



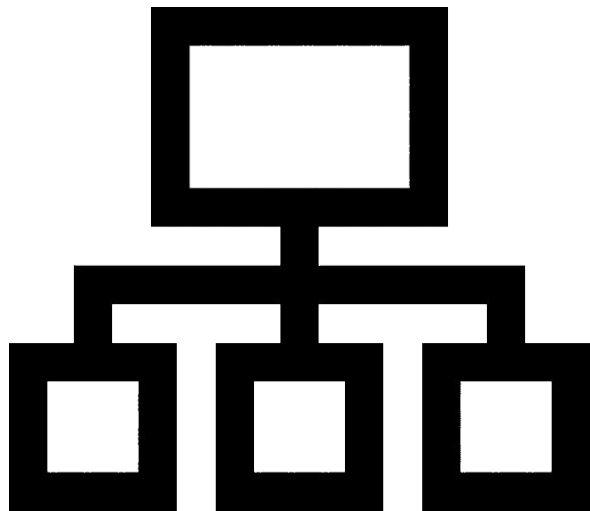
Arquitetura de Computadores

Noções básicas de arquitetura de computadores - Aula 2

Aprofunde-se no livro texto!

Leia o capítulo 11 do Livro do Uyemura.





- ▶ Aprofundamos a noção de projeto hierárquico
 - ▶ Especialmente na descrição de funcionamento de um computador
- ▶ Vimos como a noção hierárquica é coberta nas HDLs
 - ▶ Especialmente na linguagem Verilog
- ▶ Entendemos as principais etapas de execução de um programa
 - ▶ Discutimos os tipos de instruções e quais blocos funcionais que são ativados ao executá-las

- ▶ Veremos em mais detalhes o funcionamento de um processador. O esquemático geral será nosso guia de referência!
 - ▶ Suas características gerais
 - ▶ Estudo inicial da Unidade Central de Processamento
 - ▶ Componentes da Via de Dados (*Datapath*)
 - ▶ Instruções e Via de Dados
 - ▶ Unidade de Controle
 - ▶ Arquiteturas RISC e CISC

Características Gerais

Principais componentes de um computador

Circuitos de Entrada: Fornece dados para o computador via Teclado, mouse, unidades de disco, CD-ROM, scanners...

Circuitos de Saída: Codificam os resultados binários das operações para apresentá-los via monitor, unidade de disco no modo escrit:

Memória: Armazena programas, dados e tudo mais necessário, como o Sistema Operacional.

Via de dados: Representa o caminho que o dado percorre durante os eventos de processamento.

Unidade de Controle: Garante que o dado seja enviado para conjunto correto de circuitos, nas mais diversas operações suportadas na via de dados.

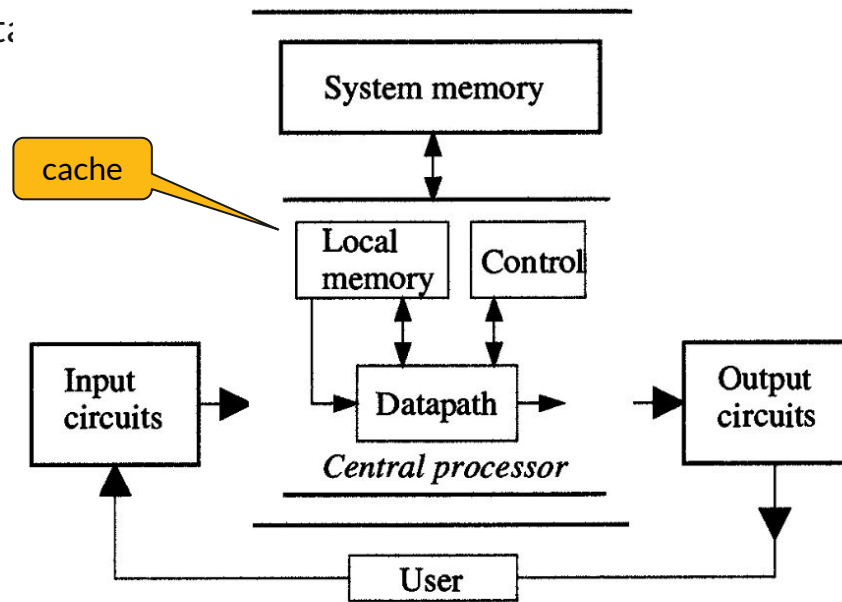


Figure 11.1 Major components of a computer

O que o computador pode fazer?

Podemos simplificar as operações que o computador realiza em duas principais:

- Movimentação de dados
- Execução de operações binárias
 - Funções lógicas: NOT, AND, XNOR, ... – usadas nas decisões
 - Funções aritméticas: soma, subtração, multiplicação...

Instrução → Cada operação que o computador realiza

Conjunto de Instruções → é o conjunto de todas as instruções suportadas
(ISA - *Instruction Set Architecture*)

Modelo de von Neumann

Composto por dois blocos principais:

- **Memória** → Armazena programa (lista de instruções) e dados
- **Unidade Central de Processamento** → via de dados + unidade de controle (+ registrador de instrução).

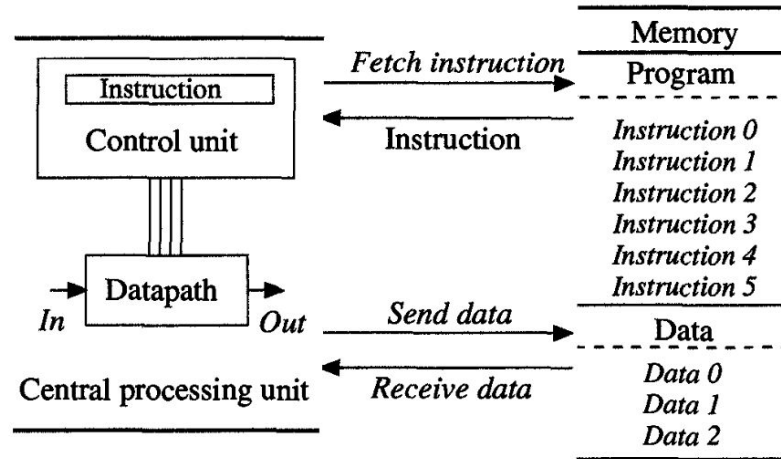


Figure 11.2 The von Neumann model of a programmable computer

Ciclos do Modelo von Neumann

O modelo von Neumann de um computador é baseado na repetição de um procedimento de quatro ciclos para execução do programa,. São eles:

1. **Busca de instrução** → Busca na memória a instrução que será armazenada na Unidade de Controle;
2. **Decodificação de instrução** → Processo de interpretar a instrução para determinar o que precisa ser feito dentro da CPU; A unidade de controle então “informa” à via de dados o que fazer;
3. **Execução da instrução** → A via de dados executa a operação, acessando as entradas, calculando, e apresentando os resultados;
4. **Armazenamento** → o resultado das operações é guardado novamente na memória.

$$t_{Inst} = t_{BI} + t_{DI} + t_{EX} + t_A$$

Programação

Programa: lista ordenada de comandos que ditam uma sequência de operações necessárias para efetuar uma determinada tarefa.

Linguagem de programação: permite descrever o programa em uma sintaxe específica, com uma semântica atrelada.

O computador só “entende” linguagem de máquina (0s e 1s).

Programas especiais, como o compilador, fazem a tradução da linguagem de programação para a linguagem de máquina.

A linguagem Assembly possui uma correspondência direta com cada comando binário em linguagem de máquina, facilitando seu entendimento.

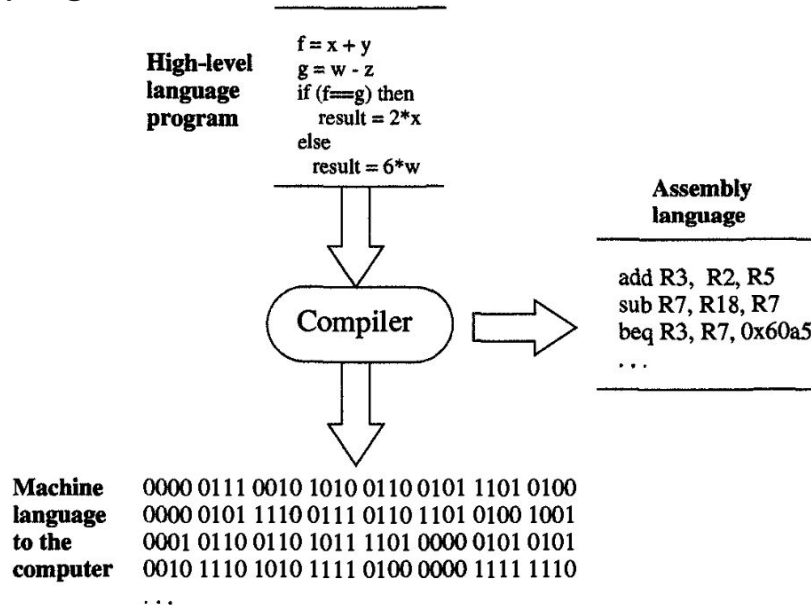


Figure 11.3 Levels of programming languages

Registradores do Computador

Registrador → é um conjunto unitário de célula de memória. São agrupados para armazenar toda uma palavra binária.

