Sistemas Microprocessados – Prova 1
Prof. Daniel Café – turma A – 2017/1

Nome: _____ Matrícula: _____

Q1a (1)	Q1b (2)	Q2(2)	Q3(2)	Q4a (1,5)	Q4b (1,5)	Total

Questão 1

Convolução

Convolução discreta é uma operação matemática bastante usada em processamento de sinais O cálculo de convolução é tipicamente encontrado em filtros digitais, cálculos de correlação e até identificação de frequência fundamental de sinais ruidosos. A convolução entre dois sinais s_1 e s_2 é dada por:

$$h[k] = \sum_{j=0}^{k} s_1[j] \cdot s_2[k-j]$$

Considere que s1 e s2 são vetores de bytes de tamanho N inicializados nas posições de memória #sig1 e #sig2. Considere também que você já dispõe de uma subrotina *mult* que realiza a multiplicação de dois bytes e retorna uma palavra de 16-bits.

1.a) Desenhe o fluxograma do seu algoritmo. (1 ponto)
1.b) Escreva uma subrotina *conv* que retorne o valor da k-ésima amostra. (2 pontos)

Convenções de *mult*:	Convenções de *conv*
• R13 -> op1 • R14 -> op2 • R15 -> resultado em 16-bits	• R5 -> valor de k • R6 -> endereço do vetor s_1 • R7 -> endereço do vetor s_2 • R8 -> retorno do k-ésima amostra

Questão 2

Multiplicador de hardware

Escreva a subrotina *mult* da questão anterior para operandos com sinal utilizando o hardware dedicado de multiplicação. Use a convenção de passagem de parâmetros pela pilha ao invés da convenção anterior (2 pontos).

Questão 3

Conjunto de instruções

Escreva o código assembly que gera as instruções abaixo. (2 pontos)

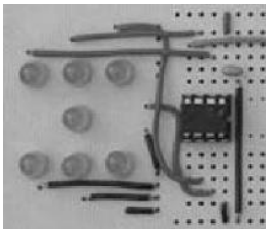
Memória	
0x30	0x55B9
0x32	0xFFFF
0x34	0x8318
0x36	0xD237
0x38	0x3FFB

Para cada instrução:

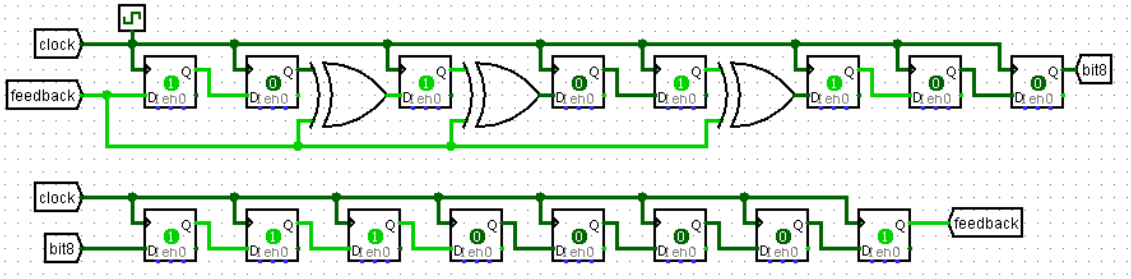
- Dizer qual é o formato da instrução,
- Explicitar o modo de endereçamento do(s) operando(s).
- Mostrar quais registradores e endereços estão sendo usados.

Questão 4

Dado digital



Nesta questão, iremos projetar um dado digital. Suponha que toda vez que apertarmos um botão, o microcontrolador executa uma rotina que gera um número de 1 a 6. Números pseudoaleatórios podem ser gerados de diferentes maneiras. Uma maneira é utilizando registradores de deslocamento com realimentação linear, em inglês: Linear Feedback Shift Registers (LFSR). O exemplo abaixo é um LFSR iniciado com o valor 0xACE1. Cada etapa embaralha os bits de acordo com um polinômio gerador, neste caso 0x3400 que indica a posição das portas XOR. Esse processo gera uma sequência de números aparentemente aleatório. Se rodarmos 3 vezes teremos a sequência [0xACE1, 0x E270, 0x 7138, 0x389C].

