PCS3616

Programação de Sistemas

(Sistemas de Programação)

Semana 3, Aula 5

Introdução

Máquina de von Neumann

Escola Politécnica da Universidade de São Paulo

Observações sobre a Máquina de Turing

- Uma **Máquina de Turing** deve ser vista como um computador com um único programa fixo. Para alterar o programa, é preciso construir outra máquina.
- Algumas Máquinas de Turing servem como **reconhecedores de linguagens**, outras podem **computar funções**.
- É possível construir uma **Máquina de Turing Universal**, a qual simula a computação de Máquinas de Turing arbitrárias sobre entradas arbitrárias.
- Eliminadas suas limitações de recursos, um **computador moderno** pode ser visto como um dispositivo similar à Máquina de Turing Universal.

Problemas Práticos da Máquina de Turing

- A Máquina de Turing se apresenta, portanto, através de um formalismo poderoso, com fita infinita e apenas quatro operações triviais: ler, gravar, avançar e recuar.

- Isso faz dela um dispositivo detalhista que oferece uma **visão microscópica** da solução do problema que pretende resolver, não permitindo ao usuário usar abstrações mais expressivas.

- É possível definir bibliotecas de Máquinas de Turing, cada qual executando uma função, de tal forma que uma solução de problema possa ser definida pela combinação adequada de Máquinas de Turing.

- Embora a Máquina de Turing Universal permita uma espécie de programação (programa armazenado), o seu código é extenso e a sua velocidade final de execução, muito baixa (desempenho).

A ideia da Máquina de von Neumann

- O **Modelo de von Neumann** procura oferecer uma alternativa prática, disponibilizando ações mais poderosas e ágeis em seu repertório de operações.

- Isso viabiliza, para os mesmos programas, codificações muito mais expressivas, compactas e eficientes.

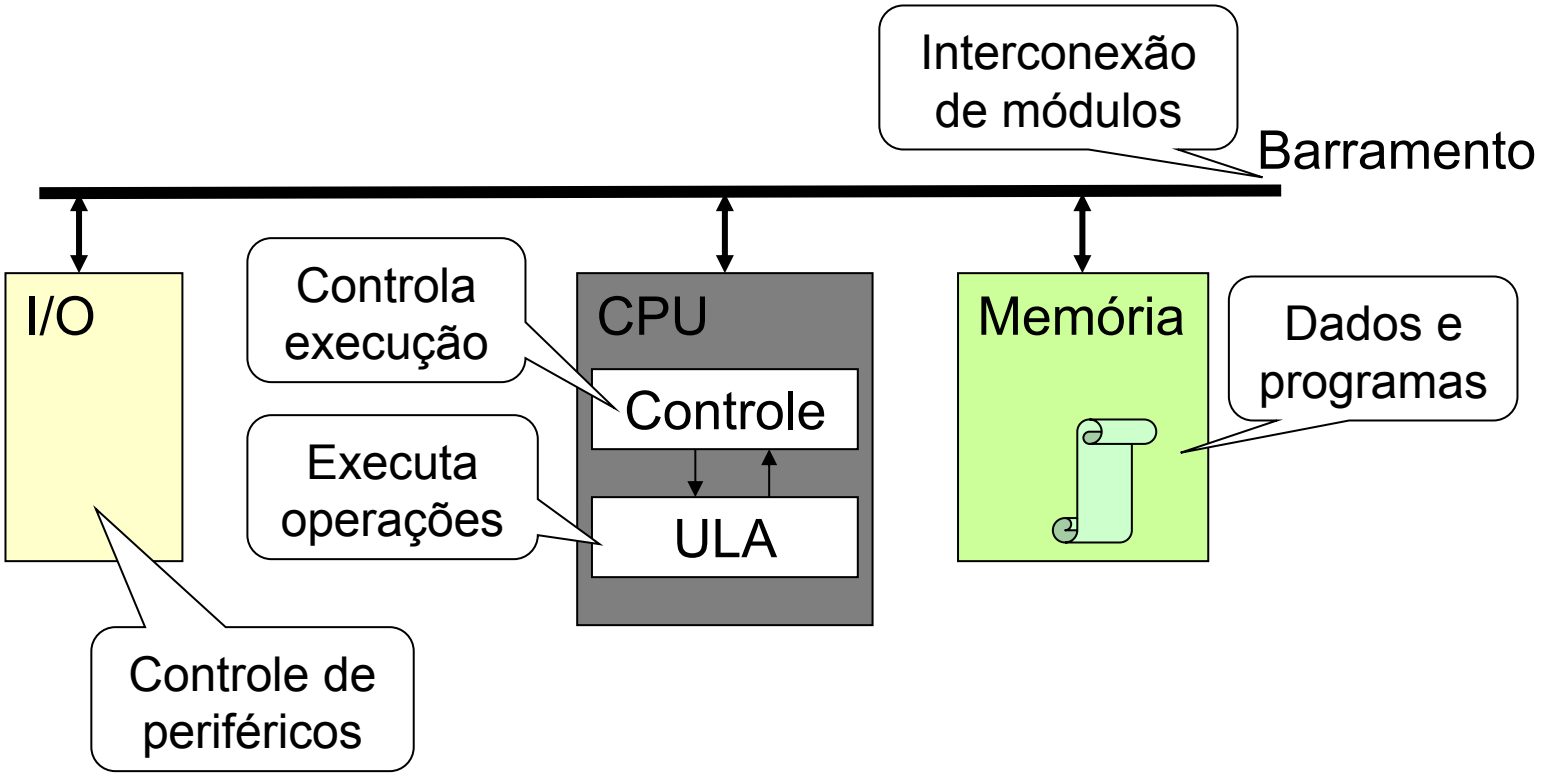
John von Neumann (1903-1957)