Atente-se à notação:

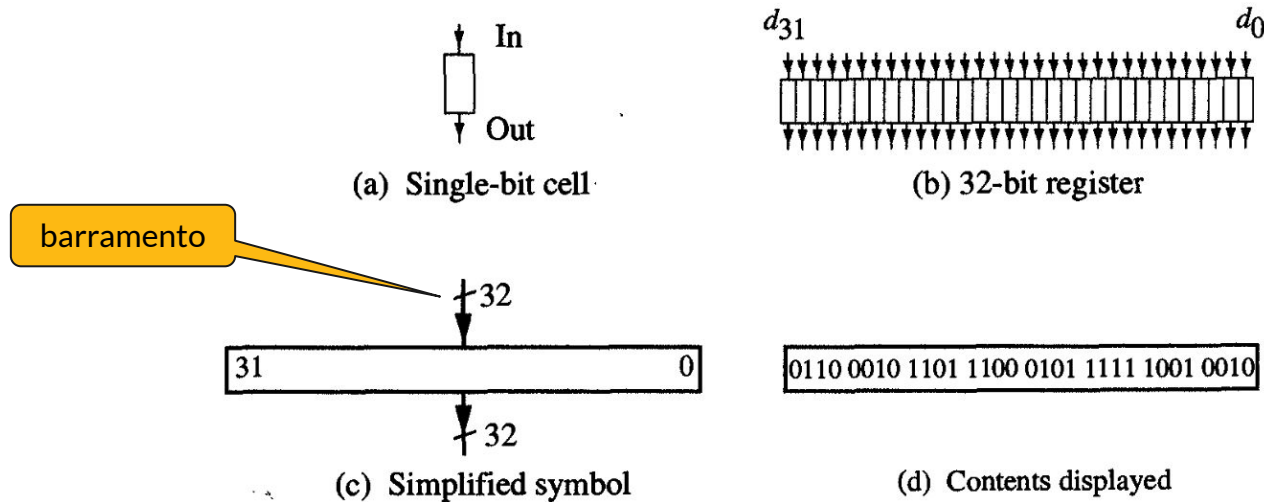


Figure 11.4 Schematic symbols used to represent registers

Estudo inicial da Unidade Central de Processamento

Circuito de busca de informação

Considere a sequência de instruções de um programa armazenado na memória, conforme segue

Address	Binary Instruction	Order
0400	01101100 11111010 11110000 11110000	<i>Inst 0</i>
0404	11000101 10110101 00001111 11110000	<i>Inst 1</i>
0408	01000110 10011111 10101010 10101010	<i>Inst 2</i>
0412	10010011 01101110 00110011 00110011	<i>Inst 3</i>
0416	10101001 01000101 11100011 11100011	<i>Inst 4</i>
0420	10001000 10001101 10011001 10011001	<i>Inst 5</i>
0424	11100010 10101001 11100010 10101010	<i>Inst 6</i>
0428	00100111 01101010 00110010 01011000	<i>Inst 7</i>
0432	10010010 11010011 10010011 01001001	<i>Inst 8</i>

Figure 11.5 A program sequence stored in memory

Circuito de busca de informação

- **Registrador de Instrução (IR)** guarda a palavra binária da instrução corrente
- **Contador de Programa (PC)** guarda o endereço de memória da próxima instrução que será buscada, controlando assim o fluxo de execução.

Para obter a próxima instrução, calculamos:

$$PC \leftarrow PC + X, \text{ (onde } X \text{ normalmente é 4)}$$

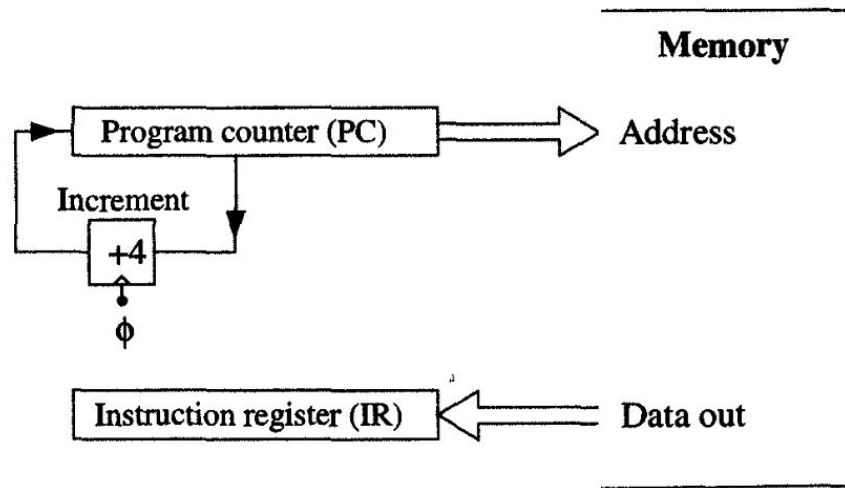
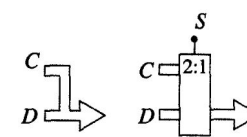
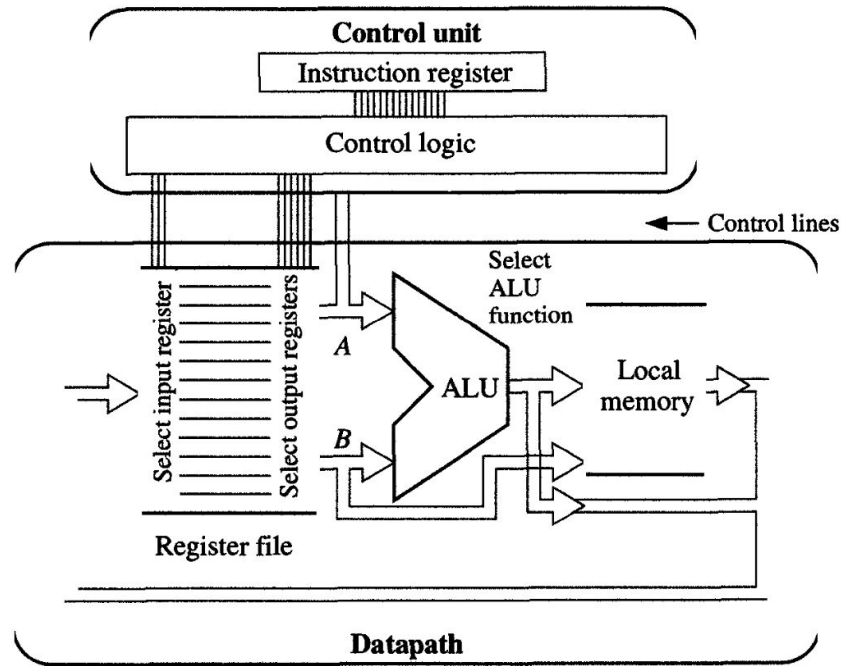


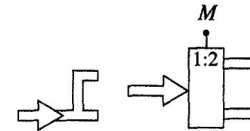
Figure 11.6 Operation of the instruction fetch (IF) network

Conceito de via de dados

A Unidade de Controle **acessa diretamente** o Registrador de Instrução. Os bits da instrução determinam quais **sinais de controle** a Lógica de Controle irá passar para as **linhas de controle**, determinando como se dará o fluxo na Via de Dados.



