Capítulo 12

Estrutura Interna de um Processador

No capítulo anterior definiu-se a arquitectura do conjunto de instruções do processador P3. A arquitectura do conjunto de instruções define a interface visível para o programador, especificando as instruções que estão disponíveis, quais os registos internos, os modos de acesso à memória e outras características relevantes do processador.

Para uma dada arquitectura do conjunto de instruções, existem muitas realizações possíveis para a estrutura do processador. As numerosas realizações possíveis resultam do grande número de escolhas que estão disponíveis ao projectista de sistemas digitais e implicam diferentes compromissos entre o número de ciclos de relógio necessários para executar cada instrução, a frequência máxima de relógio possível para o sistema e a área ocupada em silício para a realização física do processador.

Este capítulo descreve em detalhe uma realização particular do processador P3 e descreve os diversos compromissos que são inerentes às escolhas tomadas, focando, quando tal é julgado oportuno, as alternativas mais óbvias e as razões pelas quais as mesmas foram preteridas em detrimento da implementação descrita.

Como qualquer circuito complexo, é útil analisar o processador P3 considerando separadamente duas componentes principais, tal como foi estudado no Capítulo 9: o circuito de dados e o circuito de controlo.

No circuito de dados encontra-se toda a lógica regular que é usada para armazenar e processar dados do utilizador, lógica esta que opera, de uma forma geral, sobre conjuntos de dados organizados em octetos e palavras. Fazem parte do circuito de dados o banco de registos, a unidade lógica e aritmética, os circuitos de acesso a memória e portos de entrada/saída e ainda os barramentos de interligação internos.

O circuito de controlo gera os sinais que controlam o circuito de dados, por forma a que este execute a sequência de operações que são necessárias para carregar e executar cada instrução *assembly*, tendo em conta o estado do circuito de dados.

12.1 Circuito de Dados

O circuito de dados do processador P3, esquematizado na Figura 12.1, tem cinco componentes principais: o banco de registos, a unidade lógica e aritmética, o *registo de instrução* (`RI`), o registo de estado (`RE`), e, por último, os circuitos de interligação e multiplexagem de dados.

O banco de registos, cuja estrutura interna foi descrita na Secção 7.5.4, contém 16 registos, `R0` a `R15` de 16 bits cada e é acedido através de dois portos de leitura (portos A e B) e um porto de escrita (porto D). Dois destes registos são registos de uso especial, o contador de programa `PC` e o apontador para a pilha `SP`.

A unidade lógica e aritmética, descrita em detalhe na Secção 9.4, é utilizada para realizar todas as operações lógicas e aritméticas sobre os operandos que são fornecidos pelo banco de registos.

O registo de instrução, `RI`, é um registo de uso especial que não está integrado no banco de registos. Este registo não precisa de ser acedido directamente pelo circuito de dados. No entanto, todos os seus bits são usados pela unidade de controlo.

O registo de estado, `RE`, agrupa os diversos bits de estado do processador, ligados ao circuito de dados através de dois barramentos de 5 bits que permitem ler e escrever este registo.

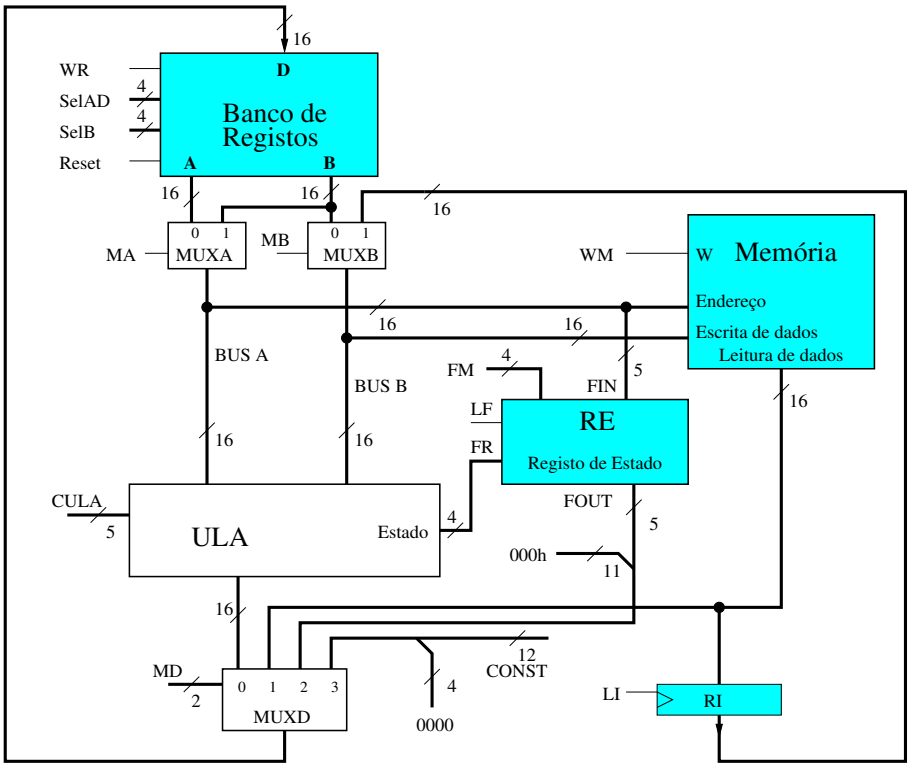

Figura 12.1: Circuito de dados do processador P3.

Os circuitos de acesso à memória são constituídos pelo barramento de en-

dereços e por dois barramentos de dados, um para escrita, outro para leitura. O barramento de endereços está ligado ao porto A do banco de registos, enquanto que o barramento de dados está ligado ao porto B. O barramento de leitura de dados está ligado ao porto de escrita do banco de registos. Estas ligações permitem executar uma leitura de memória para o banco de registos, através do controlo dos portos A e D do banco de registos, ou uma escrita na mesma, através do controlo dos portos A e B.

O funcionamento do circuito de dados é controlado pela palavra de controlo, sendo o funcionamento de cada um dos blocos descrito em detalhe nas secções seguintes.

12.1.1 Banco de Registos

O banco de registos foi estudado na Secção 7.5.4 e é utilizado quase sem modificações no circuito de dados do processador. A única alteração é que o registo `R0` toma, neste caso, sempre o valor `0`. O sinal de controlo de escrita, WR, é gerado pela unidade de controlo, assim como os valores nos barramentos $SelAD$ e $SelB$, de 4 bits cada. O valor de $SelAD$ especifica qual o registo cujo conteúdo é colocado no porto A e, simultaneamente, qual o registo em que deverá ser escrito o valor contido no porto D, se o sinal WR estiver activo. O valor de $SelB$ especifica qual o registo cujo conteúdo deverá ser colocado no porto B.

Para que seja possível flexibilizar o controlo da unidade lógica e aritmética, o registo `R0` contém sempre o valor 0. Isto permite seleccionar 0 como um dos operandos e efectuar diversas operações que, doutro modo, necessitariam de uma unidade lógica e aritmética mais complexa. Na prática, o registo `R0` não é um verdadeiro registo, uma vez que é implementado ligando as linhas do barramento directamente ao nível lógico `0`, através de *buffers* de três estados.

Com a excepção do registo `R0`, todos os outros registos podem ser usados para guardar valores. No entanto as funções de alguns dos registos estão pré-definidas, de acordo com a Tabela 12.1.

Note-se que todos os registos que estão destinados a funções específicas tem número superior a 7. Isto impede que um programa codificado ao nível do *assembly* tenha acesso aos mesmos e perturbe o funcionamento normal do processador, uma vez que, ao nível do *assembly*, o programador apenas pode aceder aos registos com números entre 0 e 7.

Entre os registos de uso especial, encontram-se o `R14` e o `R15`. O registo `R14` é o registo apontador da pilha, `SP`. É este registo que é usado para endereçar a memória quando se executa uma operação que manipule directa (`POP` ou `PUSH`) ou indirectamente (`CALL`, `INT`, `RET` e `RTI`) a pilha do processador.

O registo `R15` guarda o valor do contador de programa, `PC`, que, após a execução de cada instrução, aponta sempre para a próxima instrução que o processador irá executar.

Alterar o valor de qualquer um destes registos fora do seu contexto normal de utilização irá interromper o funcionamento normal do processador. Assim, o seu uso para quaisquer outras funções deverá ser sempre evitado.

Os registos `R11` a `R13` estão também destinados a funções específicas, mas o seu significado só se tornará claro quando se analisar a forma como são executadas as instruções *assembly*, na Secção 12.3.

Registo	Descrição
R0	Constante 0
R1	Registo de uso geral
R2	Registo de uso geral
R3	Registo de uso geral
R4	Registo de uso geral
R5	Registo de uso geral
R6	Registo de uso geral
R7	Registo de uso geral
R8	Registo de uso restrito
R9	Registo de uso restrito
R10	Registo de uso restrito
R11	Operando (SD)
R12	Endereço destino (EA)
R13	Resultado (RD)
R14	Apontador da pilha (SP)
R15	Contador programa (PC)

Tabela 12.1: Banco de registos.

12.1.2 Unidade Lógica e Aritmética

A unidade lógica e aritmética (ULA) usada por este processador é a que foi estudada na Secção 9.4, e que se reproduz na Figura 12.2.

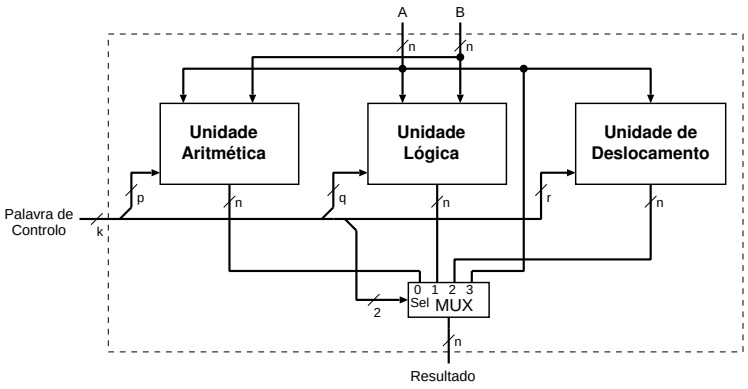

Figura 12.2: Estrutura da unidade lógica e aritmética.

A unidade lógica e aritmética é controlada por 5 bits de controlo, $CULA$. O valor destes 5 bits especifica, de acordo com a Tabela 9.1, qual a operação que a ULA executa sobre os dois operandos na sua entrada. Estes dois operandos são provenientes dos portos A e B do banco de registos.

Os quatro bits de estado gerados pela ULA estão ligados ao registo de estado, cujo funcionamento será detalhado na Secção 12.1.4.

12.1.3 Registo de Instrução

O registo de instrução, RI, encontra-se ligado directamente ao barramento de leitura de dados a partir da memória, é utilizado para guardar o código máquina da instrução *assembly* que está a ser executada.

