12. PDS8_V2 12-2
12.1 Estrutura do PDS8_V2 12-5
12.2 Diagrama de blocos do PDS8_V2 12-8
12.2.1 Módulo de controlo 12-10
12.2.2 ASM do módulo de controlo 12-11
12.2.3 ROM de descodificação 12-13

12. PDS8_V2

O módulo de controlo do PDS8_V1, não levou em consideração os tempos de reacção dos dispositivos endereçados.

No sentido de corrigir estas deficiências e de aproximar o PDS8_V1 das arquitecturas reais, vamos construir uma nova versão que denominaremos PDS8_V2, que em relação à anterior apresenta as seguintes alterações:

- Implementação do controlo de modo a produzir as necessárias temporizações no acesso aos dispositivos de memória;
- No sentido de se aproximar das arquitecturas reais, passaremos do modelo *Harvard* a *Von Neumann*, ou seja, utilização de um único espaço de memória para código e dados, como mostra a Figura 12-1.

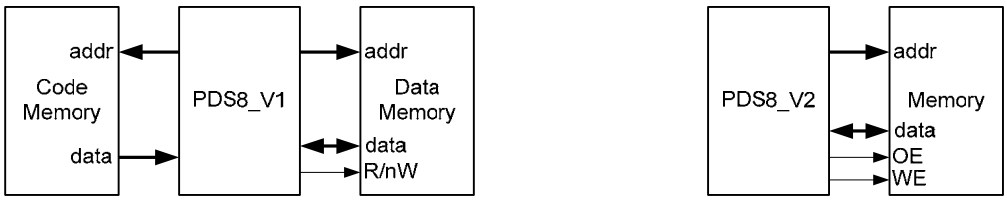

Figura 12-1 – Diagrama de blocos: Modelo *Havard* e *Von Neumann*

Antes de introduzirmos estas alterações, é importante ter presente o diagrama temporal dos ciclos de leitura (*read*) e de escrita (*write*) dos dispositivos de memória ROM e RAM estática.

A memória RAM estática poderá ser controlada de duas formas:

- um sinal **R/nW** que selecciona leitura ou escrita e outro **CE** que inibe ou desinibe a acção seleccionada.
- um sinal **OE** para accionar a leitura e outro **WE** para accionar a escrita, sendo estes sinais activados em exclusão. Para concatenação de dispositivos dispõe de um sinal **CE** para inibição e desinibição.

As memórias RAM disponíveis no mercado, permitem as duas formas de funcionamento, no entanto, por simplicidade iremos adoptar a segunda forma, o que levará a adicionar dois sinais ao CPU, um para controlo da leitura **RD** e outro para escrita **WR**.

Ciclo de leitura

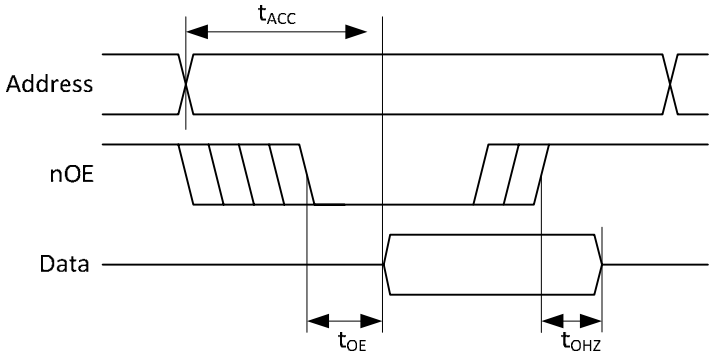

Figura 12-2 - Diagrama temporal de um *Read cycle*

t_{ACC} (*time ACCess*) tempo mínimo de acesso à informação.

t_{OE} (OE *to Output valid*) tempo que medeia entre a activação do sinal nOE e a presença no data bus de informação estável em baixa impedância posta disponível pelo dispositivo.

t_{OHZ} (*Output Disable to Output High Z*) tempo que medeia entre a desactivação do sinal nOE e libertação do data bus por parte do dispositivo.

Ciclo de escrita

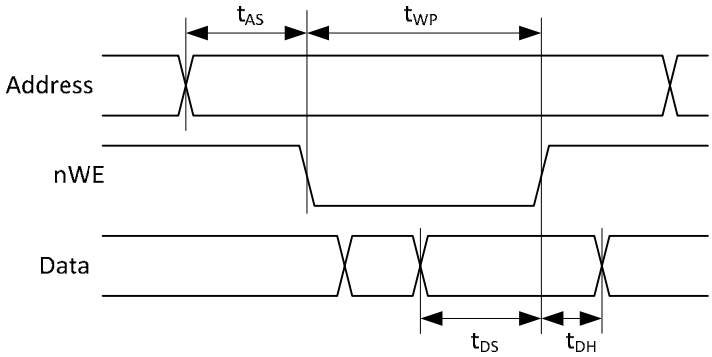

Figura 12-3 – Diagrama temporal de um *Write cycle*

t_{AS} (*Address Setup Time*) intervalo de tempo mínimo a respeitar entre o estabelecimento de um endereço e a activação do sinal nWE.

t_{WP} (*Write Pulse Width*) duração mínima do sinal nWE.

t_{DS} (*Data Setup Time*) intervalo de tempo mínimo a respeitar entre o estabelecimento de informação válida no data bus e a desactivação do sinal nWE.

t_{DH} (*Data Hold Time*) tempo mínimo durante o qual ainda se torna necessário manter os dados estáveis no bus após ter terminado o sinal de nWE.