(a) MUX equivalents

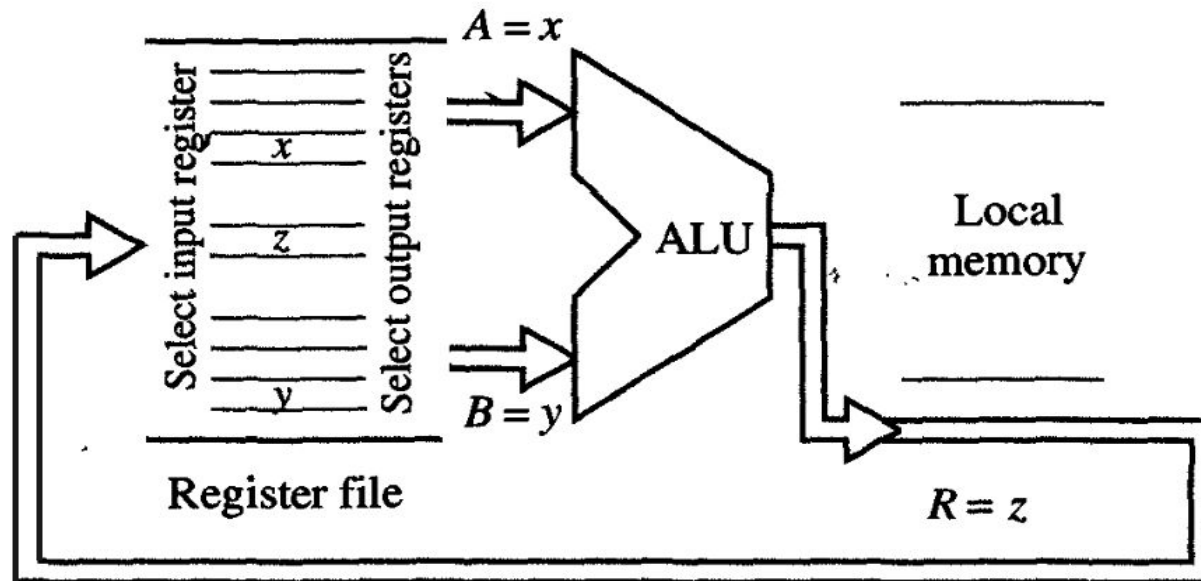


(b) DeMUX equivalents

Figure 11.7 The central processor consists of the datapath and the control unit

Operações da via de dados

Operações de Registrador para Registrador



$$R = A + B$$

$$R = A - B$$

$$R = A \cdot B$$

$$R = \bar{A}$$

Figure 11.9 Datapath for a register-to-register operation where data originates from the registers and the result is stored back into the registers

Operações da via de dados

Carregar (Load)

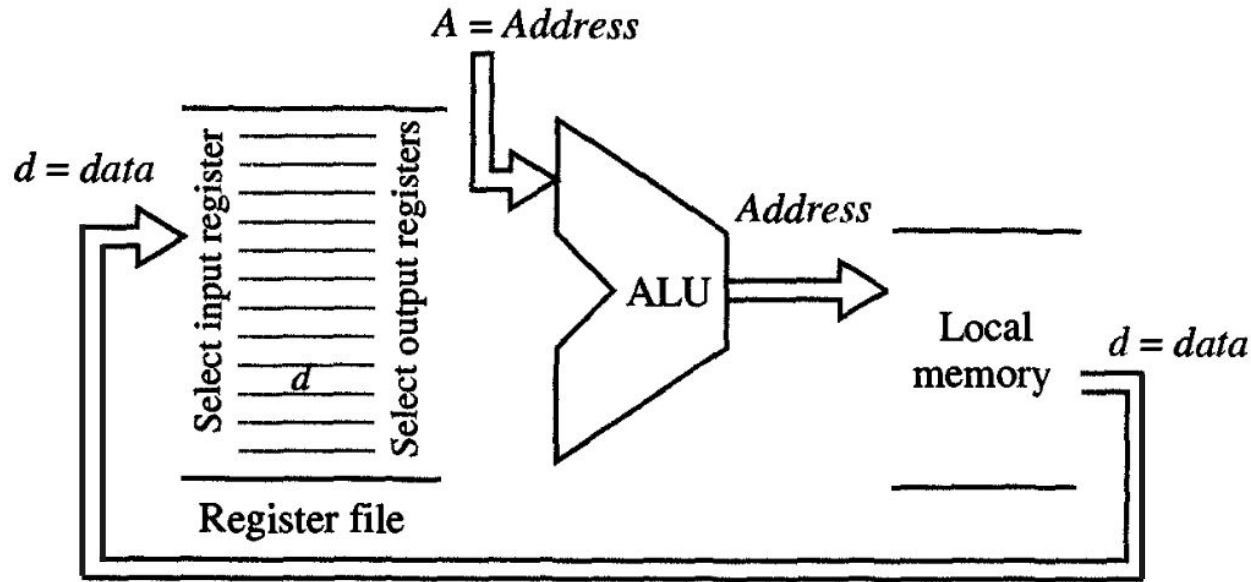


Figure 11.10 The load word instruction allows us to move a data word from the memory to a particular register

Operações da via de dados

Armazenar (Store)

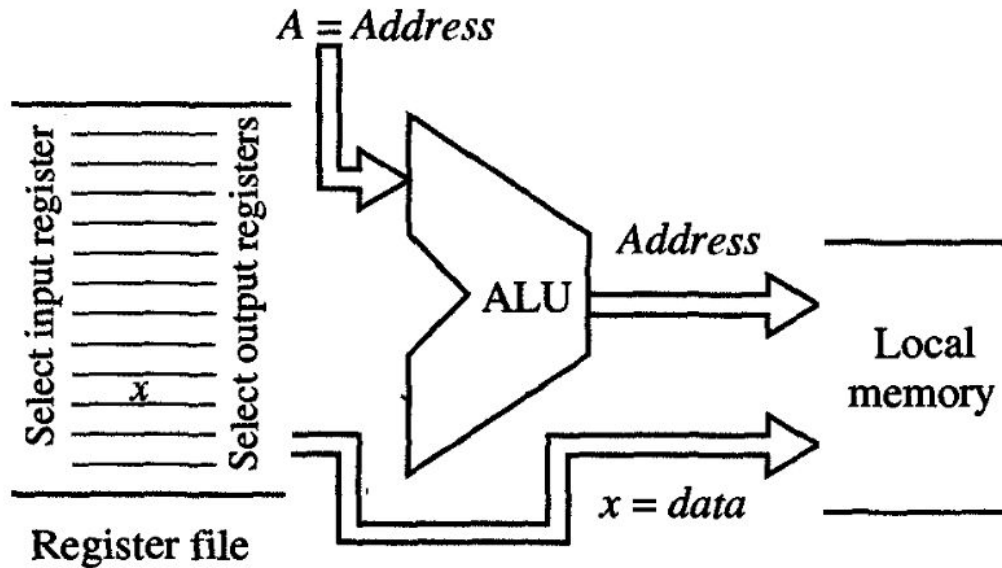


Figure 11.11 The store word instruction allows us to move a data word from a register and store it in the memory

Componentes da Via de Dados (*Datapath*)

Arquivo de Registradores (ou Banco de

No exemplo, um banco de registradores com 32 registradores de 32 bits cada, e barramentos de endereço de 5 bits.

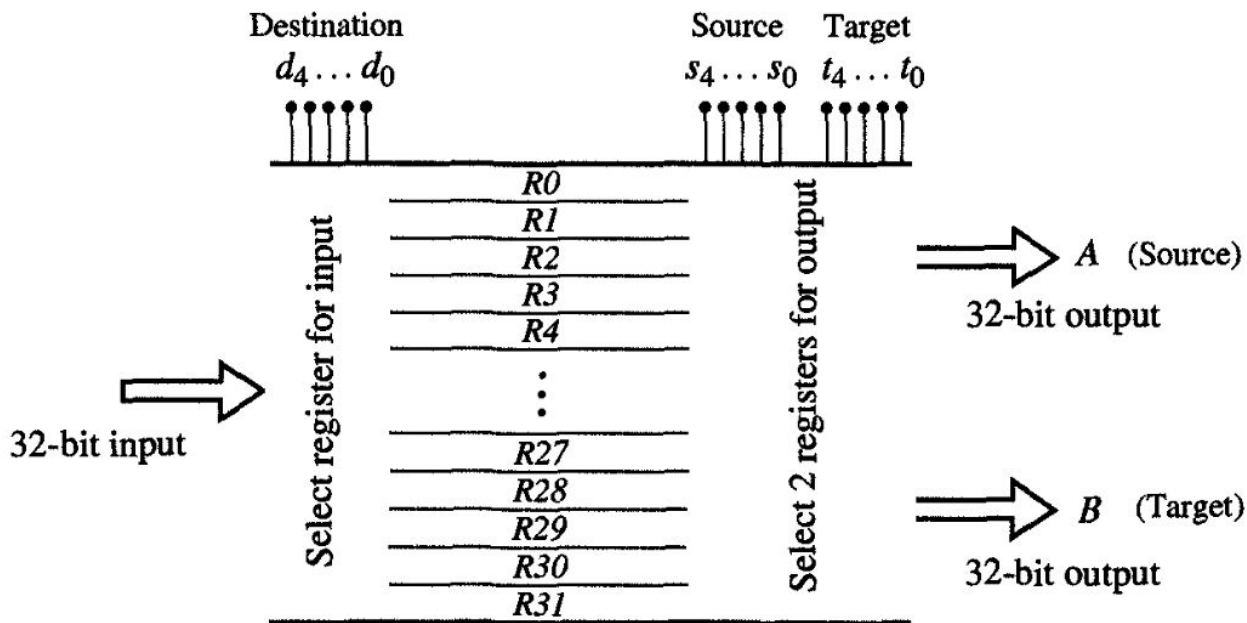


Figure 11.12 Structure of the register file

Arquivo de Registradores (ou Banco de

Detalhando um pouco mais a estrutura...

