Laboratório de Sistemas Digitais

Aula Teórico-Prática – preparação para o projeto final

Ano Letivo 2019/20

Introdução à arquitetura de computadores

Guilherme Campos, Iouliia Skliarova

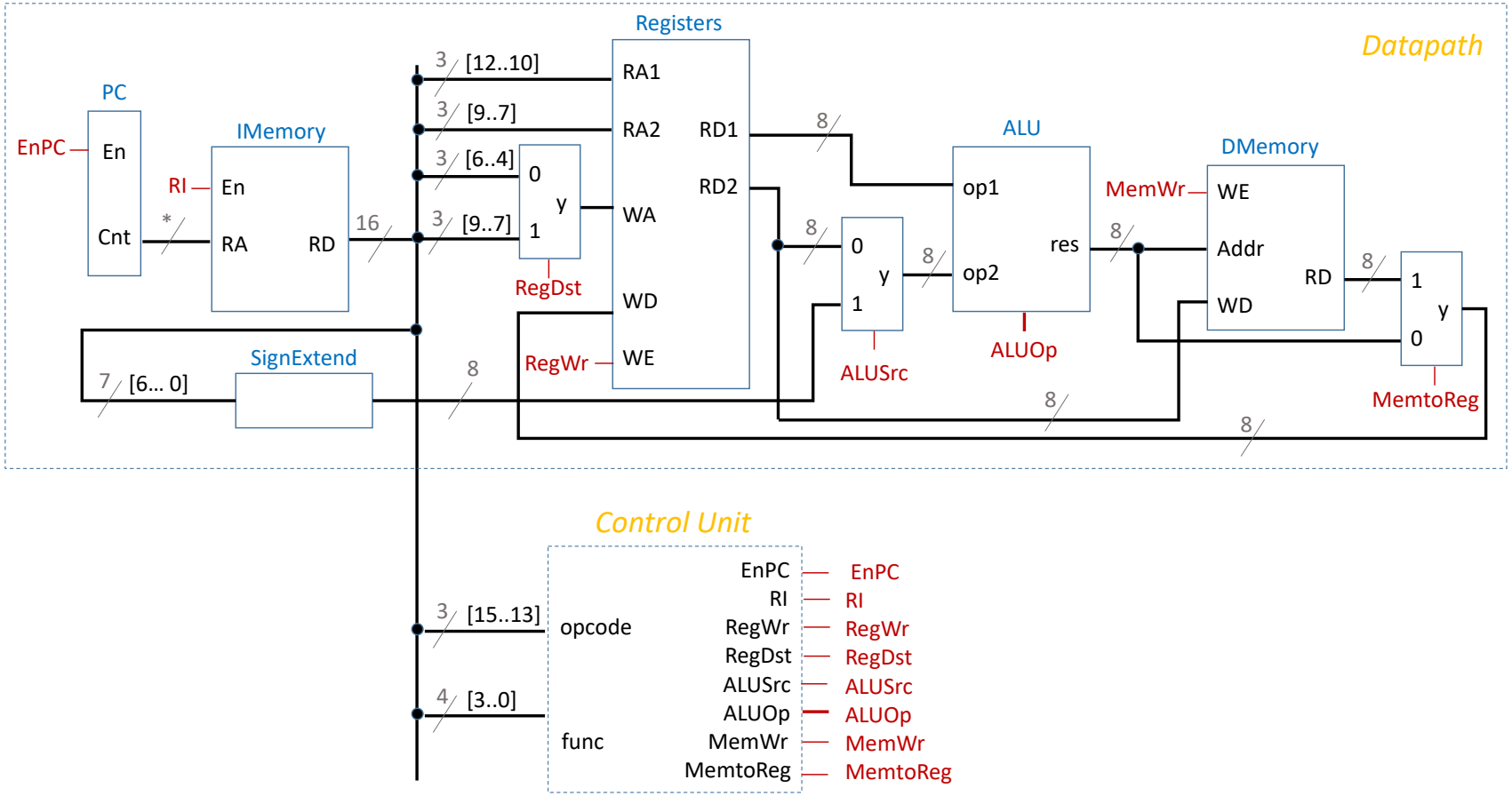
Conteúdo

- Conceitos fundamentais em arquitetura de computadores
  - O computador como sistema digital
  - Os elementos básicos de um computador
  - Exemplo dum processador simplificado
- Instruções
  - Tipos de instruções
  - Codificação de instruções
  - O ciclo básico de execução de uma instrução

Arquitetura de computadores

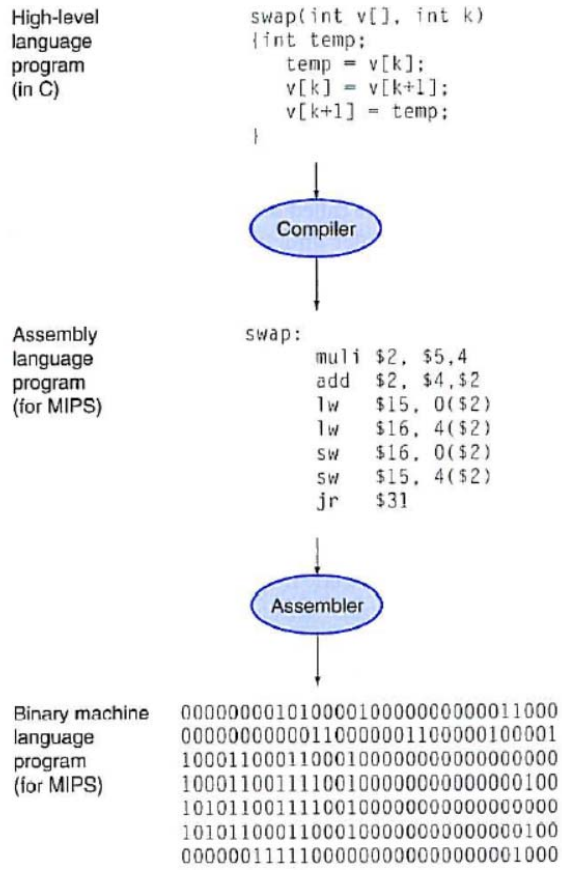
- Arquitectura de computadores é uma das áreas de aplicação direta dos conceitos, técnicas e metodologias aprendidas em ISD e LSD
- Em arquitetura de computadores, contudo, trabalha-se num nível de abstração diferente, recorrendo na maior parte das vezes a blocos funcionais complexos:
  - Unidades de entrada – permitem a receção de informação vinda do exterior (dados, programas) e que é armazenada em memória
  - Unidades de saída – permitem o envio de resultados para o exterior
  - Memória – armazenamento de:
    - programas
    - dados para processamento
    - resultados
  - CPU – processamento da informação através da execução do programa armazenado em memória
- Cada um destes blocos é um sistema digital

Arquitetura do processador a desenvolver



Níveis de representação

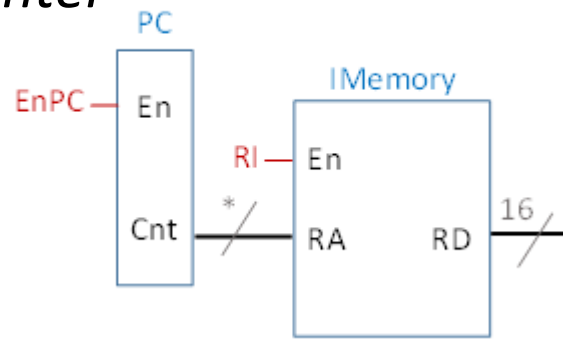
- Computadores executam comandos, chamados instruções
- Conjunto de instruções (*instruction set*) - operações que um processador suporta
- Uma instrução é uma coleção de bits:
  - Exemplo: 0010100111000000 – somar dois valores
- Instruções de mais baixo nível (0s e 1s) constituem a linguagem de máquina
- Primeiros computadores foram programados com a linguagem de máquina - pouco eficiente
- *Assembly* (linguagem de montagem) é uma notação **legível** por humanos para o código de máquina que uma arquitetura de computador específica usa
  - Exemplo: **ADD** $2, $3, $4
- *Assembler* (montador) traduz o código assembly para o código de máquina
- *Assembly* obriga o programador a pensar como computador (uma instrução por cada ação a executar) - não produtivo e requer conhecimento detalhado da arquitetura
- Linguagens de programação de alto nível (C, C++, Java…)
  - Exemplo: c = a + b;
- Compilador - traduz o código fonte de uma linguagem de programação de alto nível para uma linguagem de programação de baixo nível



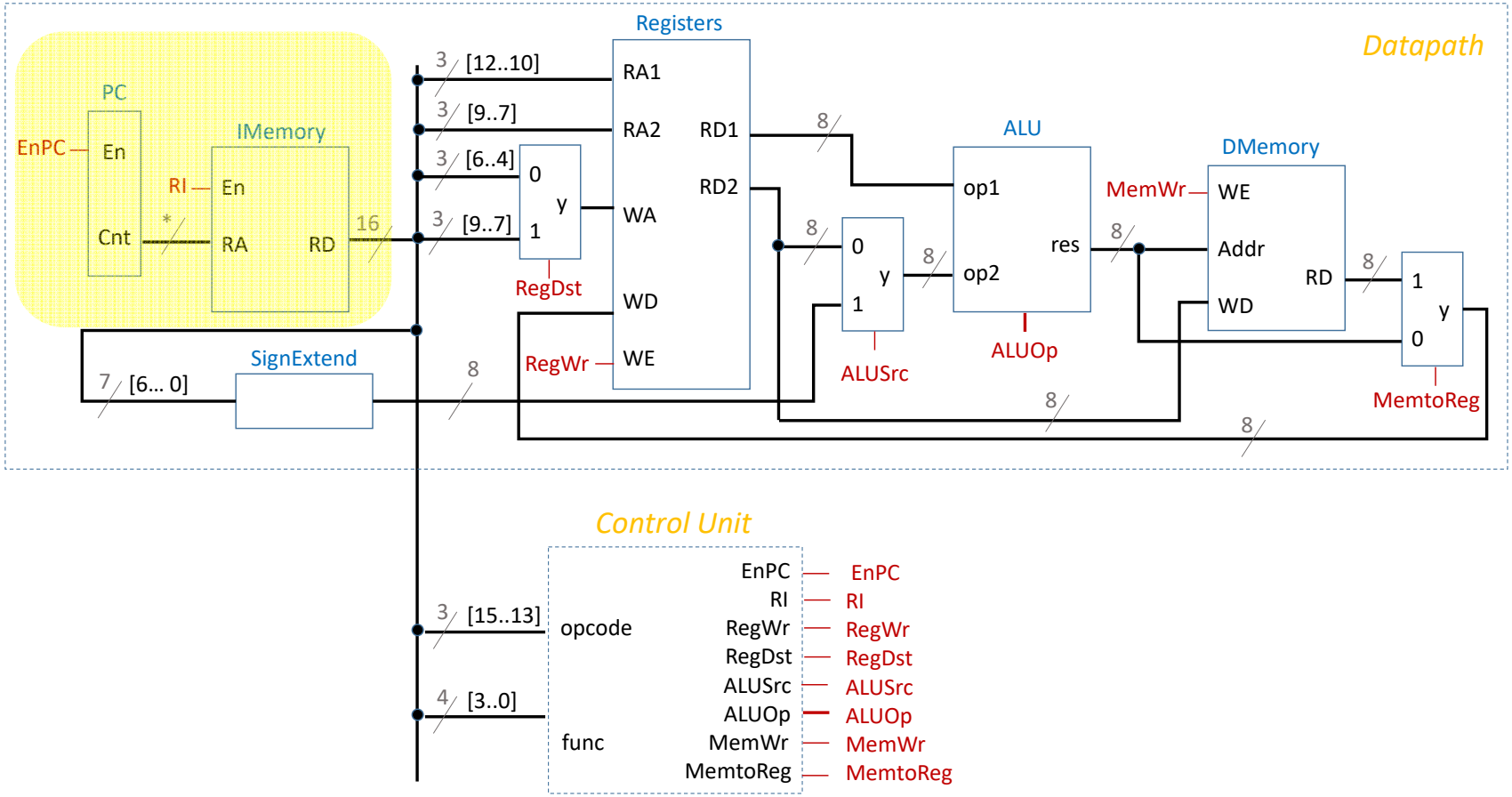
D.A. Petterson, J.L. Hennessy, Computer Organization and Design. The hardware/software interface, 4th ed., 2009