Os 16 bits deste registo codificam, como foi descrito na Secção 11.4, qual a operação que deve ser executada e quais os operandos aos quais a mesma deve ser aplicada. De uma forma geral, o conteúdo deste registo não passa no circuito de dados, excepto na primeira fase de execução de uma instrução, em que o registo de instrução é carregado a partir de memória. O carregamento deste registo é controlado pelo valor do sinal LI, gerado pela unidade de controlo.

Para algumas operações, é necessário ler um ou mais campos do registo de instrução para o circuito de dados. Assim, é possível seleccionar o conteúdo do registo de instrução para a entrada no porto B da ULA, activando o sinal MB.

12.1.4 Registo de Estado

O registo de estado, RE, guarda os bits de estado do processador, permitindo ao programador testar o resultado da operação anterior e manter diversos bits de estado, que são actualizados de acordo o resultado das operações efectuadas pela unidade lógica e aritmética.

Quando o sinal de controlo LF está a 0, o valor dos bits de estado é actualizado de acordo com o resultado da última operação efectuada pela ULA. Para isso, o correspondente bit na máscara FM deverá estar a 1, de acordo com a Figura 12.3 definida pela unidade de controlo e não visível pelo programador ao nível do *assembly*. Os *bits de micro-estado*, z e c, são actualizados em todos

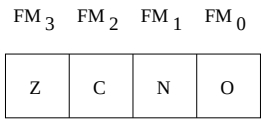

Figura 12.3: Bits da máscara FM que controla a actualização dos bits de estado.

os ciclos de relógio. A unidade de controlo define, em cada ciclo de relógio, quais os bits de estado que devem ser actualizados, de acordo com o que foi referido na Secção 11.6.1 sobre o modo como cada instrução *assembly* actualiza os mesmos.

Quando o valor de LF está a 1, o registo de estado é carregado com um valor proveniente do barramento $BUS\ A$, através do barramento FIN. Independentemente do valor dos sinais de controlo, o registo de estado pode ser carregado no banco de registos, através do porto D, usando para tal o barramento $FOUT$.

A Figura 12.4 descreve o esquema interno do registo de estado. O funcionamento do registo de estado é controlado por sinais activados pela unidade de controlo e por quatro bits de estado ZR, CR, NR, OR, gerados pela unidade lógica e aritmética.

Quando o sinal LF é activado pela unidade de controlo, o registo de estado é carregado a partir do barramento $BUS\ A$.

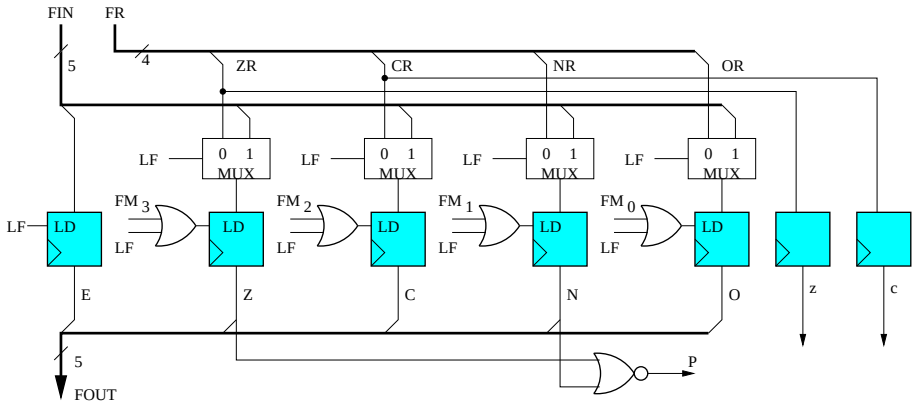

Figura 12.4: Esquema interno do registo de estado.

Além dos bits `Z`, `C`, `N` e `O`, cujo significado foi descrito na Secção 11.6.4, existe ainda o bit `E` que controla se o processador deve ou não responder a interrupções e cujo valor não vem da ULA, mas sim através da entrada FIN. Note-se que o sinal `P` não corresponde a um bit no registo de estado, uma vez que este bit é gerado pela lógica representada a partir dos valores dos bits `Z` e `N`.

12.1.5 Barramentos de Interligação

Os dois portos de leitura do banco de registos alimentam a unidade lógica e aritmética através dos multiplexadores `MUXA` e `MUXB`, que atacam os barramentos $BUS\ A$ e $BUS\ B$.

Quando é efectuado um acesso à memória, o $BUS\ A$ contém o endereço que deverá ser acedido. No caso de uma operação de escrita (sinal WM a 1), o valor no barramento $BUS\ B$ especifica qual o valor que deverá ser escrito. No caso de uma leitura (sinal WM a 0), o multiplexador $MUXD$ deverá ser controlado por forma a seleccionar o valor devolvido pela memória, colocando o valor de MD a 01. Neste caso, o valor lido da memória é escrito para o registo especificado pelo sinal de controlo $SelAD$, devendo o sinal de escrita no banco de registos, WR, estar activado.

12.1.6 Controlo do Circuito de Dados

O controlo do circuito de dados é feito através da palavra de controlo, descritos na Tabela 12.2. Cada um dos sinais descritos nesta tabela controla o funcionamento de um dos elementos do circuito de dados, tal como descrito nas secções anteriores.

Para se perceber melhor o funcionamento deste circuito de dados, consideram-se em seguida alguns exemplos que permitem ilustrar as operações de transferência entre registos possibilitadas pelo mesmo.

Suponha-se, por exemplo, que se pretende somar o conteúdo do registo `R3` com o conteúdo do registo `R7` e deixar o resultado no registo `R3`, alterando os valores dos bits de estado `C` e `Z`. Que valores devem tomar as variáveis de controlo do circuito de dados?

Sinal	# bits	Função
$Reset$	1	Inicializa o conteúdo dos registos a 0
$SelAD$	4	Controla os portos A e D do banco de registos
$SelB$	4	Controla o porto B do banco de registos
MA	1	Controlo do multiplexador A
MB	1	Controlo do multiplexador B
MD	2	Controlo do multiplexador D
WR	1	Escrita no banco de registos
WM	1	Escrita em memória
LF	1	Carrega os bits de estado
LI	1	Carrega o registo de instrução
FM	4	Controla a actualização dos bits de estado
$CULA$	5	Controla operação a executar na ULA
$CONST$	12	Valor de constante numérica

Tabela 12.2: Palavra de controlo do circuito de dados.

Em primeiro lugar, para que o banco de registos coloque nos portos A e B os registos R3 e R7, os sinais $SelAD$ e $SelB$ devem tomar os valores 0011 e 0111, respectivamente. Adicionalmente, e como se pretende que tenha lugar uma escrita no banco de registos, o valor do sinal WR deverá ser 1.

Para que os valores presentes nos portos A e B do banco de registos cheguem às entradas da unidade lógica e aritmética, o valor dos sinais MA e MB deverá ser 0.

A operação da ULA é controlada pelo valor do sinal $CULA$. Da Tabela 9.1, tira-se que $CULA$ deverá tomar o valor 00000 para que seja realizada uma operação de adição. Para conseguir que os bits de estado sejam actualizados com os valores desta operação, o sinal LF deverá tomar o valor 0 e o sinal FM deverá tomar o valor 1100.

Finalmente, e para que o porto de escrita do banco de registos receba o valor de saída da unidade lógica e aritmética, é necessário controlar MD por forma a que o multiplexador D seleccione na sua saída o valor da sua entrada 0, o que consegue colocando MD a 00.

Resta agora controlar o valor dos sinais ainda não definidos por forma a que não se realizem operações indesejadas. Uma vez que não se pretende escrever na memória nem no registo de instrução os sinais WM e LI deverão tomar o valor 0. O campo constante, não utilizado, pode tomar qualquer valor.

Concluí-se assim que a micro-operação R3←R3+R7 é executada se os sinais de controlo tomarem os valores da segunda coluna da Tabela 12.3.

Para um segundo exemplo, suponha-se agora que se pretende endereçar a memória usando o conteúdo do registo R5 e guardar o valor dessa posição de memória no registo R3, sem mexer no conteúdo de qualquer dos outros registos.

Nestas condições, será necessário forçar $SelB$ a 0101 e $SelAD$ a 0011, activando também o sinal WR.

O multiplexador MUXA deverá agora seleccionar a sua entrada 1, pelo que o valor do sinal MA deverá ser 1. Para efectuar uma leitura de memória e seleccionar o valor lido como aquele que entra no porto D do banco de registo,

Sinal	R3 ← R3+R7	R3 ← M[R5]
$Reset$	0	0
$SelAD$	0011	0011
$SelB$	0111	0101
MA	0	1
MB	0	x
MD	00	01
WR	1	1
WM	0	0
LF	0	0
LI	0	0
FM	1100	0000
$CULA$	00000	xxxxx
$CONST$	xxxxxxxxxxx	xxxxxxxxxxx

Tabela 12.3: Sinais que controlam a execução das micro-operações R3 ← R3+R7 (coluna 2) e R3 ← M[R5] (coluna 3).

o sinal WM deverá tomar o valor 0 e o sinal MD o valor 01.

Para que não haja escrita no registo de instrução nem no registo de estado, os sinais LF e LI deverão tomar o valor 0. Também o sinal FM deverá ter todos os seus bits a 0.

Neste exemplo, como em muitos outros, os valores de alguns dos sinais são irrelevantes, uma vez que controlam partes do circuito de dados que não são lidos nem guardados. É o caso dos sinais de controlo $CULA$, MB e $CONST$, que podem tomar qualquer valor sem afectar o funcionamento do circuito. A terceira coluna da Tabela 12.3 descreve os valores que os sinais devem tomar para que seja executada a micro-operação R3 ← M[R5].

12.2 Unidade de Controlo

Os sinais que controlam o circuito de dados são gerados por uma unidade de controlo micro-programada, descrita nesta secção. Como foi visto na Secção 8.3.3, a utilização de uma unidade de controlo micro-programada permite uma maior flexibilidade da unidade de controlo, e também uma organização mais estruturada que no caso em que a unidade de controlo é feita sintetizando uma máquina de estados.

A Figura 12.5 descreve a estrutura geral da unidade de controlo do processador P3. O coração da unidade de controlo é o micro-sequenciador, que controla a ordem pela quais são executadas as micro-instruções guardadas na memória de controlo. As micro-instruções definem o valor dos sinais utilizados para controlar o circuito de dados, o próprio micro-sequenciador, e diversos aspectos do funcionamento dos outros módulos que aparecem na Figura 12.5.

