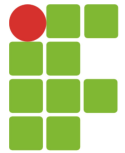
Assembly SaM

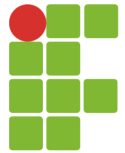
Compiladores

Prof. Daniel Saad Nogueira Nunes

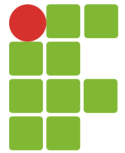
IFB – Instituto Federal de Brasília,
Campus Taguatinga



Sumário

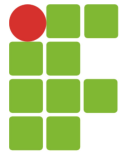


Sumário

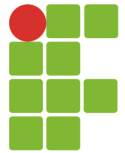


Introdução

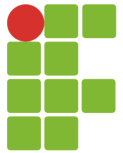
- O SaM foi escolhido nessa disciplina e será a linguagem alvo em que o código será produzido.
- Em outras palavras, os compiladores desenvolvidos deverão compilar o código na linguagem de programação prototipada para o assembly SaM.
- Trata-se de um assembly bem simples, com funções didáticas, ideal para a proposta da disciplina.



Sumário

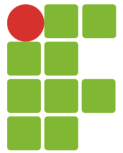


Sumário



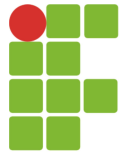
SaM: visão geral

- O nome SaM vem de Stack Machine.
- É uma abstração criada pelo prof. Keshav Pingali (University of Texas).
- É uma máquina baseada em pilhas, isto é, um dispositivo em que suas operações básicas consistem em inserções e remoções no topo de uma pilha.
- Os códigos SAMCODE executam nesta máquina. O SAMCODE é um pseudo-assembly que imita características de assemblys reais.
- Uma **máquina virtual** SaM deve conseguir interpretar um SAMCODE.



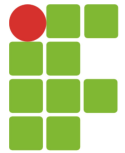
SaM: visão geral

- Todas as operações do SAMCODE são do tipo `<op-code> <operand>`, ou `<op-code>`, isto é, o código da operação seguido por 0 ou 1 operandos.
- Exemplo: `PUSHIMM 3` insere o valor 3 no topo da pilha.
- Exemplo: `ADD` insere desempilha os dois elementos do topo da pilha, realiza a soma entre eles, e insere no topo o resultado da soma.

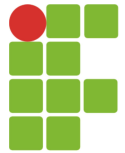


SaM: visão geral

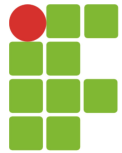
- A máquina SaM consegue lidar com inteiros de 32-bits, número ponto flutuante (float) no padrão IEEE-754, caracteres e strings.



Sumário



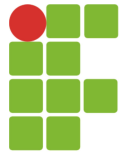
Sumário



SAM: organização

A máquina SAM tem três áreas de memória: o **programa**, a **pilha** e a **heap**.

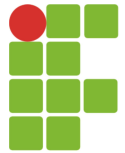
- Programa: memória em que o programa será carregado previamente à sua execução. O programa é uma coleção de **instruções** em SAMCODE.
- Pilha: memória na qual a máquina SAM armazena dados durante a sua execução.
- Heap: memória na qual a máquina SAM armazena dados alocados dinâmicamente



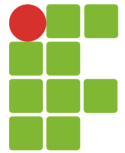
SAM: organização

A máquina SAM também conta com **registradores**, que armazenam dados relativos á execução do programa. São eles:

- PC (program counter): contém o endereço da instrução que está sendo executada.
- SP (stack pointer): contém o endereço da primeira posição **livre** da pilha. O primeiro endereço da pilha é 0 e os endereços subsequentes são incrementados de 1 em 1.
- FBR (frame based register): mantém registro das chamadas de **funções**. Inicialmente, FBR contém o endereço 0.
- HALT: um registrador que diz se a máquina SAM deve parar, ou não, de executar instruções.



Sumário

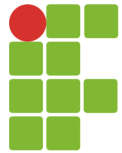


Instruções: classificação

Classificação

A máquina SAM possui 5 tipos de instruções:

- Instruções lógicas e aritméticas: lidam com operações aritméticas, como adição, subtração, multiplicação e divisão, e operações lógicas, como E, OU e NÃO, assim como comparações.
- Instruções de manipulação da pilha: copiam valores de um lugar da pilha para outro.

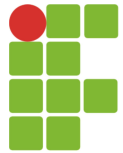


Instruções: classificação

Classificação

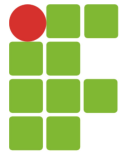
A máquina SAM possui 5 tipos de instruções:

- Instruções de manipulação de registradores: permitem que dados sejam inseridos na pilha a partir de um registrador ou retirado da pilha e armazenado em um registrador.
- Instruções de controle: permitem desvios condicionais e incondicionais no programa.
- Instruções de I/O: lidam com operações de I/O.



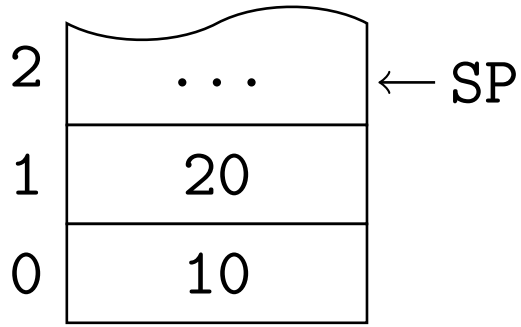
Instruções

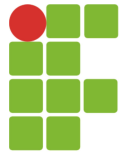
- A cada execução de uma instrução, o controle é transferido para a próxima instrução na memória de programa, e o PC é atualizado.



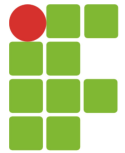
SAM: a pilha

- A pilha é a memória de uma máquina SAM em que as computações do programa são armazenadas.
- A cada dado que é inserido na pilha, o SP incrementa de uma posição.
- Abaixo temos uma figura em que dois valores, 20 e 10 são inseridos na pilha, fazendo com que o SP aponte para o endereço 2.



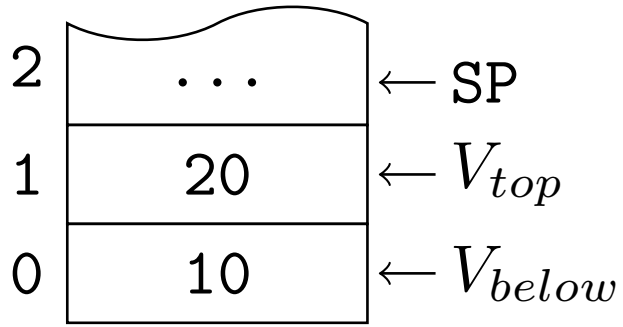


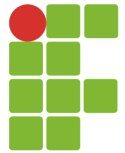
Sumário



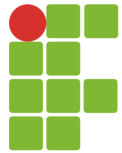
ISA

- Agora comentaremos sobre a ISA de uma máquina SAM.
- Imagine que a pilha pode ser vista como um vetor V, em que o i-ésimo elemento desse vetor é denotado por $V[i]$.
- Utilizaremos a seguinte convenção: $V_{top} = V[SP - 1]$ e $V_{below} = V[SP - 2]$. Isto é, V_{top} é o elemento que está no topo da pilha e V_{below} é o elemento abaixo do topo.