Instruções

- Independentemente do tipo de CPU e da sua estrutura interna, qualquer instrução deverá ter uma codificação que permita responder às seguintes questões:
  - Qual a operação a realizar?
  - Qual a localização dos operandos?
  - Onde colocar o resultado?
  - Qual a próxima instrução?
- As instruções são guardadas na memória e endereçadas com um contador, chamado *Program Counter*

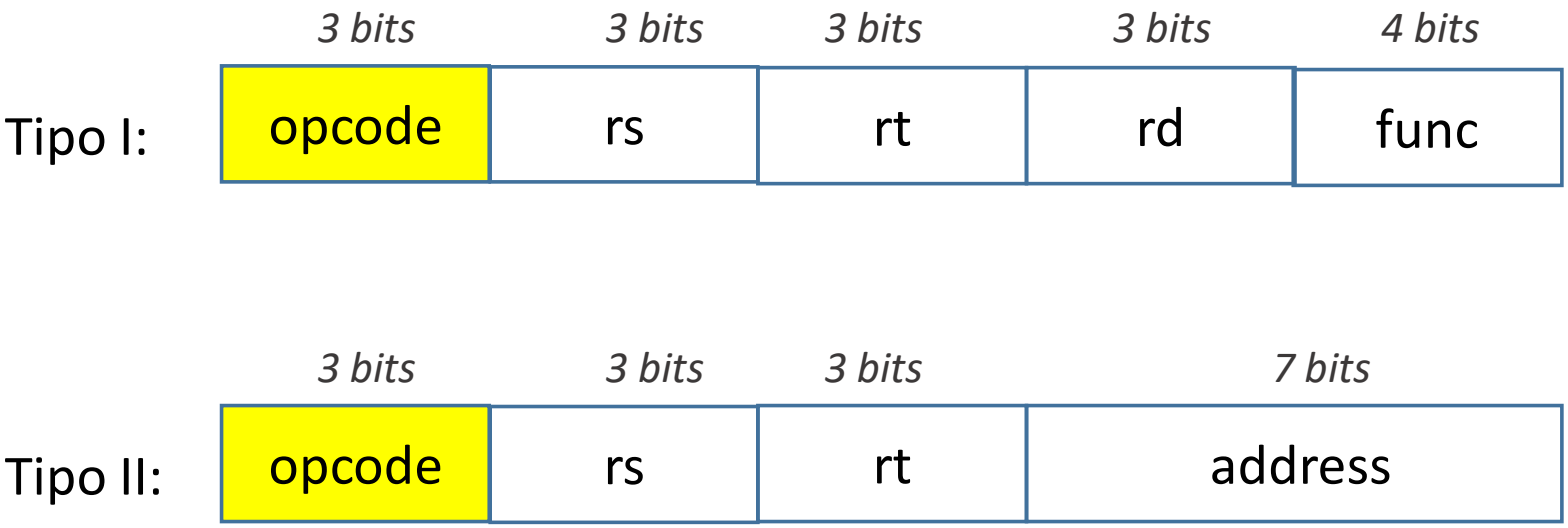


Arquitetura do processador a desenvolver



Codificação de instruções

- 16 bits para cada
- 2 formatos possíveis: I e II

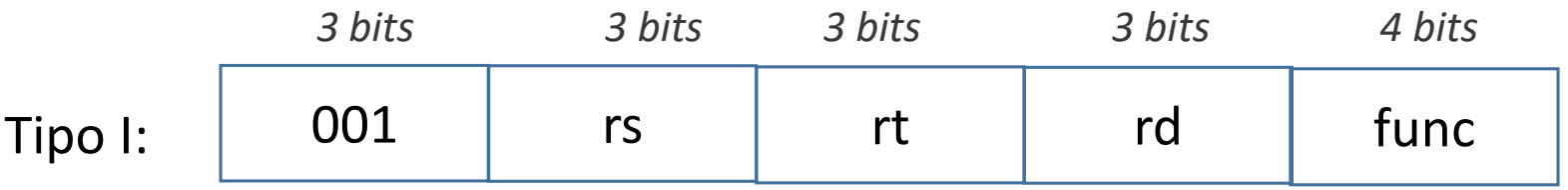


Código de operação

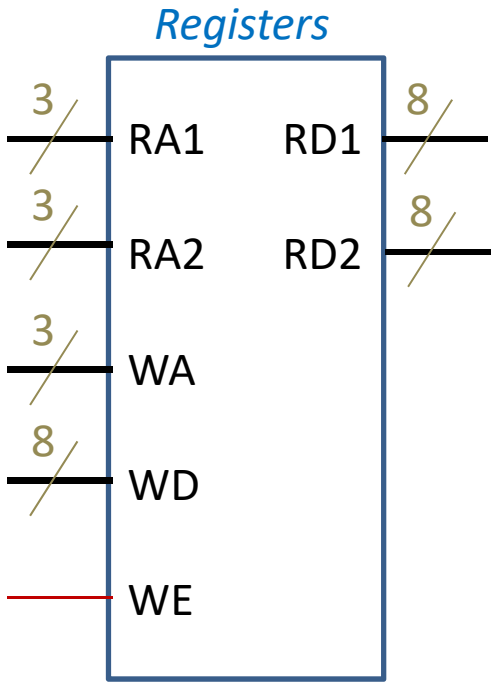
- Operações possíveis são definidas no campo *opcode (operation code)* de cada instrução

opcode	Instrução
000	**NOP** – não fazer nada
001	Todas as instruções aritméticas ou lógicas (a operação a executar é definida pelo campo *func*) – tipo I 
100	**ADDI** - soma o conteúdo dum registo com uma constante – tipo II
110	**SW** – transferir dados dum registo para a memória de dados – tipo II
111	**LW** – transferir dados da memória de dados para um registo – tipo II

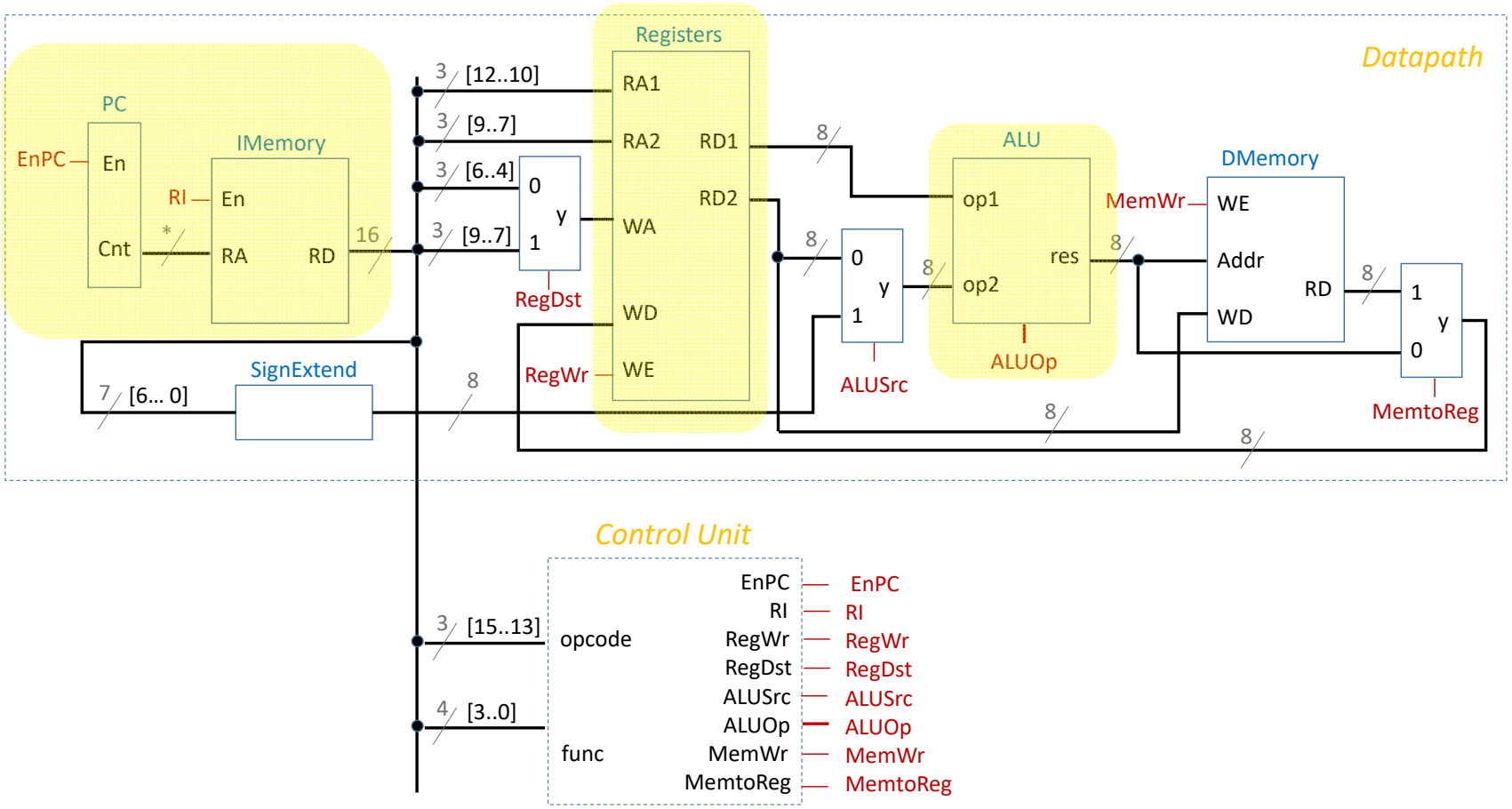
Instruções tipo I



- Realizam operações aritméticas/lógicas
- Não têm acesso à memória
- Só trabalham com operandos armazenados em registos
- Existem 8 registos de 8 bits cada um
- *rs* – 1º operando
- *rt* – 2º operando
- *rd* – resultado
- *func* – operação a executar (numa ALU)

