

Manual do PEPE

1	Introdução	1
2	Ligações do módulo PEPE.....	1
3	Registos.....	3
3.1	Registo de Estado (RE)	3
3.2	Registo SP (Stack Pointer)	4
3.3	Registo de Ligação (RL)	4
3.4	Registo Configuração do Núcleo do PEPE (RCN).....	5
4	Excepções.....	5
5	Conjunto de instruções.....	6
5.1	Instruções de dados	6
5.2	Instruções de controlo de fluxo.....	8
5.3	Instruções.....	8
6	Aspectos adicionais do assembler	16
6.1	Comentários.....	16
6.2	Literais.....	16
6.3	Etiquetas	16
6.4	Pseudo-Instruções ou Directivas.....	17

1 Introdução

O módulo PEPE implementa um microprocessador de 16 bits. Este manual é apenas um resumo das suas características.

2 Ligações do módulo PEPE

O módulo PEPE tem as seguintes ligações:

Designação	Nº de bits	Descrição
RESET	1	Inicialização (activo a 0)
INT0	1	Interrupção 0 (nível de activação programável)
INT1	1	Interrupção 1 (nível de activação programável)
INT2	1	Interrupção 2 (nível de activação programável)
INT3	1	Interrupção 3 (nível de activação programável)
INTA	1	(<i>Interrupt Acknowledge</i>) Atendimento da Interrupção (activo a 0; só para a interrupção 3)
CLOCK	1	Entrada de relógio
DATAH	8	D(15..8) - Byte de maior peso do barramento de dados
DATAL	8	D(7..0) - Byte de menor peso do barramento de dados
A15..A0	16	A(15..0) - Barramento de endereços
BA	1	(<i>Byte Addressing</i>) Endereçamento de byte BA=1 – acessos à memória em byte BA=0 – acessos à memória em palavra
RD	1	Activo a 0 nos ciclos de leitura à memória
WR	1	Activo no flanco 0 para 1 nos ciclos de escrita na memória
WAIT	1	WAIT=1 – prolonga o ciclo de acesso à memória WAIT=0 – ciclo de acesso à memória com duração mínima
BRQ	1	(<i>Bus Request</i>) Pedido de DMA, activo a 1
BGT	1	(<i>Bus Grant</i>) Autorização para DMA, activo a 1

Os pinos INT0 a INT3 permitem gerar interrupções externas. Estes pinos são activos no flanco de 0 para 1. Os pinos estão ordenados por prioridade, sendo INT0 o mais prioritário e o INT3 o menos prioritário. É possível inibir todas as interrupções externas e cada interrupção em particular de forma independente das restantes.

A interrupção correspondente ao pino INT3 é vectorizada e exige um controlador de interrupções (PIC – *Programmable Interrupt Controller*) adequado (não descrito neste documento). Quando atende esta interrupção, o processador leva o pino INTA (*Interrupt Acknowledge*) a 0. Esse pino deve ligar ao PIC, que nessa altura é suposto colocar no byte de menor peso do barramento de dados um número (0 a 255) que a rotina de atendimento desta excepção pode usar para identificar a fonte da interrupção.

O pino BA destina-se a suportar o endereçamento de byte, juntamente com o bit A0 do barramento de endereços. Os bits A15 a A1 definem qual a palavra de memória endereçada, ficando o acesso a cada um dos bytes dessa palavra dependente dos valores de A0 e BA, de acordo com a seguinte tabela.

Tipo de acesso	BA	A0	Byte da palavra acedido
Palavra (16 bits)	0	0	Os dois (acesso a toda a palavra)
		1	Illegal. Acesso de palavra desalinhado. Gera excepção.
Byte (8 bits)	1	0	Acède só ao byte no endereço par
		1	Acède só ao byte no endereço ímpar

O pinos RD e WR são usados no ciclo de acesso à memória para leitura e escrita, respectivamente, e são activos a 0. Num dado ciclo, apenas um deles estará activo. A transferência de dados está completa quando estes pinos transitam de 0 para 1.

3 Registos

A tabela seguinte indica quais os registos do PEPE.

Número	Sigla	Nome e descrição
---	PC	Contador de Programa (<i>Program Counter</i>)
0 a 10	R0 a R10	Registos de uso geral
11	RL ou R11	Registo de Ligação (usado para guardar o PC nas instruções CALLF e RETF, para otimizar as chamadas a rotinas que não chamam outras)
12	SP ou R12	Apontador da Pilha (<i>Stack Pointer</i>)
13	RE ou R13	Registo de Estado (<i>flags</i>)
14	BTE ou R14	Base da Tabela de Excepções
15	TEMP ou R15	Registo temporário, usado na implementação de algumas instruções (não usar em programação do utilizador)

3.1 Registo de Estado (RE)

O RE (Registo de Estado), contém os bits de estado e de configuração que interessa salvar (na chamada de rotinas e atendimento de excepções) e repôr (no retorno), com a disposição e significado indicados na figura e tabela seguintes. A operação de *reset* do processador coloca todos os bits do Registo de Estado a 0.

