Insper

Elementos de Sistemas

Conteúdo 12 – Programação em Assembly

“A carta que escrevi hoje é mais longa que o usual pois não tive tempo de fazer ela mais curta."
“The letter I have written today is longer than usual because I lacked the time to make it shorter."
Blaise Pascal (1623–1662) **matemático frances**

Engenharia - Luciano Soares <lpsoares@insper.edu.br>

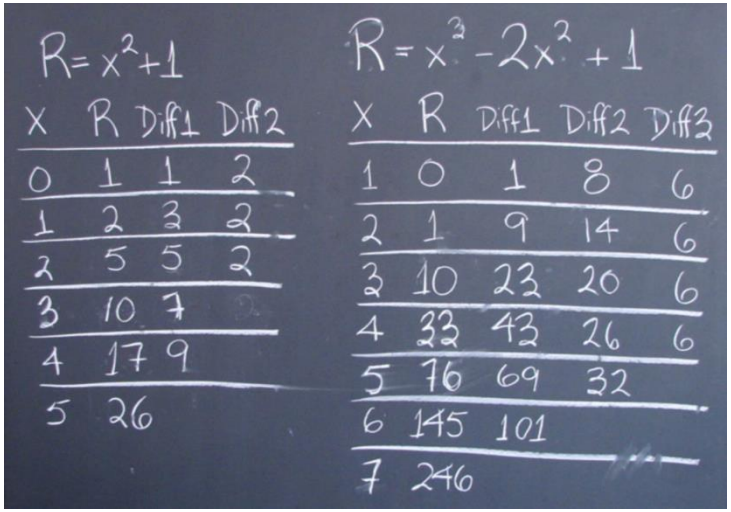
Charles Babbage

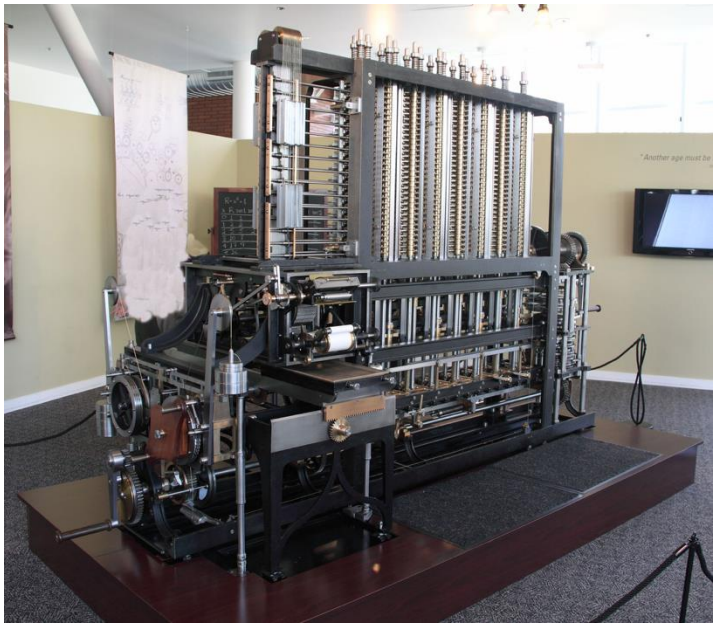
Charles Babbage projetou e construiu maquinas para imprimir tabulações de funções polinomiais de forma automática.

https://www.youtube.com/watch?v=QVxbNZWLP60

Máquinas de Babbage

Máquina Diferencial usava o princípio da diferença dividida para os cálculos das funções polinomiais.



