Ciclo Busca – Decodifica – Executa
Projeto da Via de Dados

Rápido Flashback...

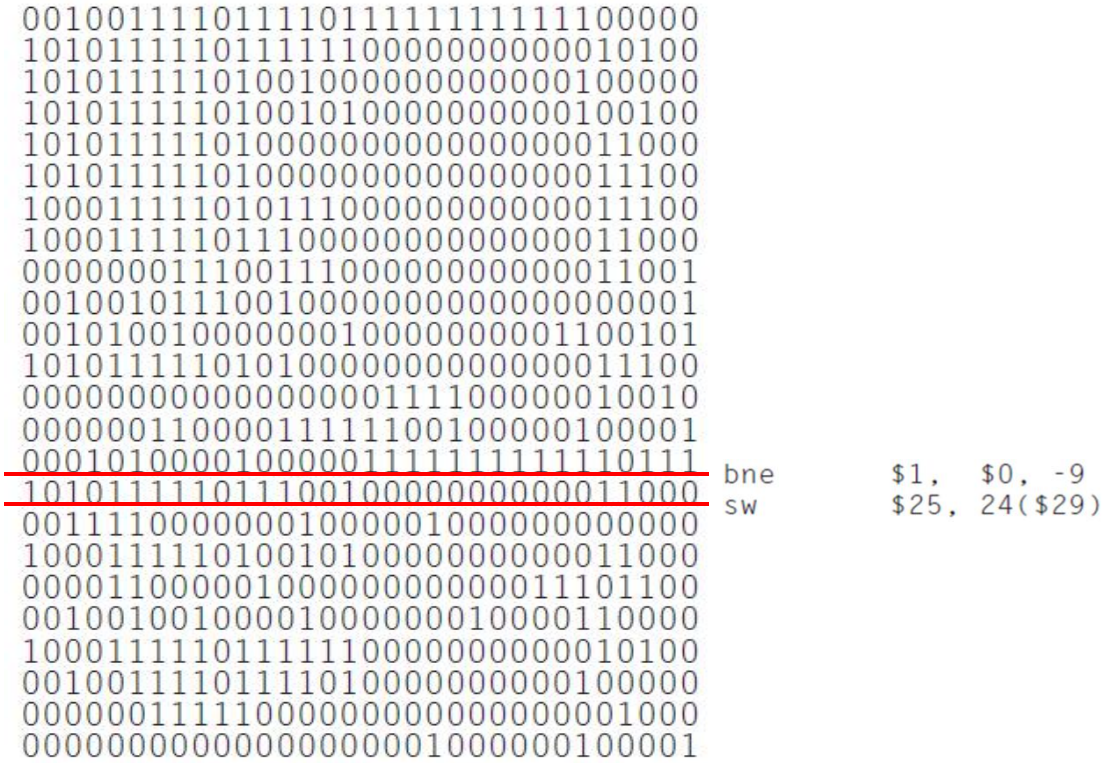
Ciclo de busca & decodifica & executa

Código em linguagem C

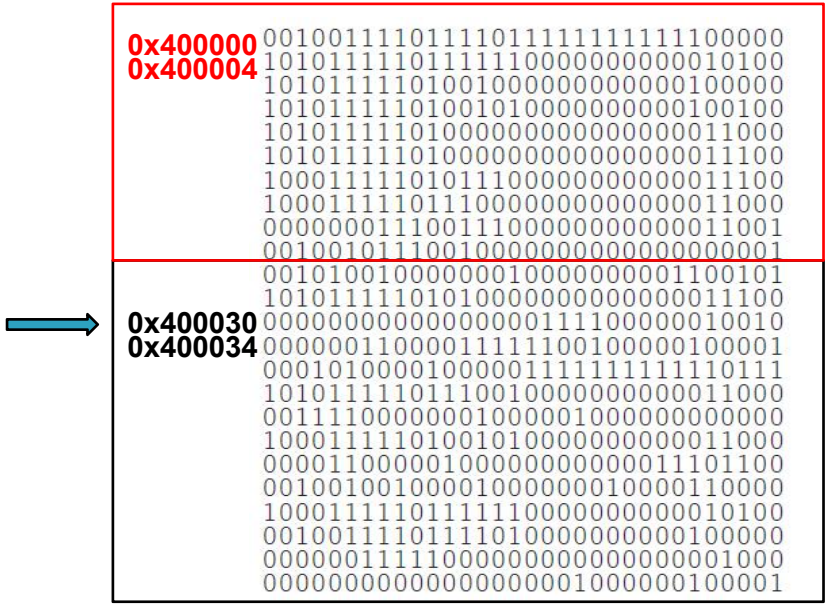
```
#include <stdio.h>

int
main (int argc, char *argv[])
{
    int i;
    int sum = 0;

    for (i = 0; i <= 100; i = i + 1) sum = sum + i * i;
    printf ("The sum from 0 .. 100 is %d\n", sum);
}
```