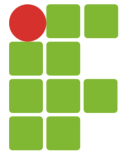


Sumário



Instruções lógicas e aritméticas

- As instruções lógicas e aritméticas são responsáveis por executar operações aritméticas, lógicas e comparações.
- Operações sobre inteiros são sempre truncadas.
- Valores booleanos podem ser encarados como inteiros, 1 para verdadeiro e 0 para falso.

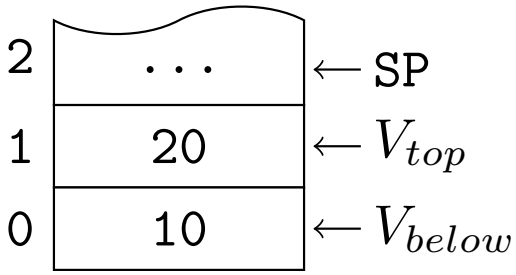


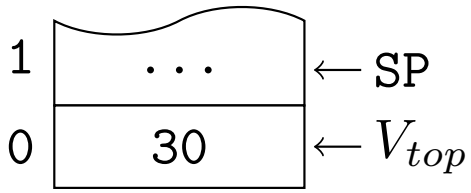
Instruções lógicas e aritméticas

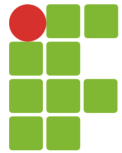
ADD

Sintaxe: `ADD`

- Desempilha V_{top}, V_{below} e Insere na pilha o resultado de $V_{below} + V_{top}$.





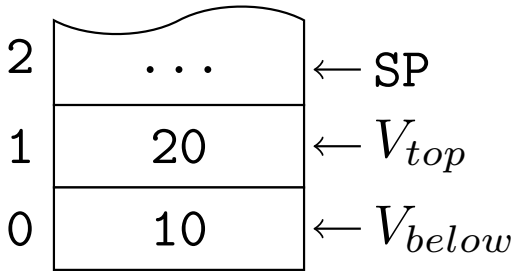


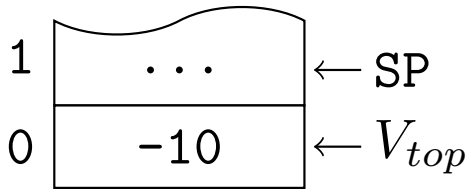
Instruções lógicas e aritméticas

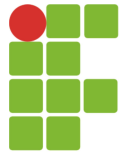
SUB

Sintaxe: `SUB`

- Desempilha V_{top}, V_{below} e insere na pilha o resultado de $V_{below} - V_{top}$.





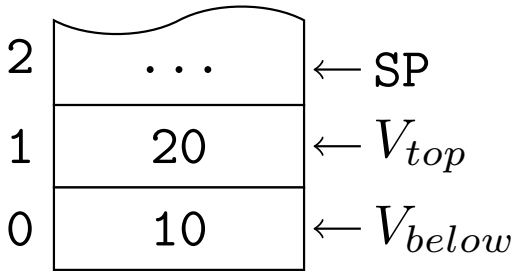


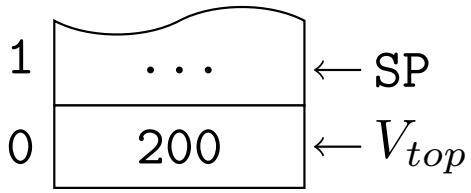
Instruções lógicas e aritméticas

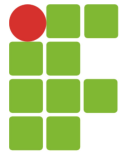
TIMES

Sintaxe: `TIMES`

- Desempilha V_{top}, V_{below} e insere na pilha o resultado de $V_{below} \cdot V_{top}$.





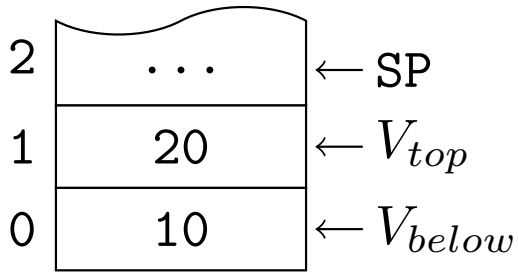


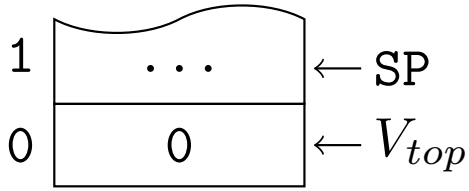
Instruções lógicas e aritméticas

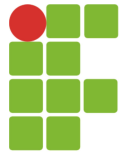
DIV

Sintaxe: `DIV`

- Desempilha V_{top}, V_{below} e insere na pilha o resultado de $\frac{V_{below}}{V_{top}}$. A divisão é inteira.





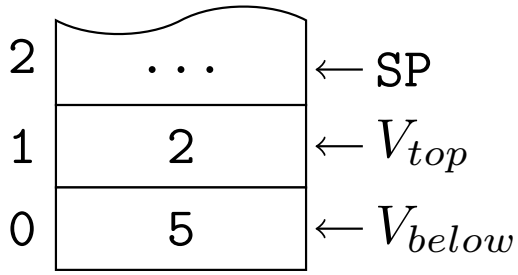


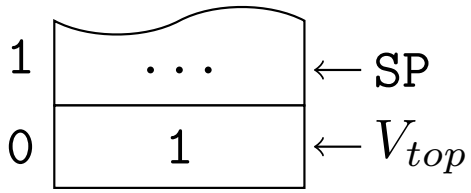
Instruções lógicas e aritméticas

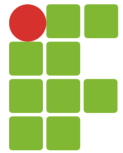
MOD

Sintaxe: `MOD`

- Desempilha V_{top},V_{below} e insere na pilha o resultado de V_{below} mod V_{top}.





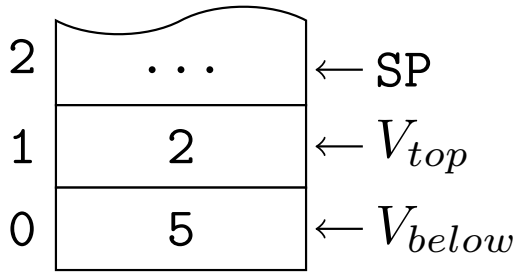


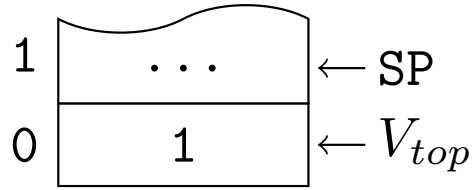
Instruções lógicas e aritméticas

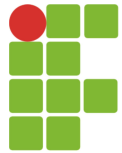
MOD

Sintaxe: `MOD`

- Insere na pilha o resultado de $V_{below} \mod V_{top}$.







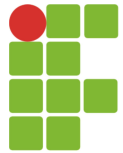
Instruções lógicas e aritméticas

LSHIFT

Sintaxe: `LSHIFT b`

- Insere na pilha o resultado de $V_{top} << b$.

Figura: Resultado de `LSHIFT 2`.



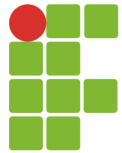
Instruções lógicas e aritméticas

RSHIFT

Sintaxe: `RSHIFT b`

- Desempilha V_{top} e insere na pilha o resultado de $V_{top} >> b$.

Figura: Resultado de `RSHIFT 2`.