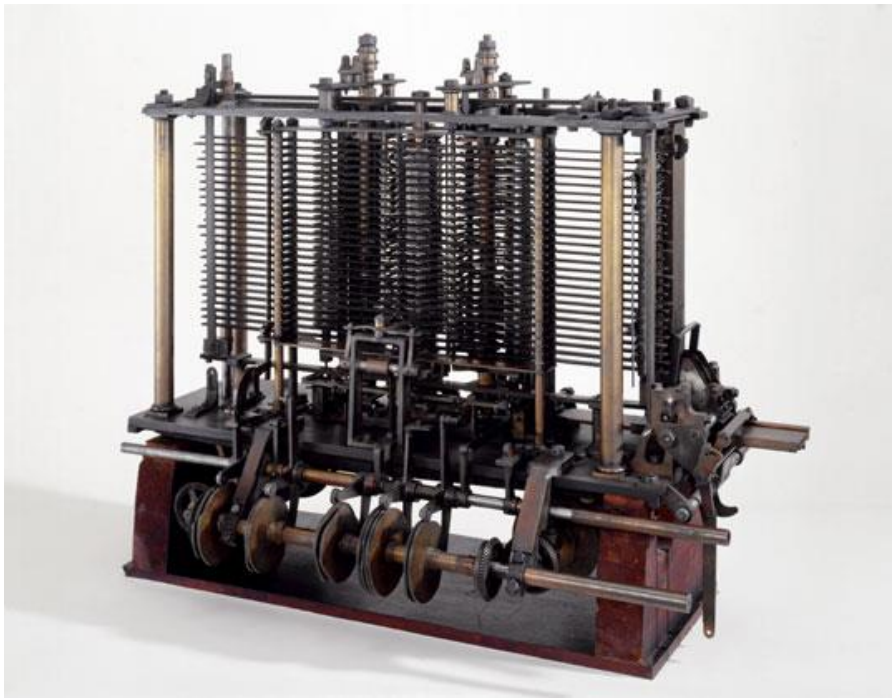


https://www.youtube.com/watch?v=qctHEGKr9Zs

Insper

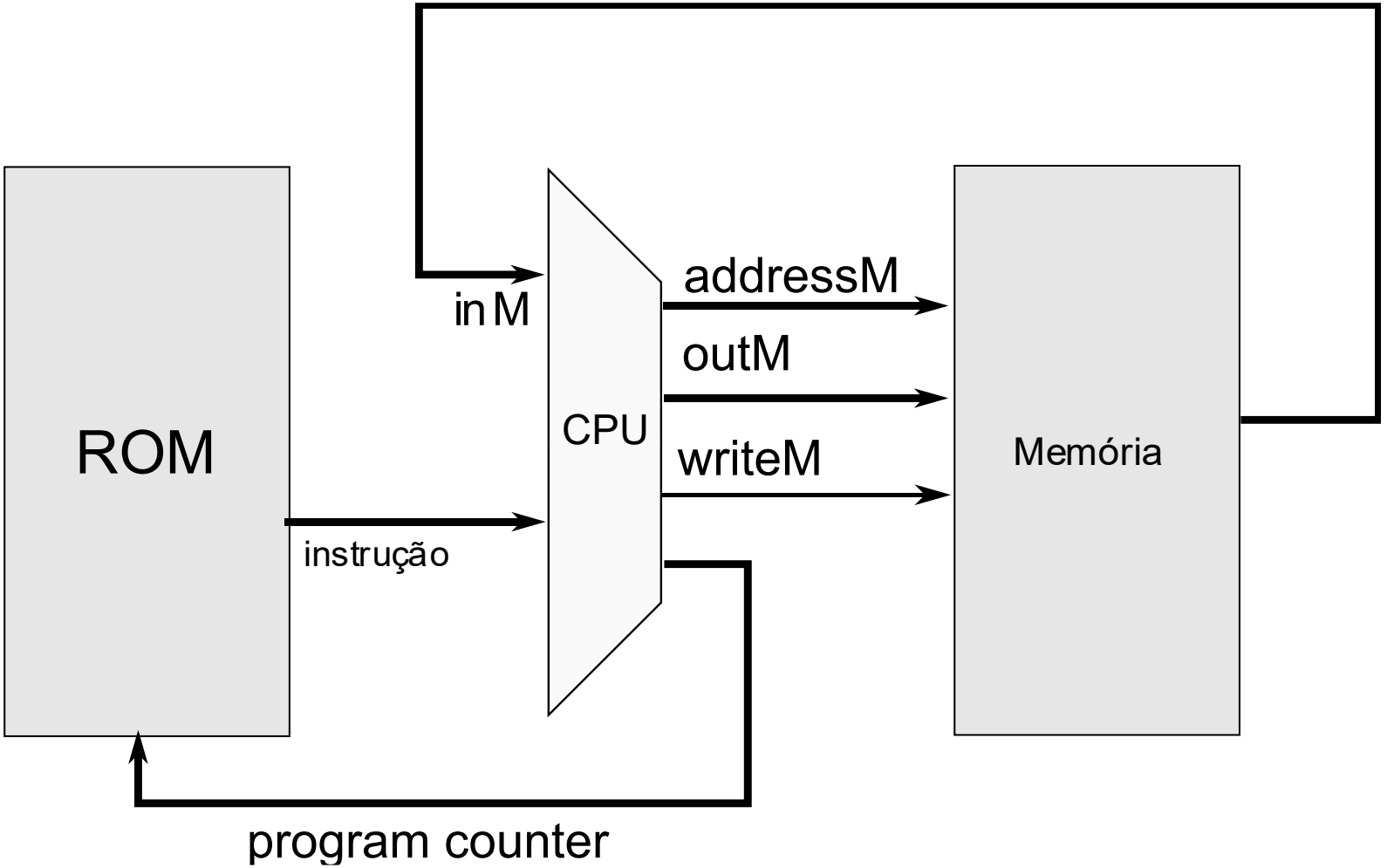
Máquina Analítica

Porém conforme Babbage ia trabalhando em suas máquinas, percebeu que ela poderia ser mais genérica, o que levou ao desenvolvimento da Máquina Analítica que apresenta conceitos usados nos computadores atuais.

