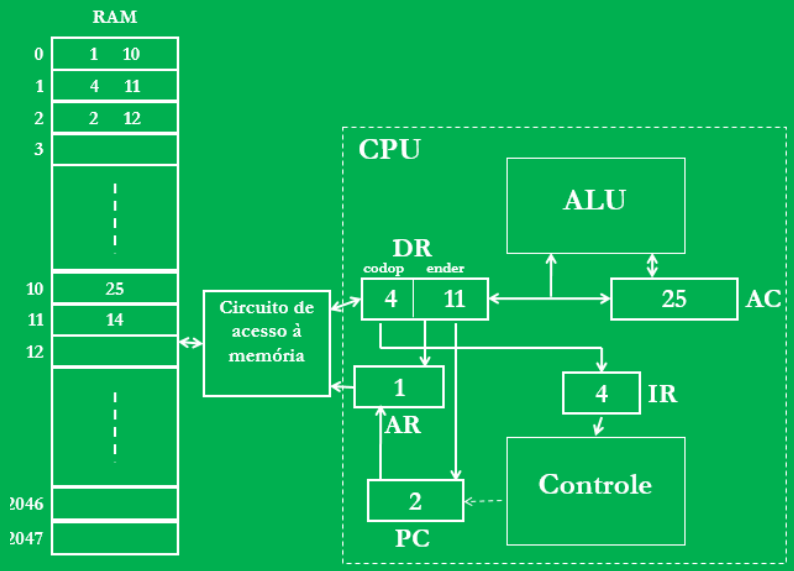
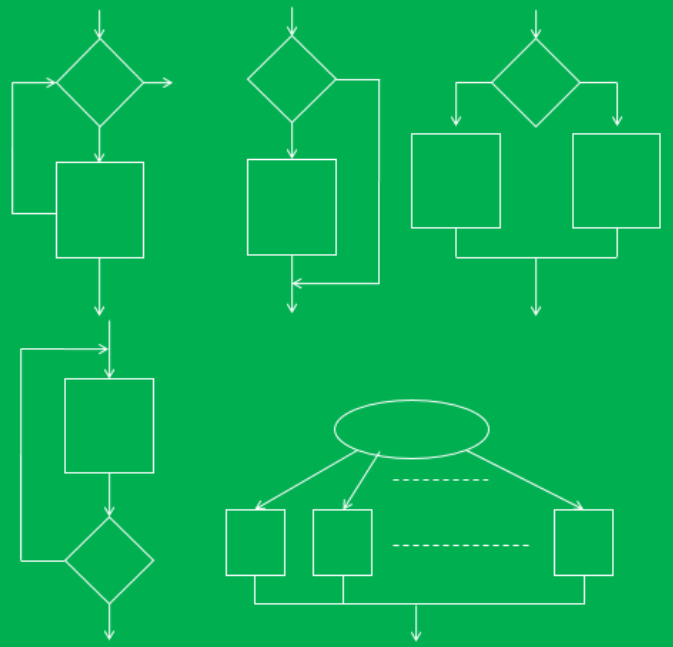
COMPUTADORES, ALGORITMOS E PROGRAMAS: MANUAL INTRODUTÓRIO DE ENSINO

Fábio Carneiro Mokarzel
Nei Yoshihiro Soma





Esta obra é a primeira de uma série de três livros que pretendem servir como um guia para a preparação de aulas de disciplinas introdutórias de Computação e Programação, em cursos de Engenharia.
Inicialmente, são apresentados os conceitos básicos de computadores e entidades correlatas como programas, hardware e software; são descritos alguns princípios de funcionamento desses dispositivos; a estrutura e a natureza binária das informações por eles manipuladas é devidamente detalhada e é exposto um panorama da evolução da comunicação humano-computador e das linguagens de programação.
Em seguida é abordada a forma usual de se resolver problemas por computador, começando pela escolha de um método, transcrevendo-o para algoritmo e obtendo-se um programa correspondente escrito em uma linguagem convencional de programação.
Os outros dois livros devem tratar, respectivamente, da programação unimodular e da programação multimodular, apresentando grande ferramental de programação, incluindo comandos de diversos tipos, muitas formas de estruturar os dados a serem processados e importantes técnicas de organizar os módulos do programa, visando alcançar racionalidade no desenvolvimento e eficiência na execução.
O conjunto desses três livros apresenta ilustrações com exaustivos detalhes, visando dar ao professor diversas opções para tornar suas aulas instrutivas e agradáveis de serem assistidas.

Computadores, Algoritmos e Programas: Manual Introdutório de Ensino

Fábio Carneiro Mokarzel

Nei Yoshihiro Soma

Computadores, Algoritmos e Programas: Manual Introdutório de Ensino

Fábio Carneiro Mokarzel

Nei Yoshihiro Soma



Reservado para ISBN

FÁBIO CARNEIRO MOKARZEL

Disse o Senhor Deus, Criador e Sustentador dos céus e de todo o universo: "Buscar-me-eis e me achareis, quando me buscardes de todo o vosso coração"

Dedico este trabalho ao Senhor Jesus Cristo, Meu Criador, Salvador e Libertador de minha alma e à minha esposa Ilda Alves dos Santos Mokarzel

NEI YOSHIHIRO SOMA

Não tenho compromisso com minhas ideias, pois o único compromisso que se pode ter em termos de ideias é com a busca da verdade **(Prof. Anísio Teixeira)**

A meus pais, Yoshio e Verônica (*in memoriam*), irmã Nanci, esposa Akemi e filhos Renato, Ryuji e Daniel.dmc.

Os autores também dedicam esta obra aos professores que vierem a abraçar a responsabilidade de ministrar disciplinas introdutórias sobre computadores, algoritmos e programas

Agradecimentos

Às nossas esposas, Ilda e Akemi, pelas incessantes orações, apoio, incentivo e constantes exortações para que perseverássemos na elaboração deste livro.

A todos os alunos do ITA para os quais ministramos as disciplinas de Introdução à Computação e à Programação, pelo retorno que nos proporcionaram sobre o material didático que lhes temos apresentado durante as diversas etapas de sua produção.

Prefácio

Desde os Anos 80, quando, no Instituto Tecnológico de Aeronáutica, iniciamos nossa carreira como Professor Universitário, vimos ministrando, entre outras disciplinas de Ciência e Engenharia de Computação, disciplinas introdutórias de Computação e de Programação. Para nos auxiliar na transmissão dos conhecimentos sobre estes assuntos, vimos produzindo, modernizando e aperfeiçoando material didático para apresentar aos alunos em classe e deixa-lo disponível para estudos extra classe.

Começamos pela elaboração de manuscritos a serem copiados na lousa, solicitando que os alunos os copiassem em seus cadernos. Pouco tempo depois, substituímos a lousa e o giz por retroprojetor e tela de projeção e transcrevemos os manuscritos para transparências, usando canetas coloridas apropriadas e fazendo cópias dessas transparências em papel para serem adquiridas pelos alunos. Depois, usando ainda o referido dispositivo, passamos a produzir documentos no aplicativo WORD e a copia-los nas transparências a serem projetadas, fornecendo a versão eletrônica aos alunos, uma vez que os notebooks passaram a ser material escolar. Com a difusão dos projetores denominados inicialmente *data-shows*, passamos a transcrever nosso material didático para arquivos a serem executados no aplicativo PowerPoint, utilizando, no início, poucos recursos desse aplicativo. Finalmente aperfeiçoamos esse material, aplicando em sua totalidade técnicas de animação, o que veio a tornar nossas aulas muito mais dinâmicas, atrativas e agradáveis de serem acompanhadas, não somente em classe, mas também nos estudos extra classe.

Nesse estágio de desenvolvimento de nosso material didático, passamos a receber de nossos alunos opiniões altamente positivas sobre a sua qualidade didática, bem como de colegas professores para os quais emprestamos o material para consultas e também para utiliza-lo em suas aulas.

Assim, decidimos elaborar o presente trabalho, usando o formato *pdf*, para disponibiliza-lo em papel e em meio eletrônico à comunidade acadêmica, servindo-lhe como um manual para a preparação de aulas teóricas de disciplinas introdutórias de Computação e Programação, em cursos de Engenharia.

O presente trabalho destina-se ao preparo de aulas para aproximadamente um terço das aulas teóricas de um curso de Introdução à Programação. Os assuntos abordados são conceitos primários sobre computadores, algoritmos e programas. Os outros dois terços deverão ser por nós apresentados em dois outros livros cujos temas centrais serão, respectivamente, os princípios de programação unimodular e de subprogramação.

Num futuro próximo, pretendemos publicar também livros-manuais sobre o preparo de aulas práticas sobre Introdução à Programação. Além desta disciplina, há boas perspectivas de fazermos o mesmo trabalho para as disciplinas de Estruturas de dados e de Compiladores.

Foi escolhida a Linguagem C para ilustrar todos os princípios e técnicas de Programação apresentados neste trabalho.

RESUMO DO CONTEÚDO DOS CAPÍTULOS

O Capítulo zero, *Apresentação*, é um esquema para a aula inaugural da disciplina. Começa pelos seus objetivos, vindo a seguir um histórico sobre o aparecimento, a evolução e a atuação do computador na sociedade, e a relevância da disciplina em cursos de Engenharia. Depois é apresentada uma divisão em capítulos e um cronograma a ser seguido na ministração da disciplina e por fim é apresentada uma Bibliografia para enriquecer os conhecimentos a serem transmitidos.

O Capítulo I, *Estrutura de um Computador*, apresenta conceitos básicos sobre computadores e entidades correlatas como programas, hardware e software, descreve alguns princípios de funcionamento desses equipamentos e mostra esquemas primitivos e modernos das unidades componentes de sua arquitetura.

O Capítulo II, *Informações Manipuladas por um Computador*, expõe a estrutura e a natureza binária das informações por eles manipuladas, incluindo o formato de armazenamento de números inteiros e reais, de textos, de imagens e de instruções a serem executadas.

O Capítulo III, *Evolução das Linguagens de Programação*, mostra um panorama da evolução da comunicação humano-computador e das linguagens de programação, abordando, inicialmente, a linguagem de máquina, mas, depois, discorrendo sobre a linguagem Assembly, as linguagens tradicionais de programação e as linguagens aplicativas.

O Capítulo IV, *Elementos Básicos de Algoritmos e Programas*, apresenta o conceito de algoritmos e seu importante papel na elaboração de programas para computadores.

O Capítulo V, *Linguagens para Algoritmos*, discorre sobre algumas linguagens comumente usadas para a elaboração de algoritmos, dentre as quais, o pseudocódigo, os fluxogramas e a linguagem de programação aplainada.

O Capítulo VI, *Propriedades dos Bons Algoritmos*, enumera várias propriedades desejáveis que um algoritmo deve apresentar.

Finalmente o Capítulo VII, *Estrutura de um programa em C*, coloca os alunos em contato com a estrutura de programas em C, apresentando suas partes componentes.

Nossa expectativa com esta e com as duas próximas obras é a de que professores que venham a ministrar as referidas disciplinas possam traçar um plano seguro para suas aulas teóricas e tenham substancial fonte de conhecimentos e ilustrações para enriquece-las.

Sumário

Dedicatória 7

Agradecimentos 9

Prefácio 11

Sumário 15

Capítulo Zero - Apresentação 19

- **0.1 Objetivos de uma disciplina de Introdução à Programação** 21
- **0.2 O computador na sociedade** 22
- **0.3 Relevância da Introdução à Programação** 65
- **0.4 Programa de Introdução à Programação** 76
- **0.5 Bibliografia** 78

Capítulo I – Estrutura de um Computador 93

1.1 Conceito de Computador 95

1.2 Modelo hipotético de um computador 104

1.3 Unidades básicas de um computador 202

Capítulo II – Informações manipuladas por um computador 230

2.1 Natureza das informações 232

2.2 Sistemas de numeração e mudança de base 249

2.3 Aritmética binária e operações lógicas 265

2.4 Representação de números inteiros 289

2.5 Representação de números reais 307

2.6 Representação de textos 326

2.7 Representação de imagens 337

2.8 Representação de instruções 339

Capítulo III – Evolução das linguagens de programação 344

3.1 A linguagem de máquina 346

3.2 Linguagem Assembly 480

3.3 Linguagens tradicionais de programação 491

3.4 Linguagens aplicativas **531**
Capítulo IV – Elementos básicos de algoritmos e programas **534**
4.1 A necessidade de métodos e algoritmos **536**
4.2 Algoritmos executados por seres humanos **540**
4.3 Algoritmos para computadores **546**
4.4 Programas a partir de algoritmos **588**
4.5 Produção de um software **607**
Capítulo V – Linguagens para algoritmos **618**
5.1 Pseudocódigo **620**
5.2 Fluxogramas **623**
5.3 Linguagem de programação aplainada **637**
Capítulo VI Propriedades dos bons algoritmos **651**
6.1 Tempo finito de execução **654**
6.2 Comandos definidos **655**
6.3 Comandos efetivos **659**
6.4 Início e término dos algoritmos **661**
6.5 Entrada de dados e saída de resultados **663**

6.6 Algoritmos amigáveis **666**

6.7 Algoritmos bem-estruturados, legíveis e de fácil correção **670**

6.8 Algoritmos tolerantes a falhas **676**

6.9 Outros ingredientes **677**

Capítulo VII – Estrutura de um programa em C **678**

7.1 Partes componentes de um programa em C **680**

7.2 Cabeçalho de uma função **685**

7.3 Declarações globais e locais **689**

7.4 Diretivas de pré-processamento **691**

7.5 Comentários em programas **721**

Computadores, Algoritmos e Programas: Manual Introdutório de Ensino

Capítulo Zero
Apresentação

Capítulo Zero - Apresentação

0.1 – Objetivos de uma Disciplina de Introdução à Programação

0.2 – O computador na sociedade

0.3 – Relevância da Introdução à Programação

0.4 – Programa de Introdução à Programação

0.5 – Bibliografia

0.1 – Objetivos de uma Disciplina de Introdução à Programação

- **Conceituação** de computadores, linguagens, algoritmos, programas e compiladores
- Desenvolvimento de **algoritmos**
- **Programação unimodular** numa linguagem imperativa estruturada:
  - Linguagem escolhida: **C**
- **Programação multimodular** – Subprogramas
- Este manual cobre o **primeiro** destes objetivos

0.2 – O Computador na Sociedade

0.2.1 – Precursores do computador eletrônico digital

A tarefa de **calcular** vem sendo automatizada há muitos anos:

- **Ábacos:**

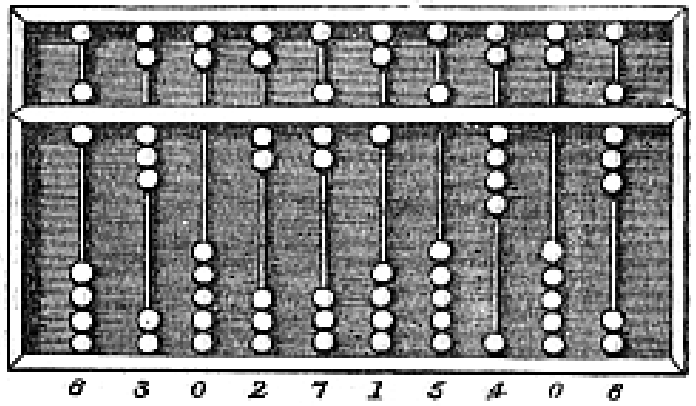
  origem provável:

  Mesopotâmia

  há mais de 5000 anos

Ábaco versão chinesa (Wikipedia)

Site 01



Primeira calculadora utilizada pelo homem: um ábaco representando o número 6302715406.

- **Réguas de cálculo:**

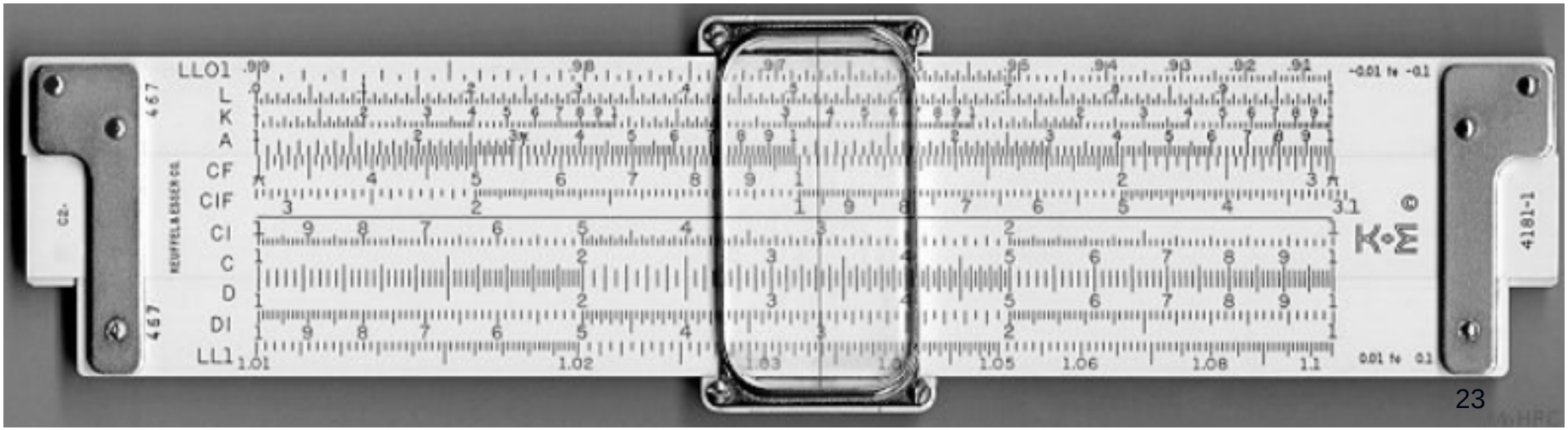
  - Criadas pelo padre inglês William Oughtred, em 1638

  - Aperfeiçoadas e muito usadas até a década de 1970

  - Substituídas pelas calculadoras eletrônicas

  - Faziam cálculos complexos como potenciação, radiciação, logaritmos e funções trigonométricas

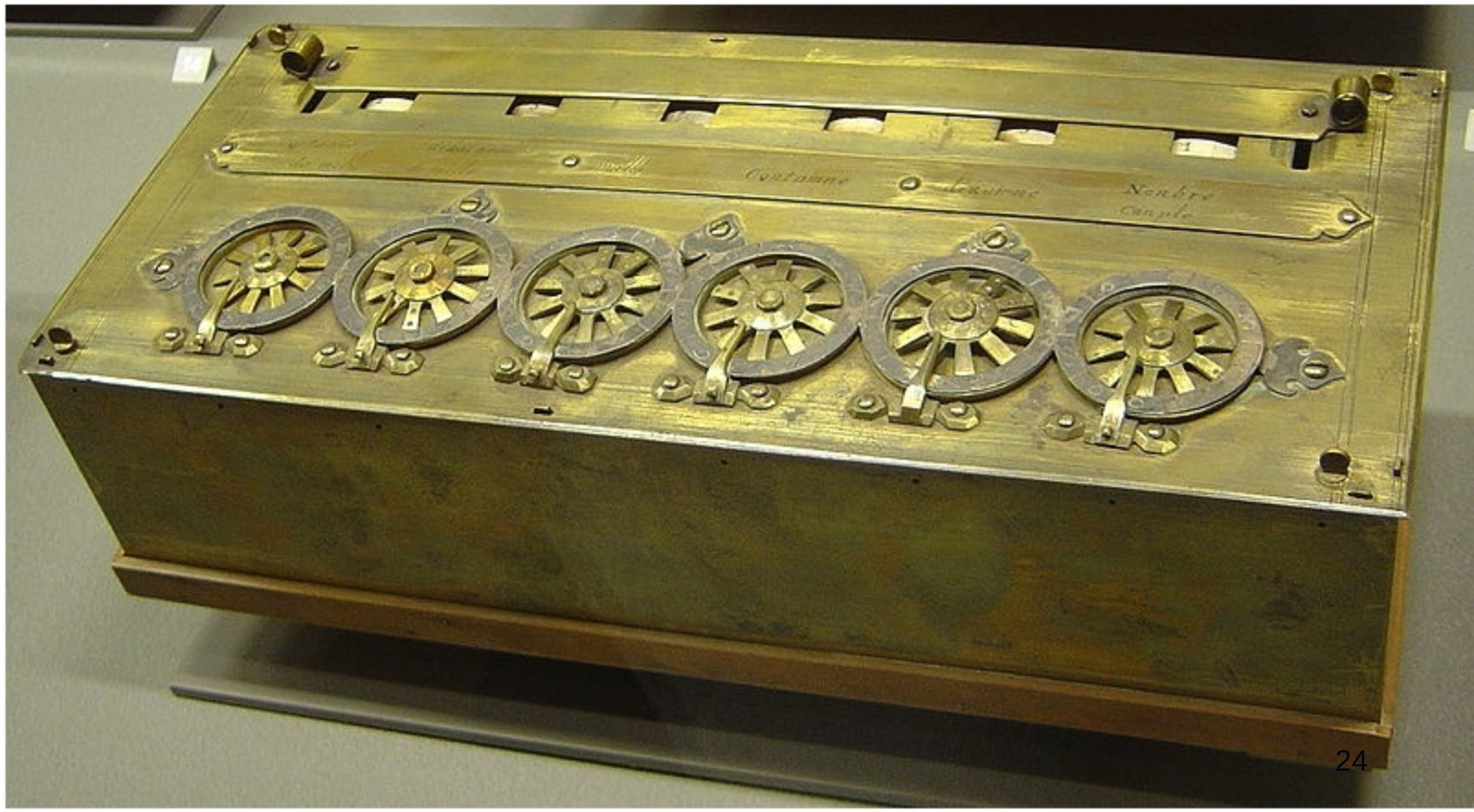
Site 02



- **Calculadoras mecânicas:**

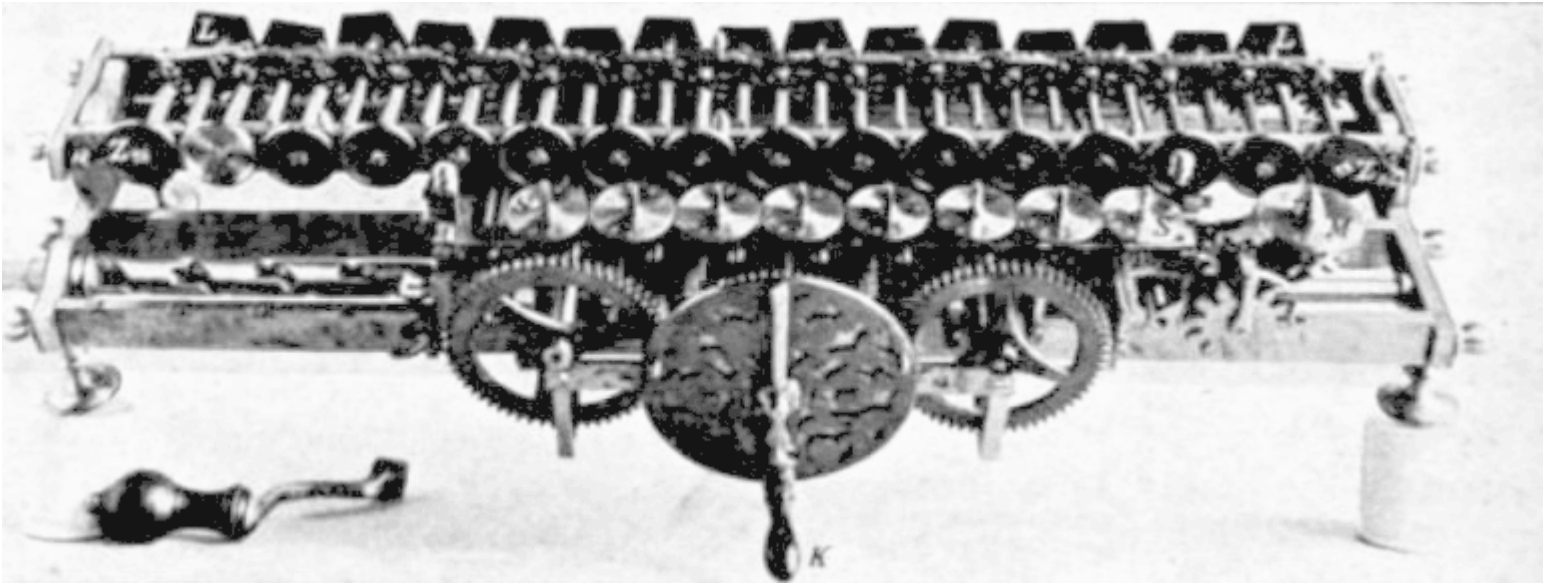
  - **Pascalina:** construída pelo francês Blaise Pascal, em 1642
  - Fazia somas e subtrações

Site 03



- **Calculadoras mecânicas:**

  - **Calculadora de Leibniz:** construída na Alemanha por Gottfried Leibniz, por volta de 1694
  - Fazia somas, subtrações, multiplicações e divisões
  - As duas últimas aplicavam sucessivas execuções das duas primeiras



- **Calculadoras mecânicas:**

  - **Arithmometer:** construída pelo francês Charles Xavier Thomas, em 1820
  - Fazia somas, subtrações, multiplicações e divisões
  - Primeira calculadora realmente comercializada com sucesso: até 1850, vendidos cerca de 1500 exemplares

Site 05



- **Máquina analítica:** concepção de um computador mecânico, em 1833, pelo inglês Charles Babbage

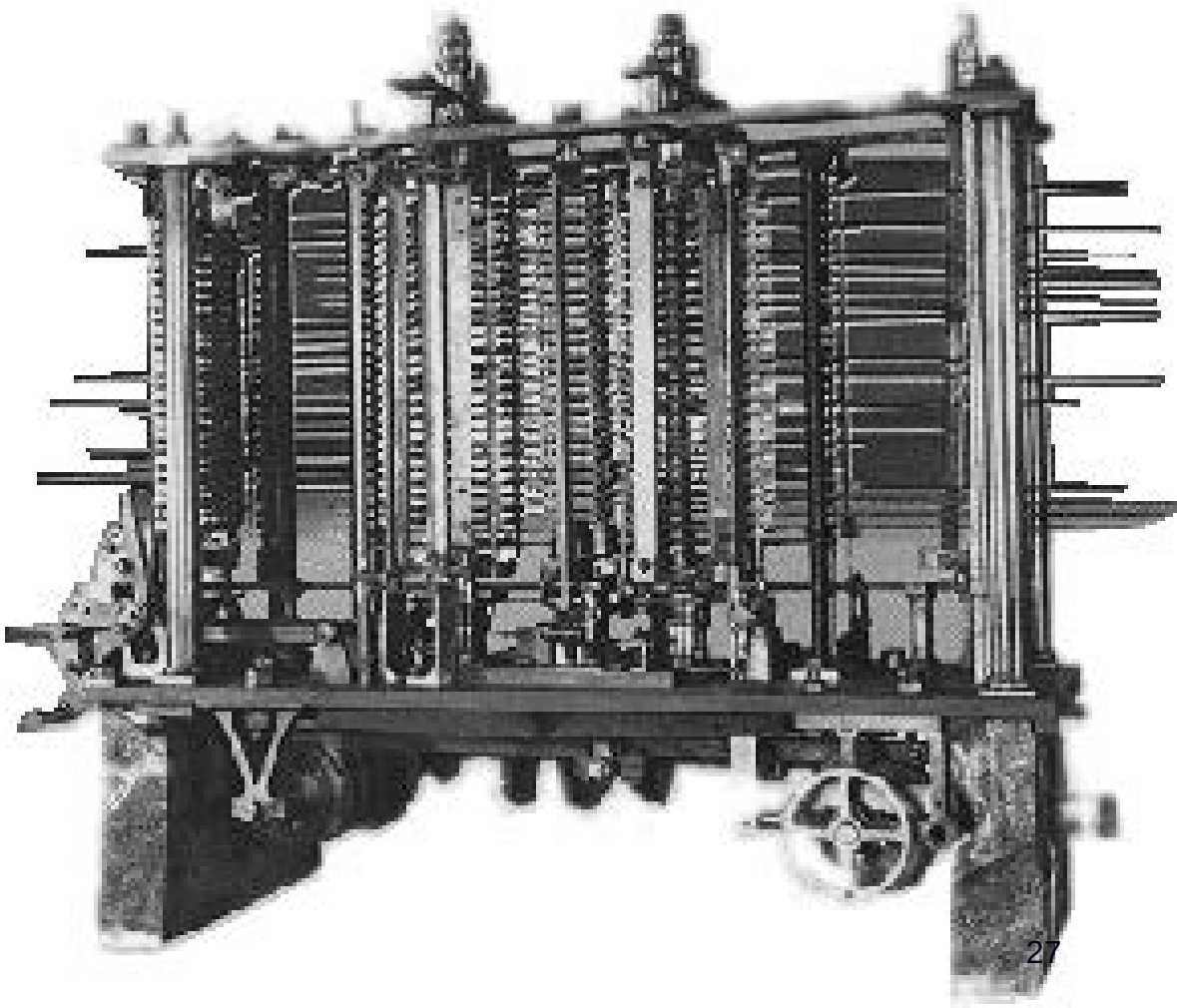
Com unidade de controle de memória aritmética

Com entrada e saída de dados

Programada por meio de instruções em cartões perfurados

A sequência de execução dos cartões podia ser alterada

Site 06

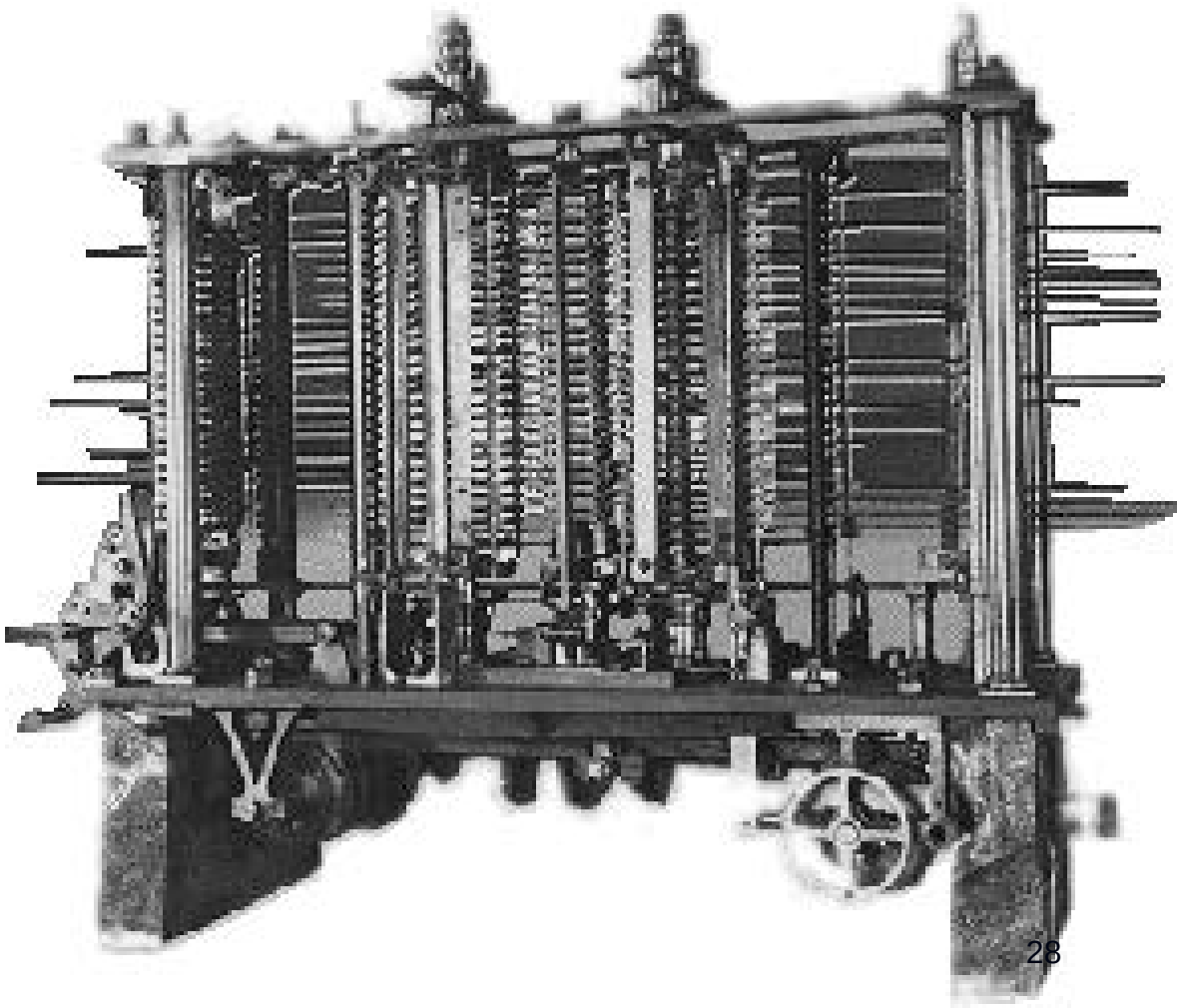


- **Máquina analítica:** concepção de um computador mecânico, em 1833, pelo inglês Charles Babbage

Exigia tecnologia mecânica muito avançada para a época

Não foi efetivamente construída naquele século

Site 06

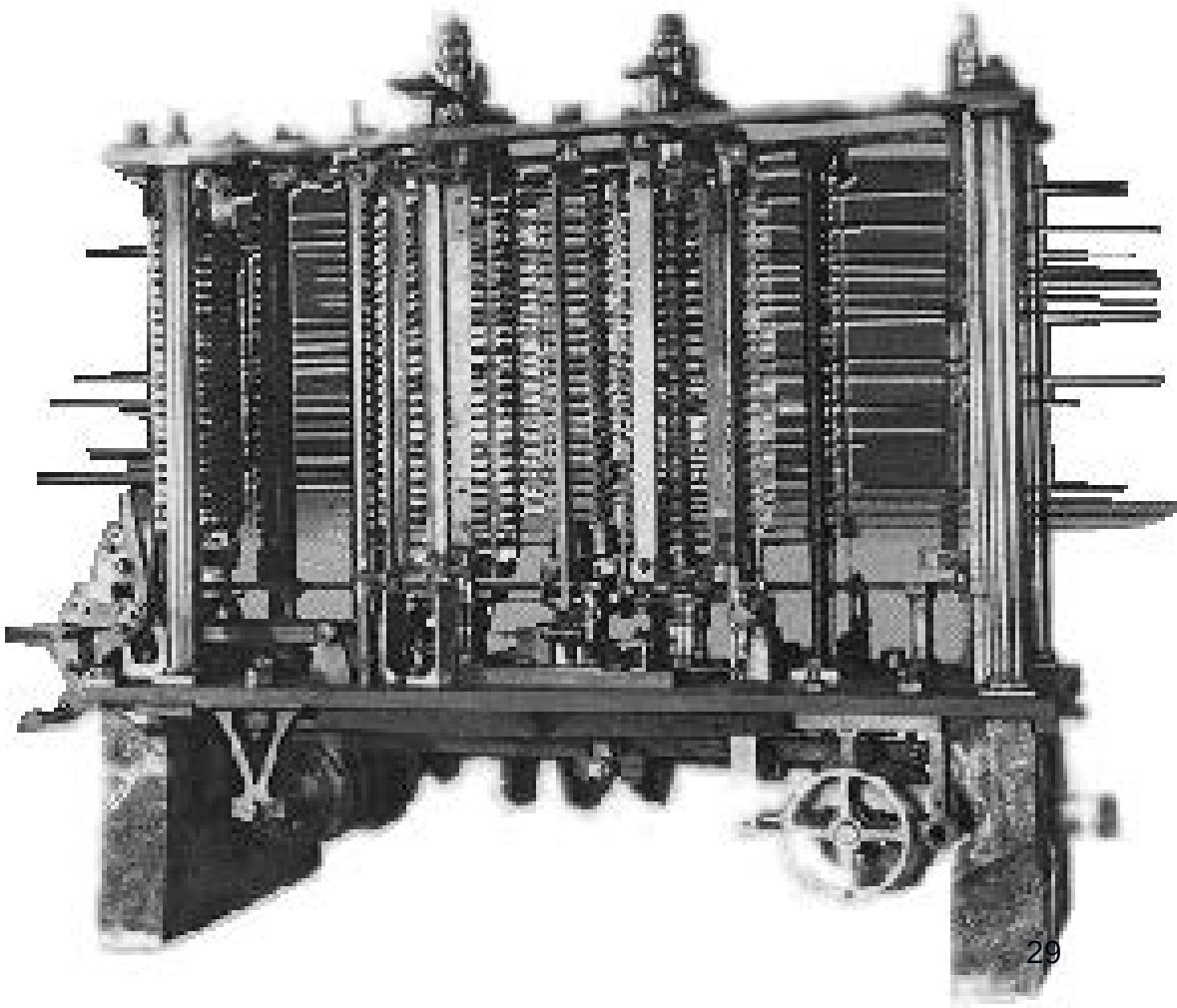


- **Máquina analítica:** concepção de um computador mecânico, em 1833, pelo inglês Charles Babbage

Na década de 1980, pelo seu valor histórico, um grupo resolveu construí-la

Ela funcionou perfeitamente

Site 06

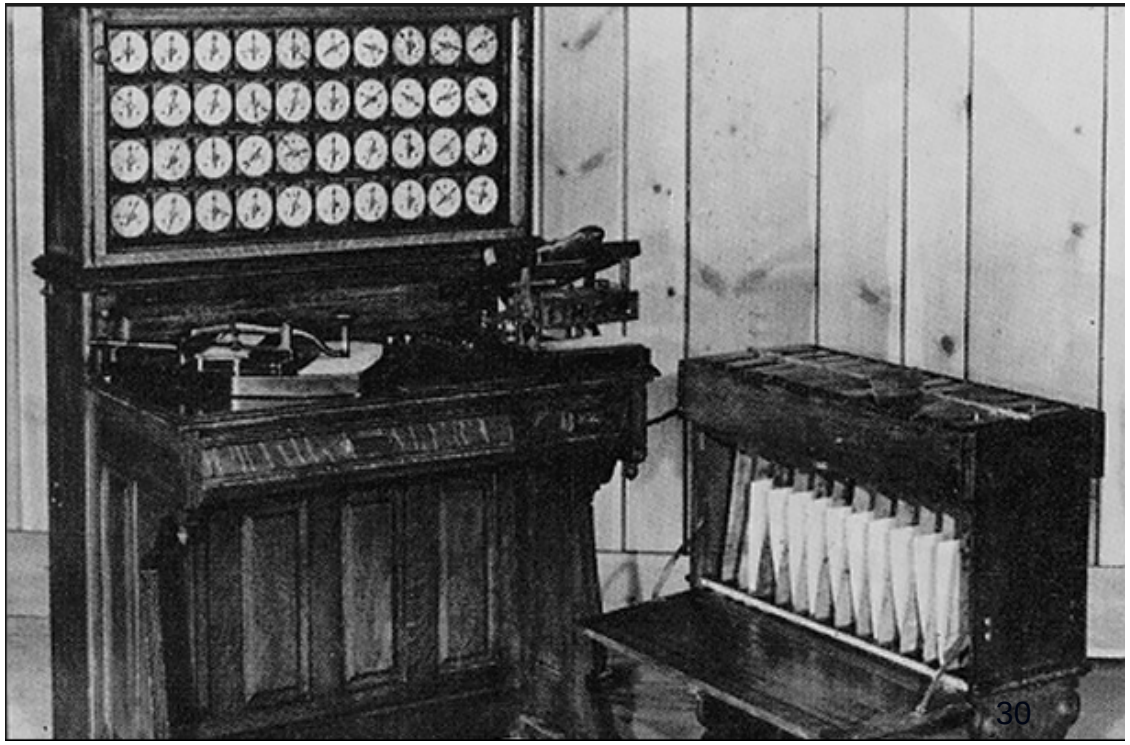


- **Máquina recenseadora:** construída em 1886, por Herman Hollerith, funcionário do Departamento de Recenseamento dos Estados Unidos

Utilizava cartões perfurados que, lidos eletricamente:

- **Controlavam o funcionamento**
- **Continham dados de entrada**
- **Eram usados para imprimir resultados**

Site 07



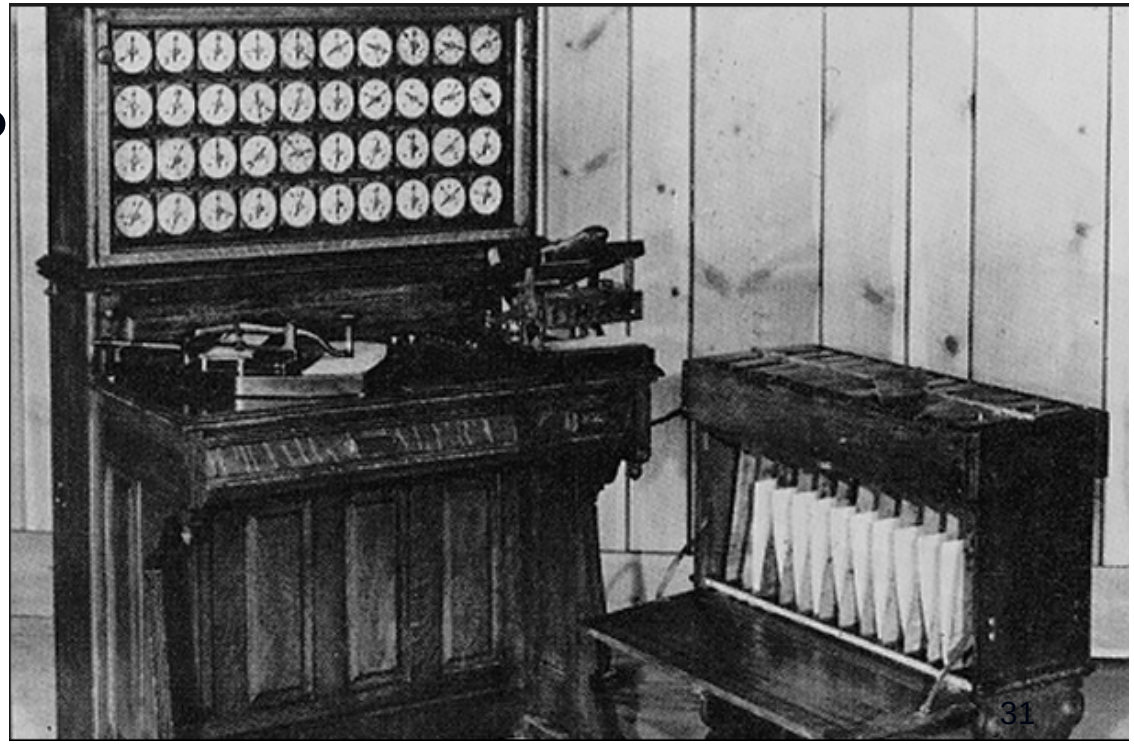
- **Máquina recenseadora:** construída em 1886, por Herman Hollerith, funcionário do Departamento de Recenseamento dos Estados Unidos

Cartões perfurados para programas e dados foram usados por muito tempo

Só foram substituídos pelo teclado e vídeo

Herman Hollerith foi um dos fundadores da International Business Machines (IBM) em 1924

Site 07



- **Computadores eletrônicos analógicos:** utilizam grandezas elétricas como tensão e corrente para modelar um problema a ser resolvido

  - Constituídos de amplificadores, filtros e circuitos somadores, integradores e diferenciadores, além de outros

  - Eram o estado da arte dos computadores antes da Segunda Guerra Mundial

  - Até a década de 1960, competiam com os computadores digitais

- **Computadores eletrônicos analógicos:**

  - Tinham vantagem frente aos primeiros computadores digitais, pois eram capazes de resolver problemas mais complexos

  - Nessa época, os programas de computador não eram ainda muito populares

- **Computadores eletrônicos analógicos:**

  - A solução é apresentada em forma de curvas em gráficos; os resultados são exatos

  - Números reais não são representados exatamente nos computadores digitais; é uma representação discreta

- **Computadores eletrônicos analógicos:**

  - Ao contrário dos computadores digitais modernos, computadores analógicos não são muito flexíveis

  - Precisam ser reprogramados manualmente para trocar o problema em que vão trabalhar

- **Computadores eletrônicos analógicos:**

  - Com o desenvolvimento dos transistores, os computadores digitais começaram a imperar

  - Mas, até hoje, os computadores analógicos são usados na resolução de alguns problemas específicos

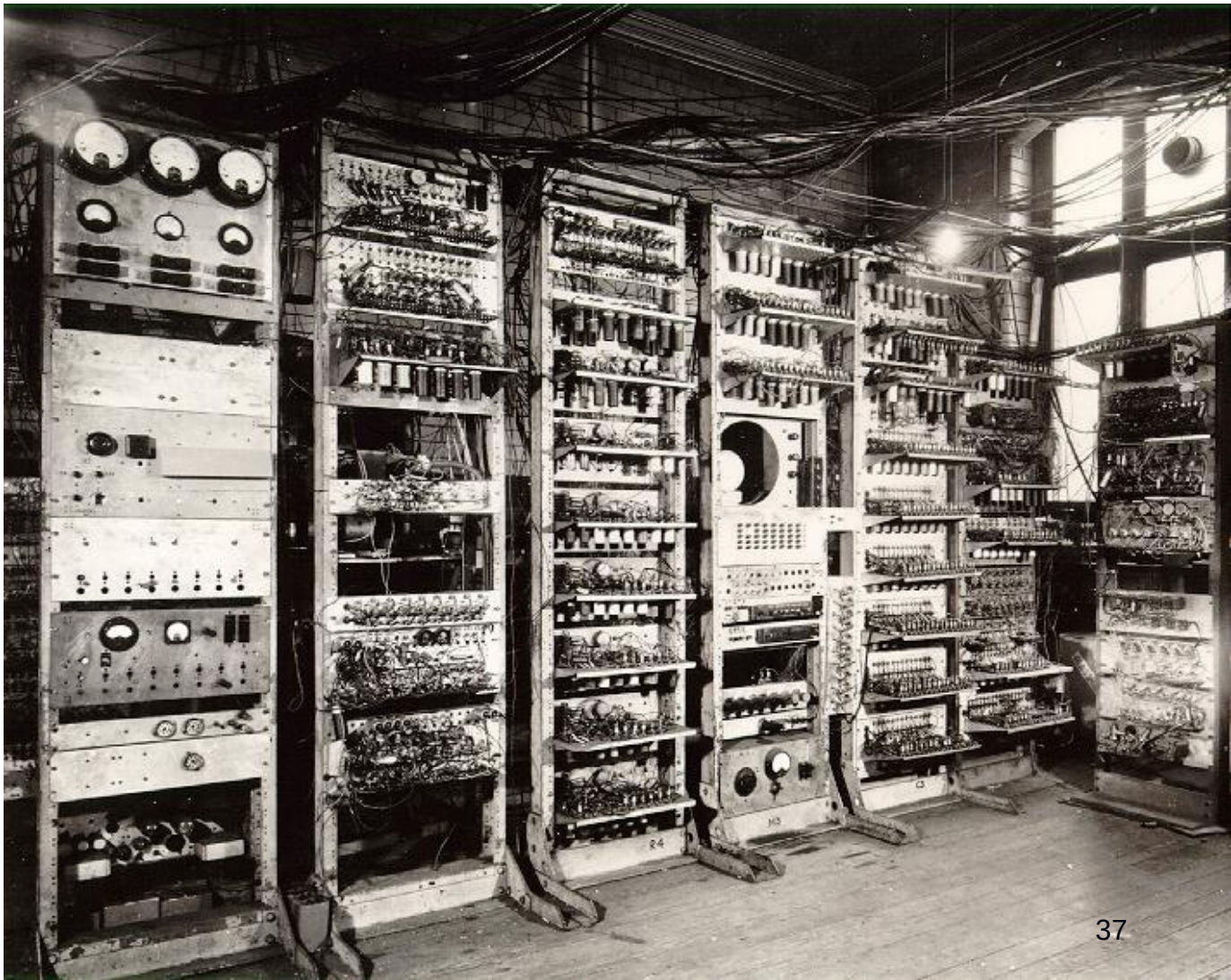
- **Mark I:** primeiro computador digital eletro-mecânico; construído entre 1937 e 1944, por Howard Aiken, na Universidade de Harvard

Site 08

Utilizava relés e válvulas eletrônicas

Media 2,5 m de altura e 18 m de comprimento

Gerava imensa quantidade de calor



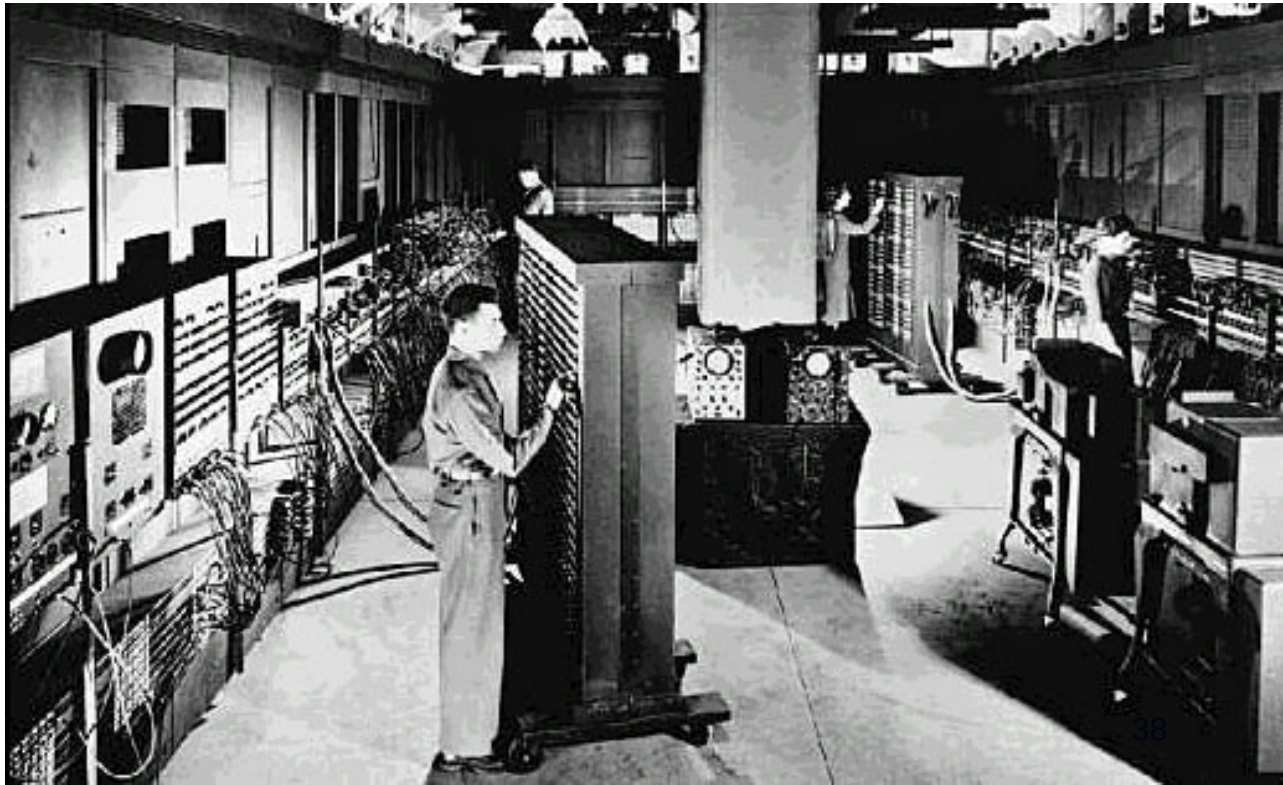
- **ENIAC (Electronic Numeric Integrator and Calculator):** primeiro computador eletrônico digital

Construído em 1946, pelos cientistas norte-americanos John Presper Eckert e John Mauchly

Site 09

Chegava a ser, em algumas operações, mil vezes mais rápido que o MARK I.

Ocupava 170 m^2, pesava 30 toneladas, consumia 150 Kwatts



0.2.2 – Evolução da tecnologia eletrônico-digital

Desde o ENIAC, houve um desenvolvimento acelerado da tecnologia eletrônico-digital:

- **Válvulas eletrônicas:**

  Dispositivos que conduzem a corrente elétrica num só sentido

  Dissipam muito calor

  Quebram após não muitas horas de uso



Site 10

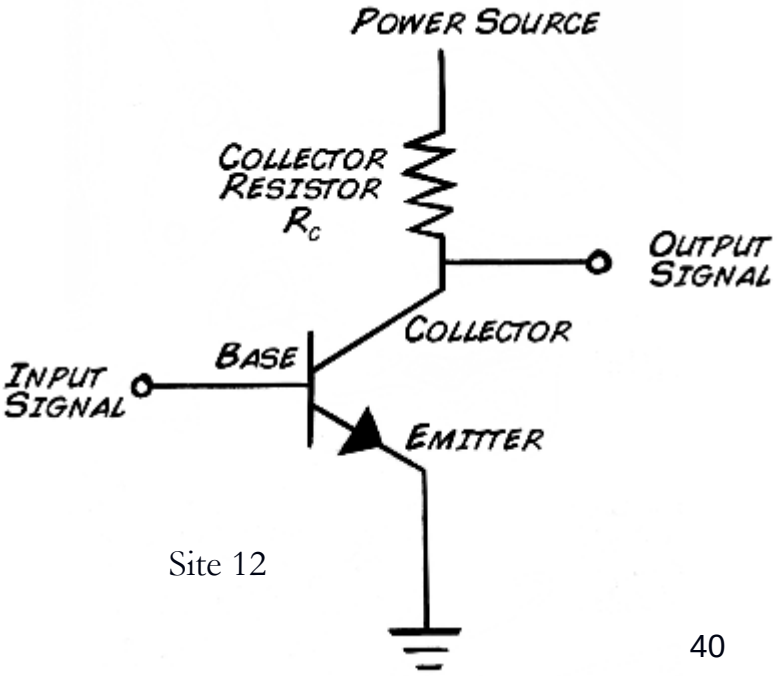
Transistores:

Site 11



Utilizados principalmente como amplificadores e interruptores de sinais elétricos

Têm o mesmo papel das válvulas porém são bem menores e dissipam muito menos calor



Site 12

- **Circuitos integrados:** vários transistores e outros componentes eletrônicos em um só circuito



Site 13

Site 13

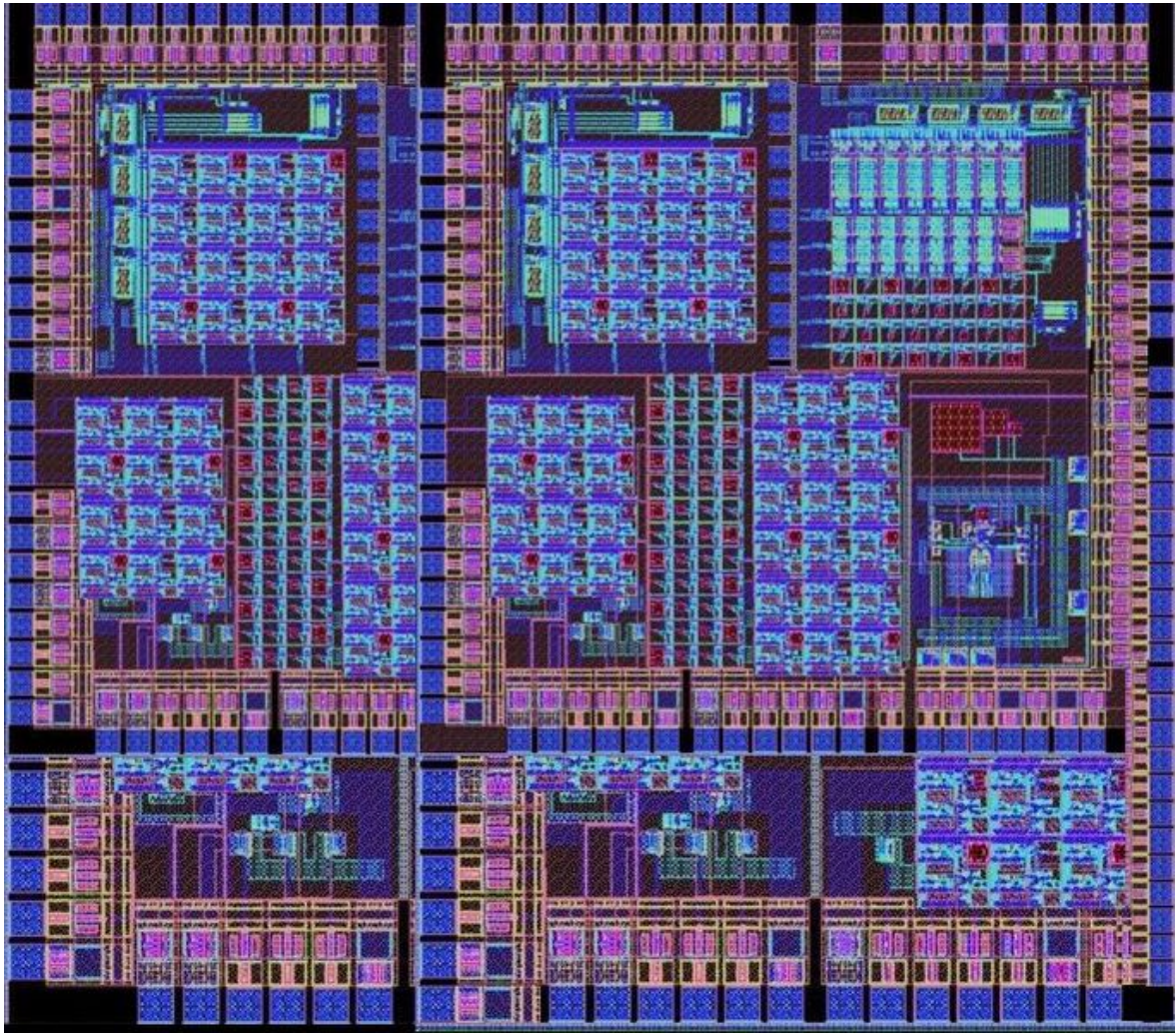


Imagem aumentada mais de 1000 vezes

Usados hoje em quase todos os equipamentos eletrônicos

Revolucionaram o mundo da eletrônica

- **Circuitos integrados:** vários transistores e outros componentes eletrônicos em um só circuito



Site 13

Site 13

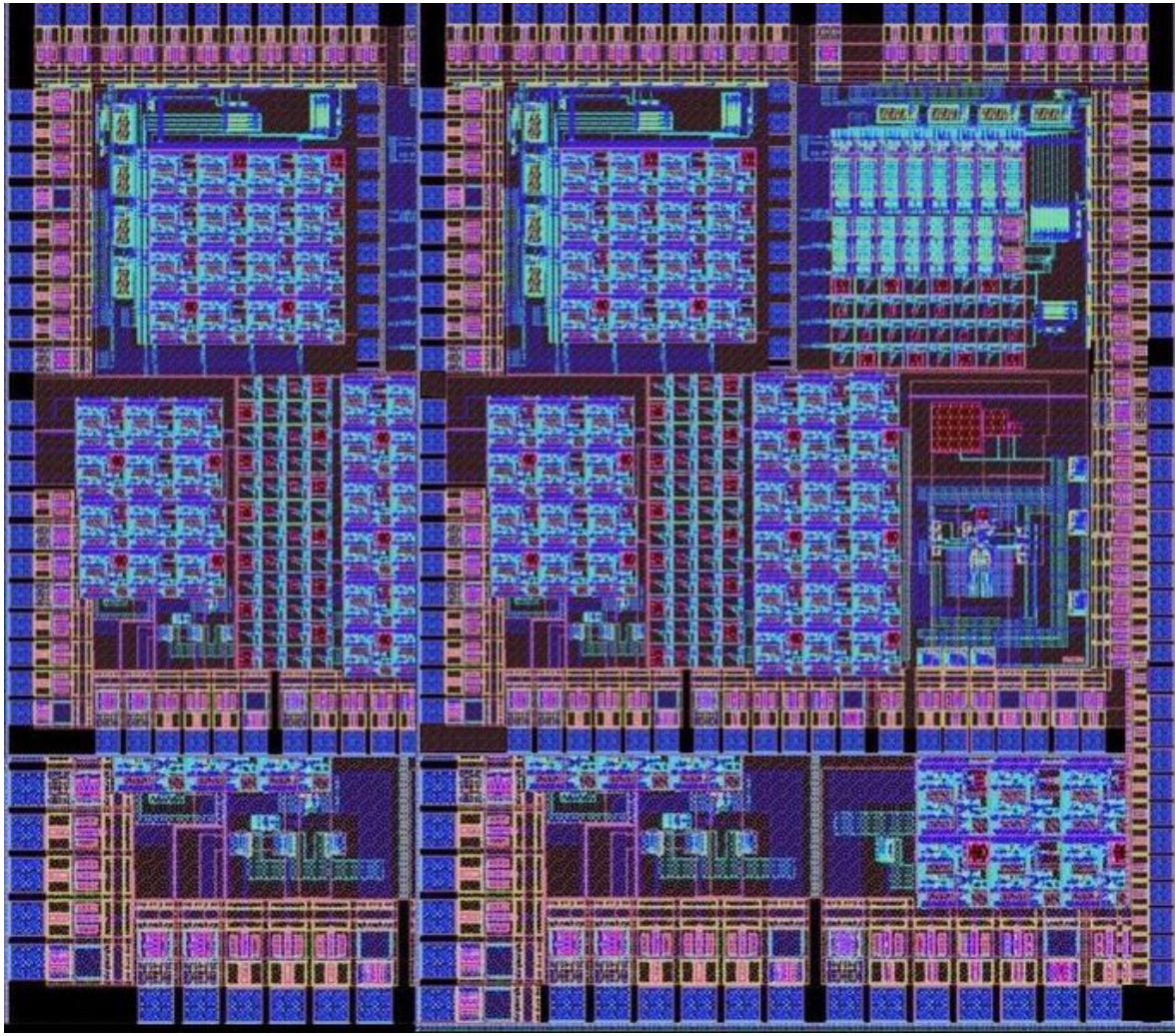


Imagem aumentada mais de 1000 vezes

Invenção de Jack Kilby da Texas Instruments, em 1958

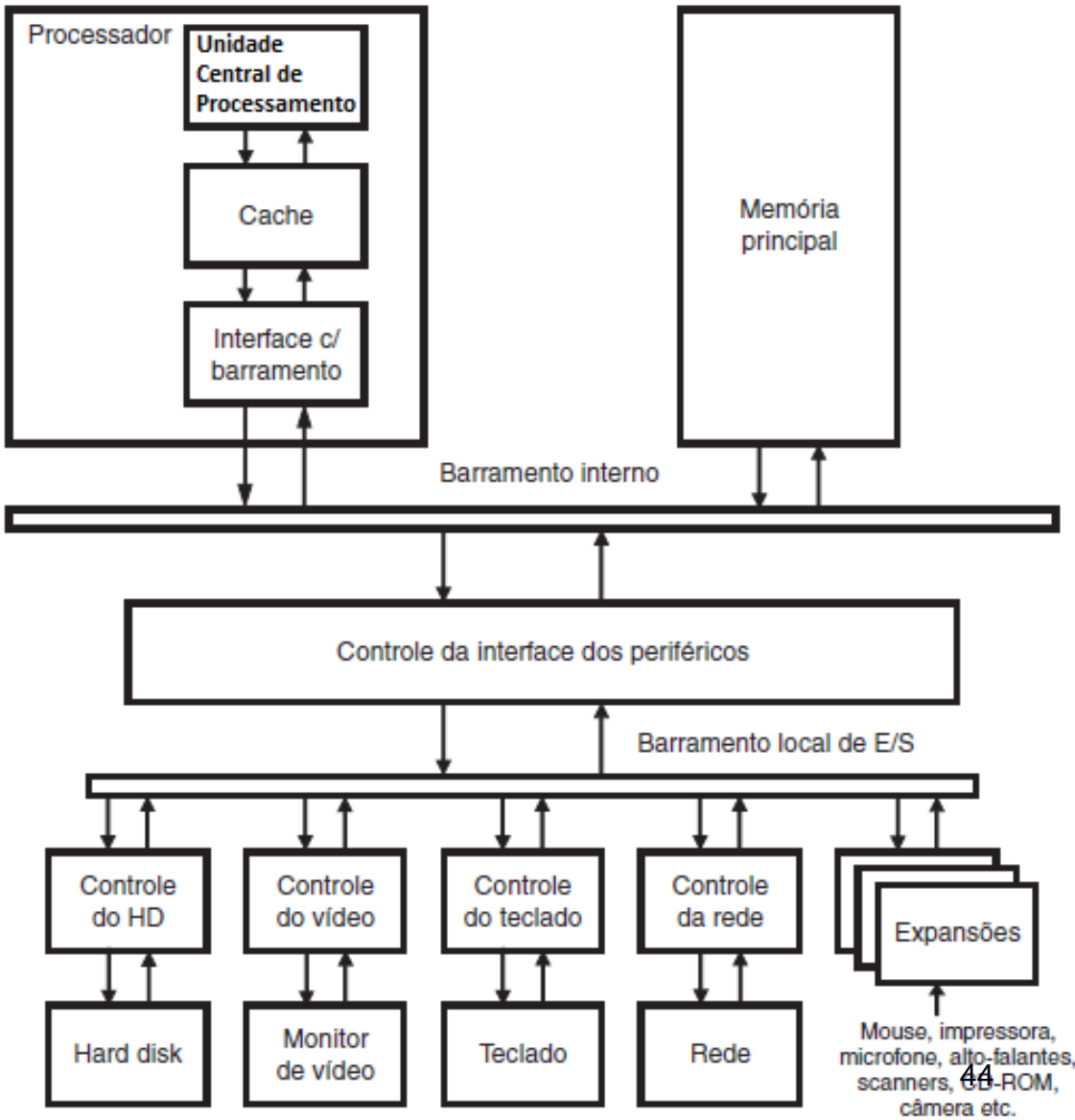
Em 2000, Kilby ganhou o Prêmio Nobel por essa invenção

- **Circuitos integrados:** escalas (Wikipedia)

Abreviatura e época	Denominação	Nº de transistores
SSI 1958	Small Scale Integration	10 a 99
MSI anos 60	Medium Scale Integration	100 a 999
LSI anos 70	Large Scale Integration	1.000 a 9.999
VLSI anos 80	Very Large Scale Integration	10.000 a 99.999
ULSI anos 90	Ultra Large Scale Integration	100.000 a 999.999
SLSI anos 2000	Super Large Scale Integration	1.000.000 a 10.000.000

- **Microprocessadores:** circuitos integrados coordenadores de todas as atividades de um computador

Esquema de um computador pessoal moderno ⟶



Livro 01

Microprocessadores:

Site 14

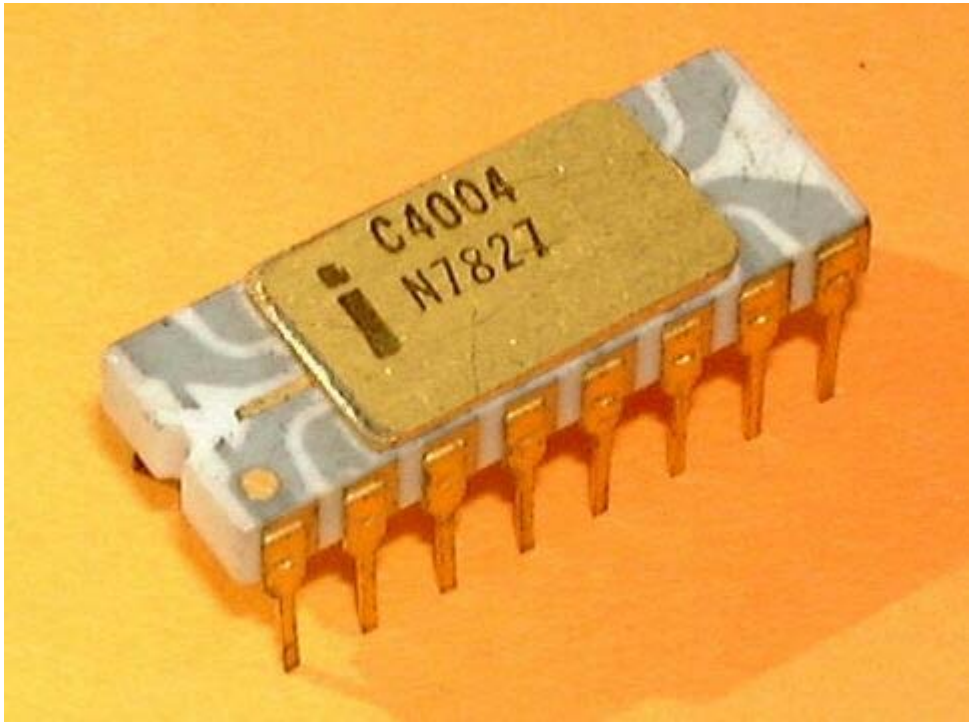
4004: primeiro microprocessador comercial

Inventado pela Intel em 1971

Aritmética binária de 4 bits

Clock de 740 KHz

Possuia 2.300 transistores



- **Microprocessadores:**

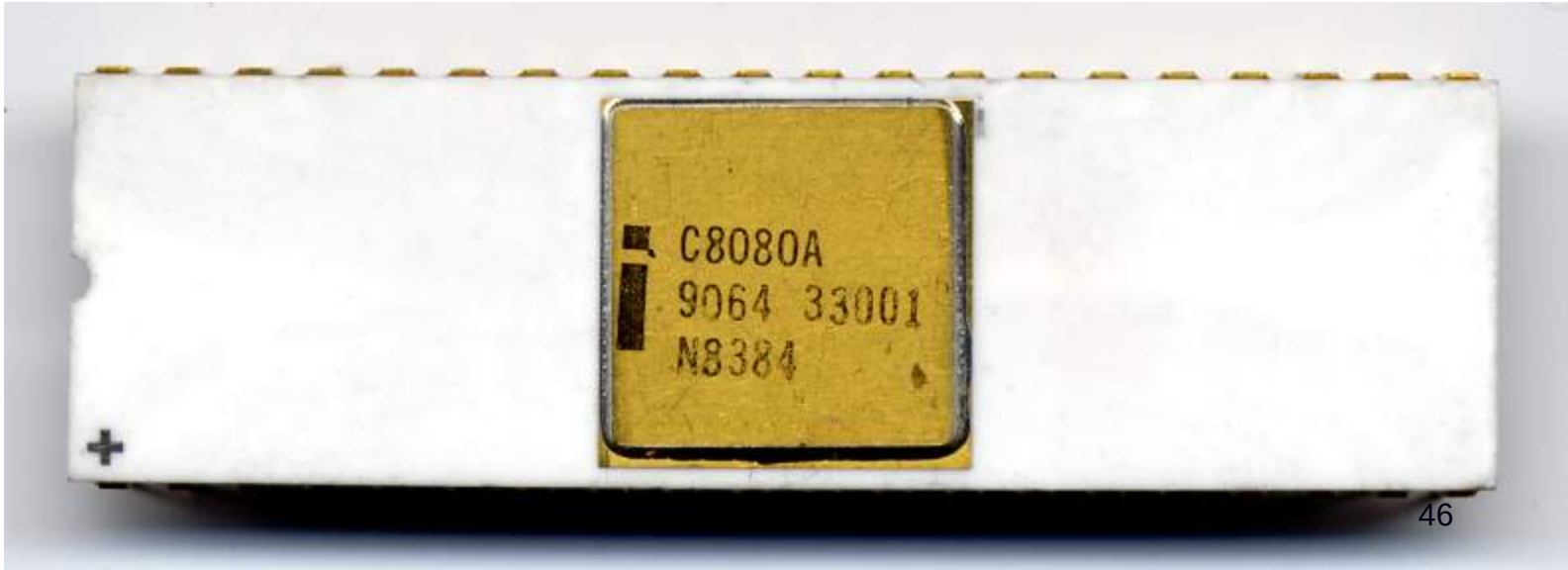
8080: microprocessador criado pela Intel em 1974

Aritmética binária de 8 bits

Clock de 2 MHz

Possuia 6.000 transistores

Site 15



Microprocessadores:

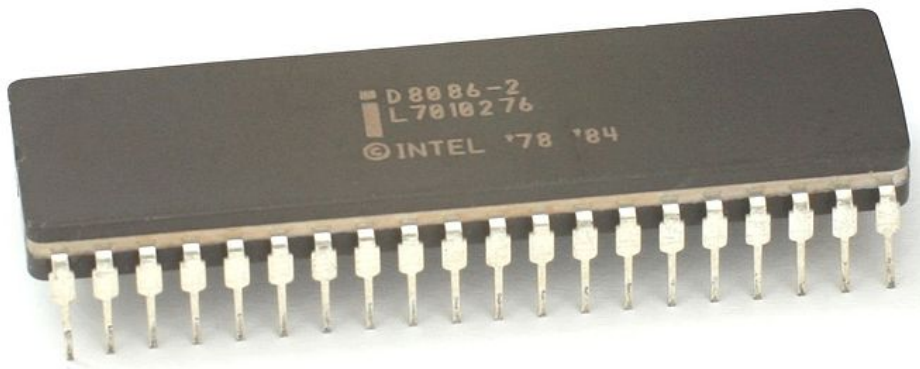
8086: microprocessador criado pela Intel em 1978

Aritmética binária de 16 bits

Clock de 5 a 10 MHz

Possuia 20.000 transistores

Site 16



- **Microprocessadores:**

80386: microprocessador criado pela Intel em 1986

Aritmética binária de 32 bits

Clock de 33 MHz

Possuia 275.000 transistores

Site 17



- **Microprocessadores:**

Pentium IV: microprocessador criado pela Intel em 2001

Site 18

Aritmética binária de 64 bits

Clock de 1.3 a 2.0 GHz

Possui cerca de 5.500.000 transistores



- **Novos microprocessadores da Intel:**
  - **Core 2 Duo:** 2 processadores (núcleos) em 1
  - **i3:** 2 processadores (núcleos) em 1; emula 4 núcleos
  - **i5:** 2 e 4 processadores (núcleos) em 1; emula 4 núcleos
  - **i7:** 4 e 6 processadores (núcleos) em 1; emula 8 e 12 núcleos

- **Microprocessadores, outras aplicações:**
  - **Processadores digitais de sinais:** processam sinais de áudio, vídeo, etc.; por exemplo, em aparelhos de CD, DVD e televisores digitais
  - **Microcontroladores:** embarcados em dispositivos (geralmente produtos comercializados) para controlar suas ações
  - **Processadores gráficos:** utilizados em placas de vídeo para fazer computação gráfica

Mais avanços na tecnologia eletrônico-digital:

- **Fibras óticas:**
  - Podem substituir com vantagens os fios de cobre
  - Não são susceptíveis à interferência eletromagnética
  - Aplicadas na medicina (endoscopias por exemplo)
  - Aplicadas em telecomunicações (principalmente na **Internet**)

Mais avanços na tecnologia eletrônico-digital:

- **Comunicação sem fio –** grande variedade de aplicações:
  - Controle remoto de equipamentos elétricos e eletrônicos
  - Teleprocessamento
  - Comunicações via satélite
  - Redes locais, Internet e telefonia celular

Mais avanços na tecnologia eletrônico-digital:

- **Raio laser –** grande variedade de aplicações:
  - Cirurgias e fisioterapia
  - Comunicação por fibras ópticas e leitura de códigos de barra
  - Computadores, aparelhos de som e imagem

Mais avanços na tecnologia eletrônico-digital:

- **Supercondutividade:**
  - Certos materiais, quando se esfriam a temperaturas extremamente baixas, conduzem corrente sem resistência nem perdas
  - Isso tem sido usado na pesquisa de circuitos eletrônicos cada vez mais rápidos

0.2.3 – Impactos na sociedade

- Os computadores tornaram-se:
  - Muito menores
  - Muito mais poderosos
  - Muito menos consumidores de energia
  - Muito menos dissipadores de calor

- Hoje no mercado:
  - **Poderosos laptops e desktops:** muito mais potentes que volumosos computadores de décadas passadas
  - **Supercomputadores:** cada um com milhares ou até milhões de processadores
- **Redes de computadores:**
  - Locais caseiras e empresariais
  - Internet, a rede mundial

- No início, os computadores eram usados apenas para manipular números

- Pouco tempo depois, o processamento **não numérico** tornou-se muito mais importante que o processamento **numérico**

- Hoje os computadores processam informações de diversas espécies: números, textos, sons, imagens, fenômenos físicos, químicos, biológicos e sociais

A **comunicação humano-computador** evoluiu muito:

- Programação no ENIAC, em 1946:
  - Entrada e saída de dados usava cartões perfurados
  - Programas eram introduzidos mediante manipulação de conexões eletrônicas
  - Instalação de um novo programa demorava um ou dois dias

A **comunicação humano-computador** evoluiu muito:

- Para o EDSAC (Electronic Delay Storage Automatic Calculator), em 1949:
  - Descobriu-se que programas poderiam ser lidos, tal como os dados de entrada, e armazenados na memória (conceito de **programa armazenado**)
  - Os programas passaram a ser escritos em cartões perfurados e instalados na memória por meio de leitora de cartões

Novos meios de programar computadores foram surgindo:

- Os resultados passaram a ser impressos em papel, por impressoras
- Programas passaram a ser introduzidos por teclado, ficando armazenados em arquivos
- Os resultados passaram a ser exibidos em monitores de vídeo

Novos meios de programar computadores foram surgindo:

- Ambientes de programação foram criados, disponibilizando ao programador:
  - Confortáveis editores de textos, manipuladores de arquivos, compiladores
  - Botões a serem acionados pelo mouse para salvar arquivos, compilar, executar e corrigir programas, usar a impressora, etc.

Outras atividades, além de programar:

- Edição de documentos, figuras e apresentações: Word, Adobe, Paint, Power-Point, etc.
- Manipulação de tabelas para contabilidade: Excel
- Navegadores da Internet: Explorer, Mozilla, etc.
- Manipulação de arquivos de áudio e vídeo: iTunes, Emule, Ares, MP3Gain, Nero, Roxio, YouTube, etc
- Interfaces para controle de processos
- Etc.

Hoje os computadores são utilizados por pessoas totalmente leigas em Ciência da Computação

Hoje os sistemas computacionais estão integrados em praticamente todas as atividades da sociedade:

Atividades técnico-científicas	Literatura e artes
Medicina e saúde	Cinema
Indústria e comércio	Música
Atividades administrativas e governamentais	Pesquisa escolar e caseira (serve como enciclopédia sempre atualizada)
Educação	Entretenimento

0.3 – Relevância da Introdução à Programação

- A maioria dos ambientes computacionais de trabalho e dos programas científicos são escritos em linguagens de programação, tais como **Fortran**, **Pascal**, **C**, **Java**, etc.

- Não raro, um engenheiro necessita desenvolver programas ou ambientes para novas aplicações, em sua área de atuação

- Diversas áreas da Eletrônica trabalham diretamente com softwares para fazer **interface** com dispositivos eletrônicos ou para serem neles **embarcados**

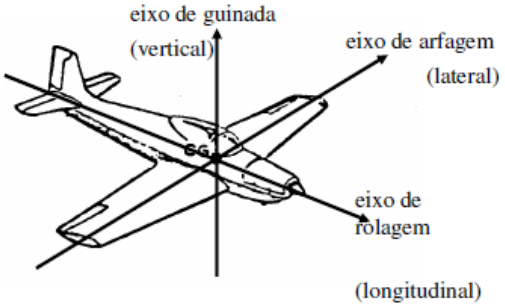
- A Matemática, a Física e várias áreas da Engenharia trabalham com **simulações** na busca da melhor solução para um problema

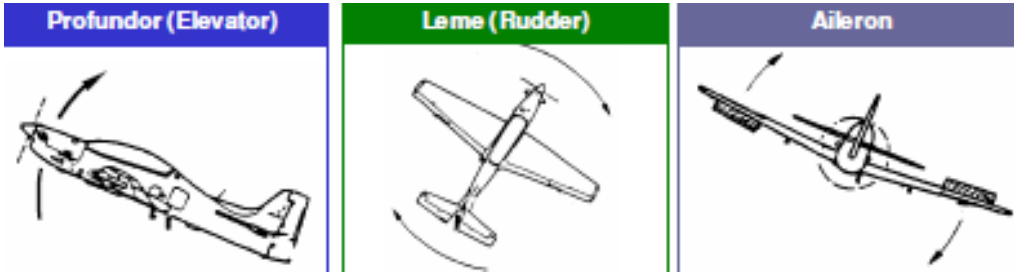
- Na área financeira, a programação de macros para o **Excel** dá um grande diferencial

- Muitos **aviões** trabalham com softwares embarcados que controlam diversos de **seus sub-sistemas**

Sub-sistemas de um avião









- O conhecimento dos **princípios de programação** numa das referidas linguagens é portanto imprescindível para um engenheiro

- Este manual ensina os princípios mais elementares de **computadores**, **algoritmos** e **programas**, integrantes de disciplinas introdutórias de **programação de computadores**

- Para alcançar sucesso numa disciplina como esta, é preciso muita **dedicação** e realizar **muitos exercícios**

- No entanto, apenas **um semestre de treinamento** em programação é insuficiente para um engenheiro

- Muitos problemas de Engenharia encontram soluções nos vários ramos da Matemática
- Alguns deles exigem quantidade descomunal de operações matemáticas
- A utilização de sistemas computacionais para solucioná-los é portanto indispensável
- Disciplinas comumente denominadas de **Cálculo Numérico** ou **Matemática Computacional** apresentam algoritmos para realizar essas operações por computador, de forma eficiente
- São disciplinas comumente inseridas em **cursos básicos** de Engenharia

- Muitas atividades de Engenharia envolvem a manipulação de quantidades imensas de informações

- As formas com que essas informações se interligam entre si apresentam diversos níveis de complexidade

- Um engenheiro deve saber decidir qual o melhor modelo de armazenamento para um sistema de informações e os melhores algoritmos para gerenciá-lo

- Disciplinas comumente denominadas **Algoritmos e Estruturas de Dados** aborda essa necessidade

- É uma disciplina que está integrada em alguns **cursos básicos** de Engenharia

Por que a Linguagem C como primeira linguagem de programação?

- Há especialistas que a consideram facilitadora de **indisciplina** e de confecção de programas **desestruturados** e confusos

- Preferem linguagens como **Pascal** que inibem um pouco essa indisciplina

Por que a Linguagem C como primeira linguagem de programação?

- Outros especialistas preferem **Java** ou **C++**
- Para inserir logo o aluno no combate à **crise de software**
  - Rápido crescimento da **demanda** por software
  - Rápido crescimento da **complexidade** dos problemas a serem resolvidos
  - Carência de técnicas para o desenvolvimento de sistemas que **funcionem adequadamente** ou possam ser **validados**
- **Java** é a linguagem básica para transações na **Internet**

Por que a Linguagem C como primeira linguagem de programação?

- No ITA, em 1998, **Pascal** foi substituída pela **Linguagem C**
- **Pascal** havia caído em desuso
- Os mesmos princípios de **programação estruturada** podem ser aplicados em ambas as linguagens

Por que a Linguagem C como primeira linguagem de programação?

- Num Curso de Bacharelado em Ciência da Computação, **Java** é uma forte opção
- Programas em **Java** são mais ineficientes do que em **C** – isso é inconveniente em **Engenharia**
- **Java** e **C++** usam programação **orientada a objetos**; isso é bem difundido só em **Engenharia de Computação**
- Outras modalidades de Engenharia têm a programação **estruturada** bem arraigada

Por que a Linguagem C como primeira linguagem de programação?

- **C** possibilita a confecção de **programas eficientes**
- Permite acesso até certo ponto à **estrutura interna** do computador
- Ainda é uma linguagem **largamente utilizada**

0.4 – Programa de Introdução à Programação

Cap. I	Conceitos primários	3.5 semanas
Cap. II	Algoritmos e programas	1.5 semanas
Cap. III	Declarações e comandos de atribuição	1 semana
Cap. IV	Comandos de controle de fluxo	1 semana
Cap. V	Comandos de entrada e saída	1 semana
Cap. VI	Variáveis indexadas	2.5 semanas
Cap. VII	Tipos enumerativos e estruturas	1 semana
Cap. VIII	Subprogramação	1.5 semana
Cap. IX	Ponteiros	1.5 semana
Cap. X	Metodologia top-down com subprogramação	0.5 semana
Cap. XI	Noções de estruturas de dados	1 semana

0.4 – Programa de Introdução à Programação

- **Observação: este é um programa completo para uma disciplina de Introdução à Programação**

- **Este manual cobre os dois primeiros capítulos deste programa:**

Cap. I	**Conceitos primários**	**3.5 semanas**
Cap. II	**Algoritmos e programas**	**1.5 semanas**

- **A divisão em capítulos deste manual é diferente da divisão em capítulos deste programa**

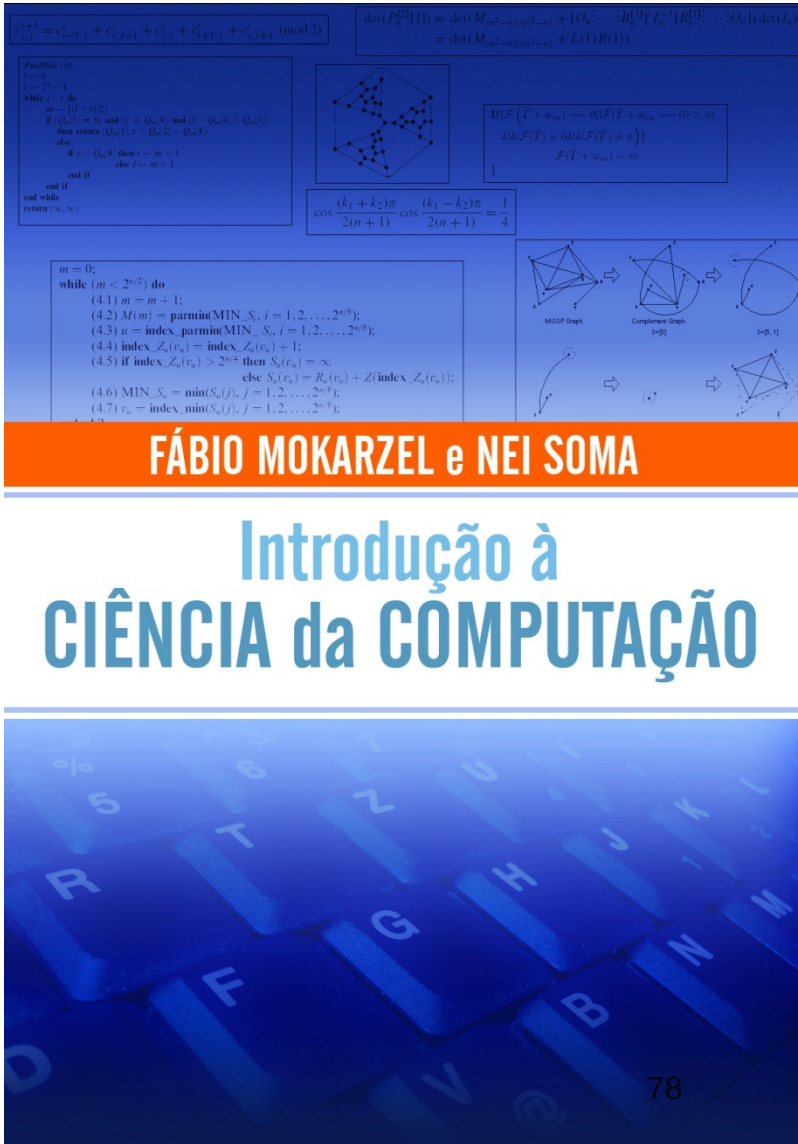
0.5 – Bibliografia

0.5.1 – Livros

Livro 01:

Mokarzel, F.C.; Soma, N.Y.

Introdução à Ciência da Computação. Rio de Janeiro: Campus-Elsevier, 2008.



Espinha dorsal da disciplina

Resultado de muitos anos ministrando a disciplina

0.5 – Bibliografia

0.5.1 – Livros

Livro 02:

Mizrahi, V.V. ***Treinamento em Linguagem C***. 2ª Edição. São Paulo: Pearson, 2008.

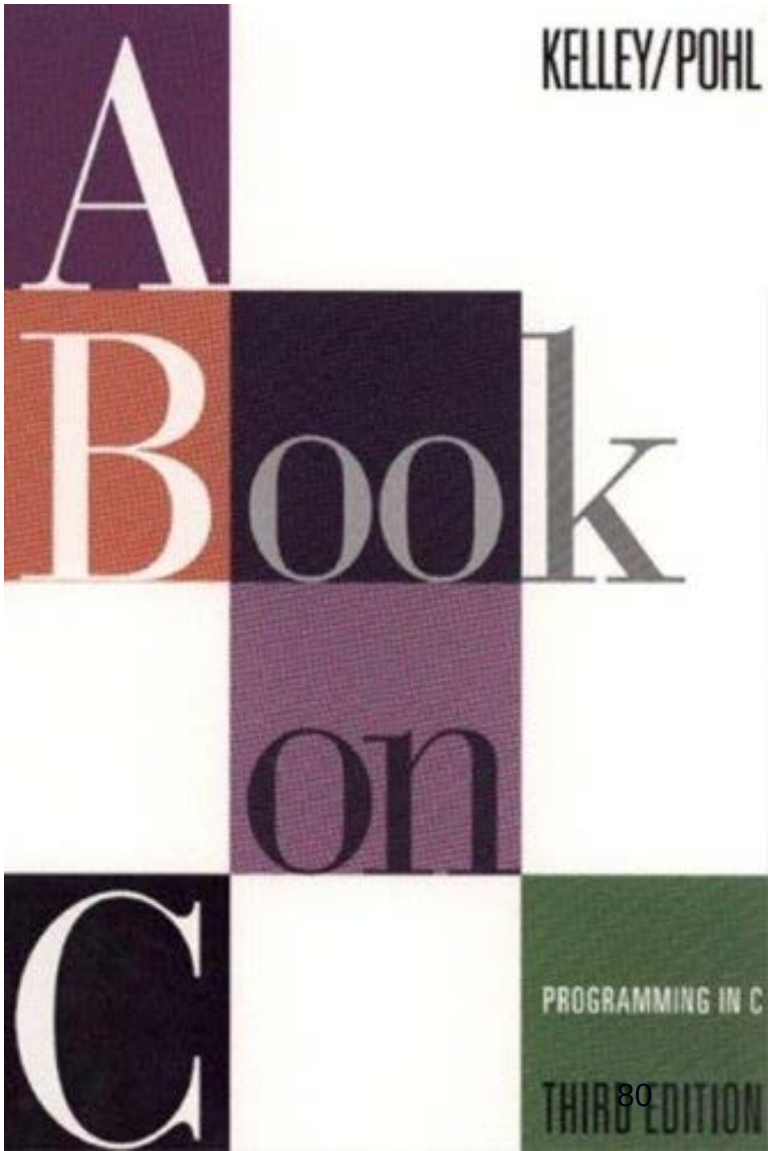


0.5 – Bibliografia

0.5.1 – Livros

Livro 03:

Kelley, A.; Pohl, I. ***A Book on C***. Redwood City: Benjamin Cummings, 1995.



0.5 – Bibliografia

0.5.1 – Livros

Livro 04:

Saliba, W.L.C. ***Técnicas de Programação: uma Abordagem Estruturada***. São Paulo: Makron, 1992.



0.5 – Bibliografia

0.5.1 – Livros

Livro 05:

Schildt, H. *C Completo e Total*. 3ª Edição São Paulo: Makron, 1997.

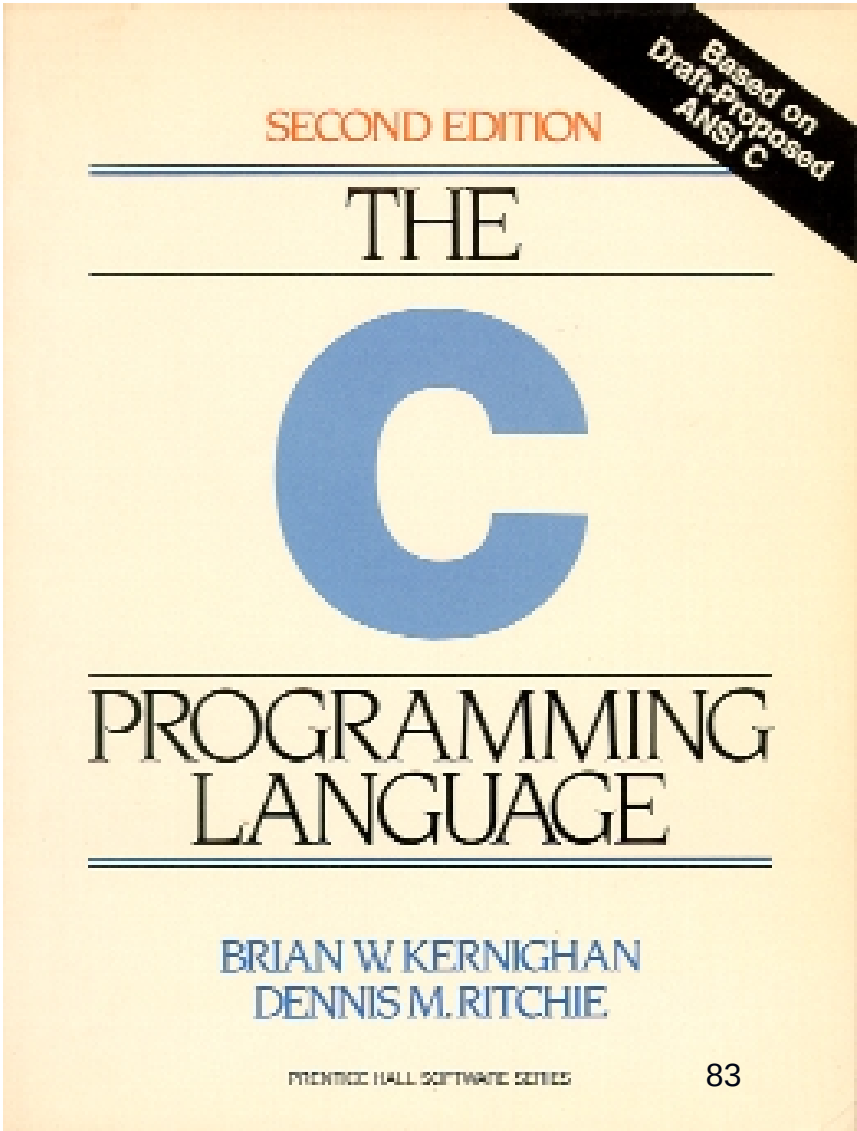


0.5 – Bibliografia

0.5.1 – Livros

Livro 06:

Kernighan B.W.; Ritchie D.M. ***The C Programming Language***. 2nd Edition Englewood Cliffs: Prentice Hall, 1988.



0.5 – Bibliografia

0.5.2 – Slides em Power-Point

PPT 01:

Slides de CES-10 –

Prof. Mokarzel, 2011



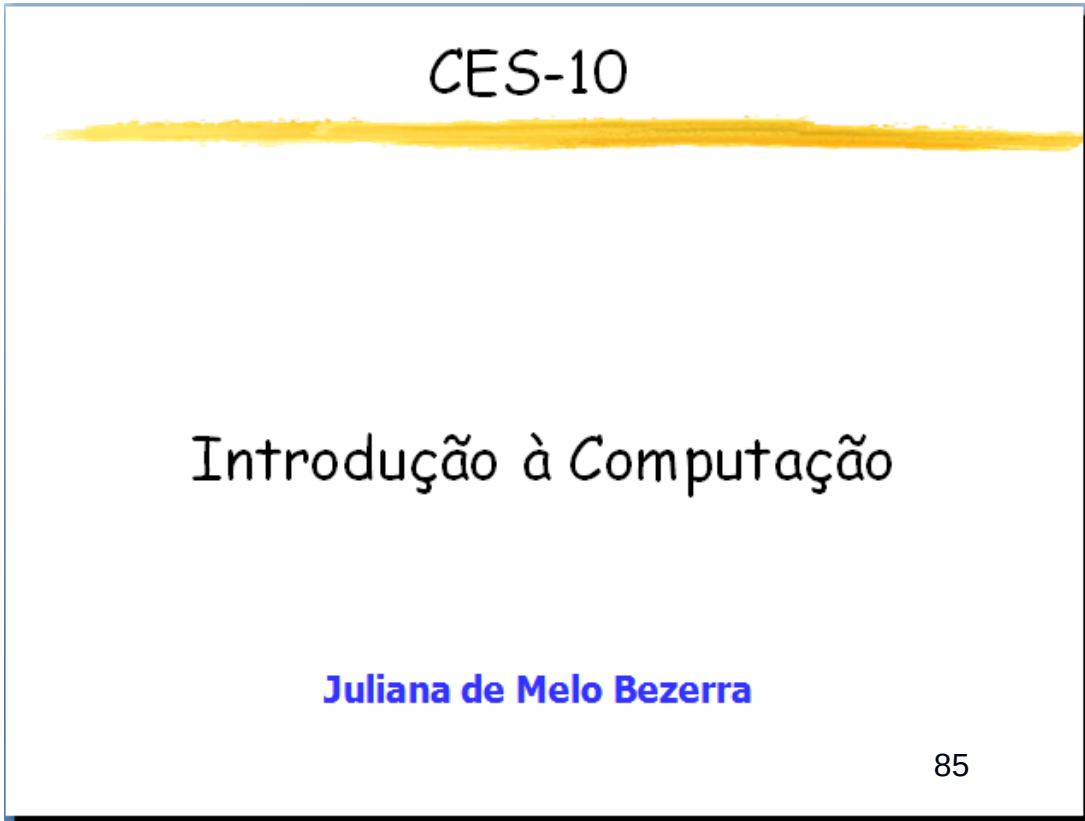
0.5 – Bibliografia

0.5.2 – Slides em Power-Point

PPT 02:

Slides de CES-10 –

Profa. Juliana, 2009



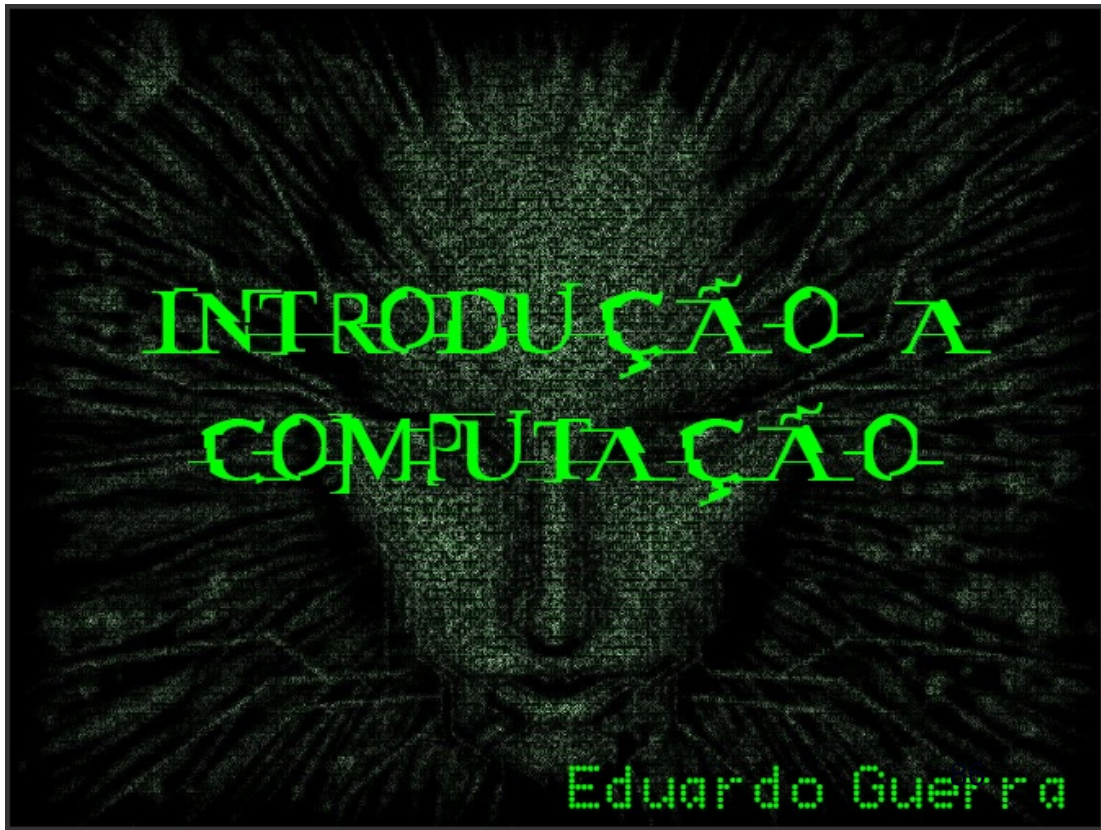
0.5 – Bibliografia

0.5.2 – Slides em Power-Point

PPT 03:

Slides de CES-10 –

Prof. Guerra, 2009



0.5 – Bibliografia

0.5.3 – Sites da Internet

- Site 01:

  https://pt.wikipedia.org/wiki/%C3%81baco

- Site 02:

  http://producao.virtual.ufpb.br/books/camyle/introducao-a-computacao-livro/livro/livro.chunked/ch01s01.html

- Site 03:

  https://pt.wikipedia.org/wiki/La_pascaline

0.5 – Bibliografia

0.5.3 – Sites da Internet

- Site 04:

  https://commons.wikimedia.org/wiki/Category:Leibniz_calculator#/media/File:Leibniz_Stepped_Reckoner_mechanism.png

- Site 05:

  https://wikivisually.com/wiki/Charles_Xavier_Thomas

- Site 06:

  http://www.dsc.ufcg.edu.br/~pet/jornal/marco2014/materias/historia_da_computacao.html

0.5 – Bibliografia

0.5.3 – Sites da Internet

- Site 07:

  http://ahistoriadainformaticaprofmarcos.blogspot.com/p/herman-hollerith.html

- Site 08:

  https://erealityhome.wordpress.com/2008/09/20/1944-mark-i/

- Site 09:

  https: //thenewstack.io/reading-the-manual-for-eniac-the-worlds-first-electronic-computer/

0.5 – Bibliografia

0.5.3 – Sites da Internet

- Site 10:

  https://www.altanatubes.com.br/webstore/?id=1898&t=Valvula-811A-Shuguang-Fisioterapia-por-ondas-curtas

- Site 11:

  https://www.infoescola.com/eletronica/transistor/

- Site 12:

  https://backyardbrains.com/experiments/transistorDesign

0.5 – Bibliografia

0.5.3 – Sites da Internet

- Site 13:

  https://pt.wikipedia.org/wiki/Circuito_integrado

- Site 14:

  http://www.ic.uff.br/~otton/graduacao/informaticaI/computadores_por_imagens.html

- Site 15:

  https://pt.wikipedia.org/wiki/Intel_8080

0.5 – Bibliografia

0.5.3 – Sites da Internet

- Site 16:

  https://www.rockarama.com.br/executivos-da-intel-comemoram-as-quase-quatro-decadas-do-processador-x86/

- Site 17:

  https://en.wikipedia.org/wiki/Intel_80386

- Site 18:

  https://pt.wikipedia.org/wiki/Pentium_4

Computadores, Algoritmos e Programas: Manual Introdutório de Ensino

Capítulo I

Estrutura de um Computador

Capítulo I – Estrutura de um Computador

1.1 – Conceito de computador

1.2 – Modelo hipotético de um computador

1.3 – Unidades básicas de um computador

1.1 – Conceito de Computador

Computadores: equipamentos eletrônicos digitais destinados ao processamento de informações dos mais variados tipos, em altíssima velocidade

No início de sua existência:

- Ocupavam **salas enormes**
- Somente manipulavam **números**
- Seus **circuitos** executavam apenas **soma e subtração de inteiros**
- **Multiplicação** e **divisão** de inteiros e manipulação de **números reais** eram feitas por **programas lentos** e não por circuitos

Ao longo do tempo:

- **Circuitos** cada vez mais poderosos e **espaço** ocupado cada vez menor
- Agilidade no processamento de **números reais**, **textos**, **imagens** e **sons**
- Computadores **pessoais** e **portáteis**
- **Redes de computadores:**
  - Compartilhamento de recursos
  - Informações localizadas **em todo o mundo** acessadas de qualquer lugar, rapidamente

Um computador é capaz de executar enorme **diversidade de tarefas**

- Algumas bem **simples**, como:
  - Solução de problemas aritméticos
  - Ordenação alfabética de nomes
  - Desenho de figuras geométricas

Um computador é capaz de executar enorme **diversidade de tarefas**

- Outras de grande **complexidade**, como:

Controle de processos industriais	Controle de complexos robôs
Execução remota de complicadas cirurgias	Gerenciamento de empresas, bancos, etc.
Detecção de desmatamento	Simulações para Engenharia
Previsão de condições climáticas e catástrofes atmosféricas	Monitoramento de jogos Software embarcado

Por que um **mesmo dispositivo eletrônico** (computador), com os **mesmos circuitos**, executa, em momentos diferentes, trabalhos de natureza totalmente **diversa**?

- É que, em um desses momentos, ele está **programado** para executar uma determinada tarefa

- Em outro, para outra tarefa totalmente **diferente**

Programa:

- Entidade pela qual se consegue **instruir** um computador para que ele **faça o que se deseja**, num dado momento

- Sequência de **instruções** que, ao serem executadas por tal máquina, **realizam** uma determinada tarefa

Hardware:

- Conjunto de todos os circuitos eletrônicos de um computador

Software:

- Conjunto de todos os programas residentes no sistema de memória de um computador

Software:

- **Sistema operacional**: controla e coordena o uso do hardware entre os vários programas aplicativos para os vários usuários

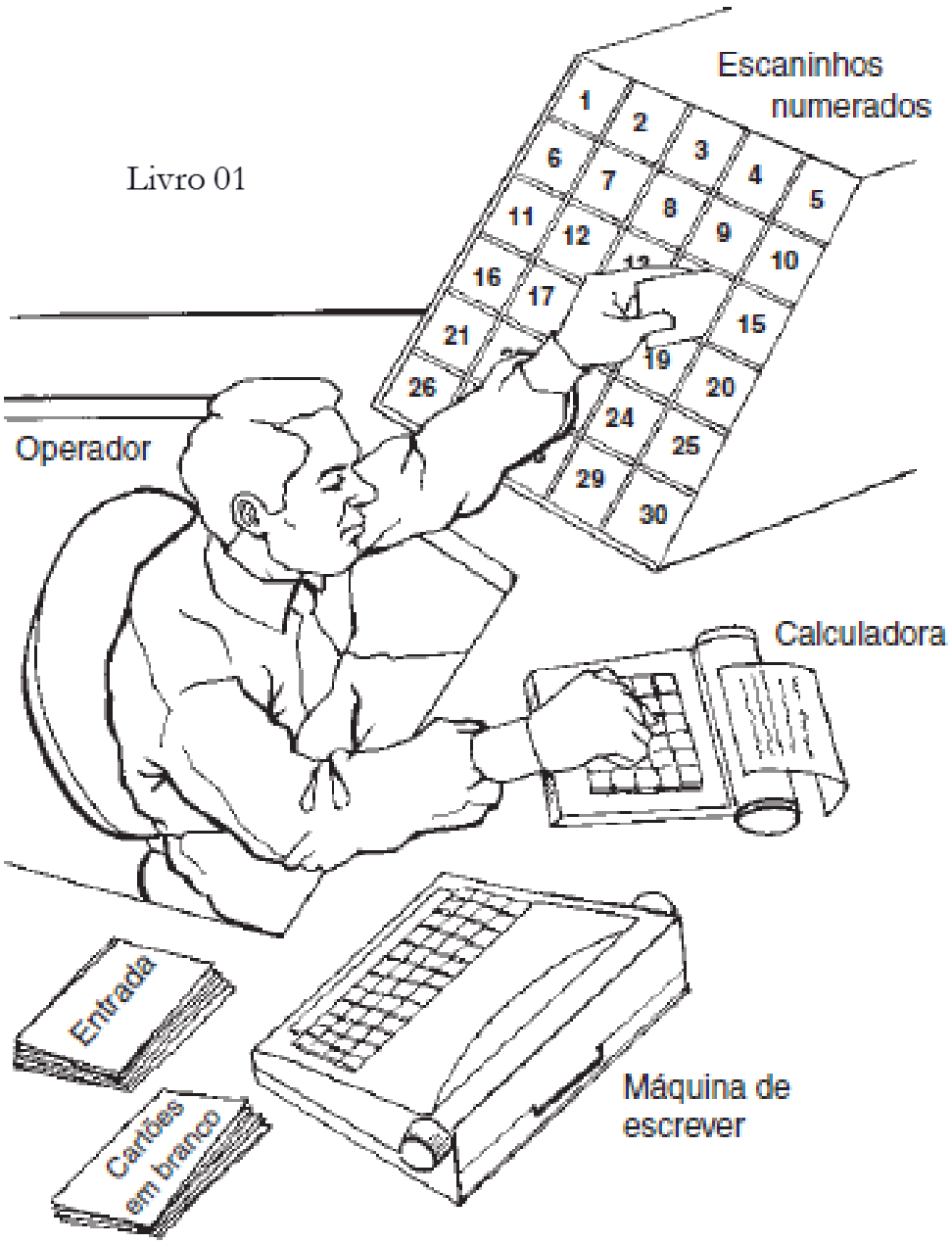
- **Software básico:** programas que auxiliam os usuários em suas tarefas básicas (compiladores, Word, Excel, Power-Point, etc.)

- **Utilitários em geral:** programas instalados para propósitos específicos e/ou particulares (eMule, iTunes, Nero, editores de fotos, etc.)

- **Programa feitos pelos usuários:** programas-tarefas escolares, projetos de pesquisa, etc.

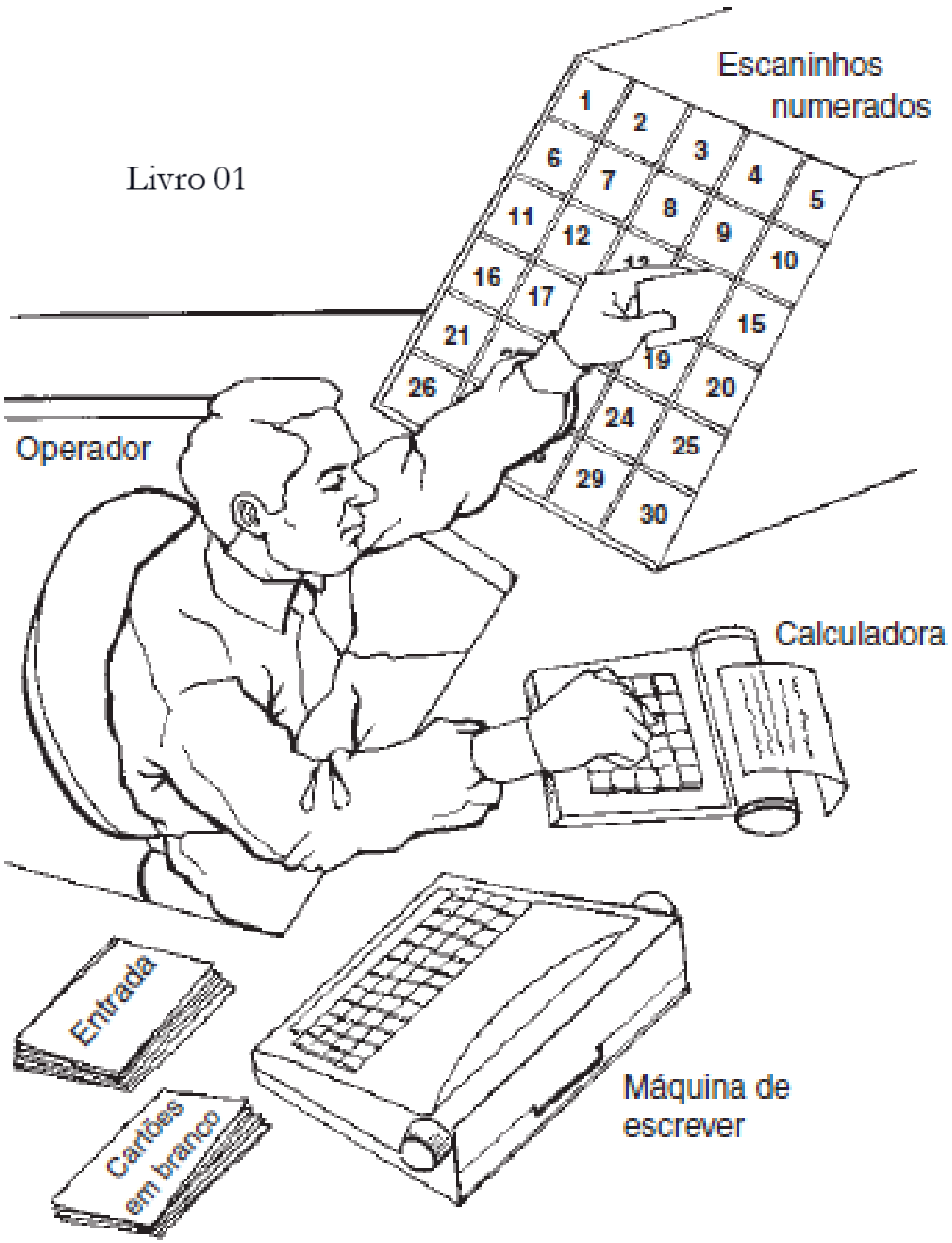
1.2 – Modelo hipotético de um computador

- Os computadores modernos têm **sofisticada** estrutura interna
- No entanto, alguns **princípios** de seu funcionamento podem ser explicados de forma relativamente **simples**
- A seguir, um **modelo hipotético** de um computador, formado por elementos presentes em **escritórios**, antes do domínio dos computadores



O operador trabalha com:

- Calculadora
- Máquina de escrever
- Estante com 30 escaninhos numerados
- Pilha de cartões de entrada
- Pilha de cartões em branco



Em cada escaninho um só cartão, no qual pode estar escrito:

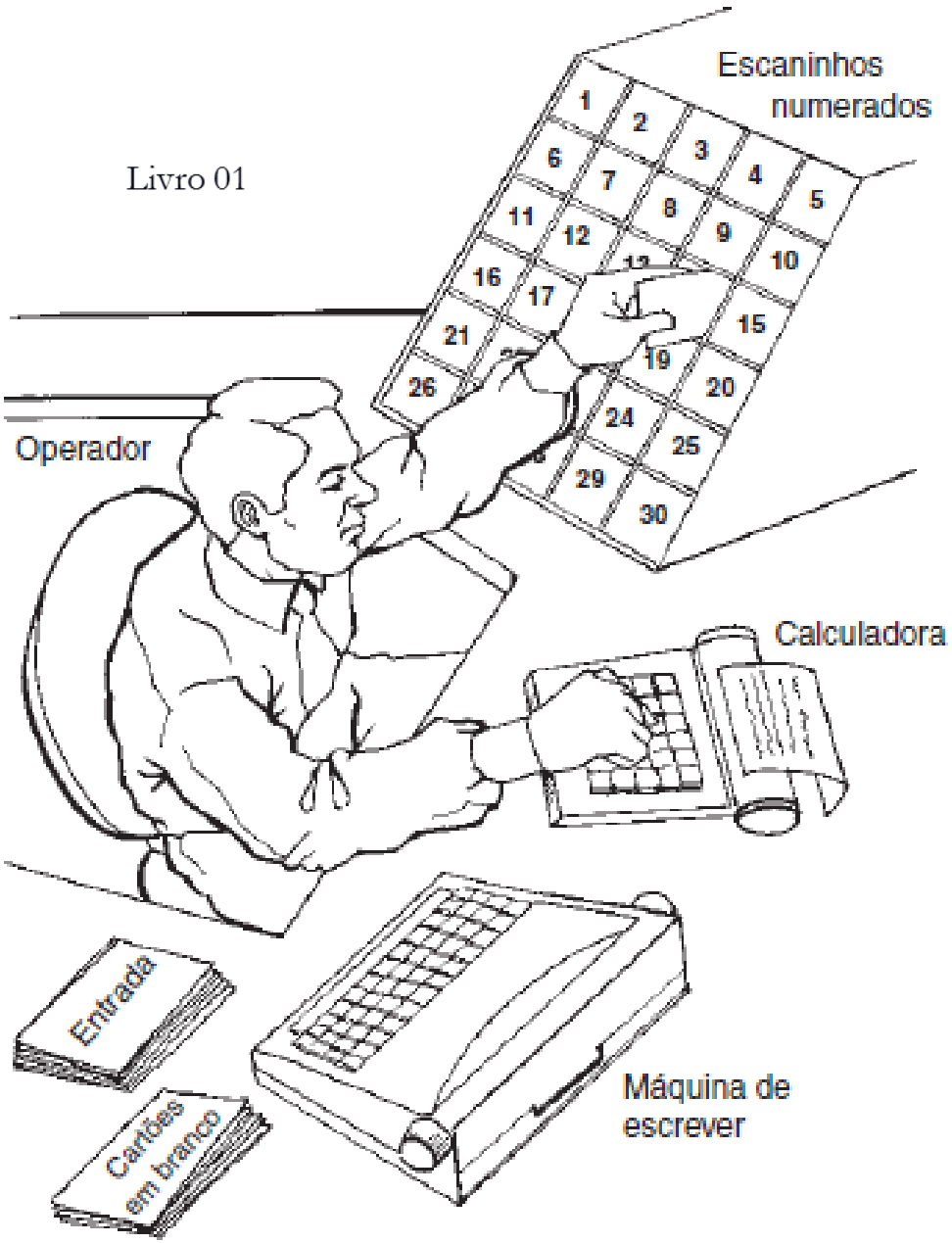
- Um número ou
- Uma instrução a ser executada

Nomenclatura:

Sendo $1 \leq i \leq 30$, então:

Esc i é o escaninho nº i

Funcionamento:



- O operador retira, lê e executa a instrução de **Esc 1**
- Em seguida, faz o mesmo com a de **Esc 2**, **Esc 3**, ... , etc.
- As instruções vão sendo executadas sequencialmente
- No entanto, há instruções que mandam quebrar a sequência

Exemplo: sejam as seguintes instruções de **Esc 1** a **Esc 12**

1	2	3	4	5
Esc 21 ← 0	**Esc 22 ← 0**	**Esc 23 ← 1**	**Se entrada vazia, Exec Esc 9**	**Ler número para Esc 24**
6	**7**	**8**	**9**	**10**
Esc 21 ← Esc 21 + Esc 24	**Esc 22 ← Esc 22 + Esc 23**	**Exec Esc 4**	**Se Esc 22 = 0, Exec Esc 12**	**Esc 25 ← Esc 21 / Esc 22**
11	**12**			
Imprimir Esc 25	**Finalizar**			

No início, o conteúdo de **Esc 21** a **Esc 25** é indefinido

1	2	3	4	5
Esc 21 ← 0	**Esc 22 ← 0**	**Esc 23 ← 1**	**Se entrada vazia, Exec Esc 9**	**Ler número para Esc 24**

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	**Esc 22 ← Esc 22 + Esc 23**	**Exec Esc 4**	**Se Esc 22 = 0, Exec Esc 12**	**Esc 25 ← Esc 21 / Esc 22**

11	12
Imprimir Esc 25	**Finalizar**

21	22	23	24	25
?	?	?	?	?

Sejam também alguns **cartões de entrada**:

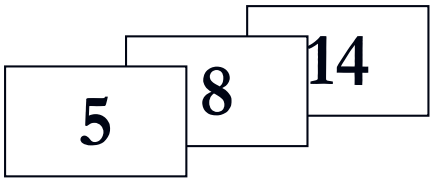
1	2	3	4	5
Esc 21 ← 0	**Esc 22 ← 0**	**Esc 23 ← 1**	**Se entrada vazia, Exec Esc 9**	**Ler número para Esc 24**

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	**Esc 22 ← Esc 22 + Esc 23**	**Exec Esc 4**	**Se Esc 22 = 0, Exec Esc 12**	**Esc 25 ← Esc 21 / Esc 22**

11	12
Imprimir Esc 25	**Finalizar**

21	22	23	24	25
?	?	?	?	?

Entrada

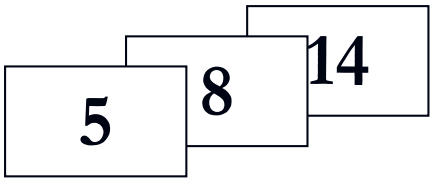


Seja agora a **execução das instruções**:

1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24
6	**7**	**8**	**9**	**10**
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22
11	**12**			
Imprimir Esc 25	Finalizar			

21	22	23	24	25
?	?	?	?	?

Entrada



Operador

1	2	3	4	5
	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

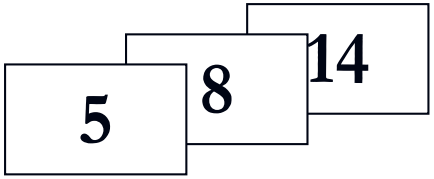
11	12
Imprimir Esc 25	Finalizar

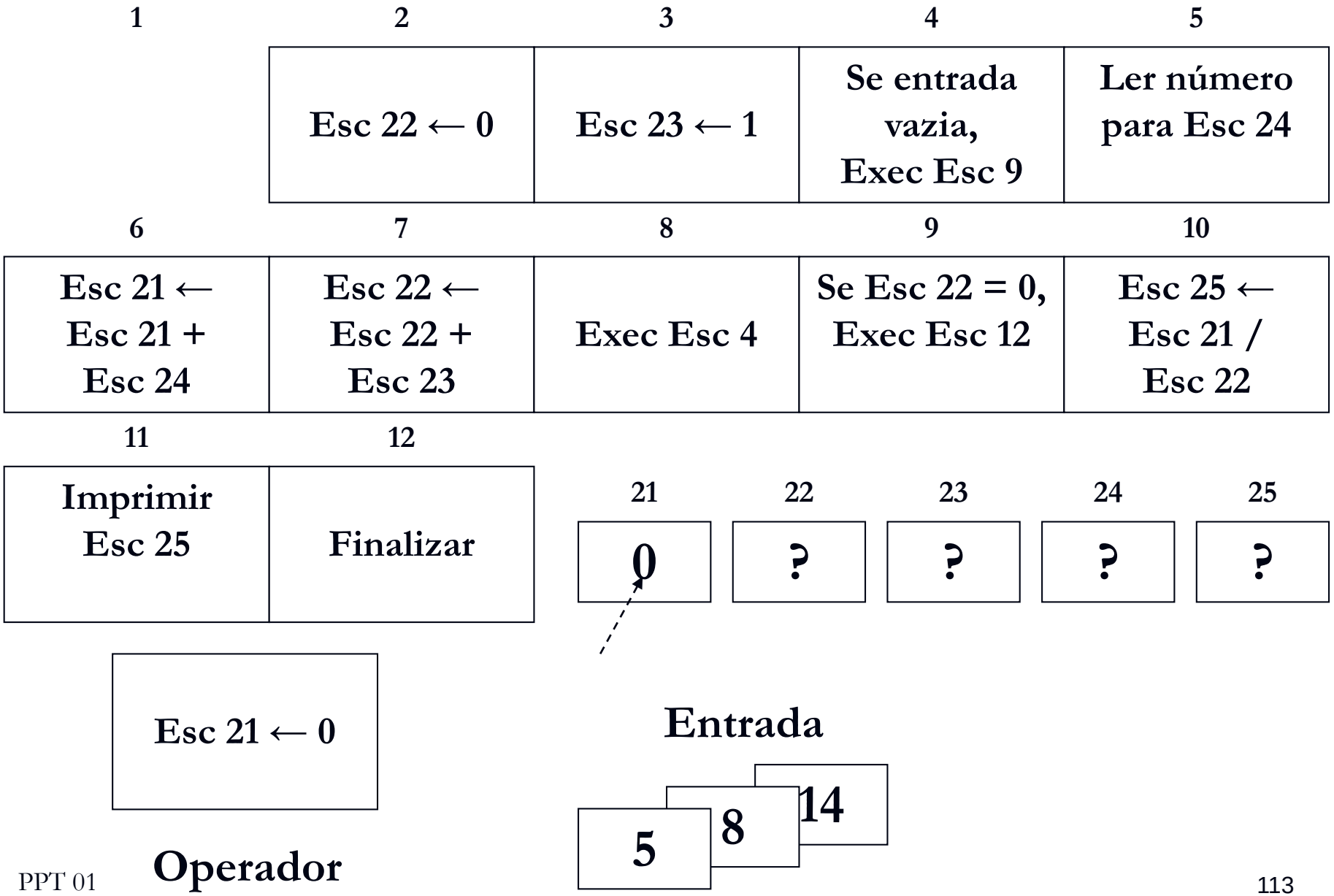
21	22	23	24	25
?	?	?	?	?

Esc 21 ← 0

Operador

Entrada

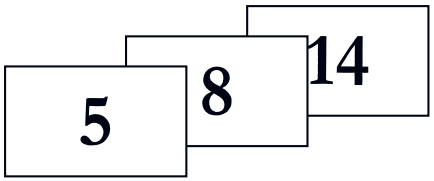



1
2
3
4
5
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 21 ← Esc 21 + Esc 24
Esc 22 ← Esc 22 + Esc 23
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
0
?
?
?
?
Esc 21 ← 0
Operador
Entrada
5
8
14

1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24
6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22
11	12			
Imprimir Esc 25	Finalizar			

21	22	23	24	25
0	?	?	?	?

Entrada



Operador

1	2	3	4	5
Esc 21 ← 0		Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

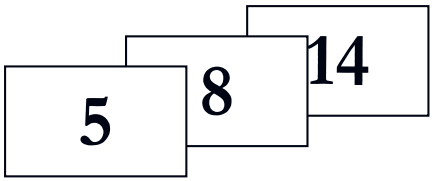
11	12
Imprimir Esc 25	Finalizar

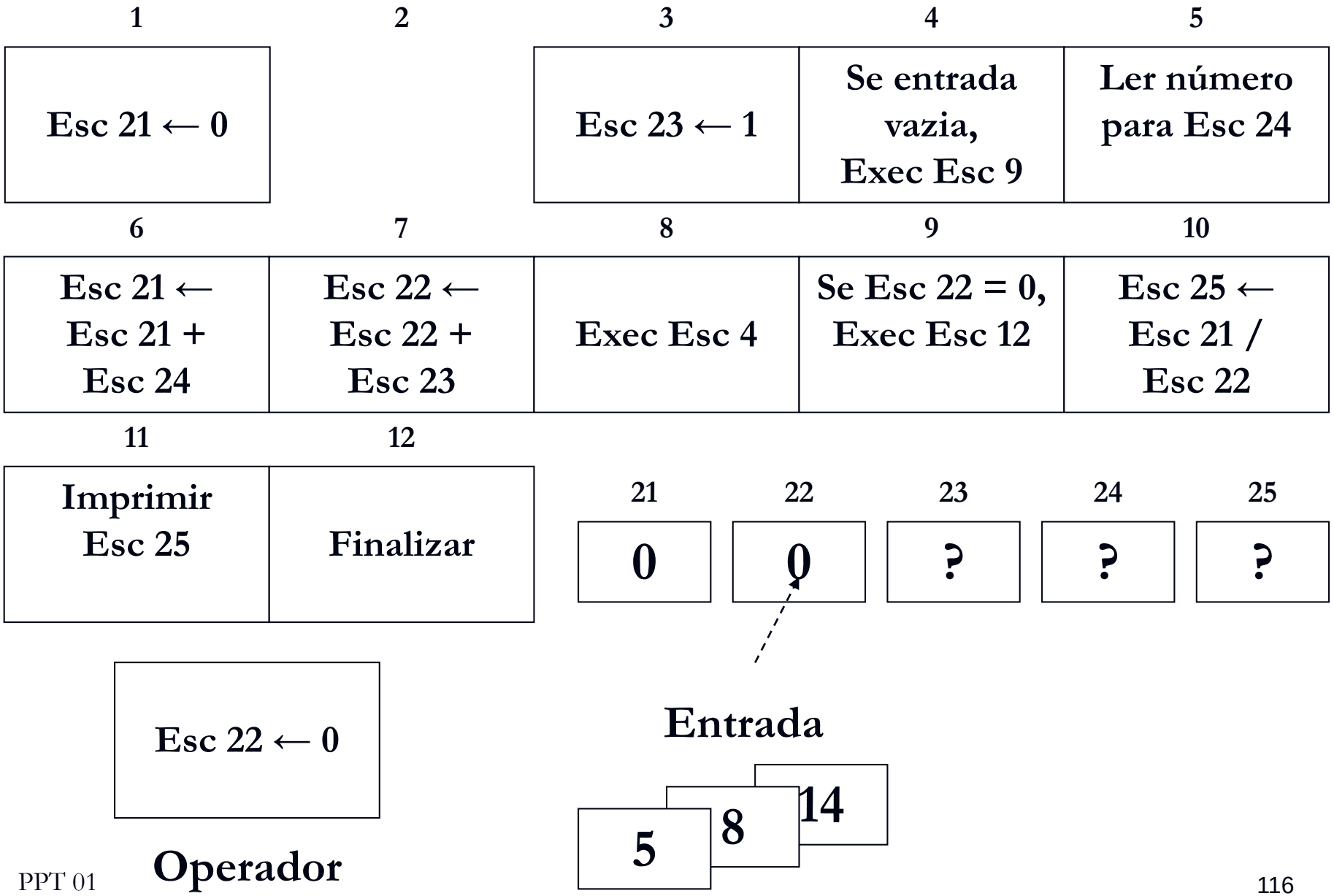
21	22	23	24	25
0	?	?	?	?

Esc 22 ← 0

Operador

Entrada




1
2
3
4
5
Esc 21 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 21 ← Esc 21 + Esc 24
Esc 22 ← Esc 22 + Esc 23
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
0
0
?
?
?
Entrada
5
8
14
Esc 22 ← 0
Operador

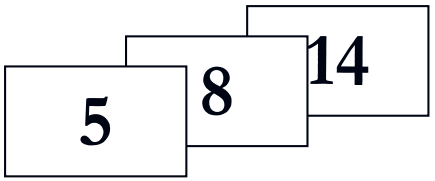
1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

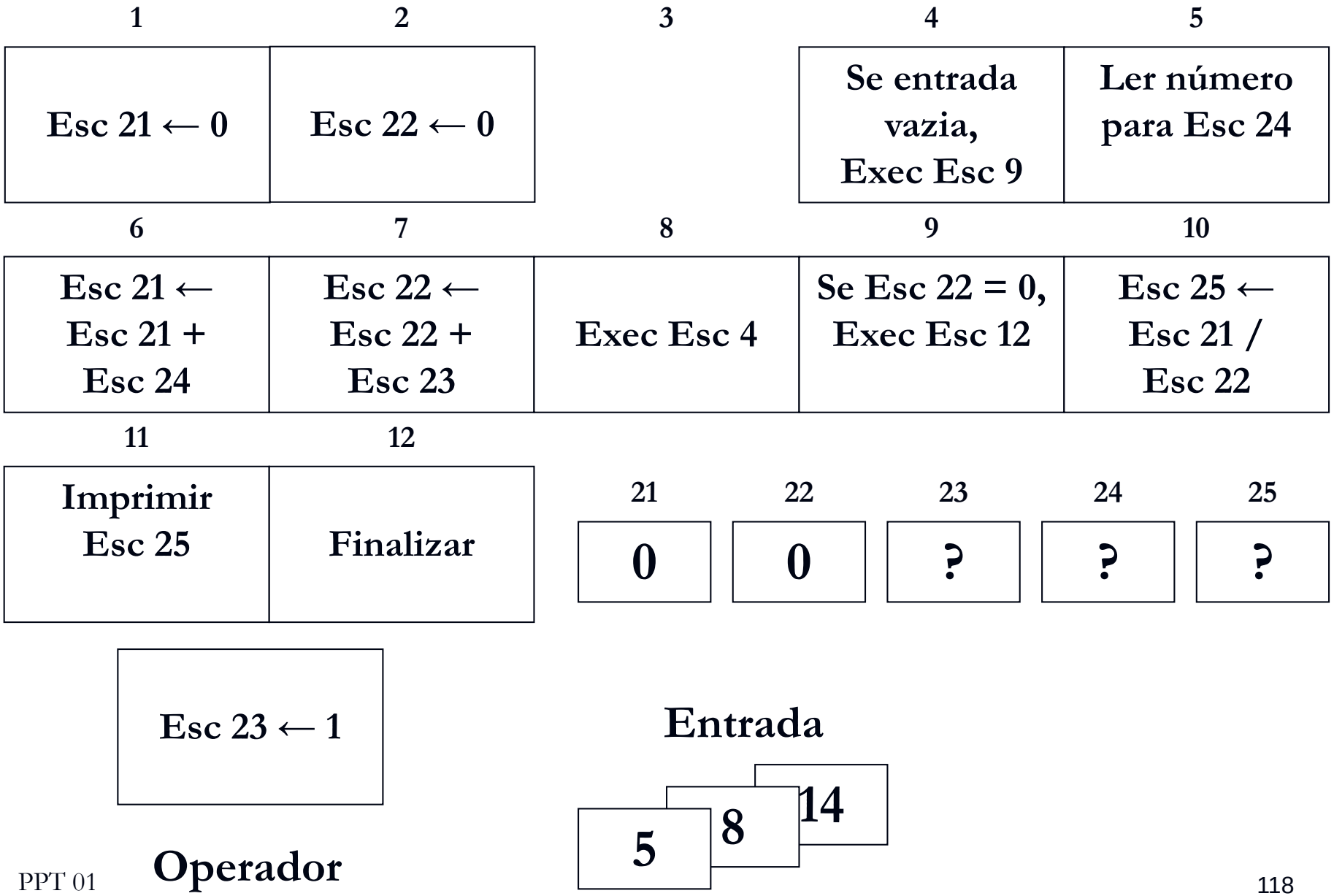
11	12
Imprimir Esc 25	Finalizar

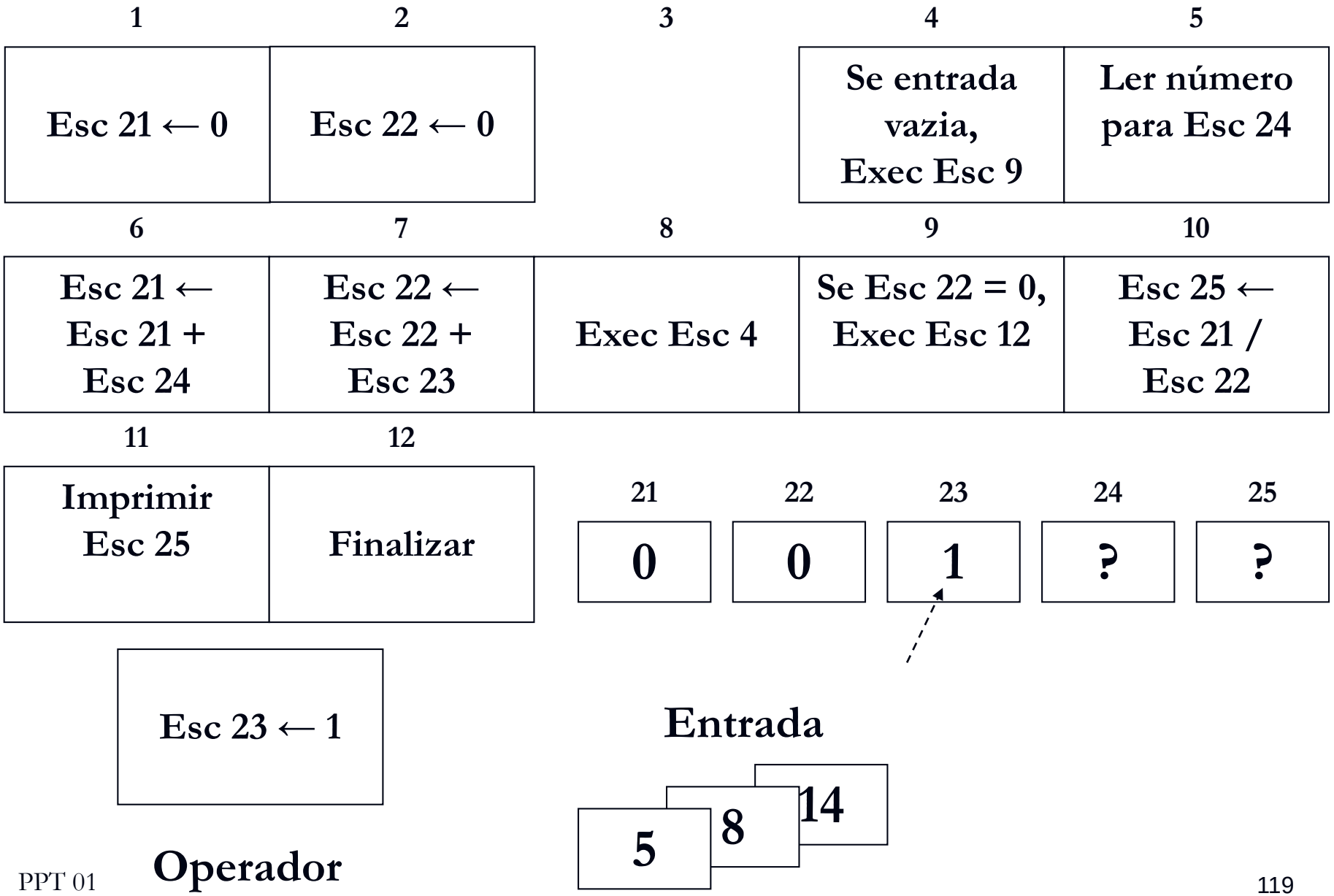
21	22	23	24	25
0	0	?	?	?

Entrada



Operador


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 21 ← Esc 21 + Esc 24
Esc 22 ← Esc 22 + Esc 23
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
0
0
?
?
?
Esc 23 ← 1
Operador
Entrada
5
8
14


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 21 ← Esc 21 + Esc 24
Esc 22 ← Esc 22 + Esc 23
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
0
0
1
?
?
Esc 23 ← 1
Operador
Entrada
5
8
14

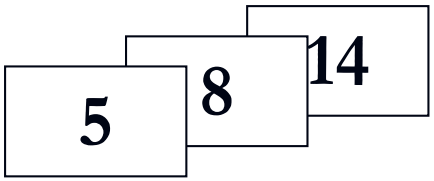
1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

11	12
Imprimir Esc 25	Finalizar

21	22	23	24	25
0	0	1	?	?

Entrada



Operador

1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1		Ler número para Esc 24

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

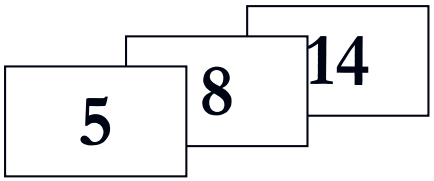
11	12
Imprimir Esc 25	Finalizar

21	22	23	24	25
0	0	1	?	?

Se entrada vazia,
Exec Esc 9

Operador

Entrada



Há cartões na entrada: manter a sequência

1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1		Ler número para Esc 24

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

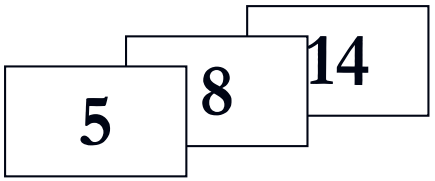
11	12
Imprimir Esc 25	Finalizar

21	22	23	24	25
0	0	1	?	?

Se entrada vazia, Exec Esc 9

Operador

Entrada



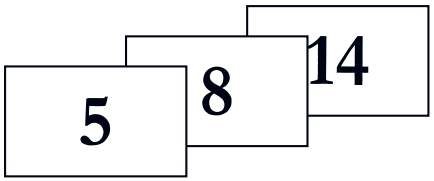
1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

11	12
Imprimir Esc 25	Finalizar

21	22	23	24	25
0	0	1	?	?

Entrada



Operador

1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

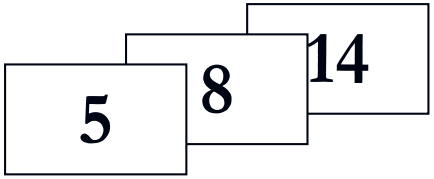
11	12
Imprimir Esc 25	Finalizar

21	22	23	24	25
0	0	1	?	?

Ler número para Esc 24

Operador

Entrada



1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

11	12
Imprimir Esc 25	Finalizar

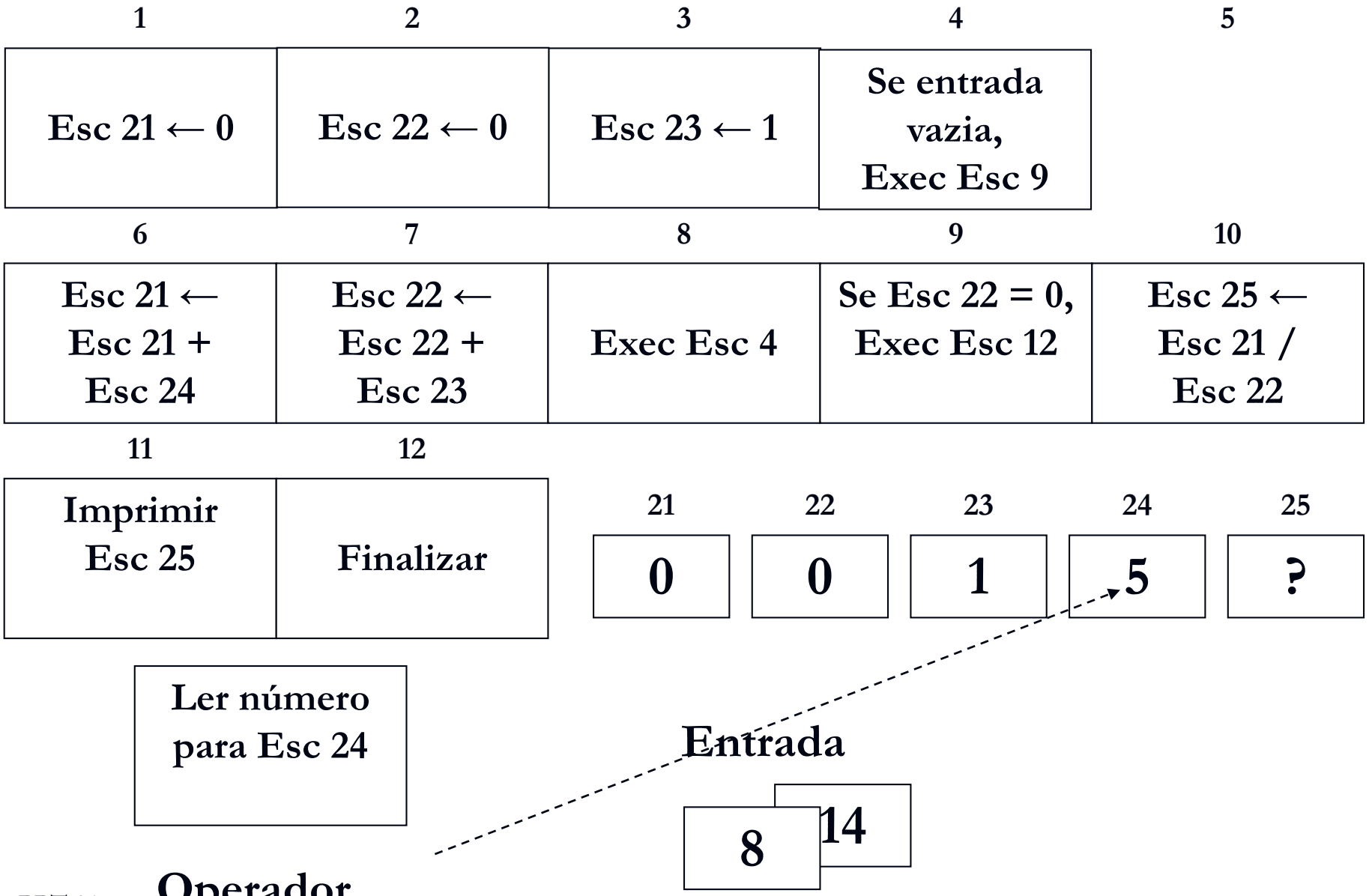
21	22	23	24	25
0	0	1	?	?

