência máxima a que a estrutura pode ser sujeita. Para ultrapassar este facto, foram desenvolvidas arquitecturas com melhor desempenho, denominadas de *pipe line*. Estas arquitecturas não serão abordadas no âmbito deste documento, embora o seu princípio já tenha sido aflorado aquando do estudo das técnicas de implementação das máquinas de estado no sentido de aumentar a frequência de funcionamento.

14.4 Sincronização e interacção com dispositivos externos

Dada a estrutura do PDS16, com um único espaço de endereçamento, subentende-se que este espaço irá ser povoado por diferentes tipos de dispositivos. Teremos dispositivos de memória não volátil (ROM, flash, etc.) para conter código, dispositivos de memórias suportando leitura e escrita (SRAM, DRAM, etc.) para conter dados e ainda a existência de dispositivos que permitam a entrada e saída de dados para o exterior do sistema. Esta multiplicidade de dispositivos e interacções implica que o processador tenha mecanismos de sincronização, que lhe permita adaptar-se as especificidades temporais de cada um dos dispositivos sem comprometer o desempenho global do sistema.

14.5 RDY

A inserção de um dispositivo lento (TOE e TWP muito grande, ver Capitulo 12 Figuras 12-2,3) no sistema não deve implicar uma velocidade uniforme do CPU tal que a temporização desse dispositivo seja satisfeita. Com este objectivo foi adicionada ao PDS16 uma entrada denominada RDY (*Ready*) que, quando activa, significa que o CPU poderá finalizar a acção de leitura ou escrita em curso. Este sinal será testado pelo CPU antes de finalizar o ciclo de leitura ou escrita e, caso este se encontre desactivo, o CPU entra em estado de espera (*Wait state*), prolongando indefinidamente o ciclo em curso até que o sinal RDY se torne activo. No diagrama temporal da Figura 14-13 é mostrado o instante de observação do sinal RDY (a meio do sinal nRD, nWRL ou nWRH) e a respectiva consequência no comportamento do bus do CPU.

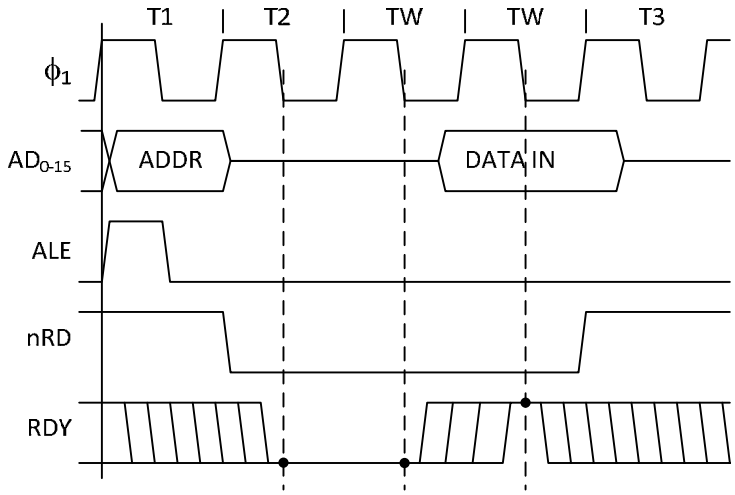

Figura 14-13 – Diagrama temporal do sinal RDY num ciclo de leitura

No exemplo da Figura 14-13 podemos observar que foram gerados dois estados de *Wait* o que levou a que o sinal nRD ficasse activo durante 3 ciclos do sinal de *clock*. A gestão desta entrada pode ser realizada numa de duas formas, nomeadamente: *normally ready* ou *normally not ready*. No modo *normally ready* a entrada RDY é mantida sempre activa sendo colocado a zero pelo dispositivo endereçado, caso pretenda gerar estados de espera. No modo *normally not ready* o sinal