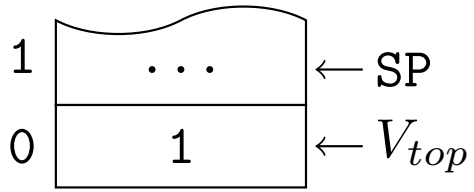


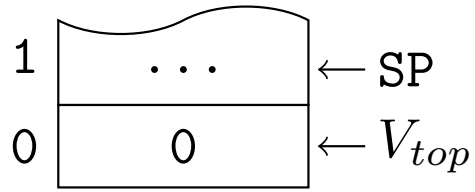
Instruções lógicas e aritméticas

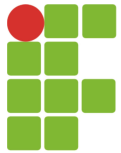
NOT

Sintaxe: `NOT`

- Desempilha V_{top} e insere na pilha o resultado de $\neg V_{top}$. Isto é, se $V_{top} \neq 0$, 0 é inserido, caso contrário, 1 é inserido.





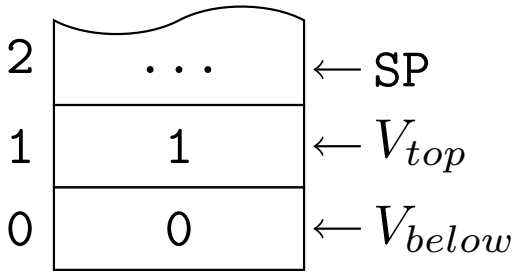


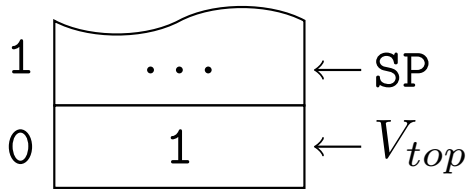
Instruções lógicas e aritméticas

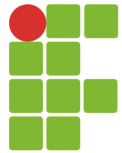
OR

Sintaxe: `OR`

- Desempilha V_{top}, V_{below} e insere na pilha o resultado de $V_{below} \vee V_{top}$.





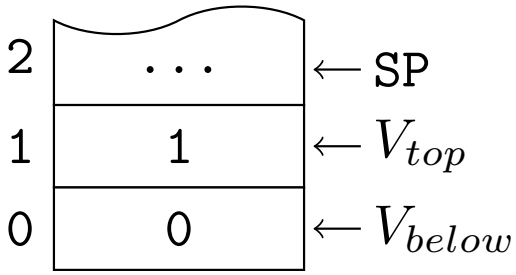


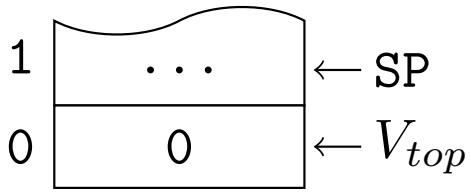
Instruções lógicas e aritméticas

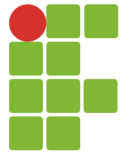
AND

Sintaxe: `AND`

- Desempilha V_{top}, V_{below} e insere na pilha o resultado de $V_{below} \wedge V_{top}$.





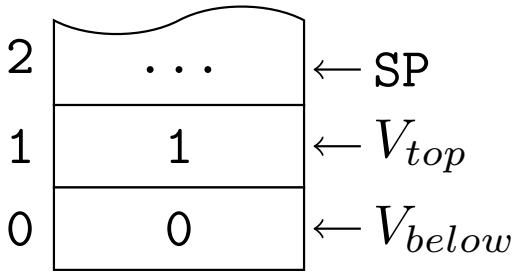


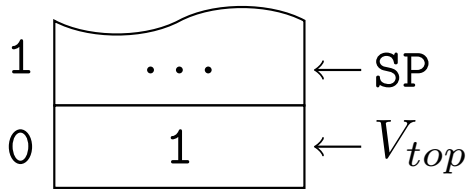
Instruções lógicas e aritméticas

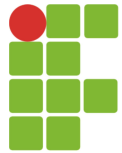
XOR

Sintaxe: `XOR`

- Desempilha V_{top}, V_{below} e insere na pilha o resultado de $V_{below} \oplus V_{top}$.





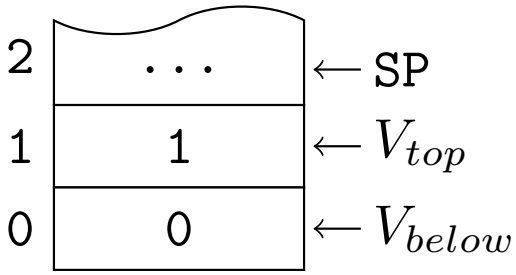


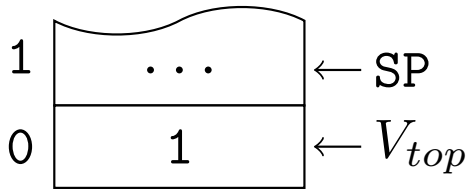
Instruções lógicas e aritméticas

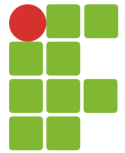
NAND

Sintaxe: `NAND`

- Desempilha V_{top}, V_{below} e insere na pilha o resultado de $\neg(V_{below} \wedge V_{top})$.





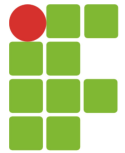


Instruções lógicas e aritméticas

BITNOT

Sintaxe: `BITNOT`

- Desempilha V_{top} e insere na pilha o resultado de $\sim V_{top}$, considera-se inteiros de 32-bits.
- Se $V_{top} = 0$ então $\sim V_{top} = \underbrace{111\ldots1}_{32}$

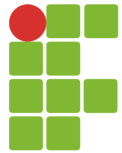


Instruções lógicas e aritméticas

BITAND

Sintaxe: `BITAND`

- Desempilha V_{top}, V_{below} e insere na pilha o resultado de V_{below} & V_{top}, considera-se inteiros de 32-bits.
- Se $V_{below} = 2$ e $V_{top} = 4$, então 0 será inserido na pilha.

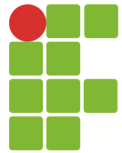


Instruções lógicas e aritméticas

BITOR

Sintaxe: `BITOR`

- Desempilha V_{top}, V_{below} e insere na pilha o resultado de $V_{below} \mid V_{top}$, considera-se inteiros de 32-bits.
- Se $V_{below} = 2$ e $V_{top} = 4$, então 6 será inserido na pilha.

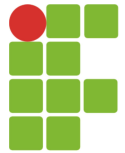


Instruções lógicas e aritméticas

BITXOR

Sintaxe: `BITXOR`

- Insere na pilha o resultado de V_{below} ^ V_{top}, considera-se inteiros de 32-bits.
- Se $V_{below} = 3$ e $V_{top} = 6$, então 5 será inserido na pilha.

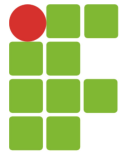


Instruções lógicas e aritméticas

BITNAND

Sintaxe: `BITNAND`

- Insere na pilha o resultado de $\sim (V_{below} \ \& \ V_{top})$, considera-se inteiros de 32-bits.
- Se $V_{below} = \underbrace{101010\ldots}_{32}$ e $V_{top} = \underbrace{010101\ldots}_{32}$, então, $\underbrace{11111\ldots}_{32}$ será inserido na pilha.