Trecho de Código para computar e imprimir a soma dos quadrados dos inteiros entre 0 e 100.

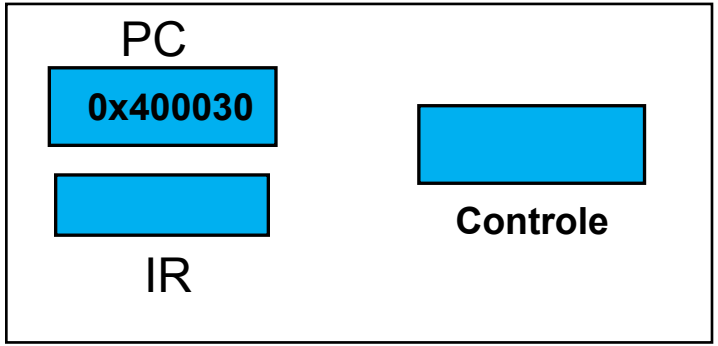
Linguagem de máquina MIPS



Ciclo de Busca-Execução

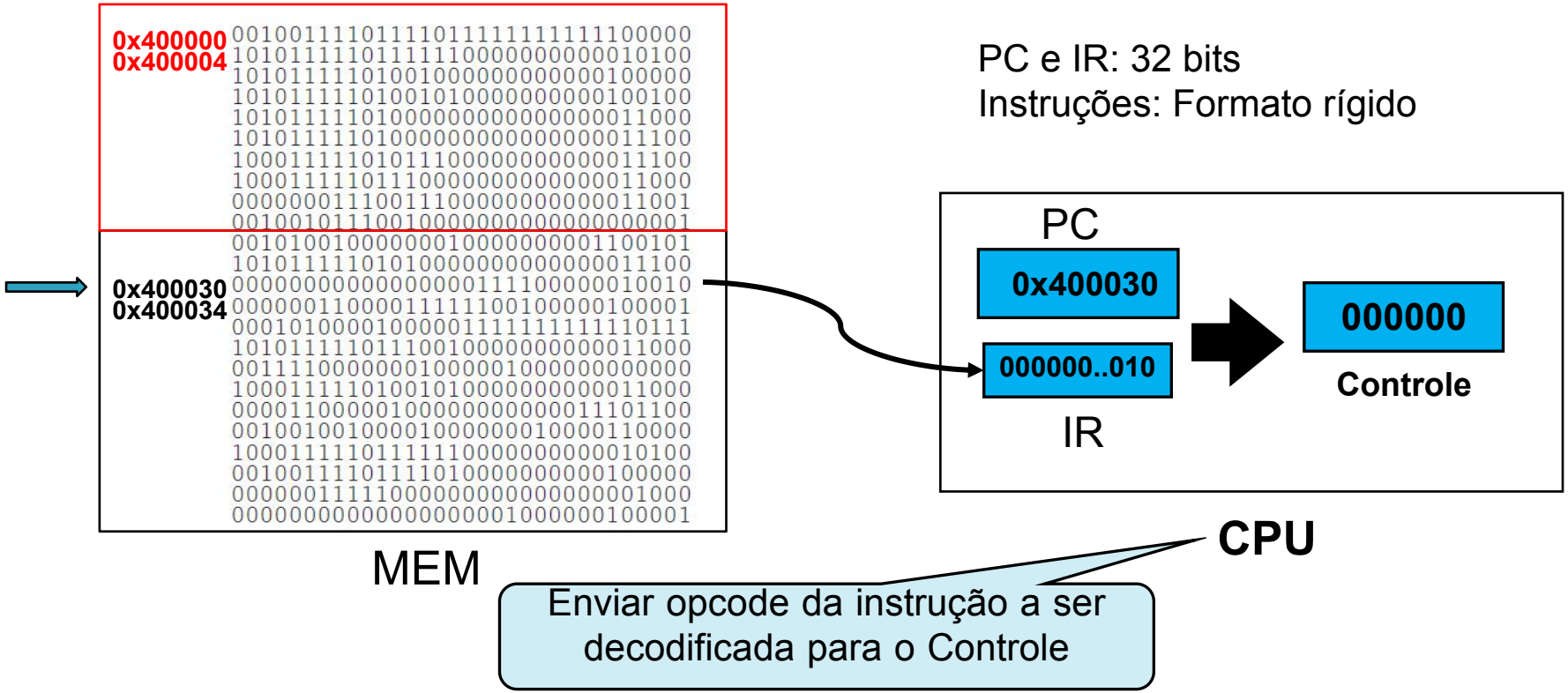


PC e IR: 32 bits
Instruções: Formato rígido

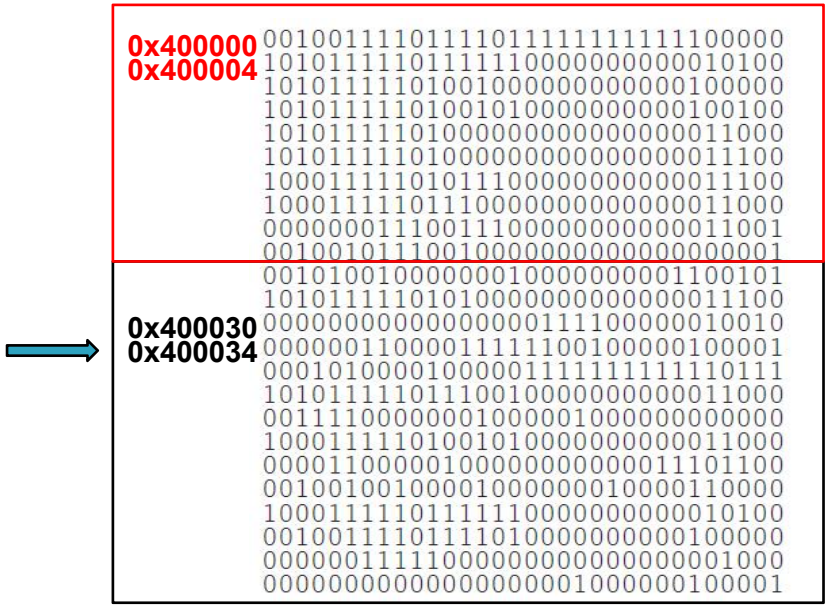


Buscar uma instrução
no segmento de código

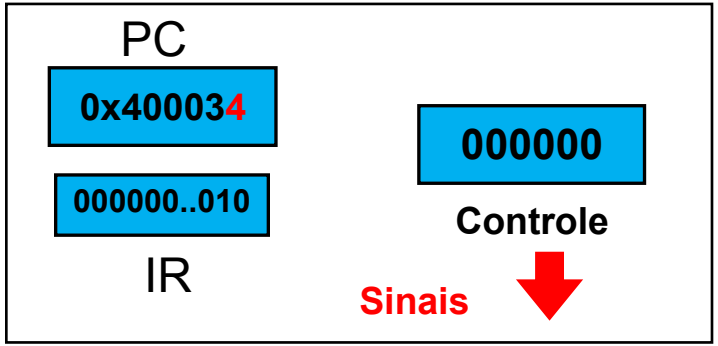
Ciclo de Busca-Execução



Ciclo de Busca-Execução