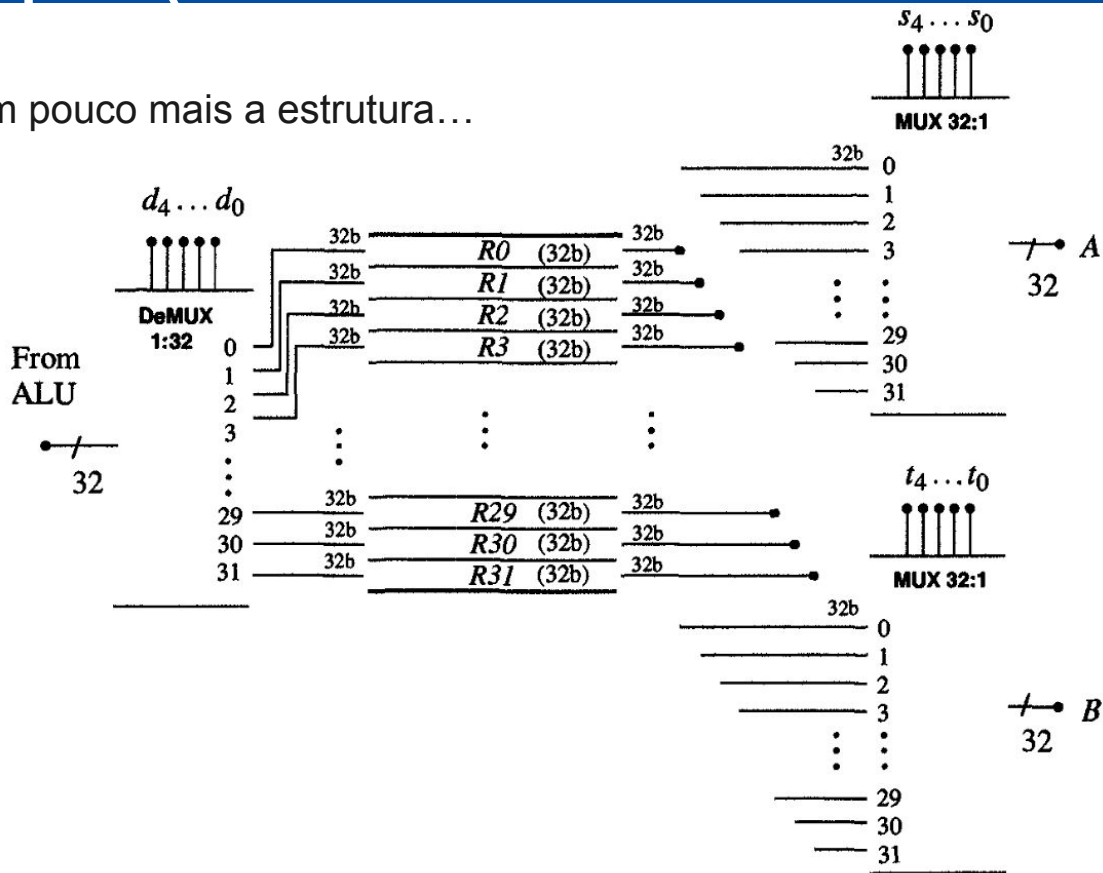


Figure 11.13 A unit-level view of the register file structure

Seção responsável por executar as funções aritméticas requeridas (ADD, SUB, etc.) e todas as operações lógicas (NOT, AND, XOR, etc.)

Operações variam de processador para processador, mas muitas operações são comuns a todos.

A ULA é controlada pela palavra de **seleção de função** f .

No exemplo, assumimos que todos os operandos têm 32 bits.

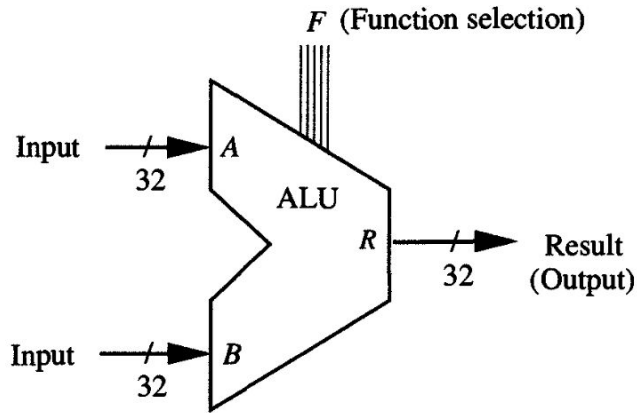


Figure 11.15 Symbol for the arithmetic and logic unit (ALU)

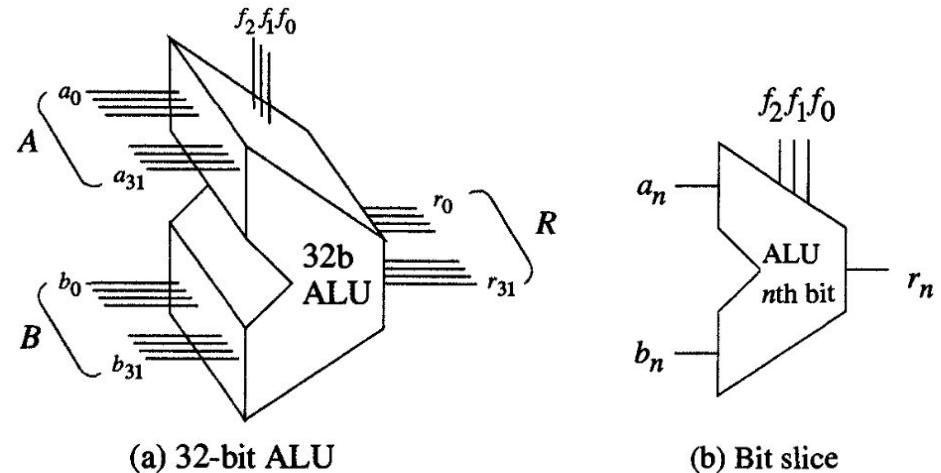


Figure 11.17 Concept of a bit slice of the ALU

Memória Local

Memória utilizada para operações rápidas de *load* e *store*, genericamente chamada de **cache**.

É conectada entre o caminho do fluxo de dados e a memória principal.

Operacionalmente idêntica a uma matriz genérica de memória escrita/leitura. O valor do sinal R/W especifica qual operação será executada.

