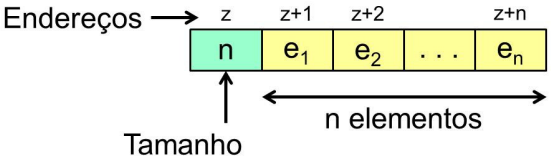
Módulo 1: Programação Assembly

ORIENTAÇÃO:
Este módulo é composto por 15 exercícios e pelo Problema 1. Os exercícios não precisam ser apresentados, ou seja, eles não recebem visto. Servem apenas para que o aluno desenvolva sua habilidade de programação. Apenas o Problema 1 deve receber o visto (nota), que só será validado após o upload de seu programa solução. Serão especificadas as datas limite para o visto e o upload.

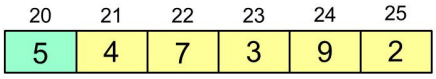
OBJETIVO DESTE MÓDULO:
Usar a solução de pequenos programas para desenvolver o entendimento e o uso do assembly do MSP430.

Introdução: declarando vetores na memória

Para este módulo usaremos vetores de números ou letras. 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). Para o caso de letras, usaremos a Tabela ASCII, disponível no final deste documento.



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



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

IMPORTANTE: A solução sempre será uma sub-rotina, por isso, 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
```

Exercícios:

Exercício 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 (veja tabela no final deste documento) 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,"JOAQUIMJOSE"
```

Dica 1: Você pode declarar o vetor em várias linhas, caracter a caracter

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

Ou colocar tudo junto: `vetor:      .byte 11,"JOAQUIMJOSE"`

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.

Exercício 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 Exercício 1, mas agora 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, "JOAQUIMJOSE",0
```

Observação: apenas para ampliar o conhecimento, também seria possível fazer a declaração da forma mostrada abaixo. Para escrever o programa, 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 3: Se você estiver utilizando o visualizador de memória, é recomendável alterar a visualização para "16-Bit Hex - TI Style"

Exercício 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 Exercício 1, declare um vetor de bytes formado pela concatenação do código ASCII dos nomes completos de cada membro da equipe.

Exercício 4:
Escreva subrotina **EXTREMOS** que recebe em R5 o endereço de início de um vetor com palavras de 16 bits (W16) com sinal e retorna:

- **R6** → menor elemento, **R7** → maior elemento

Para este exercício, vamos formar o vetor usando o número de matrícula e o ano de nascimento de cada membro da equipe. Veja o exemplo para uma equipe com 2 alunos:
Aluno 1: matrícula = 12/1234567 e nasceu em 1990 → 121, 234, 567, -1990.
Aluno 2: matrícula = 11/786745 e nasceu em 1980 (está velho) → 117, 867, 45, -1980

```
            .data
; Declarar vetor com 8 elementos [121, 234, 567, -1990, 117, 867, 45, -1980]
vetor:      .word       8, 121, 234, 567, -1990, 117, 867, 45, -1980
```

Exercício 5:
Escreva a subrotina **SUM16** que armazena a soma (elemento a elemento) de dois vetores de 16 bits de mesmo tamanho.

R5 = 0x2400 → endereço do vetor 1;
R6 = 0x2410 → endereço do vetor 2;
R7 = 0x2420 → endereço do vetor soma.

```
            .data
; Declarar os vetores  com 7 elementos
Vetor1:     .word       7, 65000, 50054, 26472, 53000, 60606,   814, 41121
Vetor2:     .word       7,   226,  3400, 26472,   470,  1020, 44444, 12345
```

Exercício 6:
Escreva a subrotina **SUM_TOT** que armazena o somatório de todos os elementos dos vetores 1 e 2 do exercício anterior. O resultado desta soma precisa de 32 bits e para tanto use a concatenação dos registradores R8 (MSW) e R7 (LSW).
MSW = Most Significant Word e LSW = Least Significant Word

Exercício 7:
Escreva a subrotina **FIB**, que armazena na memória do MSP a partir da posição 0x2400 os primeiros 20 números da sequência de Fibonacci. Use representação de 16 bits sem sinal.

Exercício 8:
Escreva a subrotina **FIB16**, que armazena em R10 o maior número da sequência de Fibonacci a "caber" dentro da representação de 16 bits.

Exercício 9:
Escreva a subrotina **FIB32**, que armazena em R11 (MSW) e R10 (LSW) o maior número da sequência de Fibonacci a "caber" dentro da representação de 32 bits.

Exercício 10:
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 hexadecimal de cada nibble (4 bits). A sugestão é para criar a subrotina **NIB_ASC**, que converte um nibble em ASCII e depois usar essa subrotina 4 vezes. Use R5 como ponteiro para escrita na memória. Veja o exemplo abaixo:

Recebe: R6 = 35243 (0x89AB em hexadecimal)
Retorna em 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
```

Exercício 11:
Escreva subrotina **ASC_W16** que faz a operação inversa do Exercício 5. 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.

