Mestrado Integrado em Eng. de Comunicações

Directivas do Assembler

Microprocessadores

2º Ano – A10

Linguagem assembly

- Nenhum programador utiliza a linguagem máquina;
- Utilizam as mnemónicas da linguagem assembler;
- Um “assemblador” converte essas mnemónicas em código máquina;
- A linguagem assembler é muito próxima da linguagem máquina;
- Cada instrução é convertida directamente em bytes de código máquina (*opcode*);
- Produz código mais eficiente mas é mais trabalhoso para o programador.

Linguagem Assembly

i. *Apresenta um nível de abstracção intermédia entre os dois extremos: linguagem máquina e linguagem alto-nível*

ii. *Facilita a programação pela substituição do código binário da linguagem máquina com símbolos*

    Escrita usando labels (etiquetas), mnemónicas, etc.

iii. *Um programa em assembly não é executável*

    Todos os símbolos tais como mnemónicas, etiquetas devem ser traduzidos para código binário

Assembly->Código Máquina

Dependendo da complexidade do ambiente de programação pode envolver várias etapas até a produção do código executável

i. *Assembler*
   a. *Traduz um programa em assembly para programa em linguagem máquina (código objecto)*
   b. *O código objecto pode estar na forma absoluta ou forma relocatable*

ii. *Linker*
   a. *Combina vários programas objectos na forma relocatable, produzindo um programa executável através da atribuição de endereços absolutos*
   b. *Produz também um ficheiro contendo o mapa de memória e tabela de símbolos*

Contador de Localização

- *O assembler possui um contador de localização para cada um dos cincos segmentos*
  1. *CODE (0000h - FFFFH) Código – Ap: PC*
  2. *DATA (00H – FFH)/(00H–7FH+SFR) Interna de Dados*
  3. *IDATA (00H – FFH)/(00H–7FH+80H-FFH) Interna de Dados Indirecta - Ap:Ri*
  4. *BIT (00H –FFH)/(20H – 2FH) Interna de Dados: Bit Space 00-7Fh*
  5. *XDATA (0000H – FFFFH) Externa de Dados – Ap:Ri e DPTR*

- *Cada contador de localização é inicializado com o valor zero e pode posteriormente ser alterado usando as directivas do assembler*

- *O símbolo $ pode ser usado para especificar o valor do contador de localização do segmento activo:*

```
TABLE:      DB      1,2,3,4,5,6,7,8,9
LEN         EQU     $-TABLE
```

Salto/Invocação genérico

- O ASM51 permite o uso das mnemónicas genéricas JMP ou CALL. Evitando em muitas situações o uso de SJMP, AJMP, LJMP ou ACALL, LCALL;

- A conversão da mnemónica genérica para instrução real segue a seguinte regra:

  1. JMP é substituído por SJMP se não for usada *forward reference* e o destino do salto estiver na gama de [–128,+127] localizações

     » *forward reference*: uso de um símbolo antes da respectiva definição

  2. JMP/CALL é substituído por AJMP/ACALL se não for usada *forward reference* e a instrução que segue JMP/CALL pertencer à mesma página de 2Kbytes que a instrução destino

  3. Caso contrário, será usado a conversão para LJMP/LCALL

Endereçar ao bit

Muitas vezes é necessário realizarmos operações sobre bits.

A linguagem assembly permite o endereçamento ao bit utilizando uma das seguintes formas:

	ON	EQU	7
Explicitamente pelo endereço	SETB	0E7H	
	JNB	99H, $	
Usando o operador .	SETB	ACC.7	
	SETB	224.ON	
	SETB	0E0H.7	
Usando um símbolo pré-definido	JNB	TI, $	
	CLR	C	

Bases numéricas

Decimal	MOV	A, #15
	MOV	A, #15**D**
Binário	MOV	#1111**B**
Octal	MOV	A, #17**Q**
	MOV	A, #17**O**
Hexadecimal	MOV	A, #0F**H**
	MOV	A, #0A5**H**
	~~MOV~~	~~A, #A5**H**~~ ?

