Tutorial Assembly Parte 2: O renascer de um Tutorial! ☺
By Mr. Maggots29A - Data: 17/12/2005

Nota do Autor:

É pessoal depois de muita demora em lançar novos tutoriais, por causa de problemas pessoais, principalmente no trabalho (Sendo este que resolvi larga-lo para dar continuidade aos meus estudos e a este tutorial). Estou de volta para tentar terminar o que comecei. Devo lembrar que isso tudo aconteceu depois dos e-mails e mensagens no MSN de pessoas pedindo a continuação do mesmo. Então fica este sendo meu presente de natal para você.... haha dingo bell, dingo bell, dingo Mel Lisboa! O.o

O Show tem que continuar...

Bem o problema de ficar tanto tempo fora, é que as vezes você pode chegar em casa e encontrar as coisas meio desarrumadas ou fora do lugar, então é momento de se ambientar e colocar as coisas no lugar.

A primeira pergunta é onde paramos?

A segunda pergunta é para onde vamos?

Bem a primeira pergunta é fácil de responder, pois paramos naquela micro pontinha do Iceberg, e estamos dando os primeiros passos, ainda estamos vendo arquivos .COM, onde é preciso organizar o inicio da memória em 100h Bytes (Hexadecimal), ou 256 Bytes (decimal). E não precisamos nos preocupar com Segmento e Offset de memória por enquanto, mas é por enquanto hein. ☺

Já a segunda pergunta só o tempo dirá! ☺

Falando em tempo, vamos relembrar o que você viu no primeiro tutorial

- - O PROCESSADOR
- - NÚMEROS BINÁRIOS:
- - BASE 10 / DECIMAL:
- - BASE 2 / BINÁRIO:
- - BASE 16 / HEXADECIMAL:
- - DADOS: BIT, NIBBLE, BYTE, WORD, DOUBLE WORD
- - FAIXA DE ALCANCE:
- - COMPLEMENTO DE 2:
- - SISTEMAS DE ARMAZENAMENTO:
- - TAMANHOS DE MEMÓRIA
- - ARMAZENAMENTO DE DADOS:
- - DEBUG:
- - REGISTRADORES:
- - INTERRUPÇÃO:
- - MOV = MOVER OU COPIAR?
- - INT 21:
- - FASMW:

Além é claro de dois mini programas para entender o funcionamento do montador e se familiarizar com o propósito da linguagem.

Vamos continuar nessa balada para que você consiga fixar tudo aquilo que você já viu e entender melhor aquilo que está por vir.

Vamos ao Loop...

Loop em inglês ou Laço como é conhecido por aqui, tem a propriedade da repetição, provavelmente quanto mais complexo seu programa for maior número de Loops ele terá!

Vamos a um exemplo de um algoritmo que mostra a tabuada de 2 (SEM LOOP);

Inicio Do Programa;
x=0;
Imprime (x=(x+1)*2);
Imprime (x=(x+1)*2);
Imprime (x=(x+1)*2);
.
.
.
Fim Do Programa;

Note como seria arcaico fazer um programa tendo q re-escrever uma função várias vezes?
Agora imagine ter que repetir um Imprime de 0 a 1000? ☹

Por isso existe o Loop/Laço, veja como ficaria a tabuada de 2 com o Loop:

Inicio Do Programa;
x=0;
Enquanto(x for menor ou igual a 10){
Imprime (x=(x+1)*2);
}
Fim Do Programa;

Em primeiro lugar setamos **x = 0**, após isso damos inicio ao nosso Loop, com a palavra chave '**Enquanto**', e dentro de Enquanto temos uma condição: (x for menor ou igual 10) enquanto essa condição não for satisfeita o Loop continuará repetindo. Para satisfazer esse Loop, o valor de **x** precisa ser maior que **10**. Caso a condição não seja satisfeita, a próxima instrução **Imprime (x=(x+1)*2);** será executado, e assim será até **x** ser maior que **10**, e quando isso ocorrer o Loop Enquanto é terminado.

OBS: Outra coisa que você precisa saber sobre um Loop, é que você pode muito bem por maneira consciente ou não fazer um Loop infinito. O Loop consciente é o quando o programador coloca uma instrução que nunca será satisfeita, o inconsciente é quando por erro de calculo ou por uso de números sinalizados ou não-sinalizados a condição não é satisfeita. Em qualquer um dos casos você como um bom programador deve saber que tudo que tem um começo tem um fim.

