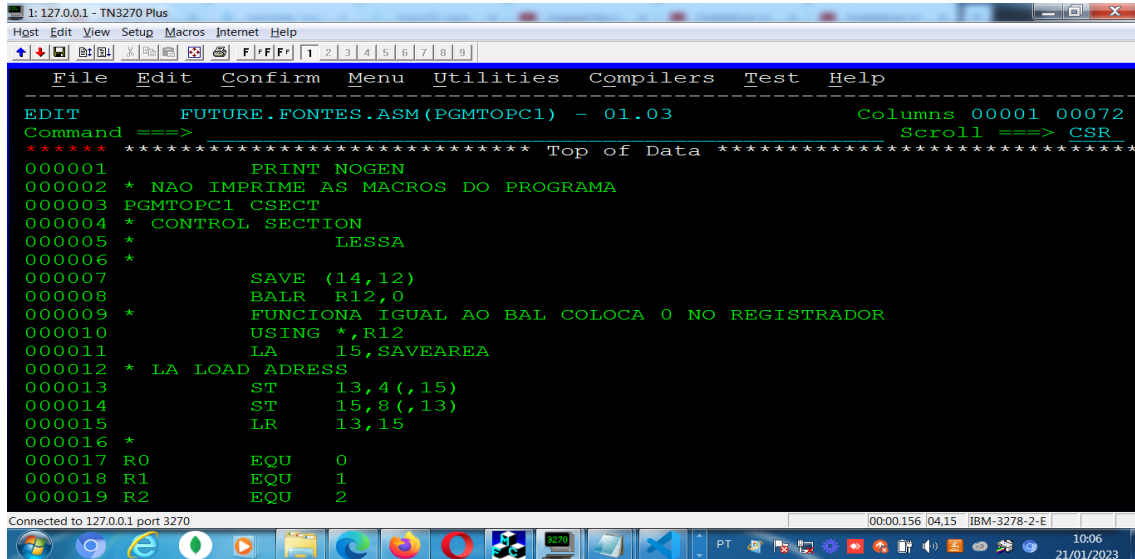


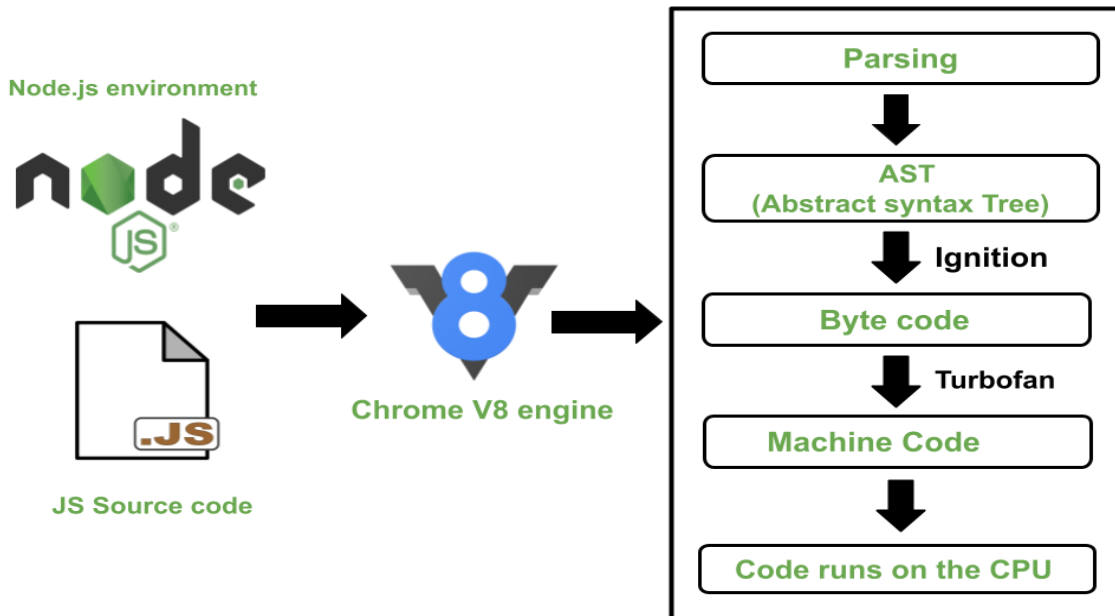
Assembly para mainframe.