O funcionamento do micro-sequenciador é controlado pela unidade de teste de condições e pela unidade de mapeamento, além dos sinais de controlo gerados directamente pelas micro-instruções. A unidade de teste de condições permite testar os bits de estado do processador e, também, outros bits internos

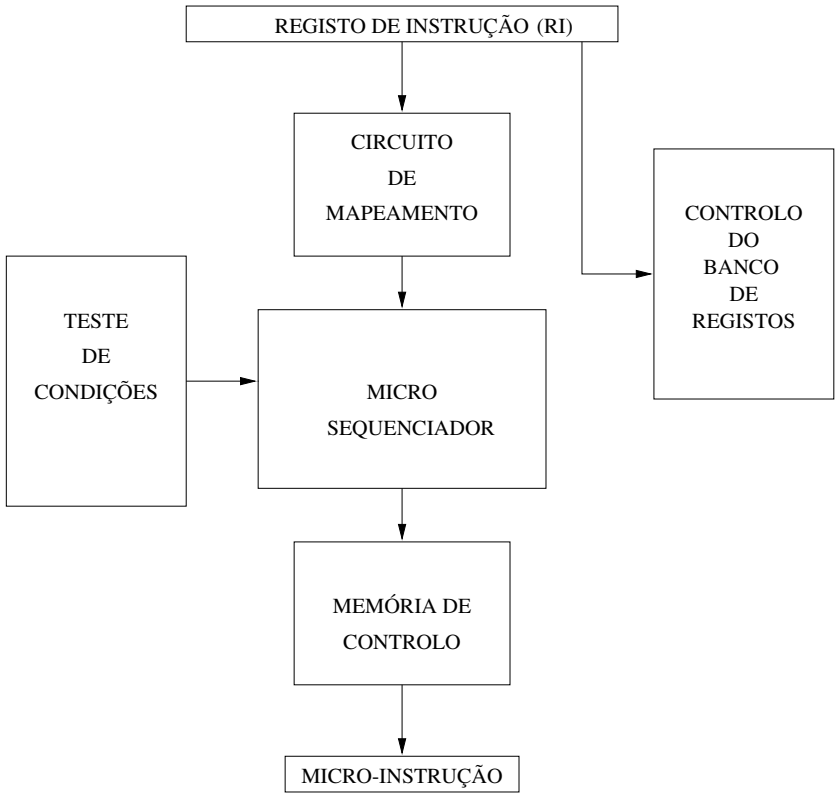

Figura 12.5: Esquema geral da unidade de controlo.

à unidade de controlo.

O controlo do banco de registos é feito por um circuito dedicado, controlado directamente por bits do registo de instrução e do registo de micro-instrução.

12.2.1 Formato das Micro-instruções

Tanto o circuito de dados como partes do próprio circuito de controlo são controlados por um conjunto de sinais que constituem a micro-instrução. Numa unidade micro-programada como esta, as micro-instruções que constituem o micro-programa são guardadas na memória de controlo, endereçada pelo registo CAR.

Uma opção possível é seleccionar uma micro-instrução que tenha um bit por cada um dos sinais de controlo que se pretendem gerar. No entanto, esta opção nem sempre se revela a mais adequada, uma vez que nem todas as combinações dos sinais de controlo são necessárias ou úteis.

A título de exemplo, no circuito da Figura 12.1, o valor do sinal $CULA$ não é importante quando se pretende carregar o valor de uma constante usando o sinal de controlo $CONST$.

Nesta realização, optou-se por utilizar dois formatos para a micro-instrução, que se distinguem entre si pelo valor do bit mais significativo da micro-instrução, F. A Figura 12.6 descreve os dois formatos possíveis para a micro-instrução.

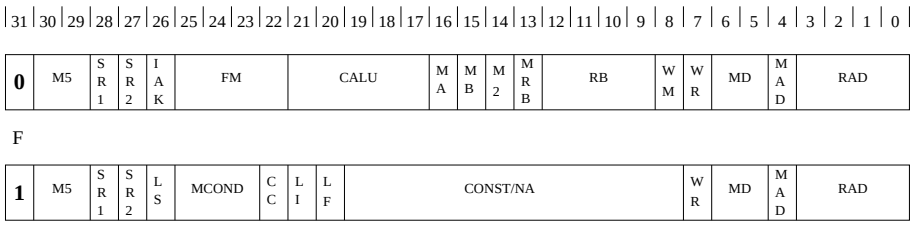

Figura 12.6: Formato das micro-instruções.

O formato correspondente a F=0 corresponde a micro-instruções que controlam fundamentalmente o circuito de dados, enquanto que o formato correspondente a F=1 é, prioritariamente, destinado a controlar a unidade de controlo, nomeadamente a unidade de teste de condições e o micro-sequenciador.

Muitas outras opções teriam sido possíveis, desde um formato único em que todos os sinais de controlo estivessem sempre disponíveis, até uma solução em que existissem mais do que dois formatos de micro-instrução. A primeira alternativa iria requerer uma micro-instrução com 50 bits, sem, no entanto, ser significativamente mais rápida, uma vez que raramente é necessário controlar todas as unidades do processador simultaneamente. Já a utilização de uma alternativa com mais formatos de micro-instrução poderia de facto reduzir o número de bits nas micro-instruções, mas implicaria uma redução significativa da velocidade de execução, por passarem a existir demasiados sinais que não podem ser controlados simultaneamente.

Pode observar-se que, na solução adoptada, alguns sinais estão presentes em ambos os formatos da micro-instrução, como, por exemplo, o sinal WR que controla a escrita no banco de registos. Estes sinais podem ser activados quer em micro-instruções do tipo 0 (F = 0) quer em micro-instruções do tipo 1 (F = 1).

Outros sinais estão presentes apenas num dos tipos de micro-instruções, como por exemplo o sinal que controla a escrita em memória, WM. Estes sinais só podem ser activados em micro-instruções desse tipo, devendo permanecer inactivos nas restantes. Isto significa que, no circuito de dados da Figura 12.1, os sinais que aparecem apenas num dos formatos de micro-instrução devem resultar da conjunção do sinal F, negado ou não, com o valor do bit da micro-instrução.

O circuito de dados, modificado para incluir explicitamente as portas lógicas que executam esta função, encontra-se representado na Figura 12.7. Note-se que a lógica adicionada força a que o controlo do multiplexador $MUXA$ e o sinal de escrita em memória fiquem activos apenas em micro-instruções do tipo 0, uma vez que o sinal que controla de facto o circuito de dados é a conjunção do sinal original com o complemento do bit F. Do mesmo modo, os 4 bits do sinal FM devem ficar activos apenas quando F=0, o que se consegue com 4 portas `AND`, na figura representadas apenas por uma delas aplicada aos 4 bits do barramento.

Os sinais de carregamento do registo de instrução e de carregamento do registo de estado, que só são gerados pela unidade de controlo em micro-instruções do tipo 1, são mascarados de forma análoga, sendo feita a conjunção com o bit F.

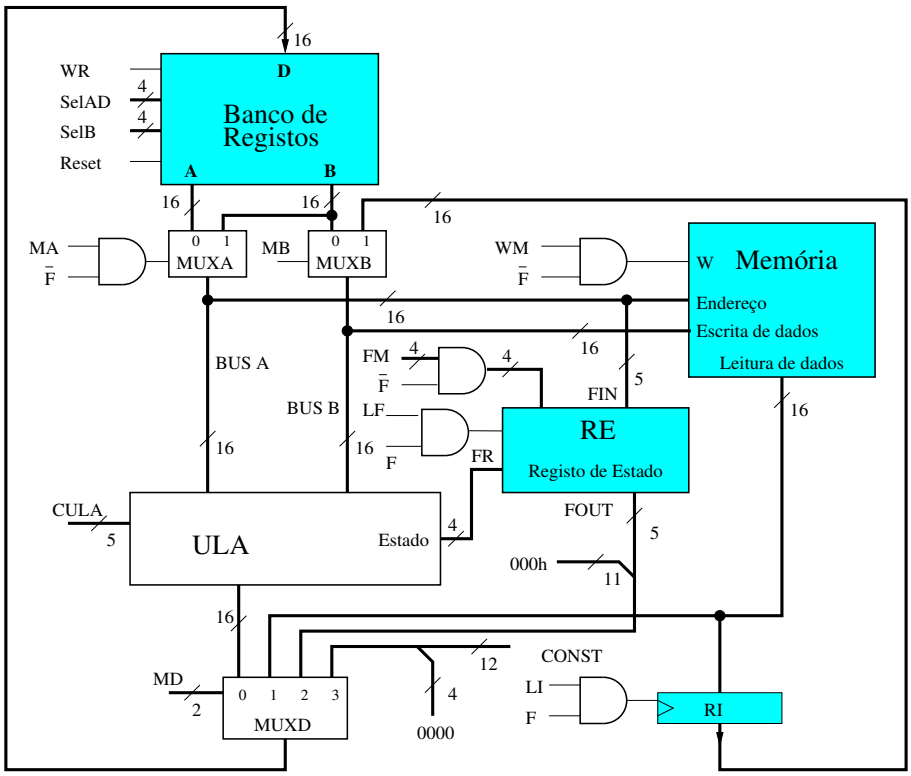

Figura 12.7: Circuito de dados e sinais de controlo.

A mesma regra é aplicável a sinais que controlam diversas partes da unidade de controlo propriamente dita.

De uma forma geral, quando um dado sinal é gerado pela unidade de controlo apenas no formato F=1, deverá ser feita uma conjunção com o sinal F, enquanto que quando esse sinal existe apenas no formato F=0, é feita a conjunção com o complemento do sinal F, isto é, $\overline{F}$.

Em alguns casos é possível poupar esta lógica se o sinal actuar partes do circuito que que não têm efeitos devido ao valor de outros sinais de controlo. Por exemplo, o sinal de controlo MB, que controla o multiplexador $MUXB$ está activo apenas no formato de micro-instrução que tem F=0. Assim, embora pudesse existir uma porta `AND` que fizesse a conjunção do sinal MB com a negação de F, esta porta revela-se inútil. De facto, uma análise cuidada do circuito de dados na Figura 12.1 mostra que, quando F=1, não existe controlo nem sobre a unidade lógica e aritmética nem sobre o sinal de escrita em memória, que está necessariamente inactivo. Assim, o valor que fica no barramento B nunca é usado, pelo que não é necessário controlar o multiplexador `MUXB`, poupando-se uma porta lógica.

12.2.2 Micro-Sequenciador

No centro da unidade de controlo encontra-se o micro-sequenciador, que gera a sequência de endereços de micro-instruções que deve ser executada. O micro-

sequenciador pode, em cada ciclo de relógio, executar uma das seguintes operações, de acordo com os sinais de controlo:

- Incrementar o endereço da micro-instrução a executar ou saltar para um endereço especificado na micro-instrução, de acordo com o valor do sinal $COND$, gerado pela unidade de teste de condições.
- Retornar de uma micro-rotina.
- Saltar para um dos endereços fornecidos pela unidade de mapeamento.