Chega de conversa, vamos ao nosso primeiro programa com o FASMW:

```
org 100h
        mov ah,02h
        mov dl,'0'
PRINCIPAL:
        int 21h
        cmp dl,'9'
        jnz INCREMENTA
        ret
INCREMENTA:
        inc dl
        jmp PRINCIPAL
```

Parece confuso? Isso acontece principalmente por que você nunca viu cmp, jne, inc, jmp! Mas o resultado do programa acima é simplesmente imprimir números de ‘0’ a ‘9’.Vamos olhar instrução sobre instrução:

Org 100h

Começamos nosso código com montando um endereço .COM

Mov ah,02h

É usado como parâmetro para int 21h saber que será um impresso na tela do computador um caractere.

mov dl,'0'

Em dl está o caractere a ser impresso, nesse caso ‘0’ = zero!

PRINCIPAL:

Você quer saber se isso faz parte da linguagem Assembly???
A Resposta pode ser um: Não ou Sim! ☺
Tanto essa “PRINCIPAL:” quanto “INCREMENTA:” são conhecidos como Label ou Etiqueta, elas são usadas para identificar um endereço da memória, como você não sabe em que ponto da memória esta uma determinada função, você pode usar uma Label como uma Etiqueta da memória, ou seja na hora de montar seu programa essa Label/Etiqueta será trocada por um valor da memória, o seu uso fará mais sentido logo adiante.

Não há restrições quanto escrita de uma Label, contando que ela não comece com um número, em todo caso leia o manual.

int 21h

É aqui que o programa começa a funcionar, se você carregar o programa até aqui acontecerá o seguinte:
int 21h lê o conteúdo de ax, sendo assim temos ah setado em 02h, isso diz a interrupção que você irá trabalhar com caracteres. O caractere por sua vez encontra-se em dl que é igual a ‘0’(zero).

Até aqui o programa imprimiu ‘0’(zero) na tela do computador, mas ainda faltam 9 números, agora vamos conhecer o CMP!

cmp dl,'9'

CMP é o mnemônico de COMPARE = COMPARAR. É como se fosse um “if” em linguagem C!
É nele que iniciamos nosso LOOP/LAÇO. No caso da instrução acima, fazemos uma comparação entre o registrador ‘dl’ e o valor ‘9’. O resultado dessa comparação resulta na sinalização dos REGISTRADOR FLAGs.

REGISTRADOR DE FLAGS DO PROCESSADOR:

Fora os registradores que você já viu, existem alguns registradores especiais do processador que são muita importantes, entres eles esta o Registrador de Flags, a única expressão em cada sinalização é um VERDADEIRO ou FALSO, sendo assim são Flags de apenas 1 bit.

Momentos em que as Flags são Sinalizadas:

ZF	**ZERO FLAG**	Quando o resultado de uma operação for 0
OF	**OVERFLOW FLAG**	Quando ocorre um Estouro/Overflow
IF	**INTERRUPT FLAG**	Quando começa uma Interrupção
CF	**CARRY FLASG**	Quando acontece famoso vai ‘1’ em uma adição ou subtração

Isso são algumas Flags iniciais, mais para frente veremos outras!
Aliando a sinalização de cada Flag aos JUMPS/SALTOS condicionais você tem um loop/laço.

JA	**JUMP IF ABOVE**	salte se acima
JAE	**JUMP IF ABOVE OR EQUAL**	salte se acima ou igual
JB	**JUMP IF BELOW**	salte se abaixo
JBE	**JUMP IF BELOW OR EQUAL**	salte se abaixo ou igual
JE	**JUMP IF EQUAL**	salte se igual
JG	**JUMP IF GREATER**	salte se maior
JGE	**JUMP IF GREATER OR EQUAL**	salte se maior ou igual
JL	**JUMP IF LESS**	salte se menor
JLE	**JUMP IF LESS OR EQUAL**	salte se menor ou igual
JNA	**JUMP IF NOT ABOVE**	salte se não acima
JNAE	**JUMP IF NOT ABOVE OR EQUAL**	salte se não acima ou igual
JNB	**JUMP IF NOT BELOW**	salte se não abaixo
JNBE	**JUMP IF NOT BELOW OR EQUAL**	salte se não abaixo ou igual
JNE	**JUMP IF NOT EQUAL)**	salte se não igual
JNG	**JUMP IF NOT GREATER**	salte se não maior
JNGE	**JUMP IF NOT GREATER OR EQUAL**	salte se não maior ou igual
JNL	**JUMP IF NOT LESS**	salte se não menor
JNLE	**JUMP IF NOT LESS OR EQUAL**	salte se não menor ou igual
JNZ	**JUMP IF NOT SERO**	salte se não zero
JZ	**JUMP IF ZERO**	salte se zero