Como se pode observar no diagrama temporal do ciclo de escrita, ver Figura 12-2, só é possível activar o sinal WE após a estabilização do endereço. Por outro lado é necessário manter o endereço e os dados estáveis depois de desactivar o sinal WE. No caso do PDS8_V1, esta especificação da memória de dados não é respeitada. Quanto ao ciclo de leitura, o comportamento do PDS8_V1 produz um conflito no bus de dados, pois quando passa de leitura a escrita não espera que o dispositivo de memória liberte o bus de dados (t_{OHZ}).

Se observarmos o diagrama temporal do ciclo de leitura, ver Figura 12-3, não existe nenhum problema se variarmos o endereço quando o sinal RD está activo, no entanto, como veremos mais adiante, o espaço de memória do CPU vai ser povoado por outro tipo de dispositivos além da memória, como sejam dispositivos para entrada e saída de dados para o exterior do sistema. Se imaginarmos um dispositivo que está a receber uma cadeia de caracteres, e que cada vez que é lido pelo CPU entrega um carácter da cadeia recebida, facilmente se percebe que a leitura descontrolada de um endereço de memória é neste caso nefasta. Por esta razão o CPU deverá gerar tempos de guarda em torno do sinal RD relativamente à variação dos endereços.

Como já vimos anteriormente, no PDS8_V1, o processamento é concretizado pela sucessão das acções *fetch execute*. Mantendo a mesma lógica, ou seja, após a leitura de uma instrução (*fetch*), passamos à execução, e assim sucessivamente, poderemos pensar num diagrama temporal para a actividade dos *buses* de endereços, dados e controlo do CPU como é mostrado na Figura 12-4.

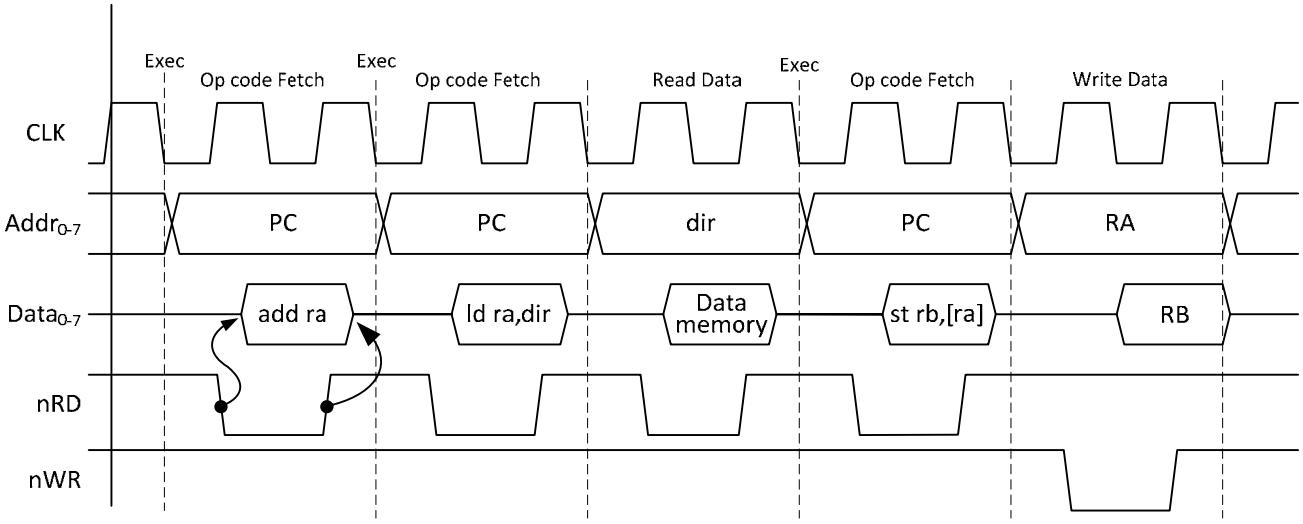

Figura 12-4 – Diagrama temporal de actividade dos *buses*

A Figura 12-4 apresenta três exemplos: um ciclo de *fetch* para uma instrução que tem execução interna ao CPU; o *fetch* de uma instrução cuja execução implica a leitura de um dado da memória; o *fetch* de uma instrução cuja execução implica a escrita de um dado na memória. Os vários ciclos do CPU têm em conta os diagramas temporais característicos dos dispositivos de memória e tendem a minimizar os tempos de acesso para assim se obter um melhor desempenho do CPU.
Dado que é necessário gerar temporizações para os vários sinais, leva a que a implementação do módulo de controlo do PDS8_V2, recorra a uma máquina de estados síncrona.