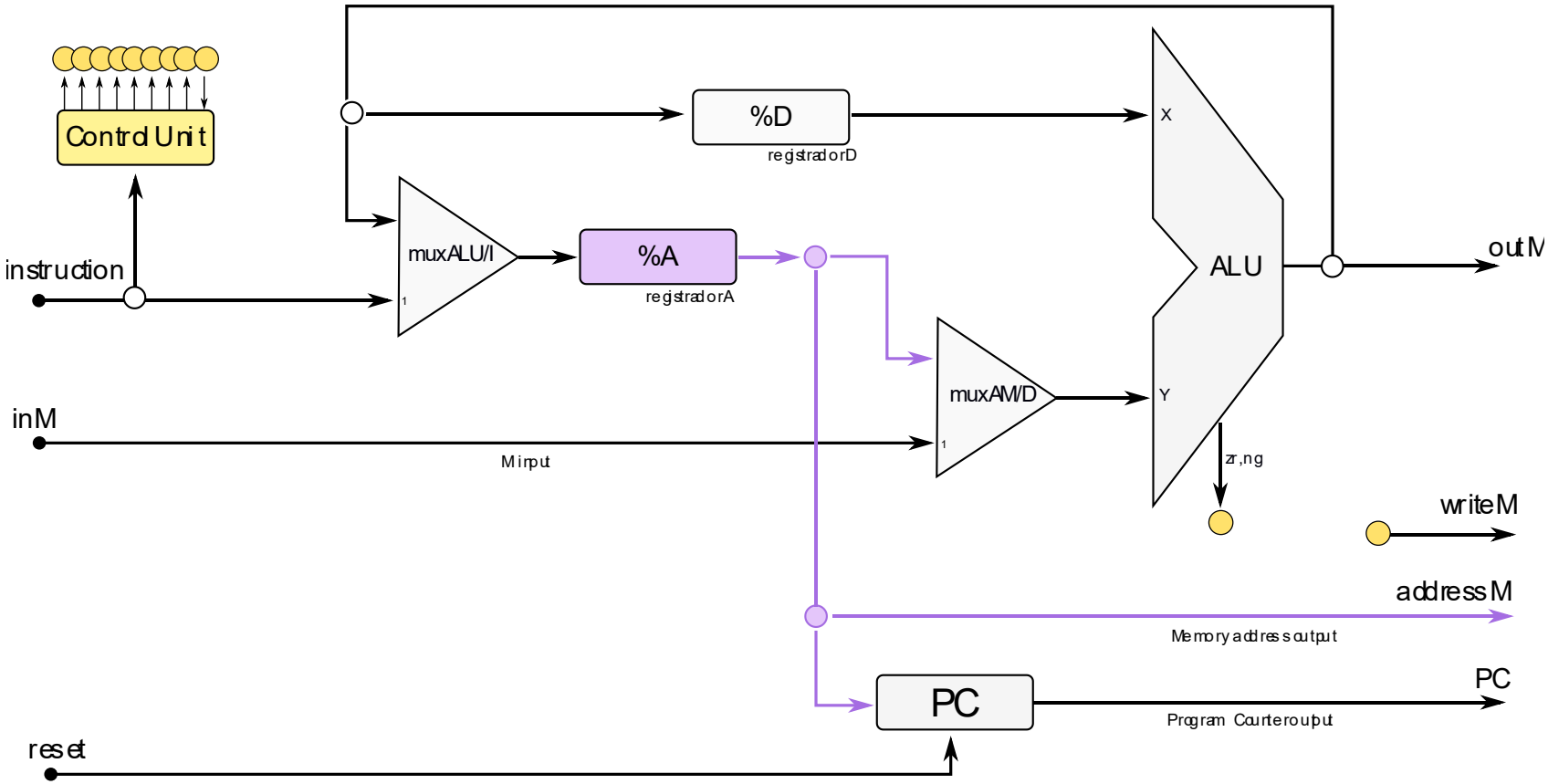


https://www.youtube.com/watch?v=BlbQsKpq3Ak

Z01

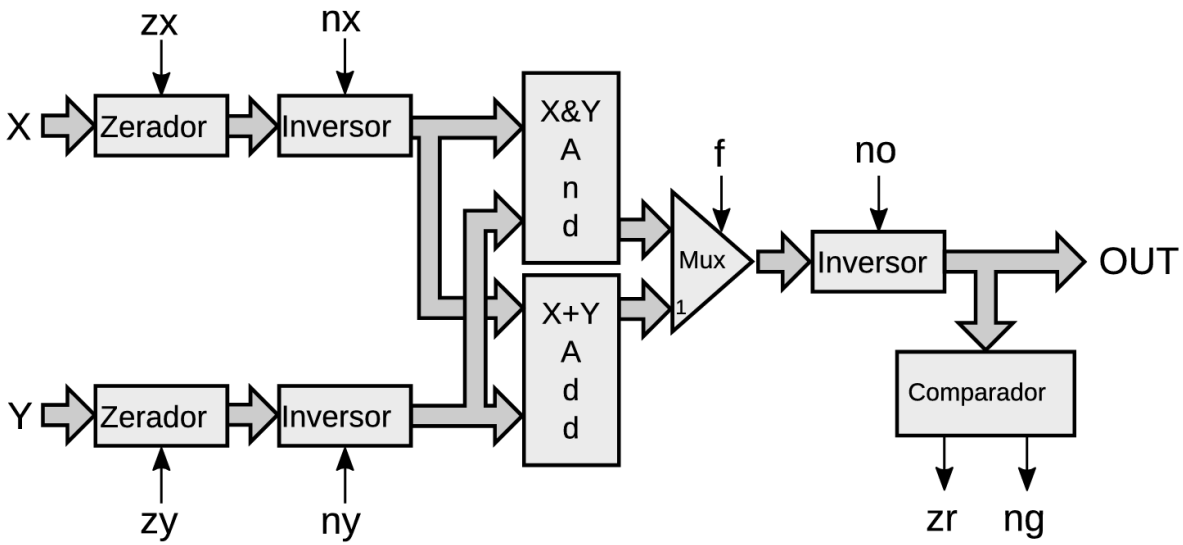

ROM
instrução
CPU
inM
addressM
outM
writeM
Memória
program counter

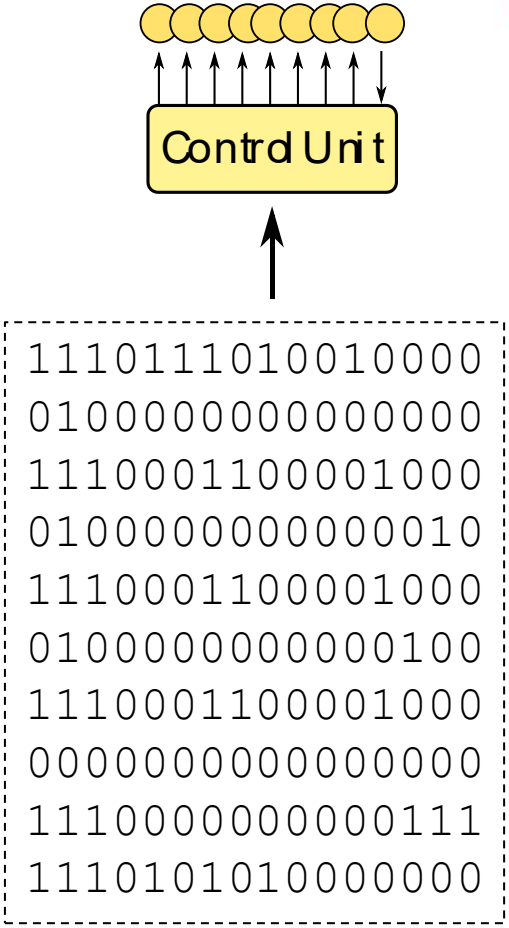
CPU


Contrd Unit
instruction
%D
registradorD
muxALU/I
1
%A
registradorA
X
ALU
Y
outM
muxAM/D
1
inM
M input
zr,ng
writeM
addressM
Memory address output
PC
PC
Program Counter output
reset

Insper

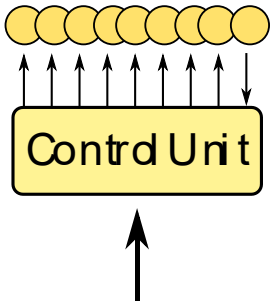
Como programar?





