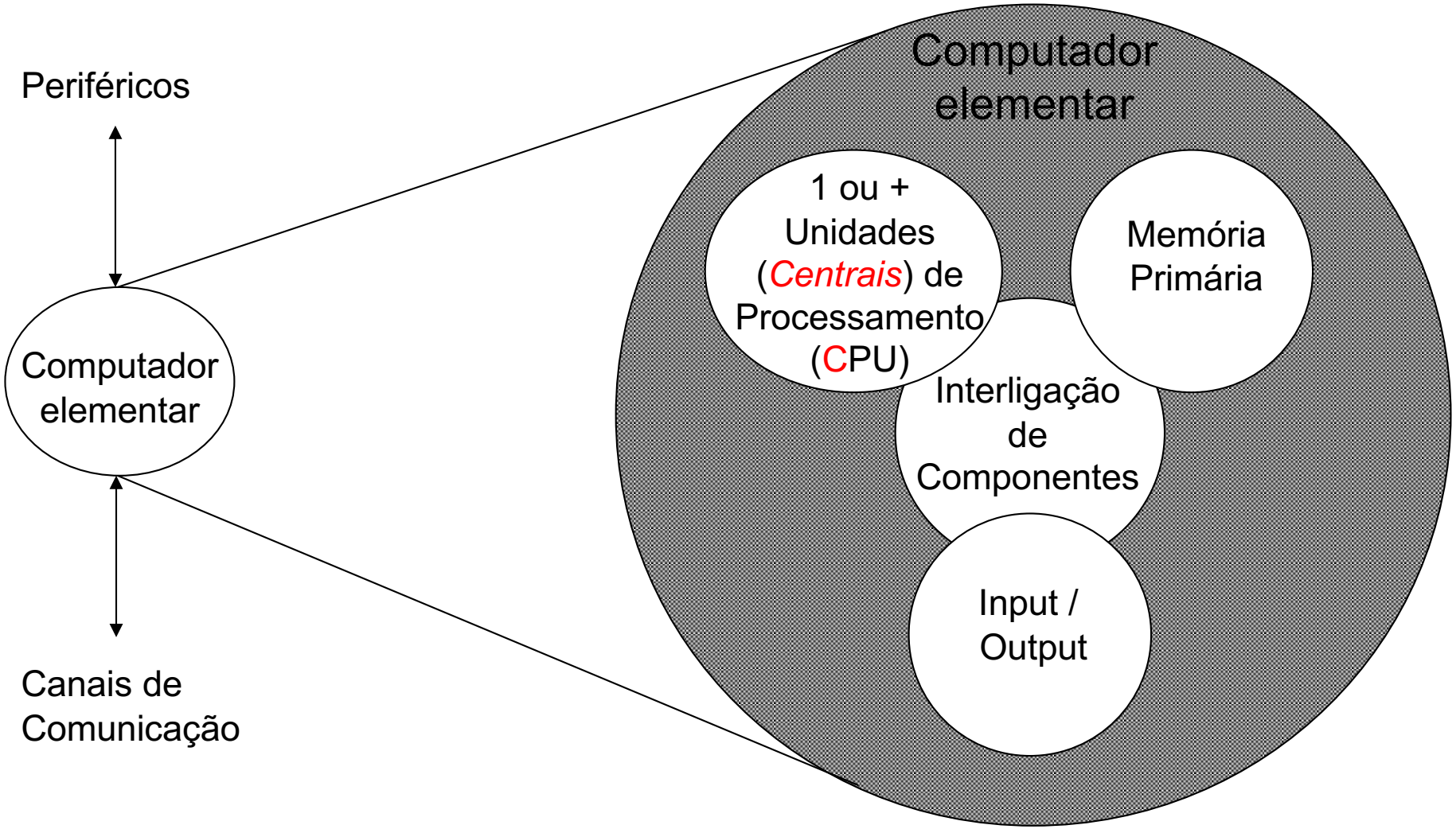
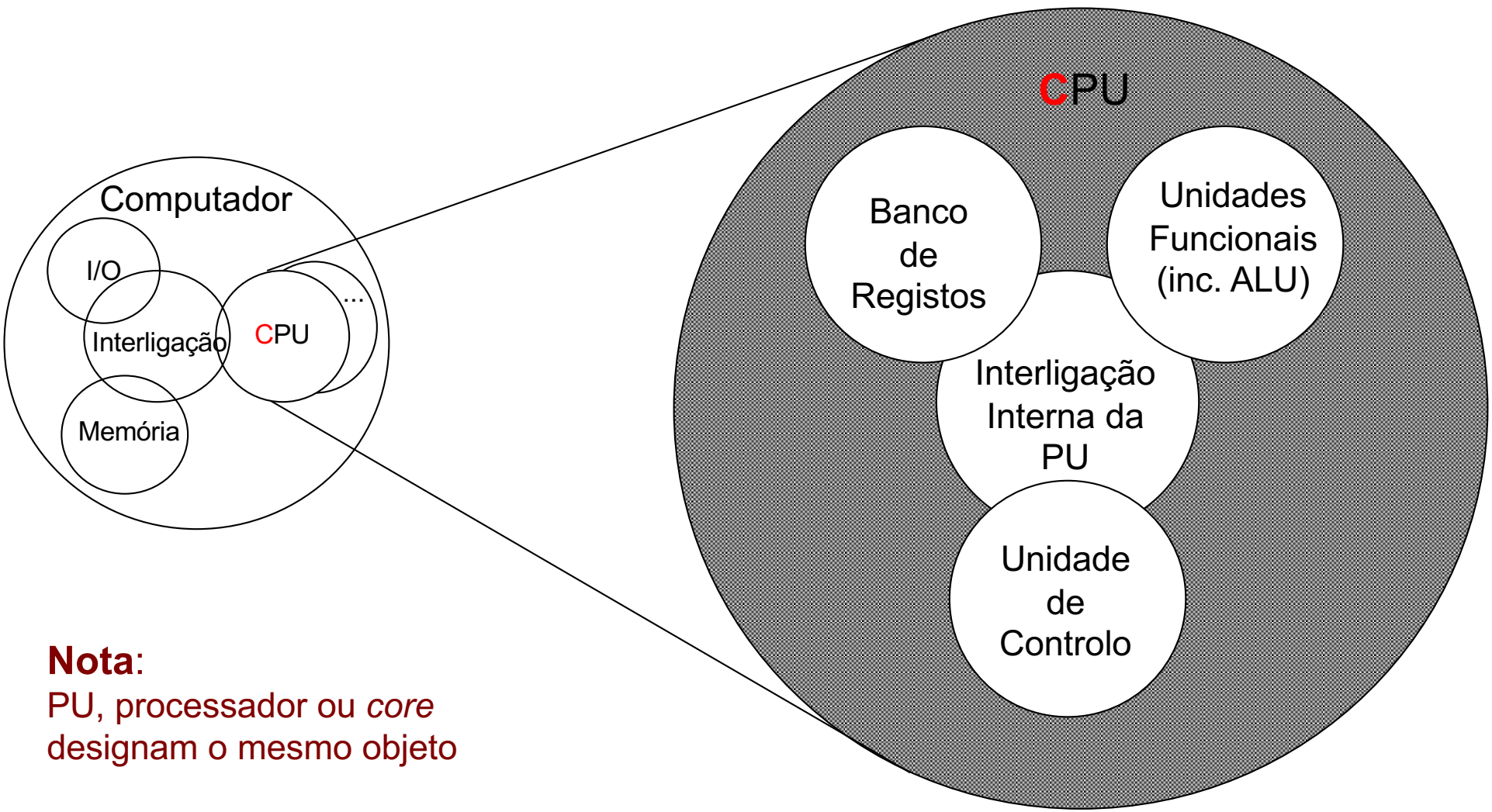
Estrutura do tema ISC