Ler número para Esc 24

Entrada

8 14

Operador 5


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
6
7
8
9
10
Esc 21 ← Esc 21 + Esc 24
Esc 22 ← Esc 22 + Esc 23
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
0
0
1
5
?
Ler número para Esc 24
Entrada
8
14
Operador

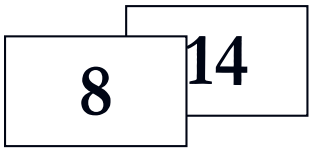
1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

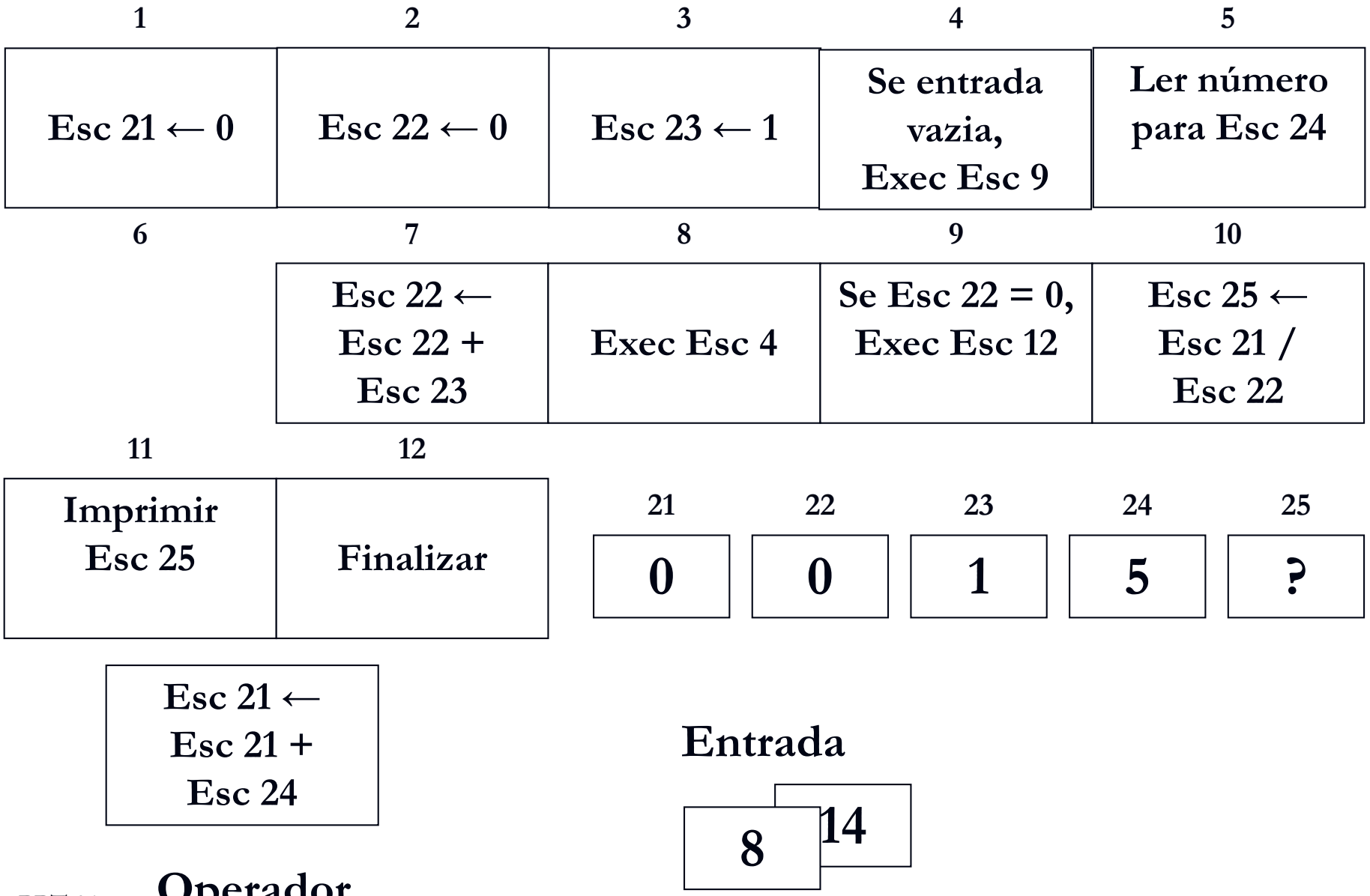
11	12
Imprimir Esc 25	Finalizar

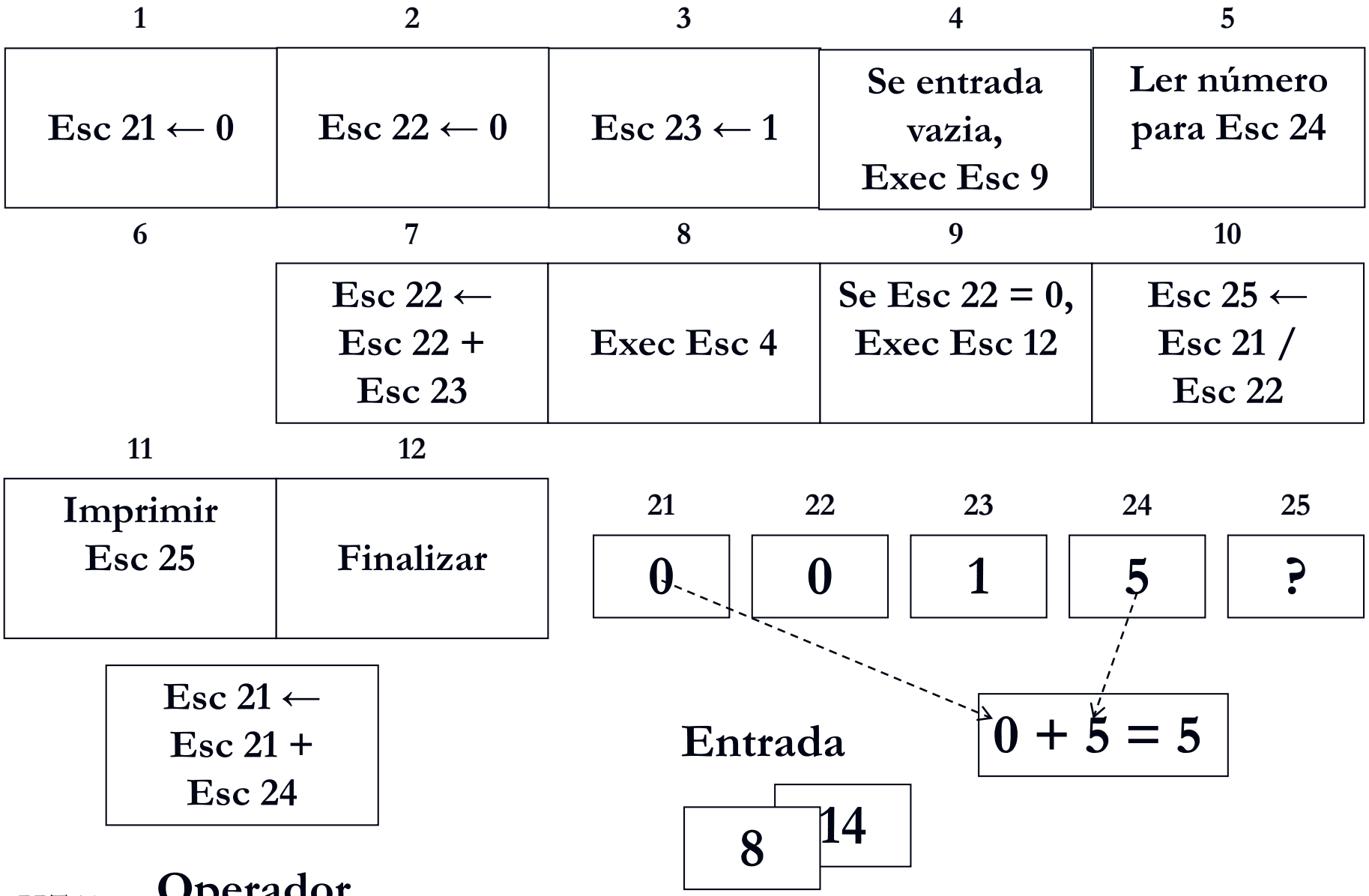
21	22	23	24	25
0	0	1	5	?

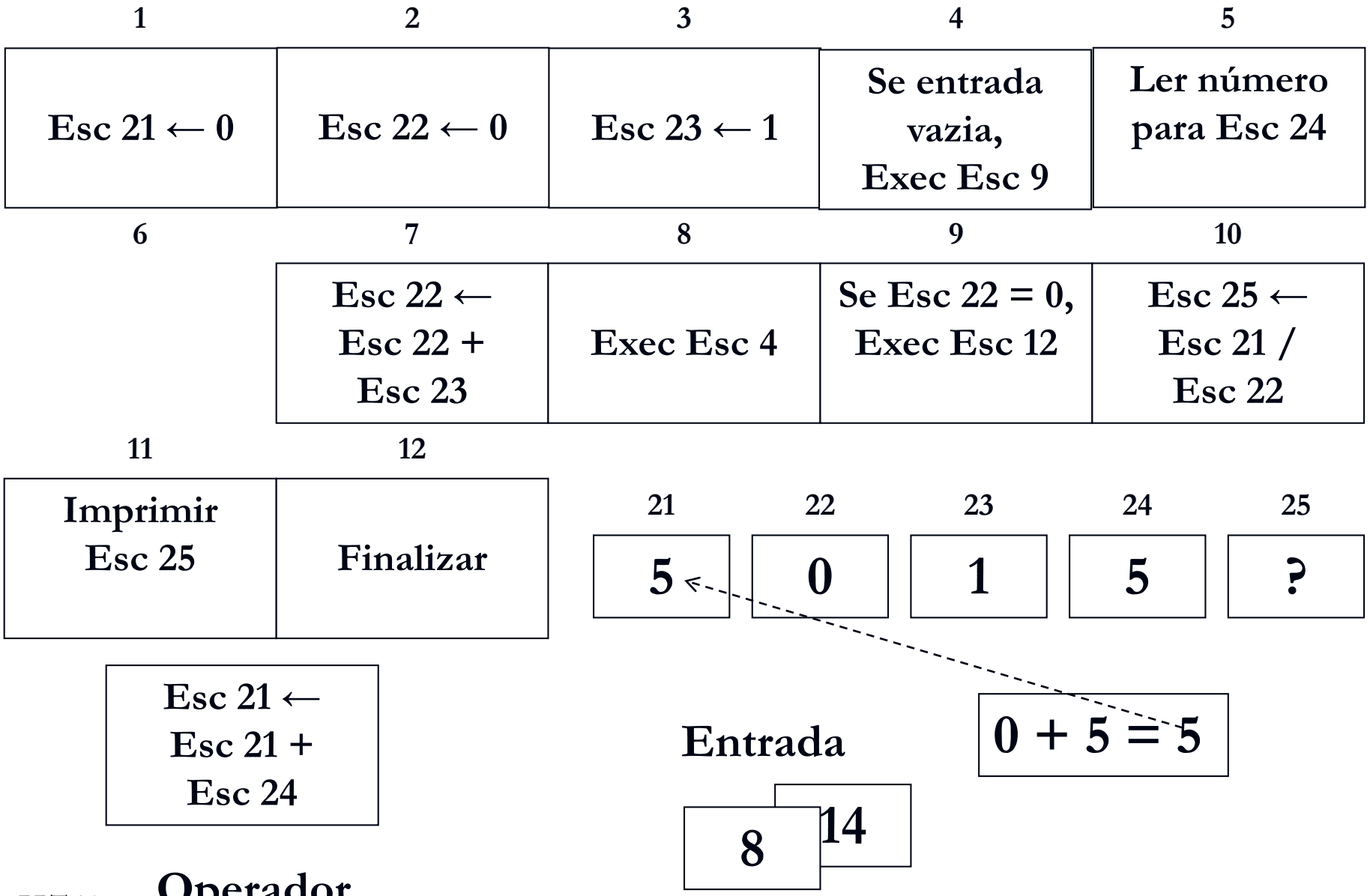
Entrada



Operador


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 22 ← Esc 22 + Esc 23
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
0
0
1
5
?
Esc 21 ← Esc 21 + Esc 24
Operador
Entrada
8
14


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 22 ← Esc 22 + Esc 23
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
0
0
1
5
?
0 + 5 = 5
Esc 21 ← Esc 21 + Esc 24
Operador
Entrada
8
14


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 22 ← Esc 22 + Esc 23
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
5
0
1
5
?
0 + 5 = 5
Esc 21 ← Esc 21 + Esc 24
Entrada
8
14
Operador

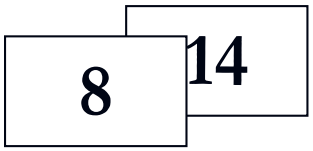
1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

11	12
Imprimir Esc 25	Finalizar

21	22	23	24	25
5	0	1	5	?

Entrada



Operador

1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24		Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

11	12
Imprimir Esc 25	Finalizar

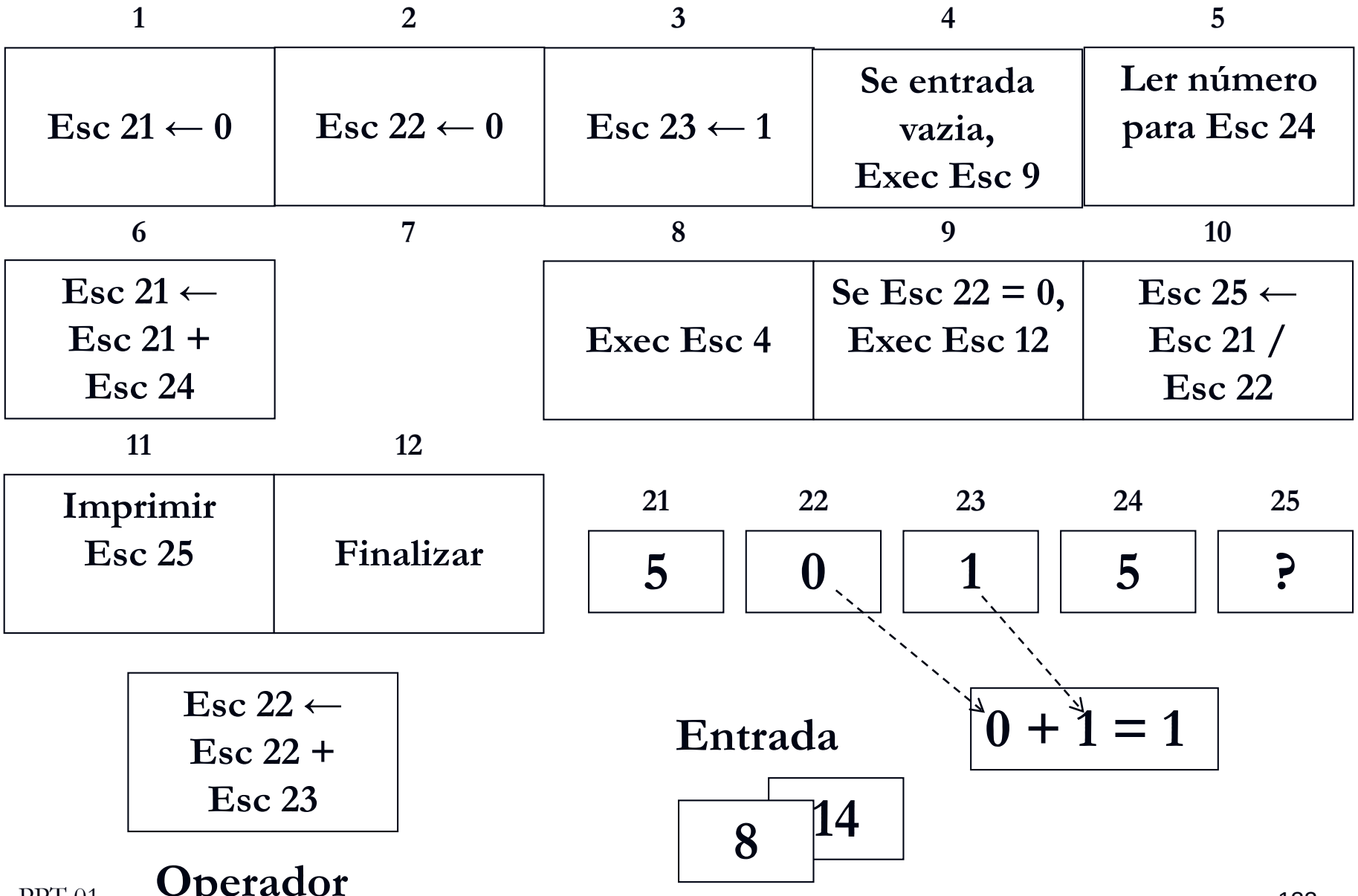
21	22	23	24	25
5	0	1	5	?

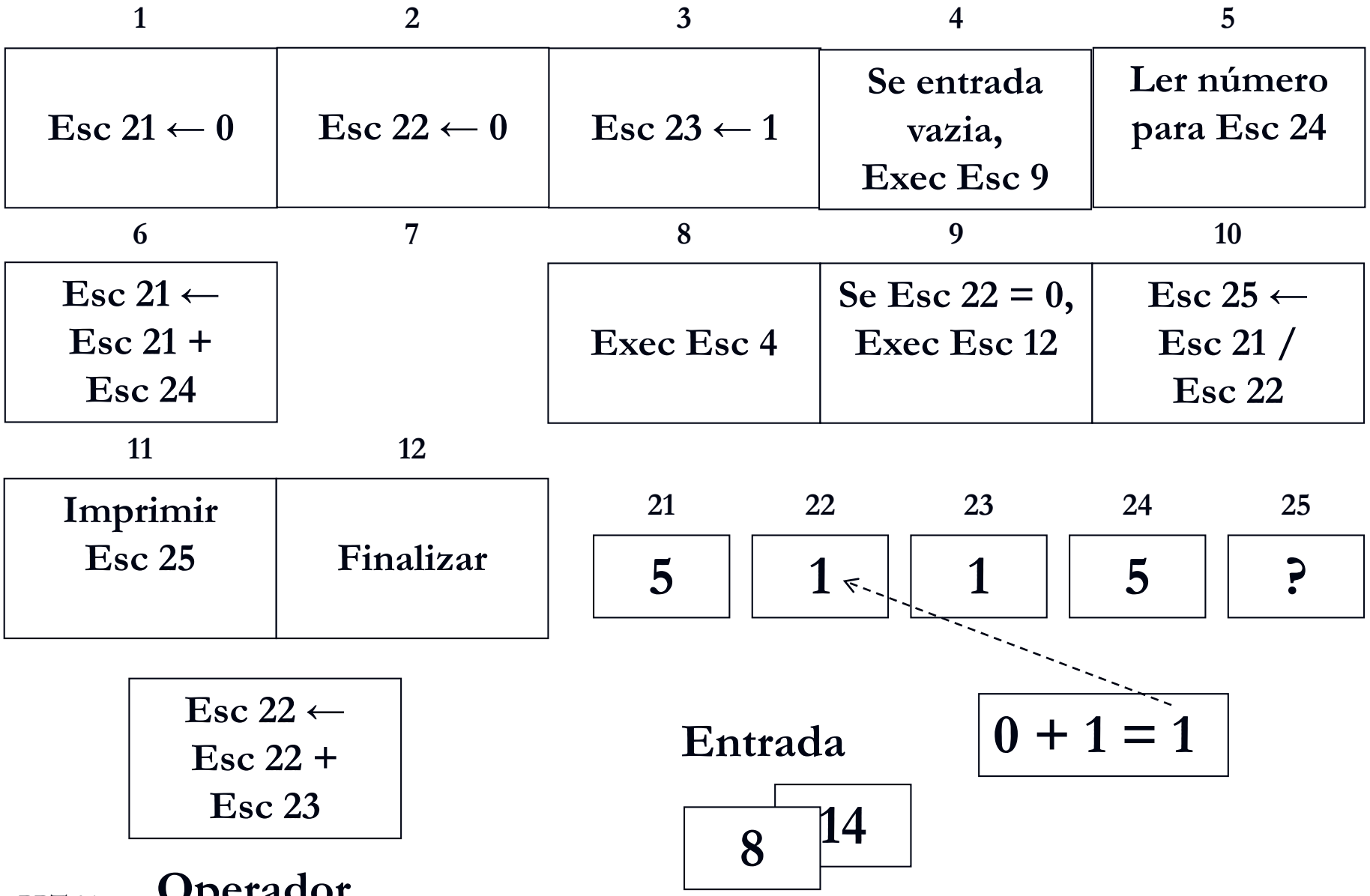
Esc 22 ← Esc 22 + Esc 23

Operador

Entrada

8 14


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 21 ← Esc 21 + Esc 24
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
5
0
1
5
?
0 + 1 = 1
Esc 22 ← Esc 22 + Esc 23
Operador
Entrada
8
14


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 21 ← Esc 21 + Esc 24
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
5
1
1
5
?
0 + 1 = 1
Esc 22 ← Esc 22 + Esc 23
Operador
Entrada
8
14

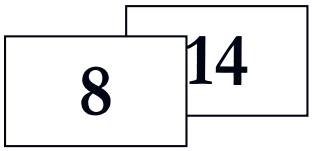
1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

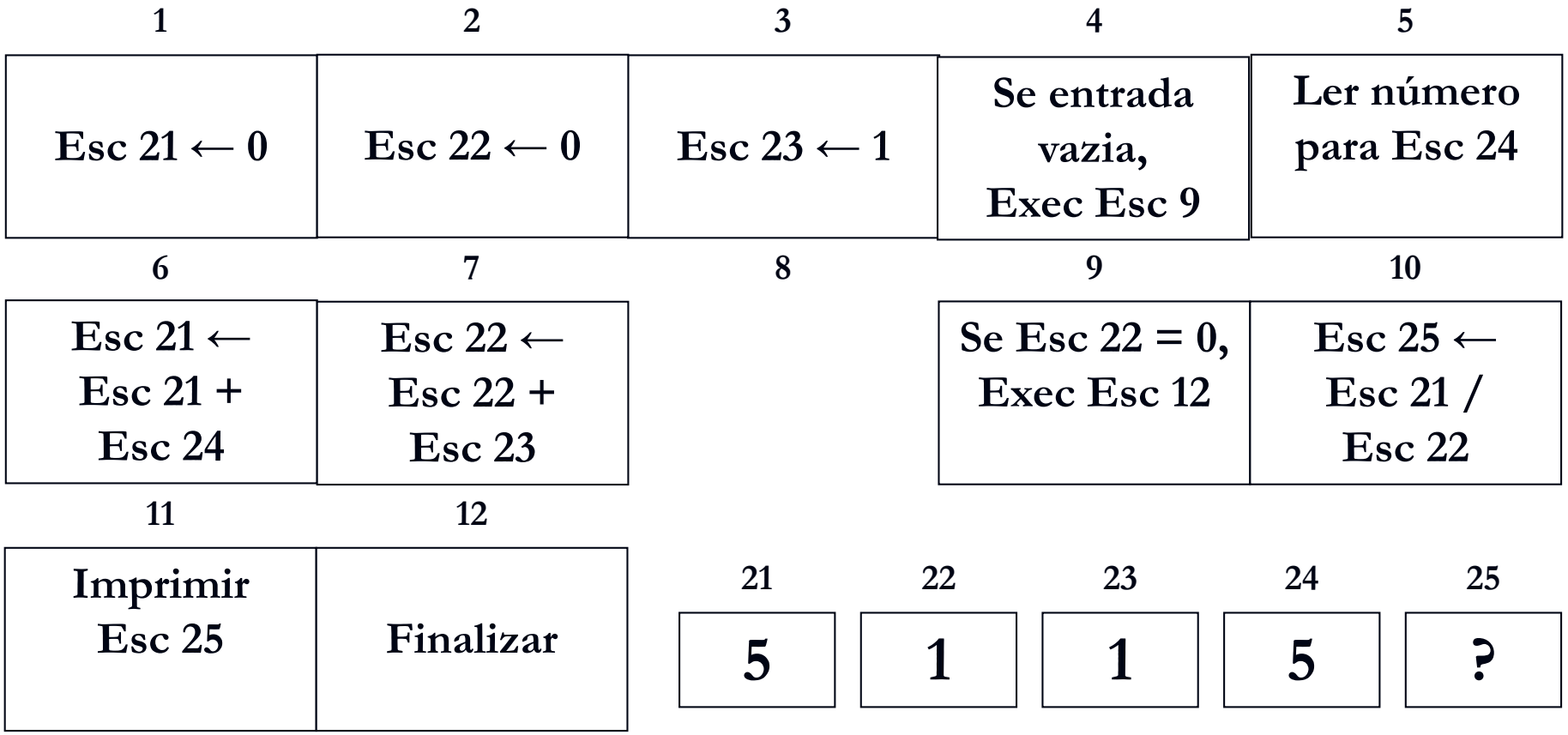
11	12
Imprimir Esc 25	Finalizar

21	22	23	24	25
5	1	1	5	?

Entrada



Operador


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 21 ← Esc 21 + Esc 24
Esc 22 ← Esc 22 + Esc 23
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
5
1
1
5
?
Exec Esc 4
Operador
Entrada
8
14

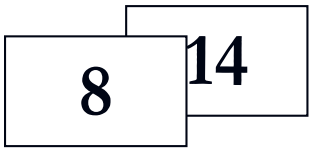
1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

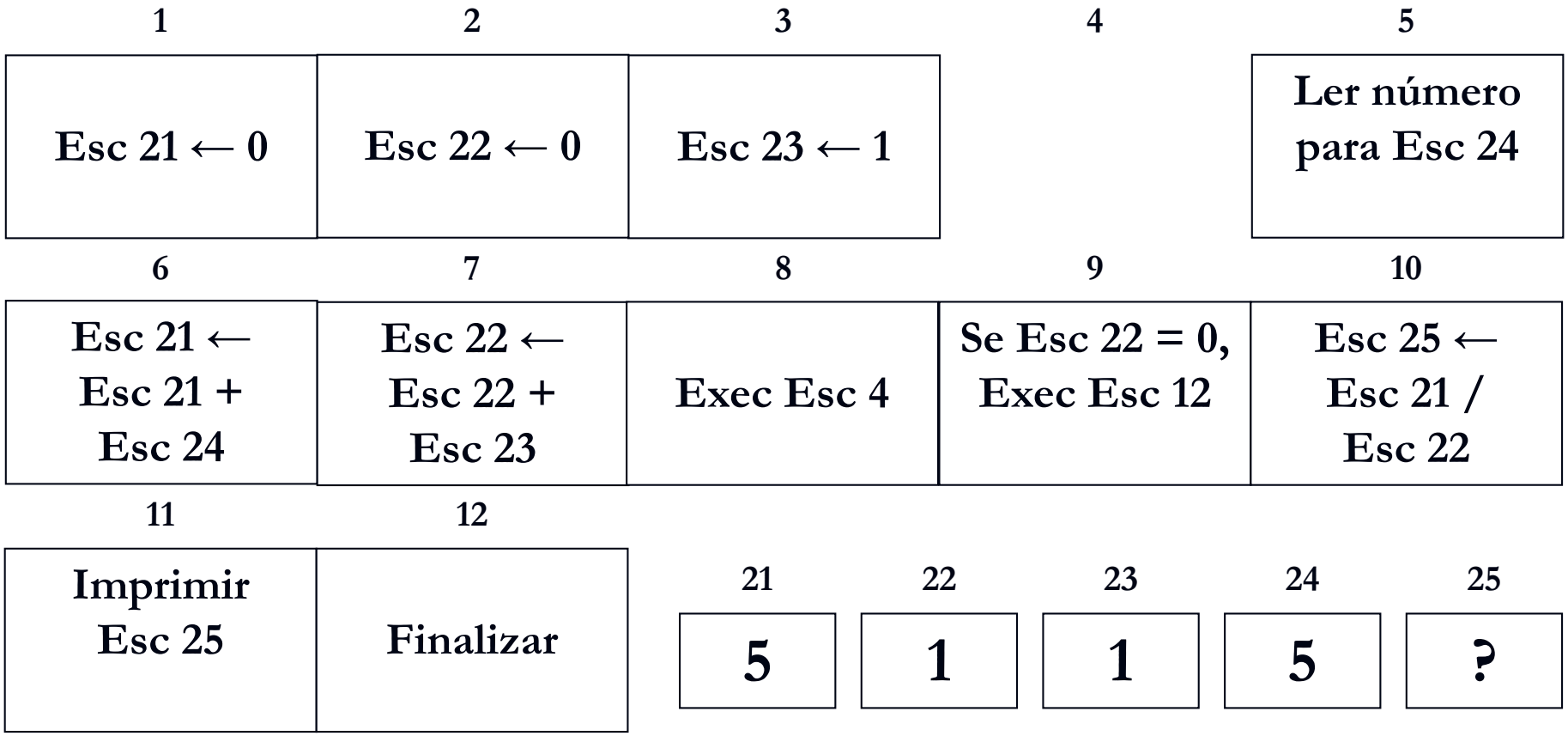
11	12
Imprimir Esc 25	Finalizar

21	22	23	24	25
5	1	1	5	?

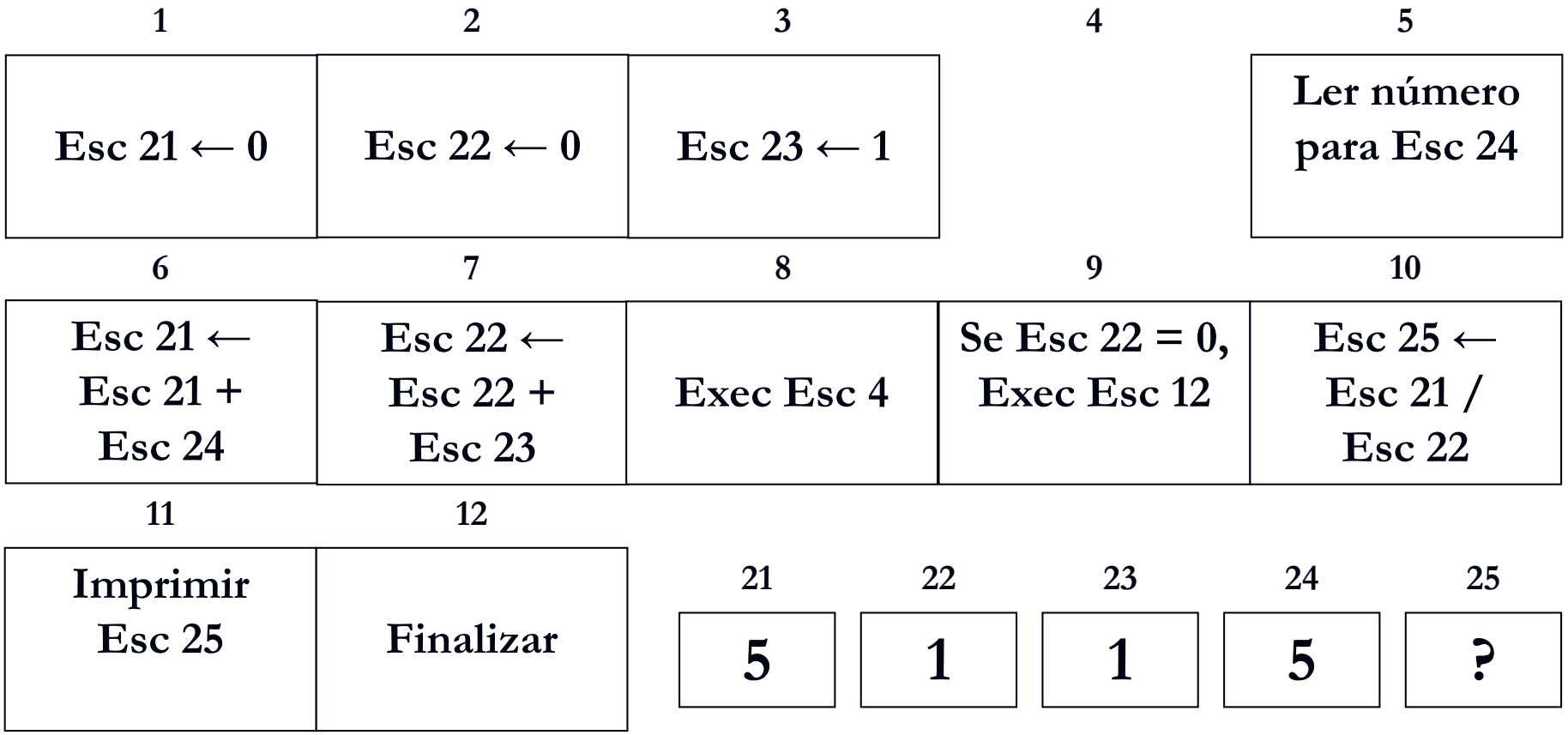
Entrada



Operador


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Ler número para Esc 24
6
7
8
9
10
Esc 21 ← Esc 21 + Esc 24
Esc 22 ← Esc 22 + Esc 23
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
5
1
1
5
?
Se entrada vazia, Exec Esc 9
Operador
Entrada
8
14

Há cartões na entrada: manter a sequência



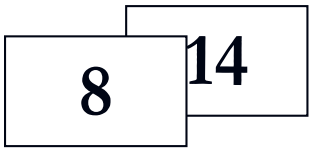
1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

11	12
Imprimir Esc 25	Finalizar

21	22	23	24	25
5	1	1	5	?

Entrada



Operador

1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

11	12
Imprimir Esc 25	Finalizar

21	22	23	24	25
5	1	1	5	?

Ler número para Esc 24

Entrada

8 14

Operador

1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

11	12
Imprimir Esc 25	Finalizar

21	22	23	24	25
5	1	1	5	?

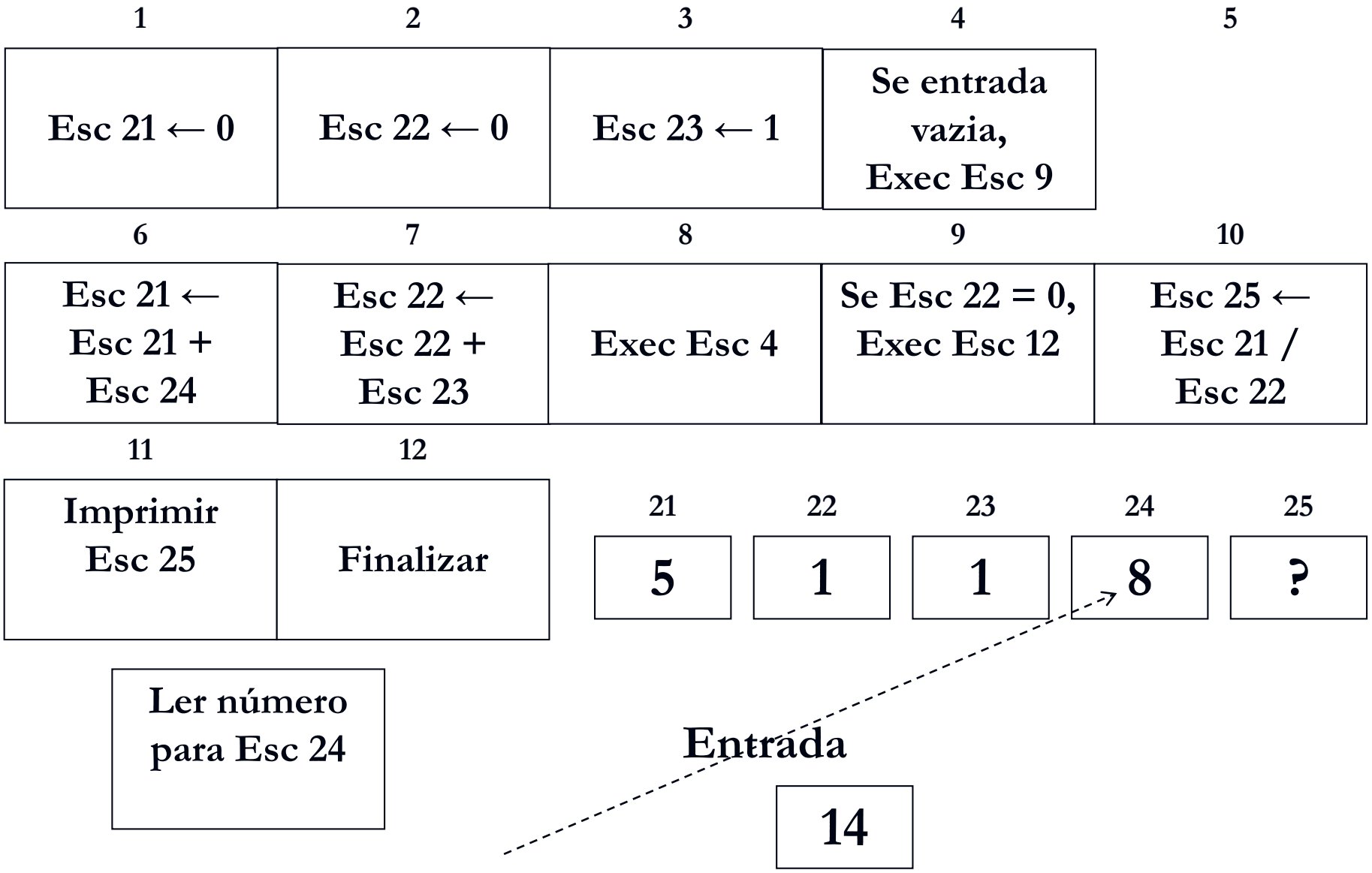
Ler número para Esc 24

Entrada

14

Operador

8


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
6
7
8
9
10
Esc 21 ← Esc 21 + Esc 24
Esc 22 ← Esc 22 + Esc 23
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
5
1
1
8
?
Ler número para Esc 24
Entrada
14
Operador

1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

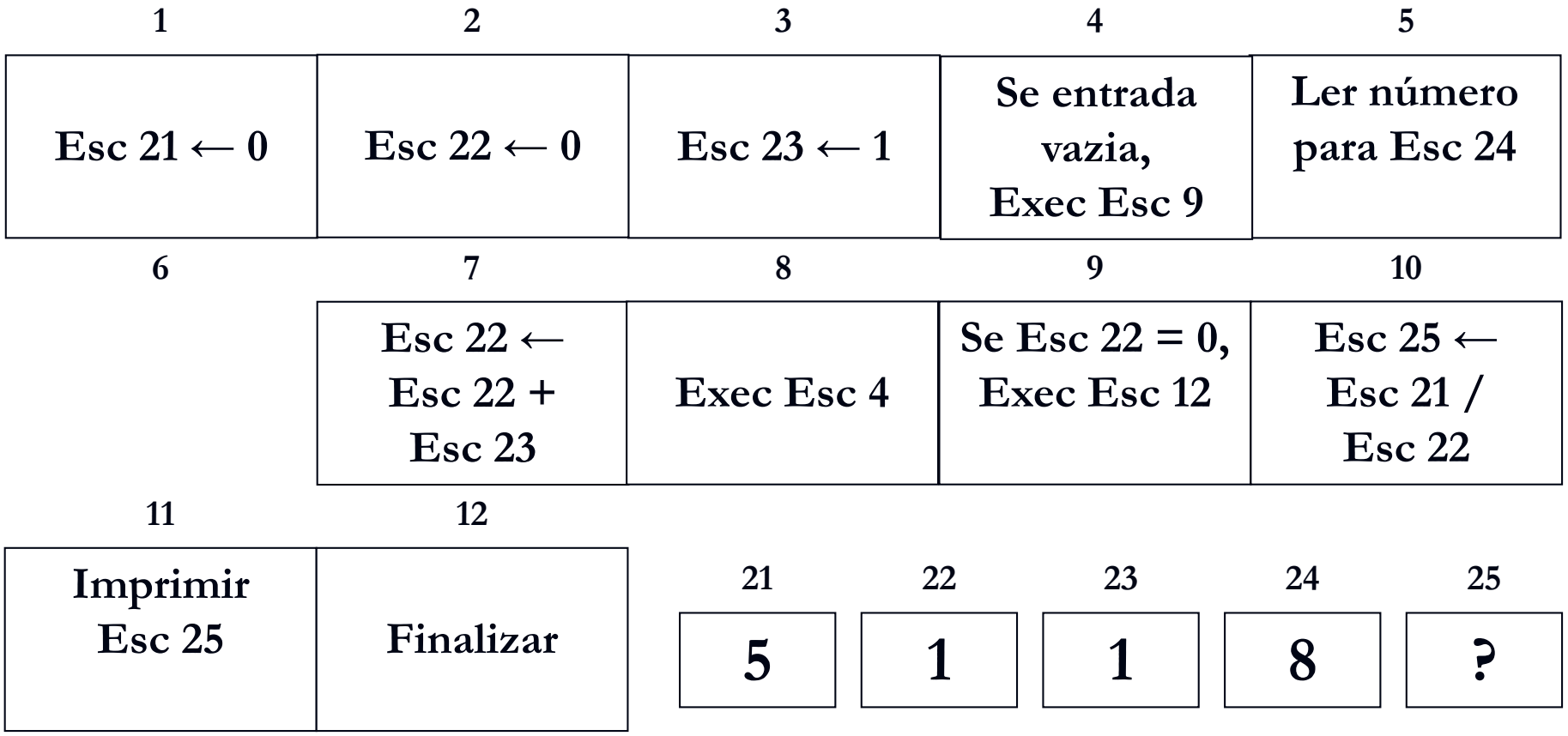
11	12
Imprimir Esc 25	Finalizar

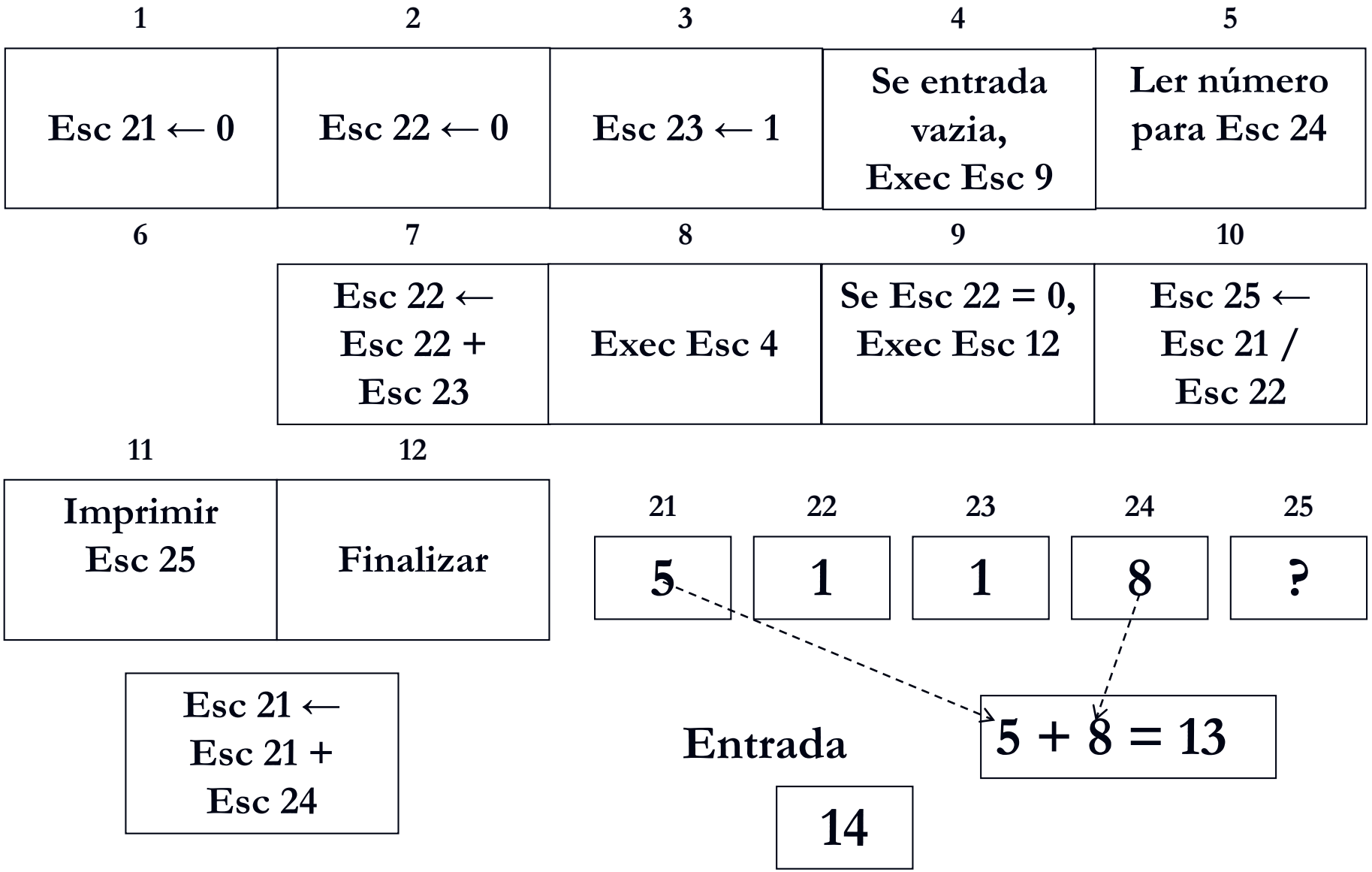
21	22	23	24	25
5	1	1	8	?

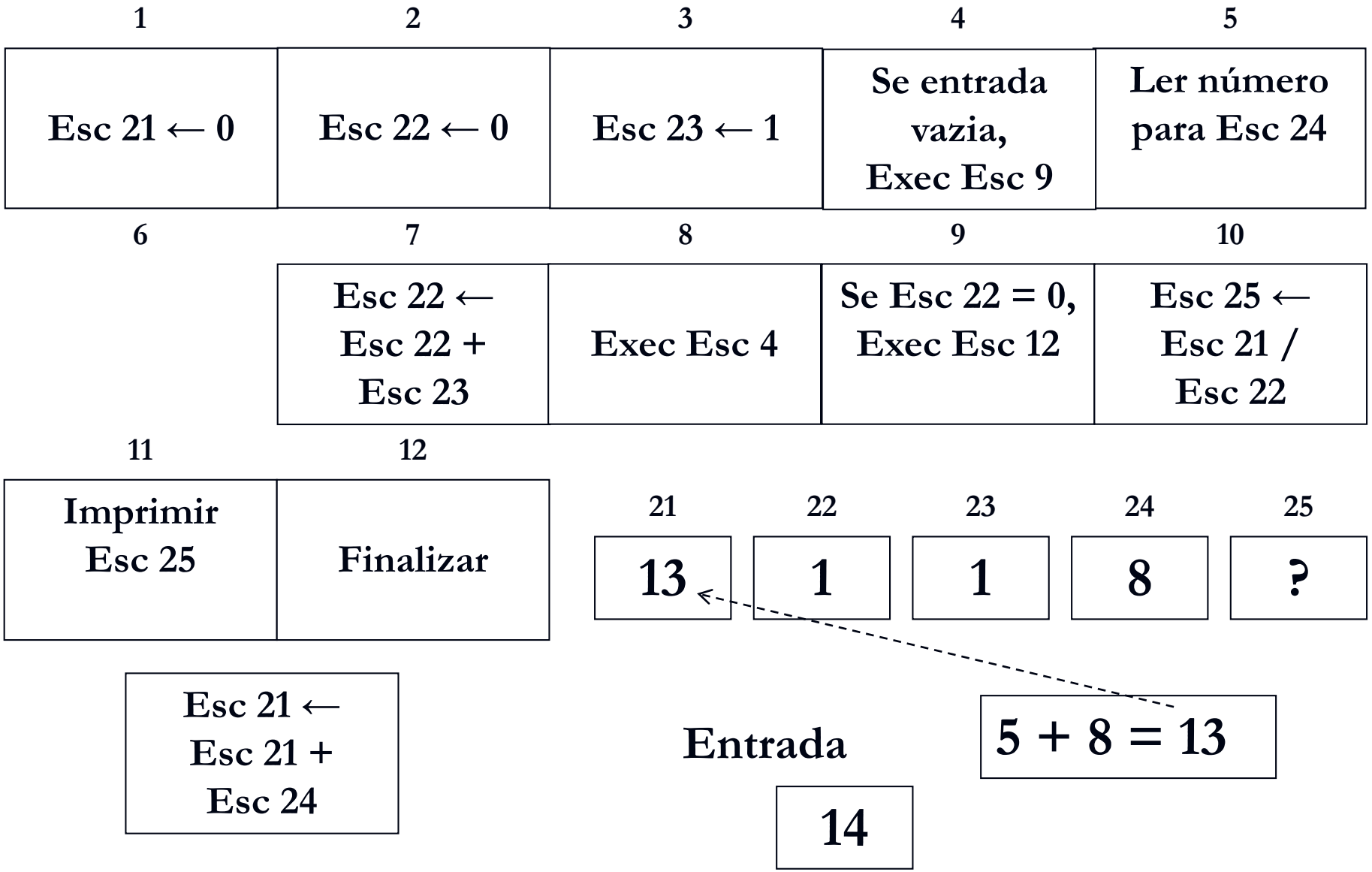
Entrada

14

Operador


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 22 ← Esc 22 + Esc 23
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
5
1
1
8
?
Esc 21 ← Esc 21 + Esc 24
Operador
Entrada
14


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 22 ← Esc 22 + Esc 23
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
5
1
1
8
?
5 + 8 = 13
Entrada
14
Esc 21 ← Esc 21 + Esc 24
Operador


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 22 ← Esc 22 + Esc 23
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
13
1
1
8
?
5 + 8 = 13
Esc 21 ← Esc 21 + Esc 24
Entrada
14
Operador

1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

11	12
Imprimir Esc 25	Finalizar

21	22	23	24	25
13	1	1	8	?

Entrada

14

Operador

1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24		Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

11	12
Imprimir Esc 25	Finalizar

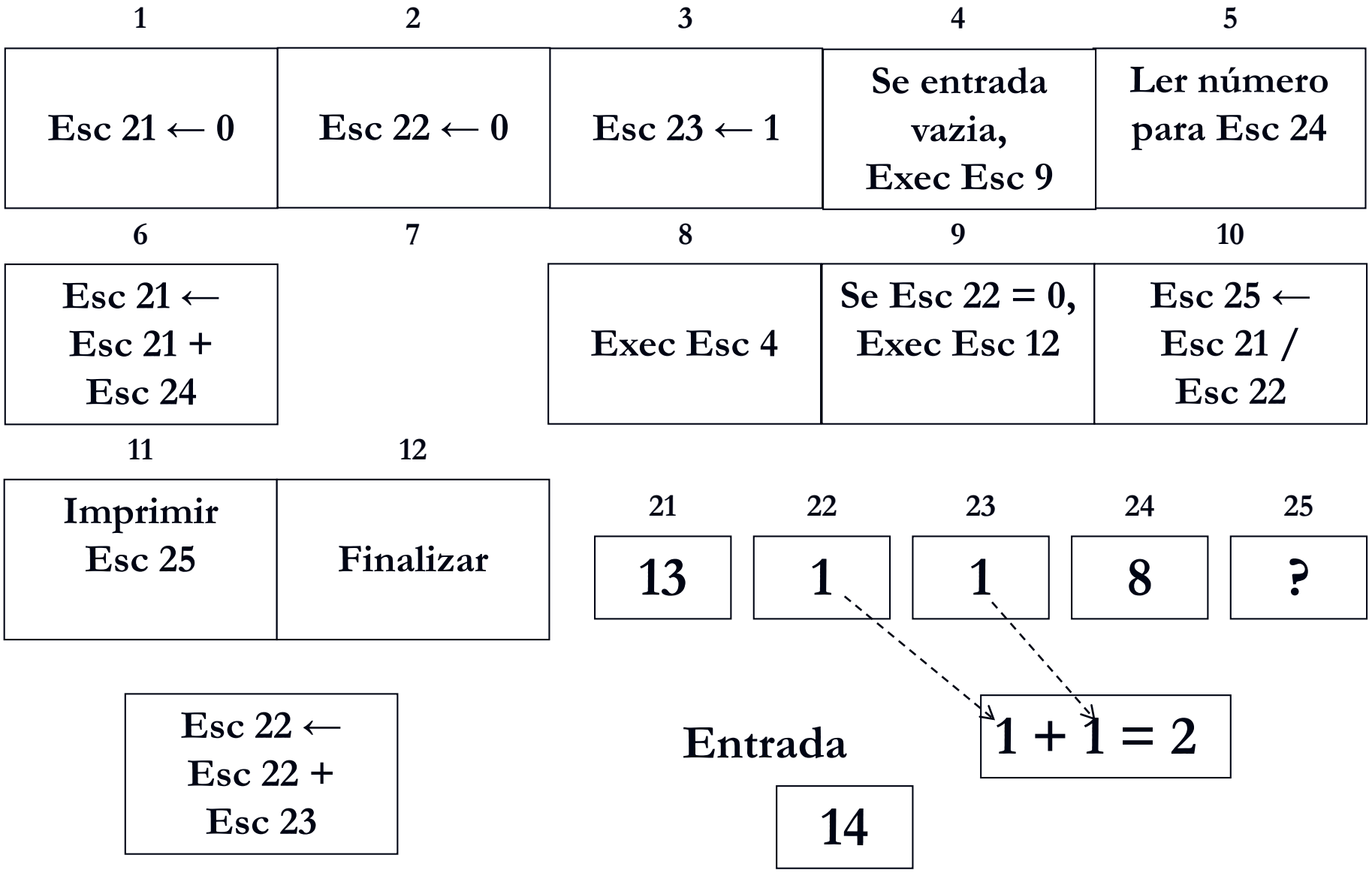
21	22	23	24	25
13	1	1	8	?

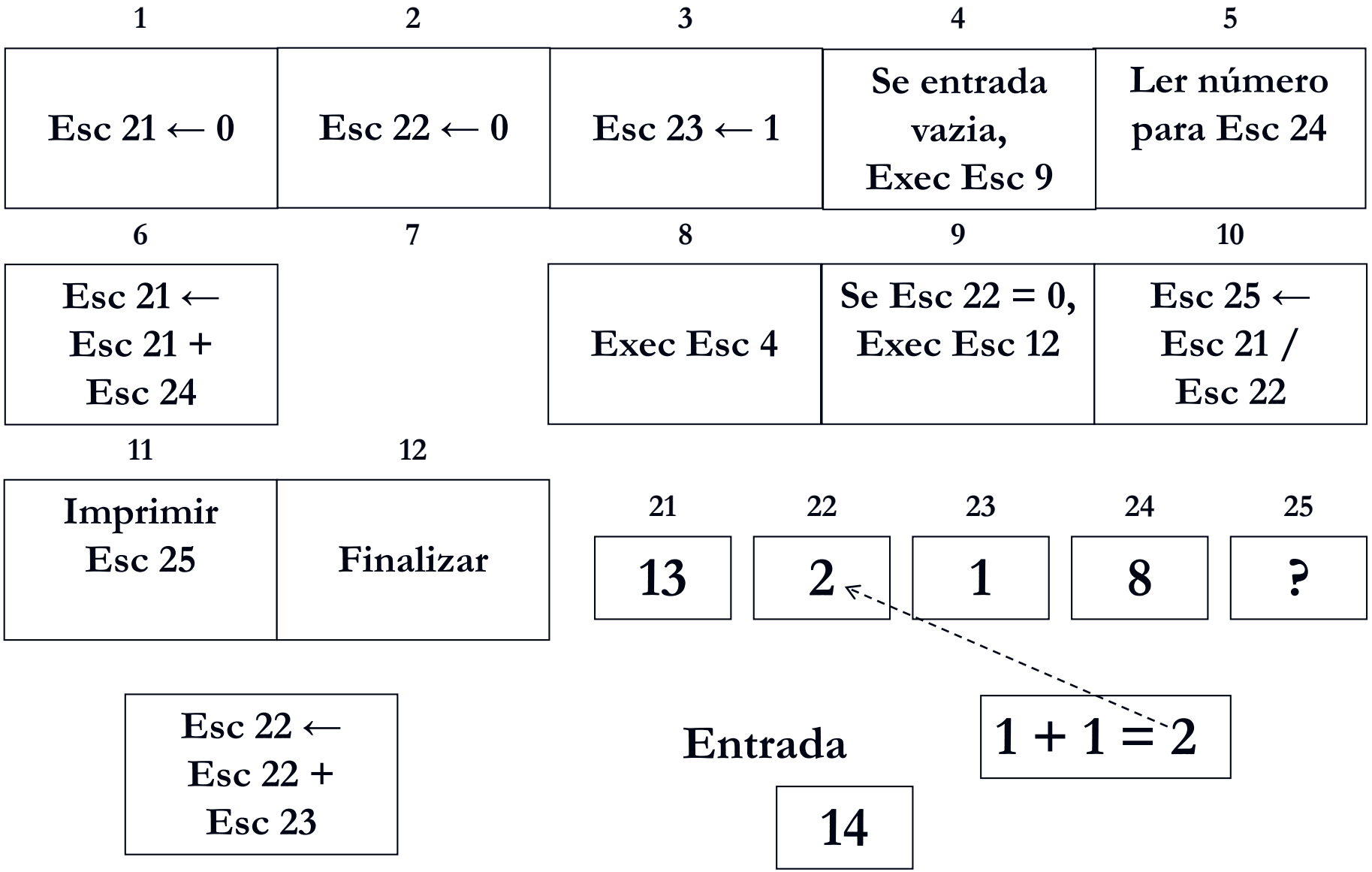
Esc 22 ← Esc 22 + Esc 23

Operador

Entrada

14


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 21 ← Esc 21 + Esc 24
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
13
1
1
8
?
1 + 1 = 2
Entrada
14
Esc 22 ← Esc 22 + Esc 23
Operador


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 21 ← Esc 21 + Esc 24
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
13
2
1
8
?
1 + 1 = 2
Esc 22 ← Esc 22 + Esc 23
Operador
Entrada
14

1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

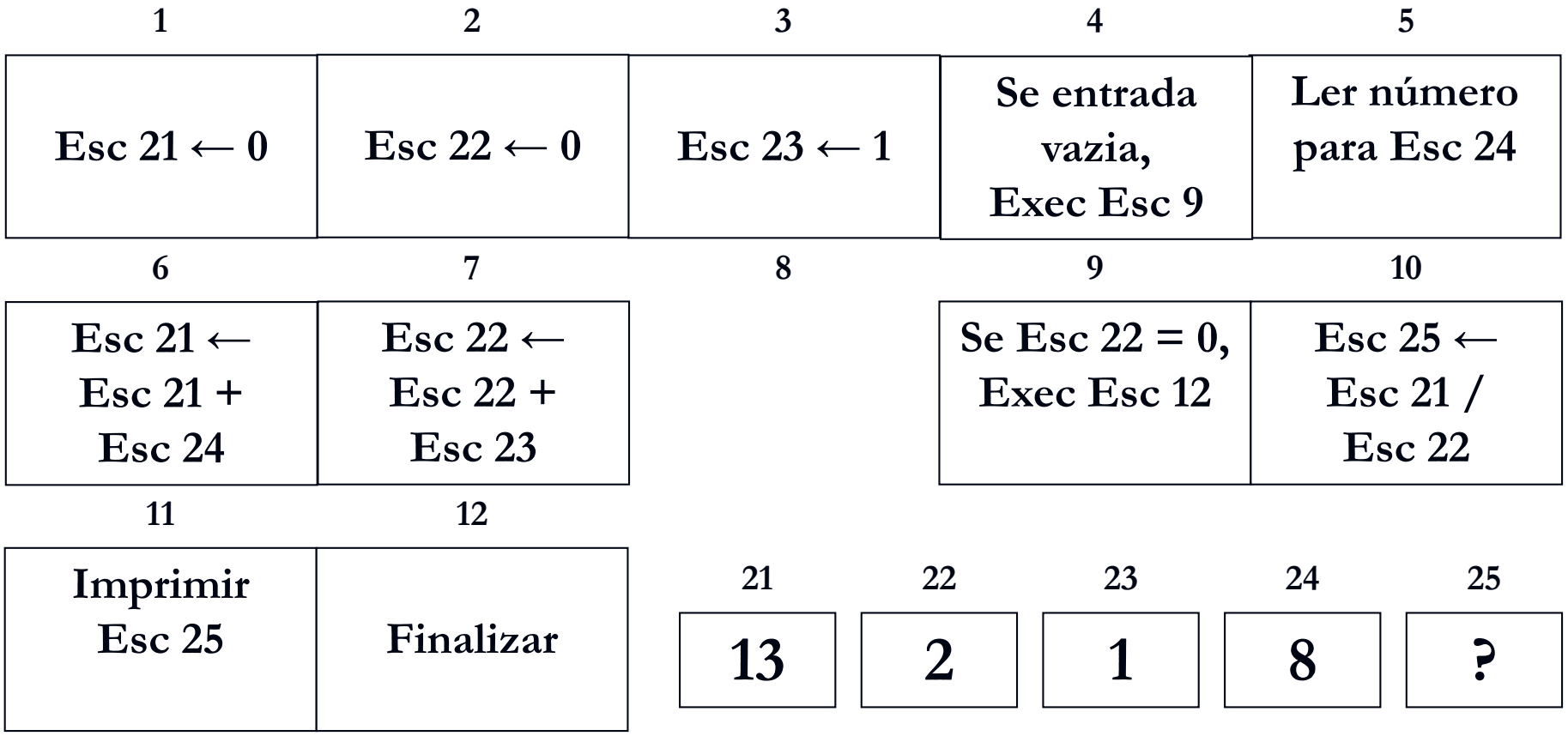
11	12
Imprimir Esc 25	Finalizar

21	22	23	24	25
13	2	1	8	?

Entrada

14

Operador


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 21 ← Esc 21 + Esc 24
Esc 22 ← Esc 22 + Esc 23
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
13
2
1
8
?
Exec Esc 4
Operador
Entrada
14

1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

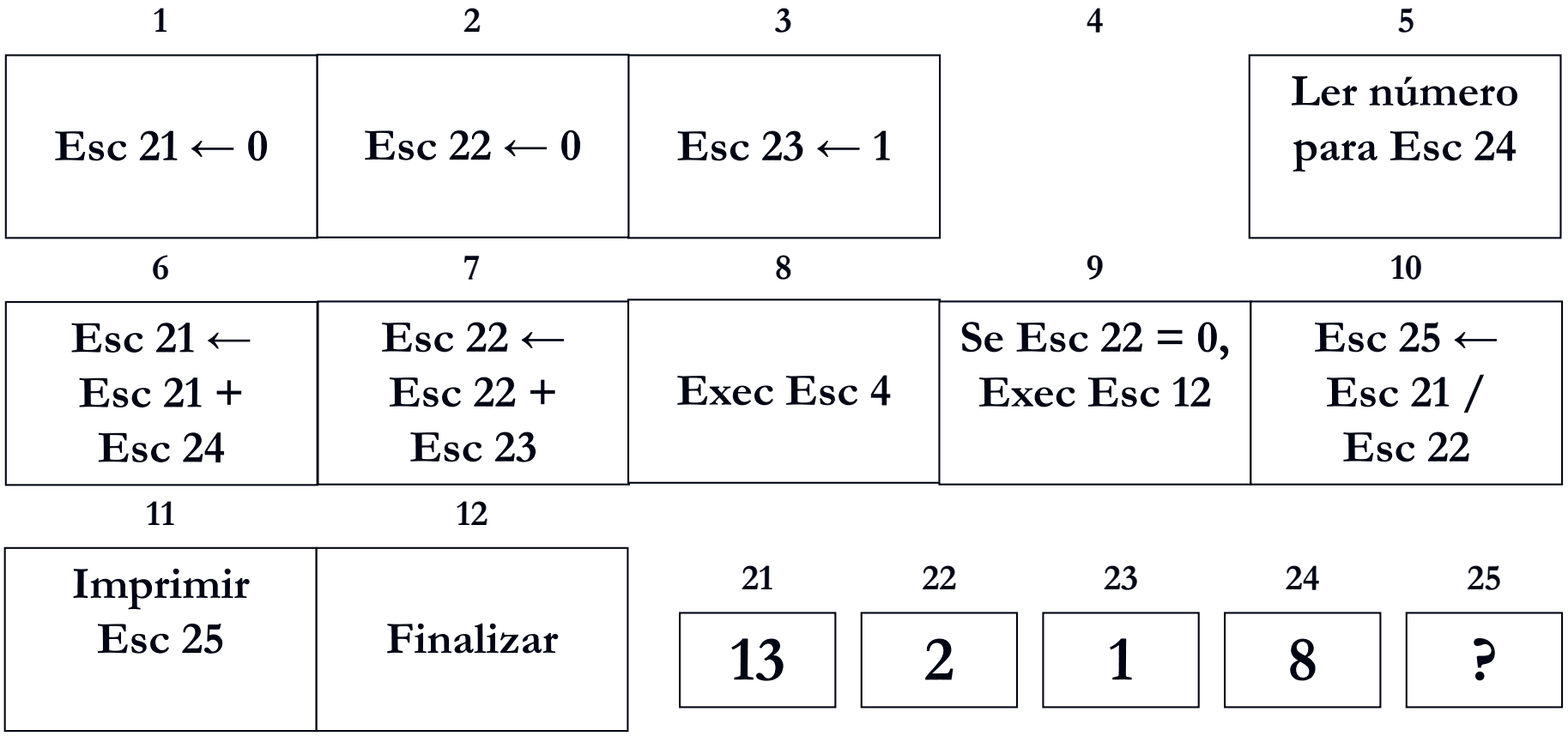
11	12
Imprimir Esc 25	Finalizar

21	22	23	24	25
13	2	1	8	?

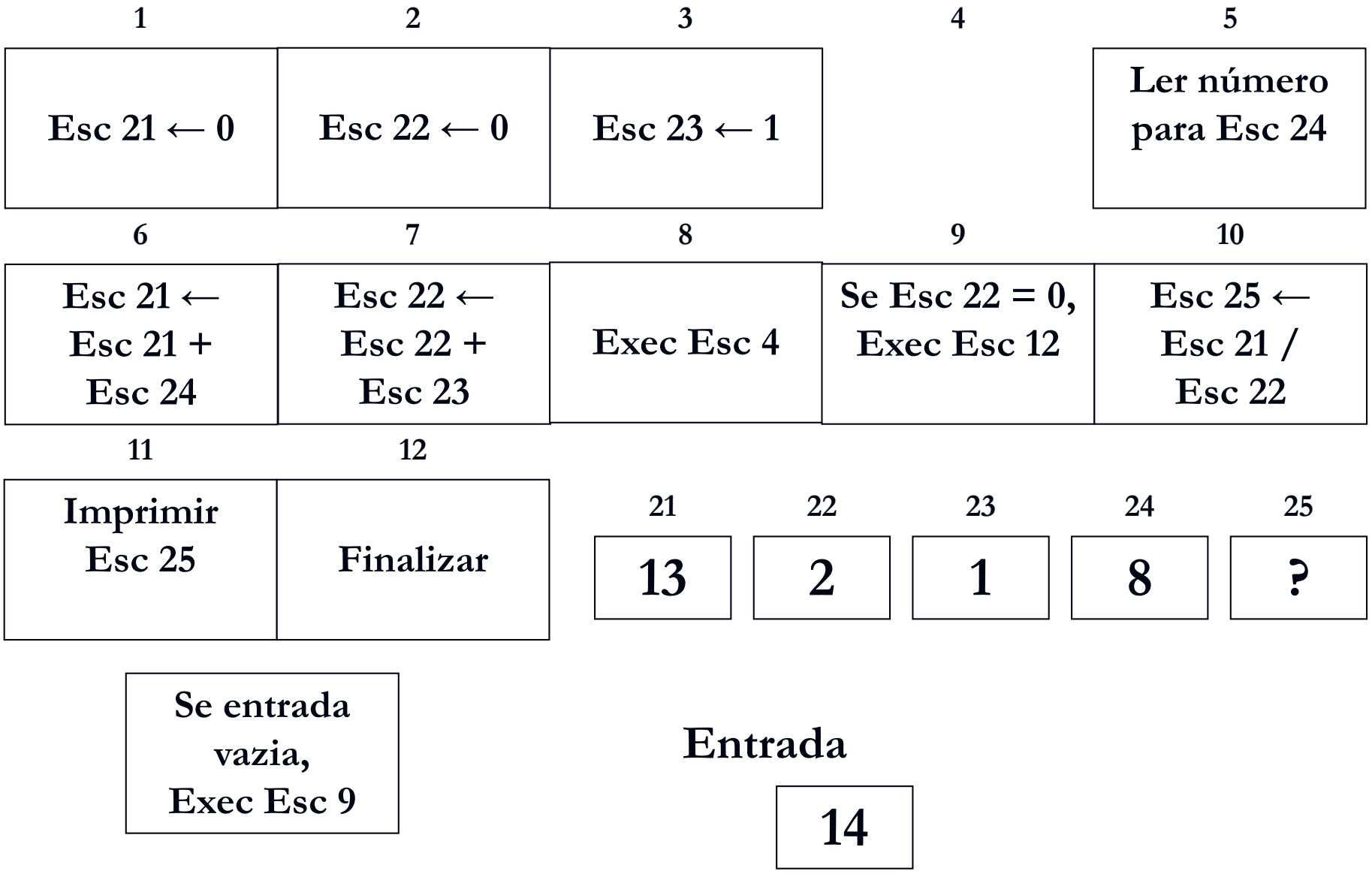
Entrada

14

Operador


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Ler número para Esc 24
6
7
8
9
10
Esc 21 ← Esc 21 + Esc 24
Esc 22 ← Esc 22 + Esc 23
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
13
2
1
8
?
Se entrada vazia, Exec Esc 9
Operador
Entrada
14

Há cartões na entrada: manter a sequência



1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

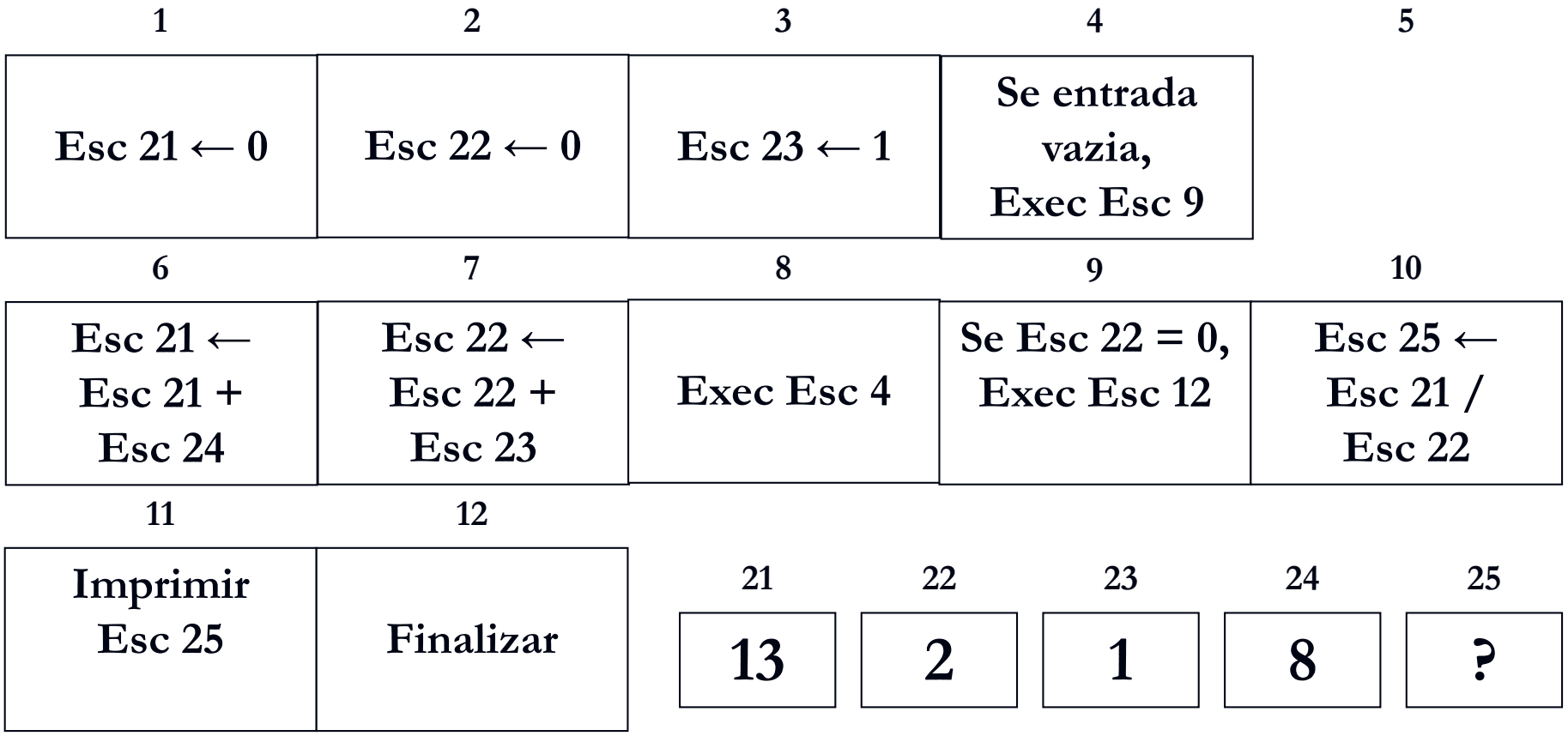
11	12
Imprimir Esc 25	Finalizar

21	22	23	24	25
13	2	1	8	?

Entrada

14

Operador


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
6
7
8
9
10
Esc 21 ← Esc 21 + Esc 24
Esc 22 ← Esc 22 + Esc 23
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
13
2
1
8
?
Ler número para Esc 24
Operador
Entrada
14

1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

11	12
Imprimir Esc 25	Finalizar

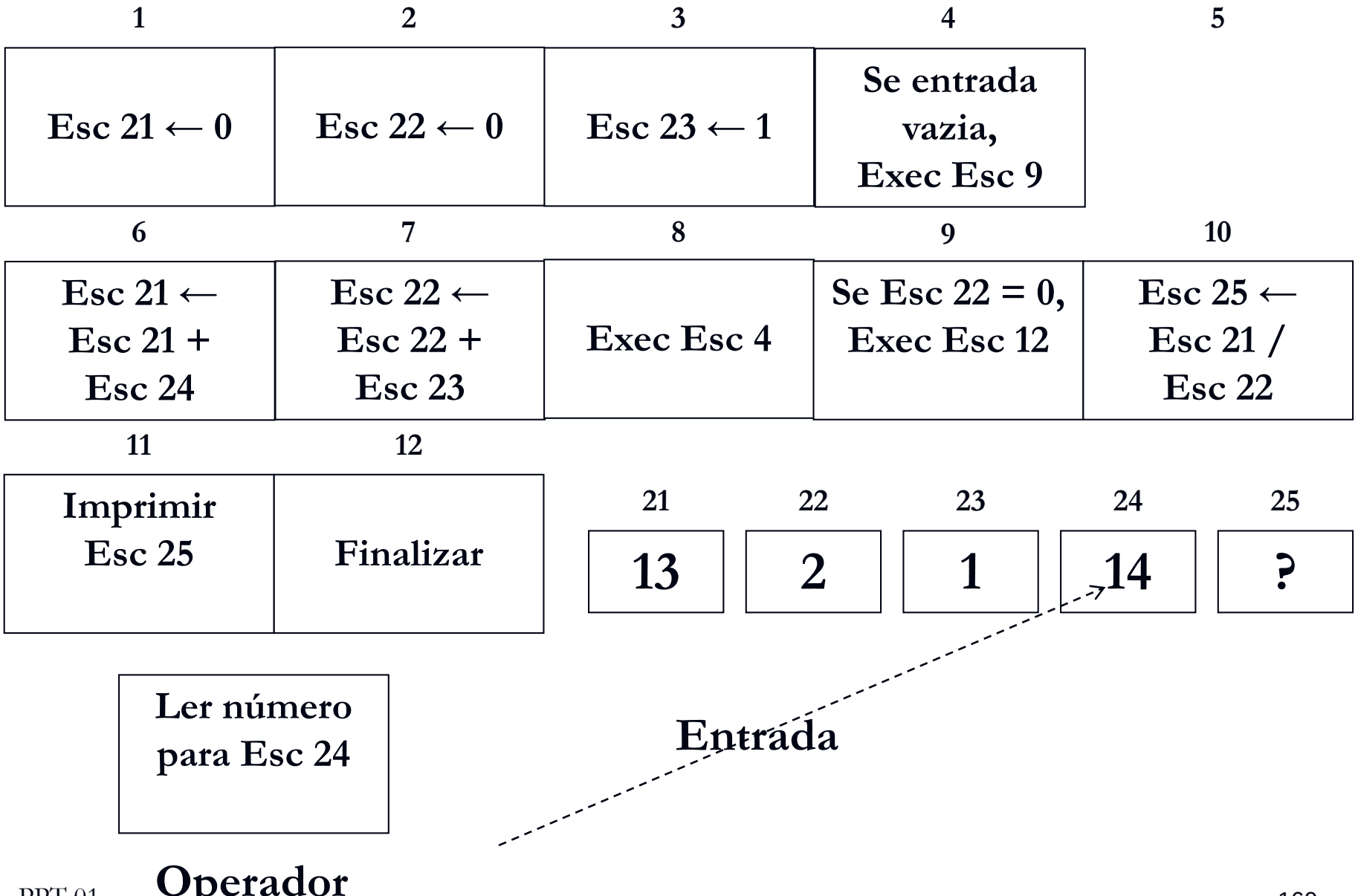
21	22	23	24	25
13	2	1	8	?

Ler número para Esc 24

Entrada

Operador

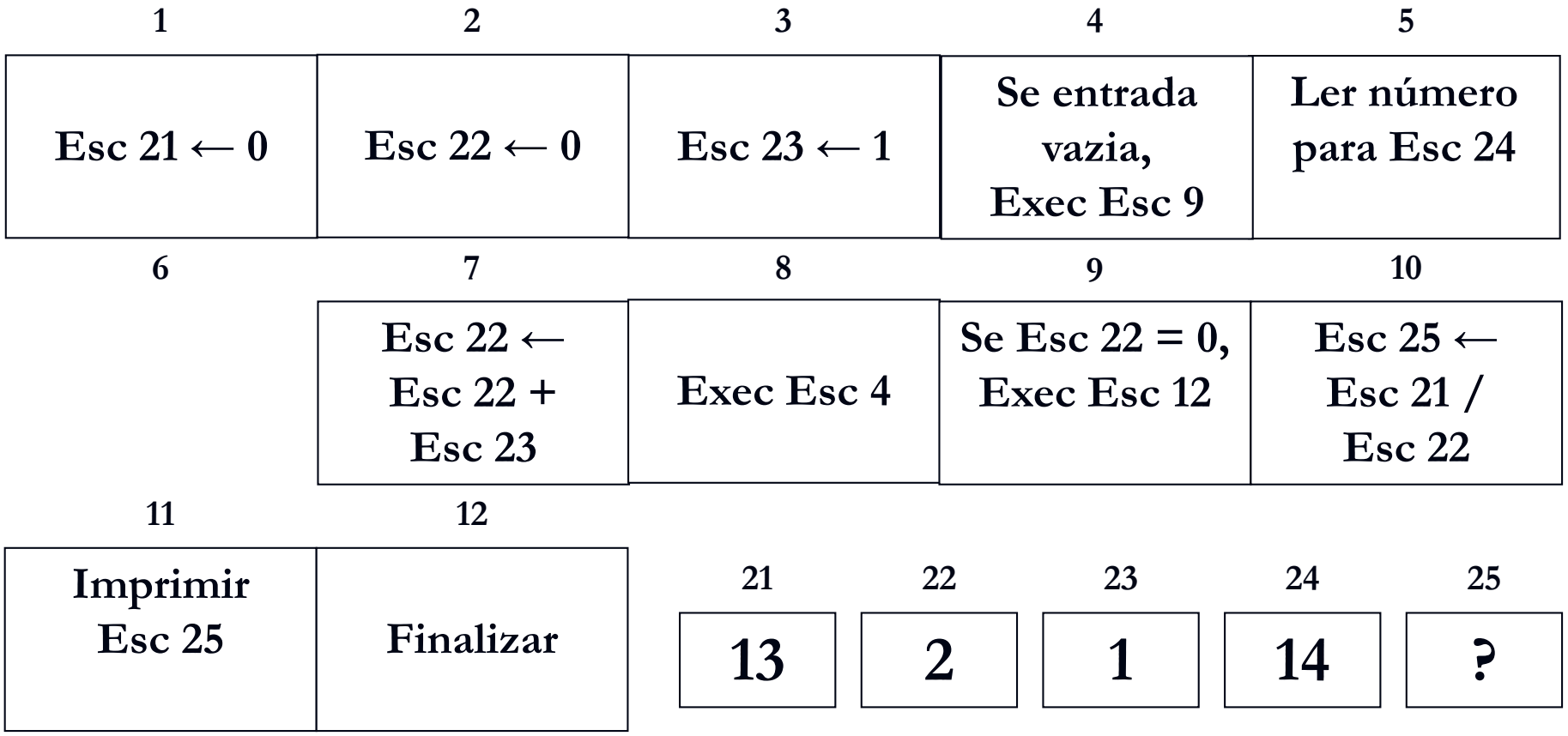
14

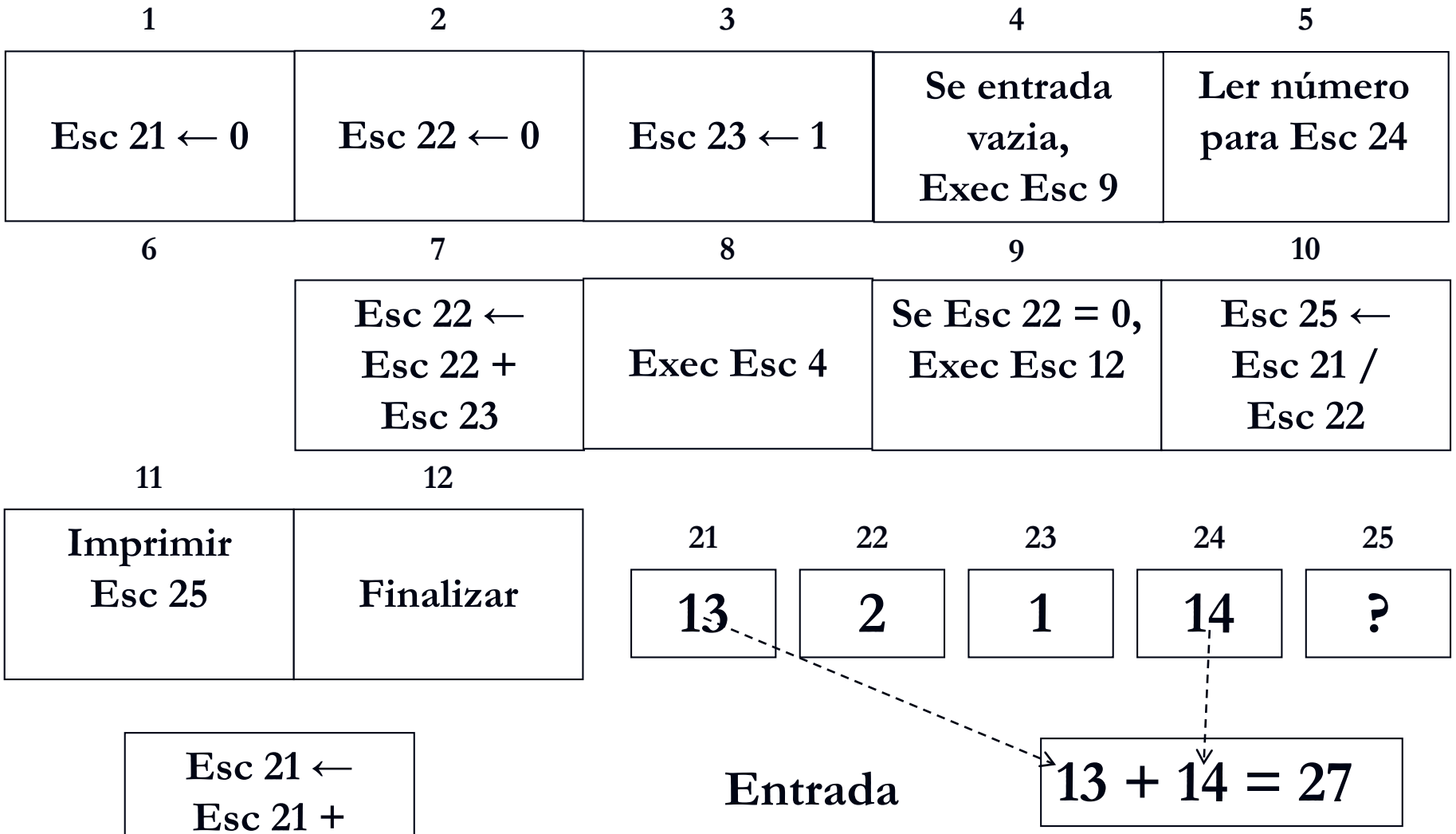

1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
6
7
8
9
10
Esc 21 ← Esc 21 + Esc 24
Esc 22 ← Esc 22 + Esc 23
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
13
2
1
14
?
Ler número para Esc 24
Entrada
Operador

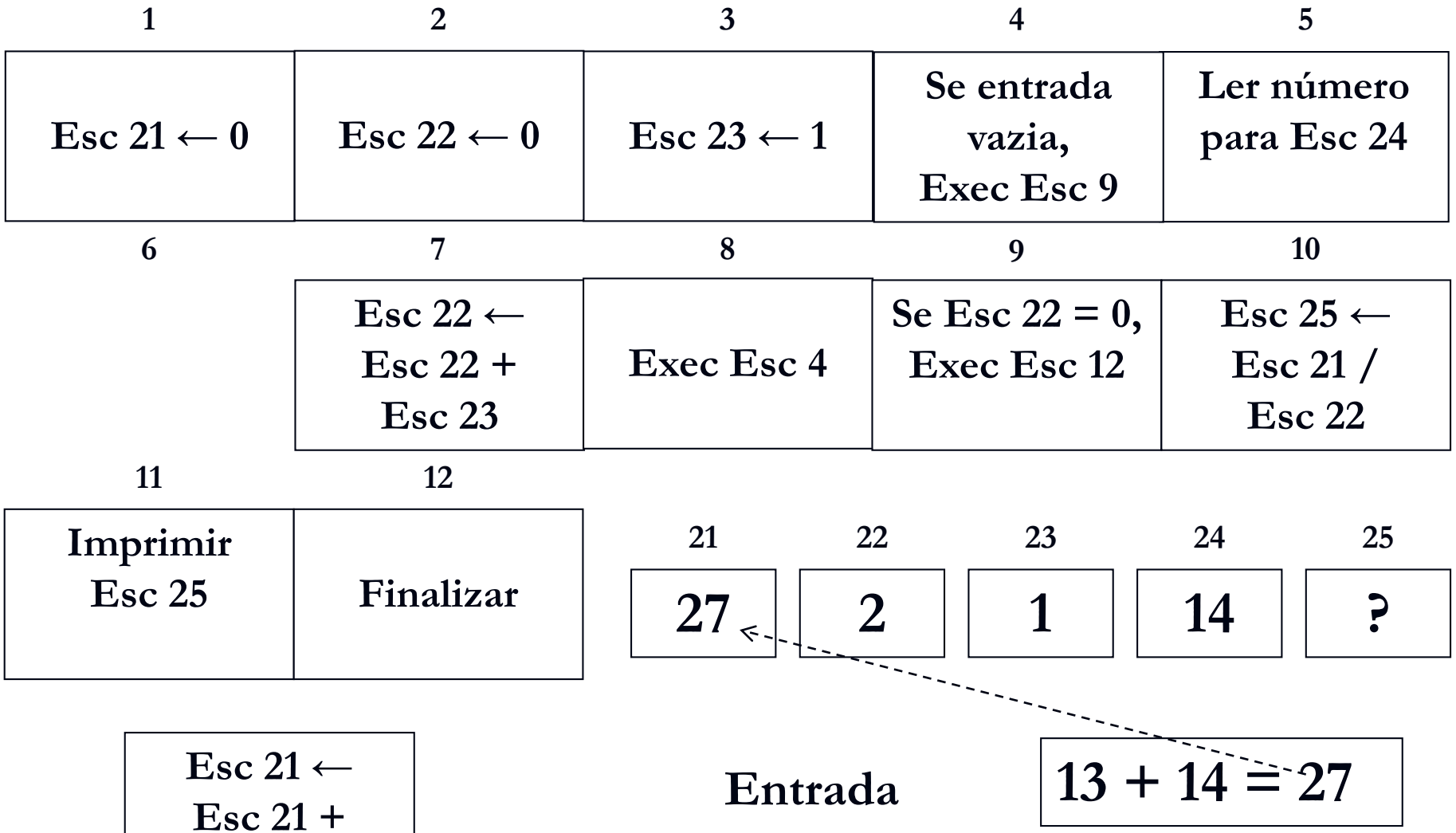
1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24
6	**7**	**8**	**9**	**10**
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22
11	**12**			
Imprimir Esc 25	Finalizar			

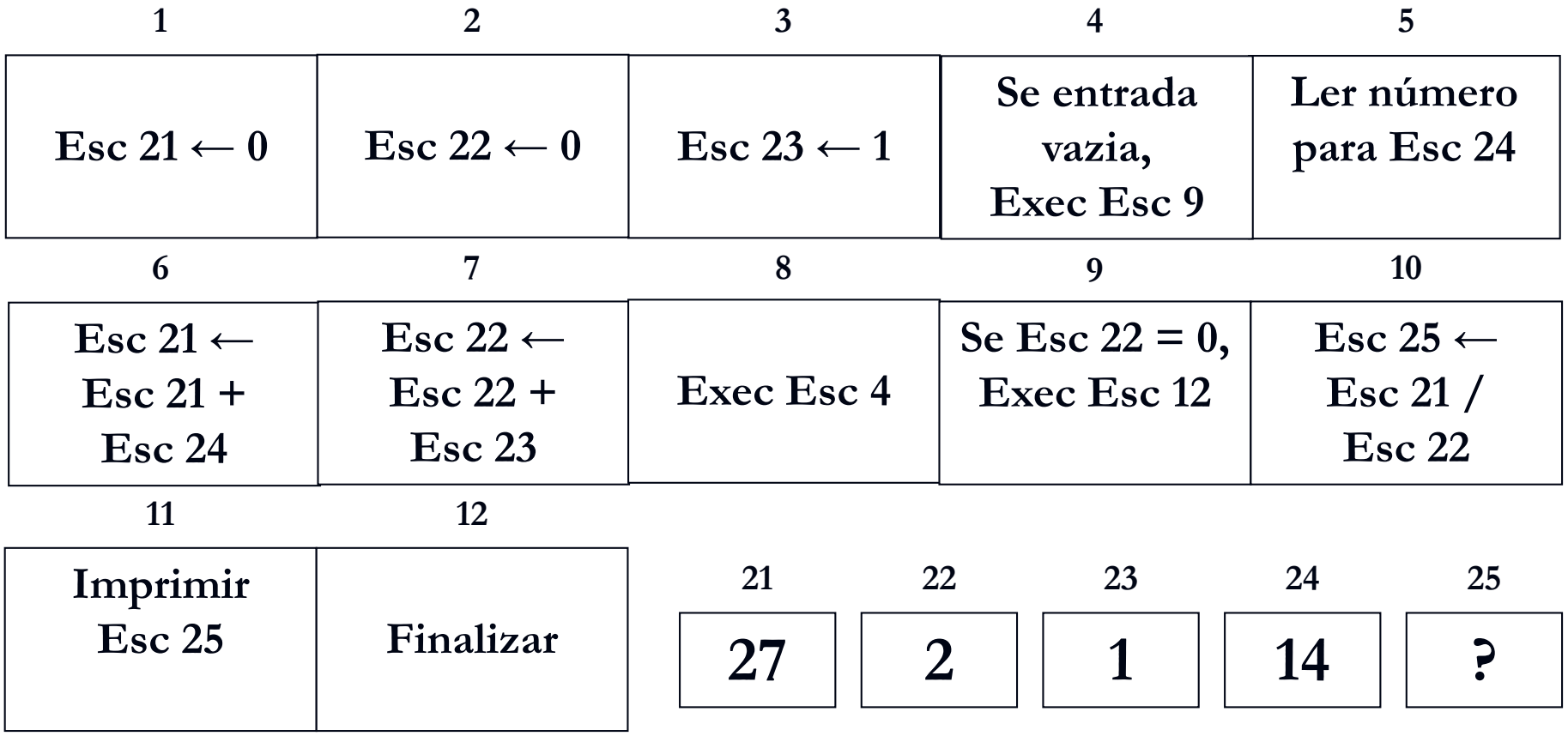
21	22	23	24	25
13	2	1	14	?

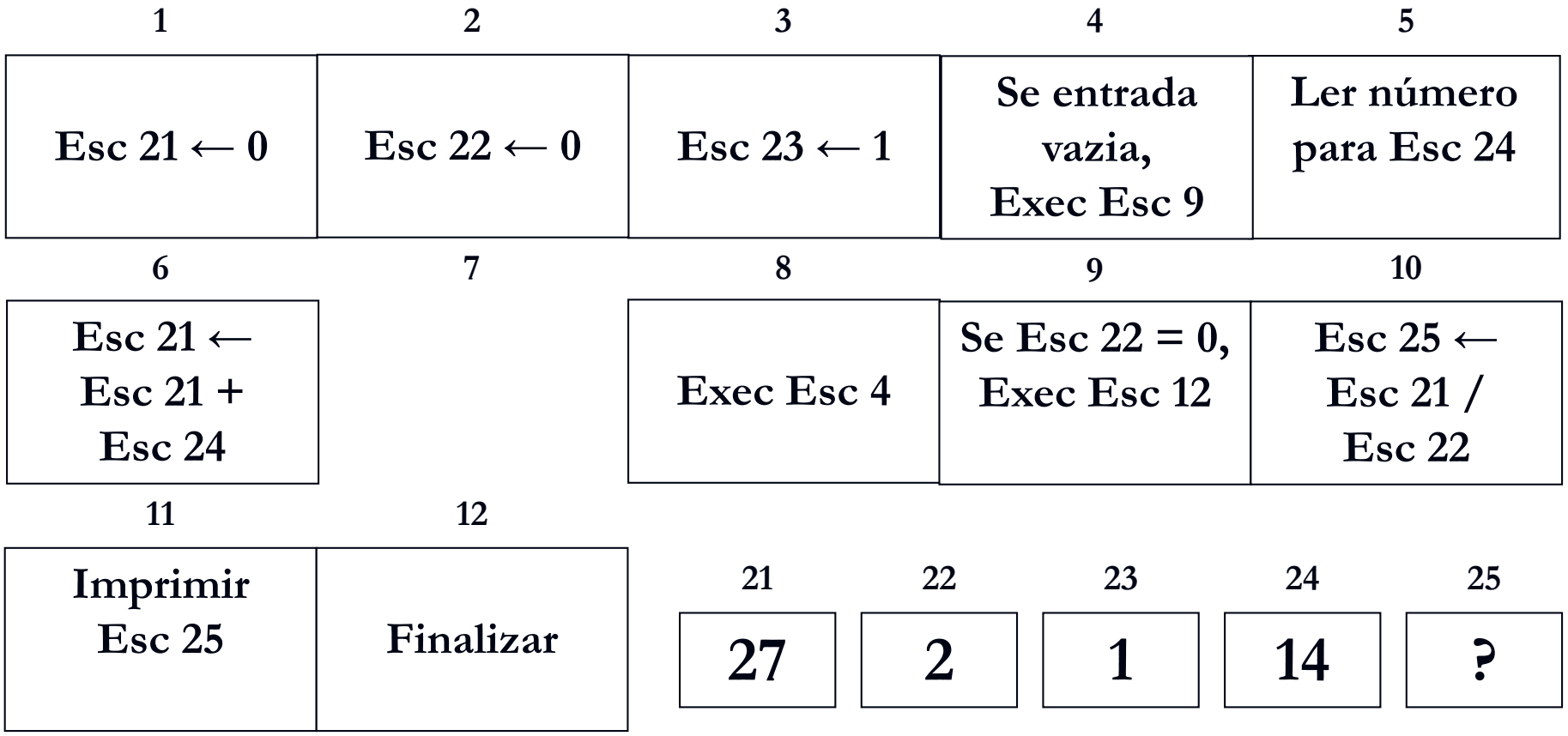
Entrada

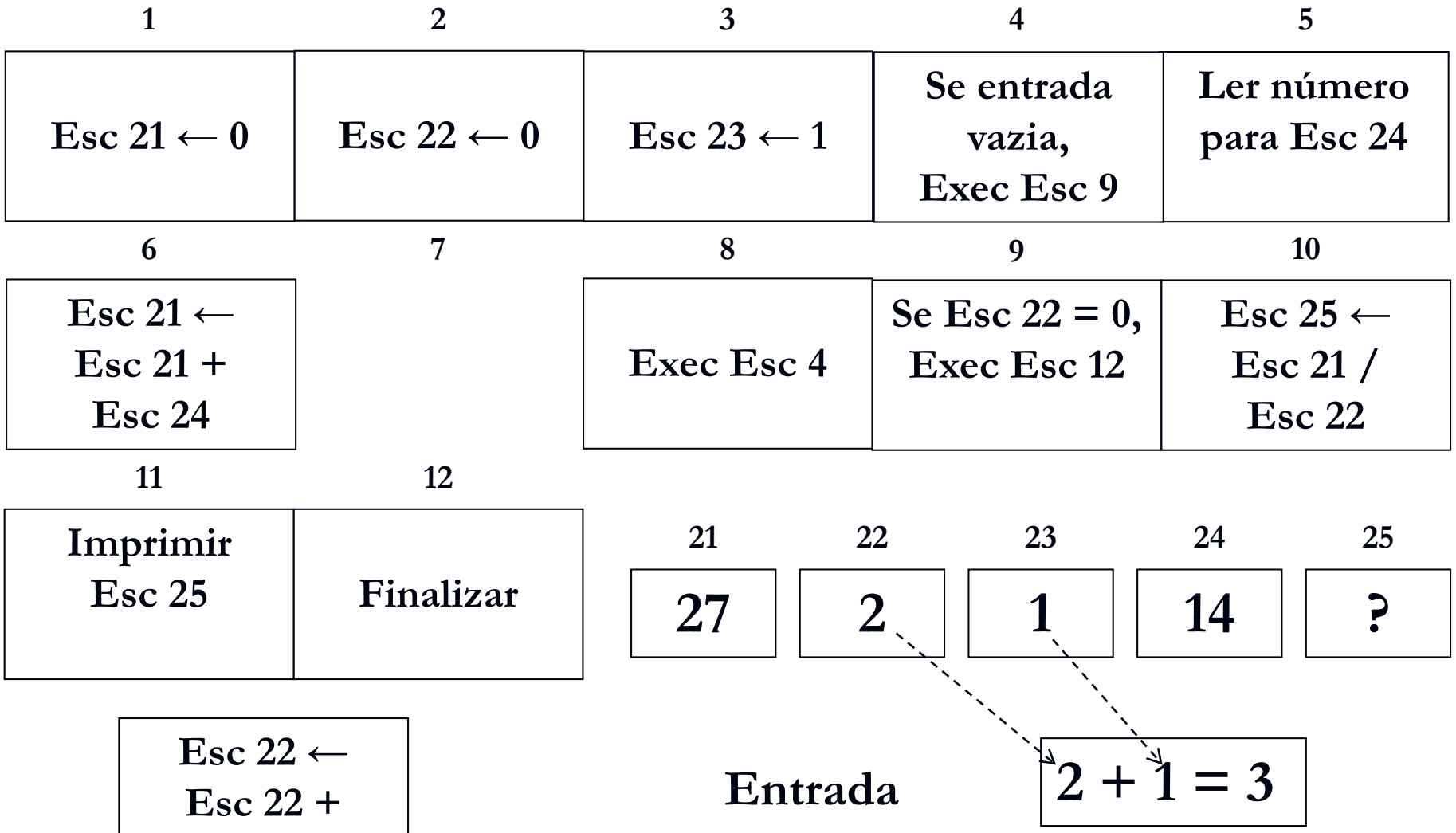

1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 22 ← Esc 22 + Esc 23
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
13
2
1
14
?
Esc 21 ← Esc 21 + Esc 24
Operador
Entrada

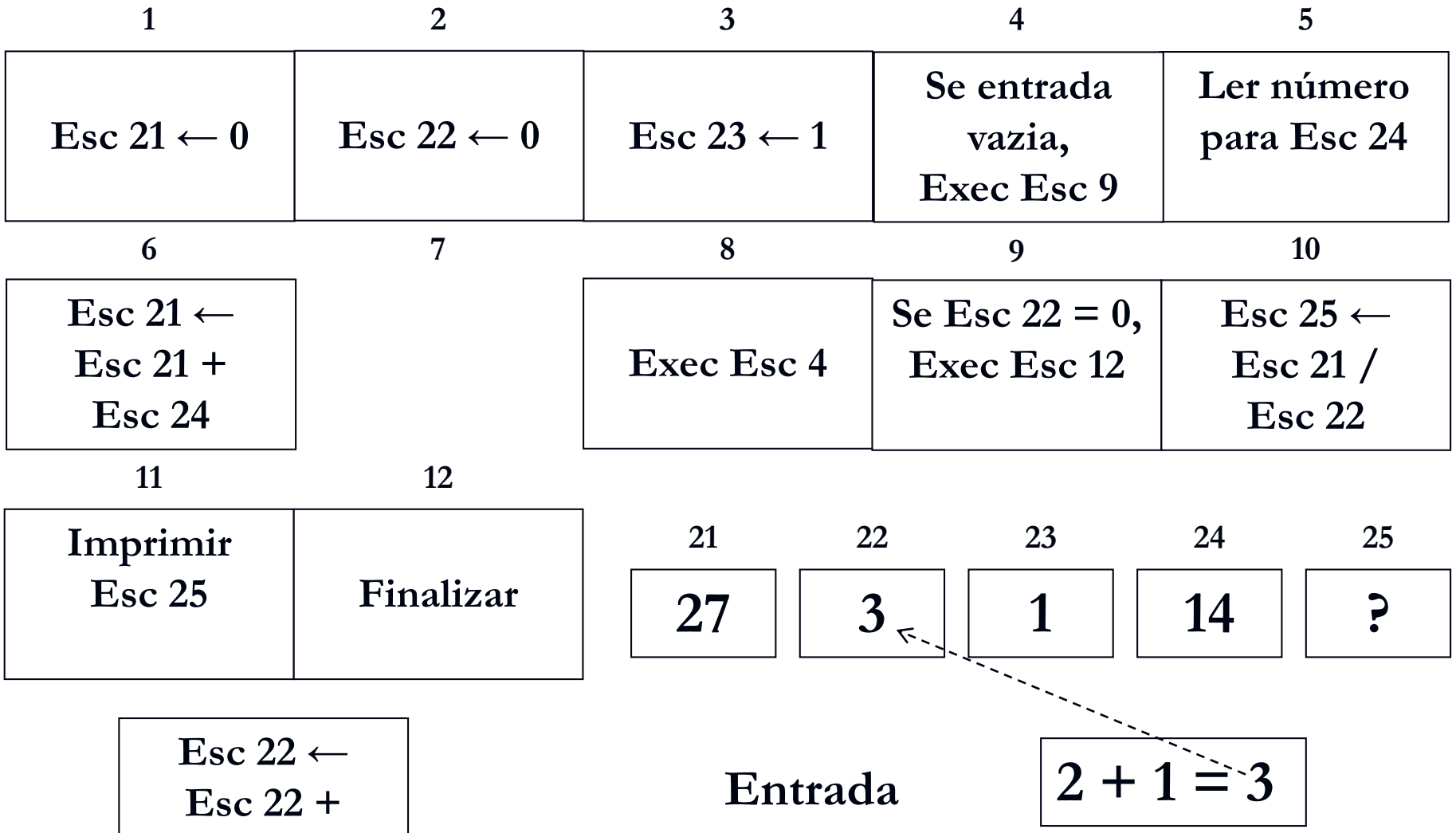

1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 22 ← Esc 22 + Esc 23
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
13
2
1
14
?
13 + 14 = 27
Entrada
Esc 21 ← Esc 21 + Esc 24
Operador


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 22 ← Esc 22 + Esc 23
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
27
2
1
14
?
13 + 14 = 27
Entrada
Esc 21 ← Esc 21 + Esc 24
Operador


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 21 ← Esc 21 + Esc 24
Esc 22 ← Esc 22 + Esc 23
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
27
2
1
14
?
Entrada


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 21 ← Esc 21 + Esc 24
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
27
2
1
14
?
Esc 22 ← Esc 22 + Esc 23
Entrada
Operador


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 21 ← Esc 21 + Esc 24
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
27
2
1
14
?
Esc 22 ← Esc 22 + Esc 23
Operador
Entrada
2 + 1 = 3


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 21 ← Esc 21 + Esc 24
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
27
3
1
14
?
Esc 22 ← Esc 22 + Esc 23
Operador
Entrada
2 + 1 = 3

1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

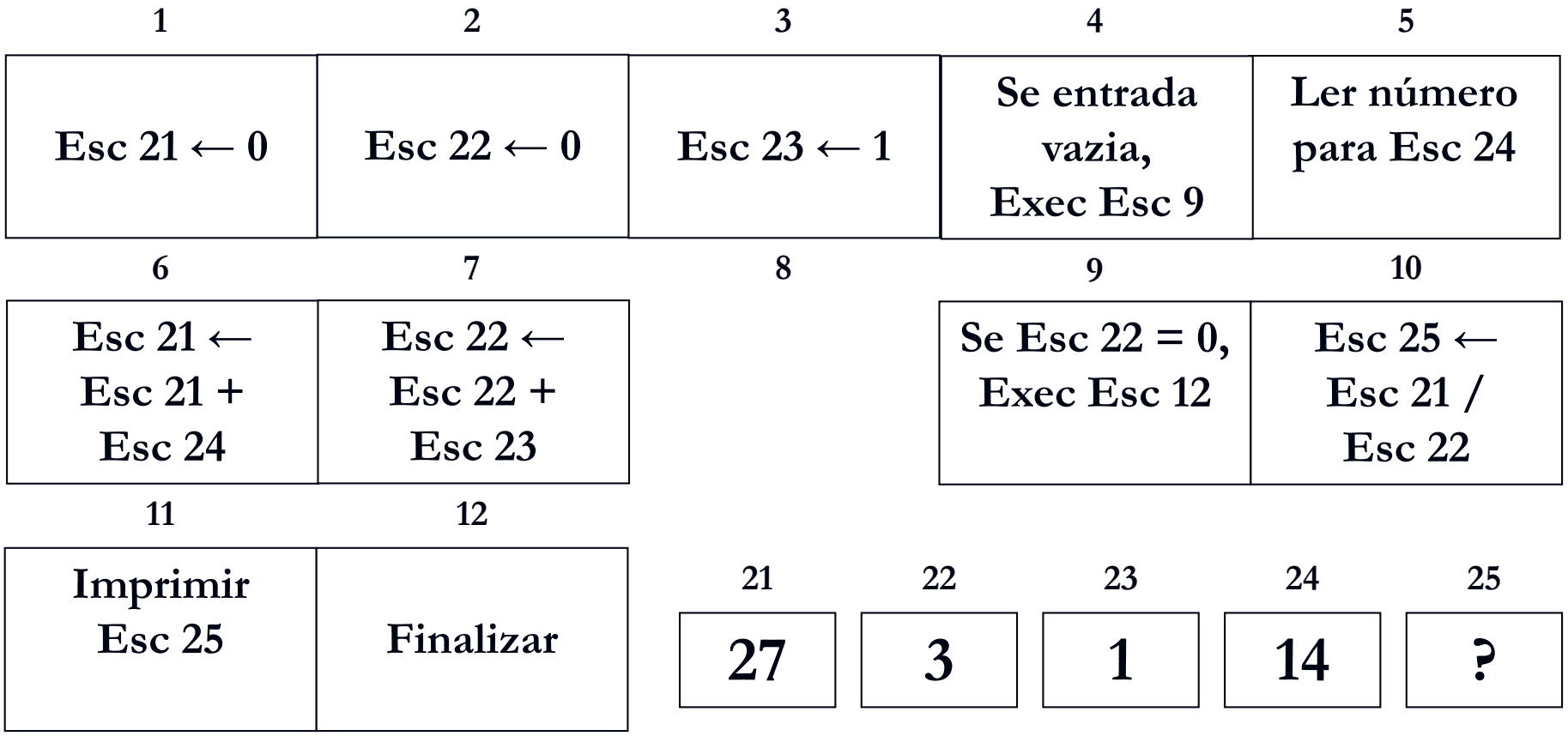
6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

11	12
Imprimir Esc 25	Finalizar

21	22	23	24	25
27	3	1	14	?

Entrada

Operador


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 21 ← Esc 21 + Esc 24
Esc 22 ← Esc 22 + Esc 23
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
27
3
1
14
?
Exec Esc 4
Operador
Entrada

1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

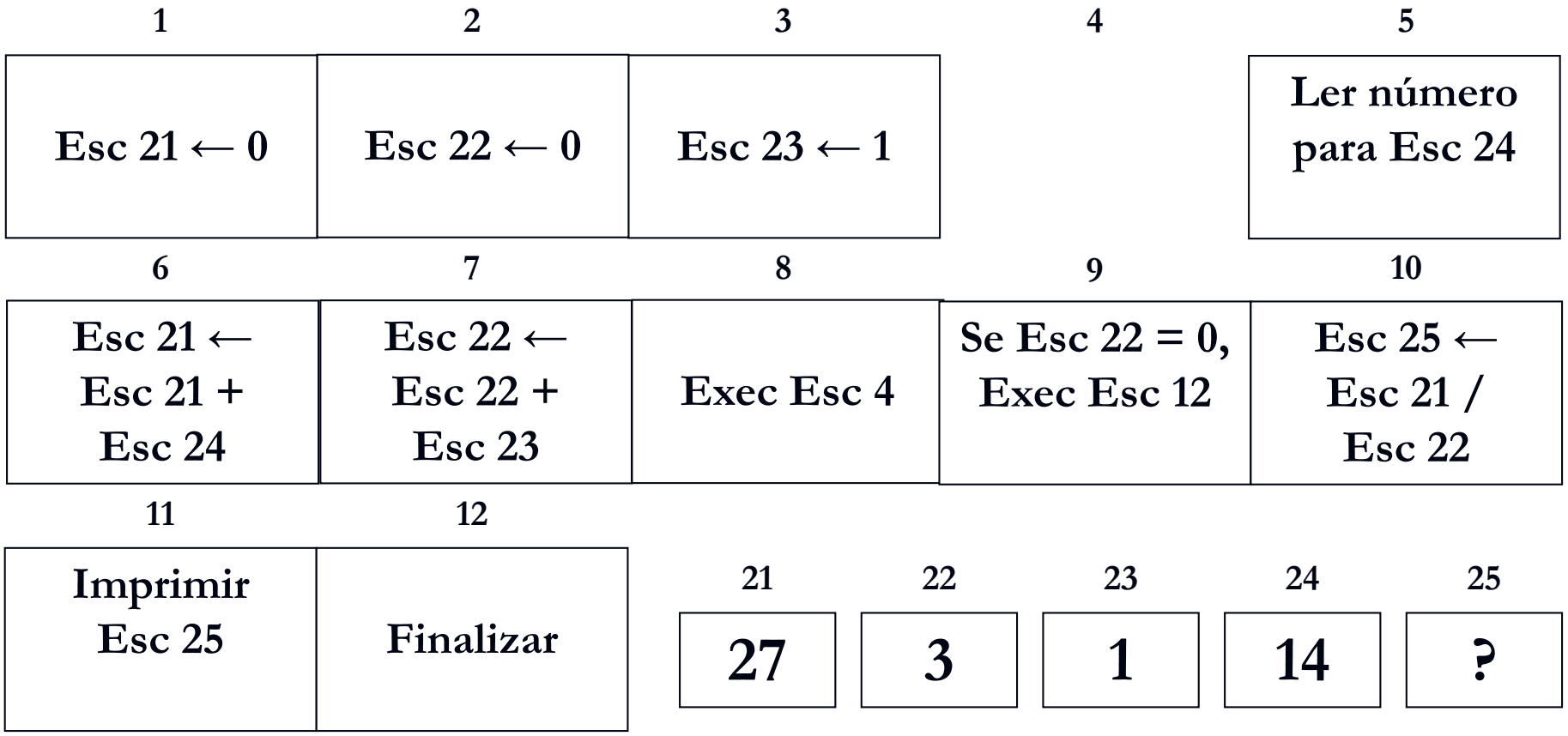
6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

11	12
Imprimir Esc 25	Finalizar

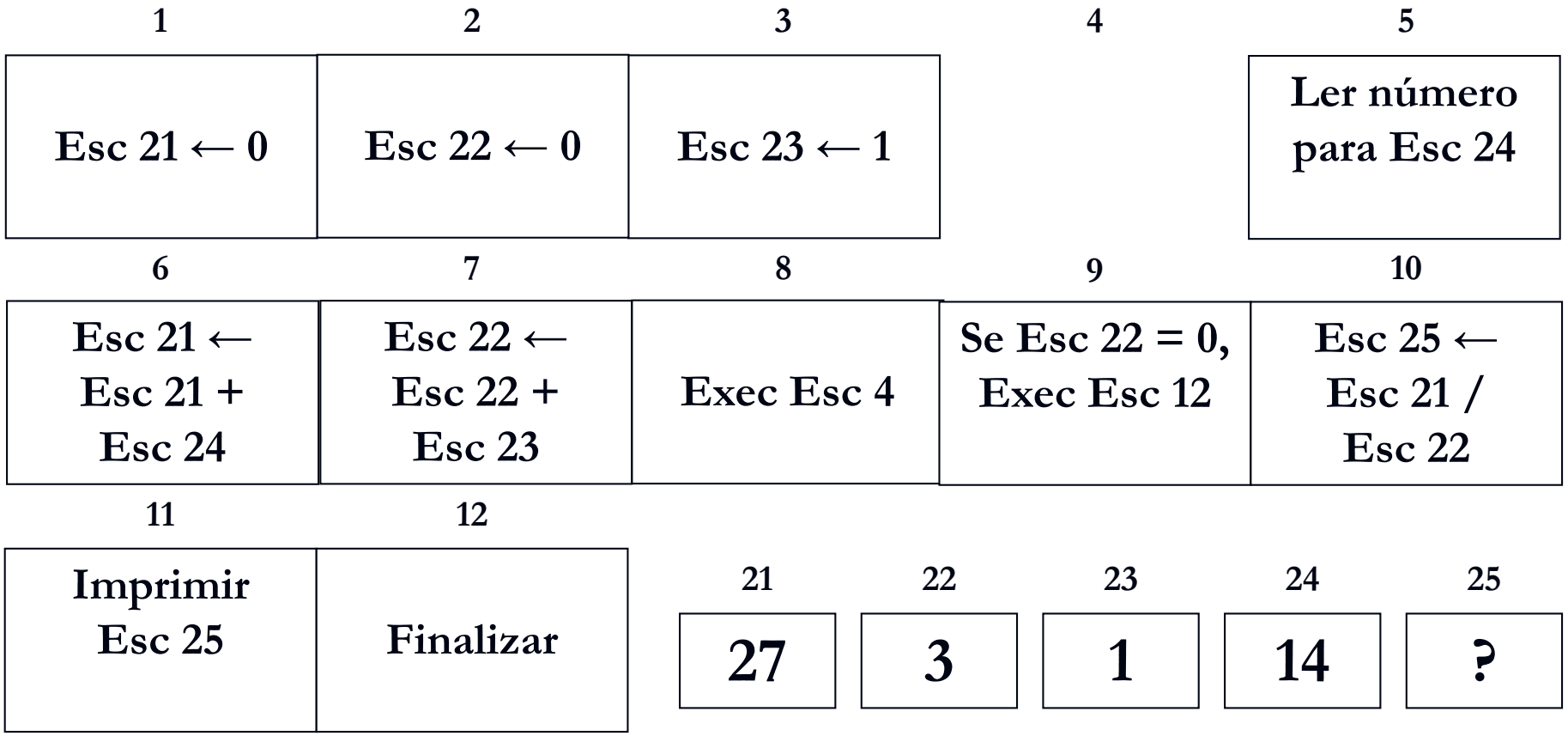
21	22	23	24	25
27	3	1	14	?

Entrada

Operador


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Ler número para Esc 24
6
7
8
9
10
Esc 21 ← Esc 21 + Esc 24
Esc 22 ← Esc 22 + Esc 23
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
27
3
1
14
?
Se entrada vazia, Exec Esc 9
Operador
Entrada

Entrada vazia: quebrar a sequência de execução



1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

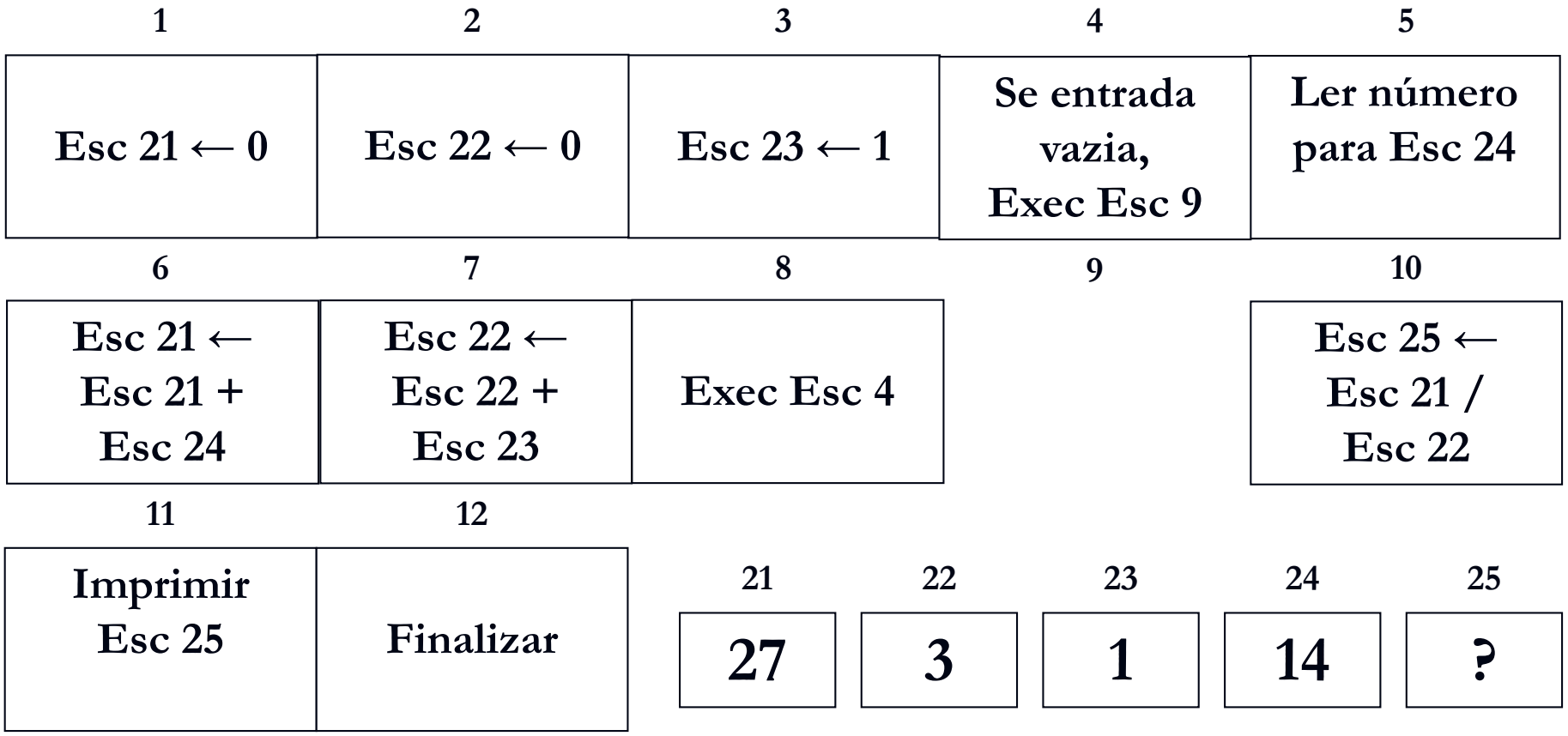
6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

11	12
Imprimir Esc 25	Finalizar

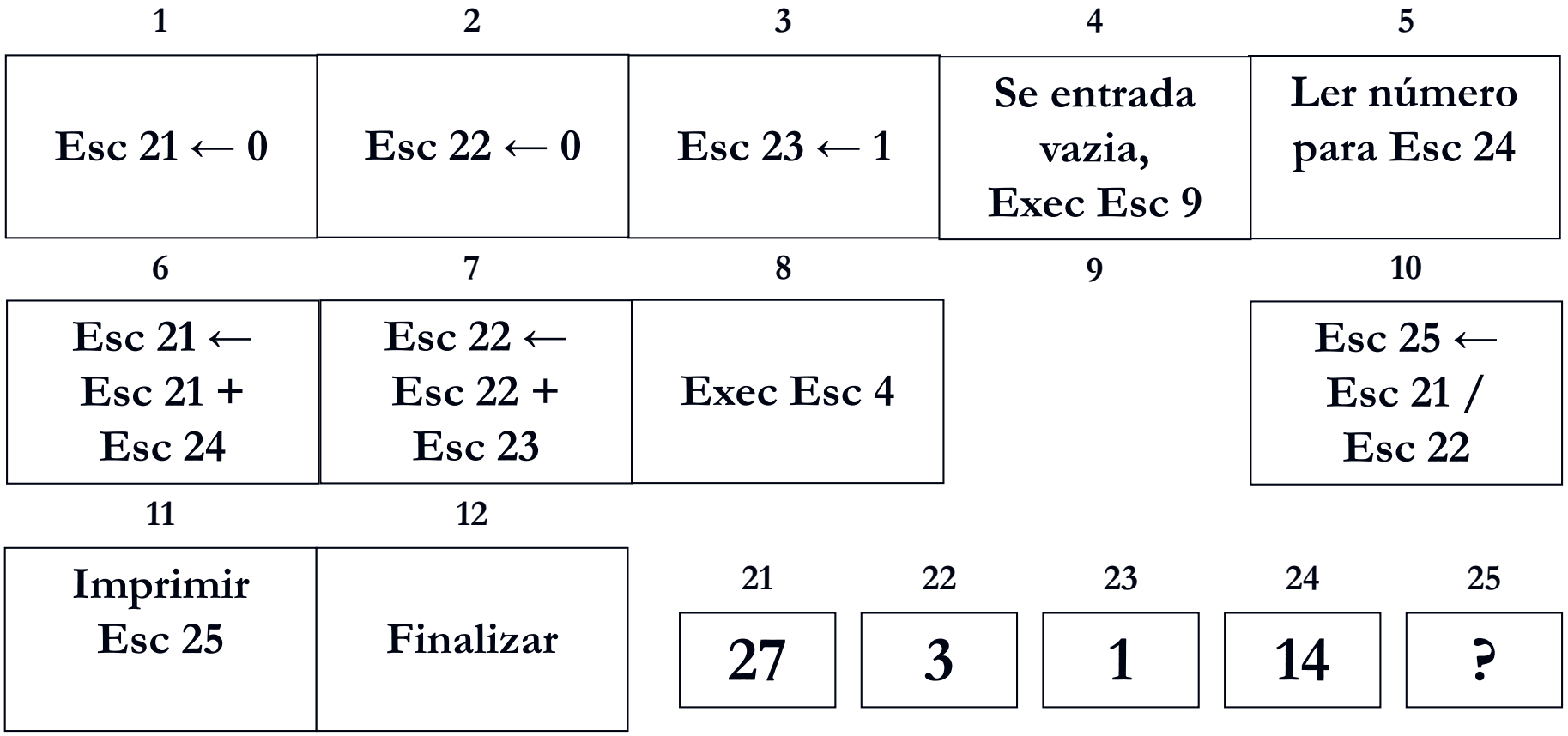
21	22	23	24	25
27	3	1	14	?

Entrada

Operador


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 21 ← Esc 21 + Esc 24
Esc 22 ← Esc 22 + Esc 23
Exec Esc 4
Esc 25 ← Esc 21 / Esc 22
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
27
3
1
14
?
Se Esc 22 = 0, Exec Esc 12
Operador
Entrada

Esc 22 ≠ 0: manter a sequência



1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

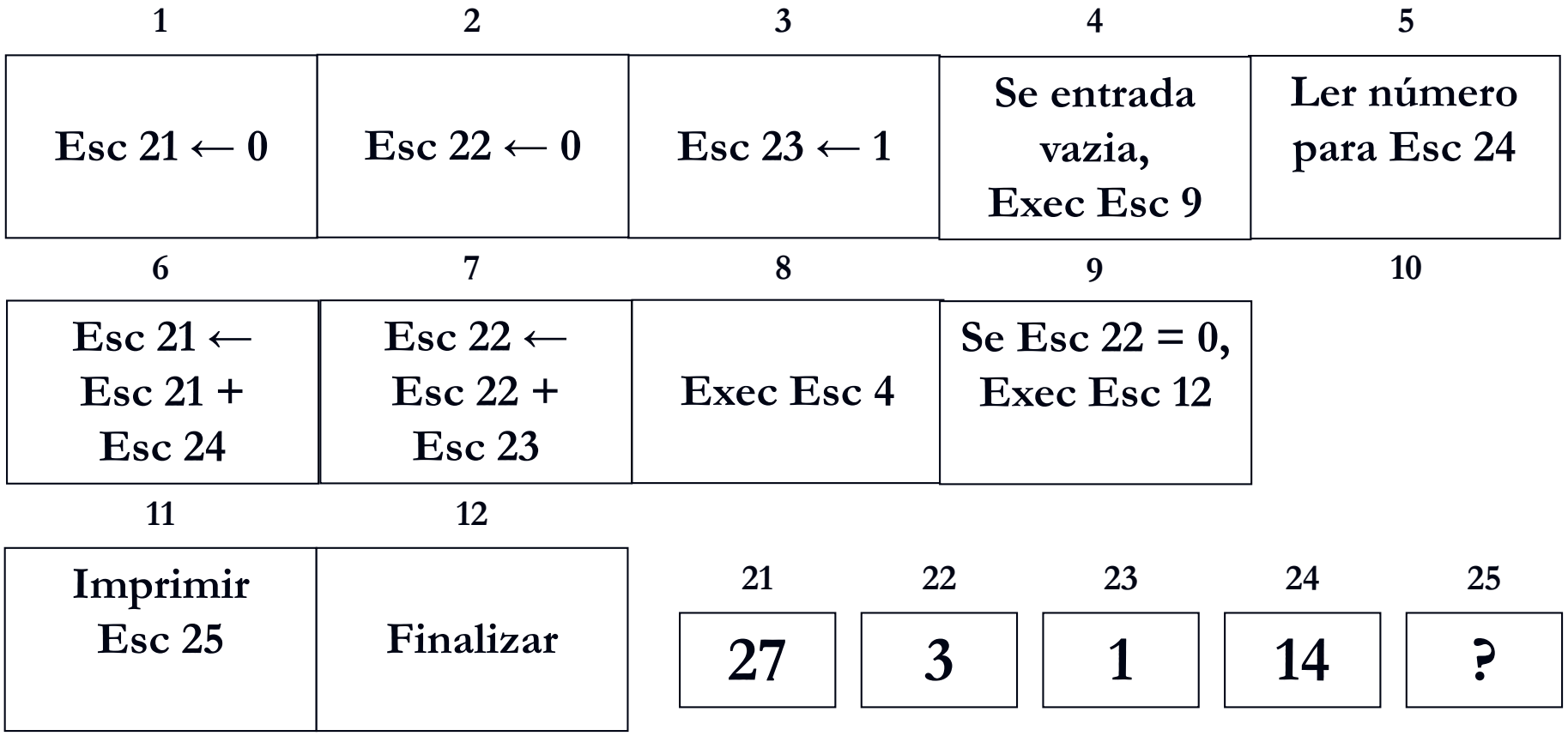
6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

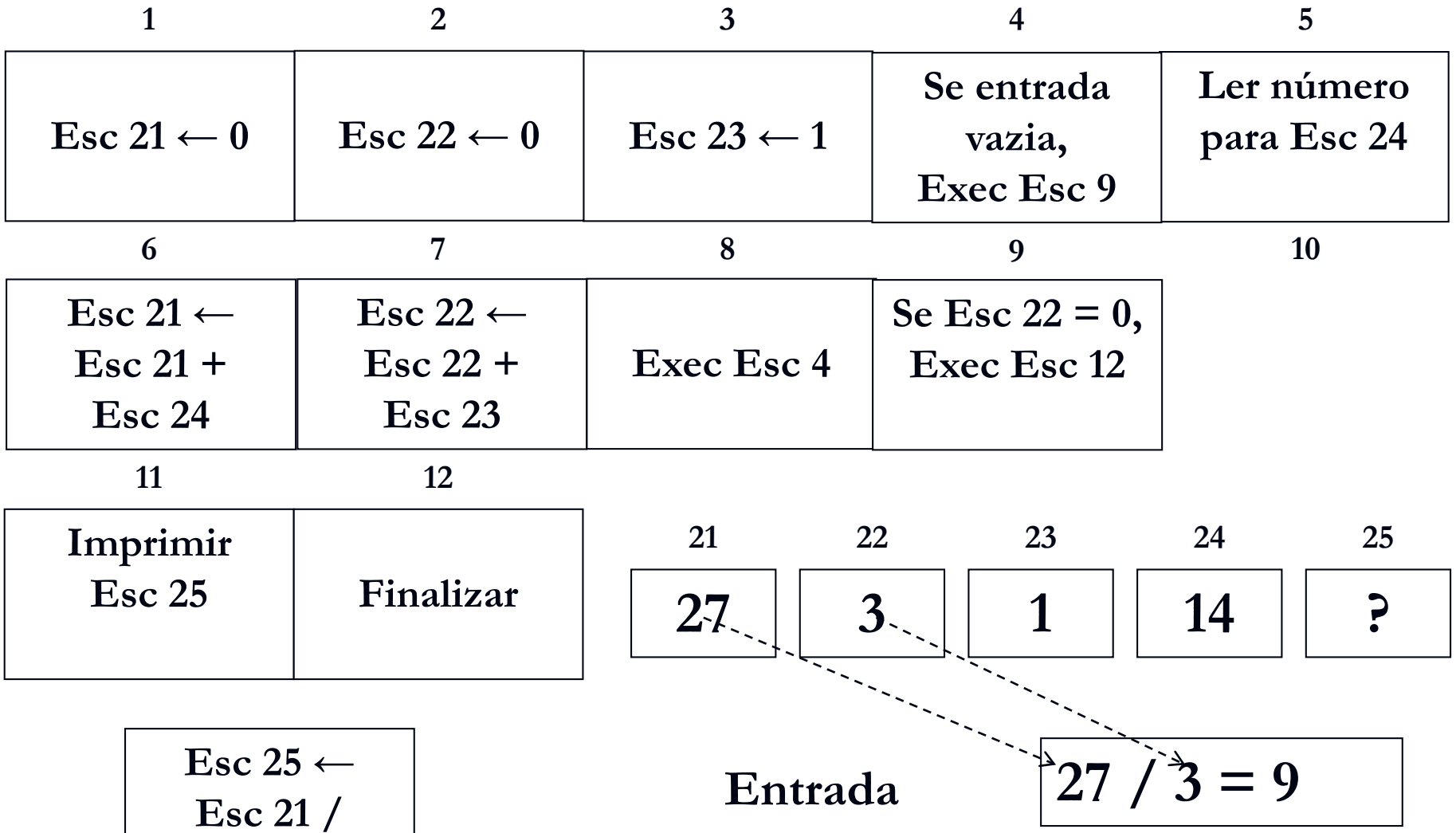
11	12
Imprimir Esc 25	Finalizar

21	22	23	24	25
27	3	1	14	?

Entrada

Operador


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 21 ← Esc 21 + Esc 24
Esc 22 ← Esc 22 + Esc 23
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
27
3
1
14
?
Esc 25 ← Esc 21 / Esc 22
Operador
Entrada


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 21 ← Esc 21 + Esc 24
Esc 22 ← Esc 22 + Esc 23
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
11
12
Imprimir Esc 25
Finalizar
21
22
23
24
25
27
3
1
14
?
Esc 25 ← Esc 21 / Esc 22
Operador
Entrada
27 / 3 = 9

1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	

11	12
Imprimir Esc 25	Finalizar

21	22	23	24	25
27	3	1	14	9

Esc 25 ← Esc 21 / Esc 22

Operador

Entrada

27 / 3 = 9

1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

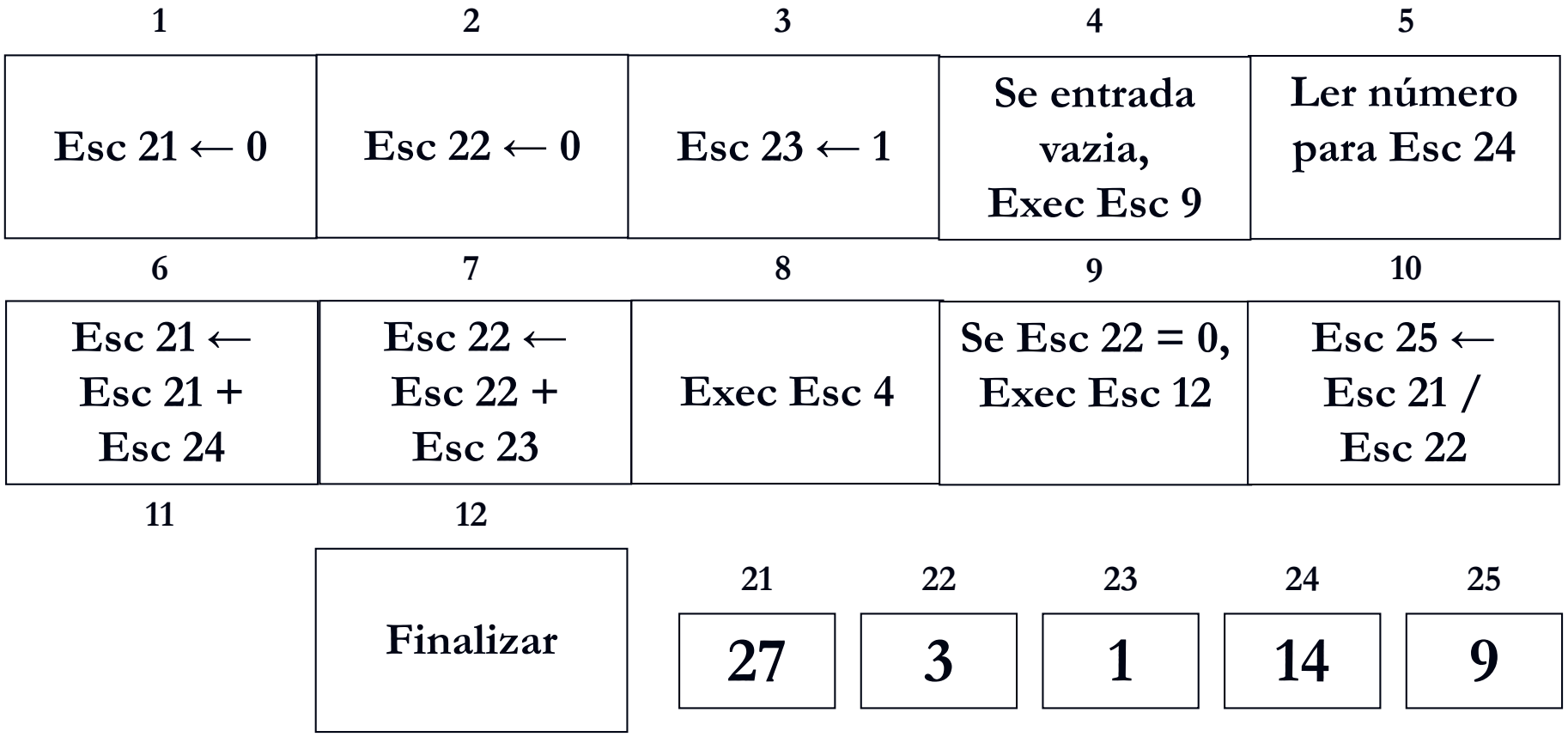
6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

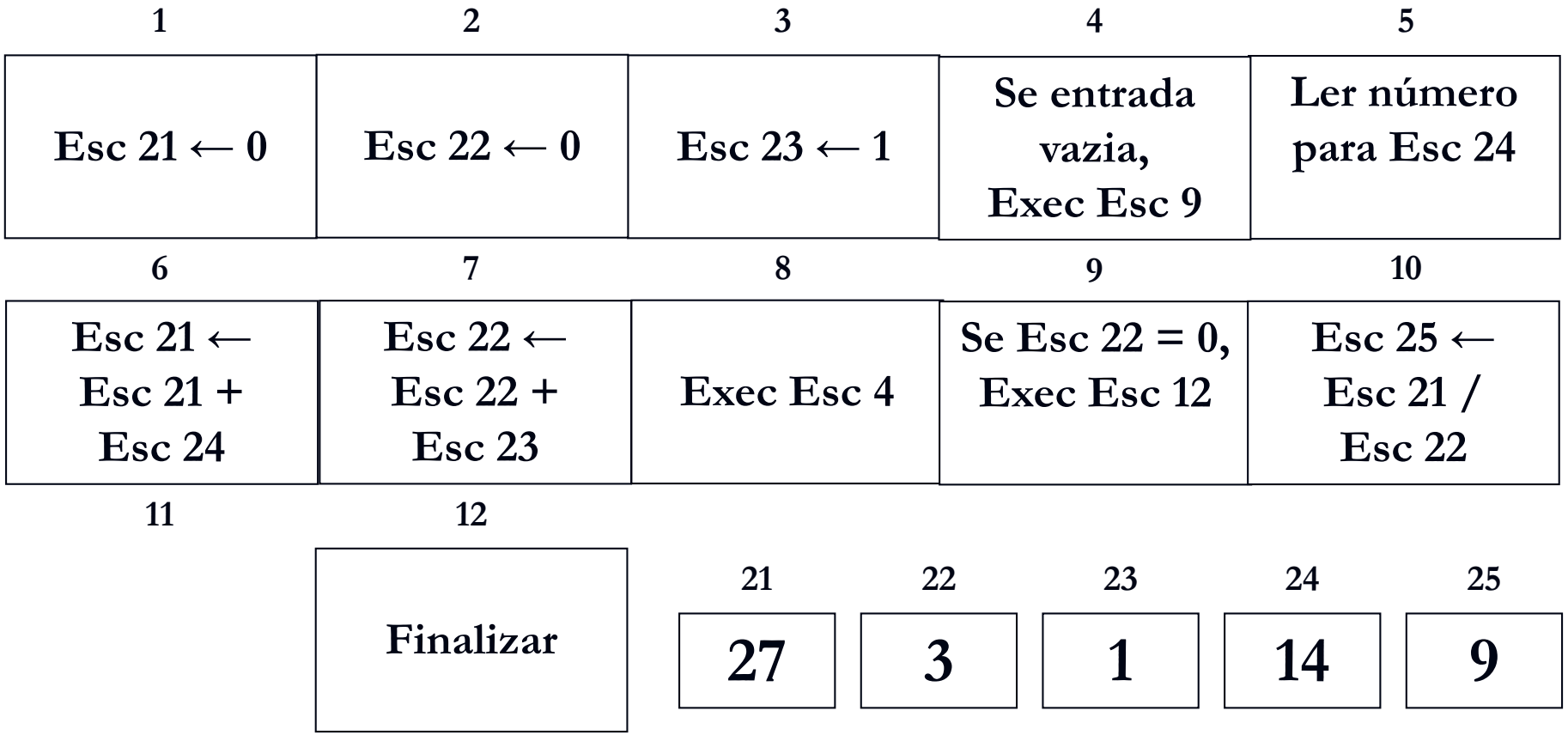
11	12
Imprimir Esc 25	Finalizar

21	22	23	24	25
27	3	1	14	9

Entrada

Operador


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 21 ← Esc 21 + Esc 24
Esc 22 ← Esc 22 + Esc 23
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Finalizar
21
22
23
24
25
27
3
1
14
9
Imprimir Esc 25
Operador
Entrada


1
2
3
4
5
Esc 21 ← 0
Esc 22 ← 0
Esc 23 ← 1
Se entrada vazia, Exec Esc 9
Ler número para Esc 24
6
7
8
9
10
Esc 21 ← Esc 21 + Esc 24
Esc 22 ← Esc 22 + Esc 23
Exec Esc 4
Se Esc 22 = 0, Exec Esc 12
Esc 25 ← Esc 21 / Esc 22
11
12
Finalizar
21
22
23
24
25
27
3
1
14
9
Imprimir Esc 25
Operador
Entrada
Folha de papel
9

1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

11	12
Imprimir Esc 25	Finalizar

21	22	23	24	25
27	3	1	14	9

Folha de papel

9

Entrada

Operador

1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

11	12
Imprimir Esc 25	

21	22	23	24	25
27	3	1	14	9

Finalizar

Operador

Entrada

Folha de papel

9

Trabalho encerrado !

1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

11	12
Imprimir Esc 25	

21	22	23	24	25
27	3	1	14	9

Finalizar

Operador

Entrada

Folha de papel

9

1	2	3	4	5
Esc 21 ← 0	**Esc 22 ← 0**	**Esc 23 ← 1**	**Se entrada vazia, Exec Esc 9**	**Ler número para Esc 24**
6	**7**	**8**	**9**	**10**
Esc 21 ← Esc 21 + Esc 24	**Esc 22 ← Esc 22 + Esc 23**	**Exec Esc 4**	**Se Esc 22 = 0, Exec Esc 12**	**Esc 25 ← Esc 21 / Esc 22**
11	**12**			
Imprimir Esc 25	**Finalizar**			

Qual a finalidade deste conjunto de instruções?