A estrutura do micro-sequenciador está descrita na Figura 12.8. O micro-

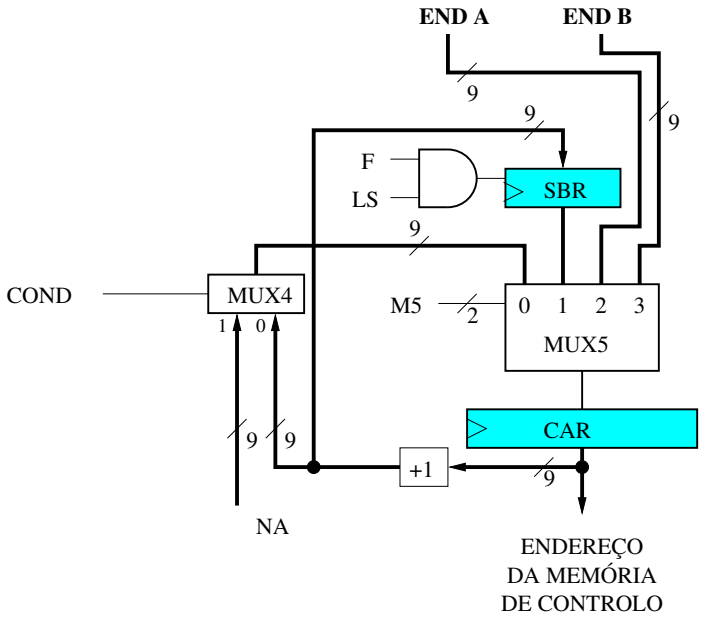

Figura 12.8: Esquema do micro-sequenciador.

sequenciador usa dois registos. O registo CAR (*Control Address Register*) contém o endereço da memória de micro-instruções onde está a micro-instrução que está a ser executada. Este registo desempenha para um micro-programa o mesmo papel que o contador de programa desempenha para um programa em *assembly* e pode também ser referido por *contador de micro-programa*. O registo SBR guarda o endereço de micro-programa para onde deverá ser transferido controlo após terminar a execução de uma micro-rotina. Uma vez que existe só um registo para guardar o endereço de retorno, apenas é possível utilizar um nível de profundidade de micro-rotinas, o que significa que uma micro-rotina não pode chamar outra.

É importante sublinhar que não existe qualquer relação entre uma micro-rotina e uma subrotina definida ao nível do *assembly*.

O funcionamento do micro-sequenciador é controlado pelos dois bits do sinal $M5$ que controla o carregamento do registo CAR, da seguinte forma:

- $M5 = 00$: CAR é incrementado se $COND$ for 0, e é carregado com o valor de NA (o endereço da memória de controlo para onde se deve saltar), se

$COND$ for 1, permitindo assim a execução de saltos condicionais dentro de micro-programas.

- $M5$ = 01 : CAR é carregado com o valor contido em SBR, sendo assim executado um retorno de micro-rotina.
- $M5$ = 10 : CAR é carregado com o valor especificado em $ENDA$ pela unidade de mapeamento.
- $M5$ = 11 : CAR é carregado com o valor especificado em $ENDB$ pela unidade de mapeamento.

A maioria das micro-instruções são executadas de forma sequencial, pelo que após a execução de uma micro-instrução o CAR deve ser incrementado para ficar a apontar para a próxima posição da memória de controlo. Este comportamento consegue-se colocando o valor de $M5$ a 00 na micro-instrução, e controlando o bloco de teste de condições por forma a que o valor de $COND$ seja 0. Caso se pretenda executar um salto condicional no micro-programa, o valor de $M5$ deverá ser 00 na mesma, e a unidade de teste de condições deverá seleccionar a condição de salto desejada.

A possibilidade de carregar o registo CAR com o conteúdo do registo SBR permite a existência de micro-rotinas. Quando se pretende chamar uma micro-rotina, o sinal LS deve ser activado e o registo CAR deve ser carregado com o valor de NA, que especifica o endereço da micro-rotina que se pretende chamar. Como o sinal LS só existe nas micro-instruções com F=1, o sinal de carregamento do registo SBR deve ser inibido quando F=0. A activação deste sinal faz com que o registo SBR seja carregado com o valor de CAR+1, que representa o endereço da primeira micro-instrução a ser executada quando a execução da micro-rotina termina. O retorno de micro-rotina é executado seleccionado $M5$ = 01 o que força o carregamento do registo CAR com o endereço de retorno.

O conteúdo do registo CAR também pode ser carregado com os valores $ENDA$ (colocando $M5$=10) ou $ENDB$ (colocando $M5$=11), gerados pela unidade de mapeamento, e cuja utilidade será estudada na Secção 12.2.4.

12.2.3 Teste de Condições

A unidade de teste de condições, descrita na Figura 12.9, tem como função seleccionar qual a condição que é testada pelo micro-sequenciador quando o mesmo executa uma micro-instrução de salto ou chamada a micro-rotina, condicional ou não.

Esta unidade tem um único bit de saída, o sinal $COND$, que indica ao micro-sequenciador se deve ou não executar um salto, tal como foi descrito na secção anterior.

Esta unidade é fundamentalmente constituída por dois multiplexadores, e algumas portas lógicas auxiliares. O multiplexador, MUXCOND, controlado pelo campo $MCOND$ da micro-instrução, permite seleccionar um dos seguintes bits:

- A constante 1, o que permite ao micro-sequenciador incrementar ou executar saltos incondicionais, dependendo do valor do sinal CC (*Complementar Condição*).

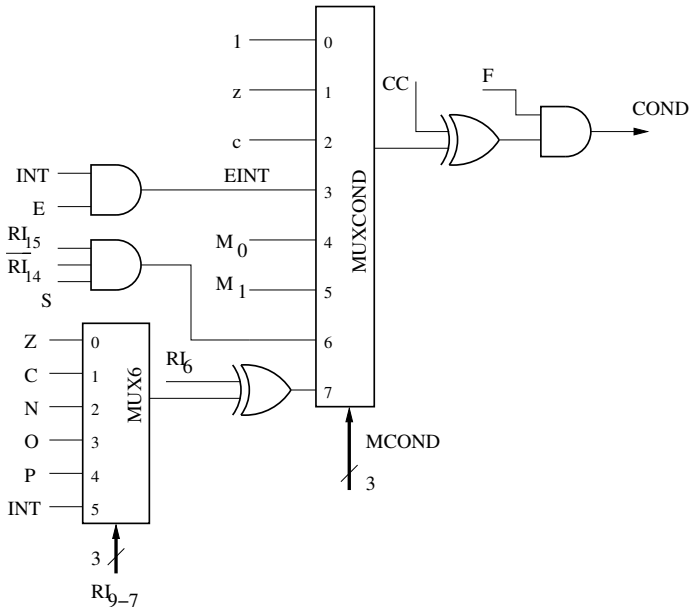

Figura 12.9: Unidade de teste de condições.

- Os bits z e c do registo de estado. Estes bits são também conhecidos por bits de micro-estado. O bit z está a 1 se a última operação na ULA deu resultado zero, enquanto que o bit c está a 1 se a última operação na ULA gerou transporte.
- A conjunção do bit `E` do registo de estado com o sinal INT que assinala a presença de uma interrupção, indicando se deve atender uma interrupção presente.
- Os bits do campo M do registo de instrução, que codificam o modo de endereçamento da instrução *assembly* que está a ser executada.
- O bit S do registo de instrução, mascarado pela expressão $RI_{15}\overline{RI_{14}}$ que indica qual o operando ao qual deve ser aplicado o modo de endereçamento da instrução, no caso de instruções de dois operandos.
- Um dos bits do registo de estado, mais os sinais P e INT, escolhido de acordo com o valor dos bits 7 a 9 do registo de instrução.

A entrada 7 do multiplexador `MUXCOND` é controlada pela saída do multiplexador `MUX6`. Uma vez que os bits 7 a 9 do registo de instrução especificam, conforme Figura 11.13) e de acordo com a Tabela 11.14, qual a condição que é testada numa dada instrução, é possível seleccionar directamente qual a condição que deverá ser testada pelo micro-sequenciador.

Uma análise da Tabela 11.14 revela que os bits 7, 8 e 9 do registo de instrução seleccionam qual a condição, enquanto que o bit 6 indica se a condição deve ser complementada. A ligação do multiplexador `MUX6` conforme indicado na figura e o uso de uma porta `XOR`, que funciona como uma negação condicional, permite que, na entrada 7 do multiplexador `MUXCOND`, esteja já seleccionada a

condição correcta. Isto leva a que o teste de uma destas condições seja efectuado numa só micro-instrução, não sendo necessário efectuar diversos testes aos valores dos bits de estado e do valor dos bits 6 a 9 do registo de instrução.

Quando se pretende que o micro-sequenciador execute as instruções de forma sequencial, deverá ser colocado o valor 0 na saída $COND$. Isto é conseguido colocando o valor 000 no campo $MCOND$ da micro-instrução e o valor 1 no campo CC da micro-instrução.

Caso se pretenda que o micro-sequenciador execute um salto incondicional, deverá ser colocado o valor 1 na saída $COND$, colocando 000 em $MCOND$ e 0 em CC.

Finalmente, caso se pretenda que o micro-sequenciador execute um salto na condição de um dado bit tomar um dado valor, o multiplexador `MUXCOND` deverá ser controlado de forma a seleccionar o bit pretendido, enquanto que o sinal CC define se a condição deverá ser complementada ou não.

Tanto o salto incondicional como o salto condicional só podem ser executados pelo micro-sequenciador quando a micro-instrução é do tipo F=1, uma vez que só neste formato estão disponíveis os campos $COND$, CC e o endereço de salto NA.

Caso a micro-instrução seja do tipo 0, o valor de $COND$ é colocado a 0 e o micro-sequenciador incrementa sempre o contador de micro-programa.

12.2.4 Unidade de Mapeamento

A unidade de mapeamento é utilizada para gerar, de forma rápida, os endereços das micro-rotinas chamadas durante a execução de instruções. Com efeito, em diversos passos da execução de uma instrução *assembly* torna-se necessário saltar para uma micro-rotina ou troço de micro-código, de acordo com o valor presente num dado campo do registo de instrução.

Por exemplo, o valor contido nos seis bits mais significativos do registo de instrução representa o código de instrução e define qual a operação que deverá ser executada. Este valor é utilizado na fase de descodificação de uma instrução *assembly*, para gerar o endereço da memória de controlo que corresponde às micro-instruções que implementam a instrução *assembly*.

Noutra fase da execução da instrução, é necessário saltar para um dado endereço de micro-código, de acordo com o modo de endereçamento utilizado, e especificado no campo M do registo de instrução.

A unidade de mapeamento é utilizada em diversas fases da execução de uma instrução. No processador P3, esta execução é feita nos seguintes passos:

1. Carregamento do registo de instrução.
2. Descodificação do código de operação e carregamento dos operandos.
3. Execução do micro-programa que implementa a instrução.
4. Escrita do resultado.
5. Teste de pedidos de interrupção.