Como é sabido, nas máquinas de estado síncronas, as evoluções de estado dão-se com o intervalo mínimo de um período do sinal MCLK, enquanto as saídas, se gerarmos um sinal em contra fase com o sinal MCLK, poderemos ter acontecimentos com duração de meio período de MCLK.
Assim sendo, e como acontecia no PDS8_V1, poderemos gerar dois sinais $\phi 1$ e $\phi 2$, em contra fase, um para o módulo de controlo e outro para o módulo funcional.

12.1 Estrutura do PDS8_V2

A nova especificação do PDS8_V2 implica as seguintes alterações à estrutura do PDS8_V1:

- Como vimos anteriormente, durante todo o ciclo de leitura ou escrita, é necessário assegurar a estabilidade do bus de endereços. No PDS8 o bus de endereços é estabelecido pelos conteúdos dos registos PC, RA ou RA+RB, através de um multiplexer. No sentido de assegurar a estabilidade do bus de endereços iremos adicionar um registo denominado MAR (***M**emory **A**ddress **R**egister*) no qual, no início de cada ciclo de acesso à memória, é registado o endereço que se pretende aceder para leitura ou escrita;

- Como se pode observar na Figura 12-4, no ciclo máquina de leitura de memória (*fetch* ou *execute*), o sinal RD fica activo durante um período de MCLK. Internamente ao CPU a informação lida da memória pode ser captada em dois momentos distintos: a meio do sinal de RD com a transição descendente de MCLK, ou no final do sinal de RD. A captação a meio do sinal RD implica que o t_{OE} do dispositivo acedido, somado com o tempo de propagação no bus, somado com o t_{DS} do registo do CPU não possa exceder meio período de MCLK. Por esta razão iremos adoptar a captação no final do sinal RD. A captação no final do RD leva a que o momento em que o CPU armazena internamente a informação lida, é o mesmo em que indica à memória a finalização da leitura. Esta simultaneidade leva a que se possam gerar meta-estados no registo que estava a captar a informação, adulterando a informação lida. Para evitar este facto, vamos adicionar um registo, denominado por MBR (*Memory Buffer Register*). Este registo deverá ter como característica principal um t_{DH} muito pequeno a fim de evitar a meta-estabilidade. A introdução do registo MBR como elemento de recepção dos dados vindos do exterior, tem também como vantagem apresentar ao exterior um único elemento da estrutura, definindo assim, uma impedância característica de entrada do bus de dados do CPU;

- Na fase *fetch*, o processador lê da memória a instrução a executar. Como a instrução tem que permanecer na entrada do módulo de controlo até ao momento da execução, é necessário adicionar um registo para armazenar a instrução lida da memória. Este registo normalmente denominado por IR (***I**nstruction **R**egister*), assegura a estabilidade dos vários sinais durante a fase de execução e da preparação do próximo valor do registo PC;

- Ao registo PC é adicionado uma entrada de *Enable* para condicionar a sua evolução a dois momentos: na fase *fetch* para incrementar de 1, e na fase execute da instrução jump.

- A acção de Reset, embora com início assíncrono passa a ter finalização síncrona;

A nova estrutura assim constituída apresenta um módulo funcional relativamente complexo e implica sequencialidade nas transferências entre registos, tornando enfadonha e confusa a sua descrição através de um ASM e tabelas de verdade. Por esta razão iremos recorrer numa fase inicial à descrição da arquitectura em *basic schemata*.