Encontrar a média aritmética dos números lidos do balcão de entrada

Finalidade dos escaninhos **Esc 21** a **Esc 25:**

1	2	3	4	5
Esc 21 ← 0	**Esc 22 ← 0**	**Esc 23 ← 1**	**Se entrada vazia, Exec Esc 9**	**Ler número para Esc 24**

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	**Esc 22 ← Esc 22 + Esc 23**	**Exec Esc 4**	**Se Esc 22 = 0, Exec Esc 12**	**Esc 25 ← Esc 21 / Esc 22**

11	12
Imprimir Esc 25	**Finalizar**

21	22	23	24	25
27	3	1	14	9

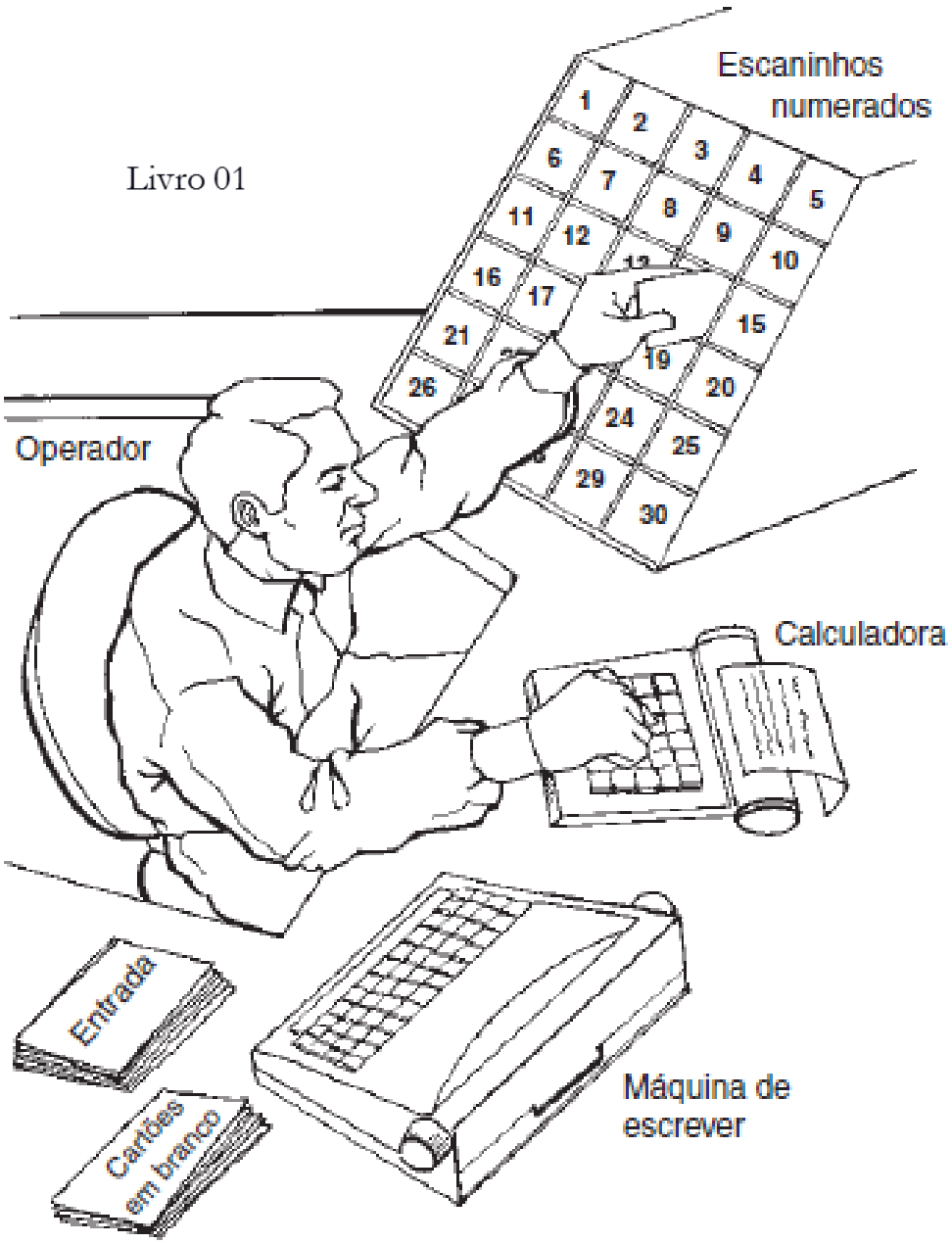
Esc 21: guardar a somatória dos números lidos

Esc 22: guardar a quantidade de números lidos

Esc 23: guardar a constante 1

Esc 24: guardar cada número lido

Esc 25: guardar a média aritmética



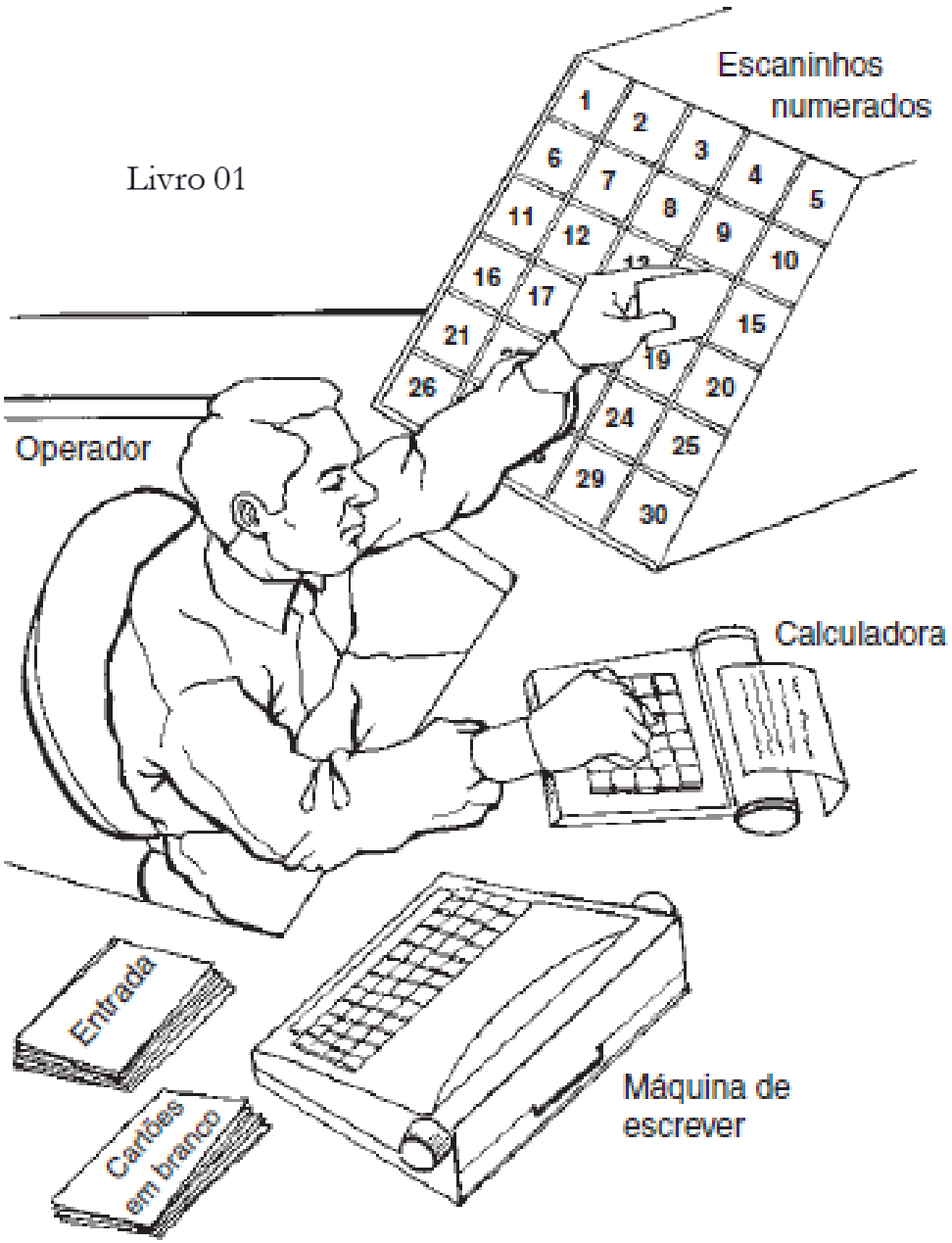
Hardware:

- Operador
- Calculadora
- Máquina de escrever
- Estante com escaninhos

Software:

- Conteúdo dos escaninhos
- Cartões brancos e de entrada

Programa:



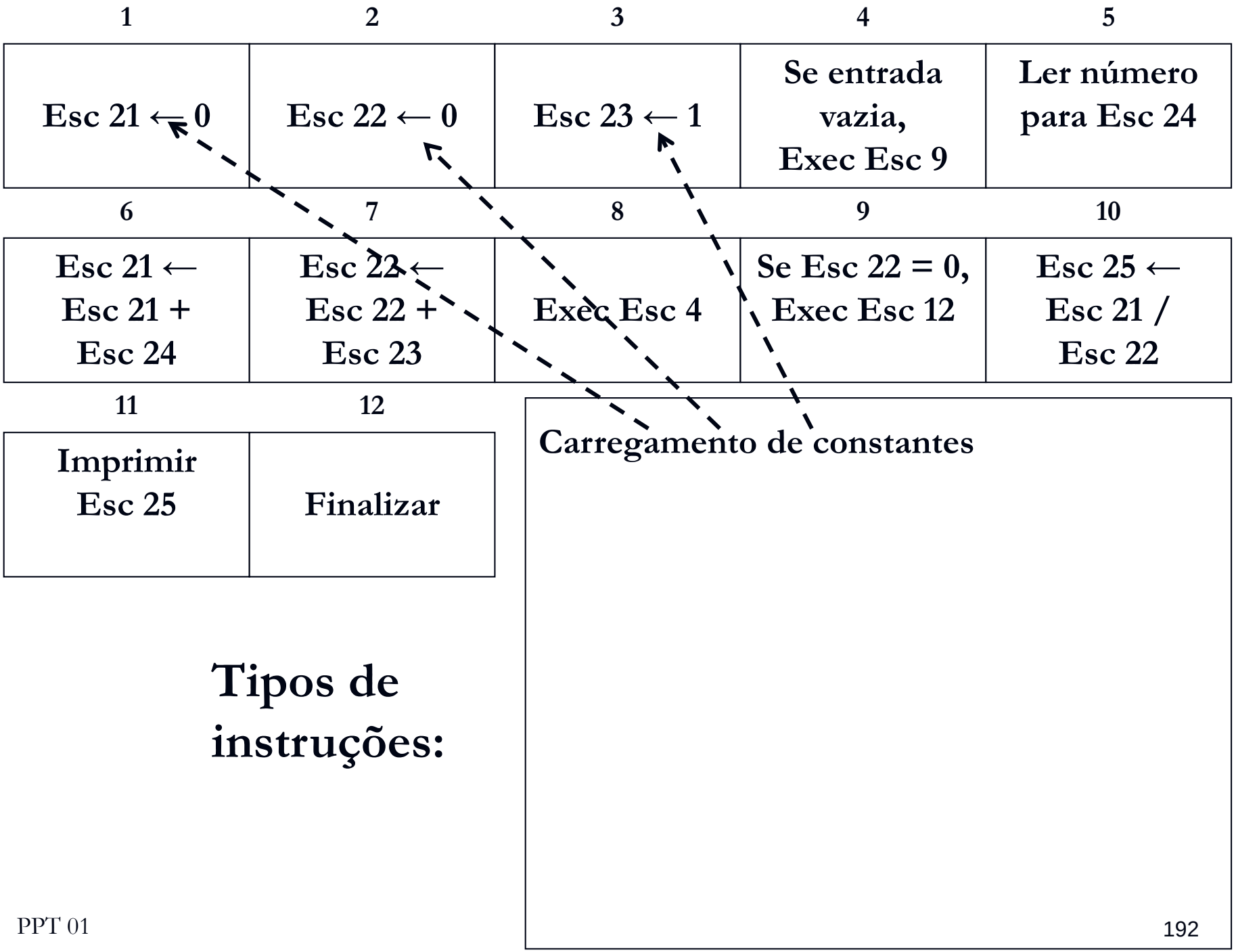
- Conteúdo dos escaninhos
- Para calcular a **média de outros números**: trocar os cartões de entrada
- Para **rodar outro programa**: trocar as instruções dos escaninhos

1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

11	12
Imprimir Esc 25	Finalizar

Tipos de instruções:

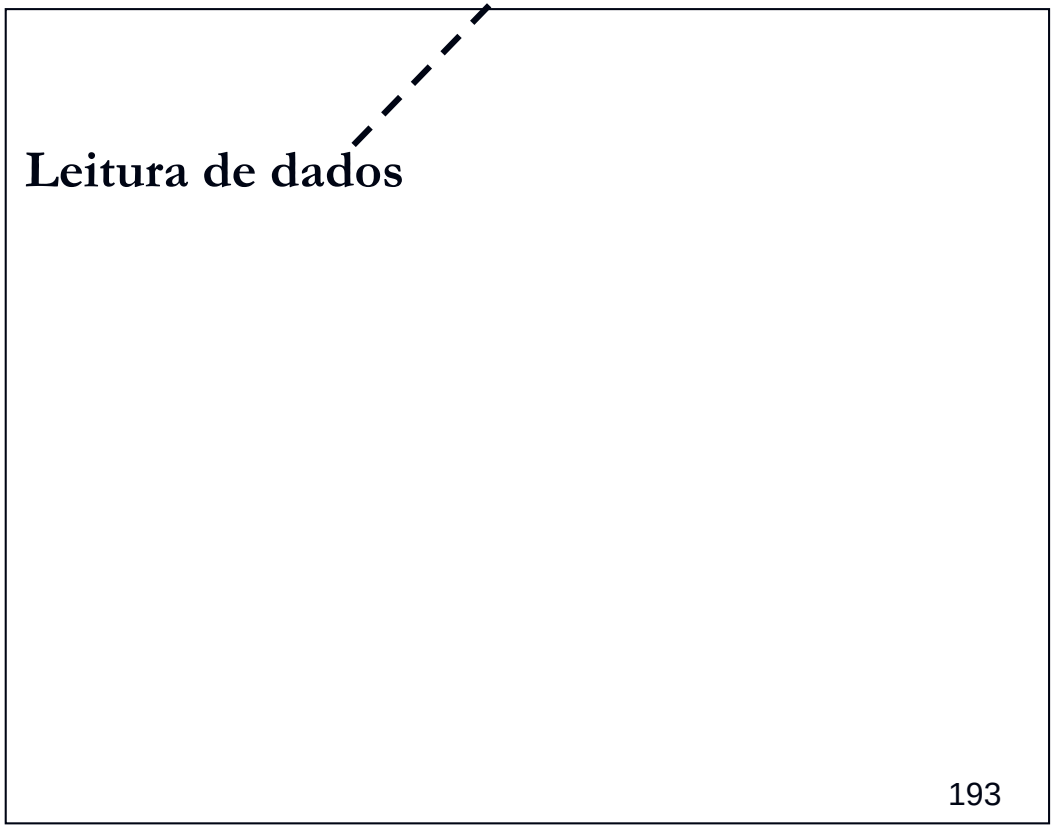


Tipos de instruções:

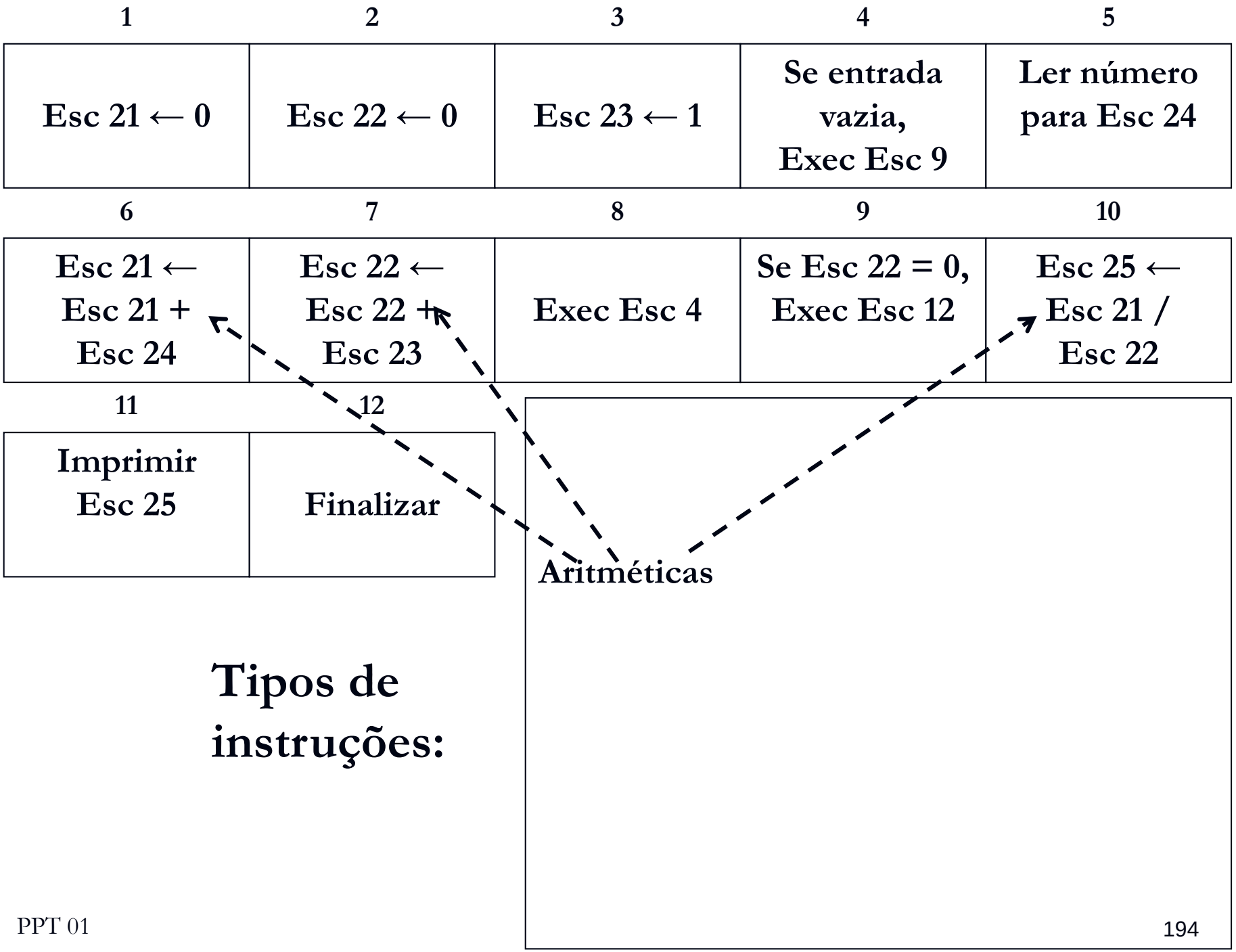
1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

11	12
Imprimir Esc 25	Finalizar



Tipos de instruções:



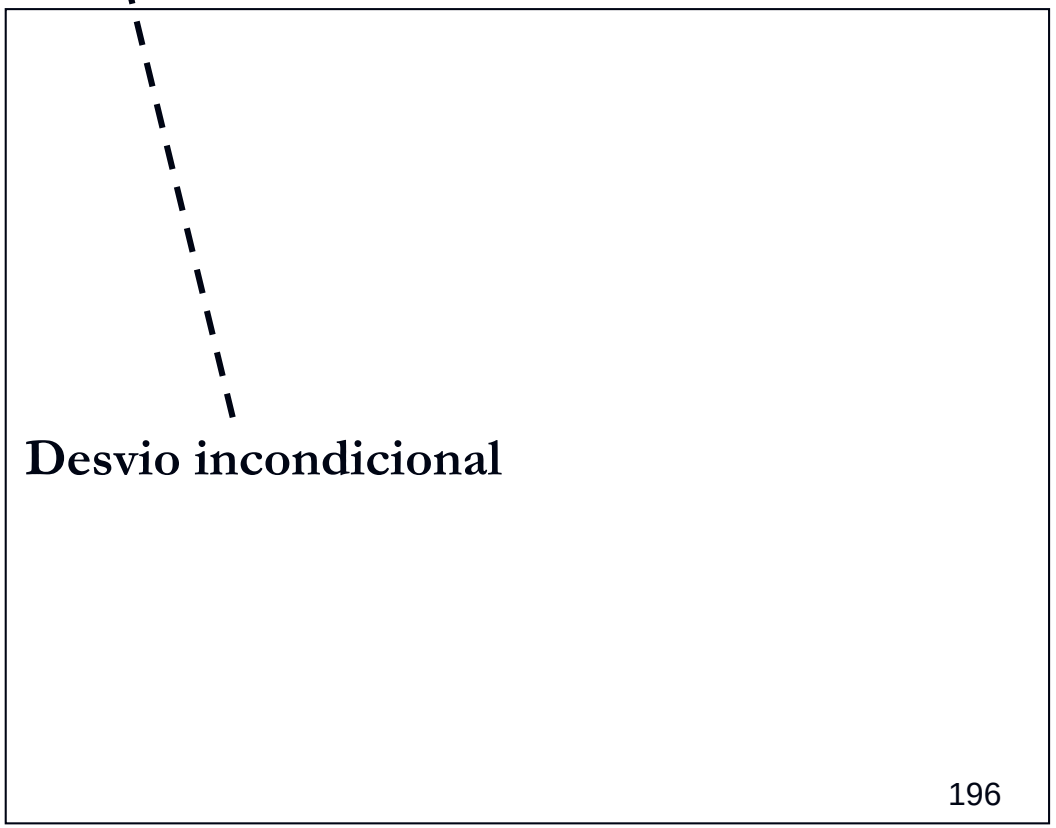
Tipos de instruções:

1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24
6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22
11	12			
Imprimir Esc 25	Finalizar			

Emissão de resultados

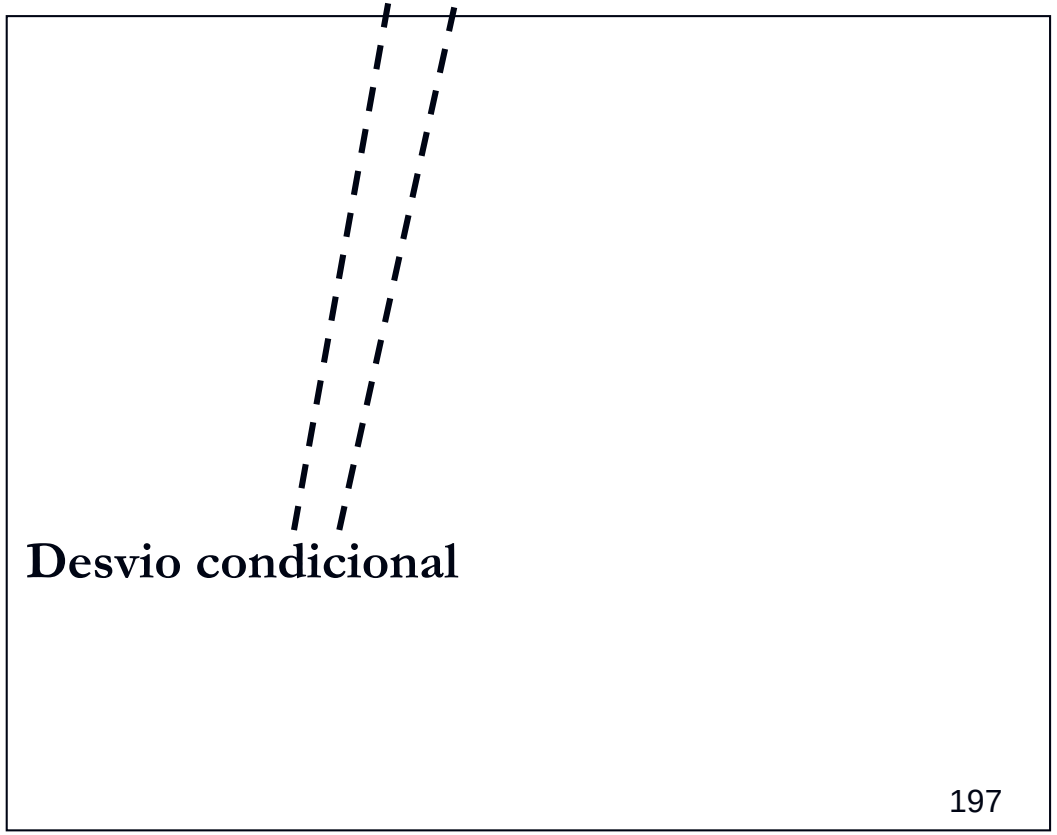
Tipos de instruções:

1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24
6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22
11	12			
Imprimir Esc 25	Finalizar			



Tipos de instruções:

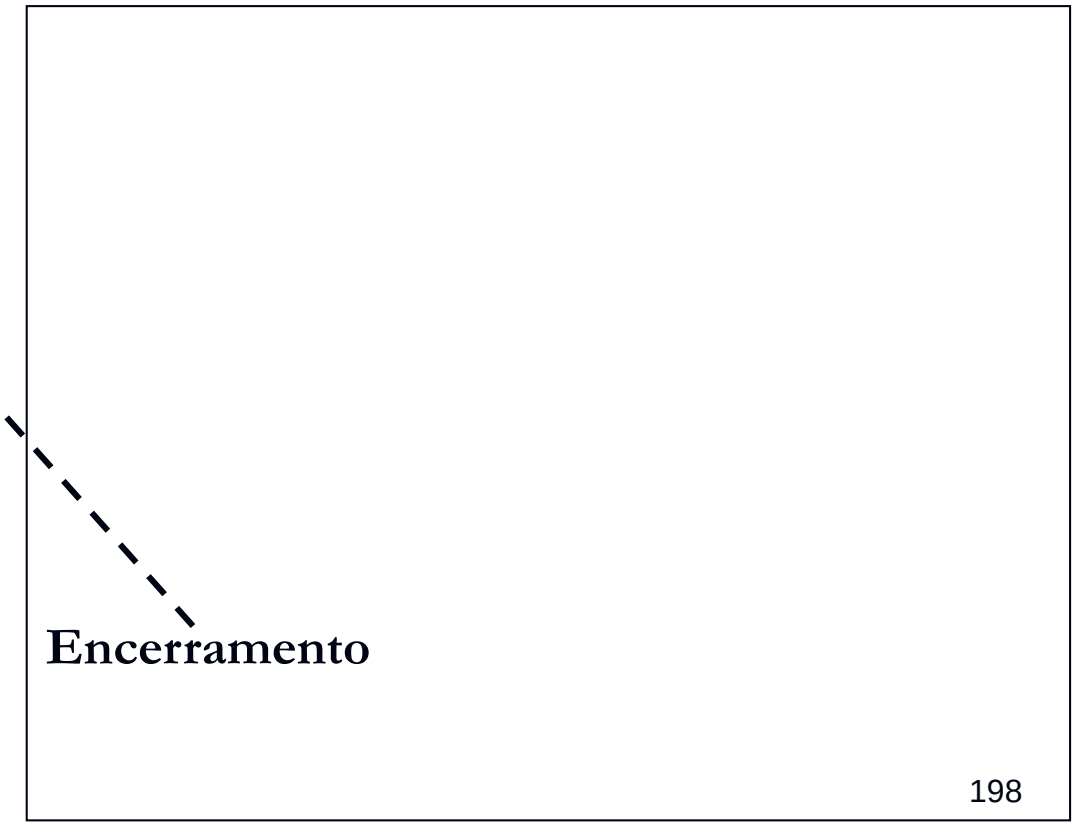
1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24
6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22
11	12			
Imprimir Esc 25	Finalizar			



Tipos de instruções:

1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24
6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22
11	12			
Imprimir Esc 25	Finalizar			

Tipos de instruções:



1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

11	12
Imprimir Esc 25	Finalizar

Tipos de instruções:

Cópia de conteúdo: Esc i ← Esc j

1	2	3	4	5
Esc 21 ← 0	Esc 22 ← 0	Esc 23 ← 1	Se entrada vazia, Exec Esc 9	Ler número para Esc 24

6	7	8	9	10
Esc 21 ← Esc 21 + Esc 24	Esc 22 ← Esc 22 + Esc 23	Exec Esc 4	Se Esc 22 = 0, Exec Esc 12	Esc 25 ← Esc 21 / Esc 22

11	12
Imprimir Esc 25	Finalizar

Tipos de instruções:

- Carregamento de constantes
- Leitura de dados
- Aritméticas
- Emissão de resultados
- Desvio incondicional
- Desvio condicional
- Encerramento
- Cópia de conteúdo: Esc i ← Esc j

Exercícios 1.2:

Escrever, nos mesmos moldes das instruções apresentadas anteriormente, um programa para:

1) Calcular e imprimir o fatorial de um número fornecido em cartão de entrada

2) Calcular e imprimir a soma dos termos de uma PA para a qual o primeiro termo, o número de termos e a razão devem ser fornecidos em cartões de entrada (sem usar a fórmula da soma)

3) Calcular e imprimir o MDC de dois números fornecidos em cartões de entrada

1.3 – Unidades básicas de um computador

PPT 01

Analogia	
Modelo hipotético	**Computador primitivo**
Operador	Unidade de controle
Calculadora	Unidade funcional (ALU)
Escaninhos	Memória
Cartões de entrada	Unidade de entrada
Máquina de escrever	Unidade de saída

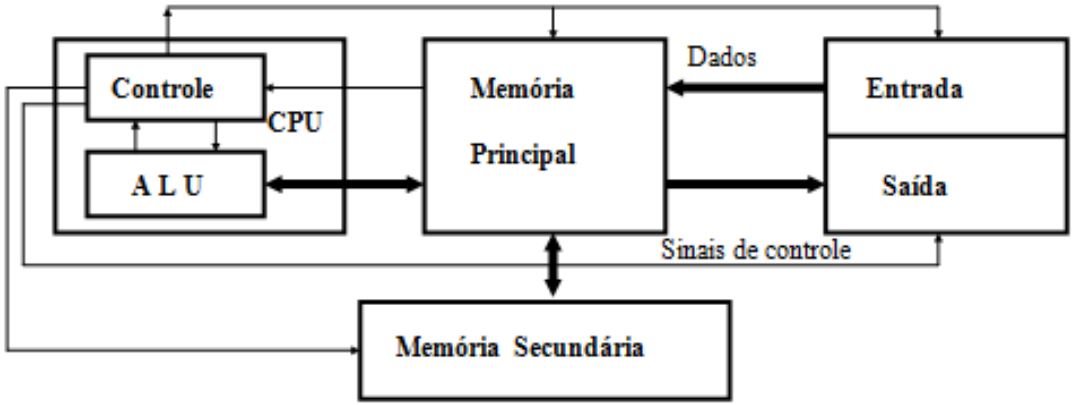
Modelo hipotético

Livro 01



Computador primitivo

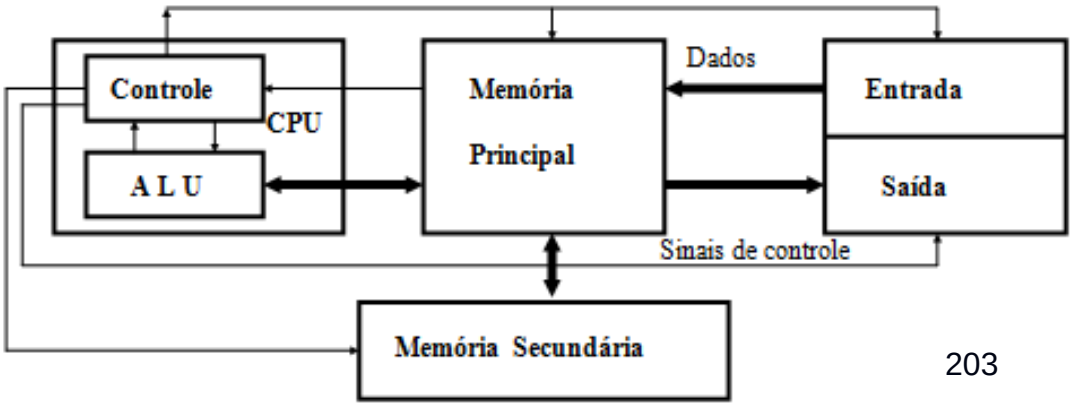
PPT 01



CPU – Unidade Central de Processamento:

- É o conjunto formado pela **unidade de controle** e pela **unidade funcional** (antiga ALU – unidade lógica e aritmética)

- Ali as instruções são encaminhadas para execução e os dados são intensamente usados na produção de resultados intermediários e finais



Funcionamento análogo ao do modelo hipotético:

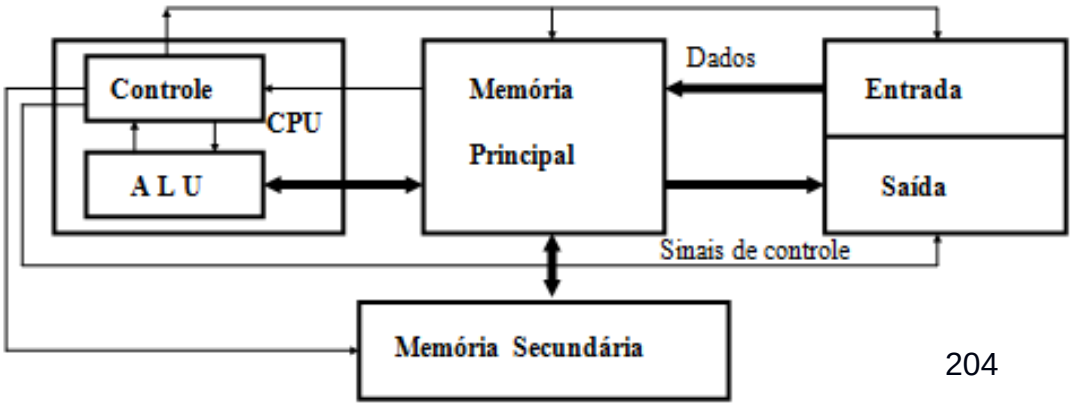
- A memória é constituída de vários compartimentos numerados, denominados **palavras**

- Ali são colocadas:

  - Instruções de programas a serem executados
  - Resultados de cálculos intermediários e finais

PPT 01

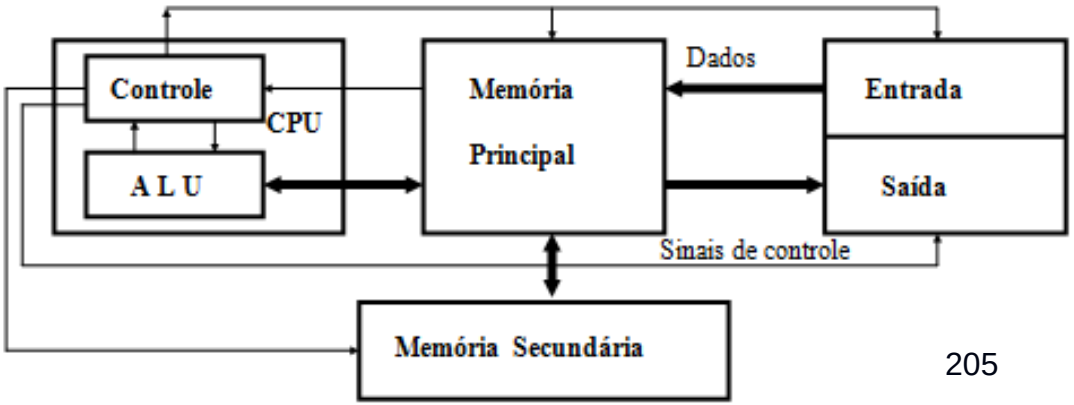
O número que identifica cada palavra é seu **endereço**



Funcionamento análogo ao do modelo hipotético:

- A **unidade de controle** lê e interpreta a instrução de cada uma dessas palavras
- Ela seleciona e aciona o **dispositivo** que irá executá-la
- Se **não** for uma instrução de **desvio**, a próxima a ser lida e executada será a **subsequente**

PPT 01



Memória principal:

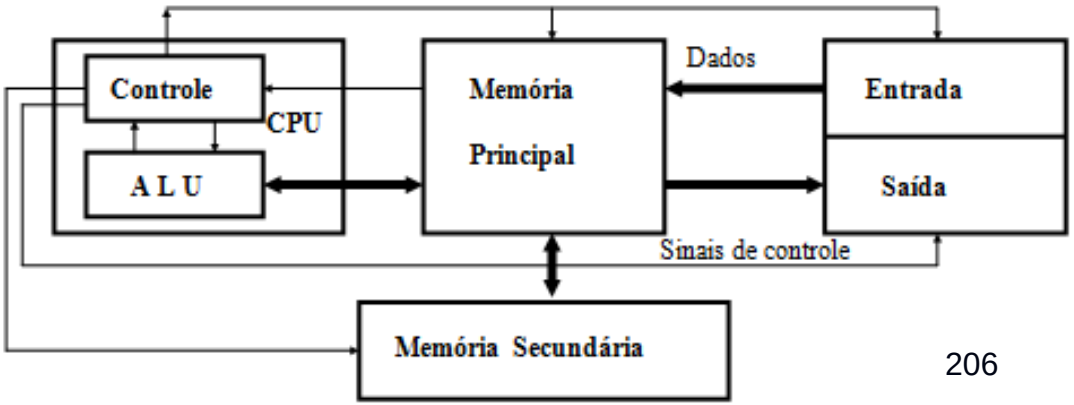
- Armazena os programas em processamento pela CPU, num dado momento

Memória secundária:

- Armazena todos os programas e toda diversidade de informações residentes no computador, mesmo com ele desligado

PPT 01

Exemplos: sistemas bancários, bibliotecas, estoques industriais e comerciais, etc.



Por que não guardar tudo na memória principal:

- A atual **tecnologia** de memória principal a torna **volátil**, ou seja, toda a informação é perdida ao se **desligar o computador**

- Isso não ocorre com a tecnologia de **memória secundária**

Por que não guardar tudo na memória principal:

- Memória principal se comunica **intensamente** com a CPU, necessitando ser de **rápido acesso**; isso exige tecnologia **sofisticada**, o que a torna **cara**

- Para guardar tudo ali, seu volume seria **imenso**, encarecendo **desnecessariamente** o computador

- A grande maioria dessas informações é usada **esporadicamente**, podendo ser armazenada num meio de **acesso mais lento**, porém bem mais **barato**

- Hoje a **memória principal** é referenciada como **RAM** (random access memory – acesso aleatório)

- **Acesso aleatório:** o tempo de acesso a qualquer de suas palavras é constante e igual ao de todas as outras

- A **memória secundária** é referenciada hoje como **HD** (hard disk – disco rígido), interno e fixo na torre dos desk-tops ou na base dos lap-tops

- Também podem ser considerados partes da memória secundária elementos portáteis como **CD's**, **DVD's**, **pen-drives**, **disquetes** (antigos) e **HD's externos**

Unidades de entrada:

- Responsáveis por levar informações **de fora para dentro** dos computadores

- As mais conhecidas: **teclado** e **mouse**: suas ações geralmente são visualizadas na tela do monitor

- **Leitora de cartões perfurados:** muito usada nas décadas de 1950 e 1960

Unidades de entrada:

- Também podem ser consideradas unidades de entrada: HD interno e externo, drive de CD's/DVD's, porta USB, drive de disquetes (antigo), unidade de fita magnética

  - Programas podem ler informações de arquivos acessados por esses dispositivos

- Dispositivos para interfaceamento digital: leitora óptica, célula fotosensora, microfone, câmera, joystick, scanner, sensores para controle de processos industriais

Unidades de saída:

- Responsáveis por levar informações **de dentro para fora** dos computadores

- As mais conhecidas: **vídeo**, **impressoras** (laser, jato de tinta e impacto – essas últimas já não tão usadas), **auto-falantes** e **plotters** (traçadores de gráficos)

- Há também **impressoras multi-funcionais**: imprimem, tiram cópias, digitalizam, revelam fotos

- **Impressoras de papel contínuo:** muito usadas nas décadas de 1950 e 1960

Unidades de saída:

- Também podem ser consideradas unidades de saída: HD interno e externo, drive de CD's/DVD's, porta USB, drive de disquetes (antigo), unidade de fita magnética
  - Programas podem escrever informações em arquivos acessados por esses dispositivos
- Equipamentos industriais controlados por computador: prensas, tanques para banhos químicos, robôs, etc.

Unidades funcionais:

- **Unidade funcional:** confere ao computador a capacidade de realizar operações matemáticas
- **Operandos** para essas operações são obtidos do sistema de **memória**
- **Resultados** também são ali guardados
- Cada **operação** é realizada por um **circuito específico**

- Operações muito comuns: as quatro **operações aritméticas**

- Outras muito comuns: **operações lógicas** (definidas mais adiante), cujos resultados só podem assumir os valores **0** (zero) e **1** (um)

- Operações matemáticas mais **complexas**: trigonométricas, logarítmicas, exponenciais, etc

- Nos computadores primitivos, tais circuitos integravam uma única unidade: **unidade lógica e aritmética** (ALU)

- Uma ALU podia executar num dado instante **apenas uma** dessas operações

- Hoje os circuitos funcionais podem trabalhar **em paralelo**, permitindo a **execução simultânea** de várias dessas operações

- **Supercomputadores** possuem **várias unidades** para uma **mesma operação**: várias somas, várias multiplicações, etc., simultâneas

Unidade de controle:

- **Lê** e **interpreta** cada instrução do programa em execução
- **Aciona a unidade** do computador que **executará** a instrução
- Obtém os eventuais **operandos** da instrução
- Faz o **sequenciamento** das instruções de um programa
- **Instruções de desvio** podem quebrar a **sequência linear** das instruções de um programa

Unidade central de processamento (CPU): composta por

- Unidade de controle
- Unidades funcionais
- Conjunto de registradores de propósitos gerais
- Registradores de propósitos específicos

Conjunto de registradores de propósitos gerais:

- Pequeno módulo de memória, de **acesso muito mais rápido** que o acesso à memória principal (RAM)
- Cada **registrador** é análogo a uma **palavra** da RAM: possui seu **endereço** dentro do conjunto
- **Finalidade:** armazenar resultados intermediários de cálculos, evitando guardá-los na RAM

Conjunto de registradores de propósitos gerais:

- A **velocidade** dos circuitos da **CPU** é bem maior que a velocidade de acesso às palavras da **RAM**

- Sem esses registradores, o número de acessos à RAM seria muito **elevado**, deixando a **CPU ociosa** durante boa percentagem do tempo de execução dos programas

Registradores de propósitos específicos:

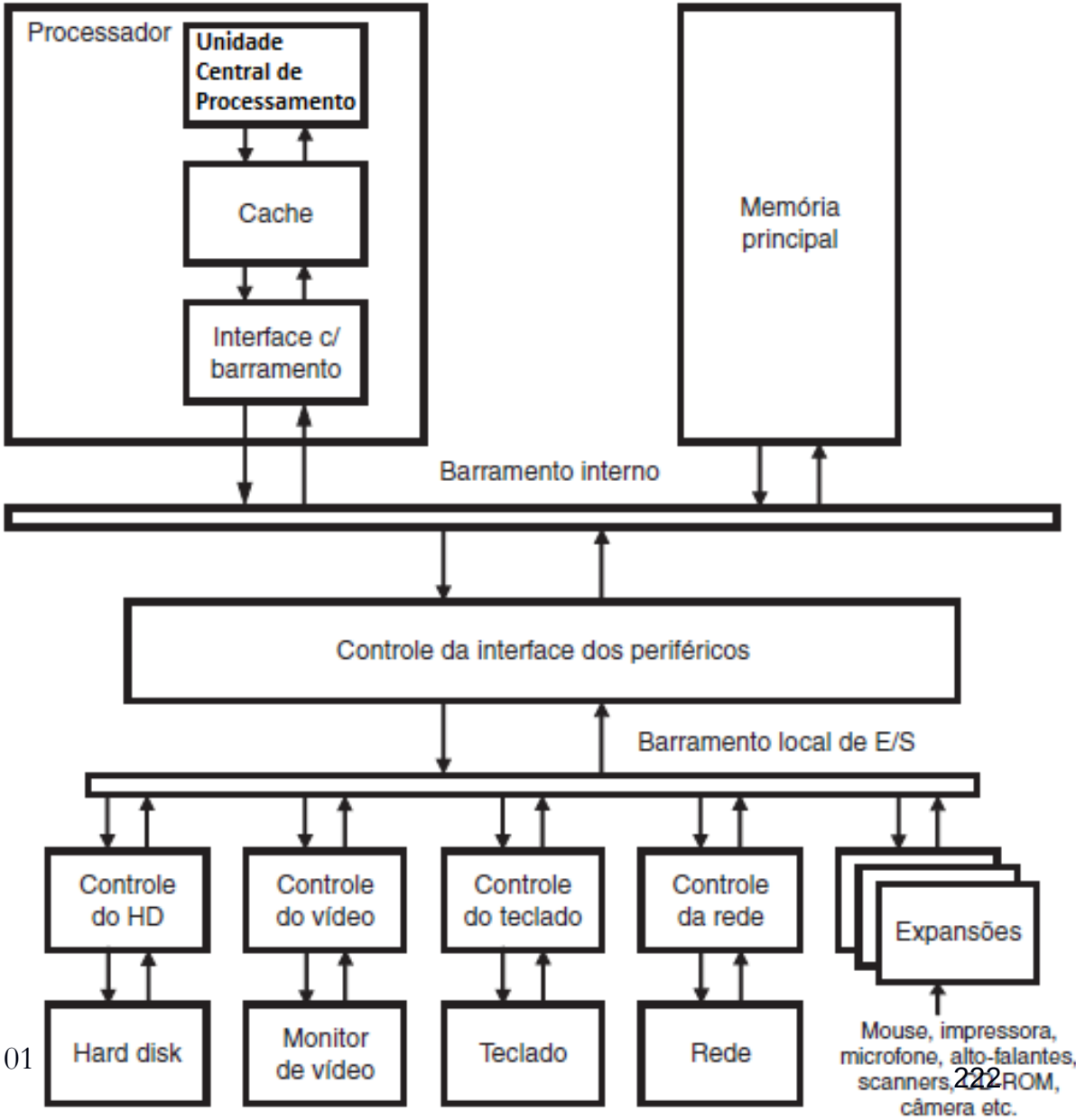
- De **acesso tão rápido** quanto os de propósitos gerais
- Cada um tem sua finalidade:
  - Guardar o endereço da palavra da memória que está sendo acessada
  - Guardar o conteúdo lido da RAM ou a ser nela escrito
  - Guardar o endereço da próxima instrução a ser executada
  - Guardar a instrução a ser interpretada
  - Etc.

Organização de um computador pessoal moderno:

Três componentes:

- **Processador**
- **Memória principal**
- **Sistema de entrada e saída (E/S)**

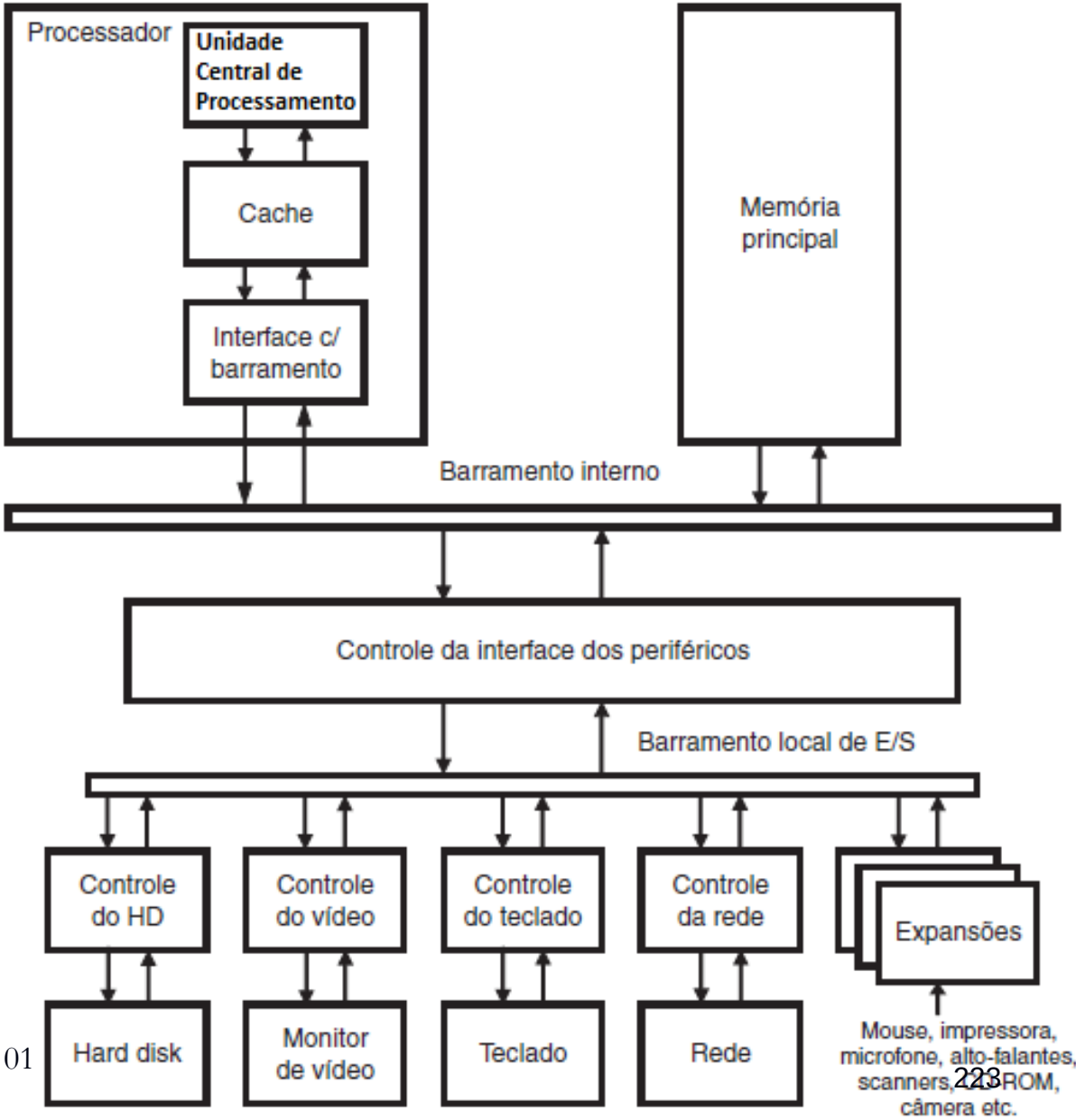
Ligados por um barramento interno



Organização de um computador pessoal moderno:

Processador composto por:

- **CPU**
- **Memória cache**
- **Interface com o barramento**



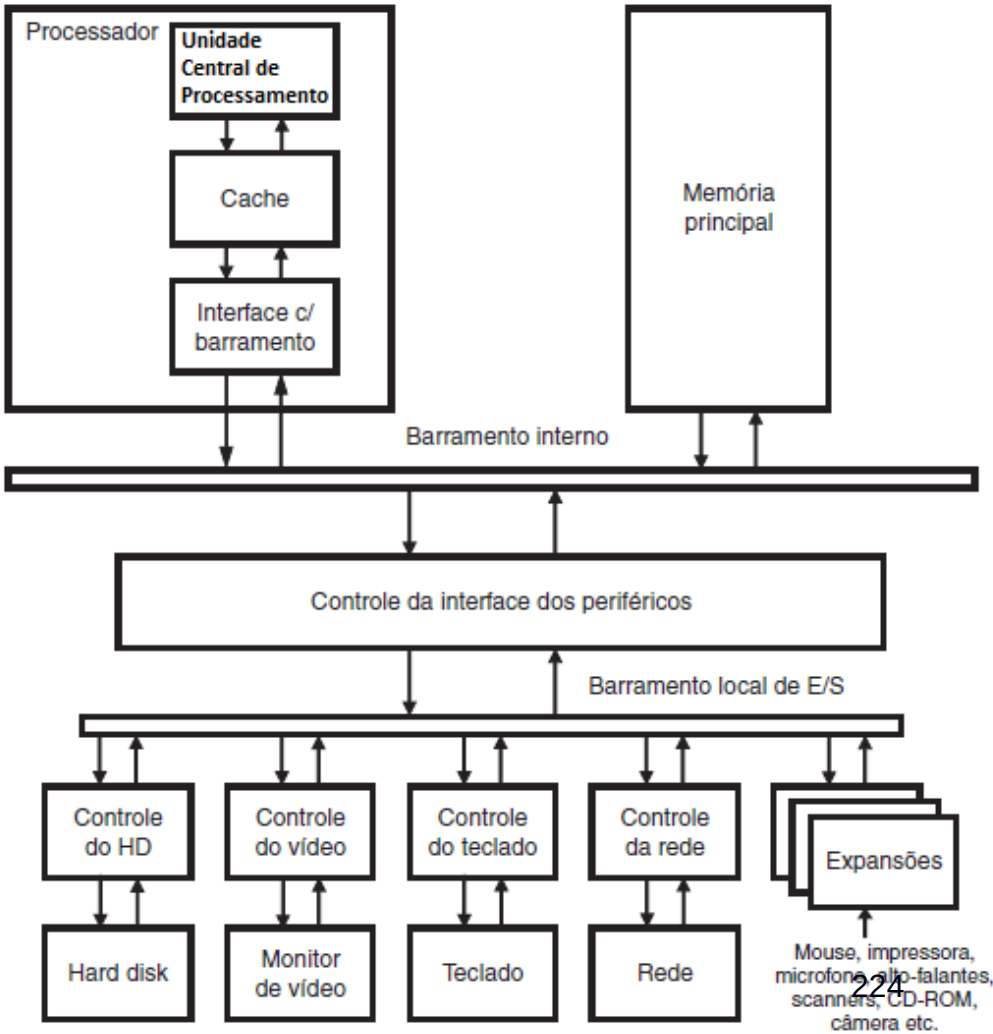
Organização de um computador pessoal moderno:

Memória cache:

De acesso mais rápido que o da RAM

Mais lento que o dos registradores

Guarda cópia das palavras da RAM mais intensamente acessadas, num passado bem recente



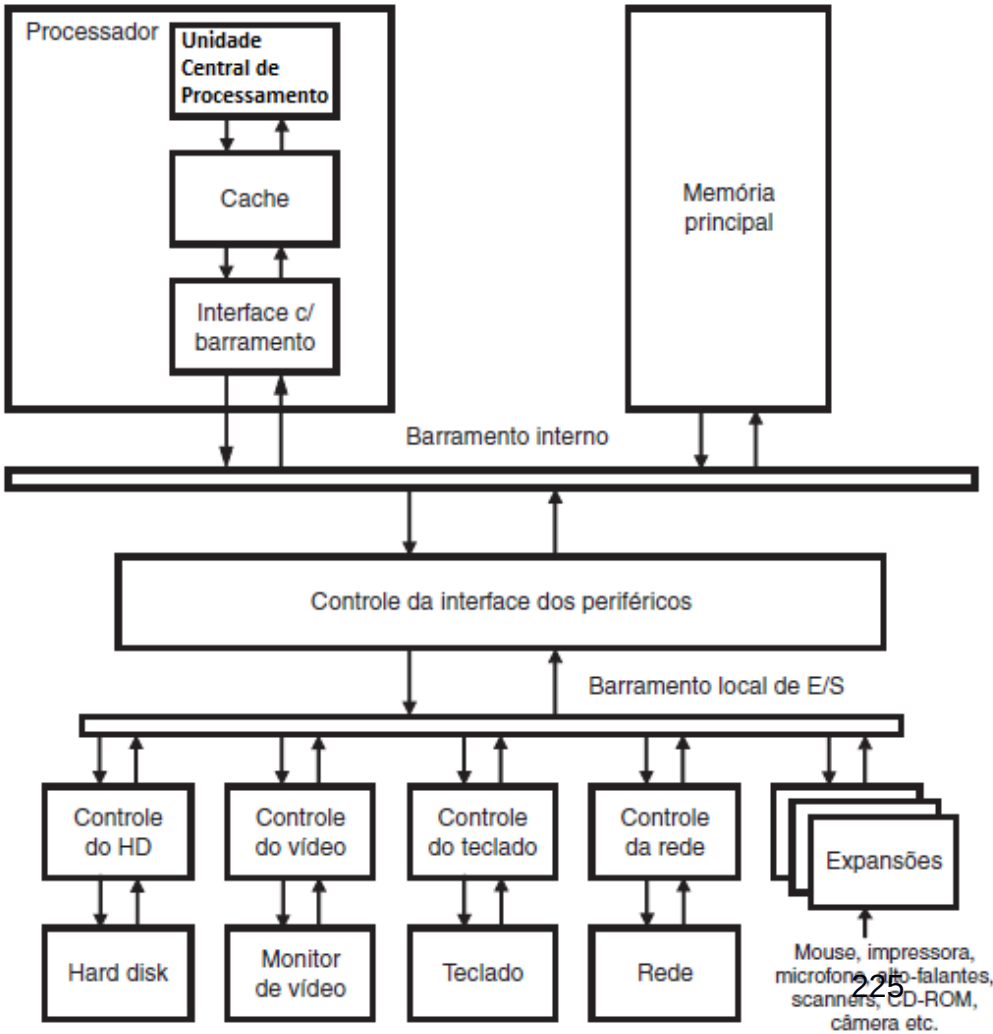
Organização de um computador pessoal moderno:

Memória cache:

Finalidade: tentar atender aos pedidos de acesso à RAM

Pode-se reduzir bem o tempo total de acesso à RAM

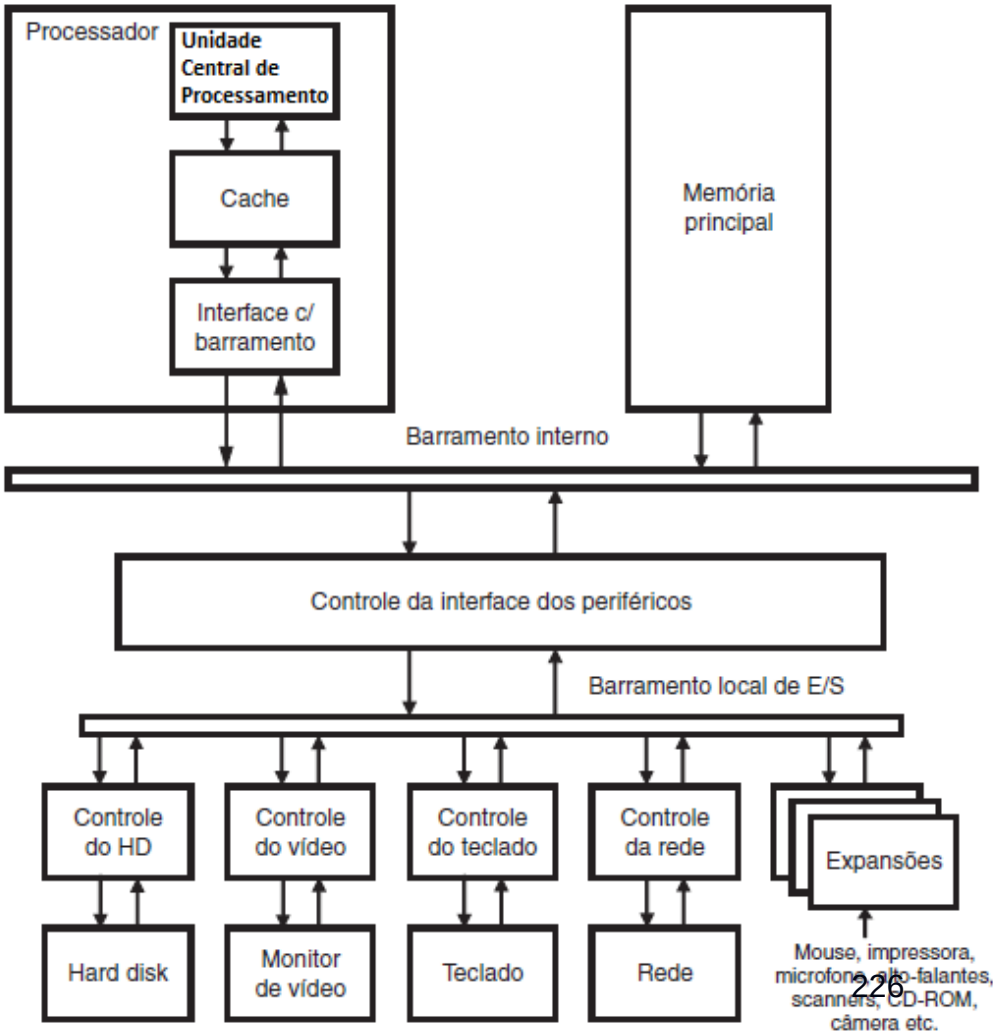
Melhora muito o desempenho do computador



Organização de um computador pessoal moderno:

Sistema de memória composto hierarquicamente por:

- **Registradores**
- **Memória cache**
- **RAM (memória principal)**
- **Memória secundária**

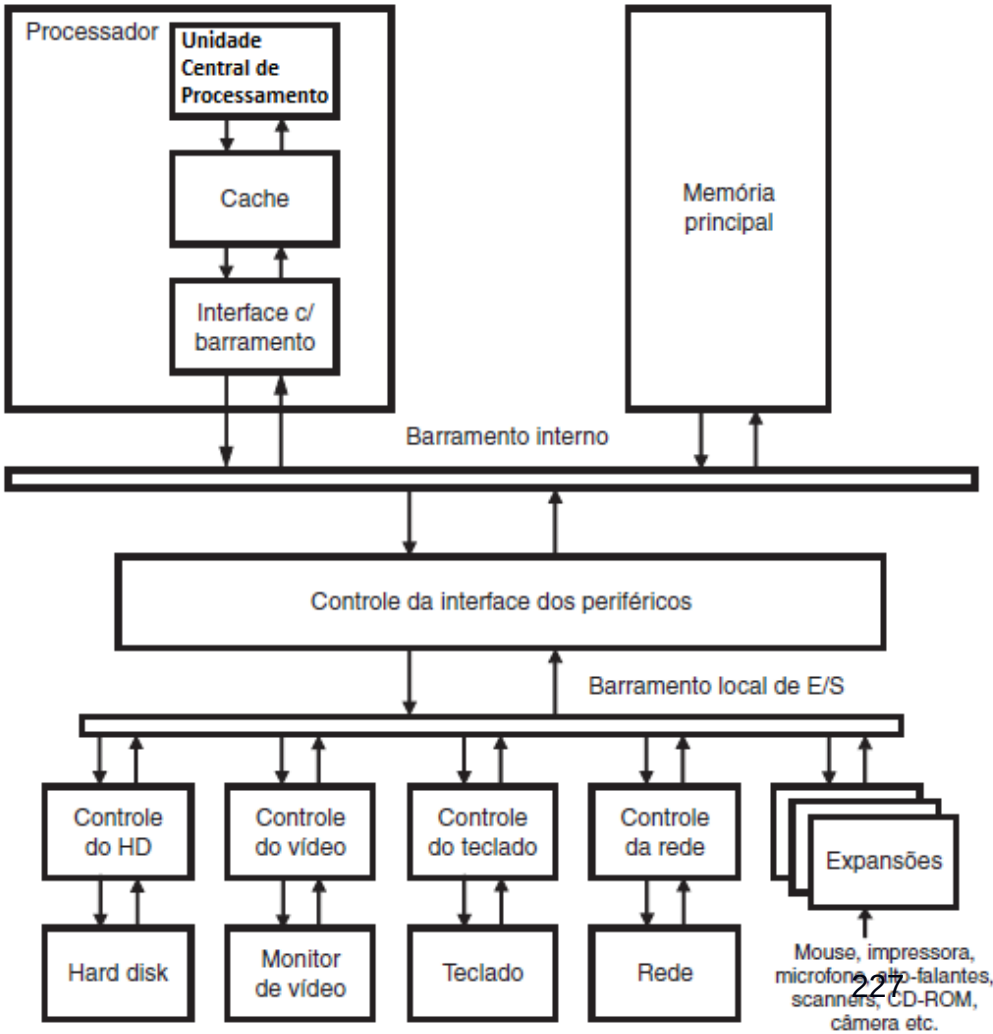


Organização de um computador pessoal moderno:

Periféricos:

Todos os equipamentos do sistema de E/S

HD, vídeo, teclado, placa de rede, mouse, impressora, microfone, auto-falantes, scanner, drive de CD/DVD, câmera, etc.



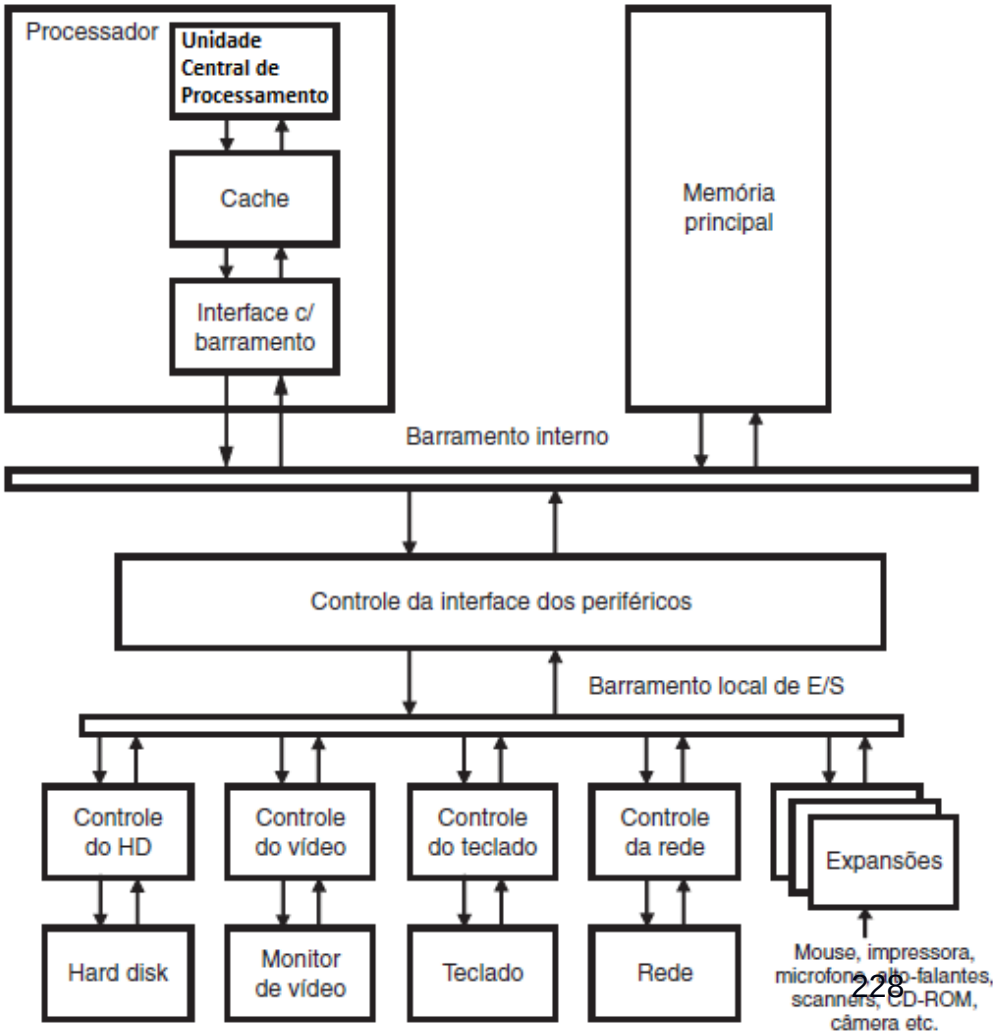
Computadores com mais de um processador:

Processador i5 da Intel:

2 e 4 processadores equivalentes ao da figura ao lado

Processador i7 da Intel:

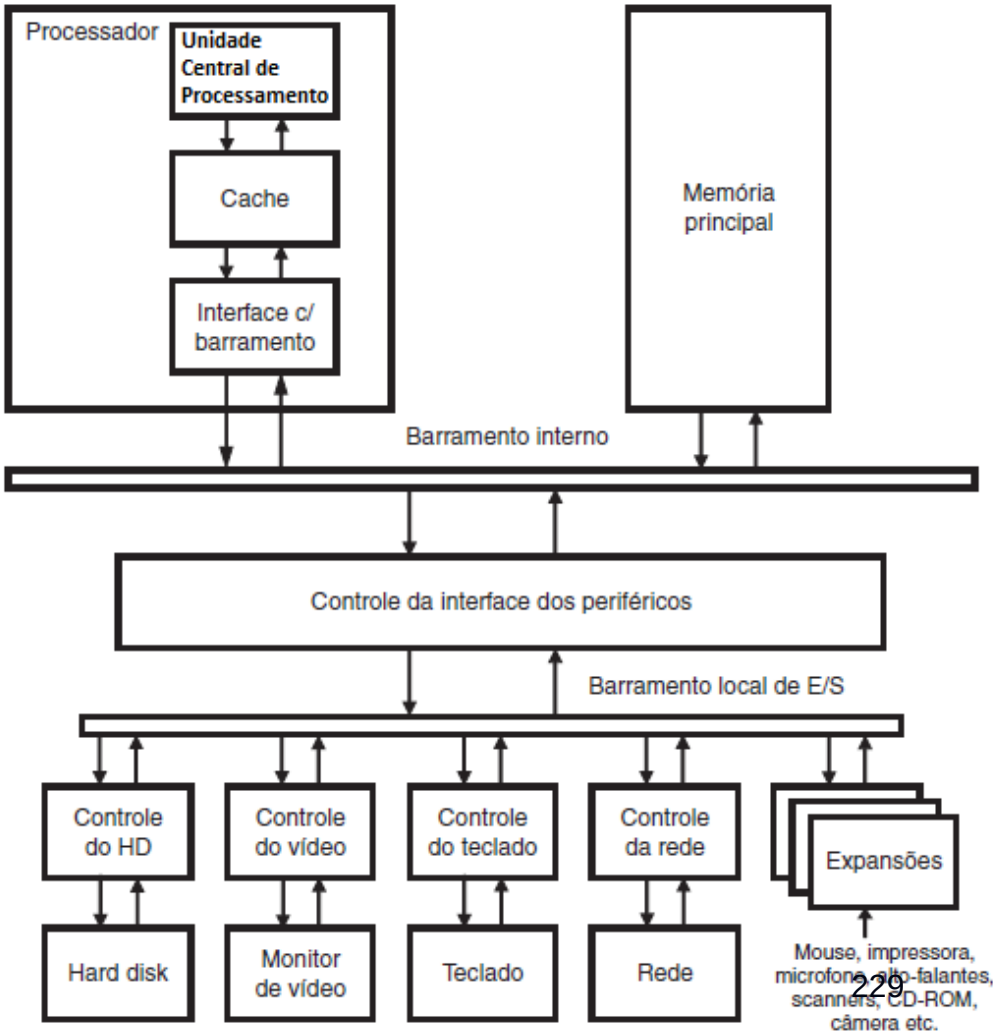
4 e 6 processadores equivalentes ao da figura ao lado



Computadores com mais de um processador:

Supercomputadores:

- Milhares de processadores
- Milhares de módulos de memória
- Multiprocessamento: execução simultânea de vários programas
- Processamento paralelo: execução simultânea de vários trechos de um mesmo programa



Computadores, Algoritmos e Programas: Manual Introdutório de Ensino

Capítulo II

Informações Manipuladas por um Computador

Capítulo II – Informações Manipuladas por um Computador

2.1 – Natureza das informações

2.2 – Sistemas de numeração e mudança de base

2.3 – Aritmética binária e operações lógicas

2.4 – Representação de números inteiros

2.5 – Representação de números reais

2.6 – Representação de textos

2.7 – Representação de imagens

2.8 – Representação de instruções

2.1 – Natureza das Informações

- Um computador é capaz de manipular
  - Números
  - Textos
  - Sons
  - Imagens
  - Composições desses ingredientes

- **Molécula:**
  - **Menor** porção de uma **substância pura**
  - **Porção qualquer** de uma substância pura: **conjunto de moléculas**
- **Bit:**
  - **Menor** unidade de **informação** em um computador
  - **Informação qualquer** na memória de um computador: **conjunto de bits**

- **Bit (bi**-nary digi-**t):** unidade de armazenamento que pode assumir um de apenas dois valores possíveis: o **0 (zero)** e o **1 (um)**

- Uma **informação** qualquer na memória de um computador é uma composição de **0's** e **1's**

- **Números** em um computador são guardados no **sistema binário de numeração**, em vez do **sistema decimal**, usado comumente pelos seres humanos

- **Dispositivos físicos para armazenar um bit:** devem ser capazes de estar em apenas um de dois estados possíveis:
  - O primeiro representando o valor **0 (zero)**
  - O segundo representando o valor **1 (um)**
- **Exemplo:** recipiente capaz de conter apenas uma **gota** de líquido
  - Se ele estiver **vazio**: **estado 0**
  - Se, com um conta-gotas, for-lhe **introduzida uma gota**, ele passa para o **estado 1**

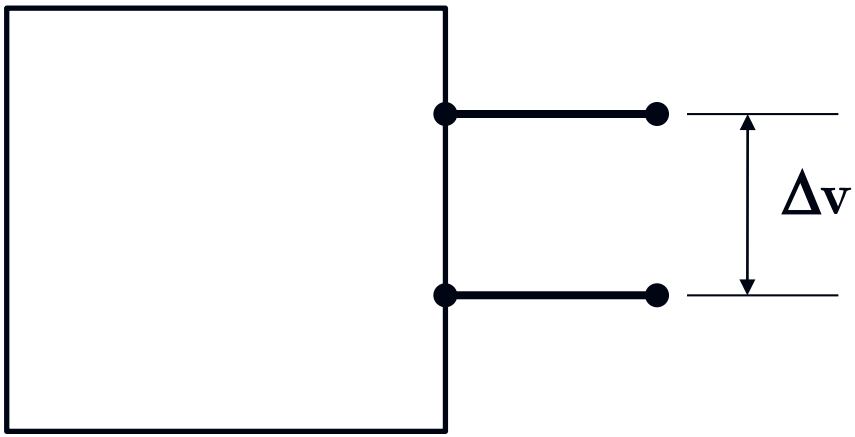
- Há **três diferentes meios físicos** para armazenar um **bit** em um computador:

  - **Meio eletrônico –** usado principalmente dentro do **processador** e da **memória principal**, mas também em **pen-drives**

  - **Meio magnético –** usado em **HD's** e **fitas magnéticas**

  - **Meio óptico –** usado em **CD's** e **DVD's**

Meio eletrônico: dispositivo de nome **flip-flop**



Três valores de tensão:

Zero: desligado

Baixo: estado 0

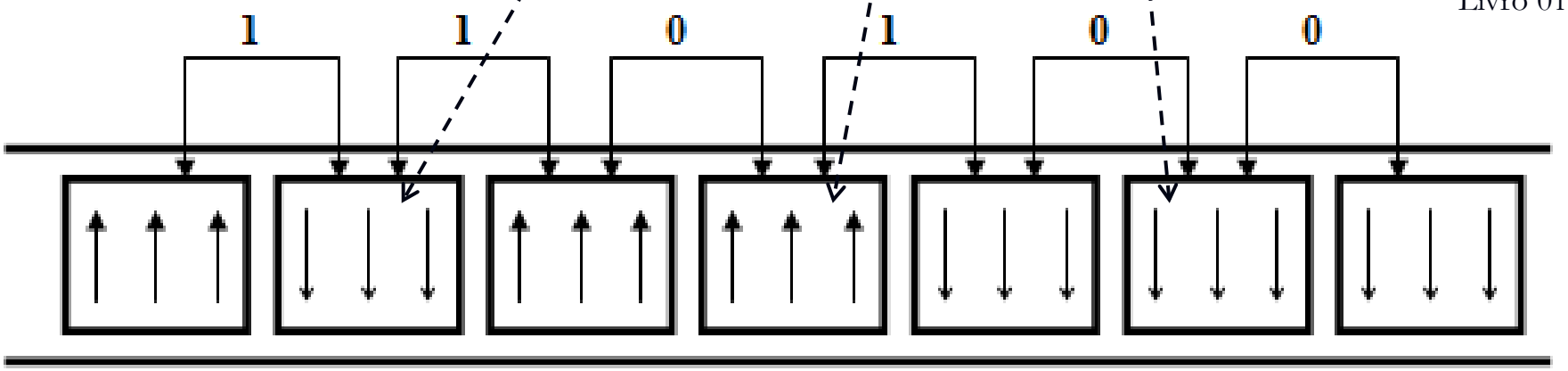
Alto: estado 1

Na realidade, um flip-flop tem outros terminais não abordados em CES-10

Meio magnético: trilhas de **micro-magnetos**

- Cada magneto é magnetizado em um de **dois** possíveis **sentidos**
- Observar dois **magnetos adjacentes**: se iguais, o bit é **0**, se diferentes, o bit é **1**

Livro 01



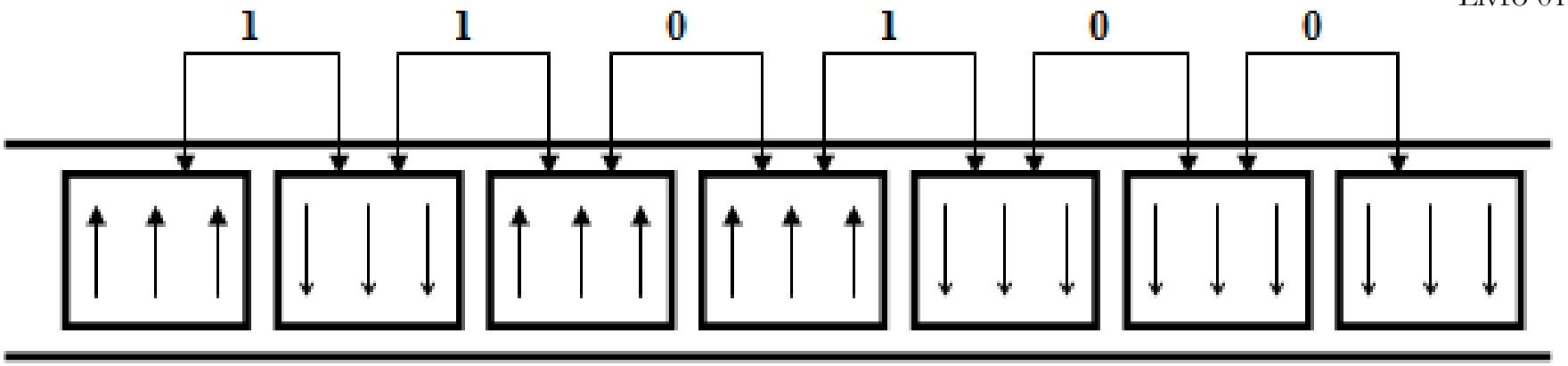
Trilha magnética de bits

Meio magnético: trilhas de **micro-magnetos**

- Na figura, os bits armazenados são **110100**

- A leitura da trilha é feita por uma cabeça detectora de **alteração do sentido** de magnetização dos magnetos

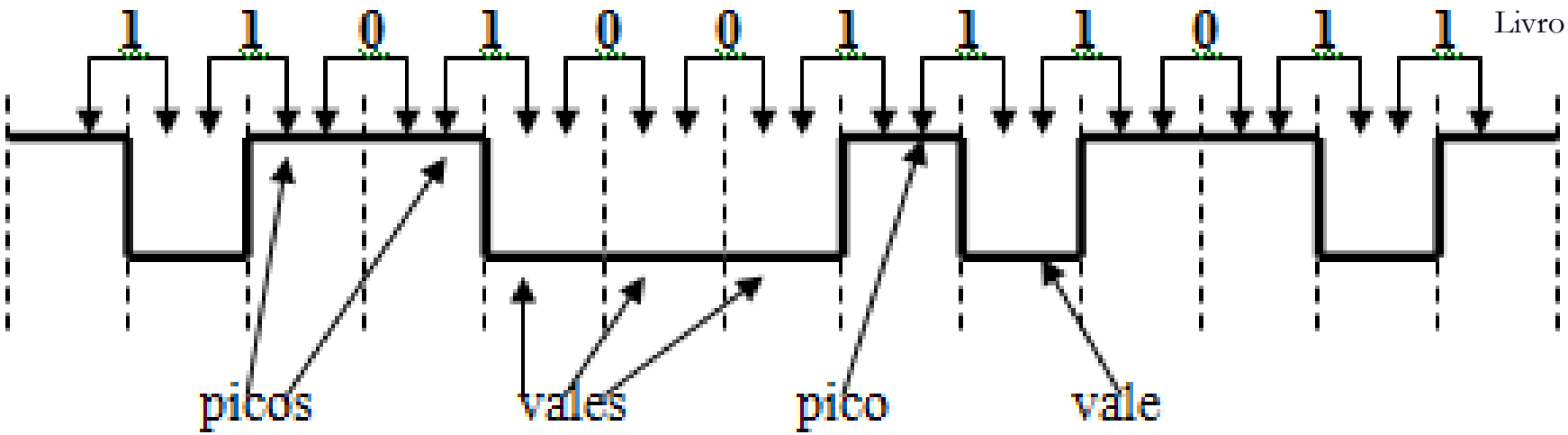
Livro 01



Trilha magnética de bits

Meio óptico: trilhas numa **superfície espelhada**

- Cada trilha contém sequências de minúsculos **picos** e **vales**
- **Bit 1:** alteração de nível; **Bit 0:** manutenção de nível

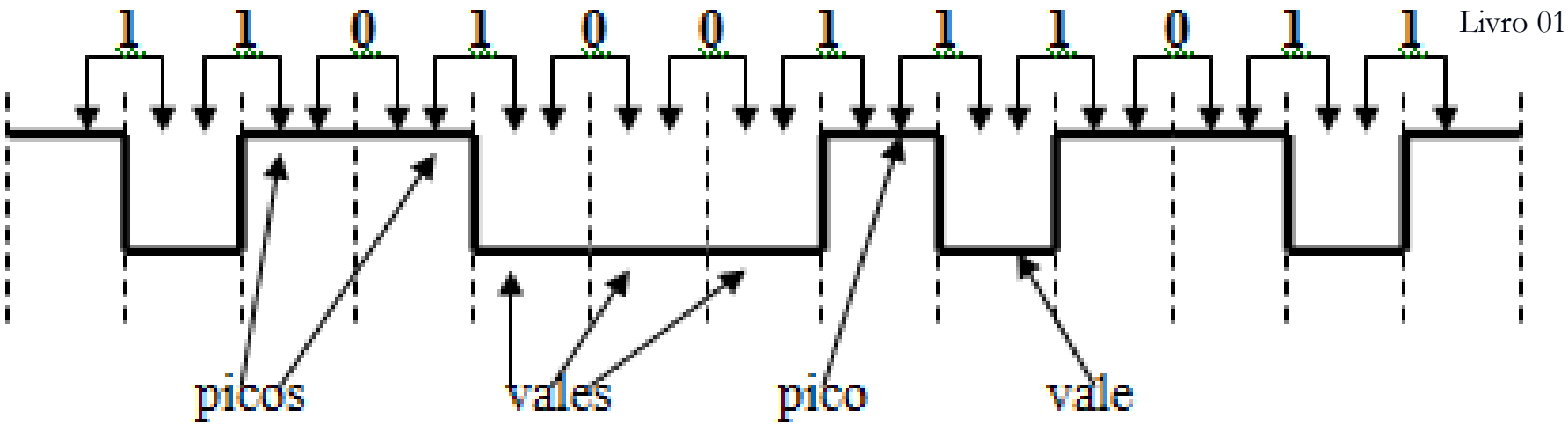


Livro 01

Trilha de um dispositivo óptico digital

Meio óptico: trilhas numa **superfície espelhada**

- A leitura da trilha é feita pela emissão de **raio laser** sobre ela
- O **ângulo** de reflexão do raio é **diferente para picos e vales**; isso é captado pelo receptor do reflexo

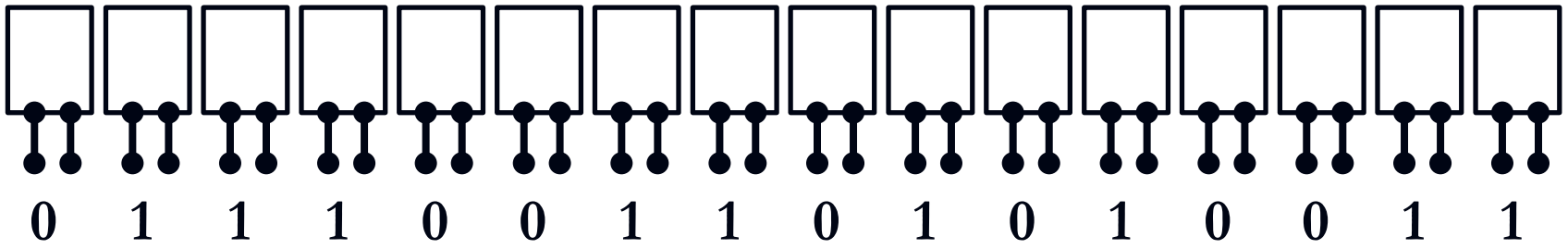


Trilha de um dispositivo óptico digital

Palavra:

- Na RAM, na cache e nos registradores da CPU, é uma **concatenação de flip-flops**

PPT 01



Palavra de 16 bits

- Num instante qualquer da execução, ela pode estar guardando um conjunto qualquer de **0's** e **1's**

Byte: conjunto de 8 bits



- **Importância:** cada **caractere** que compõe um texto é armazenado em **um byte**
- **Caracteres:** letras, dígitos decimais, operadores aritméticos, pontuação, etc.
- **Exemplo:** o byte acima representa a letra **'s'** (minúscula)

Byte: conjunto de 8 bits



- O **tamanho da palavra** de um computador pode ser expresso em número de **bits** ou de **bytes**

- A **palavra** de um módulo de memória pode ser um conjunto de **1, 2, 4, 8 ou 16 bytes adjacentes** desse módulo

- Os computadores atuais podem **variar o tamanho** de suas palavras mediante programação

- Ora ele pode enxergá-las como tendo apenas **1 byte**, ora como tendo **2**, **4**, **8** ou **16 bytes**

- Então o sistema de **endereçamento** de seus módulos de memória **pode variar** mediante programação

- **Exemplo:** RAM de 2 megabytes

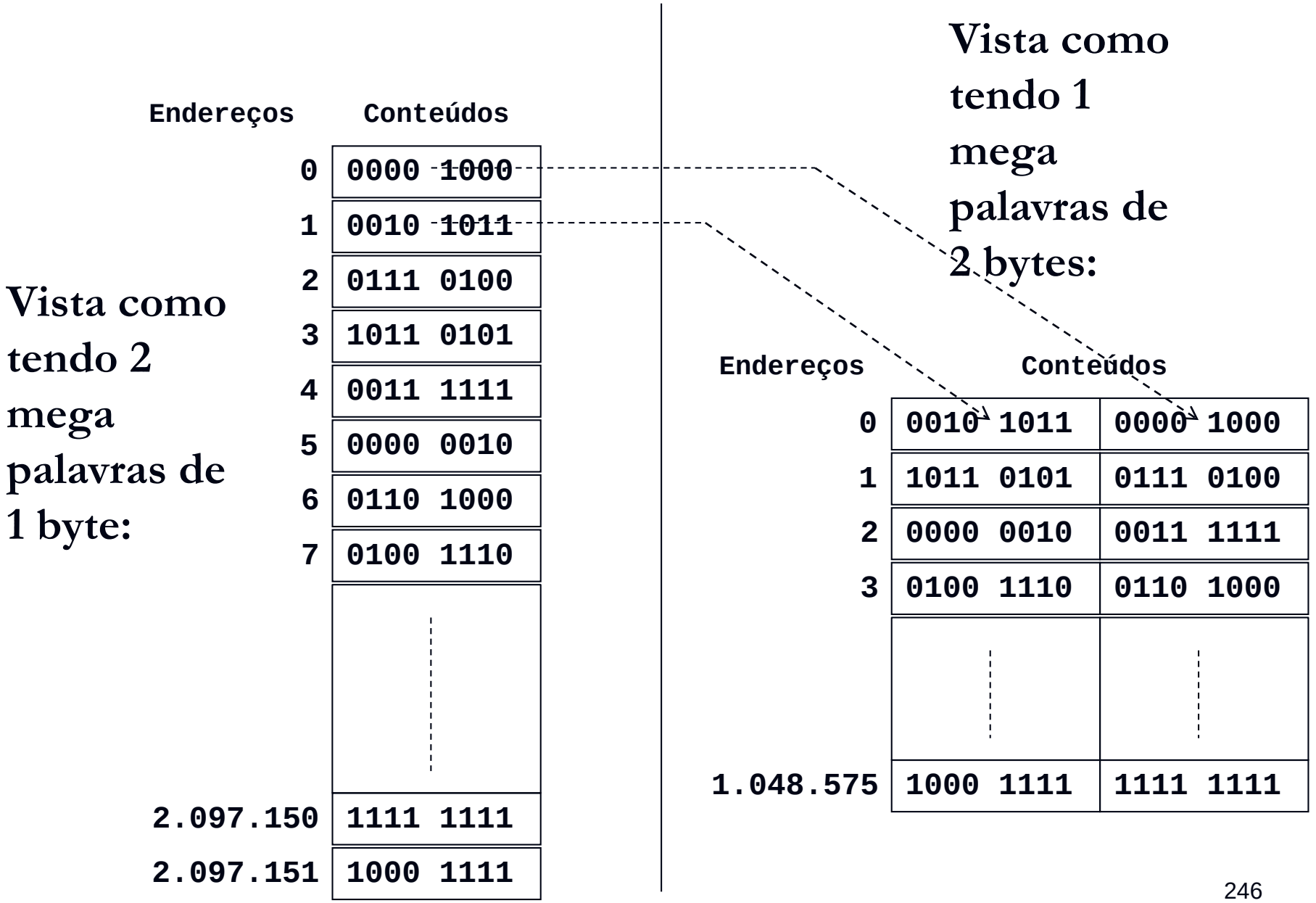


Tabela dos prefixos kilo, mega, giga, etc.:

Prefixos	Valores		
	Em potência de 2	Em decimal	Próximos de
Kilo	2^{10}	1.024	10^{3}
Mega	2^{20}	1.048.576	10^{6}
Giga	2^{30}	1.073.741.824	10^{9}
Tera	2^{40}	1.099.511.627.776	10^{12}
Peta	2^{50}	1.125.899.906.842.624	10^{15}

Observação:

- Sabe-se que toda informação na memória é um conjunto de **0's** e **1's**

- A priori, não se sabe se o **conteúdo** de um trecho de memória é um conjunto de **números**, ou um **texto**, ou uma **imagem**, ou um **som** digitalizado

- O **programa** que vai trabalhar com esse trecho deve estar **preparado** para enxergá-lo da maneira correta

- Vê-se que muita coisa em **computação** é uma questão de **programação**

2.2 – Sistemas de Numeração e Mudança de Base

- Desde a antiguidade, os seres humanos utilizam o **sistema decimal** de numeração **(base 10)**

- **Alfabeto** do sistema decimal:

$$\Sigma_{10} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

- Valores numéricos **acima de 9**, utilizam **mais de um dígito decimal** (Exemplos: 15, 26, 367, 289170)

- Num computador, todas as informações são sequências de **0's** e **1's**

- Nos computadores então, nada mais natural do que manipular valores numéricos no **sistema binário** de numeração **(base 2)**

- **Alfabeto** do sistema binário:

$$\Sigma_2 = \{0, 1\}$$

- Valores numéricos **acima de 1**, utilizam **mais de um dígito binário** (Exemplos: 10, 101, 1111, 100010)

- Qualquer número natural pode ser **base** de um sistema de numeração:

Sistema	Base	Alfabeto
Decimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Binário	2	0, 1
Ternário	3	0, 1, 2
Quaternário	4	0, 1, 2, 3
Octal	8	0, 1, 2, 3, 4, 5, 6, 7
Hexadecimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

- **Equivalência** entre os sistemas:

$$(29)_{10} = (11101)_2 = (1002)_3 = (131)_4 = (35)_8 = (1D)_{16}$$

- Em Computação usa-se muito os sistemas **binário**, **decimal**, **octal** e **hexadecimal**

Sistema	Base	Alfabeto
Decimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Binário	2	0, 1
Ternário	3	0, 1, 2
Quaternário	4	0, 1, 2, 3
Octal	8	0, 1, 2, 3, 4, 5, 6, 7
Hexadecimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Conversão de base:

- Nesta disciplina, para se converter um número de uma base $\mathbf{b_1}$ para outra $\mathbf{b_2}$:

1. Converte-se de $\mathbf{b_1}$ para a **base 10**
2. Converte-se da **base 10** para $\mathbf{b_2}$

Assim evita-se fazer cálculos fora da base 10

Conversão de base para números inteiros:

- Seja o seguinte número inteiro **N** escrito numa **base b** qualquer:

$$(N)_b = (A_n\ A_{n-1}\ A_{n-2}\ .....\ A_2\ A_1\ A_0)_b$$

- A_n, A_{n-1}, A_{n-2}, ... A_2, A_1 e A_0 são dígitos do alfabeto do sistema de numeração de **base b**.

- A conversão da base **b** para a base **10** utiliza o seguinte polinômio:

$$(N)_{10} = A_n\ b^n + A_{n-1}\ b^{n-1} + A_{n-2}\ b^{n-2} + ... + A_2\ b^2 + A_1\ b^1 + A_0\ b^0$$

Exemplos:

1. $(11100101)_2 =$

   $= 1 * 2^7 + 1 * 2^6 + 1 * 2^5 + 0 * 2^4 + 0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0$

   $= 2^7 + 2^6 + 2^5 + 2^2 + 2^0$

   $= (229)_{10}$

2. $(271)_8$

   $= 2 * 8^2 + 7 * 8^1 + 1 * 8^0$

   $= (185)_{10}$

3. $(2AB3)_{16}$

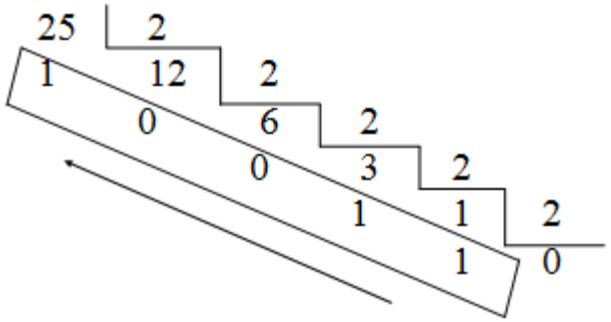
   $= 2 * 16^3 + 10 * 16^2 + 11 * 16^1 + 3 * 16^0$

   $= (10931)_{10}$

- Na conversão da base **10** para a base **b**, divide-se sucessivamente o número na base **10** por **b**, colhendo-se os **resto**s e **invertendo**-se a ordem dos mesmos

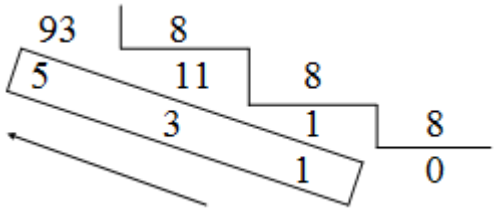
- **Exemplos:**

$(25)_{10}$ para a base 2



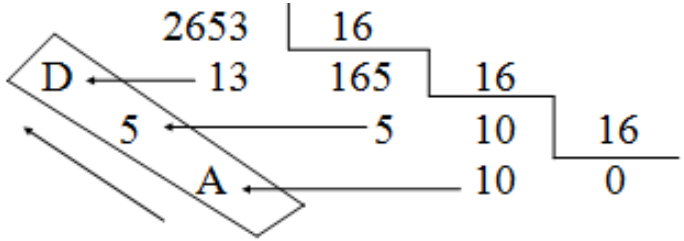
$(25)_{10} = (11001)_2$

$(93)_{10}$ para a base 8



$(93)_{10} = (135)_8$

$(2653)_{10}$ para a base 16



$(2653)_{10} = (A5D)_{16}$

Conversão de base para números reais:

- Converte-se **separadamente** a parte **inteira** da parte **fracionária**
- A conversão da **parte inteira** acaba de ser **vista**
- Será vista agora a conversão da **parte fracionária**

Conversão de base para números reais:

- Seja o seguinte número fracionário **N** escrito numa **base b** qualquer:

$$(N)_b = (0.A_{-1}\ A_{-2}\ A_{-3}\ .....\ A_{-n+2}\ A_{-n+1}\ A_{-n})_b$$

- A_{-1}, A_{-2}, A_{-3}, ... A_{-n+2}, A_{-n+1} e A_{-n} são dígitos do alfabeto do sistema de numeração de **base b**.

- Para números fracionários, a conversão da base **b** para a base **10** utiliza o seguinte polinômio:

$$(N)_{10} = A_{-1}b^{-1} + A_{-2}b^{-2} + A_{-3}b^{-3} + \ldots. + A_{-n+2}b^{-n+2} + A_{-n+1}b^{-n+1} + A_{-n}b^{-n}$$

Exemplos:

1. $(0.1011)_2$

   $= 1 * 2^{-1} + 1 * 2^{-3} + 1 * 2^{-4}$

   $= (0.6875)_{10}$

2. $(0.307)_8$

   $= 3 * 8^{-1} + 7 * 8^{-3}$

   $= (0.388671875)_{10}$

3. $(0.2B)_{16}$

   $= 2 * 16^{-1} + 11 * 16^{-2}$

   $= (0.16796875)_{10}$

Conversão de um número fracionário da base 10 para a base b:

1. Multiplicar o número na base 10 por b, separando a parte inteira do resultado

2. Aplicar o passo 1 à parte fracionária restante, repetidamente, até obter zero para a nova parte fracionária

3. Caso nunca se chegue ao zero, pode ser uma dízima periódica

4. As partes inteiras obtidas são os dígitos da conversão

Exemplo: converter $(0.7265625)_{10}$ para a base 8

$$8 * 0.7265625 = \mathbf{5} + 0.8125$$
$$8 * 0.8125 = \mathbf{6} + 0.5$$
$$8 * 0.5 = \mathbf{4} + 0.0$$

$$(0.7265625)_{10} = (0.564)_8$$

Exemplo: converter $(0.169189453125)_{10}$ para a base 16

$$
\begin{aligned}
16 * 0.169189453125 &= \mathbf{2} + 0.70703125 \\
16 * 0.70703125 &= \mathbf{11} + 0.3125 \\
16 * 0.3125 &= \mathbf{5} + 0.0
\end{aligned}
$$

$$(0.169189453125)_{10} = (0.2B5)_{16}$$

Exemplo: converter $(0.3)_{10}$ para a base 2

$$
\begin{aligned}
2 * 0.3 &= \mathbf{0} + 0.6 \\
2 * 0.6 &= \mathbf{1} + 0.2 \\
2 * 0.2 &= \mathbf{0} + 0.4 \\
2 * 0.4 &= \mathbf{0} + 0.8 \\
2 * 0.8 &= \mathbf{1} + 0.6 \\
2 * 0.6 &= \mathbf{1} + 0.2 \\
2 * 0.2 &= \mathbf{0} + 0.4 \\
2 * 0.4 &= \mathbf{0} + 0.8 \\
2 * 0.8 &= \mathbf{1} + 0.6 \\
2 * 0.6 &= \mathbf{1} + 0.2
\end{aligned}
$$

Dízima periódica

Um mesmo número pode ser expresso por dízimas periódicas em algumas bases e em outras não

Porém a racionalidade de um número independe da base

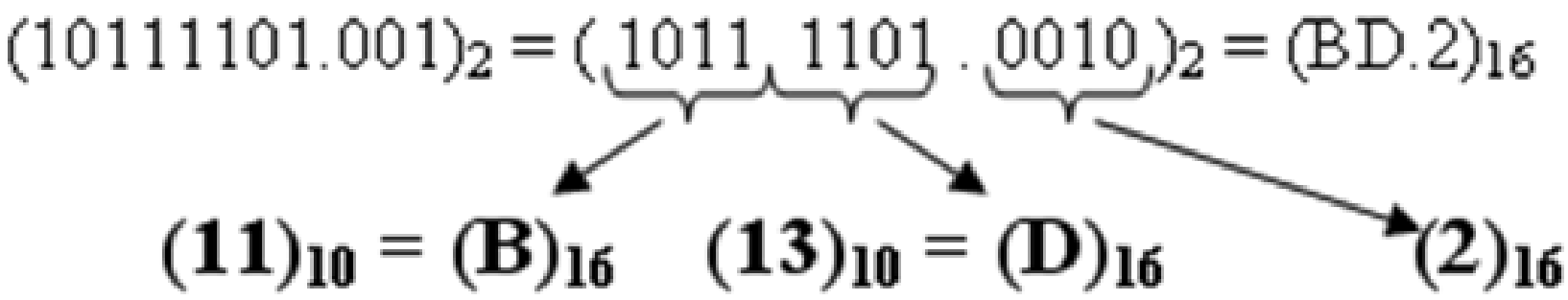
$$(0.3)_{10} = (0,0\ \underbrace{1001\ 1001}\ 1\ldots)_2$$

Conversão entre sistema binário e sistemas de bases potências de 2:

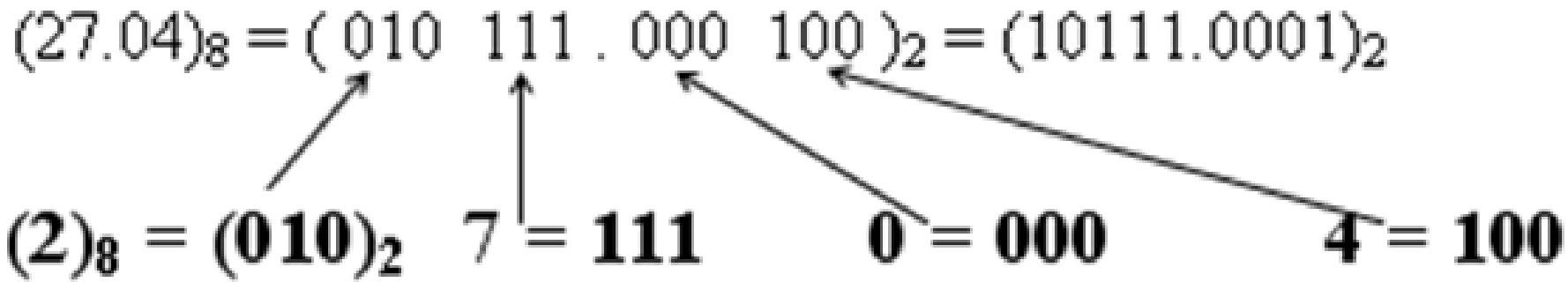
- A **base 10** não será usada como **intermediária**

- Cada grupo de **n dígitos** de um número escrito numa **base b** qualquer corresponde a **um único dígito** do mesmo número escrito na **base b^n**

Exemplo: base 2 para base 16



Exemplo: base 8 para base 2



Exercícios 2.2:

Efetuar as seguintes mudanças de base:

1. 13A5C da base 16 para a base 5
2. 1001101101 da base 2 para a base 10
3. B12F da base 16 para a base 10
4. 184 da base 10 para a base 2
5. 1632 da base 10 para a base 8
6. 23.6 da base 8 para a base 3
7. 11010111.01011 da base 2 para a base 8, sem usar base intermediária
8. 42.B6 da base 16 para a base 2 sem usar base intermediária

2.3 – Aritmética binária e operações lógicas

- **Circuitos aritméticos** baseiam-se nas operações aritméticas no **sistema binário**

- **Exemplo:** soma, subtração e multiplicação de números binários: a seguir

Importante:

carry e borrow

Também há carry na multiplicação

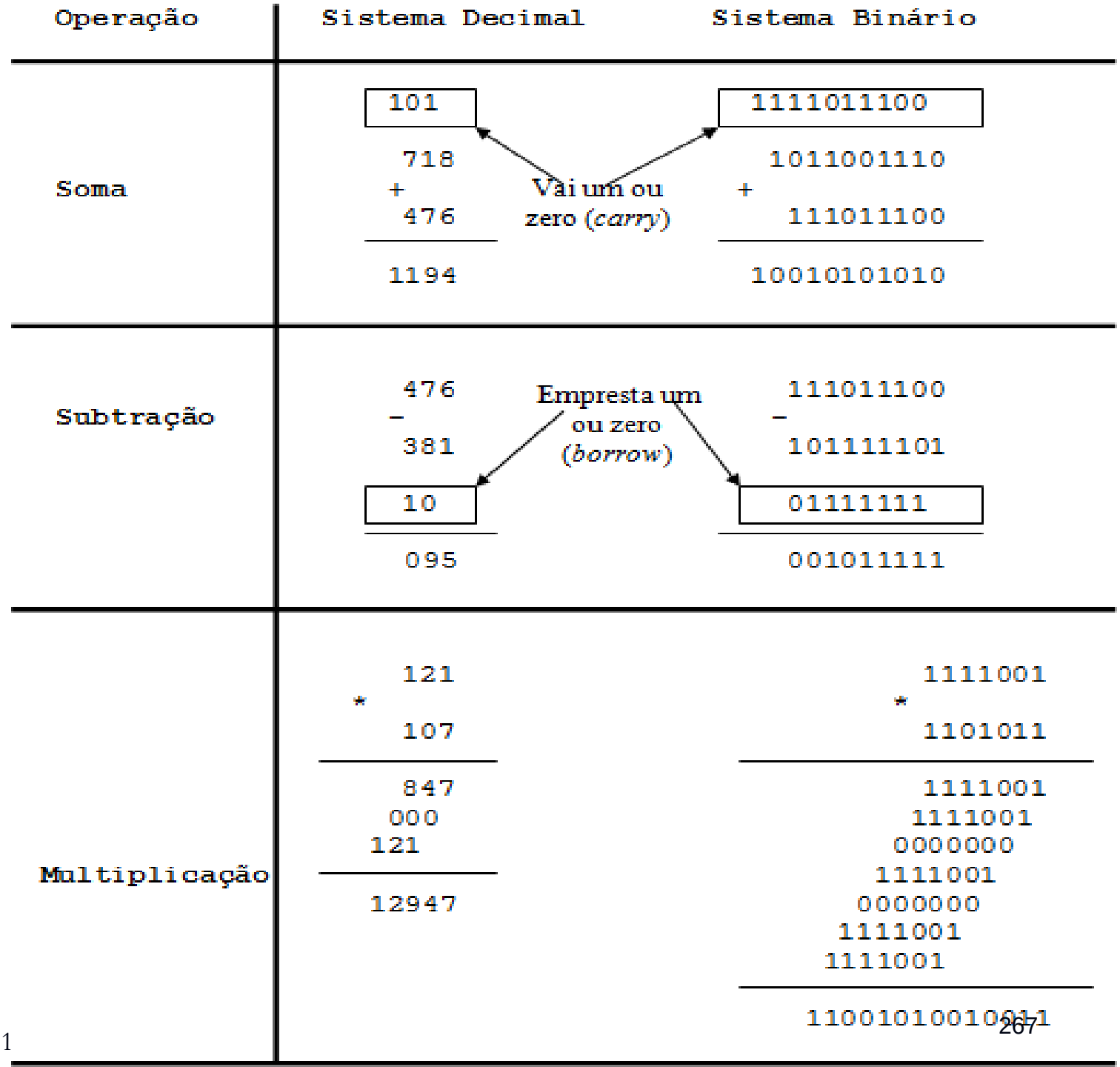
Operação	Sistema Decimal	Sistema Binário
Soma	101 718 + 476 1194 Vai um ou zero (*carry*)	1111011100 1011001110 + 111011100 10010101010
Subtração	476 – 381 10 095 Empresta um ou zero (*borrow*)	111011100 – 101111101 01111111 001011111
Multiplicação	121 * 107 847 000 121 12947	1111001 * 1101011 1111001 1111001 0000000 1111001 0000000 1111001 1111001 1100101001011

Exemplo: seja a soma

Os cálculos vão da direita para a esquerda

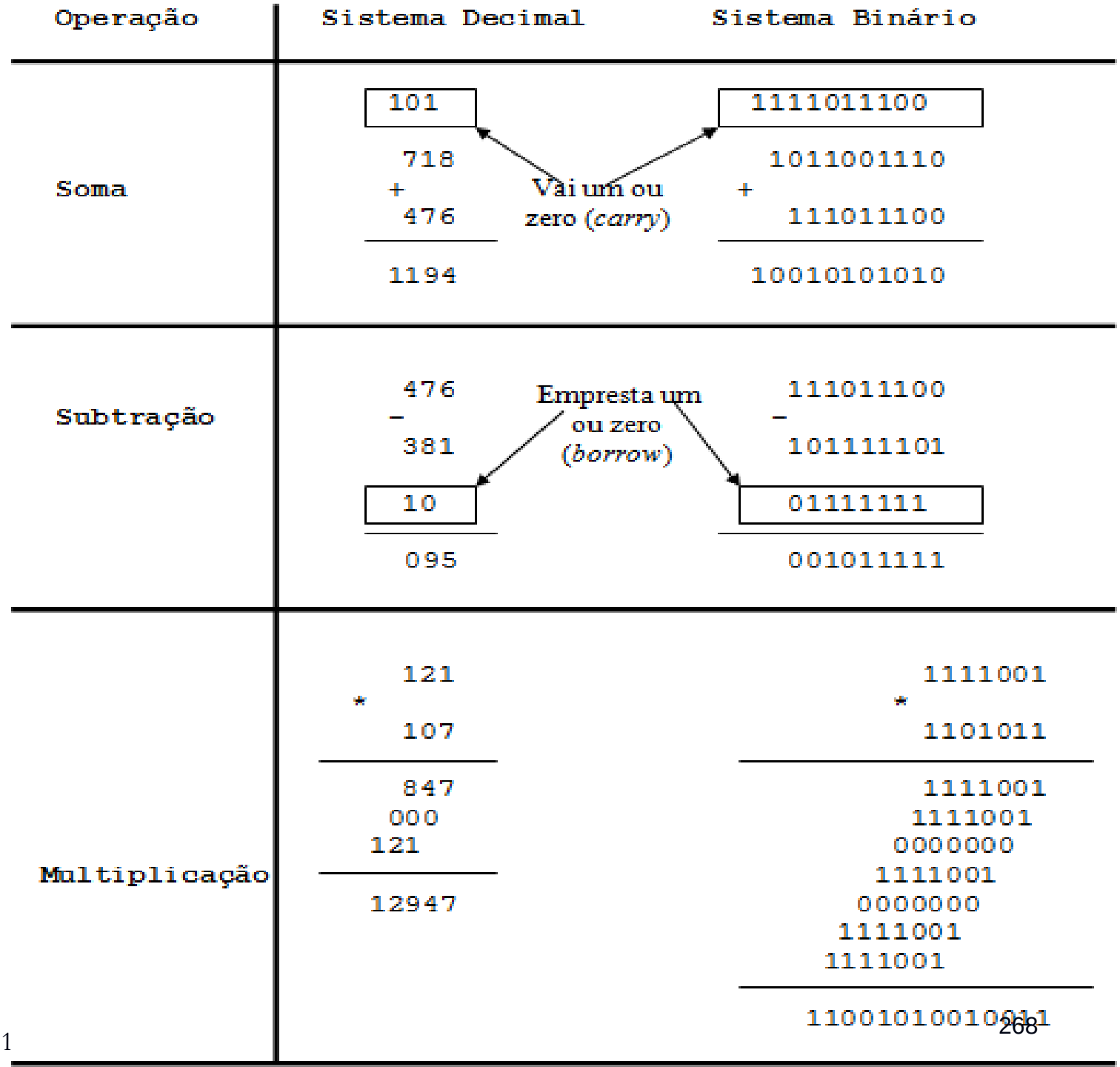
Primeiro cálculo: dois bits de entrada

Produz-se dois bits: soma e carry



Segundo cálculo: três bits de entrada (carry do cálculo anterior)

Também produz-se dois bits: soma e carry



Tabuada da aritmética binária:

Operandos		a + b		a - b		a * b
a	b	soma	carry	subtração	borrow	produto
0	0	0	0	0	0	0
0	1	1	0	1	1	0
1	0	1	0	1	0	0
1	1	0	1	0	0	1

Valores e operações lógicas:

- O resultado do teste de uma **condição** é um de apenas dois valores possíveis:

Verdade ou **Falso**

- Estes valores são denominados **valores lógicos**

- **Valores lógicos** não são números, mas podem ser convenientemente mapeados nos **dígitos binários**:

1 = Verdade e **0 = Falso**

Exemplos: nas instruções

- Se **não há cartões na entrada**, executar **Esc 9**;
- Se **Esc 22 = 0**, executar **Esc 12**;
- Se **Esc 24 > Esc 21**, executar **Esc 10**;

Não há cartões na entrada

Esc 22 = 0

Esc 24 > Esc 21

São condições a serem testadas

Seus valores só podem ser **verdade** ou **falso**

Operadores relacionais: <, ≤, >, ≥, =, ≠

- Comparam duas grandezas
- Seu resultado é um valor lógico

Operadores lógicos: and, or, not, exor

- Seus operandos são grandezas lógicas
- Seu resultado é um valor lógico

Exemplos com operadores lógicos:

- Seja a frase:

Se (meu carro tem combustível) e (sua mecânica está em bom estado), então (saio com ele)

- A ação **(saio com ele)** só será realizada se ambas as condições

(meu carro tem combustível) e
(sua mecânica está em bom estado)

forem verdade

A palavra **"e"** é um operador lógico

Seu resultado é **verdade**, se ambos os operandos forem **verdade**

Exemplos com operadores lógicos:

- Seja a frase:

Se (meu carro tem combustível) e (sua mecânica está em bom estado), então (saio com ele)

- A ação **(saio com ele)** só será realizada se ambas as condições

(meu carro tem combustível) e
(sua mecânica está em bom estado)

forem verdade

Em CES-10, no lugar da palavra **"e"** será usado **"and"** ou então o símbolo **"&&"**

Exemplos com operadores lógicos:

- Seja a frase:

Se (meu carro está em ordem) ou (tenho carona) então (vou ao teatro)

- A ação **(vou ao teatro)** será realizada se pelo menos uma das condições

(meu carro está em ordem) ou **(tenho carona)**

for verdade

A palavra **"ou"** é um operador lógico

Seu resultado é **verdade**, se pelo menos um dos operandos for **verdade**

Exemplos com operadores lógicos:

- Seja a frase:

Se (meu carro está em ordem) ou (tenho carona) então (vou ao teatro)

- A ação **(vou ao teatro)** será realizada se pelo menos uma das condições

(meu carro está em ordem) ou **(tenho carona)**

for verdade

Em CES-10, no lugar da palavra **"ou"** será usado **"or"** ou então o símbolo **"||"**

Exemplos com operadores lógicos:

- Seja a frase:

  Se não (estou gripado) então (vou ao clube nadar)

- A ação **(vou ao clube nadar)** será realizada se a condição

  (estou gripado)

  for falsa

A palavra **"não"** é um operador lógico

Seu resultado é **verdade**, se seu operando for **falso**

Exemplos com operadores lógicos:

- Seja a frase:

  Se não (estou gripado) então (vou ao clube nadar)

- A ação **(vou ao clube nadar)** será realizada se a condição

  (estou gripado)

  for falsa

Em CES-10, no lugar da palavra **"não"** será usado **"not"** ou então o símbolo **"!"**

Outro operador lógico - exor:

- **exor:** exclusive-or (ou-exclusivo)
- Operador lógico de dois operandos
- O resultado só será **verdade** se os operandos forem **diferentes entre si**

Tabuada dos operadores lógicos:

Operandos		a and b	a or b	a exor b	not a
a	b				
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

- Pela Álgebra Booleana, **exor** pode ser expressa em termos de **and**, **or** e **not**:

a exor b = ((not a) and b) or (a and (not b))

Tabuada dos operadores lógicos:

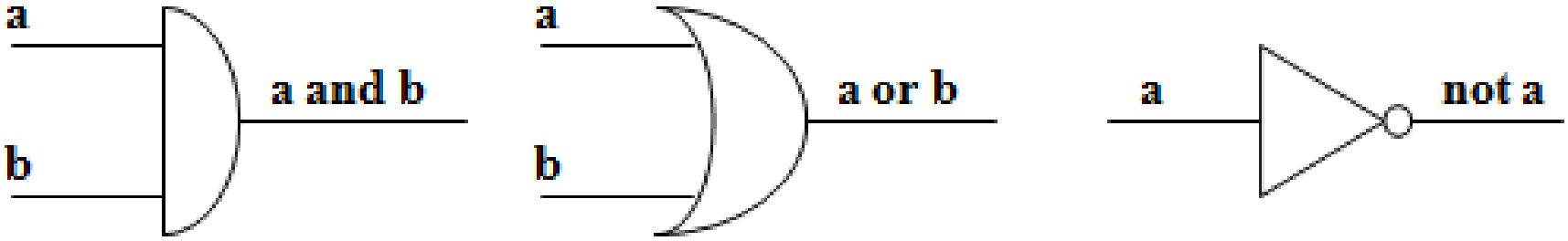
Operandos		a and b	a or b	a exor b	not a
a	b				
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

- Cada coluna desta tabela é a **tabela-verdade** de um operador

Circuitos aritméticos digitais:

- No mercado, existem componentes para os operadores **and**, **or** e **not**:

Livro 01



- Esses componentes são **portas lógicas** para esses operadores

- As portas **and** e **or** podem ter mais de duas entradas

Circuitos aritméticos digitais:

- Equivalência entre tabelas-verdades aritméticas e lógicas:

Operandos		a + b		a - b		a * b
a	b	soma	carry	subtração	borrow	produto
0	0	0	0	0	0	0
0	1	1	0	1	1	0
1	0	1	0	1	0	0
1	1	0	1	0	0	1

Operandos		a and b	a or b	a exor b	not a
a	b				
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Soma = Subtração = exor

Carry = Produto = and

Operandos		a + b		a - b		a * b
a	b	soma	carry	subtração	borrow	produto
0	0	0	0	0	0	0
0	1	1	0	1	1	0
1	0	1	0	1	0	0
1	1	0	1	0	0	1

Operandos		a and b	a or b	a exor b	not a
a	b				
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

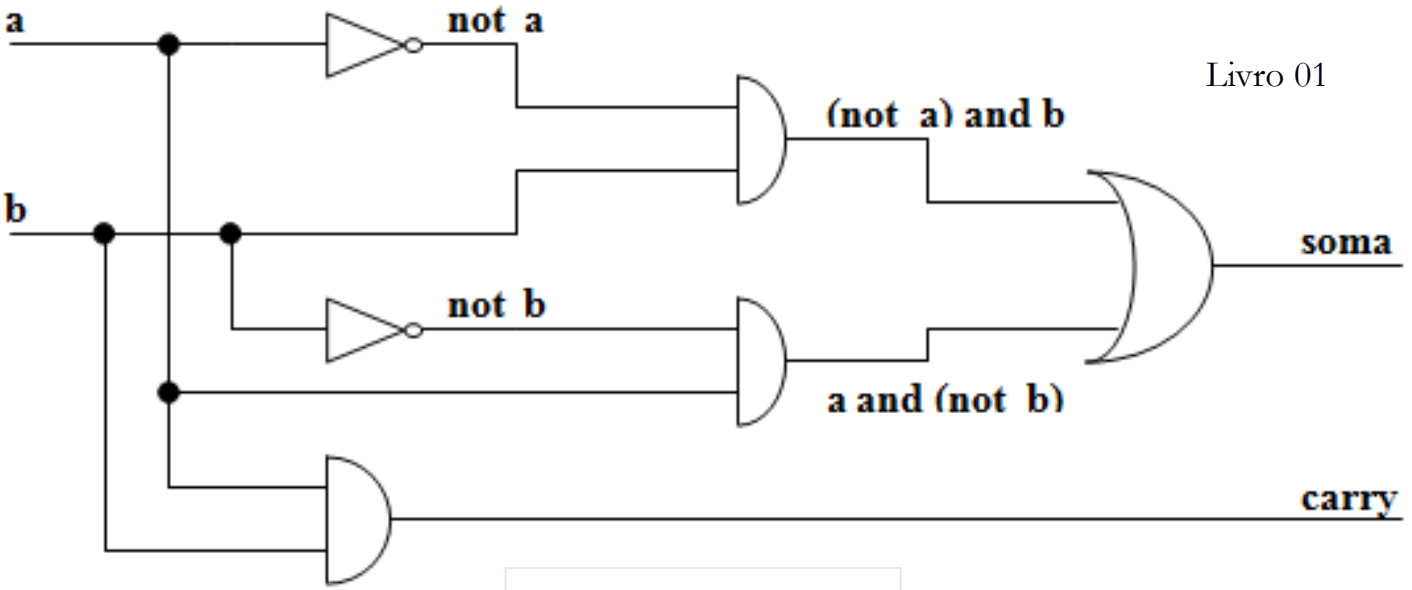
Assim:

soma = a + b = a exor b = ((not a) and b) or (a and (not b))

carry = a and b

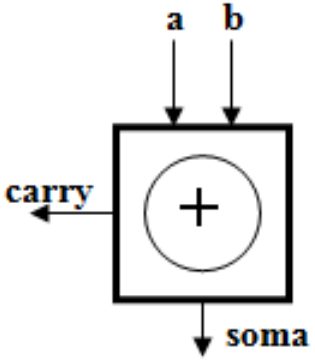
soma = a + b = a exor b = ((not a) and b) or (a and (not b))
carry = a and b

- Circuito somador de dois números de um bit:

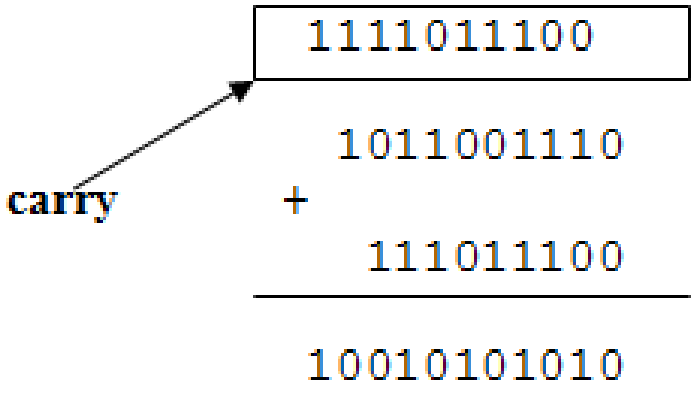


Livro 01

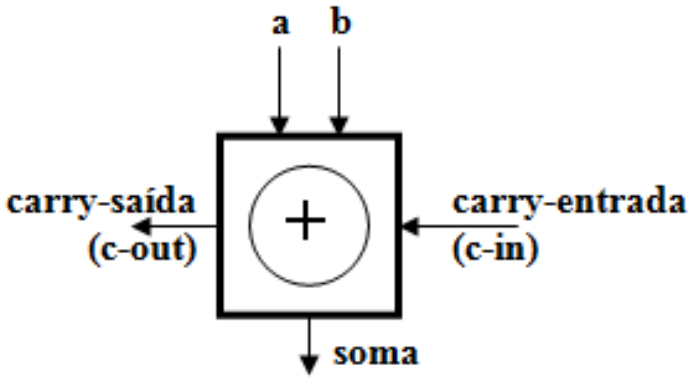
- Abreviadamente:



Somador de dois números com dois ou mais bits:



- Pode ser a **concatenação** de vários somadores de **três** bits cada:

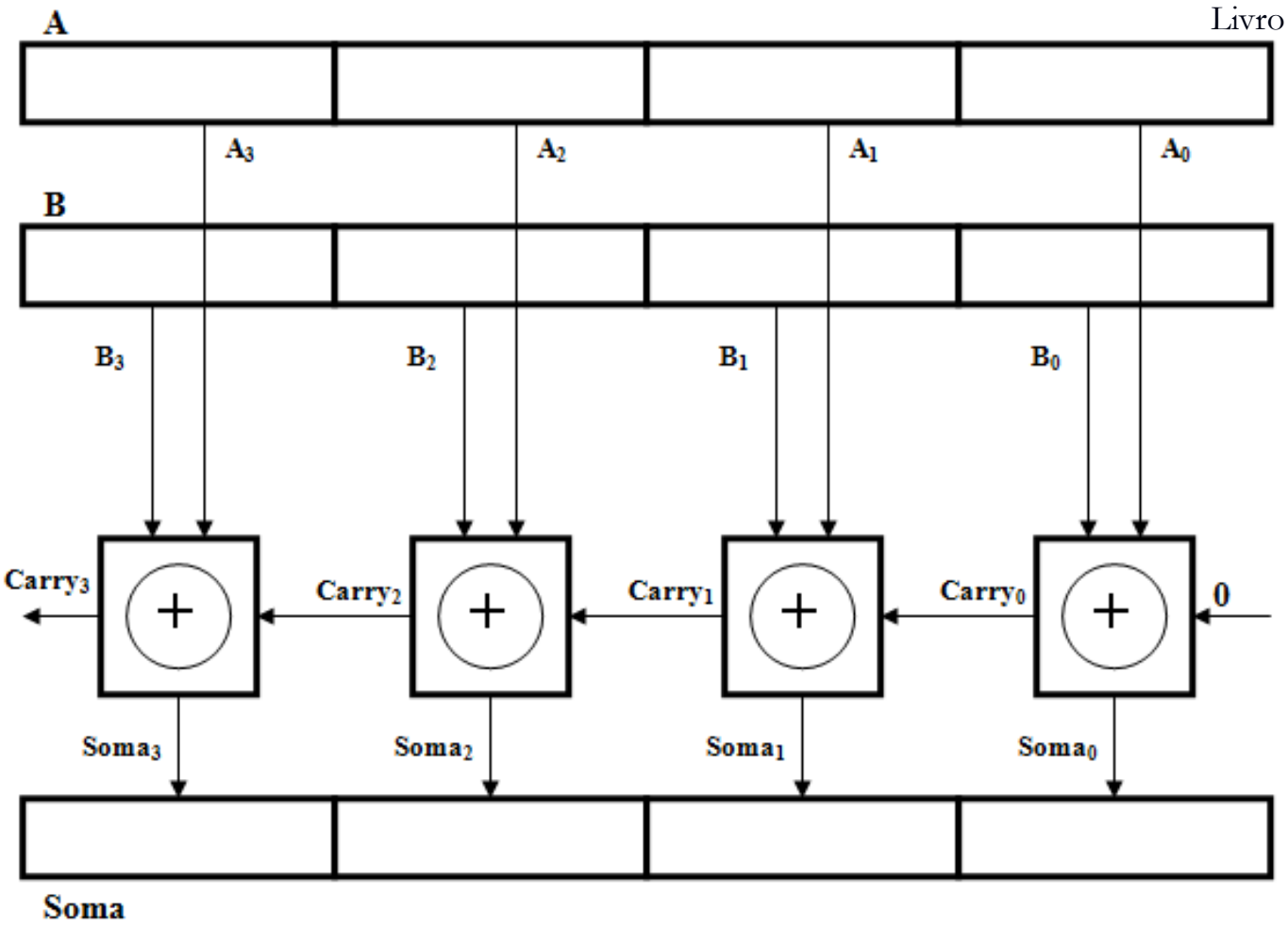


soma = a exor b exor c-in

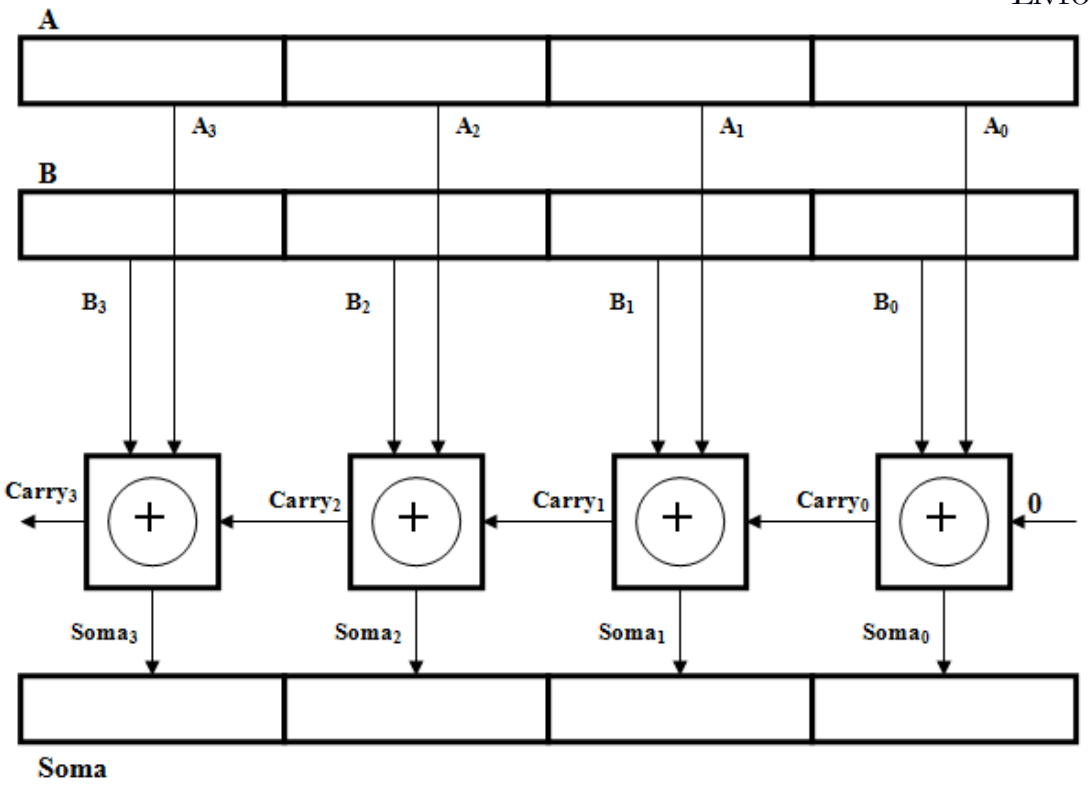
c-out = (a and b) or (a and c-in) or (b and c-in)

Exemplo: somador de dois números de 4 bits:

Livro 01



- Seu nome é **somador com propagação de carry**



- **Caráter sequencial:** cada somador de bit precisa do carry do somador da direita

- Há somadores que **dispensam** a sequencialidade: são mais sofisticados e mais **rápidos**

2.4 – Representação de números inteiros

Assuntos relacionados:

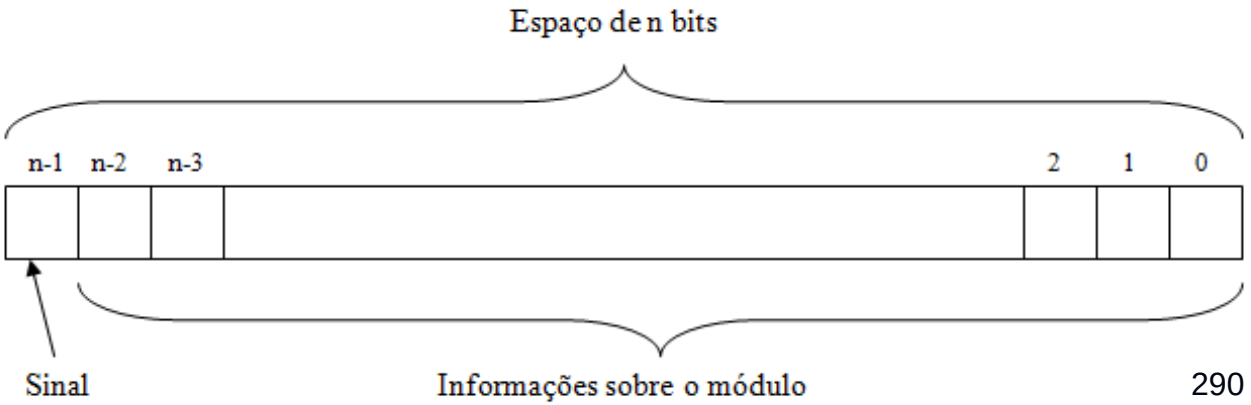
- Regras de **conversão** para o **sistema binário** de numeração (vistas anteriormente)
- **Espaço em bits** ou bytes para guardar números: varia com cada **hardware** e com cada **software** que os manipula
- Tratamento de **números negativos**: para inteiros, armazenar o **sinal** e o **módulo** (visto a seguir)

Sejam usados **n bits** para representar números inteiros

Representação do sinal:

- Usar o bit mais à esquerda como **bit-sinal**; bit **1** para os números negativos e bit **0** para os outros números

- Os outros bits devem guardar informações sobre o **módulo**



Representação do módulo por sinal + magnitude:

- Idéia imediatista: armazenar o **valor do módulo** em binário (fácil interpretação)

- Dado o **conteúdo** de um espaço em bits na memória, onde está guardado um **número inteiro**, dizer qual é esse número, na base 10:

  - Examina-se o **bit mais à esquerda** para saber o **sinal**

  - **Converte-se** os outros bits para o sistema decimal para saber o **módulo**

Exemplo: seja um sistema que utiliza **5 bits** para armazenar inteiros; 1 para o sinal e 4 para o módulo

Quais os números armazenados pelos conteúdos **01110** e **10110**?

- **01110:** sinal não negativo; módulo $(1110)_2 = (14)_{10}$

  01110 corresponde a $\mathbf{+(14)_{10}}$

- **10110:** sinal negativo; módulo $(0110)_2 = (6)_{10}$

  10110 corresponde a $\mathbf{-(6)_{10}}$

Inconvenientes da representação sinal + magnitude:

1. Possui dois zeros: **00000** e **10000**

   Complica o esquema dos circuitos aritméticos

2. Nos circuitos aritméticos, o **bit-sinal** deve ter **tratamento diferenciado** dos demais bits

   Circuitos aritméticos mais **caros** e mais **lentos**

- Essa representação **não** é mais **utilizada**

Exemplo: erro pelo tratamento não diferenciado do bit-sinal

- Seja a soma **10 + (-5)**, usando 5 bits para inteiros

- **$(10)_{10} \rightarrow$ 01010; $(-5)_{10} \rightarrow$ 10101;**

```
  0 1 0 1 0
  +
  1 0 1 0 1
  ---------
  1 1 1 1 1 = -15 → Erro
```

Representação do módulo em complemento de 2:

- Utiliza o bit-sinal da **mesma maneira**

- Outros bits:

  - Para números **não-negativos**: devem conter o **módulo**, tal como em sinal + magnitude

  - Para números **negativos**: devem conter o **complemento do módulo**, somado com **1**

  - **Complemento** de um conjunto de bits: todos os bits são **invertidos** (complemento de **0010** é **1101**)

Exemplo: representação em comp-2 de **5 bits** dos números +9 e -6

- **Comp-2 (+9):** bit-sinal 0; $(9)_{10} = (1001)_2$

  Comp-2 (+9) = 01001

- **Comp-2 (-6):** bit-sinal 1; $(6)_{10} = (0110)_2$

  Complemento (0110) + 1 = 1001 + 1 = 1010

  Comp-2 (-6) = 11010

Exemplo: decimais correspondentes às seguintes representações em comp-2: **01101** e **11001**

- **01101:** bit-sinal é 0; $(1101)_2 = (13)_{10}$

  01101 = comp-2 (+13)

- **11001:** bit-sinal é 1;

  complemento(1001) + 1= 0110 + 1 = 0111 = $(7)_{10}$

  11001 = comp-2 (-7)

Exemplo: números representáveis em comp-2 de 5 bits

Decimal	Comp-2	Decimal	Comp-2
+15	01111	-1	11111
+14	01110	-2	11110
+13	01101	-3	11101
+12	01100	-4	11100
+11	01011	-5	11011
+10	01010	-6	11010
+9	01001	-7	11001
+8	01000	-8	11000
+7	00111	-9	10111
+6	00110	-10	10110
+5	00101	-11	10101
+4	00100	-12	10100
+3	00011	-13	10011
+2	00010	-14	10010
+1	00001	-15	10001
0	00000	-16	10000

Livro 01

- Números fora do intervalo [-16, +15] não são representáveis em comp-2 de 5 bits

Aritmética em complemento de 2:

1. Não apresenta dois zeros: **10000 = Comp-2 (-16)**

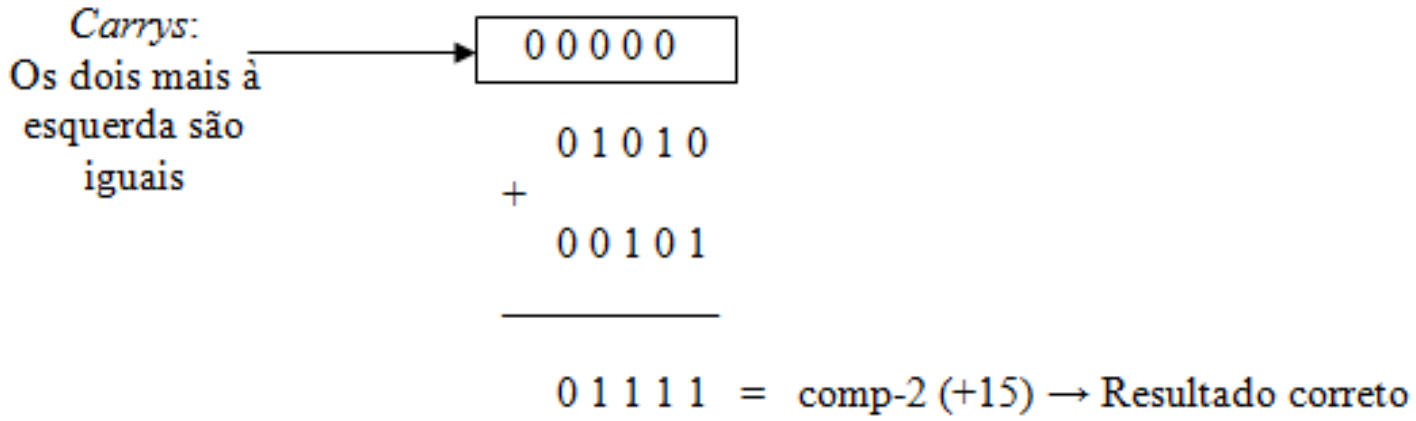
2. Nos circuitos aritméticos o bit-sinal tem o **mesmo tratamento** dos outros bits

- Para comprovar o item 2, sejam analisadas as seguintes operações de soma:

(10 + 5), (5 + (-6)), (15 + (-10)), (12 + 6), ((-9) + (-8))

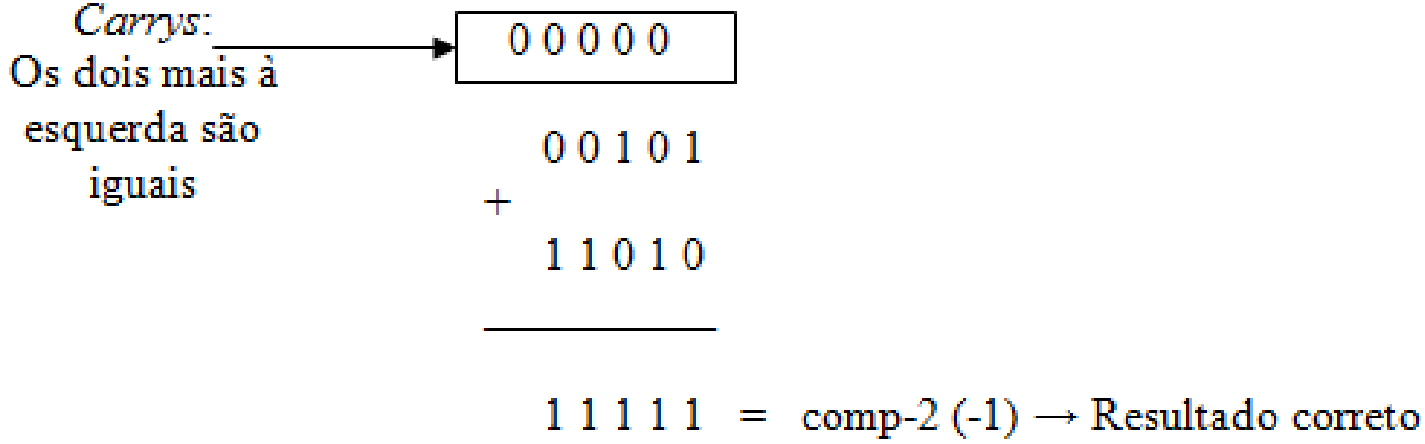
10 + 5: o resultado deve ser +15

Comp-2 (+10) = 01010; Comp-2 (+5) = 00101; Então:



5 + (-6): o resultado deve ser -1

Comp-2 (+5) = 00101; Comp-2 (-6) = 11010; Então:



15 + (-10): o resultado deve ser +5

Comp-2 (+15) = 01111; Comp-2 (-10) = 10110; Então:

Carrys: → 1 1 1 1 0

Os dois mais à esquerda são iguais

```
  0 1 1 1 1
+
  1 0 1 1 0
-----------
1 0 0 1 0 1
```

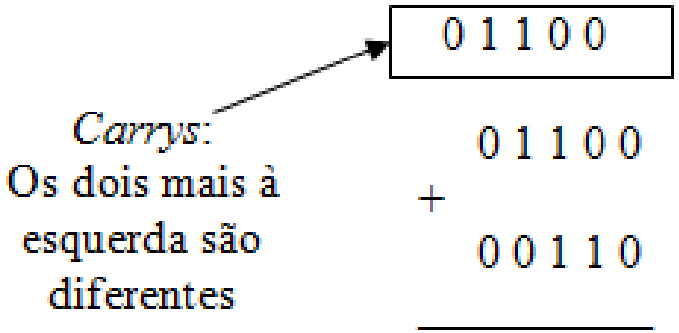
Como o número de bits do resultado deve ser 5, elimina-se o bit mais à esquerda, resultando

0 0 1 0 1 = comp-2 (+5) → Resultado correto

12 + 6: o resultado deve ser +18, fora de [-16, +15] (overflow);

Espera-se **resultado incorreto**!!!

Comp-2 (+12) = 01100; Comp-2 (+6) = 00110; Então:

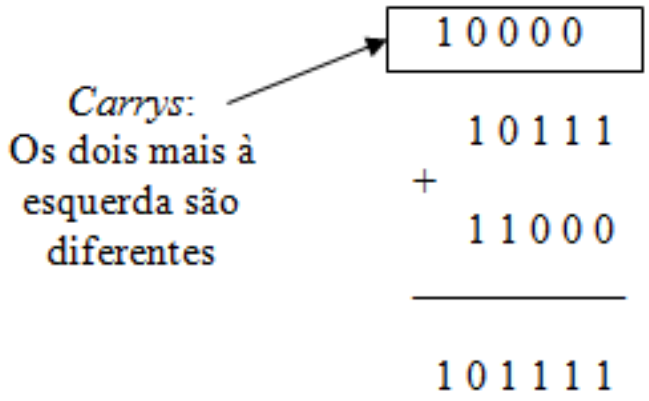


1 0 0 1 0 = comp-2 (-14) → Resultado incorreto, como esperado

(-9) + (-8): o resultado deve ser -17, fora de [-16, +15] (overflow);

Espera-se **resultado incorreto**!!!

Comp-2 (-9) = 10111; Comp-2 (-8) = 11000; Então:



Como o número de bits do resultado deve ser 5, elimina-se o bit mais à esquerda, resultando

0 1 1 1 1 = comp-2 (+15) → Resultado incorreto, como esperado

- Nas operações com resultados corretos

  Os dois carry's mais à esquerda foram iguais

- Nas operações com resultados incorretos

  Os dois carry's mais à esquerda foram diferentes

Overflow = $Carry_{n-1}$ exor $Carry_{n-2}$

Exercícios 2.4:

O códigos a seguir são números binários inteiros escritos em complemento de 2, cada um com diferente número de bits. Dizer o valor em decimal com sinal de cada um deles.

1. 11110
2. 11
3. 1010101
4. 010101010
5. 10000000

2.5 – Representação de números reais

- Qualquer número real **x** pode ser colocado na forma

$$\mathbf{x = 0.mantissa * base^{expoente}}$$

- Mantissa, base e expoente podem estar escritos em **qualquer base**

- Aqui, por razões didáticas, **base** e **expoente** serão expressos em **base 10**

Exemplos:

$(257.45)_{10}$
$= (0.25745)_{10} * 10^3$
$= (0.0025745)_{10} * 10^5$
$= (0.0000025745)_{10} * 10^8$

$(0.00101)_2$
$= (0.101)_2 * 2^{-2}$
$= (0.00000101)_2 * 2^3$

$(0.00000101)_2$
$= (0.0101)_2 * 16^{-1}$
$= (0.000000000101)_2 * 16^1$

Sistema de ponto-flutuante: representação de números reais onde um mesmo número pode ter o ponto em diferentes posições da mantissa, bastando ajustar o valor do expoente

Forma normalizada: forma em que o dígito mais significativo da mantissa está à **direita** do ponto e o mais **próximo** possível dele

Forma desnormalizada: forma em que isso não ocorre

$(257.45)_{10}$
$= (0.25745)_{10} * 10^{3}$
$= (0.0025745)_{10} * 10^{5}$
$= (0.0000025745)_{10} * 10^{8}$

$(0.00101)_{2}$
$= (0.101)_{2} * 2^{-2}$
$= (0.00000101)_{2} * 2^{3}$

$(0.00000101)_{2}$
$= (0.0101)_{2} * 16^{-1}$
$= (0.000000000101)_{2} * 16^{1}$

Armazenamento no computador:

$$x = 0.\text{mantissa} * \text{base}^{\text{expoente}}$$

- Armazena-se a **mantissa** e o **expoente** no sistema binário
- A **base** é uma **potência de 2** e não é armazenada
  - Ela é **implícita** nos **circuitos** e/ou nos **programas** que manipulam números reais

Armazenamento no computador:

$$x = 0.\text{mantissa} * \text{base}^{\text{expoente}}$$

- Utiliza-se sistema de ponto-flutuante e forma normalizada

- Tipicamente usa-se:
  - Um bit para o **sinal**
  - Um determinado número de bits para a **mantissa em comp-2**
  - Um determinado número de bits para o **expoente**

- Não há **bit-sinal** para **expoentes**

- Para se armazenar o **expoente** de um número:

  - Soma-se a ele o módulo do expoente negativo de **maior módulo** representável, obtendo-se o que se chama **característica** do número

  - Armazena-se a **característica**, em vez do expoente

Exemplo: seja um sistema de ponto-flutuante com as seguintes dimensões:

- **32 bits (4 bytes)** para armazenar n^{os} reais
- **1 bit** para o sinal
- **23 bits** para a mantissa
- **8 bits (1 byte)** para o expoente (ou característica)
- **Base = 2**

Como ficam armazenados nesse sistema os seguintes números:

$$\mathbf{(0.5625)_{10}\ ,\ (2.3)_{10}\ e\ (0.0625)_{10}\ ?}$$

- **32 bits (4 bytes)** para armazenar n^{os} reais
- **1 bit** para o sinal
- **23 bits** para a mantissa
- **8 bits (1 byte)** para o expoente (ou característica)
- **Base = 2**

Com **8 bits** para expoentes, esses podem variar no intervalo **[-128, 127]**

Expoente negativo de maior módulo: **-128**

Característica = Expoente + 128

Representação do número $(0.5625)_{10}$:

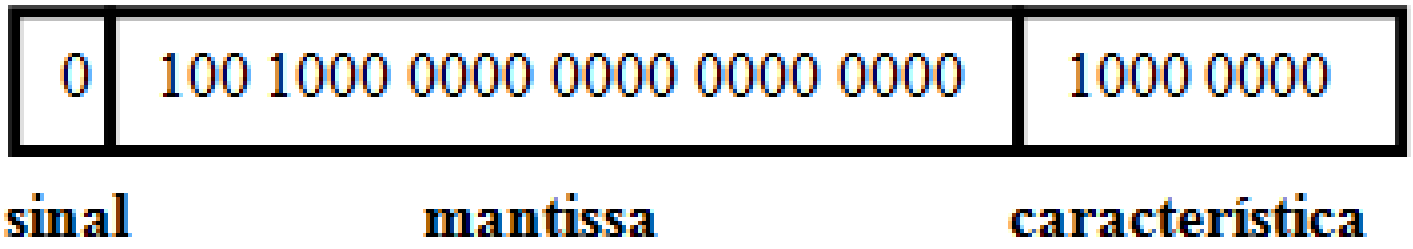
$(0.5625)_{10} = (0.1001)_2 = (0.1001)_2 * 2^0$;

Já está normalizado, logo

característica = 0 + 128 = $(128)_{10}$ = **$(1000\ 0000)_2$**

mantissa = **100 1000 0000 0000 0000 0000**

Então, $(0.5625)_{10}$ fica assim armazenado:



Representação do número $(2.3)_{10}$:

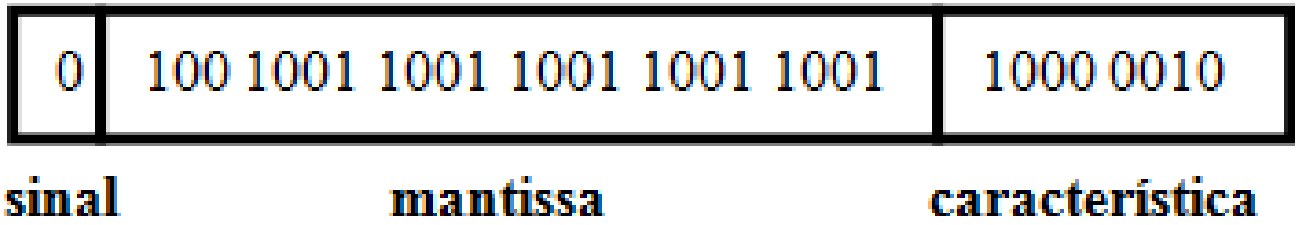
$(2.3)_{10} = (10.0\ 1001\ 1001\ 1001\ ...)_2 * 2^0$; desnormalizado

$(2.3)_{10} = (0.100\ 1001\ 1001\ 1001\ ...)_2 * 2^2$

característica $= 2 + 128 = (130)_{10} =$ **$(1000\ 0010)_2$**

mantissa = 100 1001 1001 1001 1001 1001

Então, $(2.3)_{10}$ fica assim armazenado:



A mantissa sofreu um truncamento!

Imprecisão!

Representação do número $(0.0625)_{10}$:

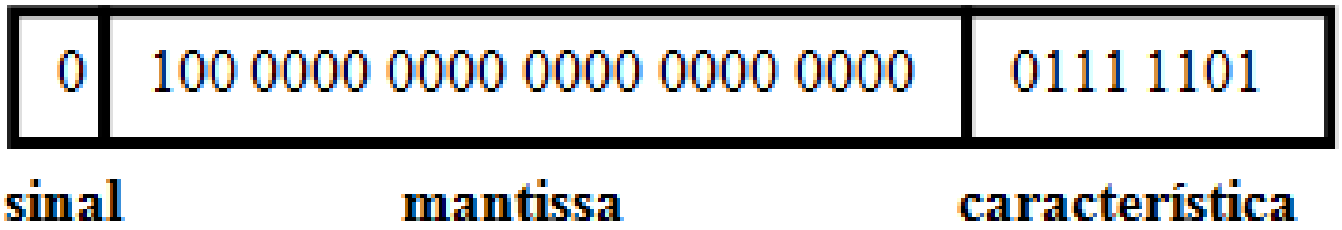
$(0.0625)_{10} = (0.0001)_2 * 2^0$; desnormalizado

$(0.0625)_{10} = (0.1)_2 * 2^{-3}$

característica = -3 + 128 = $(125)_{10}$ = **$(0111\ 1101)_2$**

mantissa = **100 0000 0000 0000 0000 0000**

Então, $(0.0625)_{10}$ fica assim armazenado:



Exemplo: seja um sistema de ponto-flutuante com as mesmas dimensões do sistema anterior, exceto:

- **Base = 16**

Como ficam armazenados nesse sistema os seguintes números:

$$\mathbf{(42.3)_{10}\ e\ (0.0625)_{10}\ ?}$$

- Uma alteração unitária no expoente corresponde a mover o ponto sobre 4 casas na mantissa

Representação do número $(42.3)_{10}$:

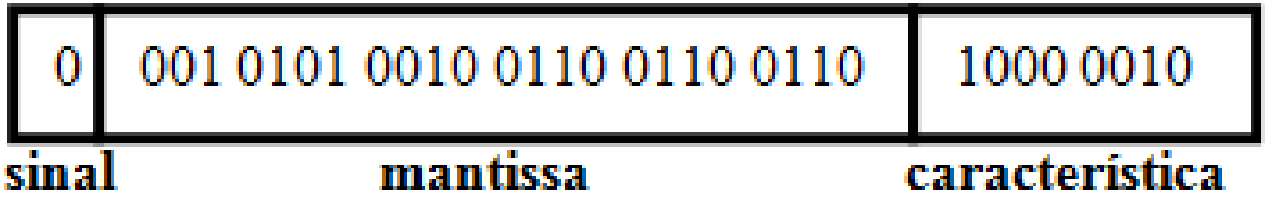
$(42.3)_{10} = (10\ 1010.0\ 1001\ 1001\ 1001\ ...)_2 * 16^0$;

$(42.3)_{10} = (0.001\ 0101\ 0010\ 0110\ 0110\ 0110\ ...)_2 * 16^2$
normalizado

característica = 2 + 128 = $(130)_{10}$ = **$(1000\ 0010)_2$**

mantissa = **001 0101 0010 0110 0110 0110**

Então, $(42.3)_{10}$ fica assim armazenado:



A mantissa sofreu um truncamento!

Imprecisão!

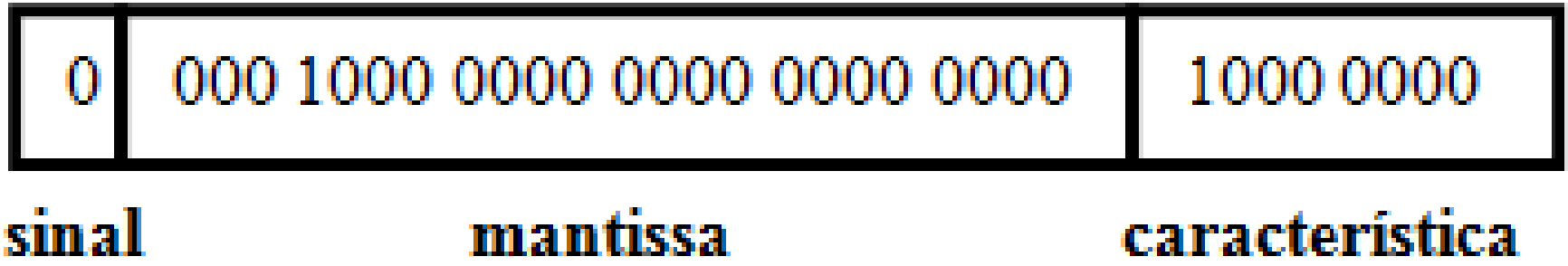
Representação do número $(0.0625)_{10}$:

$(0.0625)_{10} = (0.0001)_2 * 16^0$; já está normalizado

característica = 0 + 128 = $(128)_{10}$ = **$(1000\ 0000)_2$**

mantissa = **000 1000 0000 0000 0000 0000**

Então, $(0.0625)_{10}$ fica assim armazenado:

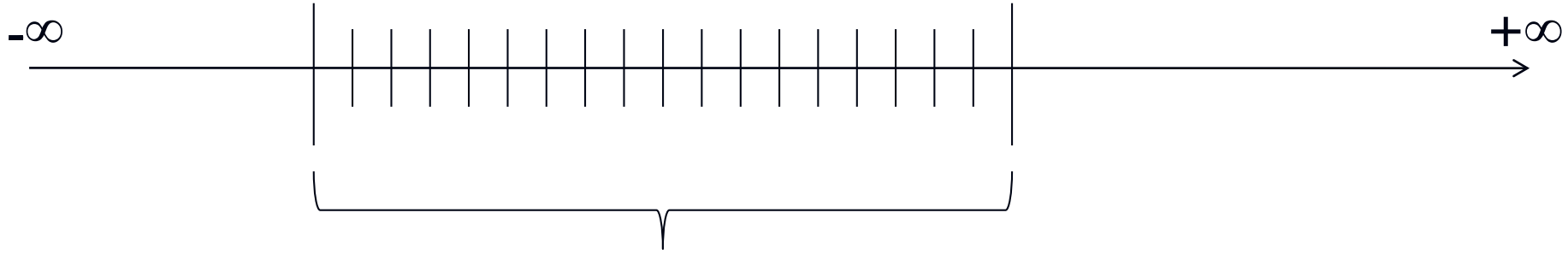


Observações:

- **Limitar o número de bits** para manipular reais delimita:
  - O **tamanho máximo** dos reais manipuláveis
  - A **precisão** com que os reais são transportados entre computador e ambiente externo

Observações:

- Há infinitos números reais



- Dentro de qualquer intervalo finito, a **quantidade de números reais é infinita**

- Dessa quantidade, apenas um **subconjunto finito** de números reais é **representável** no computador

Observações:

- Qualquer representação de números reais em computadores introduz **erros**
- É necessário **truncar a mantissa**, para que caiba no **espaço finito** de memória a ela reservado
- Quanto maior esse espaço, menores os erros introduzidos

Observações:

- Há processos numéricos que **amplificam** muito pequenos erros como esses, **comprometendo os resultados**

- Portanto a programação envolvendo números reais deve ser muito criteriosa para que os erros desses resultados não sejam significativos

Exercícios 2.5:

Um computador utiliza 32 bits (4 bytes) para armazenar números reais; 1 bit para o sinal, 8 bits para a característica e 23 bits para a mantissa.