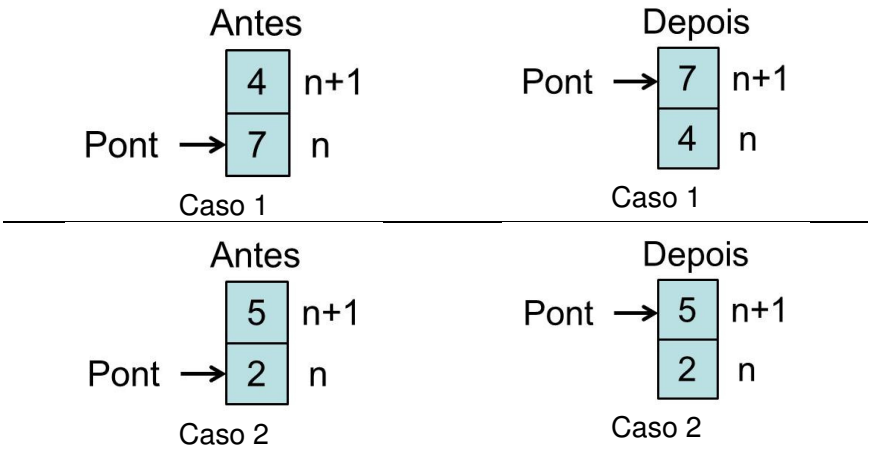
```
;-------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------
;
      mov   #MEMO,R5
      call  #ASC_W16    ;chamar subrotina
OK    jc    OK          ;travar execução com sucesso
NOK   jnc  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

Exercício 12:
Escrever subrotina que ordena, de forma crescente, um vetor. Um método muito conhecido para ordenar os elementos de um vetor é o método da BOLHA. Este método pede uma subrotina auxiliar que vamos chamar de ORD.

A subrotina ORD ordena de forma crescente duas posições de memória: a apontada pelo ponteiro e a seguinte. Veja os exemplos abaixo. No Caso 1, as duas posições foram trocadas. No Caso 2, a ordem já estava correta e nada foi feito. Note que em ambos os casos, o ponteiro terminou apontando para a segunda posição.



Vamos agora considerar um vetor com ***n*** elementos, como mostrado abaixo. Note que o vetor está na vertical. Se varrermos todo este vetor com a subrotina ORD, no topo do vetor deverá estar o maior elemento (Maior 1). Se varrermos novamente o vetor, exceto a última posição, selecionaremos o segundo maior elemento (MAIOR 2). Repetimos esse procedimento ***n-1*** vezes e o vetor será ordenado.

n	y	Maior 1	Maior 1	Maior 1	...	Maior 1
n-1	x	y	Maior 2	Maior 2	...	Maior 2
...	...	...	...	Maior 3	...	Maior 3
3	c	c	c	...	...	...
2	b	b	b	b	...	Menor 2
1	a	a	a	a	...	Menor 1
Seq.	**Original**	**Varrida 1**	**Varrida 2**	**Varrida 3**	**...**	**Varrida n-1**

Exemplo: vetor [4, 7, 3, 5, 1] com 5 elementos, na horizontal.

	Tamanho	Elementos do vetor				
Original	5	4	7	3	5	1
Varrida 1 (4 comparações)	5	4	3	5	1	**7**
Varrida 2 (3 comparações)	5	3	4	1	**5**	7
Varrida 3 (2 comparações)	5	3	1	**4**	5	7
Varrida 4 (1 comparação)	5	1	**3**	4	5	7

Escreva subrotina **ORDENA** que recebe em R5 o endereço de início de um vetor de bytes (sem sinal) e o ordena. Organize os registradores da forma abaixo:

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. Note que usamos a mesma formatação de vetor do Exercício 1.

Atenção: Não use a instrução swpb (swap bytes) pois ela opera em palavras de 16-bits e está sempre alinhada em endereços pares, ou seja, não vai funcionar em endereços ímpares.

O programa deve ter a seguinte organização:

```
;-------------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------------
            mov   #vetor,R5   ;incializar R5 com o endereço do vetor
            call  #ORDENA     ;chamar subrotina
            jmp   $           ;travar execução
            nop               ;exigido pelo montador
;
ORDENA:     ...
            ...
            Ret
            .data
; Declarar vetor com a concatenação dos nomes completos da equipe
vetor:      .byte 11, "JOAQUIMJOSE"
```

Exercício 13:
Neste exercício vamos operar com algarismos romanos. Para relembrar, indicamos o link abaixo:
https://www.somatematica.com.br/fundam/romanos.php

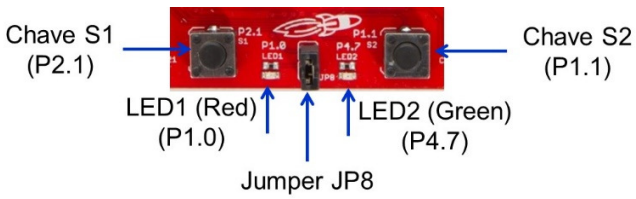
Escreva a subrotina ROM_ARAB, que recebe R6 apontando para um número representado com algarismos romanos (será uma a sequência de letras) e retorna em R5 o número correspondente. Note que o número representado com algarismos romanos é, na verdade, um vetor composto por bytes. Neste exercício, ao invés de usarmos a primeira posição para indicar o tamanho do vetor, vamos apenas marcar seu final com o byte zero(0x00).

Siga a estruturação indicada abaixo.

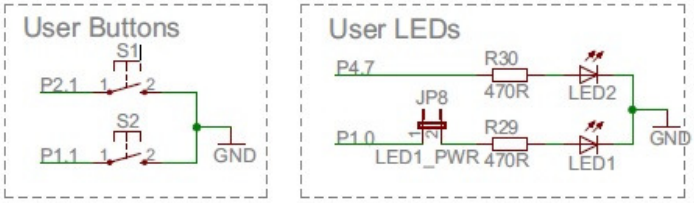
```
;-------------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------------
;
      mov   #NUM_ROM,R6 ;R6 aponta para o início do número
      call  #ROM_ARAB   ;chamar subrotina
      jmp   $           ;travar execução
;
ROM_ARAB:
      ...
      ret

;-------------------------------------------------------------------------------
; Segmento de dados inicializados (0x2400)
;-------------------------------------------------------------------------------
            .data
; Especificar o número romano, terminando com ZERO.
NUM_ROM:    .byte "MMXIX",0         ;2019
```

Utilizando Leds e Chaves:

A placa (Launch Pad) em uso tem duas chaves e dois leds. Veja a localização na figura abaixo.



De acordo com o esquemático abaixo, os leds acendem quando o pino correspondente (P1.0 ou P4.7) é colocado em nível alto. As chaves, quando acionadas são lidas como ZERO. Enquanto estiverem abertas, são lidas como UM.



A execução de uma ação ao acionar uma chave, merece um estudo mais detalhado. A chave está normalmente aberta, ou seja, o pino de I/O correspondente está em nível alto. Ao ser acionada, a chave faz um curto para a terra e com isso esse pino vai para nível baixo. A ação comandada pela chave deve acontecer quando ela passar de aberta para fechada. Uma nova ação só será executada quando a chave, novamente, passar pela transição de aberta para

fechada. Para tanto, a chave precisa deixar o estado de fechada. O fluxograma da Figura 1.a ilustra essa ideia.

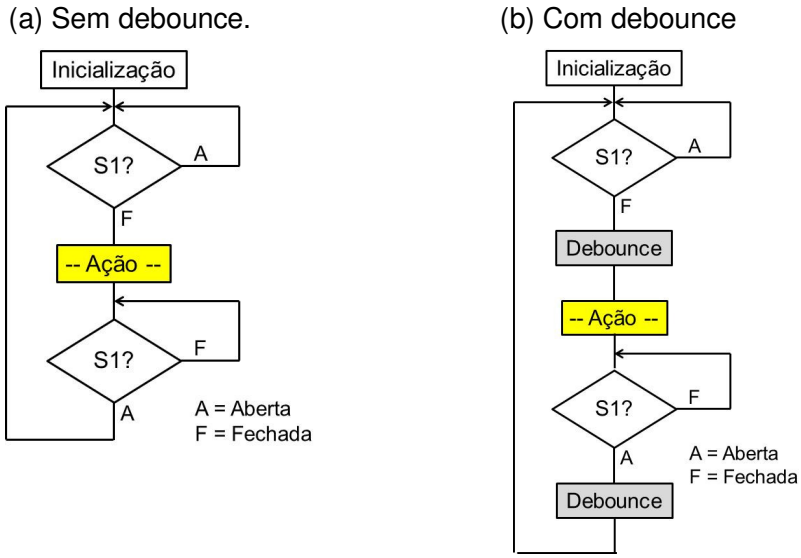


Figura 1. Fluxograma para execução de uma ação mediante o acionamento de uma chave.

As chaves mecânicas possuem contatos metálicos que são ruidosos quando fechados e quando abertos. Isto significa que ao se acionar a chave, o pino correspondente não vai imediatamente para zero (ou para um), mas sim passa por uma situação transitória ruidosa, alternando entre zero e um várias vezes. O resultado disso é que o programa é "enganado" e percebe múltiplos acionamentos. A estes múltiplos acionamentos de uma chave damos o nome de rebote ou bounce. É preciso de uma lógica (debounce) que elimine os bounces. O mais simples é, ao perceber uma transição da chave, esperar um certo tempo para que os bounces decaiam e só então prosseguir com o programa. O fluxograma da Figura 1.b, abaixo ilustra esse conceito. A caixa denominada debounce apenas consome tempo.

É uma boa prática de programação, colocar no início do programa as constantes que serão usadas. Por exemplo, na listagem abaixo, foi declarada a constante DELAY que é usada para especificar a duração do debounce. Toda vez que a palavra DELAY for encontrada, o montador a substitui pelo seu valor (1234). A declaração ".equ" é uma abreviação da palavra "equate".