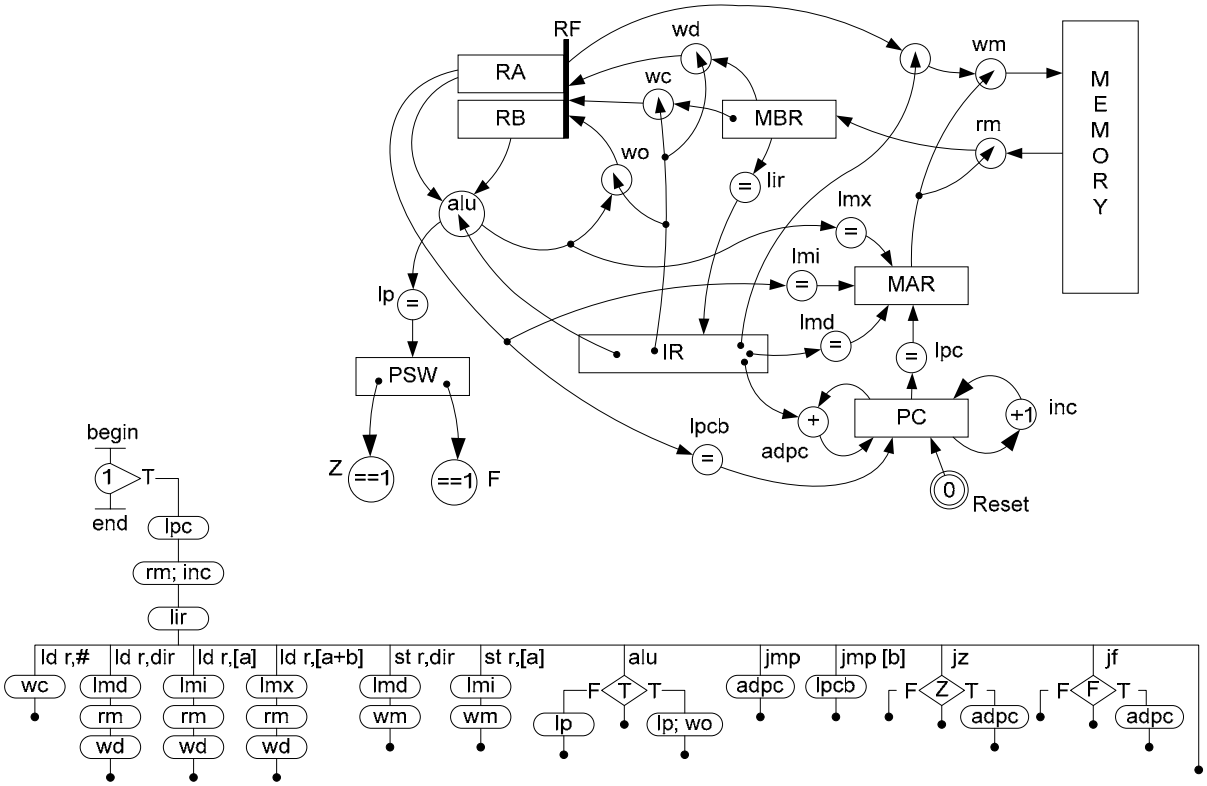

Figura 12-5 – *Basic schemata* do PDS8_v2 (EFI e ESA)

Como se pode observar no EFI da Figura 12-5, com a passagem ao modelo *Von Neumann*, o espaço de memória é único levando a que o registo MAR tenha como fonte, os registos PC para *Op code fetch* e IR, RA e a ALU para *execute*. A interface entre o CPU e a memória é concretizada pelo registo MAR que estabelece o endereço e pelo registo MBR que capta a informação lida. Na escrita, os dados são fornecidos pelo registo RA ou RB.

Os registos RA e RB são acedidos de duas formas: de forma individualizada, como acontece quando são operandos da ALU; ou como estrutura de dois registos endereçáveis pelo bit R presente no IR.

Descrição dos vários operadores presentes no EFI do PDS8_V2:

adpc	PC=PC+IR.offset	;add PC	lpc	load PC	;MAR=PC
inc	PC++	;inc PC	rm	Read mem	;mbr=memory[MAR]
lir	IR=MBR	;load IR	wc	write const	;RF[IR.R]=MBR.const
lmd	MAR=IR.direct	;load MAR direct	wd	write direct	;RF[IR.R]=MBR. direct
lmi	MAR=RA	;load MAR Indirect	wm	write mem	;memory[mar]=RF[IR.R]
lmx	MAR=RA+RB	;load MAR Indexed	wo	write result	;RF[IR.R]=alu(IR.op,RA,RB)
lp	PSW=F,Z	;load PSW			

Como se pode observar no ESA da Figura 12-5, o *fetch* é igual para todas as instruções. O fetch é concretizado pela seguinte sequência de acções: carrega no MAR o valor do PC para estabelecer o endereço da próxima instrução a ser executada; o código da instrução apontado pelo MAR é lido para o MBR; o valor do PC é incrementado no sentido de apontar para a próxima instrução; por fim o código da instrução é transferido para o registo IR. A operação de escrita sobre o conjunto dos registos RA e RB, que é concretizada pelos operadores wd, wc e wo, é vectorizada pelo bit R contido no código da instrução.

12.2 Diagrama de blocos do PDS8_V2

Na Figura 12-6 está representado um possível diagrama de blocos da arquitectura PDS8_V2, no qual podemos observar as alterações propostas relativamente ao PDS8_V1.