1. Supondo que a base para a característica seja 2, dizer como ficam armazenado nesse computador os números $(242.6)_{10}$ e $(0.0859375)_{10}$

2. Supondo agora que a base seja 16, dizer como ficam armazenado nesse computador os números $(115,9)_{10}$ e $(0.0234375)_{10}$

2.6 – Representação de textos

- Textos são importantes informações manipuladas por computadores

- **Exemplos:**

  - Empregados de uma empresa
  - Livros de uma biblioteca
  - Conteúdo didático de aulas e livros

- **Texto** é uma sequência linear de símbolos de grafia denominados **caracteres**

- **Exemplos de caracteres:**

  - Letras maiúsculas e minúsculas: **A, B, ... , Z, a, b, ... , z**
  - Dígitos decimais: **0, 1, 2, ... , 8, 9**
  - Operadores aritméticos: **+, -, *, /**
  - Separadores: **(,), [,], {, }**
  - Caracteres de pontuação: **. , ; ! ? :**
  - Caracteres especiais: **@, #, $, %, &**
  - Caracteres de controle: muda linha **(nl)**, tabulação **(ht),** etc.

- Cada caractere ocupa **1 byte** (8 bits) na memória

- A introdução de textos na memória pelo teclado se dá por meio de **conversão de cada tecla** digitada para um determinado conjunto de **8 bits**

- Cada caractere tem seu conjunto específico denominado **padrão de bits**

- Com 8 bits, existem 256 padrões distintos, possibilitando representar 256 caracteres diferentes

- A determinação dos padrões de bits para cada caractere é estabelecido por padronizações internacionais denominadas **códigos para informações**

- O código mais conhecido é o **ASCII** (American Standard Code for Information Interchange)

- Ver tabela a seguir:

Tabela para o código ASCII:

Livro 01

	0	1	2	3	4	5	6	7	8	9
0	nul							bel	bs	ht
10	nl			cr						
20								esc		
30			sp	!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~	del		

Exemplos:

Livro 01

	0	1	2	3	4	5	6	7	8	9
0	nul							bel	bs	ht
10	nl			cr						
20								esc		
30			sp	!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~	del		

Caractere 'K':

linha 70 + coluna 5 = $(75)_{10} = (1001011)_2$;

ASCII ('K') = 01001011

Caractere '&':

linha 30 + coluna 8 = $(38)_{10} = (100110)_2$;

ASCII ('&') = 00100110

Caractere '{':

linha 120 + coluna 3 = $(123)_{10} = (1111011)_2$;

ASCII ('{') = 01111011

Para maior clareza, os códigos estarão em decimal

Livro 01

	0	1	2	3	4	5	6	7	8	9
0	nul							bel	bs	ht
10	nl			cr						
20								esc		
30			sp	!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	|	}	~	del		

Exemplos:

ASCII ('K') = 75

ASCII ('&') = 38

ASCII ('{') = 123

Esses valores são as representações internas dos caracteres

Significado de alguns caracteres de controle:

Caractere	Representação interna	Significado
nl	10	Iniciar nova linha (*new line*)
ht	9	Tabulação horizontal
cr	13	Voltar ao início da mesma linha (*carriage return*)
bs	8	Voltar um espaço (*back space*)
bel	7	Tocar a campainha
nul	0	Caractere nulo

- O caractere **bel** toca uma campainha, durante um curtíssimo período de tempo

- O caractere **nul** é muito usado como finalizador de sequências de caracteres

Exemplo: armazenamento do texto

"Cada caractere tem seu padrao de 8 bits; <nl> <nl>
Existem 256 padroes de bits distintos."

Livro 01

End	Carac	Repr. int	End	Carac	Repr. int	End	Carac	Repr. int	End	Carac	Repr. int
407	C	01000011	428	u	01110100	449	E	01000101	470	e	01100101
408	a	01100001	429	sp	00100000	450	x	01110111	471	sp	00100000
409	d	01100100	430	p	01110000	451	i	01101001	472	b	01100010
410	a	01100001	431	a	01100001	452	s	01110011	473	i	01101001
411	sp	00100000	432	d	01100100	453	t	01110100	474	t	01110100
412	c	01100011	433	r	01110010	454	e	01100101	475	s	01110011
413	a	01100001	434	a	01100001	455	m	01101101	476	sp	00100000
414	r	01110010	435	o	01101111	456	sp	00100000	477	d	01100100
415	a	01100001	436	sp	00100000	457	2	00110010	478	i	01101001
416	c	01100011	437	d	01110000	458	5	00110101	479	s	01110011
417	t	01110100	438	e	01100101	459	6	00110110	480	t	01110100
418	e	01100101	439	sp	00100000	460	sp	00100000	481	i	01101001
419	r	01110010	440	8	00111000	461	p	01110000	482	n	01101110
420	e	01100101	441	sp	00100000	462	a	01100001	483	t	01110100
421	sp	00100000	442	b	01100010	463	d	01110000	484	o	01101111
422	t	01110100	443	i	01101001	464	r	01110010	485	s	01110011
423	e	01100101	444	t	01110100	465	o	01101111	486	.	00101110
424	m	01101101	445	s	01110011	466	e	01100101	487	nul	00000000
425	sp	00100000	446	;	00111011	467	s	01110011	488		
426	s	01110011	447	nl	00001010	468	sp	00100000	489		
427	e	01100101	448	nl	00001010	469	d	01110000	490		

Armazenamento a partir do endereço 407

"Cada caractere tem seu padrao de 8 bits; <nl> <nl>
Existem 256 padroes de bits distintos."

Livro 01

End	Carac	Repr. int	End	Carac	Repr. int	End	Carac	Repr. int	End	Carac	Repr. int
407	C	01000011	428	u	01110100	449	E	01000101	470	e	01100101
408	a	01100001	429	sp	00100000	450	x	01110111	471	sp	00100000
409	d	01100100	430	p	01110000	451	i	01101001	472	b	01100010
410	a	01100001	431	a	01100001	452	s	01110011	473	i	01101001
411	sp	00100000	432	d	01100100	453	t	01110100	474	t	01110100
412	c	01100011	433	r	01110010	454	e	01100101	475	s	01110011
413	a	01100001	434	a	01100001	455	m	01101101	476	sp	00100000
414	r	01110010	435	o	01101111	456	sp	00100000	477	d	01100100
415	a	01100001	436	sp	00100000	457	2	00110010	478	i	01101001
416	c	01100011	437	d	01110000	458	5	00110101	479	s	01110011
417	t	01110100	438	e	01100101	459	6	00110110	480	t	01110100
418	e	01100101	439	sp	00100000	460	sp	00100000	481	i	01101001
419	r	01110010	440	8	00111000	461	p	01110000	482	n	01101110
420	e	01100101	441	sp	00100000	462	a	01100001	483	t	01110100
421	sp	00100000	442	b	01100010	463	d	01110000	484	o	01101111
422	t	01110100	443	i	01101001	464	r	01110010	485	s	01110011
423	e	01100101	444	t	01110100	465	o	01101111	486	.	00101110
424	m	01101101	445	s	01110011	466	e	01100101	487	nul	00000000
425	sp	00100000	446	;	00111011	467	s	01110011	488		
426	s	01110011	447	nl	00001010	468	sp	00100000	489		
427	e	01100101	448	nl	00001010	469	d	01110000	490		

- A tabela ASCII vista abrange o intervalo **[0, 127]**; é a tabela básica

- No intervalo [128, 255], caracteres especiais:

  - Letras acentuadas: **ç, á, ü, etc.**
  - Letras de alfabetos diversos: **α, β, غ, ض, بش, etc.**
  - Símbolos matemáticos: **∞, ∫, ≠, ↔, √, etc.**
  - Figuras especiais; **♥, ♫, ©, ☺, ♣, etc.**

- Esse intervalo tem sido usado de forma **diferente** pelos vários fabricantes de hardware e de software, dificultando a **portabilidade** dos programas

2.7 – Representação de imagens

- A tela de vídeo é dividida retangularmente em um conjunto de minúsculos quadrados chamados ***pixels*** (Picture elements)

- Vídeo em preto e branco

  - **0** para branco e **1** para preto

- Vídeo colorido - Padrão RBG (Red Blue Green)

  - Diferentes cores são obtidas a partir de intensidades diferentes dessas **três cores primárias**

  - Um **pixel** é armazenado em **3 bytes** (1 byte por cor básica)

  - **Exemplo:** imagem com 1024 linhas de 1024 pixels ocupará **3 Megabytes**

  - Existem técnicas de **compressão de dados** para reduzir o **gasto excessivo** de memória

2.8 – Representação de instruções

- Um **programa** é um conjunto de **instruções**
- Essas instruções devem estar armazenadas na RAM, para que sejam executadas pela CPU
- Na RAM devem ficar também os **dados** (números, textos, imagens) a serem manipulados por essas instruções
- É necessária **cuidadosa programação** para que a CPU não tome **dado por instrução** e vice-versa

- Uma instrução é dividida tipicamente em duas partes:
  - **Código da operação**, que é a especificação da operação a ser realizada pela instrução
  - Informações sobre os **operandos**
- Há instruções com **zero**, **um**, **dois** ou até **mais** operandos
- Para cada operando, a informação pode ser o **próprio** ou o seu **endereço**

Classes de instruções:

- **Entrada e saída de dados:** comunicação com o meio externo ao computador ou com sua memória secundária, para receber e/ou enviar informações

- **Movimentação de dados:** transporte de dados entre a RAM e os registradores da CPU e carregamento de constantes em um desses módulos de memória

Classes de instruções:

- **Operações aritméticas e lógicas:** cálculos entre conteúdos de palavras do sistema de memória e/ou constantes especificadas na própria instrução

- **Controle do fluxo de execução:** alteração condicional ou incondicional da sequência de instruções

- O **espaço ocupado** por uma instrução depende:
  - Da **natureza** da operação a ser realizada
  - Do **número** de operandos
  - Da forma de **referenciá-los**
  - Da **arquitetura** do computador
- Há instruções que ocupam apenas **1 byte**; outras ocupam **dois ou mais bytes**

Computadores, Algoritmos e Programas: Manual Introdutório de Ensino

Capítulo III

Evolução das Linguagens de Programação

Capítulo III – Evolução das Linguagens de Programação

3.1 – A linguagem de máquina

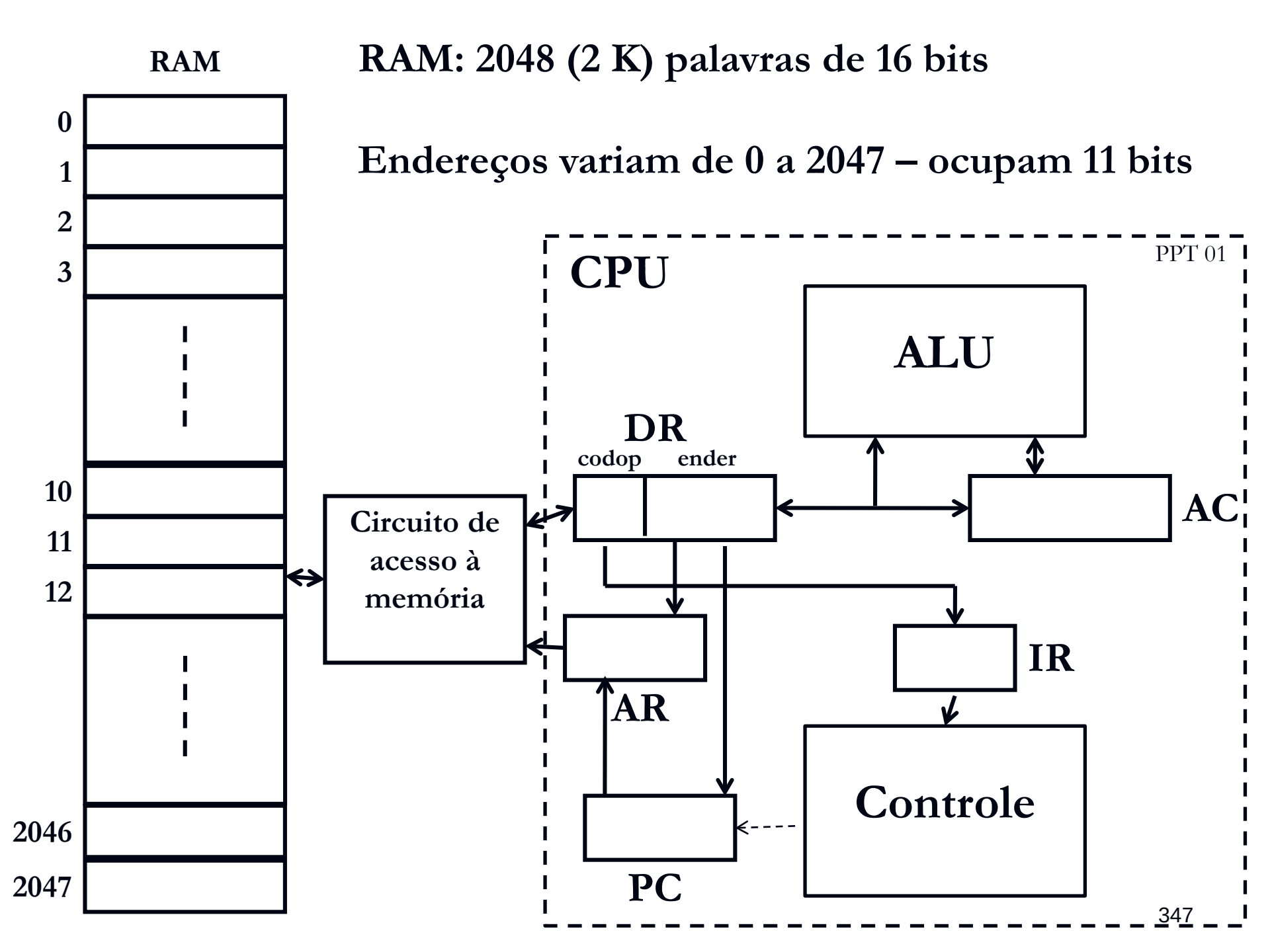
3.2 – Linguagem Assembly

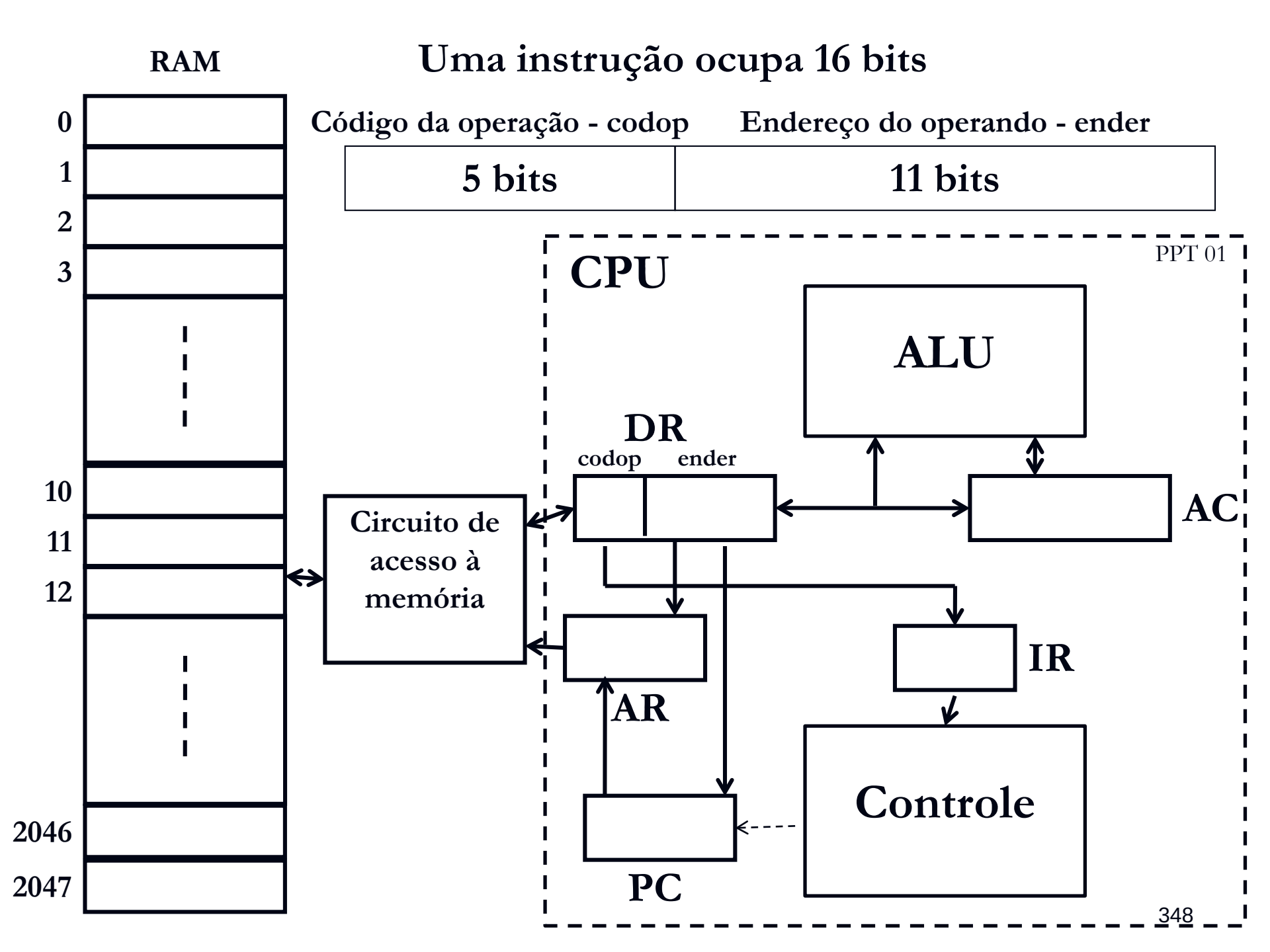
3.3 – Linguagens tradicionais de programação

3.4 – Linguagens aplicativas

3.1 – A linguagem de máquina

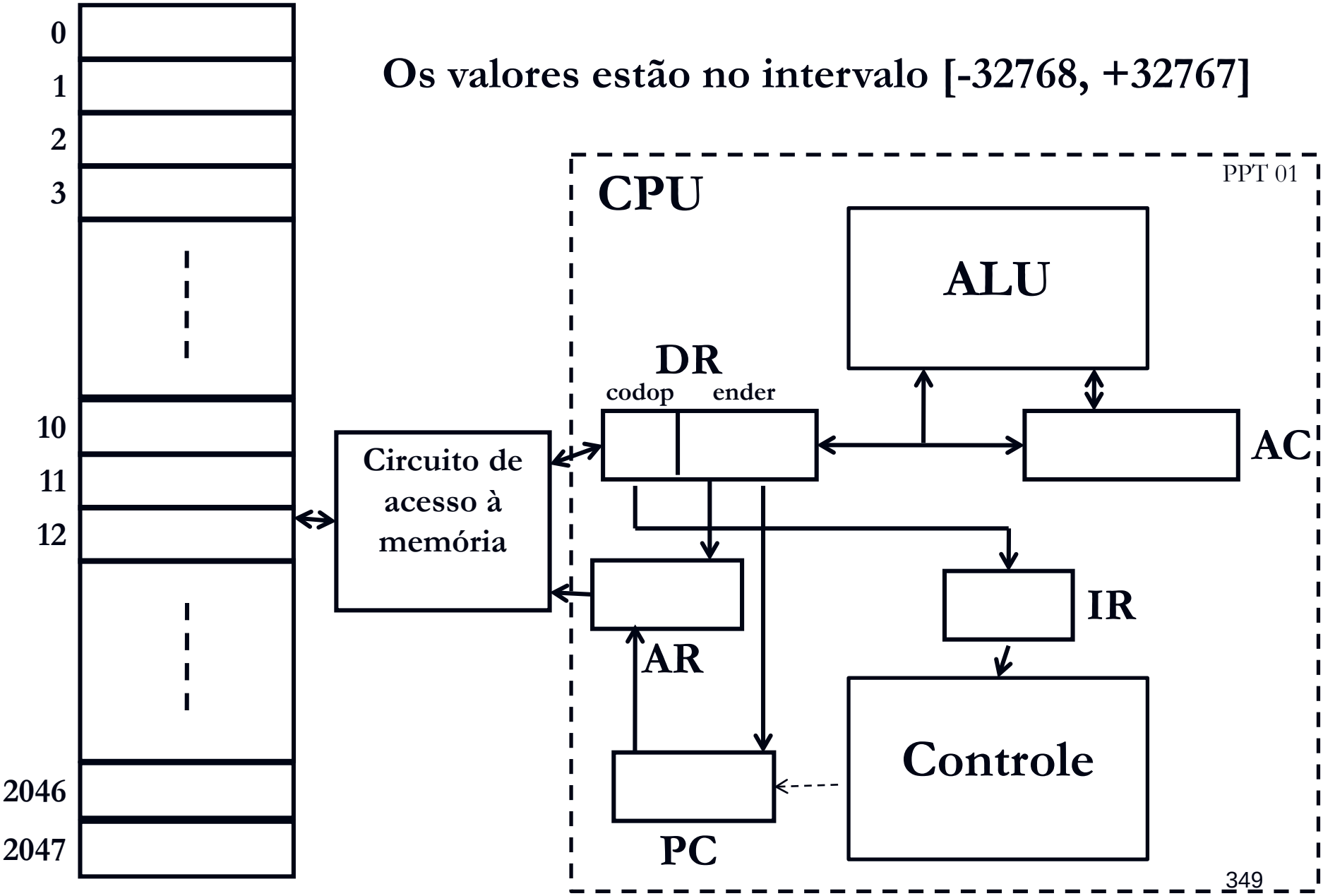
- **Linguagem de máquina:** conjunto de instruções que o hardware de um computador é capaz de realizar
- Cada instrução é expressa por uma sequência de bits
- Seja um computador bem primitivo mostrado a seguir


RAM
RAM: 2048 (2 K) palavras de 16 bits
Endereços variam de 0 a 2047 – ocupam 11 bits
0
1
2
3
10
11
12
2046
2047
CPU
PPT 01
ALU
DR
codop
ender
AC
Circuito de acesso à memória
IR
AR
Controle
PC
347


RAM
Uma instrução ocupa 16 bits
0
1
2
3
10
11
12
2046
2047
Código da operação - codop
Endereço do operando - ender
5 bits
11 bits
CPU
PPT 01
ALU
DR
codop
ender
AC
Circuito de acesso à memória
IR
AR
Controle
PC
348

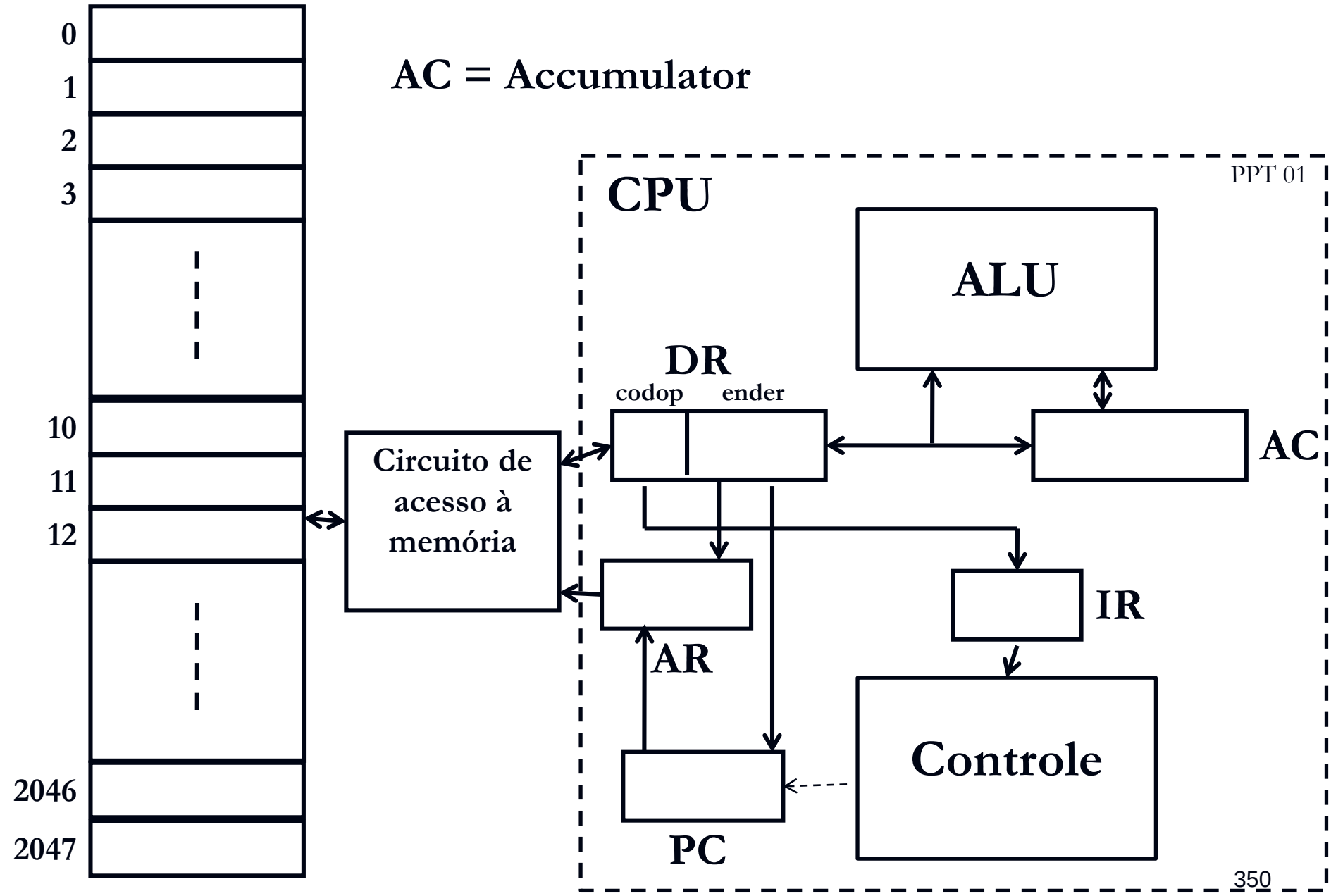
Os dados são inteiros em comp-2 de 16 bits

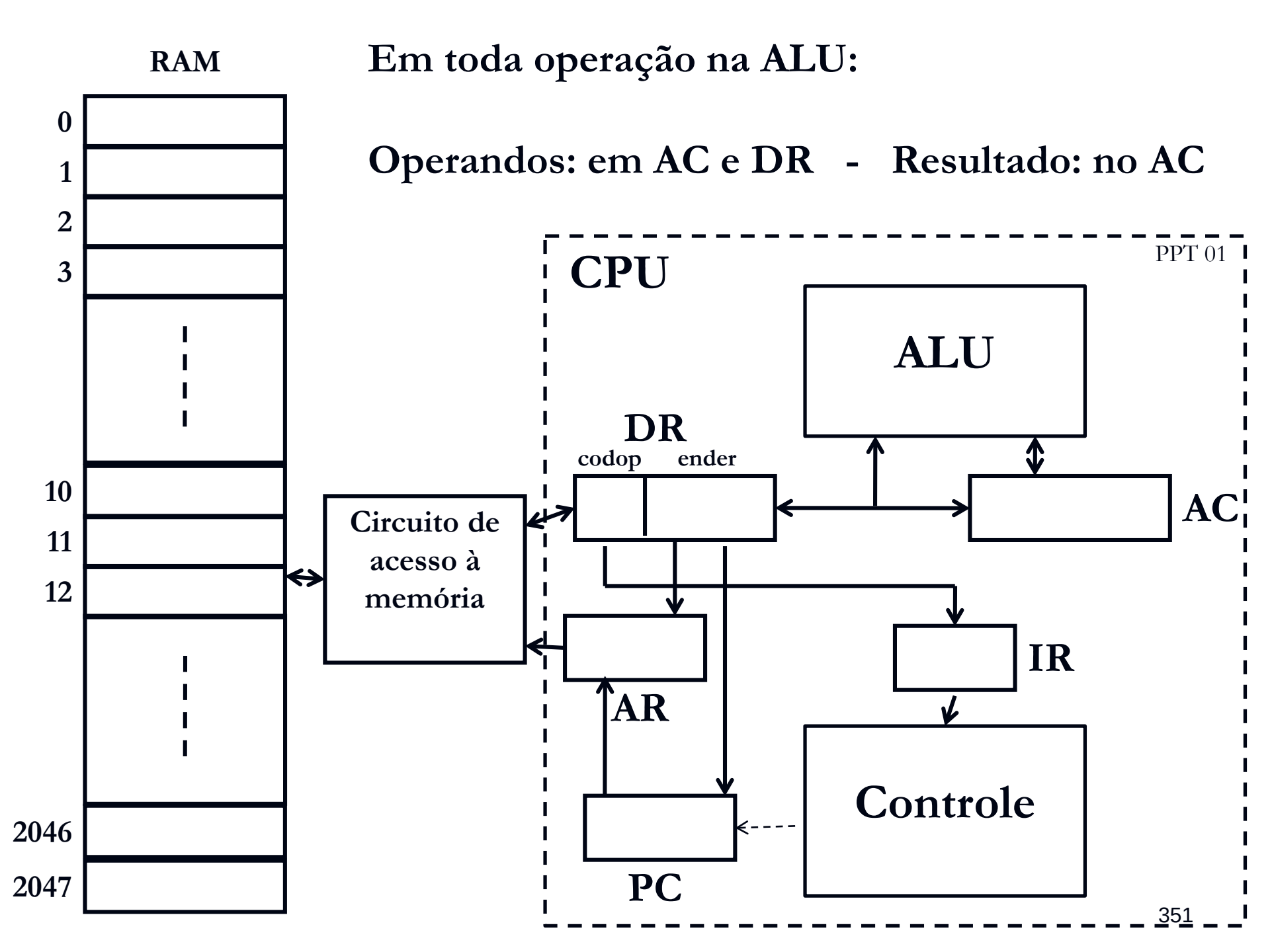
Os valores estão no intervalo [-32768, +32767]



Registrador de propositos gerais: AC (16 bits)

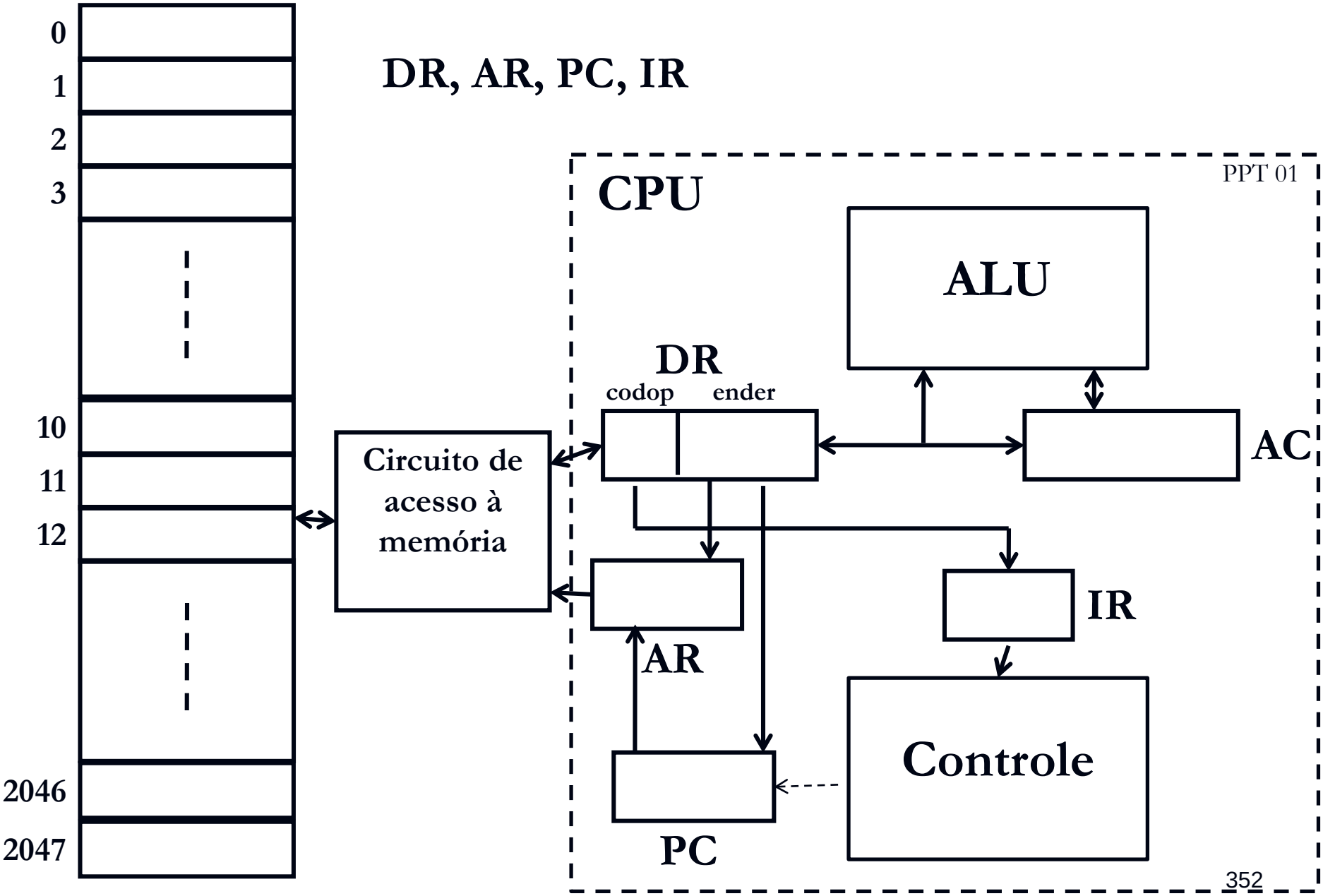
AC = Accumulator

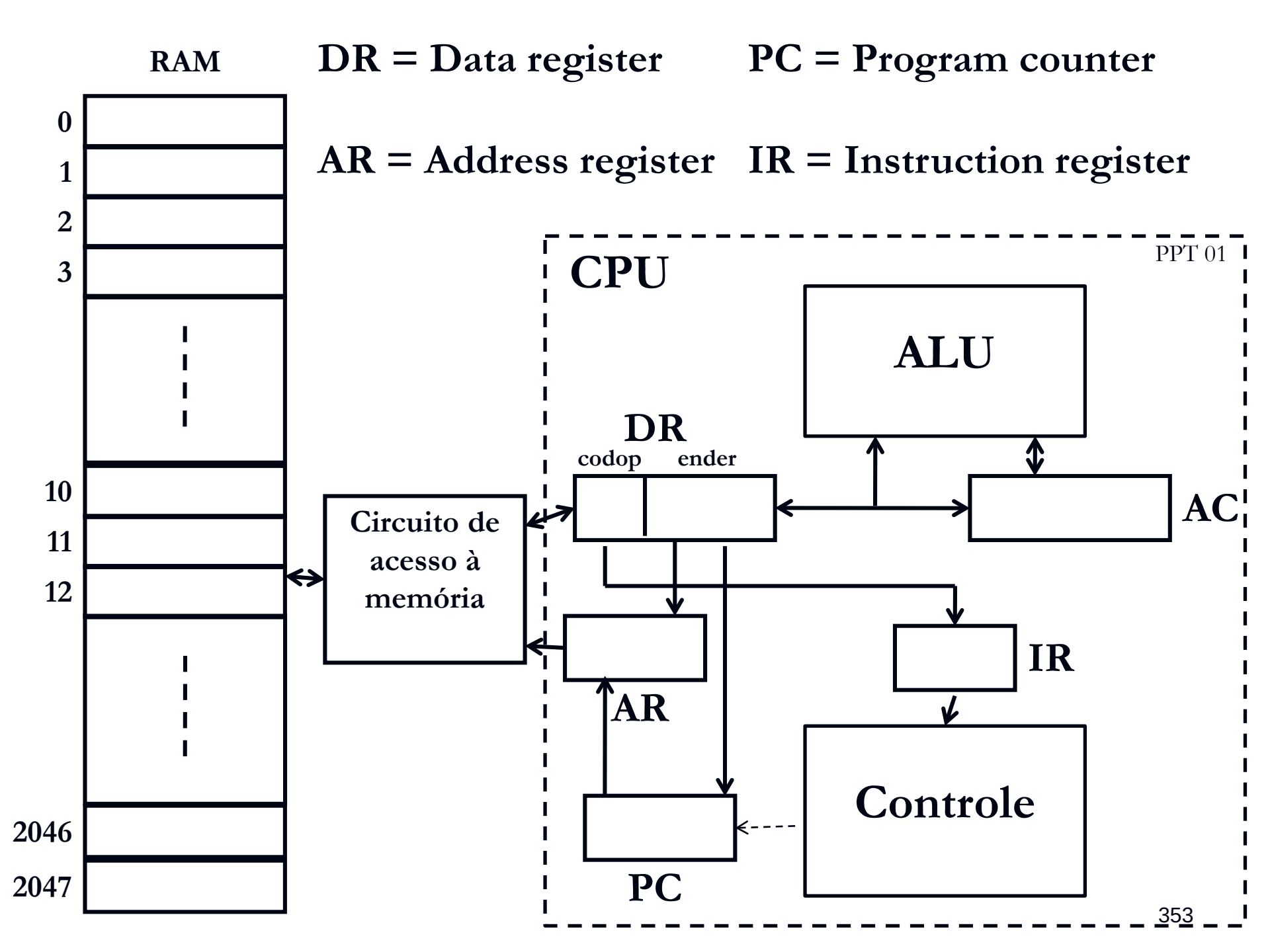



RAM
Em toda operação na ALU:
Operandos: em AC e DR - Resultado: no AC
0
1
2
3
10
11
12
2046
2047
CPU
PPT 01
ALU
DR
codop
ender
AC
Circuito de acesso à memória
IR
AR
Controle
PC
351

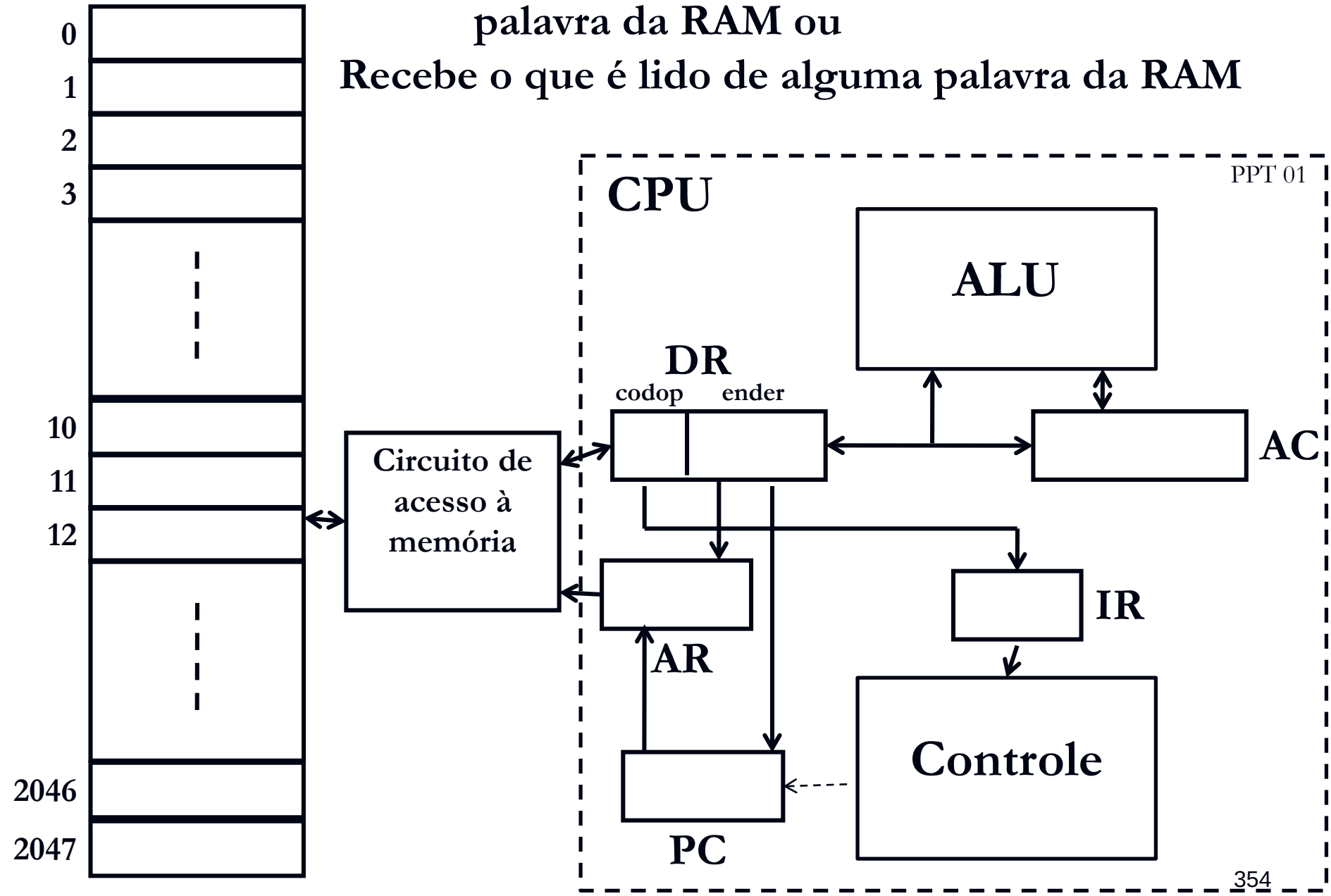
Registradores de propósitos específicos:

DR, AR, PC, IR

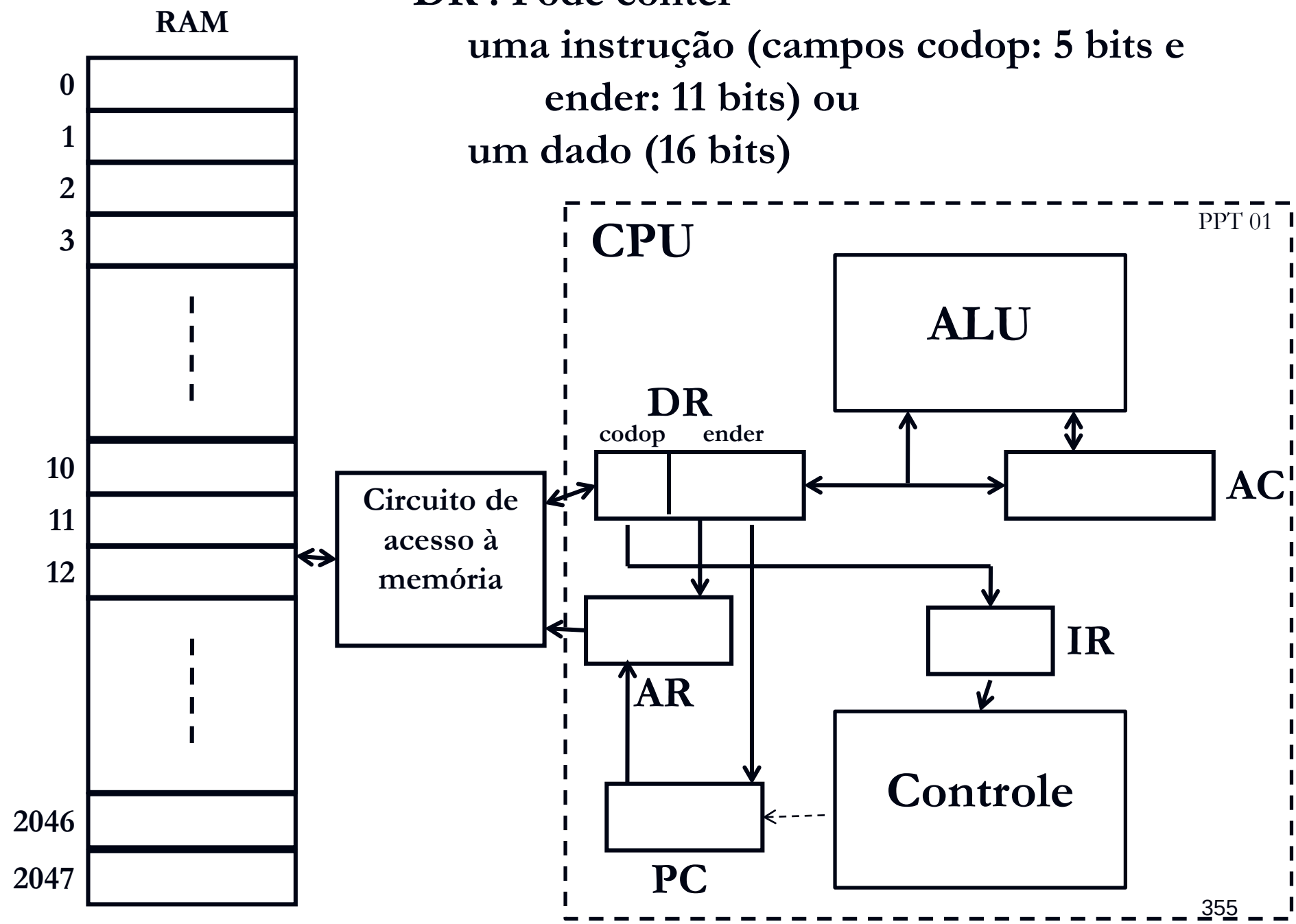



RAM
DR = Data register
PC = Program counter
AR = Address register
IR = Instruction register
0
1
2
3
10
11
12
2046
2047
CPU
PPT 01
ALU
DR
codop
ender
AC
Circuito de acesso à memória
IR
AR
Controle
PC
353

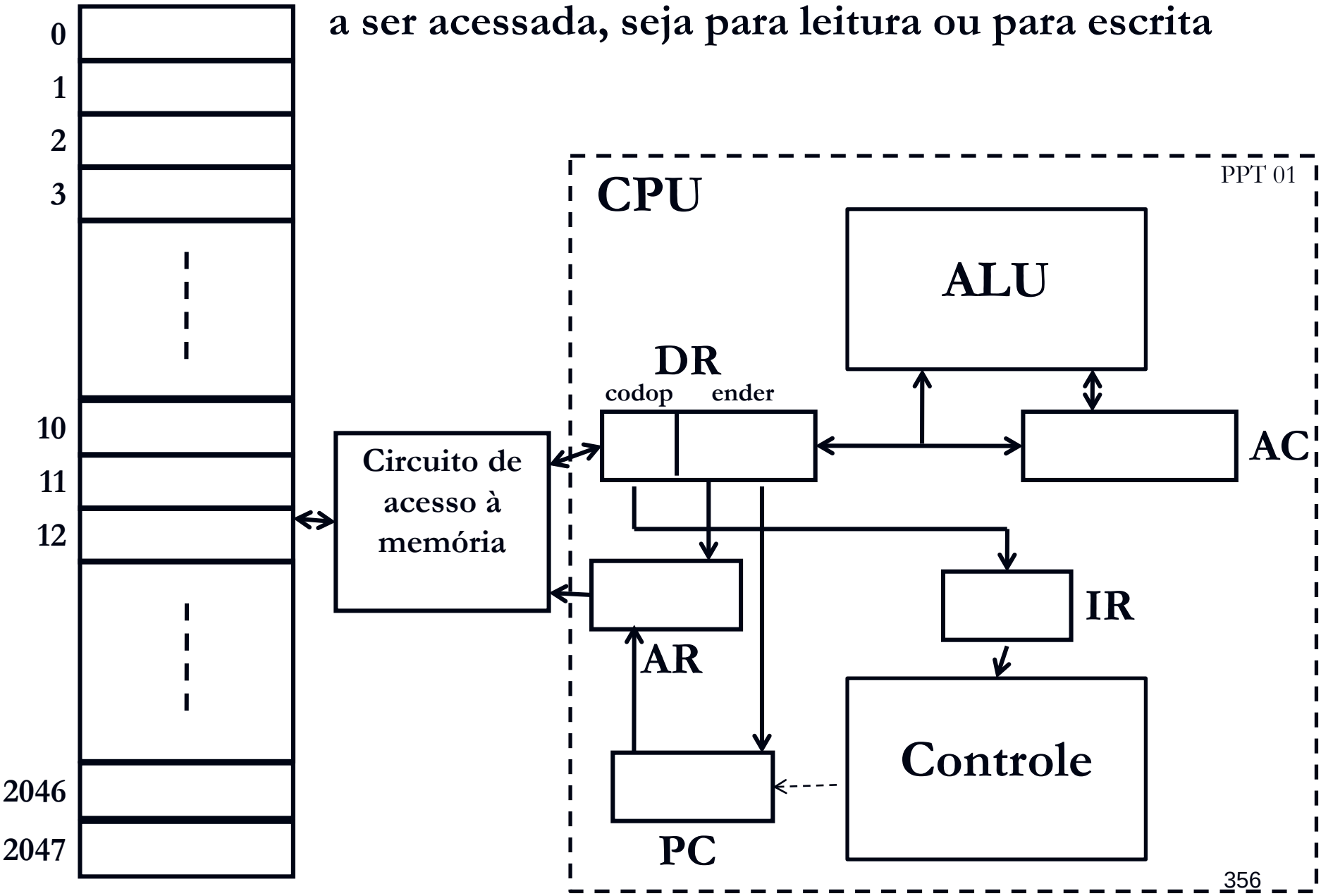
DR (16 bits): contém o que será escrito em alguma palavra da RAM ou
Recebe o que é lido de alguma palavra da RAM



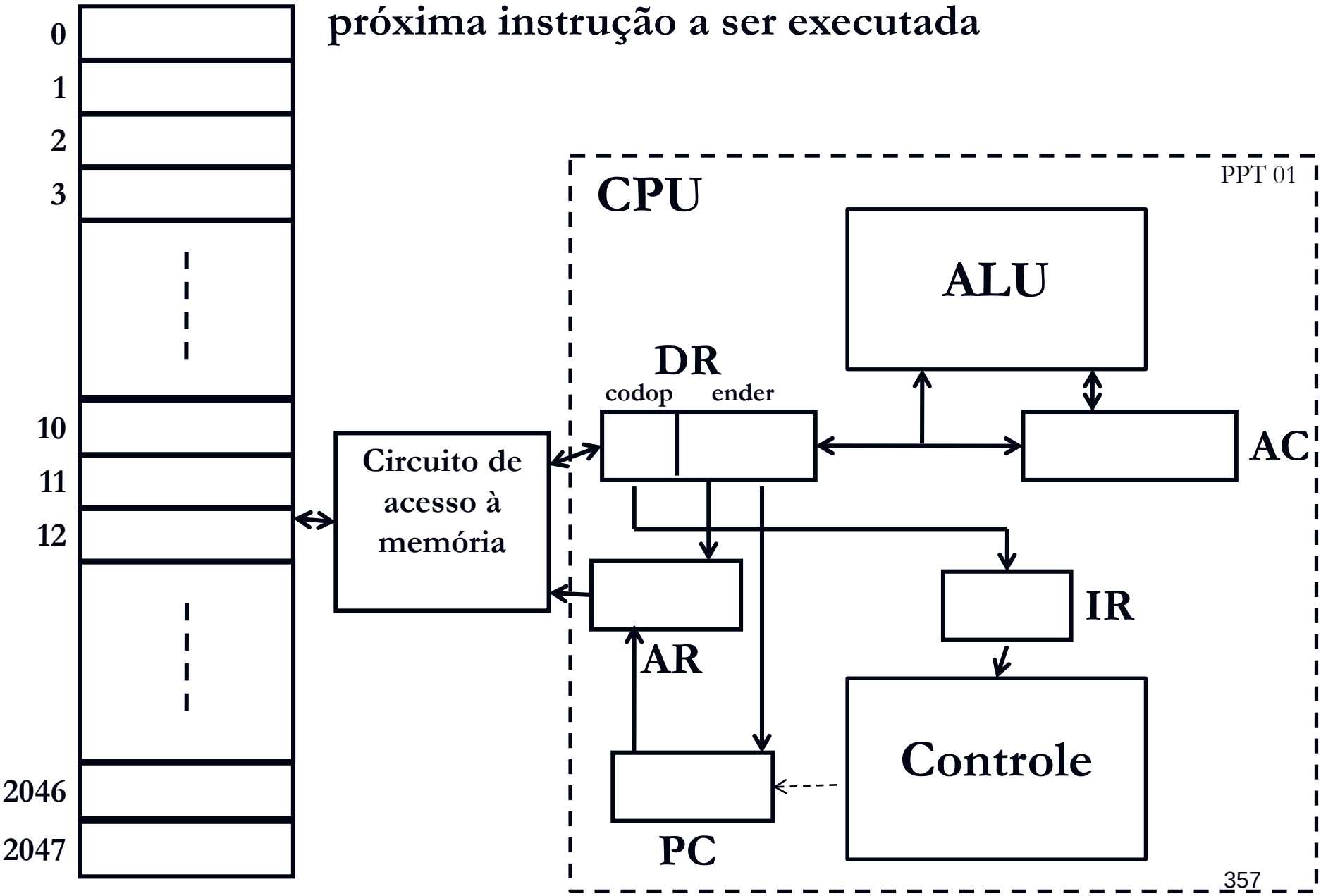
DR : Pode conter
uma instrução (campos codop: 5 bits e ender: 11 bits) ou
um dado (16 bits)



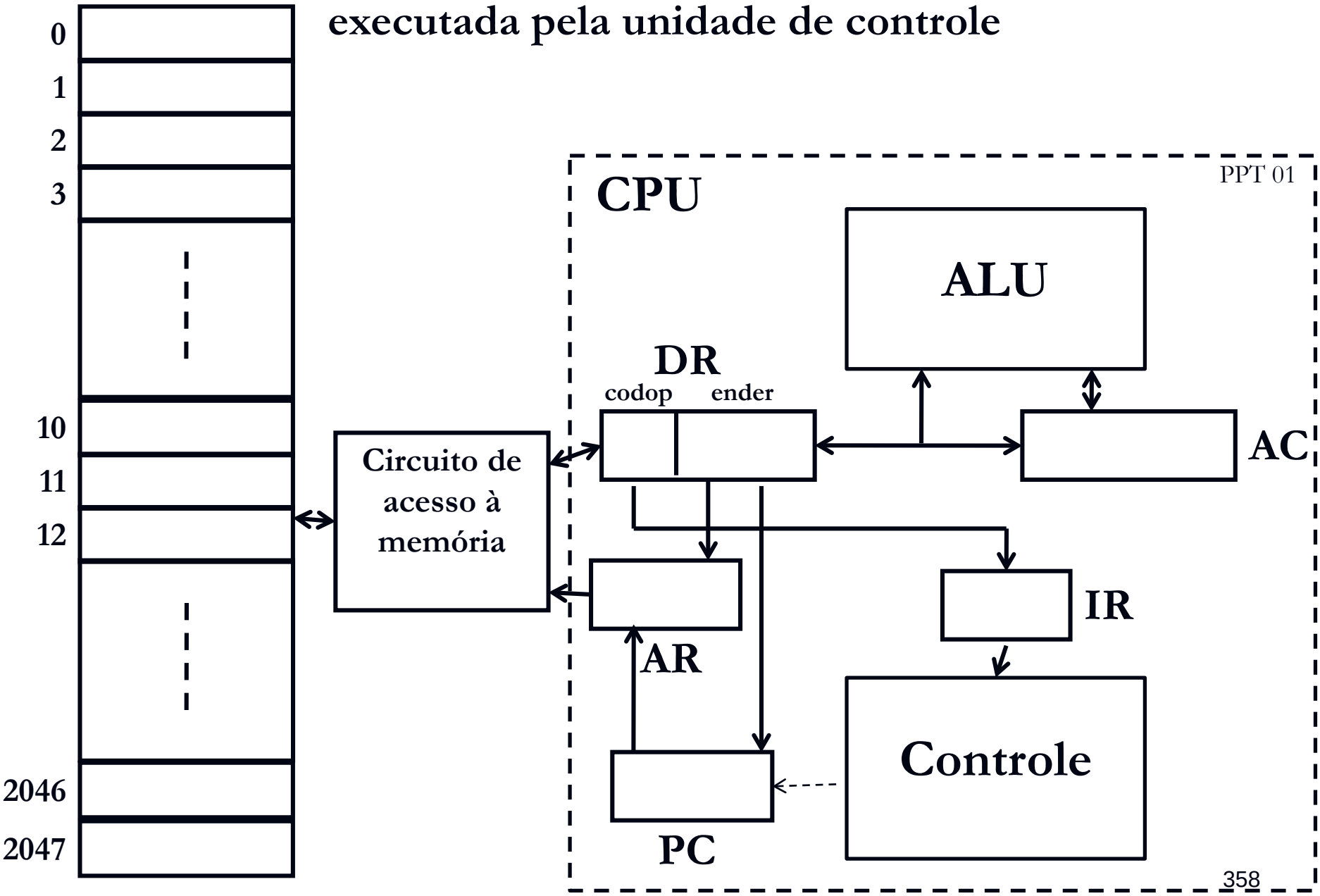
AR (11 bits): contém o endereço da palavra da RAM a ser acessada, seja para leitura ou para escrita



PC (11 bits): contém o endereço na RAM da próxima instrução a ser executada



IR (5 bits): contém o código da instrução a ser executada pela unidade de controle



Conjunto de instruções primitivo:

Nome	Mnemô-nico	Código em binário	Código em decimal	Ação realizada
Carregar no AC	LD	00001	1	AC ← Mem (Ender)
Armazenar AC	ST	00010	2	Mem (Ender) ← AC
Zerar memória	STZ	00011	3	Mem (Ender) ← 0
Somar ao AC	ADD	00100	4	AC ← AC + Mem (Ender)
Subtrair do AC	SUB	00101	5	AC ← AC - Mem (Ender)
Multiplicar pelo AC	MULT	00110	6	AC ← AC * Mem (Ender)
Dividir AC	DIV	00111	7	AC ← AC / Mem (Ender)
And no AC	AND	01000	8	AC ← AC and Mem (Ender)
Or no AC	OR	01001	9	AC ← AC or Mem (Ender)
Not AC	NOT	01010	10	AC ← not AC
Desviar incondicionalmente	JUMP	01011	11	Desviar para Mem (Ender)
Desviar se AC < 0	JN	01100	12	Se AC < 0, desviar para Mem (Ender)
Desviar se AC = 0	JZ	01101	13	Se AC = 0, desviar para Mem (Ender)
Desviar se AC > 0	JP	01110	14	Se AC > 0, desviar para Mem (Ender)
Leitura de inteiro	READ	01111	15	Ler (Mem (Ender))
Escrita de inteiro	WRITE	10000	16	Escrever (Mem (Ender))
Parar a execução	STOP	10001	17	Encerrar a execução desligando a máquina

Código da operação - codop	Endereço do operando - ender
5 bits	11 bits

Mem (Ender):

Palavra da RAM cujo endereço está no campo ender da instrução

Nome	Mnemônico	Código em binário	Código em decimal	Ação realizada
Carregar no AC	LD	00001	1	AC ← Mem (Ender)
Armazenar AC	ST	00010	2	Mem (Ender) ← AC
Zerar memória	STZ	00011	3	Mem (Ender) ← 0
Somar ao AC	ADD	00100	4	AC ← AC + Mem (Ender)
Subtrair do AC	SUB	00101	5	AC ← AC - Mem (Ender)
Multiplicar pelo AC	MULT	00110	6	AC ← AC * Mem (Ender)
Dividir AC	DIV	00111	7	AC ← AC / Mem (Ender)
And no AC	AND	01000	8	AC ← AC and Mem (Ender)
Or no AC	OR	01001	9	AC ← AC or Mem (Ender)
Not AC	NOT	01010	10	AC ← not AC
Desviar incondicionalmente	JUMP	01011	11	Desviar para Mem (Ender)
Desviar se AC < 0	JN	01100	12	Se AC < 0, desviar para Mem (Ender)
Desviar se AC = 0	JZ	01101	13	Se AC = 0, desviar para Mem (Ender)
Desviar se AC > 0	JP	01110	14	Se AC > 0, desviar para Mem (Ender)
Leitura de inteiro	READ	01111	15	Ler (Mem (Ender))
Escrita de inteiro	WRITE	10000	16	Escrever (Mem (Ender))
Parar a execução	STOP	10001	17	Encerrar a execução desligando a máquina

- Seja um programa para fazer:

Mem(12) ← Mem(10) + Mem(11)

PPT 01

Endereço	Conteúdo em binário	Conteúdo em decimal	Ação
0	00001 00000001010	1 10	AC ← Mem(10)
1	00100 00000001011	4 11	AC ← AC +Mem(11)
2	00010 00000001100	2 12	Mem(12) ← AC

- Seja o seguinte conteúdo das palavras **10** e **11** da RAM:

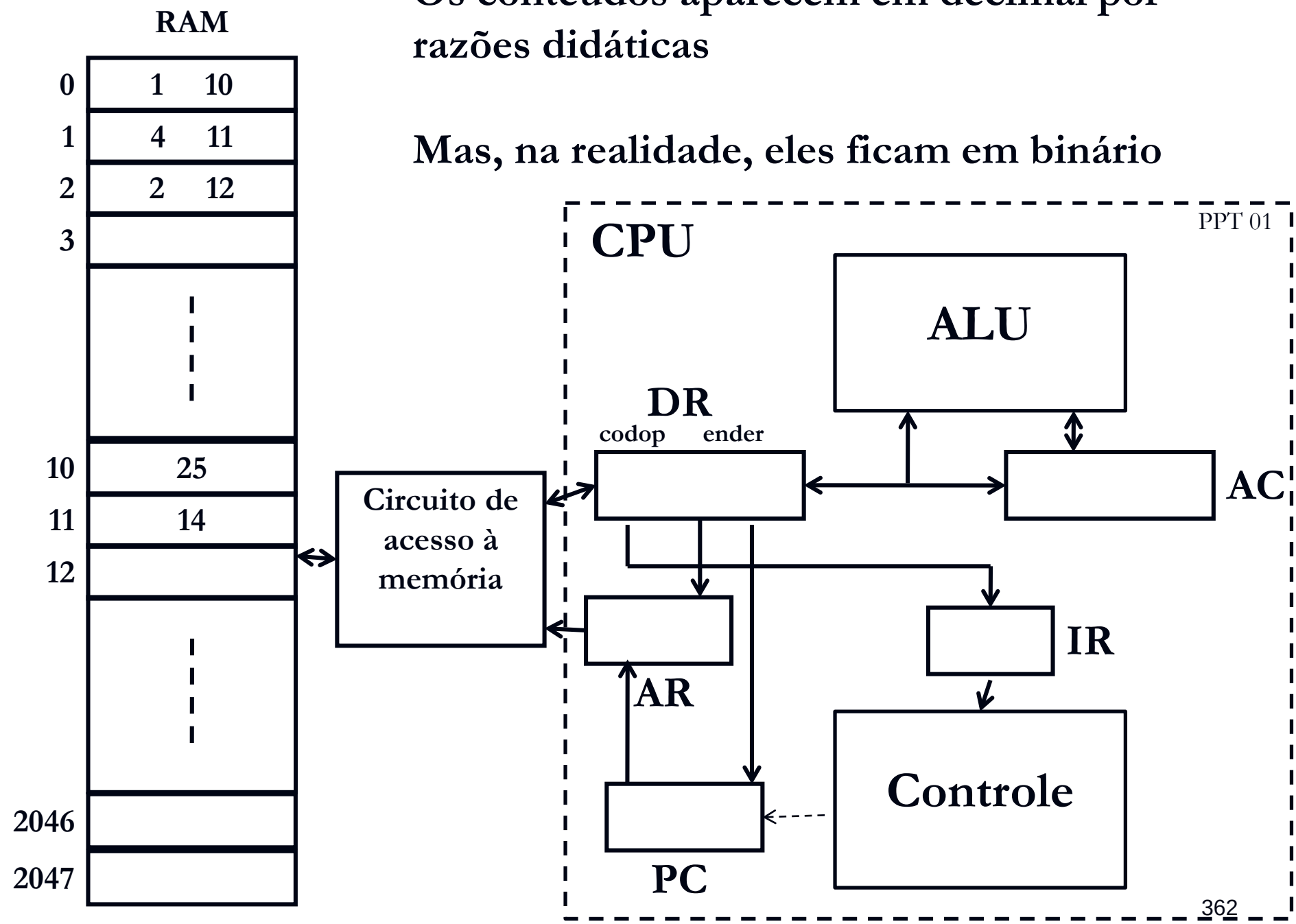
PPT 01

Endereço	Conteúdo em binário	Conteúdo em decimal
10	0000000000011001	25
11	0000000000001110	14

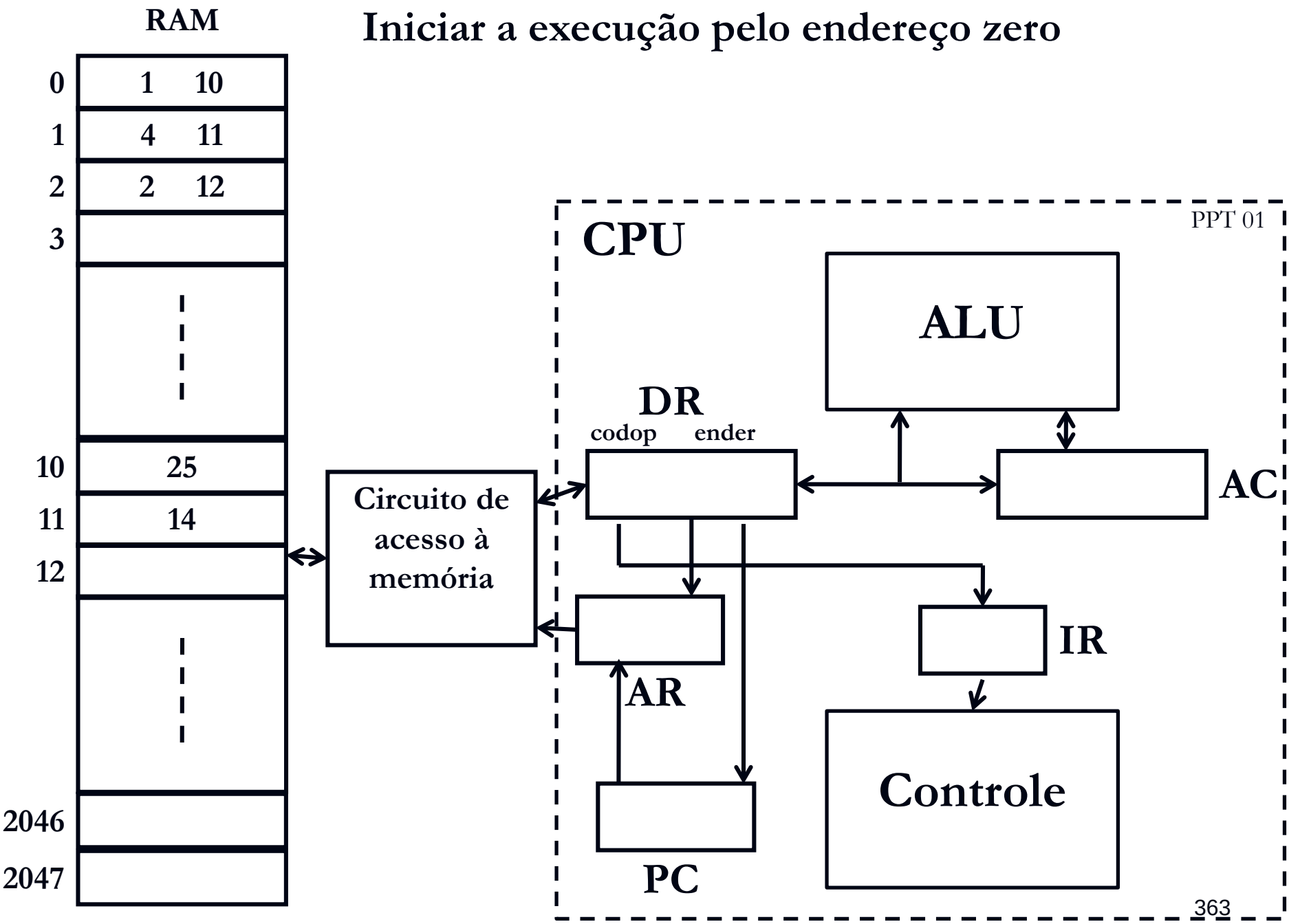
Seja tudo isso guardado na RAM do computador

Os conteúdos aparecem em decimal por razões didáticas

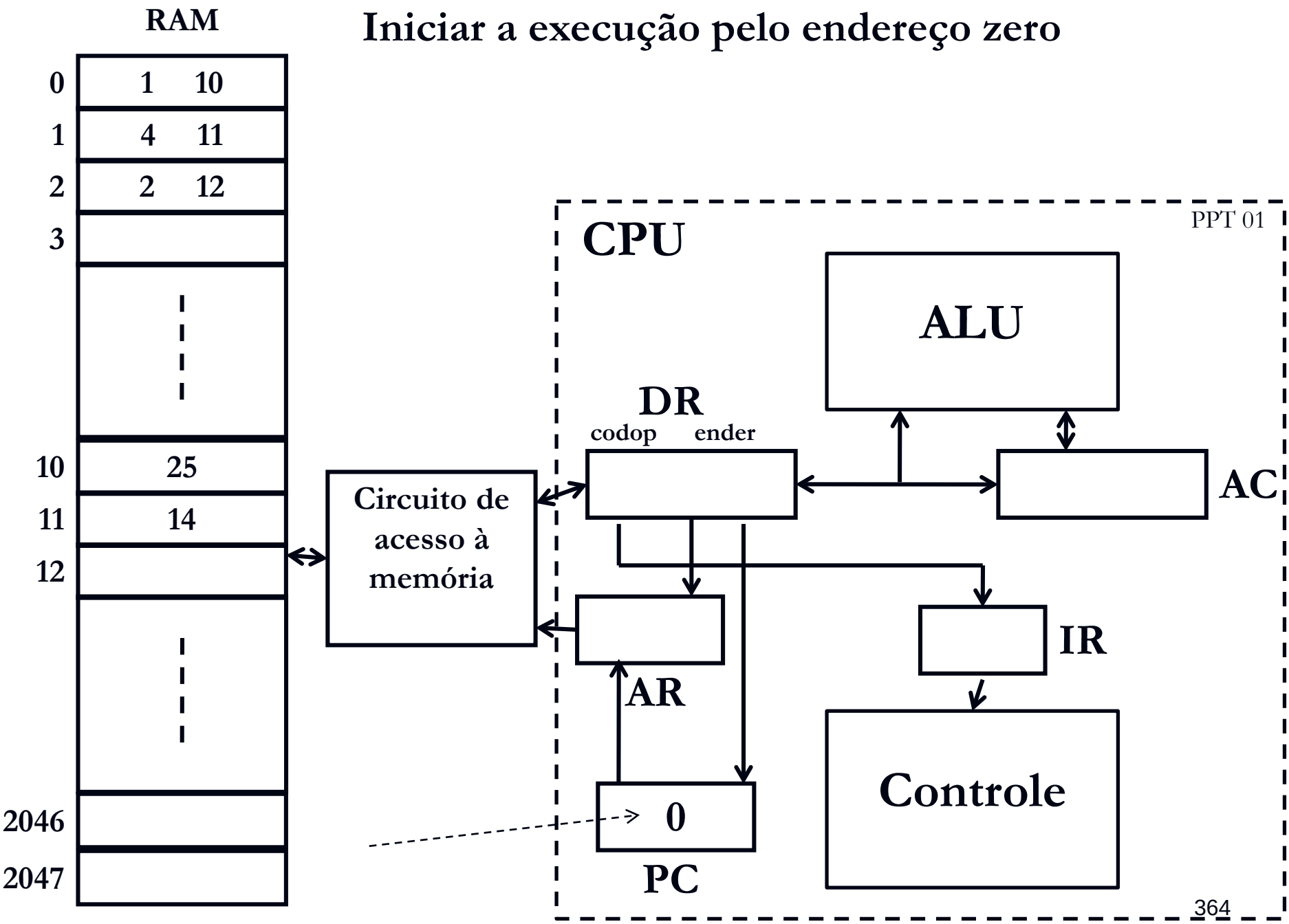
Mas, na realidade, eles ficam em binário

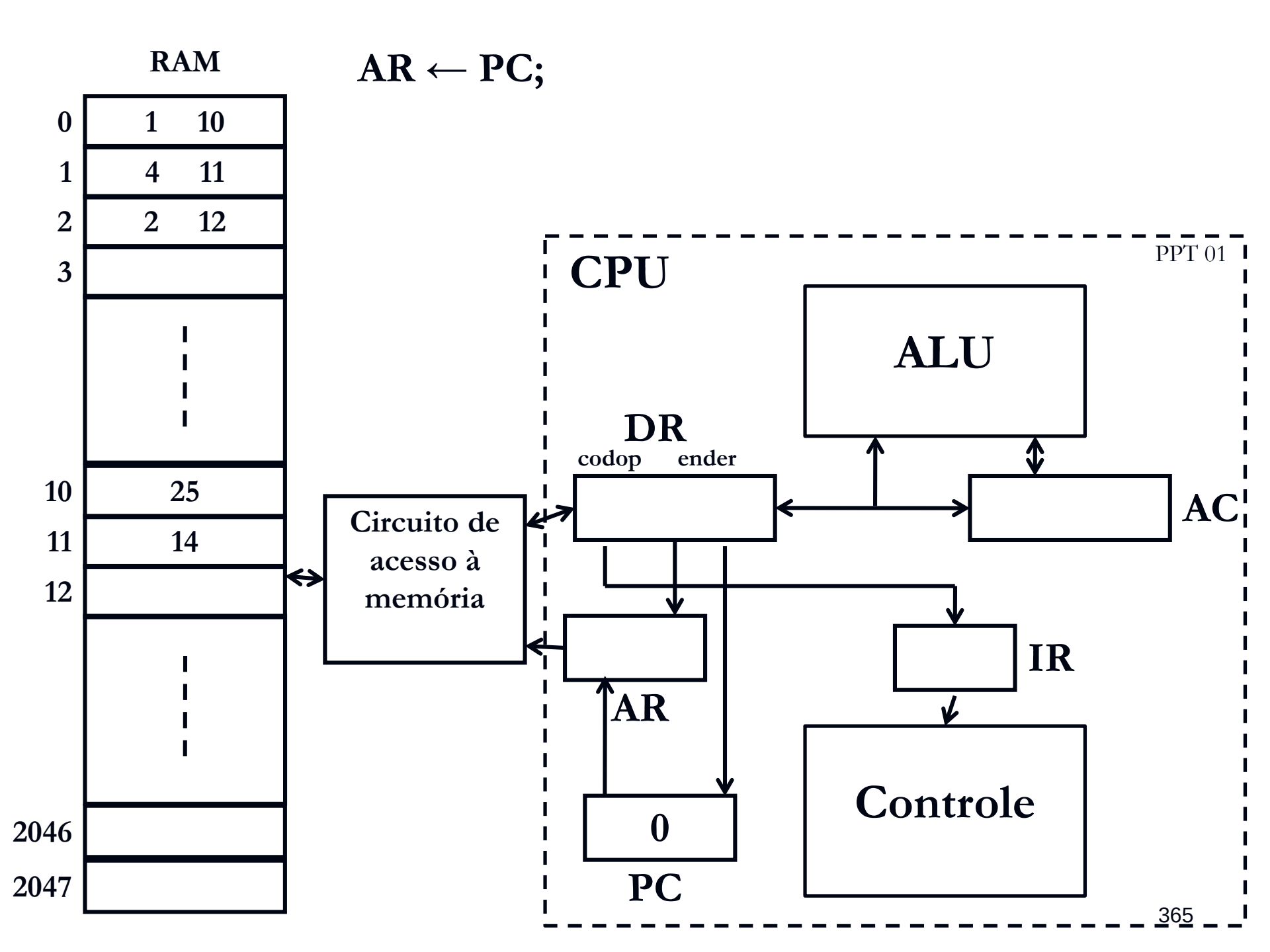


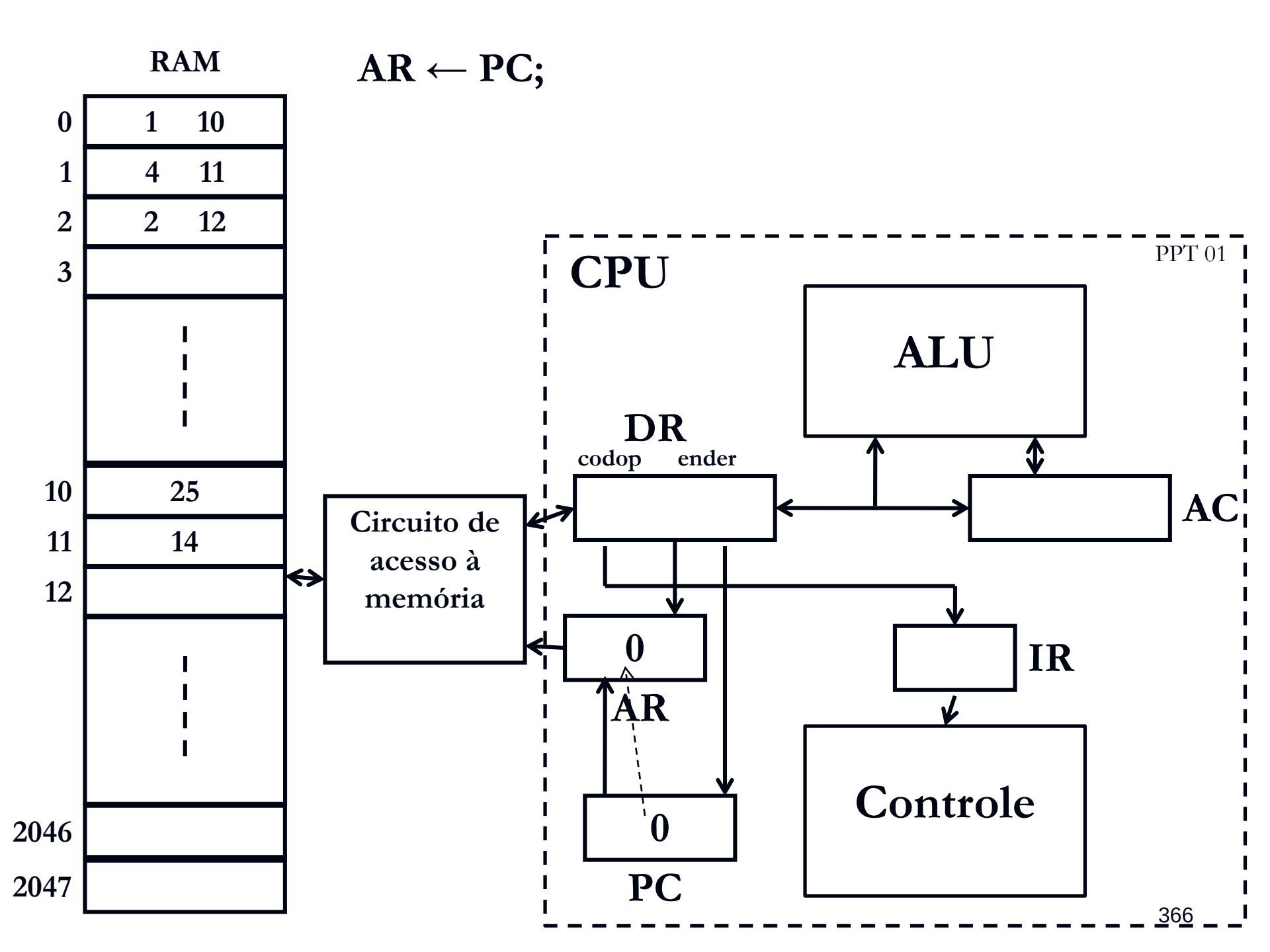
Iniciar a execução pelo endereço zero

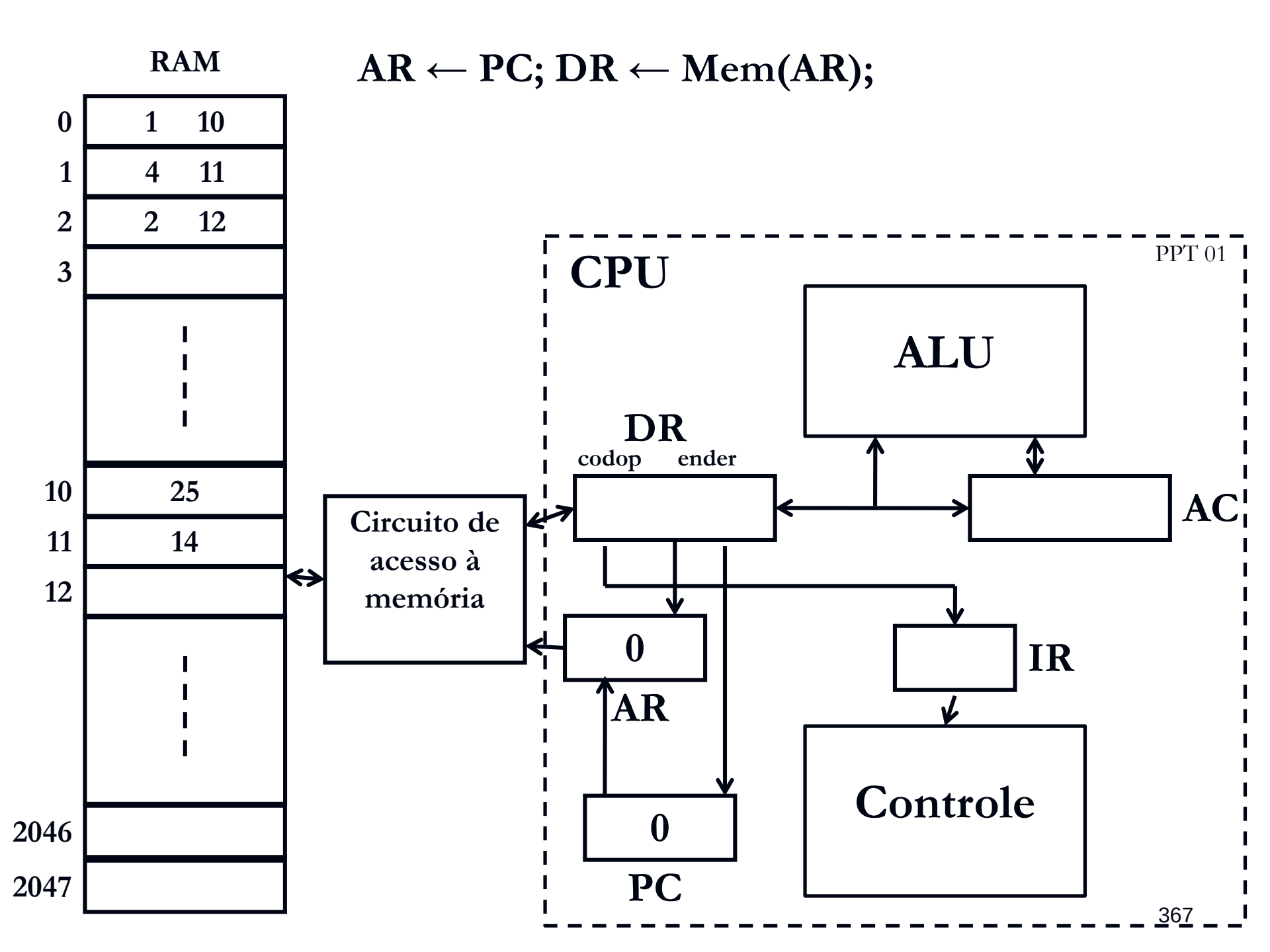


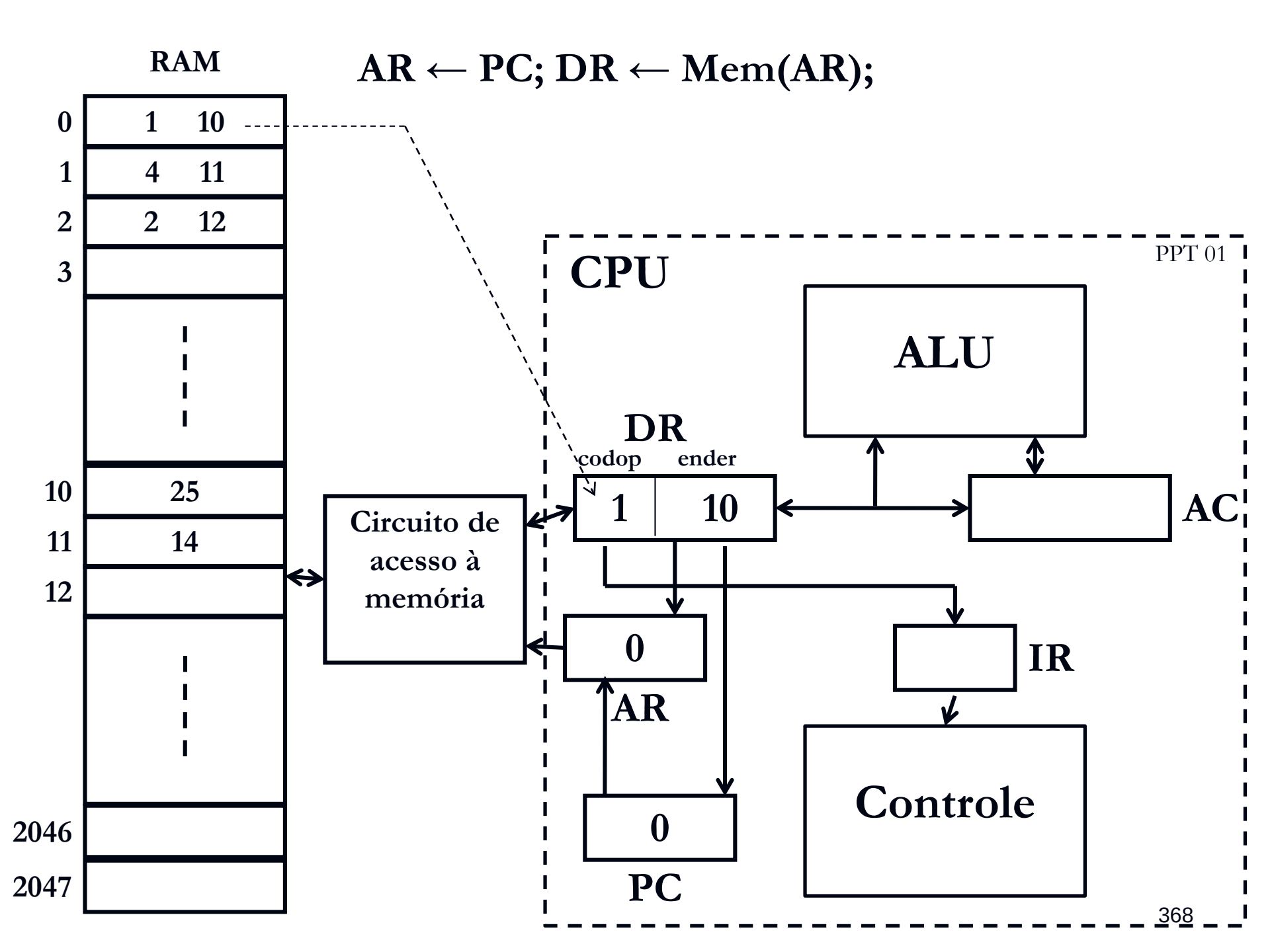
Iniciar a execução pelo endereço zero

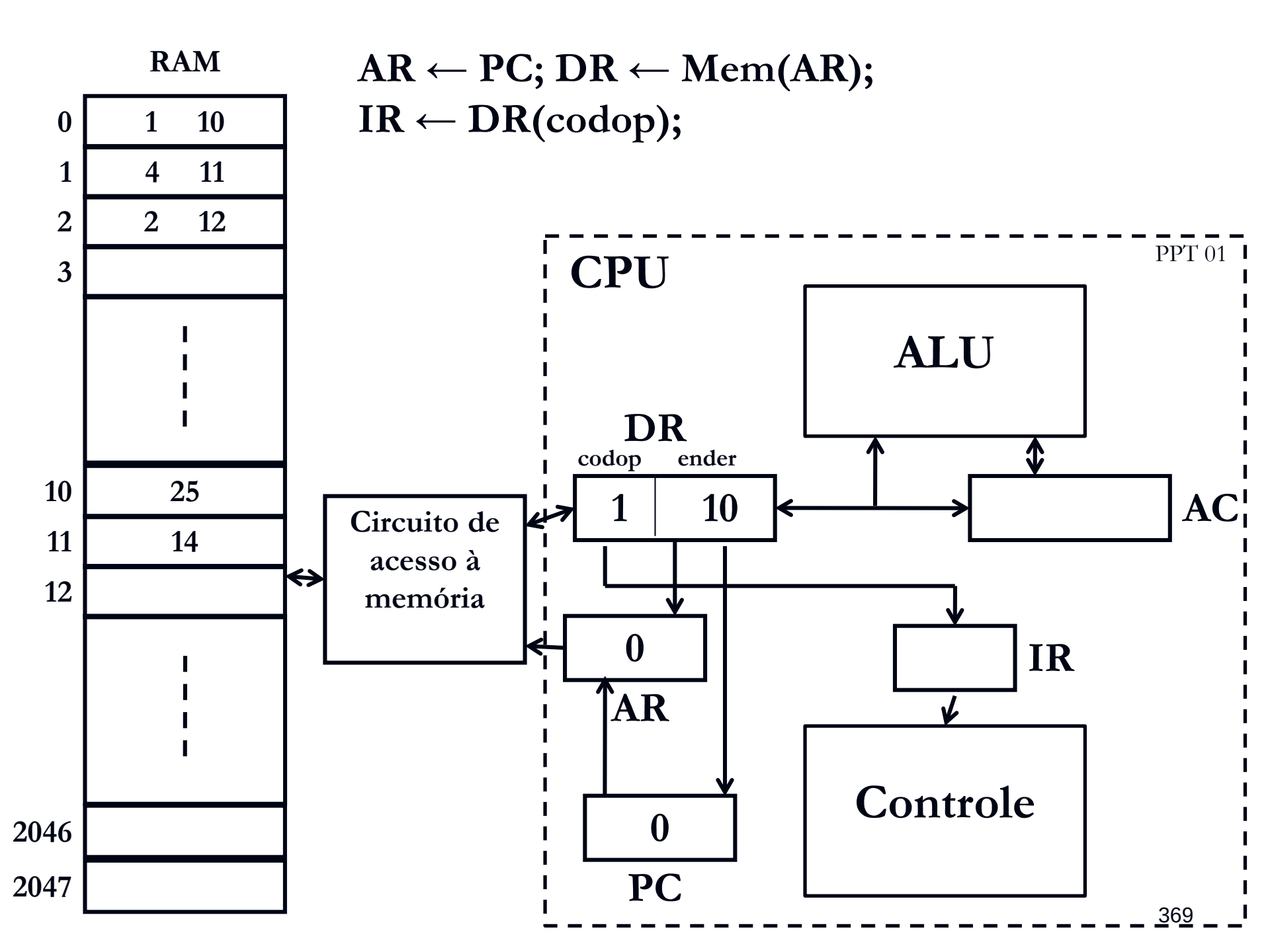


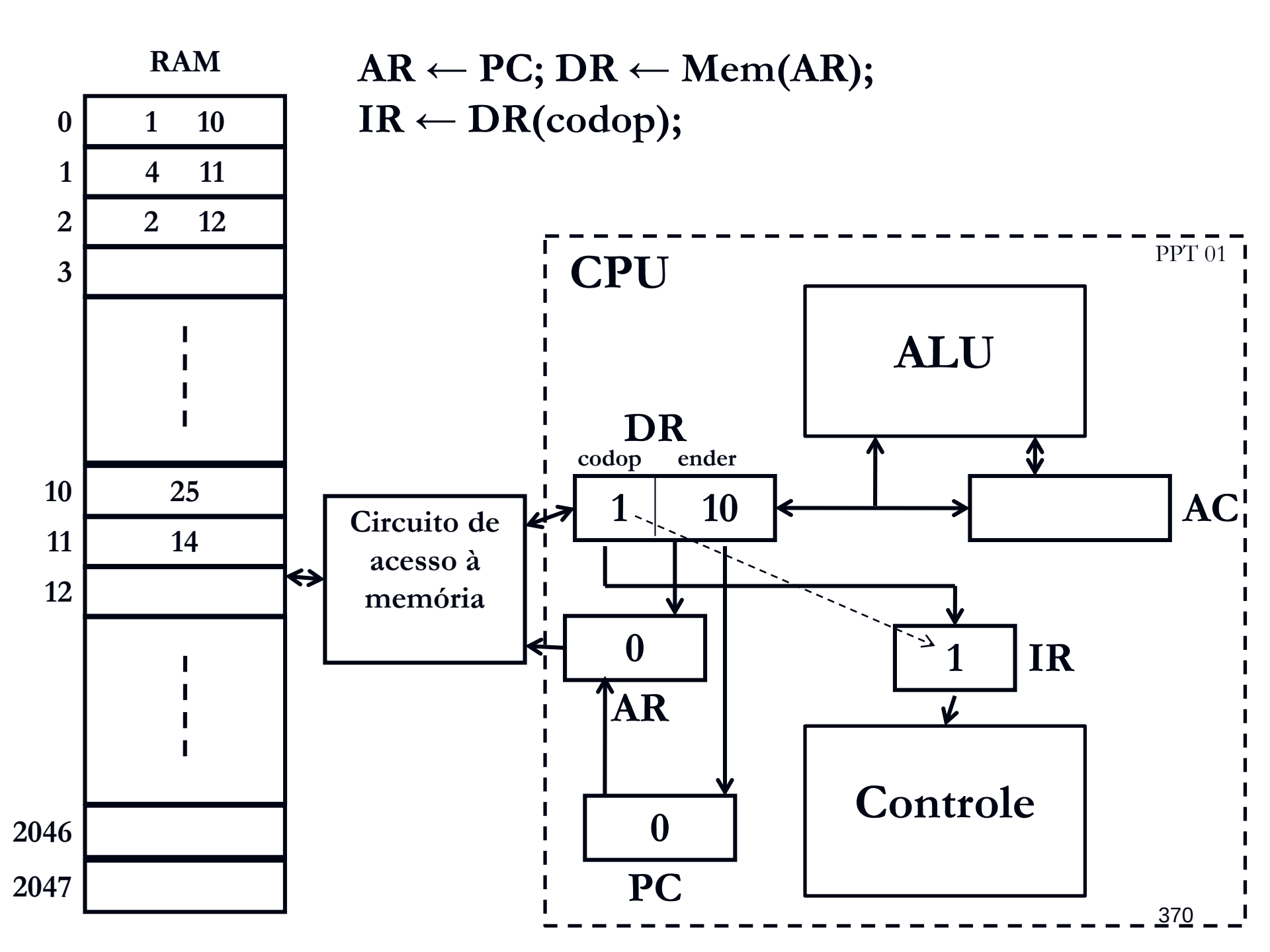

RAM
AR ← PC;
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
DR
codop
ender
AC
IR
AR
0
PC
Controle
365

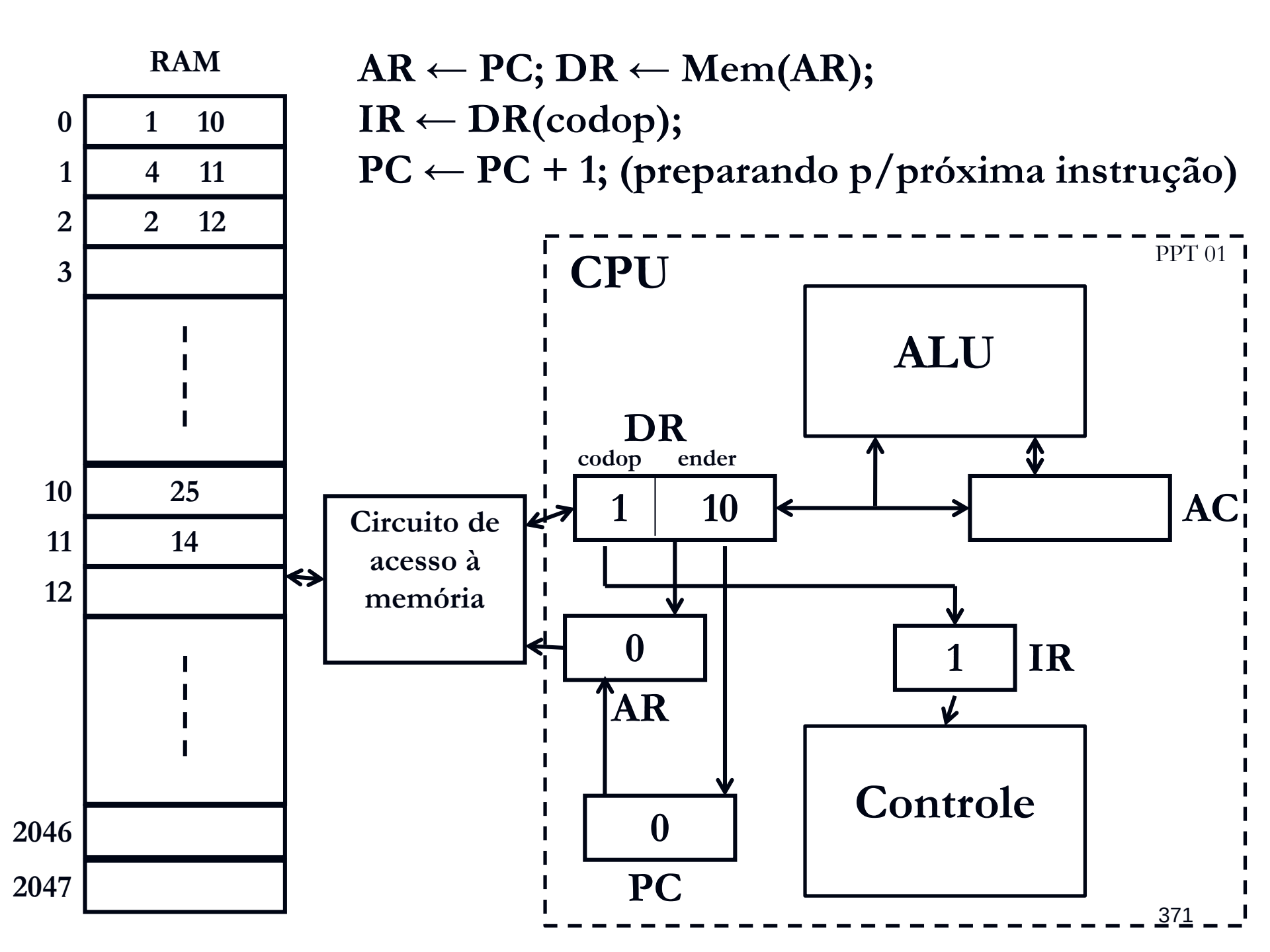

RAM
AR ← PC;
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
DR
codop
ender
AC
0
AR
IR
Controle
0
PC
366

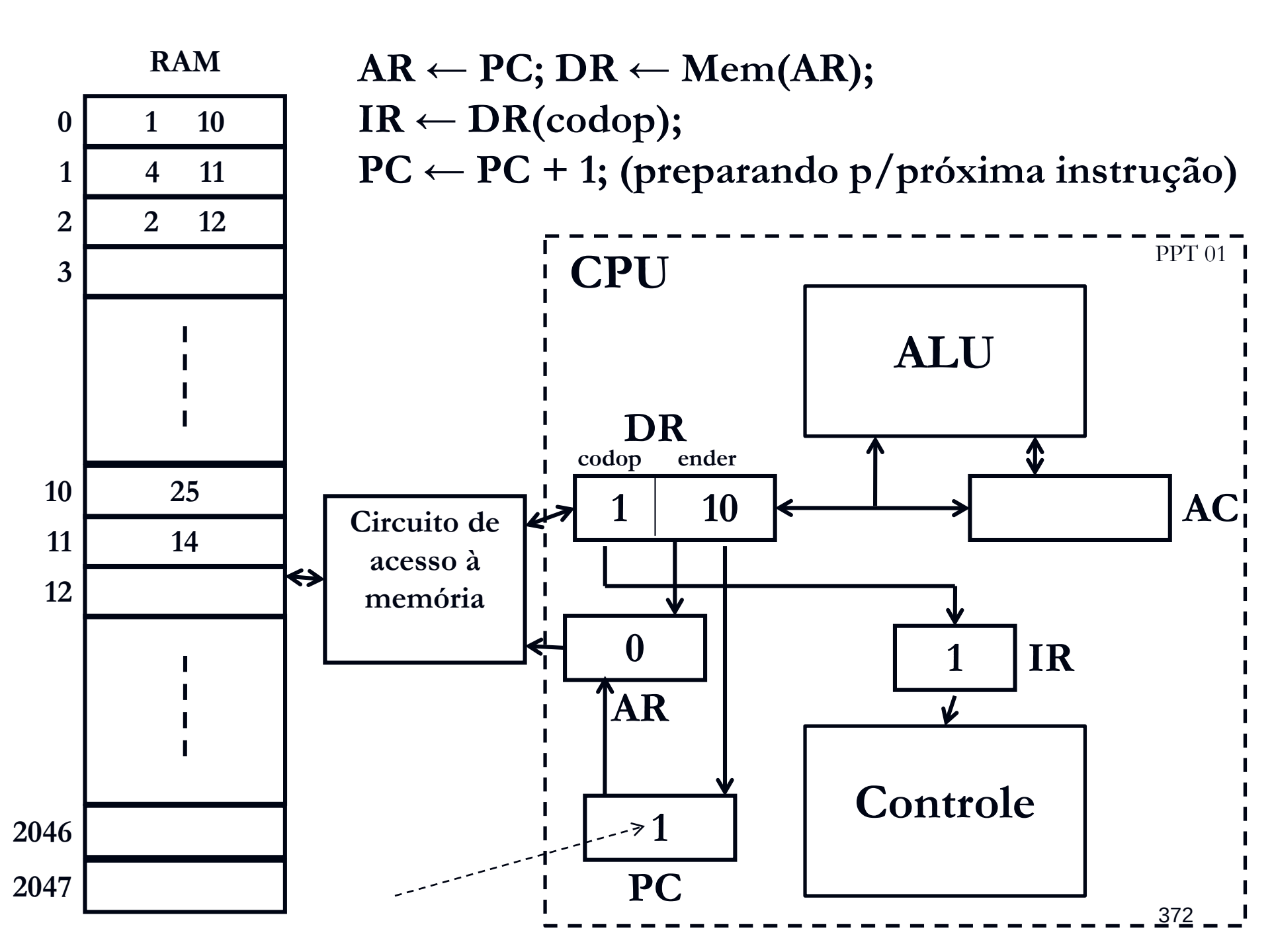

RAM
AR ← PC; DR ← Mem(AR);
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
CPU
PPT 01
ALU
DR
codop
ender
AC
Circuito de acesso à memória
0
AR
IR
Controle
0
PC
367

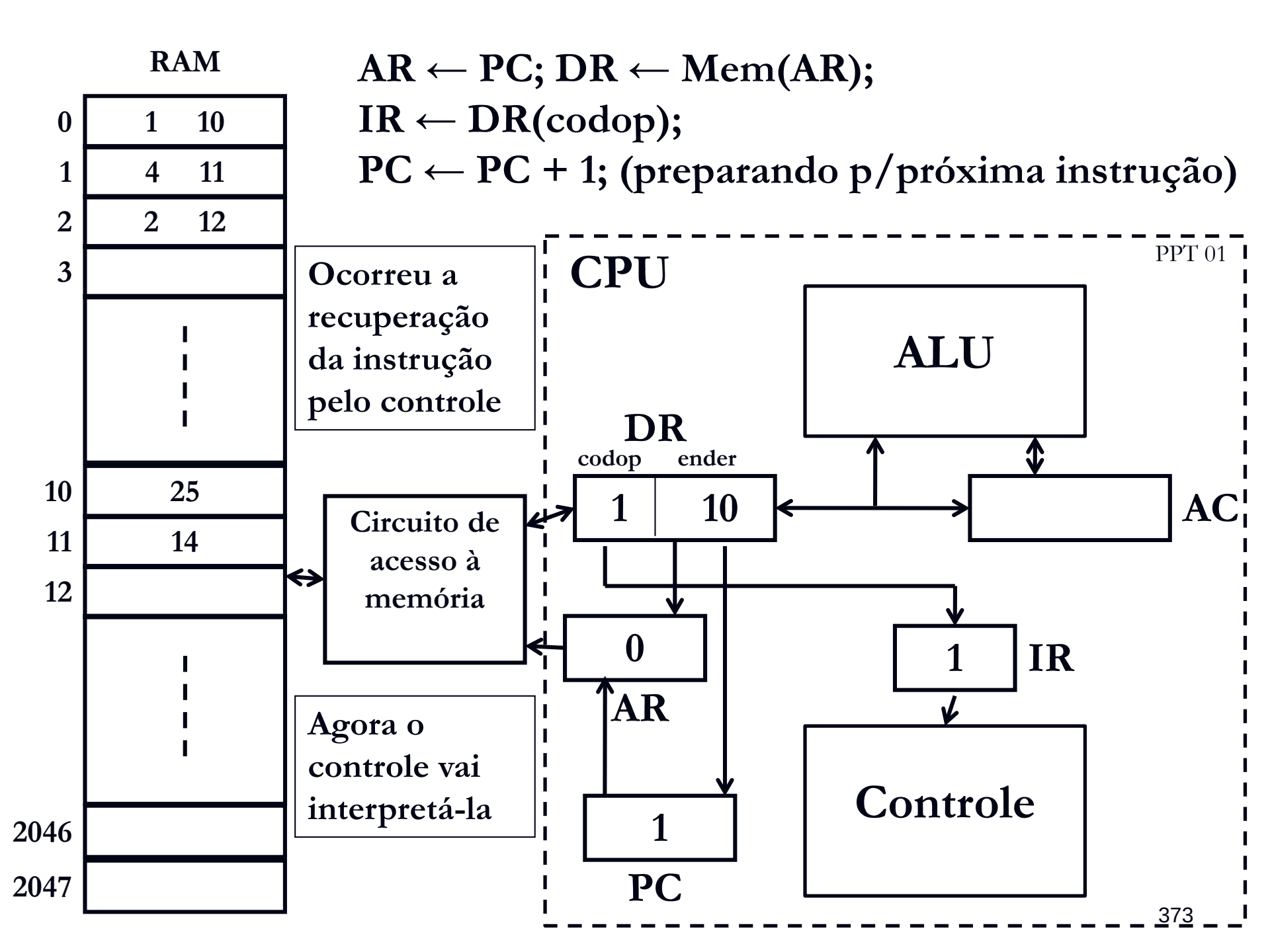

RAM
AR ← PC; DR ← Mem(AR);
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
CPU
PPT 01
ALU
DR
codop
ender
1
10
AC
Circuito de acesso à memória
0
AR
IR
Controle
0
PC
368

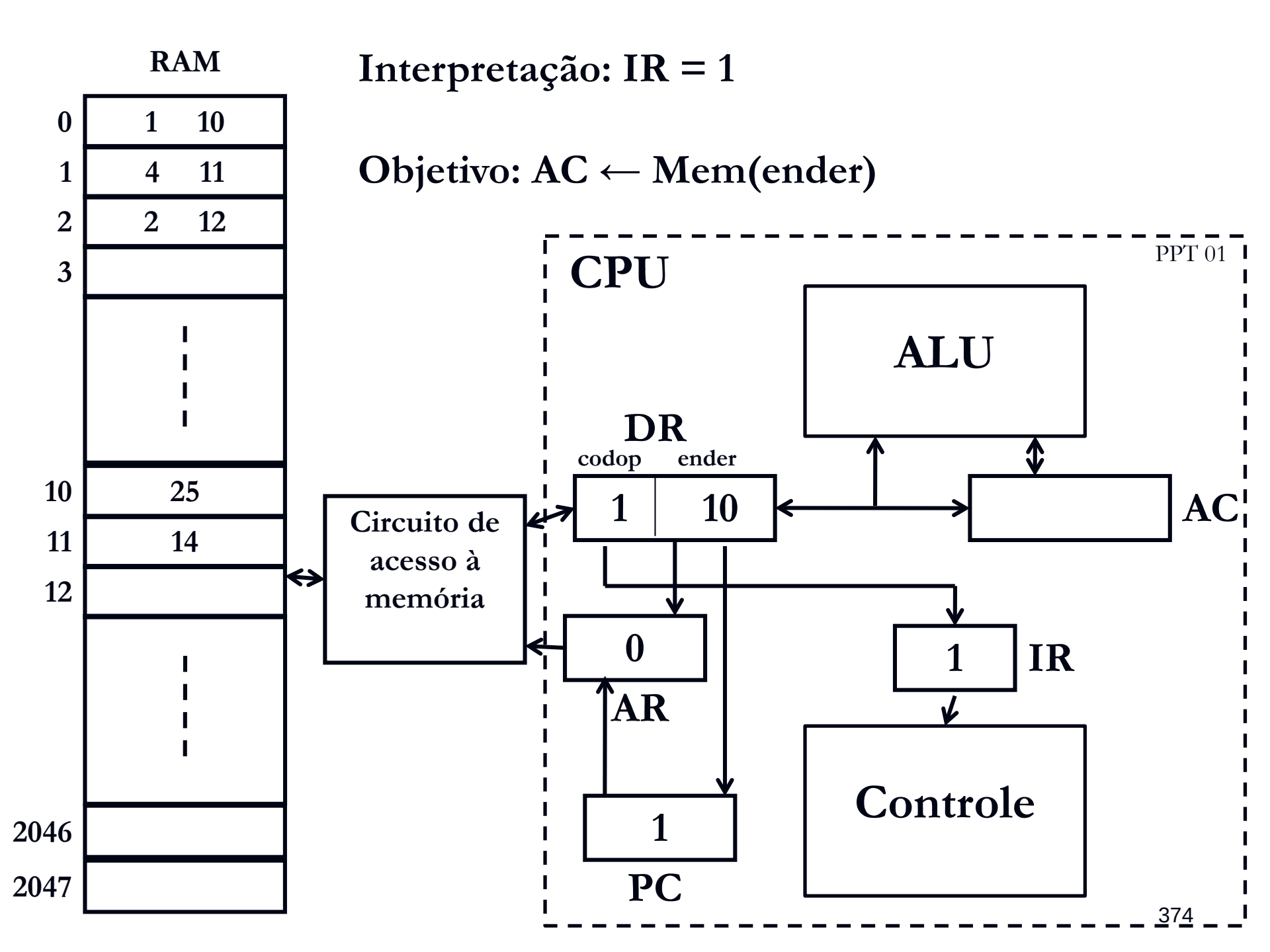

RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
AR ← PC; DR ← Mem(AR);
IR ← DR(codop);
CPU
PPT 01
ALU
DR
codop
ender
1
10
AC
Circuito de acesso à memória
0
AR
IR
Controle
0
PC
369

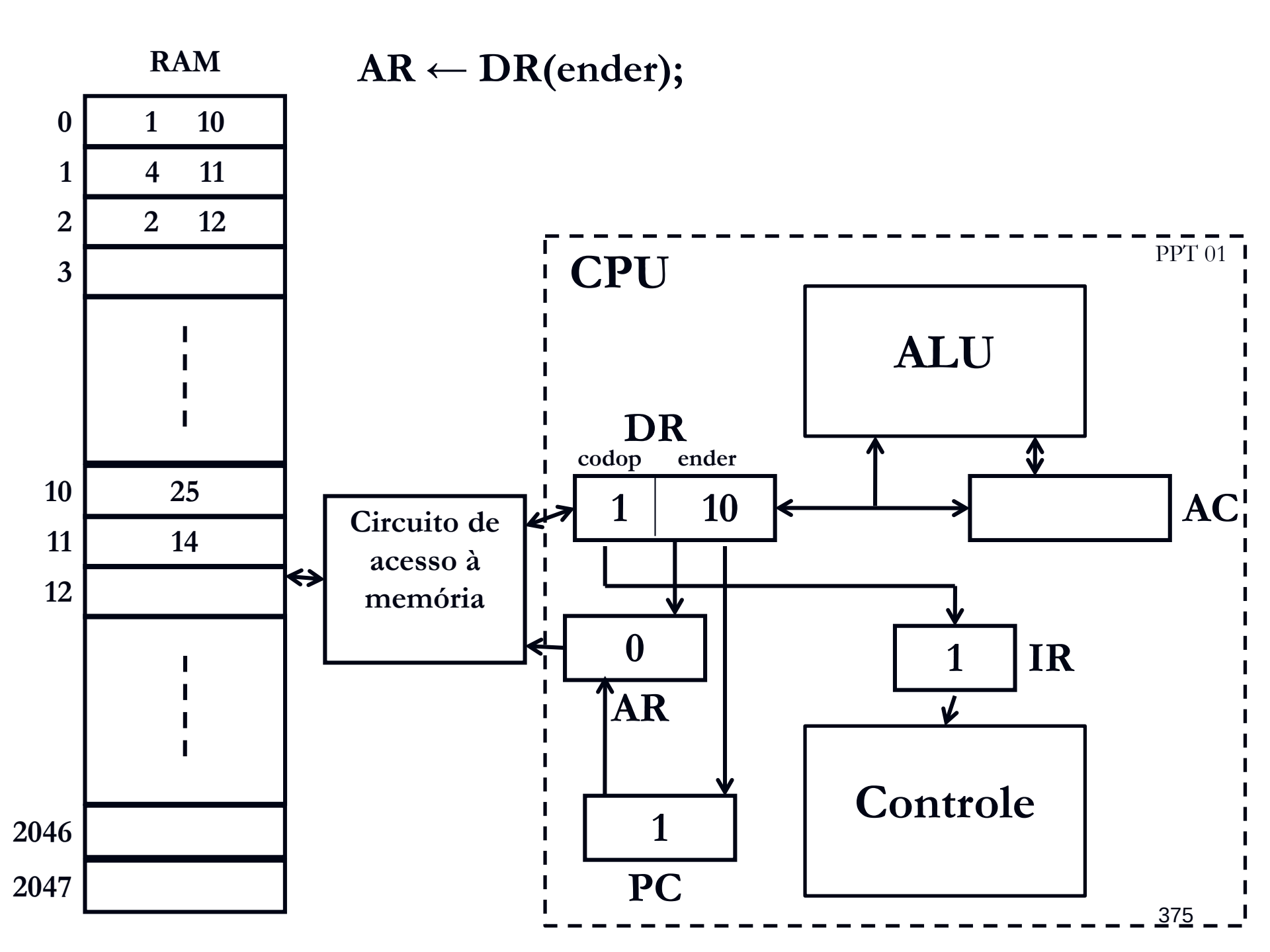

RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
AR ← PC; DR ← Mem(AR);
IR ← DR(codop);
CPU
PPT 01
ALU
DR
codop
ender
1
10
AC
Circuito de acesso à memória
0
AR
1
IR
Controle
0
PC
370

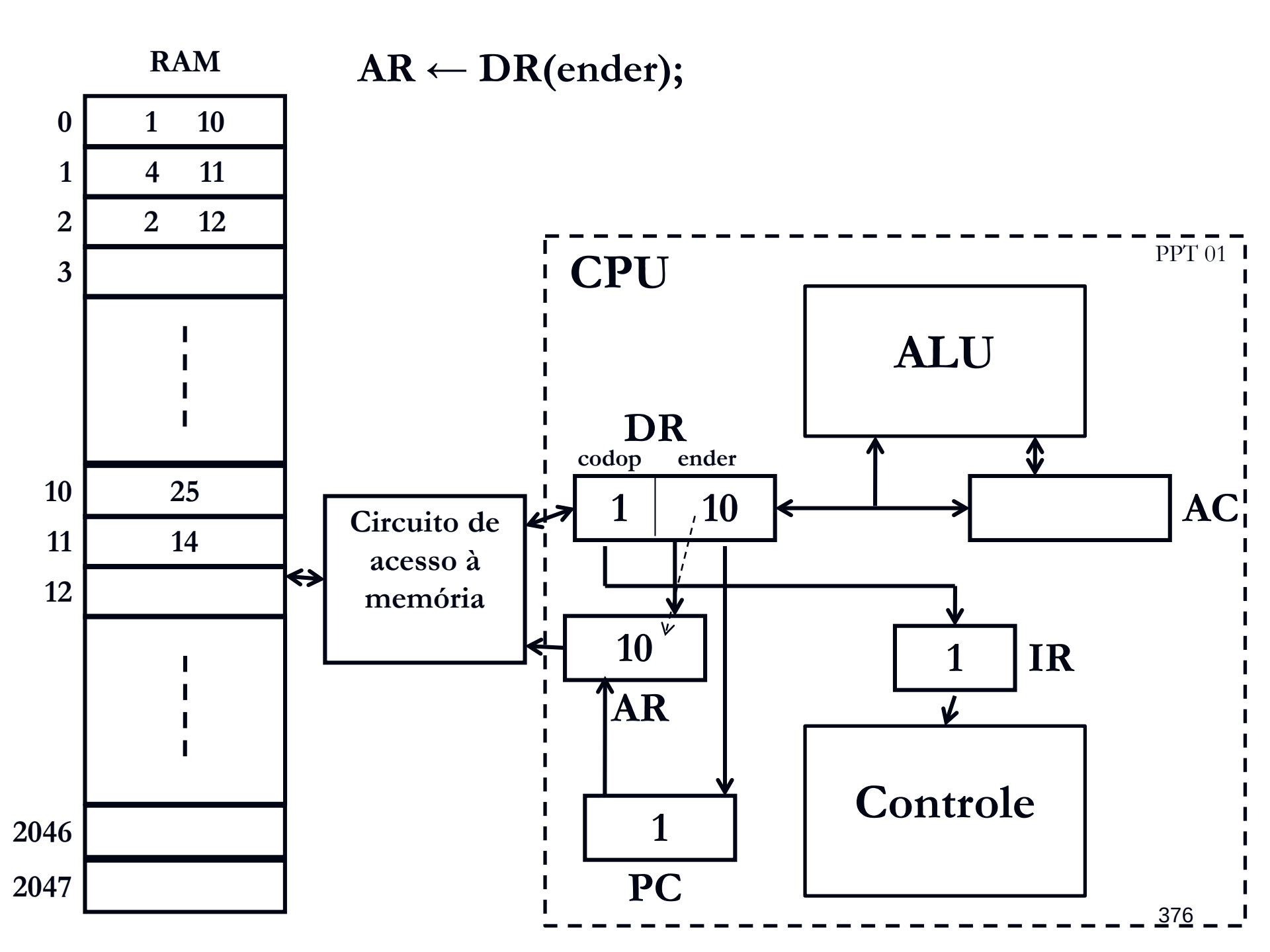

RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
AR ← PC; DR ← Mem(AR);
IR ← DR(codop);
PC ← PC + 1; (preparando p/próxima instrução)
CPU
PPT 01
ALU
DR
codop
ender
1
10
AC
Circuito de acesso à memória
0
AR
1
IR
Controle
0
PC
371

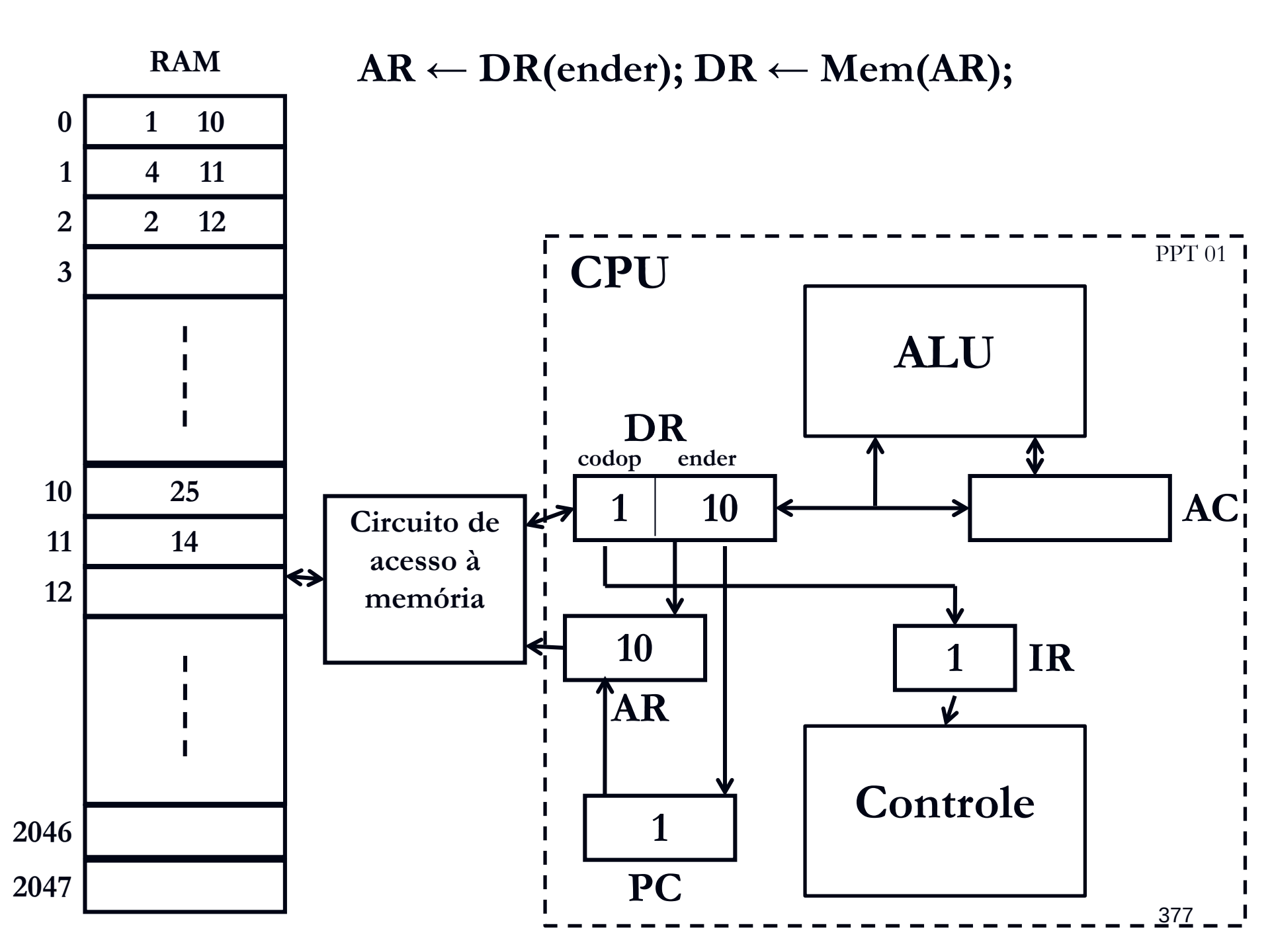

RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
AR ← PC; DR ← Mem(AR);
IR ← DR(codop);
PC ← PC + 1; (preparando p/próxima instrução)
CPU
PPT 01
ALU
DR
codop
ender
1
10
AC
Circuito de acesso à memória
0
AR
1
IR
Controle
1
PC
372

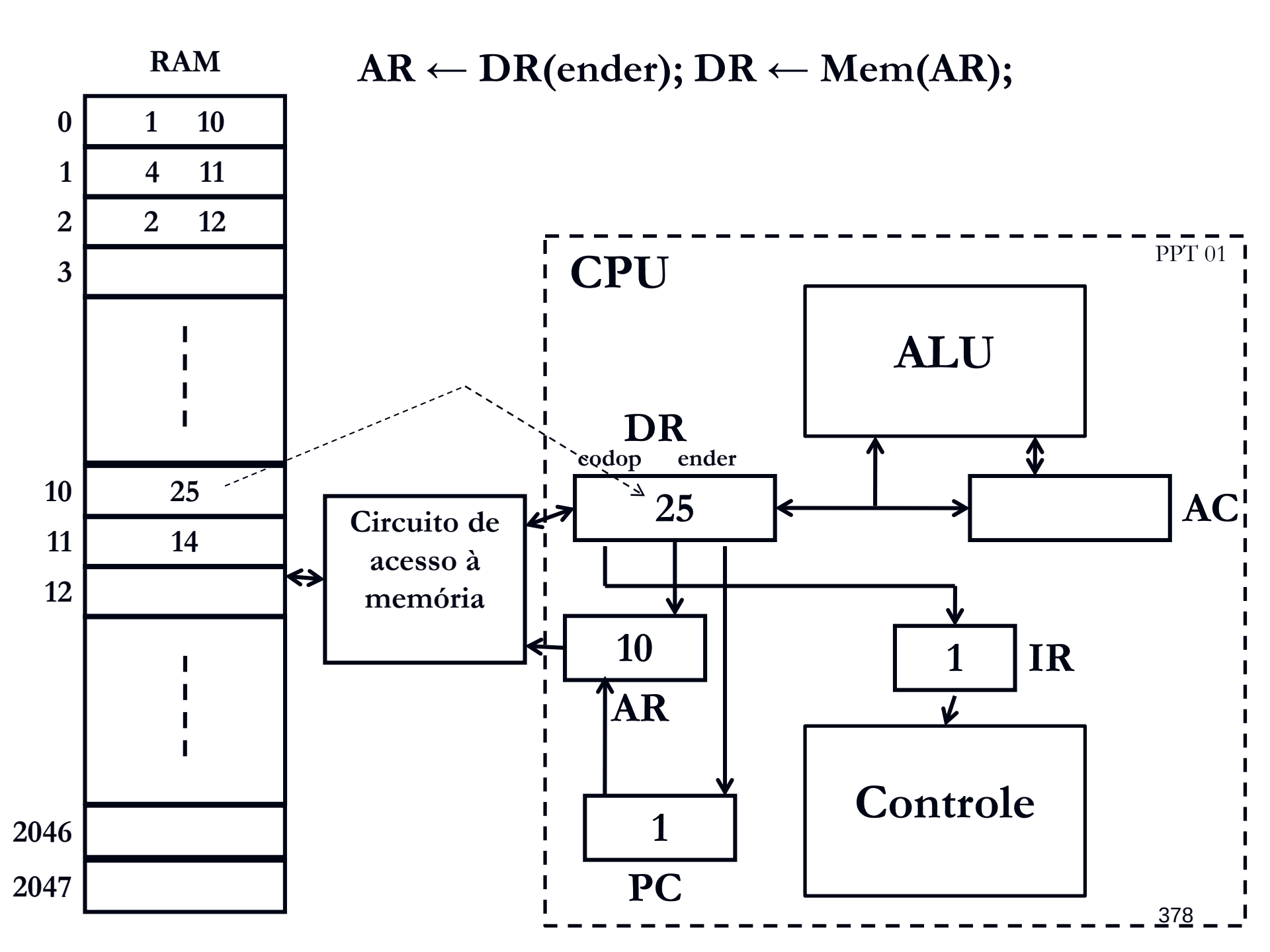

RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
AR ← PC; DR ← Mem(AR);
IR ← DR(codop);
PC ← PC + 1; (preparando p/próxima instrução)
Ocorreu a recuperação da instrução pelo controle
Circuito de acesso à memória
Agora o controle vai interpretá-la
CPU
PPT 01
ALU
DR
codop
ender
1
10
AC
0
AR
1
IR
1
PC
Controle
373

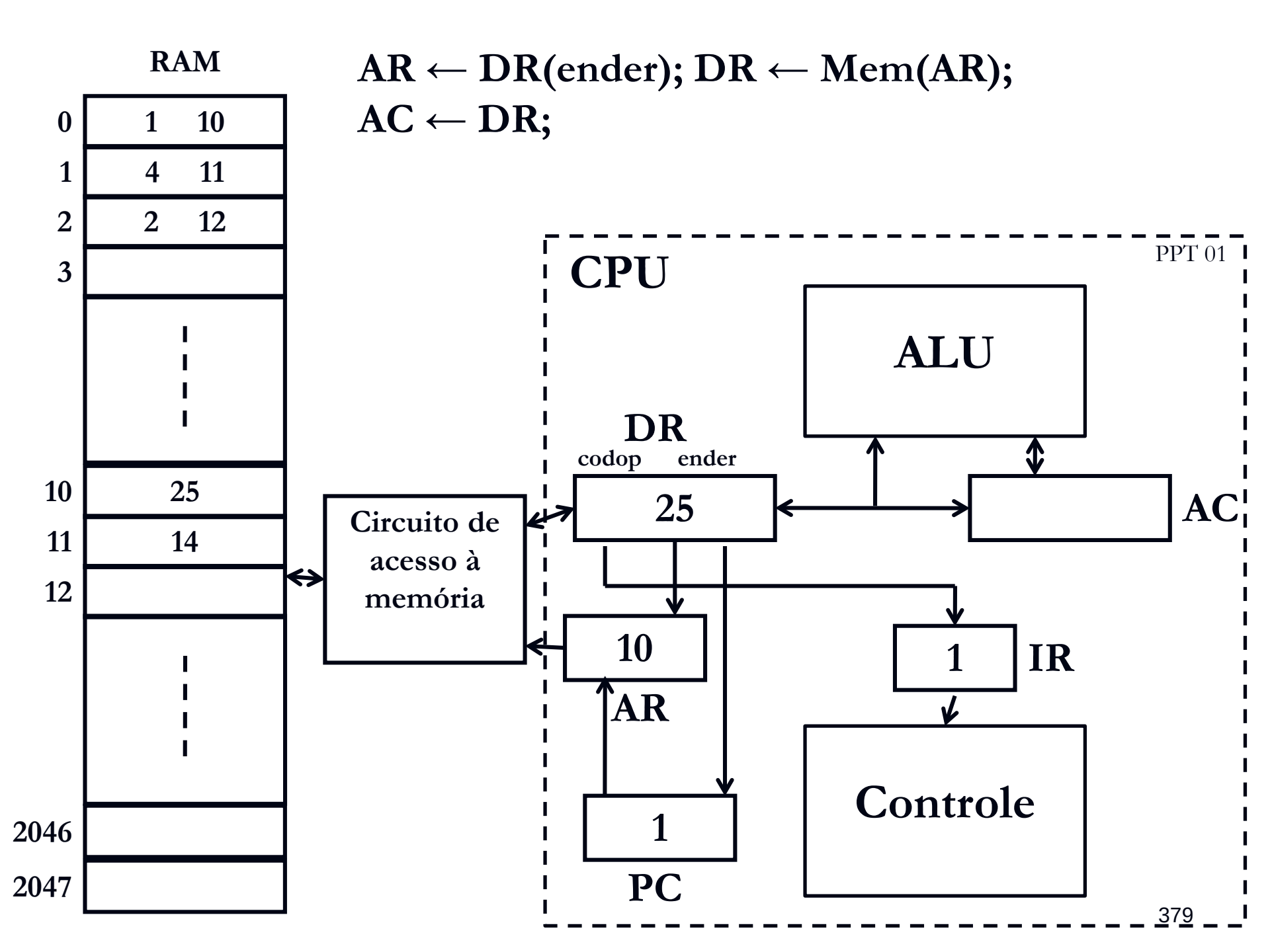

RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
Interpretação: IR = 1
Objetivo: AC ← Mem(ender)
CPU
PPT 01
ALU
DR
codop
ender
1
10
AC
Circuito de acesso à memória
0
AR
1
IR
Controle
1
PC
374

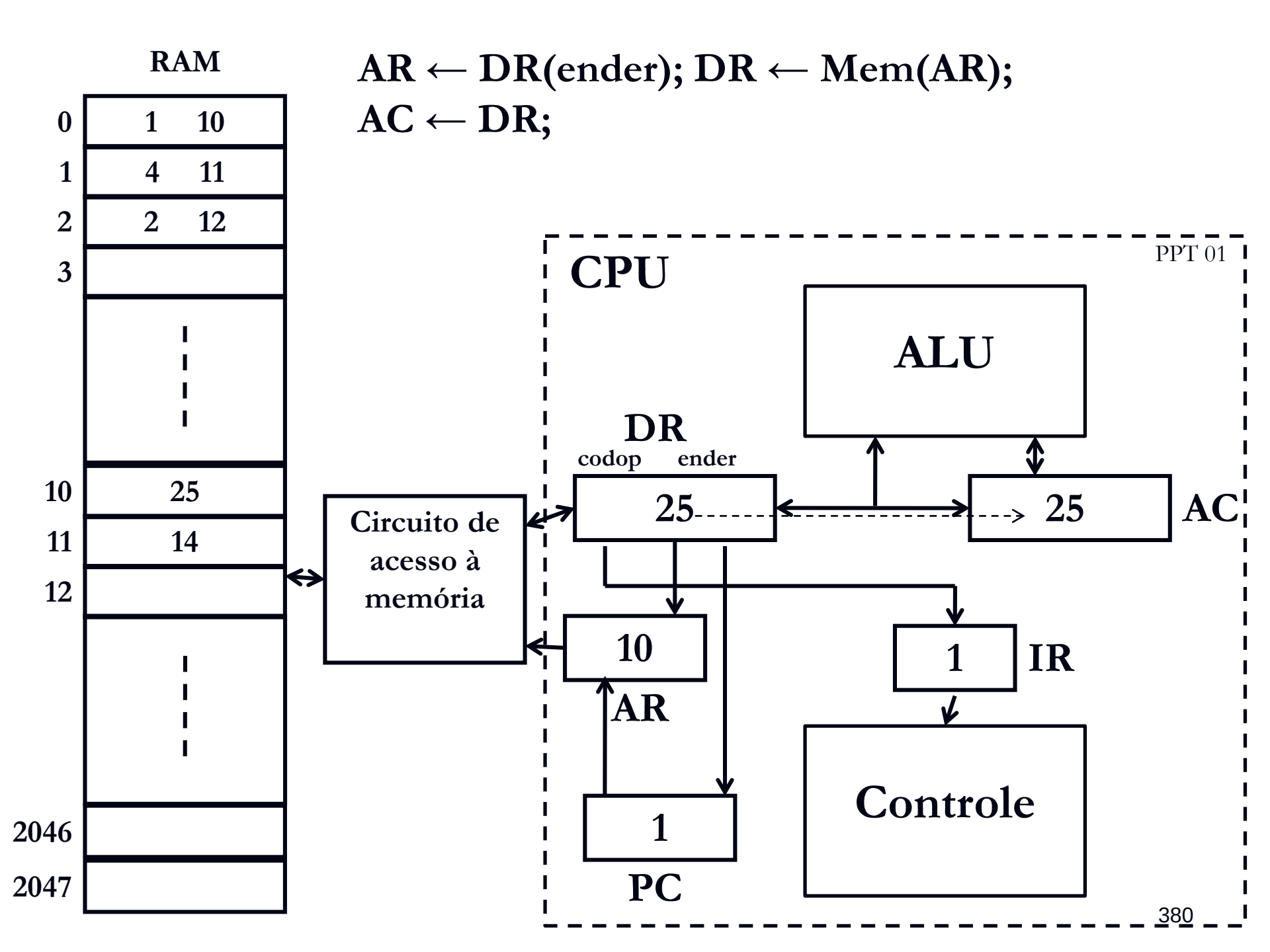

AR ← DR(ender);
RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
DR
codop ender
1 10
AC
0
AR
1
IR
Controle
1
PC
375

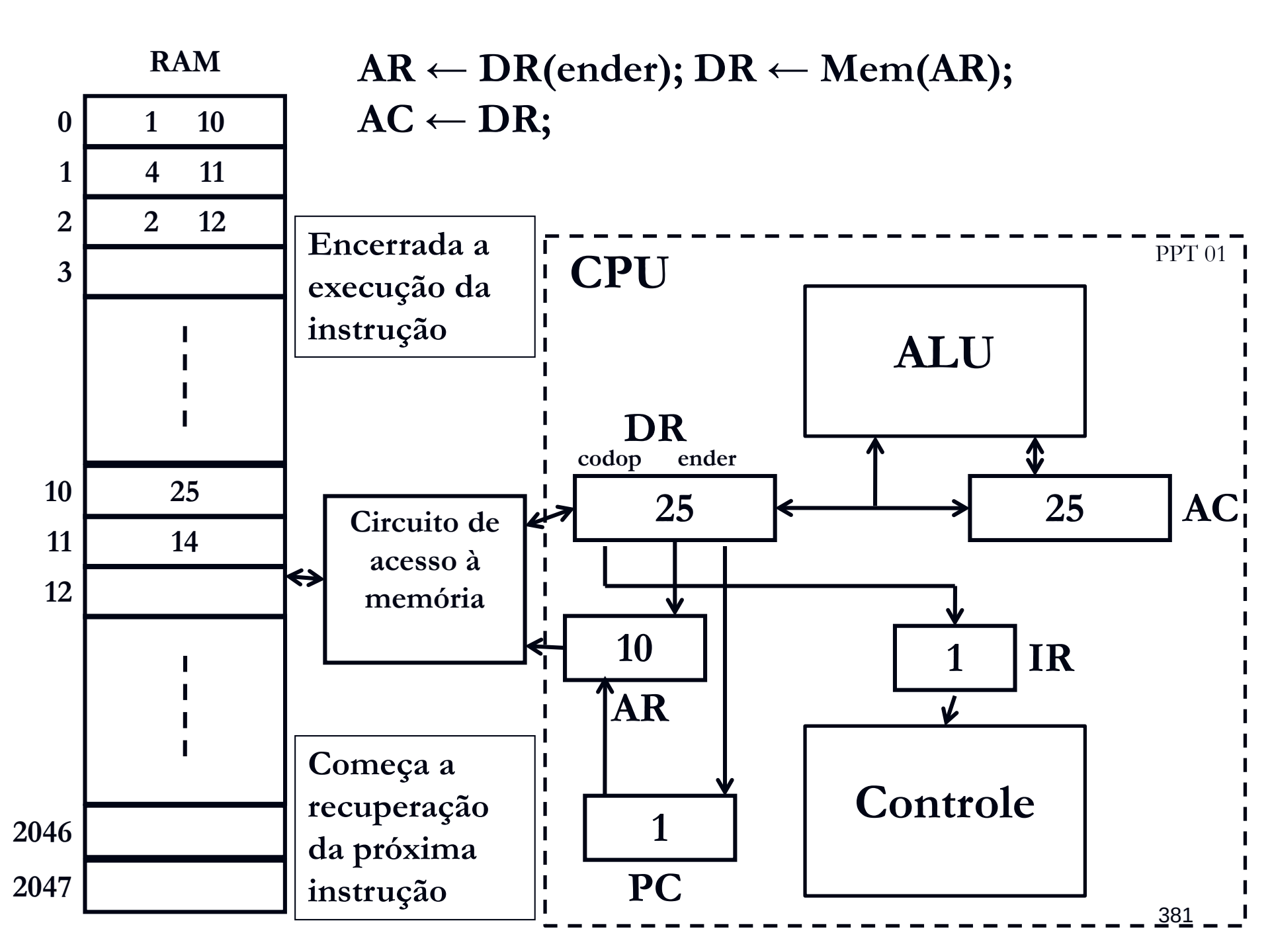

RAM
AR ← DR(ender);
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
CPU
PPT 01
ALU
DR
codop
ender
1
10
AC
Circuito de acesso à memória
10
AR
1
IR
Controle
1
PC
376

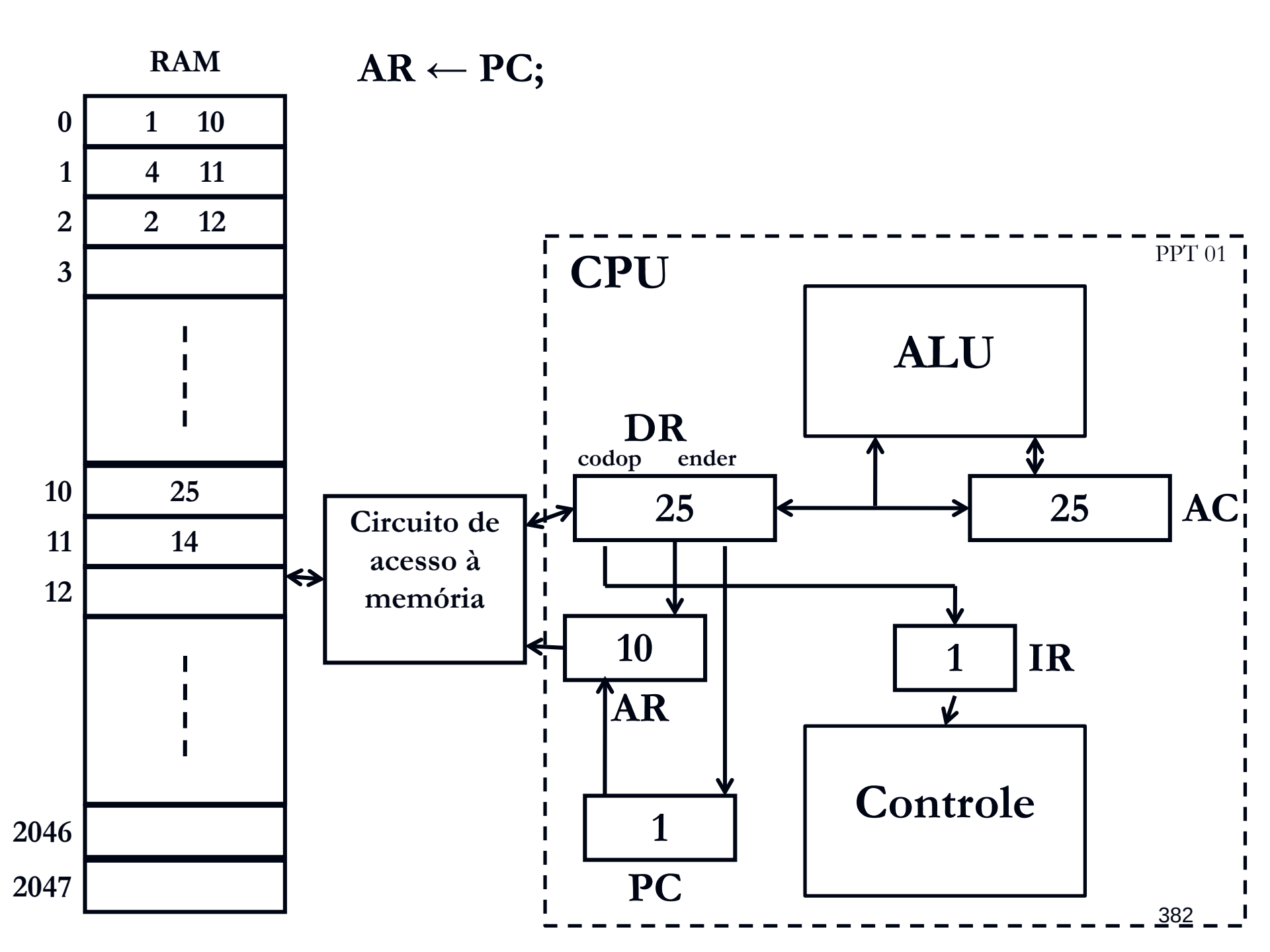

AR ← DR(ender); DR ← Mem(AR);
RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
DR
codop
ender
1
10
AC
10
AR
1
IR
Controle
1
PC
377

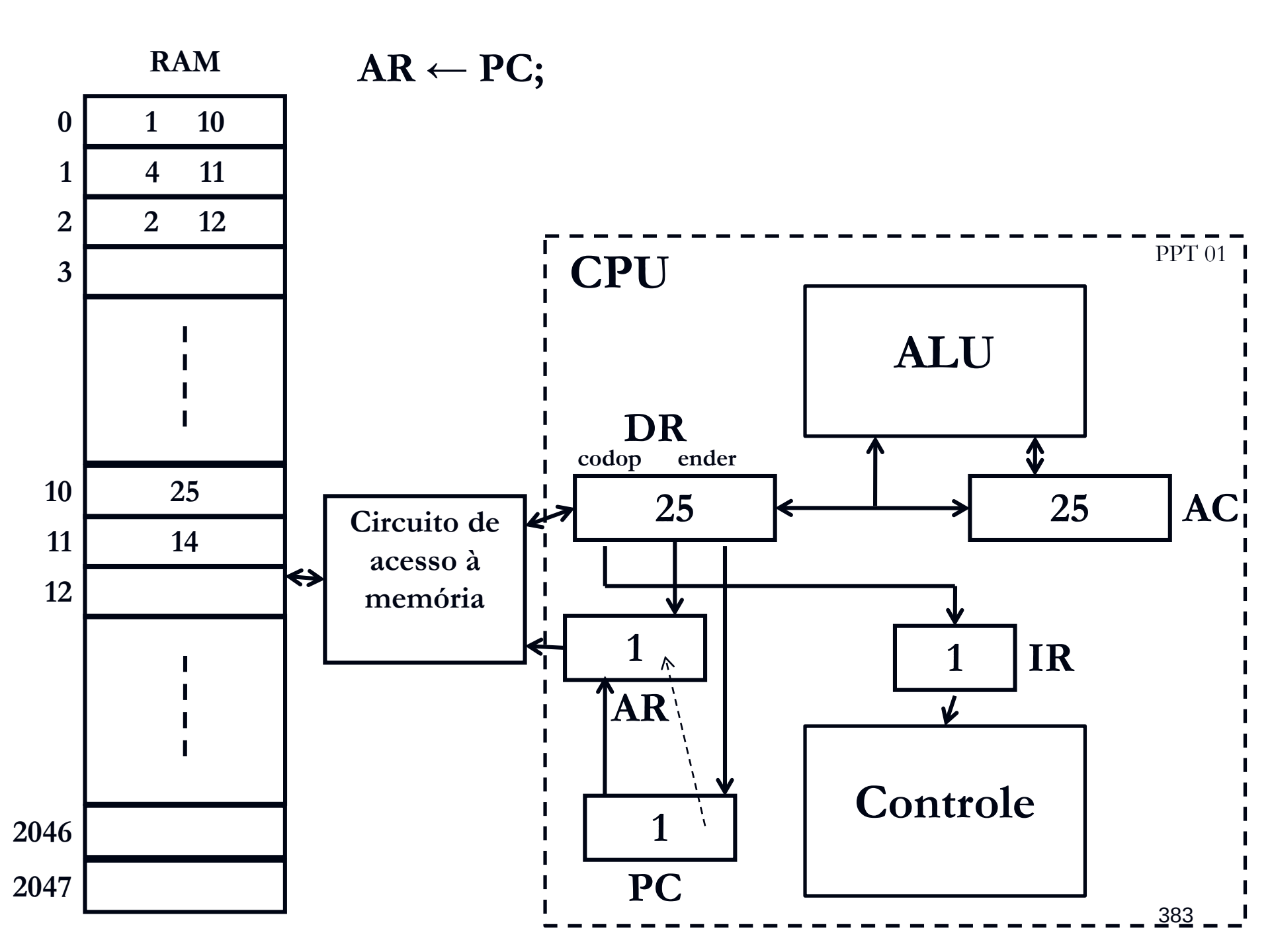

AR ← DR(ender); DR ← Mem(AR);
RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
CPU
PPT 01
ALU
DR
codop
ender
25
AC
Circuito de acesso à memória
10
AR
1
IR
Controle
1
PC
378