```
;-----------------------------------------------------------------------------
; Main loop here
;-----------------------------------------------------------------------------
DELAY       .equ        1234
```

É de se notar que os fluxogramas apresentados são pouco eficientes, pois "prendem" o processador esperando a chave fechar ou abrir. Nenhuma outra tarefa pode ser executada pela CPU. Uma situação mais realista é a de quando se precisa monitorar uma chave e executar uma tarefa de tempos em tempos. Isto significa que o processador não pode "ficar preso" esperando pela alteração na chave. A solução para este caso está apresentada na Figura 2. Note que são levados em conta dois parâmetros: o estado atual da chave (S1) e o estado anterior (o passado) da chave (PS1). É preciso explicar a notação:

S1 → chave do launch pad
S1 = A → chave S1 está aberta
S1 = F → chave S1 está fechada

PS1 → estado anterior (passado) da chave S1
PS1 = A → chave S1 estava aberta
PS1 = F → chave S1 estava fechada

As possibilidades são:
S1 = A e PS1 = A → (AA) chave estava aberta e continua aberta;
S1 = A e PS1 = F → (FA) chave estava fechada e foi liberada;
S1 = F e PS1 = A → (AF) chave estava aberta e foi acionada e
S1 = F e PS1 = F → (FF) chave estava fechada e continua fechada;

O fluxograma permite executar uma tarefa repetitiva e monitorar a chave simultaneamente. O programador decide onde ele vai realizar suas ações em função dos estados da chave S1.

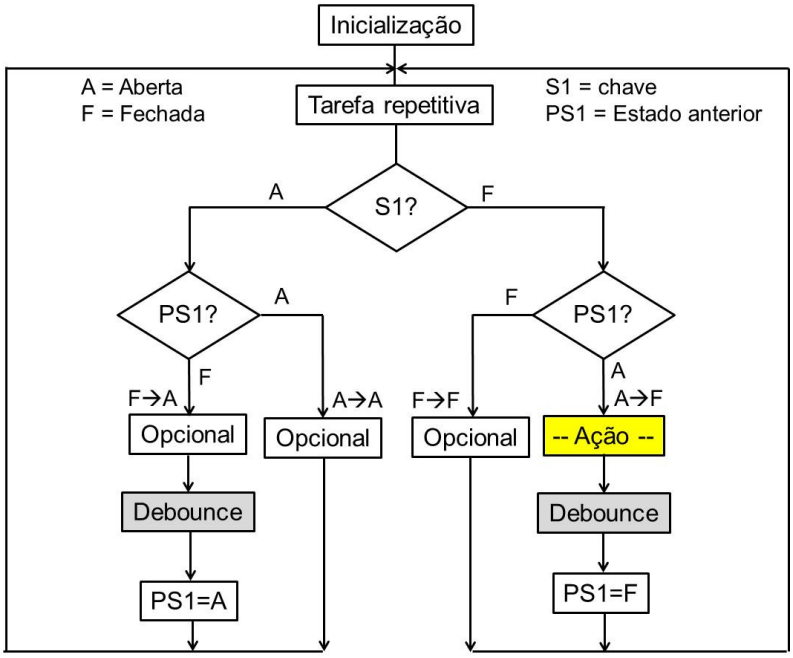
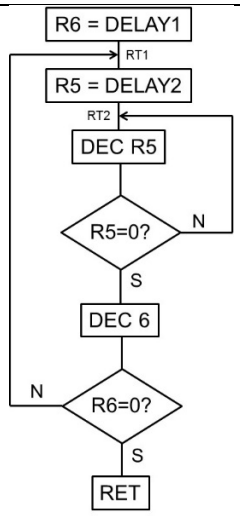


Figura 2. Fluxograma para monitorar uma chave sem "prender a execução".

Exercícios:

Exercício 14:
Faça os leds LED1 e LED2 acenderem alternadamente. Cada Led deve ficar aceso por um intervalo aproximado de 1 segundo. Como ainda não estudamos os temporizadores do MSP, vamos fazer uso da rotina **ROT_ATZ** que gera um atraso perto de 1 segundo. Uma sugestão é apresentada abaixo. Use ensaio e erro para determinar os valores de DELAY 1 e 2.