15																0
R	NP	DE	IE3	IE2	IE1	IE0	IE	TD	TV	B	A	V	C	N	Z	

Bit	Sigla	Nome e descrição	Tipo
0	Z	Zero. Este bit é colocado a 1 pelas operações da ALU que produzem zero como resultado.	Estado
1	N	Negativo. Este bit é colocado a 1 pelas operações da ALU que produzem um número negativo (bit de maior peso a 1) como resultado.	Estado
2	C	Transporte (<i>Carry</i>). Este bit é colocado a 1 pelas operações da ALU que geram transporte.	Estado
3	V	Excesso (<i>Overflow</i>). Este bit é colocado a 1 pelas operações da ALU cujo resultado é demasiado grande (em módulo) para ser representado correctamente, seja positivo ou negativo.	Estado
5, 4	B, A	Reservados para utilização futura	A definir
6	TV	Excepção em caso de excesso (<i>Trap on overflow</i>). Se este bit estiver a 1, é gerada a excepção EXCESSO na instrução que produzir o excesso. Se estiver a 0, o excesso só actualiza o bit V.	Configuração
7	TD	Excepção em caso de divisão por 0 (<i>Trap on DIV0</i>). Se este bit estiver a 1, é gerada a excepção DIV0 numa instrução DIV ou UDIV com quociente 0 (não é gerada a excepção EXCESSO nem o bit V é posto a 1)	Configuração
8	IE	Permissão de Interrupções Externas (<i>Interrupt Enable</i>). Só com este bit a 1 as interrupções externas poderão ser atendidas	Configuração
9	IE0	Permissão da Interrupção Externa 0 (<i>Interrupt Enable</i>). Só com este bit a 1 os pedidos de interrupção no pino INT0 poderão ser atendidos	Configuração
10	IE1	Idem, para a interrupção INT1	Configuração
11	IE2	Idem, para a interrupção INT2	Configuração
12	IE3	Idem, para a interrupção INT3	Configuração
13	DE	Permissão de acessos directos à memória (<i>DMA Enable</i>). Só com este bit a 1 os pedidos de DMA no pino BRQ serão tidos em conta e eventualmente atendidos pelo processador	Configuração
14	NP	Nível de Protecção. 0=Sistema; 1=Utilizador. Define o nível de protecção corrente.	Estado
15	R	Reservado para utilização futura	A definir

3.2 Registo SP (Stack Pointer)

O registo SP (*Stack Pointer*, ou Apontador da Pilha), contém o índice da última posição ocupada da pilha (topo), que cresce decrementando o SP. As operações de PUSH decrementam o SP de 2 unidades e armazenam um valor na nova posição. As operações de POP fazem a sequência inversa. Por isso, o SP deve ser inicializado com o endereço imediatamente a seguir à zona de memória atribuída à pilha (tem de ser um valor par).

3.3 Registo de Ligação (RL)

O RL (Registo de Ligação) destina-se a guardar o endereço de retorno quando a rotina invocada é terminal, isto é, não invoca outras. No retorno, o PC será actualizado a partir do RL. A vantagem deste esquema é evitar uma operação de escrita em memória, causada pelo guardar do endereço de retorno na pilha. Realmente, muitas rotinas não chamam outras, e uma simples pilha de uma posição (o RL) em registo é muito mais rápida de aceder do que uma pilha verdadeira em memória. As instruções CALL e RET usam a pilha normalmente. As instruções CALLF e RETF utilizam o RL. Cabe ao compilador (ou ao programador de *assembly*) decidir se usa umas ou outras. Naturalmente, não se pode invocar uma rotina com CALL e retornar com RETF (ou invocar com CALLF e retornar

com RET). O RL (ou R11) pode ser usado como um registo de uso geral quando não estiver em uso por um par CALLF-RETF.

3.4 Registo Configuração do Núcleo do PEPE (RCN)

Este é um registo auxiliar que pode ser escrito ou lido (com MOV) e serve para configurar a sensibilidade dos pinos de interrupção (por omissão são sensíveis ao flanco de 0 → 1). O formato dos bits deste registo é o seguinte:

Bit	Sigla	Nome e descrição
1 e 0	NSI0	Nível de sensibilidade da interrupção 0 (pino INT0). 00=flanco de 0 para 1 (com memória); 01= flanco de 1 para 0 (com memória) ; 10=nível 1 (sem memória) ; 11=nível 0 (sem memória).
3 e 2	NSI1	Idem, para o pino de interrupção INT1
5 e 4	NSI2	Idem, para o pino de interrupção INT2
7 e 6	NSI3	Idem, para o pino de interrupção INT3
15 a 9	R	Reservados

4 Excepções

Designam-se por excepções os eventos a que o processador é sensível e que constituem alterações, normalmente pouco frequentes, ao fluxo normal de instruções de um programa.

As excepções podem ter origem externa (correspondentes à activação de pinos externos do processador) ou interna (decorrentes tipicamente de erros na execução das instruções).

Existem alguns pinos do PEPE (INT0 a INT3) que originam excepções explicitamente para interromper o fluxo do programa com o fim de lidar com eventos assíncronos ao programa e associados tipicamente com os periféricos. Essas excepções designam-se por interrupções.

A cada excepção está associada uma rotina de tratamento da excepção (ou rotina de serviço da excepção, ou simplesmente rotina de excepção), cujo endereço consta da Tabela de Excepções, que contém uma palavra (o endereço da rotina de tratamento) para cada uma das excepções suportadas pelo processador.

A Tabela de Excepções começa no endereço indicado pelo registo BTE (Base da Tabela de Excepções), que deverá ser previamente inicializado com um valor adequado.

A tabela seguinte descreve as excepções que o PEPE suporta.

Endereço dentro da Tabela de Excepções	Excepção	Causa	Ocorre em	Mascarável
00H	INT0	O pino INT0 do processador é activado (com IE=1, IE0=1).	Qualquer altura	Sim
02H	INT1	O pino INT1 do processador é activado (com IE=1, IE1=1).	Qualquer altura	Sim
04H	INT2	O pino INT2 do processador é activado (com IE=1, IE2=1).	Qualquer altura	Sim
06H	INT3	O pino INT3 do processador é activado (com IE=1, IE3=1).	Qualquer altura	Sim
08H	EXCESSO	Uma operação aritmética gera excesso (<i>overflow</i>) se TV=1 no RE	Execução	Sim
0AH	DIV0	Uma operação de divisão falha por o quociente ser zero se TD=1 no RE	Execução	Sim
0CH	SOFTWARE	A instrução SWE (<i>Software Exception</i>) é executada. Usada tipicamente como chamada ao sistema operativo	Execução	Não
0EH	COD_INV	A Unidade de Controlo encontra uma combinação inválida de <i>opcode</i> . Pode ser encarada como uma excepção SWE com <i>opcode</i> próprio e portanto permite estender o conjunto de instruções por meio de software (rotina de excepção que verifica qual o <i>opcode</i> que gerou a excepção e invoca uma rotina adequada).	Descodificação	Não
10H	D_DESALINHADO	É feito um acesso de 16 bits à memória (dados) especificando um endereço ímpar	Execução	Não
12H	I_DESALINHADO	É feita uma busca à memória (<i>fetch</i>) tendo o PC um endereço ímpar	Busca	Não

5 Conjunto de instruções

5.1 Instruções de dados

A tabela seguinte sumariza os modos de endereçamento, isto é, as formas de obter os operandos.