RDY é mantido a zero e caso o dispositivo endereçado não pretenda requerer estado de espera activa a entrada RDY. Este último modo, apresenta como vantagem, permitir que a lógica associada ao dispositivo endereçado seja mais lenta a reagir sem correr o risco de não ser produzido o tempo de espera que lhe era necessário.
Para além da situação do sistema poder ser constituído por dispositivos com tempos de acesso muito díspares, este sinal também se torna indispensável num cenário onde o sistema de computação é constituído por vários CPUs com recursos próprios mas partilhando um bus comum para acesso a dispositivos do sistema. Neste cenário pode acontecer que um qualquer CPU ao tentar aceder ao bus comum, esteja em curso um acesso por parte de outro CPU. Esta situação leva a que o elemento que realiza a arbitragem dos acessos ao bus comum, desactive o sinal de RDY do CPU que pretende aceder, até à conclusão da transferência em curso.
O sinal RDY também é comummente utilizado pelo projectista de sistemas para detectar e corrigir erros de concepção ou de implementação tanto de hardware como de software. Como se pode observar na Figura 14-13, ao desactivar a entrada RDY, o CPU como que cristaliza os *buses* de endereço, dados e controlo, permitindo ao projectista verificar de forma estática qual o comportamento da lógica associada ao dispositivo que está a ser endereçado e qual o valor do dado que está a ser transferido. Se for construído um circuito que ciclo a ciclo máquina desactive o sinal de RDY, podemos seguir a execução do CPU passo a passo.

14.6 Partilha de BUS

Um outro cenário comum nos sistemas baseados em microprocessadores, que tem como objectivo diminuir o tempo de transferência de dados entre um dispositivo de I/O e a memória, é a transferência por DMA (*Direct Memory Access*), ou seja, o sistema dispõe de um dispositivo de DMA que utilizando o bus do CPU realiza a transferência de informação entre o dispositivo de I/O e a memória, sem intervenção do CPU. Nesta situação, o CPU ao receber um pedido do dispositivo de DMA, liberta os *buses* de endereço, dados e controlo (nRD e nWR), colocando-os em alta impedância e entra em estado de espera passiva (*hold*)**,** informando simultaneamente o dispositivo DMA que a transferência já se pode realizar. A libertação do BUS, ou seja, a entrada do CPU em estado de *hold*, só pode acontecer quando o ciclo de acesso à memória que por ventura esteja a decorrer tenha sido concluído, o que implica, que o dispositivo requerente do bus, aguarde por sinalização do CPU para iniciar a transferência.

BRQ/BGT

Para implementar esta funcionalidade é necessário adicionar ao CPU dois sinais, um de entrada, denominado BRQ (Bus *Request*) que o dispositivo requerente activa quando pretende realizar uma transferência e, outro de saída, denominado BGT (*Bus Grant*), que o CPU utiliza para informar o sistema requerente que o bus já foi libertado. O CPU permanecerá em estado de *hold* enquanto o sinal BRQ estiver activo e durante este tempo o CPU manterá activo o sinal BGT.
No diagrama temporal da Figura 14-14 é mostrado o instante de observação do sinal BRQ e a respectiva consequência no comportamento do CPU e como se pode observar BRQ é amostrado a meio do ciclo de *clock* que antecede o início de um acesso à memória (T3 ou T4) e caso este se encontre activo entra em estado de *hold* activando o sinal BGT. Enquanto em estado de *hold*, fica a amostrar a entrada BGT a meio dos ciclos TH

Uma vez que um pedido de BRQ só é satisfeito quando o ciclo em curso for concluído, se o CPU estiver em estado de *Wait* (RDY desactivo), só indicará BGT quando for libertado pelo sinal RDY e concluir o ciclo que estava em curso. Em caso de pedido *wait* e *hold* simultâneo, dado que o teste a estes sinais é feito após se ter iniciado o ciclo de leitura ou escrita, o sinal RDY tem prioridade sobre o sinal BRQ, caso contrário implicava interromper um dos ciclos.