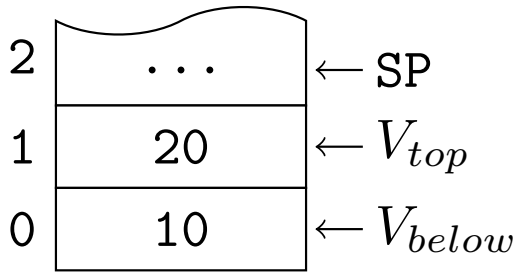


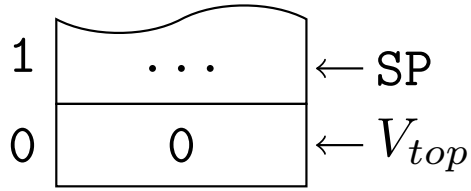
Instruções lógicas e aritméticas

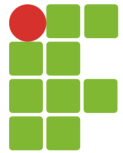
GREATER

Sintaxe: `GREATER`

- Desempilha V_{top}, V_{below} e insere na pilha 1 se $V_{below} > V_{top}$ e 0 caso contrário.





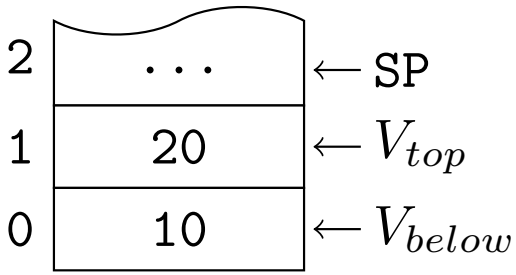


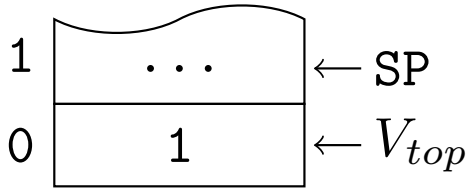
Instruções lógicas e aritméticas

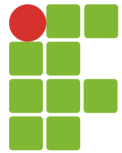
LESS

Sintaxe: `LESS`

- Desempilha V_{top}, V_{below} e insere na pilha 1 se $V_{below} < V_{top}$ e 0 caso contrário.





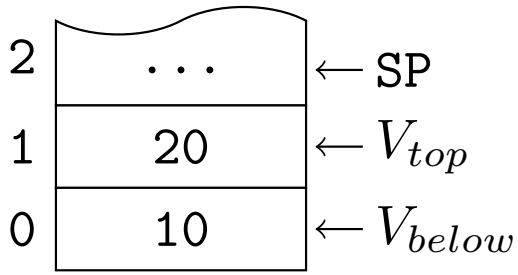


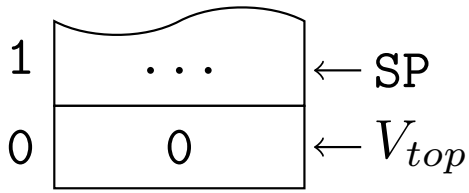
Instruções lógicas e aritméticas

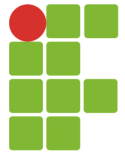
EQUAL

Sintaxe: `EQUAL`

- Desempilha V_{top}, V_{below} e insere na pilha 1 se $V_{below} = V_{top}$ e 0 caso contrário.





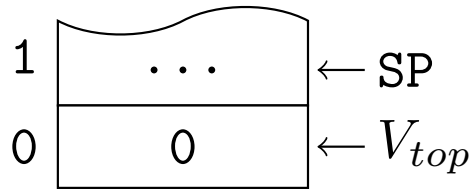


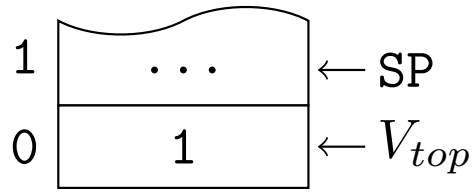
Instruções lógicas e aritméticas

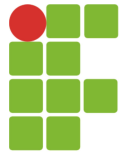
ISNIL

Sintaxe: `ISNIL`

- Desempilha V_{top} e insere na pilha 1 se $V_{top} = 0$, caso contrário, insere 0.





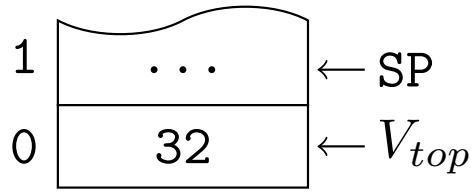


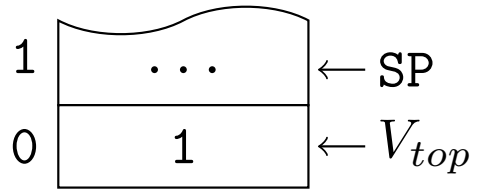
Instruções lógicas e aritméticas

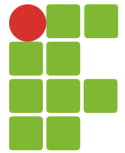
ISPOS

Sintaxe: `ISPOS`

- Insere na pilha 1 se $V_{top} > 0$, caso contrário, insere 0.





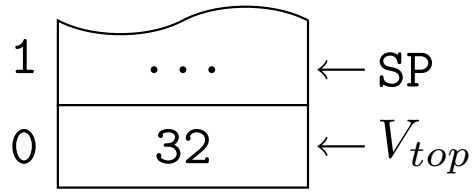


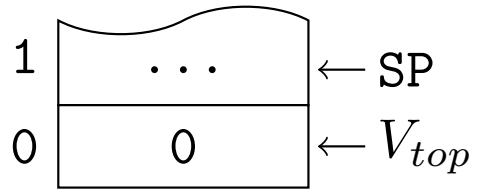
Instruções lógicas e aritméticas

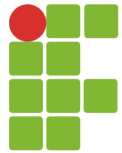
ISNEG

Sintaxe: `ISNEG`

- Insere na pilha 1 se $V_{top} < 0$, caso contrário, insere 0.







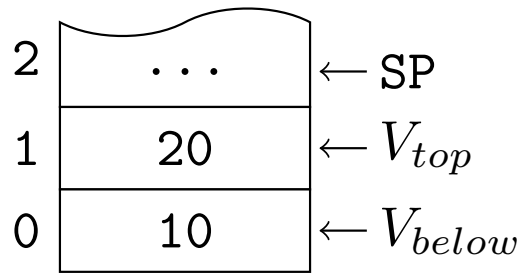
Instruções lógicas e aritméticas

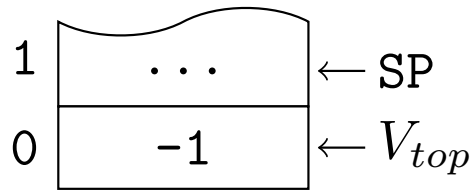
CMP

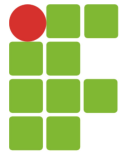
Sintaxe: `CMP`

Desempilha V_{top}, V_{below} e insere na pilha:

- -1, se $V_{below} < V_{top}$.
- 0, se $V_{below} = V_{top}$.
- 1, se $V_{below} > V_{top}$.



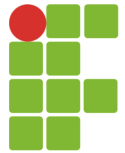




Operações em números ponto-flutuante

Temos instruções para números ponto-flutuante análogas às operações para inteiros.

- `ADDF`: adição.
- `SUBF`: subtração.
- `TIMESF`: multiplicação.
- `DIVF`: divisão.
- `CMPF`: comparação.