Modo de endereçamento	Obtenção do operando	Nº de bits na instrução	Exemplos de instruções
Imediato	Constante (dados)	4	ADD R1, 3
		8	MOVL R2, 34H MOVH R2, 0F3H
Registo	Rs	4	ADD R1, R2
Indirecto	[Rs]	4	MOV R1, [R2]
Baseado	[Rs + constante]	4 + 4	MOV R1, [R2+3]
Indexado	[Rs + Ri]	4 + 4	MOV R1, [R2+R3]
Relativo	Constante (endereços)	8	JZ 100H
		12	CALL 100H
Implícito	[SP]	0	PUSH, POP
	SP, PC	0	RET, CALL

As instruções MOVL e MOVH permitem especificar uma constante de 8 bits para inicializar apenas um dos bytes (o de menor e de maior peso, respectivamente) de um dado registo. Permitem resolver o problema de inicializar os 16 bits de um registo com instruções de apenas 16 bits (usando duas instruções). O assembler permite especificar apenas uma instrução MOV com uma constante de 16 bits, gerando as instruções necessárias de acordo com o valor da constante

ASSEMBLY	EXEMPLOS	INSTRUÇÕES EQUIVALENTES	RTL	EFEITO
MOV Rd, k	$-128 \leq k \leq +127$ MOV R0, -128 MOV R3, 0 MOV R6, +127	MOVL Rd, k	$Rd(7..0) \leftarrow k(7..0)$ $Rd(15..8) \leftarrow k(7)\{8\}$	Rd fica com uma cópia do valor da constante (de 8 bits) estendida para 16 bits com o seu bit de sinal, k(7). O valor anterior de Rd é destruído.
	$k \leq -129$ ou $k \geq 128$ MOV R7, -32768 MOV R2, -1000 MOV R5, +500 MOV R8, +32767	MOVL Rd, k(7..0) MOVH Rd, k(15..8)	$Rd(7..0) \leftarrow k(7..0)$ $Rd(15..8) \leftarrow k(7)\{8\}$ $Rd(15..8) \leftarrow k(15..8)$	Rd fica com uma cópia do valor da constante (de 16 bits). O valor anterior de Rd é destruído.

Exemplos:

ASSEMBLY	CONSTANTE (HEXADECIMAL 16 BITS)	INSTRUÇÕES MÁQUINA
MOV R1, 0	00 00H	MOVL, R1, 00H
MOV R1, +1	00 01H	MOVL, R1, 01H
MOV R1, +127	00 7FH	MOVL, R1, 7FH
MOV R1, +32767	7F FFH	MOVL, R1, FFH MOVH, R1, 7FH
MOV R1, -1	FF FFH	MOVL, R1, FFH
MOV R1, -128	FF80H	MOVL, R1, 80H
MOV R1, -32768	80 00H	MOVL, R1, 00H MOVH, R1, 80H

A tabela seguinte descreve as formas de acesso à memória em dados e a sua utilização típica.

Instrução		Modos de endereçamento	Operação com a memória	Utilização típica
MOV	Rd, [Rs]	Indirecto	Leitura da memória (16 bits)	Transferência de variáveis (16 bits) entre memória e registos
MOV	Rd, [Rs + off]	Baseado		
MOV	Rd, [Rs + Ri]	Indexado		
MOV	[Rd], Rs	Indirecto	Escrita da memória (16 bits)	
MOV	[Rd + off], Rs	Baseado		
MOV	[Rd + Ri], Rs	Indexado		
MOVB	Rd, [Rs]	Indirecto	Leitura da memória (8 bits)	Processamento de bytes individuais (cadeias de caracteres ASCII, por exemplo)
MOVB	[Rd], Rs	Indirecto	Escrita da memória (8 bits)	
MOVP	Rd, [Rs]	Indirecto	Leitura da memória (16 bits) sem usar a <i>cache</i> nem a memória virtual	Leitura de periféricos
MOVP	[Rd], Rs	Indirecto		Escrita de periféricos
SWAP	Rd, [Rs] ou [Rs], Rd	Indirecto	Troca atómica de dados (16 bits) entre memória e registo.	Troca de dados, semáforos

			Mesmo com <i>caches</i> , o acesso à memória é forçado	
PUSH	Rd	Implícito (SP)	Escrita na pilha	Guardar valores para mais tarde recuperar
POP	Rd	Implícito (SP)	Leitura da pilha	Recuperar valores guardados na pilha

5.2 Instruções de controlo de fluxo

Os aspectos mais importantes a ter em conta à partida são os seguintes:

- O PEPE suporta endereçamento de byte mas os acessos em busca de instrução têm de ser alinhados, pelo que os endereços têm de ser pares (senão é gerada uma exceção quando o acesso for feito). Para aumentar a gama de endereços que é possível atingir a partir das instruções que aceitam um operando imediato, o valor do operando codificado na instrução é entendido pelo PEPE como designando palavras (instruções) e não bytes, pelo que depois, na implementação das instruções, o PEPE multiplica automaticamente o operando por 2 (seja positivo ou negativo) antes de o utilizar no cálculo do endereço destino do salto;
- Todas as instruções de salto e de chamada de rotinas com operando imediato são relativas, isto é, o operando (em complemento para 2) é multiplicado por 2 e somado ao EIS (Endereço da Instrução Seguinte à instrução de salto). No entanto, para facilitar o utilizador, o assembler requer não uma constante numérica mas sim um endereço simbólico, ou etiqueta (*label*), e o assembler faz as contas. O assembler gera um erro caso a constante (8 ou 12 bits, depende da instrução) não seja suficiente para codificar a diferença entre o valor da etiqueta e EIS. Se for o caso, o utilizador deve usar as instruções JUMP e CALL com endereçamento por registo. Estas últimas já têm endereçamento absoluto, isto é, o valor do registo é o novo endereço da instrução a buscar (e não somado com o anterior). Note-se que

L1: JUMP L1 ; operando imediato \Rightarrow endereçamento relativo

resulta num ciclo infinito e o valor do operando codificado na instrução JUMP é -1 (o que corresponde a subtrair -2 a EIS).