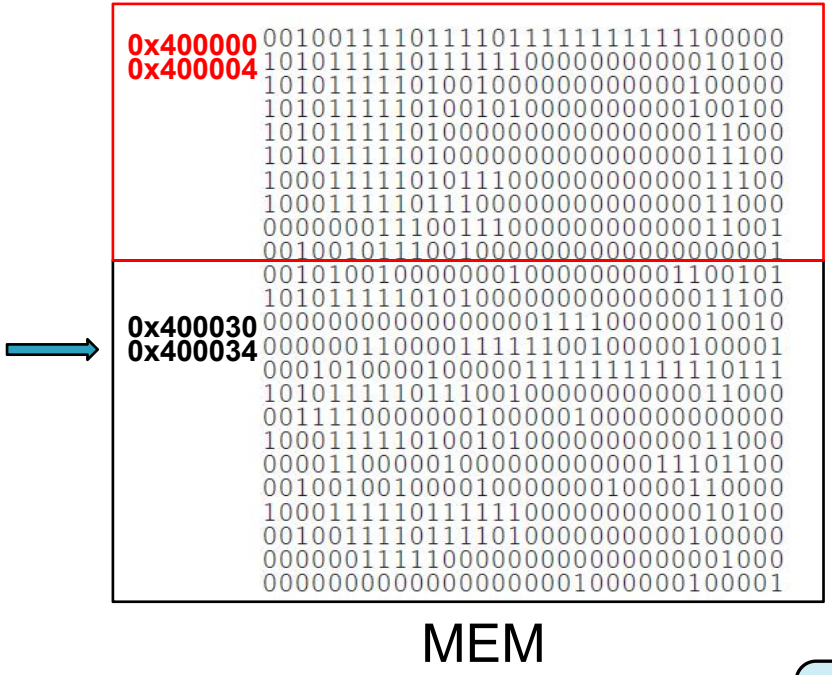


PC e IR: 32 bits
Instruções: Formato rígido

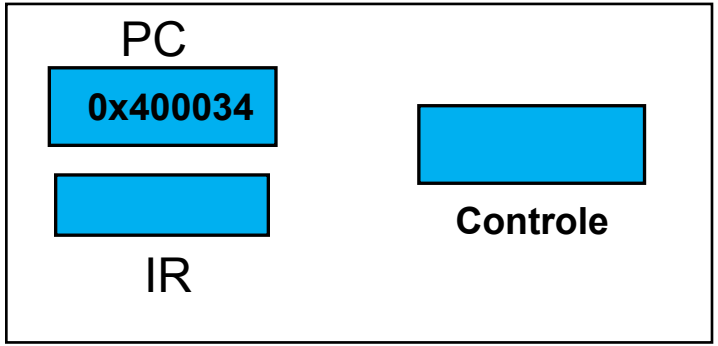


Gerar sinais de controle de partes da CPU & Incrementar PC

Ciclo de Busca-Execução



PC e IR: 32 bits
Instruções: Formato rígido



Buscar uma instrução
no segmento de código

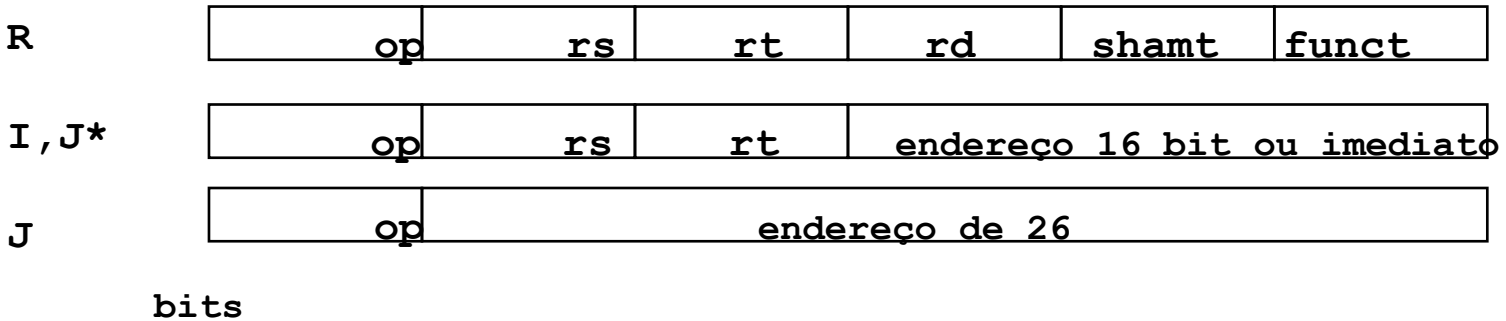
Na arquitetura MIPS

- Os dois passos iniciais são exatamente iguais para todas as classes de instruções:
  - Enviar o PC para o endereço de memória que contém a próxima instrução e trazê-la da memória;
  - Usar os campos (bits) da instrução para identificar qual(is) registrador(es) – 1 ou 2, dependendo da instrução;
- Passos seguintes dependem da classe (tipo) de instrução a ser executada;
- Vantagem MIPS: as execuções são bastante parecidas, o que agiliza a decodificação e execução, pois simplifica o hardware de controle.