Para diferenciar um dado hexadecimal imediato de um símbolo

Operadores de expressão

Símbolo do Operador	Operação
+, -,	Adição, subtracção
/, *	Divisão, multiplicação
MOD	Resto da divisão
OR	Ou-lógico
AND	E-lógico
XOR	Ou-exclusivo
NOT	Complemento
SHR	Rodar à direita
SHL	Rodar à esquerda
HIGH	Obter o MSB
LOW	Obter o LSB
EQ, =	Igual a
NE, <>	Diferente
LT, <	Menor que
LE, <=	Menor ou igual
GT, >	Maior que
GE, >=	Maior ou igual

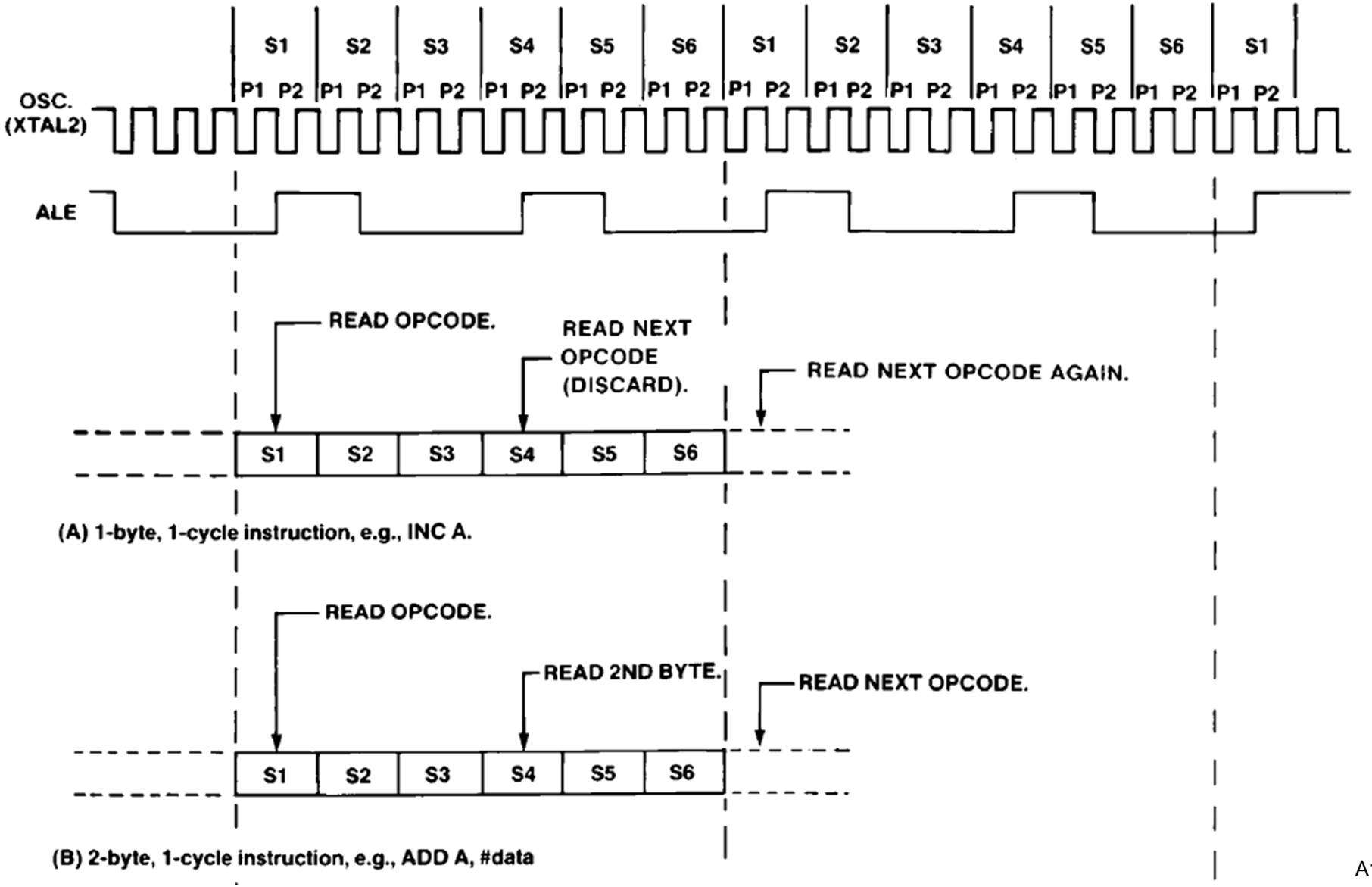
Expressão	Resultado
'A' – 'B'	**0001H**
HIGH(0AADDH)	**0AAH**
LOW(0AADDH)	**0DDH**
7 MOD 4	**3**
1000B SHR 2	**0010B**
NOT 1	**0FFFEH**
'A' SHL 8	**4100H**
5 EQ 8	**0000H**
'A' LT 'B'	**0FFFFH**
3 <= 3	**0FFFFH**
-1	**0FFFFH**
7 NE 4 ou 7<>4	**0FFFFH**
1101B XOR 0101B	**1000B**
HIGH('A' SHL 8)	**0041H**
HIGH 'A' SHL 8	**0000H**
'A' OR 'A' SHL 8	**4141H**
NOT 'A' – 1	**0FFBFH**