1. Representação de informação num computador
2. Organização e estrutura interna dum computador
3. Execução de programas num computador
4. Análise das instruções de um processador
5. Evolução da tecnologia e da eficiência

Organização e estrutura interna dum computador

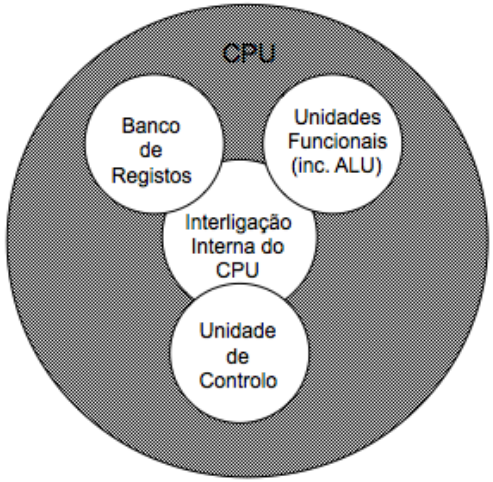


Estrutura interna dum processador (2)



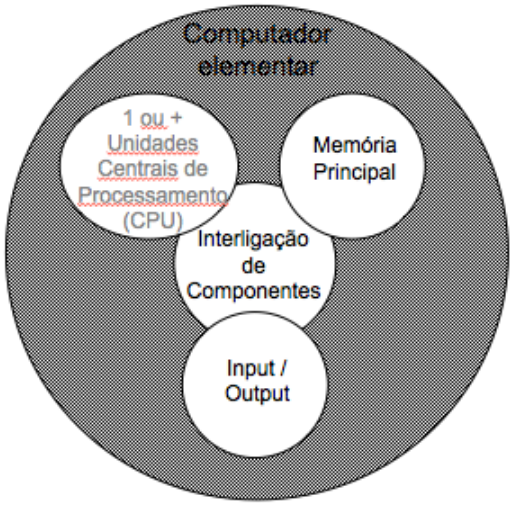
Nota:
PU, processador ou *core* designam o mesmo objeto

Estrutura interna dum processador (1)



Função dum PU:

- "motor" que continuamente
  - **lê** da memória um comando,
  - **interpreta**-o e,
  - **executa**-o (se precisar de operandos vai buscá-los e, se necessário, guarda também o resultado)
- de/para onde lê o comando:
  - da posição de memória definida no apontador p/ instrução (em registo) (**IP**, ***I**nstruction **P**ointer*, ou **PC**, ***P**rogram **C**ounter*),
  - para o registo de instrução (**IR**, ***I**nstruction **R**egister*)
- tipos básicos de comandos:
  - **operações** com dados, indo buscar os operandos se necessário e guardando o resultado no fim
  - **mover** dados de/para registos, memória ou I/O
  - **decidir** qual o (local do) próximo comando a executar



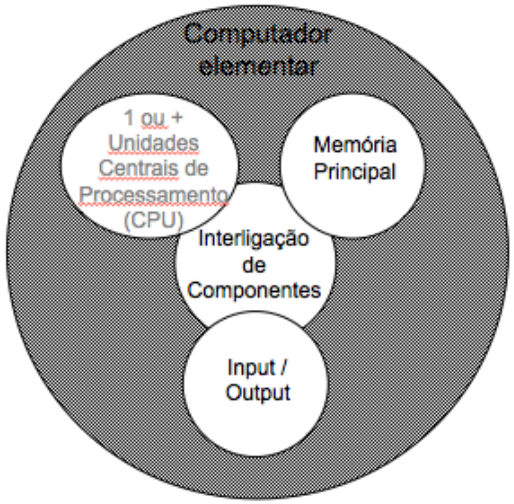
Organização duma memória principal ou primária

Função da memória primária:

- armazenar temporariamente **um programa e os dados por ele manipulados**, durante a execução de um programa
- operações que a memória executa: **ler** / **escrever**

Organização lógica:

- vetor (*array* linear) de **células**, cada com 8 bits
- cada célula é identificada pelo seu **endereço**
- dim máx definida pelos n bits do endereço: 2^n



