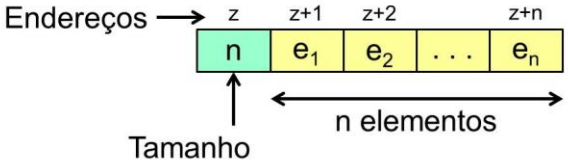
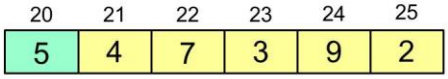
Experimento 1. Programação Assembly

OBJETIVO:
Usar a solução de pequenos programas para desenvolver o entendimento e o uso do assembly do MSP430.

DADOS:
Para os experimentos desta aula, usaremos vetores de números. Para facilitar, vamos adotar a formatação ilustrada abaixo. Note que para indicar o vetor, a única referência necessária é a do endereço de início, pois a primeira posição já indica seu tamanho. Os elementos podem ser bytes, palavras de 16 bits (W16) ou palavras de 32 bits (W32).



Exemplo: vetor[4, 7, 3, 9, 2] no endereço 0x20



Também faremos uso da Tabela ASCII, que está disponível no final deste roteiro.

O exemplo abaixo indica como inicializar a memória de dados com o vetor do exemplo acima. Note que este vetor é composto apenas com bytes.

```
;-------------------------------------------------------------------------
; Segmento de dados inicializados (0x2400)
;-------------------------------------------------------------------------
.data
; Declarar vetor com 5 elementos [4, 7, 3, 9, 2]
vetor:      .byte 0x05, 0x04, 0x07, 0x03, 0x09, 0x02
```

Todos os pedidos são para sub-rotinas. O programa deve ter a seguinte organização:

```
;-------------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------------
;
      mov   #vetor,R5   ;inicializar R5 com o endereço do vetor
      call  #subrot     ;chamar subrotina
      jmp   $           ;travar execução ao retornar da subrotina
;
subrot:     ...
      ...
      ret
```

PEDIDOS:

Programa 1:
Escreva a subrotina **MENOR** que recebe em R5 o endereço de início de um vetor de bytes (sem sinal) e retorna:

- R6 → menor elemento do vetor e
- R7 → qual sua frequência (quantas vezes apareceu)

Para este programa, declare um vetor de bytes formado pela concatenação do código ASCII dos nomes completos de cada membro da equipe. Note que o montador já converte as letras para o código ASCII correspondente, como mostrado abaixo. Use letras maiúsculas, omita os espaços e não use acentos. Preste atenção ao tipo das aspas.

```
.data
; Declarar vetor com 11 elementos [JOAQUIMJOSE]
vetor:      .byte 11,’J’,’O’,’A’,’Q’,’U’,’I’,’M’,’J’,’O’,’S’,’E’
```

Dica 1: Você pode declarar o vetor em várias linhas

Rótulo (col 1)	Instrução (col 13)	Operandos (col 21)	Comentários (col 45-80)
vetor:	.data .byte .byte .byte	 11,'J','O','A' 'Q','U','I','M' 'J','O', 'S','E'	; Início da seção de dados ; Declara o vetor de 11 ; elementos em várias ; linhas

Dica 2: Você pode visualizar a memória do MSP430 a qualquer momento no Code Composer Studio. Basta abrir a janela do navegador de memória : “Windows” → “Show View” → “Memory Browser”. Use a visualização “8-Bit Hex TI-Style” e navegue para o endereço 0x2400 para ver o seu vetor.