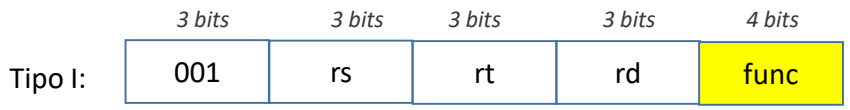


Arquitetura do processador a desenvolver

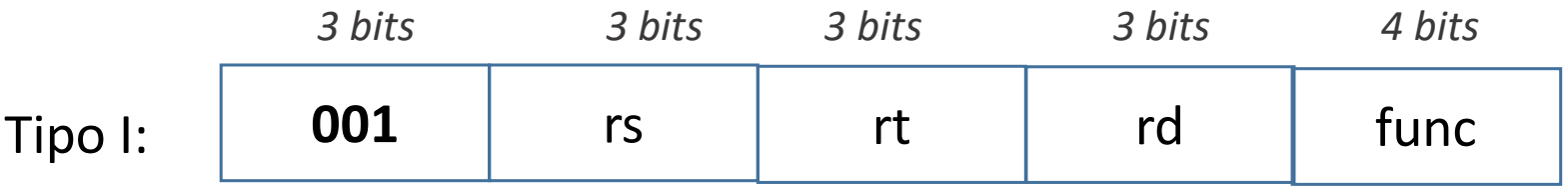


Campo *func* em instruções tipo I

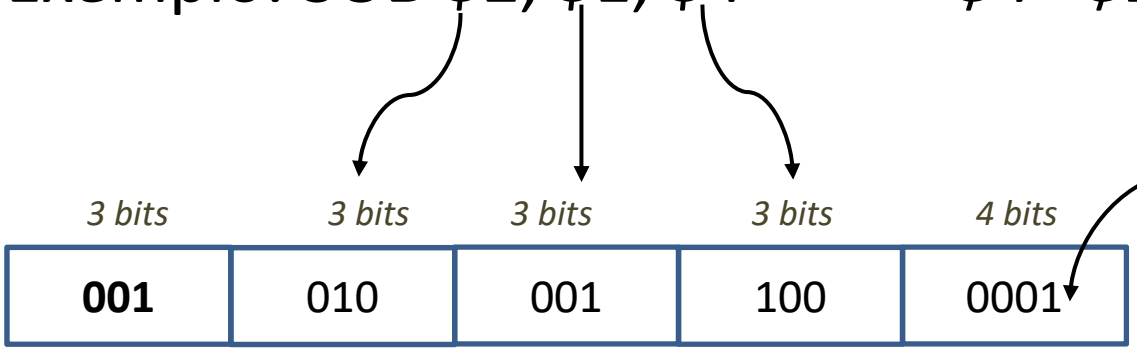
func	Operação
0000	**ADD** – soma
0001	**SUB** – subtração
0010	**AND**
0011	**OR**
0100	**XOR**
0101	**NOR**
0110	**MUU** - multiplicação sem sinal (só é usada a metade menos significativa do resultado)
0111	**MUS** - multiplicação com sinal (só é usada a metade menos significativa do resultado)
1000	**SLL** - deslocamento lógico do registo$_{rs}$ à esquerda registo$_{rt}$ bits
1001	**SRL** - deslocamento lógico do registo$_{rs}$ à direita registo$_{rt}$ bits
1010	**SRA** - deslocamento aritmético do registo$_{rs}$ à direita registo$_{rt}$ bits
1011	**EQ** – *equal* - o resultado é 1 se registo$_{rs}$ = registo$_{rt}$
1100	**SLS** – *set less than signed* – o resultado é 1 se *signed*(registo$_{rs}$) < *signed*(registo$_{rt}$)
1101	**SLU** – *set less than unsigned* – o resultado é 1 se *unsigned*(registo$_{rs}$) < *unsigned*(registo$_{rt}$)
1110	**SGS** – *set greater than signed* – o resultado é 1 se *signed*(registo$_{rs}$) > *signed*(registo$_{rt}$)
1111	**SGU** – *set greater than unsigned* – o resultado é 1 se *unsigned*(registo$_{rs}$) > *unsigned*(registo$_{rt}$)



Exemplo de codificação de instruções tipo I



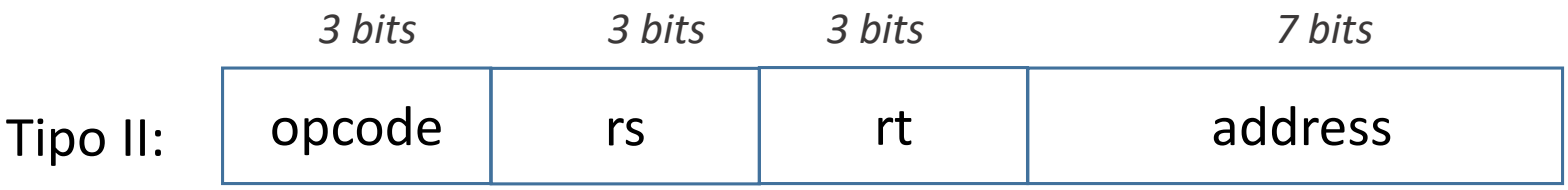
- Exemplo: **SUB** $2, $1, $4 --->>> *$4= $2 - $1*



func	Operação
0000	**ADD** – soma
0001	**SUB** – subtração
0010	**AND**
0011	**OR**

- Resultado: 0010100011000001

Instruções tipo II

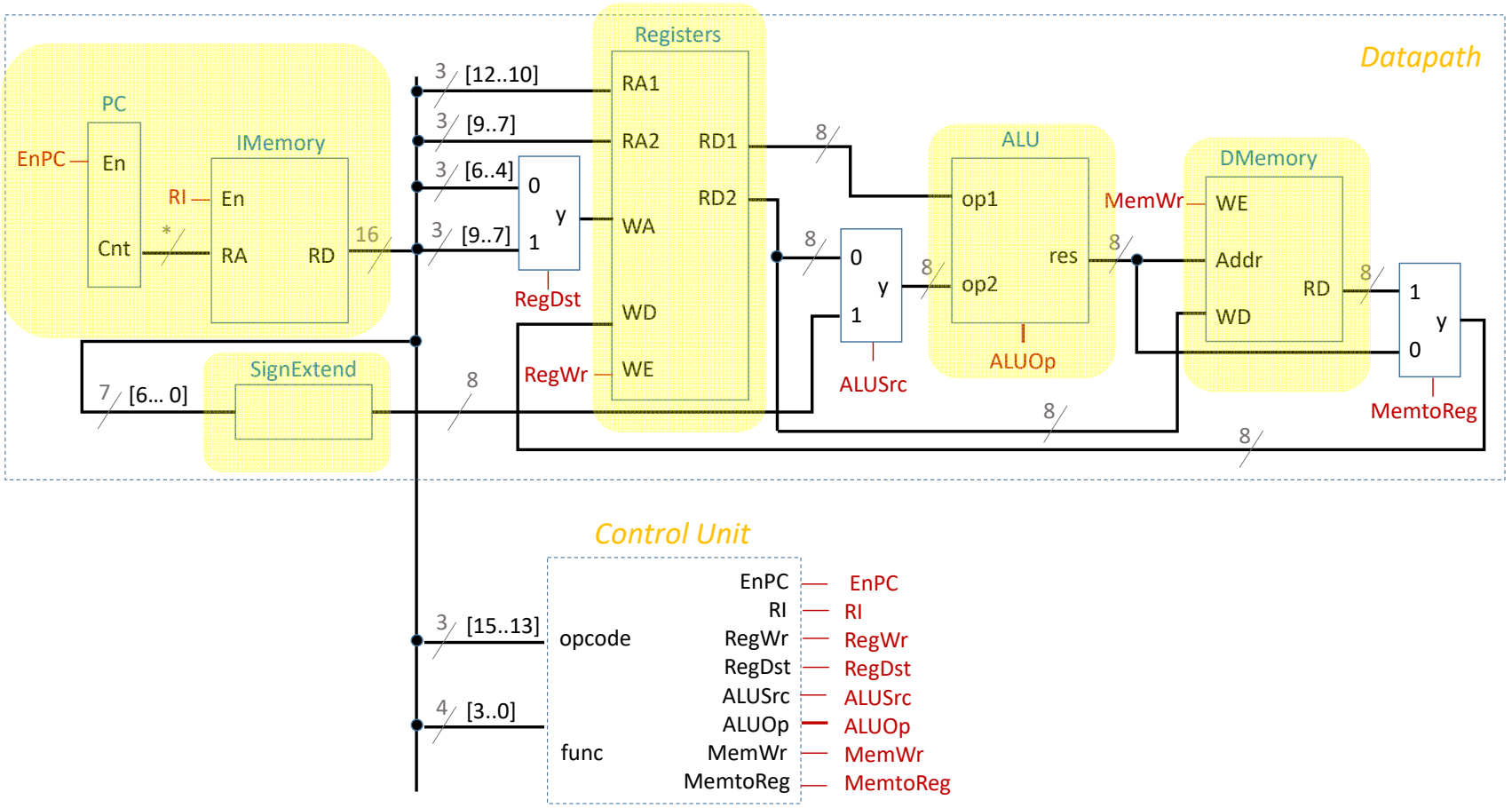


opcode	Instrução
100	ADDI
110	SW
111	LW

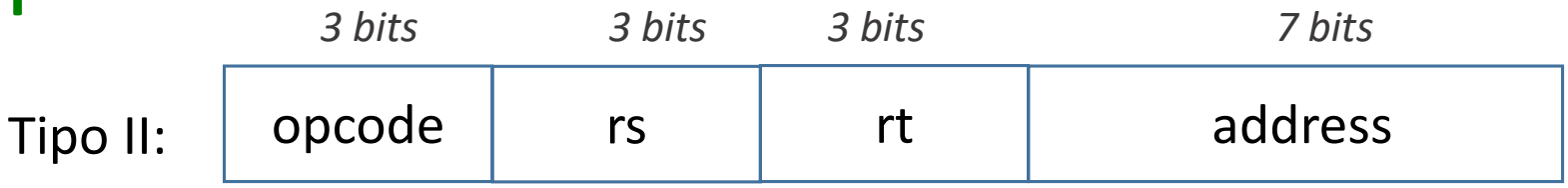
- Têm acesso à memória de dados
- Permitem trabalhar com constantes
- Com as instruções do **tipo II** é possível:
  - ler o conteúdo do registo *rt* e guardá-lo na memória de dados (*DMemory*) na posição [registo$_{rs}$ + *address*] – instrução **SW** – *store word*
  - realizar a operação inversa, i.e. ler um dado da memória, da posição [registo$_{rs}$ + *address*], e armazena-lo no registo *rt* – instrução **LW** – *load word*
  - somar o conteúdo do registo *rs* com uma constante, especificada no campo *address*, e guardar o resultado no registo *rt* – instrução **ADDI** – *add immediate*;