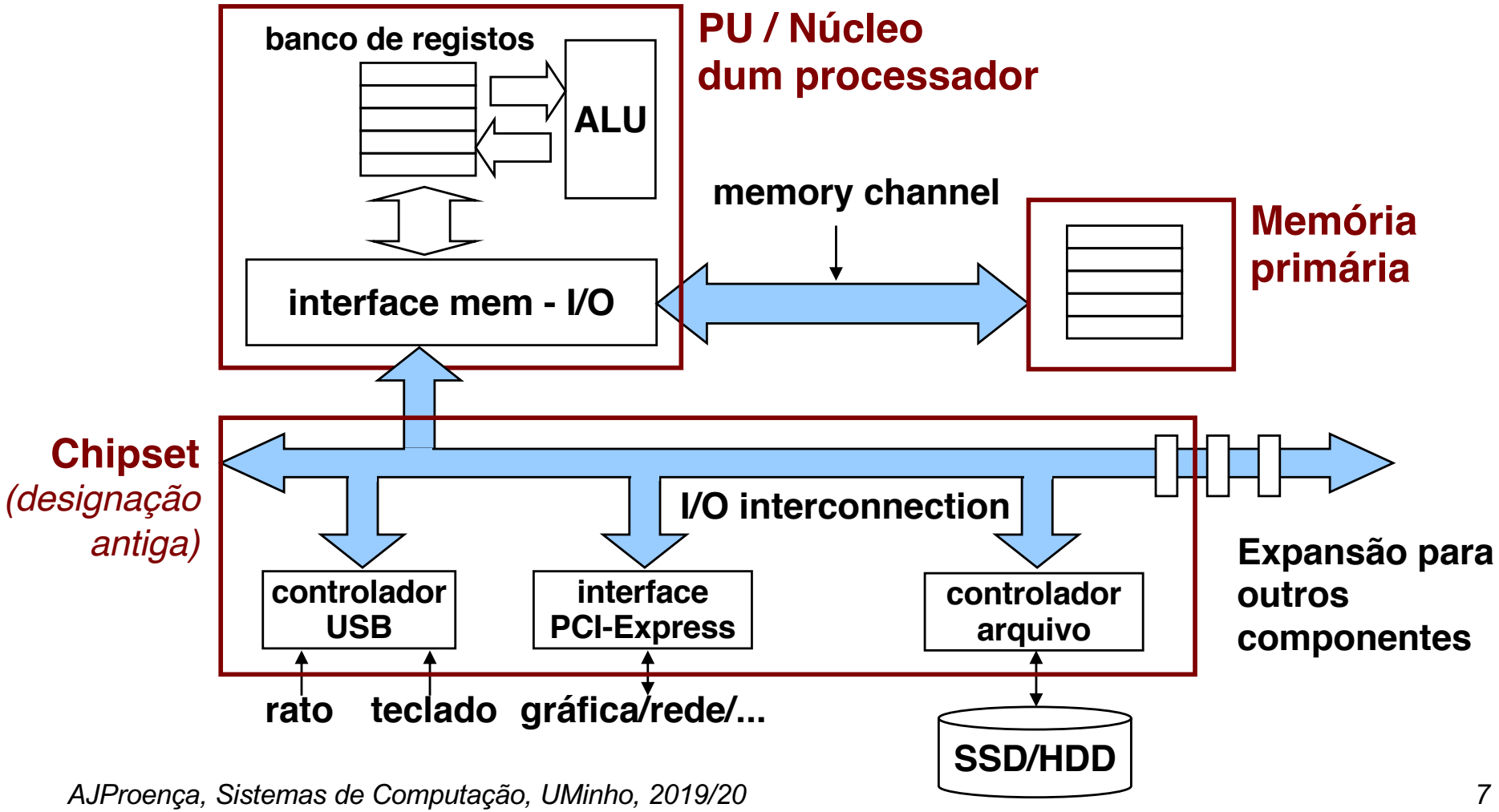
Tipos de comunicações c/ periféricos/dispositivos de I/O:

- com Humanos (monitor, teclado/rato, impressora,...)
- com máquinas (instrumentos, em sistemas embebidos, ...)
- com outros equipamentos (rede *wireless*, projetor m/média...)

Papel dos módulos de I/O:

- efetuar o interface físico e lógico entre o interior do computador e o exterior
- controlar o funcionamento de um ou mais periféricos
- fazer o intercâmbio de dados entre os periféricos e a memória principal (e/ou registos da PU)

Arquitetura base de um *laptop*:



Estrutura do tema ISC

1. Representação de informação num computador
2. Organização e estrutura interna dum computador
3. Execução de programas num computador
4. Análise das instruções de um processador
5. Evolução da tecnologia e da eficiência

```
int x = x+y;
```

- Código C
  - somar 2 inteiros (c/ sinal)

```
addl 8(%ebp),%eax
```

Idêntico à expressão

```
x = x + y
```

- *Assembly* (da GNU p/ IA-32)
  - somar 2 inteiros de 4 *bytes*
    - operandos "*long*" em GCC
    - a mesma instrução, c/ ou s/ sinal
  - operandos:
    - x: em registo %eax
    - y: na memória M[%ebp+8]

```
0x401046:    03 45 08
```

- Código *object* em IA-32
  - instrução com 3 *bytes*
  - na memória a partir do endereço `0x401046`

Níveis de abstração:

Slide anterior:

- Código C
  - somar 2 inteiros (c/ sinal)

```
int x = x+y;
```

- *Assembly* (da GNU p/ IA-32)
  - somar 2 inteiros de 4 *bytes*

```
addl 8(%ebp),%eax
```

Idêntico à expressão

```
x = x + y
```

- Código *object* em IA-32
  - instrução com 3 *bytes*
  - na memória a partir do endereço 0x401046

```
0x401046:    03 45 08
```

– nível das linguagens HLL (*High Level Languages*): as linguagens convencionais de programação (puro texto)
  - » imperativas e OO (Basic, Fortran, C/C++, Java, ...)
  - » funcionais (Lisp, Haskell, ...)
  - » lógicas (Prolog, ...)

– nível da linguagem *assembly* (de "montagem"): uma linguagem intermédia (comandos da PU em formato texto)

– nível da linguagem máquina: a linguagem de comandos, específica p/ cada PU ou família de PU's (em binário puro)
  - » arquiteturas CISC (*Complex Instruction Set Computers*)
  - » arquiteturas RISC (*Reduced Instruction Set Computers*)

Ciclo de execução de instruções:

1. **Leitura** de uma instrução da memória
   ... e incremento do IP
2. **Descodificação** da instrução
3. **Execução** da operação
   - cálculo da localização do(s) operando(s),
     e ir buscá-lo(s), se necessário
   - execução da ação especificada
   - guardar resultado, se necessário