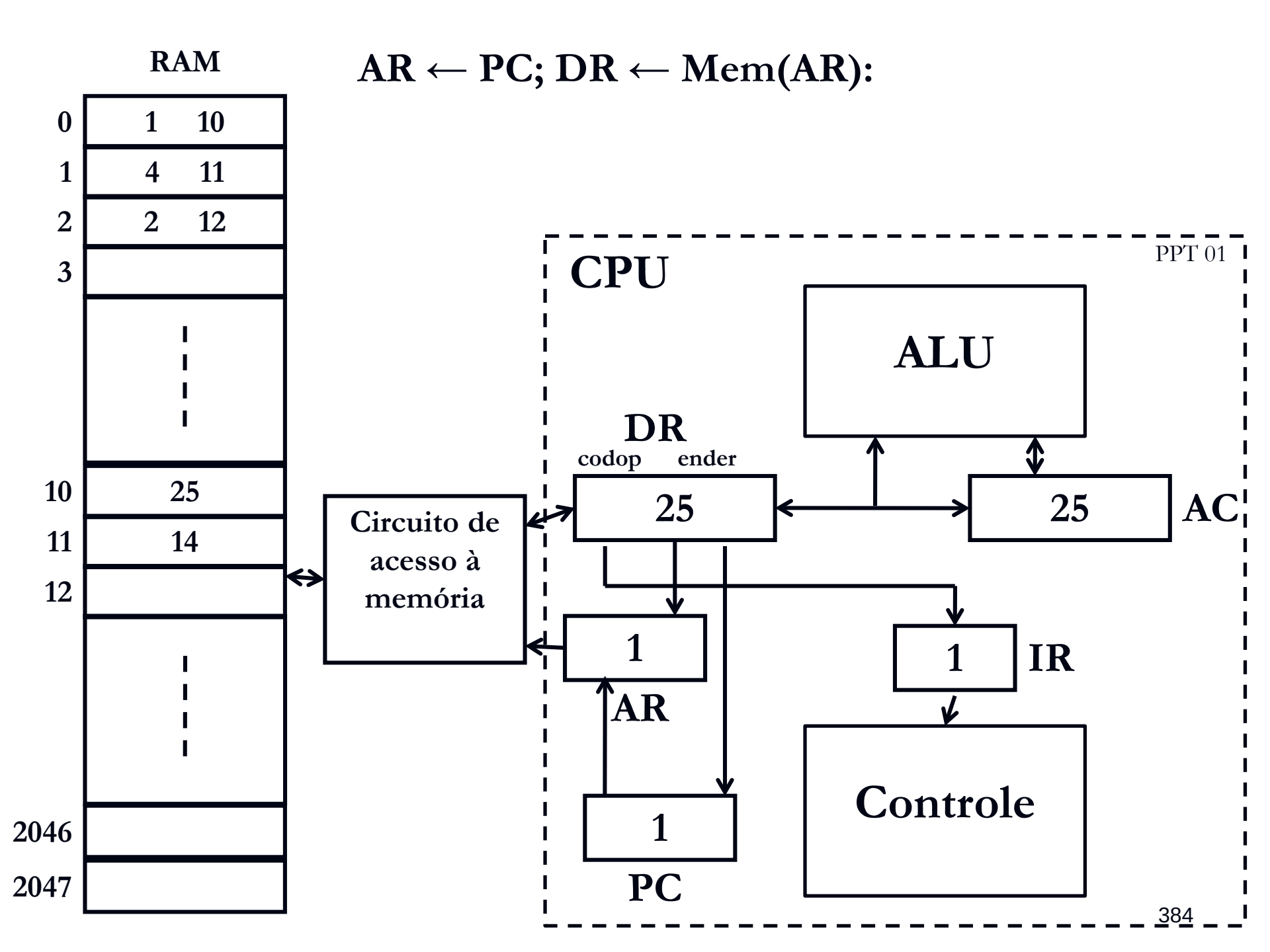

RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
AR ← DR(ender); DR ← Mem(AR);
AC ← DR;
CPU
PPT 01
ALU
DR
codop ender
25
AC
Circuito de acesso à memória
10
AR
1
IR
Controle
1
PC
379

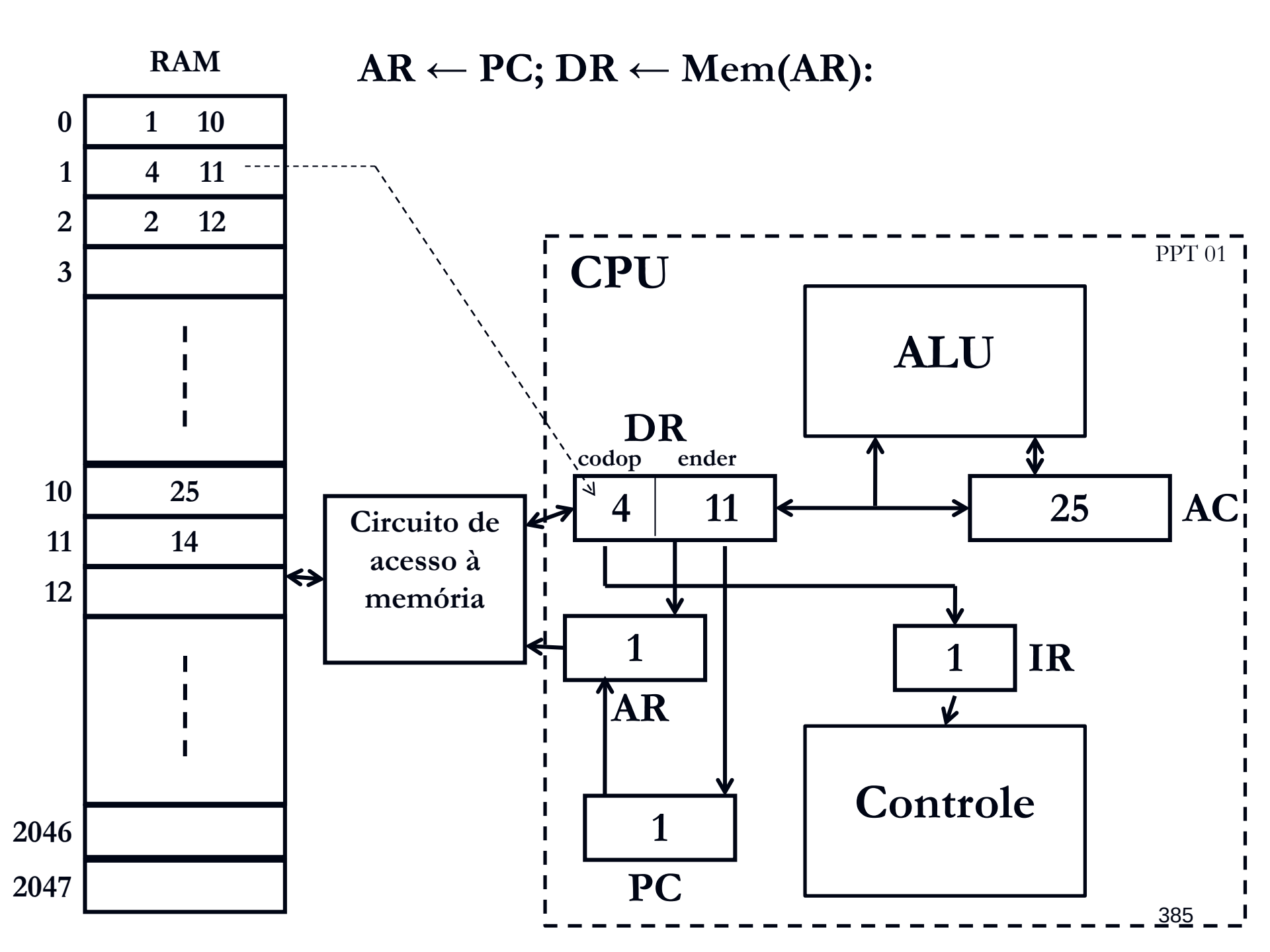

RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
AR ← DR(ender); DR ← Mem(AR);
AC ← DR;
CPU
PPT 01
ALU
DR
codop ender
25
25
AC
Circuito de acesso à memória
10
AR
1
IR
Controle
1
PC
380

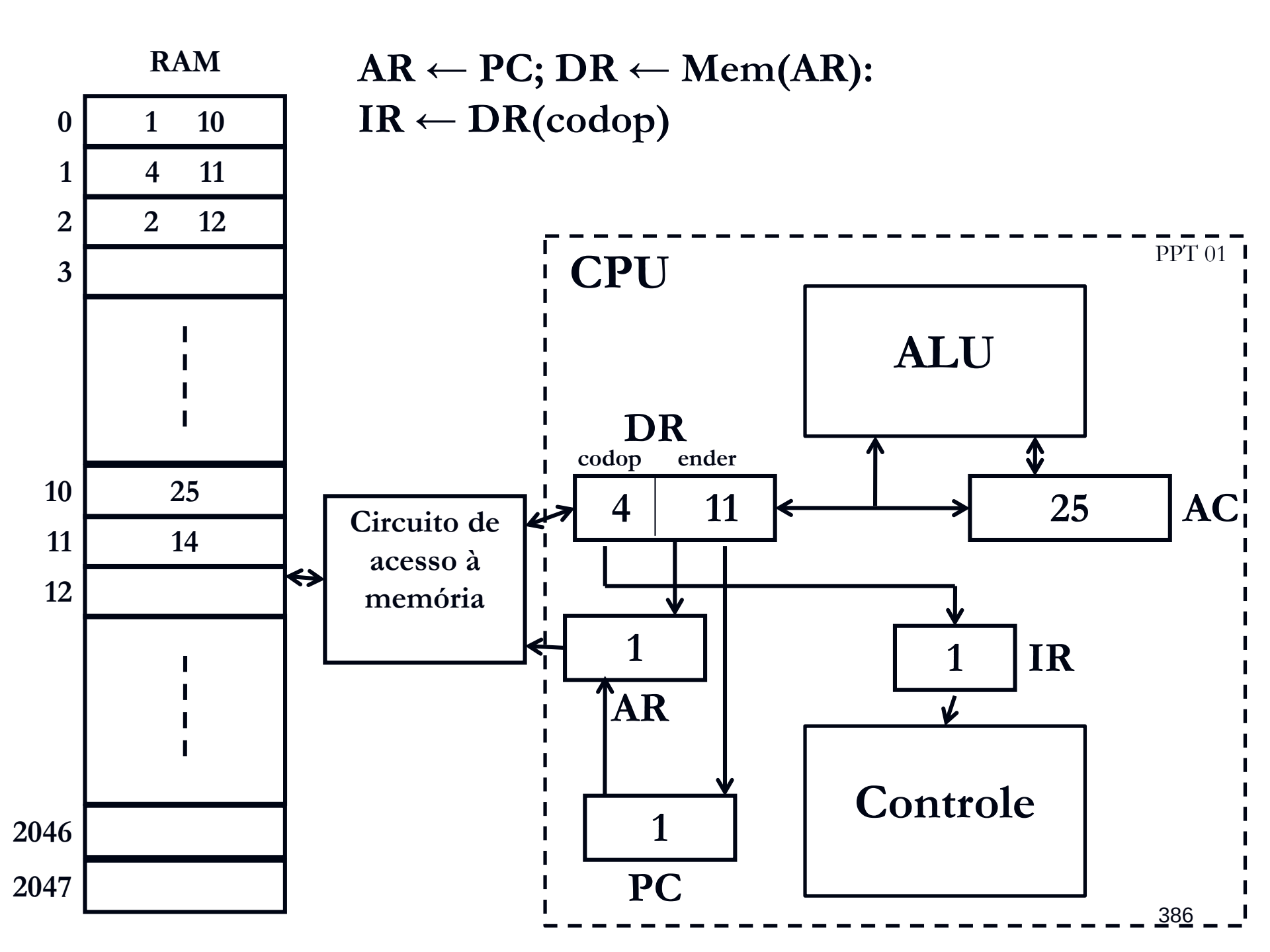

RAM
AR ← DR(ender); DR ← Mem(AR);
AC ← DR;
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
Encerrada a execução da instrução
Circuito de acesso à memória
Começa a recuperação da próxima instrução
CPU
PPT 01
ALU
DR
codop ender
25
25
AC
10
AR
1
IR
1
PC
Controle
381

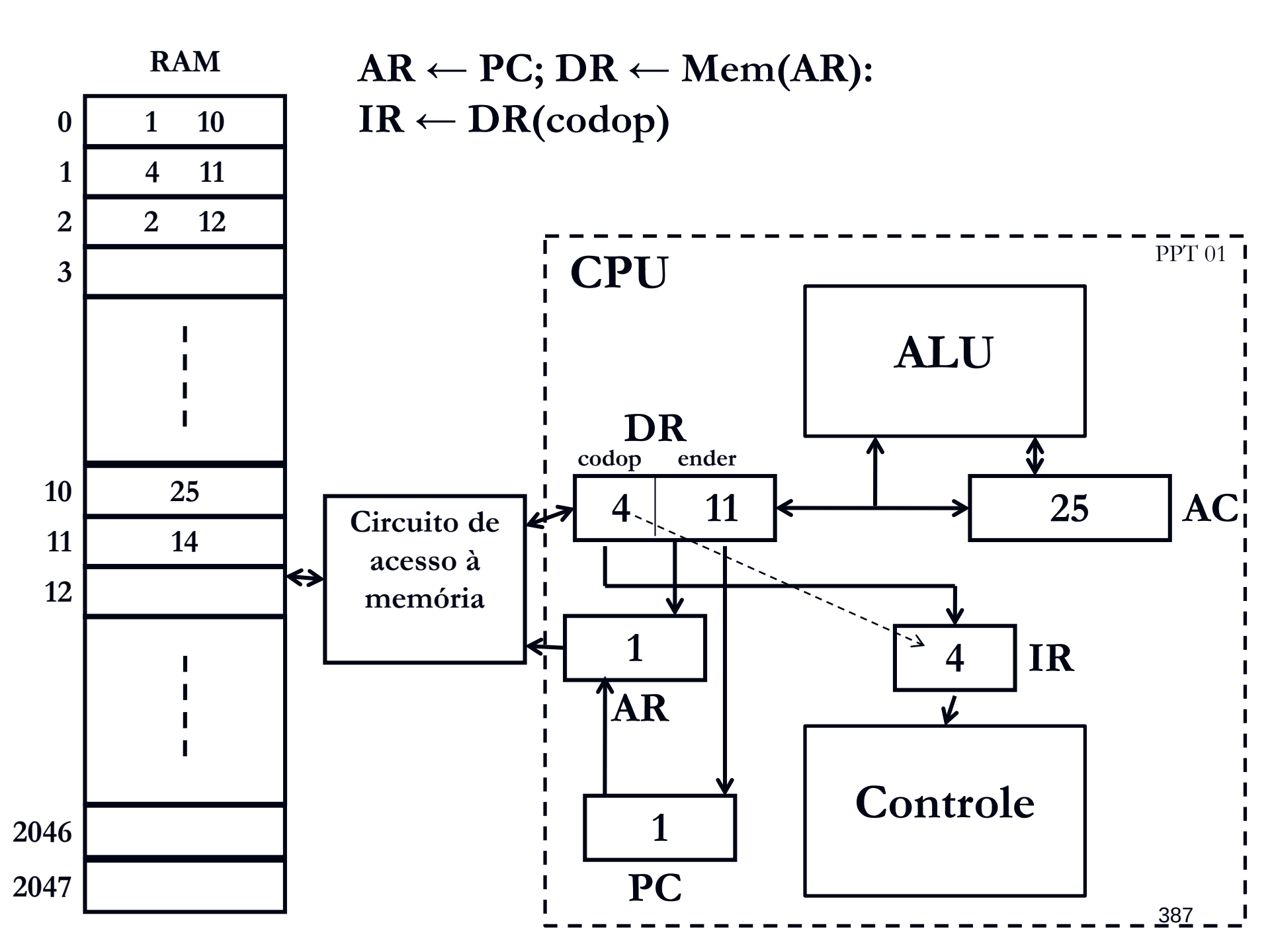

RAM
AR ← PC;
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
DR
codop
ender
25
25
AC
10
AR
1
IR
Controle
1
PC
382

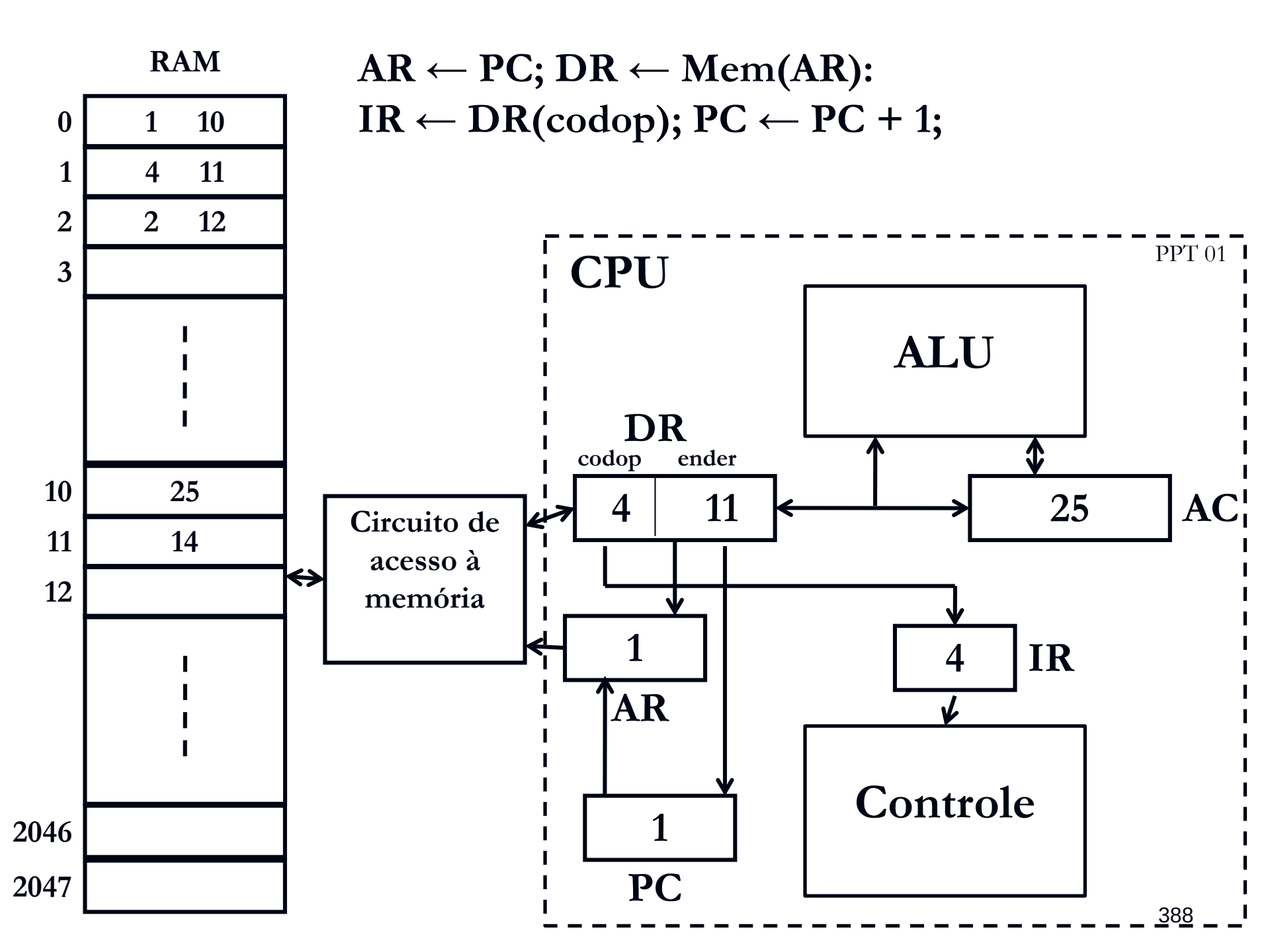

RAM
AR ← PC;
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
DR
codop ender
25
25
AC
1
AR
1
IR
Controle
1
PC
383

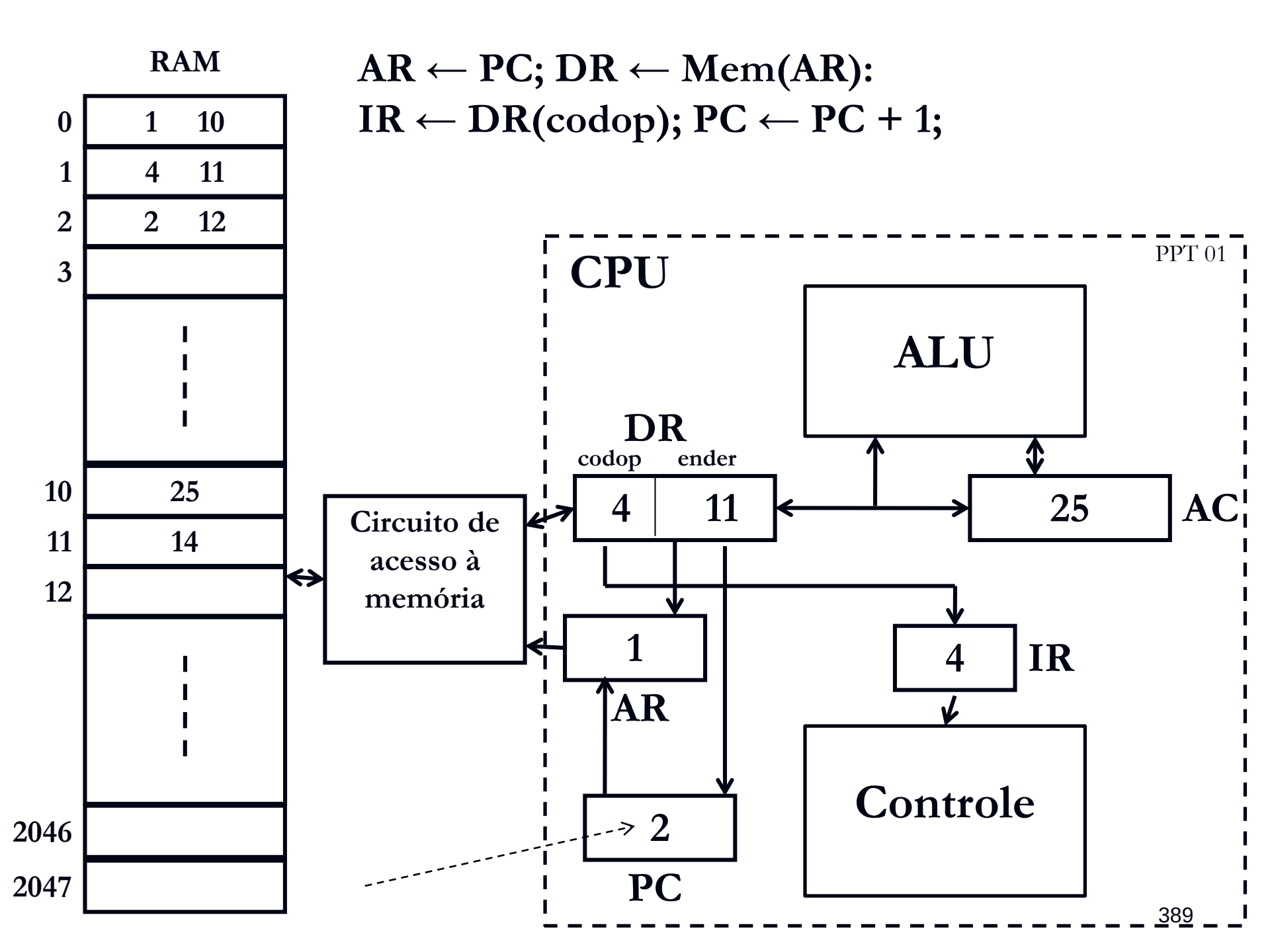

RAM
AR ← PC; DR ← Mem(AR):
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
CPU
PPT 01
ALU
DR
codop
ender
25
25
AC
Circuito de acesso à memória
1
AR
1
IR
Controle
1
PC
384

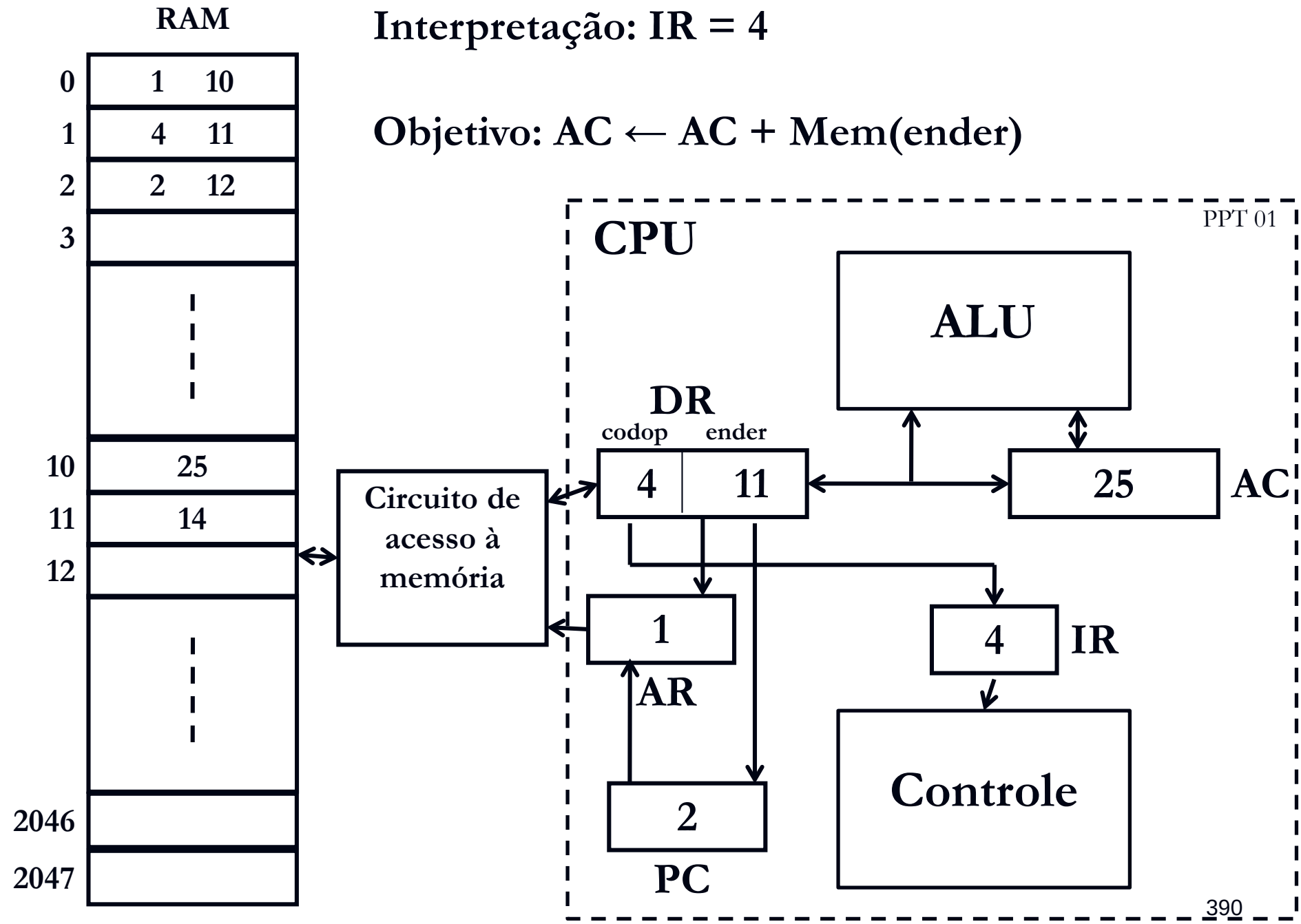

RAM
AR ← PC; DR ← Mem(AR):
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
CPU
PPT 01
ALU
DR
codop
ender
4
11
25
AC
Circuito de acesso à memória
1
AR
1
IR
Controle
1
PC
385

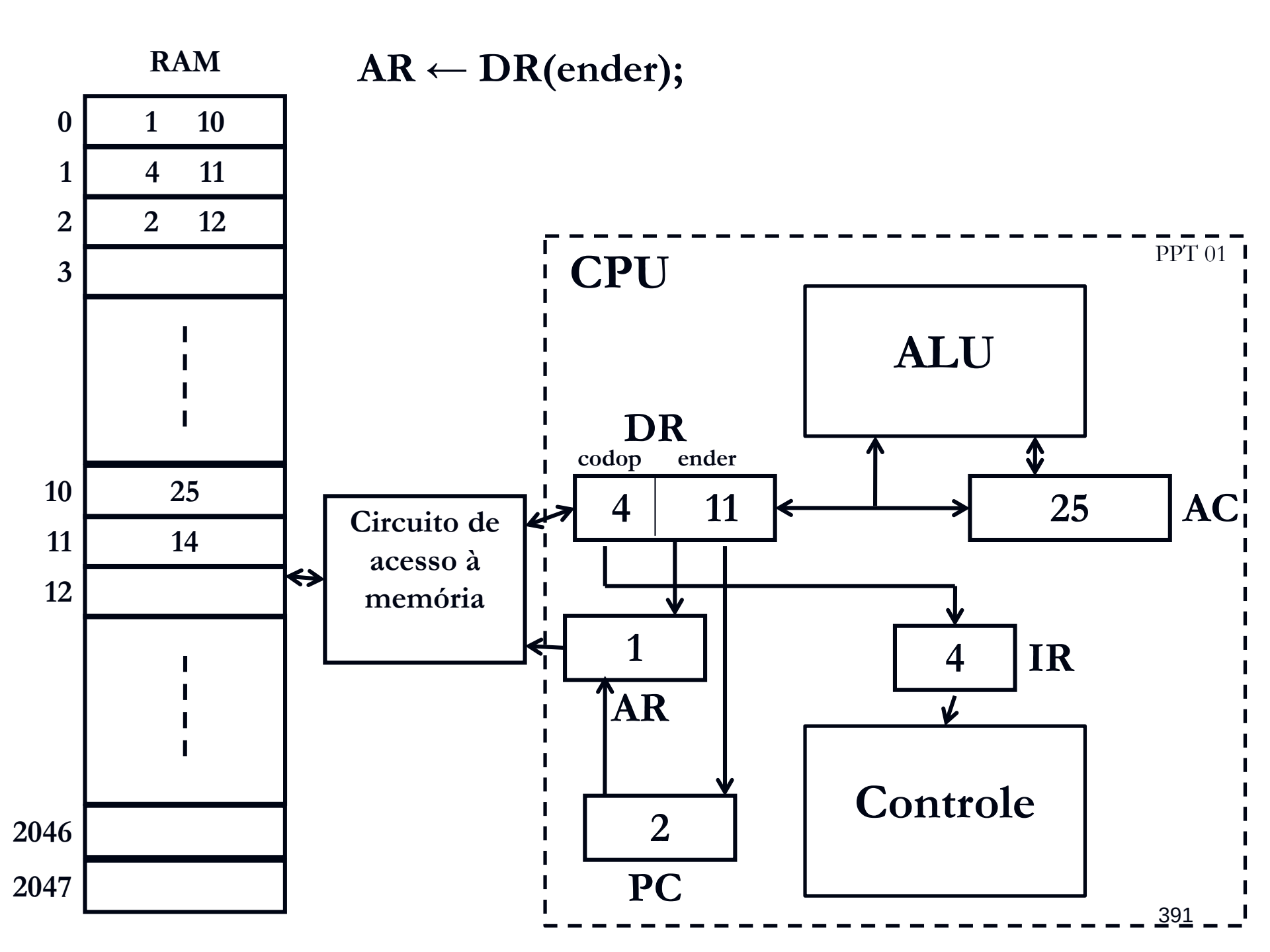

RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
AR ← PC; DR ← Mem(AR):
IR ← DR(codop)
CPU
PPT 01
ALU
DR
codop
ender
4
11
25
AC
Circuito de acesso à memória
1
AR
1
IR
Controle
1
PC
386

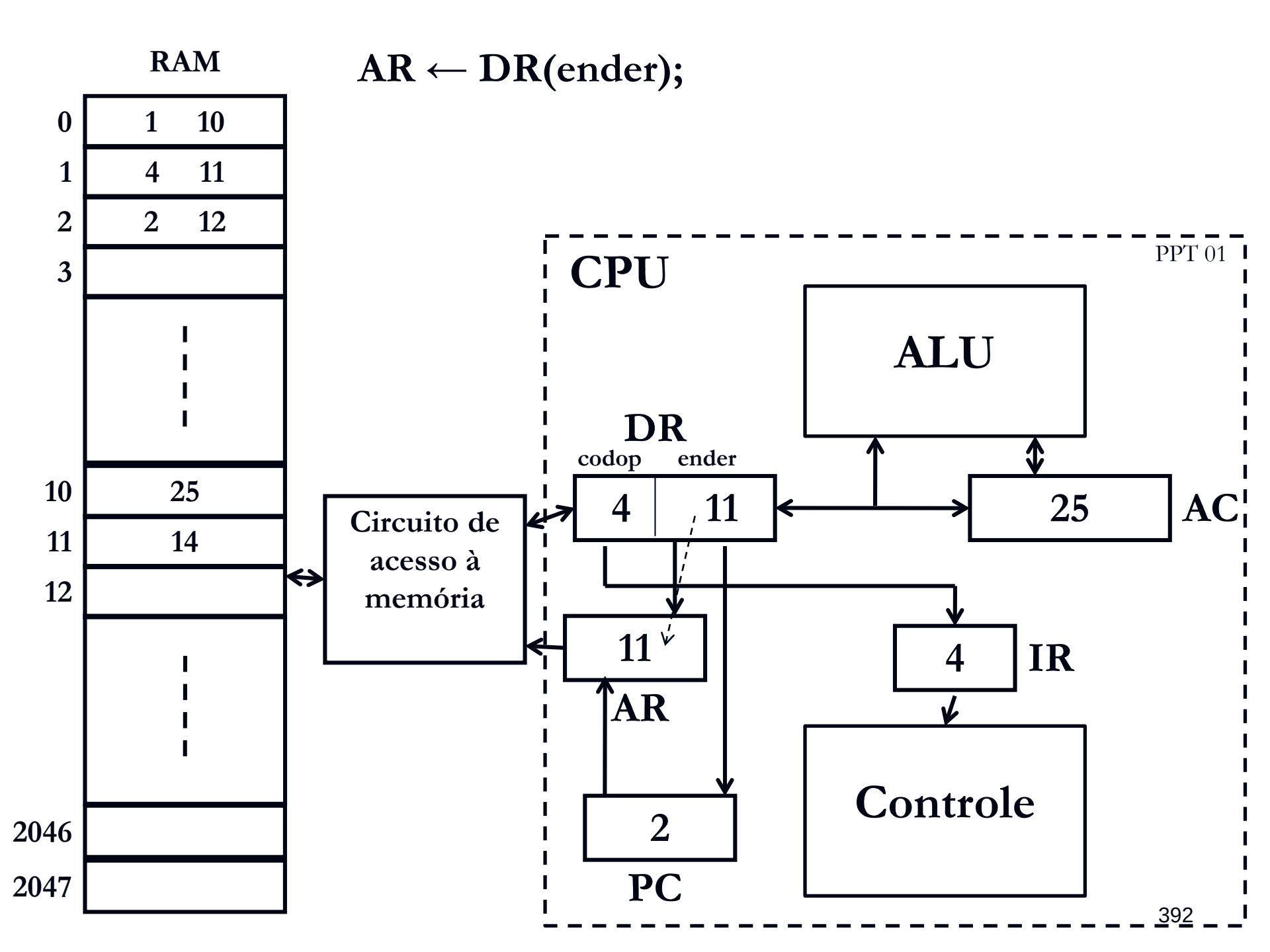

RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
AR ← PC; DR ← Mem(AR):
IR ← DR(codop)
CPU
PPT 01
ALU
DR
codop
ender
4
11
25
AC
Circuito de acesso à memória
1
AR
4
IR
Controle
1
PC
387

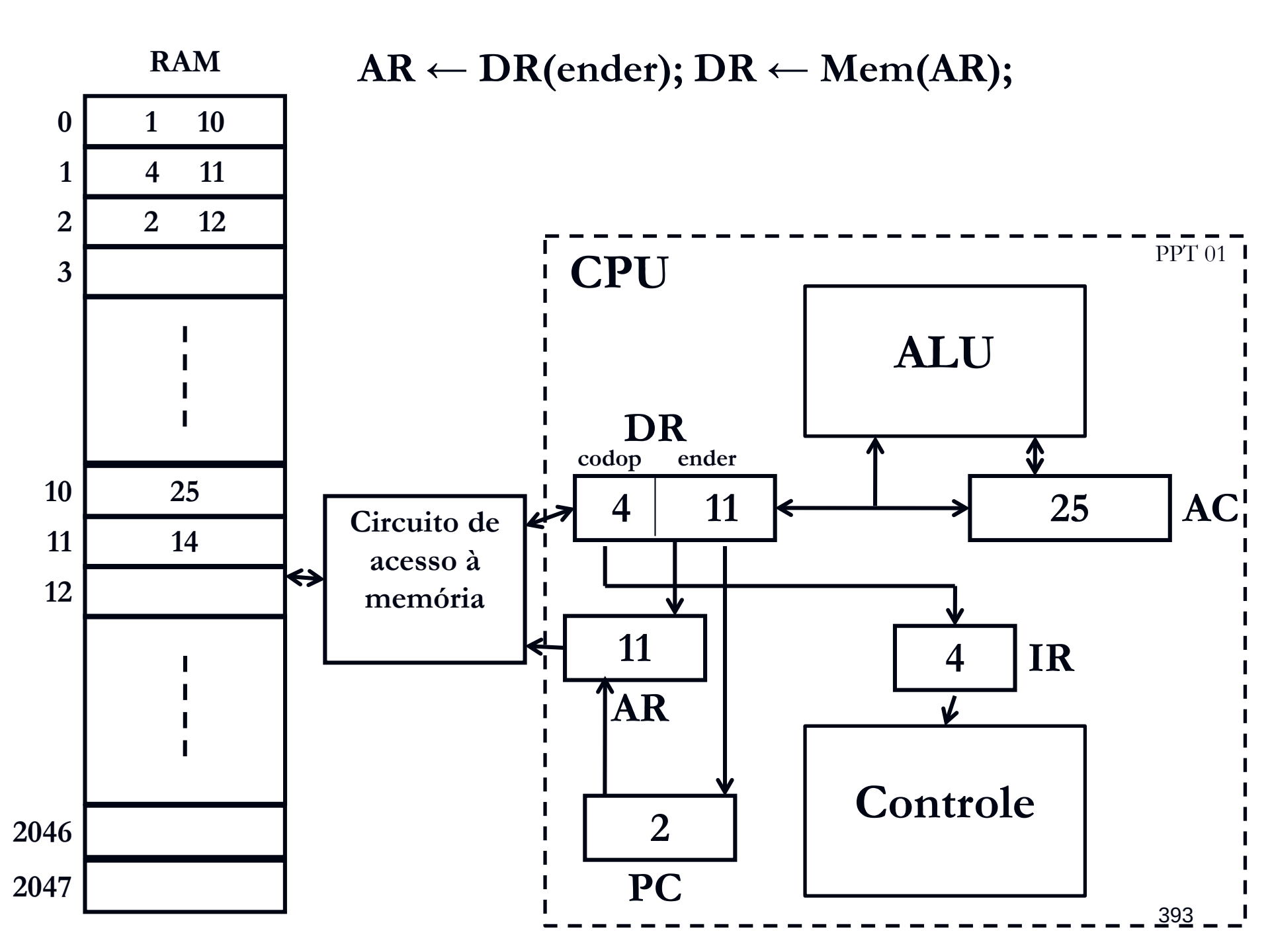

RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
AR ← PC; DR ← Mem(AR):
IR ← DR(codop); PC ← PC + 1;
CPU
PPT 01
ALU
DR
codop
ender
4
11
25
AC
Circuito de acesso à memória
1
AR
4
IR
Controle
1
PC
388

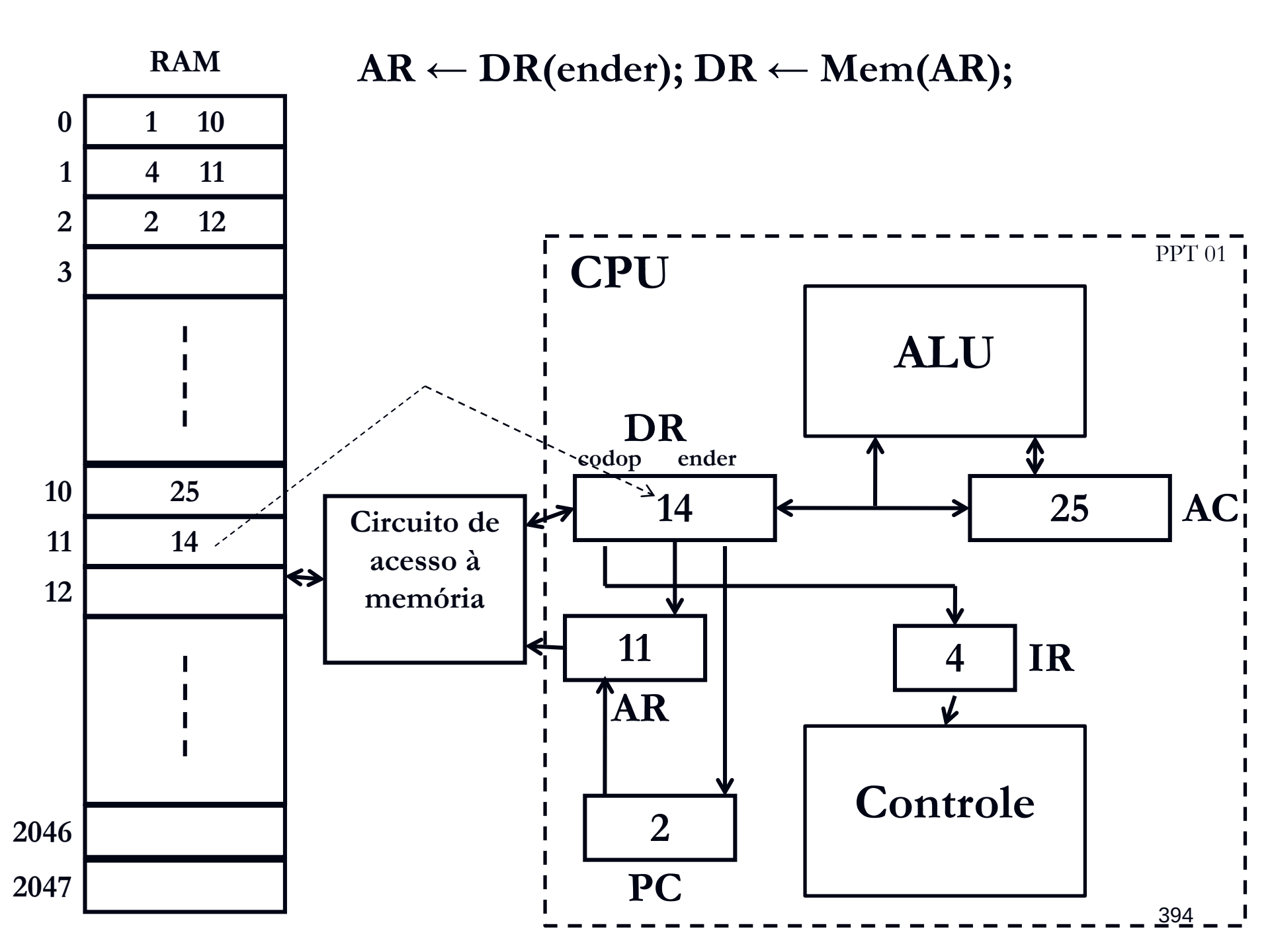

RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
AR ← PC; DR ← Mem(AR):
IR ← DR(codop); PC ← PC + 1;
CPU
PPT 01
ALU
DR
codop
ender
4
11
25
AC
Circuito de acesso à memória
1
AR
4
IR
Controle
2
PC
389

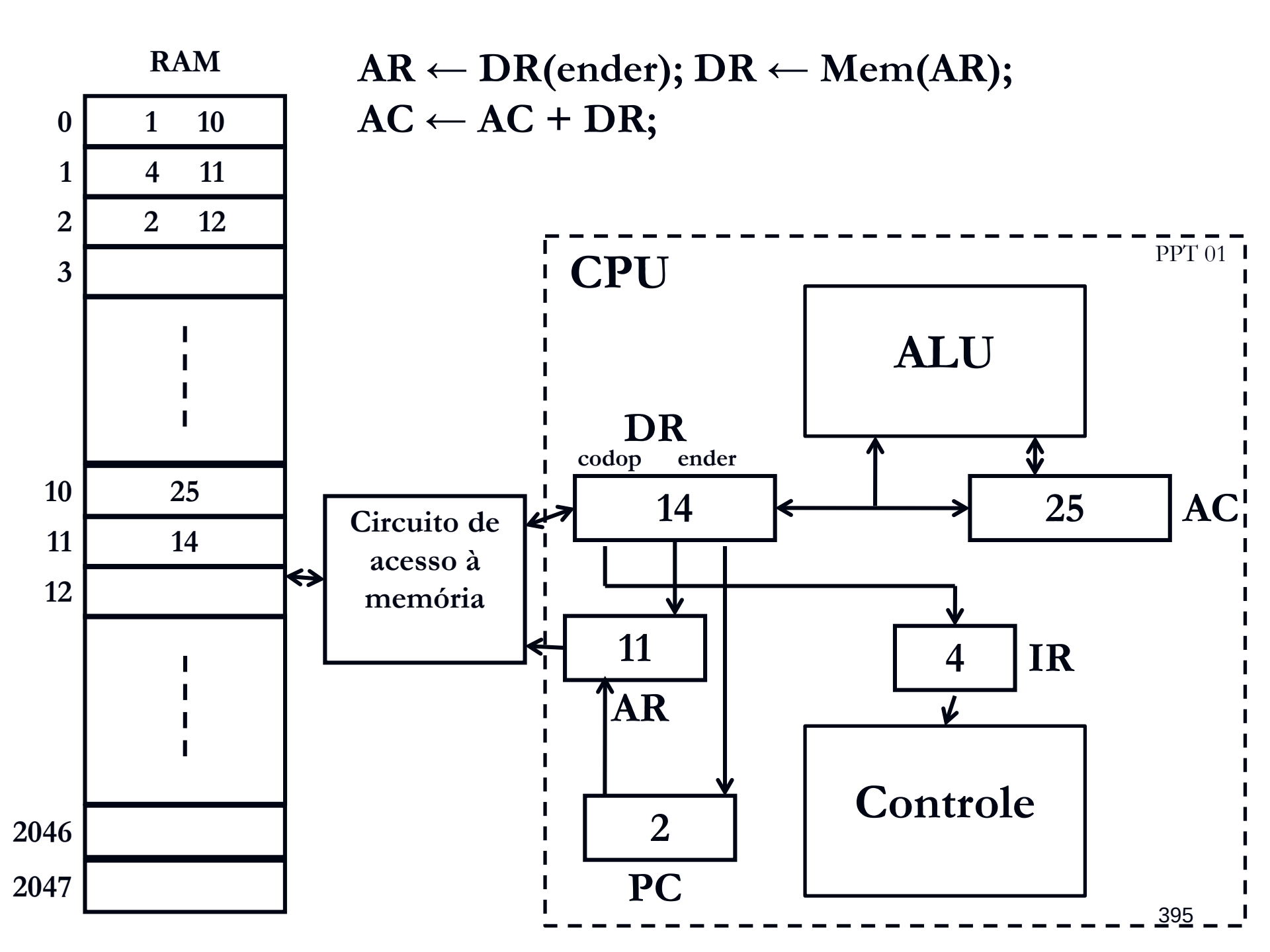

RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
Interpretação: IR = 4
Objetivo: AC ← AC + Mem(ender)
CPU
PPT 01
ALU
DR
codop
ender
4
11
25
AC
Circuito de acesso à memória
1
AR
4
IR
Controle
2
PC
390

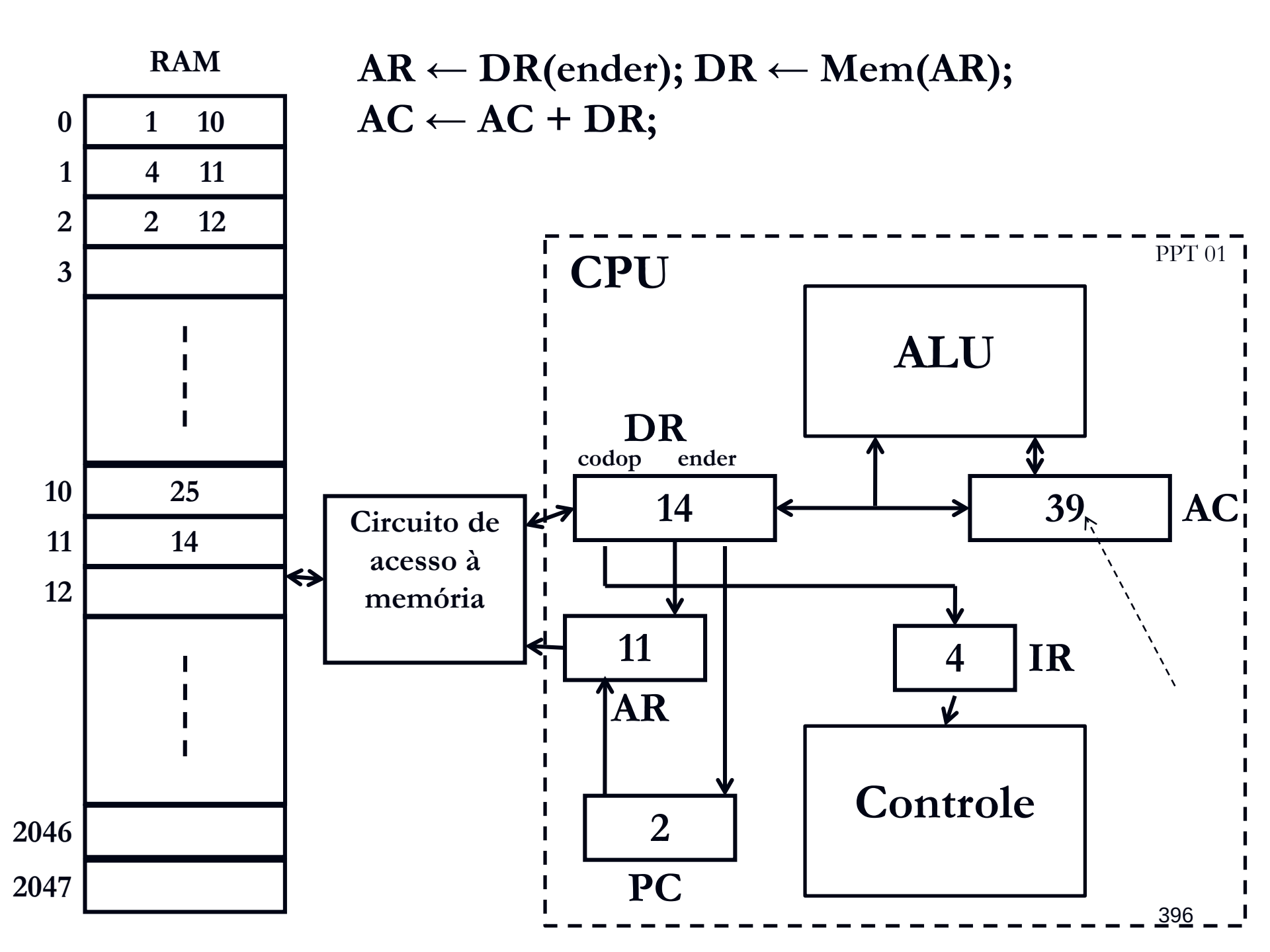

RAM
AR ← DR(ender);
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
DR
codop
ender
4
11
25
AC
1
AR
4
IR
Controle
2
PC
391

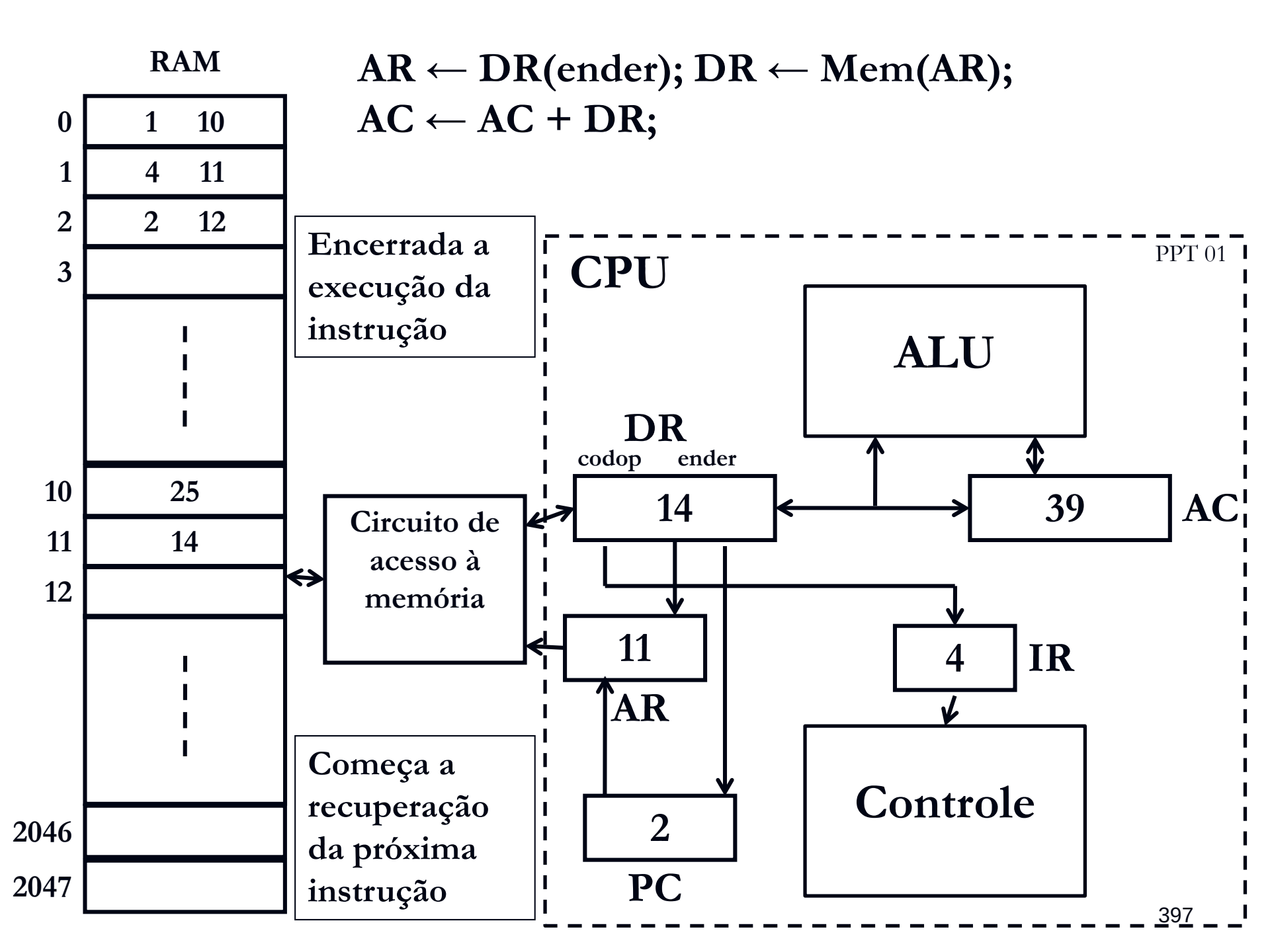

RAM
AR ← DR(ender);
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
CPU
PPT 01
ALU
DR
codop
ender
4
11
25
AC
Circuito de acesso à memória
11
AR
4
IR
Controle
2
PC
392

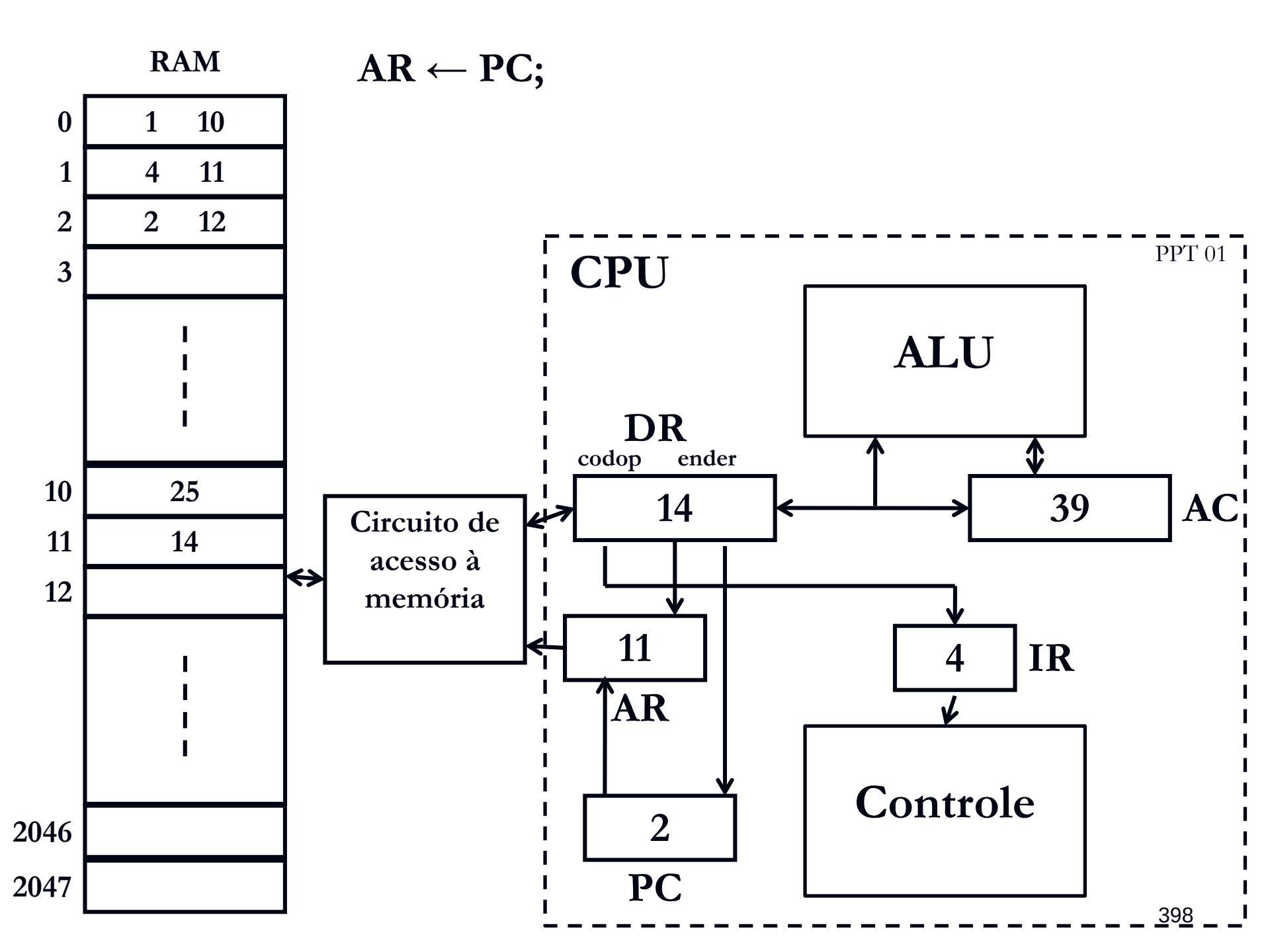

AR ← DR(ender); DR ← Mem(AR);
RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
DR
codop
ender
4
11
25
AC
11
AR
4
IR
Controle
2
PC
393

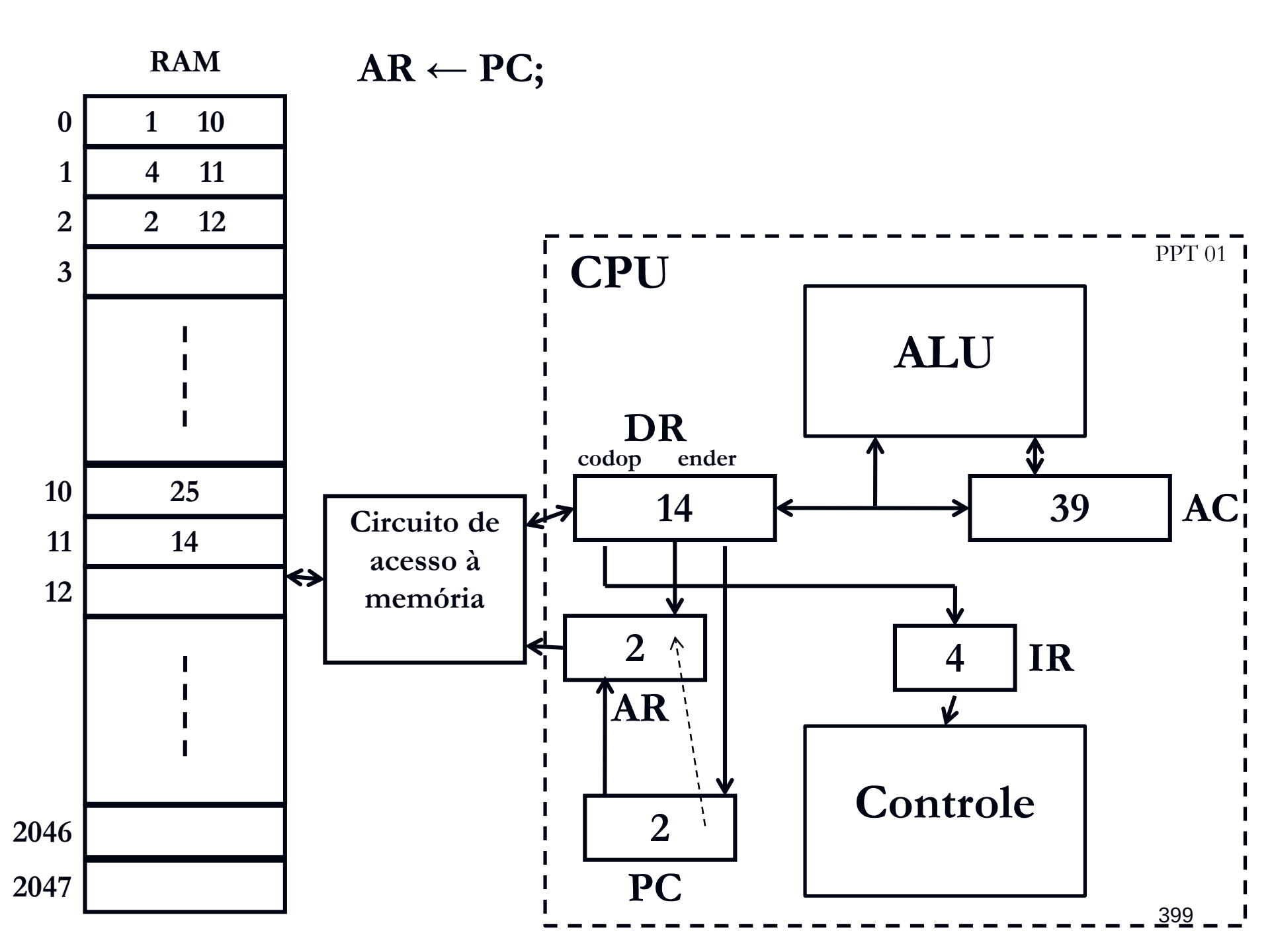

AR ← DR(ender); DR ← Mem(AR);
RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
CPU
PPT 01
ALU
DR
codop ender
14
25
AC
Circuito de acesso à memória
11
AR
4
IR
Controle
2
PC
394

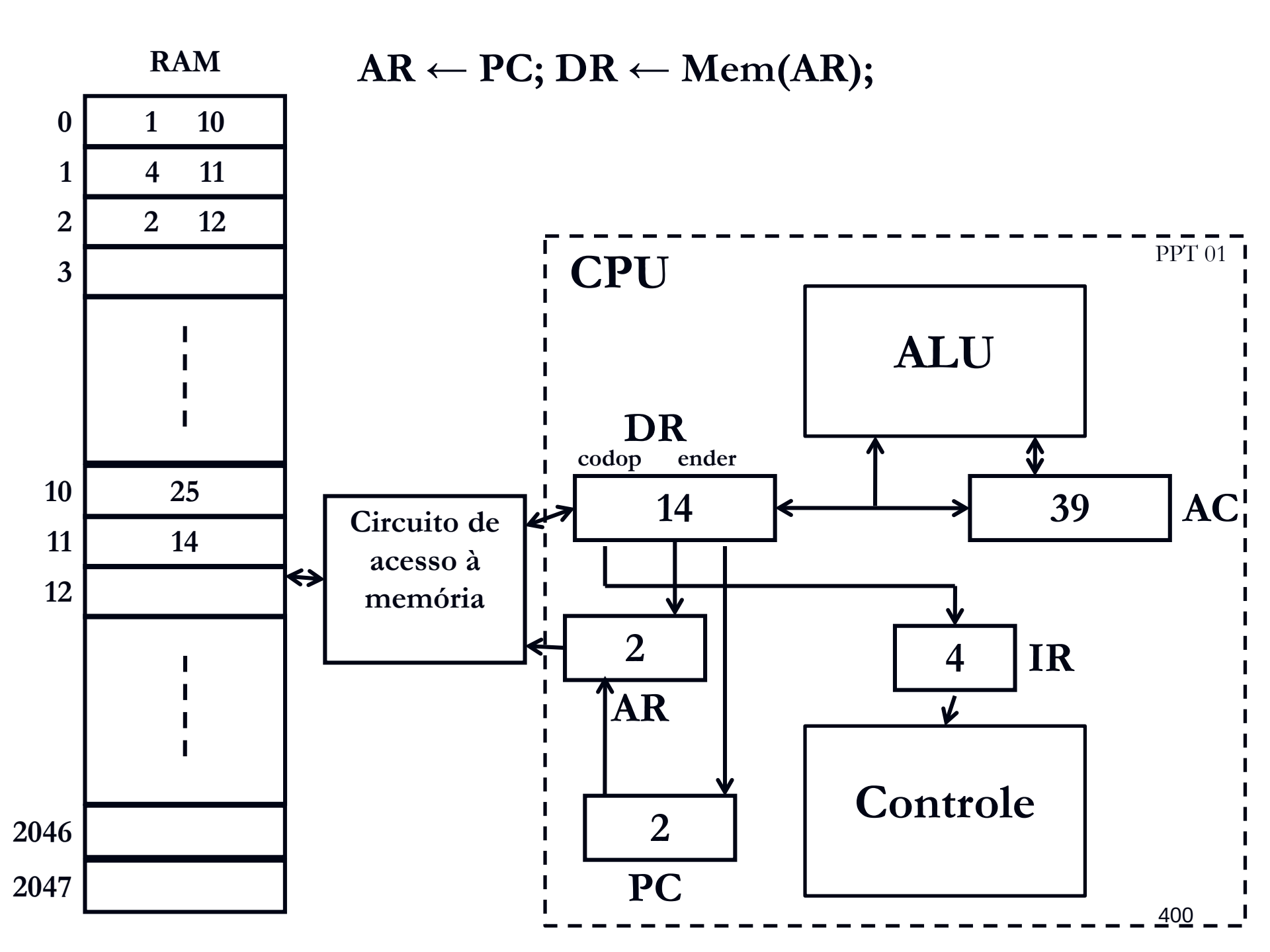

RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
AR ← DR(ender); DR ← Mem(AR);
AC ← AC + DR;
CPU
PPT 01
ALU
DR
codop ender
14
25
AC
Circuito de acesso à memória
11
AR
4
IR
Controle
2
PC
395

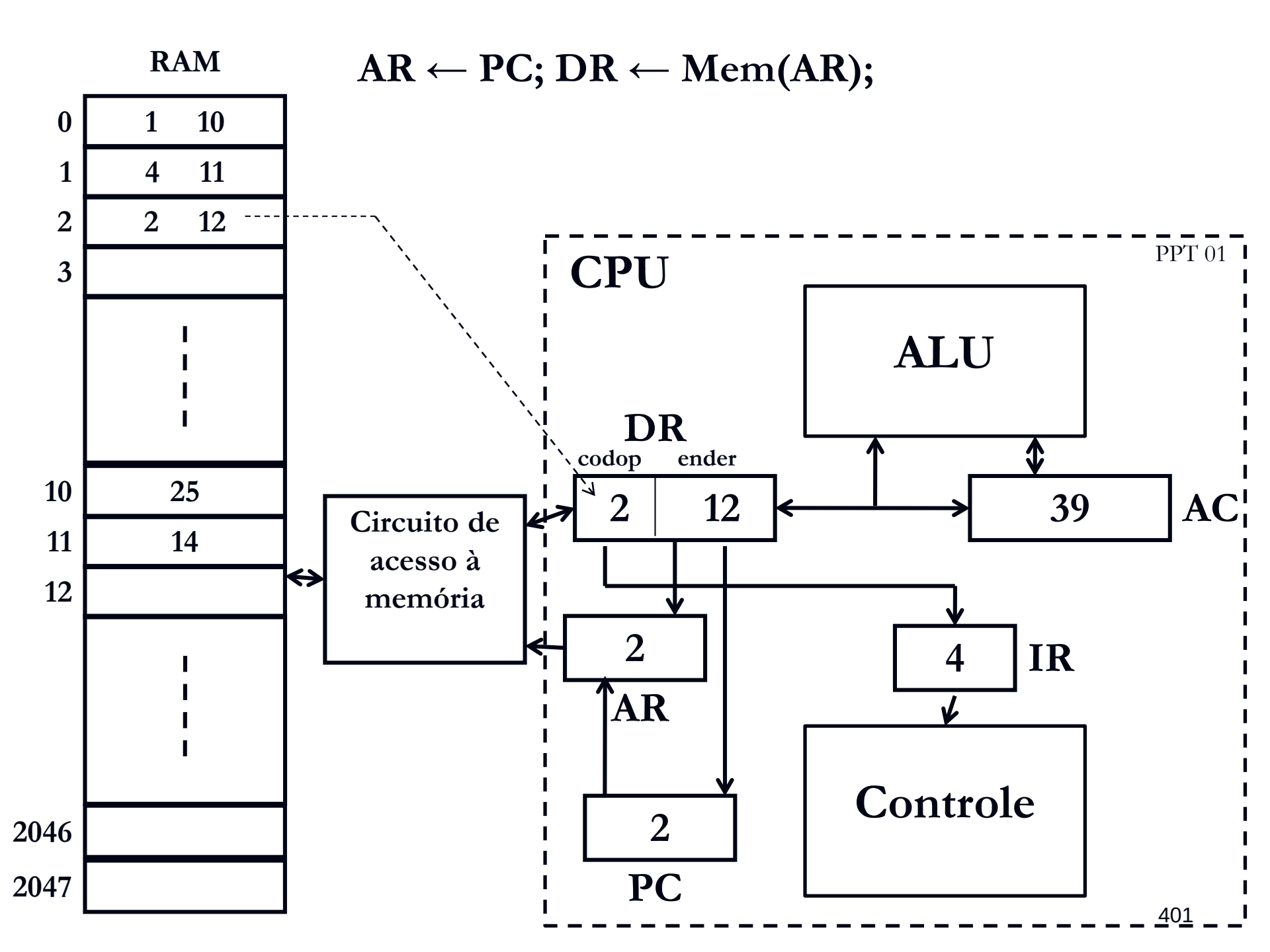

RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
AR ← DR(ender); DR ← Mem(AR);
AC ← AC + DR;
CPU
PPT 01
ALU
DR
codop ender
14
39
AC
Circuito de acesso à memória
11
AR
4
IR
Controle
2
PC
396

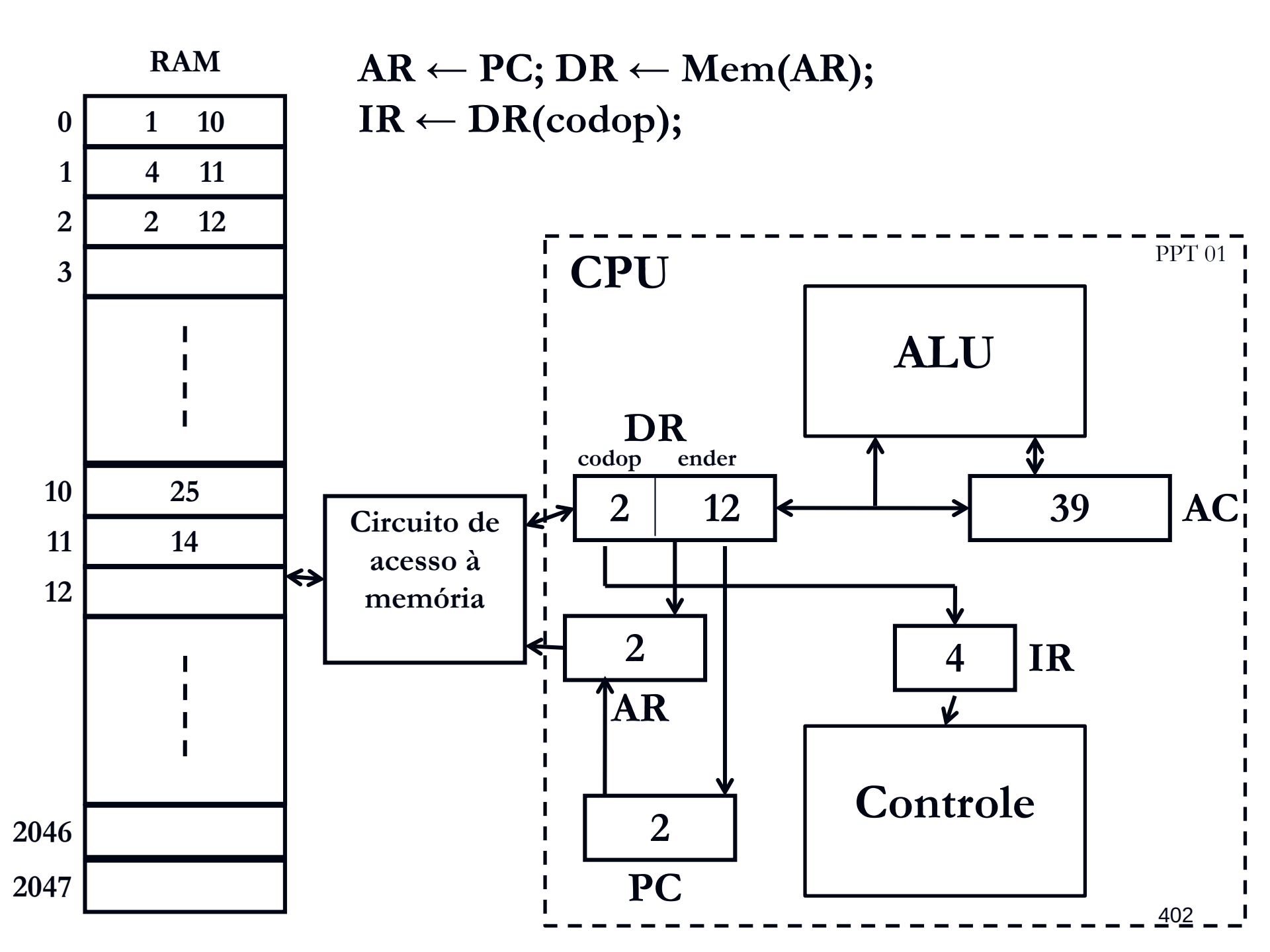

RAM
AR ← DR(ender); DR ← Mem(AR);
AC ← AC + DR;
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
Encerrada a execução da instrução
Circuito de acesso à memória
Começa a recuperação da próxima instrução
CPU
PPT 01
ALU
DR
codop ender
14
39
AC
11
AR
4
IR
Controle
2
PC
397

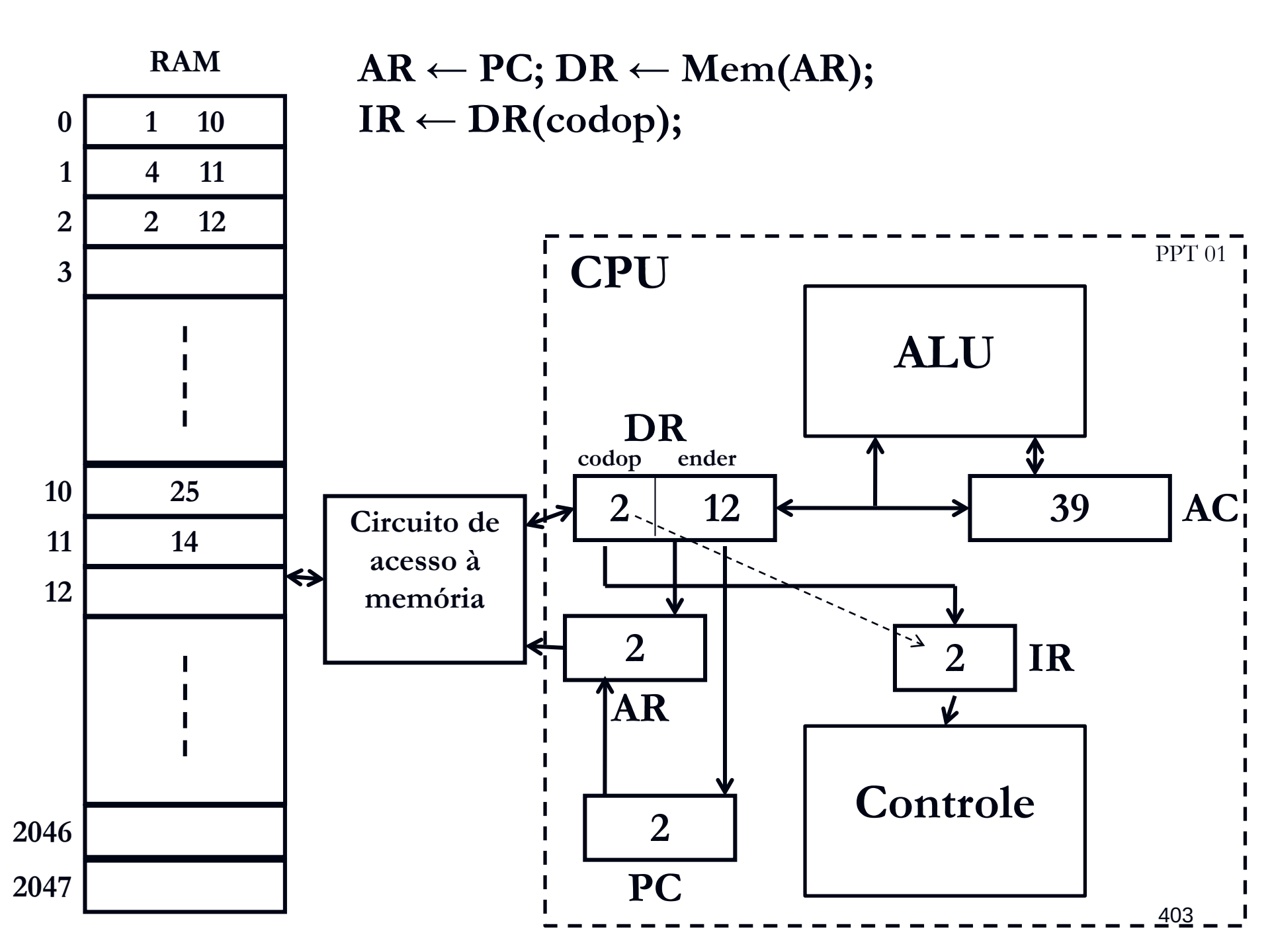

RAM
AR ← PC;
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
DR
codop
ender
14
39
AC
11
AR
4
IR
Controle
2
PC
398

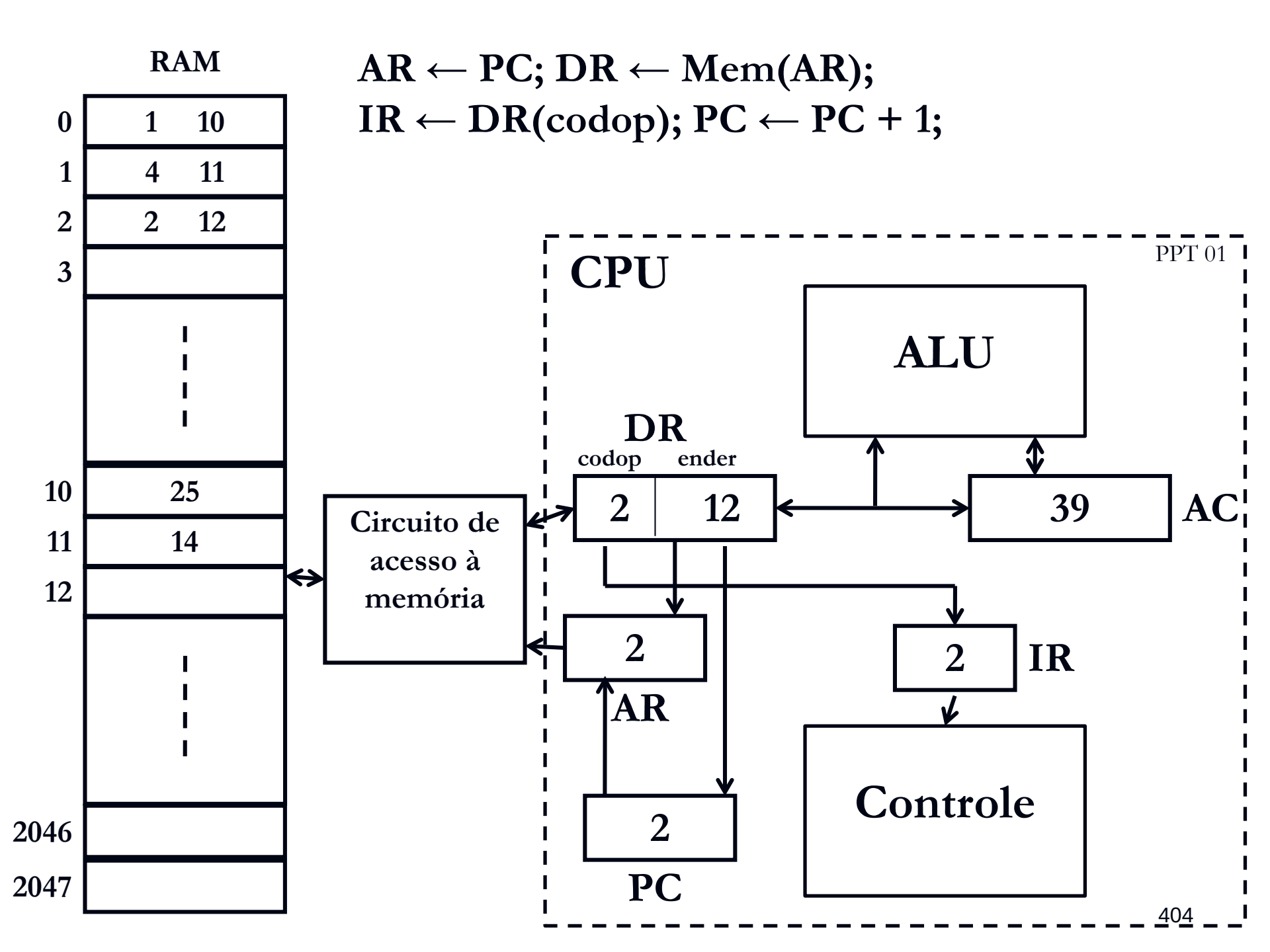

RAM
AR ← PC;
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
DR
codop ender
14
39
AC
2
AR
4
IR
Controle
2
PC
399

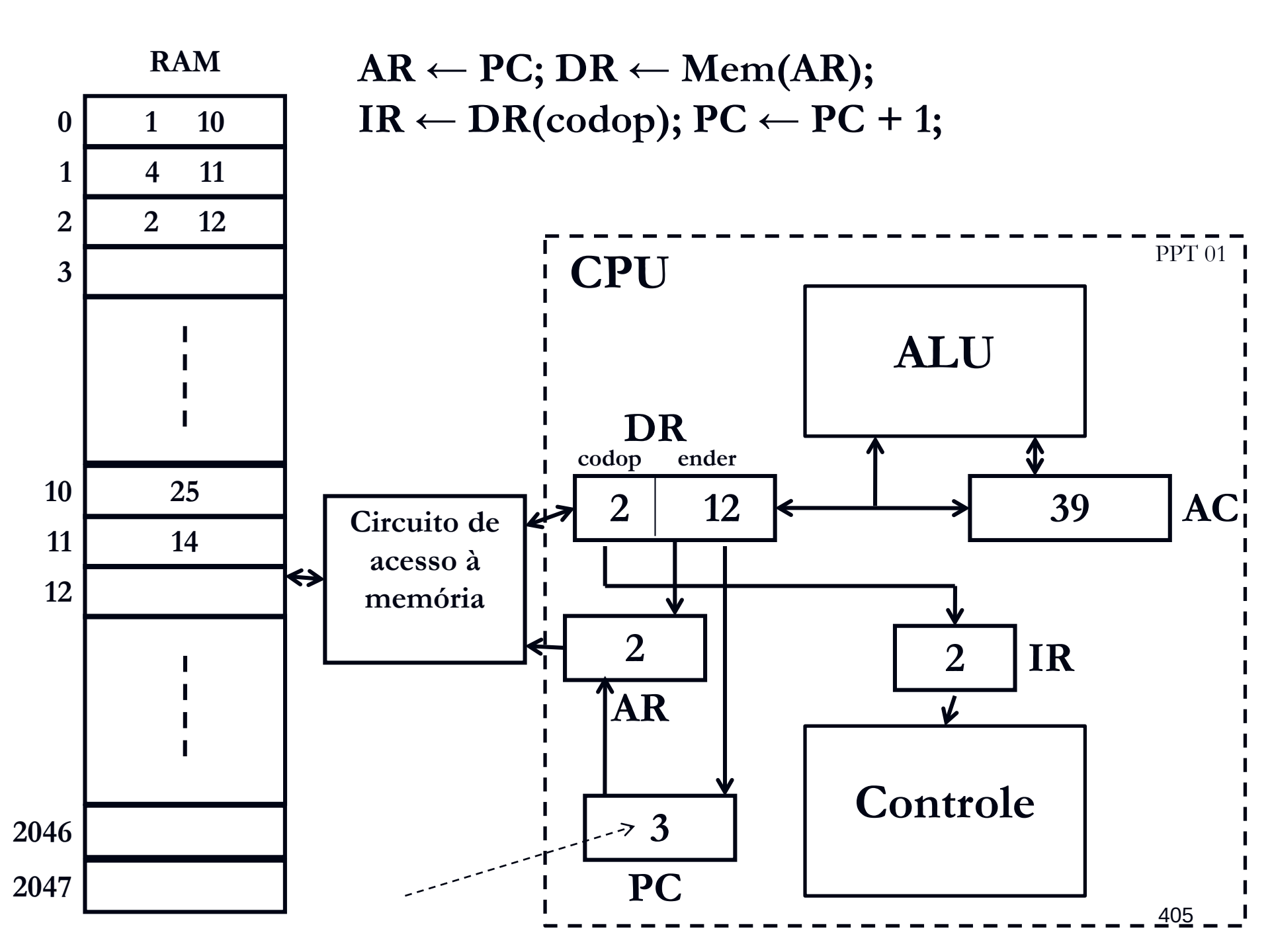

RAM
AR ← PC; DR ← Mem(AR);
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
CPU
PPT 01
ALU
DR
codop
ender
14
39
AC
Circuito de acesso à memória
2
AR
4
IR
Controle
2
PC
400

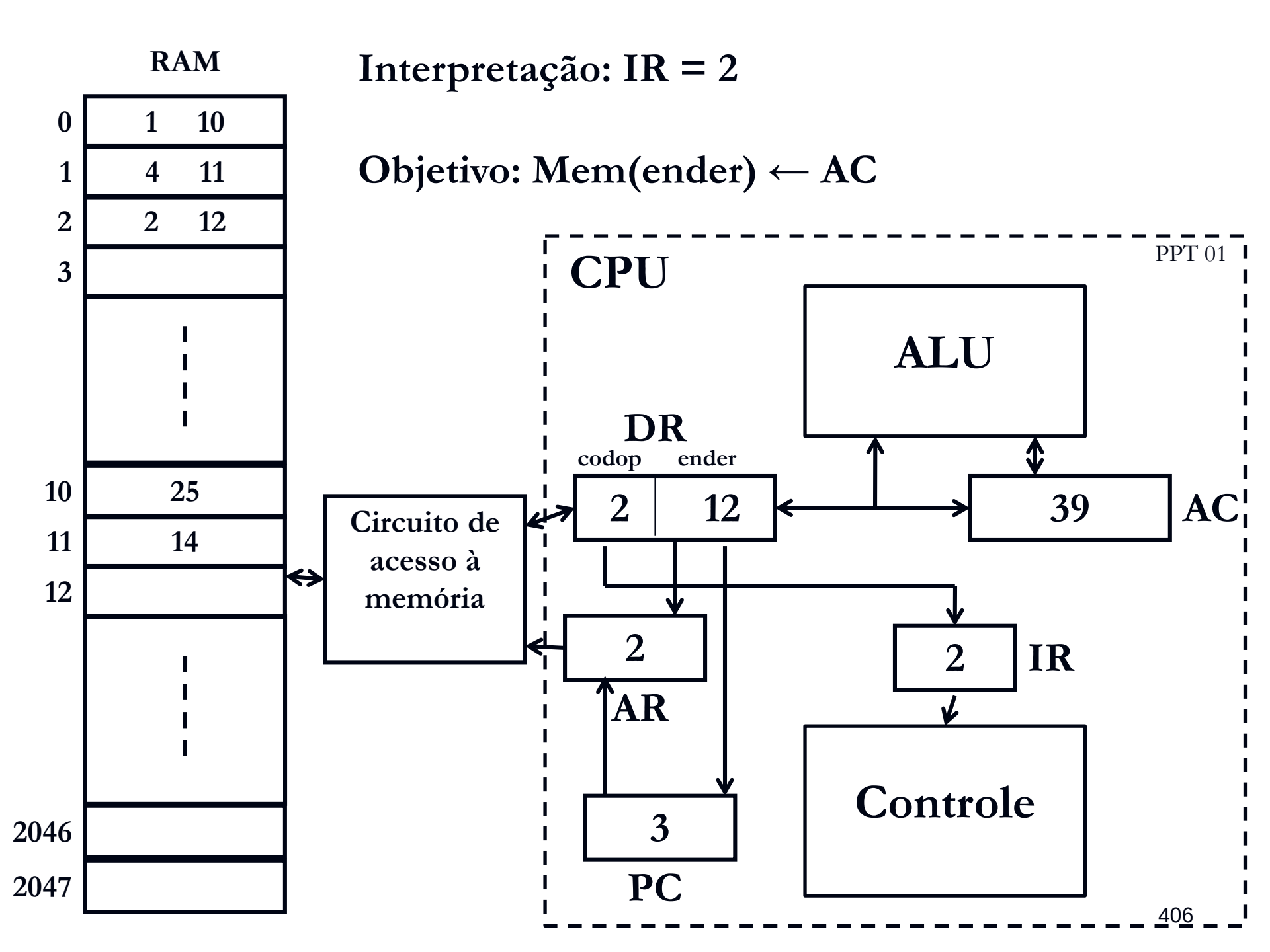

RAM
AR ← PC; DR ← Mem(AR);
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
CPU
PPT 01
ALU
DR
codop
ender
2
12
39
AC
Circuito de acesso à memória
2
AR
4
IR
Controle
2
PC
401

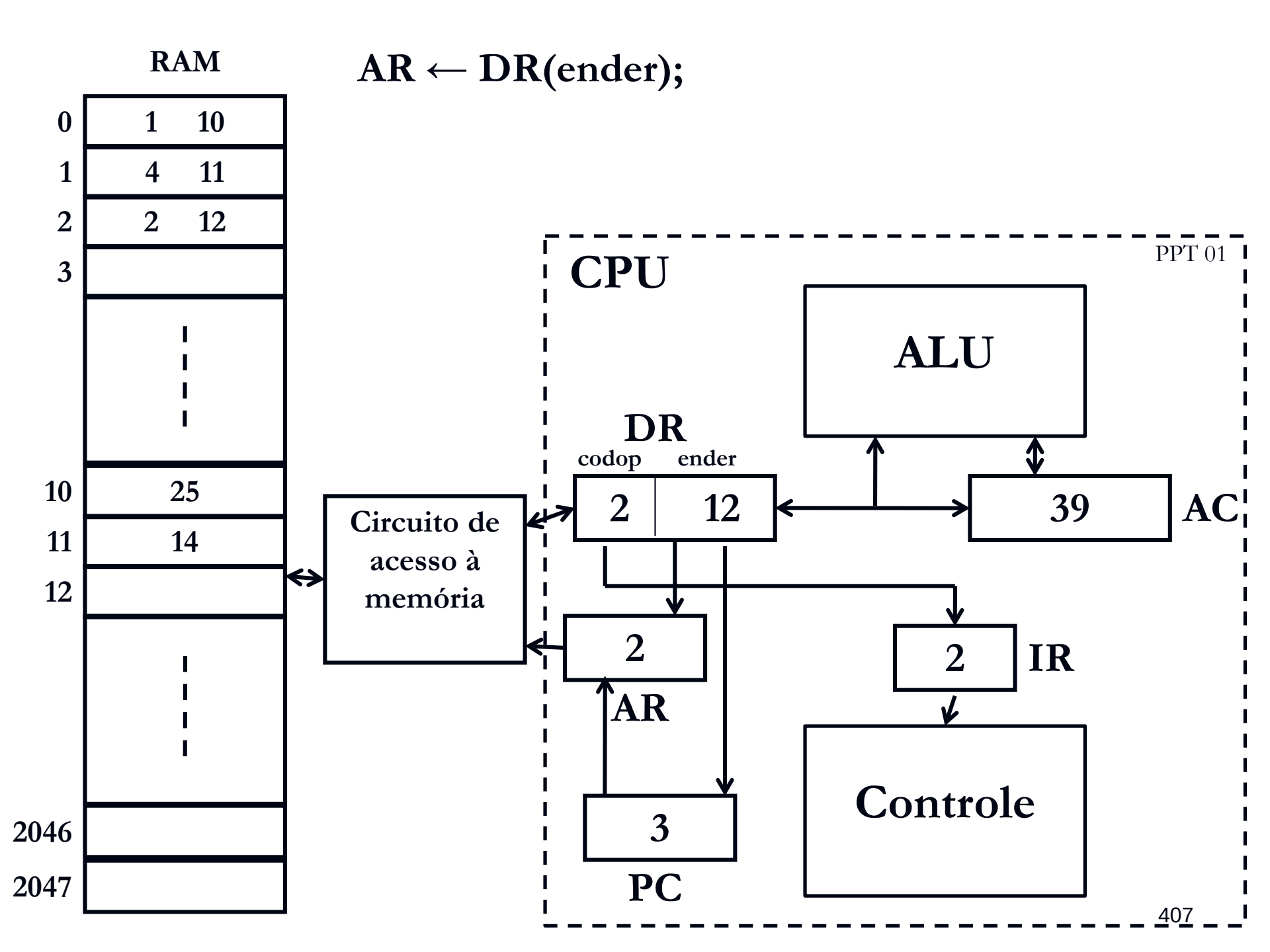

RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
AR ← PC; DR ← Mem(AR);
IR ← DR(codop);
CPU
PPT 01
ALU
DR
codop
ender
2
12
39
AC
Circuito de acesso à memória
2
AR
4
IR
Controle
2
PC
402

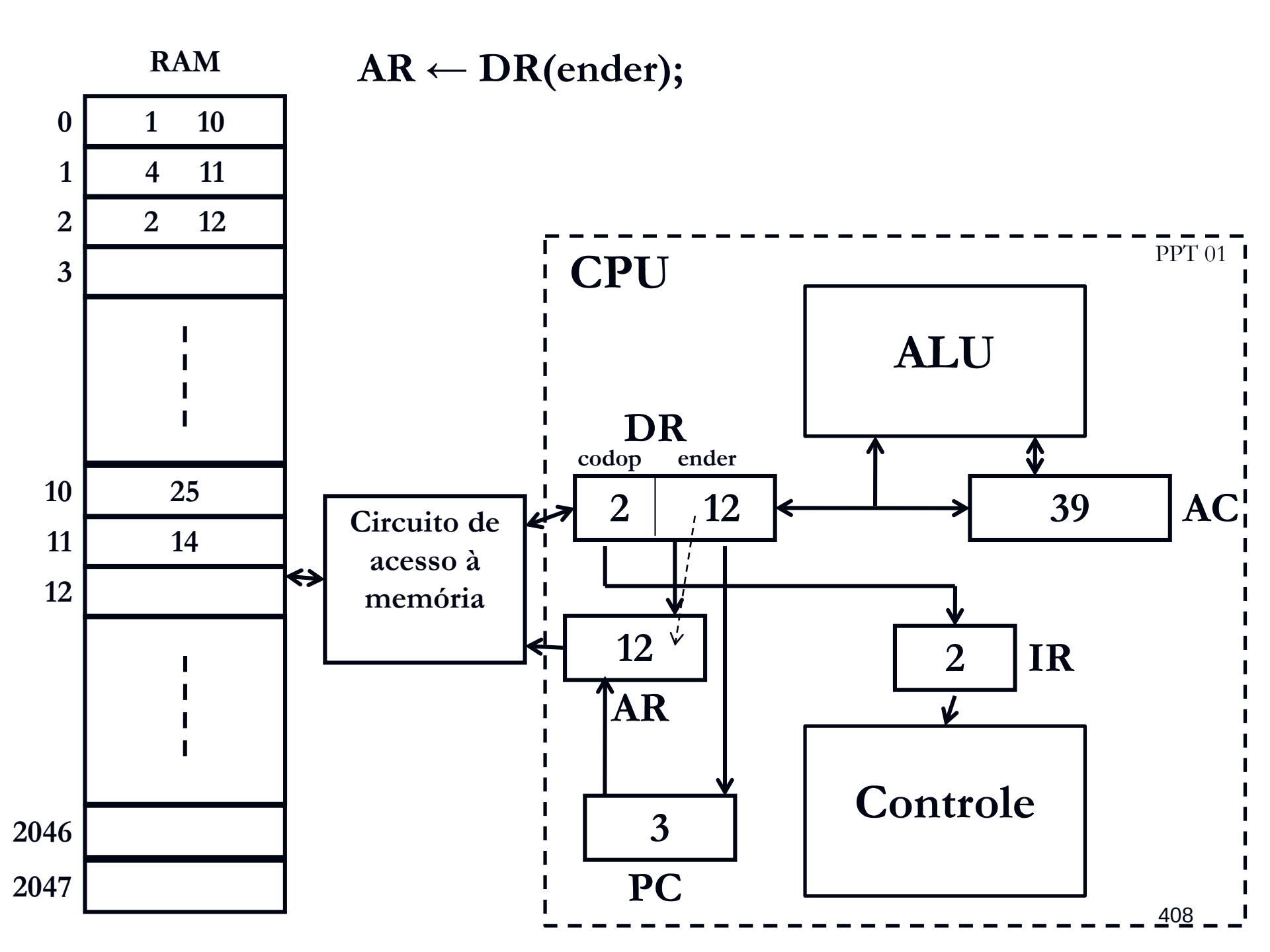

RAM
AR ← PC; DR ← Mem(AR);
IR ← DR(codop);
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
CPU
PPT 01
ALU
DR
codop
ender
2
12
39
AC
Circuito de acesso à memória
2
AR
2
IR
2
PC
Controle
403

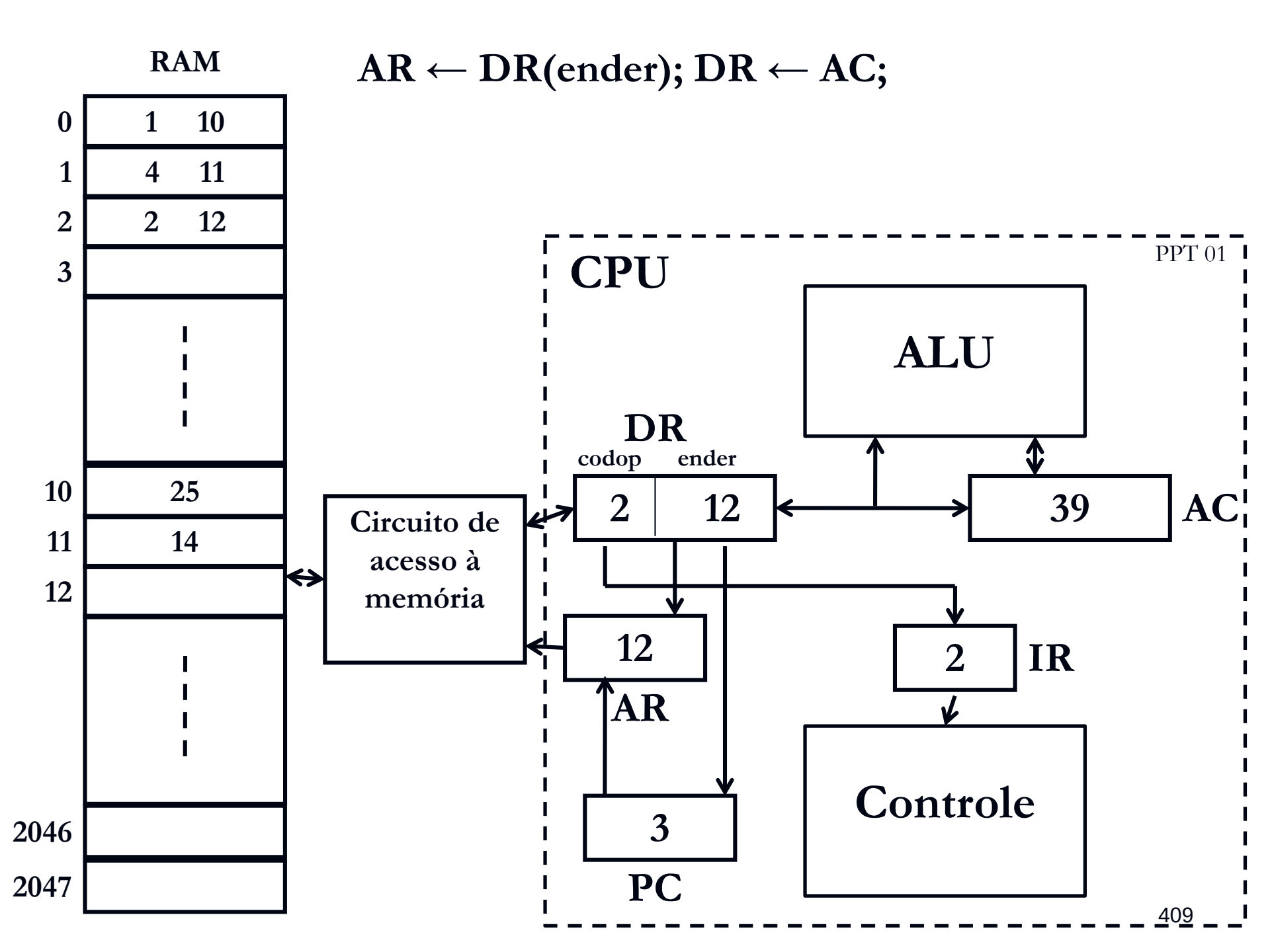

RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
AR ← PC; DR ← Mem(AR);
IR ← DR(codop); PC ← PC + 1;
CPU
PPT 01
ALU
DR
codop
ender
2
12
39
AC
Circuito de acesso à memória
2
AR
2
IR
Controle
2
PC
404

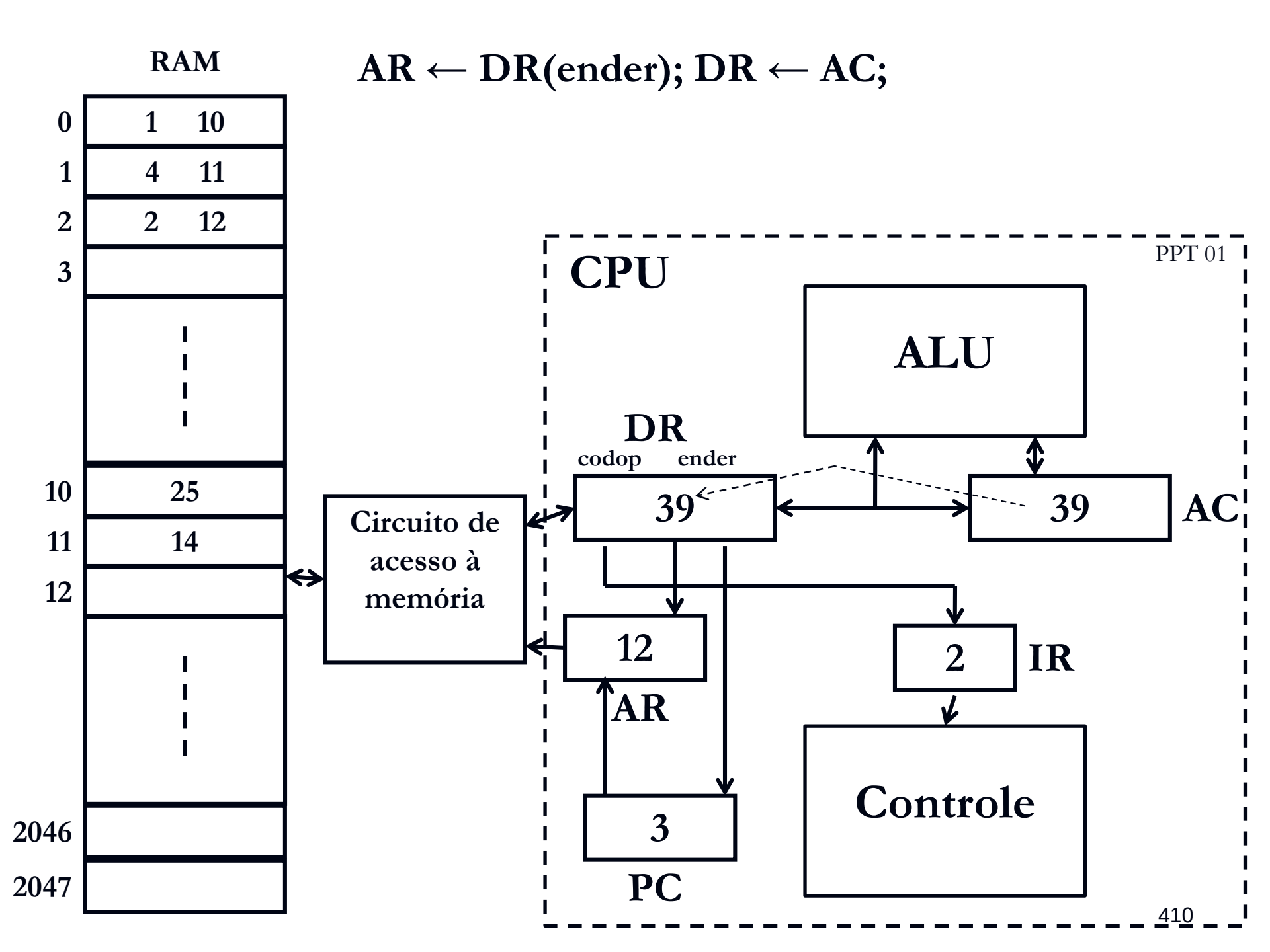

RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
AR ← PC; DR ← Mem(AR);
IR ← DR(codop); PC ← PC + 1;
CPU
PPT 01
ALU
DR
codop
ender
2
12
39
AC
Circuito de acesso à memória
2
AR
2
IR
Controle
3
PC
405

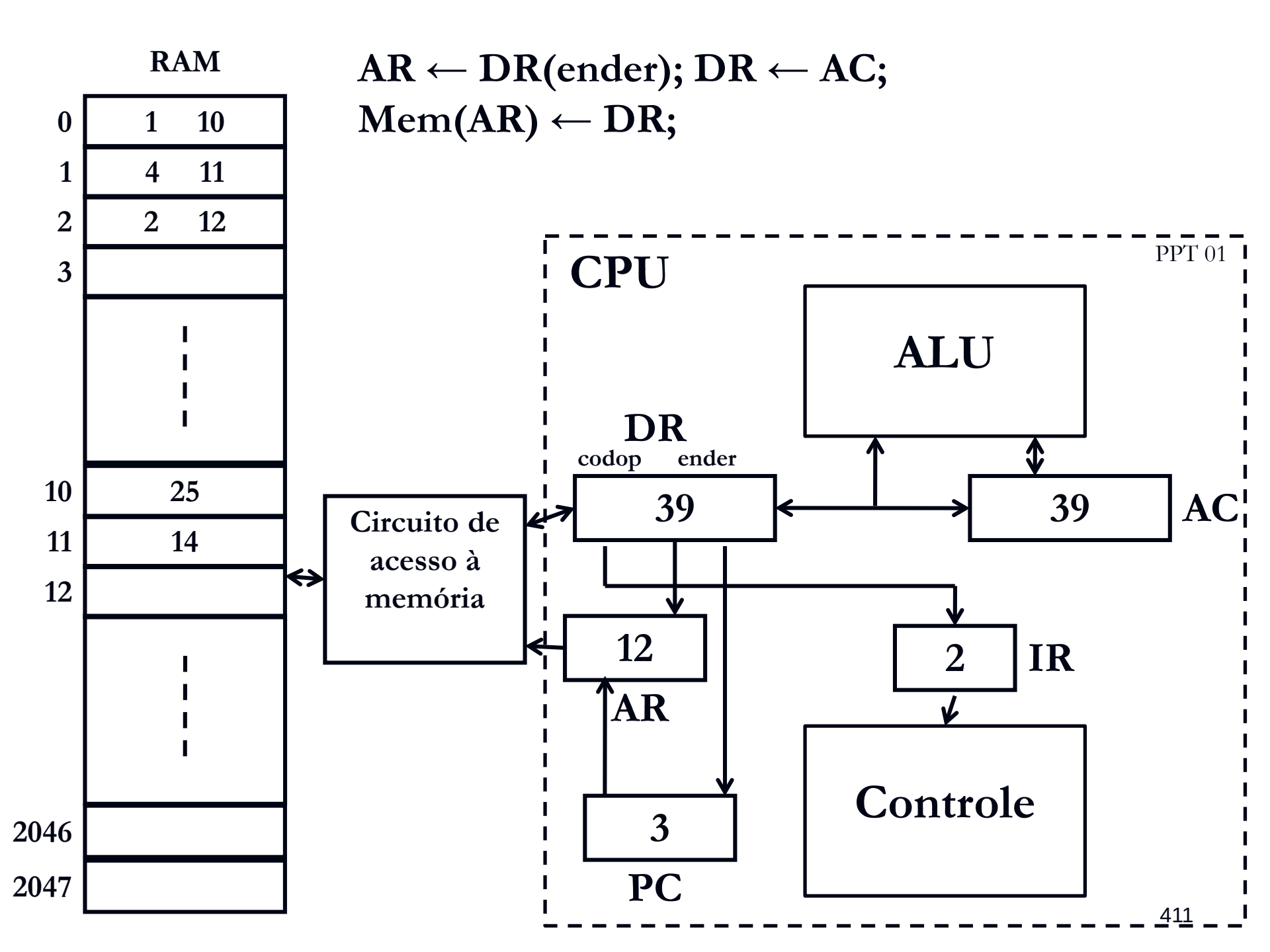

RAM
Interpretação: IR = 2
Objetivo: Mem(ender) ← AC
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
CPU
PPT 01
ALU
DR
codop
ender
2
12
39
AC
Circuito de acesso à memória
2
AR
2
IR
Controle
3
PC
406

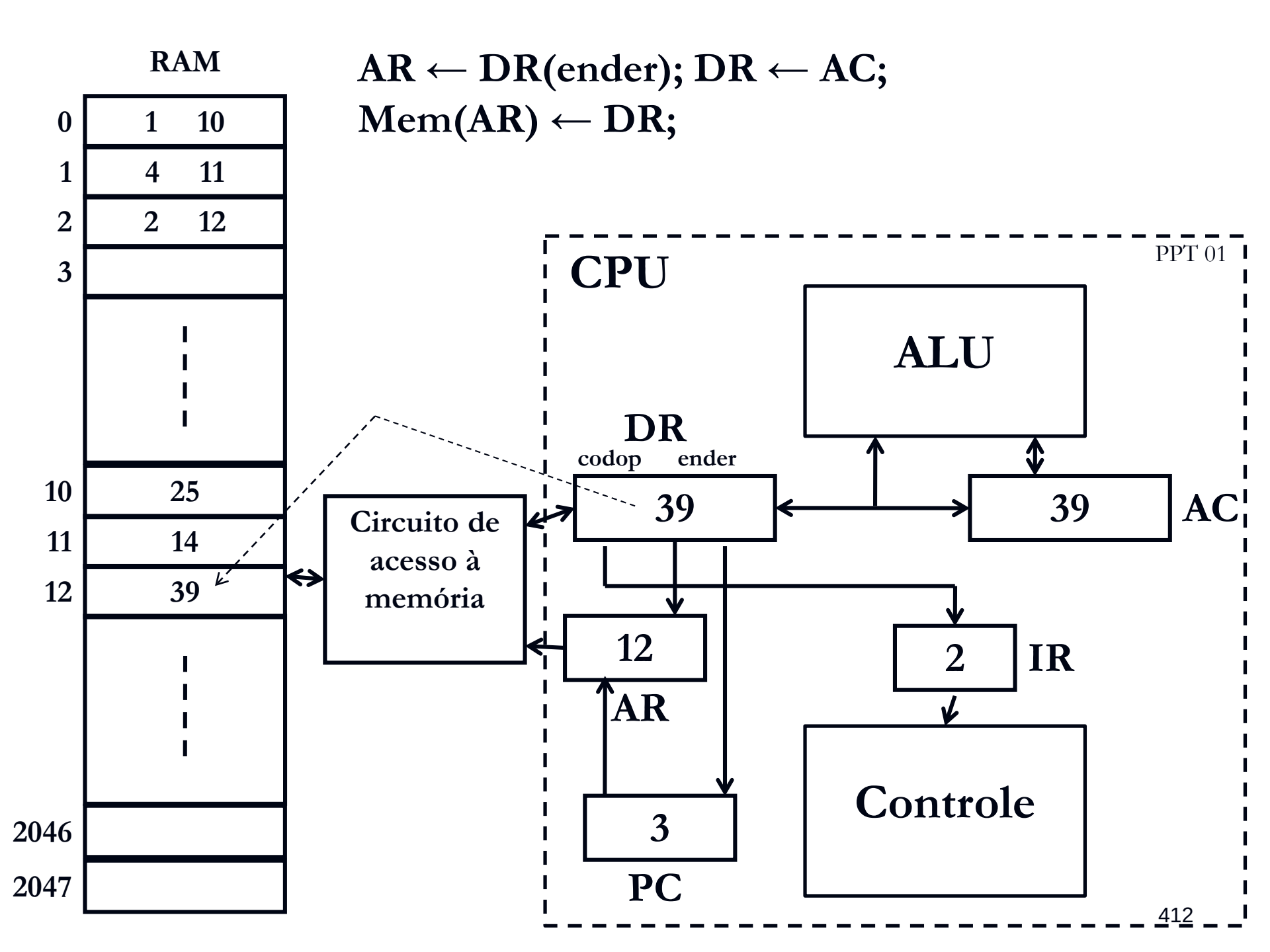

RAM
AR ← DR(ender);
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
CPU
PPT 01
ALU
DR
codop
ender
2
12
39
AC
Circuito de acesso à memória
2
AR
2
IR
Controle
3
PC
407

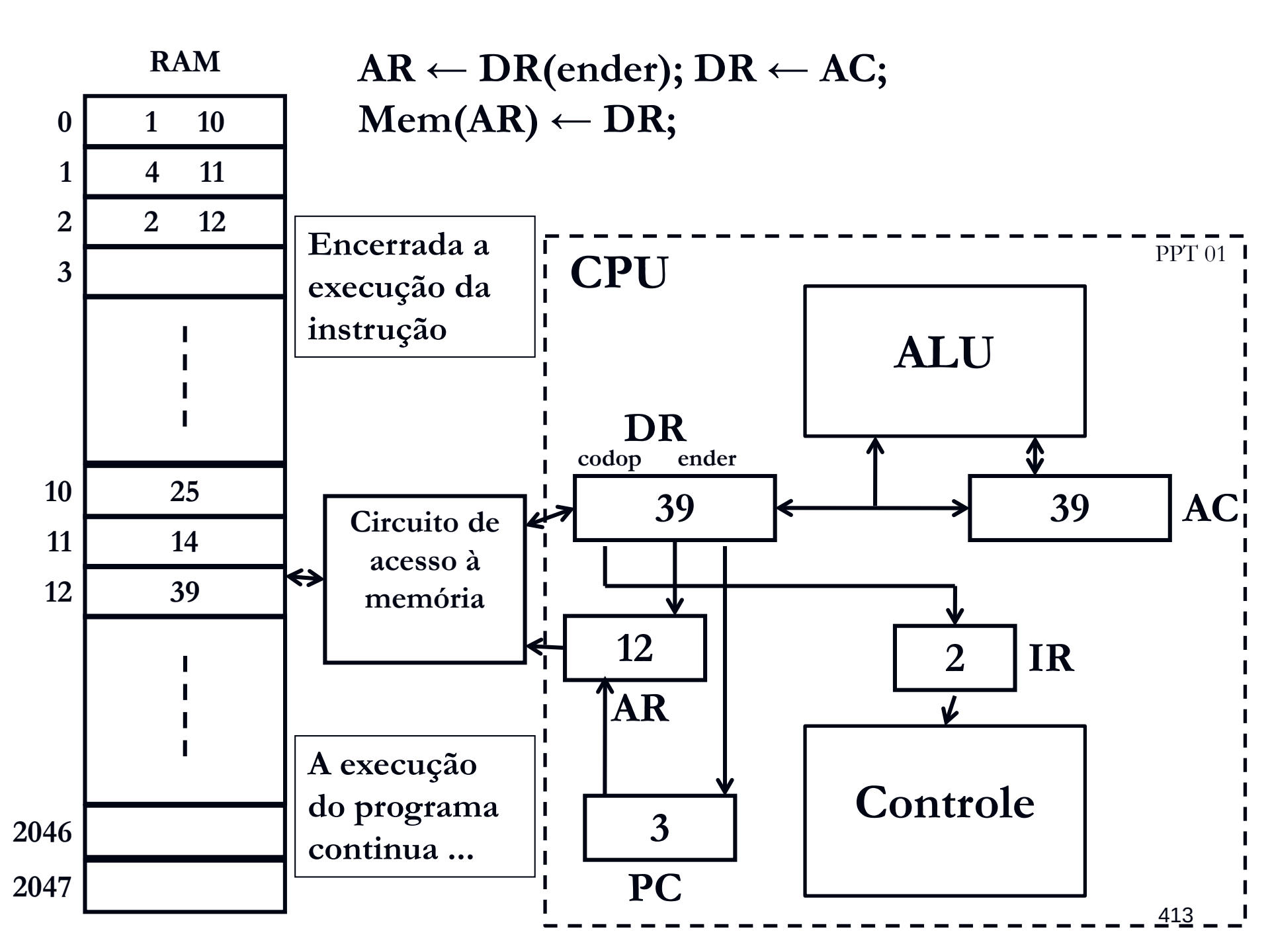

AR ← DR(ender);
RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
DR
codop
ender
2
12
39
AC
12
AR
2
IR
Controle
3
PC
408


AR ← DR(ender); DR ← AC;
RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
CPU
PPT 01
ALU
DR
codop
ender
2
12
39
AC
Circuito de acesso à memória
12
AR
2
IR
Controle
3
PC
409

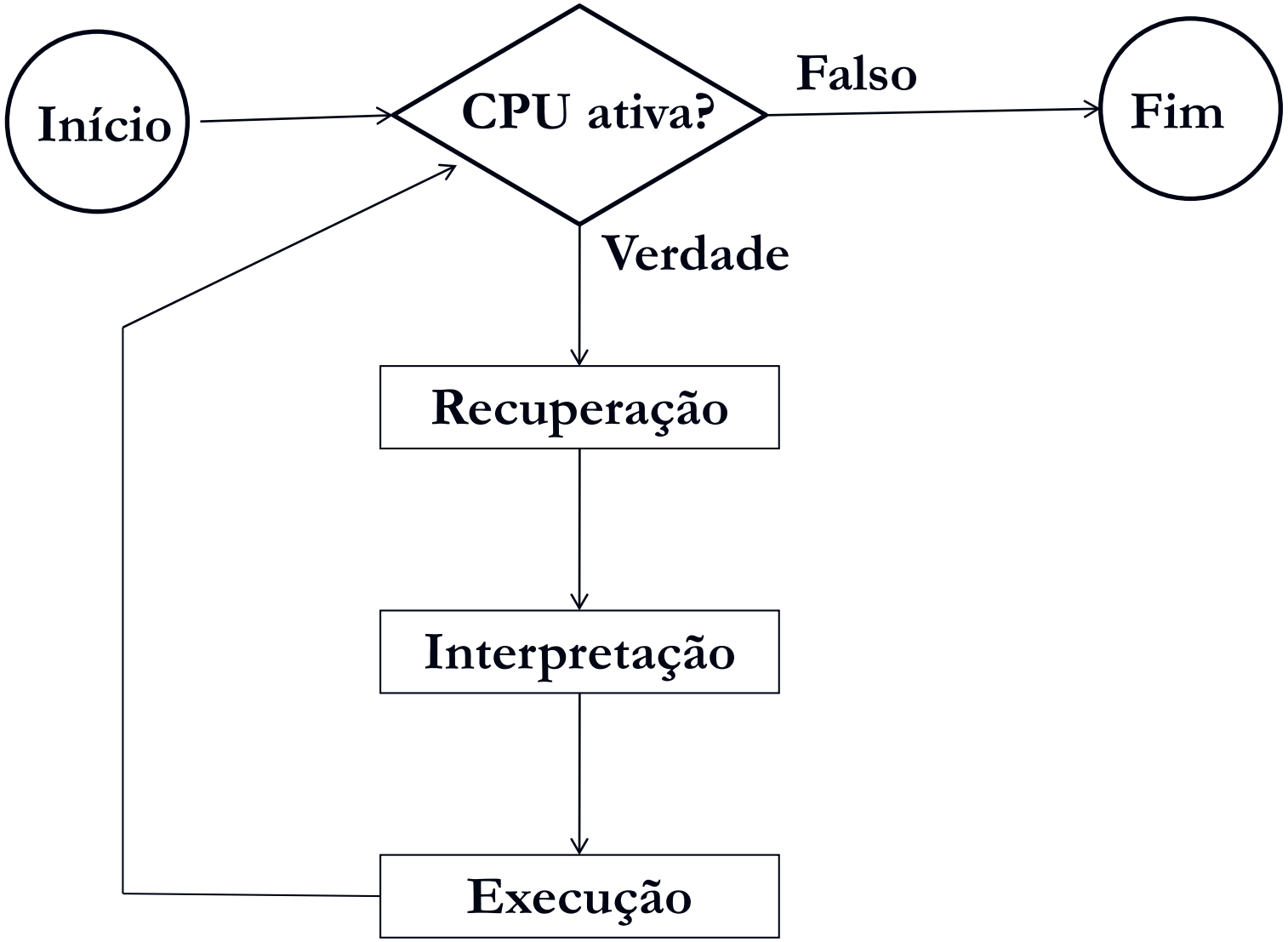

AR ← DR(ender); DR ← AC;
RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
DR
codop ender
39
39
AC
12
AR
2
IR
Controle
3
PC
410


RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
AR ← DR(ender); DR ← AC;
Mem(AR) ← DR;
CPU
PPT 01
ALU
DR
codop ender
39
39
AC
Circuito de acesso à memória
12
AR
2
IR
Controle
3
PC
411

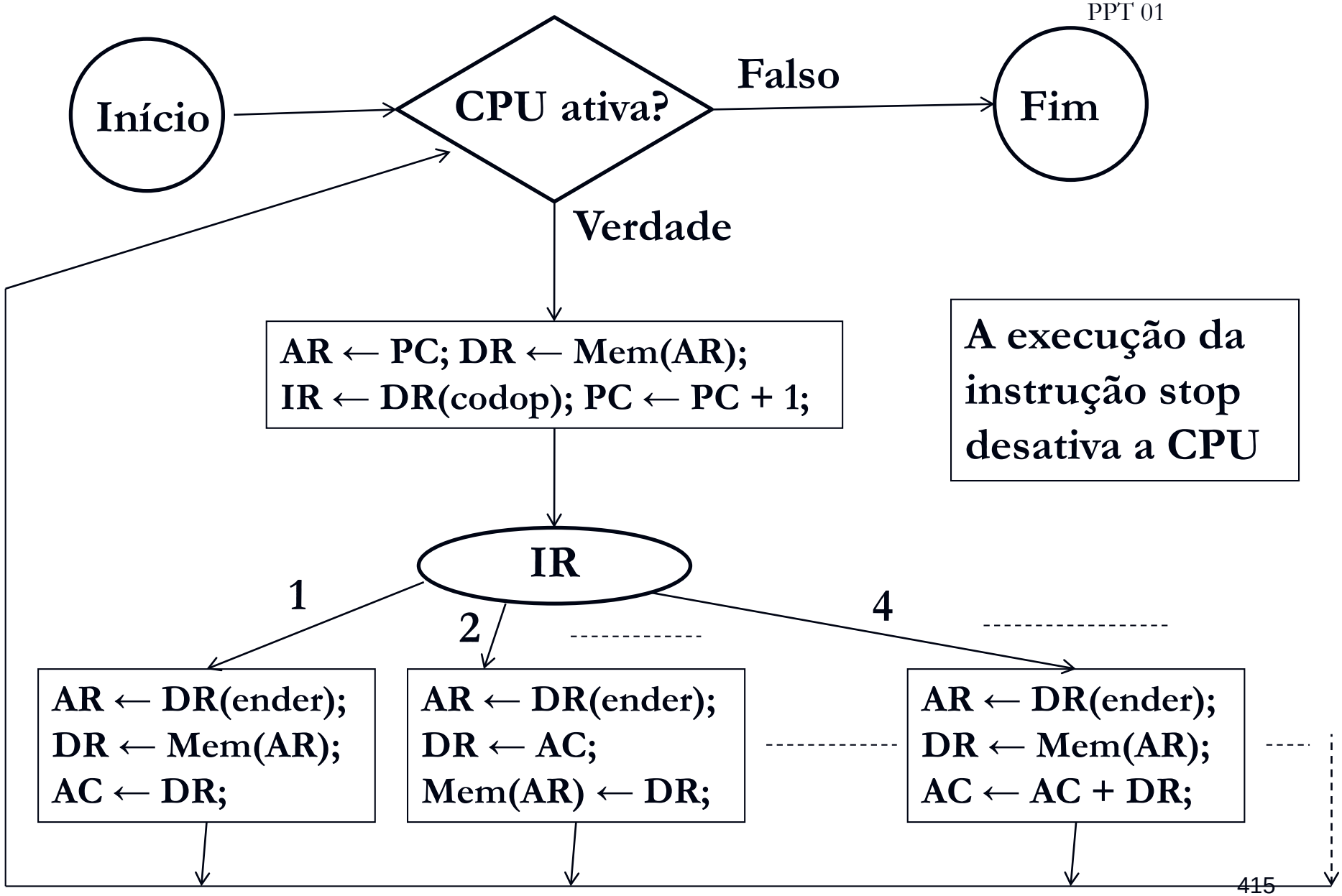

RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
39
2046
2047
AR ← DR(ender); DR ← AC;
Mem(AR) ← DR;
CPU
PPT 01
ALU
DR
codop ender
39
39
AC
Circuito de acesso à memória
12
AR
2
IR
Controle
3
PC
412

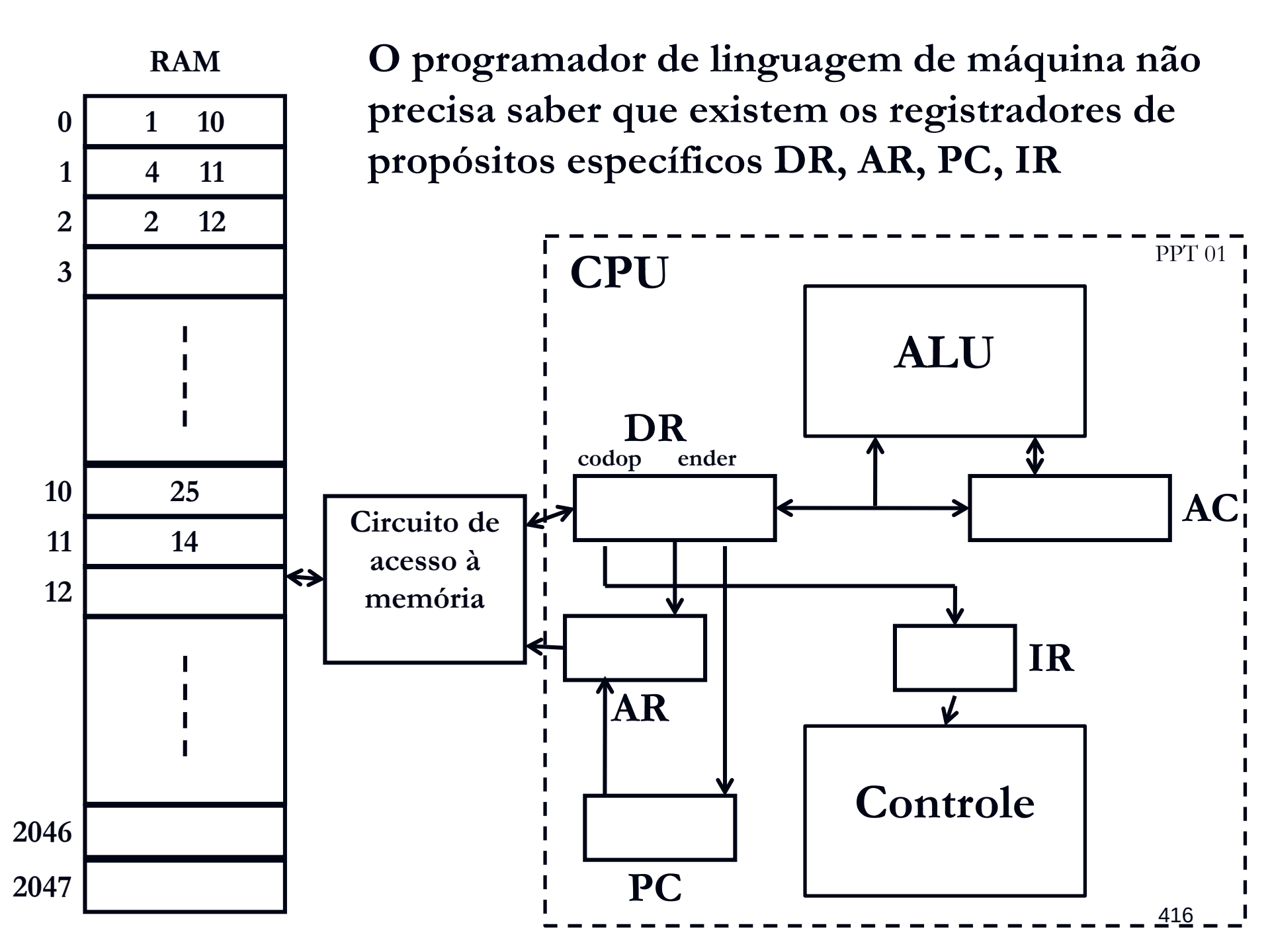

RAM
AR ← DR(ender); DR ← AC;
Mem(AR) ← DR;
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
39
2046
2047
Encerrada a execução da instrução
A execução do programa continua ...
Circuito de acesso à memória
CPU
PPT 01
ALU
DR
codop ender
39
39
AC
12
AR
2
IR
Controle
3
PC
413

Ciclo de execução das instruções:

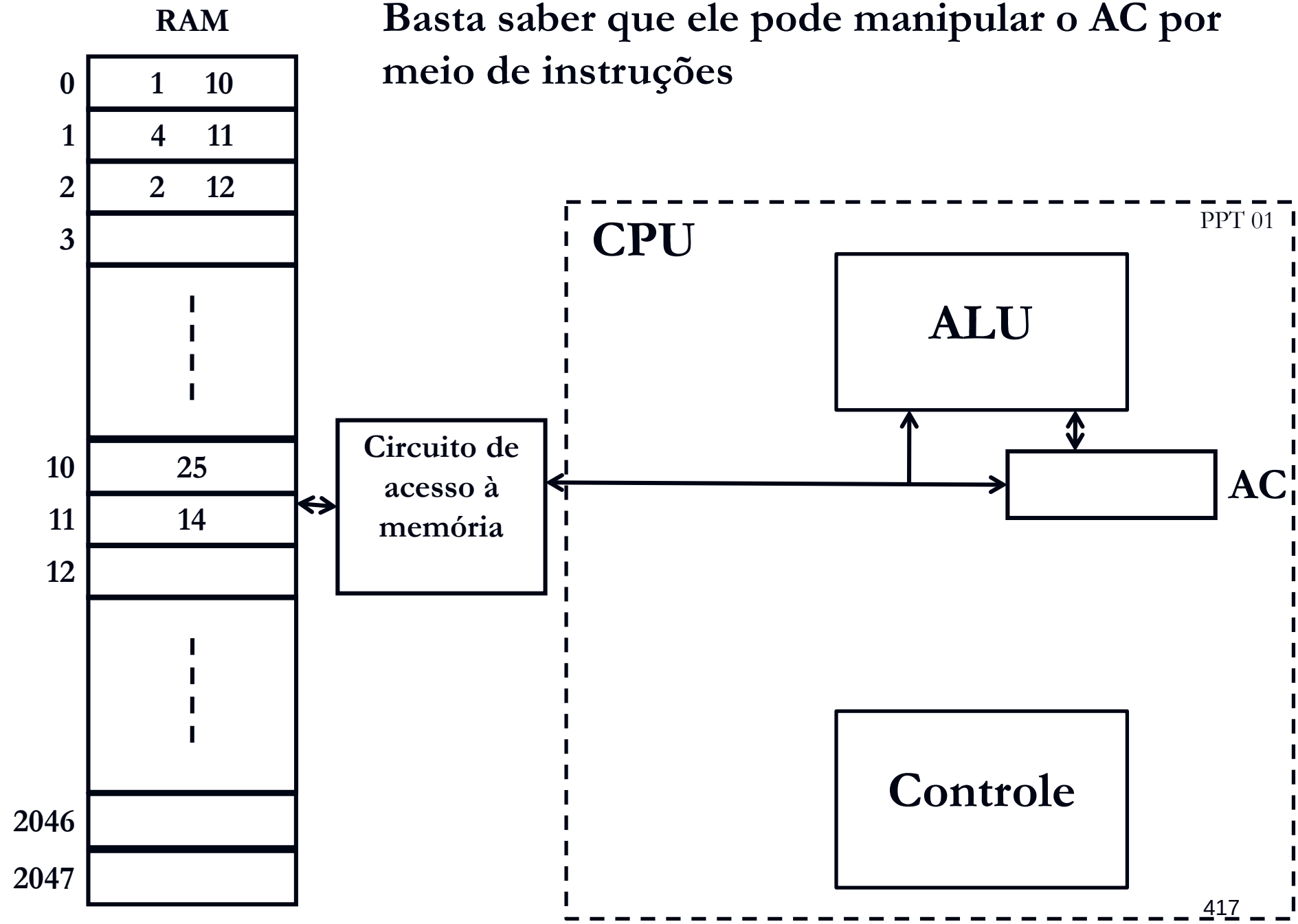


Microprograma da CPU:

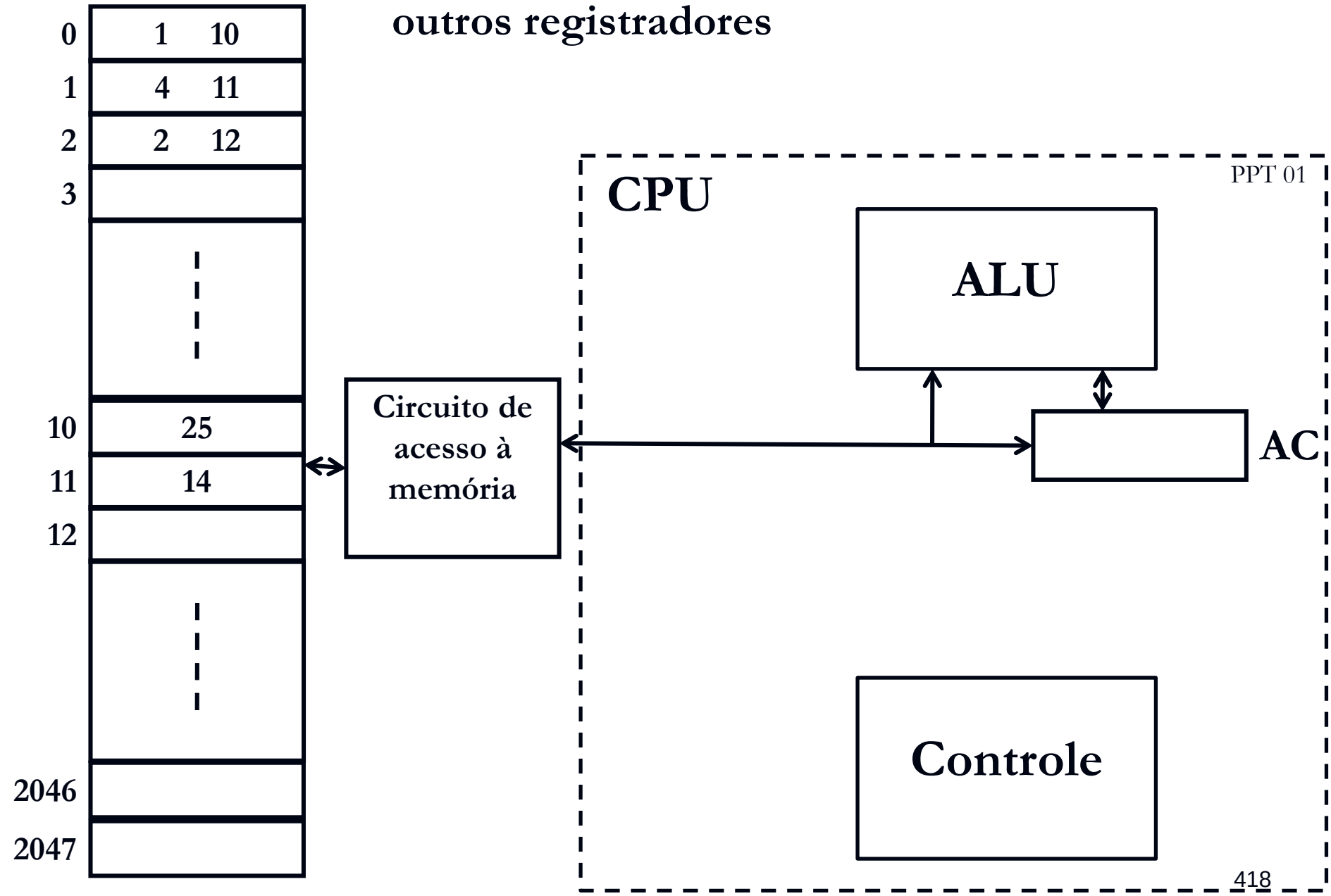


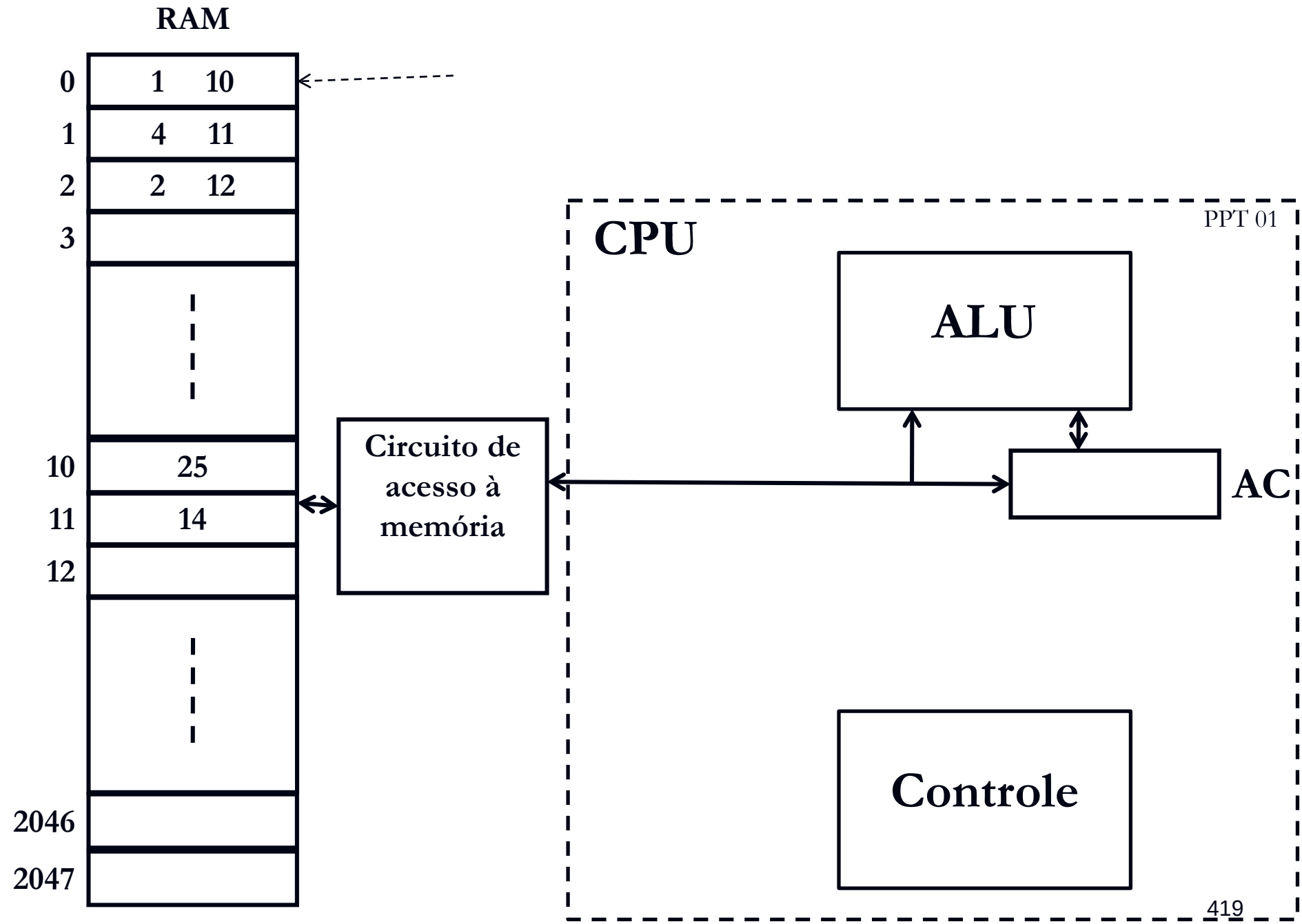

RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
O programador de linguagem de máquina não precisa saber que existem os registradores de propósitos específicos DR, AR, PC, IR
CPU
PPT 01
ALU
DR
codop ender
AC
Circuito de acesso à memória
IR
AR
Controle
PC
416

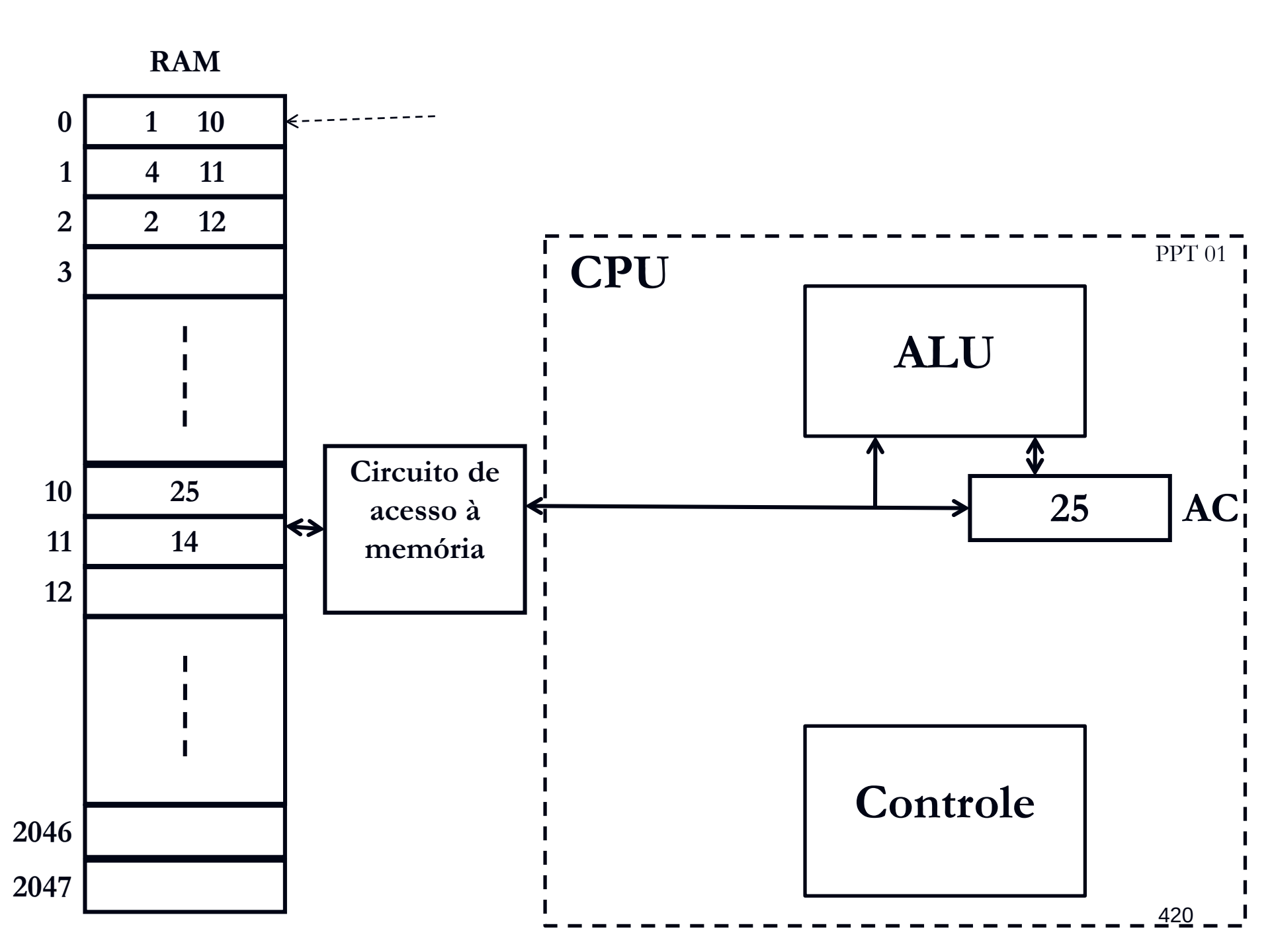
Basta saber que ele pode manipular o AC por meio de instruções