- Exemplos de instruções do **tipo II**:
  - **SW** $*rs*, $*rt*, address --->>> *DMemory*[registo$_{rs}$ + address] = registo$_{rt}$
  - **LW** $*rs*, $*rt*, address --->>> registo$_{rt}$ = *DMemory*[registo$_{rs}$ + address]
  - **ADDI** $*rs*, $*rt*, address --->>> registo$_{rt}$ = registo$_{rs}$ + address

Arquitetura do processador a desenvolver

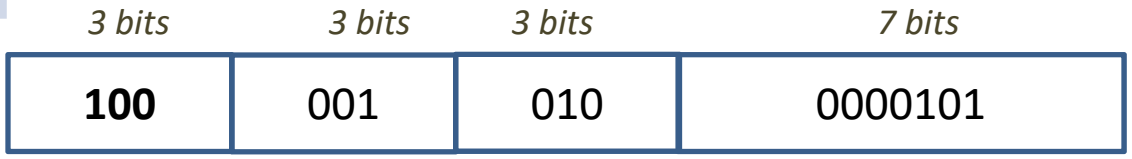


Exemplo de codificação de instruções tipo II



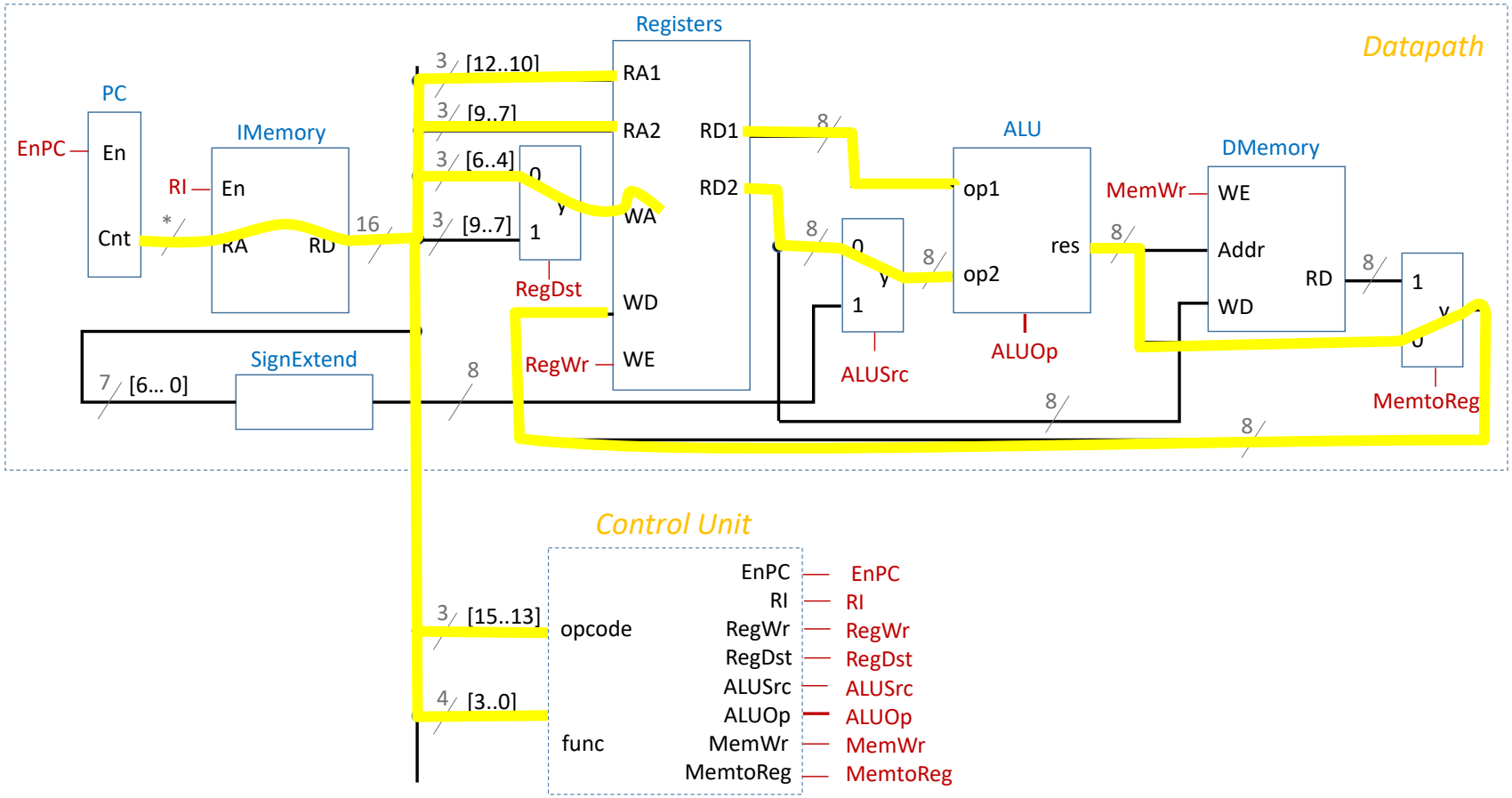
- Exemplo: **ADDI** $1, $2, 5 --->>> *$2* = *$1* + 5

opcode	Instrução
100	ADDI
110	SW
111	LW

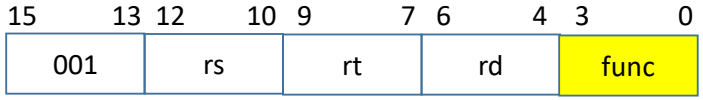


- Resultado: 1000010100000101

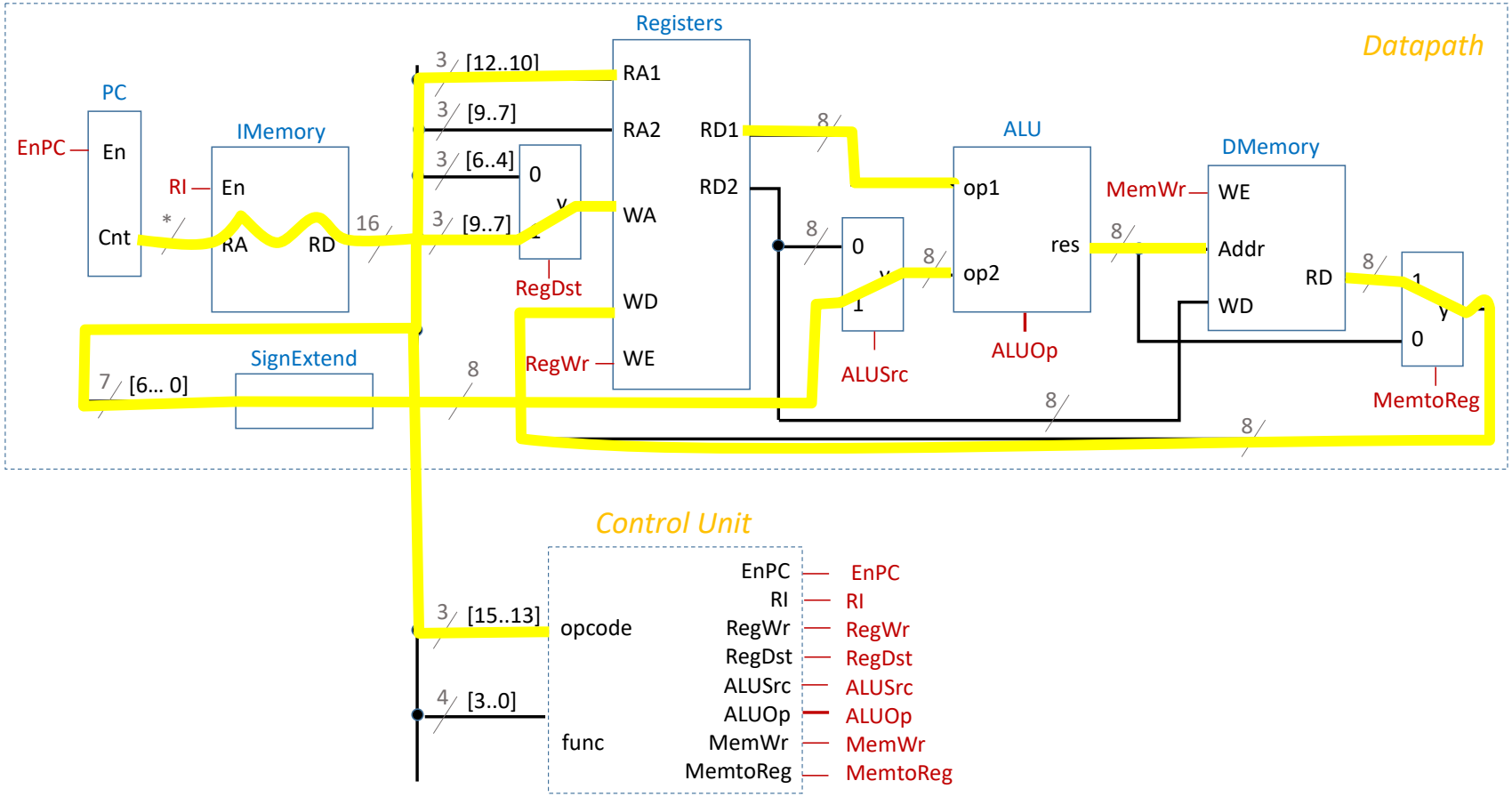
Fluxo de execução de instruções tipo I

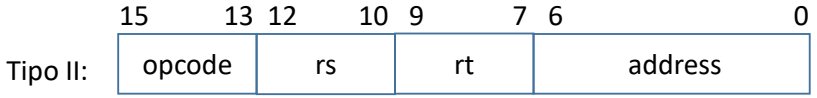


Tipo I:

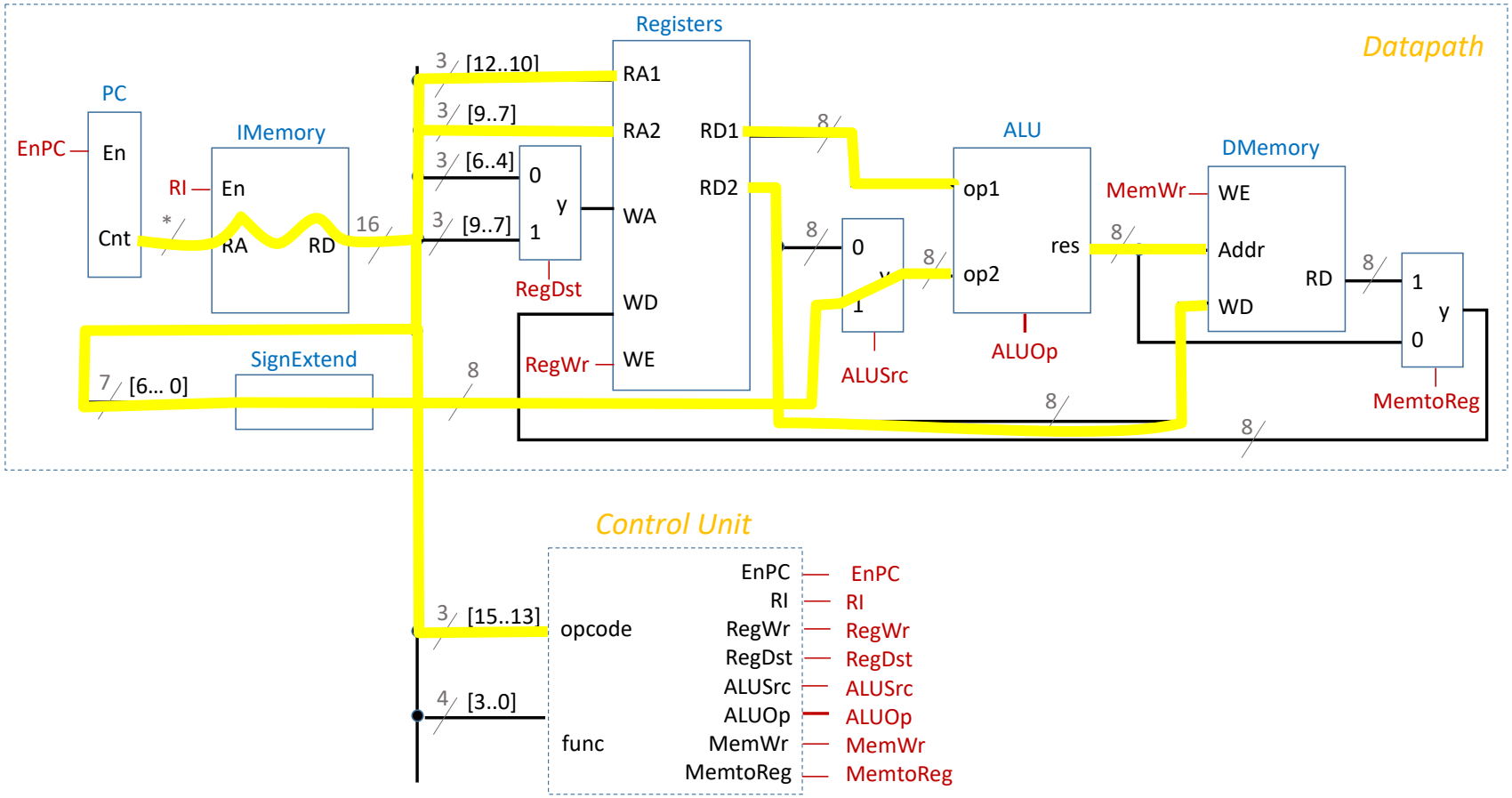


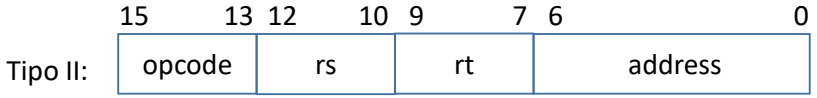
Fluxo de execução de instruções tipo II – LW



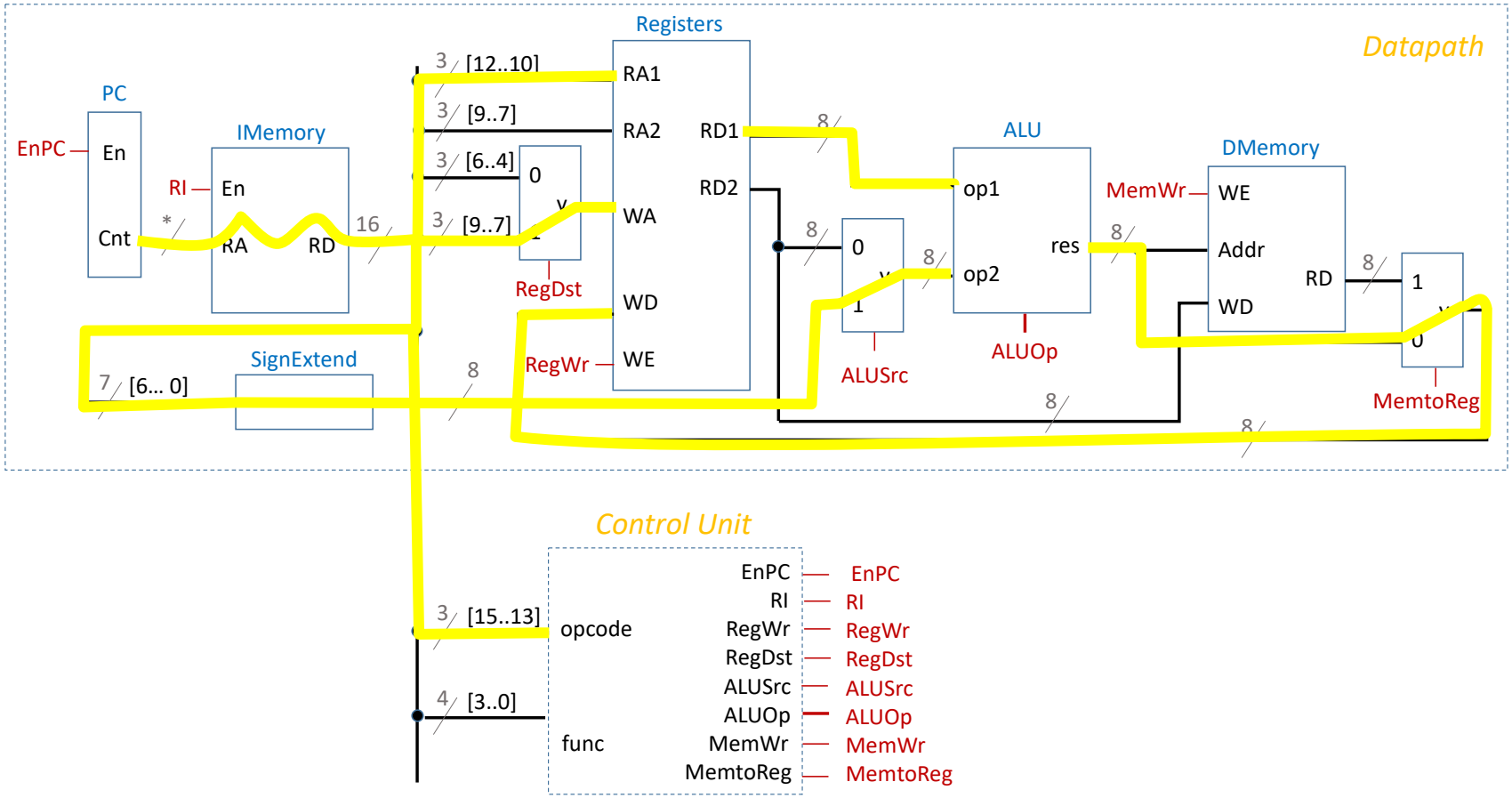


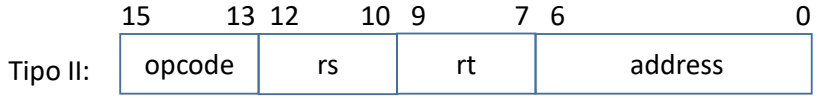
Fluxo de execução de instruções tipo II – SW