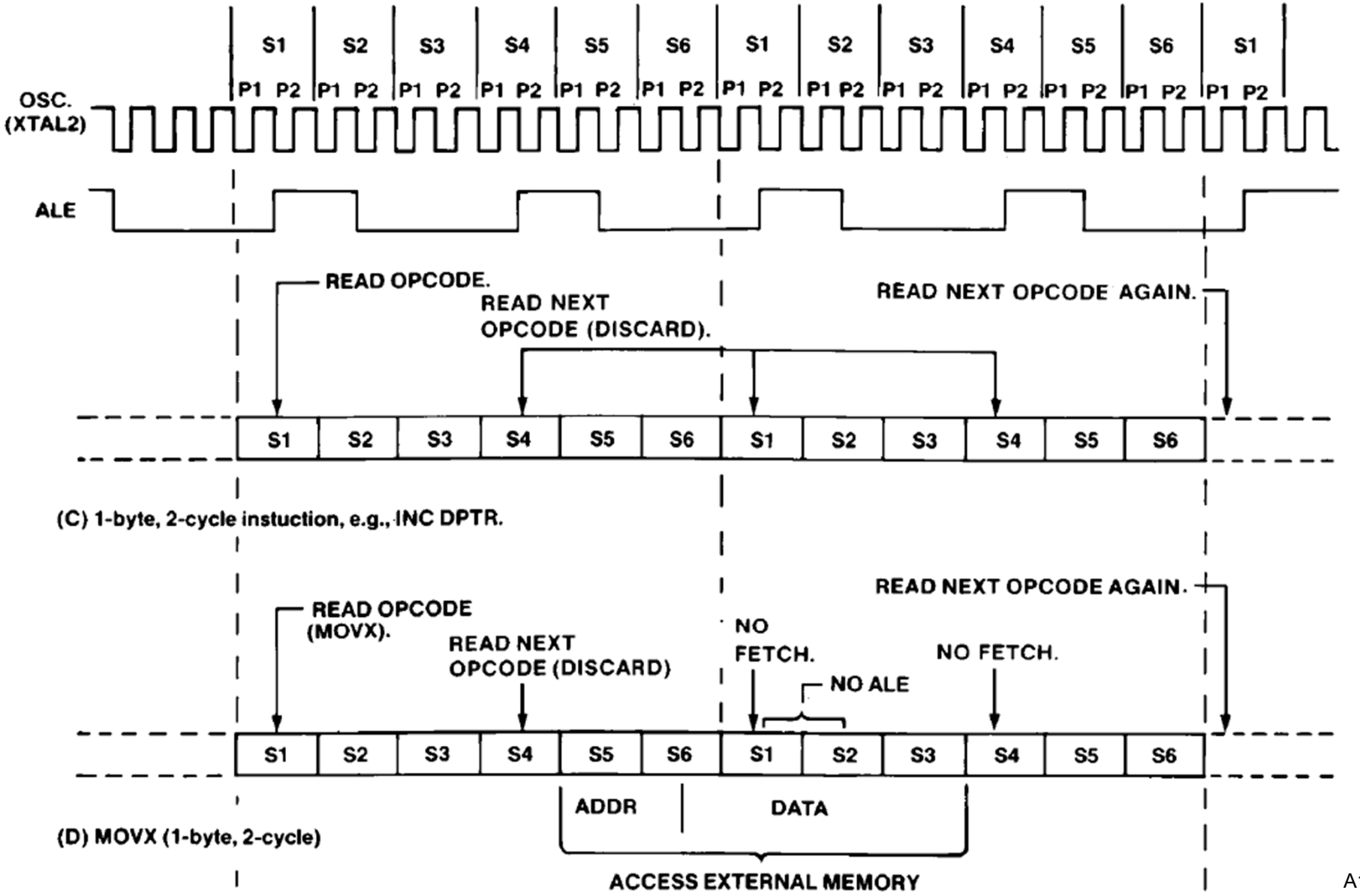
Directivas

Categoria	Directiva	Sintaxe			Função
Controlo do estado	**ORG**		**ORG**	expressão	Especifica um valor para contador de localização do segmento activo
	END		**END**		Indica ao *assembler* o fim do programa fonte
	USING		**USING**	expressão	Indica ao *assembler* o banco de registo usado no código que vem a seguir à directiva. Repare que a comutação do banco de registo deve ser efectuada usando apenas instruções do 8051
Definição de símbolos	**SEGMENT**	Símbolo	**SEGMENT**	tipo_de_segmento	Declara um símbolo como sendo um segmento *relocatable* de um dado tipo. Para começar a usar o segmento, deve-se usar a directiva RSEG
	EQU	Símbolo	**EQU**	expressão	Atribuí um valor a um símbolo
	SET	Símbolo	**SET**	expressão	Igual ao EQU, exceptuando o facto de permitir a redefinição o símbolo
	DATA	Símbolo	**DATA**	expressão	Atribui ao símbolo um endereço directo da RAM interna
	IDATA	Símbolo	**IDATA**	expressão	Atribui um endereço da RAM interna indirectamente endereçável ao símbolo
	XDATA	Símbolo	**XDATA**	expressão	Atribui ao símbolo um endereço da memória externa
	BIT	Símbolo	**BIT**	expressão	Atribuí um endereço directo da área de memória endereçável ao bit a um símbolo
	CODE	Símbolo	**CODE**	expressão	Atribuí um endereço da memória de código ao símbolo