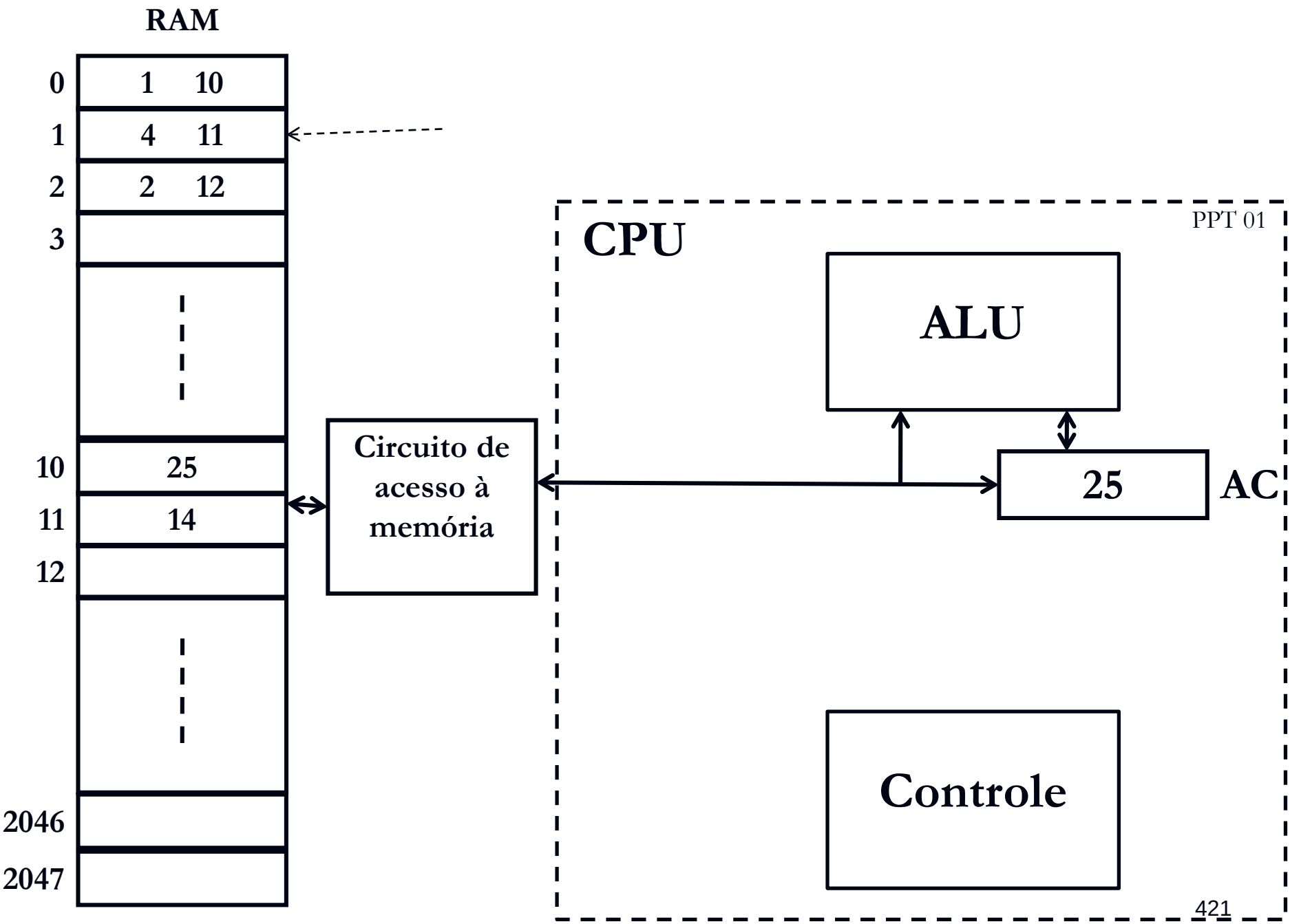


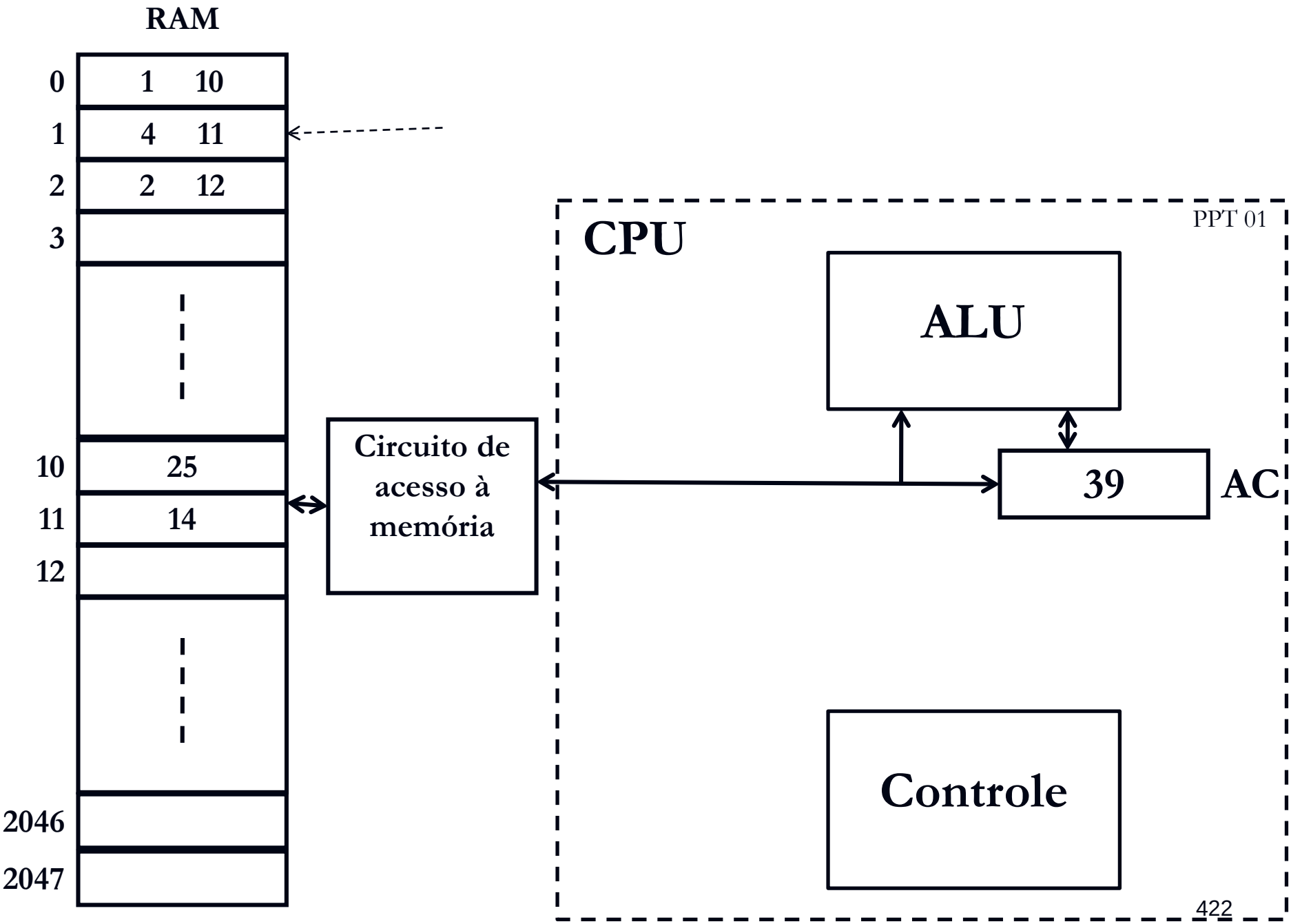
Seja então a execução, abstraindo-se dos outros registradores

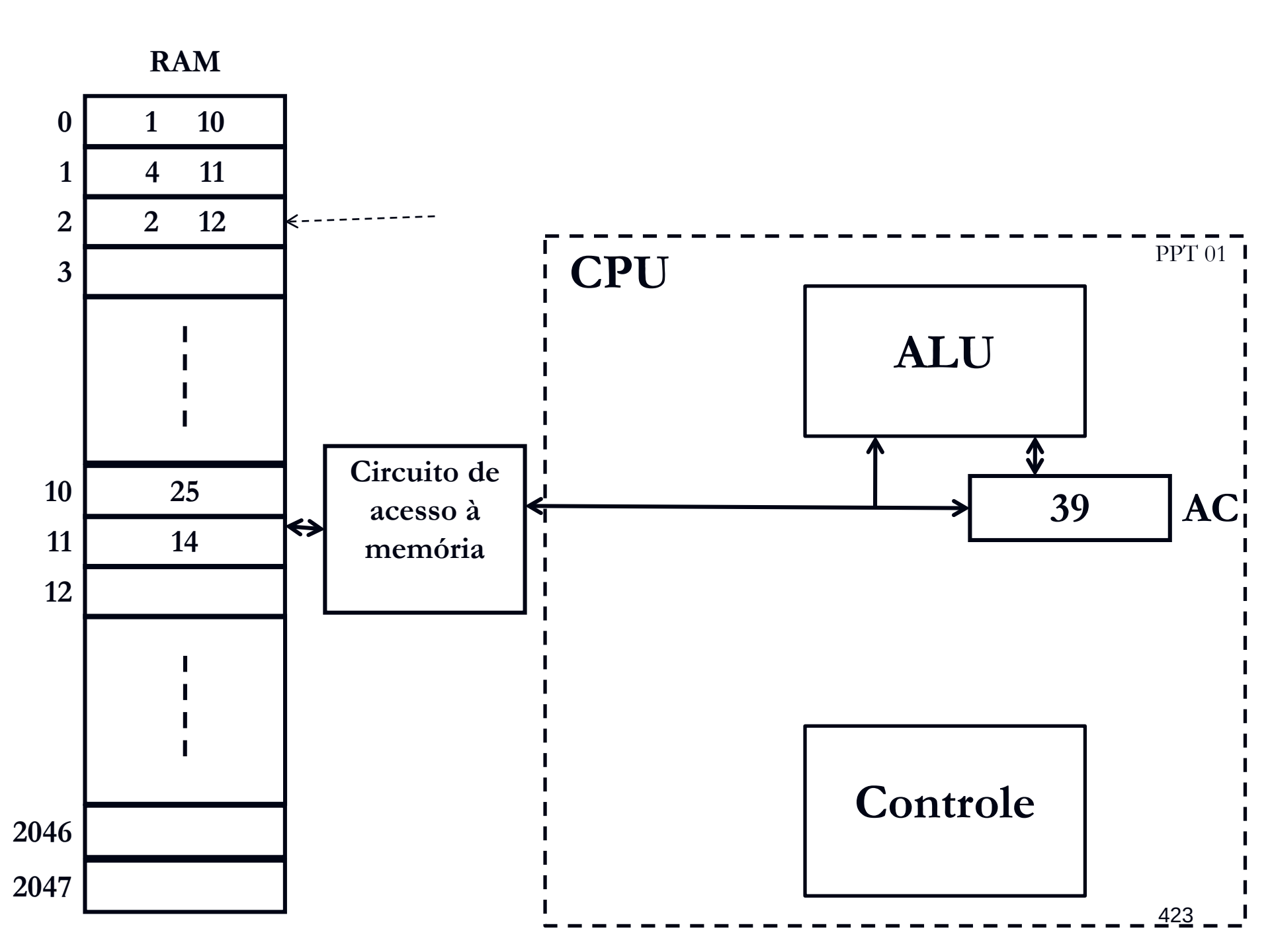


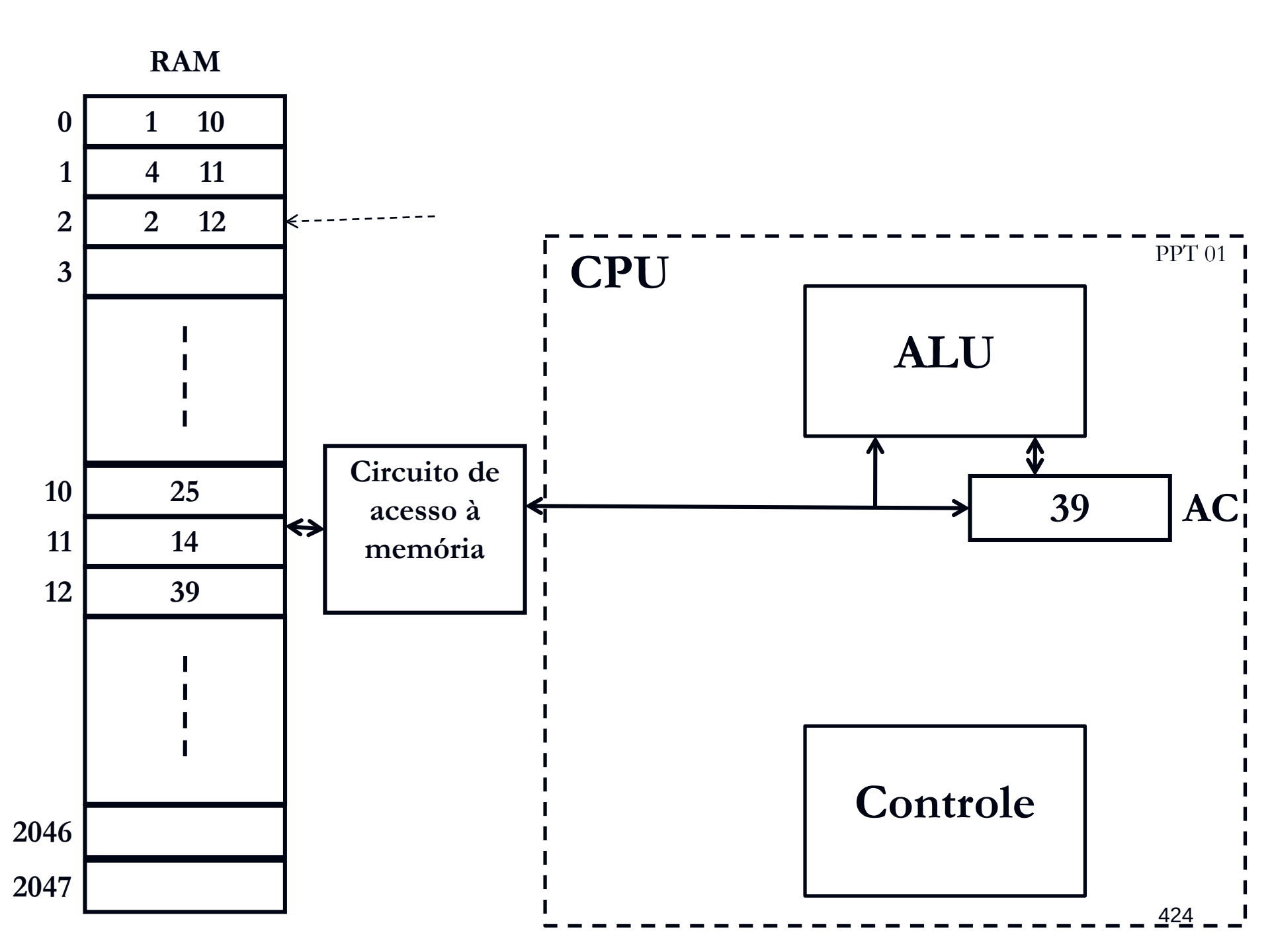

RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
AC
Controle
419


RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
25
AC
Controle
420


RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
25
AC
Controle
421


RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
39
AC
Controle
422

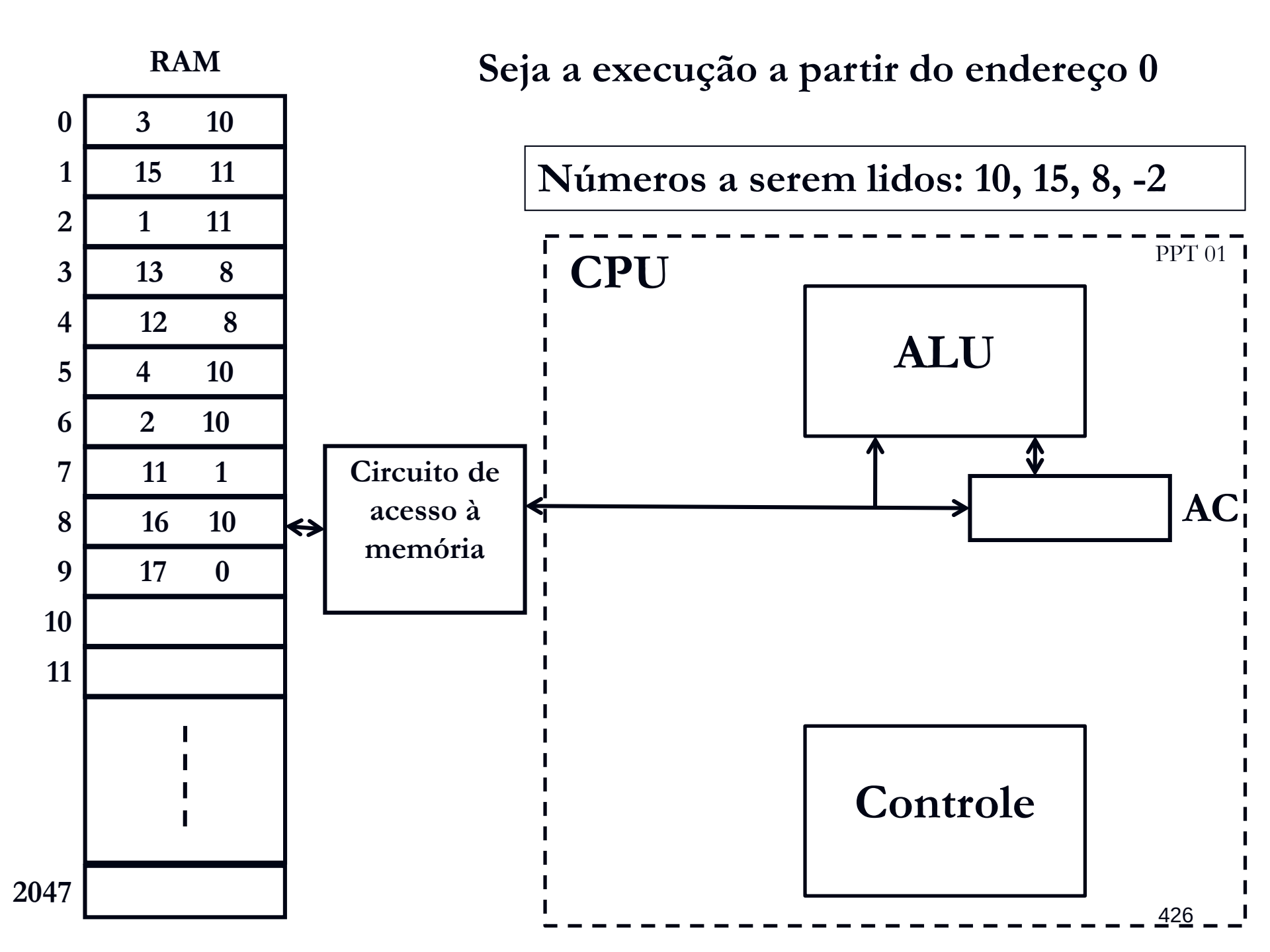

RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
2046
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
39
AC
Controle
423

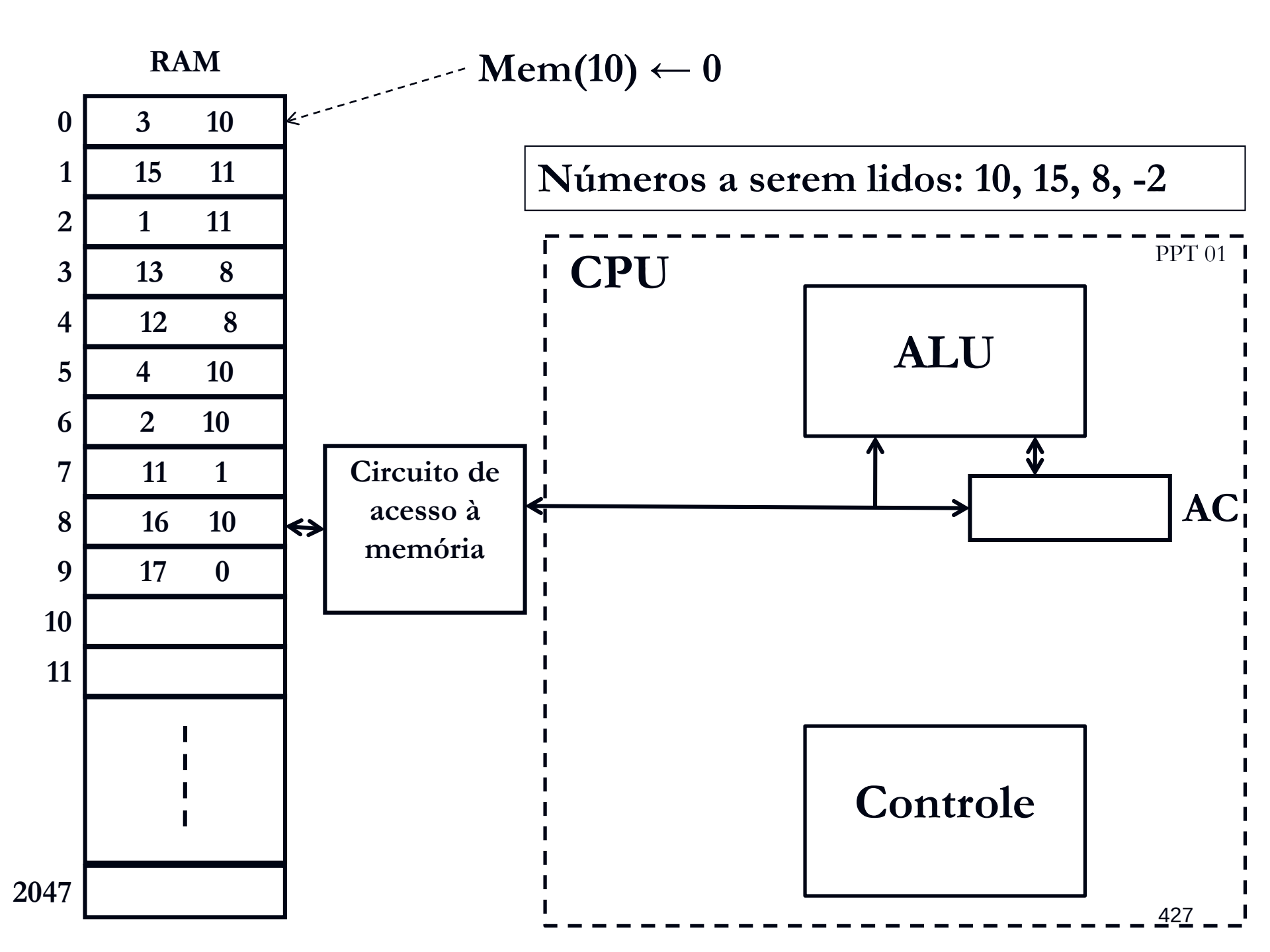

RAM
0
1 10
1
4 11
2
2 12
3
10
25
11
14
12
39
2046
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
39
AC
Controle
424

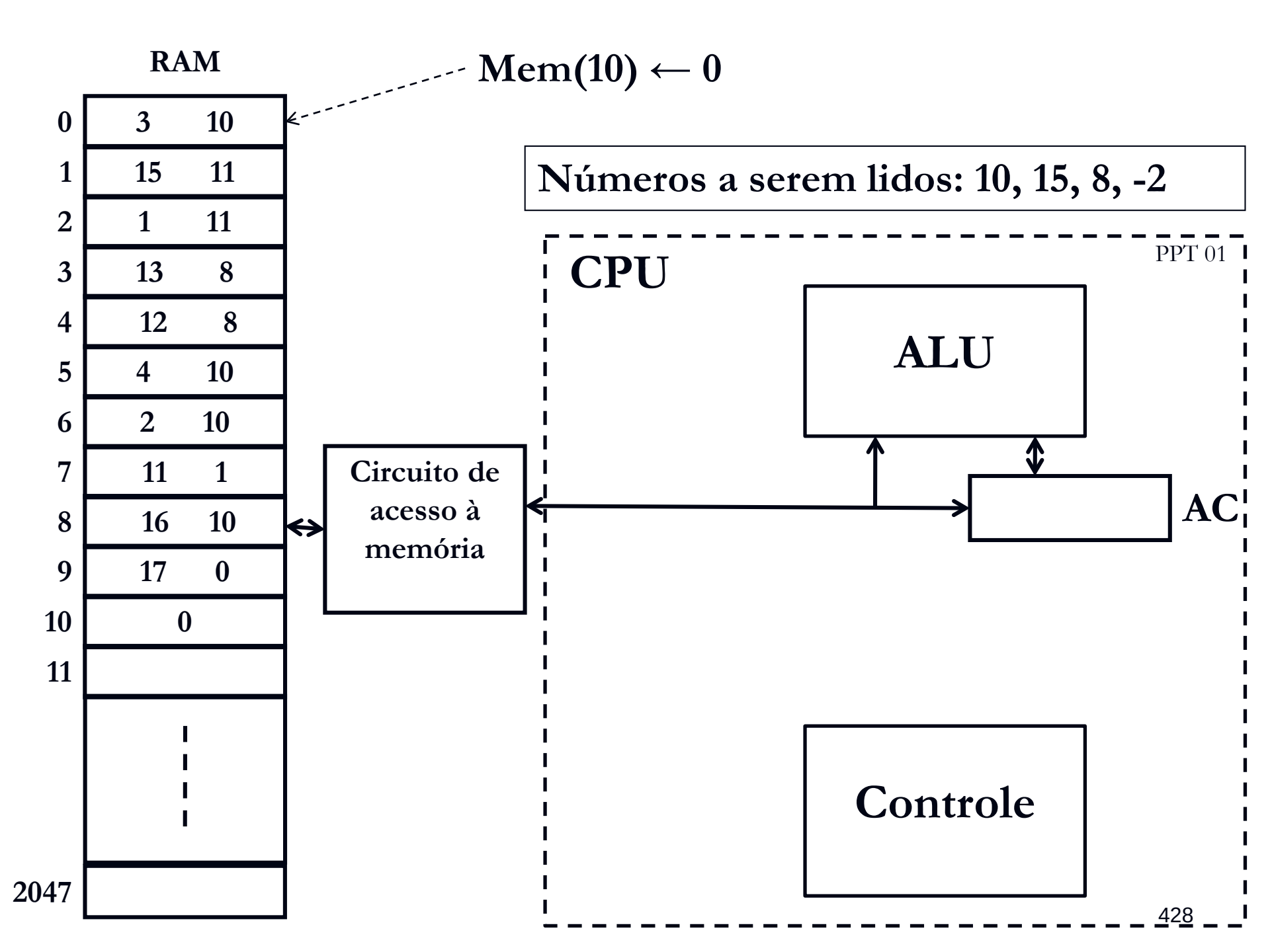
Programa para somar números inteiros positivos lidos:

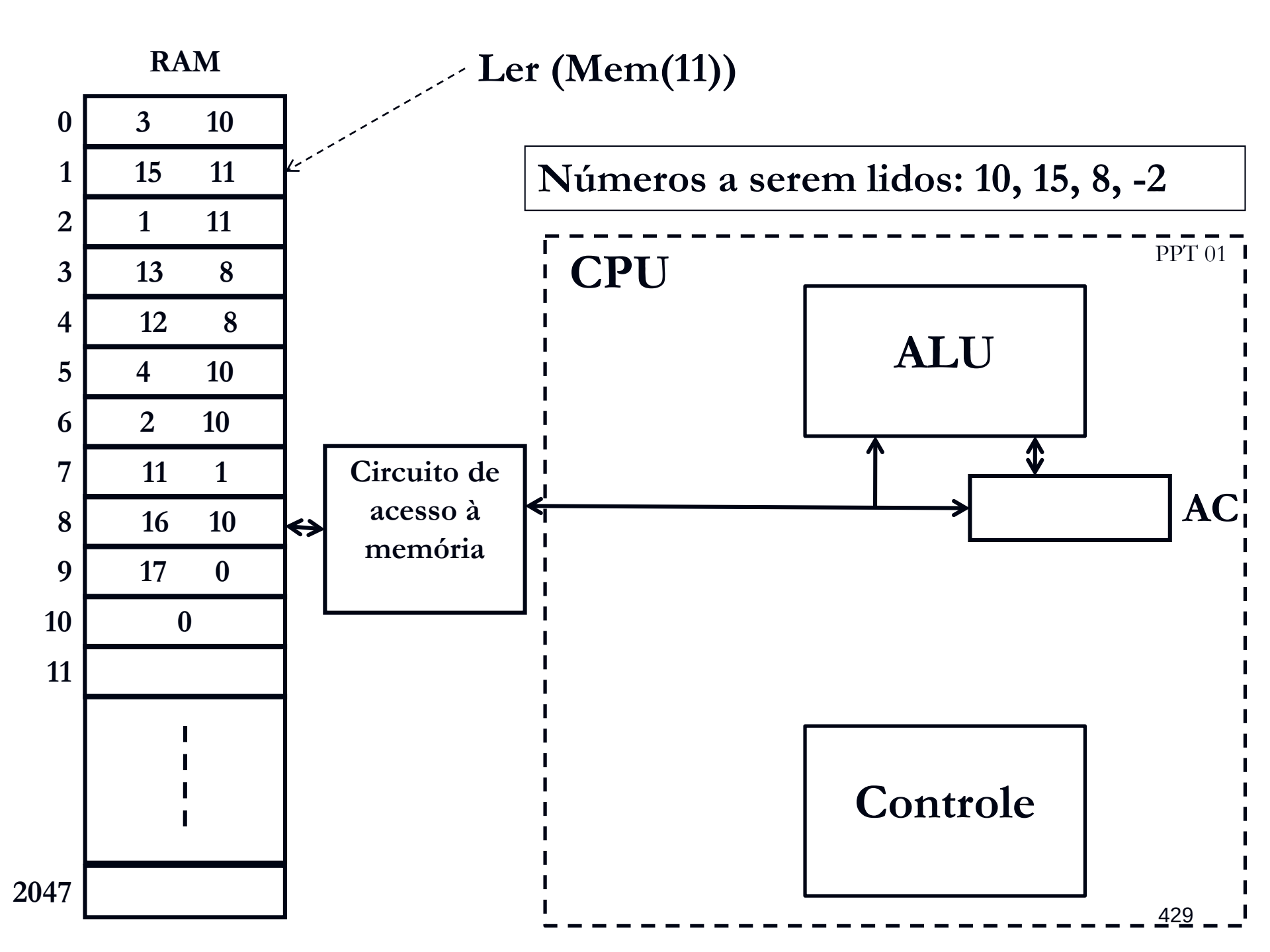
Livro 01

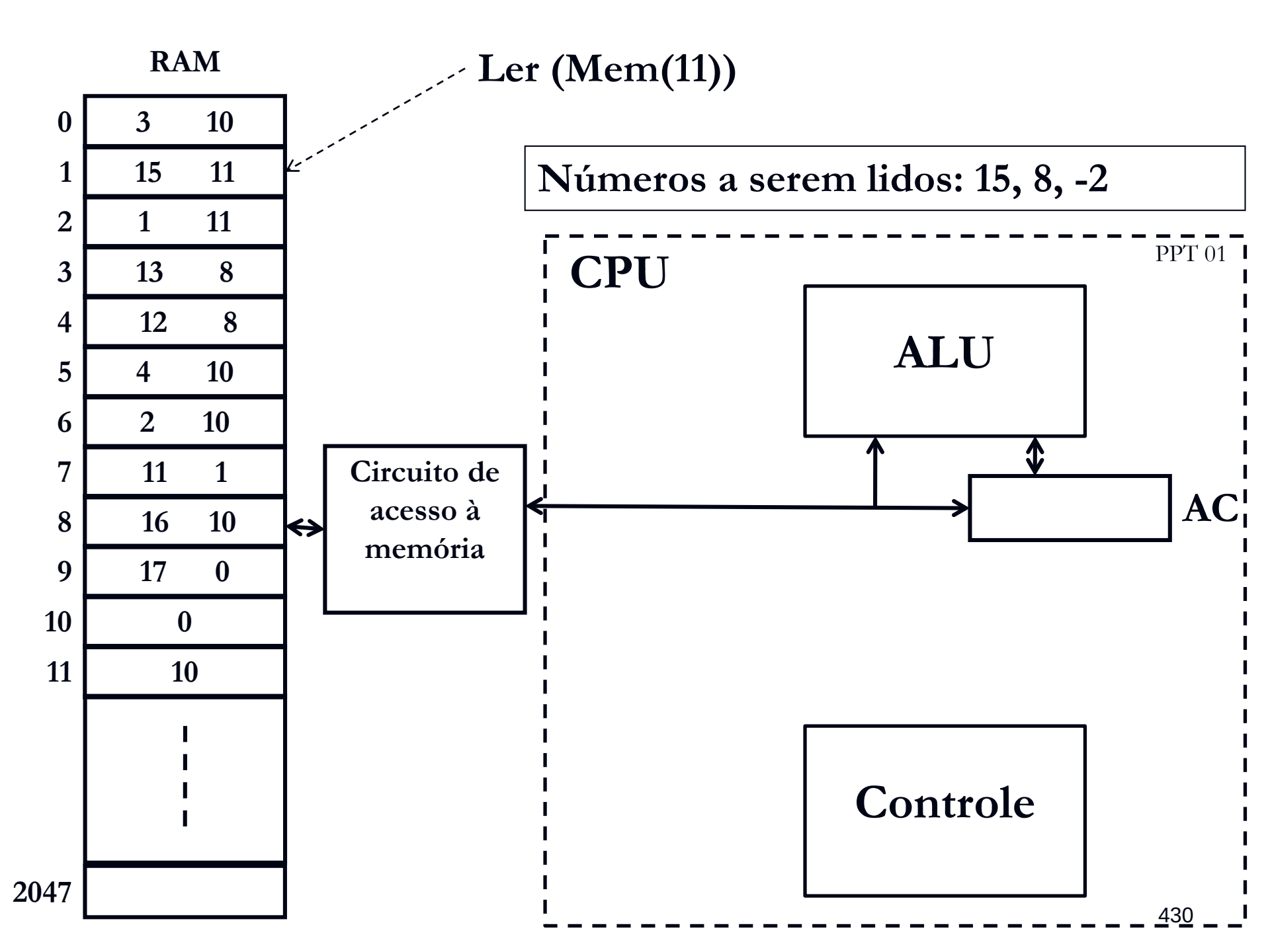
Endereço das instruções na memória	Instruções em linguagem de máquina binária		Instruções em linguagem de máquina decimal		Significado
	Código	Endereço	Código	Endereço	
0	00011	0000001010	3	10	Mem (10) ← 0
1	01111	0000001011	15	11	Ler (Mem (11))
2	00001	0000001011	1	11	AC ← Mem (11)
3	01101	0000001000	13	8	Se AC = 0 desviar p/ Mem (8)
4	01100	0000001000	12	8	Se AC < 0 desviar p/ Mem (8)
5	00100	0000001010	4	10	AC ← AC + Mem (10)
6	00010	0000001010	2	10	Mem (10) ← AC
7	01011	0000000001	11	1	Desviar para Mem (1)
8	10000	0000001010	16	10	Escrever (Mem (10))
9	10001	0000000000	17	0	Encerrar a execução

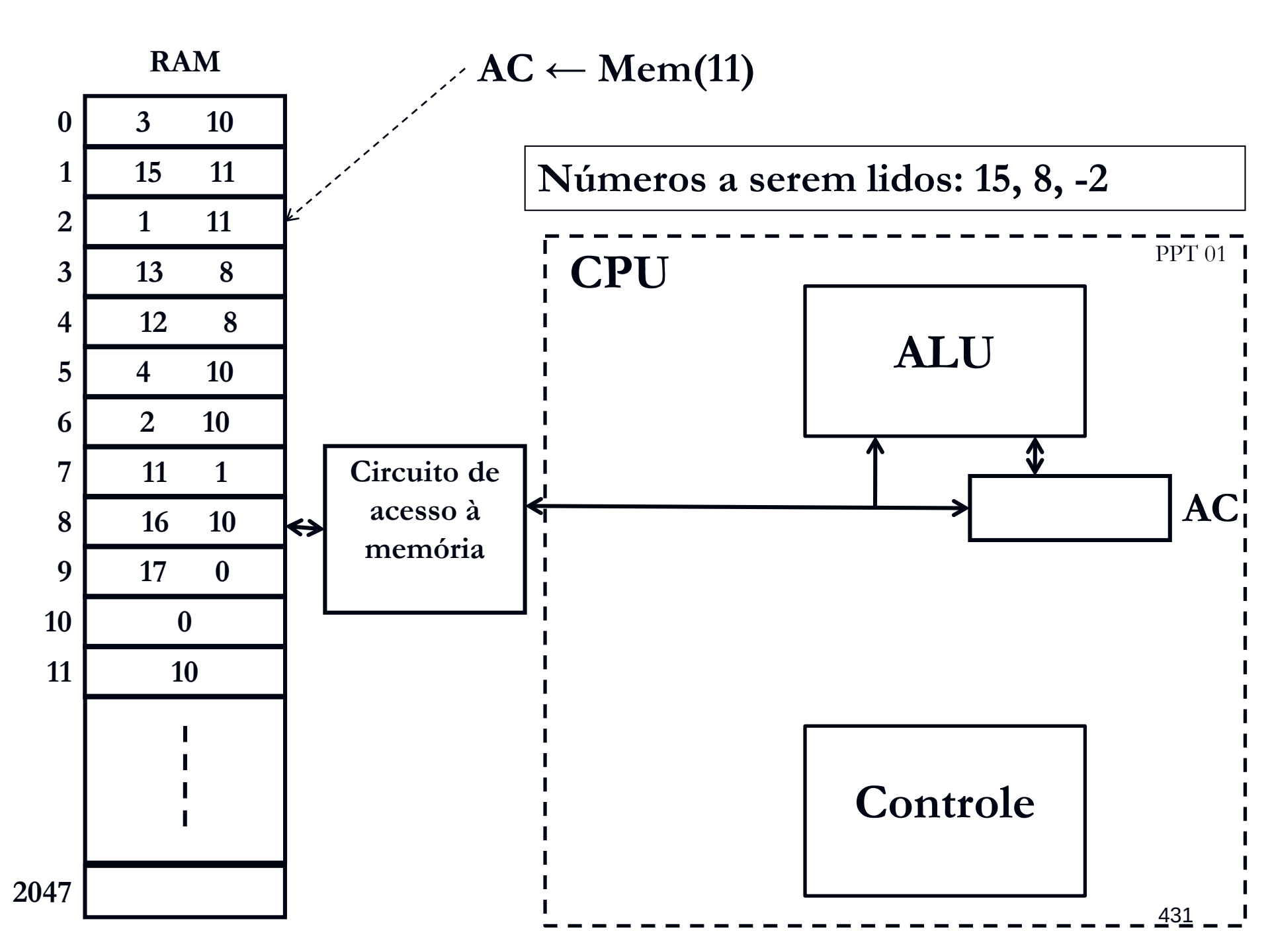

RAM
Seja a execução a partir do endereço 0
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10
11
2047
Números a serem lidos: 10, 15, 8, -2
CPU
PPT 01
ALU
AC
Circuito de acesso à memória
Controle
426

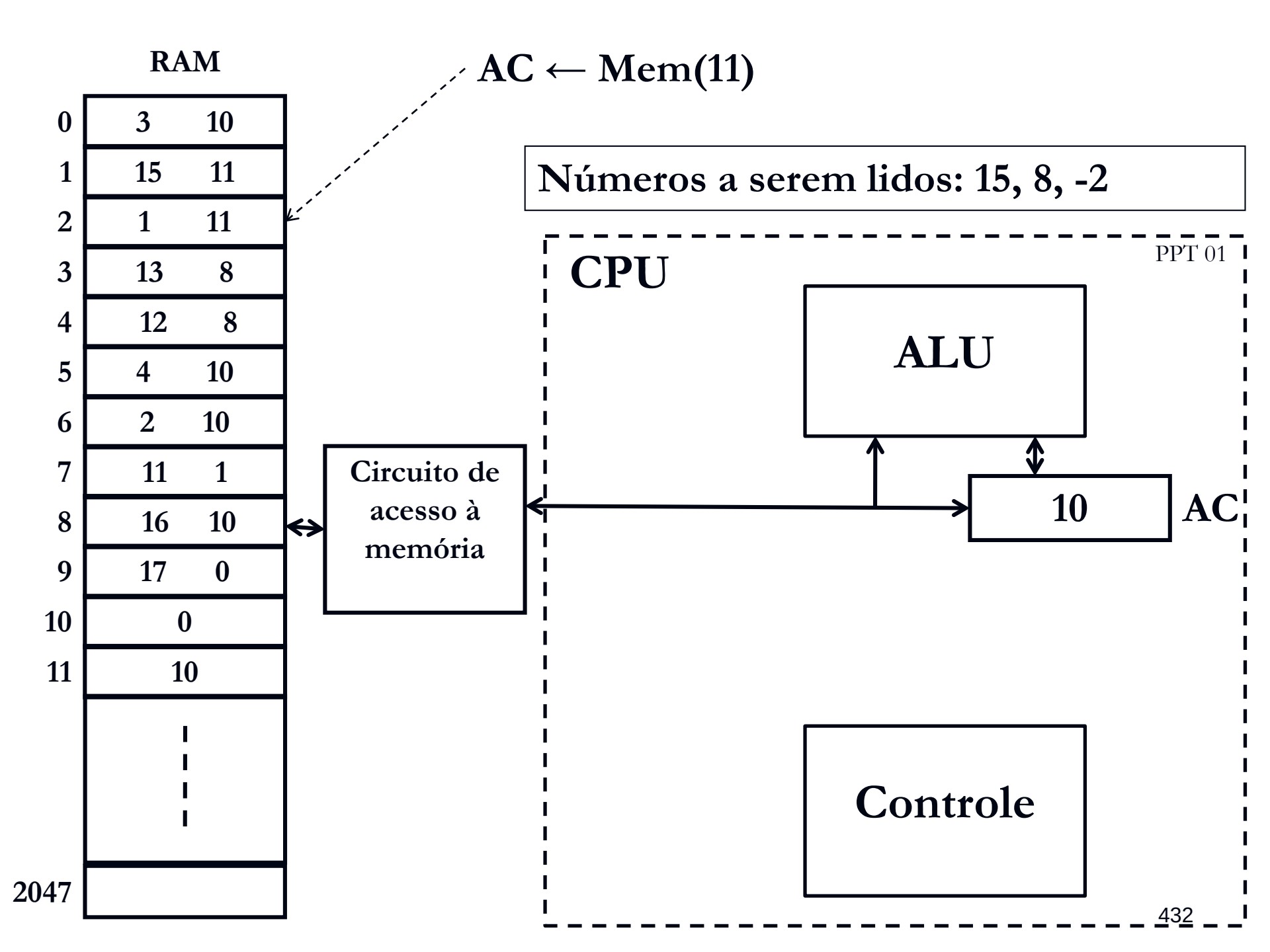

RAM
Mem(10) ← 0
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10
11
2047
Números a serem lidos: 10, 15, 8, -2
CPU
PPT 01
ALU
AC
Circuito de acesso à memória
Controle
427

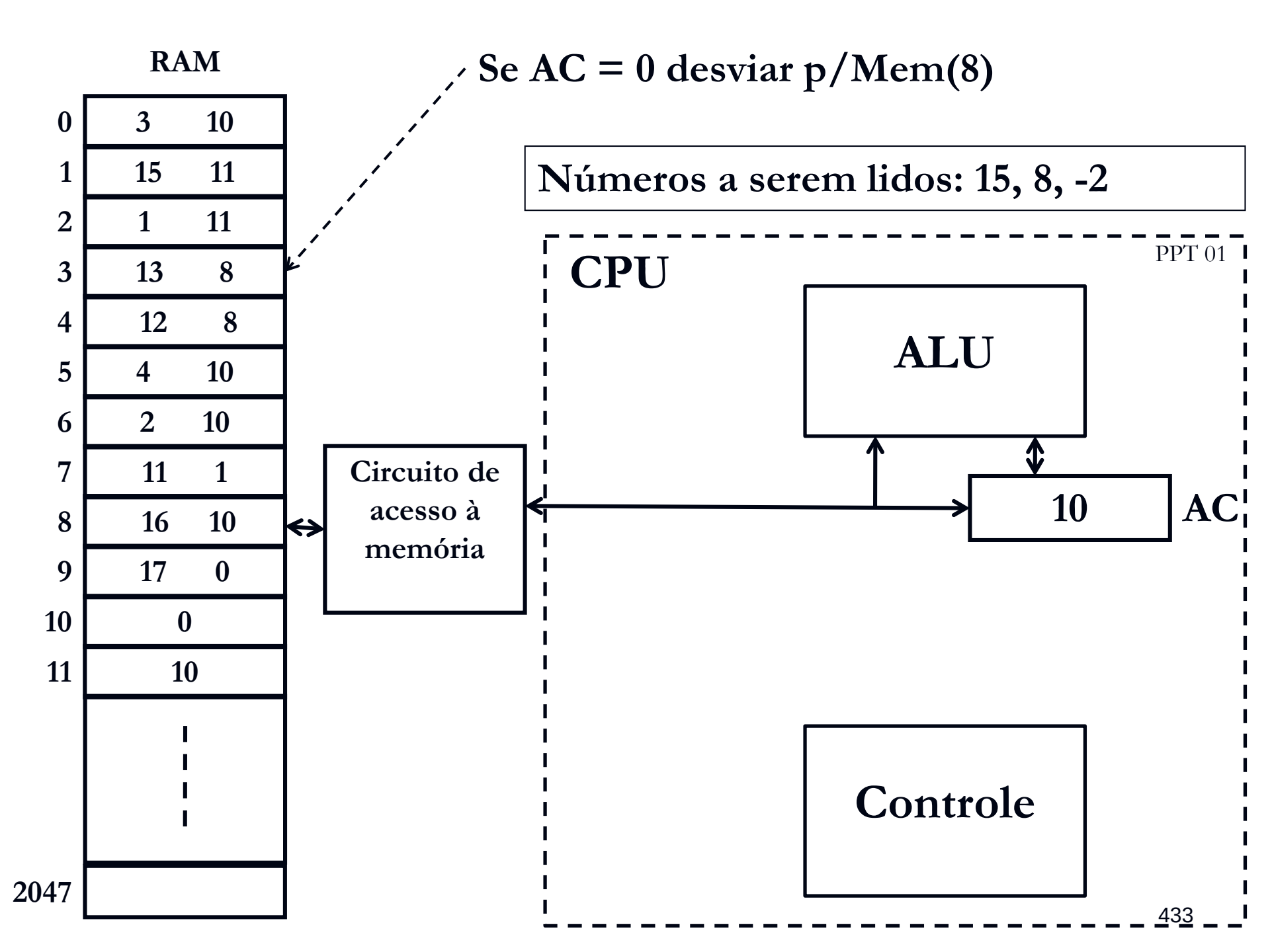

RAM
Mem(10) ← 0
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 0
11
2047
Números a serem lidos: 10, 15, 8, -2
CPU
PPT 01
ALU
AC
Circuito de acesso à memória
Controle
428

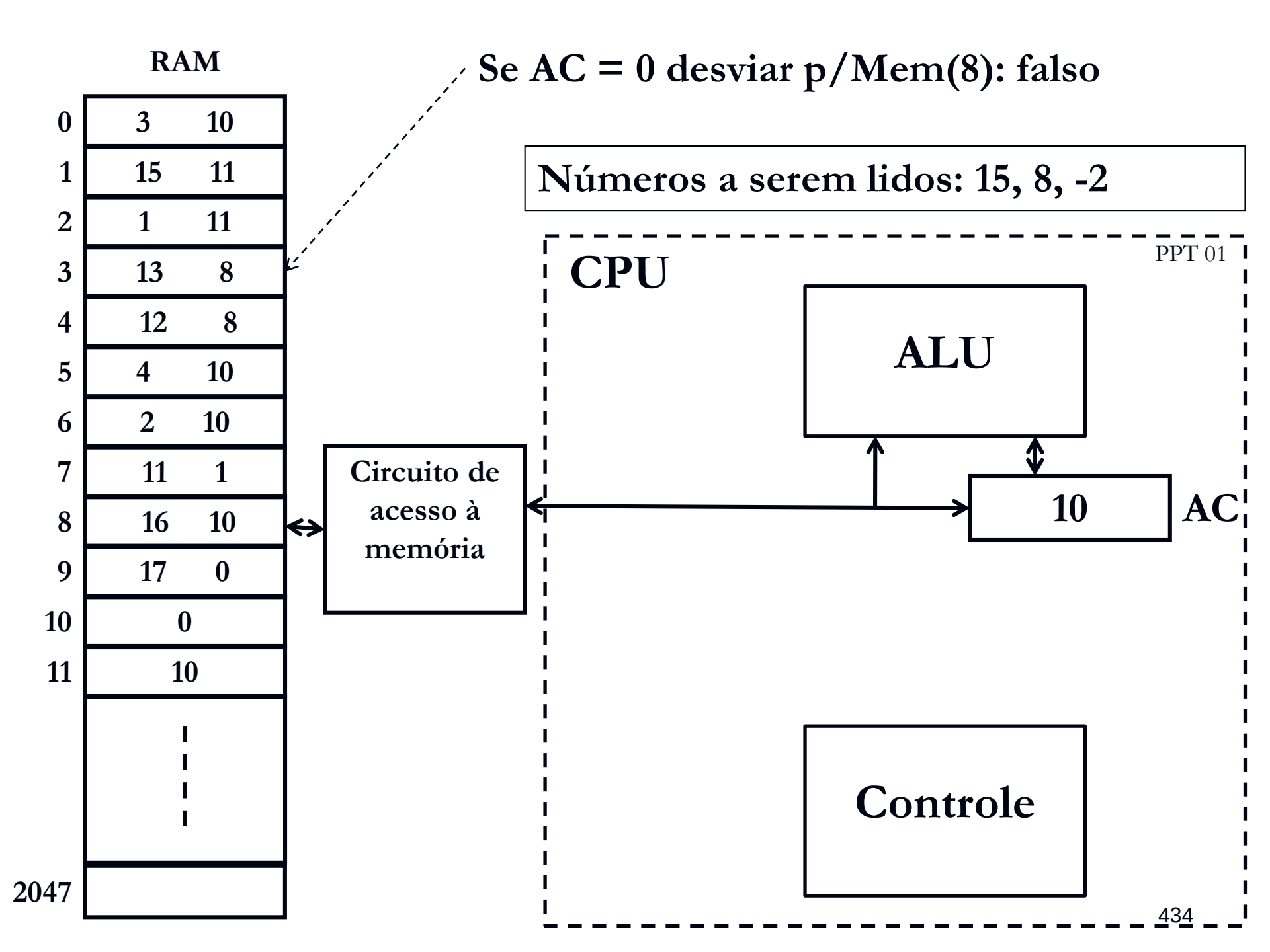

RAM
Ler (Mem(11))
Números a serem lidos: 10, 15, 8, -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 0
11
2047
CPU
PPT 01
ALU
AC
Circuito de acesso à memória
Controle
429

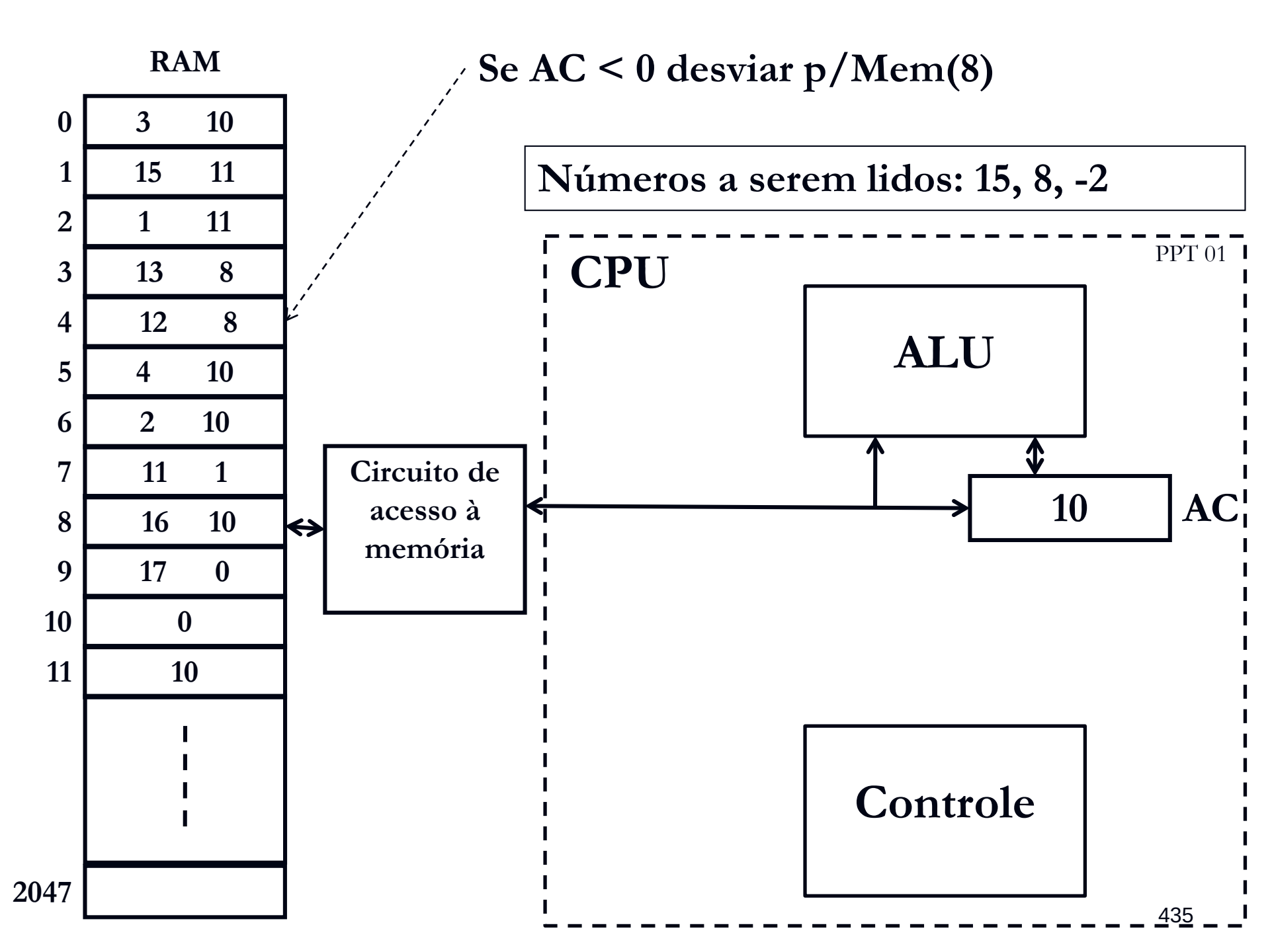

RAM
Ler (Mem(11))
Números a serem lidos: 15, 8, -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 0
11 10
2047
CPU
PPT 01
ALU
AC
Circuito de acesso à memória
Controle
430

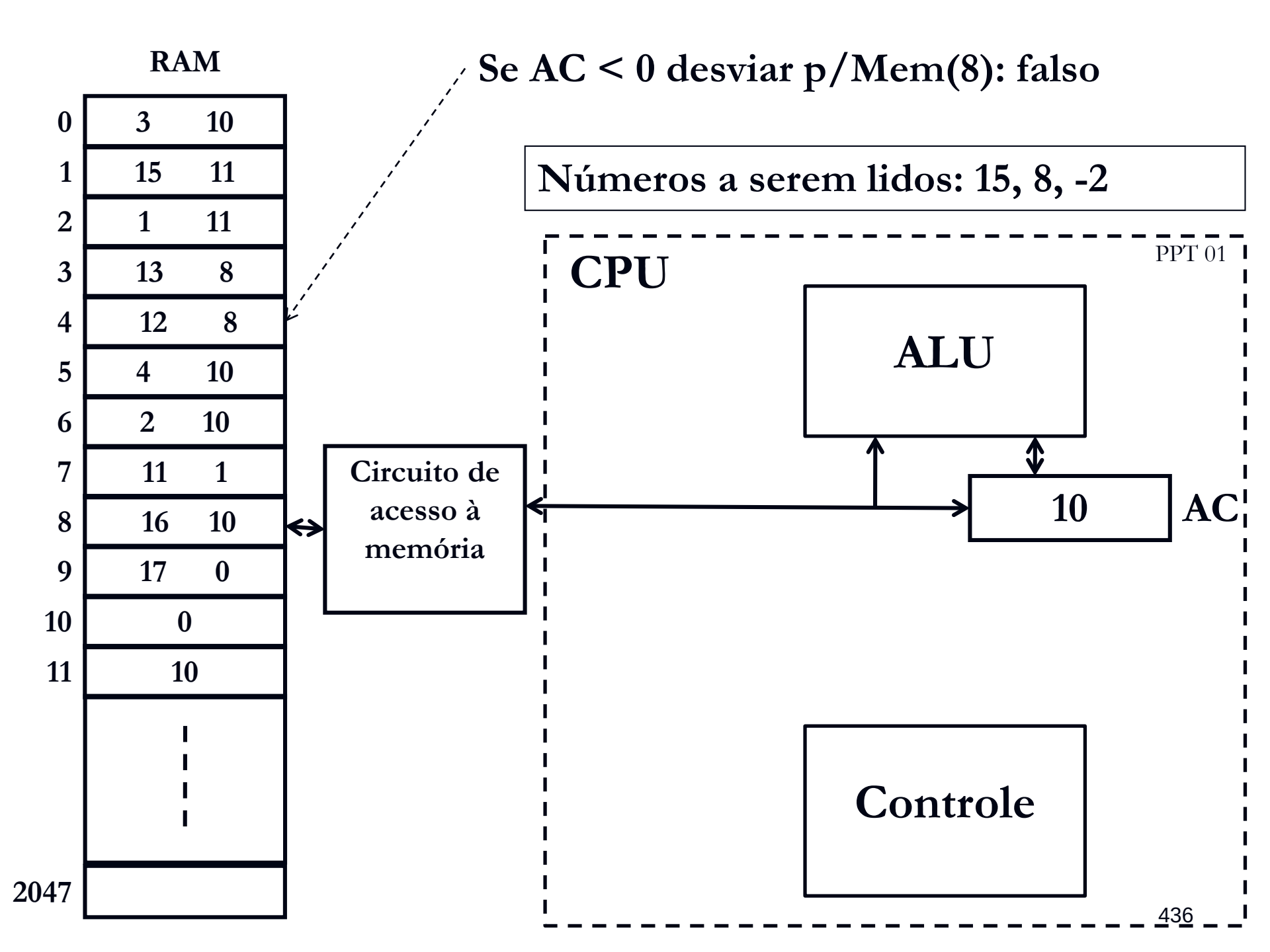

RAM
AC ← Mem(11)
Números a serem lidos: 15, 8, -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 0
11 10
2047
CPU
PPT 01
ALU
Circuito de acesso à memória
AC
Controle
431

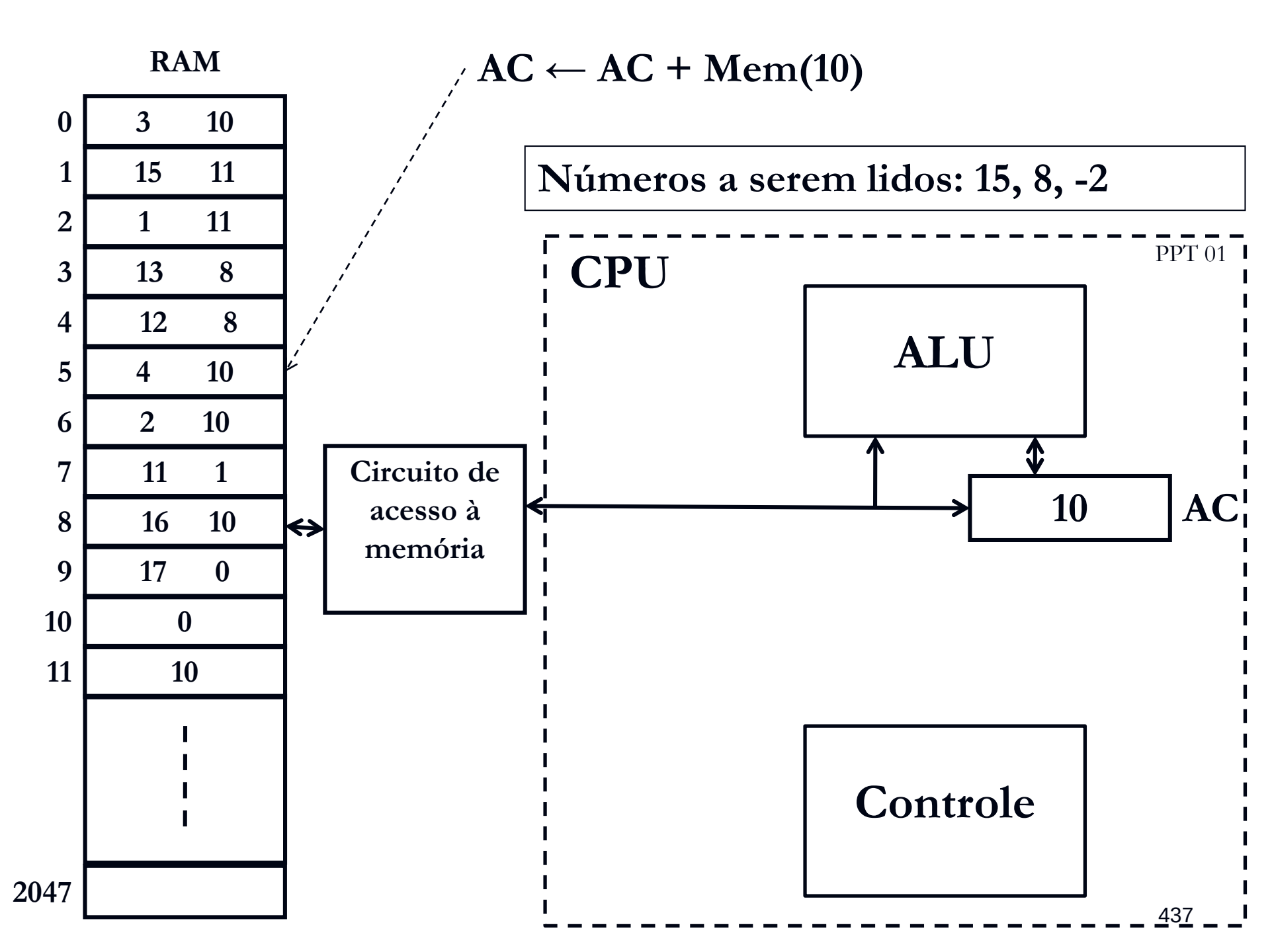

RAM
AC ← Mem(11)
Números a serem lidos: 15, 8, -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 0
11 10
2047
CPU
PPT 01
ALU
Circuito de acesso à memória
10
AC
Controle
432

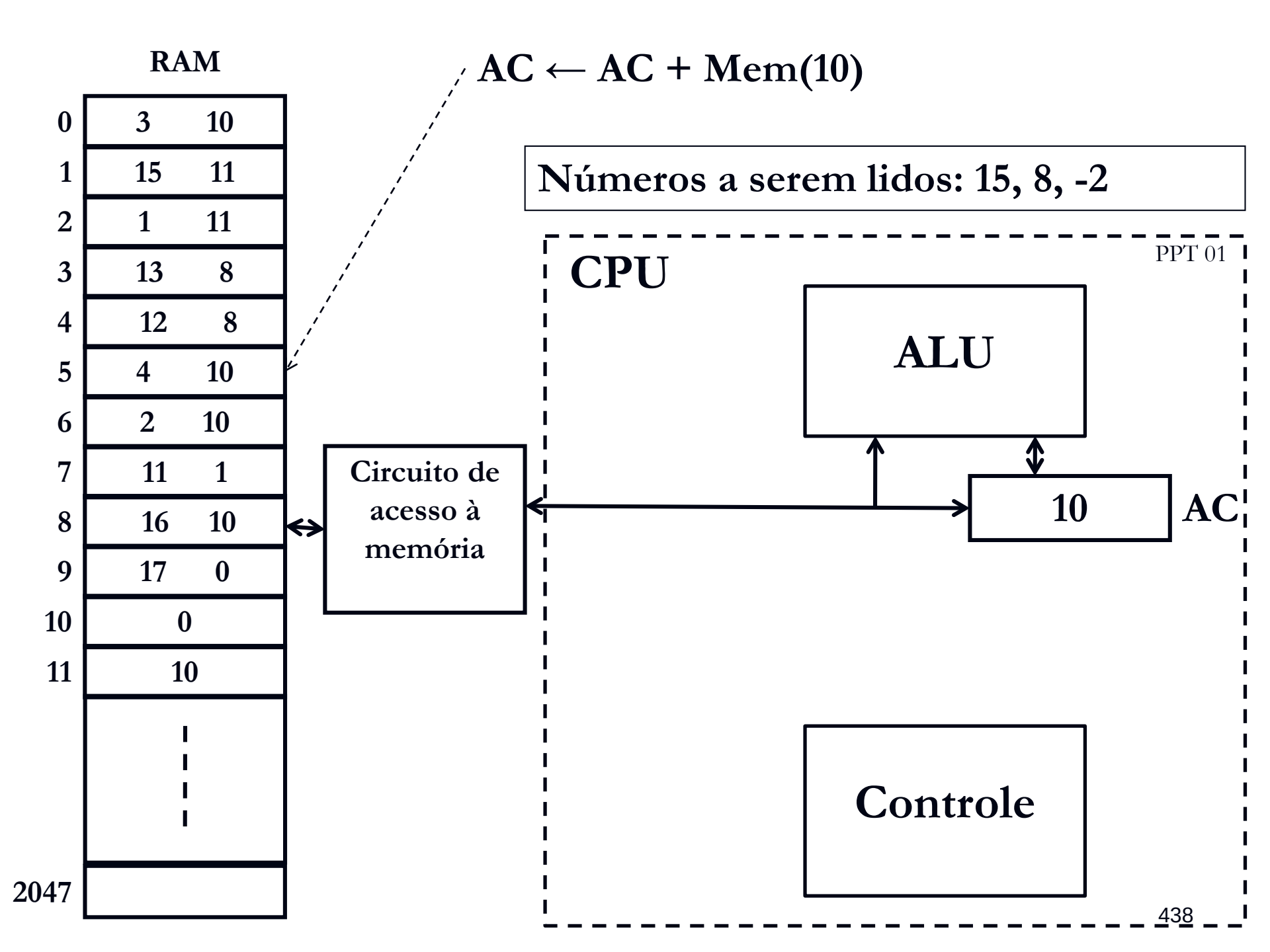

RAM
Se AC = 0 desviar p/Mem(8)
Números a serem lidos: 15, 8, -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 0
11 10
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
10
AC
Controle
433

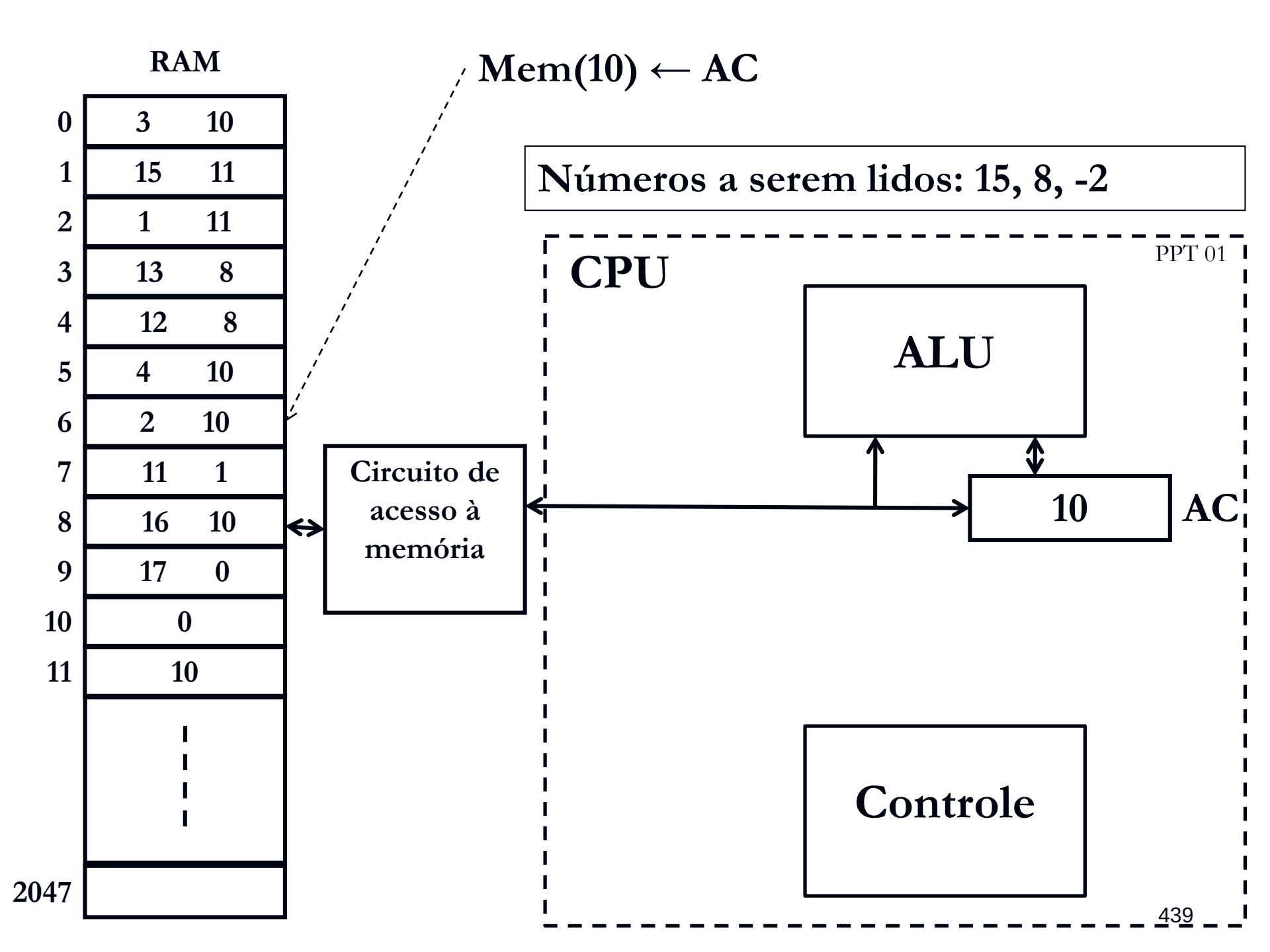

RAM
Se AC = 0 desviar p/Mem(8): falso
Números a serem lidos: 15, 8, -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 0
11 10
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
10
AC
Controle
434

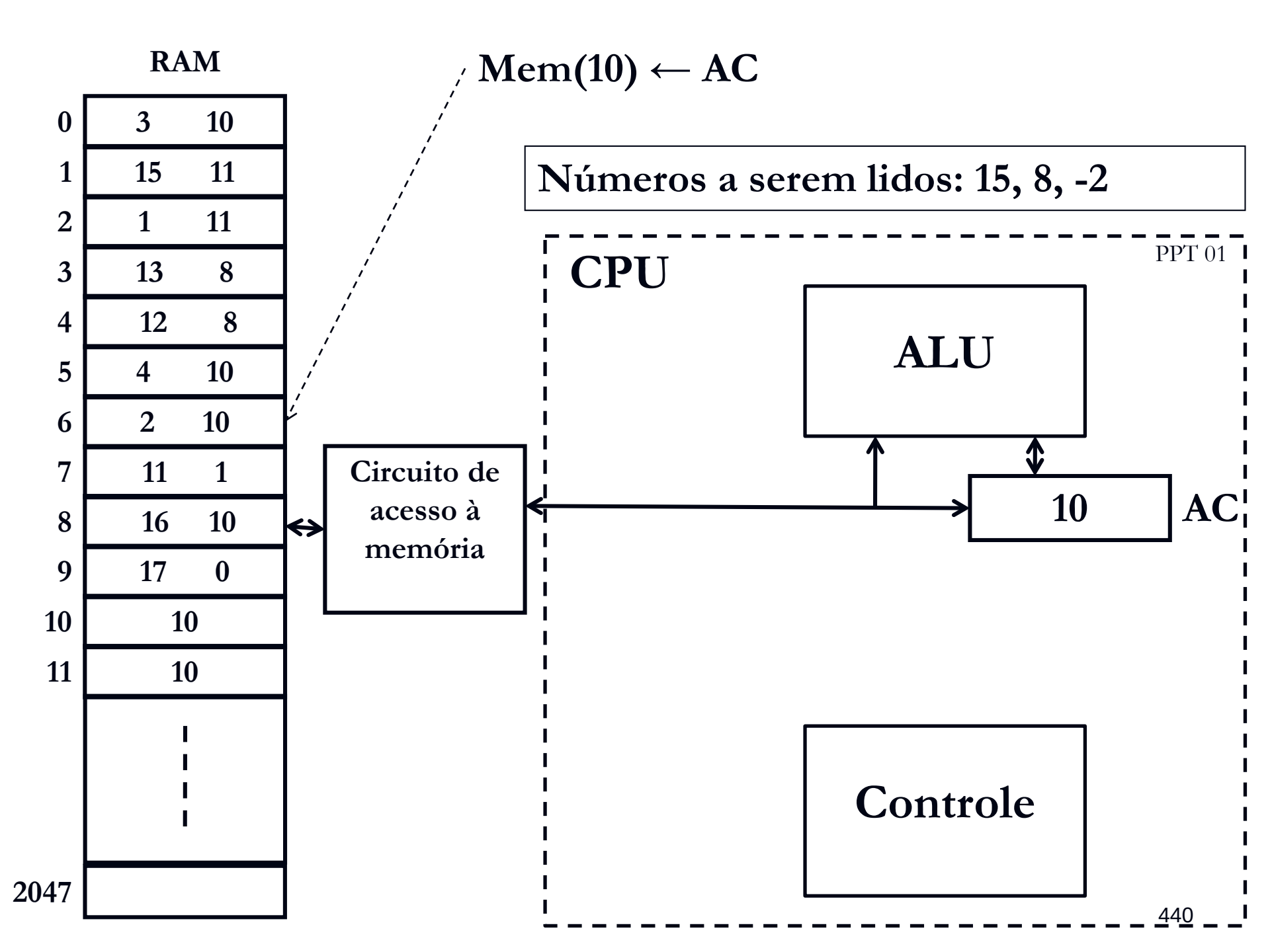

RAM
Se AC < 0 desviar p/Mem(8)
Números a serem lidos: 15, 8, -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 0
11 10
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
10
AC
Controle
435

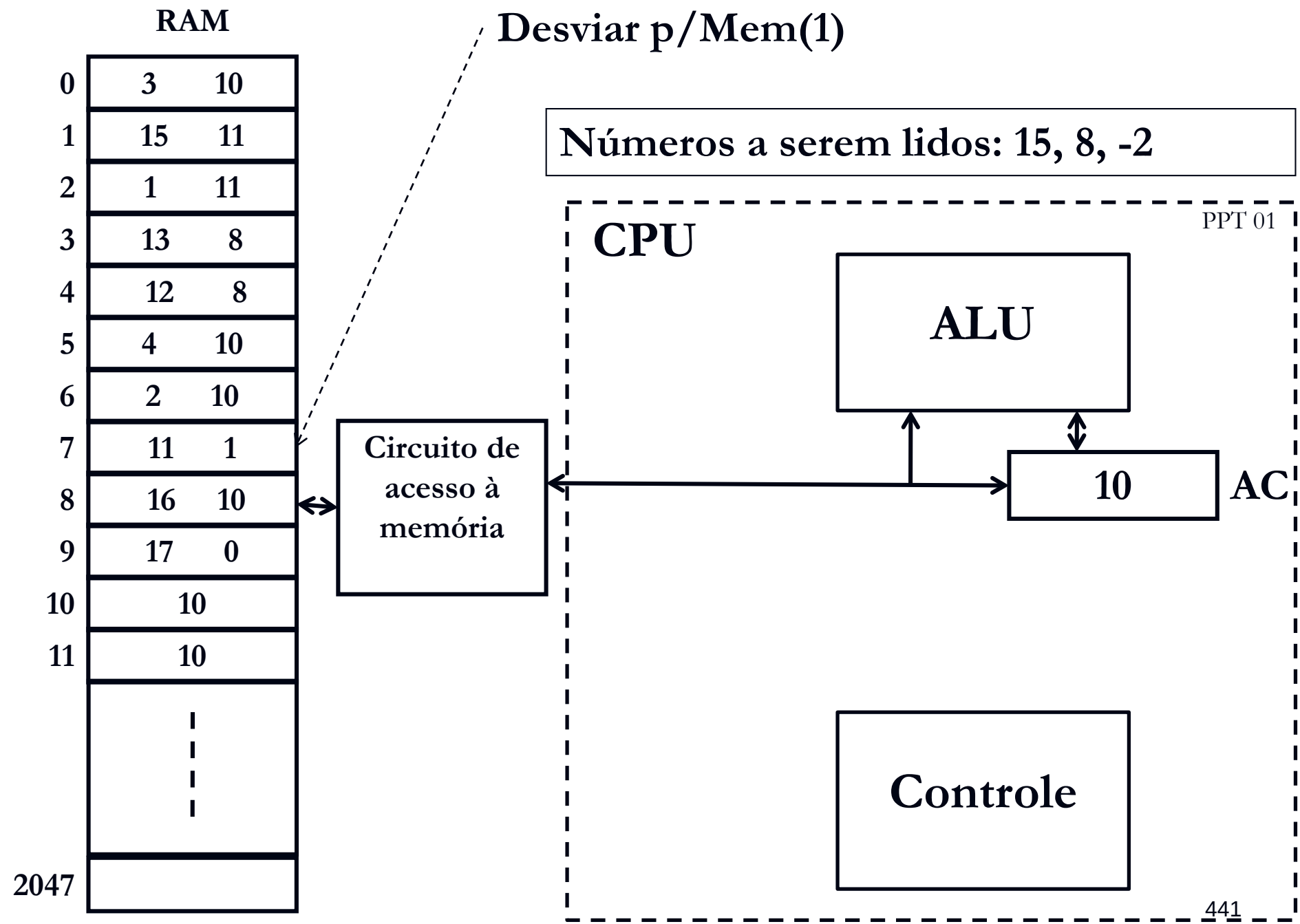

RAM
Se AC < 0 desviar p/Mem(8): falso
Números a serem lidos: 15, 8, -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 0
11 10
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
10
AC
Controle
436

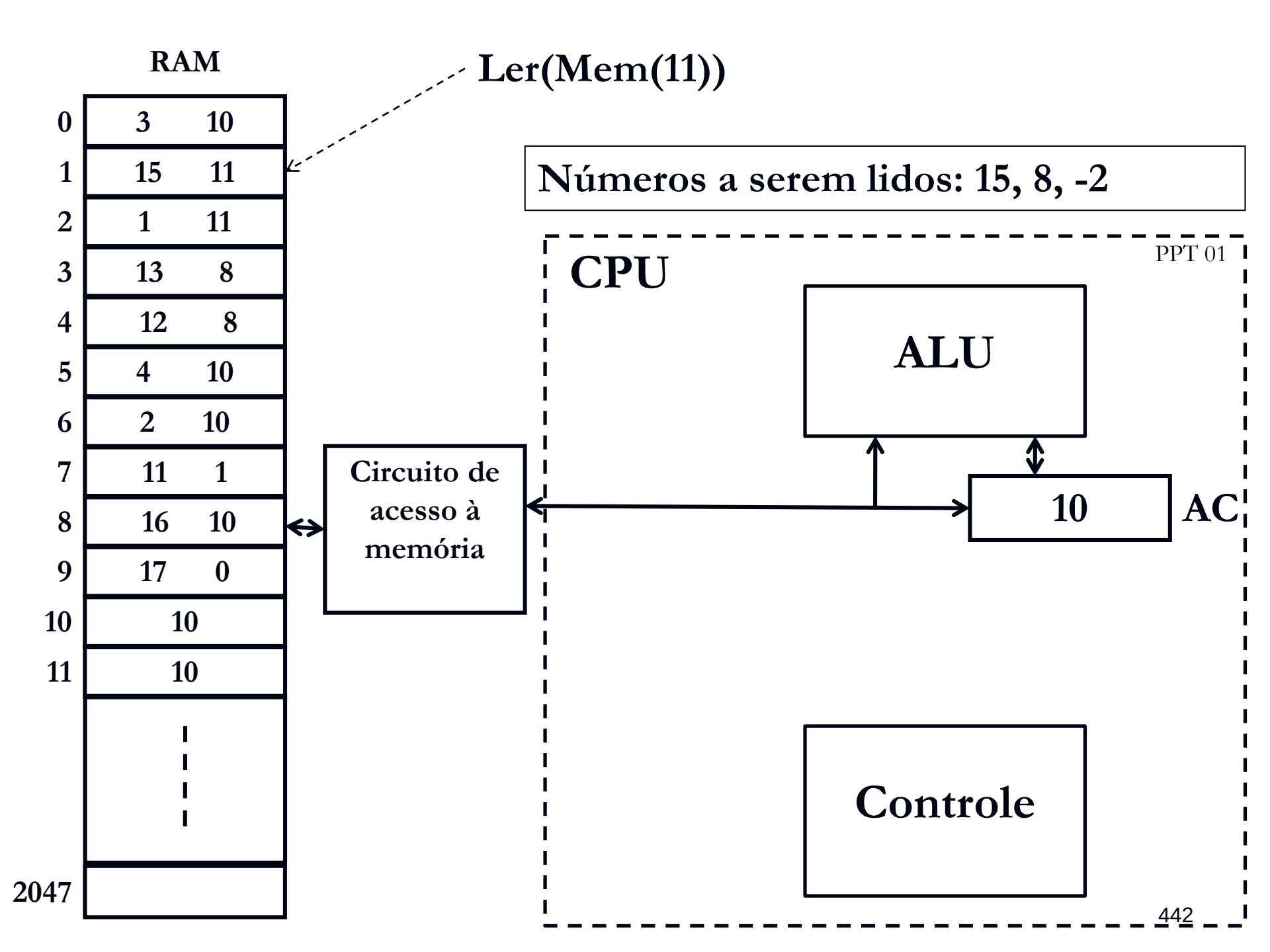

RAM
AC ← AC + Mem(10)
Números a serem lidos: 15, 8, -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 0
11 10
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
10
AC
Controle
437

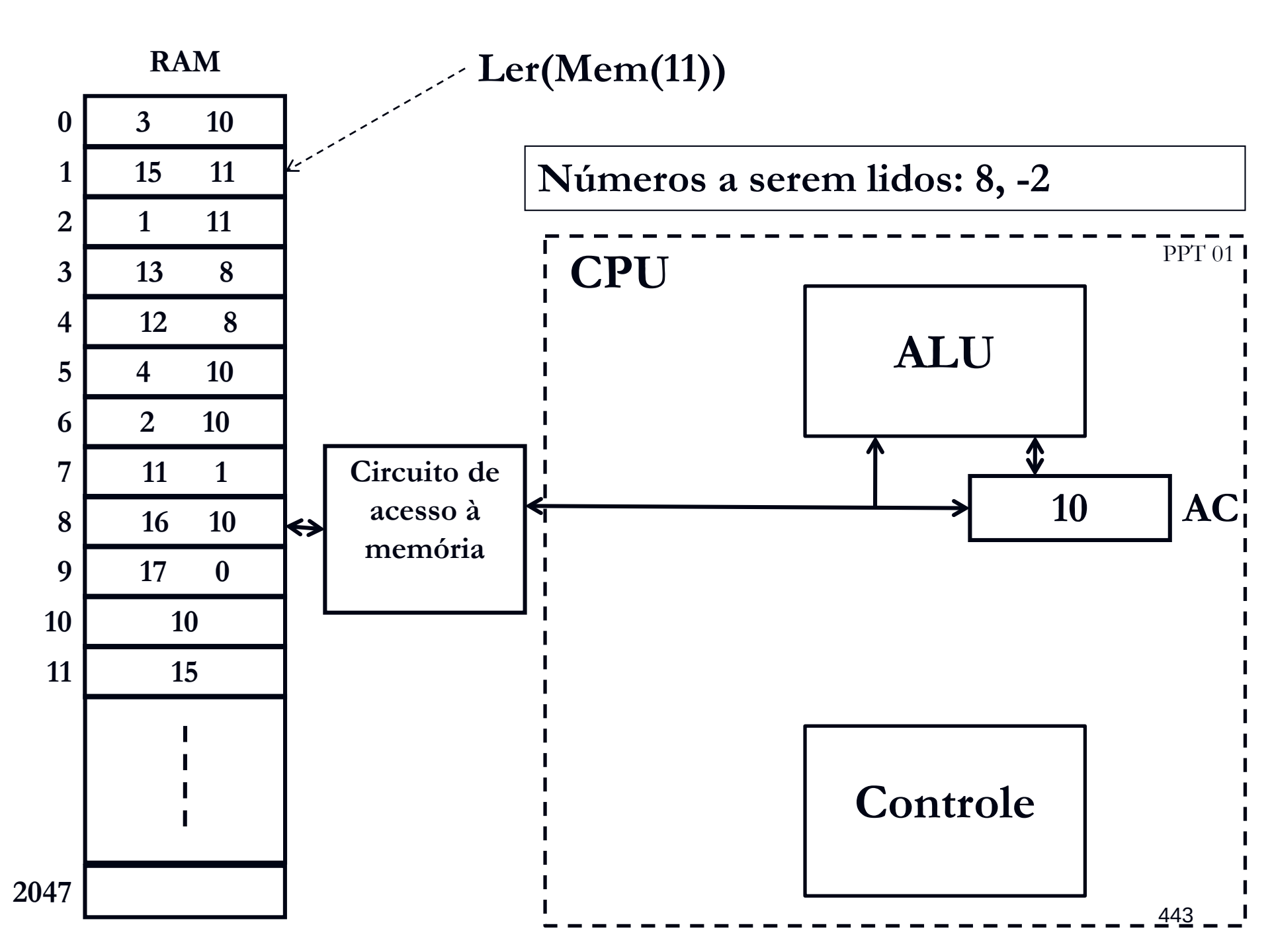

RAM
AC ← AC + Mem(10)
Números a serem lidos: 15, 8, -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 0
11 10
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
10
AC
Controle
438

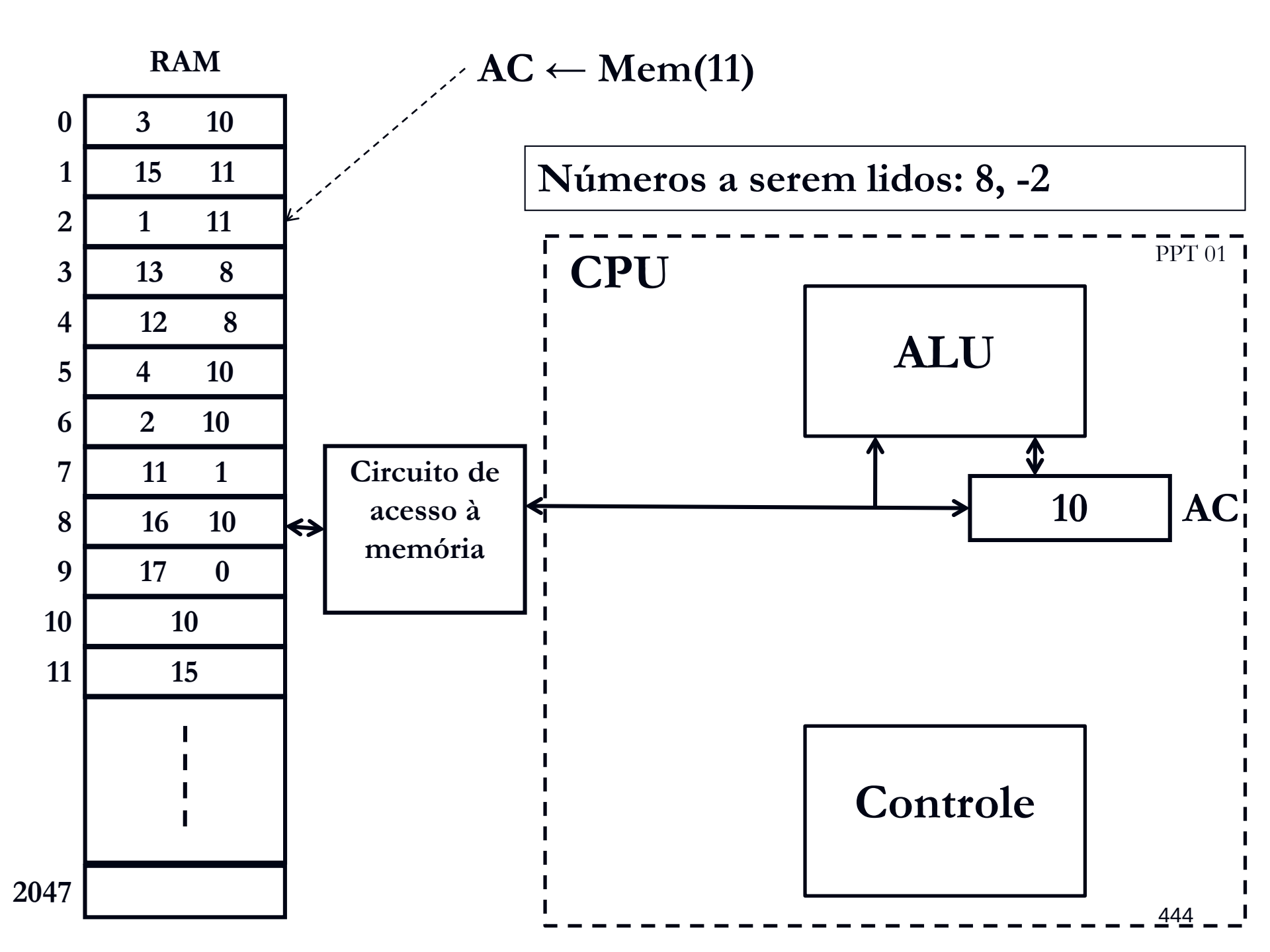

RAM
Mem(10) ← AC
Números a serem lidos: 15, 8, -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 0
11 10
2047
CPU
PPT 01
ALU
Circuito de acesso à memória
10
AC
Controle
439

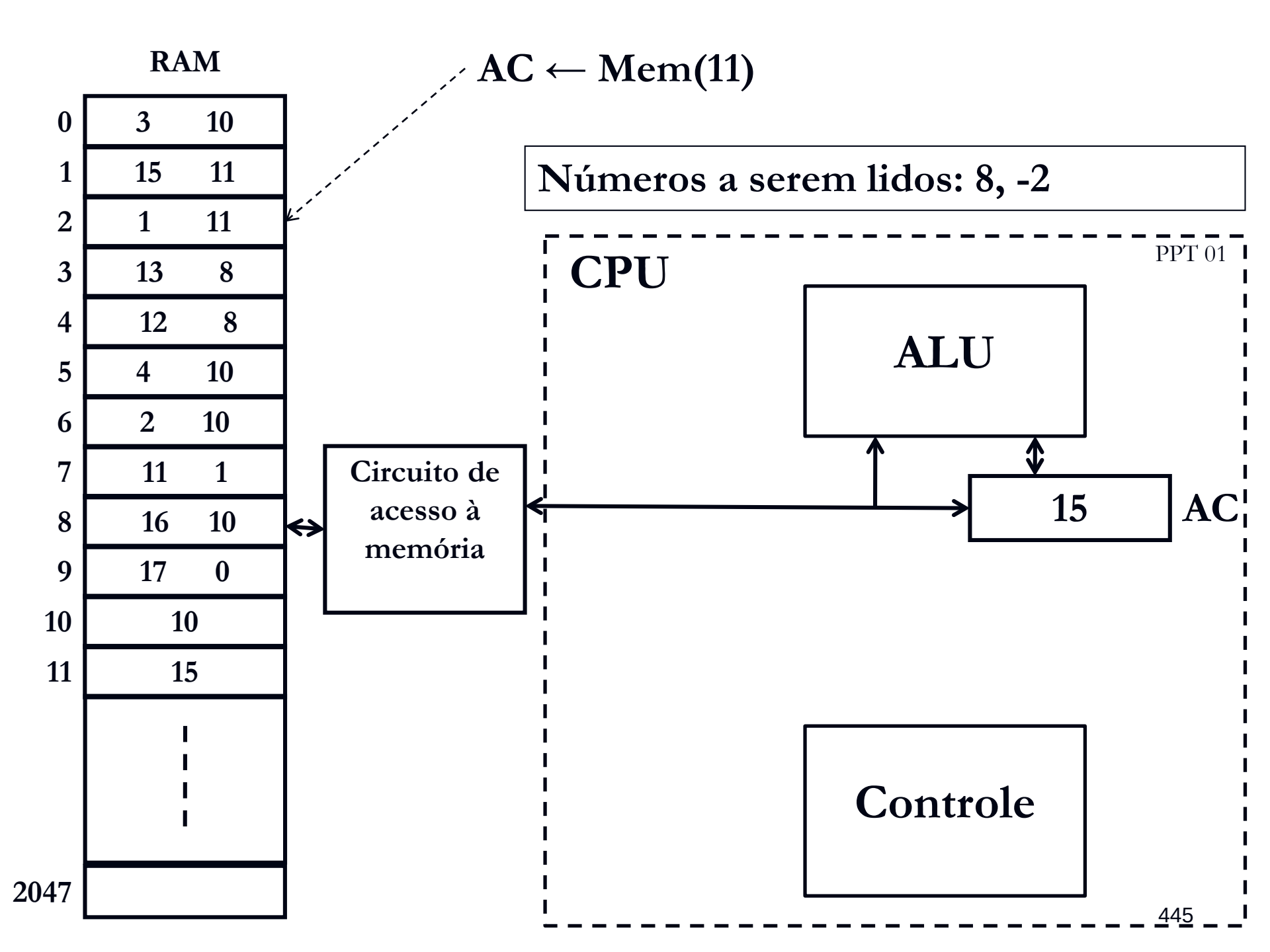

RAM
Mem(10) ← AC
Números a serem lidos: 15, 8, -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 10
11 10
2047
CPU
PPT 01
ALU
Circuito de acesso à memória
10
AC
Controle
440

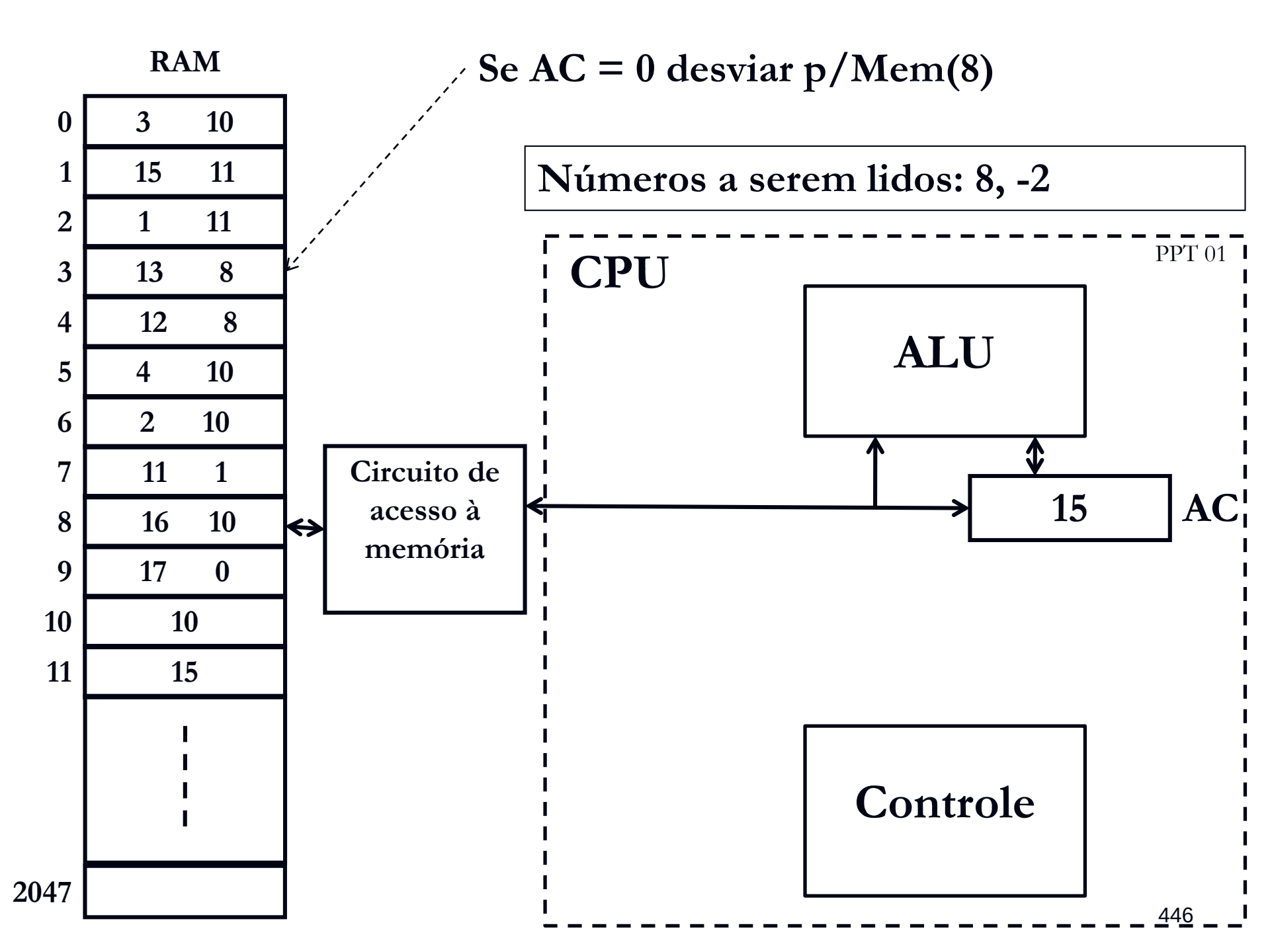

RAM
Desviar p/Mem(1)
Números a serem lidos: 15, 8, -2
CPU
PPT 01
ALU
Circuito de acesso à memória
10
AC
Controle
441
0
1
2
3
4
5
6
7
8
9
10
11
2047
3 10
15 11
1 11
13 8
12 8
4 10
2 10
11 1
16 10
17 0
10
10

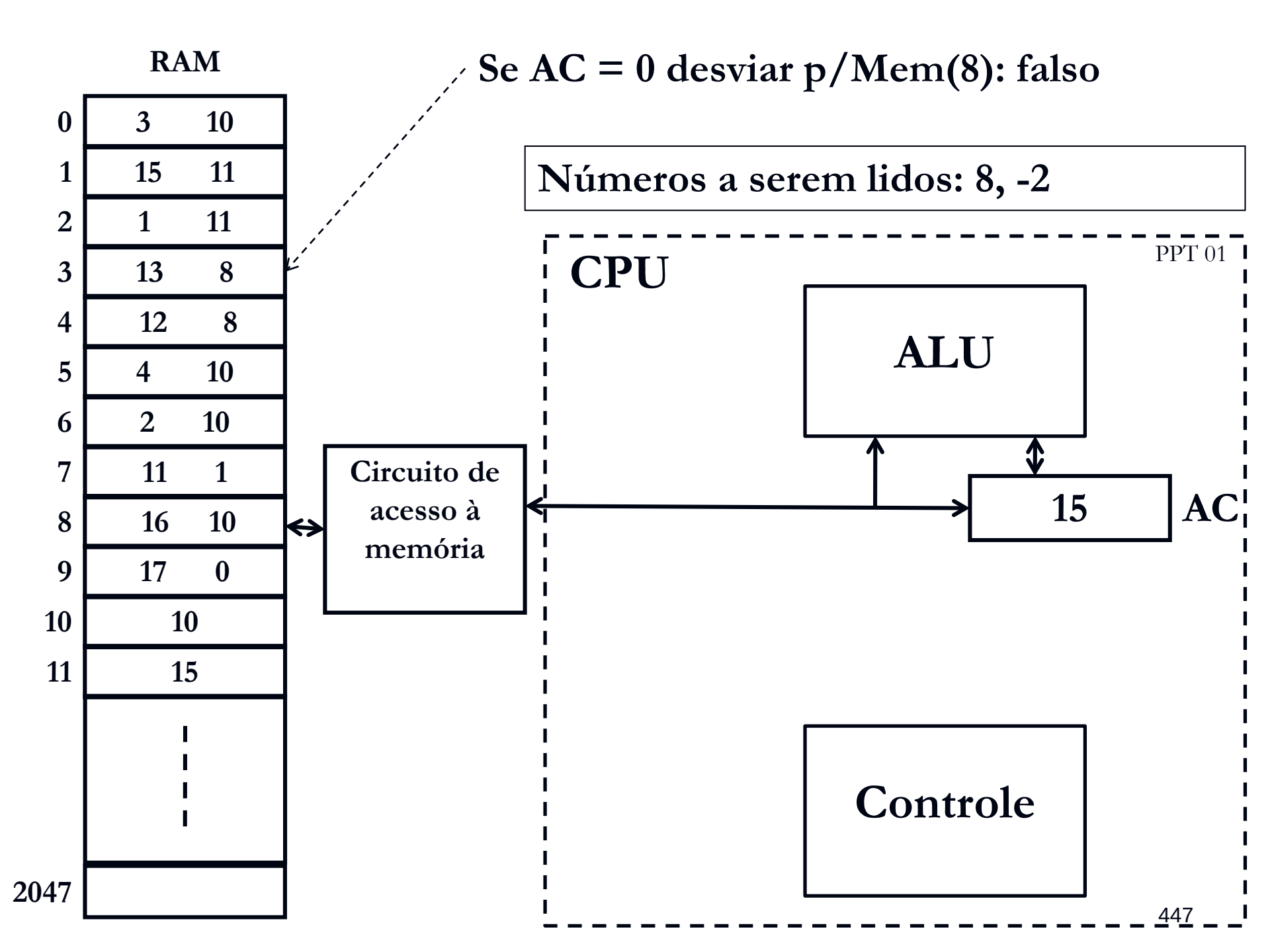

RAM
Ler(Mem(11))
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 10
11 10
2047
Números a serem lidos: 15, 8, -2
CPU
PPT 01
ALU
Circuito de acesso à memória
10
AC
Controle
442

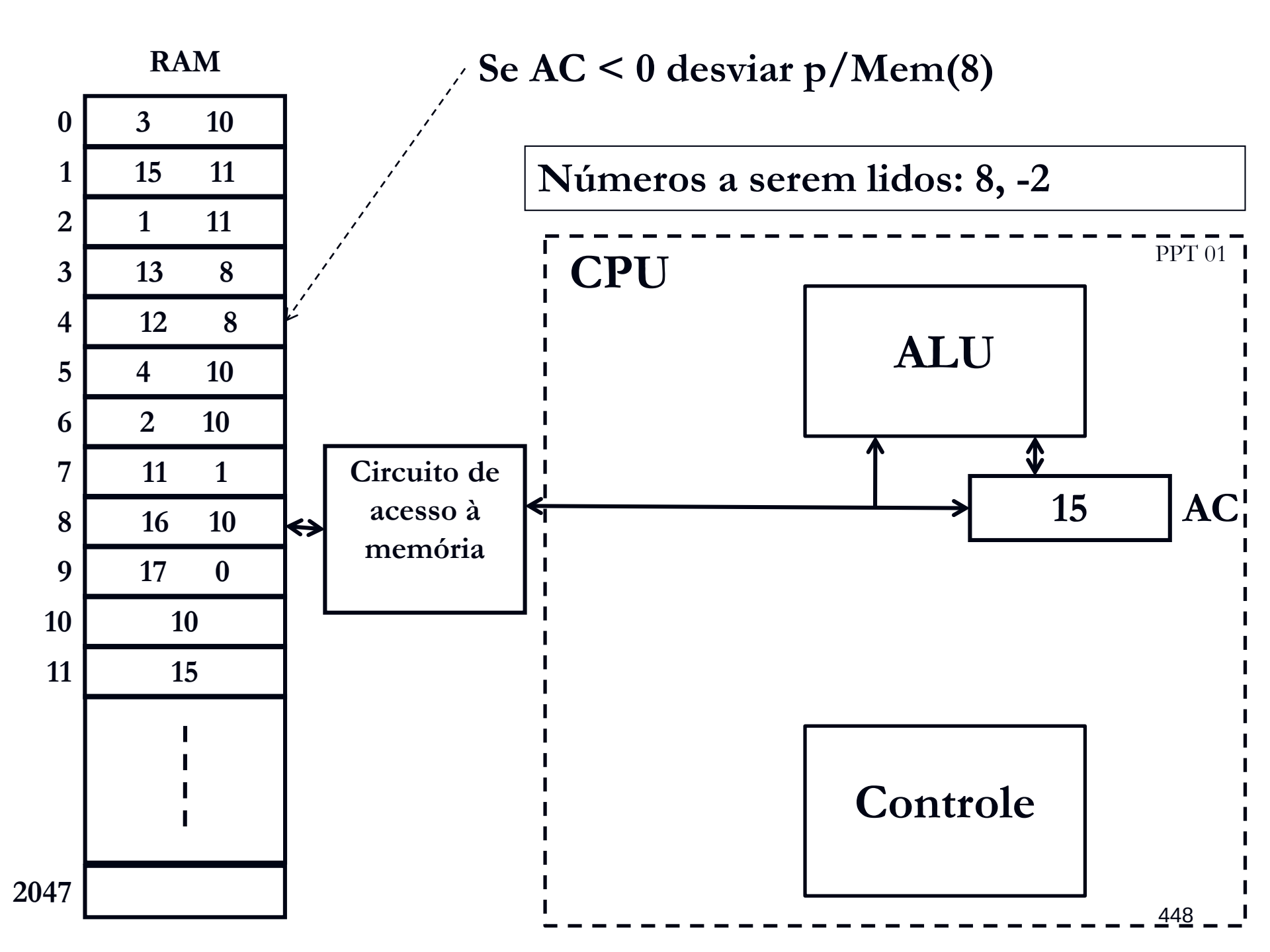

RAM
Ler(Mem(11))
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 10
11 15
2047
Números a serem lidos: 8, -2
CPU
PPT 01
ALU
Circuito de acesso à memória
10
AC
Controle
443

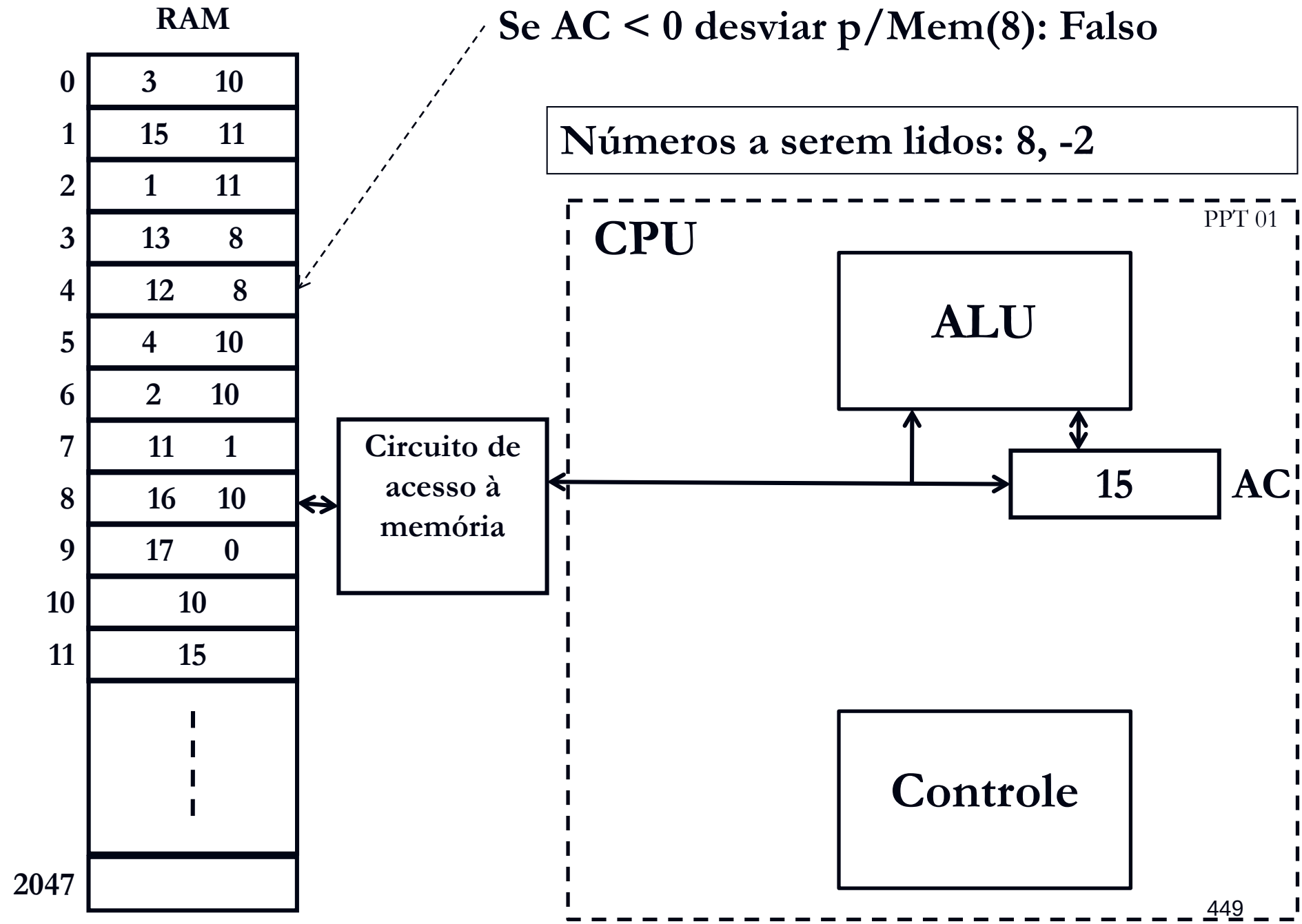

RAM
AC ← Mem(11)
Números a serem lidos: 8, -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 10
11 15
2047
CPU
PPT 01
ALU
Circuito de acesso à memória
10
AC
Controle
444

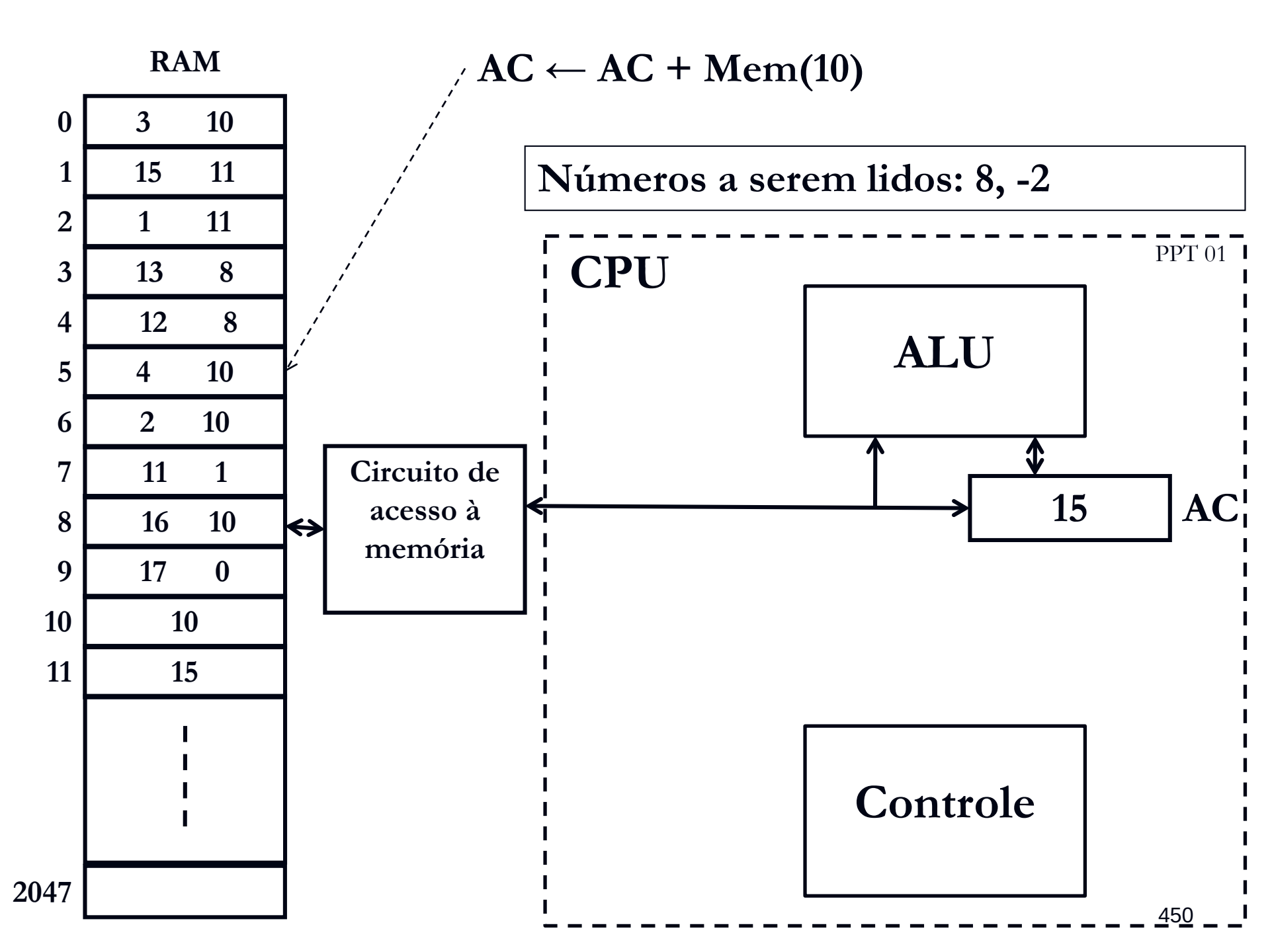

RAM
AC ← Mem(11)
Números a serem lidos: 8, -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 10
11 15
2047
CPU
PPT 01
ALU
Circuito de acesso à memória
15
AC
Controle
445

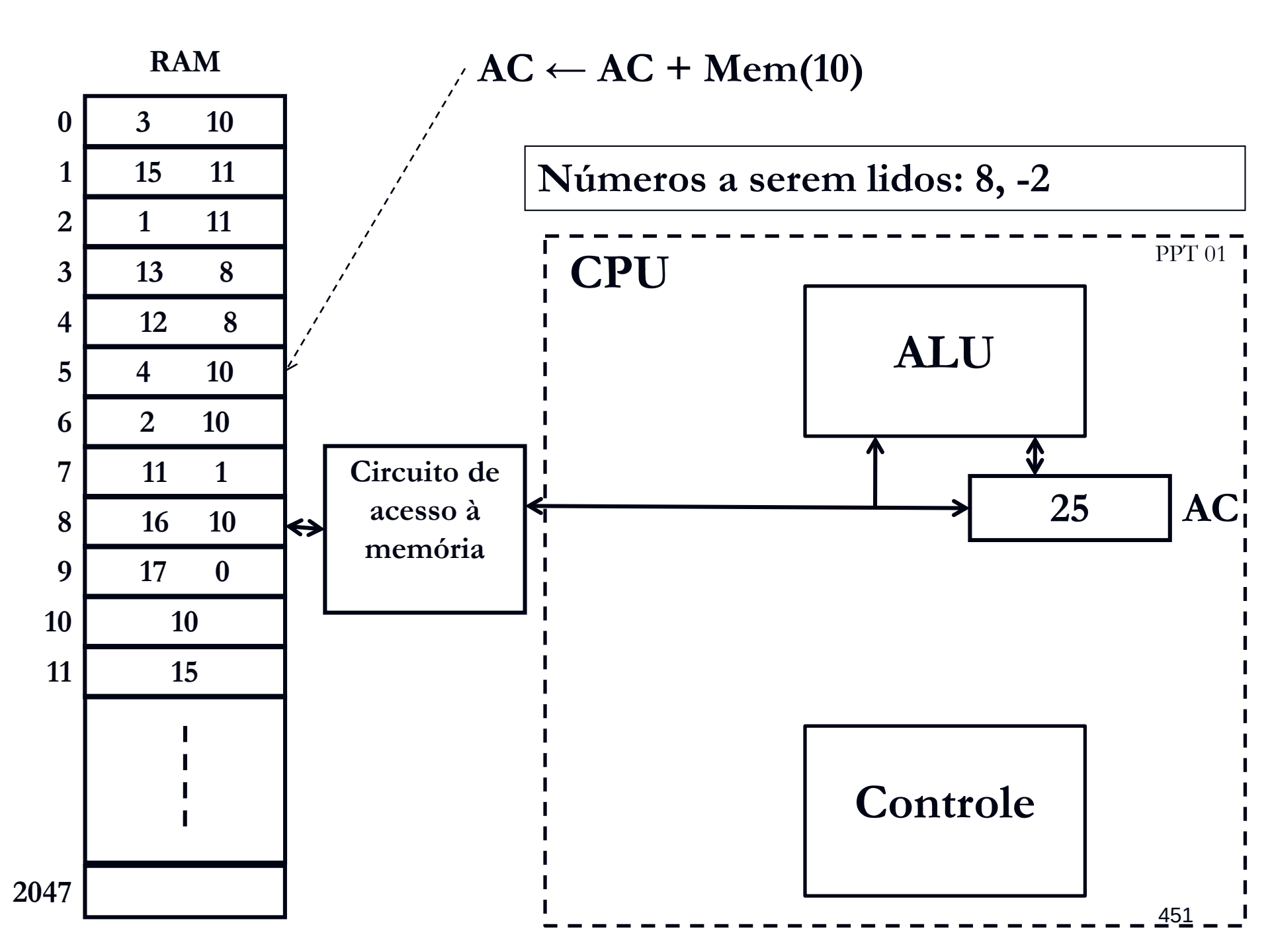

RAM
Se AC = 0 desviar p/Mem(8)
Números a serem lidos: 8, -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 10
11 15
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
15
AC
Controle
446

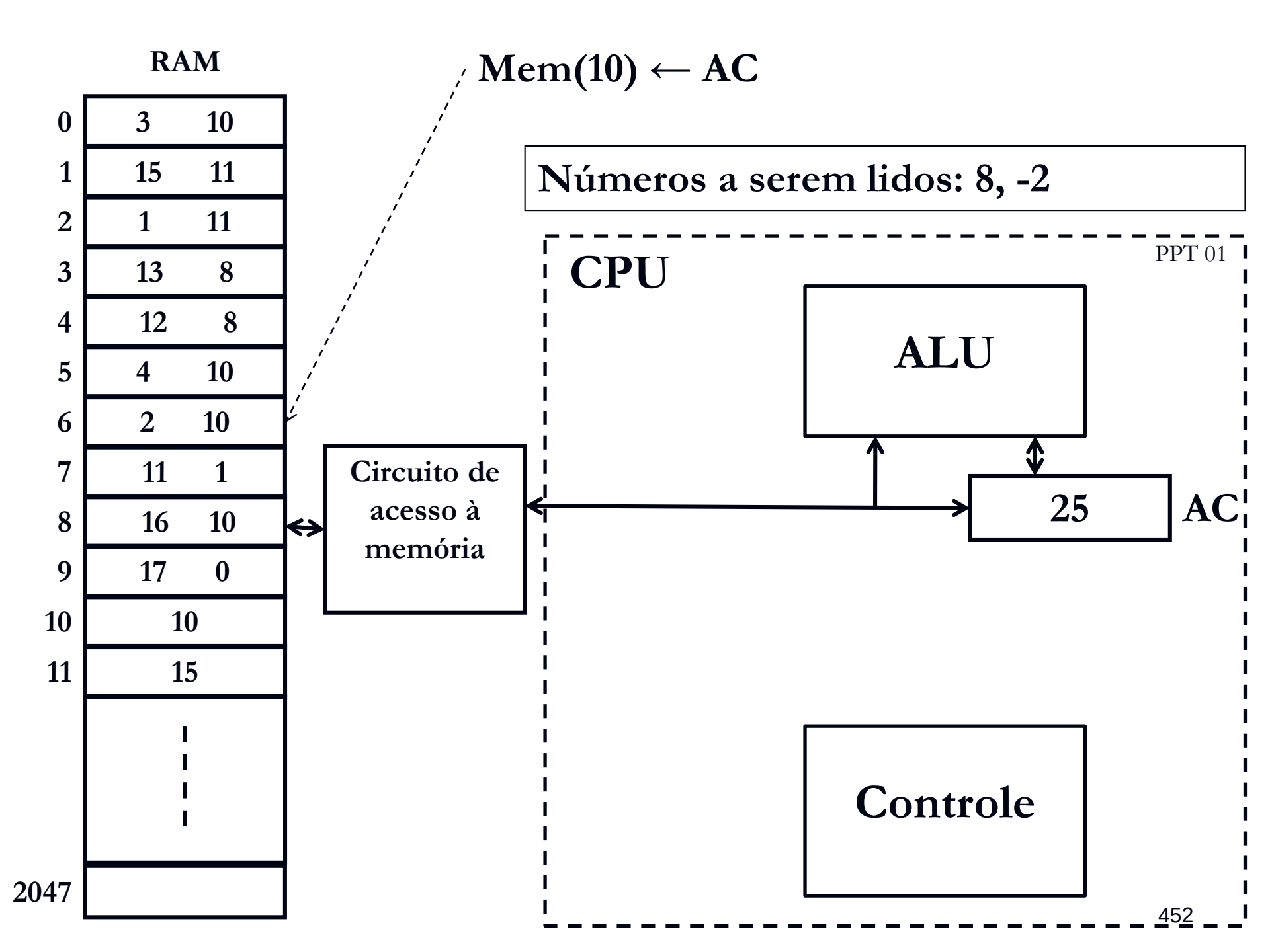

RAM
Se AC = 0 desviar p/Mem(8): falso
Números a serem lidos: 8, -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 10
11 15
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
15
AC
Controle
447

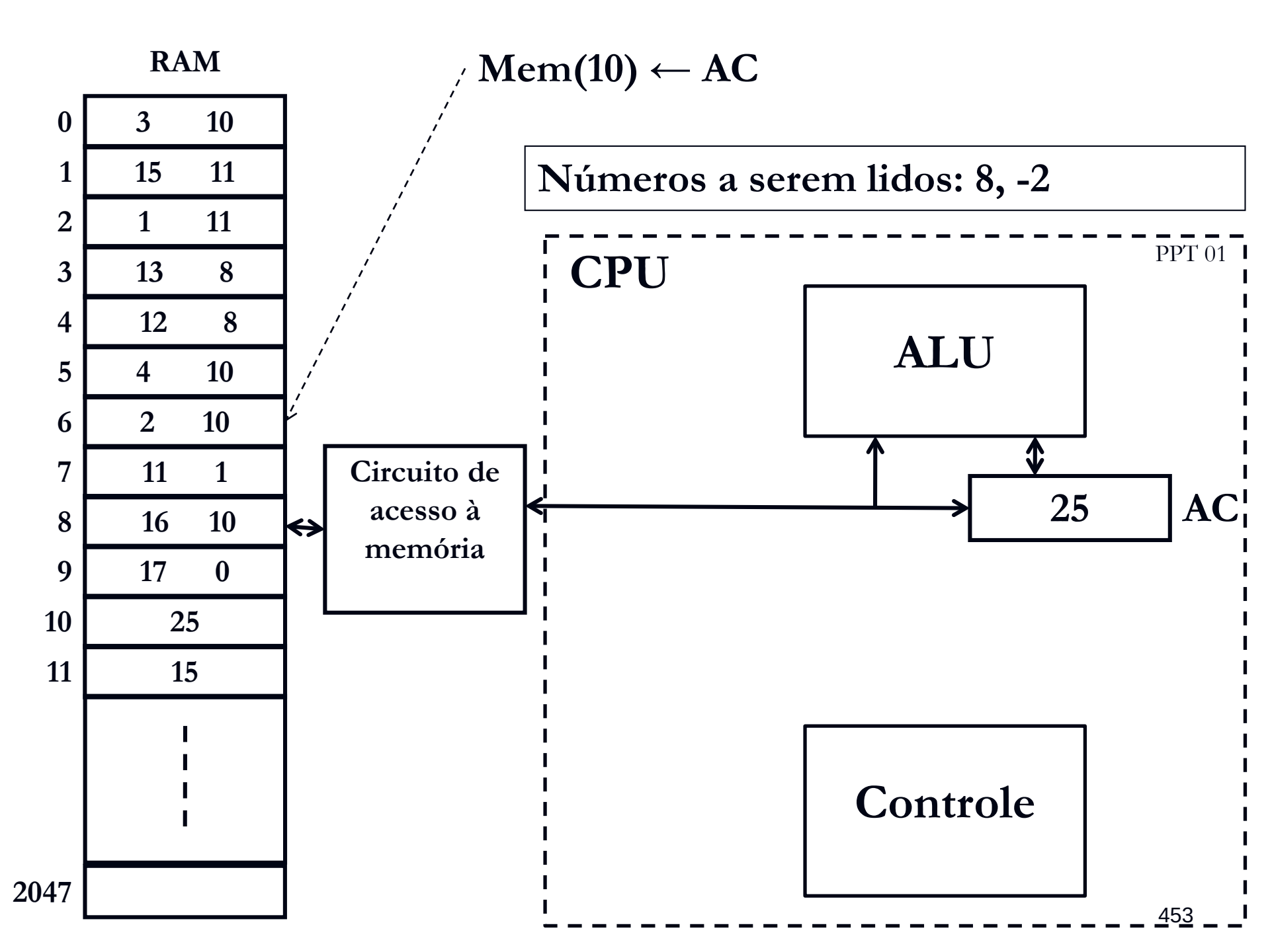

RAM
Se AC < 0 desviar p/Mem(8)
Números a serem lidos: 8, -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 10
11 15
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
15
AC
Controle
448

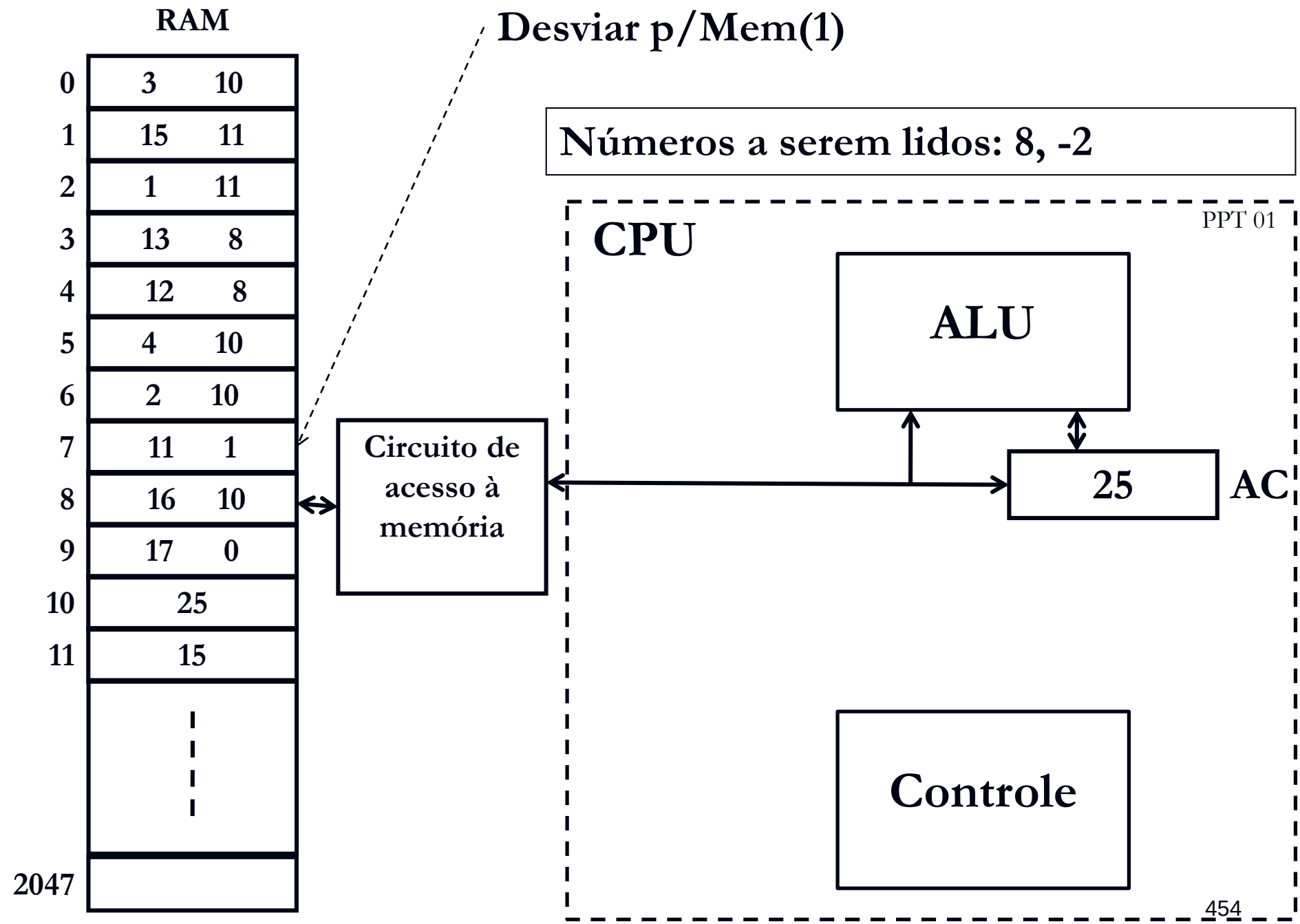

RAM
Se AC < 0 desviar p/Mem(8): Falso
Números a serem lidos: 8, -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 10
11 15
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
15
AC
Controle
449

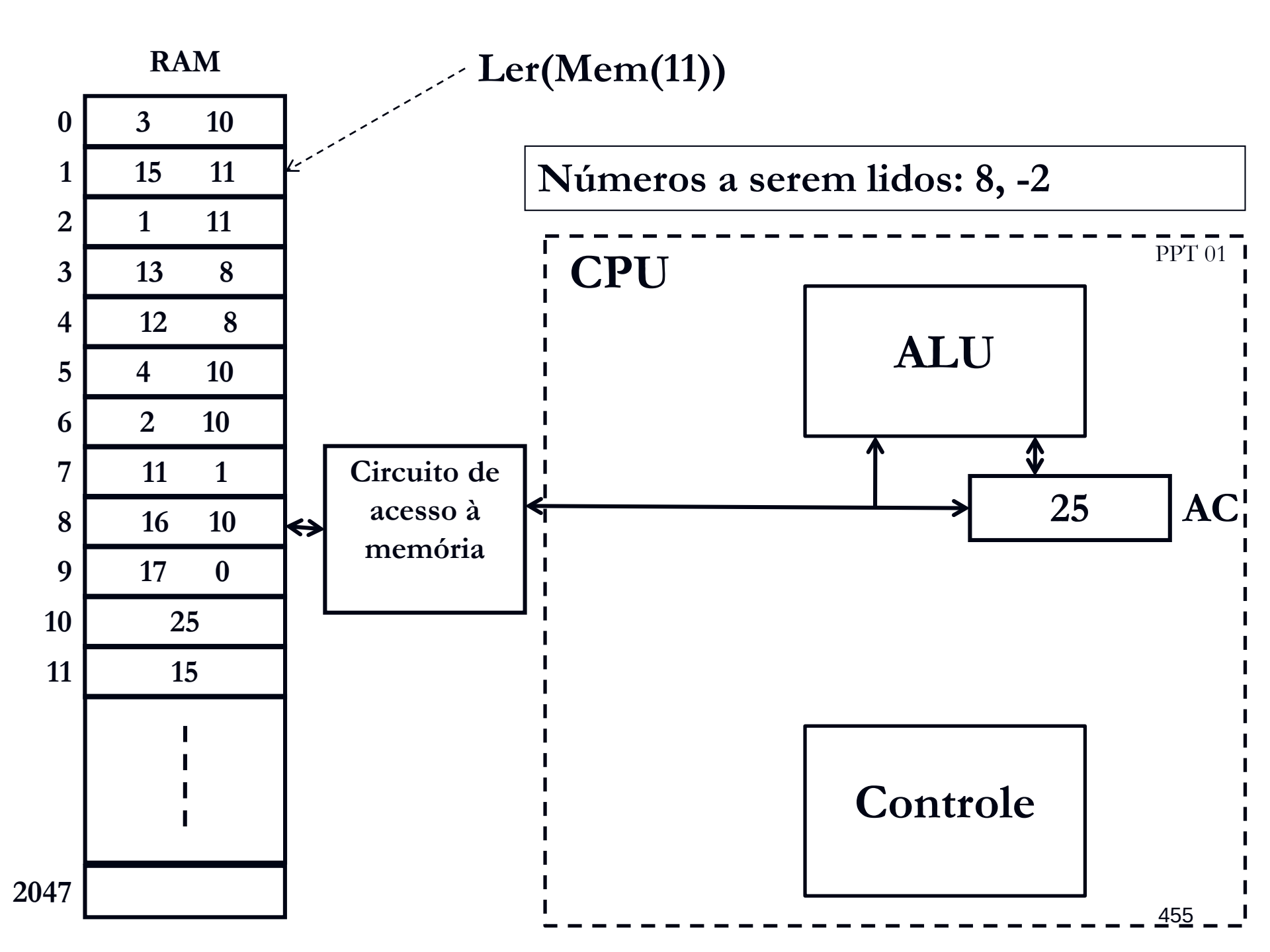

RAM
AC ← AC + Mem(10)
Números a serem lidos: 8, -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 10
11 15
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
15
AC
Controle
450

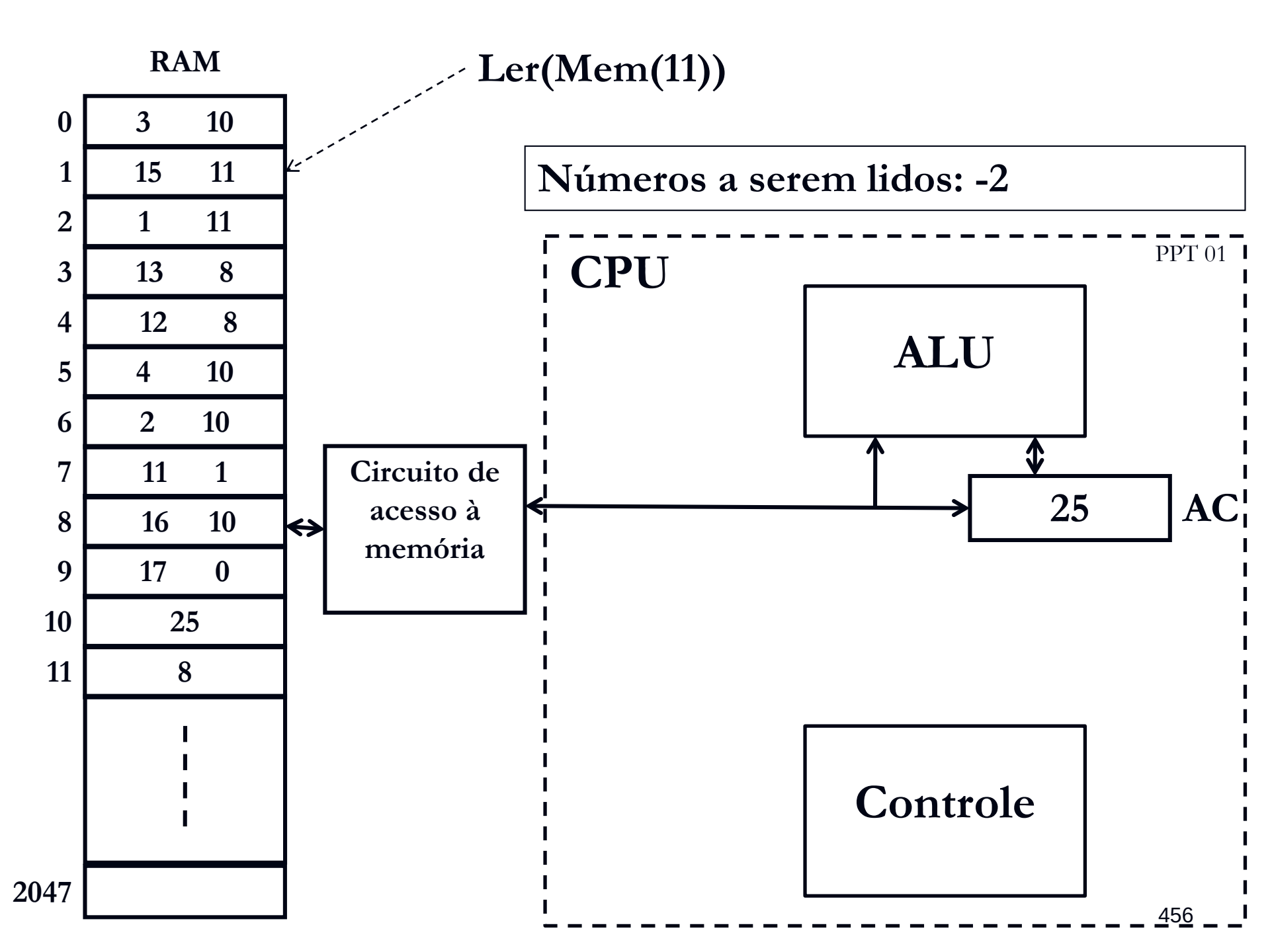

RAM
AC ← AC + Mem(10)
Números a serem lidos: 8, -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 10
11 15
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
25
AC
Controle
451

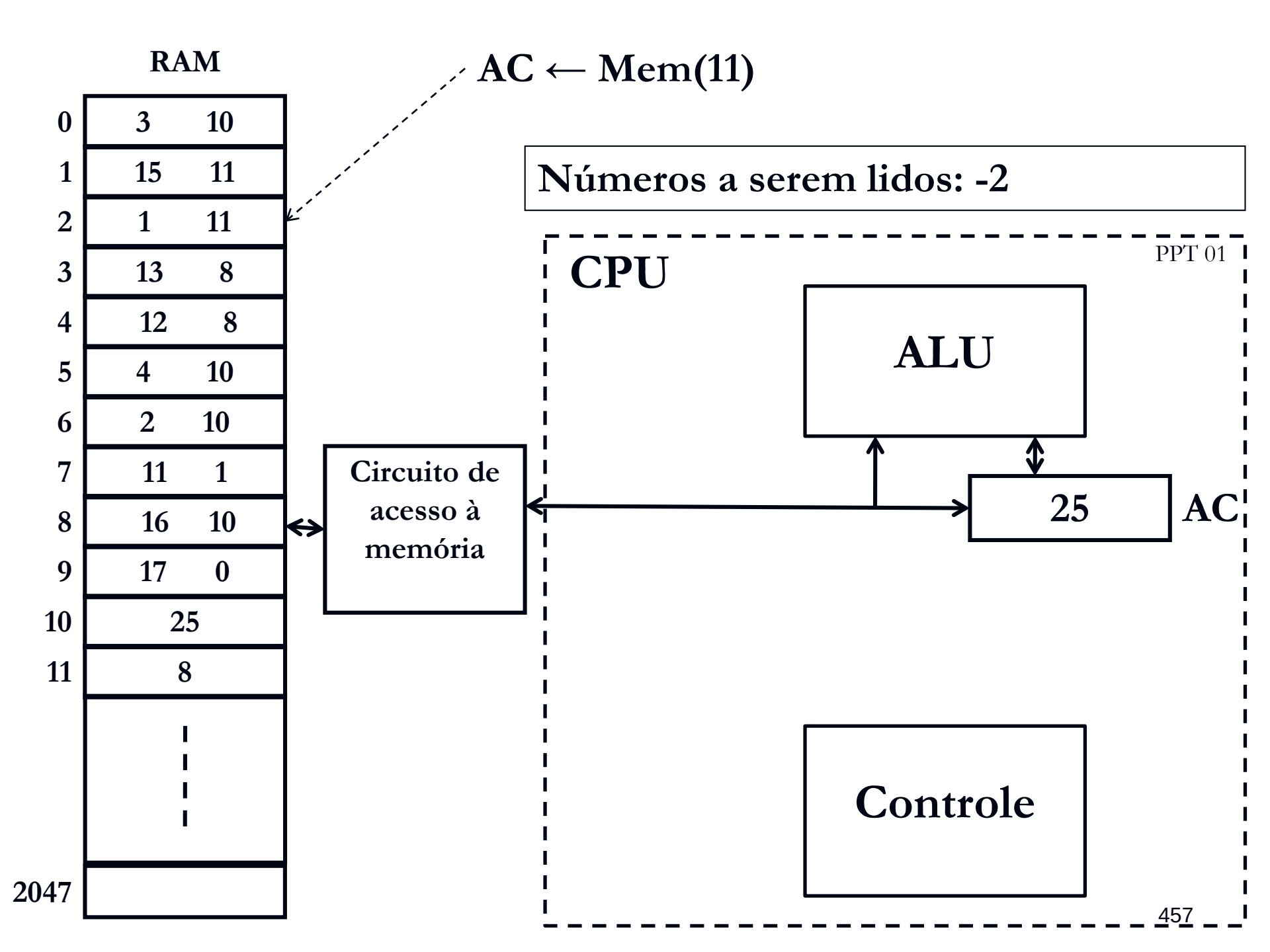

RAM
Mem(10) ← AC
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 10
11 15
2047
Números a serem lidos: 8, -2
CPU
PPT 01
ALU
Circuito de acesso à memória
25
AC
Controle
452

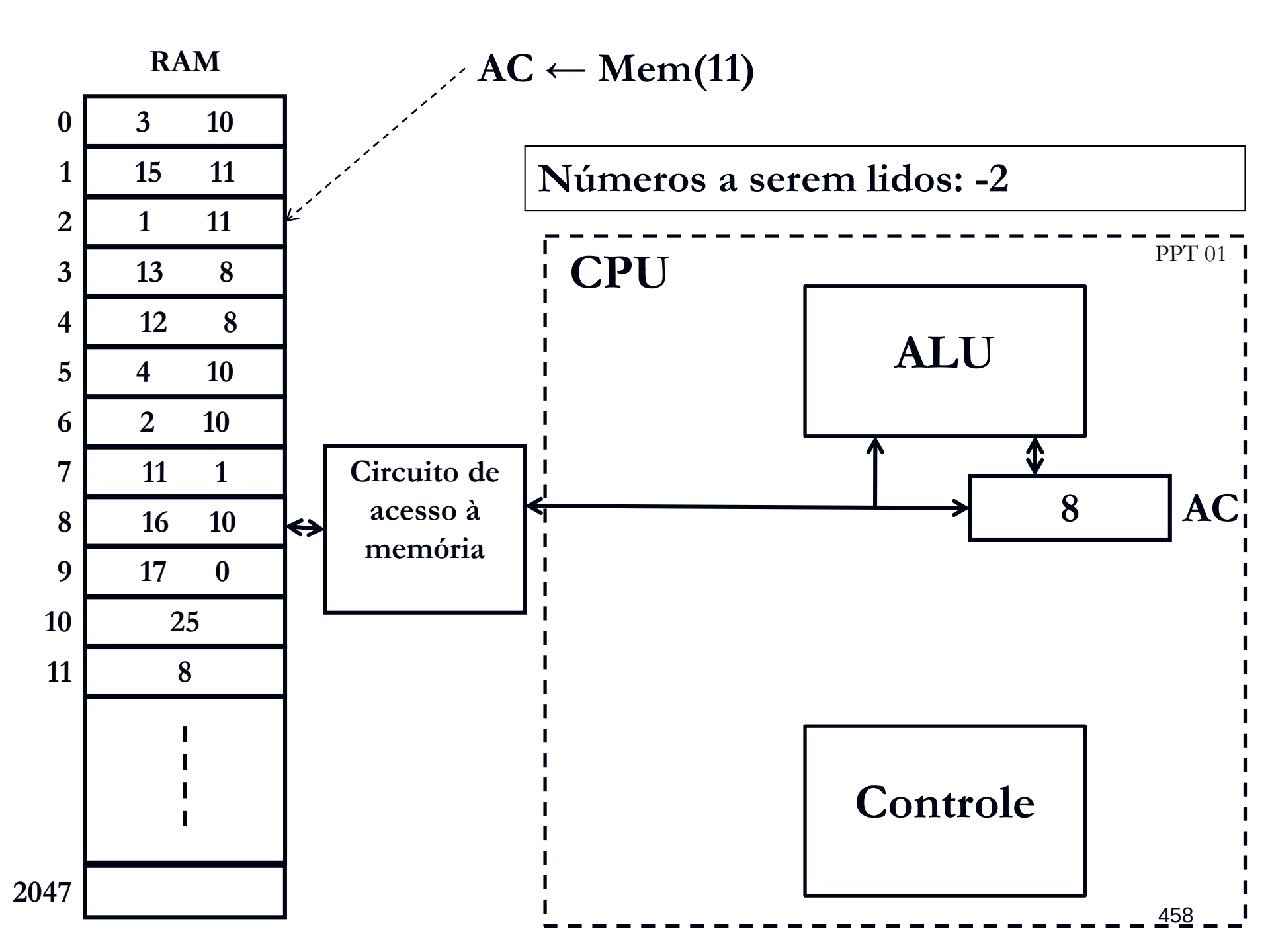

RAM
Mem(10) ← AC
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 25
11 15
2047
Números a serem lidos: 8, -2
CPU
PPT 01
ALU
Circuito de acesso à memória
25
AC
Controle
453

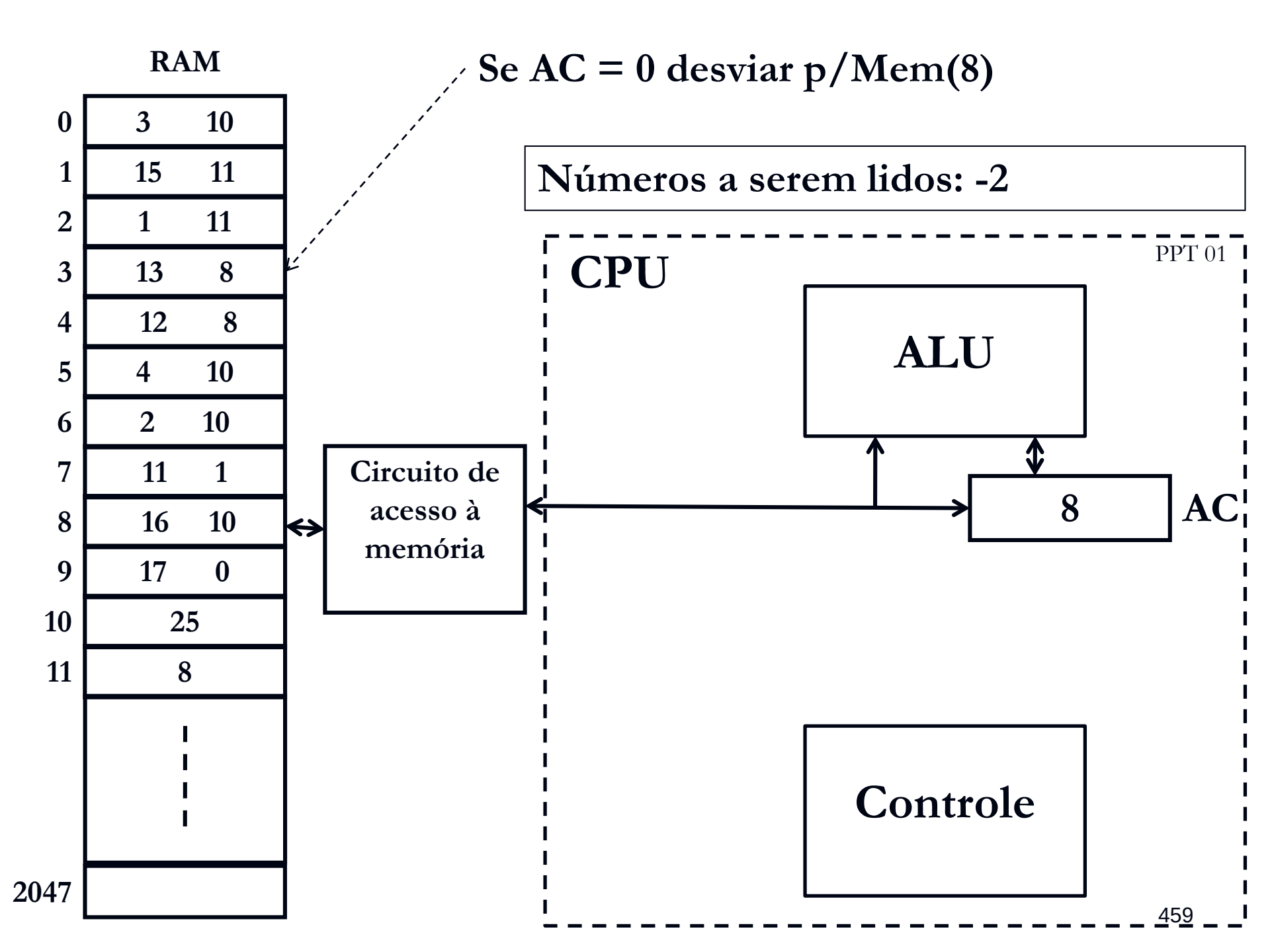

RAM
Desviar p/Mem(1)
Números a serem lidos: 8, -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 25
11 15
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
25
AC
Controle
454