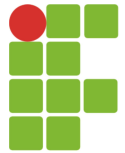


Conversão inteiro para ponto-flutuante

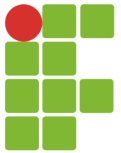
Conversão

As seguintes instruções convertem inteiros para ponto-flutuante ou vice-versa.

- `ITOF` : converte V_{top} para a sua versão ponto-flutuante.
- `FTOI` : converte V_{top} para a sua versão inteira.
- `FTOI` : converte o arredondamento de V_{top} para a sua versão inteira.



Sumário



Instruções de manipulação da pilha

PUSHIMM

Sintaxe: `PUSHIMM x`.

- Insere o inteiro x na pilha.

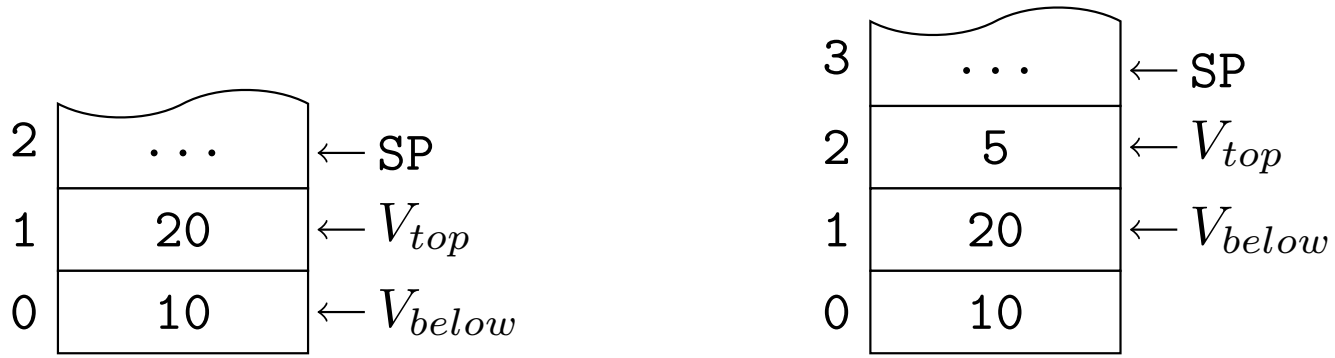
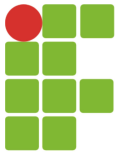


Figura: Pilha após `PUSHIMM 5`.



Instruções de manipulação da pilha

PUSHIMMF

Sintaxe: `PUSHIMMF x`.

- Insere o número ponto-flutuante x na pilha.

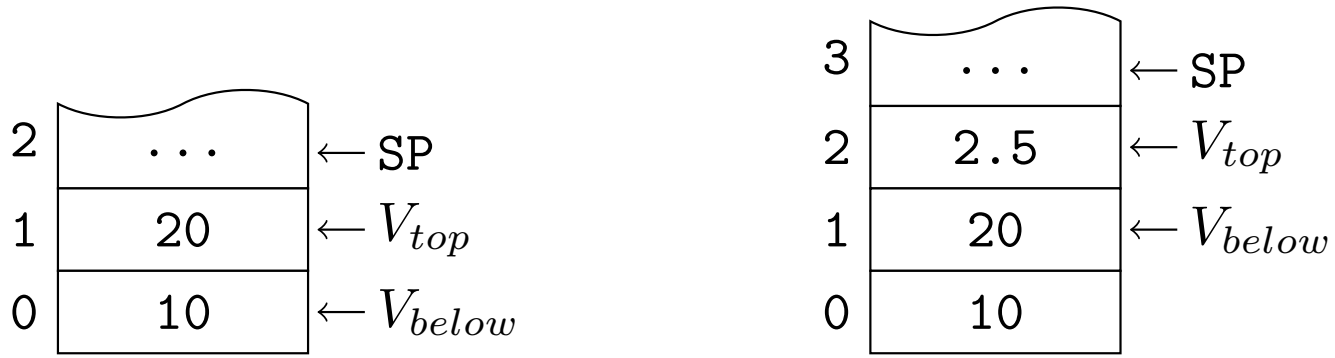
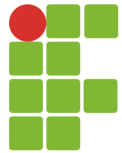


Figura: Pilha após `PUSHIMMF 2.5`.



Instruções de manipulação da pilha

PUSHIMMCH

Sintaxe: `PUSHIMMCH c`.

- Insere o caractere c na pilha. Lembrando que caracteres são inteiros no intervalo da tabela ASCII, logo, o caractere 'c', por exemplo, equivale ao inteiro 99.

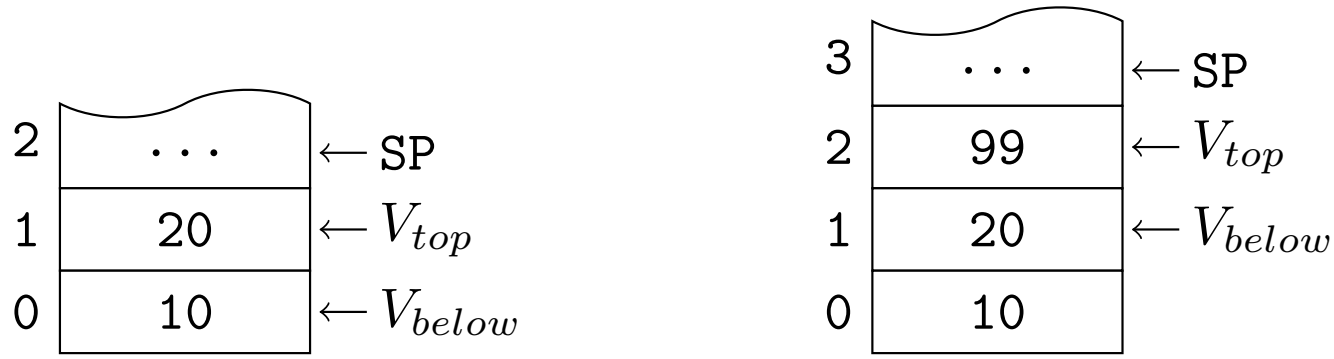
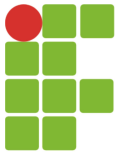


Figura: Pilha após `PUSHIMMCH 'c'`.



Instruções de manipulação da pilha

PUSHIMMSTR s

Sintaxe: `PUSHIMMSTR s`.

- Insere a string s na heap e insere o endereço de s na pilha.

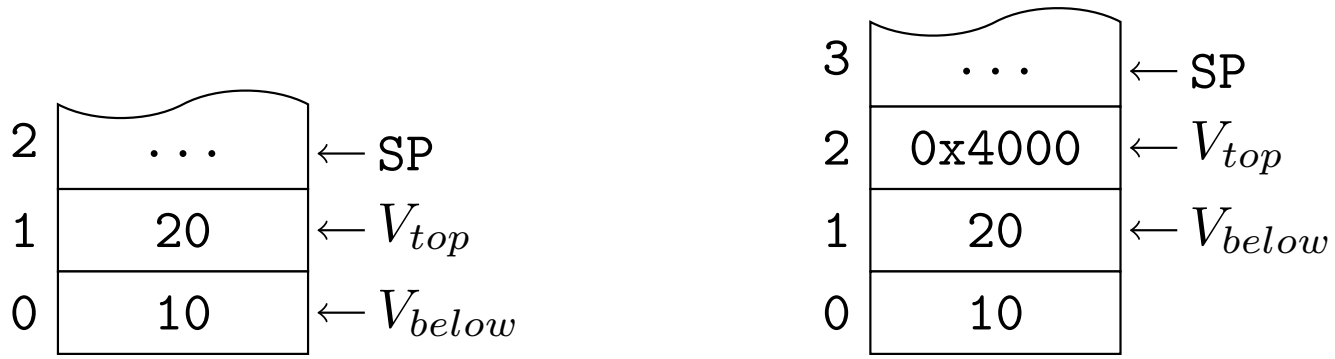
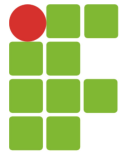


Figura: Pilha após `PUSHIMMSTR "abracadabra"`, assumindo que a string ocupa a posição 0x4000 da heap.