Directivas

Categoria	Directiva	Sintaxe			Função
Incialização e reserva de armazenamento	**DS**	[LABEL:]	**DS**	expressão	Reserva espaços em múltiplos de *bytes*. Não pode ser utilizado com segmento do tipo BIT. O valor da expressão deve ser conhecida pelo *assembler*
	DBIT	[LABEL:]	**DBIT**	expressão	Reserva espaços em múltiplos de bits. O valor da expressão deve ser conhecida pelo *assembler*
	DB/DW	[LABEL:]	**DB/DW**	expressão	Inicializa a memória de código com valores do tipo byte/word
Program linkage	**PUBLIC**		**PUBLIC**	Símbolo [, símbolo] [...]	Define uma lista de símbolos que tornam visíveis e utilizáveis a partir de outros módulos
	EXTRN		**EXTRN**	Tipo_segmento(símbolo [,símbolo] [...], ...)	Informa o *assembler* da lista de símbolos definidos noutros módulos e que vão ser utilizados neste. O tipo de segmento pode ser CODE, DATA, XDATA, IDATA, BIT e um especial designado por NUMBER que especifica um símbolo definido por EQU
	NAME		**NAME**	Nome_do_módulo	
Selecção de Segmentos	**RSEG**		**RSEG**	Nome_do_segmento	Ao encontrar uma directiva de selecção de segmento, o *assembler* direcciona o código
	CSEG		**CSEG**	[**AT** endereço]	ou dado que lhe segue para o segmento
	...		**DSEG**	[**AT** endereço]	seleccionado até que seja seleccionado um
	XSEG		**XSEG**	[**AT** endereço]	outro segmento