Existe uma versão mais recente: Webassembly.

Assembler não é linguagem mas sim o compilador.



```
EDIT          FUTURE.FONTES.ASM (PGMTOPC1) - 01.03          Columns 00001 00072
Command ==>          Scroll ==> CSR
*****          ***** Top of Data          *****
000001          PRINT NOGEN
000002          * NAO IMPRIME AS MACROS DO PROGRAMA
000003          PGMTOPC1 CSECT
000004          * CONTROL SECTION
000005          *          LESSA
000006          *
000007          SAVE (14,12)
000008          BALR R12,0
000009          *          FUNCIONA IGUAL AO BAL COLOCA 0 NO REGISTRADOR
000010          USING *,R12
000011          LA 15,SAVEAREA
000012          * LA LOAD ADDRESS
000013          ST 13,4(,15)
000014          ST 15,8(,13)
000015          LR 13,15
000016          *
000017          EQU 0
000018          EQU 1
000019          EQU 2
```

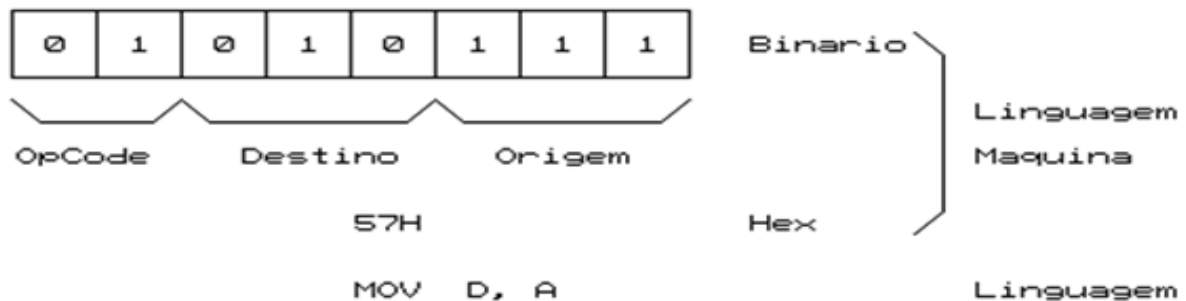


## Assembly

Linguagem de baixo nível seria língua mais próximas do binário, linguagem que trabalha diretamente com os registros do hardware. De alto nível seria próximo ao inglês ou da nossa linguagem escrita.

O que é um linguagem de alto nível?

É uma linguagem cuja sintaxe – entenda “sintaxe” como o padrão de formação das frases de um idioma – está mais próxima da nossa linguagem que dá do computador. Em outras palavras, é mais fácil entender os comandos, já que eles utilizam palavras como “print” ou “delete”.



<i>Assembly Language</i>	<i>Machine Language</i>
ST 1, [801]	00100101 11010011
ST 0, [802]	00100100 11010100
TOP: BEQ [802], 10, BOT	10001010 01001001 11110000
INCR [802]	01000100 01010100
MUL [801], 2, [803]	01001000 10100111 10100011
ST [803], [801]	11100101 10101011 00000010
JMP TOP	00101001
BOT: LD A, [801]	11010101
CALL PRINT	11010100 10101000
	10010001 01000100



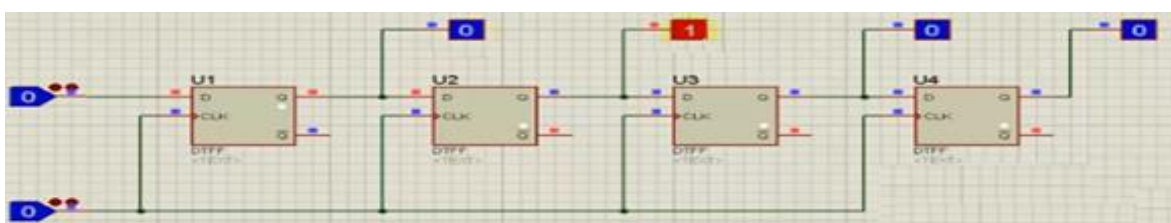
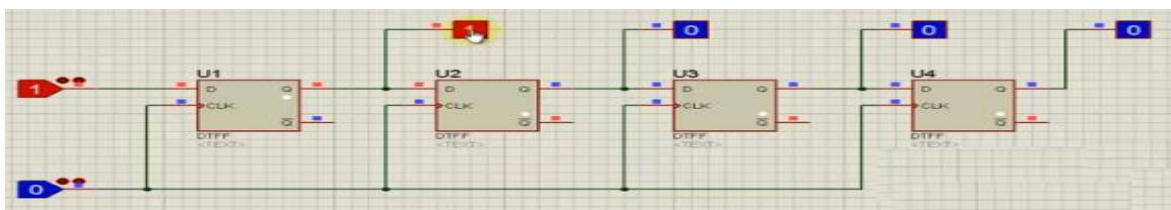
## Como funcionam os Registradores Assembly:

Os registradores armazenam valores, assim como as variáveis que utilizamos em nossa linguagem de programação como (int valor = 5) em assembly ficaria (MOV AX, 5).

## Registros especiais

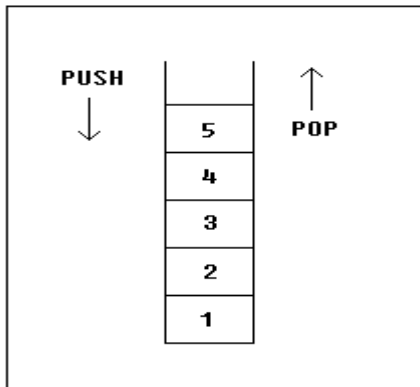
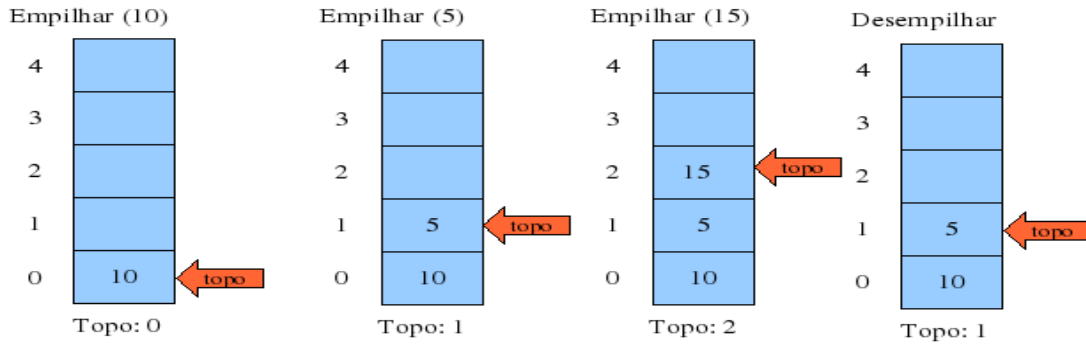
- R0 – Program Counter (PC)
  - Endereço da próxima instrução.
- R1 – Stack Pointer (SP)
  - Endereço da pilha
- R2 – Status Register (SR)
  - Guarda as flags da última operação
  - Carry, overflow, Negative, Zero
- R3 – Constant Generator (CG)

# Flip-flop



Hexadecimal	Binário		Hexadecimal	Binário
0	0000		8	1000
1	0001		9	1001
2	0010		A	1010
3	0011		B	1011
4	0100		C	1100
5	0101		D	1101
6	0110		E	1110
7	0111		F	1111

Uma pilha, em inglês *stack*, é uma estrutura de dados LIFO -- Last In First Out-- onde o último dado a entrar é o primeiro a sair. Imagine uma pilha de livros onde você vai colocando um livro sobre o outro e, após empilhar tudo, você resolve retirar um de cada vez.