A ideia da Máquina de von Neumann

- Para isso, a Máquina de von Neumann utiliza:
  - **Memória endereçável**, usando acesso aleatório.
  - **Programa armazenado** na memória, para definir diretamente a função corrente da máquina (ao invés da MEF).
  - **Dados** representados na memória (ao invés da fita).
  - Codificação numérica **binária** em lugar da unária.
  - **Instruções variadas e expressivas** para a realização de operações básicas muito frequentes (ao invés de sub máquinas específicas).
  - **Maior flexibilidade** para o usuário, permitindo operações de entrada e saída, comunicação física com o mundo real e controle dos modos de operação da máquina.
  - Capacidade de comunicação com dispositivos de **entrada e saída (E/S)**

A ideia da Máquina de von Neumann

- Desenho esquemático de uma Máquina de von Neumann:
  - Os detalhes serão deixados para a disciplina de Organização de Sistemas Digitais



Funcionamento de um Simulador

Deve-se distinguir entre dois conceitos independentes na lógica de um simulador:

- **Comandos de controle do simulador**: esta parte do simulador independe da arquitetura do computador que se está simulando. Sua função é orientar a operação do programa simulador e permitir ao usuário observar e alterar o conteúdo dos componentes do processador simulado
  - São executados via **linha de comando**

- **Execução das instruções do processador simulado**: esta parte do simulador depende fortemente da arquitetura da máquina simulada. É ela que implementa um modelo da máquina simulada no nível de granularidade mais conveniente desejado em cada caso.
  - São executados na forma de um **programa em linguagem de máquina**

Elementos da Arquitetura a Simular

- Nesta disciplina pretende-se simular um **processador muito simples**, porém estruturalmente similar aos disponíveis de fato.
  - Processadores reais costumam incluir mais instruções, registradores adicionais, vários níveis de memória, etc..