Sumário instruções MIPS

Instrução	Significado
`add $s1,$s2,$s3`	`$s1 = $s2 + $s3`
`sub $s1,$s2,$s3`	`$s1 = $s2 – $s3`
`lw $s1,100($s2)`	`$s1 = Memory[$s2+100]`
`sw $s1,100($s2)`	`Memory[$s2+100] = $s1`
`bne $s4,$s5,L`	Prox. instr. label `L` se `$s4 != $s5`
`beq $s4,$s5,L`	Prox. instr. label `L` se `$s4 == $s5`
`j L`	Prox. instr. label `L`

- Somente 3 formatos possíveis:



Sumário: Registradores MIPS

	Operandos MIPS	
Nome	**Exemplo**	**Comentários**
32 registradores	`$s0-$s7, $t0-$t9, $zero, $a0-$a3, $v0-$v1, $gp, $fp, $sp, $ra, $at`	Localizações rápidas de dados. No MIPS, os dados devem estar nos registradores aritméticos. O $zero no MIPS armazena sempre igual a 0. Registrador $at é reservado para o montador manipular constantes grandes.
2^{30} memory words	Memory[0], Memory[4], ..., Memory[4294967292]	Acessado somento por instruções de transferência de dados. MIPS usa endereços de bytes, e palavras sequenciais diferenciam de 4. Memória armazena estruturas de dados, como vetores, e registros encadeados, como aqueles salvos em procedure calls.

PROCESSADOR MIPS

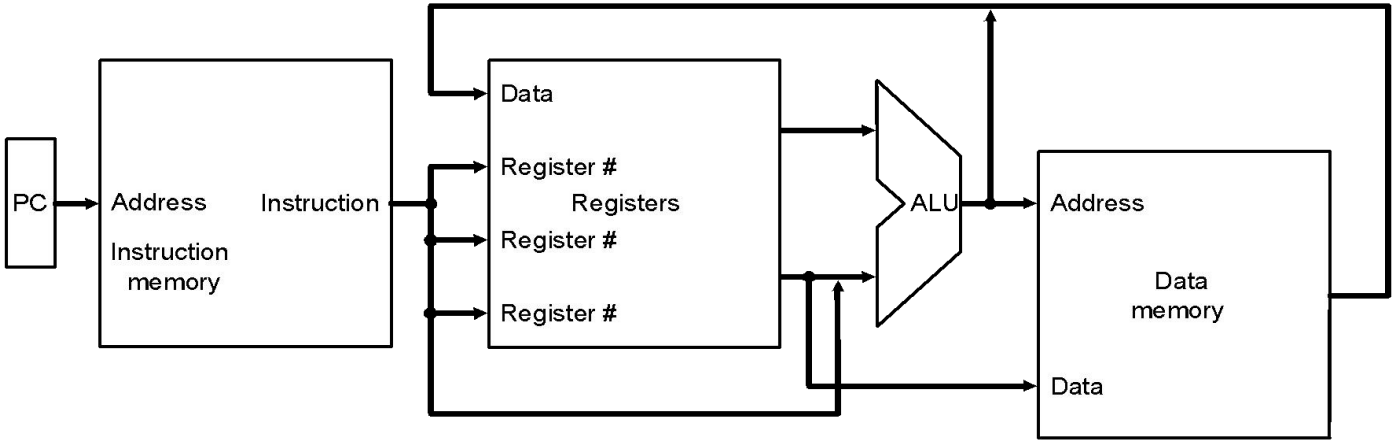
CPU = [Via de dados] + [Controle] + [Registradores e Cache]

Projeto/Implementação da CPU

- Projeto simplificado para dar suporte (apenas) às instruções:
  - Referência à memória: `lw, sw`
  - Lógicas e aritméticas: `add, sub, and, or, slt`
  - Controle de Fluxo: `beq, j`
- Implementação genérica:
  - Usa o contador de programa (PC) para fornecer o endereço da instrução;
  - Pega a instrução na memória;
  - Lê e escreve registradores;
  - Usa campos da instrução para decidir o que fazer (decodificação).
- Todas as instruções usam a ULA após ler os registradores!