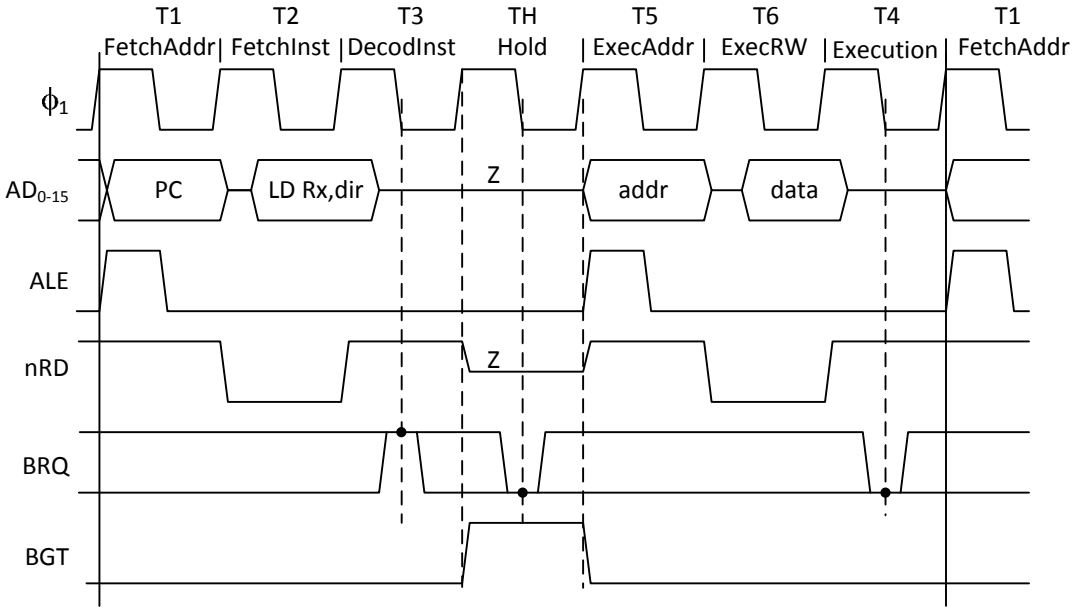

Figura 14-14 Diagrama temporal dos sinais BRQ e BGT

Como se pode observar na Figura 14-7, a adição destes três sinais RDY, BRQ e BGT implica a adição de um registo para sincronização dos sinais RDY e BRQ com o flanco descendente de *clock* (ϕ2), para garantir a estabilidade dos sinais aquando do teste por parte do módulo de controlo do CPU.

O módulo de controlo passará a ter o seguinte comportamento quanto à transição de estados:

```
sequence [ASM_curr_stat] {
  presente RESET
                  if (Q_brqFF) next HOLD_EXEC;
                  if (!Q_brqFF) next FETCH_ADDR;
  presente FETCH_ADDR:
                  next FETCH_INST;
  presente FETCH_INST:
                  if (!Q_rdyFF)next WAIT_FETCH;
                  if (Q_rdyFF) next FETCH_DECOD;
  presente FETCH_DECOD:
                  if (loadStoreMem & Q_brqFF) next HOLD_EXEC;
                  if (loadStoreMem & !Q_brqFF) next EXEC_ADDR;
                  default next EXECUTION;
  presente EXECUTION:
                  if (Q_brqFF) next HOLD_FETCH;; return;
                  if (Q_intpFF & !Q_brqFF) next INTERRUPT;
                  if (breakCondition & !Q_intpFF & !Q_brqFF) next BREAK;
                  default next FETCH_ADDR;
```

```
                break;
  presente EXEC_ADDR:
                next EXEC_RW;
  presente EXEC_RW:
                if (!Q_rdyFF) next WAIT_EXEC;
                default next EXECUTION;
  presente INTERRUPT:
                if (Q_brqFF) next HOLD_FETCH;
                if (breakCondition() & !Q_brqFF) next BREAK;
                default next FETCH_ADDR;
  presente BREAK:
                if (!Go()) next BREAK;
                default next FETCH_ADDR;
  presente HOLD_FETCH:
                if (Q_brqFF) next HOLD_FETCH;
                if (breakCondition()& !Q_brqFF) next BREAK;
                default next FETCH_ADDR;
  presente HOLD_EXEC:
                if (Q_brqFF) next HOLD_EXEC
                default next EXEC_ADDR;
  presente WAIT_FETCH:
                if (!Q_rdyFF) next WAIT_FETCH;
                default next FETCH_DECOD;
  presente WAIT_EXEC:
                if (!Q_rdyFF)next WAIT_EXEC;
                default next EXECUTION;
}
```

14.7 Informação de estado (S1, S0)

O PDS16 codifica, para o exterior, através de dois pinos (S_1 e S_0) a natureza do ciclo em que o CPU está envolvido. Estes sinais são posicionados no início de T1 e assim permanecendo ao longo de todo o ciclo, permitindo assim saber antecipadamente qual o tipo de ciclo que o CPU vai realizar. Na Tabela 14-1 estão representados os códigos associados a cada um dos estados.

S_1	S_0	Estado
0	0	*Fetch*
0	1	*Execute*
1	0	*Break*
1	1	*Interrupt*

Tabela 14-1