S1 S2 S3 S4 S5 S6 S1 S2 S3 S4 S5 S6 S1
P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2
OSC. (XTAL2)
ALE
READ OPCODE.
READ NEXT OPCODE (DISCARD).
READ NEXT OPCODE AGAIN.
S1 S2 S3 S4 S5 S6
(A) 1-byte, 1-cycle instruction, e.g., INC A.
READ OPCODE.
READ 2ND BYTE.
READ NEXT OPCODE.
S1 S2 S3 S4 S5 S6
(B) 2-byte, 1-cycle instruction, e.g., ADD A, #data


S1 S2 S3 S4 S5 S6 S1 S2 S3 S4 S5 S6 S1
P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2
OSC. (XTAL2)
ALE
READ OPCODE.
READ NEXT OPCODE (DISCARD).
READ NEXT OPCODE AGAIN.
S1 S2 S3 S4 S5 S6 S1 S2 S3 S4 S5 S6
(C) 1-byte, 2-cycle instuction, e.g., INC DPTR.
READ OPCODE (MOVX).
READ NEXT OPCODE (DISCARD)
NO FETCH.
NO ALE
NO FETCH.
READ NEXT OPCODE AGAIN.
S1 S2 S3 S4 S5 S6 S1 S2 S3 S4 S5 S6
ADDR
DATA
(D) MOVX (1-byte, 2-cycle)
ACCESS EXTERNAL MEMORY

Rotinas de atraso (delay)

```
DELAY1:  MOV   R7,#X
         DJNZ  R7,$
         RET
```

O nº de ciclos máquina desta rotina é:

NC=1+2*R7+2=3+2*R7

R7=1⇒NC=5 R7=0⇒NC=515

delay=12*NC/(frequência cristal)

f_c=12MHz ⇒ delay=NC µs

```
DELAY2:  MOV   R6,#Y
D2LOOP:  MOV   R7,#X
         DJNZ  R7,$
         DJNZ  R6,D2LOOP
         RET
```

O nº de ciclos máquina desta rotina é:

NC=(3+2*R7)*R6+3

```
DELAY3:  MOV   R5,#Z
D3L1:    MOV   R6,#Y
D3L2:    MOV   R7,#X
         DJNZ  R7,$
         DJNZ  R6,D3L2
         DJNZ  R5,D3L1
         RET
```

O nº de ciclos máquina desta rotina é:

NC=((3+2*R7)*R6+3)*R5+3

Rotinas de atraso (delay)

- ***Delay* por software**:
  - O tempo de espera é estabelecido pelo número de ciclos máquina necessários para executar a rotina de *delay*. O microprocessador fica bloqueado, ou seja, durante a execução do *delay* não pode executar outro código;
- **Difíceis de controlar**:
  - Para além de ocuparem registos, o valor a colocar em cada registo não é "simples" de obter. Muitas vezes opta-se por implementar uma rotina de *delay* fixo (ex:1000µs) e invocá-la várias vezes;
- **Dependem**:
  - Do número de registos e dos seus valores, do nº de ciclos máquina necessários à execução das instruções e do oscilador utilizado.

Rotinas de atraso (delay)

- Suponham que pretendemos gerar uma onda quadrada no pino P1.0. Qual a maior frequência possível e qual o *duty-cycle* dessa onda?

```
ONDA:   SETB    P1.1    ;NC=1⇒1µs
        CLR     P1.1    ;NC=1⇒1µs       T=4µs⇒f=250KHz
        SJMP    ONDA    ;NC=2⇒2µs       D=t_on/T*100=1µs/4µs=25%
```

- Altere o código de modo a garantir um *duty-cycle* de 50%. Qual a frequência da onda quadrada?

- Faça um rotina que permita gerar uma onda quadrada de 20KHz com um *duty-cycle* de 50%.