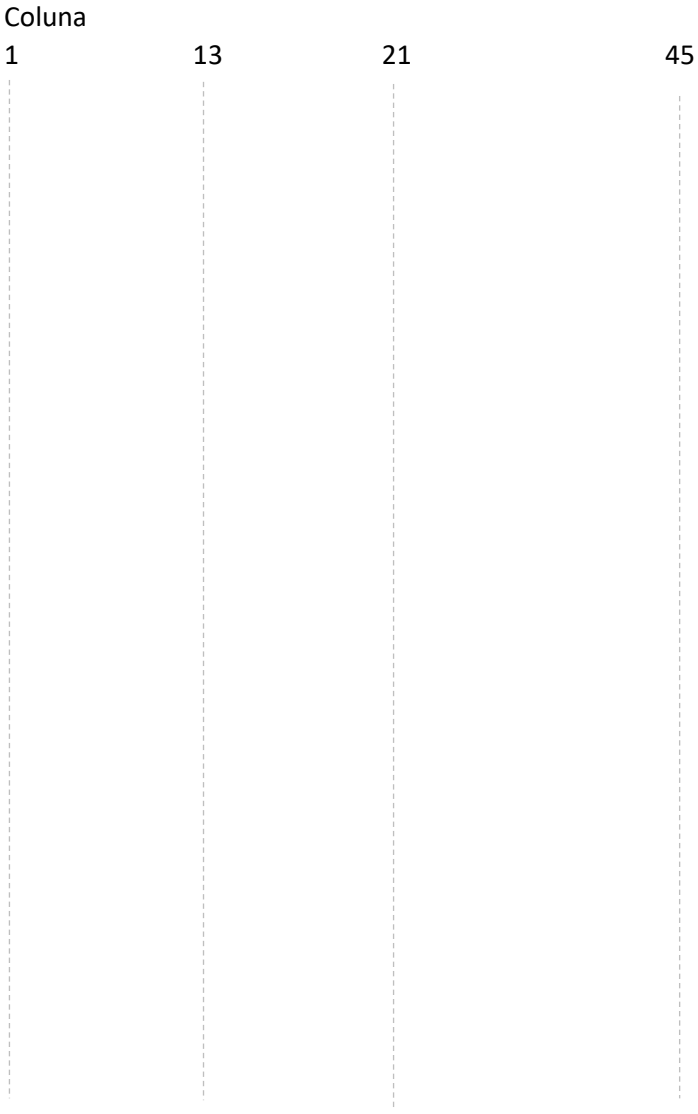


4.a) Escreva uma subrotina *randnum* que gere um número pseudoaleatório com base no princípio ilustrado acima. (1,5 pontos)