MOV AX, 03h ; AX = 03h

PUSH AX ; PUSH AX na pilha (coloca no topo)

MOV AX, 04Eh ; AX = 04Eh

POP AX ; AX = 03h

### Tratamento de sequências

1 2

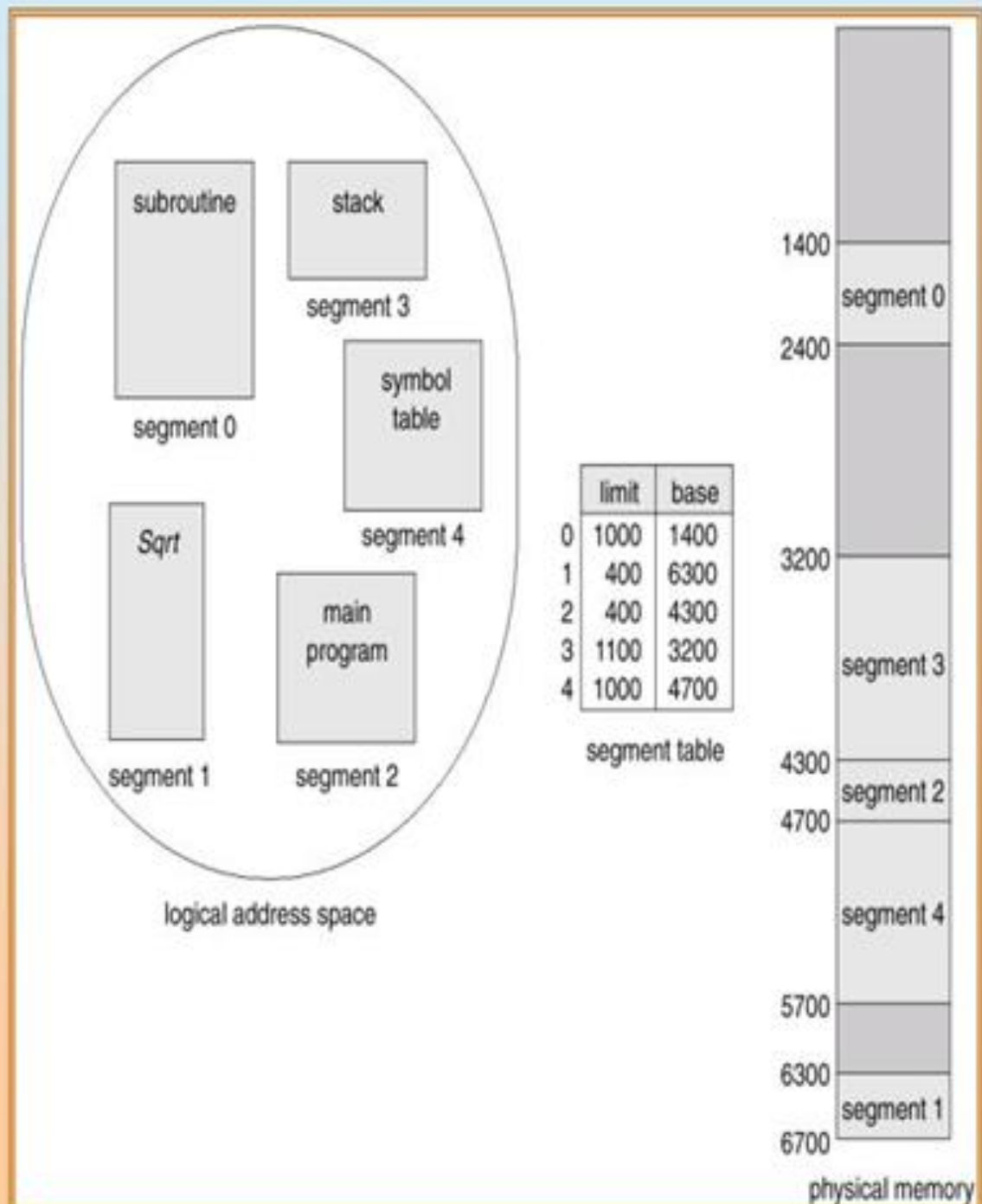
#### Pilhas



#### Matrizes



# Exemplo de Segmentação



**Tabela 9.3** Operações comuns de conjuntos de instruções

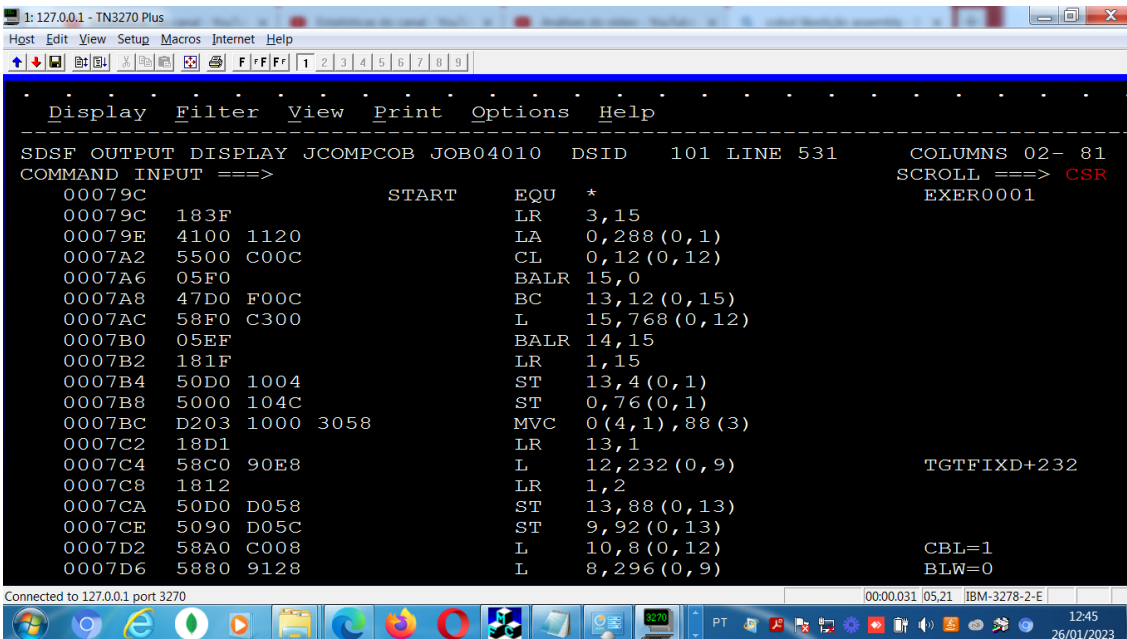
Tipo	Nome da operação	Descrição	
Operações de transferência de dados	Move	Transfere uma palavra ou bloco da fonte para o destino	
	Store	Transfere uma palavra do processador para a memória	
	Load	Transfere uma palavra da memória para o processador	
	Exchange	Troca os conteúdos dos operandos fonte e de destino	
	Clear	Transfere uma palavra contendo 0s para o destino	
	Set	Transfere um palavra contendo 1s para o destino	
	Push Pop	Transfere uma palavra da fonte para o topo da pilha Transfere uma palavra do topo da pilha para o destino	
Operações aritméticas	Add	Soma dois operandos	
	Subtract	Calcula a diferença entre dois operandos	
	Multiply	Calcula o produto de dois operandos	
	Divide	Calcula o quociente de dois operandos	
	Absolute	Substitui o operando pelo seu valor absoluto	
	Negate	Muda o sinal do operando	
	Increment Decrement	Soma 1 ao operando Subtrai 1 do operando	
Operações lógicas	AND OR NOT (Complemento) Exclusive-OR Test	Efetua a operação lógica especificada, bit a bit	
	Compare		Testa a condição especificada; atualiza códigos de condição ( <i>flags</i> ), de acordo com o resultado