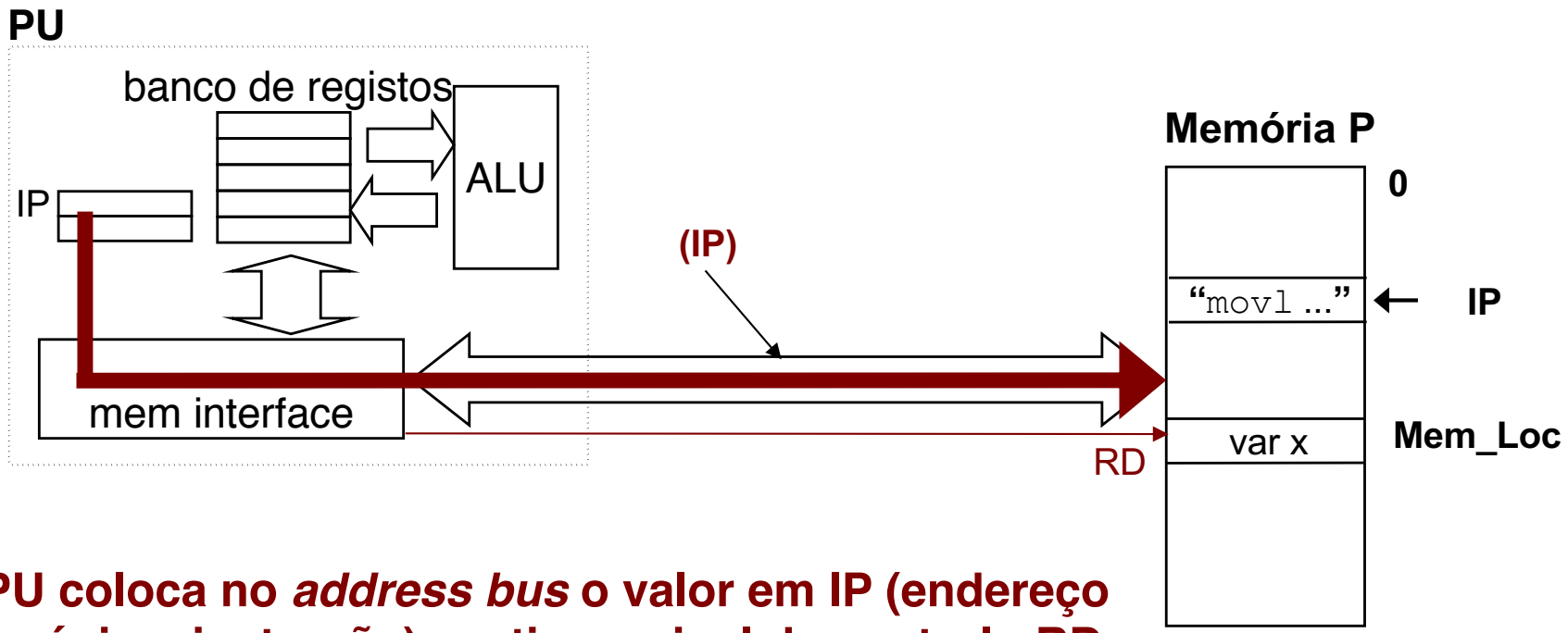
Análise de um exemplo: `movl Mem_Loc,%eax`

Mecanismos de conversão entre níveis de abstração

Modelo de computação de von Neumann (1945)

Ex.: `movl Mem_Loc,%eax`

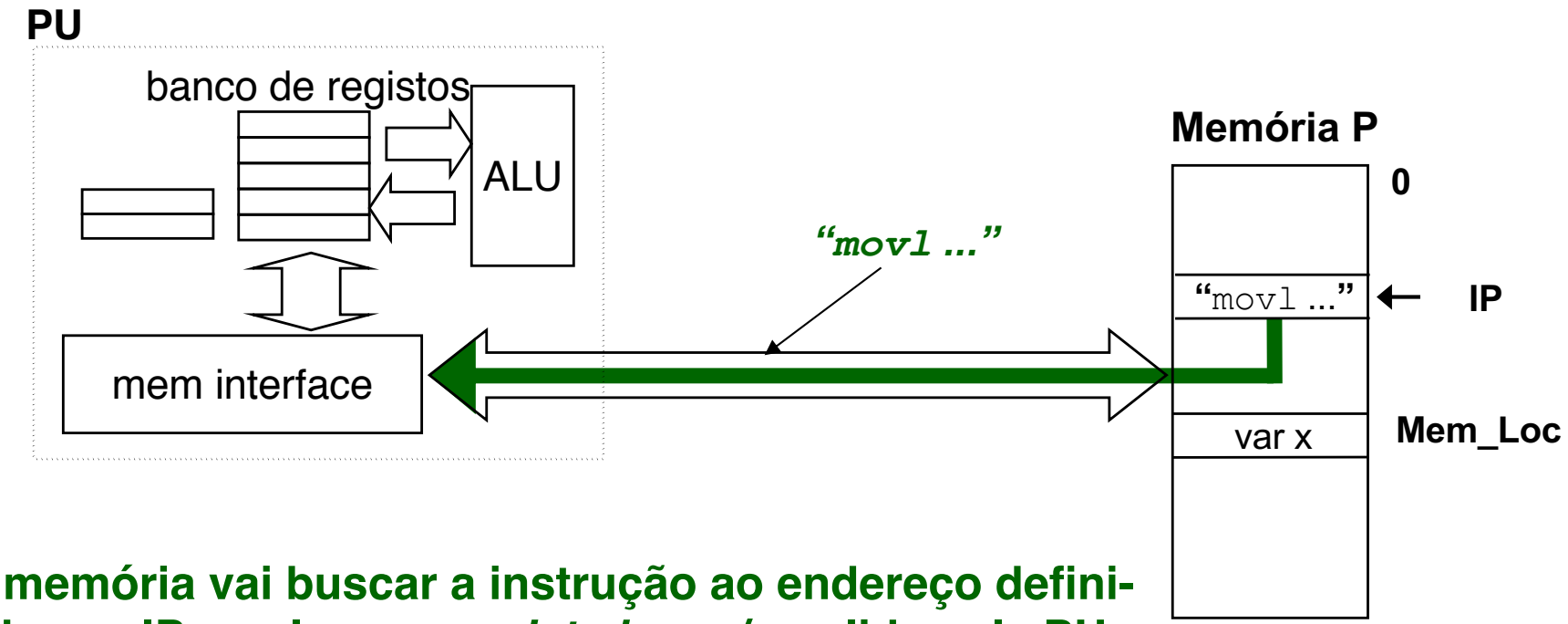
1. Leitura da instrução na memória (1)



A PU coloca no *address bus* o valor em IP (endereço p/ próxima instrução), e ativa o sinal de controlo RD