Insper

Como programar?



Instruções do tipo A

```
se bit 17 == 0:
    transfere 16 bits para o registrador A
```

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	X	v	v	v	v	v	v	v	v	v	v	v	v	v	v	v	v

[15:0] : Palavra de 16 bits

Instruções do tipo C

```
se bit 17 == 1:
    executa ação
```

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	r0	c5	c4	c3	c2	c1	c0	0	d2	d1	d0	j2	j1	j0

[13:0] : Indica ação a ser executada pela CPU

Como programar?

Cálculo

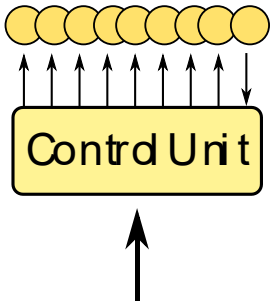
mux		zx	nx	zy	ny	f	no
r0 = 0	**r0 = 1**	**c5**	**c4**	**c3**	**c2**	**c1**	**c0**
0	-	1	0	1	0	1	0
1	-	1	1	1	1	1	1
-1	-	1	1	1	0	1	0
D	-	0	0	1	1	0	0
A	(A)	1	1	0	0	0	0
!D	-	0	0	1	1	0	1
!A	!(A)	1	1	0	0	0	1
-D	-	0	0	1	1	1	1
-A	-(A)	1	1	0	0	1	1
D+1	-	0	1	1	1	1	1
A+1	(A)+1	1	1	0	1	1	1
D-1	-	0	0	1	1	1	0
A-1	(A)-1	1	1	0	0	1	0
D+A	D+(A)	0	0	0	0	1	0
D-A	D-(A)	0	1	0	0	1	1
A-D	(A)-D	0	0	0	1	1	1
D&A	D&(A)	0	0	0	0	0	0
D|A	D|(A)	0	1	0	1	0	1

Destino

	(A)	D	A
Dest	**d2**	**d1**	**d0**
NULL	0	0	0
A	0	0	1
D	0	1	0
(A)	1	0	0
DA	0	1	1
(A)A	1	0	1
(A)D	1	1	0
(A)AD	1	1	1

Jump

	<0	=0	>0
Caso	**j2**	**j1**	**j0**
não	0	0	0
JG	0	0	1
JE	0	1	0
JGE	0	1	1
JL	1	0	0
JNE	1	0	1
JLE	1	1	0
JMP	1	1	1



Insper

Arquivos Assembly

Arquivos Assembly tem a extensão .nasm

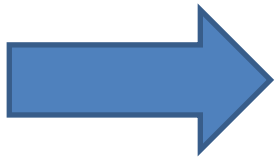
Convenções:

- Comentários são feitos com ponto e vírgula “;”
- Números (constantes) sempre positivos
- Mnemônicos em minúsculo
- Variáveis e Labels em maiúsculo

```
leaw R2, %A
movw $0, (%A)
LOOP:
leaw R1, %A
decw (%A)
movw (%A), %D
leaw END, %A
jl
nop
leaw R0, %A
movw (%A), %D
leaw R2, %A
addw (%A), %D
leaw LOOP, %A
jmp
nop
END:
```

Assembler fazer a conversão

```
movw $-1, %D
leaw 16384, %A
movw %D, (%A)
leaw 16386, %A
movw %D, (%A)
leaw 16388, %A
movw %D, (%A)
leaw 0, %A
jmp
nop
```



```
1110111010010000
0100000000000000
1110001100001000
0100000000000010
1110001100001000
0100000000000100
1110001100001000
0000000000000000
1110000000000111
1110101010000000
```

Assembly

Duas Sintaxes:

AT&T

O formato padrão das instruções:

mnemônico origem, destino

Intel

O formato padrão das instruções:

mnemônico destino, origem

Assembly AT&T

Registradores

Prefixo o sinal de porcentagem '%'.

Exemplo: %A ou %D.

Valores Literais

Os valores tem como prefixo o sinal de cifrão '$'.

Exemplo: $55, $376

Endereçamento de Memória

A memória é referenciada com parêntese em volta do registrador que armazena o endereço.