Você também pode fazer um JUMP/SALTO incondicional sem depender das sinalizações das FLAGs com:

JMP	**JUMP**	Salto = Salto Incondicional

OK? Mas faça me um favor, não tente guardar todos esses JUMPS OU FLAGS nesse exato momento, pois isso vem com o tempo o importante é você saber que uma coisa depende da outra! Continuando:

cmp dl,'9'

Teremos a comparação entre o registrador 'dl' e o número '9' o resultado dessa comparação estará na FLAG-ZERO, nesse caso quando setada em 1 = TRUE/VERDADEIRO o Loop/Laço continuará repetindo.

Em seguida temos:

jne INCREMENTA

jnz = JUMP if NOT ZERO / SALTA se NÃO é ZERO:

Enquanto o resultado da comparação entre o "dl" e o número "9" não for satisfeita o JNZ salta para a LABEL INCREMENTA, que nada mais é que uma posição na memória.

OBS: Note no código que após a instrução **jnz INCREMENTA** temos:

```
    ret
INCREMENTA:
    inc dl
    jmp PRINCIPAL
```

O **ret** não será processado por que **jnz INCREMENTA** saltou para a Label **INCREMENTA:** e ela possui duas instruções **inc dl** e **jmp PRINCIPAL**

INC = INCREMENT / INCREMENTA Adiciona 1 ao destino, nesse caso **'dl'**

Logo em seguida temos um salto incondicional:

jmp PRINCIPAL

Aqui não importa qual o estado da FLAG o salto a LABEL ou uma posição na memória é executado e pronto.

O programa está praticamente pronto, vamos revisa-lo:

```
org 100h                ; Montamos o endereço .COM;
        mov ah,02h      ; Movemos/Copiamos o valor 02h para o registrador ah;
        mov dl,'0'      ; Movemos/Copiamos o valor 02h para o registrador dl;
PRINCIPAL:              ; Label apenas para nos posicionarmos no Loop/Laço;
        int 21h         ; Interrupção 21h, lê o ah e imprime o caracter que está em dl;
        cmp dl,'9'      ; Compara dl,'9' salvando a informação na FLAG Zero-Flag;
        jnz INCREMENTA  ; Salta para Label INCREMENTA dl não for igual a '9';
        ret             ; Sai do Programa, só será executado com dl for igual a '9';
INCREMENTA:             ; Label apenas para nos posicionarmos no Loop/Laço;
        inc dl          ; Incrementa dl;
        jmp PRINCIPAL   ; Salto incondicional a Label PRINCIPAL;
```

Ai está o programa que imprime números de '0' a '9', mas ainda tem mais, vou refazer o mesmo programa, mas agora usando uma maneira diferente e otimizada:

```
org 100h
        mov ah,02h
        mov dl,'0'
PRINCIPAL:
        int 21h
        inc dl
        cmp dl,'9'
        jbe PRINCIPAL
        ret
```

Note que estruturei o programa dentro de uma Label, e isso só foi possível por que mudei o tipo de JUMP/SALTO que agora é **'jbe PRINCIPAL'** e o '**inc dl'** passou para dentro da "estrutura" **'PRINCIPAL:'** (Que não é uma estrutura de verdade, mas trate-a como se fosse), e logo após o **'int 21h'**. E a razão para '**inc dl'** estar depois da **'int 21h'** e antes da **cmp 'destino','fonte'**, é que toda operação aritmética como uma soma ou subtração por exemplo, pode influenciar no estado da FLAG, e isso pode causar erro no Loop.
Como é o caso se você colocar um **inc 'destino'** após o **cmp 'destino','fonte'**. Então:

CERTO:	**ERRADO:**
inc 'destino' cmp 'destino','fonte' jmp label	cmp 'destino','fonte' inc 'destino' jmp label

Mais uma vez tenha muito cuidado com a estrutura do seu programa e principalmente com as FLAGS e com o retorno de cada instrução.

Antes de continuarmos para o próximo programa vamos rever o que é a TABELA ASCII:

ASCII - Stands For American Standard Code For Information Interchange.

ASCII É uma tabela de códigos que define todos Caracteres como o alfabeto, números e caracteres especiais como ´@´,´#´,´$´,´%´.