Ex.: `movl Mem_Loc,%eax`

1. Leitura da instrução na memória (2)

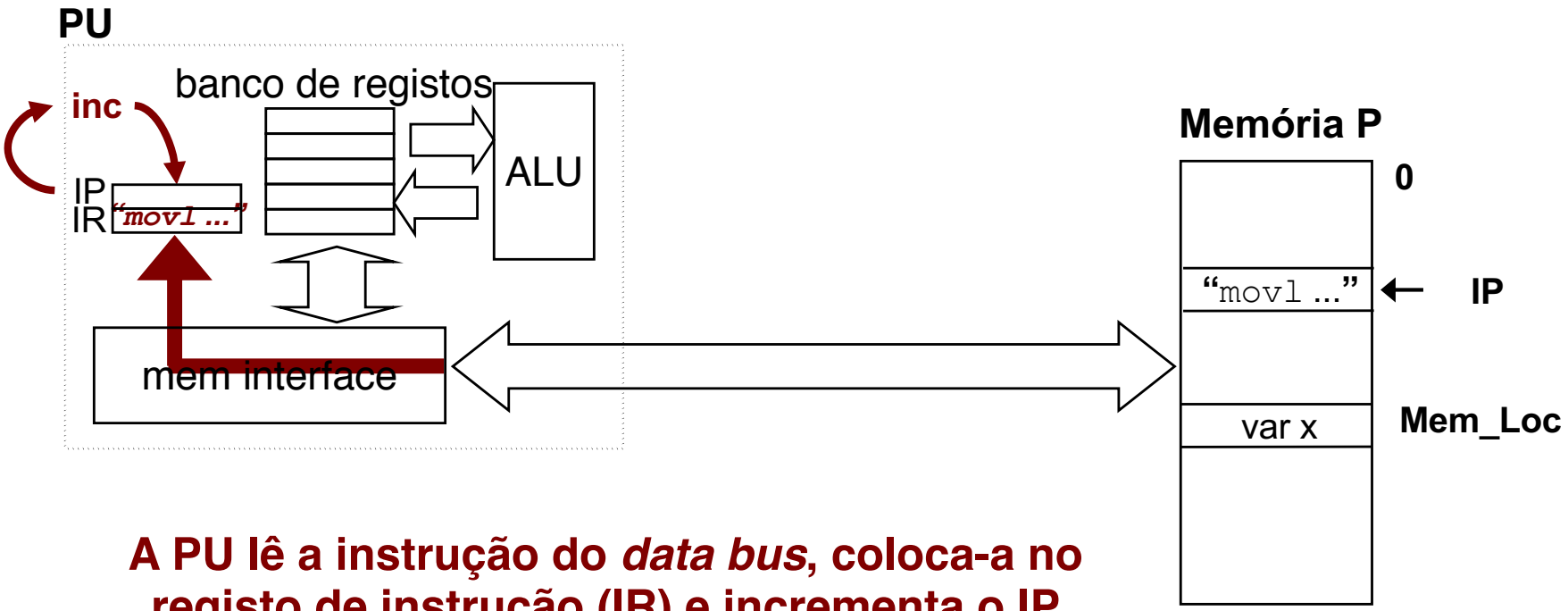


A memória vai buscar a instrução ao endereço definido por IP e coloca-a no *data bus* p/ ser lida pela PU

Ex.: `movl Mem_Loc,%eax`

1. Leitura da instrução na memória (3)

... e incremento do IP



A PU lê a instrução do *data bus*, coloca-a no registo de instrução (IR) e incrementa o IP

Exemplo de execução de uma instrução em linguagem máquina (4)

Ex.: `movl Mem_Loc,%eax`

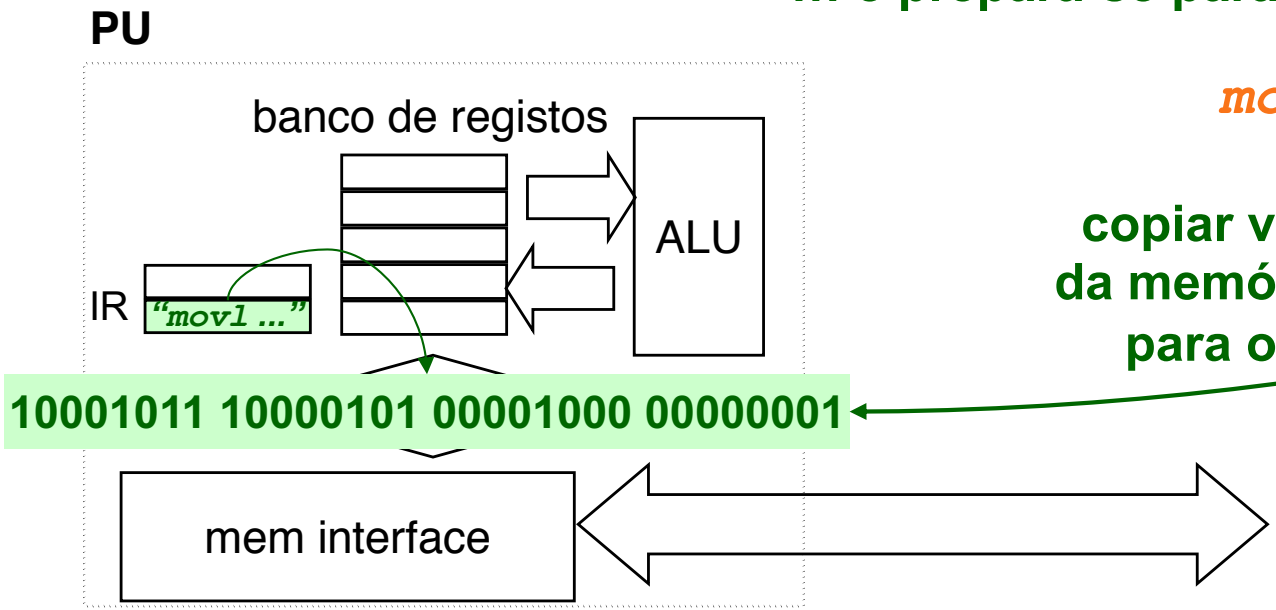
2. Descodificação da instrução

A unidade de controlo da PU descodifica a instrução…

… e prepara-se para executar a operação:

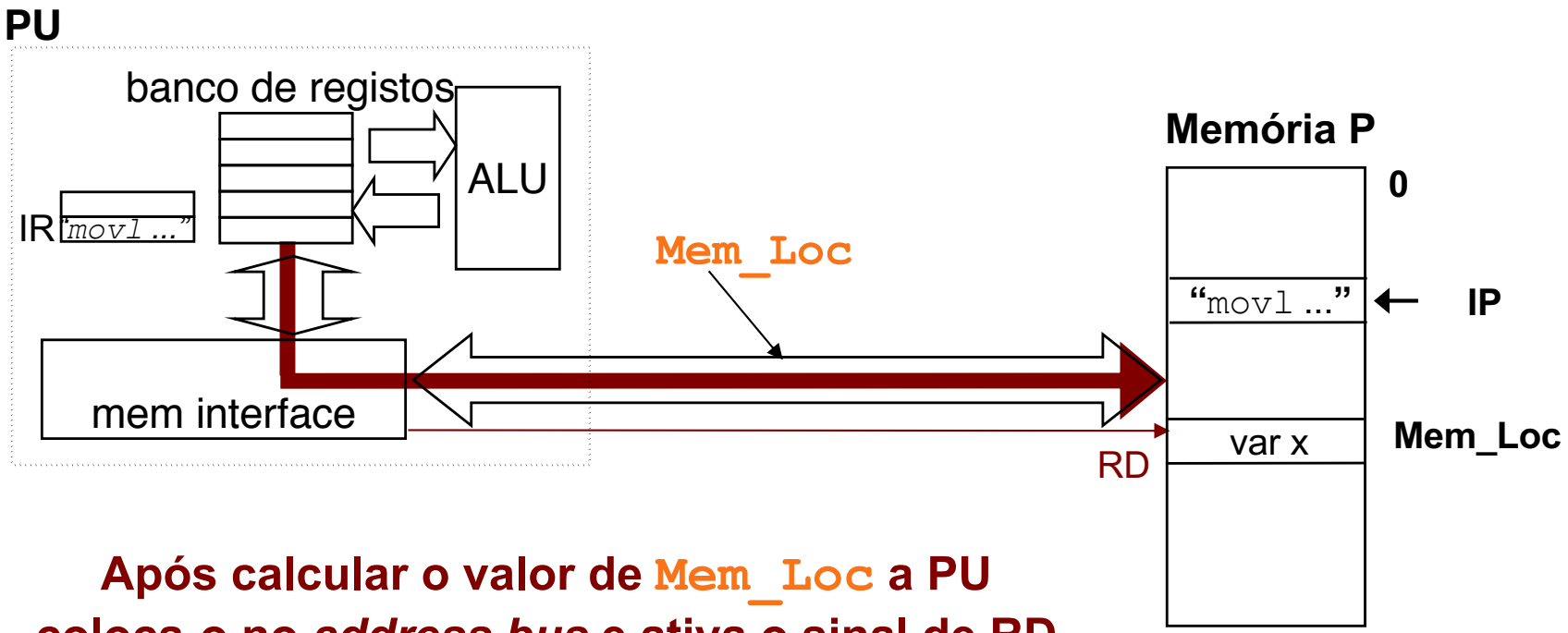
move long

copiar valor com 32 bits da memória, em Mem_Loc para o registo %eax



Ex.: `movl Mem_Loc,%eax`

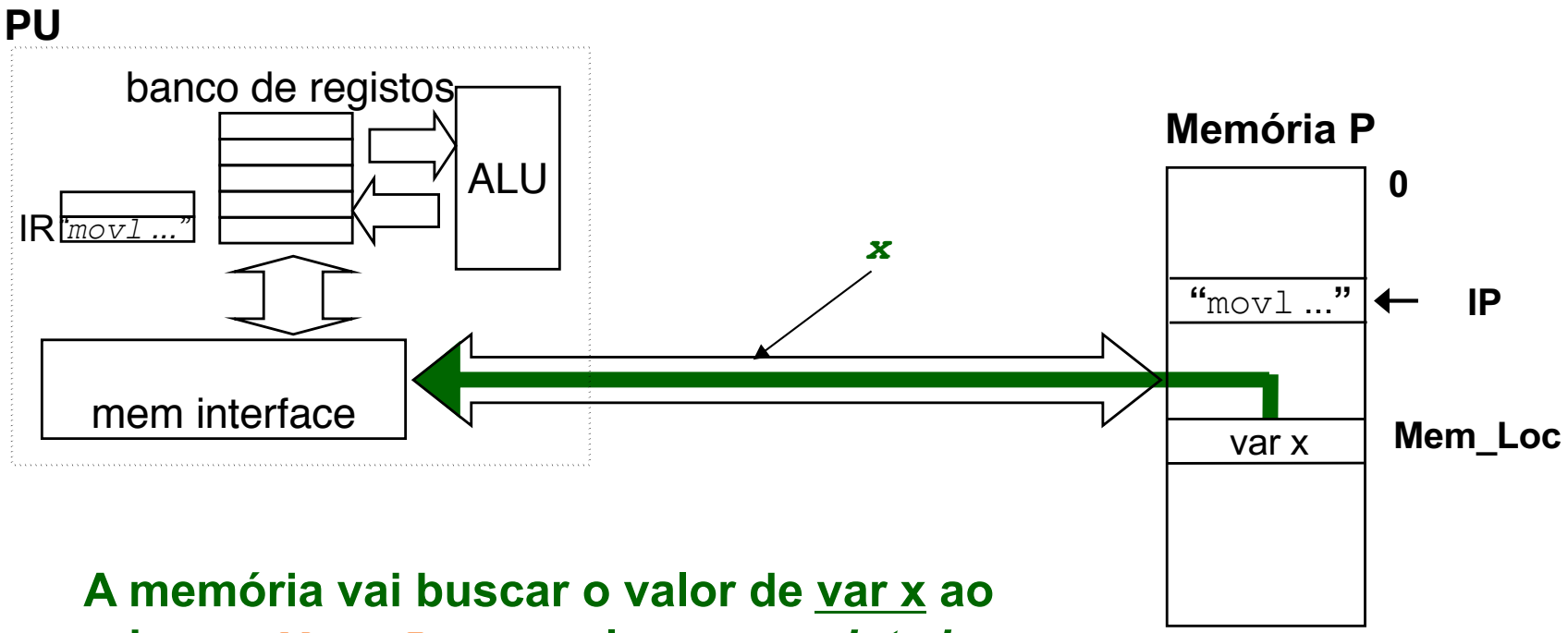
3. Execução da operação (1)



Após calcular o valor de `Mem_Loc` a PU coloca-o no *address bus* e ativa o sinal de RD