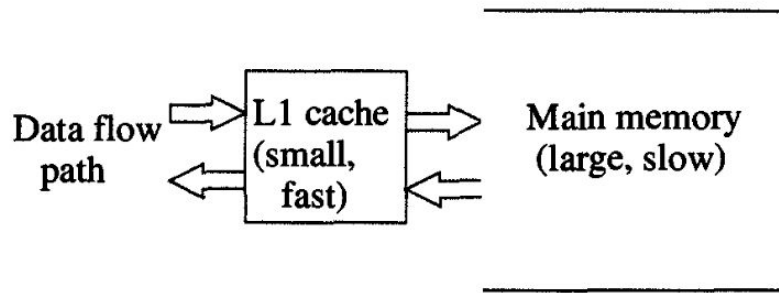


Figure 11.20 Concept of local memory

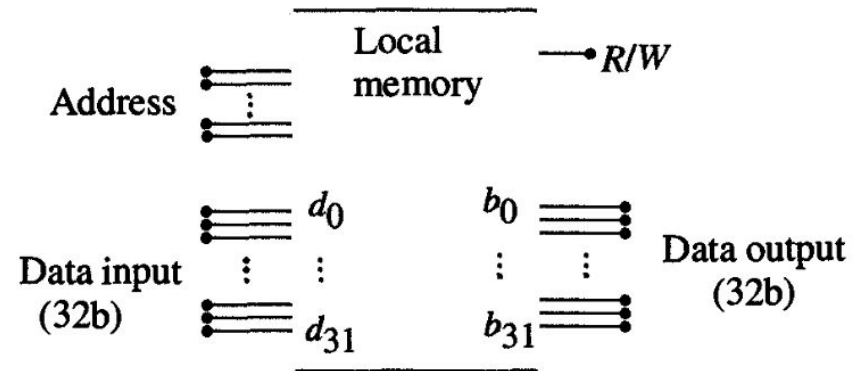


Figure 11.21 Operational model for the local memory

Retomada da última aula

Fizeram a implementação da ULA?

Retomada da última aula

Fizeram a implementação da ULA?

Como?

Bitslice?

Hierárquica com ripple carry?

ISA (Instruction Set Architecture)

Corresponde à especificação do conjunto de instruções em nível de software que um dado processador suporta.

Micro arquitetura

Corresponde às técnicas de projeto que um dado processador utiliza para implementar as instruções da ISA.

NOTEM!!

Computadores com micro arquiteturas diferentes podem compartilhar o mesmo conjunto de instruções

Ex.: Intel Pentium e AMD Athlon: quase a mesma ISA x86, mas com implementações diferentes.

Instruções e Via de Dados (*Datapath*)

Continuaremos na próxima aula

Instruções e Via de Dados

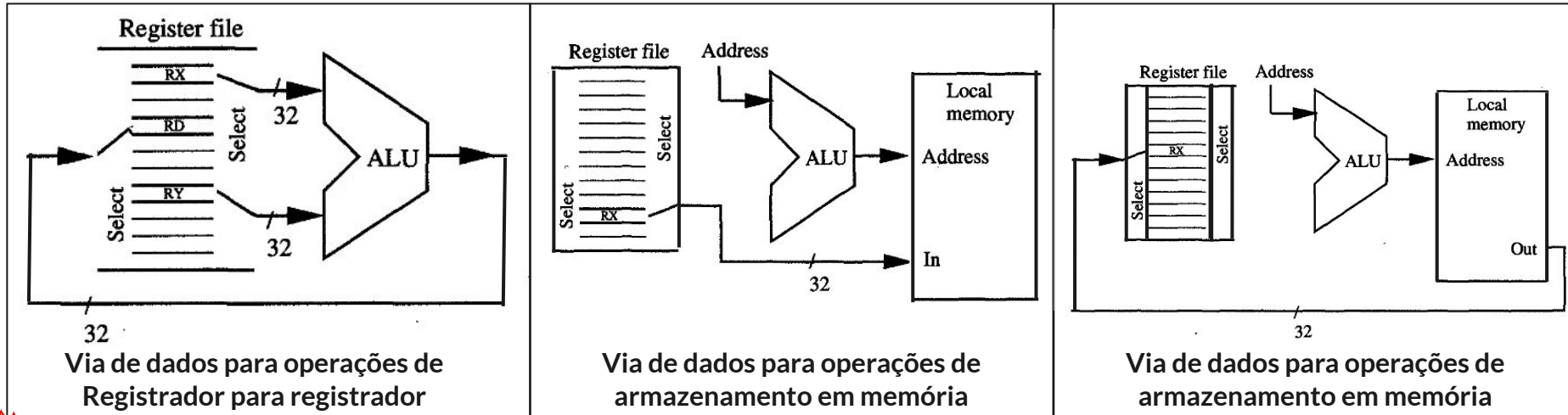
É constituída por três seções principais:

Banco de Registradores, ULA, e unidade de memória

ULA → suas funções determinam o tipo de operações lógicas ou aritméticas

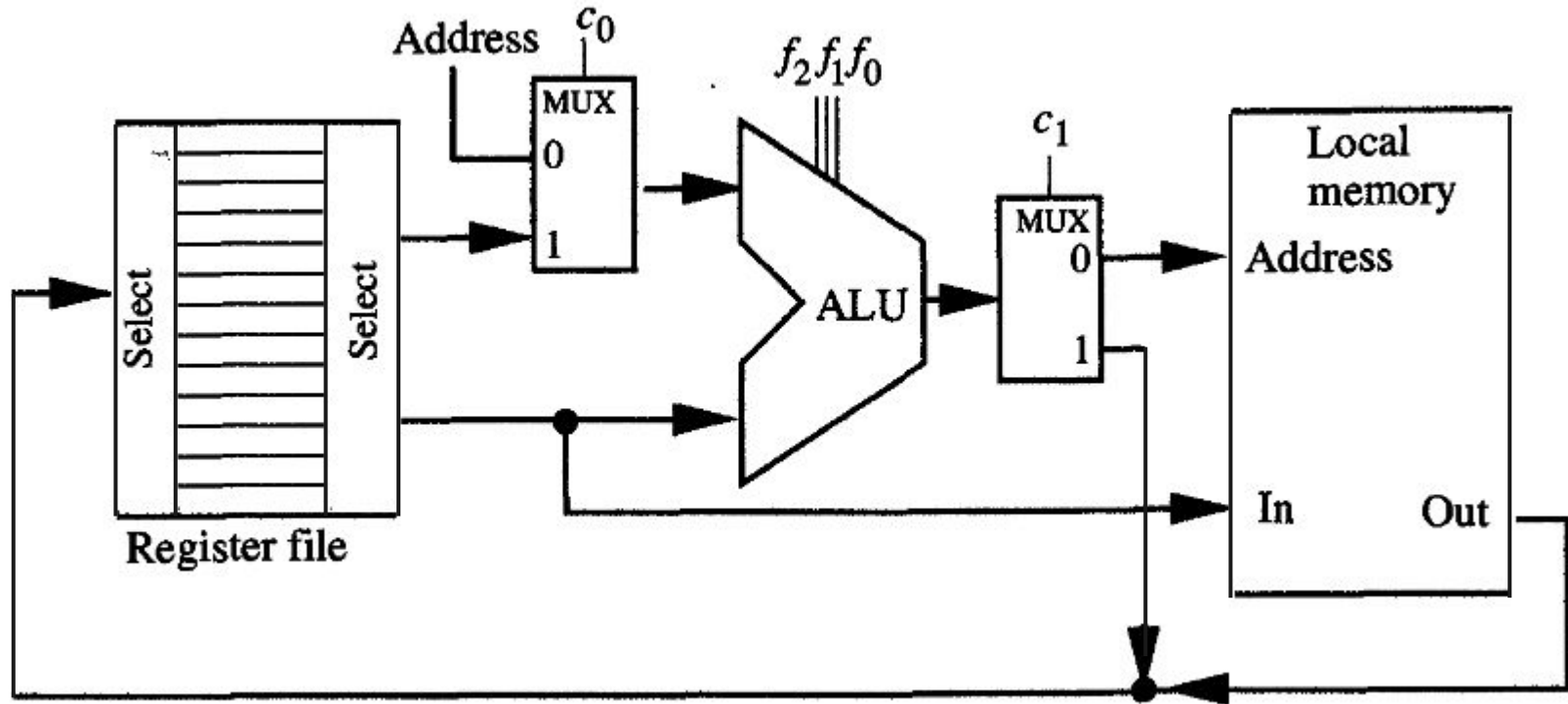
Banco de Registradores → Fornecem posições de armazenamento rápido