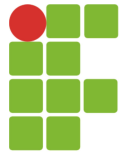


Instruções de manipulação da pilha

PUSHIMMPA label

Sintaxe: `PUSHIMMPA` .

- Insere o endereço de programa rotulado por label na pilha.

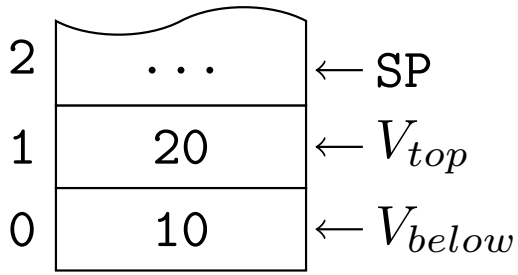


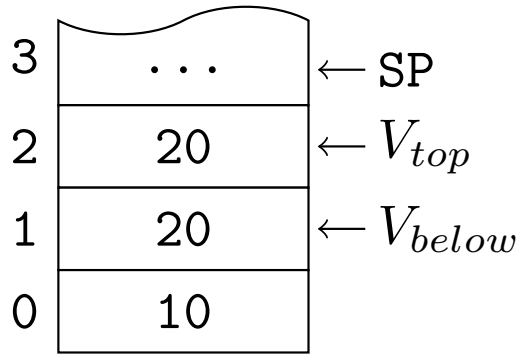
Instruções de manipulação da pilha

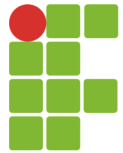
DUP

Sintaxe: `DUP`.

- Duplica V_{top}.





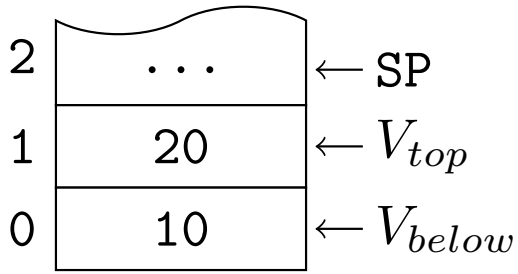


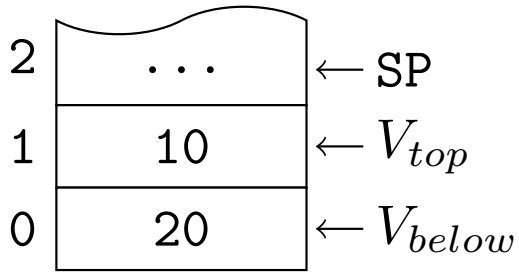
Instruções de manipulação da pilha

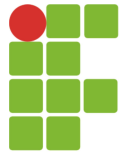
SWAP

Sintaxe: `SWAP`.

- Troca os valores de V_{top} com V_{below}.





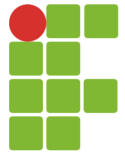


Instruções de manipulação da pilha

MALLOC

Sintaxe: `MALLOC`.

- Aloca um espaço de $V_{top} + 1$ (unidades de 32-bits) na Heap e insere na pilha o endereço do espaço alocado.
- A primeira célula de memória contém o tamanho do espaço alocado.

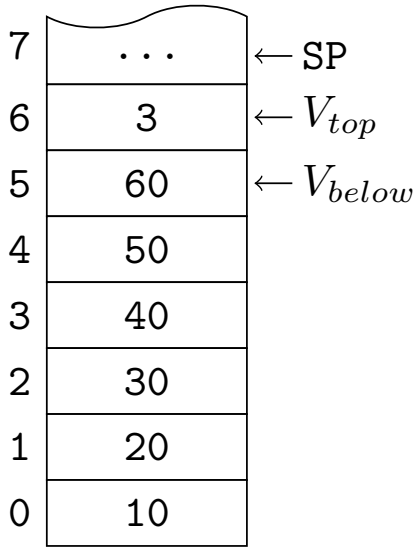


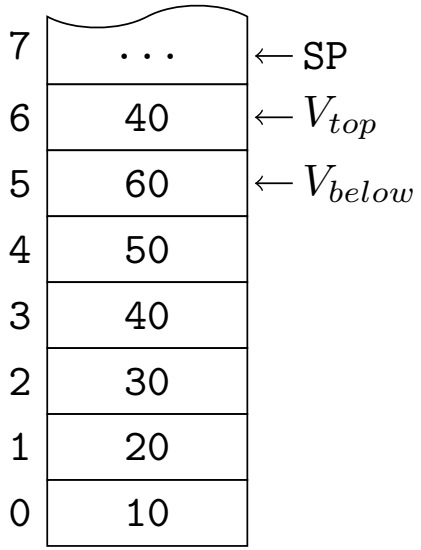
Instruções de manipulação da pilha

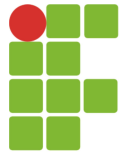
PUSHIND

Sintaxe: `PUSHIND`.

- Insere $V[m]$ na pilha, em que m é o valor de V_{top}.





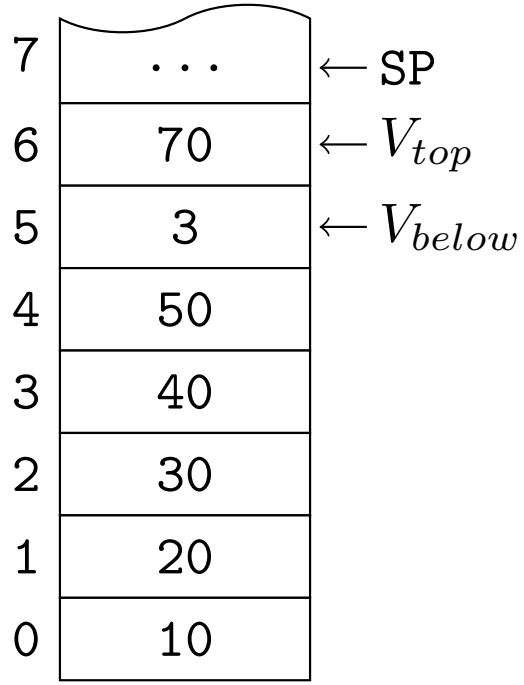


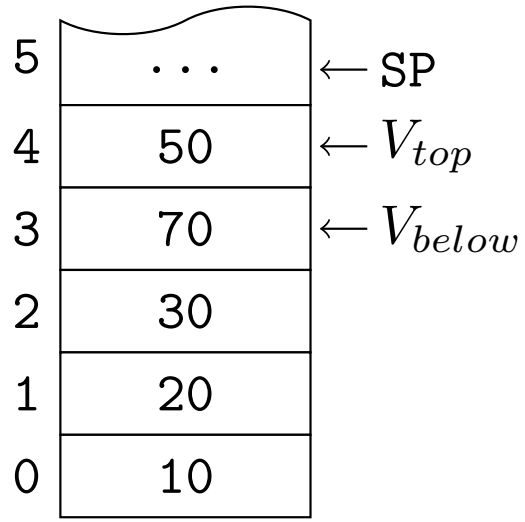
Instruções de manipulação da pilha

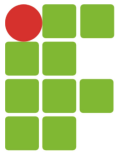
STOREIND

Sintaxe: `STOREIND`.