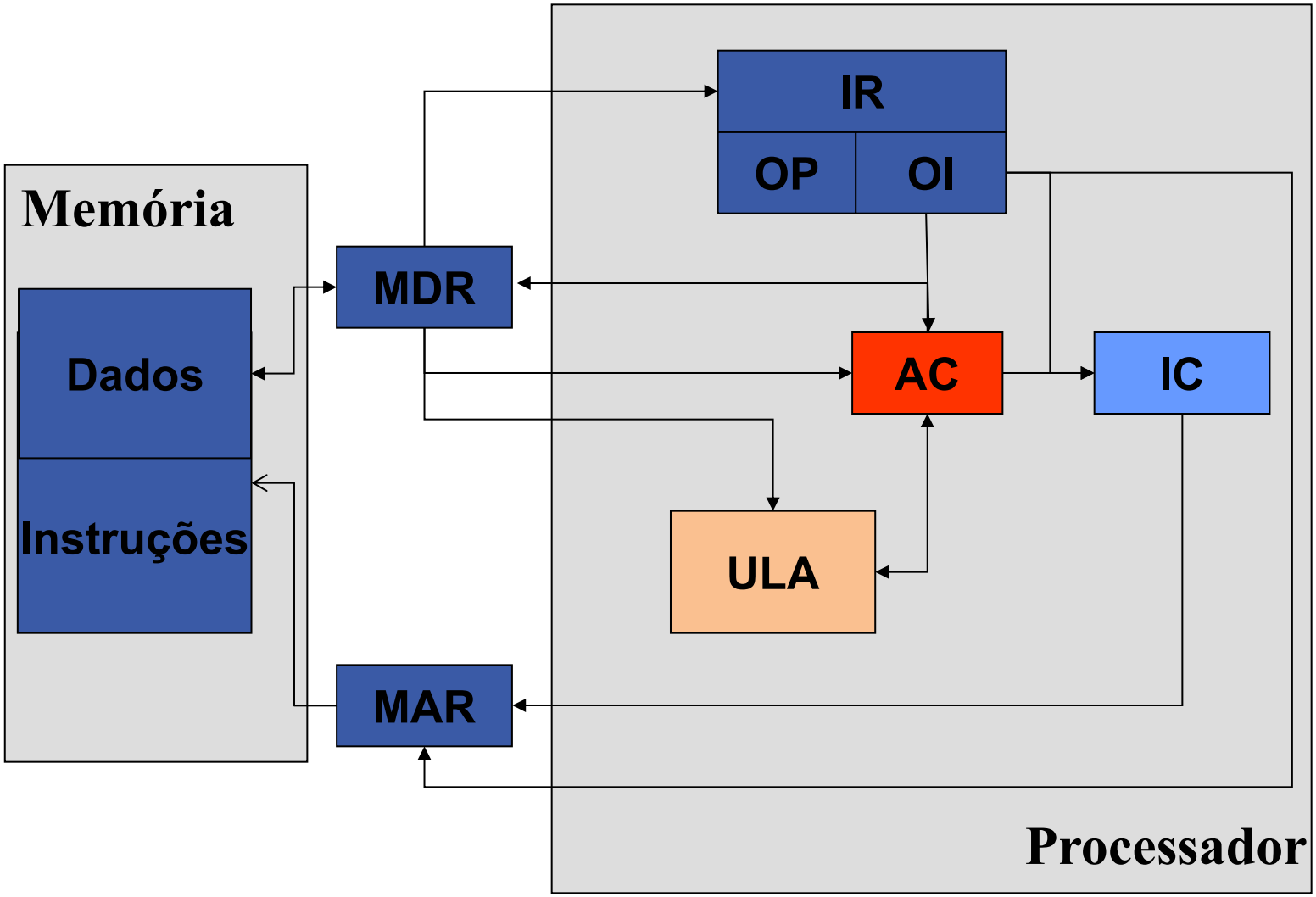
- O processador tem um conjunto de elementos físicos de armazenamento de informações:
  - **Memória Principal:** para armazenar programas e dados.
  - **Acumulador (AC):** funciona como área de trabalho, para a execução de operações aritméticas e lógicas.
  - Outros **registradores auxiliares**: empregados em diversas operações intermediárias no processamento dos programas.

- O conjunto de dados neles contidos em cada instante constitui o **estado instantâneo** do processamento.

Elementos da Arquitetura a Simular

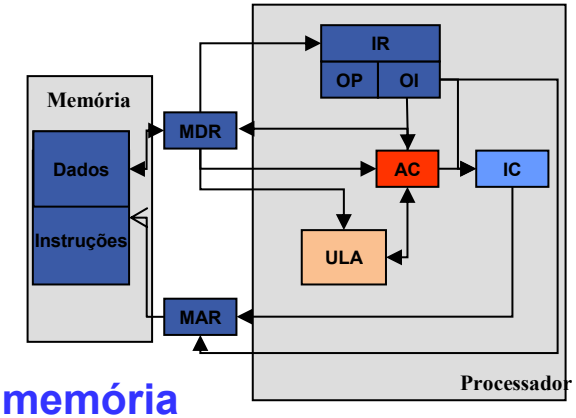
- Os **Registradores Auxiliares** são:
  - **Registrador de Dados da Memória (MDR)** – serve como ponte para os dados que trafegam entre a memória e os outros elementos da máquina.
  - **Registrador de Endereço da Memória (MAR)** – indica qual é a origem ou o destino, na memória principal, dos dados contidos no registrador de dados da memória.
  - **Registrador de Endereço de Instrução (IC)** – indica em cada instante qual será a próxima instrução a ser executada pelo processador.
  - **Registrador de Instrução (IR)** – contém a instrução em execução
    - **Código de Operação (OP)** – parte do registrador de instrução que identifica a instrução que está sendo executada
    - **Operando da Instrução (OI)** – complementa a instrução indicando o dado ou o endereço sobre o qual ela deve agir.