5.3 Instruções

As instruções sombreadas são reconhecidas pelo assembler mas na realidade podem ser sintetizadas com recurso a outras, pelo que não gastam codificações de instruções. São oferecidas apenas como notação alternativa para comodidade do programador de linguagem *assembly* e maior clareza dos programas.

As linhas marcadas com “Livre” correspondem às codificações possíveis e ainda não ocupadas.

Os campos marcados com “XXXX” não são relevantes e podem ter qualquer valor (são ignorados pelo PEPE).

Na coluna “Acções” indica-se o significado de cada instrução numa linguagem de transferência de registos (RTL), cujos aspectos essenciais são indicados pela tabela seguinte.

Se o RE for o destino de uma operação, no RE fica exactamente o resultado dessa operação. Neste caso em particular, os bits de estado não são afectados pelo valor do resultado ($Z \leftarrow 1$ se o resultado for 0000H, por exemplo) como nas outras operações, mas ficam directamente com os bits correspondentes do resultado.

Simbologia	Significado	Exemplo
Ri	Registo principal <i>i</i> (R0 a R15, incluindo RL, SP, RE, BTE e TEMP)	R1
PC	Registo <i>Program Counter</i> . Só usado do lado esquerdo da atribuição.	PC \leftarrow expressão
EIS	Endereço da Instrução Seguinte. Não é um registo, mas apenas uma notação que representa o valor do endereço da instrução seguinte (ou seja, é o endereço da instrução corrente acrescido de 2 unidades).	EIS
Mw[<i>end</i>]	Célula de memória de 16 bits que ocupa os endereços <i>end</i> e <i>end</i> +1 (<i>end</i> tem de ser par, senão gera uma excepção). O PEPE usa o esquema Big-Endian, o que significa que o byte de <u>menor</u> peso de Mw[<i>end</i>] está no endereço <i>end</i> +1.	Mw[R1+2] Se R1=1000H, o byte de menor peso está em 1003H e o de maior peso em 1002H
Mb[<i>end</i>]	Célula de memória de 8 bits cujo endereço é <i>end</i> (que pode ser par ou ímpar)	Mb[R3+R4]
(i)	Bit <i>i</i> de um registo ou de uma célula de memória	R2(4) Mw[R1](0)
Ra(i..j)	Bits <i>i</i> a <i>j</i> (contíguos) do registo Ra (<i>i</i> \geq <i>j</i>)	R2(7..3)
bit{n}	Sequência de <i>n</i> bits obtida pela concatenação de <i>n</i> cópias de <i>bit</i> , que é uma referência de um bit (pode ser 0, 1 ou Ra(i))	0{4} equivale a 0000 R1(15){2} equivale a R1(15) R1(15)
<i>dest</i> \leftarrow <i>expr</i>	Atribuição do valor de uma expressão (<i>expr</i>) a uma célula de memória ou registo (<i>dest</i>). Um dos operandos da atribuição (expressão ou destino) tem de ser um registo ou um conjunto de bits dentro do processador. O operando da direita é todo calculado primeiro e só depois se destrói o operando da esquerda, colocando lá o resultado de <i>expr</i> . <i>dest</i> e <i>expr</i> têm de ter o mesmo número de bits.	R1 \leftarrow M[R2] M[R0] \leftarrow R4 + R2 R1(7..0) \leftarrow R2(15..8)
Z, N, C, V, IE, IE0 a IE4, DE, NP	Bits de estado no RE – Registo de Estado	V \leftarrow 0
<i>Expr</i> : <i>acção</i>	Executa a <i>acção</i> se <i>expr</i> for verdadeira (<i>expr</i> tem de ser uma expressão booleana)	((N \oplus V) \vee Z)=1 : PC \leftarrow EIS + 2
\wedge , \vee , \oplus	E, OU, OU-exclusivo	R1 \leftarrow R2 \wedge R3
	Concatenação de bits (os bits do operando da esquerda ficam à esquerda, ou com maior peso)	R1 \leftarrow R2(15..8) 00H

Classe	Sintaxe em <i>assembly</i>		Campos da instrução (16 bits)				Acções	Flags afectadas	Comentários
			1º opcode (4bits)	2º opcode (4bits)	1º operando (4bits)	2º operando (4bits)			
Instruções aritméticas	ADD	Rd, Rs	ARITOP	ADD	Rd	Rs	$Rd \leftarrow Rd + Rs$	Z, N, C, V	
		Rd, k		ADDI	Rd	k	$Rd \leftarrow Rd + k$	Z, N, C, V	$k \in [-8 .. +7]$
	ADDC	Rd, Rs		ADDC	Rd	Rs	$Rd \leftarrow Rd + Rs + C$	Z, N, C, V	
		Rd, Rs		SUB	Rd	Rs	$Rd \leftarrow Rd - Rs$	Z, N, C, V	
		Rd, k		SUBI	Rd	k	$Rd \leftarrow Rd - k$	Z, N, C, V	$k \in [-8 .. +7]$
	SUBB	Rd, Rs		SUBB	Rd	Rs	$Rd \leftarrow Rd - Rs - C$	Z, N, C, V	
		Rd, Rs		CMP	Rd	Rs	$(Rd - Rs)$	Z, N, C, V	Rd não é alterado
	CMP	Rd, k		CMPI	Rd	k	$(Rd - k)$	Z, N, C, V	$k \in [-8 .. +7]$ Rd não é alterado
	MUL	Rd, Rs		MUL	Rd	Rs	$Rd \leftarrow Rd * Rs$	Z, N, C, V	O registo Rs é alterado
	DIV	Rd, Rs		DIV	Rd	Rs	$Rd \leftarrow \text{quociente}(Rd / Rs)$	Z, N, C, V $V \leftarrow 0$	Divisão inteira
	MOD	Rd, Rs		MOD	Rd	Rs	$Rd \leftarrow \text{resto}(Rd / Rs)$	Z, N, C, V $V \leftarrow 0$	Resto da divisão inteira
	NEG	Rd		NEG	Rd	xxxx	$Rd \leftarrow -Rd$	Z, N, C, V	Complemento para 2 $V \leftarrow 1$ se Rd for 8000H
	Livre								
	Livre								