	Set control variables	Efetua uma comparação lógica ou aritmética de dois ou mais operandos; atualiza códigos de condição ( <i>flags</i> ), de acordo com o resultado	
	Shift	Classe de instruções para especificar informação de controle, para fins de proteção, tratamento de interrupção, controle de temporização etc.	
	Rotate	Deslocamento de operando para a esquerda (direita), introduzindo constantes no final	
	Operações de transferência de controle	Jump (branch) Jump conditional	Desvio incondicional; carrega o PC com o endereço especificado
		Jump to subroutine Return	Testa a condição especificada; carrega ou não o PC com o endereço especificado, conforme o resultado do teste
Execute		Armazena informação de controle do programa corrente em uma posição conhecida; desvia para o endereço especificado	
Skip Skip conditional		Substitui o conteúdo do PC e de outros registradores com os valores armazenados em uma posição conhecida	
Halt Wait (hold)		Busca o operando em uma posição especificada e executa o valor desse operando como uma instrução; não modifica o PC	
No operation		Incrementa o PC (para o endereço da próxima instrução)	
		Testa a condição especificada; desvia ou não com base no resultado do teste	
	Pára a execução do programa		
	Pára a execução do programa; testa a condição especificada repetidamente; retoma a execução quando a condição é satisfeita		
	Não efetua nenhuma operação e continua a execução do programa		