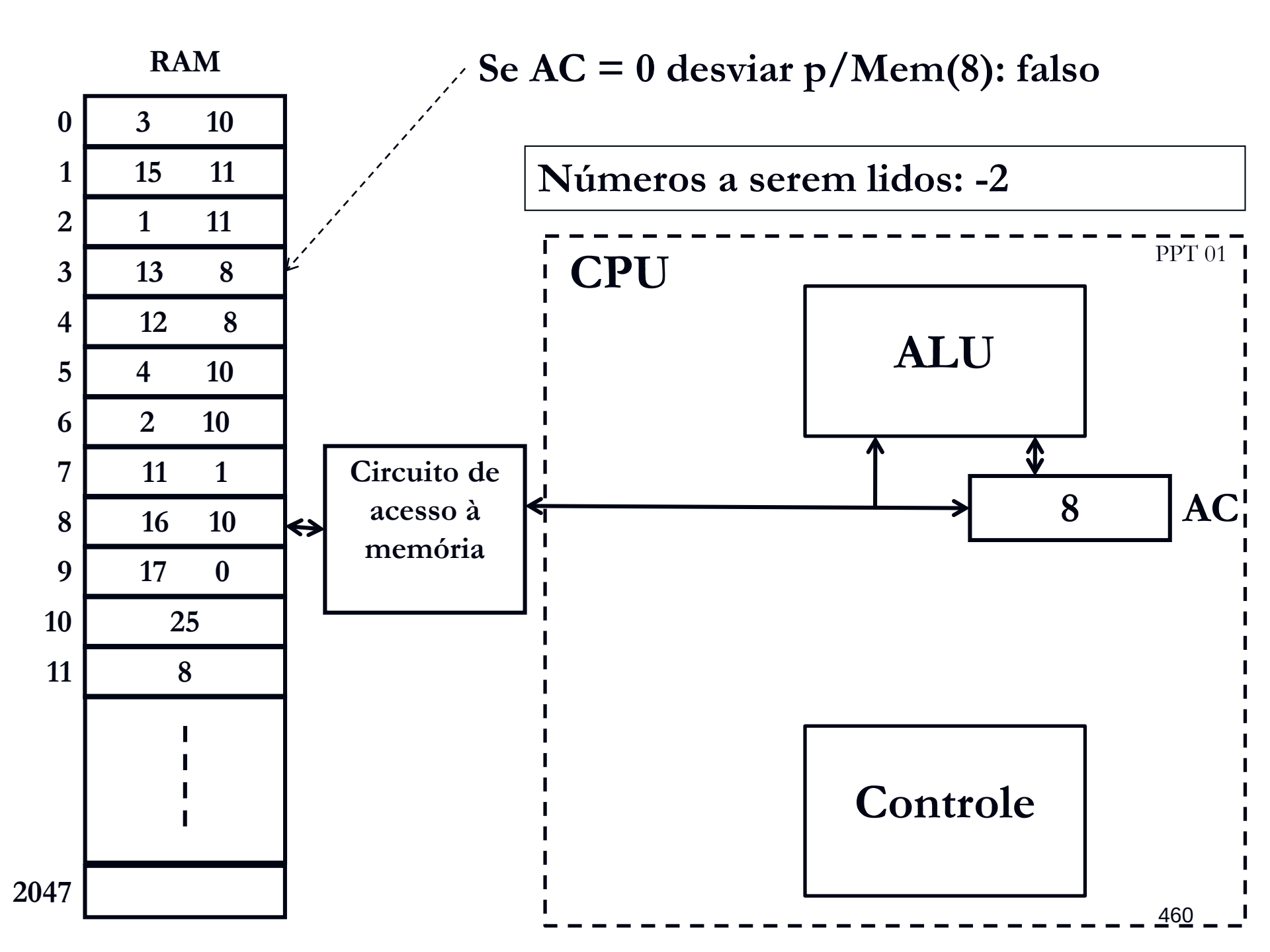

RAM
Ler(Mem(11))
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 25
11 15
2047
Números a serem lidos: 8, -2
CPU
PPT 01
ALU
Circuito de acesso à memória
25
AC
Controle
455

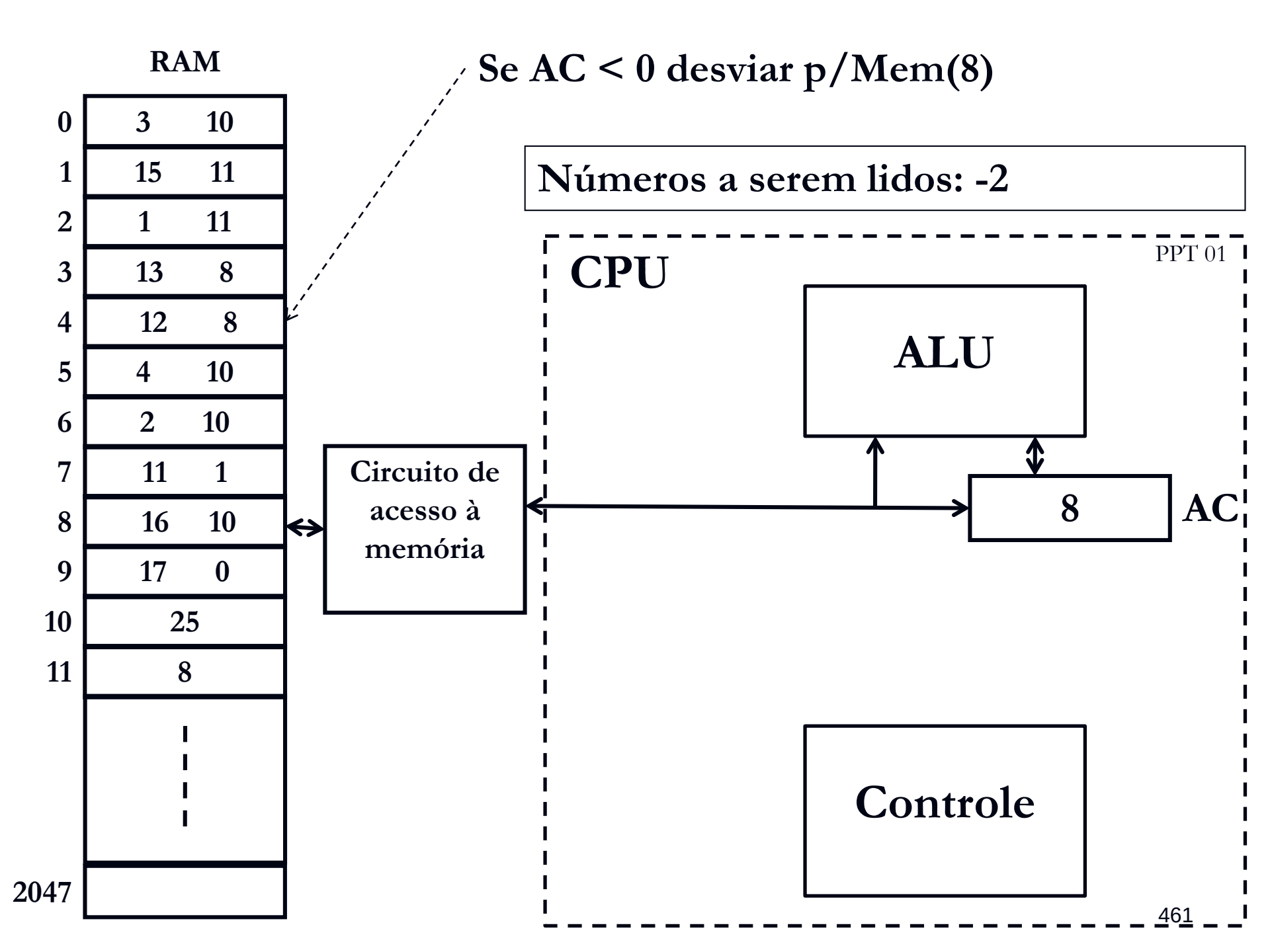

RAM
Ler(Mem(11))
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 25
11 8
2047
Números a serem lidos: -2
CPU
PPT 01
ALU
Circuito de acesso à memória
25
AC
Controle
456

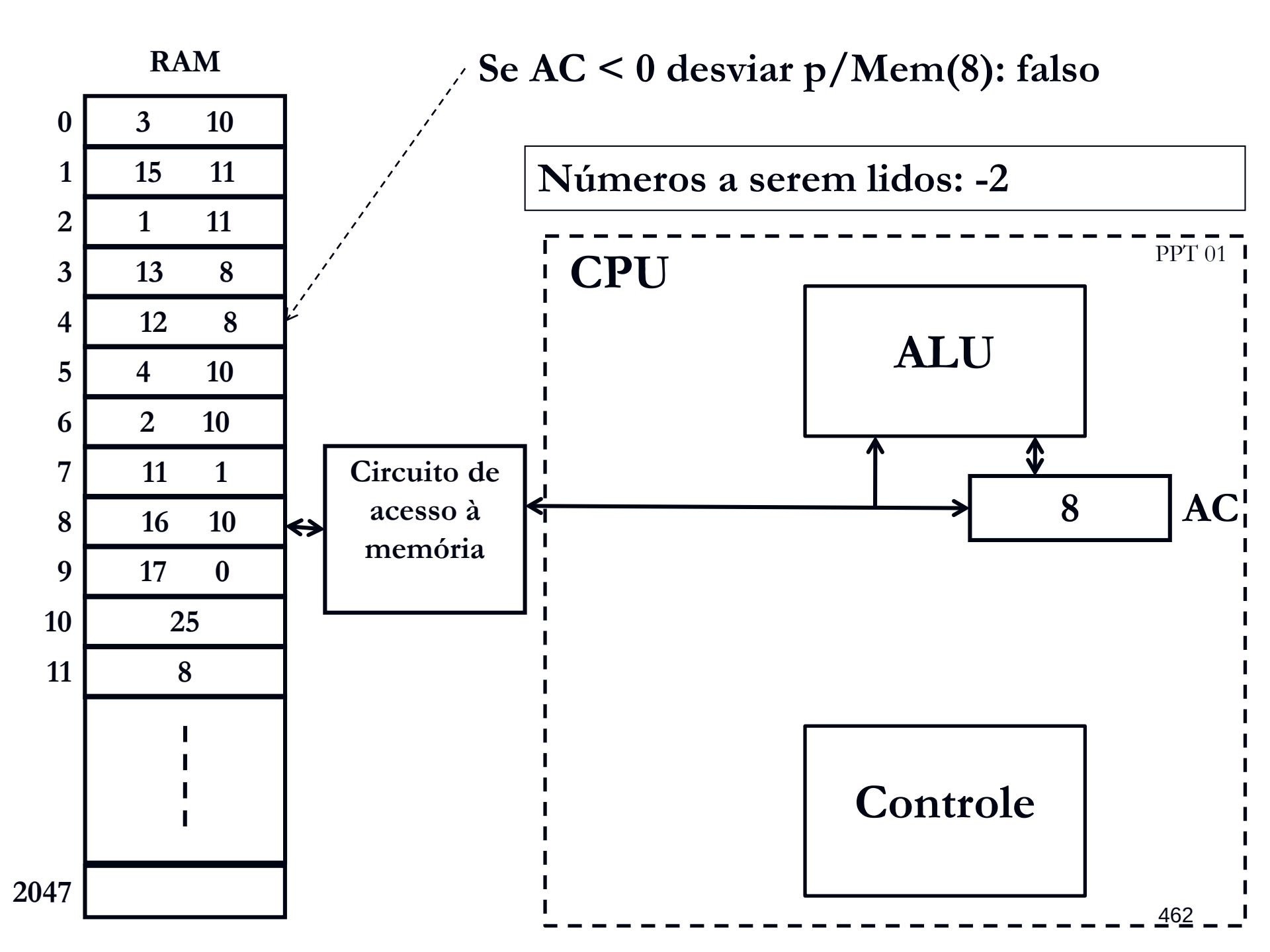

RAM
AC ← Mem(11)
Números a serem lidos: -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 25
11 8
2047
CPU
PPT 01
ALU
Circuito de acesso à memória
25
AC
Controle
457

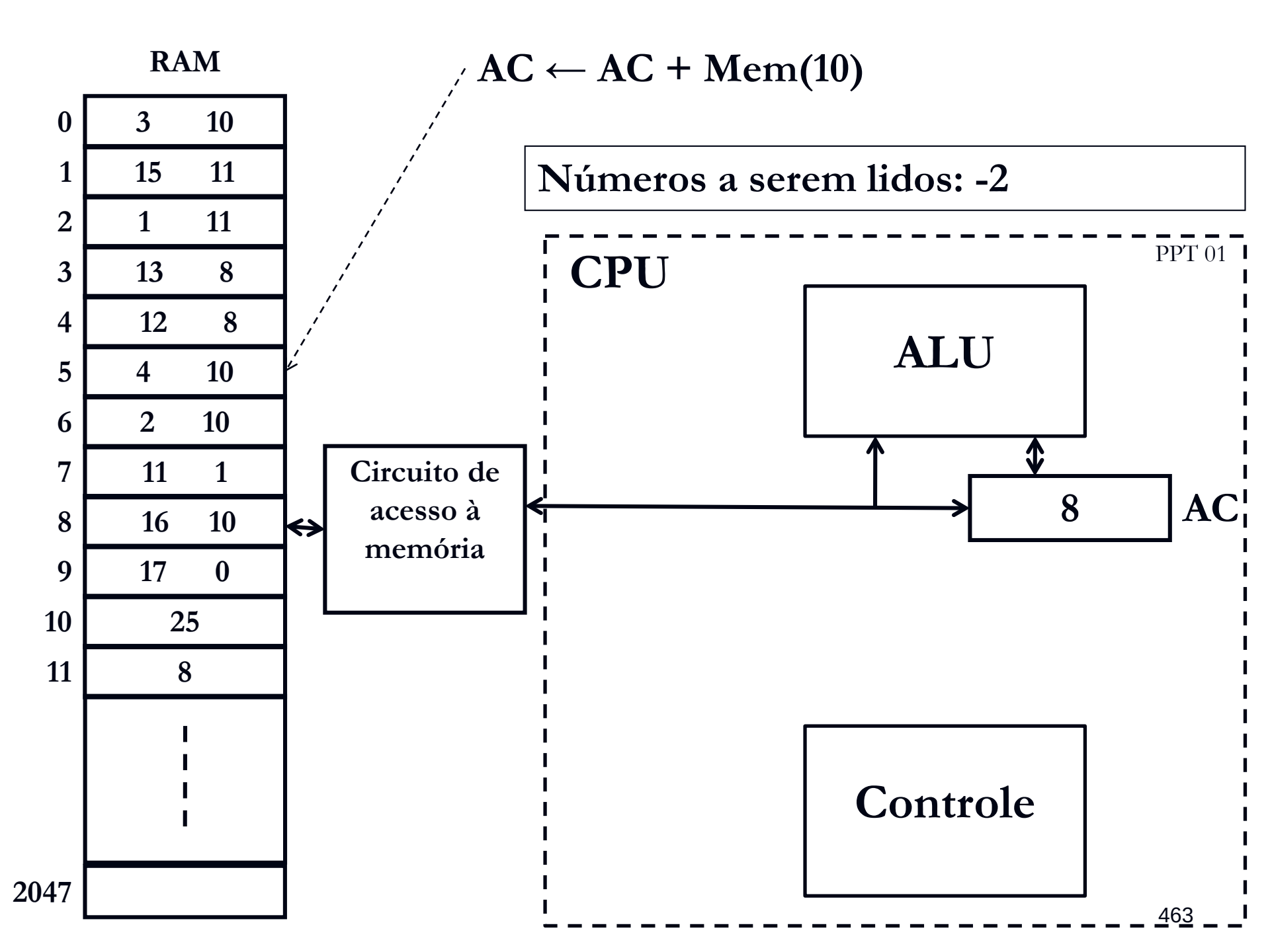

RAM
AC ← Mem(11)
Números a serem lidos: -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 25
11 8
2047
CPU
PPT 01
ALU
Circuito de acesso à memória
8
AC
Controle
458

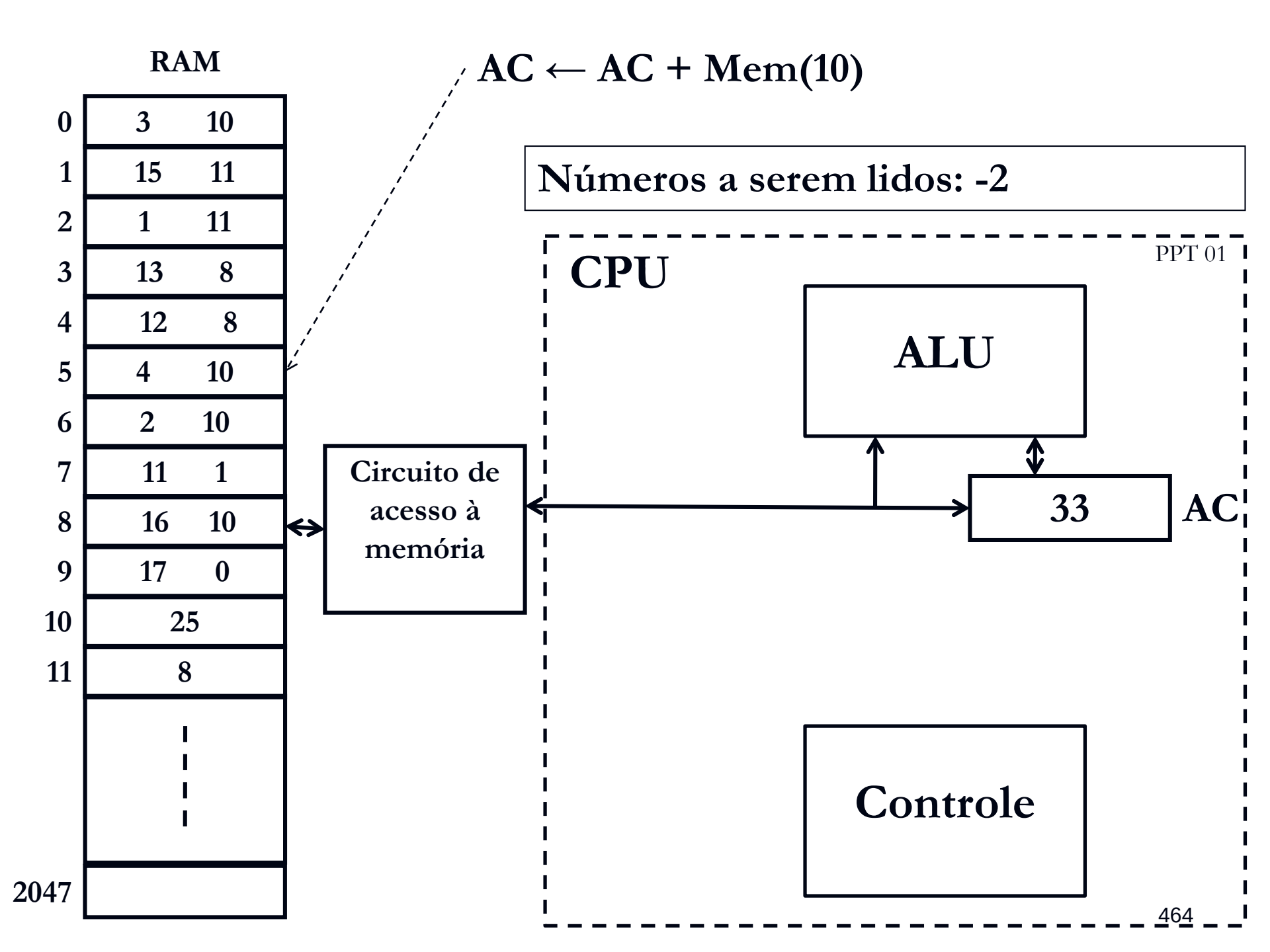

RAM
Se AC = 0 desviar p/Mem(8)
Números a serem lidos: -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 25
11 8
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
8
AC
Controle
459

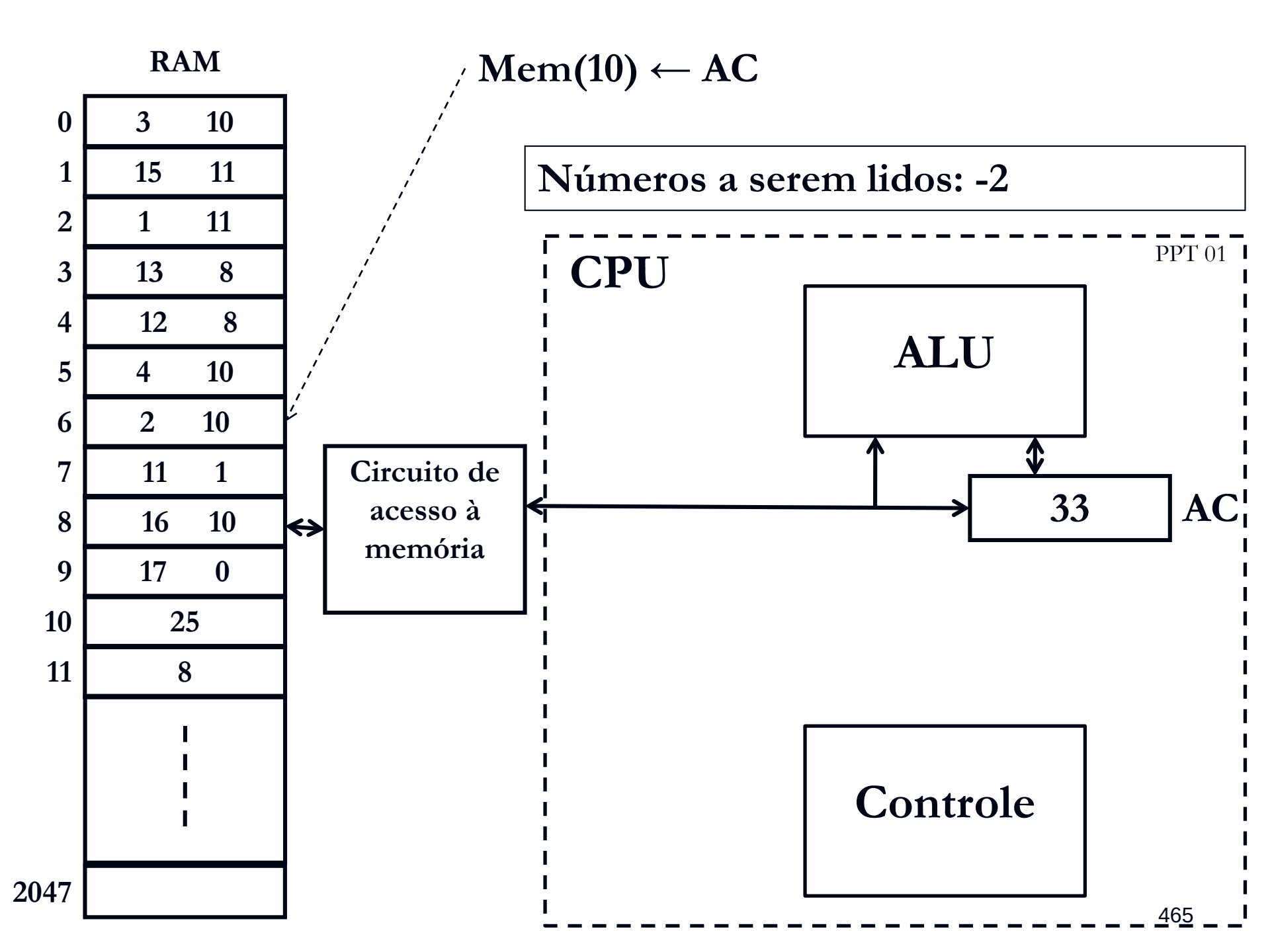

RAM
Se AC = 0 desviar p/Mem(8): falso
Números a serem lidos: -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 25
11 8
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
8
AC
Controle
460

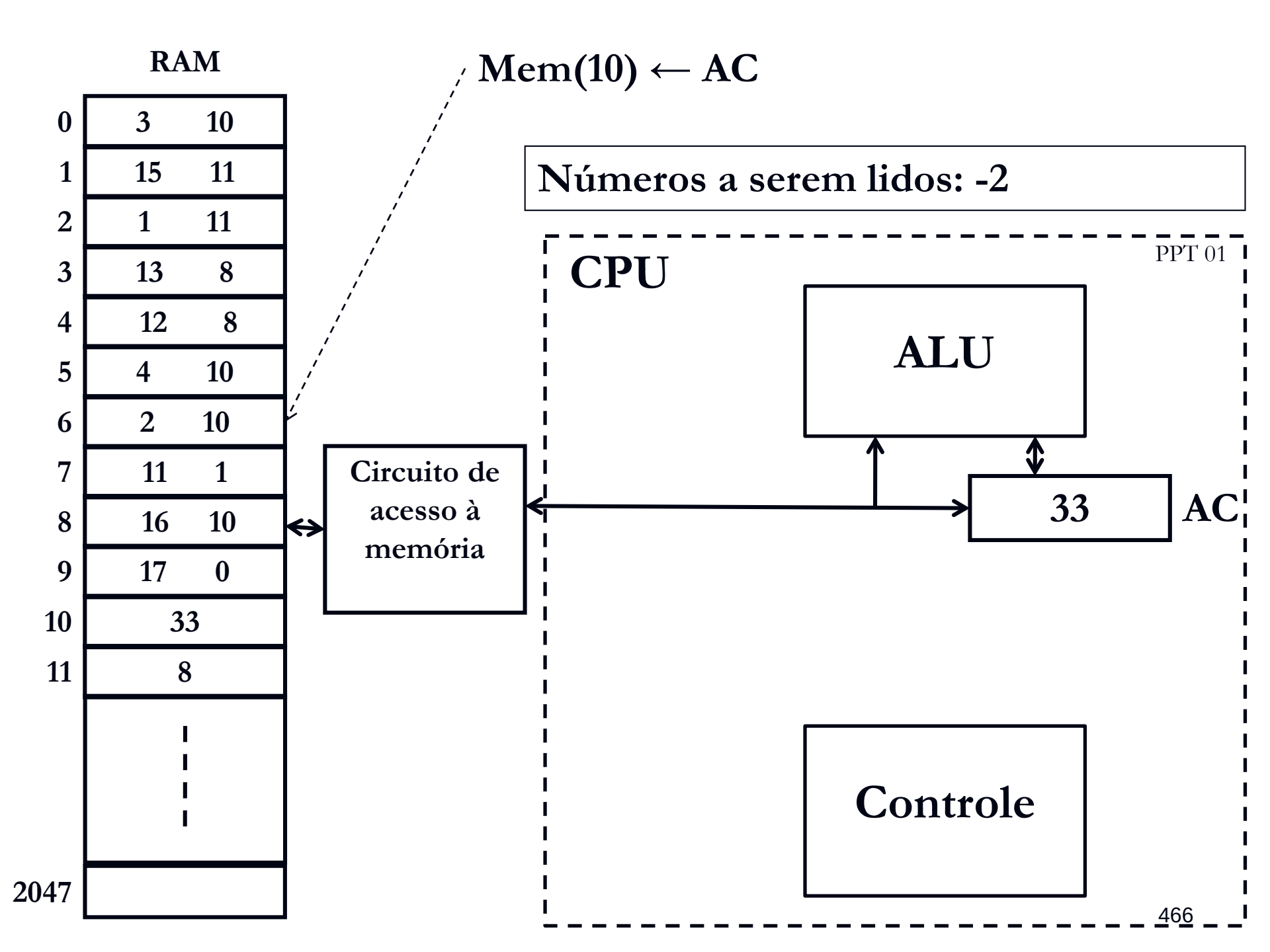

RAM
Se AC < 0 desviar p/Mem(8)
Números a serem lidos: -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 25
11 8
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
8
AC
Controle
461

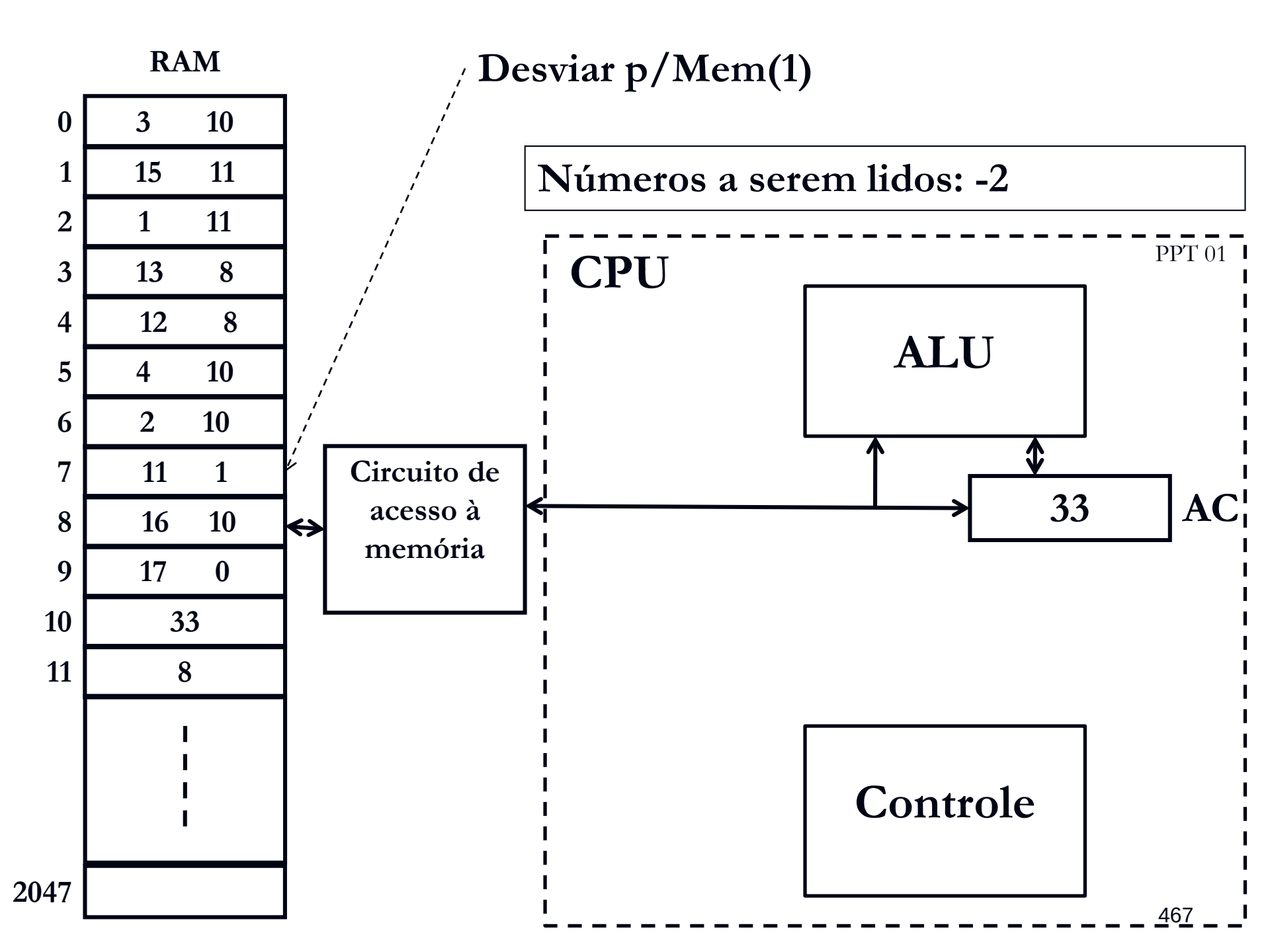

RAM
Se AC < 0 desviar p/Mem(8): falso
Números a serem lidos: -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 25
11 8
2047
CPU
PPT 01
ALU
Circuito de acesso à memória
8
AC
Controle
462

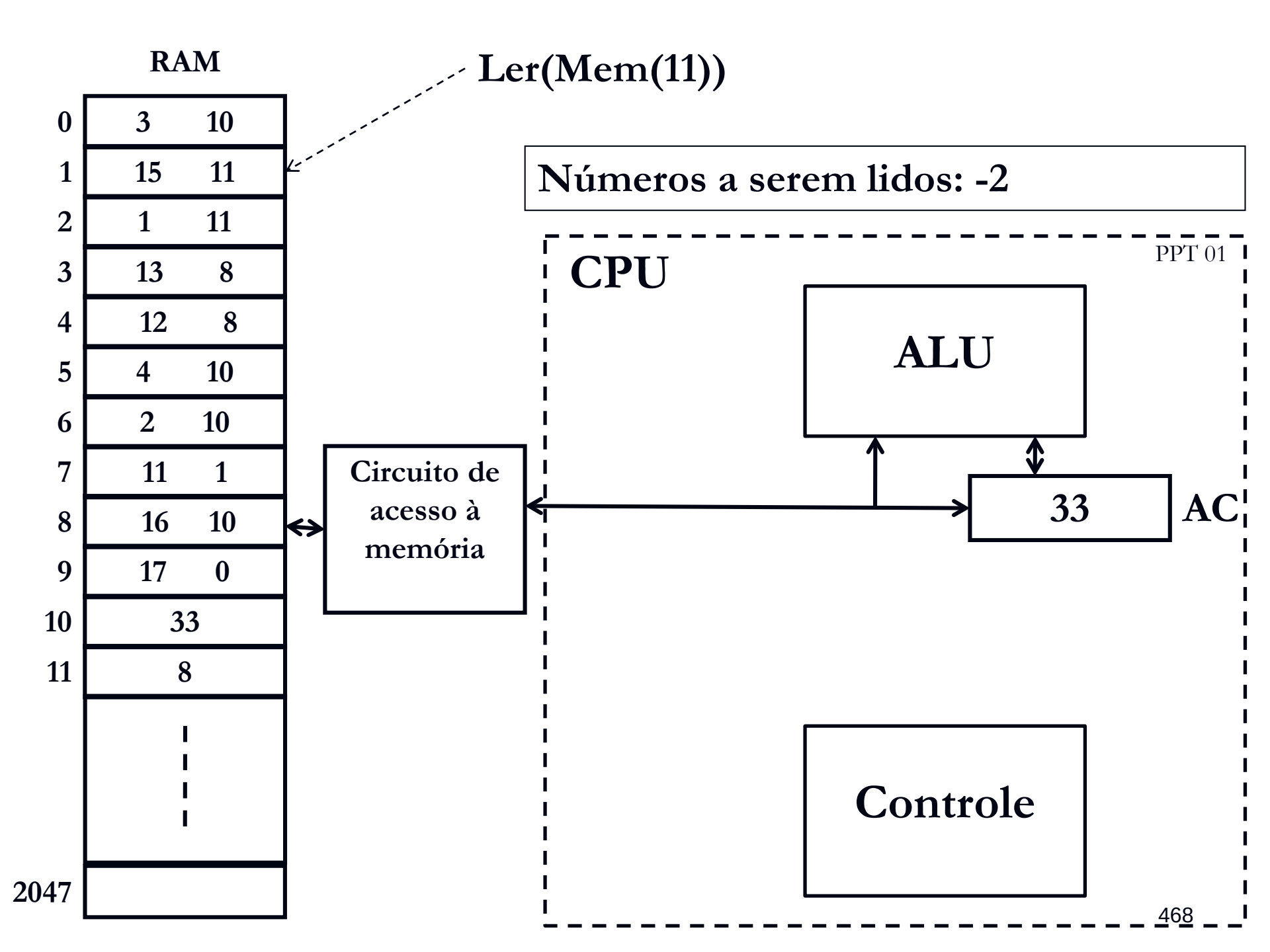

RAM
AC ← AC + Mem(10)
Números a serem lidos: -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 25
11 8
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
8
AC
Controle
463

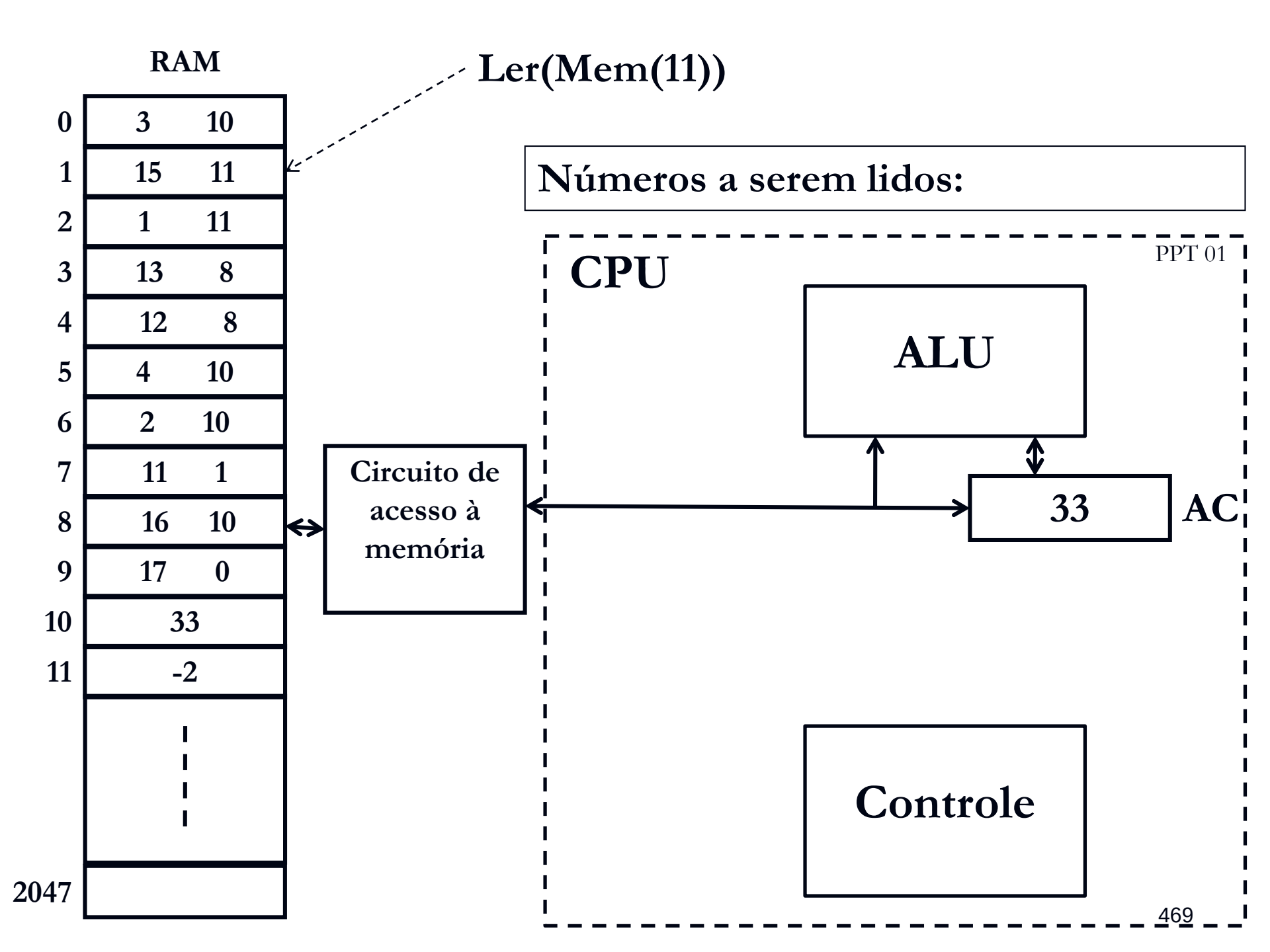

RAM
AC ← AC + Mem(10)
Números a serem lidos: -2
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 25
11 8
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
33
AC
Controle
464

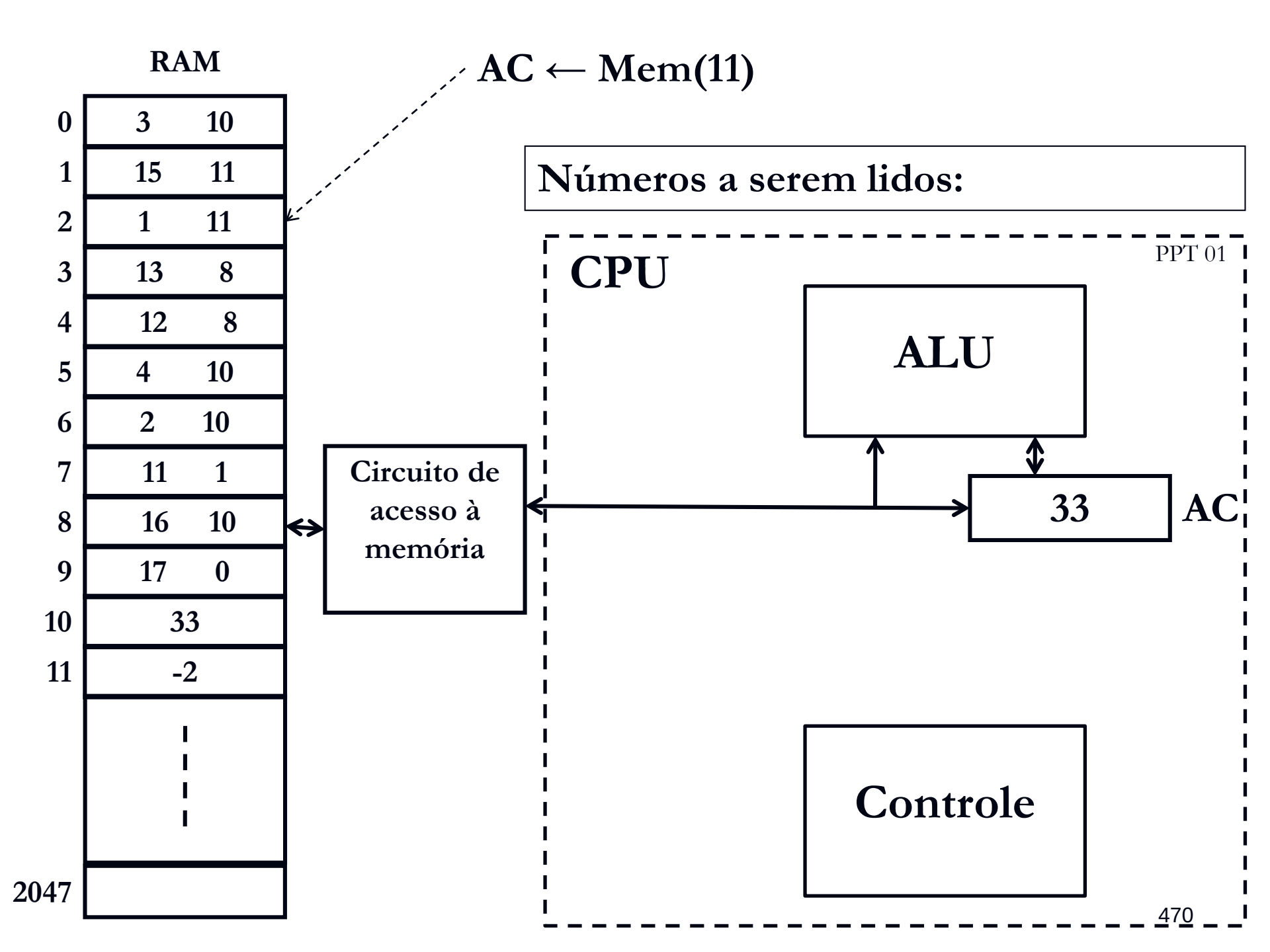

RAM
Mem(10) ← AC
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 25
11 8
2047
Números a serem lidos: -2
CPU
PPT 01
ALU
Circuito de acesso à memória
33
AC
Controle
465

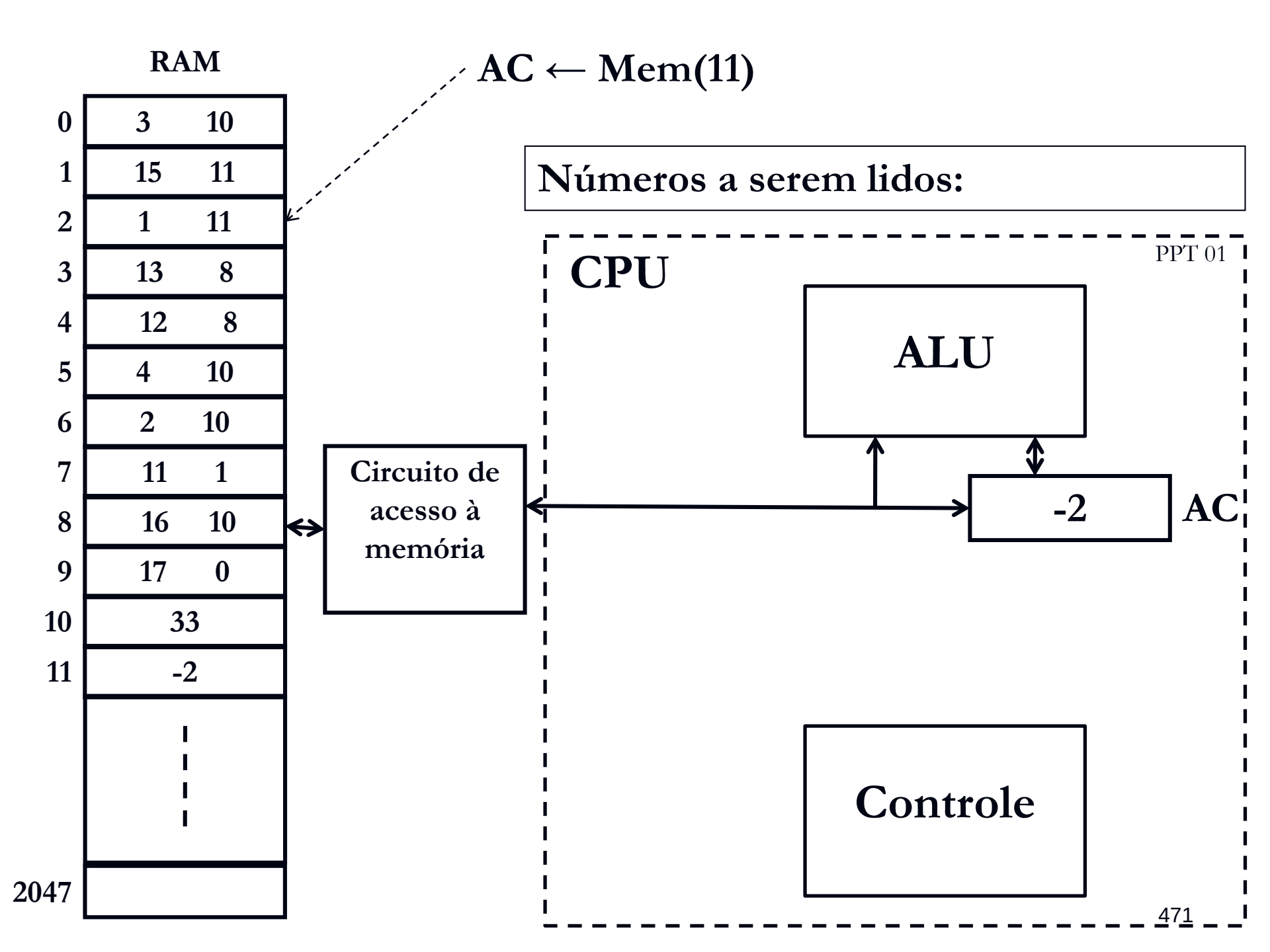

RAM
Mem(10) ← AC
Números a serem lidos: -2
CPU
PPT 01
ALU
Circuito de acesso à memória
33
AC
Controle
0
1
2
3
4
5
6
7
8
9
10
11
2047
3 10
15 11
1 11
13 8
12 8
4 10
2 10
11 1
16 10
17 0
33
8
466

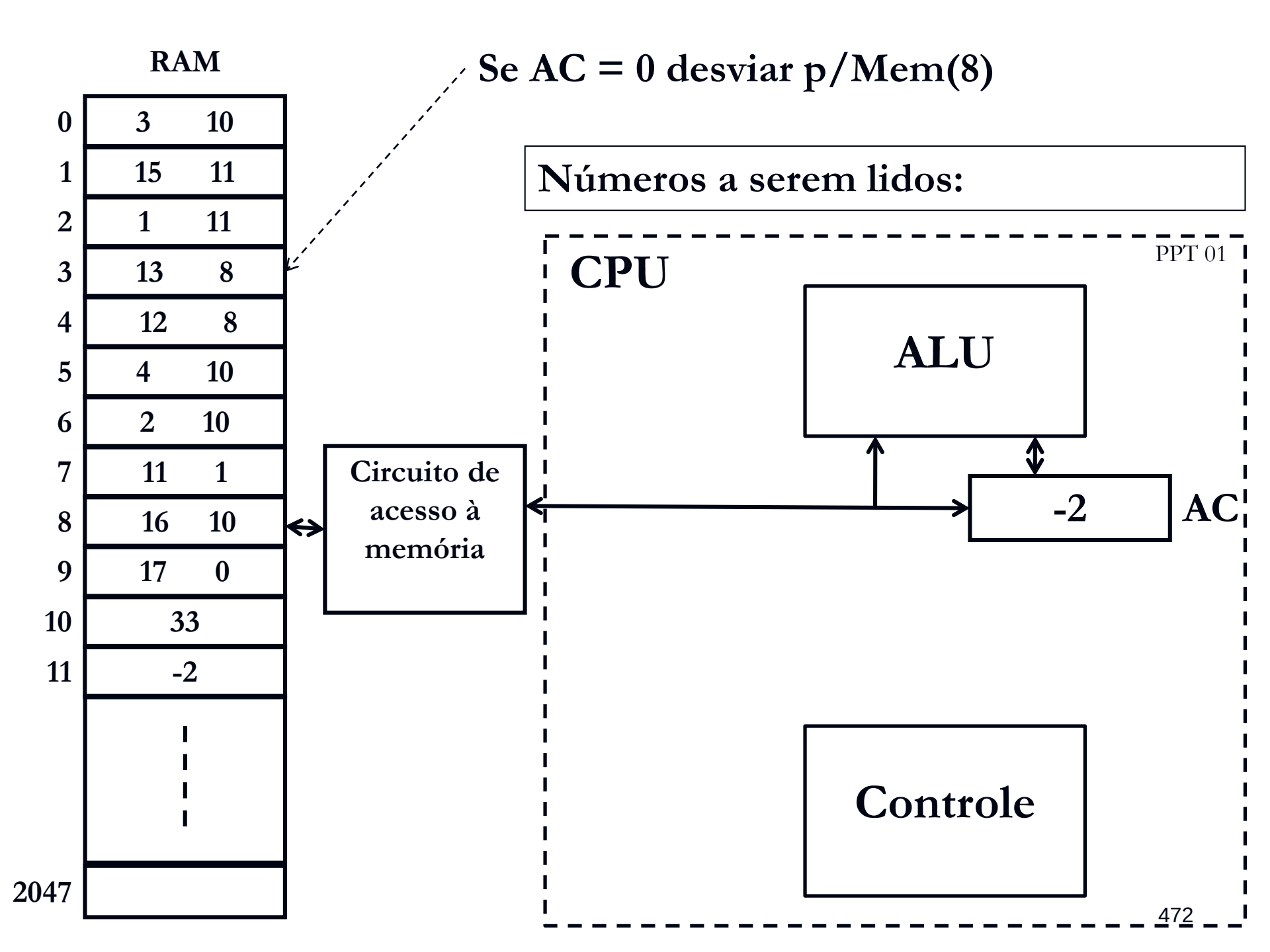

RAM
Desviar p/Mem(1)
Números a serem lidos: -2
CPU
PPT 01
ALU
Circuito de acesso à memória
33
AC
Controle
0
3 10
1
15 11
2
1 11
3
13 8
4
12 8
5
4 10
6
2 10
7
11 1
8
16 10
9
17 0
10
33
11
8
2047
467

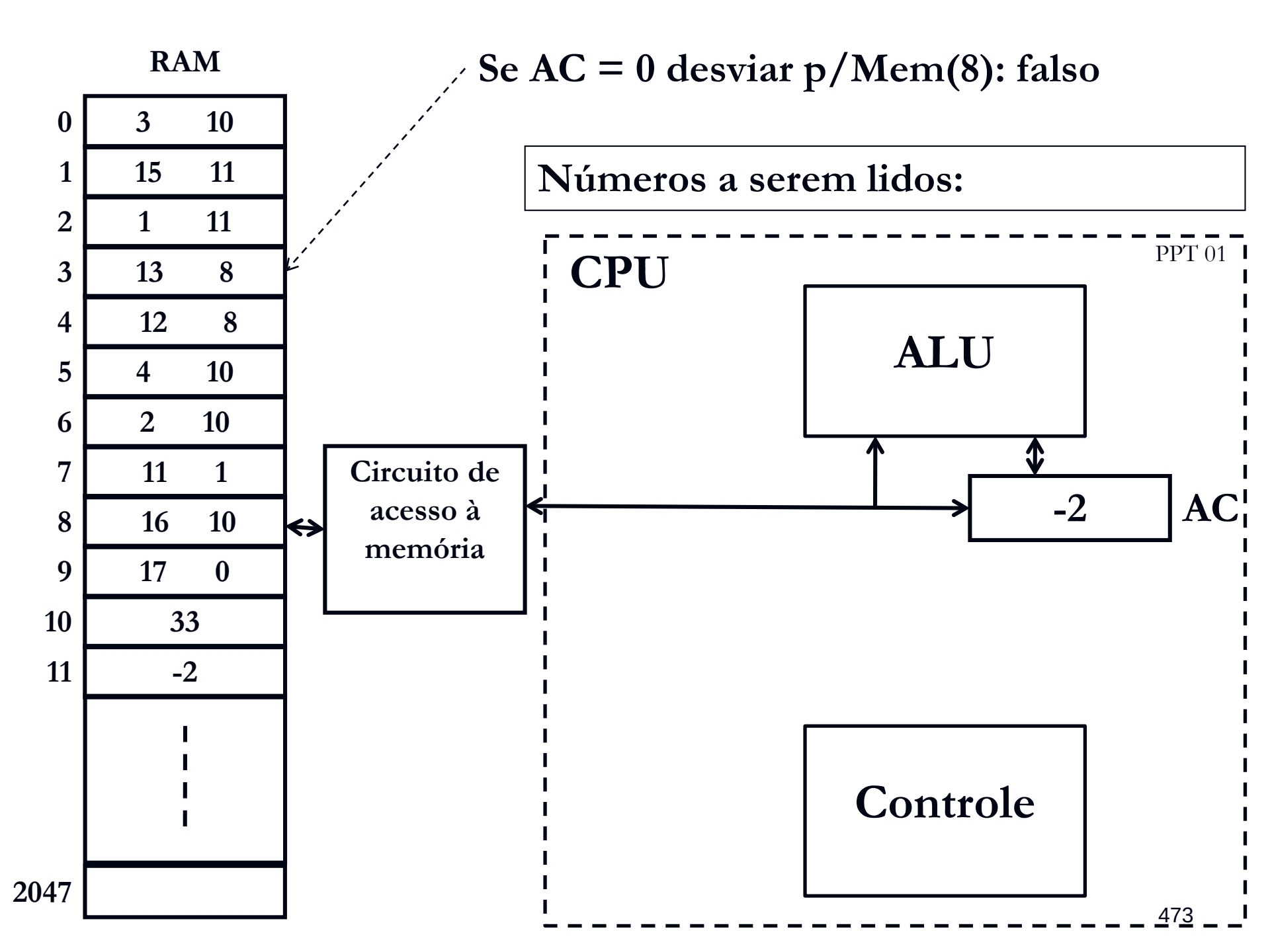

RAM
Ler(Mem(11))
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 33
11 8
2047
Números a serem lidos: -2
CPU
PPT 01
ALU
Circuito de acesso à memória
33
AC
Controle
468

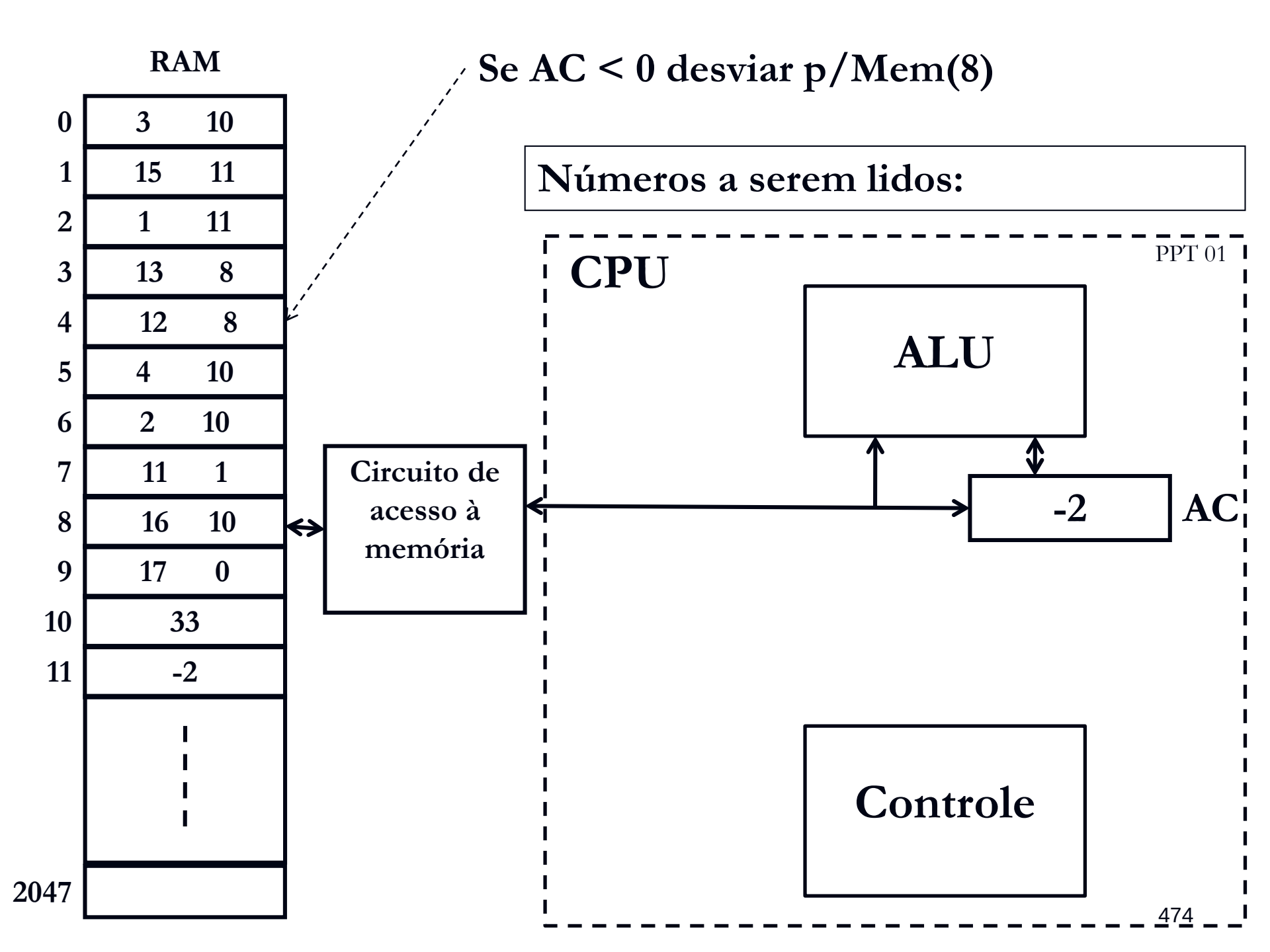

RAM
Ler(Mem(11))
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 33
11 -2
2047
Números a serem lidos:
CPU
PPT 01
ALU
Circuito de acesso à memória
33
AC
Controle
469

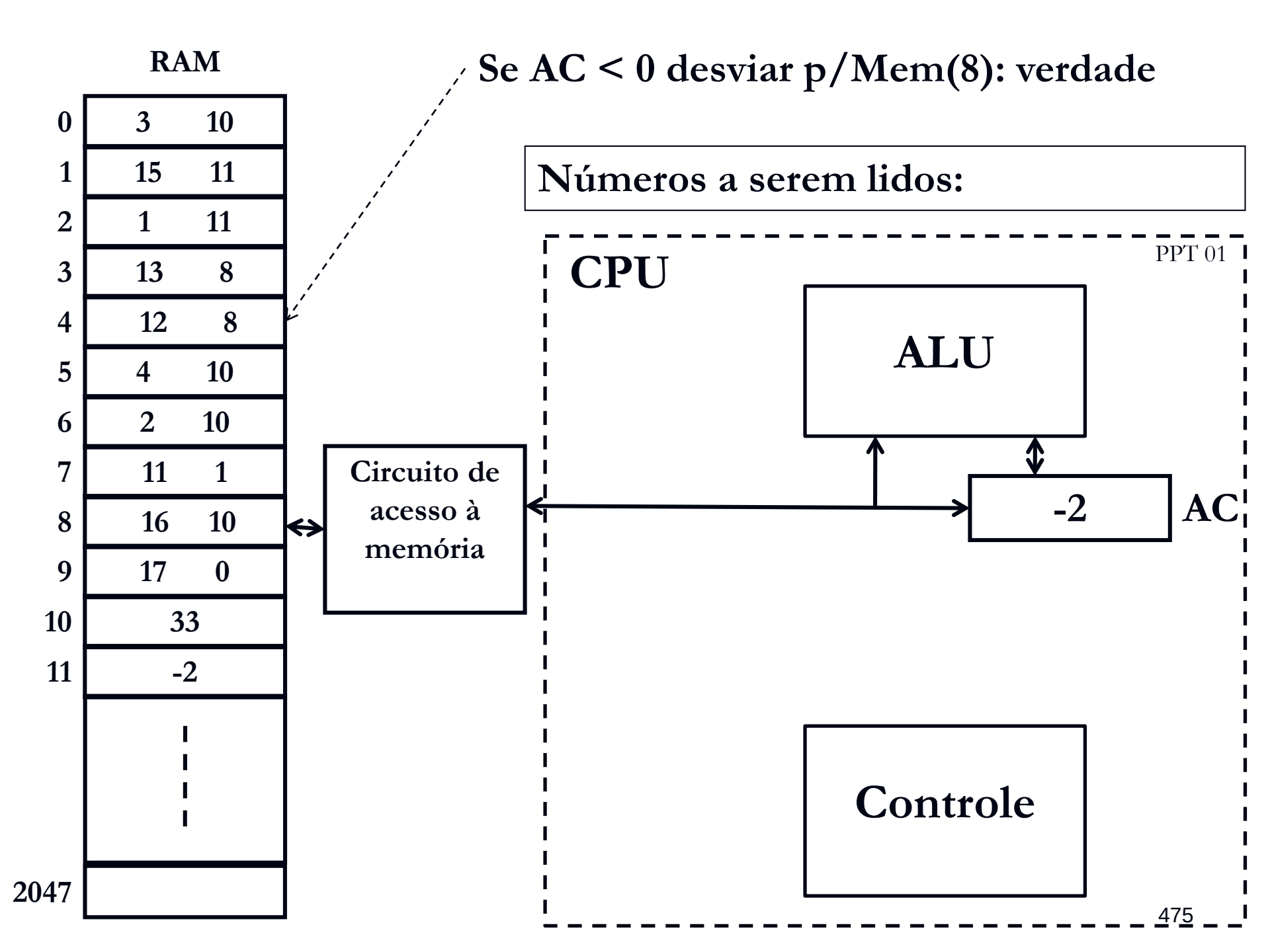

RAM
AC ← Mem(11)
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 33
11 -2
2047
Números a serem lidos:
CPU
PPT 01
ALU
Circuito de acesso à memória
33
AC
Controle
470

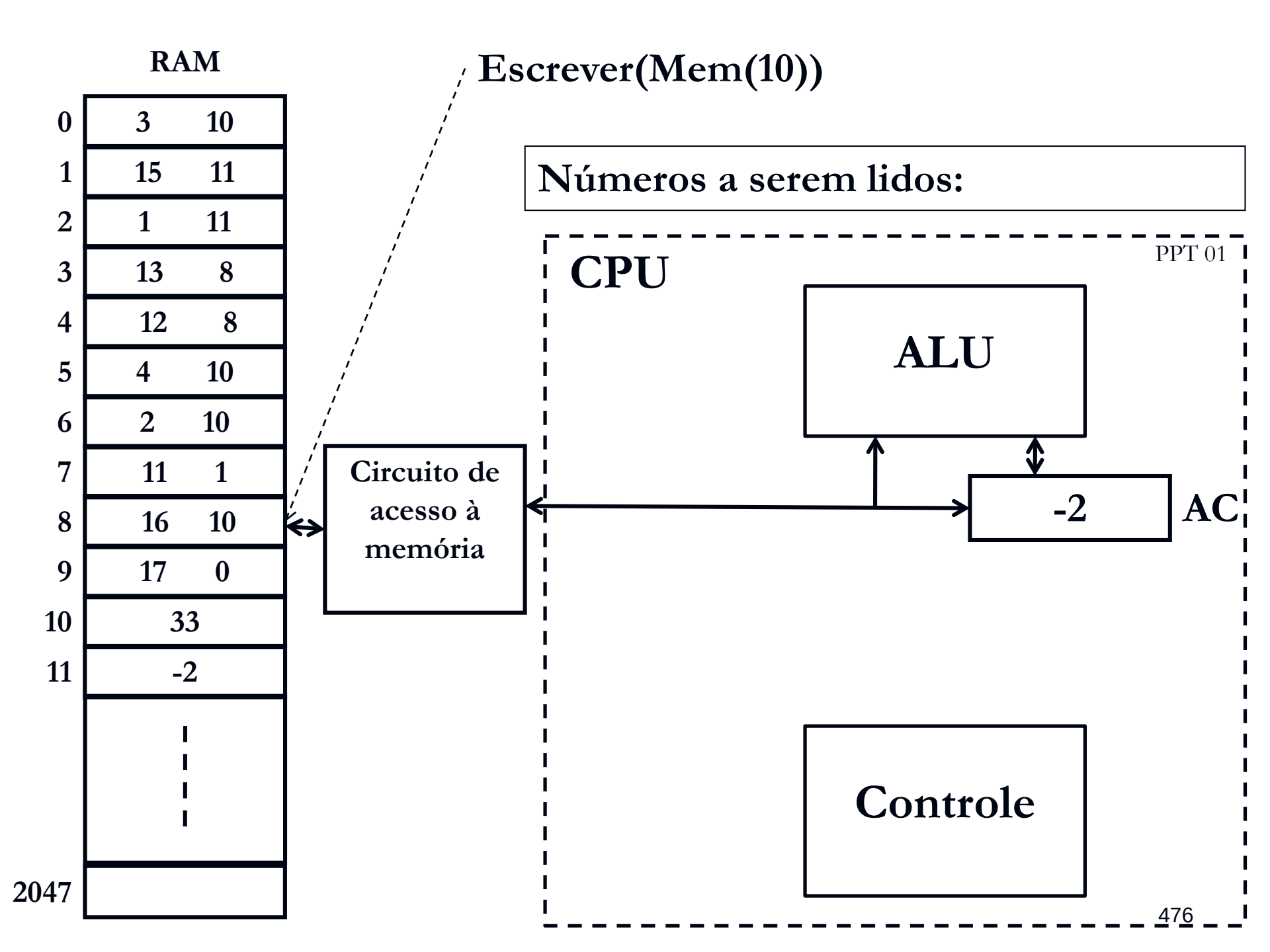

RAM
AC ← Mem(11)
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 33
11 -2
2047
Números a serem lidos:
CPU
PPT 01
ALU
Circuito de acesso à memória
-2
AC
Controle
471

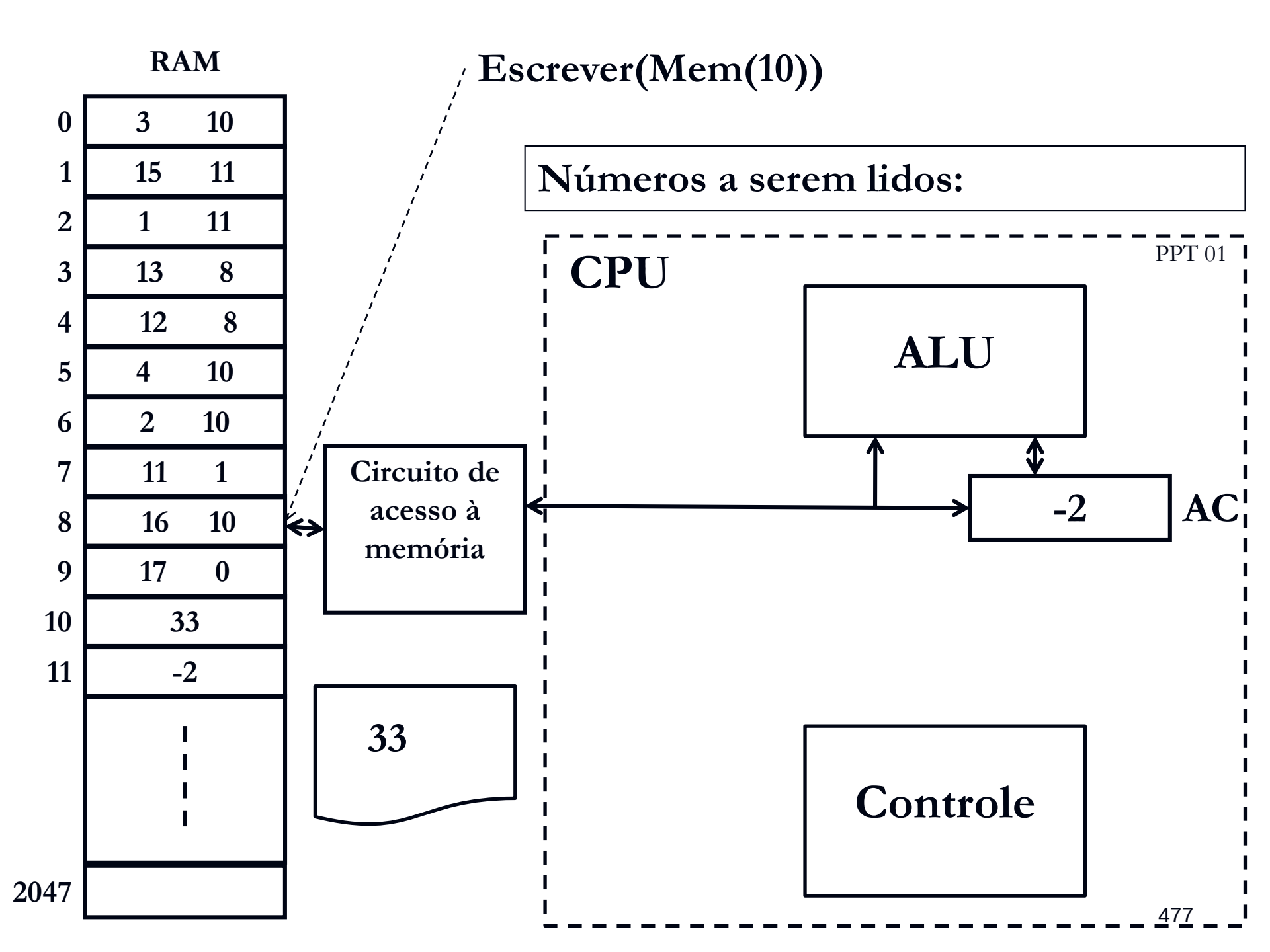

RAM
Se AC = 0 desviar p/Mem(8)
Números a serem lidos:
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 33
11 -2
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
-2
AC
Controle
472

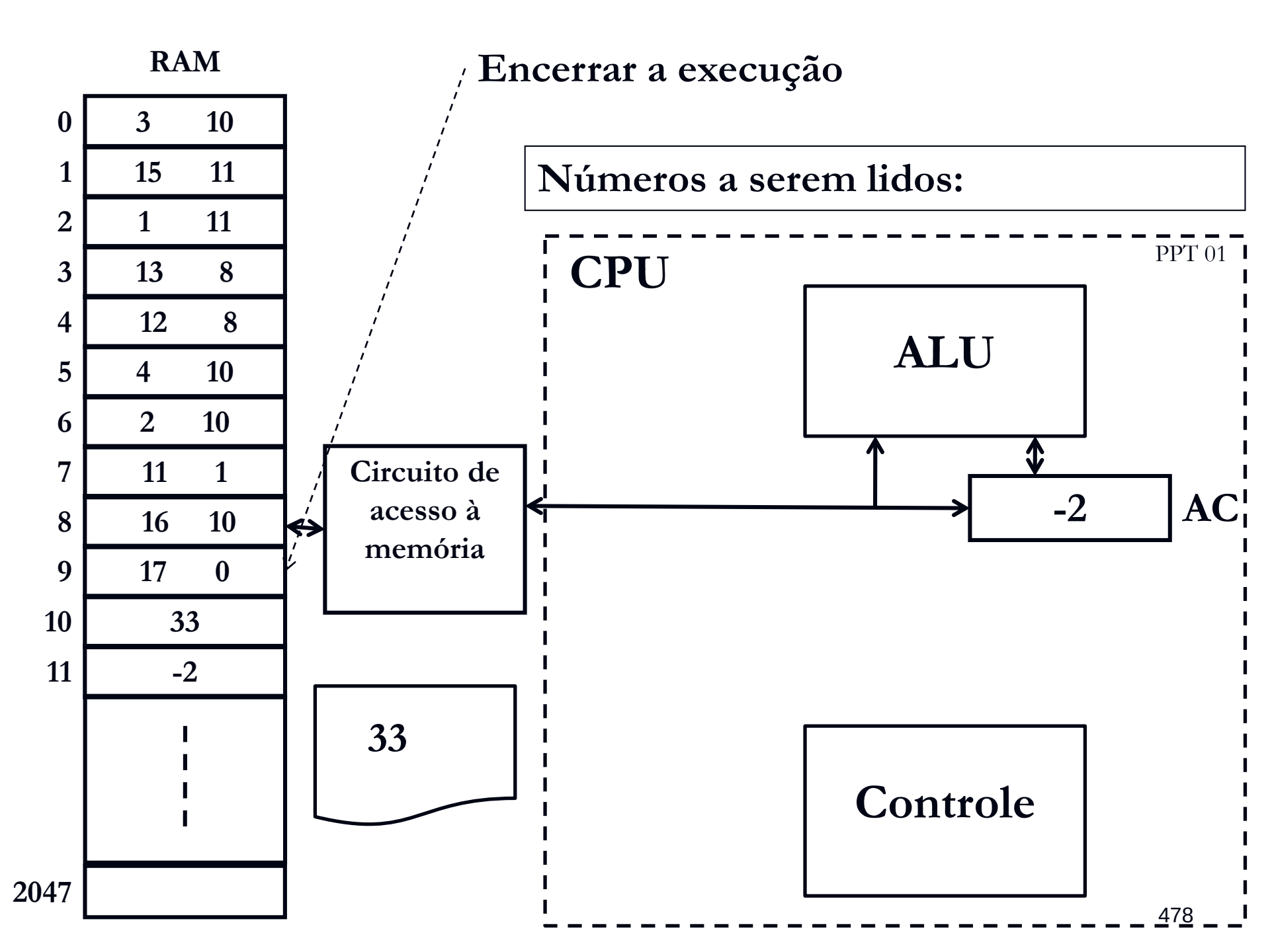

RAM
Se AC = 0 desviar p/Mem(8): falso
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 33
11 -2
2047
Números a serem lidos:
CPU
PPT 01
ALU
Circuito de acesso à memória
-2
AC
Controle
473


RAM
Se AC < 0 desviar p/Mem(8)
Números a serem lidos:
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 33
11 -2
2047
Circuito de acesso à memória
CPU
PPT 01
ALU
-2
AC
Controle
474

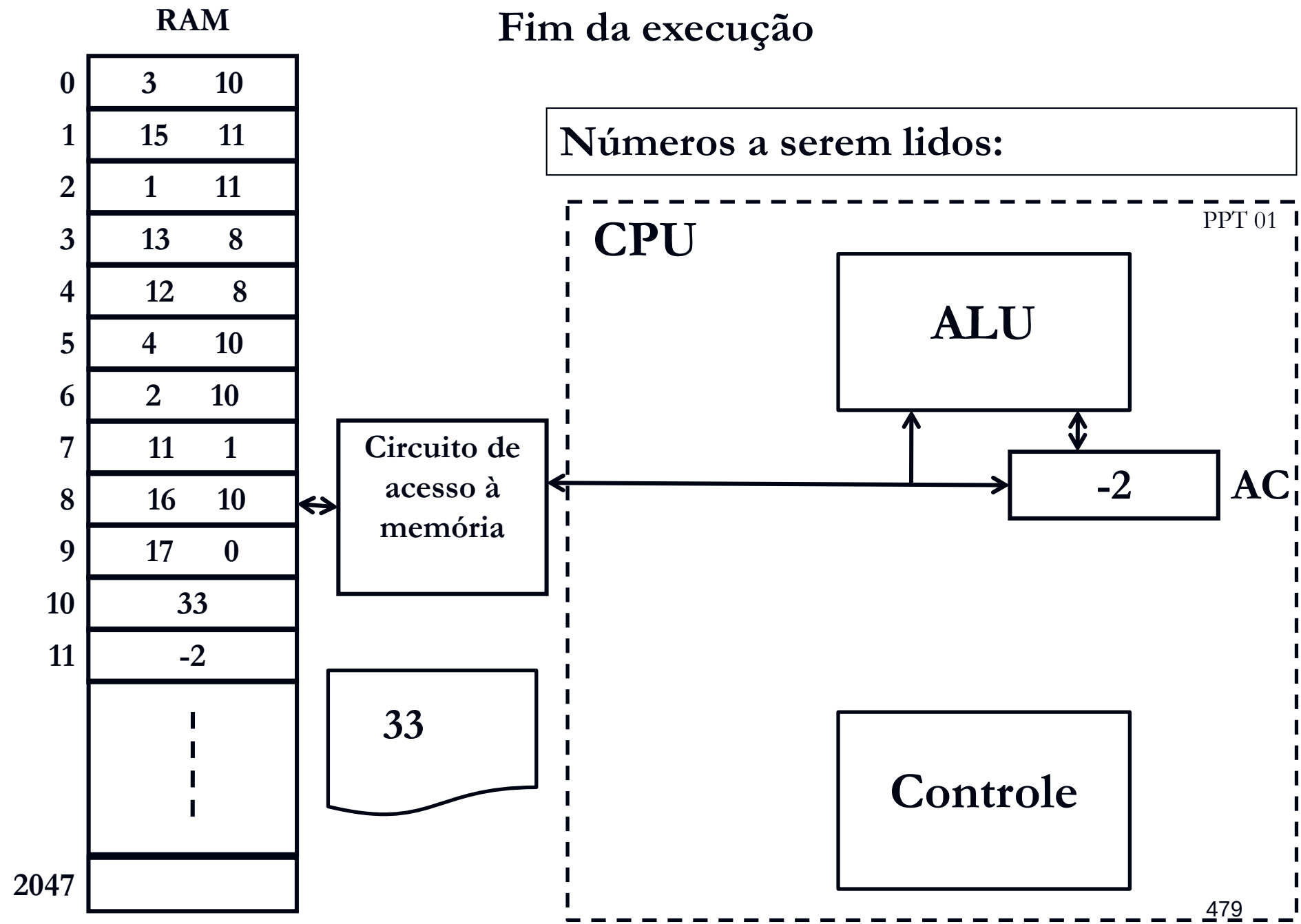

RAM
0 3 10
1 15 11
2 1 11
3 13 8
4 12 8
5 4 10
6 2 10
7 11 1
8 16 10
9 17 0
10 33
11 -2
2047
Se AC < 0 desviar p/Mem(8): verdade
Números a serem lidos:
CPU
PPT 01
ALU
Circuito de acesso à memória
-2
AC
Controle
475


RAM
0
3 10
1
15 11
2
1 11
3
13 8
4
12 8
5
4 10
6
2 10
7
11 1
8
16 10
9
17 0
10
33
11
-2
2047
Escrever(Mem(10))
Números a serem lidos:
CPU
PPT 01
ALU
Circuito de acesso à memória
-2
AC
Controle
476


RAM
Escrever(Mem(10))
Números a serem lidos:
CPU
PPT 01
ALU
-2
AC
Circuito de acesso à memória
33
Controle
0
3 10
1
15 11
2
1 11
3
13 8
4
12 8
5
4 10
6
2 10
7
11 1
8
16 10
9
17 0
10
33
11
-2
2047
477


RAM
Encerrar a execução
Números a serem lidos:
CPU
PPT 01
ALU
Circuito de acesso à memória
-2
AC
33
Controle
0
3 10
1
15 11
2
1 11
3
13 8
4
12 8
5
4 10
6
2 10
7
11 1
8
16 10
9
17 0
10
33
11
-2
2047
478

Fim da execução



3.2 – Linguagem Assembly

- Um programa em linguagem de máquina sofre de total **falta de clareza**

O programa é uma sequência de linhas numeradas

Cada linha contém uma instrução composta de um ou dois números:

- **Código de operação**
- **Endereço de operando**

É obscura a associação de cada número com a ação ou grandeza por ele representada

Programa em linguagem de máquina para somar números lidos

Livro 01

Endereço na memória	Código	Endereço
0	3	10
1	15	11
2	1	11
3	13	8
4	12	8
5	4	10
6	2	10
7	11	1
8	16	10
9	17	0

- No início dos anos 50, apareceram as chamadas **linguagens de montagem**, ou **linguagens Assembly**, ou simplesmente **Assembly's**

- **Finalidade:** expressar as instruções de máquina de **forma mais clara**

- Códigos de operação expressos por **mnemônicos**

- Informações sobre **operandos** e seus endereços expressos por **nomes** que lembrem seu significado (**endereçamento simbólico**)

- **Exemplo:** mnemônicos do computador primitivo

Nome	Mnemô-nico	Código em decimal	Ação realizada
Carregar no AC	LD	1	AC ← Mem (Ender)
Armazenar AC	ST	2	Mem (Ender) ← AC
Zerar memória	STZ	3	Mem (Ender) ← 0
Somar ao AC	ADD	4	AC ← AC + Mem (Ender)
Subtrair do AC	SUB	5	AC ← AC - Mem (Ender)
Multiplicar pelo AC	MULT	6	AC ← AC * Mem (Ender)
Dividir AC	DIV	7	AC ← AC / Mem (Ender)
And no AC	AND	8	AC ← AC and Mem (Ender)
Or no AC	OR	9	AC ← AC or Mem (Ender)
Not AC	NOT	10	AC ← not AC
Desviar incondicionalmente	JUMP	11	Desviar para Mem (Ender)
Desviar se AC < 0	JN	12	Se AC < 0, desviar para Mem (Ender)
Desviar se AC = 0	JZ	13	Se AC = 0, desviar para Mem (Ender)
Desviar se AC > 0	JP	14	Se AC > 0, desviar para Mem (Ender)
Leitura de inteiro	READ	15	Ler (Mem (Ender))
Escrita de inteiro	WRITE	16	Escrever (Mem (Ender))
Parar a execução	STOP	17	Encerrar a execução desligando a máquina

Exemplo: Programa para somar números inteiros positivos lidos

Livro 01

Endereço das instruções na memória	Instruções em linguagem de máquina		Instruções em Assembly		
	Código	Endereço	Rótulo	Código	Endereço
0	3	10		STZ	Soma
1	15	11	Leitura:	READ	Numero
2	1	11		LD	Numero
3	13	8		JZ	Escrita
4	12	8		JN	Escrita
5	4	10		ADD	Soma
6	2	10		ST	Soma
7	11	1		JUMP	Leitura
8	16	10	Escrita:	WRITE	Soma
9	17	0		STOP	
10			Soma:	CONST	0
11			Numero:	CONST	0

Montador ou Assembler: software que traduz um programa escrito em **Assembly** para linguagem de máquina

CONST: não é uma instrução executável mas sim uma **pseudo-instrução**

Livro 01

Endereço das instruções na memória	Instruções em linguagem de máquina		Instruções em Assembly		
	Código	Endereço	Rótulo	Código	Endereço
0	3	10		STZ	Soma
1	15	11	Leitura:	READ	Numero
2	1	11		LD	Numero
3	13	8		JZ	Escrita
4	12	8		JN	Escrita
5	4	10		ADD	Soma
6	2	10		ST	Soma
7	11	1		JUMP	Leitura
8	16	10	Escrita:	WRITE	Soma
9	17	0		STOP	
10			Soma:	CONST	0
11			Numero:	CONST	0

Soma: CONST 0 e Numero: CONST 0

O **Assembler** reserva as palavras **10** e **11** para as variáveis **Soma** e **Numero** e coloca ali o valor **0**

Livro 01

Endereço das instruções na memória	Instruções em linguagem de máquina		Instruções em Assembly		
	Código	Endereço	Rótulo	Código	Endereço
0	3	10		STZ	Soma
1	15	11	Leitura:	READ	Numero
2	1	11		LD	Numero
3	13	8		JZ	Escrita
4	12	8		JN	Escrita
5	4	10		ADD	Soma
6	2	10		ST	Soma
7	11	1		JUMP	Leitura
8	16	10	Escrita:	WRITE	Soma
9	17	0		STOP	
10			Soma:	CONST	0
11			Numero:	CONST	0

Soma e Numero são variáveis

Variável: nome ao qual está associado um local na memória, cujo valor pode ser alterado durante a execução

Constante: idem, só que o valor não pode ser alterado durante a execução

Endereço das instruções na memória	Instruções em linguagem de máquina		Instruções em Assembly		
	Código	Endereço	Rótulo	Código	Endereço
0	3	10		STZ	Soma
1	15	11	Leitura:	READ	Numero
2	1	11		LD	Numero
3	13	8		JZ	Escrita
4	12	8		JN	Escrita
5	4	10		ADD	Soma
6	2	10		ST	Soma
7	11	1		JUMP	Leitura
8	16	10	Escrita:	WRITE	Soma
9	17	0		STOP	
10			Soma:	CONST	0
11			Numero:	CONST	0

	STZ	Soma
Leitura:	READ	Numero
	LD	Numero
	JZ	Escrita
	JN	Escrita
	ADD	Soma
	ST	Soma
	JUMP	Leitura
Escrita:	WRITE	Soma
	STOP	
Soma:	CONST	0
Numero:	CONST	0

Livro 01

Soma:	CONST	0
Numero:	CONST	0
Inicio:	STZ	Soma
Leitura:	READ	Numero
	LD	Numero
	JZ	Escrita
	JN	Escrita
	ADD	Soma
	ST	Soma
	JUMP	Leitura
Escrita:	WRITE	Soma
	STOP	
	END	Inicio

Livro 01

- São dois programas equivalentes

- **Pseudo-instrução END:**
  - Indica ao Assembler o final do trecho a ser traduzido
  - Contém o rótulo da primeira instrução executável

	STZ	Soma
Leitura:	READ	Numero
	LD	Numero
	JZ	Escrita
	JN	Escrita
	ADD	Soma
	ST	Soma
	JUMP	Leitura
Escrita:	WRITE	Soma
	STOP	
Soma:	CONST	0
Numero:	CONST	0

Livro 01

```
Soma:     CONST    0
Numero:   CONST    0
Inicio:   STZ      Soma
Leitura:  READ     Numero
          LD       Numero
          JZ       Escrita
          JN       Escrita
          ADD      Soma
          ST       Soma
          JUMP     Leitura
Escrita:  WRITE    Soma
          STOP
          END      Inicio
```

Livro 01

- A **primeira instrução** executável não precisa ser a da **primeira linha** do programa

- As pseudo-instruções **CONST** não precisam estar no **final** do programa

- Cada computador tem sua **linguagem de máquina** e seu **Assembly** particular, por serem linguagens fortemente dependentes de sua **estrutura interna**

- Atualmente as CPU's são bem **mais poderosas**, sendo assim também as **instruções**, as **linguagens de máquina** e os **Assembly's**

- Por exemplo, as CPU's costumam ter diversos **registradores de propósitos gerais**, podendo ser acessados através de **instruções**

- Numa instrução de soma, é necessário especificar os **registradores envolvidos**

Exercícios 3.2: Livro 01

1. Escrever um programa em Assembly para ler vários pares de números inteiros positivos e, para cada par, calcular e imprimir seu MDC e seu MMC. Utilizar os mesmos mnemônicos apresentados no início da Seção 1.3.2.

2. Escrever um programa em Assembly para ler um conjunto de **n** números inteiros positivos, em que o valor **n** deve ser lido inicialmente. O programa deve também calcular e imprimir a soma desses números e a soma de seus quadrados. Deve ainda contabilizar e imprimir quantos números ímpares e quantos números pares estão nesse conjunto. Utilizar os mesmos mnemônicos apresentados no início da Seção 1.3.2.

3.3 – Linguagens tradicionais de programação

- Programas escritos em **Assembly** também carecem de **clareza**

Livro 01

- **Exemplo:**

Não é de imediato que se percebe que é a execução da atribuição

$$X = \frac{(A + B) * C}{(D - E) * (F + G)}$$

```
A:       CONST   0          LD      D
B:       CONST   0          SUB     E
C:       CONST   0          ST      X
D:       CONST   0          LD      F
E:       CONST   0          ADD     G
F:       CONST   0          MULT    X
G:       CONST   0          ST      X
X:       CONST   0          LD      A
Inicio:  READ    A          ADD     B
         READ    B          MULT    C
         READ    C          DIV     X
         READ    D          ST      X
         READ    E          END     Inicio
         READ    F
         READ    G
```

- Em 1954 surgiu **Fortran** (**For**-mula **tran**-slation): a primeira linguagem de **propósitos gerais** para a solução de problemas matemáticos e científicos

- É uma linguagem intensamente usada **até hoje**, principalmente por **pesquisadores** científicos

- Seus mantenedores a têm **atualizado** para mantê-la **viva**, mesmo com o surgimento de linguagens mais modernas

Cálculo da expressão:

$$\frac{(A+B)*C}{(D-E)*(F+G)}$$

Em Assembly:

```
A:       CONST   0       LD      D
B:       CONST   0       SUB     E
C:       CONST   0       ST      X
D:       CONST   0       LD      F
E:       CONST   0       ADD     G
F:       CONST   0       MULT    X
G:       CONST   0       ST      X
X:       CONST   0       LD      A
Inicio:  READ    A       ADD     B
         READ    B       MULT    C
         READ    C       DIV     X
         READ    D       ST      X
         READ    E       END     Inicio
         READ    F
         READ    G
```

Livro 01

Em Fortran:

```
INTEGER A, B, C, D, E, F, G, X
READ*, A, B, C, D, E, F, G
X = (A + B) * C / ((D - E) * (E + F))
STOP
END
```

- Com o passar do tempo, linguagens mais bem **estruturadas** e mais **poderosas** foram surgindo

- Eis o nome de algumas delas:

  COBOL, ALGOL, PASCAL, Modula-2, PL-1, C, C++, Java, C#, Delphy, LISP, PROLOG, etc.

- Esta disciplina utilizará a **Linguagem C** para ilustrar os princípios de **programação** aqui abordados

- Foi a linguagem utilizada para desenvolver importantes componentes do **software básico** de computadores

- É uma linguagem que permite a aplicação de técnicas para **boa estruturação** dos programas

- É uma linguagem **viva**, ou seja, ainda é muito utilizada para o desenvolvimento de programas

Exemplo: Programa para somar números inteiros positivos lidos

Em Assembly:

```
Soma:       CONST    0
Numero:     CONST    0
Inicio:     STZ      Soma
Leitura:    READ     Numero
            LD       Numero
            JZ       Escrita
            JN       Escrita
            ADD      Soma
            ST       Soma
            JUMP     Leitura
Escrita:    WRITE    Soma
            STOP
            END      Inicio
```

Livro 01

Em C:

```
#include <stdio.h>
main () {
        int numero, soma;
        soma = 0;
        scanf ("%d", &numero);
        while (numero > 0) {
                soma = soma + numero;
                scanf ("%d", &numero);
        }
        printf ("%d", soma);
}
```

PPT 01

- A programação em C não requer conhecimento da estrutura interna do computador

- O programador não precisa saber quantos registradores de propósitos gerais possui a CPU

- No programa ao lado, não há qualquer referência explícita ou implícita ao registrador AC ou a qualquer outro de mesma finalidade

- Pessoas cada vez menos familiarizadas com Arquitetura de Computadores podem programá-los

```
#include <stdio.h>
main () {
        int numero, soma;
        soma = 0;
        scanf ("%d", &numero);
        while (numero > 0) {
                soma = soma + numero;
                scanf ("%d", &numero);
        }
        printf ("%d", soma);
}
```

PPT 01

Endereço na memória	Código	Endereço
0	3	10
1	15	11
2	1	11
3	13	8
4	12	8
5	4	10
6	2	10
7	11	1
8	16	10
9	17	0

Linguagem de máquina

Livro 01

```
Soma:     CONST   0
Numero:   CONST   0
Inicio:   STZ     Soma
Leitura:  READ    Numero
          LD      Numero
          JZ      Escrita
          JN      Escrita
          ADD     Soma
          ST      Soma
          JUMP    Leitura
Escrita:  WRITE   Soma
          STOP
          END     Inicio
```

Assembly

- **Cada instrução em Assembly corresponde a uma instrução em linguagem de máquina**
- **É relativamente simples traduzir de Assembly para linguagem de máquina**

C

PPT 01

```
#include <stdio.h>
main () {
        int numero, soma;
        soma = 0;
        scanf ("%d", &numero);
        while (numero > 0) {
                soma = soma + numero;
                scanf ("%d", &numero);
        }
        printf ("%d", soma);
}
```

Endereço na memória	Código	Endereço
0	3	10
1	15	11
2	1	11
3	13	8
4	12	8
5	4	10
6	2	10
7	11	1
8	16	10
9	17	0

Linguagem de máquina

Livro 01

```
Soma:     CONST   0
Numero:   CONST   0
Inicio:   STZ     Soma
Leitura:  READ    Numero
          LD      Numero
          JZ      Escrita
          JN      Escrita
          ADD     Soma
          ST      Soma
          JUMP    Leitura
Escrita:  WRITE   Soma
          STOP
          END     Inicio
```

Assembly

- **A estrutura de um programa em C é totalmente diferente das outras duas**
- **A tradução de C para Assembly ou linguagem de máquina é bem mais complexa**

C

PPT 01

```
#include <stdio.h>
main 0 {
        int numero, soma;
        soma = 0;
        scanf ("%d", &numero);
        while (numero > 0) {
                soma = soma + numero;
                scanf ("%d", &numero);
        }
        printf ("%d", soma);
}
```

Endereço na memória	Código	Endereço
0	3	10
1	15	11
2	1	11
3	13	8
4	12	8
5	4	10
6	2	10
7	11	1
8	16	10
9	17	0

Linguagem de máquina

Livro 01

```
Soma:     CONST   0
Numero:   CONST   0
Inicio:   STZ     Soma
Leitura:  READ    Numero
          LD      Numero
          JZ      Escrita
          JN      Escrita
          ADD     Soma
          ST      Soma
          JUMP    Leitura
Escrita:  WRITE   Soma
          STOP
          END     Inicio
```

Assembly

- **Compilador: software que traduz de uma linguagem de programação para Assembly ou linguagem de máquina**

C

PPT 01

```
#include <stdio.h>
main 0 {
        int numero, soma;
        soma = 0;
        scanf ("%d", &numero);
        while (numero > 0) {
                soma = soma + numero;
                scanf ("%d", &numero);
        }
        printf ("%d", soma);
}
```

- **Em muitos ambientes de programação:**

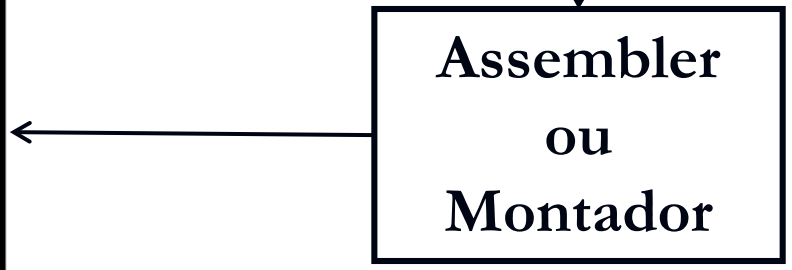
Programa em linguagem de programação

```
#include <stdio.h>
main () {
        int numero, soma;
        soma = 0;
        scanf ("%d", &numero);
        while (numero > 0) {
                soma = soma + numero;
                scanf ("%d", &numero);
        }
        printf ("%d", soma);
}
```



Programa em Assembly

```
Soma:     CONST   0
Numero:   CONST   0
Inicio:   STZ     Soma
Leitura:  READ    Numero
          LD      Numero
          JZ      Escrita
          JN      Escrita
          ADD     Soma
          ST      Soma
          JUMP    Leitura
Escrita:  WRITE   Soma
          STOP
          END     Inicio
```



Programa em linguagem de máquina

Endereço na memória	Código	Endereço
0	3	10
1	15	11
2	1	11
3	13	8
4	12	8
5	4	10
6	2	10
7	11	1
8	16	10
9	17	0

- A seguir, os processos de **compilação** e de **montagem** serão ilustrados, usando-se um programa em **C** para o cálculo do **fatorial** de um número lido

- Antes porém, será comentado sobre a **divisão de um programa em módulos**

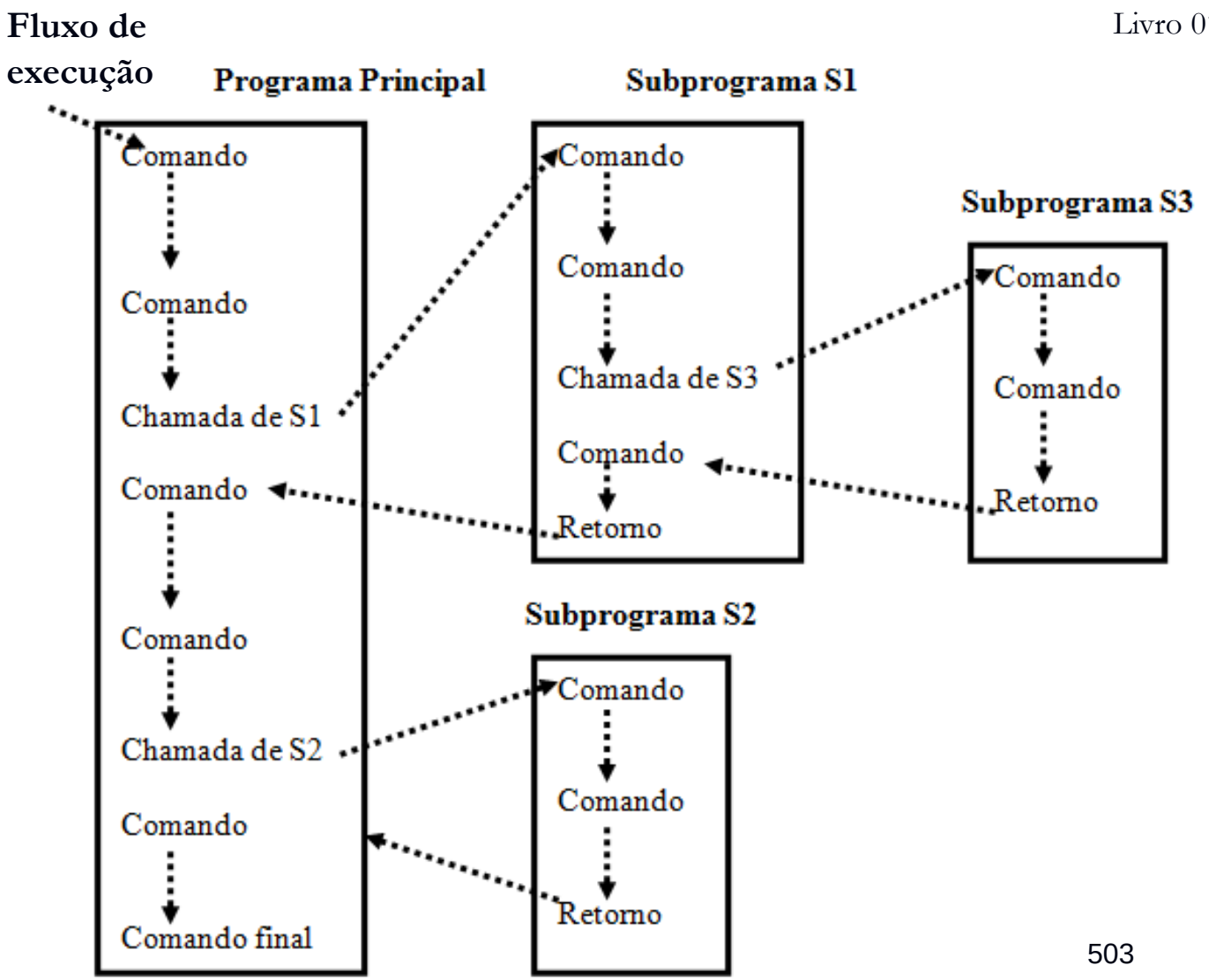
Divisão de um programa em módulos e sua execução:

O programa principal é obrigatório

Por ele começa a execução

Os outros módulos são auxiliares do principal e não são obrigatórios

Eles são chamados de subprogramas



Livro 01

Exemplo: Compilação do programa do fatorial

```
#include <stdio.h>
int main ( )
{
        int n, fat, i;
        scanf (“%d”, &n);
        fat = 1;
        i = 2;
        while (i <= n) {
                fat = fat * i;
                i = i + 1;
        }
        printf (“%d”, fat);
}
```

Livro 01

C1:	CONST	1
C2:	CONST	2

```
#include <stdio.h>
int main ( )
{
        int n, fat, i;
        scanf (“%d”, &n);
        fat = 1;
        i = 2;
        while (i <= n) {
                fat = fat * i;
                i = i + 1;
        }
        printf (“%d”, fat);
}
```

Livro 01

Primeiramente, reserva de espaço para as constantes 1 e 2

C1:	CONST	1
C2:	CONST	2
n:	CONST	0
fat:	CONST	0
i:	CONST	0

```
#include <stdio.h>
void main ( )
{
        int n, fat, i;
        scanf (“%d”, &n);
        fat = 1;
        i = 2;
        while (i <= n) {
                fat = fat * i;
                i = i + 1;
        }
        printf (“%d”, fat);
}
```

Livro 01

Em seguida, reserva de espaço para as variáveis n, i, fat

```
#include <stdio.h>
void main ( )
{
        int n, fat, i;
        scanf (“%d”, &n);
        fat = 1;
        i = 2;
        while (i <= n) {
                fat = fat * i;
                i = i + 1;
        }
        printf (“%d”, fat);
}
```

Livro 01

C1:	CONST	1
C2:	CONST	2
n:	CONST	0
fat:	CONST	0
i:	CONST	0

Agora a tradução dos comandos

Rótulo da 1ª instrução executável:

inic

```
#include <stdio.h>
void main ( )
{
        int n, fat, i;
        scanf (“%d”, &n);
        fat = 1;
        i = 2;
        while (i <= n) {
                fat = fat * i;
                i = i + 1;
        }
        printf (“%d”, fat);
}
```

```
C1:     CONST   1
C2:     CONST   2
n:      CONST   0
fat:    CONST   0
i:      CONST   0
inic:   READ    n
```

Livro 01

Na realidade, a tradução de scanf é algo mais complexo:

É uma chamada de subprograma

Rótulo da 1ª instrução executável:

inic

```
#include <stdio.h>
void main ( )
{
        int n, fat, i;
        scanf (“%d”, &n);
        fat = 1;
        i = 2;
        while (i <= n) {
                fat = fat * i;
                i = i + 1;
        }
        printf (“%d”, fat);
}
```

Livro 01

```
C1:     CONST   1
C2:     CONST   2
n:      CONST   0
fat:    CONST   0
i:      CONST   0
inic:   READ    n
        LD      C1
        ST      fat
        LD      C2
        ST      i
```

Rótulo da 1ª instrução executável:

inic

```
#include <stdio.h>
void main ( )
{
        int n, fat, i;
        scanf (“%d”, &n);
        fat = 1;
        i = 2;
        while (i <= n) {
                fat = fat * i;
                i = i + 1;
        }
        printf (“%d”, fat);
}
```

Livro 01

A instrução “loop” deveria ser LD i, mas antes dela e da instrução JUMP, o registrador AC já estará com o valor de “i”

```
C1:     CONST   1
C2:     CONST   2
n:      CONST   0
fat:    CONST   0
i:      CONST   0
inic:   READ    n
        LD      C1
        ST      fat
        LD      C2
        ST      i
loop:   SUB     n
        JP      escrever

        JUMP    loop
```

“escrever” é o rótulo da instrução logo após JUMP

Rótulo da 1ª instrução executável:

inic

```
#include <stdio.h>
void main ( )
{
        int n, fat, i;
        scanf (“%d”, &n);
        fat = 1;
        i = 2;
        while (i <= n) {
                fat = fat * i;
                i = i + 1;
        }
        printf (“%d”, fat);
}
```

Livro 01

```
C1:     CONST   1
C2:     CONST   2
n:      CONST   0
fat:    CONST   0
i:      CONST   0
inic:   READ    n
        LD      C1
        ST      fat
        LD      C2
        ST      i
loop:   SUB     n
        JP      escrever
        LD      fat
        MULT    i
        ST      fat
        LD      i
        ADD     C1
        ST      i
        JUMP    loop
```

Rótulo da 1ª instrução executável:

inic

```
#include <stdio.h>
void main ( )
{
        int n, fat, i;
        scanf (“%d”, &n);
        fat = 1;
        i = 2;
        while (i <= n) {
                fat = fat * i;
                i = i + 1;
        }
        printf (“%d”, fat);
}
```

Livro 01

Na realidade, a tradução de printf é algo mais complexo:

É uma chamada de subprograma

```
C1:     CONST   1
C2:     CONST   2
n:      CONST   0
fat:    CONST   0
i:      CONST   0
inic:   READ    n
        LD      C1
        ST      fat
        LD      C2
        ST      i
loop:   SUB     n
        JP      escrever
        LD      fat
        MULT    i
        ST      fat
        LD      i
        ADD     C1
        ST      i
        JUMP    loop
escrever:WRITE  fat
```

Rótulo da 1ª instrução executável:

inic

```
#include <stdio.h>
void main ( )
{
        int n, fat, i;
        scanf (“%d”, &n);
        fat = 1;
        i = 2;
        while (i <= n) {
                fat = fat * i;
                i = i + 1;
        }
        printf (“%d”, fat);
}
```

Livro 01

```
C1:     CONST   1
C2:     CONST   2
n:      CONST   0
fat:    CONST   0
i:      CONST   0
inic:   READ    n
        LD      C1
        ST      fat
        LD      C2
        ST      i
loop:   SUB     n
        JP      escrever
        LD      fat
        MULT    i
        ST      fat
        LD      i
        ADD     C1
        ST      i
        JUMP    loop
escrever:WRITE  fat
        STOP
        END     inic
```

```
#include <stdio.h>
void main ( )
{
        int n, fat, i;
        scanf (“%d”, &n);
        fat = 1;
        i = 2;
        while (i <= n) {
                fat = fat * i;
                i = i + 1;
        }
        printf (“%d”, fat);
}
```

Livro 01

Final da compilação

Agora vem a montagem

PPT 01

```
C1:       CONST     1
C2:       CONST     2
n:        CONST     0
fat:      CONST     0
i:        CONST     0
inic:     READ      n
          LD        C1
          ST        fat
          LD        C2
          ST        i
loop:     SUB       n
          JP        escrever
          LD        fat
          MULT      i
          ST        fat
          LD        i
          ADD       C1
          ST        i
          JUMP      loop
escrever: WRITE     fat
          STOP
          END       inic
```

```
C1:     CONST   1
C2:     CONST   2
n:      CONST   0
fat:    CONST   0
i:      CONST   0
inic:   READ    n
        LD      C1
        ST      fat
        LD      C2
        ST      i
loop:   SUB     n
        JP      escrever
        LD      fat
        MULT    i
        ST      fat
        LD      i
        ADD     C1
        ST      i
        JUMP    loop
escrever:WRITE  fat
        STOP
        END     inic
```

rótulo	endereço

O Assembler monta uma tabela de rótulos para ajudar a preencher o programa em linguagem de máquina

endereço	codop	ender
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		515

```
C1:       CONST   1
C2:       CONST   2
n:        CONST   0
fat:      CONST   0
i:        CONST   0
inic:     READ    n
          LD      C1
          ST      fat
          LD      C2
          ST      i
loop:     SUB     n
          JP      escrever
          LD      fat
          MULT    i
          ST      fat
          LD      i
          ADD     C1
          ST      i
          JUMP    loop
escrever: WRITE   fat
          STOP
          END     inic
```

rótulo	endereço
C1	0
C2	1
n	2
fat	3
i	4

endereço	codop	ender
0	1	
1	2	
2	0	
3	0	
4	0	
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		516

```
C1:      CONST   1
C2:      CONST   2
n:       CONST   0
fat:     CONST   0
i:       CONST   0
inic:    READ    n
         LD      C1
         ST      fat
         LD      C2
         ST      i
loop:    SUB     n
         JP      escrever
         LD      fat
         MULT    i
         ST      fat
         LD      i
         ADD     C1
         ST      i
         JUMP    loop
escrever:WRITE   fat
         STOP
         END     inic
```

rótulo	endereço
C1	0
C2	1
n	2
fat	3
i	4
inic	5

endereço	codop	ender
0	1	
1	2	
2	0	
3	0	
4	0	
5	15	2
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		517

```
C1:       CONST   1
C2:       CONST   2
n:        CONST   0
fat:      CONST   0
i:        CONST   0
inic:     READ    n
          LD      C1
          ST      fat
          LD      C2
          ST      i
loop:     SUB     n
          JP      escrever
          LD      fat
          MULT    i
          ST      fat
          LD      i
          ADD     C1
          ST      i
          JUMP    loop
escrever: WRITE   fat
          STOP
          END     inic
```

rótulo	endereço
C1	0
C2	1
n	2
fat	3
i	4
inic	5

endereço	codop	ender
0	1	
1	2	
2	0	
3	0	
4	0	
5	15	2
6	1	0
7	2	3
8	1	1
9	2	4
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		518

```
C1:       CONST   1
C2:       CONST   2
n:        CONST   0
fat:      CONST   0
i:        CONST   0
inic:     READ    n
          LD      C1
          ST      fat
          LD      C2
          ST      i
loop:     SUB     n
          JP      escrever
          LD      fat
          MULT    i
          ST      fat
          LD      i
          ADD     C1
          ST      i
          JUMP    loop
escrever: WRITE   fat
          STOP
          END     inic
```

rótulo	endereço
C1	0
C2	1
n	2
fat	3
i	4
inic	5
loop	10

endereço	codop	ender
0	1	
1	2	
2	0	
3	0	
4	0	
5	15	2
6	1	0
7	2	3
8	1	1
9	2	4
10	5	2
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		519

```
C1:       CONST   1
C2:       CONST   2
n:        CONST   0
fat:      CONST   0
i:        CONST   0
inic:     READ    n
          LD      C1
          ST      fat
          LD      C2
          ST      i
loop:     SUB     n
          JP      escrever
          LD      fat
          MULT    i
          ST      fat
          LD      i
          ADD     C1
          ST      i
          JUMP    loop
escrever: WRITE   fat
          STOP
          END     inic
```

rótulo	endereço
C1	0
C2	1
n	2
fat	3
i	4
inic	5
loop	10
escrever	???

endereço	codop	ender
0	1	
1	2	
2	0	
3	0	
4	0	
5	15	2
6	1	0
7	2	3
8	1	1
9	2	4
10	5	2
11	14	???
12		
13		
14		
15		
16		
17		
18		
19		
20		520

```
C1:       CONST   1
C2:       CONST   2
n:        CONST   0
fat:      CONST   0
i:        CONST   0
inic:     READ    n
          LD      C1
          ST      fat
          LD      C2
          ST      i
loop:     SUB     n
          JP      escrever
          LD      fat
          MULT    i
          ST      fat
          LD      i
          ADD     C1
          ST      i
          JUMP    loop
escrever: WRITE   fat
          STOP
          END     inic
```

rótulo	endereço
C1	0
C2	1
n	2
fat	3
i	4
inic	5
loop	10
escrever	???

endereço	codop	ender
0	1	
1	2	
2	0	
3	0	
4	0	
5	15	2
6	1	0
7	2	3
8	1	1
9	2	4
10	5	2
11	14	???
12	1	3
13	6	4
14	2	3
15	1	4
16	4	0
17	2	4
18	11	10
19		
20		521

```
C1:     CONST   1
C2:     CONST   2
n:      CONST   0
fat:    CONST   0
i:      CONST   0
inic:   READ    n
        LD      C1
        ST      fat
        LD      C2
        ST      i
loop:   SUB     n
        JP      escrever
        LD      fat
        MULT    i
        ST      fat
        LD      i
        ADD     C1
        ST      i
        JUMP    loop
escrever:WRITE  fat
        STOP
        END     inic
```

rótulo	endereço
C1	0
C2	1
n	2
fat	3
i	4
inic	5
loop	10
escrever	19

endereço	codop	ender
0	1	
1	2	
2	0	
3	0	
4	0	
5	15	2
6	1	0
7	2	3
8	1	1
9	2	4
10	5	2
11	14	19
12	1	3
13	6	4
14	2	3
15	1	4
16	4	0
17	2	4
18	11	10
19	16	3
20		522

```
C1:      CONST   1
C2:      CONST   2
n:       CONST   0
fat:     CONST   0
i:       CONST   0
inic:    READ    n
         LD      C1
         ST      fat
         LD      C2
         ST      i
loop:    SUB     n
         JP      escrever
         LD      fat
         MULT    i
         ST      fat
         LD      i
         ADD     C1
         ST      i
         JUMP    loop
escrever:WRITE   fat
         STOP
         END     inic
```

rótulo	endereço
C1	0
C2	1
n	2
fat	3
i	4
inic	5
loop	10
escrever	19

endereço	codop	ender
0	1	
1	2	
2	0	
3	0	
4	0	
5	15	2
6	1	0
7	2	3
8	1	1
9	2	4
10	5	2
11	14	19
12	1	3
13	6	4
14	2	3
15	1	4
16	4	0
17	2	4
18	11	10
19	16	3
20	17	0

```
C1:      CONST   1
C2:      CONST   2
n:       CONST   0
fat:     CONST   0
i:       CONST   0
inic:    READ    n
         LD      C1
         ST      fat
         LD      C2
         ST      i
loop:    SUB     n
         JP      escrever
         LD      fat
         MULT    i
         ST      fat
         LD      i
         ADD     C1
         ST      i
         JUMP    loop
escrever:WRITE   fat
         STOP
         END     inic
```

rótulo	endereço
C1	0
C2	1
n	2
fat	3
i	4
inic	5
loop	10
escrever	19

Endereço inicial da execução: 5

Essa informação deve acompanhar o programa em linguagem de máquina

endereço	codop	ender
0	1	
1	2	
2	0	
3	0	
4	0	
5	15	2
6	1	0
7	2	3
8	1	1
9	2	4
10	5	2
11	14	19
12	1	3
13	6	4
14	2	3
15	1	4
16	4	0
17	2	4
18	11	10
19	16	3
20	17	0

- O programa em linguagem de programação é denominado **programa-fonte**
- O programa gerado pelo Assembler é denominado **programa-objeto**
- O programa-objeto foi montado a partir do **endereço zero** da RAM
- Esse programa é guardado num **arquivo** (extensão **obj**, ou **o**, no Code Blocks)

endereço	codop	ender
0	1	
1	2	
2	0	
3	0	
4	0	
5	15	2
6	1	0
7	2	3
8	1	1
9	2	4
10	5	2
11	14	19
12	1	3
13	6	4
14	2	3
15	1	4
16	4	0
17	2	4
18	11	10
19	16	3
20	17	0

Endereço inicial da execução: 5

- O local para execução é estabelecido pelo **sistema operacional** do computador
- Esse local depende da **disponibilidade** da RAM
- E se o local **não for o endereço zero** (por exemplo, endereço 3000)?

endereço	codop	ender
0	1	
1	2	
2	0	
3	0	
4	0	
5	15	2
6	1	0
7	2	3
8	1	1
9	2	4
10	5	2
11	14	19
12	1	3
13	6	4
14	2	3
15	1	4
16	4	0
17	2	4
18	11	10
19	16	3
20	17	0

PPT 01

Endereço inicial da execução: 5

- Os locais para C1, C2, n, fat e i não são mais **0**, **1**, **2**, **3** e **4**, mas sim **3000**, **3001**, **3002**, **3003** e **3004**

- Os rótulos **inic**, **loop** e **escrever** mudarão para os endereços **3005**, **3010** e **3019**

- Então todos os campos **ender** das instruções estarão com um erro **(deslocamento de 3000 posições)**

- Isso tem de ser **corrigido** antes da execução

endereço	codop	ender
3000	1	
3001	2	
3002	0	
3003	0	
3004	0	
3005	15	2
3006	1	0
3007	2	3
3008	1	1
3009	2	4
3010	5	2
3011	14	19
3012	1	3
3013	6	4
3014	2	3
3015	1	4
3016	4	0
3017	2	4
3018	11	10
3019	16	3
3020	17	0

PPT 01

Endereço inicial da execução: 3005

- Quem corrige isso é o software denominado **carregador (loader)**

- A correção é feita quando o programa for **carregado** para ser **executado**

endereço	codop	ender
3000	1	
3001	2	
3002	0	
3003	0	
3004	0	
3005	15	2
3006	1	0
3007	2	3
3008	1	1
3009	2	4
3010	5	2
3011	14	19
3012	1	3
3013	6	4
3014	2	3
3015	1	4
3016	4	0
3017	2	4
3018	11	10
3019	16	3
3020	17	0

Endereço inicial da execução: 3005

- Antes de corrigir os endereços do programa-objeto, é necessário juntar a ele todos os **subprogramas auxiliares** pertencentes à **biblioteca** da linguagem

- **Exemplos:** funções para **entrada e saída** (scanf, printf, etc.), funções **matemáticas** (sqr, pow, sqrt, log, sin, cos, etc.)

- Esse trabalho de juntar o programa-objeto com tais subprogramas é feito por um software denominado **editor de ligações (linkage-editor)**

- O produto do editor de ligações é um arquivo denominado **programa-executável** (extensão **exe**)

- Quando o **executável** for trazido à RAM para ser executado, só então o **carregador** fará a **correção dos endereços**

Exercício 3.3:

O programa em C a seguir calcula a soma de uma PA, onde o primeiro termo, a razão e o número de termos são lidos. Escrever um programa em Assembly e um programa em Linguagem de Máquina para este programa. Utilizar os mesmos mnemônicos e códigos de operação apresentados na Seção 1.3.1 e 1.3.2. Estudar no livro-texto e utilizar devidamente a pseudo-instrução **BLOCK** para manipular cadeias de caracteres.

Livro 01

```
void main () {
      int a1, r, n, soma, aq, i;
      scanf ("%d", &a1); scanf ("%d", &r); scanf ("%d", &n);
      soma = 0; aq = a1; i = 1;
      while (i <= n) {
            soma = soma + aq; aq = aq + r; i = i + 1;
      }
      printf ("Soma PA = %d", soma);
}
```

3.4 – Linguagens aplicativas

- Os **propósitos** das linguagens tradicionais de programação são **gerais e inúmeros**

- Com elas foram criados programas muito trabalhosos para criar linguagens de **propósitos específicos**

- Nesses programas, a **entrada de dados** consiste de **comandos** de uma nova linguagem, para a execução de determinadas tarefas atuando sobre determinados parâmetros

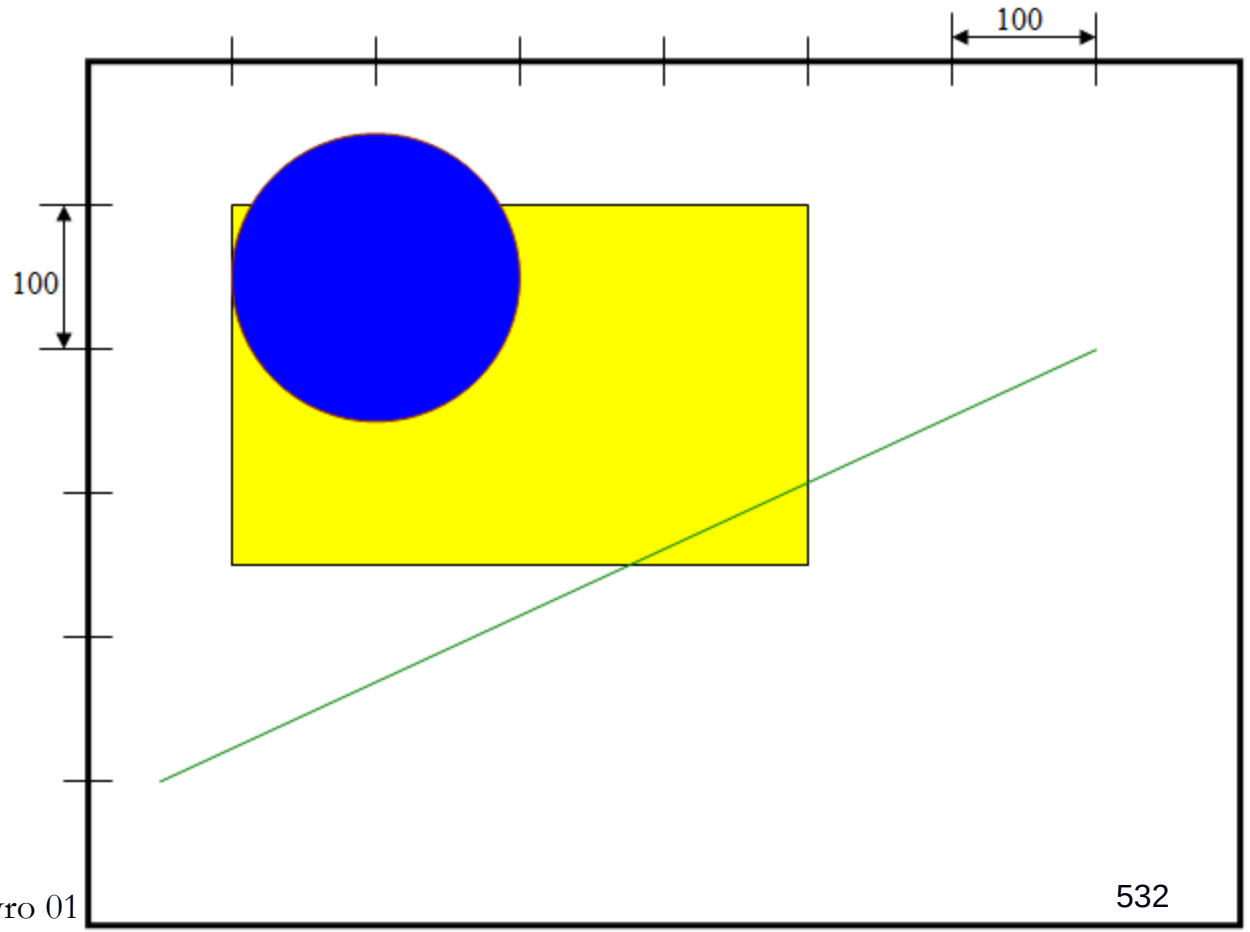
- Por exemplo, uma linguagem para desenho geométrico, pode ter comandos tais como:

Retângulo: 400 X 250 – canto superior esquerdo [100, 100] – cor [PRETO, AMARELO]

Circulo: raio 100 – centro [200, 150] – cor [MARROM, AZUL]

Reta: [50, 500] [700, 200] – cor VERDE

- Resultado ---------->



É imensa a quantidade de linguagens para aplicações específicas; exemplos de propósitos para essas linguagens:

Simulação de fenômenos de natureza diversa	Manipulação de fórmulas matemáticas
Manipulação de banco de dados	Planilhas de cálculo
Edição e formatação de textos	Controle dos mais variados tipos de processos industriais
Edição de partituras musicais	Realização de experimentos, etc.

Computadores, Algoritmos e Programas: Manual Introdutório de Ensino

Capítulo IV

Elementos Básicos de Algoritmos e Programas

Capítulo IV – Elementos Básicos de Algoritmos e Programas

4.1 – A necessidade de métodos e algoritmos

4.2 – Algoritmos executados por seres humanos

4.3 – Algoritmos para computadores

4.4 – Programas a partir de algoritmos

4.5 – Produção de um software

4.1 – A necessidade de métodos e algoritmos

Objetivo principal de um computador:

- Realizar tarefas que envolvam **intenso processamento de informações**
- Livrando os seres humanos de **esforços repetitivos, tediosos e sujeitos a erros**
- Possibilitando-lhes também a obtenção de **resultados confiáveis em tempo hábil**

- Para cada **tarefa** a ser realizada, o computador deve estar devidamente **programado**

- Caso não haja **software** pronto que realize a tarefa requerida, alguém, que recebe a denominação de **programador**, deve elaborar um **programa**

- **Programa:** sequência de **instruções** que, ao serem executadas por um computador, **realizam** uma determinada tarefa

- O **programa** deve estar escrito numa **linguagem de programação** (pode até ser **Assembly**)

Primeiros passos para a elaboração de um programa:

- **Determinação da tarefa** a ser automatizada, com detalhes minuciosos
- **Escolha do método** que irá fundamentar as ações a serem realizadas pelas instruções do programa
- **Elaboração de um algoritmo**, que é uma sequência de passos para a aplicação do método escolhido

Algoritmo: sequência finita e ordenada de **passos** (comandos executáveis e não ambíguos), que levam à aplicação de um **método** para a execução de uma **tarefa** ou resolução de um problema

Primeiros passos para a elaboração de um programa:

- **Determinação da tarefa** a ser automatizada, com detalhes minuciosos
- **Escolha do método** que irá fundamentar as ações a serem realizadas pelas instruções do programa
- **Elaboração de um algoritmo**, que é uma sequência de passos para a aplicação do método escolhido
- **Elaboração do programa**, que é a tradução do algoritmo para a linguagem de programação escolhida

4.2 – Algoritmos executados por seres humanos

- Além de computadores, **outras entidades** podem executar algoritmos

- Muitas atividades rotineiras dos **seres humanos** podem ser descritas por algoritmos. **Exemplos:**

  - Preparo de receita culinária
  - Troca de pneu furado
  - Troca de lâmpada queimada
  - Atividades de uma pessoa, desde o momento em que ela acorda, até sua chegada ao local de trabalho

Exemplo: algoritmo para troca de pneu furado


```
TrocarPneuFurado {
        Se (o estepe estiver murcho)
                Chamar o borracheiro;
        Senão {
                Pegar o macaco e a chave de boca no porta-malas;
                Afrouxar com a chave de boca todos os parafusos da roda;
                Levantar o carro com o macaco;
                Retirar todos os parafusos da roda;
                Retirar o pneu furado colocando-o no porta-malas;
                Posicionar o estepe no local em que será acoplado;
                Recolocar e apertar ligeiramente os parafusos;
                Baixar o carro;
                Apertar fortemente os parafusos;
                Guardar as ferramentas usadas;
                Levar o pneu furado para o borracheiro;
        }
}
```

- **Comando condicional se-senão**
- Escopo do ramo **"se"** não delimitado por chaves **'{'** e **'}'**
- Escopo do ramo **"senão"** delimitado por chaves **'{'** e **'}'**

Livro 01

Escopos com mais de um comando devem ser delimitados por chaves

Uso de endentações para ressaltar escopos

```
TrocarPneuFurado {
        Se (o estepe estiver murcho)
                Chamar o borracheiro;
        Senão {
                Pegar o macaco e a chave de boca no porta-malas;
                Afrouxar com a chave de boca todos os parafusos da roda;
                Levantar o carro com o macaco;
                Retirar todos os parafusos da roda;
                Retirar o pneu furado colocando-o no porta-malas;
                Posicionar o estepe no local em que será acoplado;
                Recolocar e apertar ligeiramente os parafusos;
                Baixar o carro;
                Apertar fortemente os parafusos;
                Guardar as ferramentas usadas;
                Levar o pneu furado para o borracheiro;
        }
}
```

Exemplo: algoritmo para troca de lâmpada queimada

Livro 01

```
TrocarLâmpadaQueimada {
        Se (houver na dispensa lâmpada boa de mesma potência)
                Pegar a lâmpada;
        Senão
                Comprar lâmpada de mesma potência;
        Posicionar a escada abaixo do bocal da lâmpada;
        Subir na escada até alcançar a lâmpada;
        Girar a lâmpada no sentido horário até soltá-la;
        Posicionar a nova lâmpada no bocal;
        Girá-la no sentido anti-horário até prendê-la;
        Descer da escada;
}
```

A **linguagem** para descrever o algoritmo deve ser **clara** e sem **ambiguidades** para o executor

Neste algoritmo há três **comandos repetitivos**

Livro 01

```
TrocarLâmpadaQueimada {
        Se (houver na dispensa lâmpada boa de mesma potência)
                Pegar a lâmpada;
        Senão
                Comprar lâmpada de mesma potência;
        Posicionar a escada abaixo do bocal da lâmpada;
        Subir na escada até alcançar a lâmpada;
        Girar a lâmpada no sentido horário até soltá-la;
        Posicionar a nova lâmpada no bocal;
        Girá-la no sentido anti-horário até prendê-la;
        Descer da escada;
}
```

Exercício 4.2:

1. Elaborar um algoritmo estabelecendo as atividades de um trabalhador, desde o instante em que ele acorda até o momento em que ele começa a exercer suas funções em seu ambiente de trabalho.