Elementos da Arquitetura a Simular

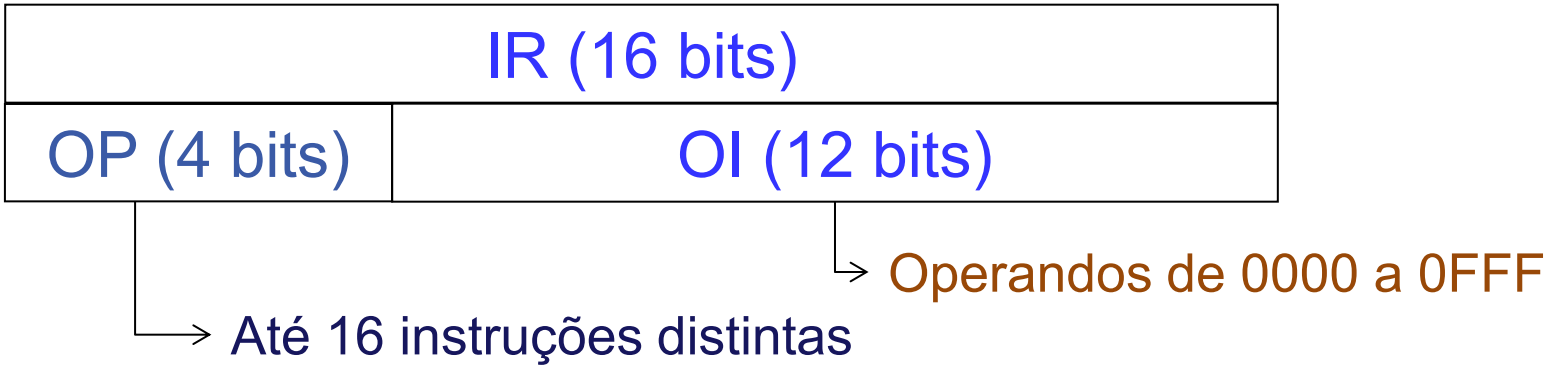


Conjunto de registradores da Máquina de von Neumann (MVN)

Arquitetura de 16 bits



MAR	Registrador de endereço de memória
MDR	Registrador de dados da memória
IC	Registrador de endereço de instrução
IR	Registrador de instrução
OP	Registrador de código de operação
OI	Registrador de operando de instrução
AC	Acumulador



Conjunto de instruções da Máquina de von Neumann (MVN)

Código (hexa)	Instrução	Operando
0	Desvio incondicional	endereço do desvio
1	Desvio se acumulador é zero	endereço do desvio
2	Desvio se acumulador é negativo	endereço do desvio
3	Deposita uma constante no acumulador	constante relativa de 12 bits
4	Soma	endereço da parcela
5	Subtração	endereço do subtraendo
6	Multiplicação	endereço do multiplicador
7	Divisão	endereço do divisor
8	Memória para acumulador	endereço-origem do dado
9	Acumulador para memória	endereço-destino do dado
A	Desvio para subprograma (função)	endereço do subprograma
B	Retorno de subprograma (função)	endereço do resultado
C	Parada	endereço do desvio
D	Entrada	dispositivo de e/s (*)
E	Saída	dispositivo de e/s (*)
F	Chamada de supervisor	constante (**)

(*) ver slides mais adiante
(**) por ora, este operando (tipo da chamada) é irrelevante, e esta instrução nada faz.

Referências Bibliográficas

Bryant R. E. and O'Hallaron, D. R. ***Computer Systems: A Programmer's Perspective***, 2010.

DONOVAN, J. ***Systems Programming***, 1972.

Leitura complementar:

UM SIMULADOR-INTERPRETADOR PARA A LINGUAGEM DE MÁQUINA DO PATINHO FEIO.

(João José Neto, Aspectos do Projeto de Software de um Minicomputador, Dissertação de Mestrado, EPUSP, S. Paulo, 1975, cap.3)

Transparências extraídas e alteradas de:

José Neto, J., Sichman, J. S., Silva, P.S.M., Rocha, R.L.A. ***Material didático da disciplina PCS 2024 – Laboratório de Fundamentos da Engenharia de Computação***, PCS/EPUSP, São Paulo, SP. 2005-2015.

PCS3616

Programação de Sistemas

(Sistemas de Programação)

Semana 3, Aula 6

Introdução

Máquina de von Neumann

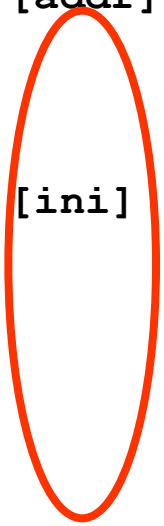
Escola Politécnica da Universidade de São Paulo

Comandos de Controle do Simulador

```
          Escola Politécnica da Universidade de São Paulo
        PCS2302/PCS2024 - Simulador da Máquina de von Neumann
      MVN versão 4.5 (Agosto/2011) - Todos os direitos reservados

 COMANDO   PARÂMETROS            OPERAÇÃO
-------------------------------------------------------------------
    i                            Re-inicializa MVN
    p     [arq]                  Carrega programa para a memória
    r     [addr] [regs]          Executa programa
    b                            Ativa/Desativa modo Debug
    s                            Manipula dispositivos de I/O
    g                            Lista conteúdo dos registradores
    m     [ini] [fim] [arq]      Lista conteúdo da memória
    h                            Ajuda
    x                            Finaliza MVN e terminal

>
```



[run] – Obtenção e Decodificação

Comando r (“run”) - Serve para promover a execução do programa, conforme o modo de operação: contínua ou uma instrução por vez

1) **Determinação da Instrução a Executar**

2) **Fase de Obtenção da Instrução**

• Obter na memória, no endereço contido no registrador de Endereço de Instrução, o código da instrução desejada

3) **Fase de Decodificação da Instrução**

• Decompor a instrução em duas partes: o código da instrução e o seu operando, depositando essas partes nos registradores de instrução e de operando, respectivamente.