```
;---------------------
; Main loop here
;---------------------
DELAY1      .equ        12
DELAY2      .equ        34

;;;; seu programa

; Rotina para gerar atraso
ROT_ATZ:    MOV         #DELAY1,R6
RT1:        MOV         #DELAY2,R5
RT2:        DEC         R5
            JNZ         RT2
            DEC         R6
            JNZ         RT1
            RET
```



Note que a sub-rotina ROT_ATZ faz uso de dois contadores aninhados. O atraso será proporcional aos valores DELAY1 e DELAY2. De forma hipotética, vamos supor que cada instrução de desvio consuma 2 µs e as demais 1 µs. Sob estas condições, qual o tempo consumido pela rotina exemplificada (DELAY1 = 12 e DELAY2 = 34)?

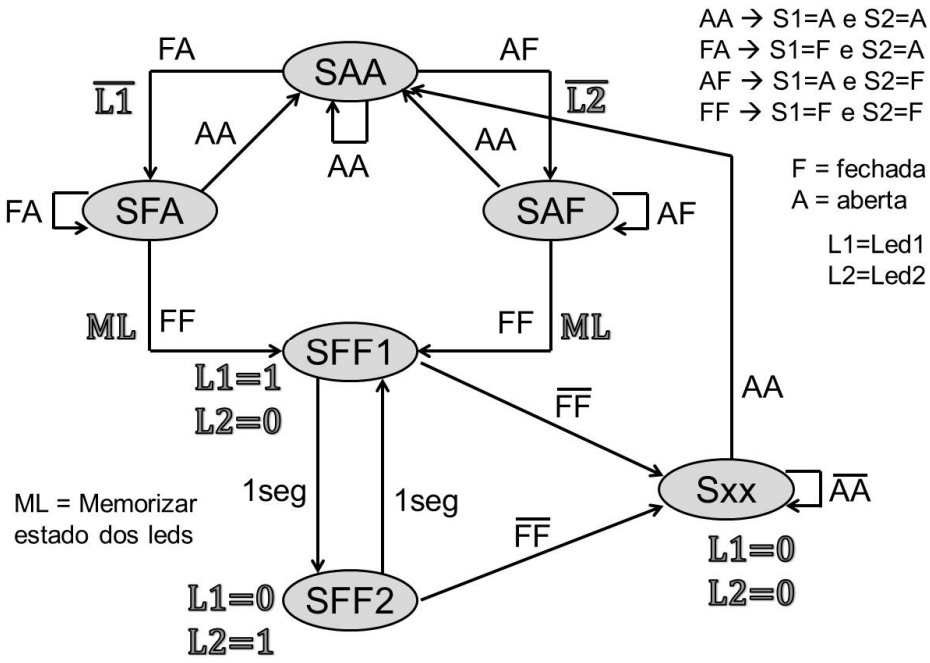
Exercício 15:
Neste exercício, LED1 deve permanecer aceso enquanto a chave S1 estiver pressionada. O mesmo deve acontecer com o LED2 e S2.

Exercício 16:
Escreva a Rotina **CONTA** que constrói um contador binário de 2 bits (0, 1, 2, 3, 0, 1, ...) com os LEDS 1 e 2. Os leds iniciam apagados e apresentam uma contagem a cada acionamento da chave S2. A contagem deve acontecer toda vez em que a chave passar do estado de aberta para fechada (transição: A→F). Um novo acionamento só pode ser aceito depois da chave passar de fechada para aberta (transição: F→A). Será preciso eliminar os rebotes desta chave (SW2), por isso use a rotina **ROT_ATZ**. É claro, determine os novos valores para DELAY1 e 2.

Exercício 17:
Escreva a Rotina **LEDS** que apresenta o seguinte controle para os leds.

- Chave S1: a cada acionamento (A→F), inverte o estado do LED 1;
- Chave S2: a cada acionamento (A→F), inverte o estado do LED 2;
- Enquanto ambas chaves estiverem acionadas (ambas fechadas), os leds piscam de forma complementar, na frequência de 1 Hz. Por complementar se entende um led aceso e o outro apagado. Quando ambas as chaves forem liberadas, é restaurado o estado dos leds, que voltam a obedecer aos controles das chaves S1 e S2.

O diagrama de estados abaixo deixa claro o funcionamento esperado.


AA → S1=A e S2=A
FA → S1=F e S2=A
AF → S1=A e S2=F
FF → S1=F e S2=F
F = fechada
A = aberta
L1=Led1
L2=Led2
SAA
SFA
SAF
SFF1
SFF2
Sxx
FA
AF
L1
L2
AA
AA
AA
FA
AF
ML
FF
FF
ML
L1=1
L2=0
FF
AA
ML = Memorizar
estado dos leds
1seg
1seg
FF
AA
L1=0
L2=0
L1=0
L2=1

PROBLEMA 1: Escrever com Algarismos Romanos