Com esta sequência de operações, é necessário descodificar o código de operação para saber quais os operandos a carregar e qual o endereço da memória de micro-programa que contém as micro-instruções a executar. Dado

que o micro-sequenciador pode testar de cada vez apenas o valor de um bit, seleccionado pela unidade de teste de condições, e que o código de operação tem 6 bits, a escolha da micro-rotina utilizando este mecanismo iria requerer seis micro-instruções só para descobrir qual o endereço da micro-rotina a efectuar. Seriam ainda necessárias mais micro-instruções para decidir quais as micro-rotinas que deveriam ser chamadas para fazer a leitura dos operandos e a escrita do resultado, o que se revelaria muito ineficiente. A unidade

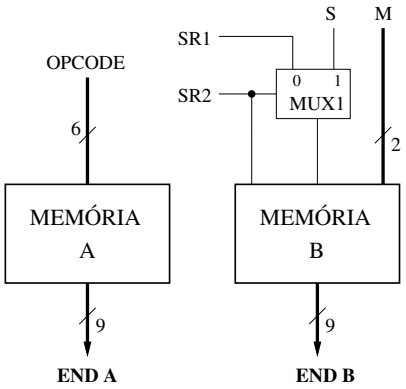

Figura 12.10: Unidade de mapeamento.

de mapeamento, esquematizada na Figura 12.10, permite que a transferência de controlo seja feita numa única micro-instrução. Isso consegue-se utilizando duas memórias de mapeamento, que são endereçadas de acordo com o código da operação, o modo de endereçamento e o valor do bit de direcção presentes no registo de instrução.

A unidade de mapeamento pode gerar dois endereços, qualquer um dos quais pode ser seleccionado pelo micro-sequenciador como o próximo endereço de micro-instrução a executar (ver Figura 12.8).

A memória A é endereçada directamente pelo código de operação (campo $OPCODE$ da instrução *assembly*) e implementa uma tabela que contém os endereços das micro-rotinas que executam as operações de transferência entre registos que realizam cada uma das instruções.

As linhas de endereço da memória de mapeamento B são controladas pelo valor dos bits $SR1$ e $SR2$ da micro-instrução e também pelo valor dos bits M e S do registo de instrução. De acordo com os valores dos bits $SR1$ e $SR2$, esta memória gera o endereço das micro-rotinas de carregamento de operandos ou de escrita do resultado.

Isto permite que, através do controlo dos bits $SR1$ e $SR2$, seja possível gerar quatro endereços diferentes. Estes endereços são usados para especificar diversas micro-rotinas de leitura e escrita, conforme especificado na Tabela 12.4.

Assim, caso se pretenda que a memória de mapeamento B gere o endereço da micro-rotina de leitura de um operando, a micro-instrução deverá colocar $SR2$ e $SR1$ a 00. Neste caso, o valor de S será ignorado, o que é correcto uma vez que este campo não tem significado quando a instrução é de um operando. Caso se pretenda o endereço da micro-rotina de leitura de dois operandos, basta forçar $SR2$ a 1, sendo nestas condições o valor de S usado para endereçar a memória, através do multiplexador `MUX1`.

SR2	SR1	S	Endereço seleccionado
0	0	-	Micro-rotina de leitura de um operando
0	1	-	Micro-rotina de escrita do resultado
1	-	0	Micro-rotina de leitura de dois operandos para S=0
1	-	1	Micro-rotina de leitura de dois operandos para S=1

Tabela 12.4: Funcionamento da memória de mapeamento B.

Para que o circuito de controlo funcione de acordo com o especificado, a memória B deverá ser carregada com os endereços das micro-rotinas correspondentes a cada uma das operações desejadas, de acordo com o esquema da Figura 12.11.

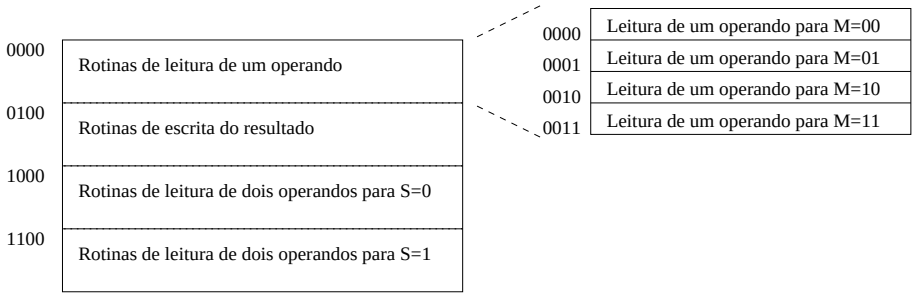

Figura 12.11: Preenchimento da memória de mapeamento B.

Para se analisar o funcionamento da unidade de mapeamento, suponha-se que se pretende controlar o micro-sequenciador por forma a transferir controlo para a primeira micro-instrução que implementa a instrução *assembly* guardada no registo de instrução. O endereço desta micro-instrução está guardado na memória de mapeamento A, que é endereçada pelos 6 bits mais significativos (campo $OPCODE$) do registo de instrução.

É necessário agora assegurar que este valor é carregado no registo CAR no próximo flanco do relógio. Por análise da Figura 12.8, verifica-se que o sinal de controlo $M5$ deverá tomar o valor 10, por forma a que o multiplexador $MUX5$ seleccione o valor gerado pela memória de mapeamento A como o próximo valor do registo CAR.

O controlo da memória de mapeamento B é ligeiramente mais complexo, mas também fácil de perceber. Suponha-se que se pretende transferir o controlo para a sequência de micro-instruções que carrega um operando. Por análise da Tabela 12.4, verifica-se que é necessário colocar os bits de controlo $SR2$ e $SR1$ a 00 para que a memória de mapeamento B gere o endereço dessa micro-rotina. Dado que valor do campo M do registo de instrução endereça directamente esta memória, basta agora colocar o sinal $M5$ a 11 para que a próxima micro-instrução a ser executada seja a desejada.

12.2.5 Controlo do Banco de Registos

A unidade de controlo controla o banco de registos através do circuito de controlo do banco de registos, descrita na Figura 12.12. Controlando o valor do

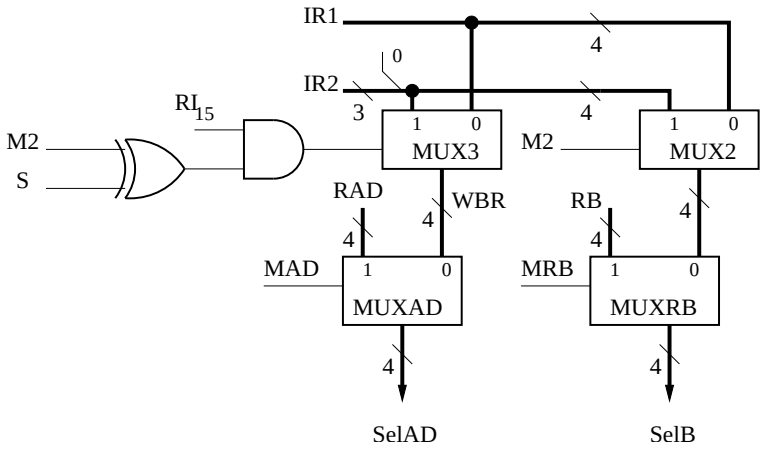

Figura 12.12: Circuito de controlo do banco de registos.

sinal MRB, disponível na micro-instrução, a unidade de controlo escolhe se o endereço do porto B do banco de registos é igual a RB (especificado na micro-instrução) ou aos valores especificados no registo de instrução. A situação é idêntica para o controlo dos portos A e D do banco de registos, sendo desta vez a escolha controlada pelo sinal MAD.

Quando o endereço do porto B é especificado directamente pelo registo de instrução, o valor do sinal de controlo $M2$ escolhe qual dos campos do registo de instrução deverá controlar este endereço.

No caso dos portos A e D, esta escolha é feita directamente por lógica que usa os valores do bit S e do bit mais significativo do código de operação. Esta lógica permite escolher o valor de WBR como sendo igual a $IR1$ ou $IR2$, de acordo com a Tabela 12.5. Apesar da aparente complexidade desta tabela, a

RI_{15}	S	M2	WBR
0	-	-	IR1
1	0	0	IR1
1	0	1	IR2
1	1	0	IR2
1	1	1	IR1

Tabela 12.5: Controlo do multiplexador `MUX3`.

sua utilização é muito simples, e é descrita na Tabela 12.6.

A lógica descrita na Figura 12.12 é necessária porque, nas instruções com dois operandos, o campo do registo de instruções que contém o registo ao qual é aplicado o modo de endereçamento pode ser $IR1$ ou $IR2$, de acordo com o valor de S. Estas instruções têm todas o bit mais significativo do registo de instrução a 1, conforme descrito na Secção 11.7.3. Todas as instruções que têm

M2	Valor seleccionado
0	Registo usado pelo primeiro ou único operando
1	Registo usado pelo segundo operando, quando exista

Tabela 12.6: Valor seleccionado pelo multiplexador `MUX3`.

o bit mais significativo do registo de instrução a 0 devem escrever o resultado no registo especificado pelo campo $IR1$, o que é conseguido com a porta `AND` da figura.

Para ilustrar o funcionamento desta unidade do micro-controlador, suponha-se que se pretende operar com registos definidos pelos valores dos campos da micro-instrução (`RAD` e `RB`), sem ter em atenção quais os registos definidos pela instrução *assembly* propriamente dita. Neste caso, há que endereçar o banco de registos com o valor definido pelos campos RAD e RB da micro-instrução, pelo que os sinais MAD e MRB deverão ser colocados a 1.

Para uma utilização mais complexa, suponha-se que se está na fase final de execução da instrução *assembly* `ADD R1, M[R7+00A0h]`. A codificação desta instrução já foi estudada na Secção 11.7.5, onde se viu que esta instrução é representada pela sequência de duas palavras `1000011001110111b` (`8677h`) e `0000000010100000b` (`00A0h`). Uma vez que se está na fase final de execução da instrução, o resultado da adição, já calculado, está guardado no registo `R13`. Pretende-se carregar este resultado no registo de destino especificado pela instrução *assembly*. Como se viu na Secção 11.7.5, o registo de destino encontra-se especificado no campo $IR2$ do registo de instrução. Isto acontece porque, na codificação desta instrução, o modo de endereçamento indexado se aplica ao segundo operando da instrução, o que é indicado pelo valor S=1.

Para conseguir executar esta operação, há, em primeiro lugar, que garantir que o valor presente no porto B do banco de registos se propague até ao porto de escrita do mesmo. Por análise da Figura 12.1 e da Tabela 9.1, verifica-se que é necessário colocar os valores no campos da micro-instrução de acordo com a Tabela 12.7.

Sinal	Valor
MB	0
MD	00
WR	1
WM	0
LF	0
LI	0
FM	0000
CULA	11XXX

Tabela 12.7: Sinais que controlam a execução da micro-operação `R1` ← `R13`.

É agora necessário controlar o circuito da Figura 12.12 por forma a conseguir que o registo usado como primeiro operando da instrução *assembly* seja escrito com o resultado. O valor de MRB deverá ser 1, para que o campo RB da micro-instrução possa especificar o registo `R13`. Já o valor de MAD deverá

ser 0, para que o valor seleccionado pelo multiplexador $MUXAD$ seja o que provêm do multiplexador $MUX3$.

Para controlar adequadamente o multiplexador $MUX3$, basta colocar $M2$ a 0, como descrito na Tabela 12.6. Pode verificar-se que o valor do bit RI_{15} do registo de instrução toma o valor necessário para que o controlo do multiplexador $MUX3$ fique com o valor adequado, que, neste caso, é 1.