Para definir os caracteres usasse uma combinação numérica de 7 ou 8 bits.
Com 7 bits podemos definir os caracteres na faixa de 0 a 127 combinações. E com 8 bits podemos definir os caracteres na faixa de 0 a 255. Mas para entender melhor veja a tabela abaixo retirada do site: www.lookuptables.com :

Tabela Padrão 7 bits de 0 a 127:

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Tabela Estendida 8 bits de 0 a 255:

128	Ç	144	É	161	í	177	▒	193	┴	209	╤	225	ß	241	±
129	ü	145	æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
130	é	146	Æ	163	ú	179	│	195	├	211	╙	227	π	243	≤
131	â	147	ô	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
132	ä	148	ö	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
133	à	149	ò	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
134	å	150	û	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
135	ç	151	ù	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
136	ê	152	_	169	_	185	╣	201	╔	217	┘	233	Θ	249	·
137	ë	153	Ö	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
138	è	154	Ü	171	½	187	╗	203	╦	219	▉	235	δ	251	√
139	ï	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	_
140	î	157	¥	173	¡	189	╜	205	═	221	▌	237	ϕ	253	²
141	ì	158	_	174	«	190	╛	206	╬	222	▐	238	ε	254	■
142	Ä	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	
143	Å	160	á	176	░	192	└	208	╨	224	α	240	≡		

Source: www.LookupTables.com

Não podemos ficar parados, agora é uma boa hora para vermos mais um novo programa, desta vez sua propriedade é imprimir o alfabeto maiúsculo de '**A**' a '**W**'!

Vamos ao código:

A funcionalidade desse programa é semelhante ao programa que imprime números de ´0´ a ´9´, mas dessa vez usei letras para representar o Alfabeto:

```
org 100h

        mov dl,41h            ; Movendo 41h = ´A´ na tabela ASCII;
        mov ah,02h            ; Movendo 02h = caracter;
MAIN_PRINCIPAL:               ; Label com o nome MAIN_PRINCIPAL;
        int 21h               ; Interrupção 21h lê ah = 02h imprime conteúdo de dl;
        inc dl                ; Incrementa dl;
        cmp dl,57h            ; Compara dl com 57h = ´W´ na tabela ASCII;
        jbe MAIN_PRINCIPAL    ; Salta se dl não for igual a 57h = ´W´;
        ret                   ; Sai do programa;
```

Observação: Olhando a Tabela ASCII você notará que a combinação numérica entre 'A' e 'W' é sempre crescente, ou seja se o caractere 'A' eqüivale a 41h, 'B' será 42h, 'C' será 43h, até chegarmos ao 'W' que é igual a 57h! Por isso o Inc dl funciona tão bem.

Sem muitas delongas com o programa acima por que ele é muito simples, vamos partir agora para o PILHA, PUSH e POP, e de quebra alterar o código acima, para que cada letra do alfabeto saia em uma nova_linha.

A PILHA / STACK:

A Pilha que descreverei aqui não é aquilo que você usa em seu rádio, brinquedo (se você ainda for uma criança) ou no controle remoto de sua TV para mante-los funcionando. Ou seja essa não é alcalina! ☺

O melhor modo de descrever uma **Pilha** é dizendo que ela se parece com uma ´**PILHA DE CAIXAS**´! É um lugar onde você pode depositar coisas, e lógico que essas coisas não são caixas, mas sim dados(de maneira temporária)!

Os Registradores SP:SS contem a localização da Pilha, sendo que SS (Stack Segment) aponta para o inicio da Pilha e SP (Stack Pointer) a atual posição em que se encontra dentro da Pilha!

Como uma ´PILHA DE CAIXAS´ o último item a entrar é o primeiro a sair, e o primeiro item a entrar na Pilha será o último a sair, sendo assim é um sistema FILO (FIRST IN LAST OUT)! Vamos ver no gráfico.

	ENDEREÇO	**PILHA**
SP	108	ITEM Nº 5
	106	ITEM Nº 4
	104	ITEM Nº 3
	102	ITEM Nº 2
SS	100	ITEM Nº 1

SS: É o começo da Pilha.
SP: É o local vazio da Pilha onde poderá colocar um dado.

Sendo assim o ´**ITEM N.º 5**´ será o primeiro item a sair da nossa Pilha, enquanto o ´**ITEM N.º 1**´ só sairá quando todos os posteriores estiverem fora!