Este problema deve receber o visto (nota), que só será validado após seu upload. Faça o upload de apenas um programa por equipe. O programa solução entregue deve estar completo e pronto para ser "carregado" e executado no Code Composer. Será verificado o correto funcionamento de todo programa. **Caso o programa solução entregue não funcione, a nota será zerada**.

ÉTICA E HONESTIDADE ESTUDANTIL:
Será verificada a similaridade entre os programas entregues e os demais programas, incluindo os dos semestres anteriores. Caso a similaridade entre dois programas seja grande o suficiente para caracterizar a "cola", as equipes estão **automaticamente reprovadas**. Oportunamente serão especificadas as datas limite para o visto e para o upload da solução.

OBJETIVO:
Escrever subrotina que representa com algarismos romanos um número entre 1 e 3999. Para lembrar sobre a numeração com algarismos romanos, é indicado o link abaixo.
https://www.somatematica.com.br/fundam/romanos.php

A resposta será uma sequência de letras armazenadas na memória, obrigatoriamente terminada com o byte zero (0x000), como é usual quando se trabalha em C.

Problema 1 (programa para receber visto):

Apresente subrotina **ALG_ROM** que recebe em R5 um número entre 1 e 3999 e o escreve com algarismos romanos a partir da posição de memória apontada por R6. O fim desse número deve ser indicado como o byte igual a zero (0x00). O programa deve ter a seguinte organização:

```
;-------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------
NUM         .equ  2019        ;Indicar número a ser convertido
;
            mov   #NUM,R5     ;R5 = número a ser convertido
            mov   #RESP,R6    ;R6 = ponteiro para escrever a resposta
            call  #ALG_ROM    ;chamar subrotina
            jmp   $           ;travar execução
            nop               ;exigido pelo montador
;
ALG_ROM:    ...
            ret

            .data
; Local para armazenar a resposta (RESP = 0x2400)
RESP:       .byte       "RRRRRRRRRRRRRRRRR",0
```

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

Códigos especiais usados na Tabela ASCII

Hexa	ASCII	Significado
00	NUL	**NUL**L
01	SOH	**S**tart **O**f **H**eading
02	STX	**S**tart of **TeX**t
03	ETX	**E**nd of **TeX**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 **AcK**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