Detalhes da implementação

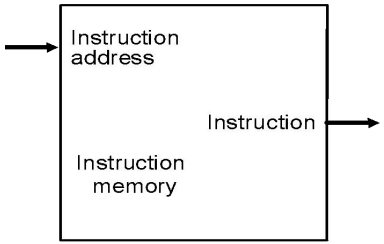
- Visão Geral:



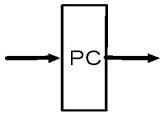
- Dois tipos de unidades funcionais:
  - Elementos que **processam dados** (circuitos combinacionais, sem memória)
  - Elementos que **armazenam estado** do programa (circuitos sequenciais, com memória)

Simplificação da Implementação

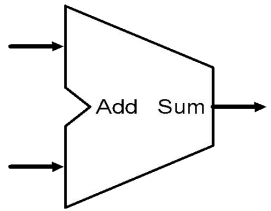
- Unidades funcionais básicas usadas na construção da CPU



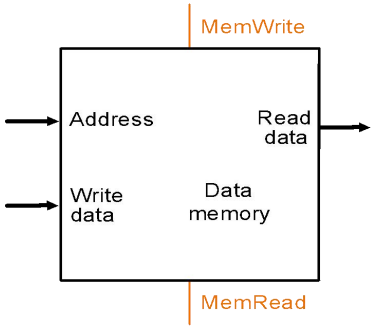
a. Instruction memory



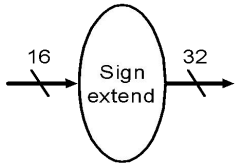
b. Program counter



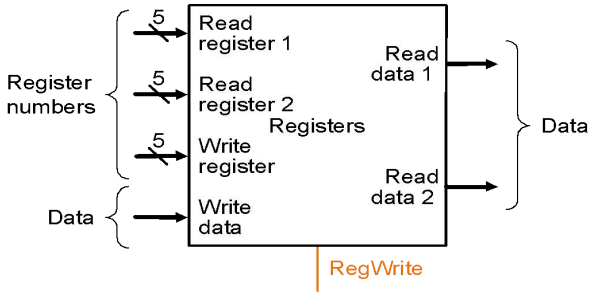
c. Adder



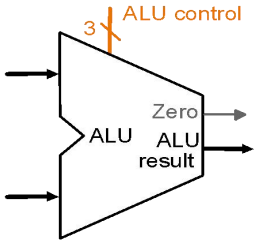
a. Data memory unit



b. Sign-extension unit



a. Registers

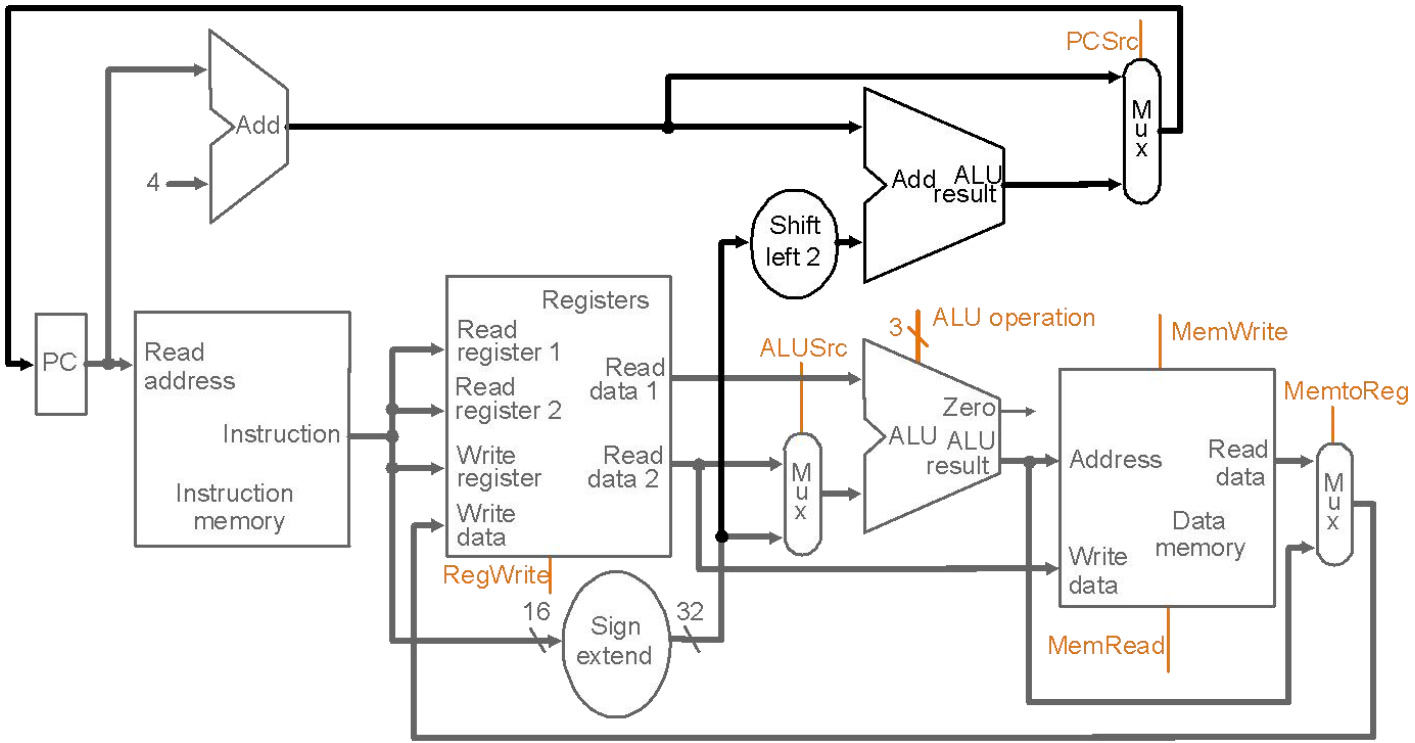


b. ALU

CONTROLE