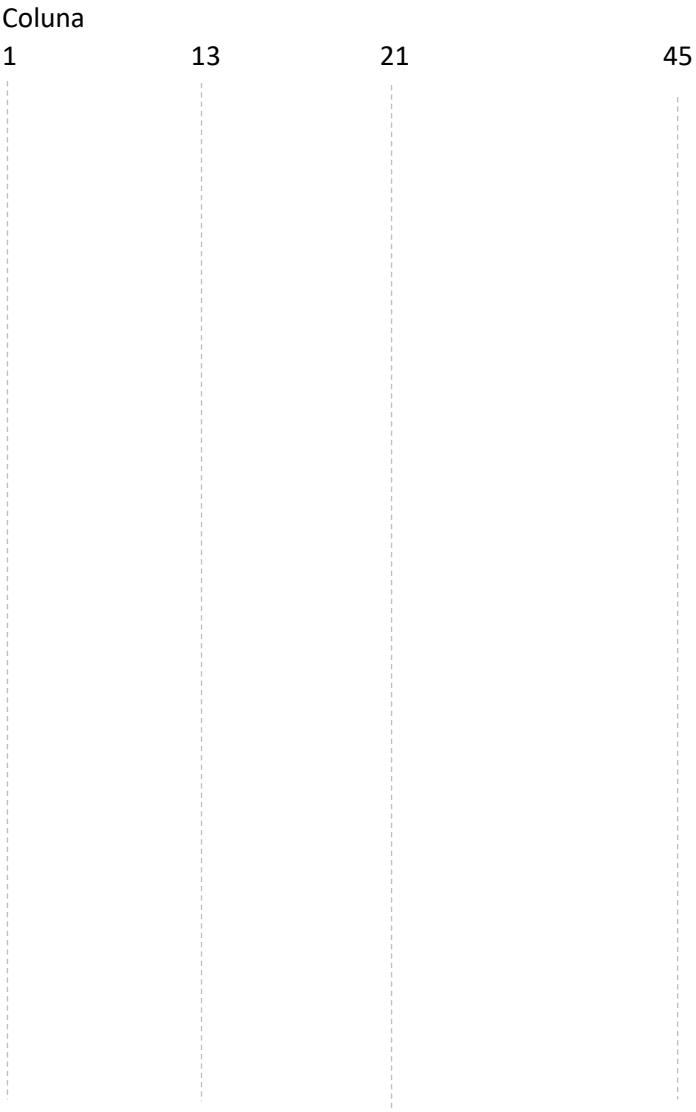
4.b) Extraia o terceiro nibble (0x0F00) do número pseudoaleatório para gerar um número de 1 a 6 que simule o resultado de uma “jogada de dados”. (1,5 pontos)

Solução 1.a)

Solução 1.b)