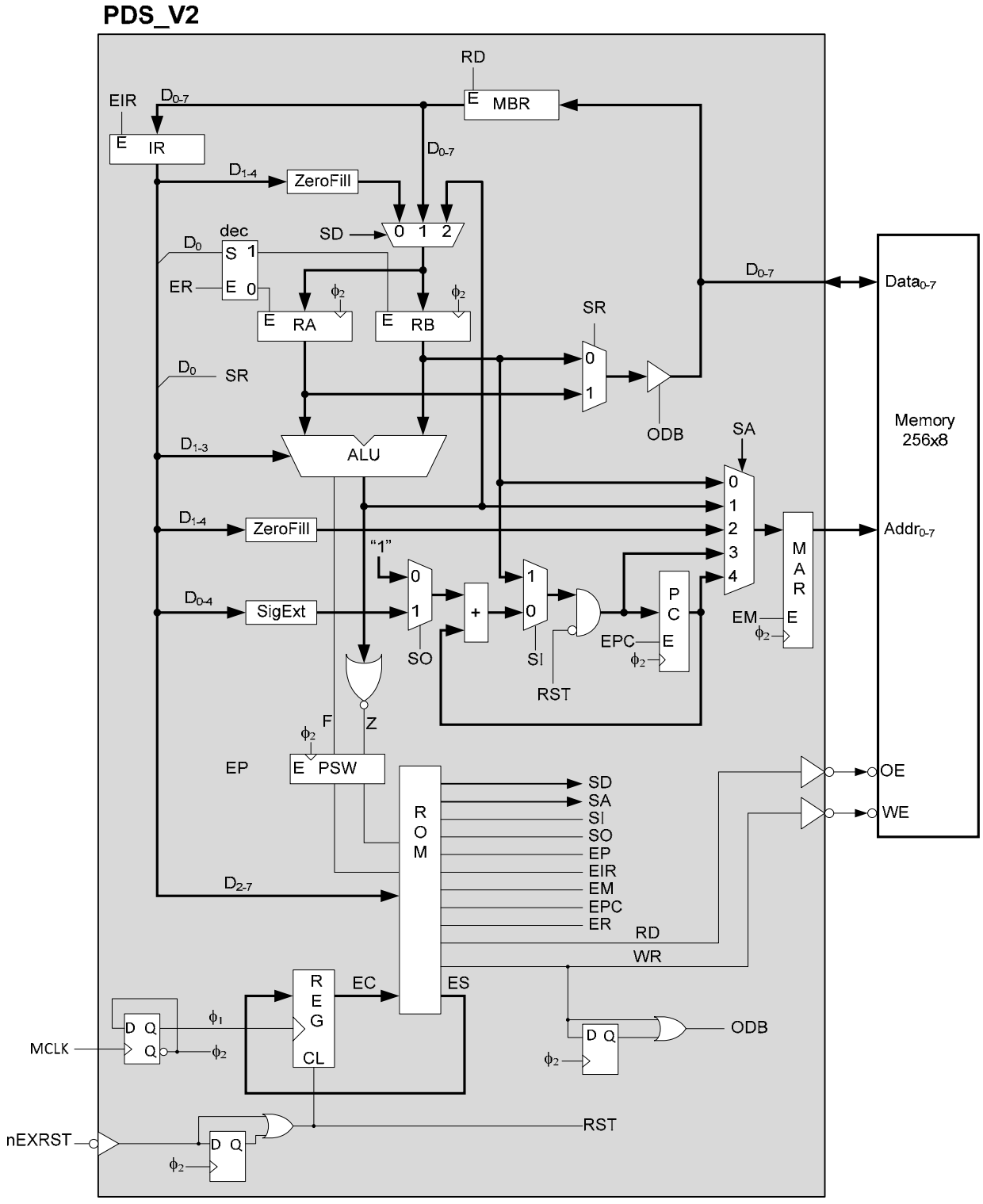

Figura 12-6 – Diagrama de blocos do PDS8_V2

Os registos constituintes da estrutura têm a seguinte especificação:

- Os registos RA, RB, PSW, PC, MAR e PC, são síncronos (*edge trigger*) com controlo de ***E**nable*;
- Os registos MBR e IR, são LATCH;

Como se pode observar na Figura 12-6 todos os registos síncronos do CPU são afectados na transição ascendente do sinal $\phi 2$.

Vejamos então a funcionalidade desempenhada por cada um dos sinais que controlam o comportamento da estrutura.

Descrição dos sinais de entrada:

O sinal MCLK (*master clock*), tal com acontecia no PDS8_V1, é dividido por 2 gerando dois sinais $\phi 1$ e $\phi 2$, em contra fase, um para o módulo de controlo, outro para módulo funcional, no sentido de gerar um diagrama temporal idêntico ao da Figura 12-4.

A activação da entrada **EXRST** leva a que os registos PC e MAR tomem o valor zero, e simultaneamente coloca o registo de estado corrente do controlo a zero. O início da acção de reset é assíncrona, mas a finalização é síncrona com o MCLK.