- Insere V_{top} na posição $V[m]$, em que m é o valor de V_{below}.







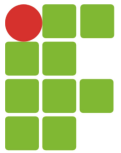
Instruções de manipulação da pilha

ADDSP

Sintaxe: `ADDSP x`.

- Faz $SP \leftarrow SP + x$.
- Útil para alocar espaço para variáveis.

Figura: Resultado de `ADDSP 3`



Instruções de manipulação da pilha

PUSHOFF

Sintaxe: `PUSHOFF k`.

- Faz $V[SP] \leftarrow V[FBR + k]$.
- Útil para recuperar valores de variáveis em funções (frames).

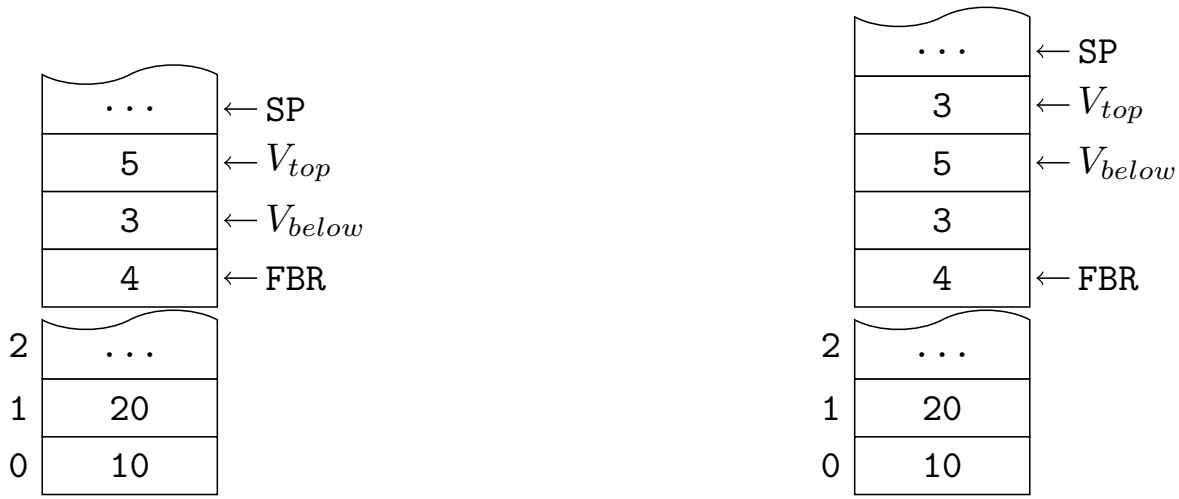
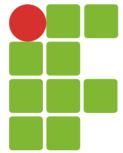


Figura: Resultado de `PUSHOFF 1`



Instruções de manipulação da pilha

STOREOFF

Sintaxe: `STOREOFF k`.

- Desempilha V_{top} e armazena o valor em $V[FBR + k]$.
- Útil para amazenar valores em variáveis em funções (frames).

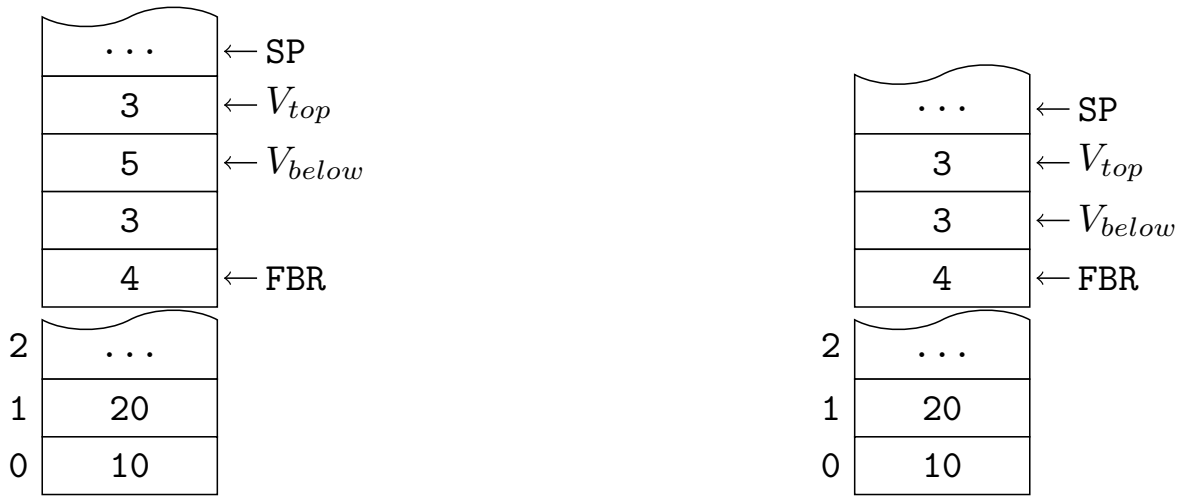
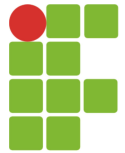
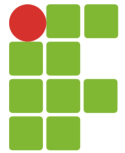


Figura: Resultado de `STOREOFF 2`



Sumário

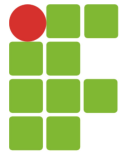


Instruções de manipulação de registradores

PUSHSP

Sintaxe: `PUSHSP`

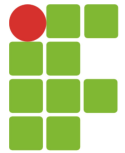
- Insere na pilha o valor de SP.



Instruções de manipulação de registradores

Sintaxe: `POPSP`

- Desempilha o valor V_{top} e o atribui a SP.

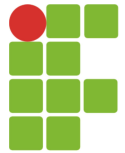


Instruções de manipulação de registradores

PUSHFBR

Sintaxe: `PUSHFBR`

- Insere o valor de FBR na pilha.
- Interessante para permitir o retorno da função para quem a invocou.

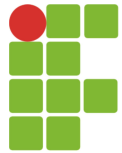


Instruções de manipulação de registradores

POPFBR

Sintaxe: `POPFBR`

- Desempilha V_{top} e o atribui a FBR.

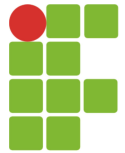


Instruções de manipulação de registradores

LINK

Sintaxe: `LINK`

- Cria um novo frame na pilha.
- Insere o valor do FBR atual na pilha e,sem seguida atribui a FBR o valor , $\mathrm{FBR} \leftarrow \mathrm{SP} - 1$.
- Similar a ao `PUSHFBR` , mas também atualiza o valor de FBR.

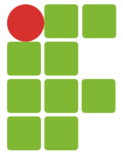


Instruções de manipulação de registradores

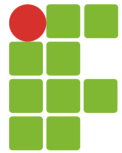
STOP

Sintaxe: `STOP`

- Atribui 1 ao registrador HALT.
- Para a execução do programa.