Fluxo de execução de instruções tipo II – **ADDI**

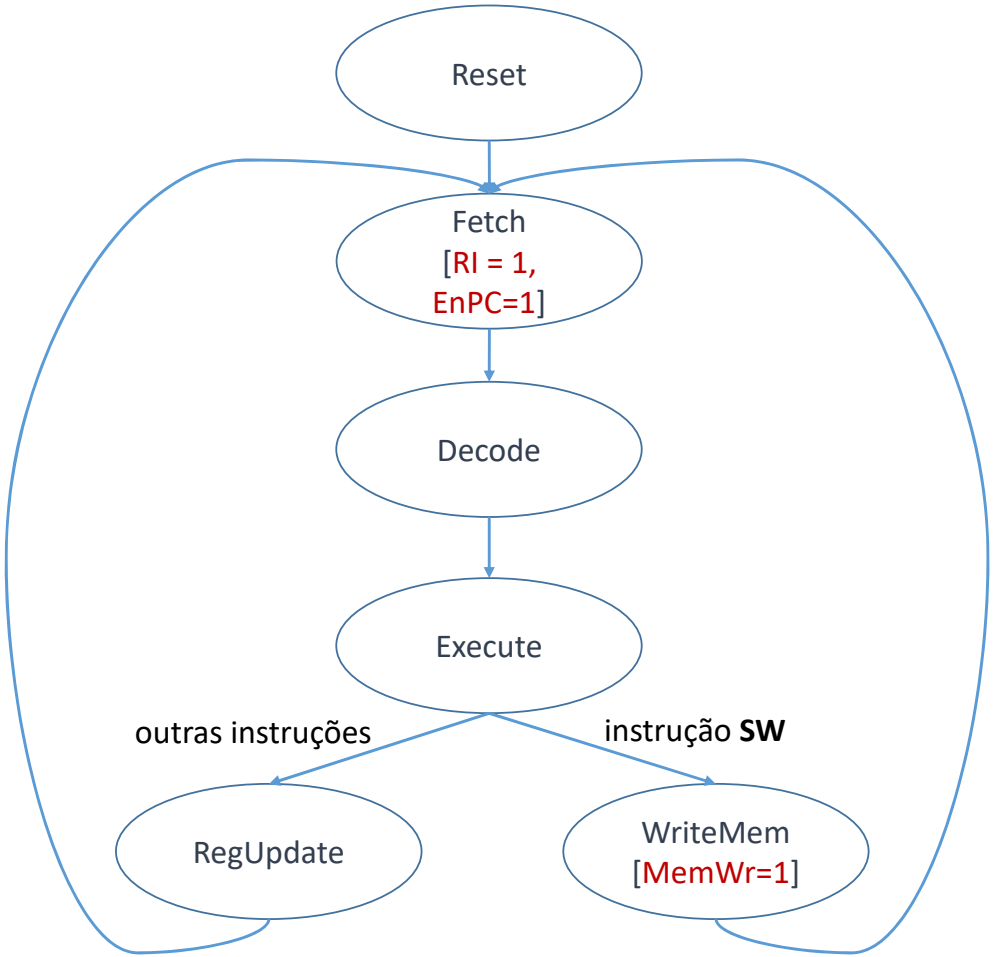




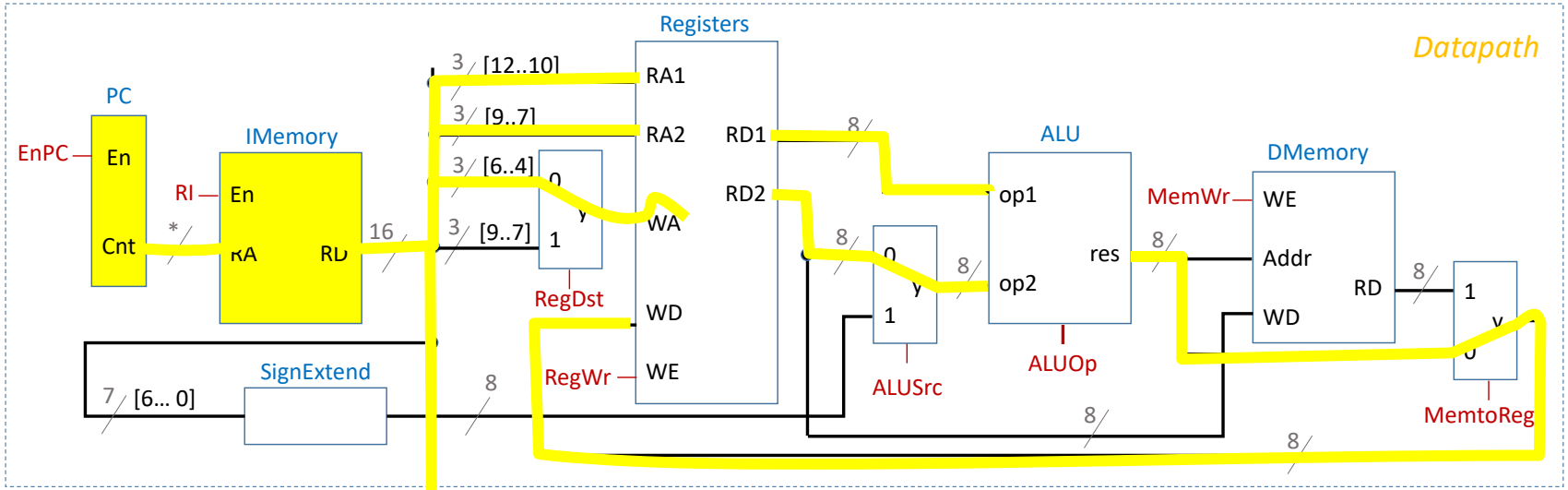
Unidade de controlo

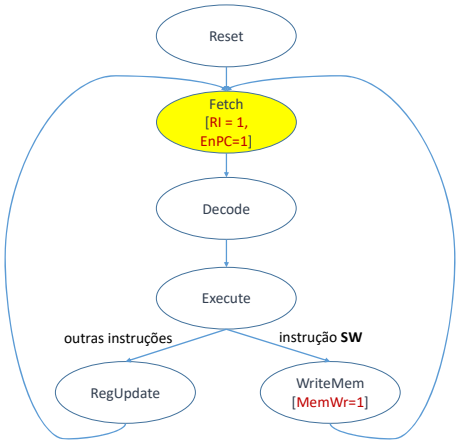
- Determina o que a unidade de execução deve fazer, gerando sinais de controlo adequados para cada fase de execução
- Para tal:
  - Força a leitura de uma instrução da memória de instruções e determina o endereço da próxima instrução *(Fetch)*
  - Descodifica o código de instrução *opcode* e o código de operação *func* *(Decode)*
  - Gera sinais de controlo necessários para executar a instrução *(Execute)*
  - Grava o resultado num registo ou na memória, completando a execução da instrução corrente *(RegUpdate / WriteMem)*

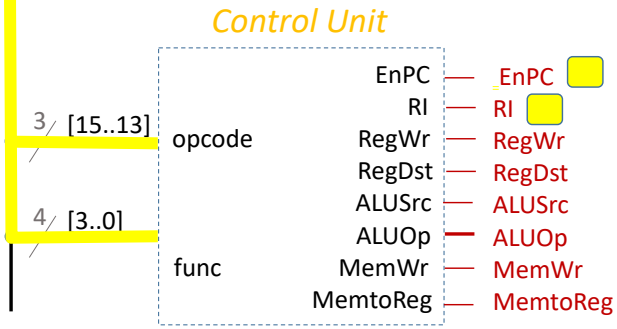
Fases de execução de uma instrução



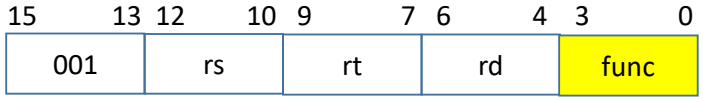
Fluxo de execução de instruções tipo I - *Fetch*



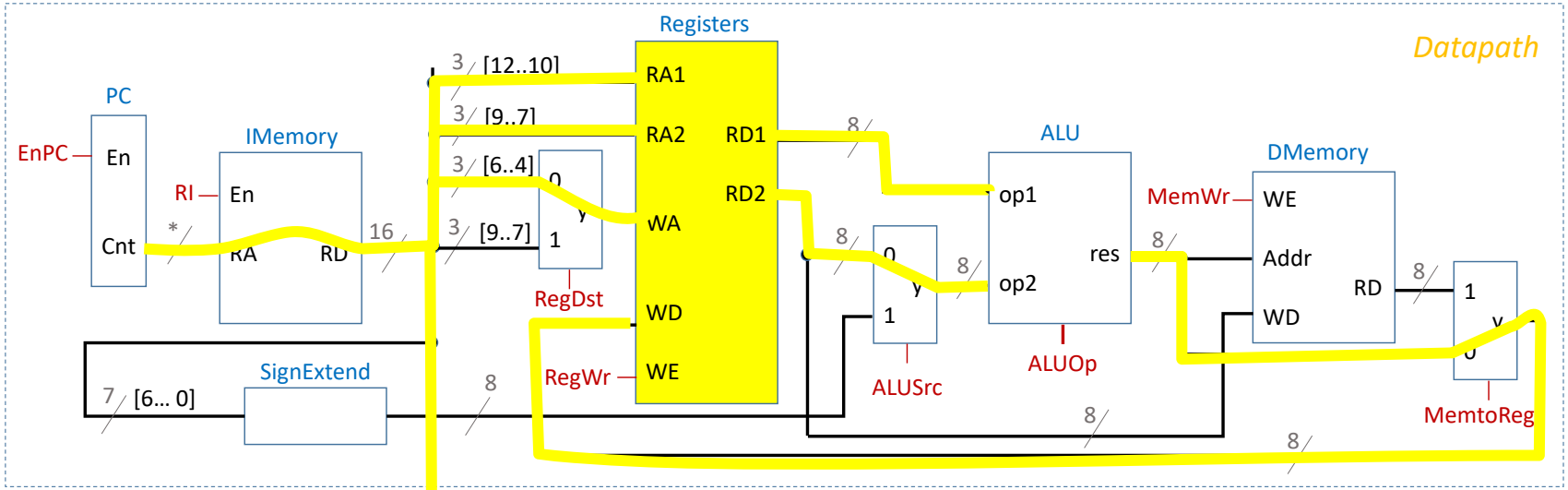


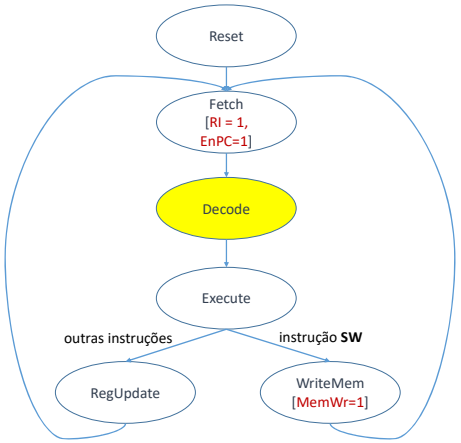


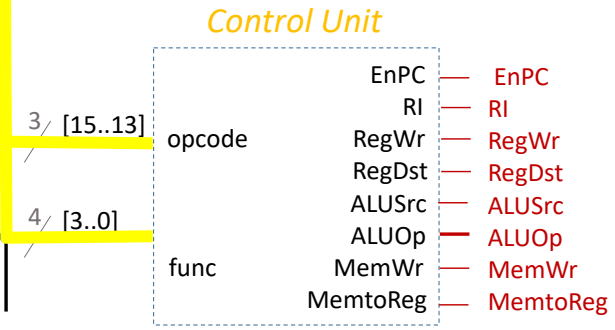
Tipo I:

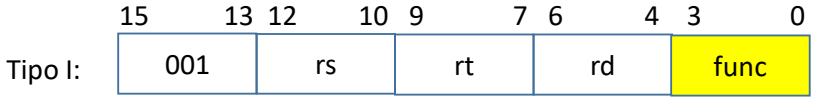


Fluxo de execução de instruções tipo I - *Decode*

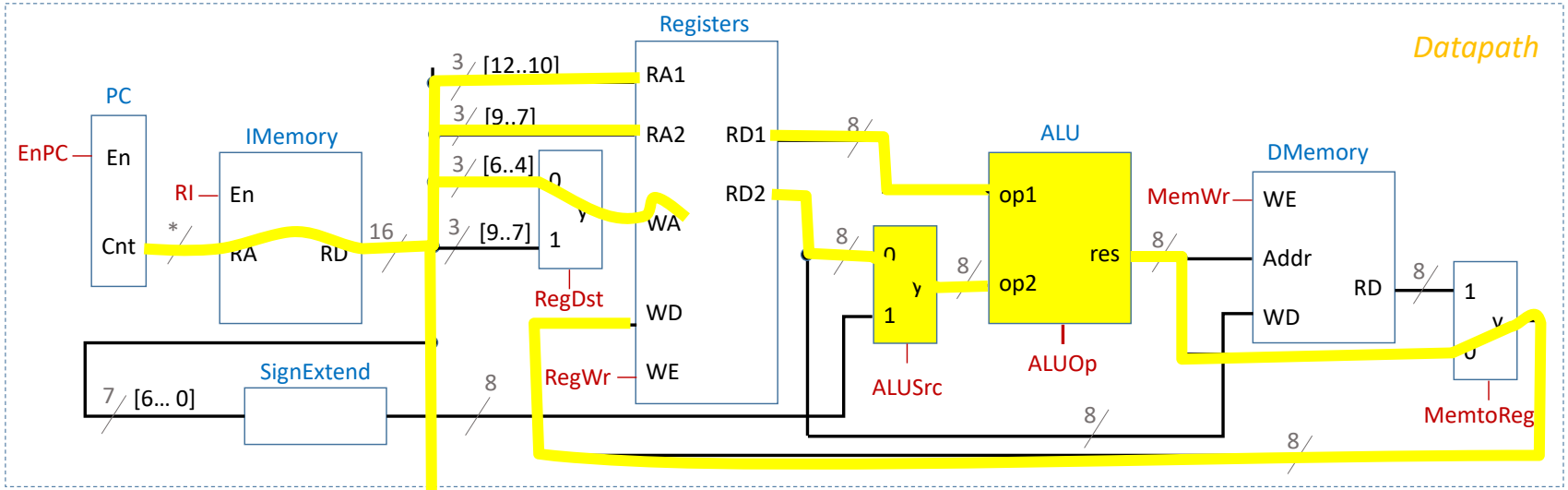


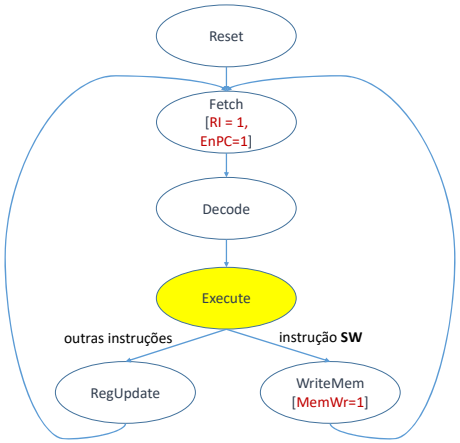


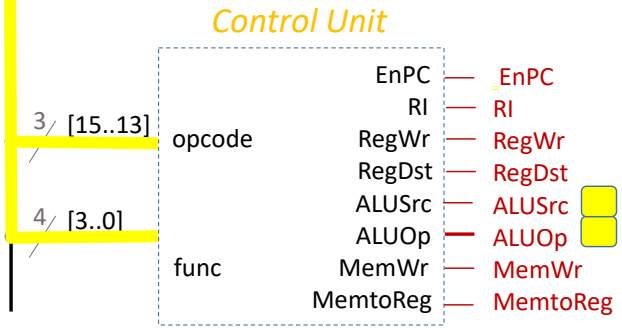


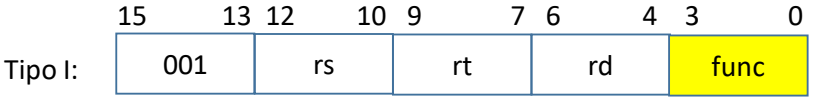


Fluxo de execução de instruções tipo I - *Execute*

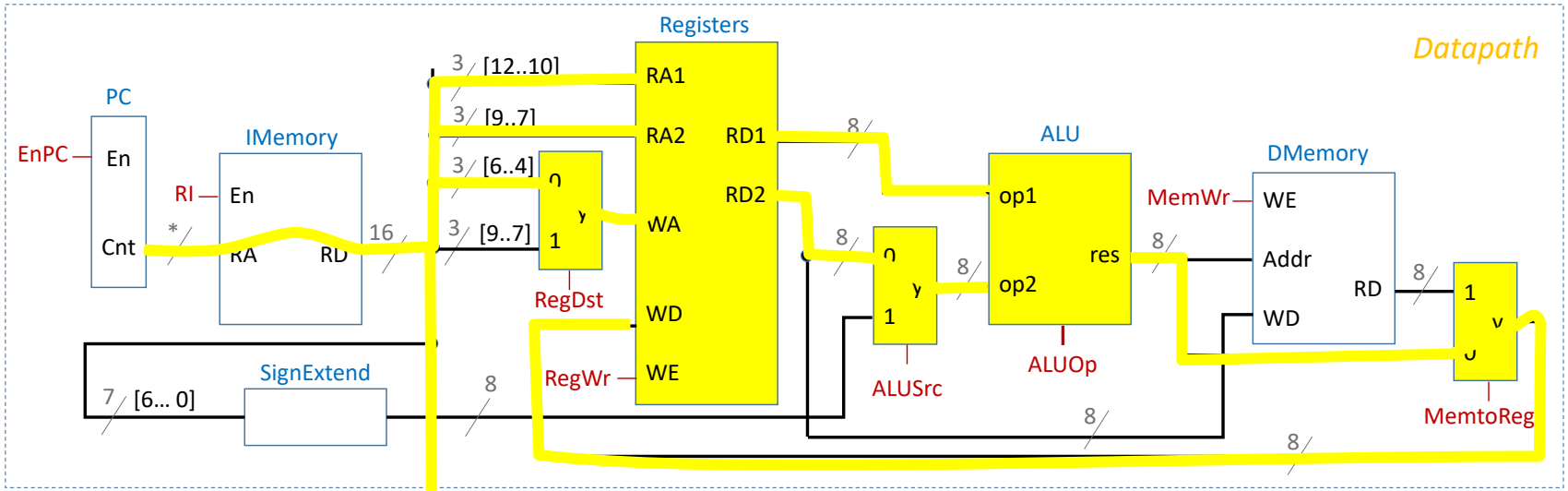


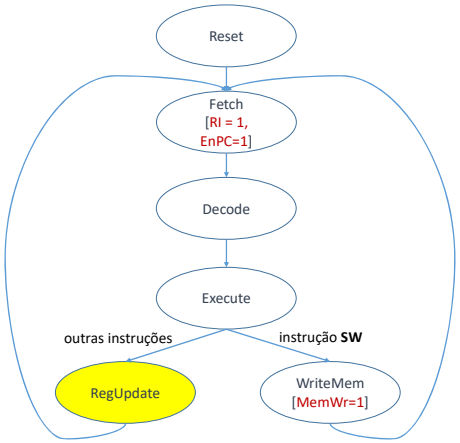


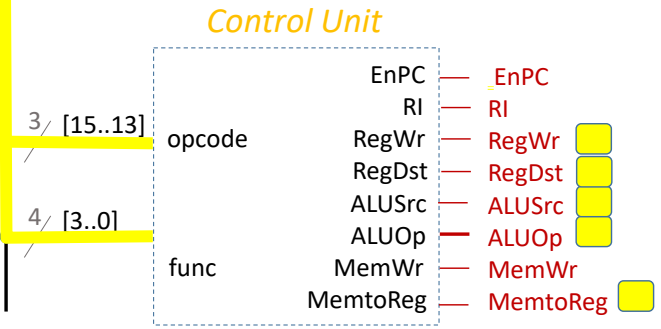


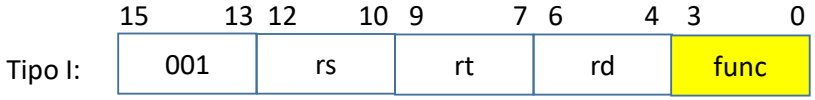


Fluxo de execução de instruções tipo I - *RegUpdate*

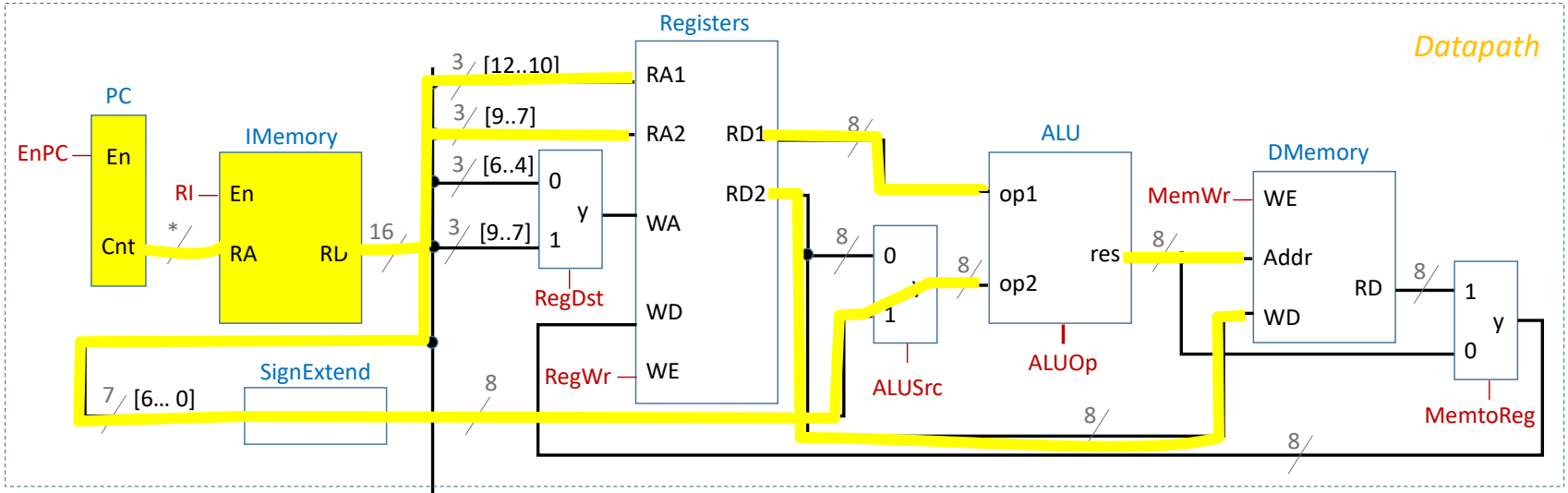


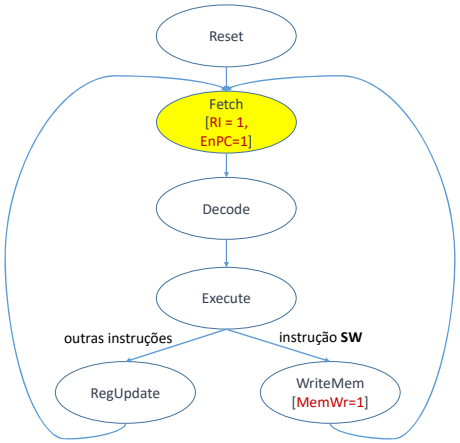


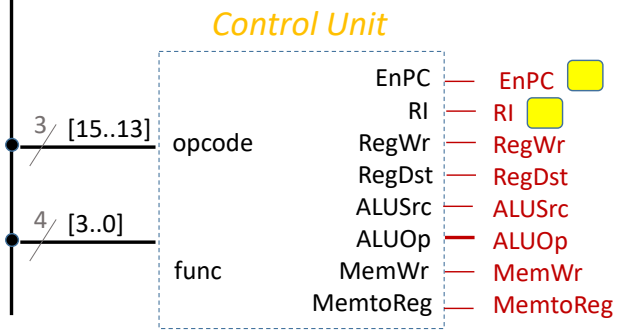




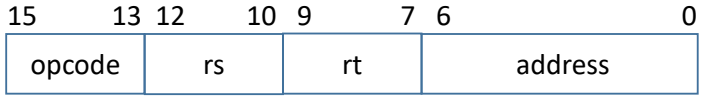
Fluxo de execução de instrução **SW** - *Fetch*