Memória cache → permite o acesso ao sistema de memória principal



Instruções e Via de Dados

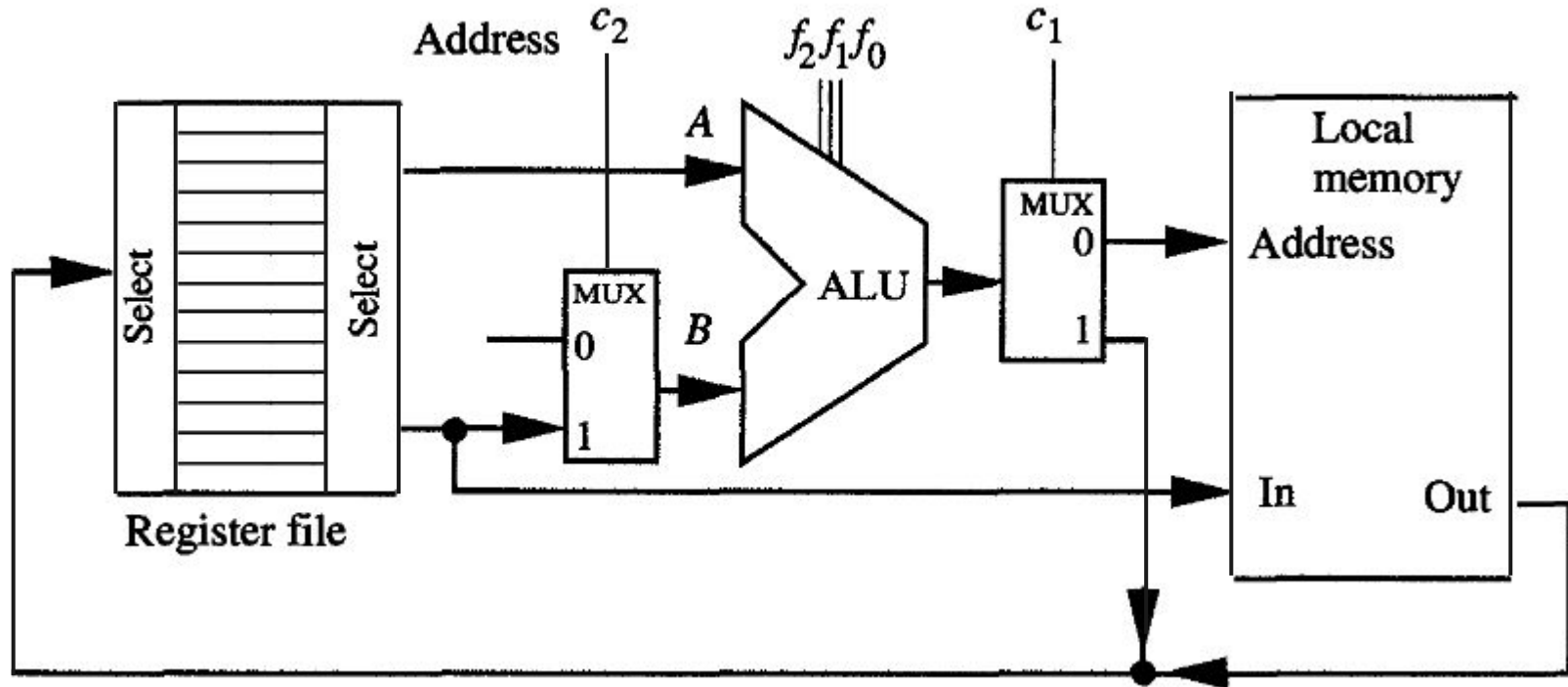
Juntando tudo:



Datapath que permite as operações de registrador para registrador, carregar e armazenar palavra

Instruções e Via de Dados

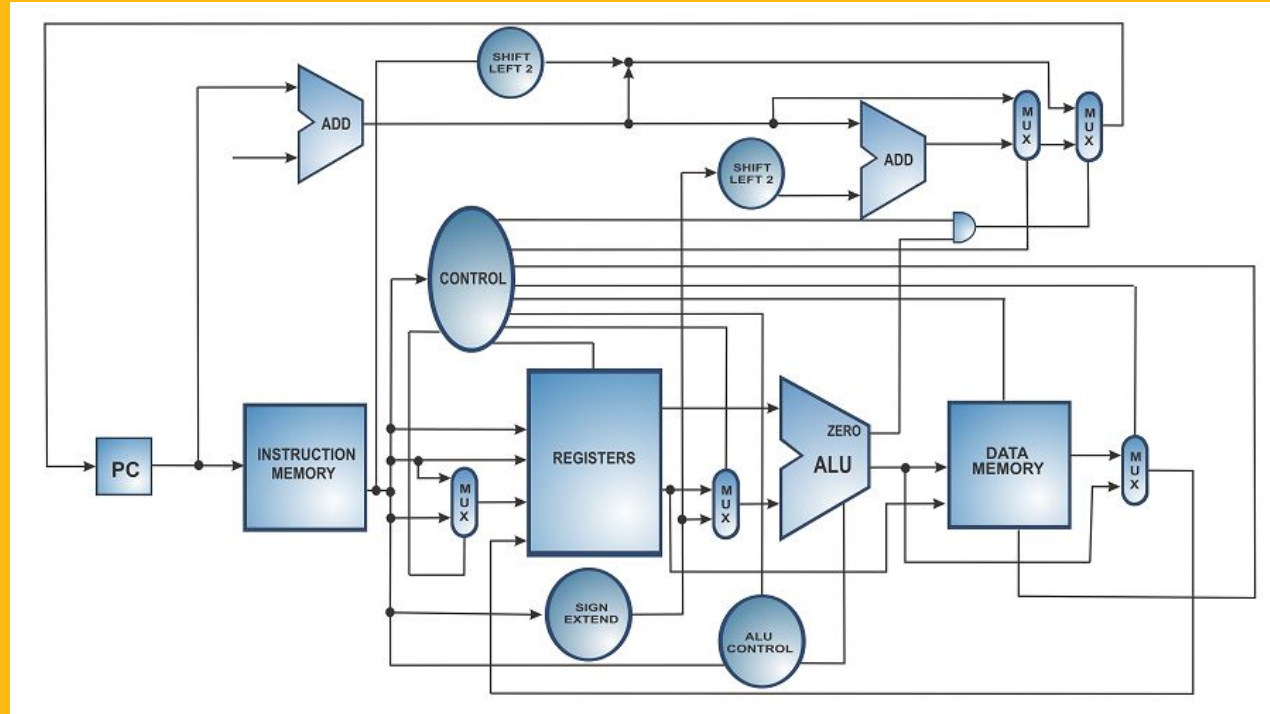
Modificações pontuais podem fazer muita diferença. Vejam a seção 11.4 do Uyemura.



Modificação no datapath para permitir mais operações

Processador MIPS

Desafio: Você consegue apontar o datapath para uma instrução de registrador a registrador?



Arquiteturas RISC e CISC

Continuaremos na próxima aula

As palavras da linguagem de um computador são chamadas de ***instruções*** e seu vocabulário é denominado de ***conjunto de instruções***.

Embora o *conjunto de instruções* seja uma característica associada a arquitetura, em geral, o conjunto de instruções de diferentes arquiteturas são bastante similares.

Isso acontece devido:

Aos projetos possuírem princípios básicos semelhantes

Algumas operações básicas devem ser oferecidas por todos os computadores

RISC vs. CISC

Solução de compromisso:

Desempenho (MIPS) Vs. Consumo de Energia (W) Vs. Custo (\$)

CISC (*Complex Instruction Set Computer*)

Grande quantidade de instruções (x86_64 → 500+)

Múltiplos modos de endereçamento

Instruções com largura variável

Ex.: IBM/360, DEC/Família PDP e VAX

RISC (*Reduced Instruction Set Computer*)

Pequena quantidade de instruções (50 a 150 instruções)

Menor quantidade de modos de endereçamento

Instruções com largura fixa

Ex.: MIPS (MIPS Tech), NIOS2 (Altera), SPARC (SUN Microsystems), RISC-V

Um pouco sobre o MIPS

Criado na década de 80 por John L. Hennessy.

Microprocessador bastante utilizado.

Em 2002, foram fabricados 100 milhões de unidades.

Encontrados em produtos de várias empresas.

ATI, Broadcom, Cisco, NEC, Nintendo, Silicon
Graphics, Sony, Texas Instrument, Toshiba, etc.

Instruções simples, sempre realizando uma única operação por instrução.



Exemplo: Instruções de Soma e Subtração

São elas:

ADD

SUB

Exemplos:

ADD a, b, c # A soma $b + c$ é armazenada em a

SUB a, a, c # A soma $a + c$ é colocada em a

Como é compilado o seguinte código C?

$a = b + c;$

$d = a - e;$

Exemplo: Instruções de Soma e Subtração

São elas:

ADD

SUB

Exemplos:

ADD a, b, c # A soma $b + c$ é armazenada em a

SUB a, a, c # A soma $a + c$ é colocada em a

Como é compilado o seguinte código C?