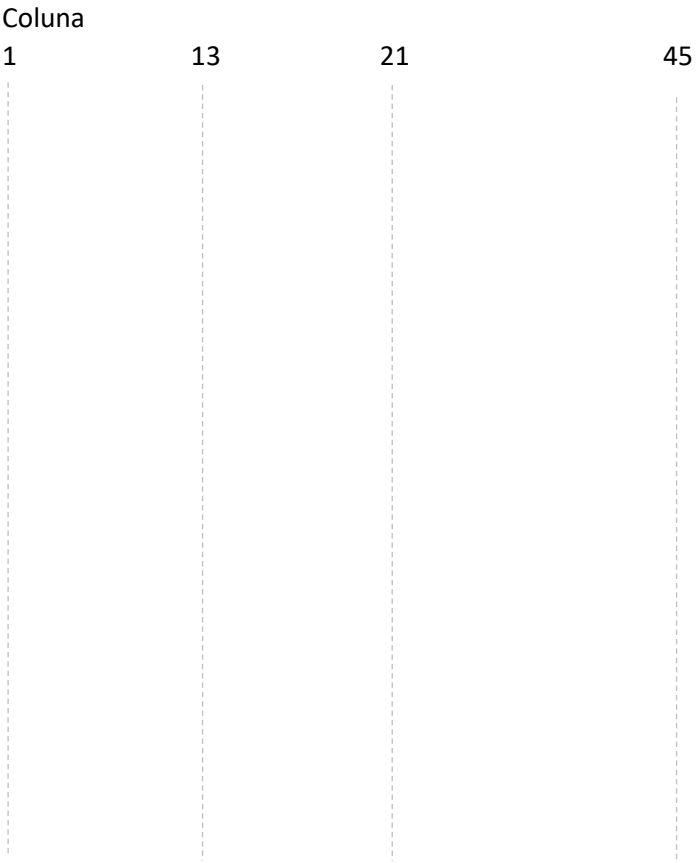
Questão 2



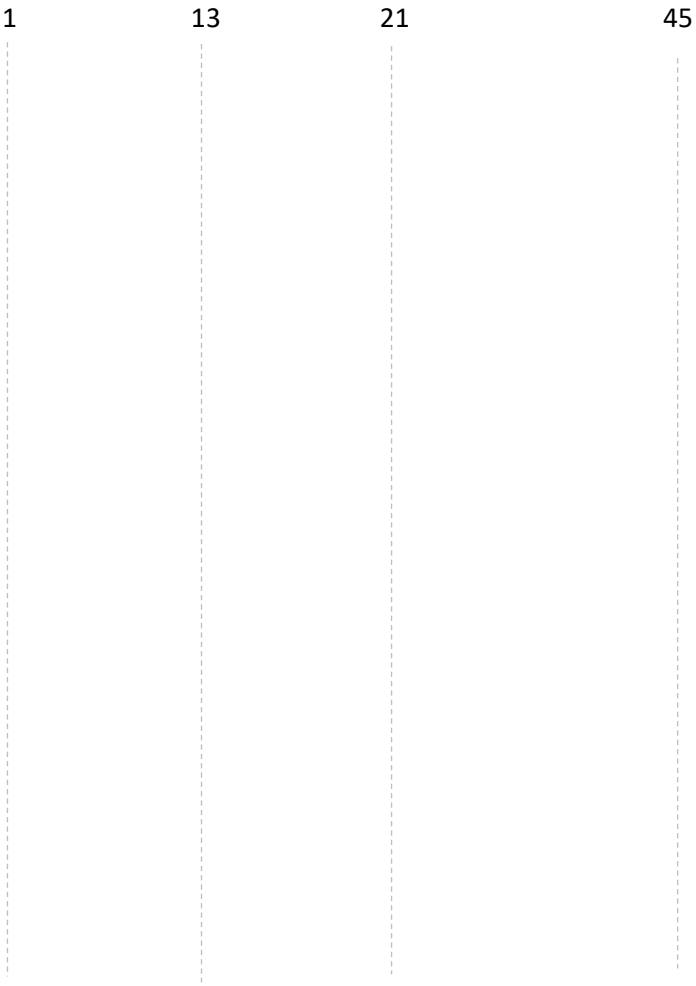
Questão 3

Memória	
0x30	0x55B9
0x32	0xFFFF
0x34	0x8318
0x36	0xD237
0x38	0x3FFB

Solução 4.a)



Solução 4.b)



Espaço em branco para eventual solução complementar

Documento para consulta

Esta folha pode ser consultada durante a prova, porém em hipótese alguma pode ser destacada do conjunto.