Mnemonic	Instruction	Action
ADC	Add with carry	$Rd := Rn + Op2 + Carry$
ADD	Add	$Rd := Rn + Op2$
AND	AND	$Rd := Rn \text{ AND } Op2$
B	Branch	$R15 := \text{address}$
BIC	Bit Clear	$Rd := Rn \text{ AND NOT } Op2$
BL	Branch with Link	$R14 := R15, R15 := \text{address}$
BX	Branch and Exchange	$R15 := Rn,$ $T \text{ bit} := Rn[0]$
CDP	Coprocessor Data Processing	(Coprocessor-specific)
CMN	Compare Negative	$CPSR \text{ flags} := Rn + Op2$
CMP	Compare	$CPSR \text{ flags} := Rn - Op2$
EOR	Exclusive OR	$Rd := (Rn \text{ AND NOT } Op2)$ $\text{OR } (Op2 \text{ AND NOT } Rn)$
LDC	Load coprocessor from memory	Coprocessor load
LDM	Load multiple registers	Stack manipulation (Pop)
LDR	Load register from memory	$Rd := (\text{address})$
MCR	Move CPU register to coprocessor register	$cRn := rRn \{<op>cRm\}$
MLA	Multiply Accumulate	$Rd := (Rm * Rs) + Rn$
MOV	Move register or constant	$Rd := Op2$
MRC	Move from coprocessor register to CPU register	$Rn := cRn \{<op>cRm\}$
MRS	Move PSR status/flags to register	$Rn := PSR$
MSR	Move register to PSR status/flags	$PSR := Rm$
MUL	Multiply	$Rd := Rm * Rs$
MVN	Move negative register	$Rd := 0xFFFFFFFF \text{ EOR } Op2$
ORR	OR	$Rd := Rn \text{ OR } Op2$
RSB	Reverse Subtract	$Rd := Op2 - Rn$
RSC	Reverse Subtract with Carry	$Rd := Op2 - Rn - 1 + Carry$
SBC	Subtract with Carry	$Rd := Rn - Op2 - 1 + Carry$
STC	Store coprocessor register to memory	$\text{address} := cRn$
STM	Store Multiple	Stack manipulation (Push)
STR	Store register to memory	$<\text{address}> := Rd$
SUB	Subtract	$Rd := Rn - Op2$
SWI	Software Interrupt	OS call
SWP	Swap register with memory	$Rd := [Rn], [Rn] := Rm$
TEQ	Test bitwise equality	$CPSR \text{ flags} := Rn \text{ EOR } Op2$
TST	Test bits	$CPSR \text{ flags} := Rn \text{ AND } Op2$



Mn	Descrição	Flags	Mn	Descrição	Flags
EQ	equal	$Z$	NE	not equal	$\bar{Z}$
CS	carry set	$C$	CC	carry clear	$\bar{C}$
HS	higher or same		LO	lower	
MI	minus/negative	$N$	PL	plus/positive	$\bar{N}$
VS	overflow	$V$	VC	no overflow	$\bar{V}$
HI	higher	$\bar{Z}C$	LS	lower or same	$Z + \bar{C}$
GE	greater or equal	$NV + \bar{N}\bar{V}$	LT	less than	$N\bar{V} + \bar{N}V$
GT	greater than	$N\bar{Z}V + \bar{N}\bar{Z}\bar{V}$	LE	less or equal	$Z + N\bar{V} + \bar{N}\bar{V}$