$a = b + c;$

$d = a - e;$



ADD a, b, c

SUB d, a, e

São os operandos do hardware de um computador

Ao contrário dos programas nas linguagens de alto nível, a quantidade de operandos das instruções aritméticas é restrita.

Os operandos de uma instrução aritmética são os registradores.

No MIPS só temos 32 registradores.

Por que tão poucos registradores?

Consumo de energia, complexidade do hardware, preço

Os registradores são os operandos do MIPS. São 32.
Alguns exemplos:

Nome	Endereço	Descrição
\$zero	\$0	Guarda a o valor constante 0
\$s0-\$s8	\$16-\$23, \$30	Registradores de uso geral
\$t0-\$t9	\$8-\$15, \$24-\$25	Registradores temporários

Tipos de Instrução

As instruções no MIPS são classificadas em 3 tipos, de acordo com o formato:

Tipo R → 3 operandos (registradores)

Tipo I → Um dos operandos vem junto com a instrução

Tipo J → Instruções de desvio incondicional. Sem operandos.

Conprocessador → Não serão abordados.

	3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	0 9	0 8	0 7	0 6	0 5	0 4	0 3	0 2	0 1	0 0	
R	OPCODE						RS					RT					RD					SA					FN						
I	OPCODE						RS					RT					IMM																
J	OPCODE						TARGET																										

O simulador MARS



MARS (MIPS Assembler Runtime Simulator)

É um simulador MIPS, alternativo ao clássico SPIM (disponível via apt).

MARS foi desenvolvido pela universidade do Estado do Missouri, EUA;

MARS foi projetado para ser um simulador de fácil utilização, para alunos de graduação;

Não apresenta recursos de simuladores mais avançados, porém sua interface é bem mais amigável;

Desenvolvido em Java;

Permite desenvolvimento de novos módulos e aplicações.

Conhecendo a interface gráfica

Et:\calebmaeal\Documentos\IFRS\Arquitortura\exemplos\le_dois_numeros_e_soma.asm - MARS 4.2

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Edit Execute

```
le_dois_numeros_e_soma.asm
17 #      System.out.println("O produto dos dois valores e: " + i*!);
18 #
19 #
20 #####
21 #####
22 #####
23 # Desenvolver o programa em assembly
24
25 .globl main
26
27 .text
28
29 main:
30     addi $v0, $zero, 5      # Solicitando ao usuário a digitacao de um valor. Para isso, $v0 deve receber o valor 5.
31     syscall                # Chamada de sistema
32     addi $a0, $zero, $v0    # Dado digitado retorna em $v0. Valor foi copiado para $a0
33
34
35     addi $v0, $zero, 5      # Solicitando ao usuário a digitacao de outro valor
36     syscall                # Chamada de sistema
37     mul $a0, $a0, $v0       # resultado digitado em $v0 é multiplicado pelo primeiro valor que estava em $a0 e armazenado em $a0
38
39     addi $v0, $zero, 1      # Imprimindo resultado da multiplicacao em tela. $v0 recebe valor 1.
40     syscall
41
42     addi $v0, $zero, 10     # Termina o programa
43     syscall
44
```

Line: 39 Column: 34 ☒ Show Line Numbers

Mars Messages Run I/O

5
10
Clear
-- program is finished running --

Registres Coproc 1 Coproc 0

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x0000000a
\$v1	3	0x00000000
\$a0	4	0x0000000a
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$a0	16	0x00000000
\$a1	17	0x00000000
\$a2	18	0x00000000
\$a3	19	0x00000000
\$a4	20	0x00000000
\$a5	21	0x00000000
\$a6	22	0x00000000
\$a7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400028
hi		0x00000000
lo		0x0000000a

Janela de Edição

Conhecendo a interface gráfica

E:\calebemicael\Documentos\IFRS\Arquitortura\exemplos\le_dois_numeros_e_soma.asm - MARS 4.2

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Registers Coproc 1 Coproc 0

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x20020005	addi \$2,\$0,0x00000005	30: addi \$v0, \$zero, 5 # Solicitando ao usuário a digita...
	0x00400004	0x0000000c	syscall	31: syscall # Chamada de sistema
	0x00400008	0x00022020	add \$4,\$0,\$2	32: add \$a0, \$zero, \$v0 # Dado digitado retorna em \$v0. V...
	0x0040000c	0x20020005	addi \$2,\$0,0x00000005	35: addi \$v0, \$zero, 5 # Solicitando ao usuário a digita...
	0x00400010	0x0000000c	syscall	36: syscall
	0x00400014	0x70822002	mul \$4,\$4,\$2	37: mul \$a0, \$a0, \$v0 # resultado digitado em \$v0 é mul...
	0x00400018	0x20020001	addi \$2,\$0,0x00000001	39: addi \$v0, \$zero, 1 # Imprimindo resultado da multipl...
	0x0040001c	0x0000000c	syscall	40: syscall
	0x00400020	0x2002000a	addi \$2,\$0,0x0000000a	42: addi \$v0, \$zero, 10 # Termina o programa
	0x00400024	0x0000000c	syscall	43: syscall

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (.data) [X] Hexadecimal Addresses [X] Hexadecimal Values [] ASCII

Mars Messages Run I/O

10
-- program is finished running --
2

Clear

Registers Coproc 1 Coproc 0

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x00000000
\$v0	2	0x00000002
\$v1	3	0x00000000
\$a0	4	0x00000000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$s8	24	0x00000000
\$s9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffc00
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400008
hi		0x00000000
lo		0x00000000

Programa em Execução

Memória de Dados

Entrada/Saída de Dados

Registradores

Sintaxe do Assembly

Comentários de linha iniciam com #.

Identificadores são sequências de caracteres **alfanuméricos**, _ e . , e não se iniciam com números.

Rótulos são colocados no começo de uma linha e seguidos de : .

Números estão na base decimal por padrão; se precedidos por **0x** são interpretados como hexadecimais.

Strings são delimitadas por aspas (“”).

Utilizadas pelo programador para instruir o assembler como traduzir um programa.

Não produz instruções de máquina.

Principais diretivas:

.asciiz → armazena caracteres de um string na memória e finalizado com o caracter null;

.ascii → armazena caracteres de um string na memória, mas não finaliza-o com null;

.data <end.> → armazena os itens na sequência no segmento de dados. Se <end.> for fornecido, os dados são armazenados a partir do endereço fornecido;

.globl → diretiva declara rótulo como global, podendo ser acessado de outros arquivos;

.text → armazena os itens na sequência no segmento de textos do usuário. Itens devem ser instruções

O MARS provê alguns serviços do sistema operacional através da instrução *syscall*.

Para utilizar um serviço:

Carregar o código do serviço no registrador **\$v0**;

Veja a tabela de serviços no próximo slide

Carregar os argumentos do serviço nos registradores **\$a0-\$a3**;

Chama a instrução ***syscall***.

Chamadas de Sistema

Serviço	Código	Argumentos	Resultado
Print_int	1	\$a0 = inteiro	
Print_float	2	\$f12=float	
Print_double	3	\$f12=double	
Print_string	4	\$a0=string	
Read_int	5		Inteiro (em \$v0)
Read_float	6		Float (em \$v0)
Read_double	7		Double (em \$v0)
Read_string	8	\$a0=buffer, \$a1=tamanho	
Exit	10		
Print_char	11	\$a0=char	

Passo a passo:

Para simular um programa em Assembly:

Utilize a aba Edit para escrever um programa ou Carregar o arquivo (menu File/Open ou ícone Open), que deve ter extensão .s ou .asm.

Ir até a opção Run >> Assemble

Run >> Go , ou ir até o ícone

Dá suporte à execução passo  passo através do botão

Entrada e saída do programa são fornecidas através da janela Run/IO. 