12.2.6 Circuito de Controlo

O circuito completo de controlo do processador P3, descrito na Figura 12.13 resulta da ligação dos diversos blocos descritos nas secções anteriores. Note-

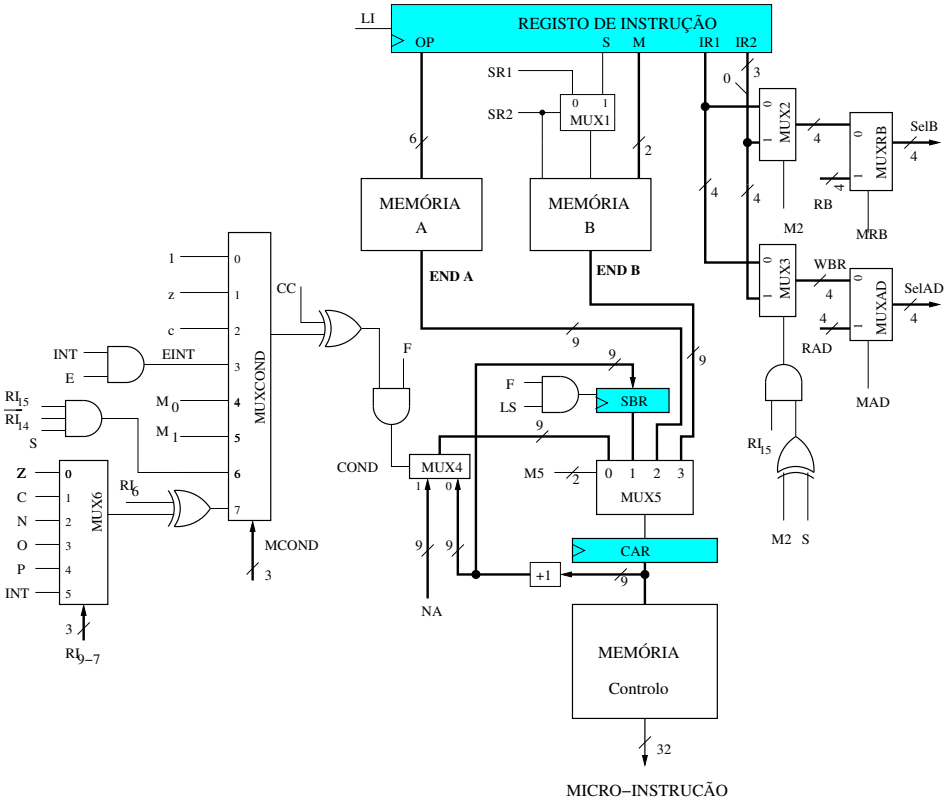

Figura 12.13: Circuito de controlo do processador P3.

se que o controlo do multiplexador `MUX5` permite seleccionar, como próximo endereço da micro-instrução a ser executada, quer o que vem do multiplexador `MUX4`, controlado pela unidade de teste de condições, quer o conteúdo do registo `SBR`, quer qualquer um dos endereços especificados pela unidade de mapeamento.

O sinal LI permite carregar o registo de instrução, que se encontra duplicado nesta figura e na Figura 12.1, que ilustra o circuito de dados.

O registo `CAR` é carregado em todos os ciclos de relógio, pelo que não são necessários sinais para controlar o seu carregamento.

12.3 Micro-Programação

Uma vez percebido o funcionamento da unidade de controlo, a programação da memória de controlo é relativamente simples. Com efeito, por análise do circuito de controlo, é possível identificar o valor que cada bit da micro-instrução deve tomar para se obtenha o funcionamento desejado do circuito de dados, assim como um comportamento correcto do próprio circuito de controlo.

O primeiro passo para a definição do conteúdo da memória de controlo é a definição da estrutura de alto nível dos micro-programas. Esta estrutura é, em grande parte, imposta pelas características do circuito de controlo e pelos tipos de operações que podem ser efectuados por este.

A sequência de operações efectuada quando uma instrução *assembly* é executada foi descrita na Secção 12.2.4. A esta sequência de operações corresponde o fluxograma da Figura 12.14. A execução de uma instrução do processador

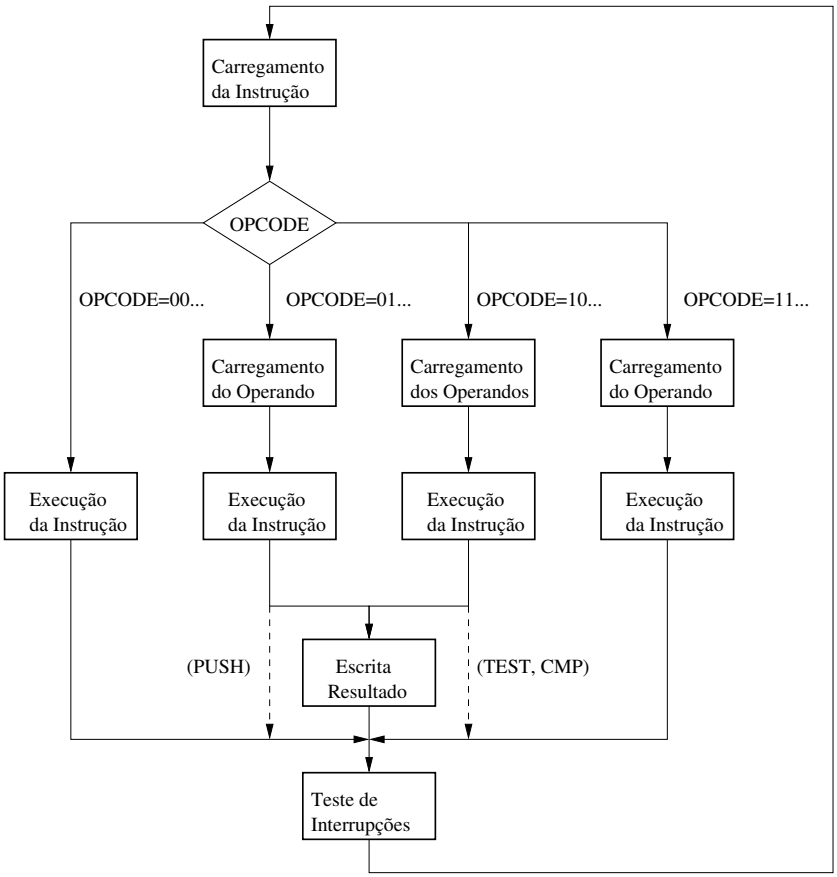

Figura 12.14: Fluxograma da execução de uma instrução *assembly*.

começa com o carregamento da instrução, da memória para o registo de instrução. Com base no código da operação, é efectuado um salto para o endereço da primeira micro-instrução que executa essa operação, usando para tal a me-

mória A do circuito de mapeamento.

Para as instruções que têm um ou dois operandos, a execução prossegue com o carregamento dos mesmos. Este carregamento é conseguido chamando uma micro-rotina, cujo endereço é fornecido pela memória B do circuito de mapeamento. A fase seguinte consiste na execução da instrução propriamente dita, usando uma sequência de micro-instruções que são específicas para cada instrução.

Após a execução da instrução, o controlo é transferido para a secção de micro-código que escreve o resultado, nos casos em que a instrução requer esta operação[1]. Para isto, é novamente utilizada a memória B do circuito de mapeamento, uma vez que a micro-rotina de escrita do resultado varia com o modo de endereçamento utilizado na instrução.

Finalmente, após a escrita do resultado (ou após a execução, no caso de instruções que não necessitam desse passo), o controlo é transferido para uma sequência de micro-instruções que verificam se existe uma interrupção pendente. Em caso afirmativo, o controlo é transferido para a micro-rotina de tratamento de interrupção. A excepção é a instrução `INT`, que não necessita de verificar se existem interrupções pendentes uma vez que faz a desactivação do bit do registo de estado que permite o atendimento de interrupções.

12.3.1 Carregamento do Registo de Instrução

Como foi visto atrás, a primeira fase da execução de uma instrução é o carregamento do registo de instrução, `RI` (em Inglês, esta fase chama-se *instruction fetch*, ou, abreviadamente, *IF*). Nesta fase, é necessário carregar o registo de instrução com o conteúdo da posição de memória apontada pelo contador de programa, `PC`. Em seguida, deve ser incrementado o valor do contador de programa, para que o mesmo fique a apontar para a próxima instrução a ser executada. A última operação consiste em transferir controlo para o troço de micro-programa que executa as operações necessárias à execução da instrução *assembly*. Ao contrário do *assembly* que apenas permite que uma e uma só operação seja feita em cada ciclo de relógio, um passo de micro-programa pode executar mais do que uma operação de transferência entre registos, desde que os circuitos de dados e controlo assim o permitam. A Tabela 12.1 descreve a sequência de operações de transferência entre registos que implementam esta fase da execução de uma micro-instrução. Como se pode observar, esta sequên-

```
IF0:  RI←M[PC]                    ; Carrega RI
IF1:  PC←PC+1, CAR←ROMA[OPCODE]   ; Incrementa PC
```

Programa 12.1: Micro-programa da fase de carregamento do registo de instrução.

cia de operações é descrita utilizando uma sequência de micro-instruções descritas na linguagem de transferência entre registos que foi descrita no Capítulo 9.

[1] Algumas instruções, como por exemplo, as instrução de `JMP`, `TEST` e `CMP` não geram qualquer resultado que necessite de ser escrito no operando da instrução.

Cada micro-instrução corresponde a uma ou mais operações de transferência entre registos, que são executadas sempre os que sinais de controlo correspondentes a essa micro-instrução estão activos. Para evitar a listagem exaustiva de todos os sinais de controlo que se encontram activos em cada micro-instrução, cada micro-instrução é precedida de um rótulo simbólico. Este rótulo corresponde aos valores dos sinais de controlo que se encontram activos durante a execução dessa micro-instrução.

Embora cada micro-instrução seja descrita como um conjunto de transferências de valores entre registos, a programação da memória de controlo é feita com uma sequência de zeros e uns que especificam o valor de cada micro-instrução, de acordo com o formato descrito na Figura 12.6. Os valores de cada bit da micro-instrução deverão ser tal que causem as transferências entre registos indicadas.

A transformação do micro-programa, descrito ao nível de transferência entre registos, na combinação de bits que controla adequadamente os circuitos do processador pode ser feita manualmente ou com o auxílio de um programa, que, neste caso, se chama micro-*assembler*. Na prática, e dado que a programação a este nível requer um conhecimento profundo dos sinais e circuitos envolvidos, a utilidade de um micro-*assembler* é consideravelmente mais reduzida que a de um *assembler*, pelo que esta transformação é feita, na maior parte dos casos, manualmente.

No caso presente, ilustra-se o funcionamento das micro-rotinas usando a linguagem de transferência entre registos, sendo a Secção 12.3.6 dedicada ao estudo do processo de tradução entre a micro-instrução e o micro-código em formato binário.