• Selecionar, com base no conteúdo do registrador de instrução, um procedimento de execução dentre os disponíveis no repertório do simulador (passo 4).

[run] – Execução de instrução (1)

4) **Fase de Execução da Instrução**

- Executar o procedimento selecionado em 3, usando como operando o conteúdo do registrador de operando, preenchido anteriormente.

4.1) Execução da instrução (decodificada em 3)

– De acordo com o código da instrução a executar (contido no registrador de instrução), executar os procedimentos de simulação correspondentes (detalhados adiante)

4.2) Acerto do registrador de Endereço de Instrução

– Caso a instrução executada não seja de desvio, incrementar o registrador de Endereço de Instrução a executar. Caso contrário, o procedimento de execução da instrução já terá atualizado convenientemente tal informação.

[run] – Execução de instrução (2)

- Obs.: Sistema de **numeração e aritmética** adotada: Binário, em complemento de dois
  - representa inteiros e executa operações em 16 bits.
  - o bit mais à esquerda é o bit de sinal (1 = negativo)

Registrador de instrução = 0 (desvio incondicional)

- modifica o conteúdo do registrador de Endereço de Instrução (**IC**) armazenando nele o conteúdo do registrador de operando (**OI**)

  IC := OI

Registrador de instrução = 1 (desvio se acumulador é zero)

- se o conteúdo do acumulador for zero, então modifica o conteúdo do registrador de Endereço de Instrução (**IC**), armazenando nele o conteúdo do registrador de operando (**OI**)

  Se AC = 0 então IC := OI

  senão IC := IC + 1

[run] – Execução de instrução (3)

Registrador de instrução = 2 (desvio se negativo)

- se o conteúdo do acumulador (**AC**) for negativo, isto é, se o bit mais significativo for 1, então modifica o conteúdo do registrador de Endereço de Instrução (**IC**) armazenando nele o conteúdo do registrador de operando (**OI**)

  Se AC < 0 então IC := OI

  senão IC := IC + 1

Registrador de instrução = 3 (constante para acumulador)

- Armazena no acumulador (**AC**) o número relativo de 12 bits contido no registrador de operando (**OI**), estendendo seu bit mais significativo (bit de sinal) para completar os 16 bits do acumulador

  AC := OI

  IC := IC +1

[run] – Execução de instrução (4)

Registrador de instrução = 4 (soma)

- Soma ao conteúdo do acumulador (**AC**) o conteúdo da posição de memória indicada pelo registrador de operando MEM[**OI**]
- Guarda o resultado no acumulador

AC := AC + MEM[OI]

IC := IC + 1

Registrador de instrução = 5 (subtração)

- Subtrai do conteúdo do acumulador (**AC**) o conteúdo da posição de memória indicada pelo registrador de operando MEM[**OI**]
- Guarda o resultado no acumulador

AC := AC - MEM[OI]

IC := IC + 1

[run] – Execução de instrução (5)

Registrador de instrução = 6 (multiplicação)

- Multiplica o conteúdo do acumulador (**AC**) pelo conteúdo da posição de memória indicada pelo registrador de operando MEM[**OI**]
- Guarda o resultado no acumulador

    AC := AC * MEM[OI]

    IC := IC + 1

Registrador de instrução = 7 (divisão inteira)

- Dividir o conteúdo do acumulador (**AC**) pelo conteúdo da posição de memória indicada pelo registrador de operando MEM[**OI**]
- Guarda a parte inteira do resultado no acumulador

    AC := int (AC / MEM[OI])

    IC := IC + 1

[run] – Execução de instrução (6)

Registrador de instrução = 8 (memória para acumulador)

- Armazena no acumulador (**AC**) o conteúdo da posição de memória cujo endereço é o conteúdo do registrador de operando MEM[**OI**]

  AC := MEM[OI]

  IC := IC + 1

Registrador de instrução = 9 (acumulador para memória)

- Guarda o conteúdo do acumulador (**AC**) na posição de memória indicada pelo registrador de operando MEM[**OI**]

  MEM[OI] := AC

  IC := IC + 1

[run] – Execução de instrução (7)

Registrador de instrução = A (desvio para subprograma)

- Armazena o conteúdo do registrador de Endereço de Instrução (**IC**), incrementado de uma unidade, na posição de memória apontada pelo registrador de operando MEM[**OI**]
- Armazena no registrador de Endereço de Instrução (**IC**) o conteúdo do registrador de operando incrementado de uma unidade (**OI**)

```
MEM[OI] := IC + 1
IC := OI + 1
```