Programa 2:
Escreva a subrotina **MAIOR16** que recebe em R5 o endereço de início de um vetor de palavras de 16 bits (words ou W16) sem sinal e retorna:

- R6 → maior elemento do vetor e
- R7 → qual sua frequência (quantas vezes apareceu)

Use o mesmo vetor do Programa 1, mas seu programa irá interpretar cada elemento como sendo composto por 2 bytes (2 letras). Assim, o tamanho do vetor deve cair para a metade. No caso de uma quantidade ímpar de letras, como acontece com o exemplo acima, é preciso um cuidado extra. Eram 11 letras, então reduzimos o tamanho para 6 (não tem sentido usar 5,5) e acrescentamos um zero no final do vetor, para completar a quantidade. Cuidado, o tamanho precisa ser declarado em 2 bytes (16 bits). No exemplo abaixo, o primeiro elemento será 0x4F4A, já que ASCII(O) = 0x4F e ASCII(J) = 0x4A. Note a inversão, pois é "little endian"!

```
.data
; Declarar vetor com 11 elementos [JOAQUIMJOSE]
vetor:      .byte 6,0,'J','O','A','Q','U','I','M','J','O','S','E',0
```

Observação: apenas para ampliar o conhecimento, também seria possível fazer a declaração da forma mostrada abaixo. Para escrever o Programa 2, não use este tipo de declaração!

```
.data
; Declarar vetor com 6 elementos de 16 bits [JOAQUIMJOSE]
vetor:      .word 6,'JO','AQ','UI','MJ','OS','E'
```

DICA: Se você estiver utilizando o visualizador de memória, é recomendável alterar a visualização para "16-Bit Hex - TI Style"

Programa 3:
Escreva a subrotina **M2M4** que recebe em R5 o endereço de início de um vetor de bytes e retorna:

- R6 → quantidade de múltiplos de 2
- R7 → quantidade de múltiplos de 4

De forma semelhante ao Programa 1, declare um vetor de bytes formado pela concatenação do código ASCII dos nomes completos de cada membro da equipe.

Programa 4:
Escreva a subrotina **W16_ASC** que recebe em R6 um número (sem sinal) de 16 bits e escreve a partir do endereço 0x2400 o código ASCII correspondente ao valor de cada nibble (4 bits). Use R5 como ponteiro para escrita na memória. Veja o exemplo abaixo:

Recebe: R6 = 35243 (0x89AB em hexadecimal)
Retorna: 0x2400: 0x38, 0x39, 0x41, 0x42

Inicialize R6 com os 5 primeiros dígitos de seu número de matrícula. Uma forma elegante de se declarar uma constante num programa é usando a diretiva ".set", como mostrado abaixo.

```
MATR        .set  35243
```

```
…
            mov    #MATR,R6
```

Programa 5:
Escreva subrotina **ASC_W16** que faz a operação inversa do programa 4. Recebe R5 apontando para um endereço (0x2400) com quatro códigos ASCII e monta em R6 a palavra de 16 bits correspondente. Inicializar a memória com seus dados do programa anterior.

Note que é necessário testar se os códigos são válidos de acordo com a Tabela ASCII (de 0x30 → 0x39 e de 0x41 → 0x46). Caso tenha sucesso, deve retornar o Carry em 1. Em caso de erro, retornar Carry em zero.

Caso de sucesso.
Recebe em 0x2400: 0x38, 0x39, 0x41, 0x42
Retorna: R6 = 0x89AB e Carry = 1.

Caso de falha.
Recebe em 0x2400: 0x38, 0x3B, 0x41, 0x42
Retorna: R6 = "don't care" e Carry = 0.

```
;-------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------
;
      mov   #MEMO,R5
      call  #ASC_W16    ;chamar subrotina
OK    jc    OK          ;travar execução com sucesso
NOK   jnc   NOK         ; travar execução com falha
;
ASC_W16:
      ...
      Ret

;-------------------------------------------------------------------------
; Segmento de dados inicializados (0x2400)
;-------------------------------------------------------------------------
                  .data
; Declarar 4 caracteres ASCII (0x38, 0x39, 0x41, 0x42)
MEMO:       .byte '8','9','A','B'
```

SUGESTÕES:

- Esboçar um fluxograma para o problema.
- Escreva os programas de forma fracionada. Faz uso de sub-rotinas. Coloque as sub-rotinas logo depois do programa principal.
- Documente as sub-rotinas, é provável que você as use em experimentos futuros.