Livro 01

4.3 – Algoritmos para computadores

- É a abordagem desta disciplina

- No início de sua existência, os computadores faziam apenas **processamento numérico**, resolvendo diversos **problemas matemáticos**

- Hoje o **processamento não numérico** é ainda mais importante que o numérico, atuando sobre informações bastante complexas, compostas de **números, textos, imagens e sons**

A seguir, **métodos** e **algoritmos** para a resolução de quatro **problemas matemáticos simples**:

- Cálculo das raízes de uma equação do segundo grau
- Cálculo do fatorial de um número inteiro
- Soma dos termos de uma progressão aritmética
- Cálculo da integral definida de uma determinada função com uma variável

Cálculo das raízes de uma equação do segundo grau

- Seja a seguinte equação genérica do 2º grau:

$$\mathbf{A*x^2 + B*x + C = 0}$$

- Onde, por hipótese,

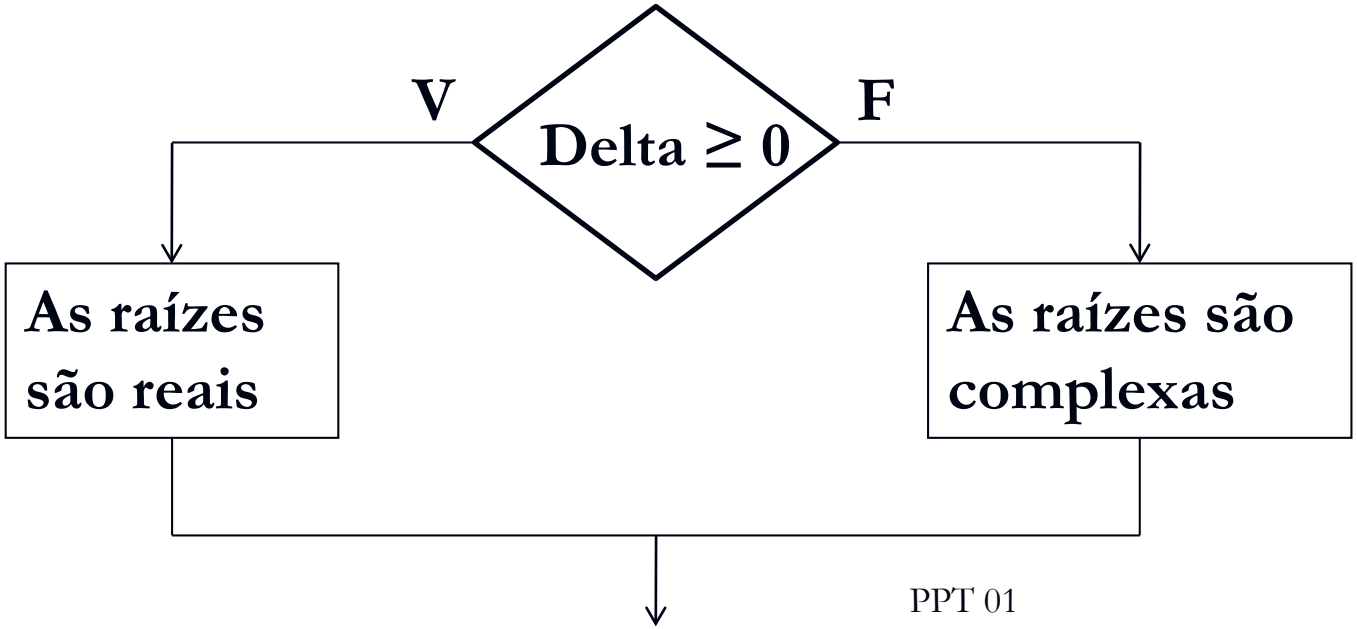
A, **B** e **C** são números reais e $\mathbf{A \neq 0}$

- O **método** escolhido para a determinação das raízes é o de **Baskara**

Fórmula de Baskara:

$$x = \frac{-B \pm \sqrt{B^2 - 4 * A * C}}{2 * A} \qquad A \neq 0$$

Discriminante **Delta**: **Delta = $B^2 - 4 * A * C$**



- No caso real, as raízes são dadas por:

$$x_1 = \frac{-B + \sqrt{Delta}}{2 * A} \quad \text{e} \quad x_2 = \frac{-B - \sqrt{Delta}}{2 * A}$$

- E no caso complexo:

$$x_1 = \frac{-B}{2 * A} + i * \frac{\sqrt{-Delta}}{2 * A} \quad \text{e} \quad x_2 = \frac{-B}{2 * A} - i * \frac{\sqrt{-Delta}}{2 * A}$$

- Então pode-se escrever o seguinte algoritmo:

RaizesEquação2Grau {

Variáveis Reais: A, B, C, X1, X2, Delta, Real, Imag;

Ler (A, B, C);

Delta ← B^2 - 4*A*C;

Se (Delta ≥ 0) {

$X1 \leftarrow \frac{-B + \sqrt{Delta}}{2*A}$; $X2 \leftarrow \frac{-B - \sqrt{Delta}}{2*A}$;

Escrever ("X1 = ", X1, " e X2 = ", X2);

}

Senão {

$Real \leftarrow \frac{-B}{2*A}$; $Imag \leftarrow \frac{\sqrt{-Delta}}{2*A}$;

Escrever ("X1 = (", Real, ") + i (", Imag, ") e X2 = (", Real, ") - i (", Imag, ")");

}

}

- **Se os valores lidos para A, B e C forem 1, -7 e 12:**

  Resultado escrito: x1 = 4 e X2 = 3

- **Se forem 1, 4 e 5**

  Resultado escrito: x1 = (-2)+i(1) e X2 = (-2)-i(1)

Livro 01

RaizesEquação2Grau {
Variáveis Reais: A, B, C, X1, X2, Delta, Real, Imag;

Ler (A, B, C);
Delta ← B^2 - 4*A*C;
Se (Delta ≥ 0) {
X1 ← $\frac{-B + \sqrt{Delta}}{2*A}$; X2 ← $\frac{-B - \sqrt{Delta}}{2*A}$;
Escrever ("X1 = ", X1, " e X2 = ", X2);
}
Senão {
Real ← $\frac{-B}{2*A}$; Imag ← $\frac{\sqrt{-Delta}}{2*A}$;
Escrever ("X1 = (", Real, ") + i (", Imag, ") e X2 = (", Real, ") - i (", Imag, ")");
}
}

- **Comando condicional (se-senão):**

Comandos de atribuição e de saída em seus escopos

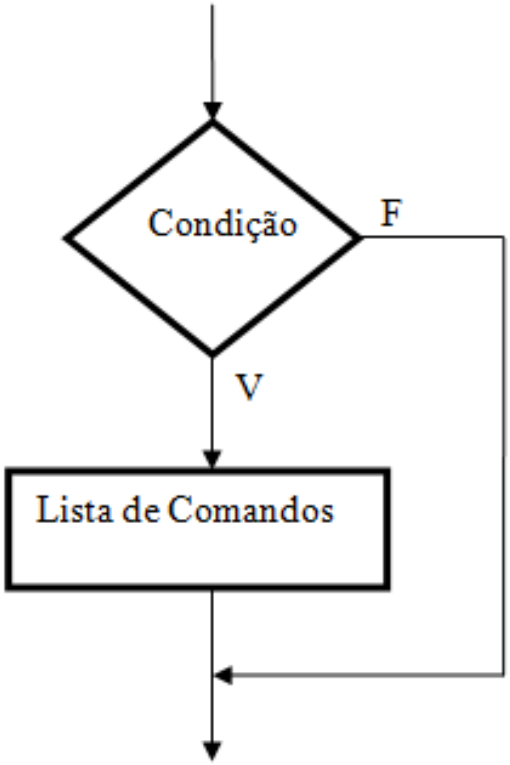
Livro 01

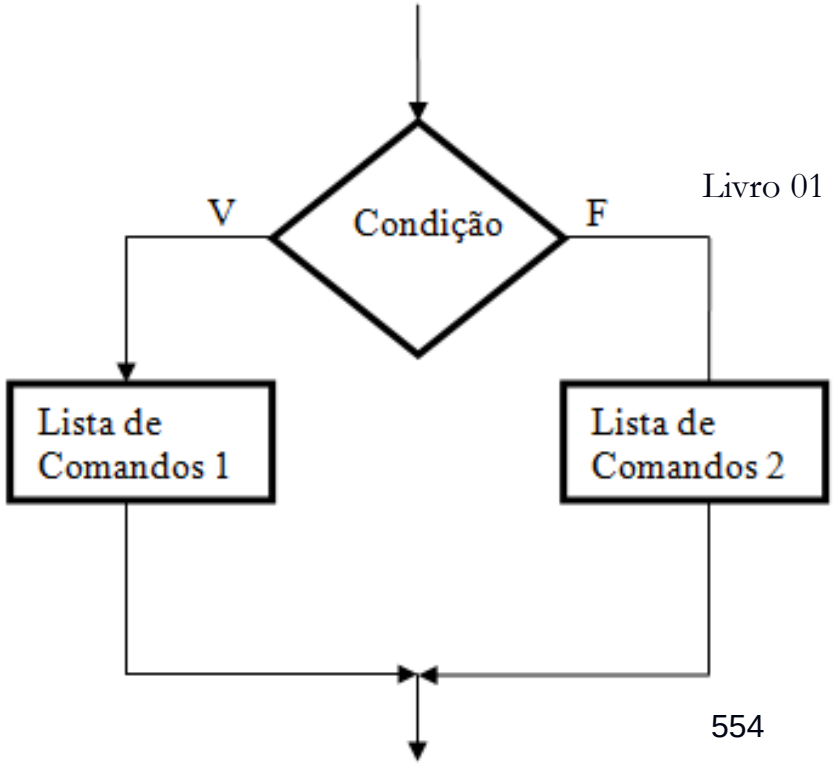
```
RaizesEquação2Grau {
        Variáveis Reais: A, B, C, X1, X2, Delta, Real, Imag;

        Ler (A, B, C);
        Delta ← B^2 - 4*A*C;
        Se (Delta ≥ 0) {
                X1 ← (-B + √Delta) / (2*A) ; X2 ← (-B - √Delta) / (2*A) ;
                Escrever ("X1 = ", X1, " e  X2 = ", X2);
        }
        Senão {
                Real ← -B / (2*A) ; Imag ← √(-Delta) / (2*A) ;
                Escrever ("X1 = (", Real, ") + i (", Imag, ") e X2 = (", Real,
                        ") - i (", Imag, ")");
        }
}
```

- Formas gerais dos **comandos condicionais**:
  - **se (condição) lista de comandos**
  - **se (condição) lista de comandos 1**
    senão lista de comandos 2

Fluxogramas explicativos:





Livro 01

- **Comandos de atribuição:**

Forma geral: Variável ← Expressão;

Variável recebe o valor calculado de expressão

```
RaizesEquação2Grau {
        Variáveis Reais: A, B, C, X1, X2, Delta, Real, Imag;

        Ler (A, B, C);
        Delta ← B^2 - 4*A*C;
        Se (Delta ≥ 0) {
                X1 ← (-B + √Delta) / (2*A); X2 ← (-B - √Delta) / (2*A);
                Escrever ("X1 = ", X1, " e X2 = ", X2);
        }
        Senão {
                Real ← -B / (2*A); Imag ← √(-Delta) / (2*A);
                Escrever ("X1 = (", Real, ") + i (", Imag, ") e X2 = (", Real,
                        ") - i (", Imag, ")");
        }
}
```

- **Comando de entrada ou de leitura:**

Forma geral: Ler (Lista de Variáveis);

```
RaizesEquação2Grau {
        Variáveis Reais: A, B, C, X1, X2, Delta, Real, Imag;

        Ler (A, B, C);
        Delta ← B^2 - 4*A*C;
        Se (Delta ≥ 0) {
                X1 ← (-B + √Delta) / (2*A) ; X2 ← (-B - √Delta) / (2*A) ;
                Escrever ("X1 = ", X1, " e  X2 = ", X2);
        }
        Senão {
                Real ← -B / (2*A) ; Imag ← √(-Delta) / (2*A) ;
                Escrever ("X1 = (", Real, ") + i (", Imag, ") e X2 = (", Real,
                        ") - i (", Imag, ")");
        }
}
```

- **Comandos de saída ou de escrita:**

Forma geral: **Escrever (Lista de Elementos de Escrita);**

Elemento de escrita: Texto entre aspas ou valor de expressão

```
RaizesEquação2Grau {
        Variáveis Reais: A, B, C, X1, X2, Delta, Real, Imag;

        Ler (A, B, C);
        Delta ← B^2 - 4*A*C;
        Se (Delta ≥ 0) {
                X1 ← (-B + √Delta) / (2*A) ; X2 ← (-B - √Delta) / (2*A) ;
                Escrever ("X1 = ", X1, " e X2 = ", X2);
        }
        Senão {
                Real ← -B / (2*A) ; Imag ← √(-Delta) / (2*A) ;
                Escrever ("X1 = (", Real, ") + i (", Imag, ") e X2 = (", Real,
                        ") - i (", Imag, ")");
        }
}
```

- **Declaração de variáveis:**

Todas as variáveis do programa são do tipo real

```
RaizesEquação2Grau {
        Variáveis Reais: A, B, C, X1, X2, Delta, Real, Imag;

        Ler (A, B, C);
        Delta ← B^2 - 4*A*C;
        Se (Delta ≥ 0) {
                X1 ← (-B + √Delta) / (2*A) ; X2 ← (-B - √Delta) / (2*A) ;
                Escrever ("X1 = ", X1, " e  X2 = ", X2);
        }
        Senão {
                Real ← -B / (2*A) ; Imag ← √(-Delta) / (2*A) ;
                Escrever ("X1 = (",  Real, ") + i (", Imag, ") e X2 = (", Real,
                        ") - i (", Imag, ")");
        }
}
```

- **O algoritmo não trabalha com variáveis do tipo complexo**

As variáveis Real e Imag auxiliam a escrita de raízes complexas

```
RaizesEquação2Grau {
        Variáveis Reais: A, B, C, X1, X2, Delta, Real, Imag;

        Ler (A, B, C);
        Delta ← B^2 - 4*A*C;
        Se (Delta ≥ 0) {
                X1 ← (-B + √Delta) / (2*A) ; X2 ← (-B - √Delta) / (2*A) ;
                Escrever ("X1 = ", X1, " e  X2 = ", X2);
        }
        Senão {
                Real ← -B / (2*A) ; Imag ← √(-Delta) / (2*A) ;
                Escrever ("X1 = (", Real, ") + i (", Imag, ") e X2 = (", Real,
                        ") - i (", Imag, ")");
        }
}
```

Cálculo do fatorial de um número inteiro

- Método:

  n! = 1 * 2 * 3 * ... * (n-1) * n (por hipótese, **n ≥ 0**)

- Lê-se o valor de **n**

- Inicializa-se o valor do fatorial com **1**

- Multiplica-se cumulativamente todos os valores inteiros do intervalo **[2, n]** pelo valor do fatorial

Algoritmo do fatorial:

```
Fatorial {
        Variáveis Inteiras: n, fat, i;

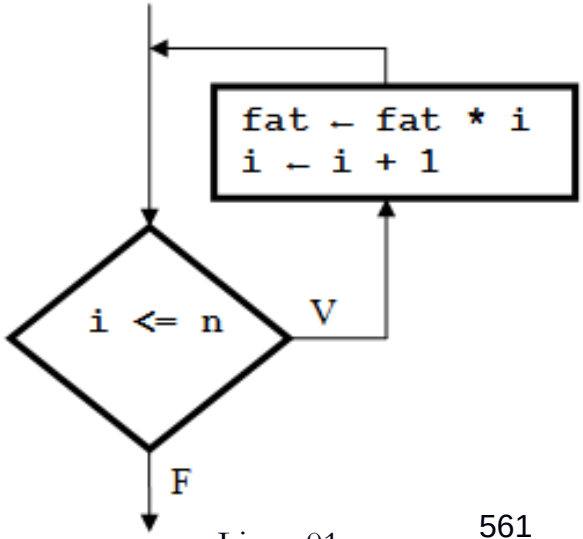
        Ler (n);
        fat ← 1; i ← 2;
        enquanto (i ≤ n) {
                fat ← fat * i;
                i ← i + 1;
        }
        Escrever ("Fatorial (", n, ") = ", fat);
}
```

Livro 01

A multiplicação cumulativa tem caráter repetitivo

Isso é expresso pelo comando "enquanto"

Fluxograma do comando enquanto:



Livro 01

Algoritmo do fatorial:

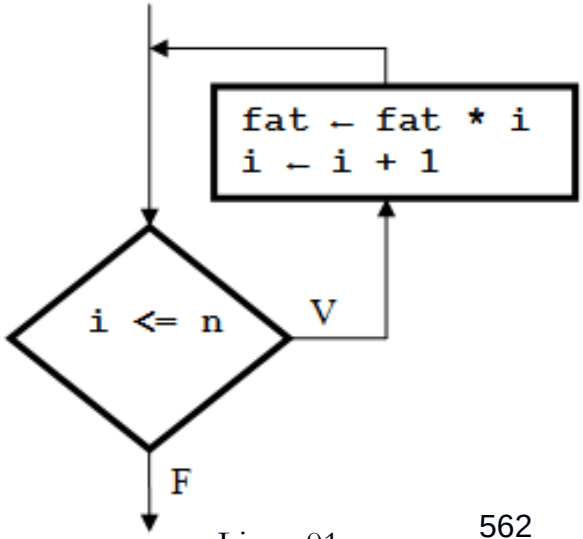
```
Fatorial {
        Variáveis Inteiras: n, fat, i;

        Ler (n);
        fat ← 1; i ← 2;
        enquanto (i ≤ n) {
                fat ← fat * i;
                i ← i + 1;
        }
        Escrever ("Fatorial (", n, ") = ", fat);
}
```

Livro 01

A multiplicação cumulativa tem caráter repetitivo

Isso é expresso pelo comando "enquanto"



Se o valor lido para n for 7:

Resultado escrito: Fatorial (7) = 5040

Livro 01

Cálculo da soma dos termos de uma PA:

- Conhecidos o 1º termo a_1, a razão **r** e o nº de termos **n**

- Método: sem usar as fórmulas

$$S = \frac{n*(a_1 + a_n)}{2} \text{ e } a_n = a_1 + (n-1)*r$$

Usando a fórmula $a_i = a_{i-1} + r$, começando por a_1 e encerrando por a_n, cada termo da PA vai sendo incluído na somatória

Exemplo: $a_1 = 2$, $r = 3$ e $n = 5$

Usando $a_i = a_{i-1} + r$, **PA = {2, 5, 8, 11, 14}**

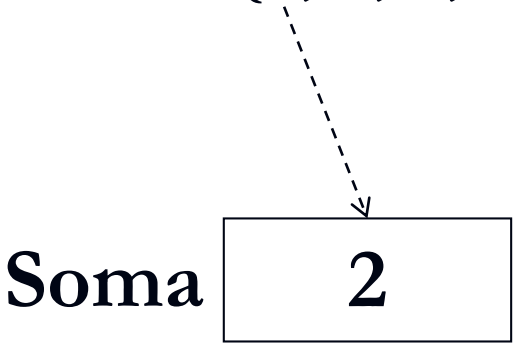
Inicialmente:

Soma | **0**

Exemplo: $a_1 = 2$, $r = 3$ e $n = 5$

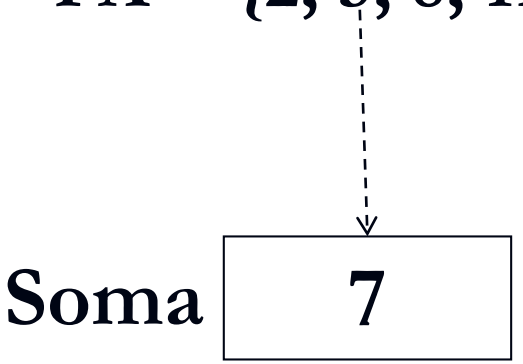
Usando $a_i = a_{i-1} + r$, **PA = {2, 5, 8, 11, 14}**

A seguir:



Exemplo: $a_1 = 2$, $r = 3$ e $n = 5$

Usando $a_i = a_{i-1} + r$, **PA = {2, 5, 8, 11, 14}**



Exemplo: $a_1 = 2$, $r = 3$ e $n = 5$

Usando $a_i = a_{i-1} + r$, **PA = {2, 5, 8, 11, 14}**



Exemplo: $a_1 = 2$, $r = 3$ e $n = 5$

Usando $a_i = a_{i-1} + r$, **PA = {2, 5, 8, 11, 14}**



Exemplo: $a_1 = 2$, $r = 3$ e $n = 5$

Usando $a_i = a_{i-1} + r$, **PA = {2, 5, 8, 11, 14}**



- Então, pode-se escrever o algoritmo a seguir

```
SomaPA {
        Variáveis Inteiras: a1, r, n, soma, aq, i;

        Ler (a1, n, r);
        soma ← 0; aq ← a1; i ← 1;
        enquanto (i ≤ n) {
                soma ← soma + aq;
                aq ← aq + r;
                i ← i + 1;
        }
        Escrever ("Progressao aritmetica:\n\n", "Primeiro termo: ", a1,
                "\nRazao: ", r, "\nNumero de termos: ", n,
                "\n\nSoma: ", soma);
}
```

aq: termo da PA a ser acrescido na somatória

Livro 01

i: número do termo a ser acrescentado na somatória

```
SomaPA {
        Variáveis Inteiras: a1, r, n, soma, aq, i;

        Ler (a1, n, r);
        soma ← 0; aq ← a1; i ← 1;
        enquanto (i ≤ n) {
                soma ← soma + aq;
                aq ← aq + r;
                i ← i + 1;
        }
        Escrever ("Progressao aritmetica:\n\n", "Primeiro termo: ", a1,
                "\nRazao: ", r, "\nNumero de termos: ", n,
                "\n\nSoma: ", soma);
}
```

Livro 01

\n: new-line **(nl)** em C; posiciona o cursor no início da linha seguinte

```
SomaPA {
        Variáveis Inteiras: a1, r, n, soma, aq, i;

        Ler (a1, n, r);
        soma ← 0; aq ← a1; i ← 1;
        enquanto (i ≤ n) {
                soma ← soma + aq;
                aq ← aq + r;
                i ← i + 1;
        }
        Escrever ("Progressao aritmetica:\n\n", "Primeiro termo: ", a1,
                "\nRazao: ", r, "\nNumero de termos: ", n,
                "\n\nSoma: ", soma);
}
```

Livro 01

Resultado para
a1 = 2, n = 7 e r = 3:

```
Progressao aritmetica:

Primeiro termo: 2
Razao: 3
Numero de termos: 7

Soma: 77
```

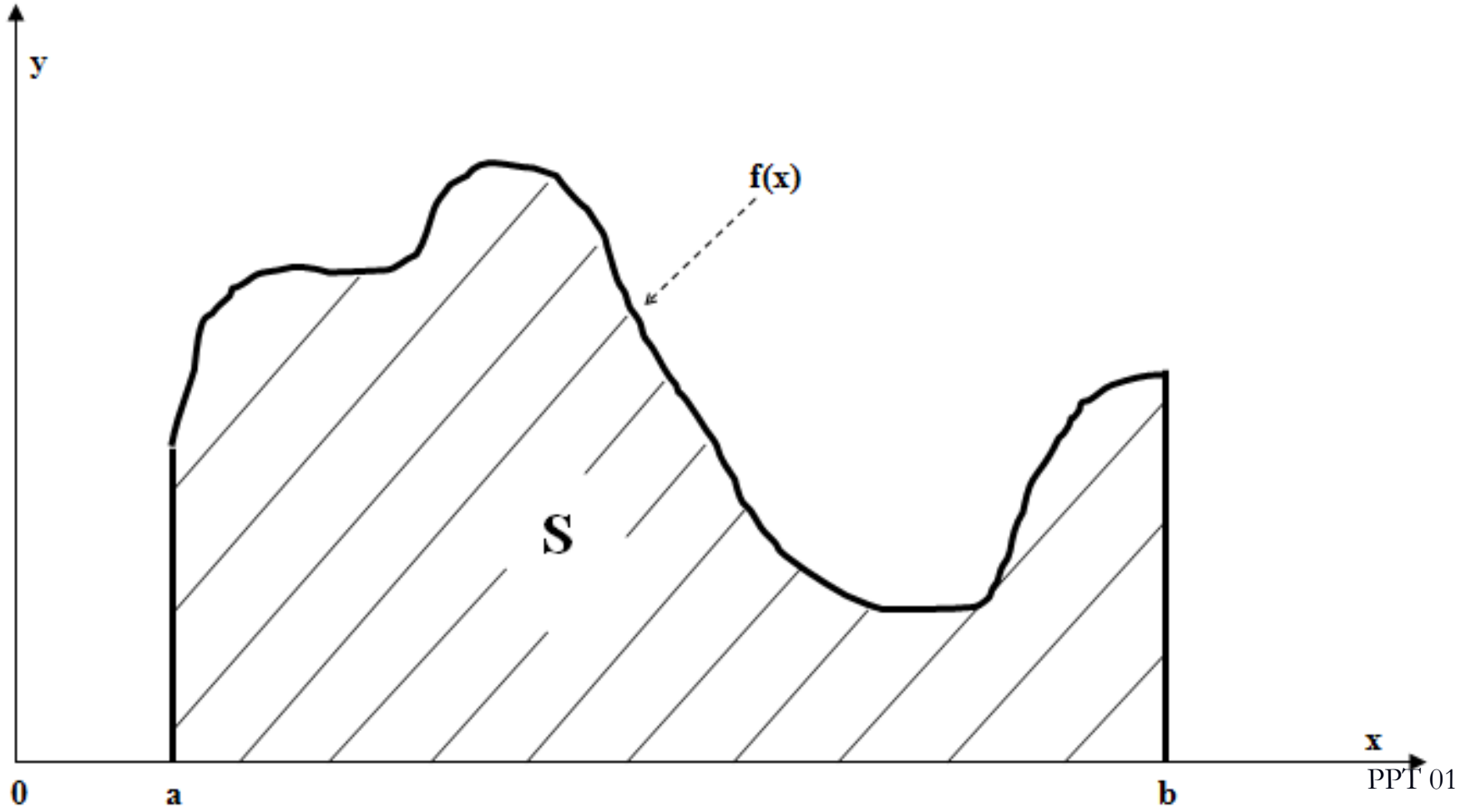
Cálculo da integral definida de uma determinada função com uma variável

Determinação da tarefa, com detalhes:

- Calcular o valor da **integral definida** de uma função **f(x)**, num dado intervalo lido **[a, b]**, com uma dada precisão lida **p**

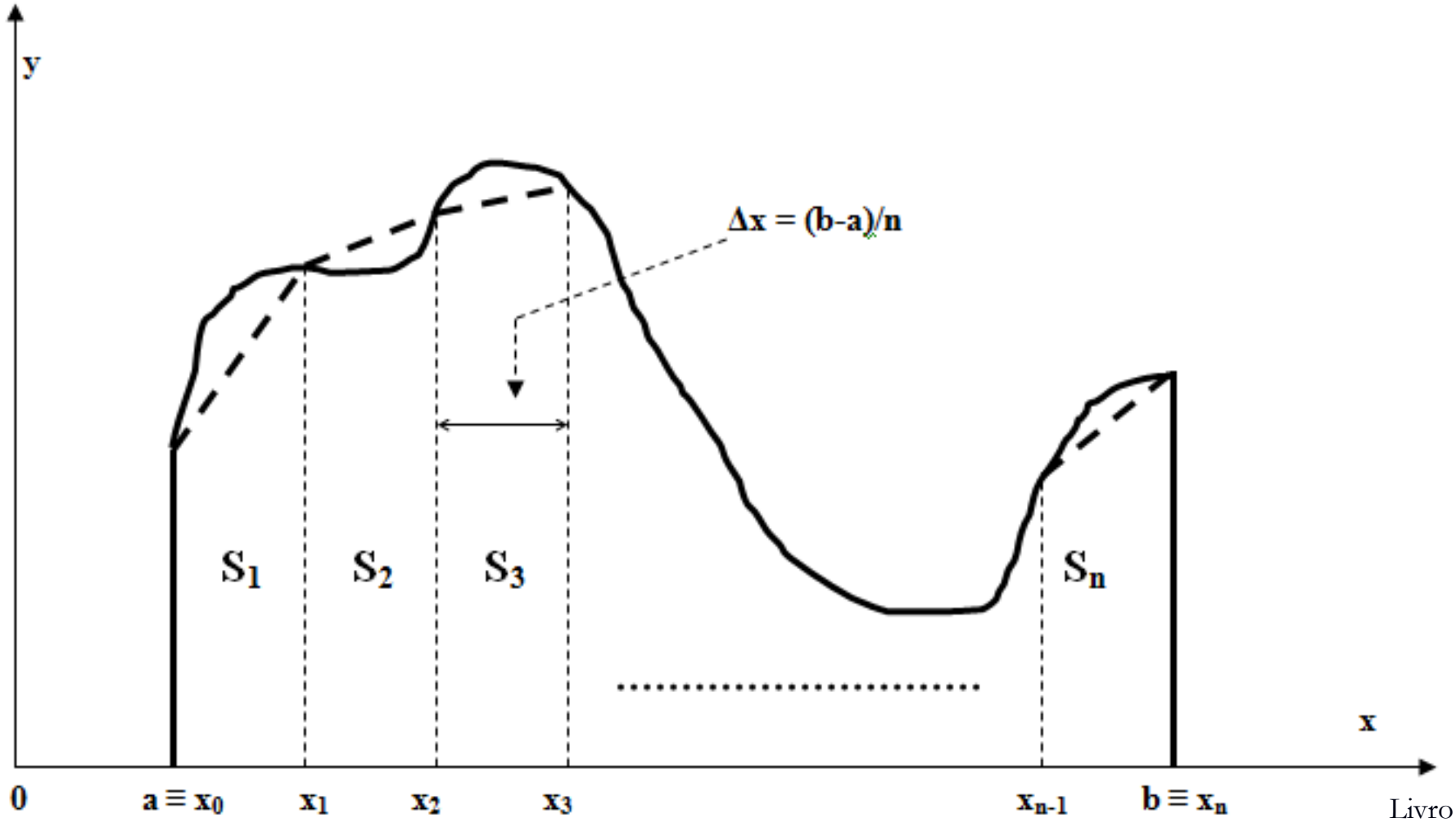
- Supor que, no referido intervalo, a função **não** assuma valores **negativos**

Interpretação gráfica de integral definida:



A integral definida de **f(x)** no intervalo **[a, b]** é a área **S**

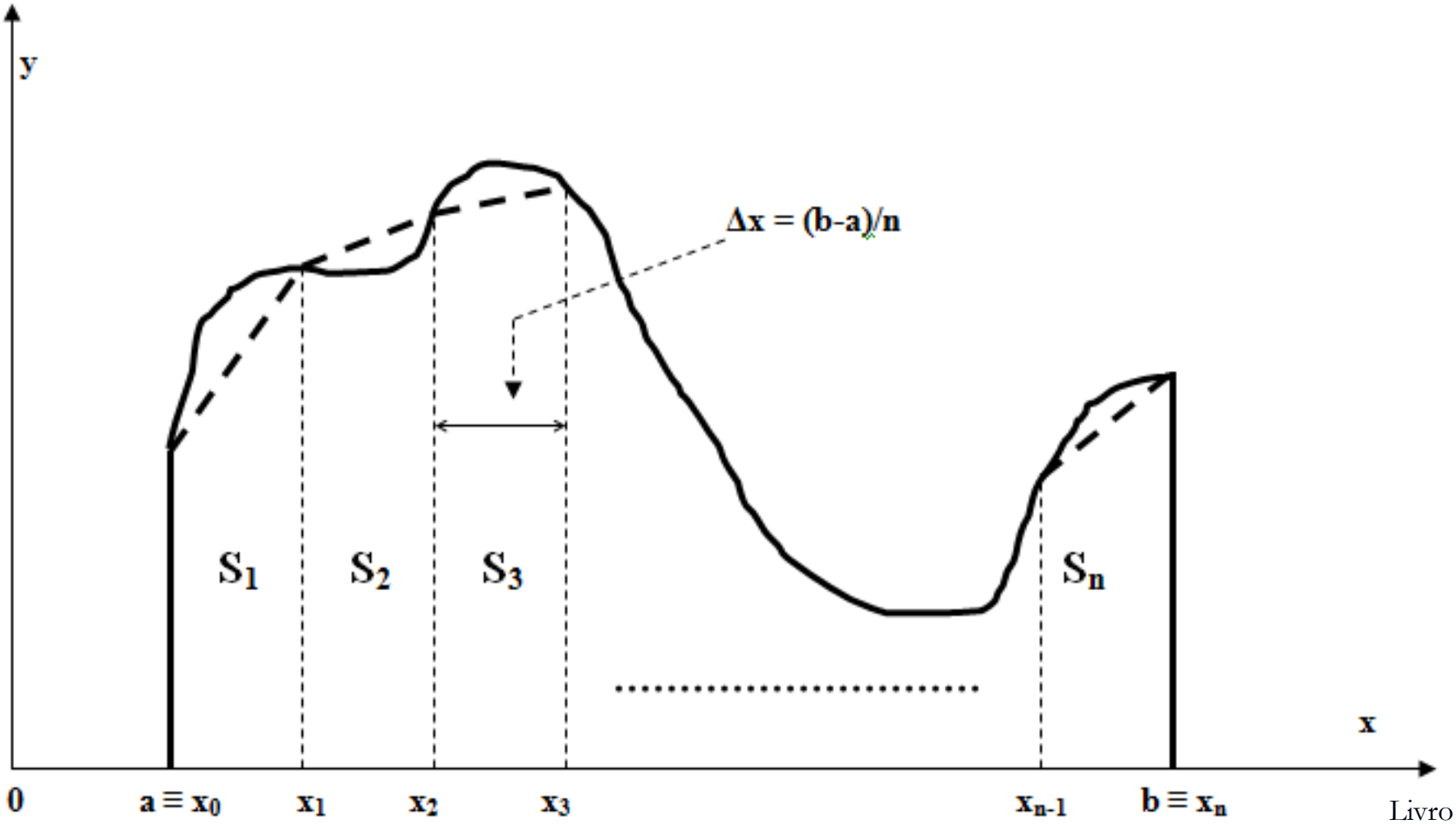
Método utilizado: Regra do Trapézio



Livro 01

- Dividir o intervalo de integração em n subintervalos de igual tamanho, determinando as sub-áreas S_1, S_2, S_3, ... , S_n

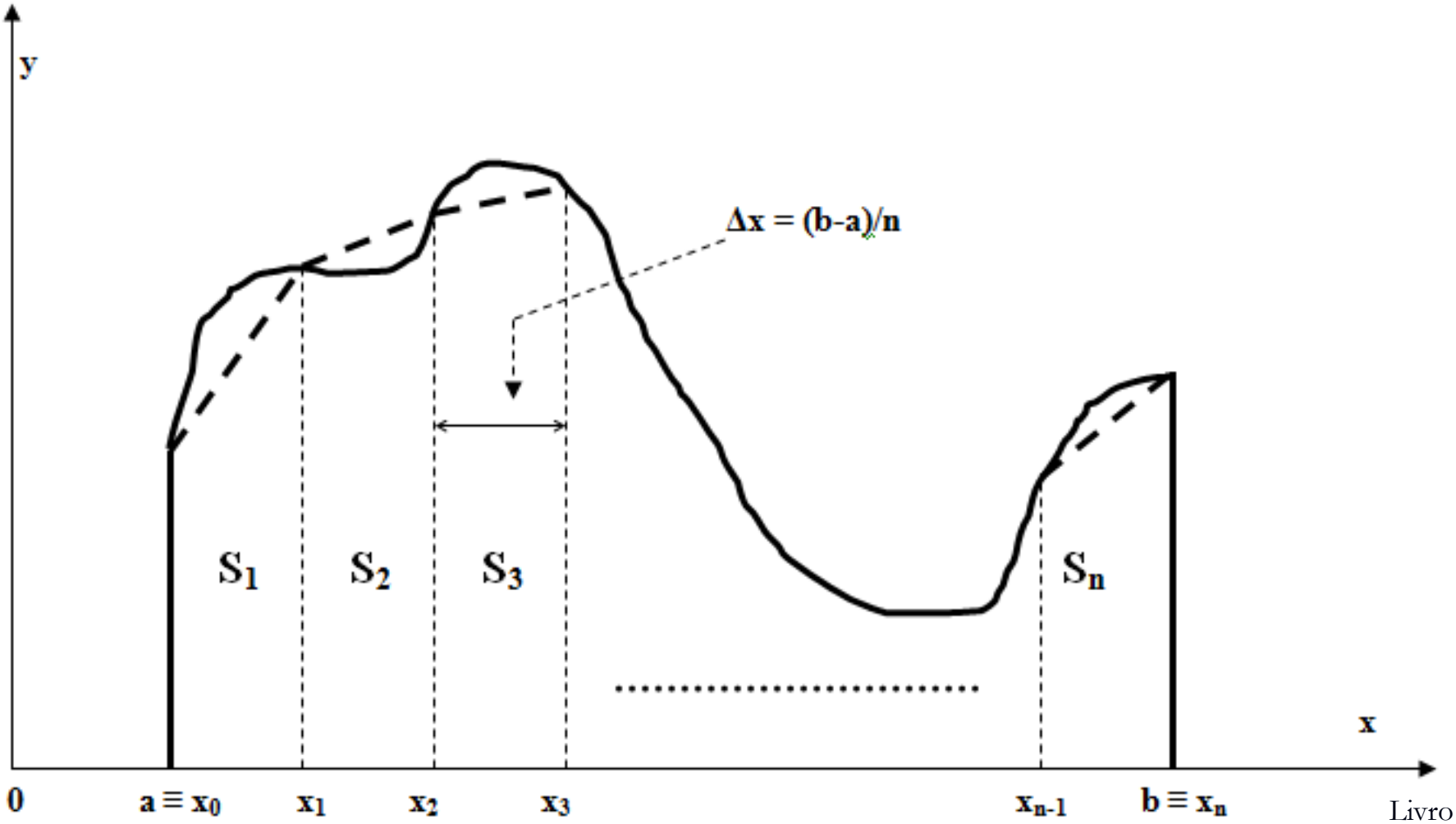
Método utilizado: Regra do Trapézio



Livro 01

- Aproximar a curva em cada sub-área para um segmento de reta
- Cada sub-área fica aproximada para um trapézio

Método utilizado: Regra do Trapézio

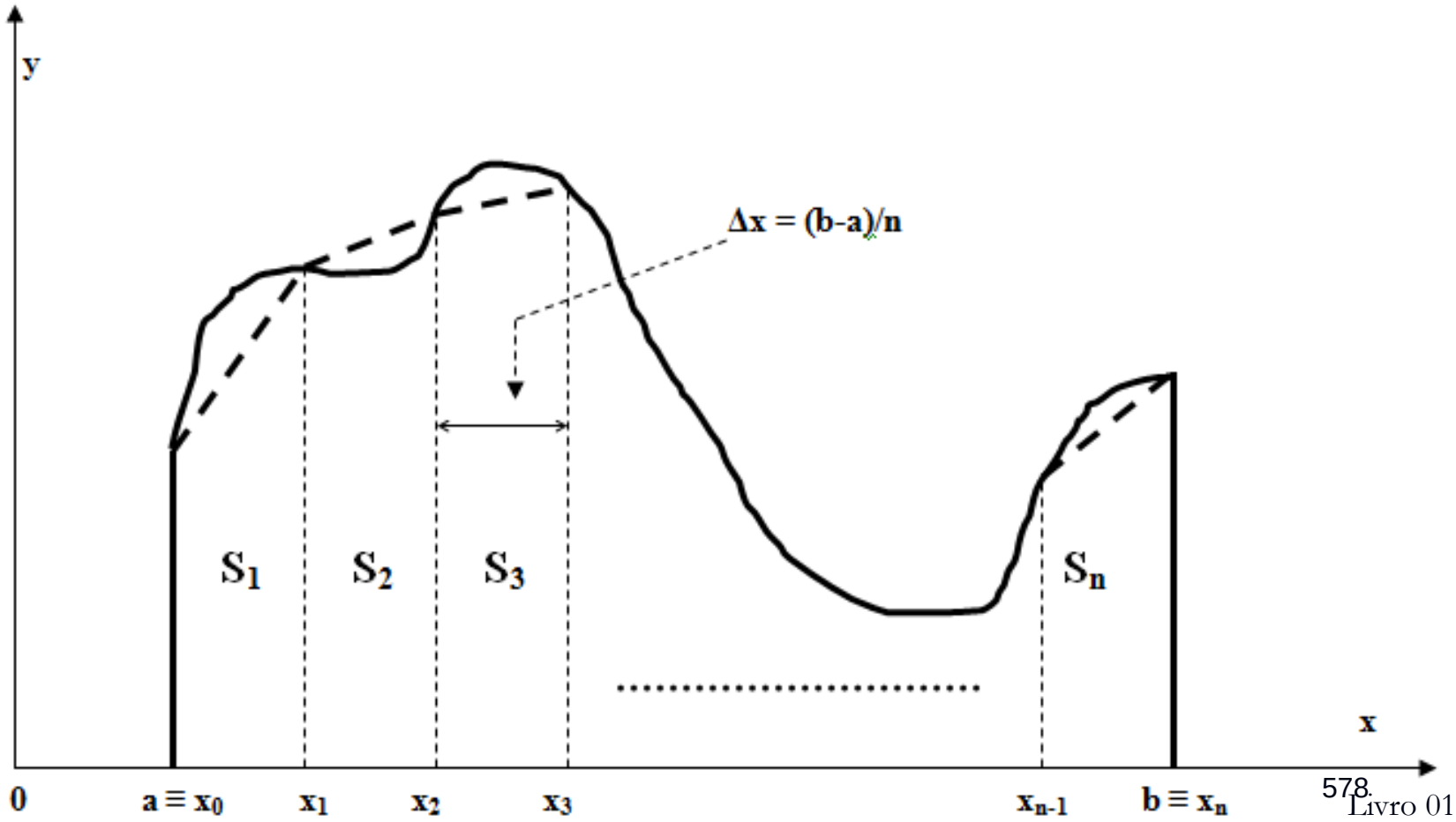


Livro 01

- Calcular o somatório das sub-áreas de todos os trapézios, que é uma aproximação para o valor procurado da integral definida

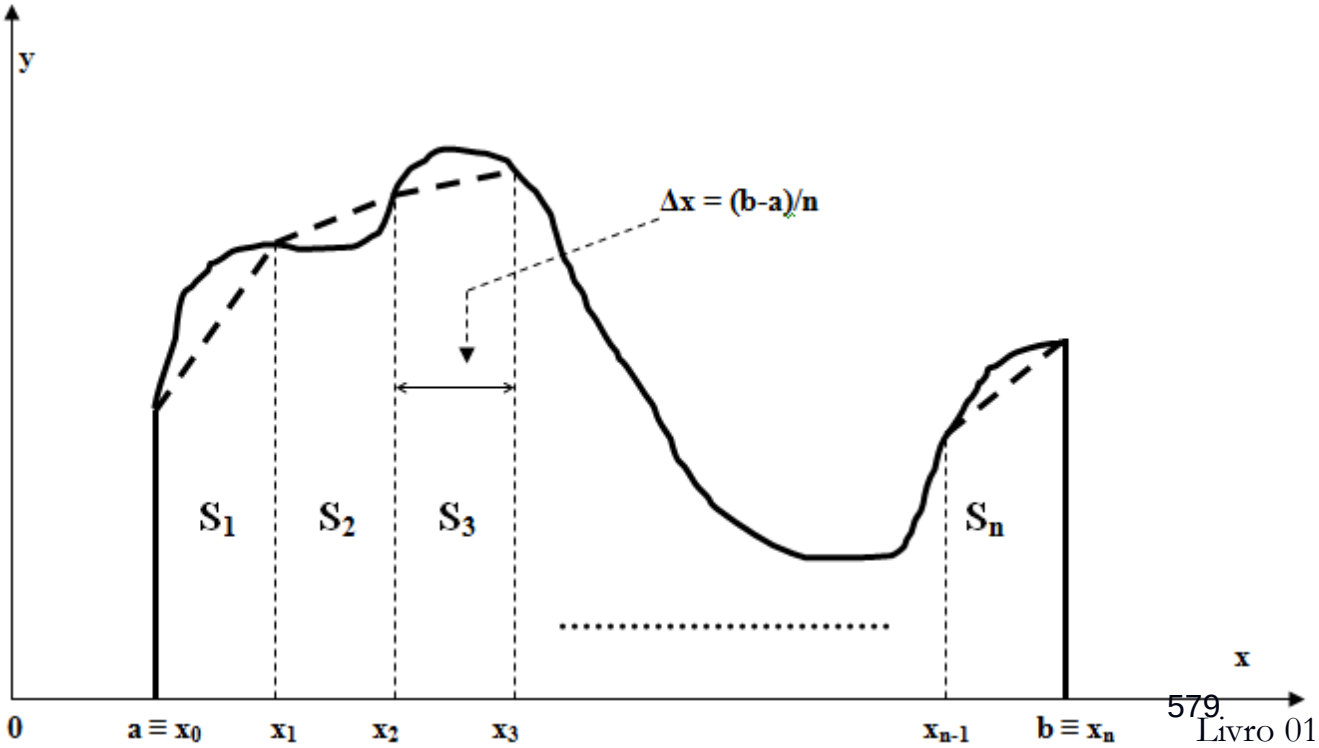
- A soma das áreas dos trapézios é dada por $S = \sum_{i=1}^{n} S_i$

- A área de cada trapézio é dada por $S_i = \frac{\Delta x * (f(x_{i-1}) + f(x_i))}{2}$



- Se **n = 10** e sendo conhecidos os valores de **a** e **b**, a **somatória** pode ser calculada com os seguintes comandos:

$$S \leftarrow 0;\ i \leftarrow 1;\ n \leftarrow 10;\ Dx \leftarrow (b-a)/n;$$
$$\text{enquanto } (i \leq n)\ \{$$
$$STrap \leftarrow \frac{Dx * (f(a + (i-1) * Dx) + f(a + i * Dx))}{2};$$
$$S \leftarrow S + STrap;\ \ i \leftarrow i + 1;$$
$$\}$$



Obtenção da precisão lida **p** no resultado:

- Calcular um valor aproximado para a área **S**, usando um valor inicial para **n** (**10** p. ex.); seja $\mathbf{S_{10}}$ esse valor
- Calcular outro valor aproximado para **S**, dobrando-se o valor de **n** (**20** p. ex.); seja $\mathbf{S_{20}}$ esse valor
- Se $\mathbf{|S_{20} - S_{10}| \leq p}$, adotar $\mathbf{S_{20}}$ como valor definitivo de **S**
  Senão, descartar $\mathbf{S_{10}}$, calcular $\mathbf{S_{40}}$ e compará-lo com $\mathbf{S_{20}}$
- Quando, para algum valor **x**, $\mathbf{|S_{2*x} - S_x| \leq p}$, adotar $\mathbf{S_{2*x}}$ como valor definitivo de **S**

Obs.: Este procedimento só é válido se **f(x)** for **bem comportada** em **[a, b]**

Algoritmo da Regra do Trapézio com precisão exigida:

Livro 01

```
IntegralTrapézioPrecisão {
        Variáveis Reais: a, b, S1, S2, STrap, Dx, p;
        Variáveis Inteiras: i, n;
        Função Real: f (Parâmetro Real);

        Ler (a, b, p);
        S2 ← 0; n ← 5;
        repetir {
                S1 ← S2; n ← 2*n; Dx ← (b-a)/n; S2 ← 0; i ← 1;
                enquanto (i ≤ n) {
                        STrap ← (Dx * (f(a + (i - 1) * Dx) + f(a + i * Dx))) / 2;
                        S2 ← S2 + STrap; i ← i + 1;
                }
        } enquanto (|S2 – S1| > p);
        Escrever ("A integral de f(x) no intervalo [", a, " , ", b,
                "], com precisão ", p, " é ", S2);
}
```

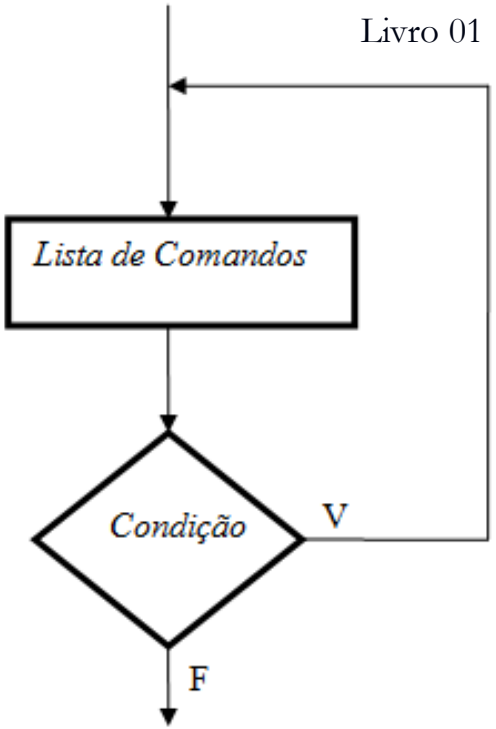
Outro tipo de comando repetitivo: **repetir-enquanto**


```
IntegralTrapézioPrecisão {
        Variáveis Reais: a, b, S1, S2, STrap, Dx, p;
        Variáveis Inteiras: i, n;
        Função Real: f (Parâmetro Real);

        Ler (a, b, p);
        S2 ← 0; n ← 5;
        repetir {
                S1 ← S2; n ← 2*n; Dx ← (b-a)/n; S2 ← 0; i ← 1;
                enquanto (i ≤ n) {
                        STrap ← (Dx * (f(a + (i - 1) * Dx) + f(a + i * Dx))) / 2;
                        S2 ← S2 + STrap; i ← i + 1;
                }
        } enquanto (|S2 – S1| > p);
        Escrever ("A integral de f(x) no intervalo [", a, " , ", b,
                "], com precisão ", p, " é ", S2);
}
```

Fluxograma do comando **repetir-enquanto**:

Livro 01

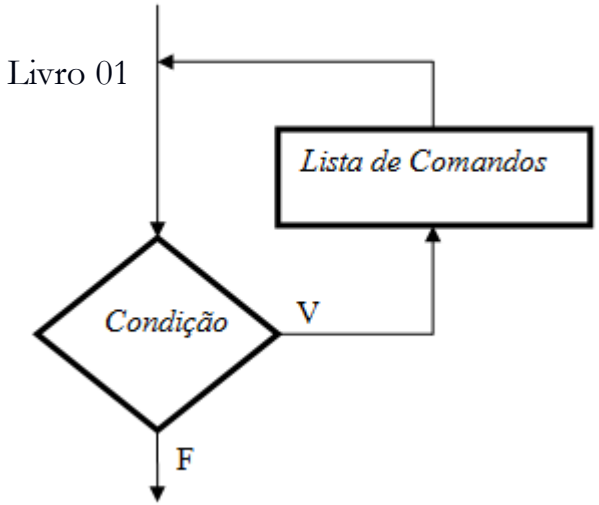


Livro 01

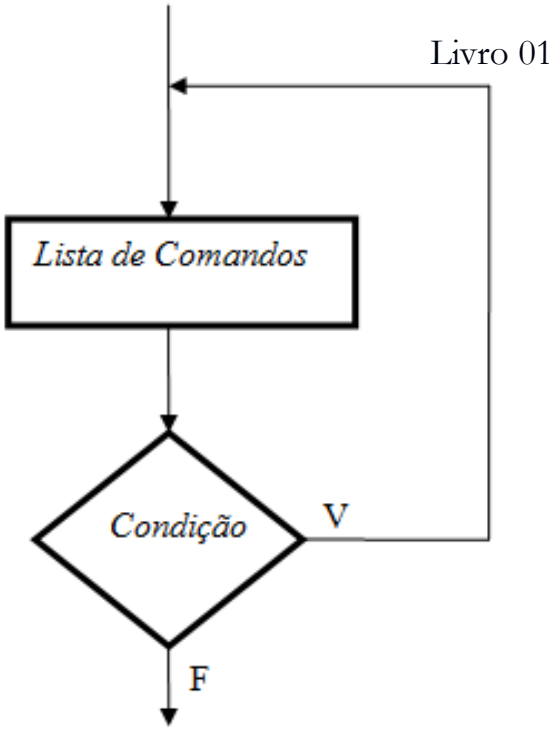
```
IntegralTrapézioPrecisão {
        Variáveis Reais: a, b, S1, S2, STrap, Dx, p;
        Variáveis Inteiras: i, n;
        Função Real: f (Parâmetro Real);

        Ler (a, b, p);
        S2 ← 0; n ← 5;
        repetir {
                S1 ← S2; n ← 2*n; Dx ← (b-a)/n; S2 ← 0; i ← 1;
                enquanto (i ≤ n) {
                        STrap ← (Dx * (f(a + (i -1) *Dx) + f(a + i * Dx))) / 2;
                        S2 ← S2 + STrap; i ← i + 1;
                }
        } enquanto (|S2 – S1| > p);
        Escrever ("A integral de f(x) no intervalo [", a, " , ", b,
                "], com precisão ", p, " é ", S2);
}
```

Comparação dos comandos **enquanto** e **repetir-enquanto**:



enquanto



repetir-enquanto

- No comando **repetir-enquanto**, a **lista de comandos** é executada pelo menos **uma vez**

As variáveis **S1** e **S2** guardam valores de cálculos consecutivos da integral, sendo **S1** o valor antigo e **S2** aquele calculado com o novo valor de **n**

Livro 01

```
IntegralTrapézioPrecisão {
        Variáveis Reais: a, b, S1, S2, STrap, Dx, p;
        Variáveis Inteiras: i, n;
        Função Real: f (Parâmetro Real);

        Ler (a, b, p);
        S2 ← 0; n ← 5;
        repetir {
                S1 ← S2; n ← 2*n; Dx ← (b-a)/n; S2 ← 0; i ← 1;
                enquanto (i ≤ n) {
                        STrap ← (Dx * (f(a + (i -1) *Dx) + f(a + i * Dx))) / 2;
                        S2 ← S2 + STrap; i ← i + 1;
                }
        } enquanto (|S2 – S1| > p);
        Escrever ("A integral de f(x) no intervalo [", a, " , ", b,
                "], com precisão ", p, " é ", S2);
}
```

O valor 5 para n nunca é usado

Antes de ser usado, ele é multiplicado por 2

- A atribuição **S2 ← 0** no início é **artificial**

- Esse valor é logo atribuído a **S1** e um novo valor para **S2** é calculado, para ser comparado com o de **S1**

Livro 01

```
IntegralTrapézioPrecisão {
        Variáveis Reais: a, b, S1, S2, STrap, Dx, p;
        Variáveis Inteiras: i, n;
        Função Real: f (Parâmetro Real);

        Ler (a, b, p);
        S2 ← 0; n ← 5;
        repetir {
                S1 ← S2; n ← 2*n; Dx ← (b-a)/n; S2 ← 0; i ← 1;
                enquanto (i ≤ n) {
                        STrap ← (Dx * (f(a + (i -1) *Dx) + f(a + i * Dx))) / 2;
                        S2 ← S2 + STrap; i ← i + 1;
                }
        } enquanto (|S2 – S1| > p);
        Escrever ("A integral de f(x) no intervalo [", a, " , ", b,
                "], com precisão ", p, " é ", S2);
}
```

Os valores de n serão 10, 20, 40, 80, 160, etc.

- Se os valores lidos para **a**, **b** e **p** forem respectivamente **1.5**, **14.8** e **0.001**, e se o valor final de **S2** for **327.181**, será escrito:

```
A integral de f(x) no intervalo [1.5, 14.8],
com precisão 0.001 é 327.181
```

```
IntegralTrapézioPrecisão {
        Variáveis Reais: a, b, S1, S2, STrap, Dx, p;
        Variáveis Inteiras: i, n;
        Função Real: f (Parâmetro Real);

        Ler (a, b, p);
        S2 ← 0; n ← 5;
        repetir {
                S1 ← S2; n ← 2*n; Dx ← (b-a)/n; S2 ← 0; i ← 1;
                enquanto (i ≤ n) {
                        STrap ← (Dx * (f(a + (i -1) * Dx) + f(a + i * Dx))) / 2;
                        S2 ← S2 + STrap; i ← i + 1;
                }
        } enquanto (|S2 – S1| > p);
        Escrever ("A integral de f(x) no intervalo [", a, " , ", b,
                "], com precisão ", p, " é ", S2);
}
```

Livro 01

4.4 – Programas a partir de algoritmos

- P**rograma** foi definido como sendo uma sequência de **instruções** que, ao serem executadas por um computador, **realizam** uma determinada tarefa

- **Algoritmo** foi definido como sendo uma sequência finita e ordenada de **passos** (comandos executáveis e não ambíguos), que levam à aplicação de um **método** para a execução de uma **tarefa** ou resolução de um **problema**

- **Programa** também pode ser definido como sendo a **tradução** de um **algoritmo** para uma **linguagem de programação**

A seguir, **programas** a partir dos **algoritmos** para:

- Cálculo das raízes de uma equação do segundo grau
- Soma dos termos de uma progressão aritmética
- Cálculo da integral definida de uma determinada função com uma variável

Cálculo das raízes de uma equação do segundo grau:

Algoritmo

```
RaizesEquação2Grau {
        Variáveis Reais: A, B, C, X1, X2, Delta, Real, Imag;

        Ler (A, B, C);
        Delta ← B^2 - 4*A*C;
        Se (Delta ≥ 0) {
                X1 ← (-B + √Delta) / (2*A) ; X2 ← (-B - √Delta) / (2*A) ;
                Escrever ("X1 = ", X1, " e  X2 = ", X2);
        }
        Senão {
                Real ← -B / (2*A) ; Imag ← √(-Delta) / (2*A) ;
                Escrever ("X1 = (", Real, ") + i (", Imag, ") e X2 = (", Real,
                        ") - i (", Imag, ")");
        }
}
```

Livro 01

```
#include <stdio.h>   <---------- Para usar scanf e printf
#include <math.h>    <---------- Para usar pow e sqrt
int main () {
    float A, B, C, X1, X2, Delta, Real, Imag;
```

Algoritmo

```
RaizesEquação2Grau {
    Variáveis Reais: A, B, C, X1, X2, Delta, Real, Imag;

    Ler (A, B, C);
    Delta ← B^2 - 4*A*C;
    Se (Delta ≥ 0) {
        X1 ← (-B + √Delta) / (2*A) ; X2 ← (-B - √Delta) / (2*A) ;
        Escrever ("X1 = ", X1, " e X2 = ", X2);
    }
    Senão {
        Real ← -B / (2*A) ; Imag ← √(-Delta) / (2*A) ;
        Escrever ("X1 = (", Real, ") + i (", Imag, ") e X2 = (", Real,
            ") - i (", Imag, ")");
    }
}
```

```
} <----------
```

```
#include <stdio.h>
#include <math.h>
int main () {
    float A, B, C, X1, X2, Delta, Real, Imag;
    scanf ("%f%f%f", &A, &B, &C);
    Delta = pow (B, 2) – 4*A*C;
}
```

"%f%f%f": Ler 3 números reais e guardá-los em

&A, &B, &C: endereços de A, B e C

Algoritmo

```
RaizesEquação2Grau {
    Variáveis Reais: A, B, C, X1, X2, Delta, Real, Imag;

    Ler (A, B, C);
    Delta ← B² - 4*A*C;
    Se (Delta ≥ 0) {
        X1 ← (-B + √Delta) / (2*A) ; X2 ← (-B - √Delta) / (2*A) ;
        Escrever ("X1 = ", X1, " e X2 = ", X2);
    }
    Senão {
        Real ← -B / (2*A) ; Imag ← √(-Delta) / (2*A) ;
        Escrever ("X1 = (", Real, ") + i (", Imag, ") e X2 = (", Real,
            ") - i (", Imag, ")");
    }
}
```

```
#include <stdio.h>
#include <math.h>
int main () {
    float A, B, C, X1, X2, Delta, Real, Imag;
    scanf (“%f%f%f”, &A, &B, &C);
    Delta = pow (B, 2) – 4*A*C;
    if (Delta >= 0) {


    } 
    else {


    }
}
```

Algoritmo

```
RaizesEquação2Grau {
    Variáveis Reais: A, B, C, X1, X2, Delta, Real, Imag;

    Ler (A, B, C);
    Delta ← B² - 4*A*C;
    Se (Delta ≥ 0) {
        X1 ← (-B + √Delta) / (2*A) ; X2 ← (-B - √Delta) / (2*A) ;
        Escrever (“X1 = ”, X1, “ e X2 = ”, X2);
    }
    Senão {
        Real ← -B / (2*A) ; Imag ← √(-Delta) / (2*A) ;
        Escrever (“X1 = (”, Real, “) + i (”, Imag, “) e X2 = (”, Real,
            “) - i (”, Imag, “)”);
    }
}
```

```
#include <stdio.h>
#include <math.h>
int main () {
    float A, B, C, X1, X2, Delta, Real, Imag;
    scanf ("%f%f%f", &A, &B, &C);
    Delta = pow (B, 2) – 4*A*C;
    if (Delta >= 0) {
        X1 = (-B+sqrt(Delta))/(2*A);  X2 = (-B-sqrt(Delta))/(2*A);
        printf ("X1 = %g e X2 = %g", X1, X2);
    }
    else {
    }
}
```

"X1 = %g e X2 = %g": cadeia de controle da escrita

Escreve tudo o que aparece menos os dois %g's

O 1º %g é para escrever o valor de X1, o 2º para X2, ambos no formato real

Se (Delta ≥ 0) {

$$X1 \leftarrow \frac{-B + \sqrt{Delta}}{2*A}; X2 \leftarrow \frac{-B - \sqrt{Delta}}{2*A};$$

Escrever ("X1 = ", X1, " e X2 = ", X2);

}

```
#include <stdio.h>
#include <math.h>
int main () {
    float A, B, C, X1, X2, Delta, Real, Imag;
    scanf ("%f%f%f", &A, &B, &C);
    Delta = pow (B,
    if (Delta >= 0) {
        X1 = (-B+sq
        printf ("X1 =
    }
    else {
        Real = -B / (2*A); Imag = sqrt(-Delta) / (2*A);
        printf ("X1 = (%g)+i(%g) e X2 = (%g)-i(%g)", Real, Imag,
                                                    Real,  Imag);
    }
}
```

Senão {

Real ← $\frac{-B}{2*A}$; Imag ← $\frac{\sqrt{-Delta}}{2*A}$;

Escrever ("X1 = (", Real, ") + i (", Imag, ") e X2 = (", Real, ") - i (", Imag, ")");

}

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
int main () {
    float A, B, C, X1, X2, Delta, Real, Imag;
    scanf ("%f%f%f", &A, &B, &C);
    Delta = pow (B, 2) – 4*A*C;
    if (Delta >= 0) {
        X1 = (-B+sqrt(Delta))/(2*A);  X2 = (-B-sqrt(Delta))/(2*A);
        printf ("X1 = %g e X2 = %g", X1, X2);
    }
    else {
        Real = -B / (2*A); Imag = sqrt(-Delta) / (2*A);
        printf ("X1 = (%g)+i(%g) e X2 = (%g)-i(%g)", Real, Imag, Real, Imag);
    }
    printf ("\n\n"); system ("pause"); return 0;
}
```

Para usar "system"

Para controlar o fechamento da tela de execução

Programa final

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
int main () {
    float A, B, C, X1, X2, Delta, Real, Imag;
    scanf (“%f%f%f”, &A, &B, &C);
    Delta = pow (B, 2) – 4*A*C;
    if (Delta >= 0) {
        X1 = (-B+sqrt(Delta))/(2*A);  X2 = (-B-sqrt(Delta))/(2*A);
        printf (“X1 = %g e X2 = %g”, X1, X2);
    }
    else {
        Real = -B / (2*A); Imag = sqrt(-Delta) / (2*A);
        printf (“X1 = (%g)+i(%g) e X2 = (%g)-i(%g)”, Real, Imag, Real, Imag);
    }
    printf (“\n\n”); system (“pause”); return 0;
}
```

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
int main () {
    float A, B, C, X1, X2, Delta, Real, Imag;
    scanf ("%f%f%f", &A, &B, &C);
    Delta = pow (B, 2) – 4*A*C;
    if (Delta >= 0) {
        X1 = (-B+sqrt(Delta))/(2*A);  X2 = (-B-sqrt(Delta))/(2*A);
        printf ("X1 = %g e X2 = %g", X1, X2);
    }
    else {
        Real = -B / (2*A); Imag = sqrt(-Delta) / (2*A);
        printf ("X1 = (%g)+i(%g) e X2 = (%g)-i(%g)", Real, Imag, Real, Imag);
    }
    printf ("\n\n"); system ("pause"); return 0;
}
```

Os detalhes da formatação de saída dos resultados em princípio não precisam aparecer no algoritmo

Apenas aparecem os elementos a serem escritos

A inclusão da biblioteca da linguagem também não precisa aparecer no algoritmo

As exigências do ambiente também não precisam aparecer no algoritmo

Soma dos termos de uma progressão aritmética:

Algoritmo

```
SomaPA {
        Variáveis Inteiras: a1, r, n, soma, aq, i;

        Ler (a1, n, r);
        soma ← 0; aq ← a1; i ← 1;
        enquanto (i ≤ n) {
                soma ← soma + aq;
                aq ← aq + r;
                i ← i + 1;
        }
        Escrever ("Progressao aritmetica:\n\n", "Primeiro termo: ", a1,
                "\nRazao: ", r, "\nNumero de termos: ", n,
                "\n\nSoma: ", soma);
}
```

```
#include <stdio.h>
#include <stdlib.h>
int main () {
    int a1, r, n, soma, aq, i;
    scanf ("%d%d%d", &a1, &n, &r);
    soma = 0; aq = a1; i = 1;
    while (i <= n) {
        soma = soma + aq;
        aq = aq + r;
        i = i + 1;
    }
}
```

Algoritmo

```
SomaPA {
    Variáveis Inteiras: a1, r, n, soma, aq, i;

    Ler (a1, n, r);
    soma ← 0; aq ← a1; i ← 1;
    enquanto (i ≤ n) {
        soma ← soma + aq;
        aq ← aq + r;
        i ← i + 1;
    }
    Escrever ("Progressao aritmetica:\n\n", "Primeiro termo: ", a1,
        "\nRazao: ", r, "\nNumero de termos: ", n,
        "\n\nSoma: ", soma);
}
```

```
#include <stdio.h>
#include <stdlib.h>
int main () {
    int a1, r, n, soma, aq, i;
    scanf ("%d%d%d", &a1, &n, &r);
    soma = 0; aq = a1; i = 1;
    while (i <= n) {
        soma = soma + aq;
        aq = aq + r;
        i = i + 1;
    }
    printf ("\nProgressao aritmetica:\n\nPrimeiro termo: %d", a1);
    printf ("\nRazao: %d\nNumero de termos: %d", r, n);
    printf ("\n\nSoma: %d", soma);
    printf ("\n\n"); system ("pause"); return 0;
}
```

Novamente, a formatação da saída não precisa aparecer no algoritmo

Inclusive os '\n's poderiam ser dele omitidos

```
Escrever ("Progressao aritmetica:\n\n", "Primeiro termo: ", a1,
        "\nRazao: ", r, "\nNumero de termos: ", n,
        "\n\nSoma: ", soma);
```

Cálculo da integral definida de função com uma variável:

O programa será particularizado para a função

$$f(x) = \log_{10}x + 5$$

No Capítulo sobre Subprogramação, será visto um programa de integrais definidas para várias funções

Algoritmo

```
IntegralTrapézioPrecisão {
        Variáveis Reais: a, b, S1, S2, STrap, Dx, p;
        Variáveis Inteiras: i, n;
        Função Real: f (Parâmetro Real);

        Ler (a, b, p);
        S2 ← 0; n ← 5;
        repetir {
                S1 ← S2; n ← 2*n; Dx ← (b-a)/n; S2 ← 0; i ← 1;
                enquanto (i ≤ n) {
                        STrap ← (Dx * (f(a + (i -1) *Dx) + f(a + i * Dx))) / 2;
                        S2 ← S2 + STrap; i ← i + 1;
                }
        } enquanto (|S2 – S1| > p);
        Escrever ("A integral de f(x) no intervalo [", a, " , ", b,
                "], com precisão ", p, " é ", S2);
}
```

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

double f (double x) {
    return (log10(x) + 5);
}

int main () {
    - - - - -
}
```

Declaração da função $f(x) = \log_{10}x + 5$

double: real de dupla precisão

Algoritmo

```
IntegralTrapézioPrecisão {
    Variáveis Reais: a, b, S1, S2, STrap, Dx, p;
    Variáveis Inteiras: i, n;
    Função Real: f (Parâmetro Real);

    Ler (a, b, p);
    S2 ← 0; n ← 5;
    repetir {
        S1 ← S2; n ← 2*n; Dx ← (b-a)/n; S2 ← 0; i ← 1;
        enquanto (i ≤ n) {
            STrap ← (Dx * (f(a + (i -1) *Dx) + f(a + i*Dx)))/2;
            S2 ← S2 + STrap; i ← i + 1;
        }
    } enquanto (|S2 – S1| > p);
    Escrever ("A integral de f(x) no intervalo [", a, " , ", b,
        "], com precisão ", p, " é ", S2);
}
```

```
int main () {
    int n, i; float a, b, p;
    double S1, S2, STrap, Dx;
    scanf ("%f%f%f", &a, &b, &p);
    S2 = 0; n = 5;
    do {

    } while (fabs (S1-S2) > p);
}
```

fabs: valor absoluto de expressão real

Algoritmo

```
IntegralTrapézioPrecisão {
    Variáveis Reais: a, b, S1, S2, STrap, Dx, p;
    Variáveis Inteiras: i, n;
    Função Real: f (Parâmetro Real);

    Ler (a, b, p);
    S2 ← 0; n ← 5;
    repetir {
        S1 ← S2; n ← 2*n; Dx ← (b-a)/n; S2 ← 0; i ← 1;
        enquanto (i ≤ n) {
            STrap ← (Dx * (f(a + (i -1) *Dx) + f(a + i*Dx))) / 2;
            S2 ← S2 + STrap; i ← i + 1;
        }
    } enquanto (|S2 – S1| > p);
    Escrever ("A integral de f(x) no intervalo [", a, " , ", b,
        "], com precisão ", p, " é ", S2);
}
```

```
int main () {
    int n, i; float a, b, p;
    double S1, S2, STrap, Dx;
    scanf ("%f%f%f", &a, &b, &p);
    S2 = 0; n = 5;
    do {
        S1 = S2; n = 2*n;
        Dx = (b-a)/n; S2 = 0; i = 1;
        while (i <= n) {
            STrap = Dx * (f(a+(i-1)*Dx) + f(a+i*Dx)) / 2;
            S2 = S2 + STrap;
            i = i + 1;
        }
    } while (fabs (S1-S2) > p);

}
```

```
repetir {
    S1 ← S2; n ← 2*n; Dx ← (b-a)/n; S2 ← 0; i ← 1;
    enquanto (i ≤ n) {
        STrap ← (Dx * (f(a + (i -1) *Dx) + f(a + i*Dx))) / 2;
        S2 ← S2 + STrap; i ← i + 1;
    }
} enquanto (|S2 – S1| > p);
```

```
int main () {
    int n, i; float a, b, p;
    double S1, S2, STrap, Dx;
    scanf ("%f%f%f", &a, &b, &p);
    S2 = 0; n = 5;
    do {
        S1 = S2; n = 2*n;
        Dx = (b-a)/n; S2 = 0; i = 1;
        while (i <= n) {
            STrap = Dx * (f(a+(i-1)*Dx) + f(a+i*Dx)) / 2;
            S2 = S2 + STrap;
            i = i + 1;
        }
    } while (fabs (S1-S2) > p);
    printf ("\nA Integral de f(x) no intervalo [%g, %g]", a, b);
    printf (" com precisao %g eh %g", p, S2);
    printf ("\n\n"); system ("pause"); return 0;
}
```

O programa não está amigável

Escrever ("A integral de f(x) no intervalo [", a, " , ", b, "], com precisão ", p, " é ", S2);

4.5 – Produção de um software

PPT 02

- **Produzir um software** é muito mais do que **sair escrevendo** um programa em uma linguagem de programação

- Quando o software é de **grande porte** seu desenvolvimento é bem **complexo** e podem ser necessárias várias **equipes**

- Isso é tema da disciplina **Engenharia de Software**:

  - **Software é um produto de Engenharia**

Etapas para produção de software de grande porte:

- Levantamento de requisitos
- Elaboração do projeto
- Implementação
- Realização de testes
- Entrega e implantação
- Operação e manutenção

Levantamento de requisitos:

- Os **requisitos** definem as **funcionalidades** de um software e as **restrições e limitações** do software que está sendo desenvolvido

- **Requisitos Funcionais**
  - Caracterizam a funcionalidade do software, ou seja, o que ele deve fazer

- **Requisitos Não-Funcionais**
  - Caracterizam restrições e limitações sobre a funcionalidade Exemplo: desempenho, segurança, etc.

Problemas para levantar requisitos:

- Clientes erram e mudam de idéia
  - Deve-se estabelecer com precisão o **escopo** do software
- Comunicação pode falhar
  - Deve-se fazer uso de **diagramas** e **documentação** para melhorar o entendimento; **formalidade** necessária
- Não é uma atividade puramente técnica
  - Requer habilidade de **relacionamento humano**

Elaboração do projeto:

- **Arquitetura do software:** pode ser a integração de vários módulos

- Escolha dos **métodos** e elaboração dos **algoritmos**

Implementação:

- Tradução do algoritmo para uma **linguagem de programação**
- Preocupações com alocação de **memória**, tipos de **dados**, **desempenho**, etc.
- Eliminação dos **erros de sintaxe**

Realização de testes:

- Verificação da implementação correta dos requisitos levantados
- Primeiramente, **cada módulo** é testado **individualmente**
- Depois os módulos são **integrados** e sua composição é testada

Tipos de teste:

- Testes com entradas **corretas**: a saída deve estar **correta**
- Testes com entradas **errôneas**: a saída deve ser **irregular**
- Testes de **desempenho**: tempo de resposta, testes com dados de entrada volumosos

- Os testes quase sempre determinam **correções** no programa
- Deve-se elaborar um **relatório** final dos testes, com resultados obtidos
- É importante o envolvimento com o **usuário**

Entrega e implantação:

- O software deve ser **instalado** em ambiente de **produção**
- Usuários devem ser **treinados**
- O **ambiente** de produção deve ser **configurado**
- Principal **propósito** desta fase:
  - Realizar **testes de aceitação** (certificar de que o software **satisfaz** os requisitos dos **usuários**)

Operação e manutenção:

- **Fase de operação:** o software passa a ser **utilizado** de fato em ambiente de produção
- **Manutenções corretivas:** correção de erros e falhas
- **Manutenções adaptativas:** (alterações no meio externo)
  - Novas versões de plataforma
  - Mudanças nas leis, políticas, etc.
- **Manutenções evolutivas:** novas funcionalidades
- **Manutenções preventivas:** garantir maior confiabilidade e possibilidade de manutenção

É importante seguir boas práticas de programação para o programa ficar fácil de ser entendido e alterado

Computadores, Algoritmos e Programas: Manual Introdutório de Ensino

Capítulo V

Linguagens para Algoritmos

Capítulo V – Linguagens para Algoritmos

5.1 – Pseudocódigo

5.2 – Fluxogramas

5.3 – Linguagem de programação aplainada

5.1 – Pseudocódigo

- No Capítulo IV, a linguagem utilizada para os algoritmos muito se **assemelha** a conhecidas linguagens de programação como **C e Pascal**

- Por isso ela se chama **pseudocódigo**

- Os comandos estão em **Português**

- Um algoritmo é expresso por seu **nome**, seguido de seu **corpo** delimitado pelas chaves **“{”** e **“}”**

- **No corpo:**
  - Declarações de variáveis
  - Comandos executáveis

- **Comandos executáveis:**
  - Comandos de atribuição
  - Comandos de entrada e saída
  - Comandos condicionais
  - Comandos repetitivos

Comandos condicionais e repetitivos possuem escopos

Esses escopos podem vir delimitados pelas chaves

Dentro dos escopos podem figurar quaisquer de todos esses comandos vistos

Exemplo: Algoritmo da equação do 2º grau:

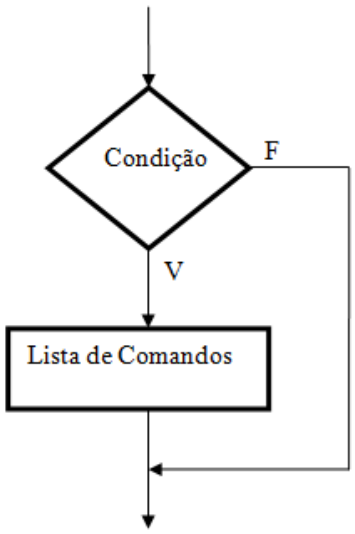
Livro 01

```
RaizesEquação2Grau {
        Variáveis Reais: A, B, C, X1, X2, Delta, Real, Imag;

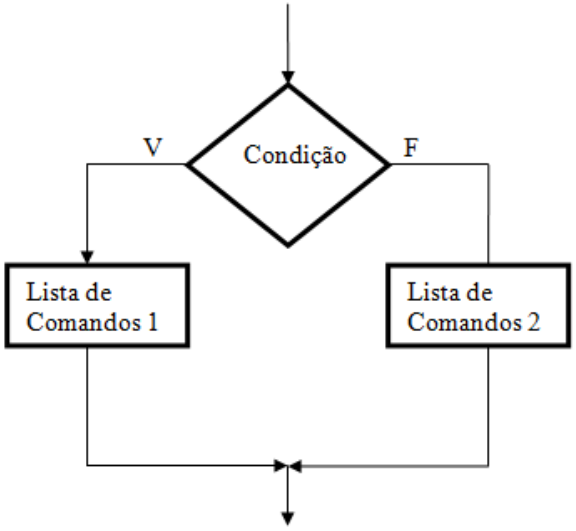
        Ler (A, B, C);
        Delta ← B^2 - 4*A*C;
        Se (Delta ≥ 0) {
                X1 ← (-B + √Delta) / (2*A) ; X2 ← (-B - √Delta) / (2*A) ;
                Escrever ("X1 = ", X1, " e X2 = ", X2);
        }
        Senão {
                Real ← -B / (2*A) ; Imag ← √(-Delta) / (2*A) ;
                Escrever ("X1 = (", Real, ") + i (", Imag, ") e X2 = (", Real,
                        ") - i (", Imag, ")");
        }
}
```

5.2 – Fluxogramas

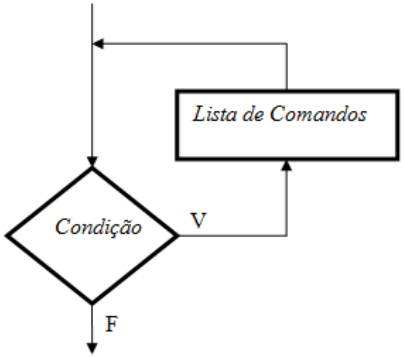
- No Tópico 2.1, foram usados **diagramas de blocos** denominados **fluxogramas**, para descrever comandos condicionais e comandos repetitivos:



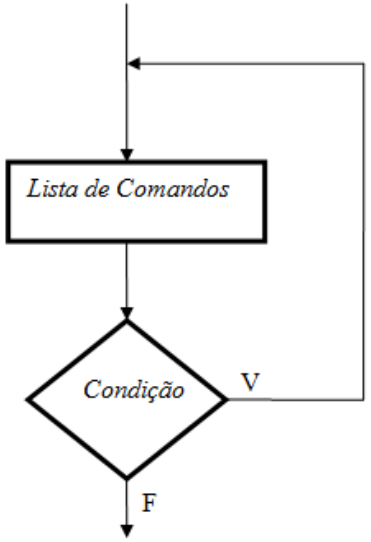
se



se-senão

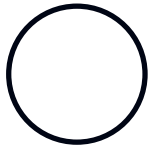


enquanto



repetir-enquanto

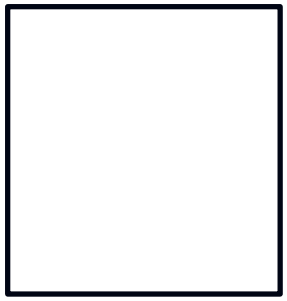
Blocos dos fluxogramas:



Círculo: início e final do algoritmo



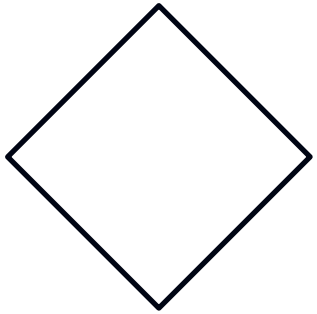
Cartão: entrada de dados



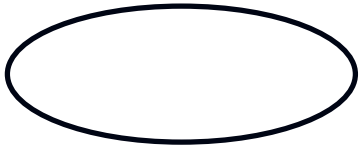
Retângulo: comandos (atribuição ou chamada de subprograma) a serem executados sequencialmente



Folha de papel: saída de resultados



Losango: decisão por um de dois caminhos alternativos



Elipse: decisão por um de vários caminhos alternativos

- **Fluxogramas** podem ser considerados como uma **linguagem** para descrição de **algoritmos**

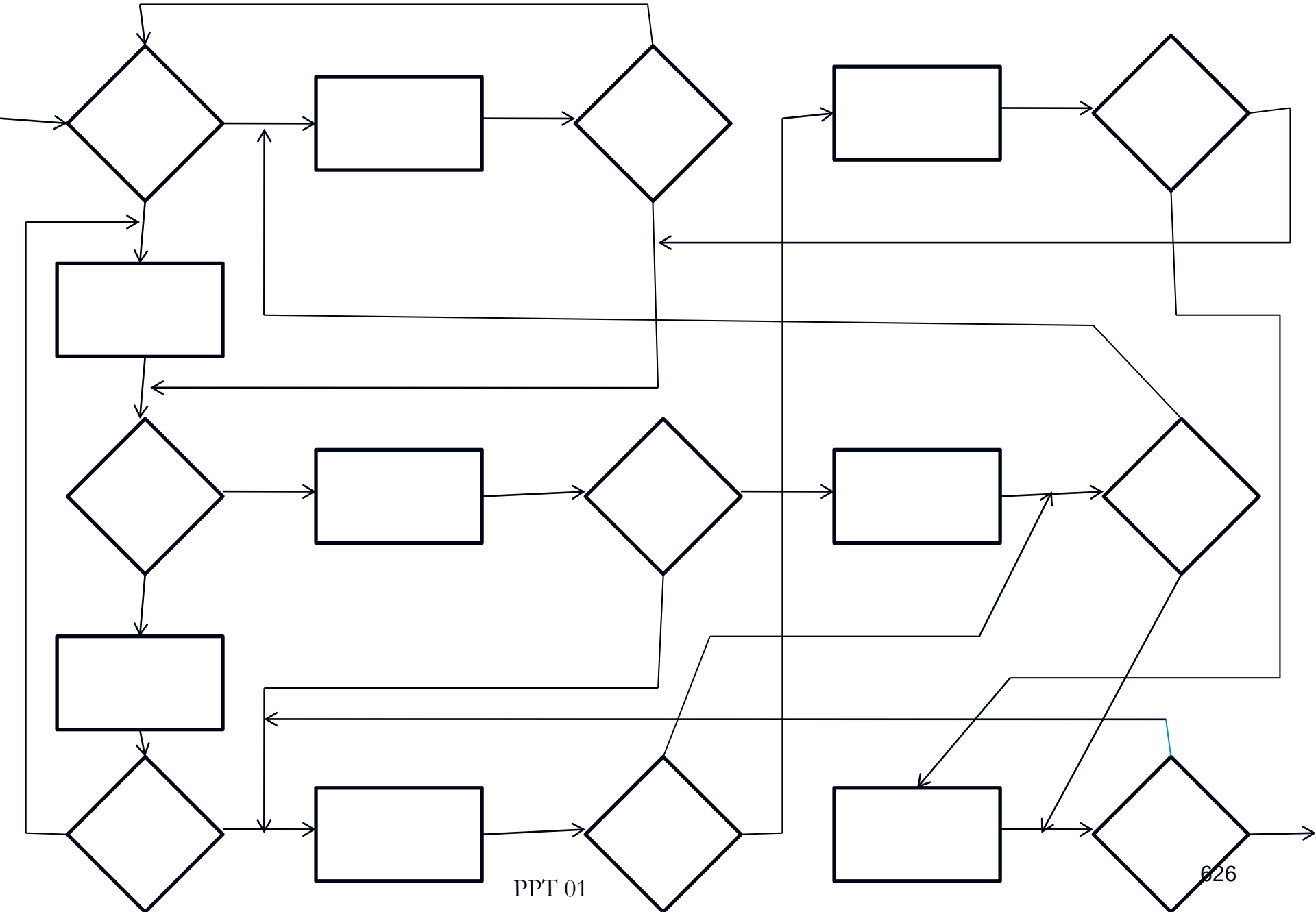
- Foram **muito usados** quando surgiram as **primeiras linguagens** de programação

- Durante **muito tempo** foram usados para esse fim

- No entanto, seu **uso indisciplinado** levava à construção de **algoritmos confusos** e difíceis de serem corrigidos

Exemplo:

Que macarronada ou que teia de aranha!!!



- Em defesa de uma **programação** mais bem **estruturada**, os fluxogramas foram substituídos pelo **pseudocódigo**

- Durante muito tempo, os fluxogramas foram **execrados na Ciência da Computação**

- No entanto, eles são providenciais para **explicar o funcionamento** de comandos mais complexos como os condicionais e os repetitivos

- Para determinados fins, principalmente didáticos, **fluxogramas** ainda são usados para descrever **algoritmos**

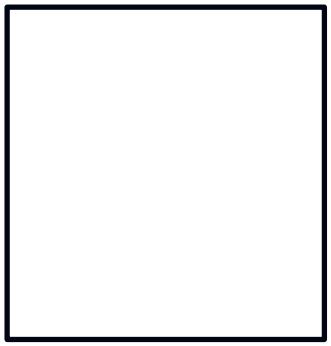
- **Programação estruturada:** forma de programação que estabelece uma **disciplina** de desenvolvimento de algoritmos

- **Objetivo:** facilitar a compreensão de programas através de **restrição** no número de **mecanismos de controle** da execução de programas

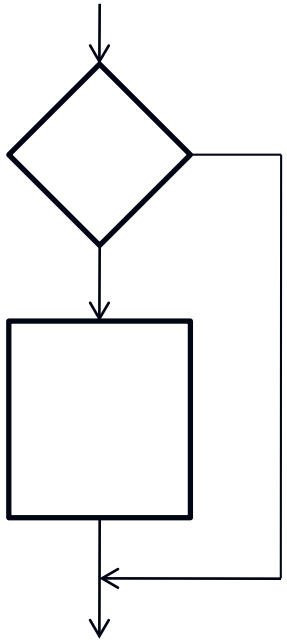
- A programação estruturada **admite** apenas **algumas construções** para fluxogramas, a saber:

Construções válidas em programação estruturada:

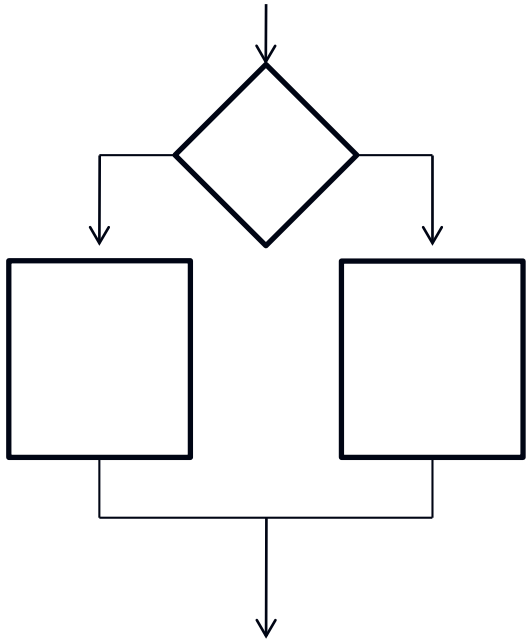
Sequência



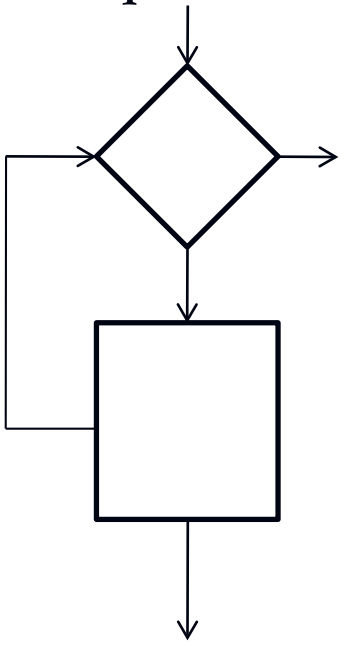
Construção Se



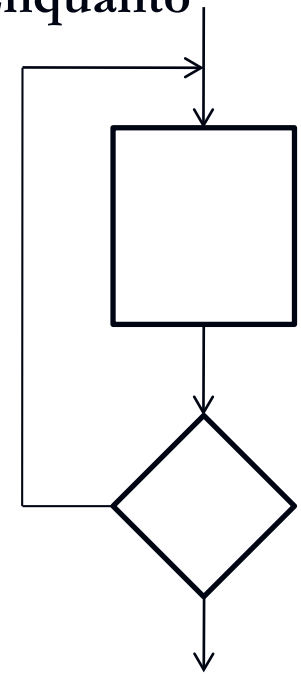
Construção Se-Senão



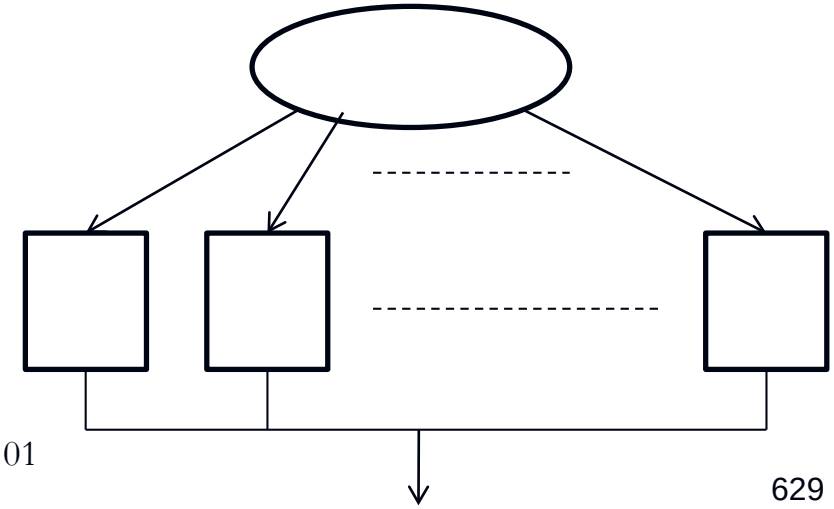
Construção Enquanto



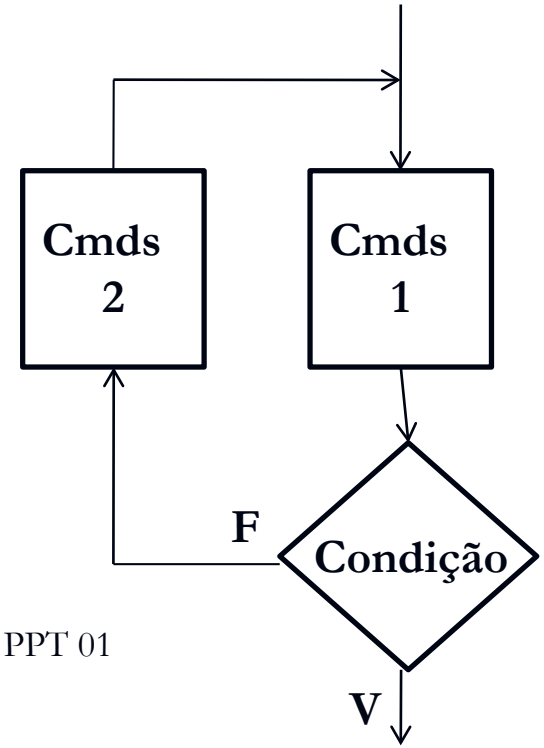
Construção Repetir-Enquanto



Construção Seleção



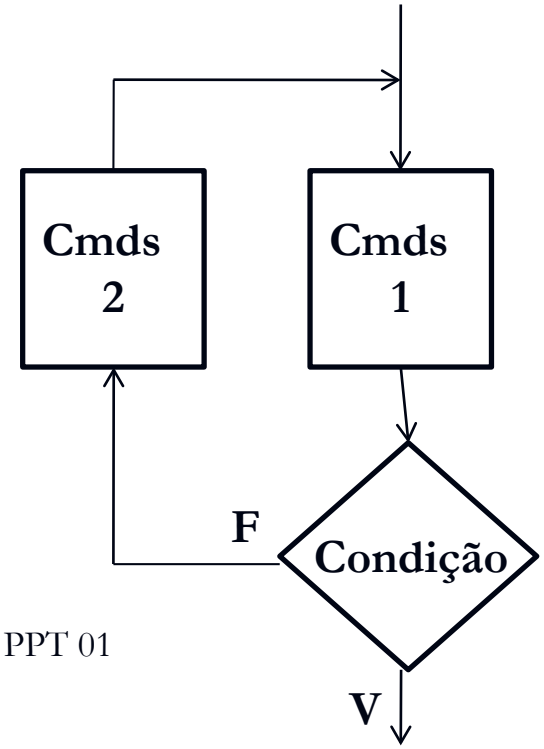
Exemplo de construção simples mas não-válida:



PPT 01

Não corresponde a nenhuma das construções válidas

Exemplo de construção simples mas não-válida:



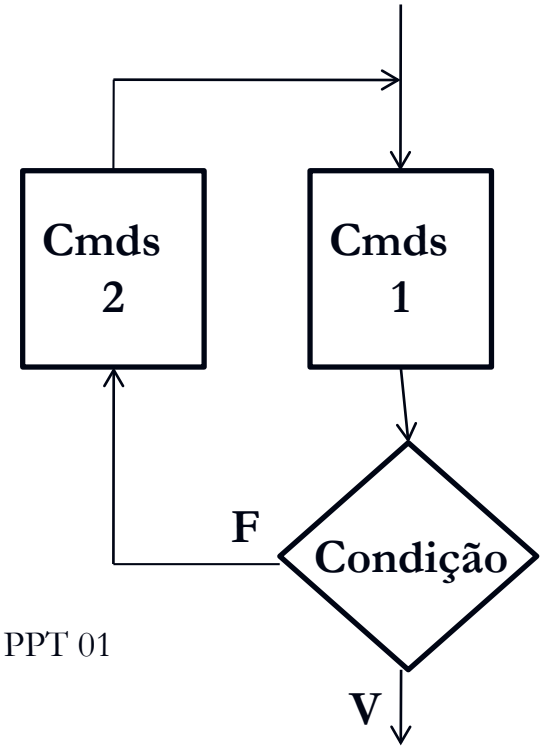
PPT 01

Possível pseudocódigo:

```
Rot1:   Cmds 1;
        Se (Condição = Verdade)
                Vá para Rot2;
        Senão {
                Cmds 2;
                Vá para Rot 1;
        }
Rot 2:  - - - - - - - -
```

Uso de rótulos e do comando "Vá para" (famoso comando "Goto")

Exemplo de construção simples mas não-válida:



PPT 01

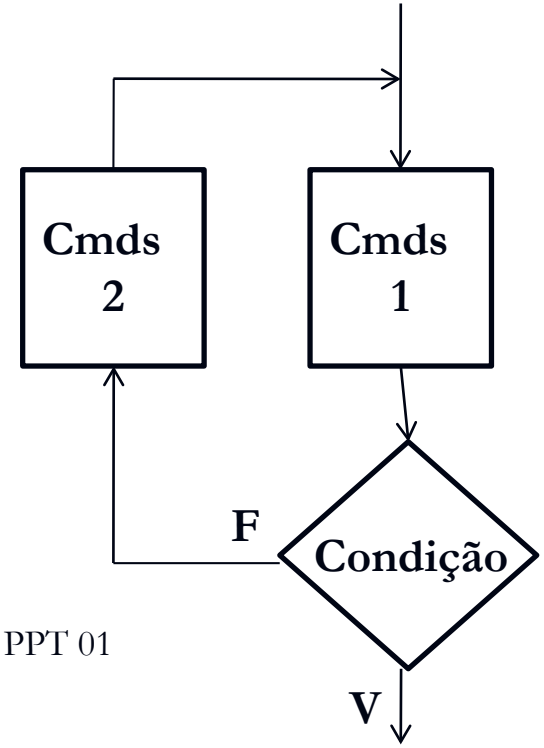
Possível pseudocódigo:

```
Rot1:   Cmds 1;
        Se (Condição = Verdade)
                Vá para Rot2;
        Senão {
                Cmds 2;
                Vá para Rot 1;
        }
Rot 2:  - - - - - - - -
```

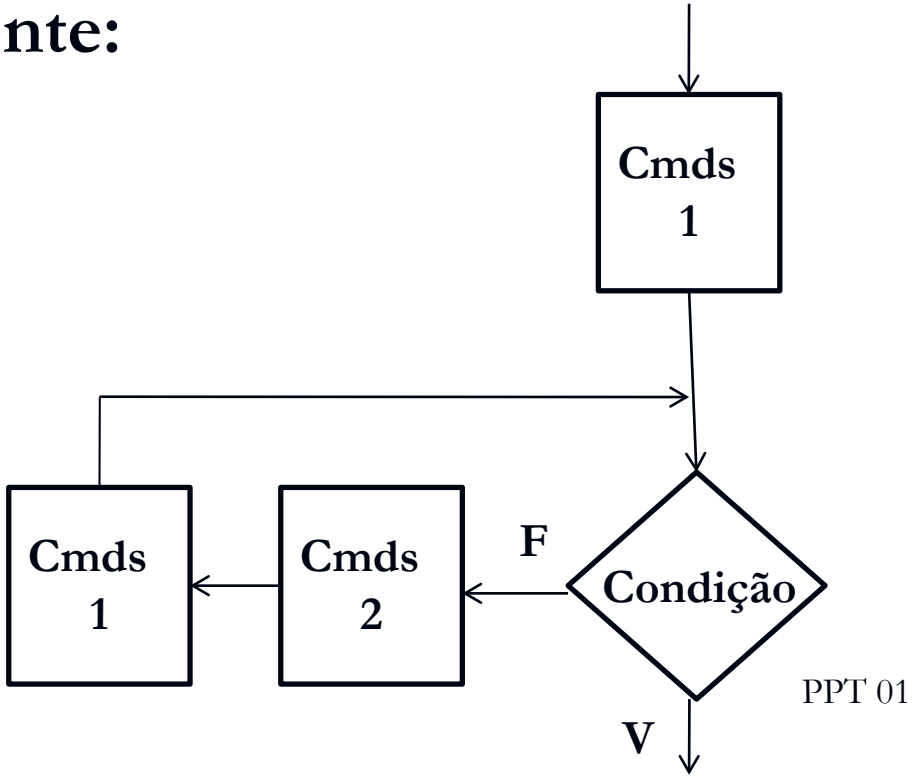
Comando "Goto": seu uso indiscriminado produz algoritmos desestruturados

Não poderá ser usado em CES-10

Fluxograma válido equivalente:



PPT 01



PPT 01

A maioria dos fluxogramas desestruturados podem ser transformados em estruturados

Pseudocódigo:

```
Cmds 1;
Enquanto (Condição = Falsa) {
        Cmds 2;
        Cmds 1;
}
```

Exemplo: equação do 2º grau

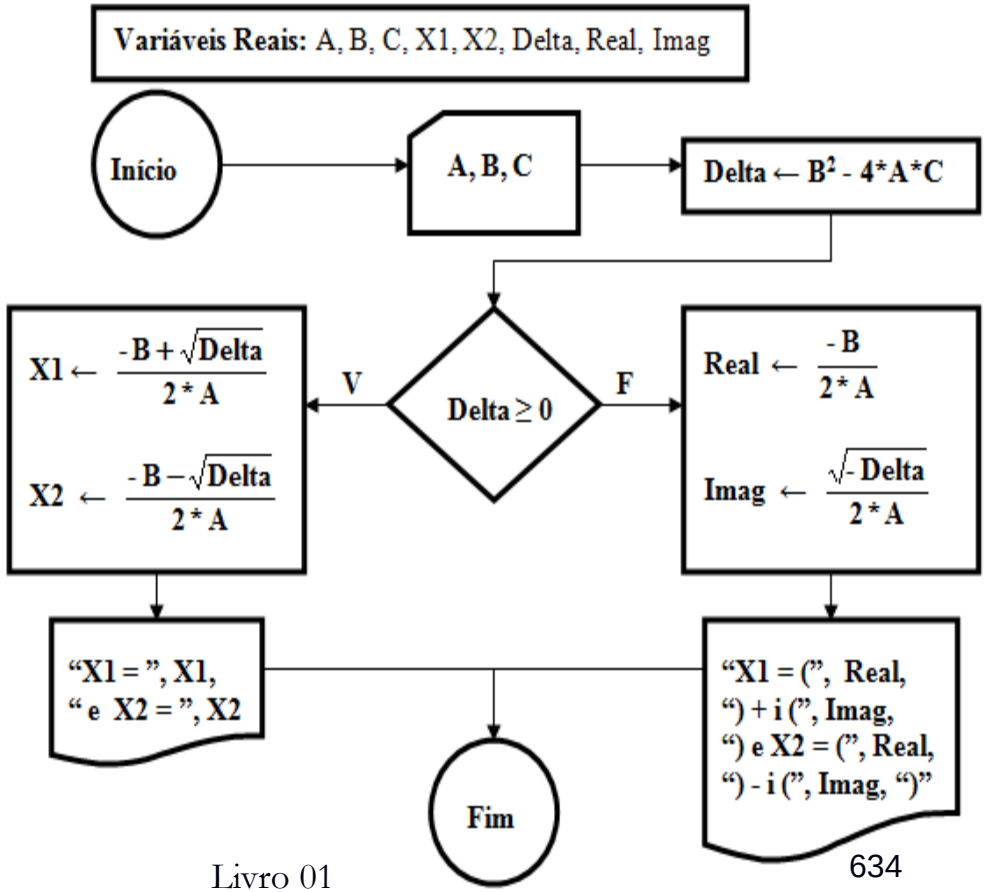
Livro 01

```
RaizesEquação2Grau {
        Variáveis Reais: A, B, C, X1, X2, Delta, Real, Imag;

        Ler (A, B, C);
        Delta ← B² - 4*A*C;
        Se (Delta ≥ 0) {
                X1 ← (-B + √Delta) / (2*A) ; X2 ← (-B - √Delta) / (2*A) ;
                Escrever ("X1 = ", X1, " e  X2 = ", X2);
        }
        Senão {
                Real ← -B / (2*A) ; Imag ← √(-Delta) / (2*A) ;
                Escrever ("X1 = (", Real, ") + i (", Imag, ") e X2 = (", Real,
                        ") - i (", Imag, ")");
        }
}
```

Pseudocódigo

Fluxograma



Exemplo: soma dos elementos de uma PA

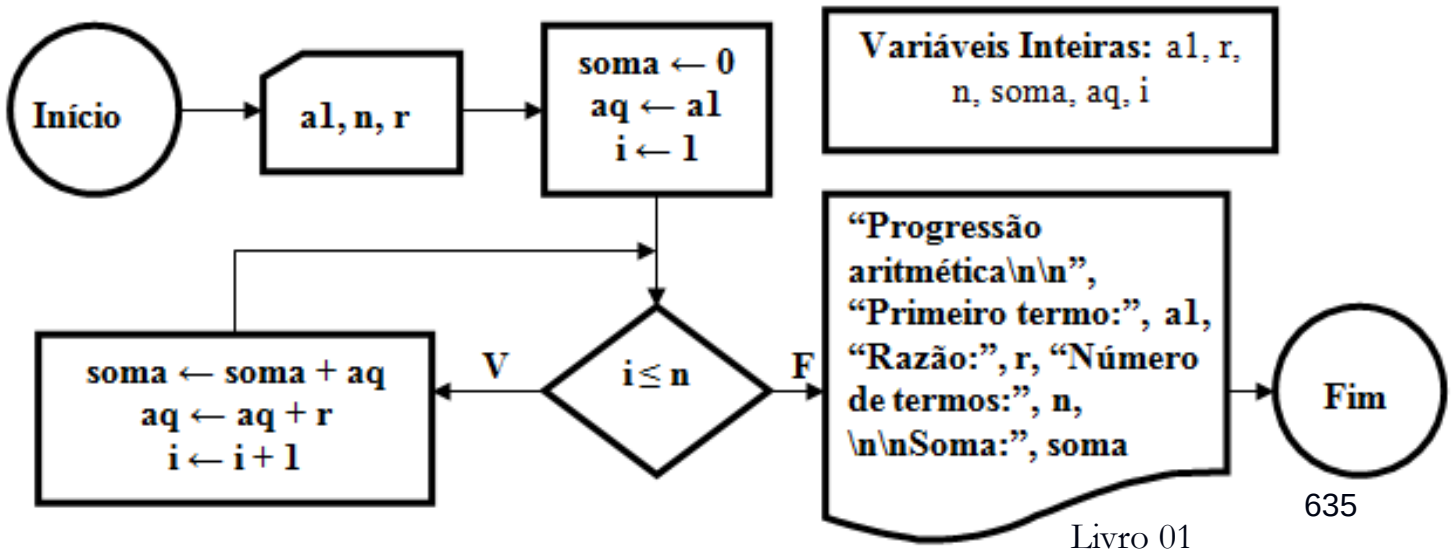
```
SomaPA {
        Variáveis Inteiras: a1, r, n, soma, aq, i;

        Ler (a1, n, r);
        soma ← 0; aq ← a1; i ← 1;
        enquanto (i ≤ n) {
                soma ← soma + aq;
                aq ← aq + r;
                i ← i + 1;
        }
        Escrever ("Progressao aritmetica:\n\n", "Primeiro termo: ", a1,
                "\nRazao: ", r, "\nNumero de termos: ", n,
                "\n\nSoma: ", soma);
}
```

Pseudocódigo

Livro 01

Fluxograma



Livro 01

Exemplo: integral definida usando Regra do Trapézio

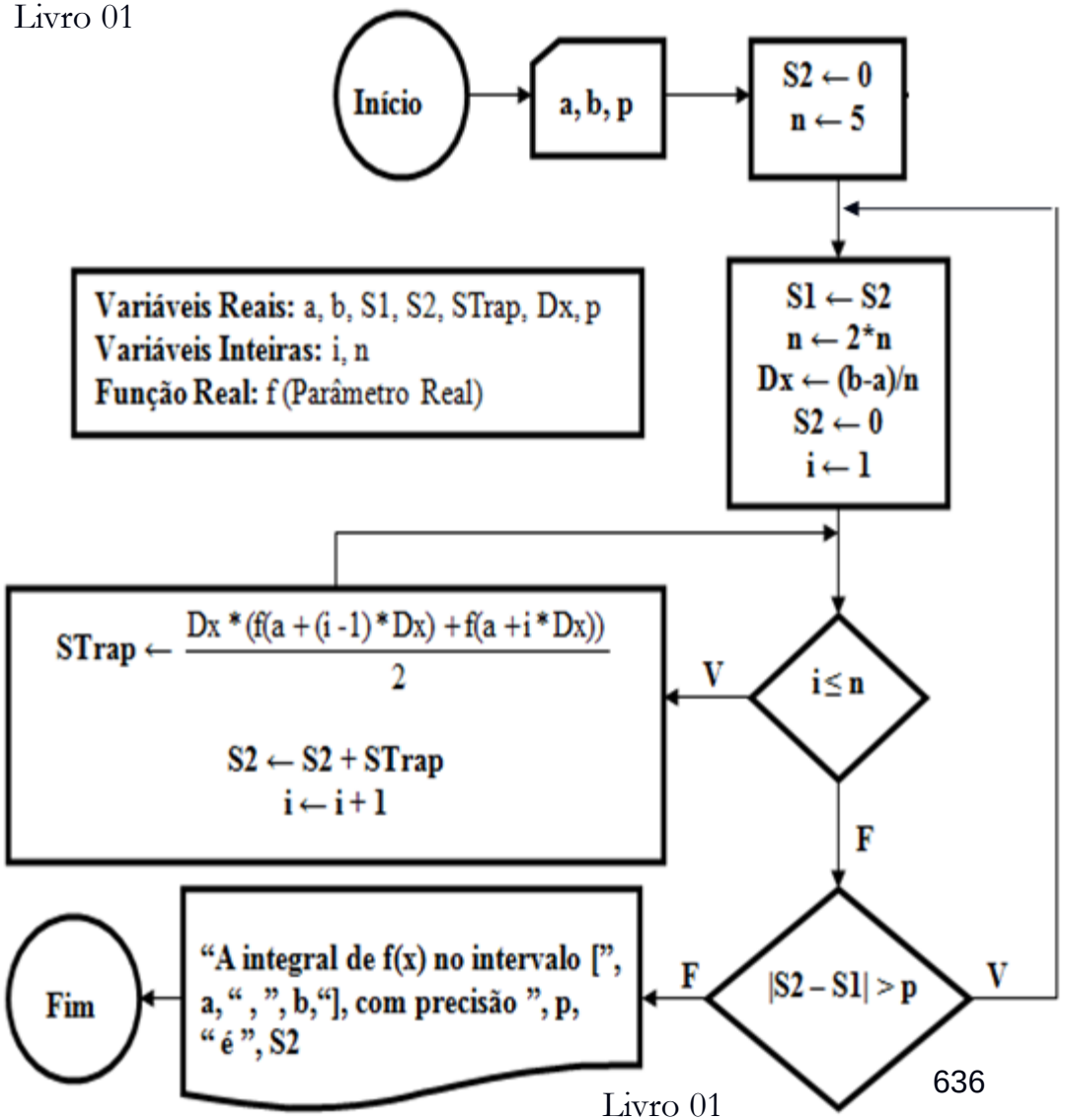
Pseudocódigo

```
IntegralTrapézioPrecisão {
    Variáveis Reais: a, b, S1, S2, STrap, Dx, p;
    Variáveis Inteiras: i, n;
    Função Real: f (Parâmetro Real);

    Ler (a, b, p);
    S2 ← 0; n ← 5;
    repetir {
        S1 ← S2; n ← 2*n; Dx ← (b-a)/n; S2 ← 0; i ← 1;
        enquanto (i ≤ n) {
            STrap ← (Dx * (f(a + (i -1) * Dx) + f(a + i * Dx))) / 2;
            S2 ← S2 + STrap; i ← i + 1;
        }
    } enquanto (|S2 – S1| > p);
    Escrever ("A integral de f(x) no intervalo [", a, " , ", b,
        "], com precisão ", p, " é ", S2);
}
```

Livro 01

Fluxograma



Livro 01

5.3 – Linguagem de programação aplainada

- Um **algoritmo** deve ser **traduzido** para a **linguagem** escolhida para sua implementação no computador

- A tradução de um **fluxograma** para a **Linguagem C** não é muito simples

- Requer encontrar uma **correspondência** entre as **construções do fluxograma** e os **comandos** da referida linguagem

- Já a tradução de um **pseudocódigo** para a **Linguagem C** é mais direta

- Requer **tradução** de alguns comandos para o **Inglês**

- Alguns **símbolos** do pseudocódigo devem ser substituídos por seus correspondentes em **C**

- **Exemplos:**

  - A atribuição **"←"** deve ser substituída por **"="**
  - As comparações **"="** e **"≠"** devem ser substituídas respectivamente por **"=="** e **"!="**
  - Etc.

- Então, por que não escrever o algoritmo **diretamente** na linguagem de programação escolhida?

- Acontece que alguns comandos dessas linguagens apresentam **obscuridade** ou **incômodos** para a confecção de algoritmos

- **Exemplos da Linguagem C:**

  - **Formatação** dos comandos de **entrada e saída** (**%f**, etc.)
  - Comandos **switch-case** (condicionais de **várias alternativas** – vistos nos próximos capítulos) que, se não usados com um certo comando **break**, podem produzir **código desestruturado**

- Nesta disciplina, a linguagem para algoritmos adotada será a **Linguagem C aplainada**:

  - Linguagem C **ligeiramente modificada**, para evitar os referidos incômodos

- A maioria dos **comandos** e as **declarações** de variáveis serão escritas em **C**

- Comandos de atribuição que, em pseudocódigo, têm a seguinte forma geral:

Variável ← Expressão;

Passam a ser aqui expressos por

Variável = Expressão;

- Comandos condicionais e repetitivos em pseudocódigo:
  - **se** (condição) comandos
  - **se** (condição) comandos 1 **senão** comandos 2
  - **enquanto** (condição) comandos
  - **repetir** comandos **enquanto** (condição);
- Em Linguagem C aplainada (tal como em C):
  - **if** (condição) comandos
  - **if** (condição) comandos 1 **else** comandos 2
  - **while** (condição) lista de comandos
  - **do** comandos **while** (condição);

Os delimitadores de escopos continuarão a ser as chaves ("{" e "}")

- Para muitos algoritmos,

  - Especificação do **formato de entrada de dados**, para as variáveis lidas

  - Especificação do **layout dos resultados**, escritos na tela ou nos arquivos em disco

não têm relevância e podem ser omitidas

- Em C aplainada, os comandos de entrada e de saída serão expressos respectivamente pelas formas:

  read (Variável, Variável, . . . , Variável);

  write (Elemento, Elemento, . . . , Elemento);

  Elemento pode ser uma expressão ou uma cadeia de caracteres entre aspas **(“ ”)**

- **Comandos condicionais de várias alternativas** serão escritos seguindo o esquema de **Pascal**, o que será abordado em capítulos posteriores

Exemplo: equação do 2º grau

Pseudocódigo

```
RaizesEquação2Grau {
        Variáveis Reais: A, B, C, X1, X2, Delta, Real, Imag;

        Ler (A, B, C);
        Delta ← B² - 4*A*C;
        Se (Delta ≥ 0) {
                X1 ← (-B + √Delta) / 2*A ; X2 ← (-B - √Delta) / 2*A ;
                Escrever ("X1 = ", X1, " e X2 = ", X2);
        }
        Senão {
                Real ← -B / 2*A ; Imag ← √(-Delta) / 2*A ;
                Escrever ("X1 = (", Real, ") + i (", Imag, ") e X2 = (", Real,
                        ") - i (", Imag, ")");
        }
}
```

Livro 01

C Aplainada

```
RaizesEquação2Grau {
        float A, B, C, X1, X2, Delta, Real, Imag;

        read (A, B, C);
        Delta = pow (B, 2) - 4*A*C;
        if (Delta >= 0) {
                X1 = (-B + sqrt (Delta)) / (2*A);
                X2 = (-B - sqrt (Delta)) / (2*A);
                write ("X1 = ", X1, " e X2 = ", X2);
        }
        else {
                Real = -b / (2*A); Imag = sqrt (-Delta) / (2*A);
                write ("X1 = (", Real, ") + i (", Imag, ") e X2 = (", Real,
                        ") - i (", Imag, ")");
        }
}
```

Livro 01

Exemplo: soma dos elementos de uma PA

Pseudocódigo

```
SomaPA {
        Variáveis Inteiras: a1, r, n, soma, aq, i;

        Ler (a1, n, r);
        soma ← 0; aq ← a1; i ← 1;
        enquanto (i ≤ n) {
                soma ← soma + aq;
                aq ← aq + r;
                i ← i + 1;
        }
        Escrever ("Progressao aritmetica:\n\n", "Primeiro termo: ", a1,
                "\nRazao: ", r, "\nNumero de termos: ", n,
                "\n\nSoma: ", soma);
}
```

Livro 01

C Aplainada

```
SomaPA {
        int a1, r, n, soma, aq, i;

        read (a1, n, r);
        soma = 0; aq = a1; i = 1;
        while (i <= n) {
                soma = soma + aq;
                aq = aq + r;
                i = i + 1;
        }
        write ("Progressao aritmetica:\n\n", "Primeiro termo: ", a1,
                "\nRazao: ", r, "\nNumero de termos: ", n,
                "\n\nSoma: ", soma);
}
```

Livro 01

Exemplo: integral definida usando Regra do Trapézio

Pseudocódigo

```
IntegralTrapézioPrecisão {
        Variáveis Reais: a, b, S1, S2, STrap, Dx, p;
        Variáveis Inteiras: i, n;
        Função Real: f (Parâmetro Real);

        Ler (a, b, p);
        S2 ← 0; n ← 5;
        repetir {
                S1 ← S2; n ← 2*n; Dx ← (b-a)/n; S2 ← 0; i ← 1;
                enquanto (i ≤ n) {
                        STrap ← (Dx * (f(a + (i -1) * Dx) + f(a + i * Dx))) / 2;
                        S2 ← S2 + STrap; i ← i + 1;
                }
        } enquanto (|S2 – S1| > p);
        Escrever ("A integral de f(x) no intervalo [", a, " , ", b,
                "], com precisão ", p, " é ", S2);
}
```

Livro 01

C Aplainada

```
IntegralTrapézioPrecisão {
        float a, b; double S1, S2, STrap, Dx, p;
        int i, n;
        double f (float);

        read (a, b, p);
        S2 = 0; n = 5;
        do {
                S1 = S2; n = 2*n; Dx = (b-a)/n; S2 = 0; i = 1;
                while (i <= n) {
                        STrap = Dx * (f(a + (i -1) * Dx) + f(a + i * Dx)) / 2;
                        S2 ← S2 + STrap; i ← i + 1;
                }
        } while (fabs (S2 – S1) > p);
        write ("A integral de f(x) no intervalo [", a, " , ", b,
                "], com precisão ", p, " eh ", S2);
}
```

Livro 01

Exercícios 5.3:

1. Desenhar um **fluxograma** para o seguinte algoritmo em C aplainada, destinado a calcular o **MDC** de um par de números:


```
MDC {
        int a, b, aux;

        write ("Calculo de MDC\n");
        write ("Par de numeros: ");
        read (a, b);
        a = abs (a);  b = abs (b);
        while  (b > 0) {
                aux = a;
                a = b;
                b = aux % b;
        }
        write ("MDC: ", a);
}
```

2. Desenhar um fluxograma para o seguinte algoritmo em C aplainada, destinado a encontrar os números perfeitos entre 1 e n:

Número perfeito é aquele cuja soma de seus divisores próprios é igual a si

Um número não é divisor próprio de si mesmo

Exemplos:

6 = 1 + 2 + 3
28 = 1 + 2 + 4 + 7 + 14

```
NumerosPerfeitos {
        long n, i, div, soma;

        write  ("Digite um numero inteiro positivo: ");
        read (n);
        write ("Numeros perfeitos entre 1 e: ", n);
        i = 1;
        while (i<=n) {
                soma = 0; div = 1;
                while (div * 2 <= i) {
                        if  (i % div == 0)
                        soma = soma + div;
                        div = div + 1;
                }
                if  (soma == i) write (i);
                i = i + 1;
        }
}
```

3. Na Seção 2.2.3 foi apresentado um algoritmo em C aplainada para calcular a integral definida de uma função, usando a **Regra do Trapézio**

- Escrever outro algoritmo para resolver esse problema, trocando a referida regra pela **Regra do Retângulo**:

  - Em vez de aproximar as sub-áreas para trapézios, aproxima-as para **retângulos**. A base deles é **Δx** e a altura é dada pela ordenada da função no **meio do sub-intervalo**

Livro 01

Computadores, Algoritmos e Programas: Manual Introdutório de Ensino

Capítulo VI

Propriedades dos Bons Algoritmos

Capítulo VI – Propriedades dos Bons Algoritmos

6.1 – Tempo finito de execução

6.2 – Comandos definidos

6.3 – Comandos efetivos

6.4 – Início e término dos algoritmos

6.5 – Entrada de dados e saída de resultados

6.6 – Algoritmos amigáveis

6.7 – Algoritmos bem estruturados, legíveis e de fácil correção

6.8 – Algoritmos tolerantes a falhas

6.9 – Outros ingredientes

Algoritmo: sequência finita e ordenada de comandos **executáveis e não ambíguos**, que levam à aplicação de um **método** para a execução de uma tarefa ou resolução de um problema

- Não é qualquer **sequência de comandos** que pode ser considerada como algoritmo
- Não é qualquer **comando** que pode pertencer a essa sequência
- Existem algumas **características obrigatórias** e outras **recomendáveis** que os algoritmos devem ter

6.1 – Tempo de execução finito

- O tempo de execução de um algoritmo deve ser **finito** para **qualquer entrada de dados**

- **Exemplo:** algoritmos de tempo infinito

Livro 01

```
Algoritmo1 {
        int n, i, m;
                .
                .
        read (n, i); m = 0;
        while (m <= n) {
                m = m + i;
        }
                .
                .
}
```

Se o valor lido de n > 0 e de i < 0: tempo infinito

```
Algoritmo2 {
        int n, i, x;
                .
                .
        read (n); i =1; x = 1;
        while (i <= n) {
                x = x * i;
        }
                .
                .
}
```

Se o valor lido de n > 0: tempo infinito

Aqui, provavelmente o programador se esqueceu de mudar o valor de i, no while

6.2 – Comandos definidos

- Todos os comandos de um algoritmo devem ser perfeitamente **definidos**, ou seja, sem **ambiguidade** ou **imprecisão**

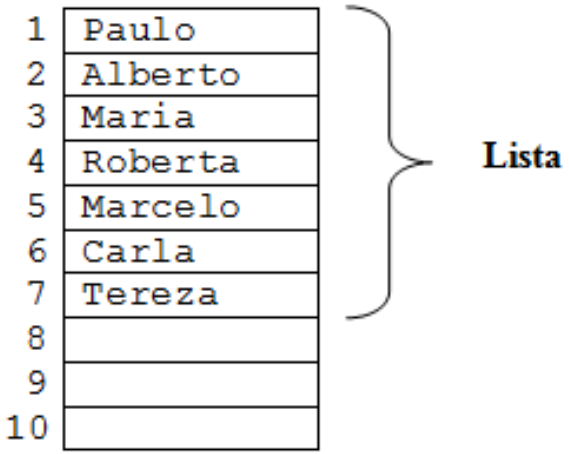
- **Exemplo:** comandos indefinidos e opções para transformá-los em comandos definidos:

Comandos indefinidos	Comandos definidos
se ($\|x - y\|$ é pequeno) $x \leftarrow y$	se ($\|x - y\| \leq 0.001$) $x \leftarrow y$
Colocar 1 pitada de sal	Colocar 2 g de sal

Livro 01

Exemplo: comando com **interpretação ambígua**

Seja a variável indexada Lista desta figura:

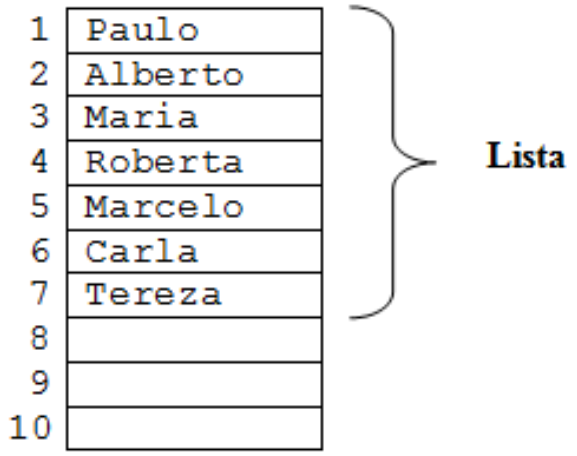


Livro 01

Obs.: variável indexada é o assunto central de um dos próximos capítulos

- O comando **(Inserir "Marcos" na posição 4)** pode ter as seguintes interpretações:
  - Trocar o nome em **Lista[4]** (**Roberta**) por **Marcos**
  - Deslocar os nomes em **Lista[4]**, **[5]**, **[6]** e **[7]** para as posições 5, 6, 7 e 8, respectivamente, e guardar **Marcos** na posição 4

Inserir “Marcos” na posição 4



Livro 01

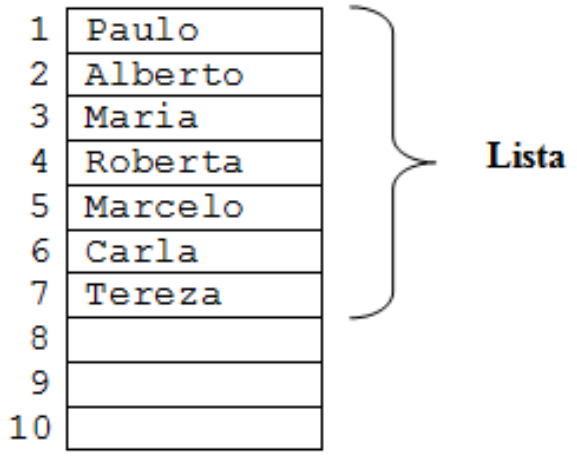
- Se a interpretação for

Trocar o nome em **Lista[4]** (**Roberta**) por **Marcos**

Então, mais definido é o comando:

Lista[4] = “Marcos”;

Inserir “Marcos” na posição 4



Livro 01

- Se a interpretação for

Deslocar os nomes em **Lista[4]**, **[5]**, **[6]** e **[7]** para as posições 5, 6, 7 e 8, respectivamente, e guardar **Marcos** na posição 4

Então, mais definidos são os comandos:

```
i = 7;
while (i >= 4) {Lista[i+1] = Lista[i]; i = i-1;}
Lista[4] = “Marcos”;
```

6.3 – Comandos efetivos

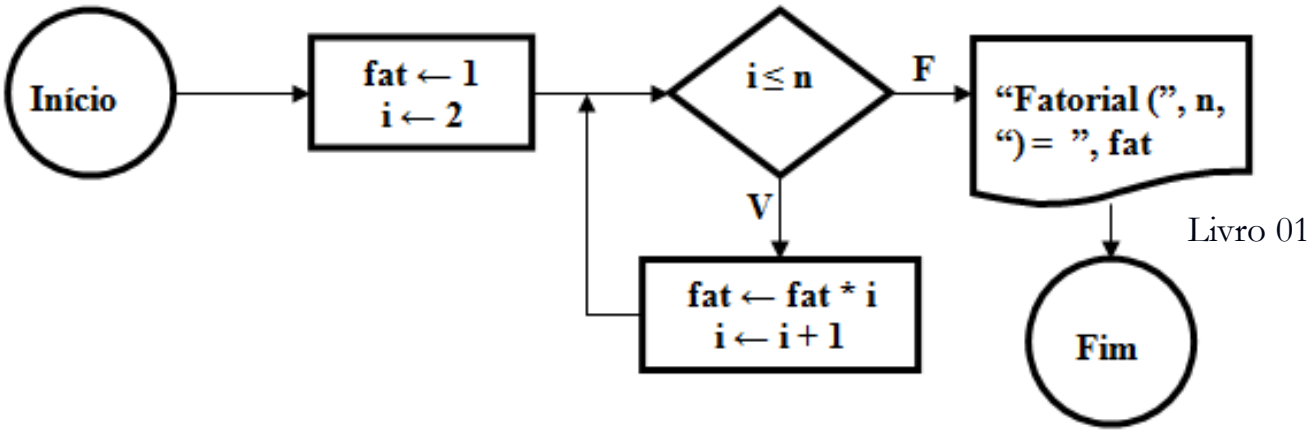
- Todos os comandos de um algoritmo devem ser **passíveis de execução**

- **Exemplo:** comando não efetivo:

  Se chover na próxima semana, esta semana vou à praia

- Outra exigência de um comando efetivo é a necessidade de as **variáveis usadas** por ele já estarem **inicializadas**

- **Exemplo:** seja o fluxograma



Livro 01

A variável **n** não recebe nenhum valor antes de ser usada no teste da condição, logo não é possível saber qual dos caminhos será tomado

6.4 – Início e término dos algoritmos

- Todo algoritmo deve ter um e um só **ponto inicial** e pelo menos um **ponto final**

- **Fluxogramas** tem potencial para **violar** essa exigência

No pseudocódigo e linguagens aplainadas:

- O **ponto inicial** é o **primeiro comando** executável da lista de comandos

- A posição após o último comando da lista é um **ponto final implícito**

- Tais linguagens costumam ter comandos tais como **encerrar**, que, independentemente de sua posição no algoritmo, **finalizam a execução**

6.5 – Entrada de dados e saída de resultados

- Há algoritmos que **não necessitam de dados externos** para serem executados com sucesso
- **Exemplo:** exibir na tela do vídeo um desenho específico
- As **medidas** desse desenho podem aparecer no algoritmo como **constantes** no meio de seus comandos
- No entanto, todo bom algoritmo deve **produzir resultados**

Formas de um algoritmo apresentar resultados:

- Exibi-los na **tela do vídeo** ou em algum equipamento de saída do computador (**impressora**, arquivo em **disco**, **alto-falante**, etc.)

- Em algoritmos compostos de **módulos**, um deles pode produzir valores a serem usados por **outros módulos** no mesmo algoritmo

- Tais valores podem ser armazenados em **variáveis** dentro da memória principal, para serem rapidamente utilizados pelos **módulos solicitantes**

Exemplo: algoritmo para resolver **sistemas de equações** lineares

- Ele pode ser usado para resolver problemas maiores, tais como:
  - Sistemas não lineares
  - Ajustes de curvas
  - Equações diferenciais parciais
  - Etc.

6.6 – Algoritmos amigáveis

- **Algoritmos amigáveis** são aqueles que indicam ao **operador** o que deseja dele receber para dar continuidade à execução

- Praticamente todos os **algoritmos** apresentados até agora neste capítulo **não são amigáveis**

- **Exemplo:** seja a seguir o algoritmo visto para o cálculo do fatorial de um número

- **O algoritmo interromperá sua execução, quando encontrar o comando read (n);**
- **Ficará aguardando o operador digitar o número para o qual se deseja calcular o fatorial**
- **Um operador desavisado não saberá o porque dessa interrupção e nem o que digitar, para dar continuidade à execução**
- **Este não é um algoritmo amigável**

```
Fatorial {
        int n, fat, i;

        read (n);
        fat = 1; i =2;
        while (i <= n) {
                fat = fat * i;
                i = i + 1;
        }
        write ("Fatorial (", n, ") = ", fat);
}
```

Livro 01

- **Agora uma versão amigável para o algoritmo:**

```
Fatorial {
        int n, fat, i;

        read (n);
        fat = 1; i =2;
        while (i <= n) {
                fat = fat * i;
                i = i + 1;
        }
        write ("Fatorial (", n, ") = ", fat);
}
```

Livro 01

```
Fatorial {
        int n, fat, i;

        write ("Calculo do fatorial de um numero\n\n");
        write ("Digite o numero: ");
        read (n);
        fat = 1; i =2;
        while (i <= n) {
                fat = fat * i;
                i = i + 1;
        }
        write ("Fatorial (", n, ") = ", fat);
}
```

Livro 01

Outro exemplo: versão amigável para o algoritmo da equação do 2º grau

Livro 01

```
RaizesEquação2Grau {
        float A, B, C, X1, X2, Delta, Real, Imag;

        write ("Solucao de uma equacao do segundo grau\n\n");
        write ("Digite os coeficientes da equacao: ");
        read (A, B, C);
        Delta = pow (B, 2) - 4*A*C;
        if (Delta >= 0) {
                X1 = (-B + sqrt (Delta)) / (2*A);
                X2 = (-B - sqrt (Delta)) / (2*A);
                write ("X1 = ", X1, " e  X2 = ", X2);
        }
        else {
                Real  =  -b / (2*A); Imag  =  sqrt (Delta) / (2*A);
                write ("X1 = (",  Real, ") + i (", Imag, ") e X2 = (", Real,
                        ") - i (", Imag, ")");
        }
}
```

6.7 – Algoritmos bem-estruturados, legíveis e de fácil correção

- Existem **metodologias** destinadas a alcançar essas características

- Dentre as quais podem ser citadas:

  - Metodologia top-down
  - Programação modular
  - Programação estruturada
  - Programação orientada a objetos

Metodologia top-down (divide and conquer)**:**

- É a decomposição de uma **grande tarefa** numa coleção de **tarefas menores** interligadas
- **Cada uma** dessas tarefas menores pode ser **decomposta da mesma forma**
- No final, chega-se a uma coleção de **tarefas triviais** interligadas
- Cada uma delas poderá ser resolvida por um **comando típico dos algoritmos**

Programação modular:

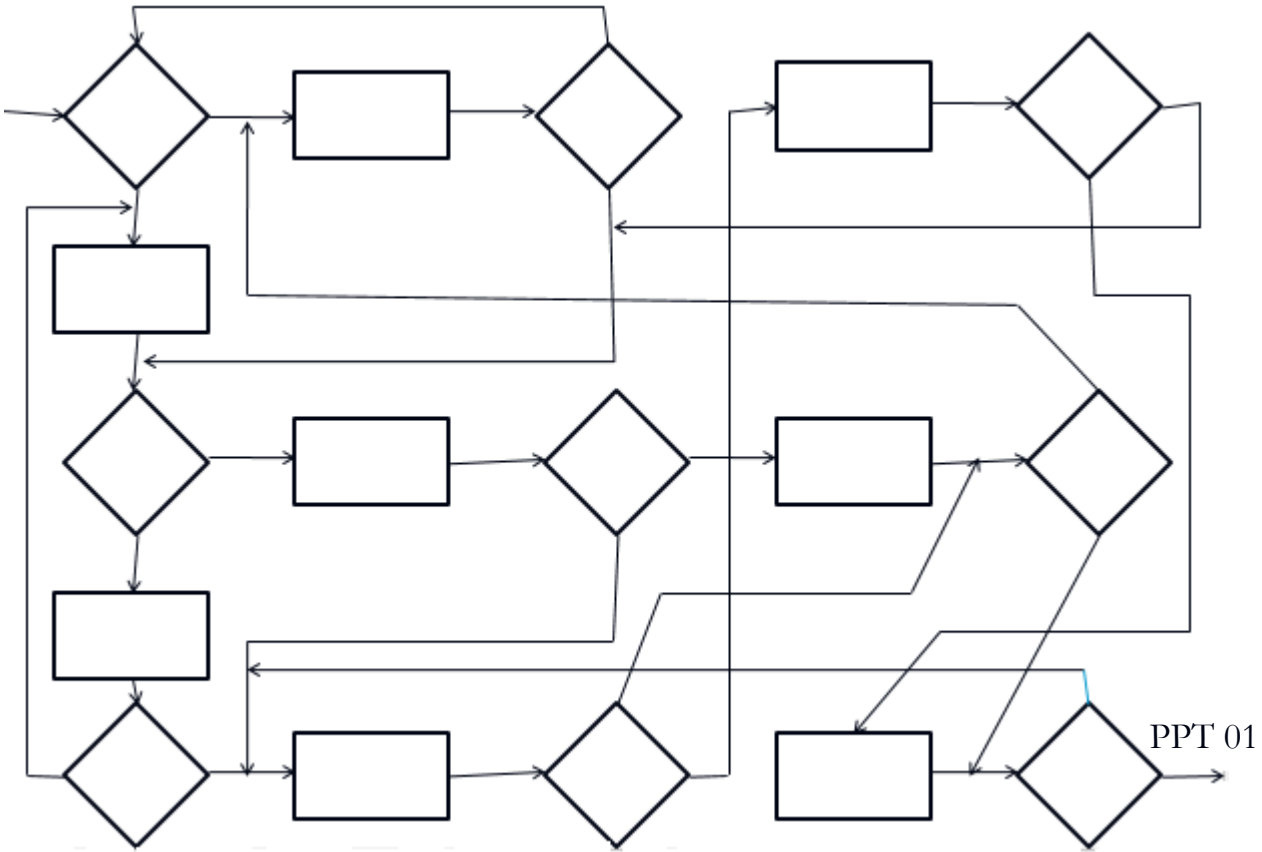
- Divide um grande algoritmo em **módulos**, cada um destinado a resolver um **problema específico**

- Esses módulos podem ser **colecionados** pelo programador, formando sua **ferramentaria** para a solução de novos grandes problemas

- A partir do capítulo sobre **subprogramação**, os programas serão construídos usando essa abordagem **modular**

Programação estruturada:

- Constituída de regras e restrições para ajudar o programador a evitar algoritmos **confusos**, **ilegíveis** e **difíceis de serem corrigidos** (macarronadas ou teias de aranha)



Programação estruturada:

- Constituída de regras e restrições para ajudar o programador a evitar algoritmos **confusos**, **ilegíveis** e **difíceis de serem corrigidos** (macarronadas ou teias de aranha)

- Em tais algoritmos, uma **alteração** num único comando do programa pode provocar a necessidade de **outras alterações** em diversas partes do algoritmo

- Todos os **algoritmos** aqui apresentados **obedecem** às regras e restrições da **programação estruturada**

Programação orientada a objetos:

- Auxiliam o programador a organizar seus algoritmos, visando facilitar a **localização de erros** e a substituição de **trechos ineficientes**, sem ter que examinar o algoritmo inteiro

- Esse tipo de programação se constitui num **paradigma diferente** daquele que será abordado nesta disciplina

- Será portanto endereçado para disciplinas específicas

- **CES-11 Algoritmos e Estruturas de Dados** tem um capítulo sobre o assunto

6.8 – Algoritmos tolerantes a falhas

- Poder trabalhar com situações excepcionais

- **Exemplos:**

  - O programa pede para digitar opção **"a"** ou **"b"**, mas o operador digita **"c"**. E agora?

  - O programa pede para digitar um **dígito decimal**, mas o operador digita uma **letra**. O que acontecerá?

- É preciso evitar **pane** no programa

6.9 – Outros ingredientes

- **Comentários** no meio dos comandos, explicando a **finalidade** de alguns trechos do algoritmo

- **Endentação** para facilitar a **visualização do escopo** dos comandos

Computadores, Algoritmos e Programas: Manual Introdutório de Ensino

Capítulo VII

Estrutura de um Programa em C

Capítulo VII – Estrutura de um Programa em C

7.1 – Partes componentes de um programa em C

7.2 – Cabeçalho de uma função

7.3 – Declarações globais e locais

7.4 – Diretivas de pré-processamento

7.5 – Comentários em programas

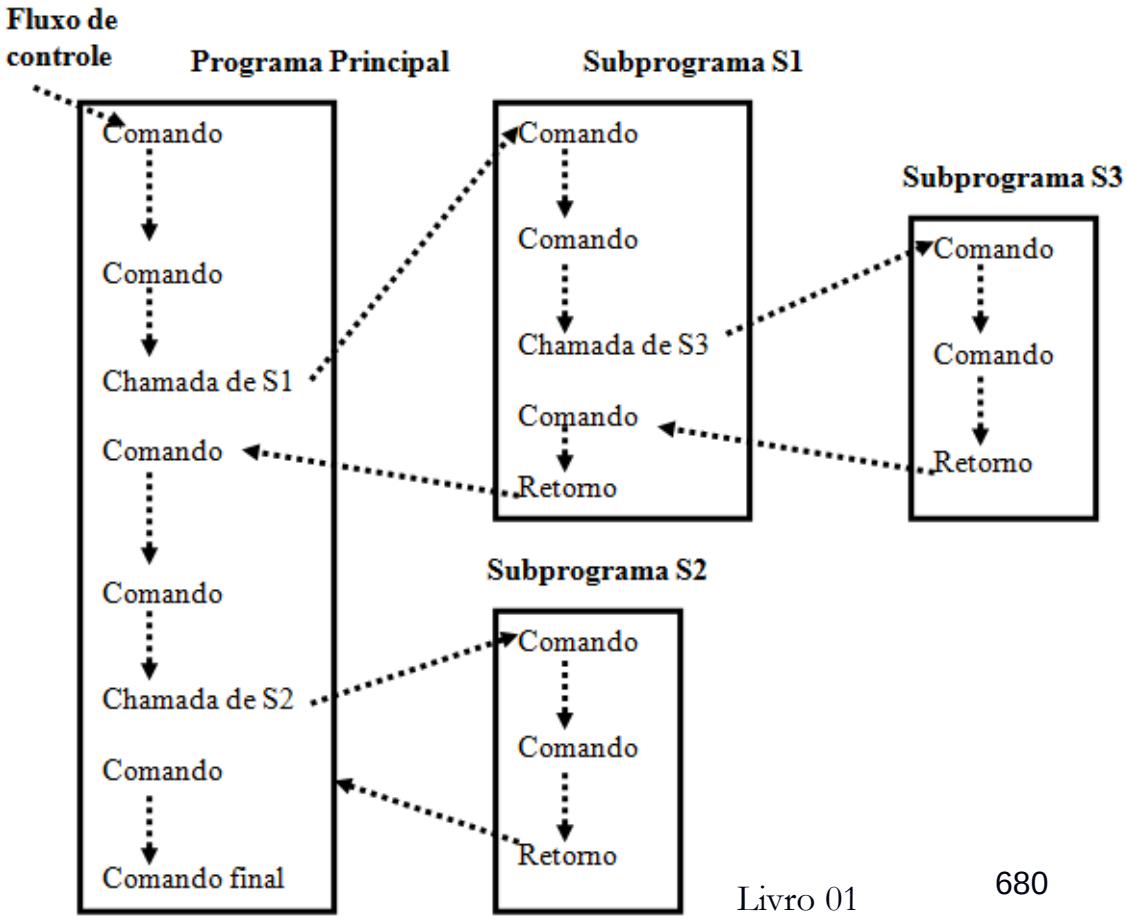
7.1 – Partes componentes de um programa em C

Como já foi comentado, um programa pode ter vários módulos

O módulo principal é obrigatório

Por ele começa a execução

Os outros módulos (opcionais) são os subprogramas



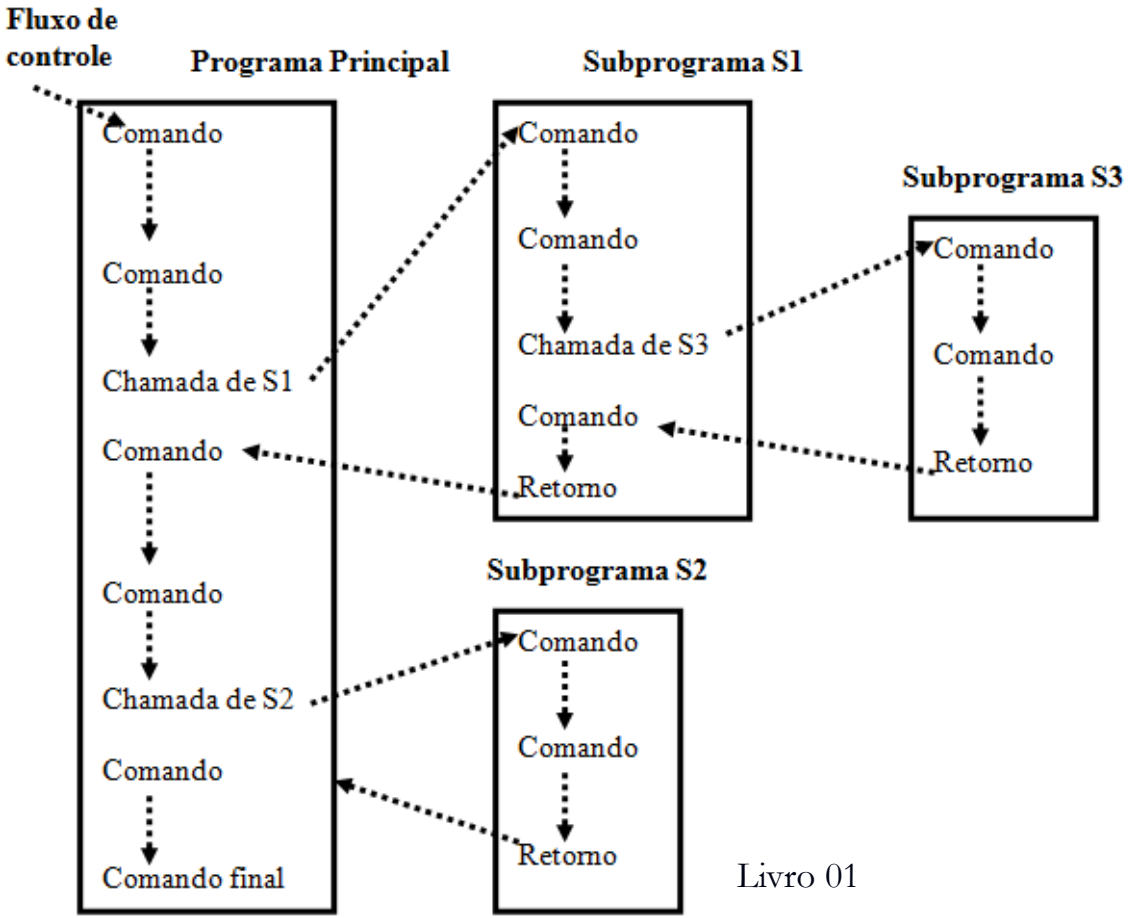
Em C, todos os módulos de um programa são denominados funções

O módulo principal tem o nome de função main

Cada módulo pode ter declarações locais

Pode haver ainda declarações fora do escopo de qualquer função

São as declarações globais



Livro 01

Declarações globais podem figurar no início ou no final do programa, ou depois de quaisquer de suas funções

Nesta disciplina elas estarão sempre no início dos programas

Estrutura básica de um programa em C, para fins didáticos:



Livro 01

A função main deve aparecer depois de suas funções auxiliares, pois em seu escopo elas são referenciadas

Se ela aparecesse antes, o compilador, ao encontrar tais referências, iria notificar o aparecimento de nomes não declarados

Toda função chamada dentro de outra deve aparecer no programa antes dessa outra

Prototipação – vista em estudos de Subprogramação – permite inverter a ordem:



Livro 01

Isso pode ajudar na aplicação da metodologia "top-down"

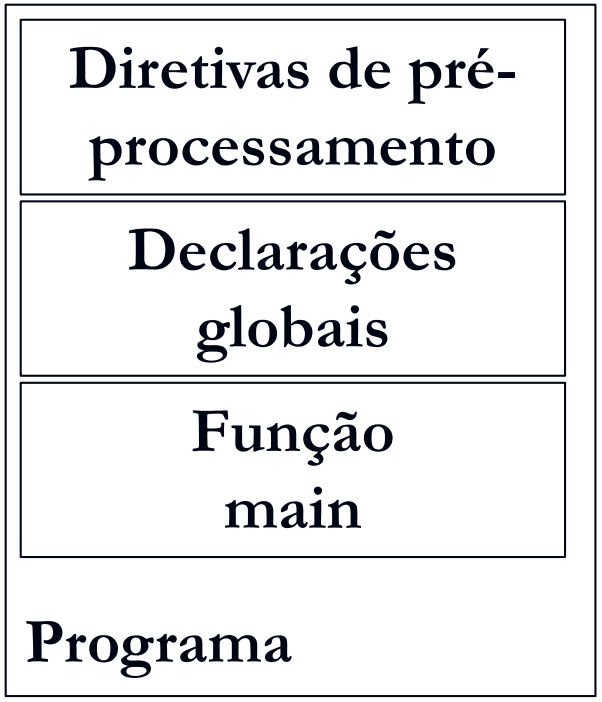
Mostra primeiro a função main

Depois suas auxiliares

Depois as auxiliares das auxiliares

E assim por diante

Antes dos estudos de Subprogramação, os **programas não terão subprogramas**:



Livro 01

Estrutura sugerida de uma função em C:

Cabeçalho {
Declarações locais
Comandos
}

Declarações poderiam vir no meio dos comandos, mas isso será evitado

7.2 – Cabeçalho de uma função

- Forma geral:

tipo nome (lista de parâmetros)

- **nome ( )** tem presença obrigatória; o restante é opcional

- Uma função pode **retornar um valor** para aquela que a invocou

  – **tipo** é o tipo do valor retornado pela função

tipo nome (lista de parâmetros)

- **Parâmetro** é uma variável local especial de uma função:
  - Destina-se a receber um **argumento** de chamada dessa função

- **Exemplo:** seja a atribuição **z = sin(x+y)**, numa função qualquer **f**

  - A função **sin** pertence à biblioteca da Linguagem C

  - Na função **f**, o valor de **x+y** é calculado; ele é o **argumento** dessa chamada da função **sin**

  - A função **sin** é então invocada e seu único **parâmetro** recebe tal **argumento**

  - A função **sin** calcula o seno do valor de seu **parâmetro**, resultando em um valor do tipo **real**

  - Finalmente a função **sin** retorna esse valor real à função **f**, que o atribui à variável **z**

- **Por default**, o tipo de uma função é **int**
- Quando uma função **não produz** nenhum **valor** a ser retornado, seu tipo deve ser **void**
- É o caso da função **main** de muitos programas, em alguns ambientes de programação
- A função **main** de muitos programas também **não tem parâmetros**

Na maioria dos casos, sua forma será:

int main ( )

Alguns ambientes como o **Borland C++** também aceitam

void main ( )

7.3 – Declarações globais e locais

- **Declarações** são usadas para:
  - Dar **nome** às **variáveis** e **constantes** do programa
  - Reservar-lhes **espaço na memória**
  - Especificar os **tipos** de valores que elas devem guardar
  - Criar **outros tipos** além daqueles disponibilizados pela linguagem

- Nos programas em **C**, toda **variável** referenciada nos comandos deve estar previamente **declarada**

- Quando **mais de uma função** de um programa referencia uma mesma **variável**, essa deve ser declarada como **global**

- Se **apenas uma função** o fizer, ela pode ser declarada como **local** a essa função

- Somente a partir do capítulo sobre **Subprogramação** os programas terão **mais de uma função**

- Até lá, suas variáveis serão **globais** ou **locais** à função **main**

7.4 – Diretivas de pré-processamento

- **Pré-processamento:** fase da compilação anterior à **tradução** do programa para a linguagem **Assembly** ou de máquina

- **Tarefas do pré-processamento** – entre outras:

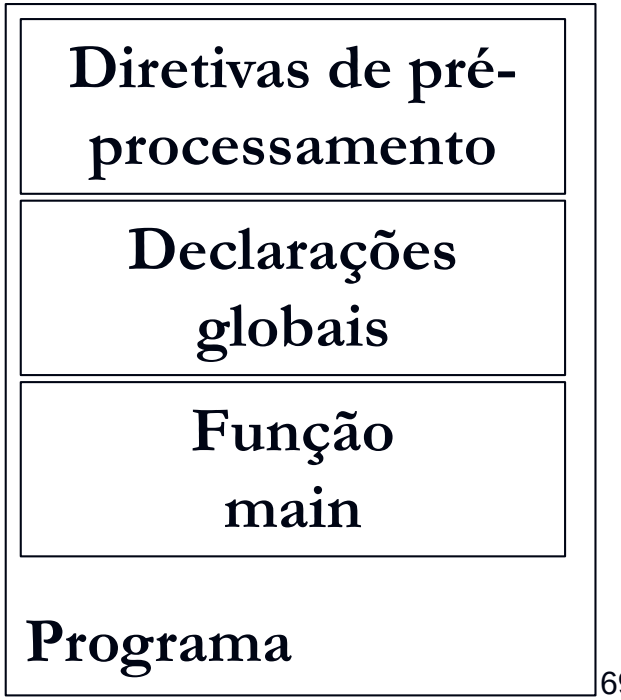
  - Inclusão de **arquivos auxiliares**

  - Tradução de **constantes simbólicas**

  - Pré-processamento de **macros**

Diretivas de pré-processamento em C:

- Toda linha iniciada com o caractere **'#'**
- Podem aparecer em **qualquer posição** nos programas
- Em **CES-10**, para fins didáticos, serão apresentadas **no início** dos programas



Inclusão de arquivos auxiliares:

- Os arquivos podem ser de dois tipos**:**
  - Arquivos do acervo do **programador**
    - Exemplo: **#include "sistemas.c"**
  - Arquivos da **biblioteca** da linguagem
    - Exemplo: **#include <math.h>**

Arquivos do acervo do programador:

- Exemplo: **#include "sistemas.c"**
- Tipicamente **escritos em C**
- Incorporados ao programa-fonte **durante o pré-processamento**
- Traduzidos para **Assembly** ou linguagem de máquina juntamente com o **restante do programa-fonte**

Arquivos da biblioteca da linguagem :

- Exemplo: **#include <math.h>**

- Contém apenas **cabeçalhos** (protótipos) de algumas funções da biblioteca, **escritos em C**

- Os **códigos-objetos** dessas funções já estão **prontos** e se encontram em algum lugar da **biblioteca** da linguagem

- Todos os **cabeçalhos** do arquivo são incorporados ao programa-fonte **durante o pré-processamento**, mesmo aqueles que não serão usados

Arquivos da biblioteca da linguagem:

- Esses **cabeçalhos** entram no processo de tradução para **Assembly** ou linguagem de máquina, juntamente com o **restante do programa-fonte**

- Durante a **edição de ligações**, os códigos-objetos das funções da biblioteca usadas no programa são **ligadas ao restante do programa-objeto**

- **Obs.:** a inclusão dos cabeçalhos no programa-fonte evita que o compilador acuse **falta de declaração**, ao encontrar a **chamada** de uma função da biblioteca

Tradução de constantes simbólicas:

- Diretiva **#define**: serve para incluir **constantes simbólicas** no programa

- Por exemplo, caso o programa contenha as seguintes diretivas:

```
#define  pi  3.14159
#define  limite  100
#define  enquanto  while
#define  se  if
#define  senao  else
#define  eq  ==
```

Durante o pré-processamento, todas as ocorrências das cadeias de caracteres:

pi, limite, enquanto, se, senao, eq

Serão substituídas respectivamente pelas cadeias:

3.14159, 100, while, if, else, ==

Utilidades de constantes simbólicas:

- Evitar distrações:
  - É comum trocar distraidamente **“==”** por **‘=’**
  - Usando **#define eq ==**

    pode-se escrever sistematicamente **“eq”** no lugar de **“==”**, evitando distrações

Utilidades de constantes simbólicas:

- Evitar **escrever** cadeias de caracteres compridas, **repetidamente**:

  - Seja um programa que **referencia muito** o valor da constante **π** (pi)

    Usando **#define pi 3.14159**

    Evita-se escrever várias vezes o extenso valor **3.14159**

Utilidades de constantes simbólicas:

- Escrever os programas em Português:
  - Com diretivas similares às seguintes:

```
#define  enquanto  while
#define  se  if
#define  senao  else
```

O programador pode **escrever** seus programas em **Português**

Exemplo com #include's e #define's:

- Seja o arquivo **preproc.c** que inclui o arquivo **defines.h**:

Livro 01

```
#include <stdio.h>
#include "defines.h"
int main () {
        int i;
        printf ("LIMITE_1: %d\n", LIMITE);
        i = 100;
        #define LIMITE 200
        SE (i EQ LIMITE)
                printf ("i: %d", i);
        SENAO
                printf ("LIMITE_2: %d", LIMITE);
}
```

preproc.c

```
#define LIMITE 100
#define EQ ==
#define SE if
#define SENAO else
```

defines.h

Seja o pré-processamento do arquivo **preproc.c**

Livro 01

```
#define LIMITE 100
#define EQ ==
#define SE if
#define SENAO else
```

defines.h

```
#include <stdio.h>
#include "defines.h"
int main () {
        int i;
        printf ("LIMITE_1: %d\n", LIMITE);
        i = 100;
        #define LIMITE 200
        SE (i EQ LIMITE)
                printf ("i: %d", i);
        SENAO
                printf ("LIMITE_2: %d", LIMITE);
}
```

preproc.c

```
#include <stdio.h>
#include "defines.h"
int main () {
    int i;
    printf ("LIMITE_1: %d\n", LIMITE);
    i = 100;
    #define LIMITE 200
    SE (i EQ LIMITE)
        printf ("i: %d", i);
    SENAO
        printf ("LIMITE_2: %d", LIMITE);
}
```

```
#define  LIMITE  100
#define  EQ  ==
#define  SE  if
#define  SENAO  else
```

defines.h

Protótipos das funções do stdio.h

Protótipos das funções do stdio.h

```
#include “defines.h”
int main () {
    int i;
    printf (“LIMITE_1: %d\n”, LIMITE);
    i = 100;
    #define LIMITE 200
    SE (i EQ LIMITE)
        printf (“i: %d”, i);
    SENAO
        printf (“LIMITE_2: %d”, LIMITE);
}
```

```
#define  LIMITE  100
#define  EQ  ==
#define  SE  if
#define  SENAO  else
```

defines.h

Protótipos das funções do stdio.h

```
#include “defines.h”
int main () {
    int i;
    printf (“LIMITE_1: %d\n”, LIMITE);
    i = 100;
    #define LIMITE 200
    SE (i EQ LIMITE)
        printf (“i: %d”, i);
    SENAO
        printf (“LIMITE_2: %d”, LIMITE);
}
```

```
#define  LIMITE  100
#define  EQ  ==
#define  SE  if
#define  SENAO  else
```

defines.h

Protótipos das funções do stdio.h

```
#define  LIMITE  100
#define  EQ  ==
#define  SE  if
#define  SENAO  else
int main () {
    int i;
    printf (“LIMITE_1: %d\n”, LIMITE);
    i = 100;
    #define LIMITE 200
    SE (i EQ LIMITE)
        printf (“i: %d”, i);
    SENAO
        printf (“LIMITE_2: %d”, LIMITE);
}
```

Original	Substituta
LIMITE	100
EQ	==
SE	if
SENAO	else

Livro 01

Protótipos das funções do stdio.h

```
int main () {
    int i;
    printf (“LIMITE_1: %d\n”, LIMITE);
    i = 100;
    #define LIMITE 200
    SE (i EQ LIMITE)
        printf (“i: %d”, i);
    SENAO
        printf (“LIMITE_2: %d”, LIMITE);
}
```

Original	Substituta
LIMITE	100
EQ	==
SE	if
SENAO	else

Livro 01

Protótipos das funções do stdio.h

```
int main () {
    int i;
    printf ("LIMITE_1: %d\n", LIMITE);
    i = 100;
    #define LIMITE 200
    SE (i EQ LIMITE)
        printf ("i: %d", i);
    SENAO
        printf ("LIMITE_2: %d", LIMITE);
}
```

Original	Substituta
LIMITE	100
EQ	==
SE	if
SENAO	else

Livro 01

Dentro das aspas
não há substituição

Protótipos das funções do stdio.h

```
int main () {
    int i;
    printf (“LIMITE_1: %d\n”, 100);
    i = 100;
    #define LIMITE 200
    SE (i EQ LIMITE)
        printf (“i: %d”, i);
    SENAO
        printf (“LIMITE_2: %d”, LIMITE);
}
```

Original	Substituta
LIMITE	100
EQ	==
SE	if
SENAO	else

Livro 01

Protótipos das funções do stdio.h

```
int main () {
    int i;
    printf (“LIMITE_1: %d\n”, 100);
    i = 100;
    #define LIMITE 200
    SE (i EQ LIMITE)
        printf (“i: %d”, i);
    SENAO
        printf (“LIMITE_2: %d”, LIMITE);
}
```

Original	Substituta
LIMITE	100
EQ	==
SE	if
SENAO	else

Livro 01

Protótipos das funções do stdio.h

```
int main () {
    int i;
    printf (“LIMITE_1: %d\n”, 100);
    i = 100;
    SE (i EQ LIMITE)
        printf (“i: %d”, i);
    SENAO
        printf (“LIMITE_2: %d”, LIMITE);
}
```

Original	Substituta
LIMITE	200
EQ	==
SE	if
SENAO	else

Livro 01

Protótipos das funções do stdio.h

```
int main () {
    int i;
    printf (“LIMITE_1: %d\n”, 100);
    i = 100;
    SE (i EQ LIMITE)
        printf (“i: %d”, i);
    SENAO
        printf (“LIMITE_2: %d”, LIMITE);
}
```

Original	Substituta
LIMITE	200
EQ	==
SE	if
SENAO	else

Livro 01

Protótipos das funções do stdio.h

```
int main () {
    int i;
    printf (“LIMITE_1: %d\n”, 100);
    i = 100;
    if (i == 200)
        printf (“i: %d”, i);
    SENAO
        printf (“LIMITE_2: %d”, LIMITE);
}
```

Original	Substituta
LIMITE	200
EQ	==
SE	if
SENAO	else

Livro 01

Protótipos das funções do stdio.h

```
int main () {
    int i;
    printf (“LIMITE_1: %d\n”, 100);
    i = 100;
    if (i == 200)
        printf (“i: %d”, i);
    SENAO
        printf (“LIMITE_2: %d”, LIMITE);
}
```

Original	Substituta
LIMITE	200
EQ	==
SE	if
SENAO	else

Livro 01

Protótipos das funções do stdio.h

```
int main () {
    int i;
    printf (“LIMITE_1: %d\n”, 100);
    i = 100;
    if (i == 200)
        printf (“i: %d”, i);
    else
        printf (“LIMITE_2: %d”, 200);
}
```

Original	Substituta
LIMITE	200
EQ	==
SE	if
SENAO	else

Livro 01

Final do pré-processamento!

Pré-processamento de macros:

- **Macros:** expressões para abreviar **códigos longos**, códigos esses usados com **muita frequência** num programa

- São **substituídas** por esses códigos durante o **pré-processamento**

- Em C, as macros são criadas também através da diretiva **#define**

Exemplo: sejam as macros

```
#define EHPAR(x) (((x)%2)?0:1)
#define ERRO(mens)  printf("ERRO:%s\n",mens)
```

- Com elas, pode-se escrever comandos tais como

```
if (EHPAR(a+b))  ..... ;
if (valor > max) ERRO("Valor muito grande");
```

- O resultado do pré-processamento é:

```
if ((((a+b)%2)?0:1))  ..... ;
if (valor > max) printf("ERRO:%s\n","Valor muito grande");
```

```
#define EHPAR(x) (((x)%2)?0:1)
#define ERRO(mens)  printf("ERRO:%s\n",mens)
```

Macros

```
if (EHPAR(a+b))  ..... ;
if (valor > max) ERRO("Valor muito grande");
```

Comandos escritos

```
if ((((a+b)%2)?0:1))  ..... ;
if (valor > max) printf("ERRO:%s\n","Valor muito grande");
```

Pré-processados

- Nos resultados, observa-se inclusive **passagem de argumento**s:
  - **x** é substituído por **a+b**
  - **mens** é substituído por **"Valor muito grande"**

- A propósito:

  ((a+b)%2)?0:1

  é denominada **expressão condicional**

- **(a+b)%2** é o resto da divisão de **(a+b)** por **2**

- A expressão toda é

  - **0** (zero), se **(a+b)%2 ≠ 0**
  - **1** (um), caso contrário

Ou seja, tal expressão verifica se (a+b) é par

- Expressões condicionais serão estudadas no capítulo de **Comandos de Controle de Fluxo**

Exemplo: seja a macro

#define cubo(x) ((x)*(x)*(x))

Os parêntesis são necessários porque '*' é realizada antes de '+' e de '-'

- Pode-se usá-la em expressões aritméticas tais como

cubo(a+b) + cubo(a-b)

- Se a macro fosse

#define cubo(x) (x*x*x)

então **cubo(a+b)** se tornaria **(a+b*a+b*a+b)**

7.5 – Comentários em programas

- Usados para **documentar** e **elucidar** trechos de programas
- Facilitam a **correção**, a **compreensão** e a **manutenção** dos programas
- O **próprio programador** ou **outras pessoas** podem se beneficiar deles, em **épocas diferentes** daquela em que o programa foi elaborado

Em **C**, duas formas de introduzir **comentários** em programas:

- Tudo entre as sub-cadeias **"/*"** e **"*/"**, sem as aspas, é considerado pelo compilador um **comentário**

- Se, numa linha do programa, aparecer a sub-cadeia **"//"**, sem as aspas, o restante da linha à direita é considerado um **comentário**

- O compilador **elimina** todos os comentários do **programa-fonte**

Programa da equação do 2º grau contendo comentários:

```
/*****************************************************************
 *     Programa para resolver uma equacao do segundo grau        *
 *     Calcula e imprime raizes reais e complexas                *
 *     Os coeficientes da equacao devem ser digitados pelo operador  *
 *****************************************************************/

/* Inclusao de arquivos da biblioteca da linguagem */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* Funcao principal  */

int main ( ) {                                          // Cabecalho
      float A, B, C, X1, X2, Delta, Real, Imag;         // Declaracao das variaveis

      printf ("Solucao de uma equacao do segundo grau\n\n"); // Escrita de titulo

      printf ("Digite os coeficientes da equacao: ");  // Entrada de dados
      scanf ("%f%f%f", &A, &B, &C);

      Delta = pow (B, 2) - 4*A*C;    // Calculo do discriminante

      if (Delta >= 0) {     // Calculo e escrita das raizes reais

            X1 = (-B + sqrt (Delta)) / (2*A);
            X2 = (-B - sqrt (Delta)) / (2*A);
            printf ("X1 = %f, e  X2 = %f", X1, X2);
      }
      else {  // Calculo e escrita das raizes complexas

            Real  =  -B / (2*A); Imag  =  sqrt (-Delta) / (2*A);
            printf ("X1 = (%f) + i (%f) e X2 = (%f) - i (%f)",
                  Real, Imag, Real, Imag);
      }
      printf ("\n\n"); system ("pause"); return 0;
}   // Fim da funcao principal
```

Fábio Carneiro Mokarzel

Professor na Divisão de Ciência da Computação do Instituto Tecnológico de Aeronáutica (ITA) desde 1978. Chefe do Departamento de Engenharia de Software do ITA desde 1996. Coordenador do Curso de Engenharia de Computação do ITA de 1999 a 2011. Engenheiro Elétrico pela UNIFEI, mestre e doutor em Computação pelo ITA.

Nei Yoshihiro Soma

Professor da Divisão de Ciência da Computação do Instituto Tecnológico de Aeronáutica (ITA) desde 1986. Chefe da mesma divisão de 2003 a 2017. Mestre em Análise de Sistemas pelo INPE, Doutor em Matemática Aplicada e Computacional pela Universidade de Sheffield, pesquisador visitante da Universidade de Bolonha. Participou do Comitê Assessor do CNPq, coordenou as Engenharias III e foi titular do Conselho Técnico Científico da Educação Superior na Capes.

Computadores, Algoritmos e Programas: Manual Introdutório de Ensino

Atualmente, disciplinas introdutórias de Computação e Programação têm importância muito grande em cursos de Engenharia de praticamente todas as modalidades, devido ao fato de os sistemas computacionais estarem integrados em todas as atividades científicas e tecnológicas da sociedade. No entanto, o ensino dessas disciplinas em diversas escolas de Engenharia tem sido delegado a novos professores que, em muitos casos, têm dificuldades em preparar material teórico e prático sobre o assunto, que seja abrangente, substancial e agradável de ser acompanhado.

Esta obra faz parte de um conjunto de três livros que destina-se a auxiliar esses professores, pois é fruto de longa experiência dos autores na ministração dessas disciplinas. Esse conjunto de livros apresenta de forma intensa os primeiros passos de programação de computadores, com inúmeras ilustrações destinadas elucidar seus princípios básicos. A Linguagem C é utilizada no desenvolvimento de programas. Este livro, particularmente, apresenta conceitos primários sobre computadores, algoritmos e programas, conceitos esses que servem como base para o ensino de Programação, abordado nos outros dois livros do conjunto