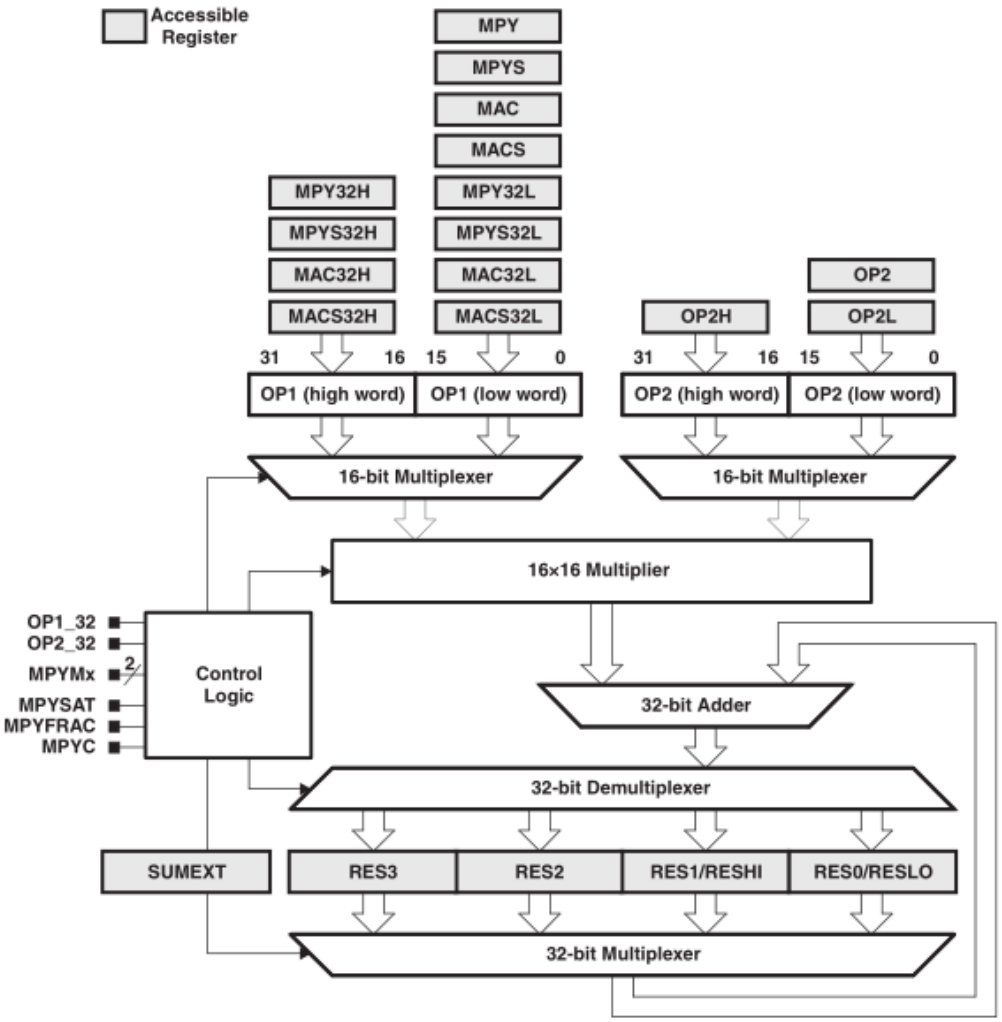
As	Ad	Modo de endereçamnento	Sintaxe	R2 (As)	R3 (As)
00	0	Registro	Rn	SR	0
01	1	Indexado | Simbólico | Absoluto	X(Rn) | ADDR | &ADDR	0	+1
10	-	Indireto	@Rn	+4	+2
11	-	Indireto c/ auto-incremento | Imediato	@Rn+ | #value	+8	-1

Multiplicador dedicado : MPY32CTL0:

15..10	8	9	7	6	5	4	3	2	1	0
xxx	MPY DL32	MPYDLY WRTEN	MPY OP2_32	MPY OP1_32	MPYMx		MPY SAT	MPY FRAC	xx	MPYC

MPYMx :

- 0 – MPY - multiplicação sem sinal,
- 1 – MPYS - multiplicação com sinal,
- 2 – MAC - multiplicação e acumulação sem sinal,
- 3 – MACS - multiplicação e acumulação com sinal.



Conjunto de Instruções

Tipos de instruções

15	14	13	12	11	10	9	8	7	6	5	4	3	0	
0	0	0	[1]	0	0	Opcode			$B/\overline{w}$	Ad		DST		Formato 2
0	0	1	Condition			PC offset (10 bit)								Saltos
Opcode				SRC			Ad		$B/\overline{w}$	As		DST		Formato 1

- Formato 1 – Até 3 palavras (Instrução + Source/Dest offset + Destination offset)
- Formato 2 – Até 2 palavras (Instrução + Destination offset)