Após a compilação de um programa COBOL, o módulo de carga é gerado pelo linkeditor através de um programa em assembly gerado na compilação(se não houver erro) .



```

SDSF OUTPUT DISPLAY JCOMPJOB04010 DSID 101 LINE 531 COLUMNS 02- 81
COMMAND INPUT ==> SCROLL ==> CSR
EXER0001
00079C          START      EQU *
00079C 183F          LR      3,15
00079E 4100 1120     LA      0,288(0,1)
0007A2 5500 C00C     CL      0,12(0,12)
0007A6 05F0          BALR   15,0
0007A8 47D0 F00C     BC      13,12(0,15)
0007AC 58F0 C300     L       15,768(0,12)
0007B0 05EF          BALR  14,15
0007B2 181F          LR      1,15
0007B4 50D0 1004     ST      13,4(0,1)
0007B8 5000 104C     ST      0,76(0,1)
0007BC D203 1000 3058 MVC     0(4,1),88(3)
0007C2 18D1          LR      13,1
0007C4 58C0 90E8     L       12,232(0,9)          TGTFIXD+232
0007C8 1812          LR      1,2
0007CA 50D0 D058     ST      13,88(0,13)
0007CE 5090 D05C     ST      9,92(0,13)
0007D2 58A0 C008     L       10,8(0,12)          CBL=1
0007D6 5880 9128     L       8,296(0,9)          BLW=0

```

## STATEMENT FORMAT

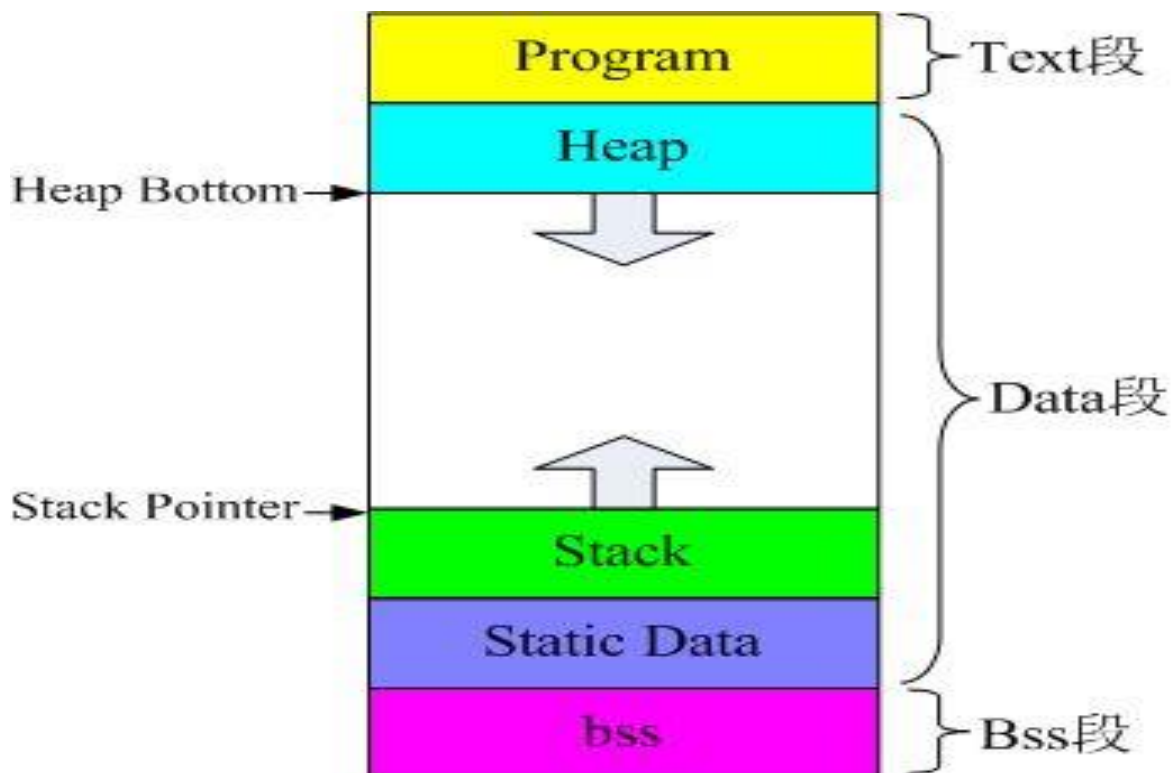
1	10	16	30
label	operation	operands	comments

Label coluna 1 e op-code coluna 10 operando coluna 16.

Comentários :

;comentário

\*comentário



**.Data definimos as constantes.**

**.BSS (Block Start Symbol) é onde definimos as variáveis que terão valores alterados.**

**.Text definimos as rotinas do programa.**