Sistemas Eletrônicos Digitais I
TRABALHO 1

v 0.0.1

Abril de 2010

Proposta

Implementar um programa que realize a divisão entre as variáveis inteiras não sinalizadas de 8 bits V1 *e* V2, *guardando o quociente na variável* QUOC *e o resto na variável* RESTO. *Ainda, converter os resultados para BCD, armazenando os novos resultados nas variáveis* QUOC_CENT, QUOC_DEZ, QUOC_UNID *e* RESTO_CENT, RESTO_DEZ e RESTO_UNID.

Para isto, deverá, obrigatoriamente, haver as seguintes subrotinas:

Divide: *realiza a divisão não sinalizada entre dois dados de 8 bits, passados por meio da pilha, sendo o primeiro (em endereço mais alto da memória) o dividendo e o segundo (em endereço menor), o divisor. Ao término da divisão, o quociente do resultado deverá estar no registrador D e o resto no registrador A.*

ConvBinBDC: *realiza a conversão de um dado de 8 bits não sinalizado de binário para BCD (decimal codificado em binário). O valor a ser convertido deverá ser passado por meio do registrador A e o resultado deverá ser armazenado nos registradores A, B e C, cada qual correspondente à unidade, dezena e centena, respectivamente. Esta subrotina deverá chamar a rotina* ***Divide****.*

Com base na proposta, são pedidos:

1. Algoritmo do programa principal e das subrotinas.
2. Programa em Assembly.
3. Código do programa gravado na memória.
4. Simulação para a divisão de 193 por 13.
5. Endereço e conteúdo de toda a pilha logo após cada CALL.

Tabela 1: Endereço das variáveis.

Variável	Endereço (H)
V1	A0
V2	A1
QUOC	B0
RESTO	B1
QUOC_UNID	B2
QUOC_DEZ	B3
QUOC_CENT	B4
RESTO_UNID	B5
RESTO_DEZ	B6
RESTO_CENT	B7

Informações Adicionais

- A memória externa é de 256 x 8.
- O vetor de reset é 00H.
- SP inicializa-se com 00H.
- O endereço das variáveis está na tabela 1.

Algoritmo de Divisão e Exemplo

O algoritmo para realizar a divisão entre dois números binários inteiros não negativos segue:

1. Carregar o dividendo.
2. Carregar o divisor.
3. Inicializar byte do quociente com FFH.
4. Deslocar o divisor à esquerda, acrescentando zeros à direita (*shift* lógico à esquerda), até alinhar seu bit mais significativo com o bit mais significativo do dividendo.
5. Realizar a subtração de divisor alterado do dividendo.
6. Deslocar, via *carry*, o byte do quociente à esquerda.
7. Se *carry* era *1*, somar o divisor alterado ao dividendo novamente.
8. Deslocar o divisor alterado à direita de uma posição, acrescentando zeros à esquerda (*shift* lógico à direita).

9. Repetir os passos de 5 a 8, até que o bit menos significativo do divisor original fique alinhado ao bit menos significativo do dividendo.

10. Inverter os bits do byte do quociente.

Exemplo:

Realizar a operação $\frac{210}{9}$.

$(210)_{10} \Longrightarrow (11010010)_2$

$(9)_{10} \Longrightarrow (1001)_2$

Operação	Díg. Quoc.	Byte Quoc.
1 1 0 1 0 0 1 0		1 1 1 1 1 1 1 1
- 1 0 0 1 0 0 0 0		
0 0 1 0 0 0 0 1 0	$\Longrightarrow$*1*	1 1 1 1 1 1 1 0
- 0 1 0 0 1 0 0 0		
1 1 1 1 1 1 0 1 0	$\Longrightarrow$*0*	1 1 1 1 1 1 0 1
+ 0 1 0 0 1 0 0 0		
1 0 1 0 0 0 0 1 0		
- 0 0 1 0 0 1 0 0		
0 0 0 0 1 1 1 1 0	$\Longrightarrow$*1*	1 1 1 1 1 0 1 0
- 0 0 0 1 0 0 1 0		
0 0 0 0 0 1 1 0 0	$\Longrightarrow$*1*	1 1 1 1 0 1 0 0
- 0 0 0 0 1 0 0 1		
0 0 0 0 0 0 0 1 1	$\Longrightarrow$*1*	1 1 1 0 1 0 0 0

$\overline{\texttt{Byte Quoc}} = (00010111)_2$ *Resposta final: Quoc:* $(10111)_2$ *Resto:* $(11)_2$

Algoritmo de Conversão Binário-BCD e Exemplo

Uma forma de se realizar a conversão de binário para BCD é a seguinte, considerando valor máximo a ser convertido de 255:

1. Dividir o valor por 100. O quociente é o dígito decimal (de 0 a 9) da centena (em binário, um valor de 0000 a 1001).

2. Dividir o resto da operação anterior por 10. O quociente será o dígito da dezena.

3. O resto da operação anterior já naturalmente é o dígito da unidade (equivalente a dividi-lo por 1).

Exemplo:

Encontrar a representação em BCD de 215.

Operação	Quociente	Resto	Cód. BCD do Quoc.
215/100	***2***	15	0010
15/10	***1***	5	0001
5/1	***5***	0	0101

Resposta final: $(0010\ 0001\ 0101)_{BCD}$

Instruções Disponíveis

Mneumônico	Opcode (H)	Descrição
ADC A, B	14	Soma A com B mais *carry*.
ADD A, B	04	Soma A com B.
CMP A, B	3C	Compara A com B (A − B).
DEC C	48	Decrementa 1 de C.
INC C	40	Incrementa 1 a C.
MOV A, B	81	A ← B
MOV B, A	82	B ← A
MOV A, C	83	A ← C
MOV C, A	84	C ← A
MOV A, [*end*]	85	A ← [*end*]
MOV [*end*], A	86	[*end*] ← A
MOV B, [*end*]	87	B ← [*end*]
MOV [*end*], B	88	[*end*] ← B
MOV A, [SP + *desl*]	89	A ← [SP + *desl*]
MOV A, [SP − *desl*]	8A	A ← [SP − *desl*]
NOT A	F6	Negação (complemento de 1).
OR A, B	0C	A ← A OU B
POP A | B | C | D	58 | 59 | 5A | 5B	Tira byte da pilha pondo-o em A | B | C | D.
PUSH A | B | C | D	50 | 51 | 52 | 53	Põe um byte na pilha de A | B | C | D.
RCL B | D	D1 | E1	Rotação de B | D à esquerda via *carry*.
RCR B | D	D2 | E2	Rotação de B | D à direita via *carry*.
ROL B | D	D3 | E3	Rotação de B | D à esquerda, com cópia a *carry*.
ROR B | D	D4 | E4	Rotação de B | D à direita, com cópia a *carry*.
SAL B | D	D5 | E5	Deslocamento aritmético de B | D à esquerda.
SAR B | D	D6 | E6	Deslocamento aritmético de B | D à direita.
SHL B | D	D7 | E7	Deslocamento lógico de B | D à esquerda.
SHR B | D	D8 | E8	Deslocamento lógico de B | D à direita.
SBB A, B	1C	Subtrai B e *carry* de A.
SUB A, B	2C	Subtrai B de A.
XOR A, B	34	A ←A OU-EXCL B
CALL end	9A	Chama subrotina.
RET	CB	Retorna de subrotina.
JMP end	EA	Pula para endereço.
JC	70	Pula se *carry* = 1.
JNC	71	Pula se *carry* = 0.
JZ	72	Pula se *zero* = 1 (*flag de zero*).
JNZ	73	Pula se *zero* = 0 (*flag de zero*).