RELATÓRIO

O relatório é individual, e deve ser entregue impresso (ou feito à mão). Em hipótese alguma será admitida a entrega do relatório de forma eletrônica.

Questão 1 (5 pontos)

Apresente cada programa pedido, comentando as partes mais importantes.

Questão 2 (1 ponto)

Deve ser feita nas listagens apresentadas para a Questão 1. Logo abaixo de cada instrução emulada, usando uma linha de comentários, escreva a instrução que a substitui.

Questão 3 (1 ponto)

Por que os vetores foram criados e manipulados a partir do endereço 0x2400?

Questão 4 (1 ponto)

Proponha uma solução para o programa 4, mas agora usando uma tabela para obter o código ASCII correspondente a um nibble.

Questão 5 (1 pontos)

O programa apresentado para responder a questão 4 é mais ou menos eficiente que seu programa 5? Por que?

TABELA ASCII PADRÃO

	0_	1_	2_	3_	4_	5_	6_	7_
_0	NUL	DLE	SP	0	@	P	`	p
_1	SOH	DC1	!	1	A	Q	a	q
_2	STX	DC2	"	2	B	R	b	r
_3	ETX	DC3	#	3	C	S	c	s
_4	EOT	DC4	$	4	D	T	d	t
_5	ENQ	NAK	%	5	E	U	e	u
_6	ACK	SYN	&	6	F	V	f	v
_7	BEL	ETB	'	7	G	W	g	w
_8	BS	CAN	(	8	H	X	h	x
_9	HT	EM	)	9	I	Y	i	y
_A	LF	SUB	*	:	J	Z	j	z
_B	VT	ESC	+	;	K	[	k	{
_C	FF	FS	,	<	L	\	l	|
_D	CR	GS	-	=	M	]	m	}

_E	SO	RS	.	>	N	^	n	~
_F	SI	UP	/	?	O	_	o	DEL

Exemplos:
símbolo 'A' = código ASCII 41h
símbolo 'B' = código ASCII 42h
símbolo 'a' = código ASCII 61h
símbolo 'z' = código ASCII 7Ah

Hexa	ASCII	Significado
00	NUL	**NUL**L
01	SOH	**S**tart **O**f **H**eading
02	STX	**S**tart of **T**e**X**t
03	ETX	**E**nd of **T**e**X**t
04	EOT	**E**nd **O**f **T**ransmission
05	ENQ	**ENQ**uire
06	ACK	**ACK**nowledge
07	BEL	**BEL**L
08	BS	**B**ack **S**pace
09	HT	**H**orizontal **T**ab
0A	LF	**L**ine **F**eed
0B	VT	**V**ertical **T**ab
0C	FF	**F**orm **F**eed
0D	CR	**C**arriage **R**eturn
0E	SO	**S**hift **O**ut
0F	SI	**S**hift **I**n
7F	DEL	**DEL**ete"

Hexa	ASCII	Significado
10	DLE	**D**ata **L**ink **E**scape
11	DC1	**D**evice **C**ontrol **1**
12	DC2	**D**evice **C**ontrol **2**
13	DC3	**D**evice **C**ontrol **3**
14	DC4	**D**evice **C**ontrol **4**
15	NAK	**N**egative **A**c**K**nowledge
16	SYN	**SYN**cronism idle
17	ETB	**E**nd of **T**ransmission **B**lock
18	CAN	**CAN**cel
19	EM	**E**nd of **M**edium
1A	SUB	**SUB**stitute
1B	ESC	**ESC**ape
1C	FS	**F**ile **S**eparator
1D	GS	**G**roup **S**eparator
1E	RS	**R**ecord **S**eparator
1F	UP	**U**nit **S**eparator
20	SP	**SP**ace