12.3.2 Carregamento dos Operandos

A fase seguinte da execução de uma instrução consiste no carregamento dos operandos, de acordo com o número dos mesmos e o seu modo de endereçamento. A micro-rotina a chamar depende da instrução *assembly* que está a ser executada, embora o procedimento seja similar em todas elas. Como foi descrito na Figura 12.10 da Secção 12.2.4, a memória de mapeamento B deve conter o endereço das micro-rotinas de leitura de operandos, quer para os casos em que existe apenas um operando, quer para os casos em que existem dois operandos.

Assim, uma instrução *assembly* que necessite apenas de um operando deverá chamar a micro-rotina correspondente usando a memória B da unidade de mapeamento, com os bits de controlo $SR1$ e $SR2$ a 0. Uma instrução que necessite de dois operandos deverá endereçar a memória B da unidade de mapeamento com o sinal $SR2$ a 1. Neste caso, o segundo bit de endereço desta memória é o bit S do registo de instrução, de acordo com o circuito da Figura 12.10.

Por forma a comunicarem facilmente entre si, os diversos passos de execução de uma instrução *assembly* usam os registos temporários de uma forma regular, de acordo com a seguinte convenção e a Tabela 12.1.

- O registo EA (R12) é usado para guardar o endereço efectivo de um operando sempre que este operando provenha de memória (do inglês, *effective address*).

- O valor do primeiro operando deve ser copiado para o registo RD (R13). Após os cálculos, o resultado da operação deve ser guardado nesse mesmo registo.

- O valor do segundo operando deve ser copiado para o registo SD (R11), sempre que a instrução use dois operandos.

As instruções de carregamento de operandos deverão funcionar de forma a deixar o endereço do operando no registo EA (R12). O valor do primeiro ou único operando deverá ser guardado no registo de destino, RD (R13). Em alguns modos de endereçamento (por exemplo, no modo imediato), não é necessário o endereço do operando e, nestes casos, o registo EA não é carregado. Quando as instruções têm dois operandos, o segundo operando deve ser deixado no registo SD (R11).

Pode-se agora examinar as micro-rotinas de carregamento de operandos, começando pelas que carregam apenas um operando. De acordo com o modo de endereçamento, o operando pode estar em várias localizações:

- Endereçamento por registo: o operando encontra-se num registo. Deve ser copiado desse registo para o registo RD. Não existe necessidade de actualizar o registo EA.

- Endereçamento indirecto por registo: o operando encontra-se na posição de memória cujo endereço está contido no registo. O valor do registo deve ser copiado para o registo EA e o valor da memória apontado por este registo deve ser copiado para o registo RD.

- Endereçamento imediato: o operando encontra-se na própria instrução, ou, mais exactamente, na posição de memória apontada pelo contador de programa, que já foi incrementado na micro-instrução com rótulo IF1. Esta posição de memória deve ser copiada para o registo RD, não havendo necessidade de actualizar o registo EA.

- Endereçamento indexado: o operando encontra-se numa posição de memória cujo endereço é obtido somando o conteúdo de um registo com o valor da posição de memória apontada pelo contador de programa. Este endereço deve ser carregado no registo EA e o seu conteúdo deverá ser copiado para o registo RD.

Em todos os casos, o registo ao qual se aplica o modo de endereçamento é o registo especificado no campo $IR1$ do registo de instrução.

Após a execução das micro-instruções que carregam o operando, o controlo deve ser retornado para o código que chamou a micro-rotina de carregamento de operando.

Torna-se agora simples especificar os micro-programas para cada uma das micro-rotinas de carregamento de um operando, descritas no Programa 12.2. A mais complexa desta micro-rotinas é a que trata do carregamento de operandos quando o modo de endereçamento é indexado, caso em que há que somar a palavra W ao valor do registo especificado em $IR1$ e endereçar a memória com o valor resultante.

As micro-rotinas de carregamento de dois operandos funcionam de forma semelhante às de carregamento de um operando. Porém, neste caso, o valor

```
F1R0:   RD←R[IR1], CAR←SBR  ; Copia operando

F1RI0:  EA←R[IR1]           ; Carrega o endereço
F1RI1:  RD←M[EA], CAR←SBR   ; Copia operando

F1IM0:  RD←M[PC]            ; Carrega o operando
F1IM1:  PC←PC+1, CAR←SBR    ; Incrementa o PC

F1IN0:  EA←M[PC]            ; Carrega a constante W
F1IN1:  PC←PC+1             ; Incrementa PC
F1IN2:  EA←EA+R[IR1]        ; Guarda o endereço
F1IN3:  RD←M[EA], CAR←SBR   ; Carrega o operando
```

Programa 12.2: Micro-rotinas de carregamento de um operando.

do bit S do registo de instrução indica se o modo de endereçamento é aplicado ao primeiro operando ou ao segundo operando. Quando o modo de endereçamento é aplicado ao primeiro operando, também se aplica ao destino da operação, uma vez que o primeiro operando especifica simultaneamente um dos operandos e o destino a dar ao resultado. Em ambos os casos, o modo de endereçamento aplica-se ao registo especificado no campo $IR1$ do registo de instrução. O outro operando, especificado pelo campo $IR2$ do registo de instrução, é sempre um registo.

```
F2R0:   RD←R[IR1]           ; Copia primeiro operando
F2R1:   SD←R[IR2], CAR←SBR  ; Copia segundo operando

F2RI0:  EA←R[IR1]           ; Guarda endereço
F2RI1:  RD←M[EA]            ; Copia primeiro operando
F2RI2:  SD←R[IR2], CAR←SBR  ; Copia segundo operando

F2IM0:  RD←M[PC]            ; Copia primeiro operando
F2IM1:  PC←PC+1             ; Incrementa o PC
F2IM2:  SD←R[IR2], CAR←SBR  ; Copia segundo operando

F2IN0:  EA←M[PC]            ; Carrega a constante W
F2IN1:  PC←PC+1             ; Incrementa o PC
F2IN2:  EA←EA+R[IR1]        ; Guarda o endereço
F2IN3:  RD←M[EA]            ; Copia primeiro operando
F2IN4:  SD←R[IR2], CAR←SBR  ; Copia segundo operando
```

Programa 12.3: Micro-rotinas de carregamento de dois operandos para $S = 0$.

O primeiro conjunto de micro-rotinas, descrito no Programa 12.3, é utilizado quando o valor de S é 0. Neste caso, o modo de endereçamento aplica-se ao primeiro operando, como, por exemplo, na instrução `ADD M[R1+30],R3`. O procedimento a adoptar é semelhante ao que foi utilizado nas instruções de um operando, existindo, tal como antes, quatro casos distintos: endereçamento por registo, indirecto, imediato e indexado. A diferença consiste fundamentalmente em copiar para o registo `SD` o valor do segundo operando, por forma

```
F2RS0:   SD←R[IR1]            ; Copia segundo operando
F2RS1:   RD←R[IR2], CAR←SBR   ; Copia primeiro operando

F2RIS0:  EA←R[IR1]            ; End.  do seg.  operando
F2RIS1:  SD←M[EA]             ; Copia segundo operando
F2RIS2:  RD←R[IR2], CAR←SBR   ; Copia primeiro operando

F2IMS0:  SD←M[PC]             ; Copia segundo operando
F2IMS1:  PC←PC+1              ; Incrementa PC
F2IMS2:  RD←R[IR2], CAR←SBR   ; Copia primeiro operando

F2INS0:  EA←M[PC]             ; Carrega a constante W
F2INS1:  PC←PC+1              ; Incrementa PC
F2INS2:  EA←EA+R[IR1]         ; Guarda endereço
F2INS3:  SD←M[EA]             ; Copia segundo operando
F2INS4:  RD←R[IR2], CAR←SBR   ; Copia primeiro operando
```

Programa 12.4: Micro-rotinas de carregamento de dois operandos para $S = 1$.

a que as instruções passem a dispor dos dois operandos nos registos RD e SD. Nos casos em que tal se aplique, o registo EA deve, tal como anteriormente, ser carregado com o valor do endereço onde se encontrava o primeiro operando. Este valor será mais tarde utilizado pelo micro-código de escrita do resultado, descrito no Programa 12.8.

Quando o valor do bit S do registo de instrução é 1, o tratamento é algo diferente. Neste caso, o modo de endereçamento aplica-se ao segundo operando, como, por exemplo, na instrução `ADD R3,M[R4+30]`. Neste caso, o papel dos registos RD e SD deve ser trocado, e não há necessidade de guardar o endereço do operando no registo EA. Note-se que o modo de endereçamento continua a aplicar-se ao registo especificado em $IR1$, só que, neste caso, o campo $IR1$ codifica o segundo operando, não o primeiro. Estas micro-rotinas estão descritas no Programa 12.4.

12.3.3 Execução das Instruções

Após a execução das micro-rotinas de carregamento de operandos, o processador pode operar sobre os dados, de acordo com a operação especificada no código de instrução.

Uma vez que os operandos estão já disponibilizados nos registos RD e SD, a operação propriamente dita é, de uma forma geral, relativamente simples. No caso vertente, ilustra-se a execução de instruções usando para tal alguns exemplos que são representativos do conjunto de instruções do processador.

Como exemplo de uma instrução aritmética, considere-se a instrução ADD. Após chamar a micro-rotina de carregamento de operandos, usando a unidade de mapeamento, esta instrução deverá somar o conteúdo dos registos RD e SD e deixar o resultado no registo RD. Deverá também actualizar os bits do registo de estado, actuando para tal os bits do campo FM.

Após ter o resultado guardado no registo RD, o controlo deve ser transferido para a micro-rotina de escrita do resultado. Uma vez que o endereço destas

micro-rotinas está guardado nas posições 4 a 7 da memória B (ver Tabela 12.4) da unidade de mapeamento, esta transferência faz-se carregando o registo CAR com o conteúdo desta memória, endereçada com os bits mais significativos ($SR2$ e $SR1$) a 01.

```
ADD0:  CAR←ROMB[1|S|M], SBR←CAR+1         ; Copia Ops
ADD1:  RD←RD+SD, FM←Fh, CAR←ROMB[0|1|M]   ; Adição
```

Programa 12.5: Micro-programa da fase de execução da instrução ADD.

Dada a simplicidade da operação de soma, a execução da instrução propriamente dita resume-se à operação RD ← RD+SD e à actualização dos bits de estado, sendo as outras duas instruções as chamadas às micro-rotinas de carregamento de operandos e de escrita do resultado.

Um exemplo ligeiramente mais complexo é o da instrução de PUSH, descrito no Programa 12.6. Esta instrução deverá guardar o seu operando na posição de memória apontada pelo registo SP e, em seguida, decrementar o mesmo. Esta instrução não tem de escrever o resultado no seu operando, uma vez que o valor do mesmo não deve ser alterado. Desta forma, o controlo é transferido directamente para a micro-rotina de tratamento de interrupções.

```
PUSH0:  CAR←ROMB[0|0|M], SBR←CAR+1  ; Copia operando
PUSH1:  M[SP]←RD, SP←SP-1           ; Escrita
PUSH2:  CAR←IH0                     ; Salto para IH
```

Programa 12.6: Micro-programa da fase de execução da instrução PUSH.