Descrição dos sinais de controlo:

- **SI** (*Select index*) permite realizar a instrução **`jmp RA`**, pois quando activo selecciona para a entrada do PC o valor de RA, estabelecendo um novo endereço de *fetch*;
- **SO** (*Select Offset*) determina se o próximo fetch se realiza no endereço dado por PC+1, ou em PC mais o parâmetro offset contido na instrução;
- processamento;
- **SD** (*Select Data*) selecciona qual a informação a carregar nos registos RA ou RB. Para a instrução **`LD,#const4`** selecciona a constante que é parâmetro da instrução. Na instrução **`LD,direct4`** selecciona o valor que está a ser lido da memória de dados. Nas instruções de processamento, selecciona o resultado da ALU.
- **SA** (*Select Address*) selecciona qual o parâmetro que estabelece o endereço. No caso de **`LD/ST R,direct4`**, selecciona o parâmetro contido na instrução. Caso o modo de endereçamento seja indirecto, selecciona o registo de indirecção RB. Se o modo é indexado, selecciona o valor calculado pela ALU, no início do *fetch* selecciona o registo PC.
- **SR** (*Select Register*) é utilizado pela instrução **`ST R,(Endereço destino)`** para seleccionar qual dos registos RA ou RB vai fornecer o valor a ser escrito na memória de dados. O parâmetro R está contido na instrução.
- **ER** (*Enable Registers*) é utilizado pela instrução **`LD R,(parâmetro fonte)`** e pelas instruções aritméticas e lógicas. No caso da instrução LD, serve para registar em R o parâmetro fonte, estando o parâmetro R (RA ou RB) também contido na instrução. Nas instruções lógicas e aritméticas para registar em R o resultado da operação realizada pela ALU.
- **EP** (*Enable PSW*) controla a escrita das *flags* no registo PSW aquando das operações lógicas e aritméticas.
- **EPC** (*Enable PC*) controla a escrita no registo PC.

- **EM** (*Enable MAR*) controla a escrita no registo MAR no início da fase *fetch*.
- **EIR** (*Enable IR*) controla a escrita no registo IR.
- **RD** (*ReaD*) controla a leitura da memória, no sentido de informar o dispositivo endereçado que deve colocar em baixa impedância os dados a serem lidos pelo CPU.
- **WR** (*WRite*) controla a escrita na memória.
- **ODB** (Output Data Bus), dado que o bus de dados é bidireccional, este sinal controla a impedância de saída do bus de dados.
- **EC** (Estado Corrente) vector que representa o estado corrente da máquina de controlo.
- **ES** (Estado Seguinte) vector que representa o estado seguinte da máquina de controlo.

12.2.1 Módulo de controlo

Antes de desenharmos o módulo de controlo do PDS8_V2, é necessário estabelecer um diagrama temporal para a actividade do bus de dados, endereço e controlo do CPU, identificando o momento de cada acção. Neste sentido, é apresentado na figura abaixo, uma proposta de diagrama temporal dos ciclos máquina *Opcode Fetch*, *Data Read* e *Data Write* do PDS8_V2 com as seguintes premissas:

- Respeitar os diagramas temporais dos dispositivos acedidos;
- Uniformizar os ciclos de leitura e escrita da memória, tanto para fetch como para execução, no sentido de simplificar o módulo de controlo;
- Minimizar o número de *clocks* necessários à preparação e execução de uma instrução.