Instruções

Instrução (.B/.W)	Args	Op-code	Descrição	Bits de Status			
				V	N	Z	C
Formato1							
MOV	src,dst	0x4	src → dst	-	-	-	-
ADD	src,dst	0x5	src + dst → dst	*	*	*	*
ADDC	src,dst	0x6	src + dst + C → dst	*	*	*	*
SUBC	src,dst	0x7	dst + not(src) + C → dst	*	*	*	*
SUB	src,dst	0x8	dst + not(src) + 1 → dst	*	*	*	*
CMP	src,dst	0x9	dst - src	*	*	*	*
DADD	src,dst	0xA	src + dst + C → dst (decimally)	*	*	*	*
BIT	src,dst	0xB	src .and. dst	0	*	*	$\bar{z}$
BIC	src,dst	0xC	not(src) .and. dst → dst	-	-	-	-
BIS	src,dst	0xD	src .or. dst → dst	-	-	-	-
XOR	src,dst	0xE	src .xor. dst → dst	*	*	*	$\bar{z}$
AND	src,dst	0xF	src .and. dst → dst	0	*	*	$\bar{z}$
Formato 2							
RRC	dst	0x000[1]	C_{n-1} → MSB →.....LSB → C	0	*	*	*
RRA	dst	0x010[1]	MSB → MSB →....LSB → C	0	*	*	*
PUSH	dst	0x120[1]	SP - 2 → SP, src → SP	-	-	-	-
SWPB	dst	0x108[1]	bit 15...bit 8 ↔ bit 7...bit 0	-	-	-	-
CALL	dst	0x128[1]	PC+2→ TOS ; #addr →PC	-	-	-	-
RETI	dst	0x130[1]		*	*	*	*
SXT	dst	0x118[1]	Extend sign bits (B/W/A)	0	*	*	$\bar{z}$
Jumps							
JNE,JNZ	label	000b[2]	Jump if zero is reset	-	-	-	-
JEQ,JZ	label	001b[2]	Jump if zero/equal	-	-	-	-
JNC	label	010b[2]	Jump if carry is reset	-	-	-	-
JC	label	011b[2]	Jump if carry is set	-	-	-	-
JN	label	100b[2]	Jump if negative set	-	-	-	-
JGE	label	101b[2]	Jump if (N xor V) = 0	-	-	-	-
JL	label	110b[2]	Jump if (N xor V) = 1	-	-	-	-
JMP	label	111b[2]	Jump unconditionally	-	-	-	-
Instruções emuladas							
ADDC	dst	ADDC #0,dst	Add carry to dst	*	*	*	*
BR	dst	MOV dst,PC	Branch	-	-	-	-
CLR	dst	MOV #0,dst	Clear dst	-	-	-	-
CLR[C/N/Z]		BIC #[1/4/2],SR	Clear [Carry/Neg/Zero] bit	-	[0]	[0]	[0]
DADC	dst	DADD #0,dst	Add Carry to dst decimally	*	*	*	*
DEC(D)	dst	SUB #[1/2],dst	Decrement by 1 (by 2)	*	*	*	*
[D/E]INT		BI[C/S] #8,SR	[Disable/Enable] Interrupts	-	-	-	-
INC(D)	dst	ADD #[1/2],dst	Increment by 1 (by 2)	*	*	*	*
INV	dst	XOR #–1,dst	Invert DST	*	*	*	*
NOP		MOV R3,R3	No operation	-	-	-	-
POP	dst	MOV @SP+,dst	Pop operand from stack	-	-	-	-
RET		MOV @SP+,PC	Return from subroutine	-	-	-	-
RL[A/C]	dst	ADD(C) dst,dst	C ← MSB← ... ← LSB ← [0/C_{n-1}]	*	*	*	*
SBC	dst	SUBC #0,dst	Subtract carry from dst	*	*	*	*
SET[C/N/Z]		BIS #[1/4/2],SR	Set [Carry/Negative/Zero] bit	-	[1]	[1]	[1]
TST	dst	CMP(.B) #0,dst	Test dst (compare with 0)	0	*	*	1

(1) – O nibble menos significativo representa apenas 1 bit do op-code (o msb)

(2) – No caso de instruções de salto, represento aqui apenas a condição de salto.