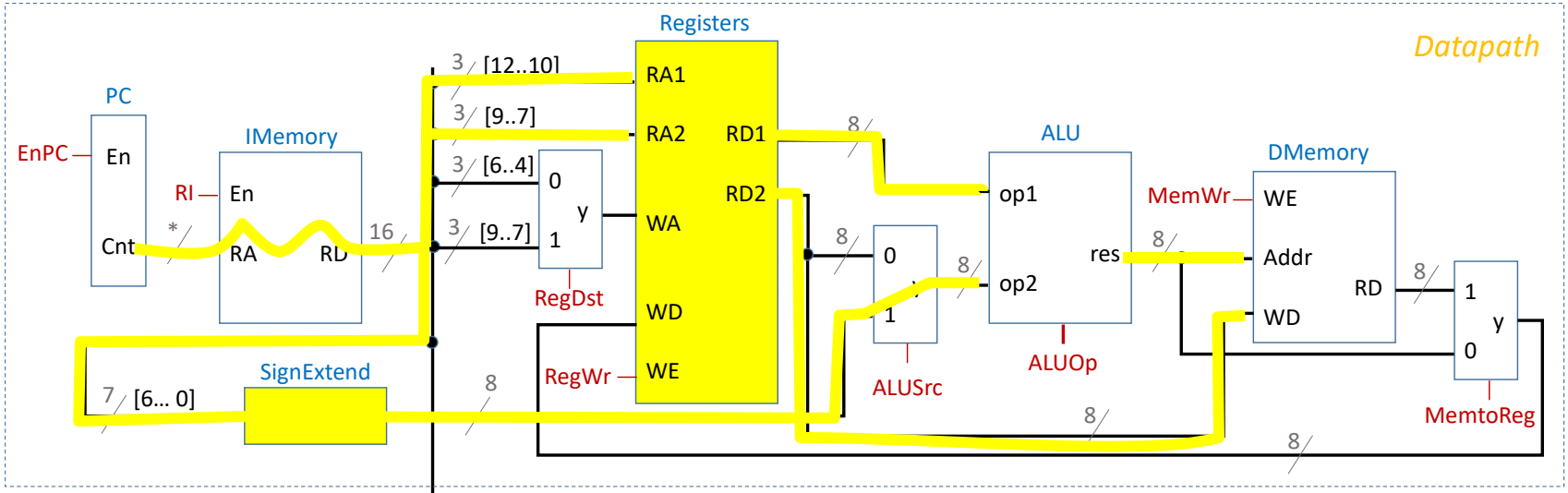


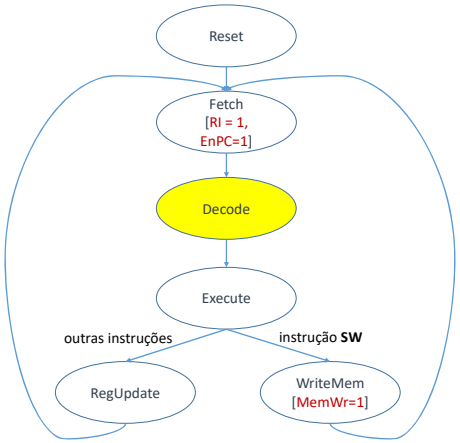


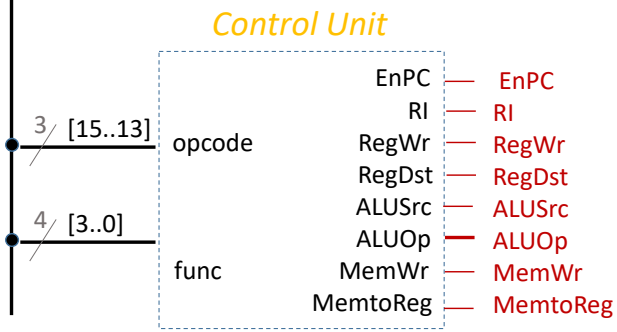
Tipo II:



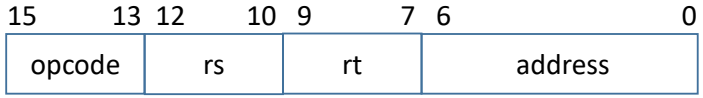
Fluxo de execução de instrução **SW** - *Decode*



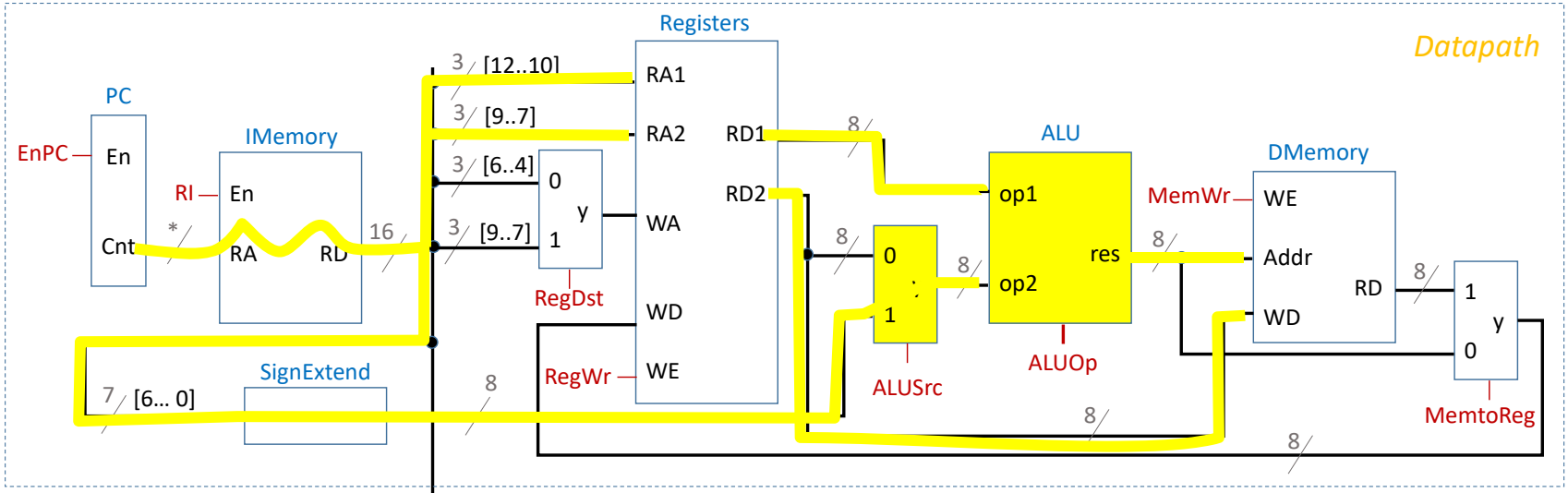


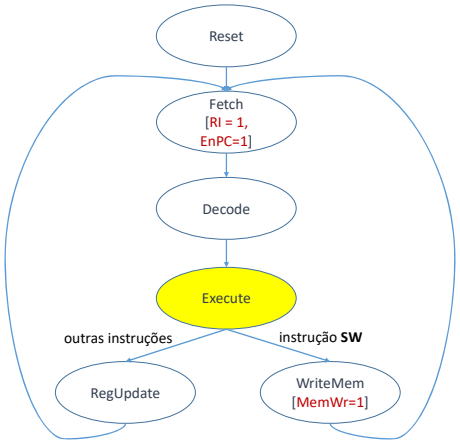


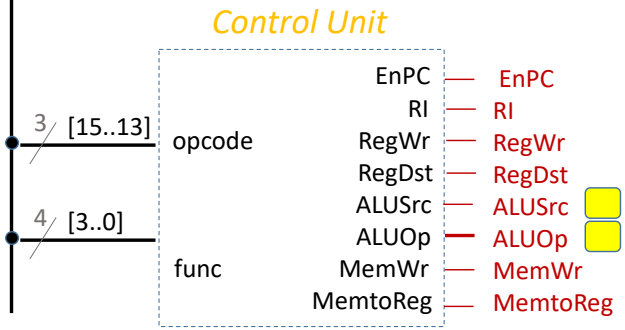
Tipo II:



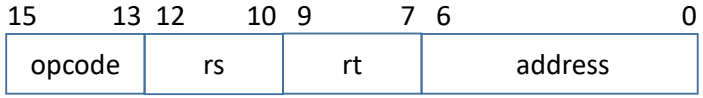
Fluxo de execução de instrução **SW** - *Execute*



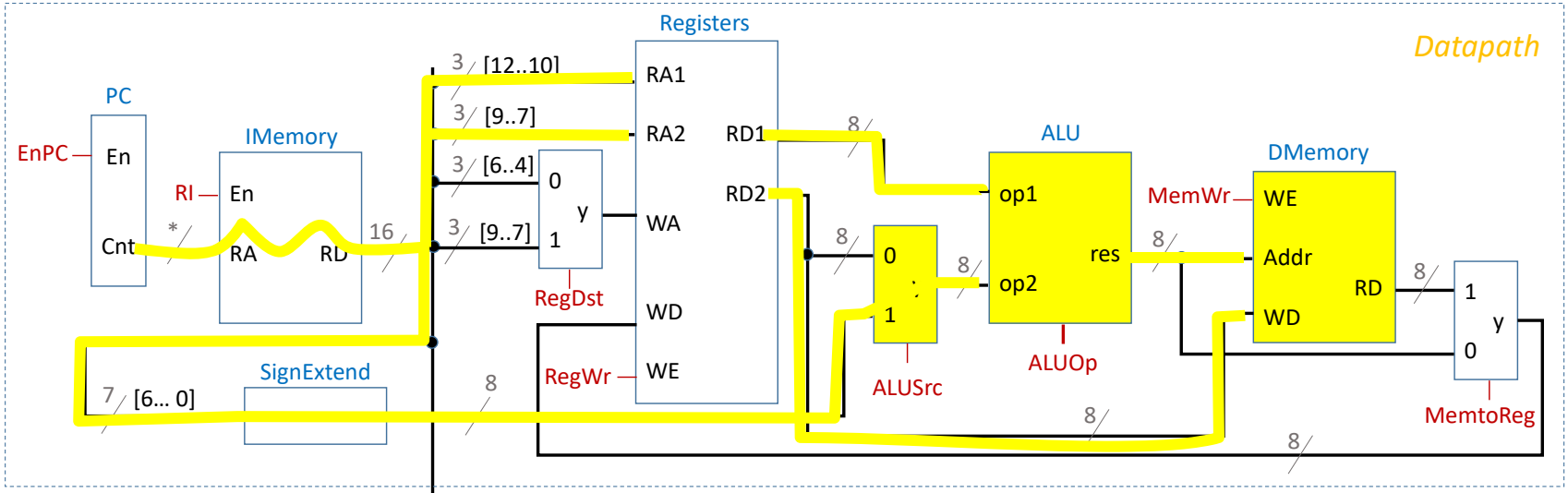


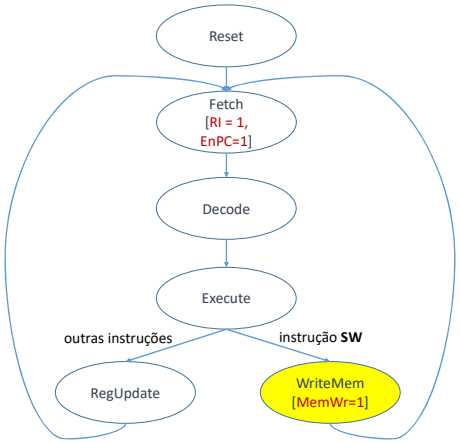


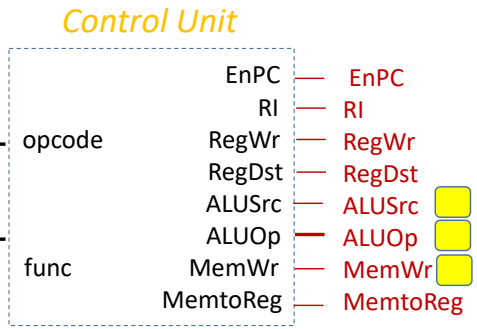
Tipo II:



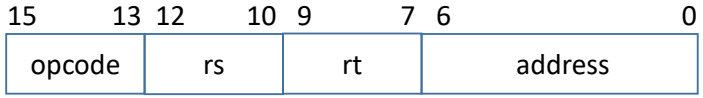
Fluxo de execução de instrução **SW** - *WriteMem*







Tipo II:



Bom Trabalho!