Classe	Sintaxe em <i>assembly</i>	Campos da instrução (16 bits)				Acções	Flags afectadas	Comentários
		1º opcode (4bits)	2º opcode (4bits)	1º operando (4bits)	2º operando (4bits)			
Instruções de bit	AND Rd, Rs	BITOP	AND	Rd	Rs	$Rd \leftarrow Rd \wedge Rs$	Z, N	
	OR Rd, Rs		OR	Rd	Rs	$Rd \leftarrow Rd \vee Rs$	Z, N	
	NOT Rd		NOT	Rd	xxxx	$Rd \leftarrow Rd \oplus FFFFH$	Z, N	Complemento para 1
	XOR Rd, Rs		XOR	Rd	Rs	$Rd \leftarrow Rd \oplus Rs$	Z, N	
	TEST Rd, Rs		TEST	Rd	Rs	$Rd \wedge Rs$	Z, N	Rd não é alterado
	BIT Rd, n		BIT	Rd	n	$Z \leftarrow Rd(k) \oplus 1$	Z	Rd não é alterado
	SET Rd, n		SETBIT	Rd	n	$Rd(n) \leftarrow 1$	Z, N ou outra (se Rd for RE)	$n \in [0 .. 15]$ Se Rd=RE, afecta apenas RE(n)
	EI		SETBIT	RE	IE_index	$RE(IE_index) \leftarrow 1$	EI	Enable interrupts
	EI0		SETBIT	RE	IE0_index	$RE(IE0_index) \leftarrow 1$	EI0	Enable interrupt 0
	EI1		SETBIT	RE	IE1_index	$RE(IE1_index) \leftarrow 1$	EI1	Enable interrupt 1
	EI2		SETBIT	RE	IE2_index	$RE(IE2_index) \leftarrow 1$	EI2	Enable interrupt 2
	EI3		SETBIT	RE	IE3_index	$RE(IE3_index) \leftarrow 1$	EI3	Enable interrupt 3
	SETC		SETBIT	RE	C_index	$RE(C_index) \leftarrow 1$	C	Set Carry flag
	EDMA		SETBIT	RE	DE_index	$RE(DE_index) \leftarrow 1$	DE	Enable DMA
	CLR Rd, n		CLRBIT	Rd	n	$Rd(n) \leftarrow 0$	Z, N ou outra (se Rd for RE)	$n \in [0 .. 15]$ Se Rd=RE, afecta apenas RE(n)
	DI		CLRBIT	RE	IE_index	$RE(IE_index) \leftarrow 0$	EI	Disable interrupts
	DI0		CLRBIT	RE	IE0_index	$RE(IE0_index) \leftarrow 0$	EI0	Disable interrupt 0
	DI1		CLRBIT	RE	IE1_index	$RE(IE1_index) \leftarrow 0$	EI1	Disable interrupt 1
	DI2		CLRBIT	RE	IE2_index	$RE(IE2_index) \leftarrow 0$	EI2	Disable interrupt 2
	DI3		CLRBIT	RE	IE3_index	$RE(IE3_index) \leftarrow 0$	EI3	Disable interrupt 3
	CLRC		CLRBIT	RE	C_index	$RE(C_index) \leftarrow 0$	C	Clear Carry flag
	DDMA		CLRBIT	RE	DE_index	$RE(DE_index) \leftarrow 0$	DE	Disable DMA
	CPL Rd, n		CPLBIT	Rd	n	$Rd(n) \leftarrow Rd(n) \oplus 1$	Z, N ou outra (se Rd for RE)	$n \in [0 .. 15]$ Se Rd=RE, afecta apenas RE(n)
	CPLC		CPLBIT	RE	C_index	$RE(C_index) \leftarrow RE(C_index) \oplus 1$	C	Complement Carry flag