Exemplo: (%A)

Assembly AT&T

Tamanho dos operadores

Instruções trabalham com diferentes tamanhos de dados, definidas pelo sufixo: b (8 bits), w (16 bits) e l (32 bits).
Exemplo: movw $2000, (%A)

Instruções de Transferência de Controle

As instruções de jump, fazem o fluxo do programa mudar, posições no programa usam marcadores (labels) queterminam com dois pontos (:).
Exemplo: loop:

Principais Instruções 1/3

LEA - Carregamento Efetivo do Endereço (Valor)
Descrição: A instrução *lea* armazena o valor passado no registrador especificado (somente o %A na implementação desenvolvida).
Exemplo: leaw $15,%A => *A = 15*

MOV – Copia Valores
Descrição: A instrução *mov,* copia o valor da primeira posição para a segunda posição (terceira e quarta opcional).
Exemplo: movw (%A),%D => *D = RAM[A]*

ADD - Adição de Inteiros
Descrição: A instrução *add* soma dois valores inteiros e armazena o resultado no terceiro parâmetro (quarto e quinto opcional).
Exemplo: addw (%A), %D, %D => *D = RAM[A]+D*

SUB - Subtração de Inteiros
Descrição: A instrução *sub,* subtrai o segundo valor do primeiro valor e armazena o resultado no terceiro parâmetro (quarto e quinto opcional).
Exemplo: subw %D, (%A), %A => *A = D-RAM[A]*

RSUB - Subtração de Inteiros Reversa
Descrição: A instrução r*sub,* subtrai o primeiro valor do segundo valor e armazena o resultado no terceiro parâmetro (quarto e quinto opcional).
Exemplo: rsubw %D, (%A), %A => *A = RAM[A]-D*

INC - Incrementa Inteiro
Descrição: A instrução *inc,* adiciona um ao valor do registrador ou memória.
Exemplo: incw %D => *D = D+1*

DEC - Decrementa Inteiro
Descrição: A instrução *dec,* diminui um ao valor do registrador ou memória.
Exemplo: decw (%A) => *RAM[A] = RAM[A]-1*

Principais Instruções 2/3

NOT – Negação por Complemento de Um
Descrição: A instrução *not,* inverte o valor de cada bit da série, ou seja, se um bit tem valor 0 fica com 1 e vice-versa.
Exemplo: notw %D => *D = !D*

NEG – Negação por Complemento de Dois
Descrição: A instrução *neg,* faz o valor ficar negativo, ou seja, um valor de x fica -x.
Exemplo: negw %A => *A = -A*

AND – Operador E (and)
Descrição: A instrução *and* faz o operador lógico E (and).
Exemplo: andw %A, %D => *D = A&D*

OR – Operador OU (or)
Descrição: A instrução *or* faz o operador lógico Ou (or).
Exemplo: Assembly: orw %D, (%A) => *RAM[A] = RAM[A]|D*

NOP – Não faz nada (Not Operation)
Descrição: A instrução *nop* não faz nada, usado para pular um ciclo de execução.
Exemplo: Assembly: nop

JMP – Jump
Descrição: A instrução *jmp* faz um salto de execução para o endereço armazenado em %A.
Exemplo: Assembly: jmp

JE – Salta Execução se Igual a Zero
Descrição: A instrução *je* faz um salto de execução para o endereço armazenado em %A, se o valor de D% for igual a zero.
Exemplo: Assembly: je

Principais Instruções 3/3

JNE – Salta Execução se Igual a Zero

Descrição: A instrução *jne* faz um salto de execução para o endereço armazenado em %A, se o valor de D% for diferente de zero.

Exemplo: Assembly: jne

JG – Salta Execução se Maior que Zero

Descrição: A instrução *jg* faz um salto de execução para o endereço armazenado em %A, se o valor de D% for maior que zero.

Exemplo: Assembly: jg

JGE – Salta Execução se Maior Igual a Zero

Descrição: A instrução *jge* faz um salto de execução para o endereço armazenado em %A, se o valor de D% for maior ou igual a zero.

Exemplo: Assembly: jge

JL – Salta Execução se Menor que Zero

Descrição: A instrução *jl* faz um salto de execução para o endereço armazenado em %A, se o valor de D% for menor que zero.

Exemplo: Assembly: jl

JLE – Salta Execução se Menor Igual a Zero

Descrição: A instrução *jle* faz um salto de execução para o endereço armazenado em %A, se o valor de D% for menor ou igual a zero.

Exemplo: Assembly: jle

Insper

www.insper.edu.br