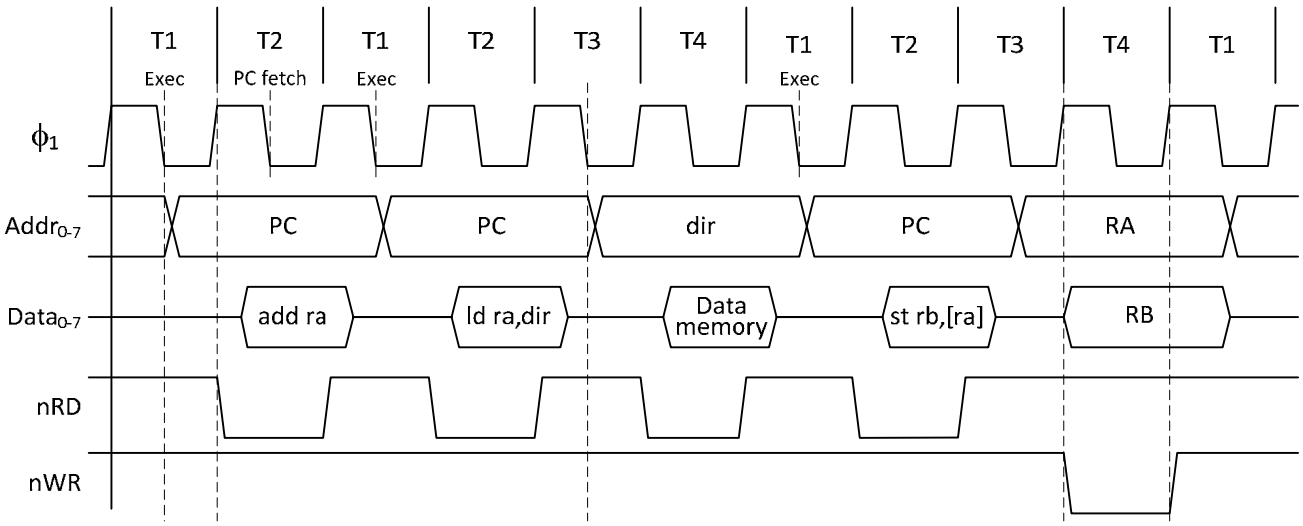

Figura 12-7 – Diagrama temporal dos *buses* do PDS8_V2

Assim sendo, o bus do PDS8_V2, apresenta as seguintes especificações:

- A duração do ciclo máquina de escrita ou de leitura são dois períodos de MCLK;
- O *Op code fetch* de uma instrução tem início a meio de T1 e termina no fim de T2;
- O *fetch* do registo PC é realizado a meio de T2;
- O código da instrução que vai ser executada é considerado disponível no CPU a meio de T2;

- A execução é realizada no fim de T2 ou T4, início de T1. Pelo que a descodificação tem que ser realizada em meio período do MCLK;
- No ciclo de escrita, como o sinal WR é activado no início T4, o tempo t_{AS} para estabilização dos endereços é de apenas meio período de MCLK. Quanto ao bus de dados fica em baixa impedância desde o início de T4 até meio de T1.

Como se pode observar no esquema bloco, o registo MAR tem como fonte o valor do registo PC e o valor a carregar no PC. A existência destes dois caminhos de dados para a entrada do MAR , deve-se ao facto de termos feito coincidir o instante de execução da instrução com início da fase *fetch* da próxima instrução. A necessidade de estabelecer como fonte do MAR o valor a carregar no PC só existe, quando a instrução a ser executada é um JUMP, pois o valor corrente do PC vai ser modificado no instante da execução, e é esse mesmo valor que corresponde ao endereço da próxima instrução a ser lida.
No esquema bloco também podemos observar que o circuito que gera o sinal ODB é sincronizado com $\phi 2$, no sentido de por disponíveis os dados a serem escritos desde o início de T4 até ao meio de T1. Esta temporização garante o t_{DH} do dispositivo seleccionado e beneficia o t_{DS} por antecipação dos dados.

12.2.2 ASM do módulo de controlo

Estabelecido o diagrama de blocos e o diagrama temporal do PDS8_V2, estamos em condições de desenhar um ASM para o controlo do CPU. Na Figura 12-8 está representado o ASM do módulo de controlo do PDS8_V2, que controla os vários sinais da estrutura, no sentido de executar as várias instruções, e gerar os vários sinais do bus com o encadeamento estabelecido no diagrama temporal da Figura 12-7.

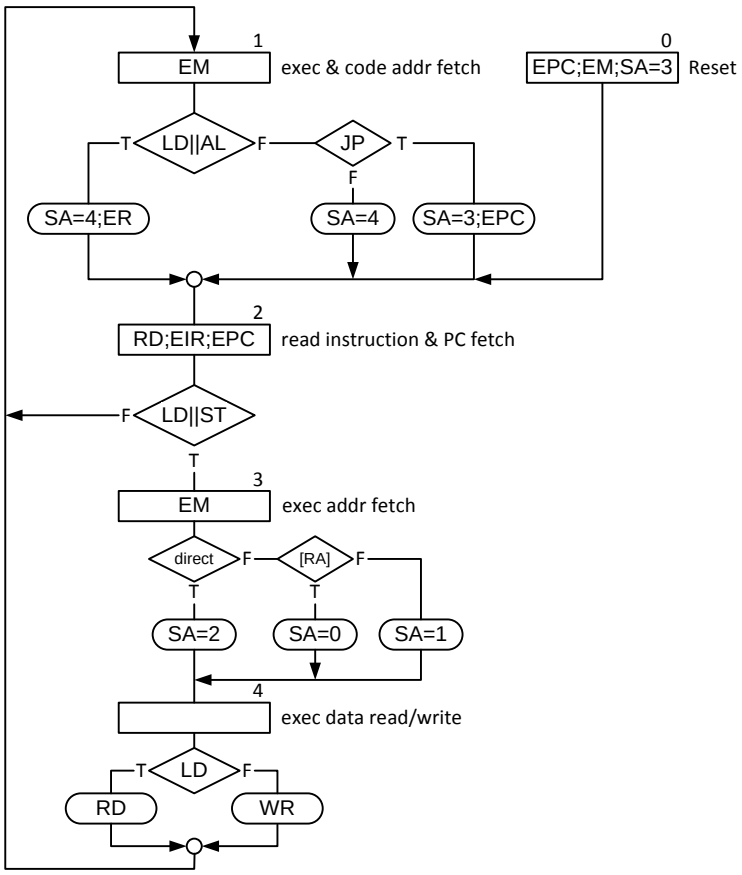

Figura 12-8 – *ASM-chart* – modulo de controlo

Estado 0: Estado inicial do controlo, só atingido por activação da entrada externa EXRST e aqui permanecendo enquanto EXRST se mantiver activa. O controlo, neste estado activa o sinal EM, o EPC e selecciona a entrada 3 do multiplexer do MAR para iniciar os registos PC e MAR com o valor zero.