Classe	Sintaxe em <i>assembly</i>	Campos da instrução (16 bits)				Acções	Flags afectadas	Comentários
		1º opcode (4bits)	2º opcode (4bits)	1º operando (4bits)	2º operando (4bits)			
Instruções de bit	SHR Rd, n	BITOP	SHR	Rd	n	$n > 0 : C \leftarrow Rd(n-1)$ $n > 0 : Rd \leftarrow 0\{n\} \parallel Rd(15..n)$	Z, N, C	$n \in [0 .. 15]$ Se $n=0$, actualiza Z e N (C não)
	SHL Rd, n		SHL	Rd	n	$n > 0 : C \leftarrow Rd(15-n+1)$ $n > 0 : Rd \leftarrow Rd(15-n..0) \parallel 0\{n\}$	Z, N, C	$n \in [0 .. 15]$ Se $n=0$, actualiza Z e N (C não)
	SHRA Rd, n	ARITOP	SHRA	Rd	n	$n > 0 : C \leftarrow Rd(n-1)$ $n > 0 : Rd \leftarrow Rd(15)\{n\} \parallel Rd(15..n)$	Z, N, C	$n \in [0 .. 15]$ Se $n=0$, actualiza Z e N (C não)
	SHLA Rd, n		SHLA	Rd	n	$n > 0 : C \leftarrow Rd(15-n+1)$ $n > 0 : Rd \leftarrow Rd(15-n..0) \parallel 0\{n\}$	Z, N, C, V	$n \in [0 .. 15]$ Se $n=0$, actualiza Z e N (C não) $V \leftarrow 1$ se algum dos bits que sair for diferente do $Rd(15)$ após execução
	ROR Rd, n	BITOP	ROR	Rd	n	$n > 0 : C \leftarrow Rd(n-1)$ $n > 0 : Rd \leftarrow Rd(n-1..0) \parallel Rd(15..n)$	Z, N, C	$n \in [0 .. 15]$ Se $n=0$, actualiza Z e N (C não)
	ROL Rd, n		ROL	Rd	n	$n > 0 : C \leftarrow Rd(15-n+1)$ $n > 0 : Rd \leftarrow Rd(15-n..0) \parallel Rd(15..15-n+1)$	Z, N, C	$n \in [0 .. 15]$ Se $n=0$, actualiza Z e N (C não)
	RORC Rd, n		RORC	Rd	n	$n > 0 : (Rd \parallel C) \leftarrow Rd(n-2..0) \parallel C \parallel Rd(15..n-1)$	Z, N, C	$n \in [0 .. 15]$ Se $n=0$, actualiza Z e N (C não)
	ROLC Rd, n		ROLC	Rd	n	$n > 0 : (C \parallel Rd) \leftarrow Rd(15-n+1..0) \parallel C \parallel Rd(15..15-n+2)$	Z, N, C	$n \in [0 .. 15]$ Se $n=0$, actualiza Z e N (C não)
	Livre							
Instruções de transferência de dados	MOV	LDO	Rd	Rs	off/2	$Rd \leftarrow Mw[Rs + off]$	Nenhuma	$off \in [-16 .. +14]$
			Rd	Rs	0000	$Rd \leftarrow Mw[Rs + 0000]$	Nenhuma	
		LDR	Rd	Rs	Ri	$Rd \leftarrow Mw[Rs + Ri]$	Nenhuma	
		STO	Rs	Rd	off/2	$Mw[Rd + off] \leftarrow Rs$	Nenhuma	$off \in [-16 .. +14]$
			Rs	Rd	0000	$Mw[Rd + 0000] \leftarrow Rs$	Nenhuma	
	MOVB	STR	Rs	Rd	Ri	$Mw[Rd + Ri] \leftarrow Rs$	Nenhuma	
		XFER	LDB	Rd	Rs	$Rd \leftarrow 0\{8\} \parallel Mb[Rs]$	Nenhuma	
			STB	Rd	Rs	$Mb[Rd] \leftarrow Rs(7..0)$	Nenhuma	O byte adjacente a $Mb[Rd]$ não é afectado
			LDBS	Rd	Rs	$Rd \leftarrow Mb[Rs](7)\{8\} \parallel Mb[Rs]$	Nenhuma	Idêntica a MOVb Rd, [Rs], mas em que é feita extensão de sinal
			LDP	Rd	Rs	$Rd \leftarrow Mw[Rs]$	Nenhuma	Não usa memória virtual nem caches (para acesso aos periféricos)
	MOVP		STP	Rd	Rs	$Mw[Rd] \leftarrow Rs$	Nenhuma	

Classe	Sintaxe em <i>assembly</i>	Campos da instrução (16 bits)				Acções	Flags afectadas	Comentários	
		1º opcode (4bits)	2º opcode (4bits)	1º operando (4bits)	2º operando (4bits)				
Instruções de transferência de dados	MOVL Rd, k	MOVL	Rd	k		Rd ← k(7){8} k	Nenhuma	k ∈ [-128 .. +127] k é estendido a 16 bits com sinal	
	MOVH Rd, k	MOVH	Rd	k		Rd(15..8) ← k	Nenhuma	k ∈ [0 .. 255] O byte de menor peso não é afectado	
	MOV Rd, k	MOVL	Rd	k		Rd ← k(7){8} k	Nenhuma	Se k ∈ [-128 .. +127]	
	MOV	Rd, k	MOVL	Rd	k(7..0)		Rd ← k(7){8} k(7..0)	Nenhuma	Se k ∈ [-32768 .. -129] ou k ∈ [+128 .. +32767]
			MOVH	Rd	k(15..8)		Rd(15..8) ← k(15..8)	Nenhuma	
			Rd, Rs	XFER	MOVRR	Rd	Rs	Rd ← Rs	Nenhuma
		Ad, Rs	MOVAR		Ad	Rs	Ad ← Rs	Nenhuma	
		Rd, As	MOVRA		Rd	As	Rd ← As	Nenhuma	
		Rd, USP	MOVUR		Rd	xxxx	Rd ← USP	Nenhuma	O SP lido é o de nível utilizador, independentemente do bit NP do RE
		USP, Rs	MOVUR	xxxx	Rs	USP ← Rs	Nenhuma	O SP escrito é o de nível utilizador, independentemente do bit NP do RE	
	SWAP	Rd, Rs	XFER	SWAPR	Rd	Rs	TEMP ← Rd Rd ← Rs Rs ← TEMP	Nenhuma	
		Rd, [Rs] ou [Rs], Rd		SWAPM	Rd	Rs	TEMP ← Mw[Rs] Mw[Rs] ← Rd Rd ← TEMP	Nenhuma	Recomeçável sem reposição de estado mesmo que um dos acessos à memória falhe
	PUSH Rs		PUSH	Rs	xxxx	Mw[SP-2] ← Rs SP ← SP – 2	Nenhuma	SP só é actualizado no fim para ser re-executável	
	POP Rd		POP	Rd	xxxx	Rd ← Mw[SP] SP ← SP + 2	Nenhuma		
	Livre								
	Livre								