Entretanto como nosso programa é .COM você não precisa fazer nada quanto a Pilha!

Agora que você já conhece um pouco a Pilha, vou explicar como você irá **INSERIR** e **TIRAR** o dados de dentro dela usando dois mnemônicos PUSH e POP!

PUSH e POP

Não PUSH e POP não são aqueles pirulitos que foram febre anos atras, eles são mnemônicos em Assembly, e seu uso é muito simples. Veja:

PUSH – Com o **PUSH** você **INSERE** um ´**DADO**´ para dentro da **Pilha**.
Esse **DADO** pode vir de **REGISTRADOR**, **SEGMENTO**,**MEMÓRIA**.
Após o PUSH a Pilha será decrementada pelo *Tamanho do DADO*.

POP – Com o POP você TIRA o ´DADO´ de dentro da Pilha.
Após o **POP** a **Pilha** será incrementada pelo *Tamanho do DADO*.

*ATENÇÃO: Sempre quando você por **INSERIR** (**PUSH**) ou **TIRAR** (**POP**) dados da pilha deverá faze-lo considerando como WORD (16 bits), sendo assim você não pode fazer nem PUSH AL ou POP AL, você deverá fazer PUSH AX ou POP AX, mesmo que seu interesse seja preservar apenas 1 byte!*

Vamos a mais um exemplo de um programa em Assembly, agora irei imprimir as letras do alfabeto, sendo um caractere por linha, e com isso mostrar o uso do PUSH e POP.

```
org 100h                    ; Montando um endereço .COM;
        mov dl,41h          ; Movendo 41h = ´A´ na tabela ASCII;
        mov ah,02h          ; Movendo 02h = caracter;
MAIN_LOOP:
        int 21h             ; Interrupção 21h lê ´ah=2´ imprime conteúdo de dl;
        cmp dl,57h          ; Compara dl com 57h = ´W´ na tabela ASCII;
        jnz NOVA_LINHA      ; Salta para Label NOVA_LINHA: se CMP não for satisfeita;
        ret                 ; Sai do programa;
NOVA_LINHA:                 ; Declaração da Label NOVA_LINHA;
        push dx             ; Salva na pilha o conteúdo do registrador dl;
        mov dl,0Ah          ; Move/Copia 0ah = 10 decimal = ASCII ´NEW_LINE/NOVA_LINHA´;
        int 21h             ; Imprime o quê está em dl;
        pop dx              ; POP Carrega o valor salvo anteriormente com o PUSH em dl;

        inc dl              ; Incrementa dl;
        JMP MAIN_LOOP       ; Salta para a Label MAIN_LOOP;
```

A funcionalidade desse programa é muito simples, como você já sabe **int 21h** deverá imprimir o conteúdo de **dl** pois assim foi indicado em **ah,02h**. Após isso é feito uma comparação **cmp dl,57h** (57h = W), se dl não for igual a W ocorre um **jnz NOVA_LINHA**, dentro da Label **NOVA_LINHA:**, temos um **push dx**, com isso estamos salvando o atual caractere que encontra-se em **dx** na pilha, com dado salvo podemos usar **dl** a vontade então fazemos um **mov dl,0Ah** (0Ah = É um Código Numérico para Line_Feed ou New_line em ASCII). Imprimimos o conteúdo de **dl** que é uma **New_line** (Nova_Linha), e após isso com **pop dx** pegamos o valor salvo anteriormente com push, e o incrementamos com **inc dl** e por fim damos um salto incondicional para **MAIN_LOOP:** Até a condição ser satisfeita!

O.k. Tudo certo? Então aqui termina a segunda parte deste tutorial e já vou escrever a 3ª parte!

E como você já sabe imprimir o alfabeto de ‘A’ e ‘Z’ que tal imprimir o alfabeto em minúsculo também? Então essa fica sendo a sua lição de casa, imprimir o alfabeto em minúsculo e em seguida em maiúsculo, atenção é só as letras do alfabeto!

*Cara estou até falando o Asmeis :D Tipo ontem minha mãe não sabia onde por a minha sobrinha, ai eu falei: Mãe “**COMPARA**” o Chiqueiro com o Berço, se nenhum dos dois tiver espaço por causa dos brinquedos, “**SALTA**” com a menina para o carrinho, após isso “**INCREMENTA**” a mamadeira com leite para ela parar de chorar, e não esqueça de “**RETORNAR**” aos seus afazeres! Hauhauha!*

Stigma29a – By: Matheus – matheus_nab@hotmail.com