Registrador de instrução = B (retorno de subprograma)

- Armazena no registrador de Endereço de Instrução (**IC**) o conteúdo que está na posição de memória apontada pelo registrador de operando MEM[**OI**]

```
IC := MEM[OI]
```

[run] – Execução de instrução (8)

Registrador de instrução = C (stop)

- Modifica o conteúdo do registrador de Endereço de Instrução (**IC**) armazenando nele o conteúdo do registrador de operando (**OI**)

  IC := OI

Registrador de instrução = D (input)

- Aciona o dispositivo indicado, fazendo a leitura de dados do mesmo
- Transfere dado para o acumulador

  (solicita dado do dispositivo)

  AC := dado de entrada

  IC := IC + 1

[run] – Execução de instrução (9)

Registrador de instrução = E (output)

- Aciona o dispositivo indicado
- Transfere o conteúdo do acumulador (**AC**) para o dispositivo, esperando que este termine de executar a operação de gravação

    dado de saída := AC

    (aciona dispositivo)

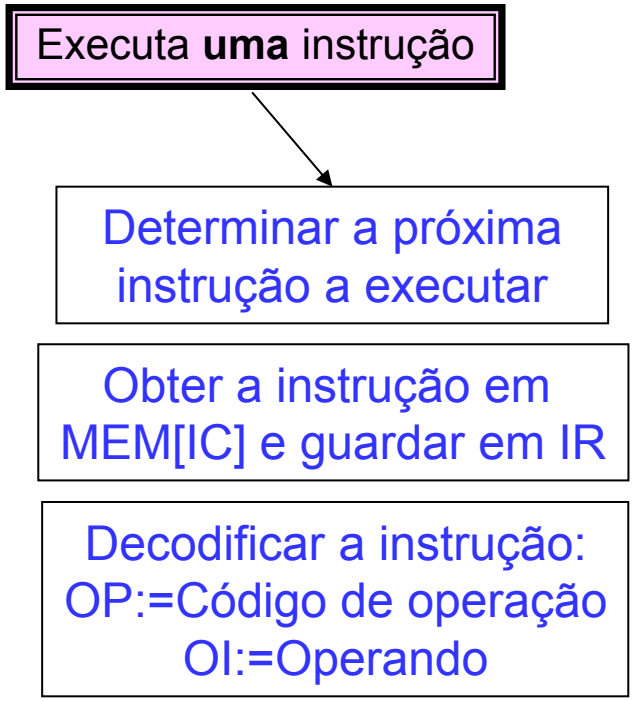
    IC := IC + 1

Registrador de instrução = F (supervisor call)

(não implementada: por enquanto esta instrução não faz nada)

    IC := IC + 1

Diagrama de fluxo do Interpretador [detalhamento da execução]



OP (hexa)	Ação a executar
0	IC:=OI
1	Se AC=0 então IC:=OI senão IC:=IC+1
2	Se AC<0 então IC:=OI senão IC:=IC+1
3	AC:=OI ; IC:=IC+1
4	AC:=AC+MEM[OI] ; IC:=IC+1
5	AC:=AC-MEM[OI] ; IC:=IC+1
6	AC:=AC*MEM[OI] ; IC:=IC+1
7	AC:=int(AC/MEM[OI]) ; IC:=IC+1
8	AC:=MEM[OI] ; IC:=IC+1
9	MEM[OI]:=AC ; IC:=IC+1
A	MEM[OI]:=IC+1; IC:=OI+1
B	IC:=MEM[OI]
C	IC:=OI
D	aguarda; AC:= dado de entrada; IC:=IC+1
E	dado de saída := AC ; aguarda ; IC:=IC+1
F	(nada faz por ora) ; IC:=IC+1

Conjunto de registradores da Máquina de von Neumann (MVN)

Operações de Entrada e Saída

OP	Tipo	Dispositivo

OP — D (entrada) ou E (saída)

Tipo — Tipos de dispositivo:

- 0 = Teclado
- 1 = Monitor
- 2 = Impressora
- 3 = Disco

Dispositivo — Identificação do dispositivo. Pode-se ter vários tipos de dispositivo, ou unidades lógicas (LU). No caso do disco, um arquivo é considerado uma unidade lógica.