Classe	Sintaxe em <i>assembly</i>	Campos da instrução (16 bits)				Acções	Flags afectadas	Comentários
		1º opcode (4bits)	2º opcode (4bits)	1º operando (4bits)	2º operando (4bits)			
Instruções de controlo de fluxo	JZ etiqueta	COND	JZ	dif =(etiqueta – EIS)/2	Z=1: PC ← EIS + (2*dif)	Nenhuma	etiqueta ∈ [EIS - 256 .. EIS + 254]	
	JNZ etiqueta		JNZ	dif =(etiqueta – EIS)/2	Z=0: PC ← EIS + (2*dif)	Nenhuma	etiqueta ∈ [EIS - 256 .. EIS + 254]	
	JN etiqueta		JN	dif =(etiqueta – EIS)/2	N=1: PC ← EIS + (2*dif)	Nenhuma	etiqueta ∈ [EIS - 256 .. EIS + 254]	
	JNN etiqueta		JNN	dif =(etiqueta – EIS)/2	N=0: PC ← EIS + (2*dif)	Nenhuma	etiqueta ∈ [EIS - 256 .. EIS + 254]	
	JP etiqueta		JP	dif =(etiqueta – EIS)/2	(N∨Z)=0: PC ← EIS + (2*dif)	Nenhuma	etiqueta ∈ [EIS - 256 .. EIS + 254]	
	JNP etiqueta		JNP	dif =(etiqueta – EIS)/2	(N∨Z)=1: PC ← EIS + (2*dif)	Nenhuma	etiqueta ∈ [EIS - 256 .. EIS + 254]	
	JC etiqueta		JC	dif =(etiqueta – EIS)/2	C =1: PC ← EIS + (2*dif)	Nenhuma	etiqueta ∈ [EIS - 256 .. EIS + 254]	
	JNC etiqueta		JNC	dif =(etiqueta – EIS)/2	C =0: PC ← EIS + (2*dif)	Nenhuma	etiqueta ∈ [EIS - 256 .. EIS + 254]	
	JV etiqueta		JV	dif =(etiqueta – EIS)/2	V=1: PC ← EIS + (2*dif)	Nenhuma	etiqueta ∈ [EIS - 256 .. EIS + 254]	
	JNV etiqueta		JNV	dif =(etiqueta – EIS)/2	V=0: PC ← EIS + (2*dif)	Nenhuma	etiqueta ∈ [EIS - 256 .. EIS + 254]	
	JEQ etiqueta		JZ	dif =(etiqueta – EIS)/2	Z=1: PC ← EIS + (2*dif)	Nenhuma	etiqueta ∈ [EIS - 256 .. EIS + 254]	
	JNE etiqueta		JNZ	dif =(etiqueta – EIS)/2	Z=0: PC ← EIS + (2*dif)	Nenhuma	etiqueta ∈ [EIS - 256 .. EIS + 254]	
	JLT etiqueta		JLT	dif =(etiqueta – EIS)/2	N⊕V =1 : PC ← EIS + (2*dif)	Nenhuma	etiqueta ∈ [EIS - 256 .. EIS + 254]	
	JLE etiqueta		JLE	dif =(etiqueta – EIS)/2	((N⊕V)∨Z)=1 : PC ← EIS + (2*dif)	Nenhuma	etiqueta ∈ [EIS - 256 .. EIS + 254]	
	JGT etiqueta		JGT	dif =(etiqueta – EIS)/2	((N⊕V)∨Z)=0 : PC ← EIS + (2*dif)	Nenhuma	etiqueta ∈ [EIS - 256 .. EIS + 254]	
	JGE etiqueta		JGE	dif =(etiqueta – EIS)/2	N⊕V =0 : PC ← EIS + (2*dif)	Nenhuma	etiqueta ∈ [EIS - 256 .. EIS + 254]	
	JA etiqueta		JA	dif =(etiqueta – EIS)/2	(C∨Z)=0 : PC ← EIS + (2*dif)	Nenhuma	etiqueta ∈ [EIS - 256 .. EIS + 254]	
	JAE etiqueta		JNC	dif =(etiqueta – EIS)/2	C =0: PC ← EIS + (2*dif)	Nenhuma	etiqueta ∈ [EIS - 256 .. EIS + 254]	
	JB etiqueta		JC	dif =(etiqueta – EIS)/2	C =1: PC ← EIS + (2*dif)	Nenhuma	etiqueta ∈ [EIS - 256 .. EIS + 254]	
	JBE etiqueta		JBE	dif =(etiqueta – EIS)/2	(C∨Z)=1 : PC ← EIS + (2*dif)	Nenhuma	etiqueta ∈ [EIS - 256 .. EIS + 254]	