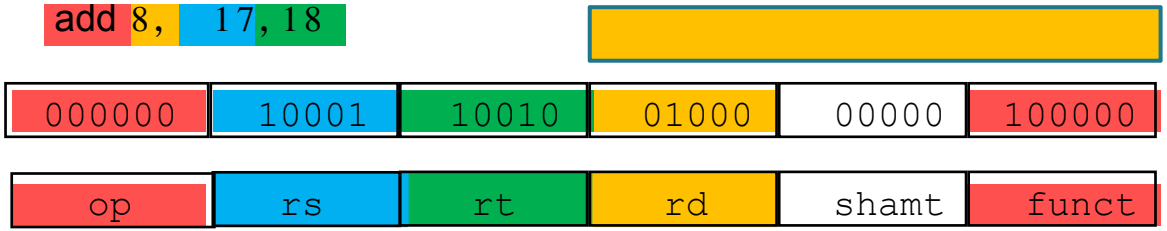
DADOS

Fluxo: Multiplexadores



Controle

- Seleção de operações a serem realizadas (ALU, read/write, etc.)
- Controle do fluxo de dados (entradas dos multiplexadores)
- Informações codificadas com os 32 bits da instrução
- Exemplo:



- Operação da ULA é baseada no tipo de instrução (campo `op`) e no código da função (campo `funct`), para instruções do tipo R

Controle: Execução na ULA

- O que a ULA deveria fazer com a instrução abaixo?

```
lw $1, 100($2)
```

35	2	1	100
op	rs	rt	16 bit offset

- Controle de entrada da ULA (define a operação aritmética)

```
000     AND                 001     OR
010     add                 110     subtract
111     set-on-less-than
```

- Outros bits: tipo de instrução:

```
00:lw, sw        01:beq   10:aritmética
```

CPU MIPS (sem jumps)