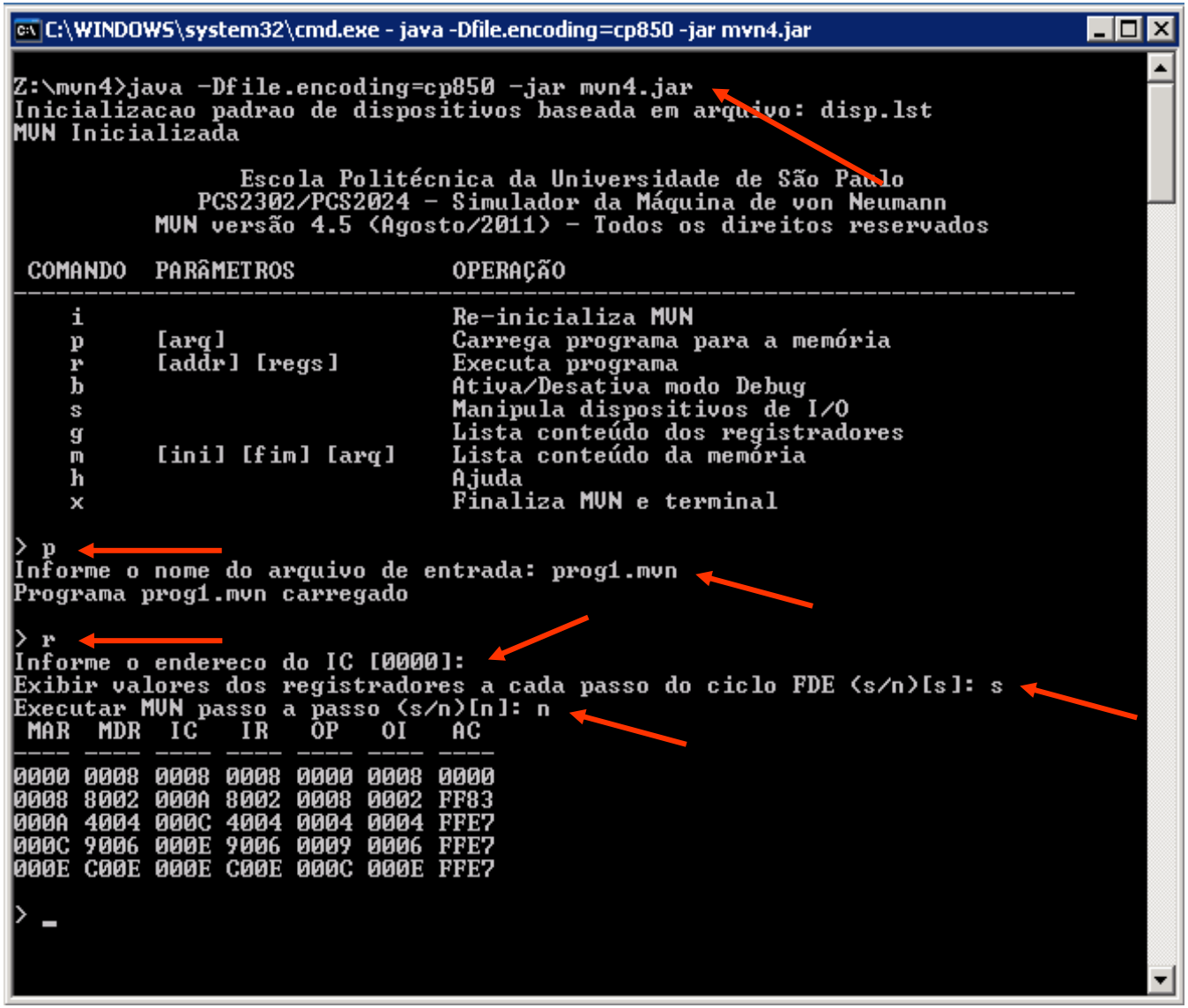
Pode-se ter, portanto, até 16 tipos de dispositivos e, cada um, pode ter até 256 unidades lógicas.

Exemplo de Programa – Prog1 (1)

- Problema: Somar o valor de duas variáveis iniciadas com os valores -125_{10} e 100_{10}, colocando o resultado em outra variável.

```
; prog1.mvn
; Soma os valores de duas posições de memória e guarda o
; resultado em outra posição de memória, parando na
; instrução final.
0000 0008  ; Ponto de entrada: salto para as instruções
; Constantes do programa
0002 FF83  ; A = 0xFF83 (-125)
0004 0064  ; B = 0x0064 (100)
; Variáveis do programa
0006 0000  ; RESULTADO deverá ser 0xFFE7 (-25)
; Instruções do programa
0008 8002  ; Carrega o valor de A no acumulador
000A 4004  ; Adiciona B ao conteúdo do acumulador
000C 9006  ; Armazena o resultado em RESULTADO
000E C00E  ; Para em 0x000E
```

Execução de Programa – Prog1 (2)



Exemplo de Programa – Prog2 (1)

- Problema: Desenvolver uma sub-rotina que subtrai dois inteiros. Os valores dos argumentos estão armazenados em duas variáveis do programa principal. O resultado é armazenado em uma variável do programa principal.

```
; prog2.mvn
; Programa de ilustração para chamada de sub-rotina
;   int subtrair(int x, int y) {
;     return x - y;
;   }
;
0000 0010  ; Ponto de entrada: pulo para as instruções
; Constantes do programa
0002 0010  ; A = 0x0010 (16)
0004 0064  ; B = 0x0064 (100)
; Variáveis do programa
0006 0000  ; RESULTADO de subtrair() = 0xFFAC (-84)
```