Sumário

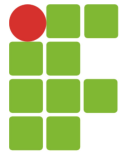


Instruções de Controle

JUMP

Sintaxe: `JUMP t`

- Jump incodicional.
- Faz $\mathrm{PC} \leftarrow t$, que pode ser um label ou um endereço inteiro.

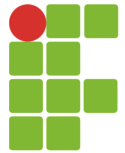


Instruções de Controle

JUMPC

Sintaxe: `JUMPC t`

- Jump condicional.
- Faz $\mathrm{PC} \leftarrow t$ somente quando o topo da pilha não é zero. O topo também é desempilhado. O operando t pode ser tanto um label quanto um endereço inteiro.

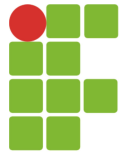


Instruções de Controle

JUMPIND

Sintaxe: `JUMPIND`

- Jump incodicional.
- Faz $\mathrm{PC} \leftarrow V_{top}$.

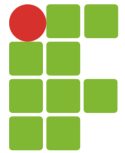


Instruções de Controle

JSR

Sintaxe: `JSR t`

- Insere $\mathrm{PC} + 1$ na pilha e faz $\mathrm{PC} \leftarrow t$, em que t que pode ser um label ou um endereço inteiro.
- Útil para invocar funções quando utilizado em conjunção com a instrução `LINK`.

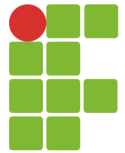


Instruções de Controle

JSRIND

Sintaxe: `JSRIND`

- Desempilha V_{top}, Faz $\mathrm{PC} \leftarrow V_{top}$, e insere $\mathrm{PC} + 1$ na pilha.
- Utilizado com a instrução `LINK` para viabilizar subrotinas.
- Diferente do `JSR` ,pois o endereço a ser atribuído ao PC está na pilha.

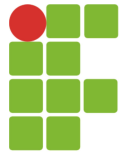


Instruções de Controle

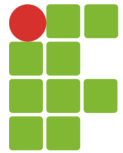
SKIP

Sintaxe: `SKIP`

- Desempilha V_{top} e faz $\mathrm{PC} \leftarrow \mathrm{PC} + V_{top} + 1$.
- Se $V_{top} = 0$, a execução segue normal. Se $V_{top} = -1$, PC continua o mesmo. Se $V_{top} = -2$, o PC aponta para a instrução anterior.
- Útil para pular instruções.



Sumário

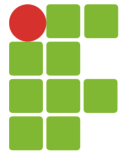


Instruções de I/O

READ

Sintaxe: `READ`

- Lê um inteiro do teclado e o insere na pilha.

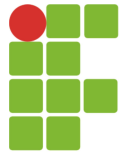


Instruções de I/O

READF

Sintaxe: `READF`

- Lê um float do teclado e o insere na pilha.

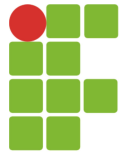


Instruções de I/O

READCH

Sintaxe: `READCH`

- Lê um caractere do teclado e o insere na pilha.

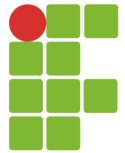


Instruções de I/O

READSTR

Sintaxe: `READSTR`

- Lê uma string do teclado, a armazena na Heap e empilha o endereço ocupado por essa string.

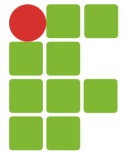


Instruções de I/O

WRITE

Sintaxe: `WRITE`

- Desempilha um inteiro e o imprime na tela.

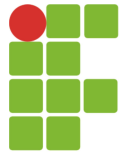


Instruções de I/O

WRITEF

Sintaxe: `WRITEF`

- Desempilha um float e o imprime na tela.

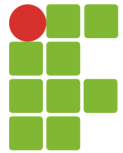


Instruções de I/O

WRITECH

Sintaxe: `WRITECH`

- Desempilha um caractere e o empilha na tela.

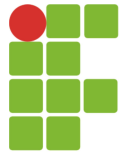


Instruções de I/O

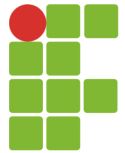
WRITESTR

Sintaxe: `WRITESTR`

- Desempilha o endereço da string na heap e imprime a string.



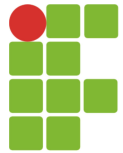
Sumário



Programas

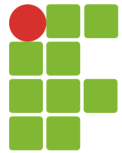
Qual seria o SAMCODE para o seguinte código?

```
main(){
     int x,y;
     x = 5;
     y = x + 6;
}
```



Programas

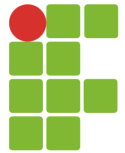
```
ADDSP 2
PUSHIMM 5
STOREOFF 0
PUSHOFF 0
PUSHIMM 6
ADD
STOREOFF 1
ADDSP -2
EXIT
```



Programas

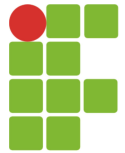
Qual seria o SAMCODE para o seguinte código?

```
main(){
    int x,y,z;
    x = 5;
    y = 3;
    if(x>y){
        z = x;
    }
    else{
        z = y;
    }
}
```



Programas

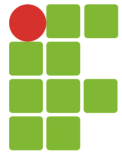
```
ADDSP 3
PUSHIMM 5
STOREOFF 0
PUSHIMM 3
STOREOFF 1
PUSHOFF 0
PUSHOFF 1
GREATER
JUMPC label_if
PUSHOFF 1
STOREOFF 2
JUMP label_endif
label_if:
PUSHOFF 0
STOREOFF 2
label_endif:
ADDSP -3
```



Programas

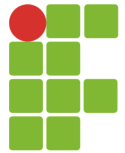
Qual seria o SAMCODE para o seguinte código?

```
main(){
    int x,y,z;
    x = 5;
    y = 3;
    while(x>y){
        z = x + y;
        y++;
    }
}
```



Programas

```
ADDSP 3
PUSHIMM 5
STOREOFF 0
PUSHIMM 3
STOREOFF 1
label_while:
PUSHOFF 0
PUSHOFF 1
GREATER
ISNIL
JUMPC label_endwhile
PUSHOFF 0
PUSHOFF 1
ADD
STOREOFF 2
PUSHOFF 1
PUSHIMM 1
ADD
STOREOFF 1
JUMP label_while
label_endwhile:
ADDSP -3
EXIT
```

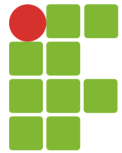


Programas

Qual seria o SAMCODE para o seguinte código?

```
int f(int x){
    x++;
    return x;
}

main(){
    int x = 2;
    x = f(x);
}
```



Programas

```
ADDSP 1
PUSHIMM 2
STOREOFF 0
PUSHIMM 0 // Adicionando o espaço para o retorno da função na pilha
PUSHOFF 0 // adicionando parâmetro da função na pilha
LINK // Cria um novo frame e salva FBR na pilha
JSR funcao // Muda o PC para o código da função
POPFBR
ADDSP -1 // Remove o parâmetro da função empilhado
STOREOFF 0 // armazena o resultado da função na variável local da main
ADDSP -1 // Remove a variável local da main
STOP
funcao:
PUSHOFF -1 // recupera valor do parâmetro e o insere na pilha
PUSHIMM 1
ADD
STOREOFF -2 // atribui ao retorno da função o resultado da função
JUMPIND // retorna para a main
```