As instruções de controlo são programadas utilizando a mesma estrutura. Neste caso, porém, estas instruções actuam directamente sobre o valor do registo PC. Por exemplo, o micro-programa que executa a instrução CALL é o do Programa 12.7.

```
CALL0:  CAR←ROMB[0|0|M], SBR←CAR+1  ; Carregar endereço
CALL1:  M[SP]←PC, SP←SP-1           ; Push do PC
CALL2:  PC←RD                       ; Carregamento do PC
CALL3:  CAR←IH0                     ; Salto para IH
```

Programa 12.7: Execução da instrução CALL.

12.3.4 Escrita do Resultado

Após a execução da instrução, o resultado deve ser escrito, em registo ou em memória, de acordo com o modo de endereçamento usado. As micro-rotinas de escrita do resultado (Programa 12.8) recebem o resultado no registo RD e escrevem-no na localização especificada pelos bits M e S do registo de instrução.

Caso o bit S seja 1, a escrita deve sempre ter lugar para um registo. O endereço deste registo é especificado directamente pela unidade de controlo do banco de registos, descrita na Figura 12.12. Caso o bit S seja 0, o valor deverá

ser escrito na posição de memória apontada pelo registo EA, caso o modo de endereçamento seja indirecto ($M = 01$) ou indexado ($M = 11$). Uma vez que só

```
WBR0:  R[WBR]←RD     ; Escrita em registo
WBR1:  CAR←IH0       ; Tratamento de interrupções

WBM0:  S: CAR←WBR0   ; Escrita em registo se S = 1
WBM1:  M[EA]←RD      ; Escrita do resultado
WBM2:  CAR←IH0       ; Tratamento de interrupções
```

Programa 12.8: Micro-rotina de escrita do resultado.

existem duas micro-rotinas de escrita do resultado, a tabela correspondente na memória de mapeamento B deve ser construída de tal forma que as entradas correspondentes aos modos de mapeamento indexado e indirecto por registo apontem para o micro-código com rótulo WBM0. A entrada nesta tabela correspondente ao modo de endereçamento imediato nunca é usado, uma vez que este modo de endereçamento não pode ser utilizado para especificar o destino de uma operação.

No caso em que o primeiro operando é especificado utilizando o modo de endereçamento por registo, a escrita do resultado é mais simples, bastando copiar o conteúdo do registo RD para o registo especificado na instrução *assembly*. O endereço deste registo é seleccionado directamente pelo circuito, de acordo com a Figura 12.12.

12.3.5 Teste de Interrupções

A fase final da execução de uma instrução é o teste à existência de interrupções pendentes. Nesta fase, verifica-se se o sinal $EINT$ está activo, o que significa que existe uma interrupção pendente e que o bit que assinala a disponibilidade para atender interrupções está activo. Em caso negativo, o controlo deverá ser transferido para a primeira micro-instrução da micro-rotina de carregamento de instruções, $IF0$, o que desencadeará a execução da próxima instrução.

Note-se que é possível realizar a primeira linha do Programa 12.9 numa só micro-instrução. A operação de transferência de registos, é sempre executada, enquanto que a operação de carregamento do CAR só é executada quando o teste ao complemento do sinal $EINT$ dá resultado verdadeiro.

Se se der início ao tratamento de uma interrupção, o registo de bits de estado e contador de programa deverão ser guardados na pilha. O bit do registo de estado que indica a disponibilidade do processador para receber interrupções deverá ser desactivado, o que se consegue carregando o registo de estado com o valor 0. Finalmente, deverá ser activado o bit IAK, indicando externamente que se vai dar início ao tratamento da última interrupção gerada.

Em resposta à activação deste sinal, o controlador de interrupções deverá colocar o vector de interrupção no barramento de dados, identificando assim o periférico responsável pela interrupção. As rotinas de tratamento às interrupções de cada periférico do sistema, para onde o processador deve passar o controlo da execução após uma interrupção de um periférico, têm os seus endereços guardados numa tabela de interrupções, com início no endereço FE00h.

O vector de interrupção serve como índice para esta tabela. Assim, este valor deverá ser somado a FE00h para se obter o endereço de memória onde se encontra o endereço com que deverá ser carregado no contador de programa (notar que $VECTINT$+FE00h=$VECTINT$-0200h e este estratagema é utilizado por, ao contrário de FE00h, o valor 0200h ser possível de representar no campo $CONST$ de 12 bits). Finalmente, deverá ser transferido controlo para a primeira micro-instrução da micro-rotina de carregamento de instrução, que, neste caso, executará a primeira instrução da subrotina de interrupção.

```
IH0:  R8←RE, EINT:  CAR←IF0      ; Guarda RE
IH1:  M[SP]←R8, SP←SP-1
IH2:  M[SP]←PC, SP←SP-1, IAK←1
IH3:  R9←VECTINT
IH4:  R8←0200h
IH5:  R9←R9-R8
IH6:  PC←M[R9]
IH7:  RE←R0, CAR←IF0
```

Programa 12.9: Micro-rotina de tratamento de interrupções.

12.3.6 Geração do Micro-código

Definida a estrutura dos micro-programas e conhecidos os detalhes de cada um dos blocos, basta agora definir o valor dos bits de cada micro-instrução. Considere-se, por exemplo, a primeira micro-instrução a ser executada durante a fase de carregamento de instrução:

```
IF0:  RI←M[PC]  ; Carrega RI
```

Em primeiro lugar, há que identificar o tipo de micro-instrução que poderá ser utilizado para efectuar as transferências entre registos indicadas. Neste caso, pretende-se carregar o registo de instrução com o valor da posição de memória apontada pelo contador de programa. O sinal que controla o carregamento do registo de instrução, LI, só está disponível no formato $F = 1$, o que define imediatamente o tipo de micro-instrução a utilizar.

Analisando agora o circuito de dados na Figura 12.1, pode-se observar que, para conseguir o funcionamento pretendido, é necessário garantir que:

1. O porto A do banco de registos seja endereçado com o número do registo que guarda o PC, ou seja, $15d = 1111b$.
2. O multiplexador MUXA seleccione a entrada 0, colocando o valor do PC no barramento de endereços da memória.
3. O sinal LI esteja activo.
4. Os sinais de controlo de escrita em memória e no banco de registos estejam inactivos.
5. O sinal que controla a escrita no registo de estado, LF, esteja inactivo.
6. O sinal que controla a escrita no registo SBR, LS, esteja inactivo.

Uma vez que o formato da micro-instrução é o formato 1, isso força automaticamente os valores pretendidos nos sinais que controlam a escrita na memória e o multiplexador `MUXA`.

Analisando o circuito de controlo, na Figura 12.13, verifica-se que é necessário garantir que:

1. O registo `CAR` seja incrementado. Para tal, é necessário:
   - Controlar o multiplexador `MUX5` com o valor 00
   - Colocar 0 no controlo do multiplexador `MUX4`, seleccionando a entrada 0 do multiplexador `MUXCOND` e colocando o sinal CC a 1, por forma a que $CCOND$ seja 0.
2. O multiplexador `MUXAD` deverá seleccionar a sua entrada RAD, que deverá tomar o valor $1111b$, uma vez que se pretende este valor em $SelAD$.

Chega-se assim à conclusão que, nesta micro-instrução, os seguintes valores deverão estar definidos:

$F = 1$
$M5 = 00$
$MCOND = 000$
$CC = 1$
$LI = 1$
$LF = 0$
$LS = 0$
$WR = 0$
$MAD = 1$
$RAD = 1111$

Estes valores definem a micro-instrução ilustrada na Figura 12.15, onde os valores que não são relevantes foram deixados em branco. Arbitrando agora

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

1	M5	SR1	SR2	LS	MCOND	CC	LI	LF	CONST/NA	WR	MD	MAD	RAD
1	00			0	000	1	1	0		0		1	1111

Figura 12.15: Codificação da micro-instrução `RI←M[PC]`.

que os valores não relevantes são colocados a 0, obtém-se finalmente o valor que os bits deverão ter para esta micro-instrução: `8060001Fh`.

Um exemplo ligeiramente mais complexo permite-nos ilustrar a utilização de uma micro-instrução do tipo F=0 e o uso das memórias de mapeamento. Considere-se então a micro-instrução:

```
IF1:  PC←PC+1, CAR←ROMA[OPCODE]
```

No circuito de dados, é necessário controlar a unidade aritmética por forma a que esta efectue um incremento, o que significa colocar $CULA$ igual a 00101 e seleccionar o `PC` no porto A do banco de registos. É também necessário seleccionar a entrada 0 do multiplexador `MUXA` e a entrada `0` do multiplexador `MUXD`.

Finalmente, é necessário activar o sinal de escrita nos registos WR e garantir que todos os outros sinais de escrita em registos estão desactivados.

Na unidade de controlo é necessário seleccionar o valor 1111 no sinal $SelAD$ assim como forçar $M5$ a 10, por forma a que a saída da memória de mapeamento A seja seleccionada. É ainda necessário garantir que o sinal IAK fica a 0.

Estas considerações conduzem à definição dos valores dos bits descritos na Figura 12.16. Arbitrando, como anteriormente, que os campos não definidos

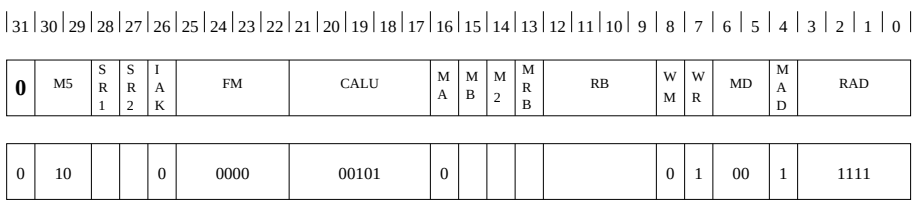

Figura 12.16: Micro-instrução `PC ← PC+1, CAR ← ROMA[OPCODE]`.

são preenchidos com o valor 0, obtém-se o valor final para esta micro-instrução, `400A009Fh`.

Sumário

Este capítulo descreveu a estrutura interna do processador P3, um processador micro-programado de 16 bits, cujo conjunto de instruções tinha sido estudado no capítulo anterior.

As duas componentes mais importantes deste processador são o circuito de dados e a unidade de controlo. O circuito de dados é constituído por um banco de registos e uma unidade lógica e aritmética, já estudadas em capítulos anteriores, além dos registos de instrução e de estado e dos diversos barramentos de interligação. A unidade de controlo é baseada num micro-sequenciador que gera a sequência de sinais que controlam o funcionamento do circuito de dados. O micro-sequenciador usa uma unidade de teste de condições, uma unidade de mapeamento e uma unidade que controla o banco de registos.

A parte final do capítulo foi dedicada a estudar a forma como se definem os micro-programas que controlam o funcionamento deste micro-processador e a forma como é realizada cada uma das suas instruções.