Ex.: `movl Mem_Loc,%eax`

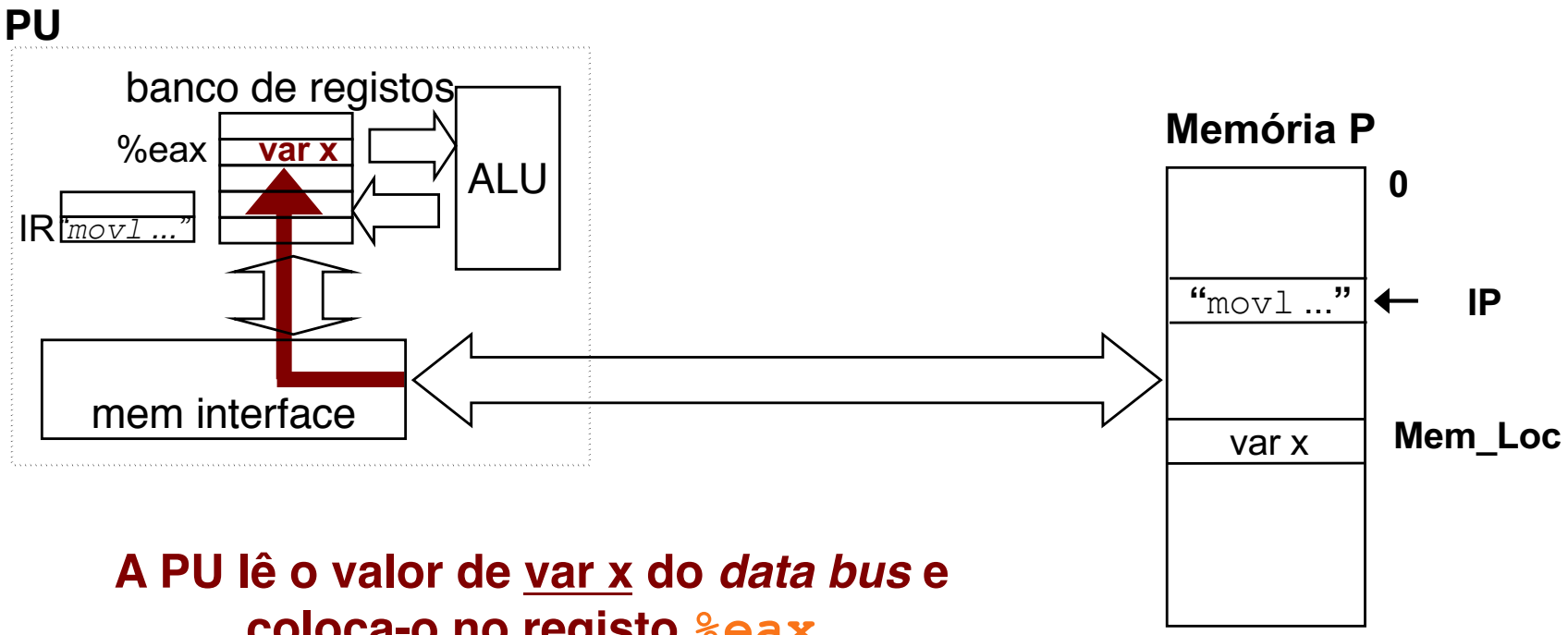
3. Execução da operação (2)



A memória vai buscar o valor de var x ao endereço `Mem_Loc` e coloca-o no *data bus*

Ex.: `movl Mem_Loc,%eax`

3. Execução da operação (3)

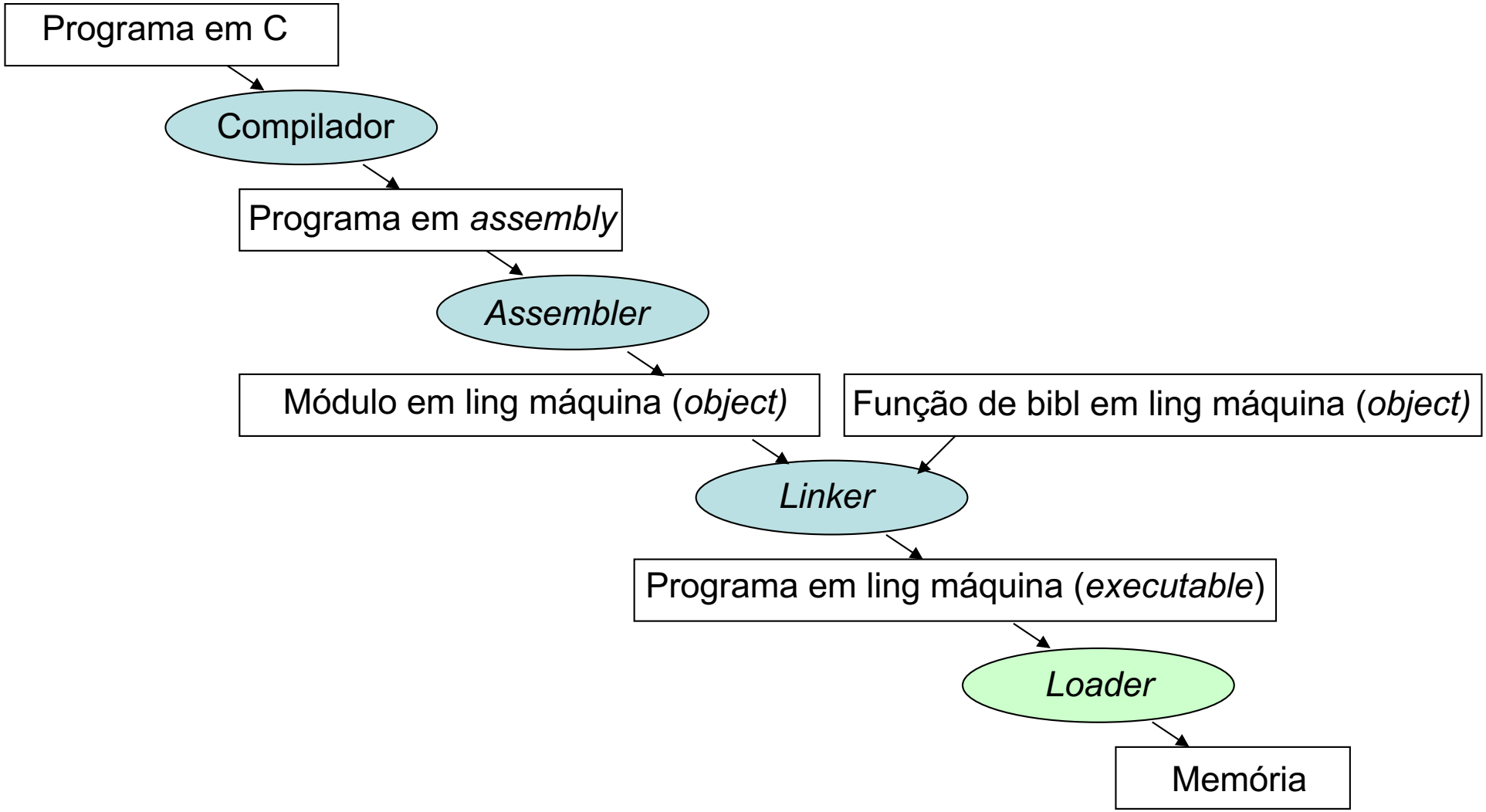


A PU lê o valor de var x do *data bus* e coloca-o no registo `%eax`

Mecanismos de conversão (para comandos da PU):