Classe	Sintaxe em <i>assembly</i>		Campos da instrução (16 bits)				Acções	Flags afectadas	Comentários
			1º opcode (4bits)	2º opcode (4bits)	1º operando (4bits)	2º operando (4bits)			
Instruções de controlo de fluxo	JMP	etiqueta	JMP	dif =(etiqueta – EIS)/2			PC ← EIS + (2*dif)	Nenhuma	etiqueta ∈ [EIS-4096 .. EIS+4094]
		Rs	CODE	JMPR	xxxx	Rs	PC ← Rs	Nenhuma	
	CALL	etiqueta	CALL	dif =(etiqueta – EIS)/2			Mw[SP-2] ← EIS PC ← EIS + (2*dif) SP ← SP - 2	Nenhuma	etiqueta ∈ [EIS-4096 .. EIS+4094] SP só é actualizado no fim para ser re-executável
		Rs	CODE	CR	xxxx	Rs	Mw[SP-2] ← EIS PC ← Rs SP ← SP - 2	Nenhuma	SP só é actualizado no fim para ser re-executável
	CALLF	etiqueta	CALLF	dif =(etiqueta – EIS)/2			RL ← EIS PC ← EIS + (2*dif)	Nenhuma	etiqueta ∈ [EIS-4096 .. EIS+4094]
		Rs	CODE	CRF	xxxx	Rs	RL ← EIS PC ← Rs	Nenhuma	
	RET			RET	xxxx	xxxx	PC ← Mw[SP] SP ← SP + 2	Nenhuma	
	RETF			RETF	xxxx	xxxx	PC ← RL	Nenhuma	
	SWE	k		SWE	k		TEMP ← RE RE ← 0 Mw[SP-2] ← EIS Mw[SP-4] ← TEMP PC ← Mw[BTE+2*k] SP ← SP - 4	Todas colocadas a zero	SP só é actualizado no fim para ser re-executável k ∈ [0 .. 255] Tem de se usar a pilha de sistema
	RFE			RFE	xxxx	xxxx	TEMP ← Mw[SP] PC ← Mw[SP+2] SP ← SP + 4 RE ← TEMP	Todas restauradas	SP só é actualizado no fim para ser re-executável Tem de se usar a pilha de sistema
	NOP			NOP	xxxx	xxxx		Nenhuma	Não faz nada
	Livre								
	Livre								
	Livre								
	Livre								
	Livre								
	Livre								
	Livre								

6 Aspectos adicionais do assembler

6.1 Comentários

Texto ignorado pelo assembler e útil apenas para informação adicional para o programador. Em cada linha, um comentário começa por “;”. Exemplo:

```
MOV R1, R2      ; copia R2 para R1
```

6.2 Literais

Os literais são valores constantes (números ou cadeias de caracteres) podem ser especificados de seis formas no código *assembly*:

- **Valor numérico em binário:** para uma constante numérica ser interpretada em binário deve ser terminada com a letra **b**; são válidos valores entre 0b e 111111111111111b.
- **Valor numérico em decimal:** qualquer valor inteiro entre –32768 e 65535. Pode opcionalmente ser terminado com a letra **d**, embora tal seja assumido quando nenhuma outra base for indicada.
- **Valor numérico em hexadecimal:** para uma constante numérica ser interpretada em hexadecimal deve ser terminada com a letra **h**; são válidos valores entre 0h e ffffh. As constantes em hexadecimal cujo dígito de maior peso é uma letra (a,b,c,d,e ou f) devem ser escritas com um zero antes da letra, de modo a distinguir a constante de uma variável. Assim a constante ffffh deverá ser escrita 0ffffh.
- **Caracter alfanumérico:** um caracter entre plicas, por exemplo, ‘g’, é convertido para o seu código ASCII.
- **Cadeia de caracteres alfanuméricos:** um conjunto de caracteres entre aspas, por exemplo “ola”, é convertido para um conjunto de caracteres ASCII.

É de notar, no entanto, que o uso de literais em código *assembly* (ou qualquer outra linguagem de programação) é desaconselhável. Em vez disso, deve-se usar o comando EQU para definir constantes (ver secção seguinte). Esta prática por um lado torna o código mais legível, pois o símbolo associado à constante dá uma pista sobre a acção que se está a tomar, e por outro lado permite uma actualização mais fácil do código, pois constantes que estão associadas não têm que ser alteradas em vários sítios dentro do código, mas simplesmente na linha do comando EQU.

6.3 Etiquetas

Para referenciar uma dada posição de memória, pode-se colocar uma etiqueta (*label*) antes da instrução que vai ficar nessa posição. A etiqueta consiste num nome (conjunto de caracteres alfanuméricos, mais o carácter ‘_’, em que o primeiro não pode ser um algarismo) seguida de ‘:’. Por exemplo,

```
AQUI: ADD R1, 1
```


Se agora se quiser efectuar um salto para esta instrução, pode-se usar:

JMP AQUI

em vez de se calcular o endereço em que a instrução ADD R1, 1 ficará depois da assemblagem.

6.4 Pseudo-Instruções ou Directivas

Chamam-se pseudo-instruções (ou directivas) ao conjunto de comandos reconhecidos pelo assembler que não são instruções *assembly*, portanto não geram código binário no ficheiro objecto. A função das pseudo-instruções é, por um lado, controlar a forma como o código é gerado (por exemplo, indicando as posições de memória onde colocar o executável ou reservando posições de memória para dados), por outro lado, permitir definir etiquetas (constantes ou posições de memória) que tornam o código mais legível e mais fácil de programar. Nesta secção descrevem-se as pseudo-instruções usadas pelo *assembler* para o processador fornecido com o simulador de circuitos de arquitectura de computadores.

PLACE

Formato: PLACE *etiqueta*

Função: O assembler usa um contador de endereços interno, que vai incrementando em cada instrução assemblada (assim, determina em que endereço fica cada instrução). O comando PLACE permite especificar no campo <endereço> um novo valor desse contador, posicionando assim no espaço de endereçamento blocos de dados ou instruções. Podem existir várias instruções PLACE no mesmo ficheiro *assembly* correspondentes a vários blocos de memória.

EQU

Formato: *símbolo* EQU *constante*

Função: o comando EQU permite associar um valor constante a um símbolo.

WORD

Formato: *etiqueta:* WORD *constante*

Função: o comando WORD permite reservar uma posição de memória para conter uma variável do programa *assembly*, associando a essa posição o nome especificado em *etiqueta*. O campo *constante* indica o valor a que essa posição de memória deve ser inicializada.

STRING

Formato: *etiqueta:* STRING *constante* [,*constante*]

Função: o comando STRING coloca em bytes de memória consecutivos cada uma das constantes nele definidas. Se qualquer dessas constantes for uma cadeia de caracteres o código ASCII de cada um deles é colocado sequencialmente na memória. A etiqueta fica com o endereço do primeiro carácter da primeira constante.

TABLE

Formato: *etiqueta:* TABLE *constante*

Função: o comando TABLE reserva o número de posições de memória especificadas no campo *constante*. *etiqueta* fica com o endereço da primeira posição.