Exemplo de Programa – Prog2 (2)

```
; Programa principal
; Chamando SUBTRAIR(A, B)
0010 8002  ; Carrega o conteúdo de A no acumulador
0012 903C  ; Armazena no parâmetro X
0014 8004  ; Carrega o conteúdo de B
0016 903E  ; Armazena no parâmetro Y
0018 A040  ; Chama a sub-rotina SUBTRAIR
001A 9006  ; Armazena o resultado em RESULTADO
001C C01C  ; Para em 0x01C
;
; Sub-rotina SUBTRAIR
; Parâmetros formais
003C 0000  ; X
003E 0000  ; Y
; Corpo da sub-rotina
0040 0000  ; Endereço de retorno
0042 803C  ; Carrega o conteúdo de X
0044 503E  ; Subtrai Y, resultado no ACUMULADOR
0046 B040  ; Retorna para o endereço contido em 0x040
```

Execução de Programa – Prog2 (3)

```
> p
Informe o nome do arquivo de entrada: prog2.mvn
Programa prog2.mvn carregado

> r
Informe o endereco do IC [0000]: 0000
Exibir valores dos registradores a cada passo do ciclo FDE (s/n)[s]: s
Executar MVN passo a passo (s/n)[n]: n
 MAR  MDR  IC   IR   OP   OI   AC
---- ---- ---- ---- ---- ---- ----
0000 0010 0010 0010 0000 0010 0000
0010 8002 0012 8002 0008 0002 0010
0012 903C 0014 903C 0009 003C 0010
0014 8004 0016 8004 0008 0004 0064
0016 903E 0018 903E 0009 003E 0064
0018 A040 0042 A040 000A 0040 0064
0042 803C 0044 803C 0008 003C 0010
0044 503E 0046 503E 0005 003E FFAC
0046 B040 001A B040 000B 0040 FFAC
001A 9006 001C 9006 0009 0006 FFAC
001C C01C 001C C01C 000C 001C FFAC

>
```

Bibliografia (Programação de Sistemas)

Relíquias Preciosas

Barron, D. W. ***Assemblers and Loaders*** (3rd. ed.) MacDonald/Elsevier, 1978

Beck, L. L. ***System Software - An Introduction to Systems Programming*** Addison-Wesley, 1996

Calingaert, P. ***Assemblers, Compilers and Program Translation*** Computer Science Press, 1979

Donovan, J. J. ***Systems Programming*** McGraw-Hill, 1972

Duncan, F.G. ***Microprocessor Programming and Software Development*** Prentice Hall, 1979.

Freeman, P. ***Software System Principles*** SRA, 1975

Gear, C. W. ***Computer Organization and Programming (3rd. ed.)*** McGraw-Hill, 1980

Graham, R. M. ***Principles of Systems Programming*** John Wiley & Sons, 1975

Gust, P. ***Introduction to Machine and Assembly Language Programming*** Prentice Hall, 1985

Maginnis, J. B. ***Elements of Compiler Construction*** Appleton-Century-Crofts, Meredith Co., 1972

Presser, L. and White, J. R. ***Linkers and Loaders*** ACM Comp. Surveys, vol. 4, n. 3, pp. 149-168, 1972

Rosen, S. (ed.) ***Programming Systems and Languages*** McGraw-Hill, 1967

Tseng, V. (ed.) ***Microprocessor Development and Development Systems*** McGraw-Hill, 1982

Ullman, J. D. ***Fundamental Concepts of Programming Systems*** Addison-Wesley, 1976

Wegner, P. ***Progr. Languages, Inf. Structures and Machine Organization*** McGraw-Hill, 1968.

Welsh, J. and McKeag, M. ***Structured System Programming*** Prentice-Hall,1980

Referências Bibliográficas

Bryant R. E. and O'Hallaron, D. R. ***Computer Systems: A Programmer's Perspective***, 2010.

DONOVAN, J. ***Systems Programming***, 1972.

Leitura complementar:

UM SIMULADOR-INTERPRETADOR PARA A LINGUAGEM DE MÁQUINA DO PATINHO FEIO.

(João José Neto, Aspectos do Projeto de Software de um Minicomputador, Dissertação de Mestrado, EPUSP, S. Paulo, 1975, cap.3)

Transparências extraídas e alteradas de:

José Neto, J., Sichman, J. S., Silva, P.S.M., Rocha, R.L.A. ***Material didático da disciplina PCS 2024 – Laboratório de Fundamentos da Engenharia de Computação***, PCS/EPUSP, São Paulo, SP. 2005-2015.