- compilador
  - traduz um programa de um nível de abstração para outro inferior (converte um ficheiro de texto noutro de texto);
    por ex., de C para *assembly*
  - normalmente inclui mais que um passo de conversão, até chegar à linguagem máquina
- *assembler* ("montador")
  - "monta" os comandos / instruções, em texto, para binário (*object*), de acordo com as regras do fabricante da PU
- interpretador
  - analisa, uma a uma, as instruções de um programa em HLL, e:
    - » **gera código** em linguagem máquina para essa instrução, **e**
    - » **executa** esse código (nota: não guarda o código gerado)

Execução de programas num computador: de HLL até à sua execução



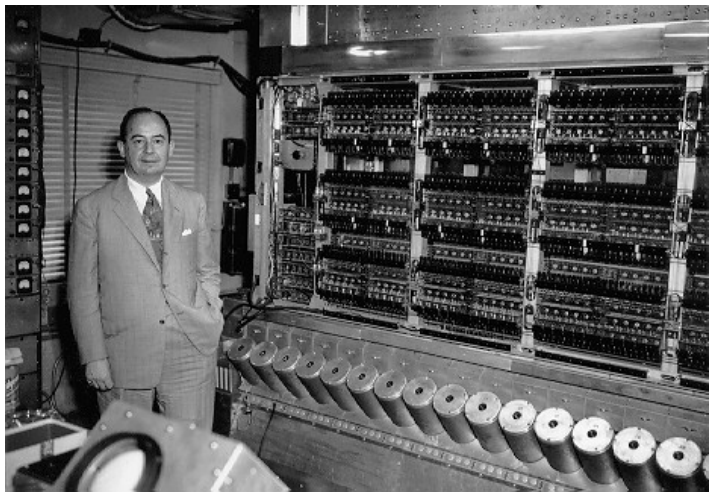
Modelo de computação de von Neumann, 1945/46 (1)

ENIAC (1ª geração, 1945)

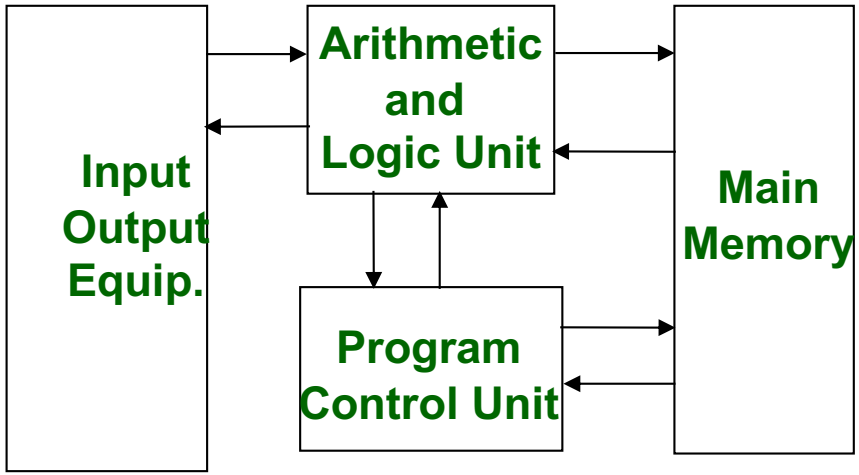
- objetivo: cálculo tabelas de artilharia (mas 1º teste foi p/ bomba H)
- máquina **decimal** (base 10)
- 17.468 válvulas, 27 toneladas
- programação: manual, alterando as conexões (cablagem)

Von Neumann introduz conceito de ***stored-program*** :

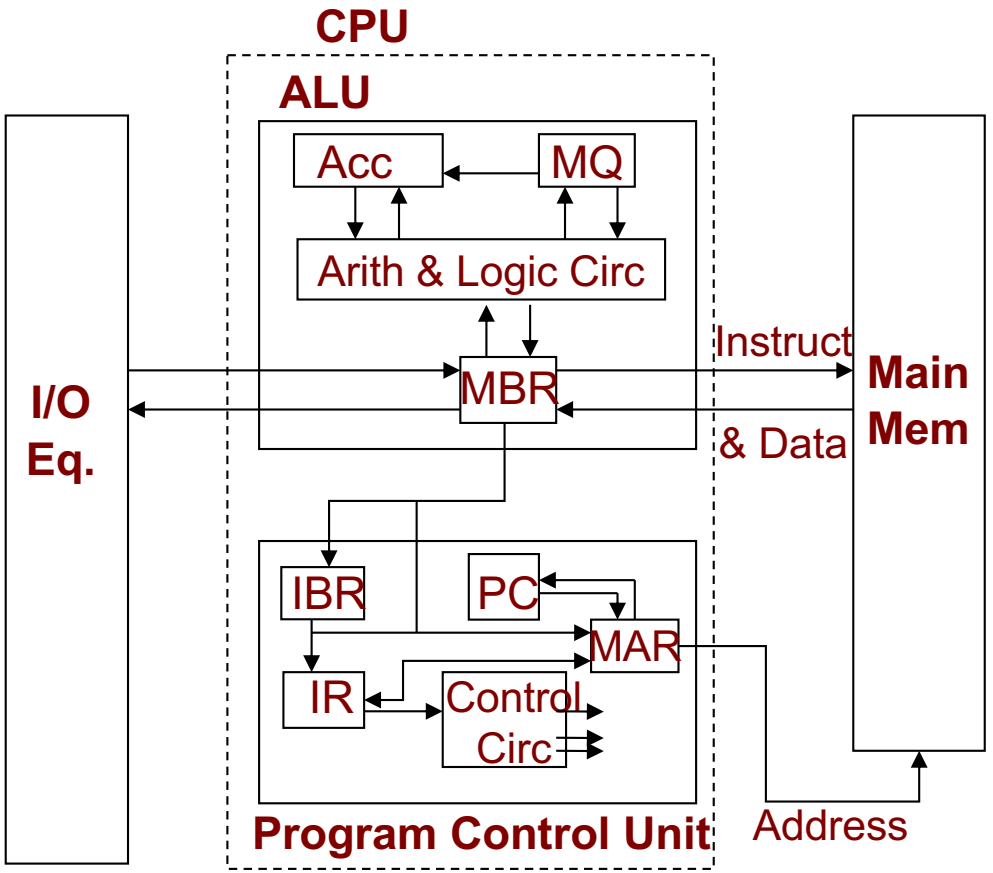
- dados e instruções em **binário**, e armazenados numa memória
- memória acedida pelo endereço da informação
- execução de instruções de modo sequencial (daí o *Program Counter*, PC), interpretadas pela unid. controlo
- constrói novo computador, IAS



Modelo de computação de von Neumann, 1945/46 (2)



Estrutura básica do IAS
(*Princeton Institute for Advanced Studies*)



Estrutura expandida do IAS