Estado 1: O estado 1 corresponde ao estado de execução e preparação de endereço. Quanto à execução, se a instrução presente no IR é uma instrução de LOAD (LD) ou aritmética (AL) activa ER e selecciona a fonte de dados. Se a instrução é de JUMP (JP) e a condição é verdadeira, activa EPC. Quanto á preparação do endereço, é activado o sinal EM e caso a instrução não seja de salto, ou sendo de salto não exista condição, é sempre selecciona a entrada 4 do multiplexer do MAR, utilizando o valor corrente do PC. Se a instrução promove a alteração do PC, selecciona a entrada 3 do multiplexer do MAR para que o PC e o MAR tomem o mesmo valor.

Estado 2: Neste estado é realizada a leitura da instrução e o *fetch* do PC. Para tal o CPU activa o sinal de RD para ler da memória a instrução a executar e, simultaneamente, incrementa o valor do registo PC para indicar o endereço da próxima instrução a ser lida. Neste estado, caso a instrução seja LOAD com acesso a memória (LM) ou STORE (ST), o próximo estado será o 3, caso contrário será o zero.

Estado3: Preparação do endereço para leitura ou escrita da memória de dados.

Estado4: Se a instrução é LOAD (LD) activa o sinal RD, caso contrário activa o sinal WR.

Para que o ASM tenha uma leitura mais fácil, os sinais de saída que não estão relacionados com a evolução de estados do processador, não foram representados.

12.2.3 ROM de descodificação

Na Tabela 12-1 está representada a programação da ROM de descodificação, na qual podemos observar o comportamento das várias saídas do controlo e a evolução de estados. Na implementação do controlo do PDS8_V2, iremos utilizar a ROM para conter o micro-code e gerar o estado seguinte. Por esta razão a dimensão da ROM será de 1024x18, pois tem como entrada os 5 bits mais significativos do código da instrução, as duas *flags* do PSW e os 3 bits que codificam o estado corrente.

	IIIII 76543ZF	EC_{0-2}	ES_{0-2}	SD	SA	SI	SO	EM	EPC	ER	EP	EIR	RD	WR
Reset	-------	0	2	-	3	-	-	1	1	0	0	0	0	0
Exec/Addr fetch														
LD R,[RA+RB]	00000--	1	2	1	4	0	-	1	0	1	0	0	0	0
LD R,[RB]	00001--	1	2	1	4	0	-	1	0	1	0	0	0	0
ST R,[RB]	00010--	1	2	-	4	0	-	1	0	0	0	0	0	0
JMP [RB]	00011--	1	2	-	3	1	-	1	1	0	0	0	0	0
LD R,direct4	001----	1	2	1	4	0	-	1	0	1	0	0	0	0
LDI R,#const4	010----	1	2	0	4	0	-	1	0	1	0	0	0	0
ST R,direct4	011----	1	2	-	4	0	-	1	0	0	0	0	0	0
JZ offset5	100--0-	1	2	-	3	0	-	1	0	0	0	0	0	0
JZ offset5	100--1-	1	2	-	4	0	1	1	1	0	0	0	0	0
JF offset5	101---0	1	2	-	3	0	-	1	0	0	0	0	0	0
JF offset5	101---1	1	2	-	4	0	1	1	1	0	0	0	0	0
JMP offset5	110----	1	2	-	3	0	1	1	1	0	0	0	0	0
ALU	111T---	1	2	2	4	0	-	1	0	T	1	0	0	0
Opcode fetch														
	AL||JP	2	1	-	-	0	0	0	1	0	0	1	1	0
	LD||ST	2	3	-	-	0	0	0	1	0	0	1	1	0
Addr fetch														
LD+ST	direct4	3	4	-	2	-	-	1	0	0	0	0	0	0
LD+ST	[RB]	3	4	-	0	-	-	1	0	0	0	0	0	0
LD+ST	[RA+RB]	3	4	-	1	-	-	1	0	0	0	0	0	0
Data Read/Write														
	LD	4	1	-	-	-	-	0	0	0	0	0	1	0
	ST	4	1	-	-	-	-	0	0	0	0	0	0	1

EC_{0-2} – Estado Corrente **ES_{0-2}** – Estado Seguinte **I_{3-7}** – bits do Instruction Register

Tabela 12-1 - Programação da ROM de descodificação