Primeiros testes

```
main: # Inicio do codigo
      # Carrega o codigo 4 no registrador $v0
      li    $t0, 4           #load immediate t0 <- 4
      li    $t1, 7           #load immediate t1 <- 7
      # Somo os valores e guardo em t0
      add    $t0, $t0, $t1    #addition t0 <- t0 + t1

      # vou imprimir o resultado na tela. O resultado tem que estar em $a0
      # para isso, o resultado tem que estar em $a0
      li    $a0, 0           # reseto o valor de $a0
      add    $a0, $t0, $zero  # uso essa artimanha para atribuir

      #preparo uma chamada de sistema para imprimir
      # Escreve o código 1 em $v0. Código 1 significa imprimir inteiro
      li    $v0, 1           # load immediate
      syscall                # Realiza a chamada e escreve um inteiro
```

Exercício 1: modifique para que os dados sejam lidos

main: # Inicio do codigo

Carrega o codigo 4 no registrador \$v0

li \$t0, 4 #load immediate t0 <- 4

li \$t1, 7 #load immediate t1 <- 7

Somo os valores e guardo em t0

add \$t0, \$t0, \$t1 #addition t0 <- t0 + t1

vou imprimir o resultado na tela. O resultado tem que estar em \$a0

para isso, o resultado tem que estar em \$a0

li \$a0, 0 # reseto o valor de \$a0

add \$a0, \$t0, \$zero # uso essa artimanha para atribuir

#preparo uma chamada de sistema para imprimir

Escreve o código 1 em \$v0. Código 1 significa imprimir inteiro

li \$v0, 1 # load immediate

syscall # Termina o programa.

DICA: Revise os slides
"Chamadas de sistema"

Exemplo 1: Hello World!

```
# Hello World in MIPS Assembly
        .data    # carrega a string no primeiro endereco
                # disponivel do proximo segmento de dados
hello_msg: .ascii "Hello World!\n"
        .text

main:   # Inicio do codigo
        # Carrega o endereco da mensagem em $a0
        la $a0, hello_msg    #load address
        # Carrega o codigo 4 no registrador $v0
        li $v0, 4            #load immediate
        syscall              #imprime uma string
```

Exemplo 2: Lê 2 Valores e Multiplica

```
.globl main
.text
main:
    addi $v0, $zero, 5    # $v0 deve receber o valor 5.
    syscall               # Chamada de sistema. Solicita a digitacao de um valor.
    add $a0, $zero, $v0   # Dado digitado retorna em $v0. Copia valor de $v0 para $a0
    addi $v0, $zero, 5    # $v0 deve receber o valor 5.
    syscall               # Chamada de sistema. Solicita digitacao de um outro valor
    mul $a0, $a0, $v0     # resultado digitado em $v0 é multiplicado pelo primeiro
                        # valor que estava em $a0 e armazenado em $a0.

    addi $v0, $zero, 1    # $v0 recebe valor 1.
    syscall               # Imprimindo resultado da multiplicacao em tela.
    addi $v0, $zero, 10   # $v0 recebe valor 1.
    syscall               # Termina o programa
```

Exercícios Sugeridos

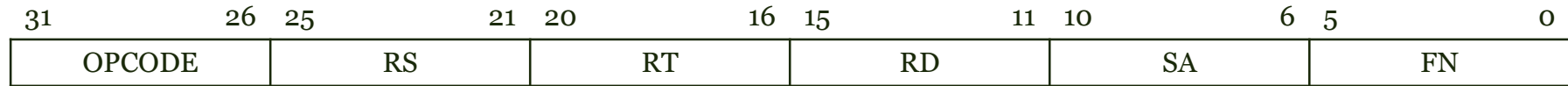
- 1) Escreva um programa que leia dois números inteiros e apresente na tela o resultado da soma, e da subtração entre eles. Assegure-se de imprimi-los em linhas diferentes.
- 2) Escreva um programa que leia os coeficientes de uma função quadrática, e calcule o valor de Delta da fórmula de Bhaskara.

Tipos de Instrução

- As instruções no MIPS são classificadas em 3 tipos, de acordo com o formato:
 - Tipo R → 3 operandos (registradores)
 - Tipo I → Um dos operandos vem junto com a instrução
 - Tipo J → Instruções de desvio incondicional. Sem operandos.

	3 1	3 0	2 9	2 8	2 7	2 6	2 5	2 4	2 3	2 2	2 1	2 0	1 9	1 8	1 7	1 6	1 5	1 4	1 3	1 2	1 1	1 0	0 9	0 8	0 7	0 6	0 5	0 4	0 3	0 2	0 1	0 0
R	OPCODE						RS					RT					RD					SA					FN					
I	OPCODE						RS					RT					IMM															
J	OPCODE						TARGET																									

Tipo R



Não há atualização do PC (além da regular)

RS e RT são operandos fonte

Possui Opcode = 000000

A operação da ALU é determinada pelo campo FN

Não há acesso à memória principal.

O resultado da operação é escrito em RD

Tipo R



31	26	25	21	20	16	15	11	10	6	5	0
OPCODE	RS	RT	RD	SA	FN						

Exemplos:

ADD \$t1 \$s1 \$s2

31	26	25	21	20	16	15	11	10	6	5	0
0	17	18	9	0	32						

31	26	25	21	20	16	15	11	10	6	5	0
000000	10001	10010	01001	00000	100000						

Tipo R



31	26	25	21	20	16	15	11	10	6	5	0
OPCODE	RS	RT	RD	SA	FN						

Exemplos:

ADD \$t1 \$s1 \$s2

31	26	25	21	20	16	15	11	10	6	5	0
0	17	18	9	0	32						

31	26	25	21	20	16	15	11	10	6	5	0
000000	10001	10010	01001	00000	100000						

SUB \$t4 \$s3 \$s7

31	26	25	21	20	16	15	11	10	6	5	0
0	19	20	12	0	34						

31	26	25	21	20	16	15	11	10	6	5	0
000000	10101	10100	01100	00000	100010						

Exemplo



$$f = (g + h) - (i + j);$$

Exemplo

$f = (g + h) - (i + j);$

add to,g,h

add t1,i,j

sub f,to,t1

o compilador cria to e t1 .

Exemplo

$f = (g + h) - (i + j);$

add to,g,h

add t1,i,j

sub f,to,t1

o compilador cria to e t1 .

Ajustando...

add \$to,\$s1,\$s2

add \$t1,\$s3,\$s4

sub \$so,\$to,\$t1

Exemplo 3

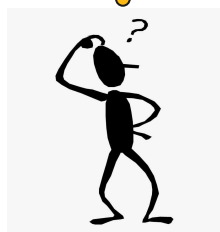
Expressões

```
1  .globl main
2  # Esse código calcula o resultado da operacao: (g+h) - (i+j)
3  # vou usar g -> $s1, h -> $s2, i-> $s3, j -> $s4 e vou guardar
4  # o resultado da expressao f em $s0. NMa expressao, vou precisar
5  # de dois temporarios: $t0 para (g+h), e $t1 para (i+j).
6  main:
7  # le o valor g
8  li $v0, 5 # configura le inteiro
9  syscall   # executa le inteiro
10 add $s1, $zero, $v0
11 # le o valor h
12 li $v0, 5 # configura le inteiro
13 syscall   # executa le inteiro
14 add $s2, $zero, $v0
15 # le o valor i
16 li $v0, 5 # configura le inteiro
17 syscall   # executa le inteiro
18 add $s3, $zero, $v0
19 # le o valor j
20 li $v0, 5 # configura le inteiro
21 syscall   # executa le inteiro
22 add $s4, $zero, $v0
23 # realiza a expressao usando os temporarios $t0 e $t1
24 add $t0,$s1,$s2 # t0 = (g + h)
25 add $t1,$s3,$s4 # t1 = (i + j)
26 sub $s0,$t0,$t1 # f = t0 - t1
27 # vou transferir o resultado de $s0 pra $a0 pra que seja impresso.
28 #add $a0, $v0, $zero
29 # ou, eu poderia usar a pseudo-instrucao a seguir
30 move $a0, $s0 # carrega o resultado em $a0, para impressao
31 li $v0 1      # configuro para imprimir um inteiro
32 syscall      # mando executar
33 li $v0 10     # configuro para encerrar a execucao
34 syscall      # mando executar
```

Exemplo 3

Expressões

E se o número
de registradores
não é
suficiente?



```
1  .globl main
2  # Esse código calcula o resultado da operacao: (g+h) - (i+j)
3  # vou usar g -> $s1, h -> $s2, i-> $s3, j -> $s4 e vou guardar
4  # o resultado da expressao f em $s0. NMa expressao, vou precisar
5  # de dois temporarios: $t0 para (g+h), e $t1 para (i+j).
6  main:
7  # le o valor g
8  li $v0, 5 # configura le inteiro
9  syscall   # executa le inteiro
10 add $s1, $zero, $v0
11 # le o valor h
12 li $v0, 5 # configura le inteiro
13 syscall   # executa le inteiro
14 add $s2, $zero, $v0
15 # le o valor i
16 li $v0, 5 # configura le inteiro
17 syscall   # executa le inteiro
18 add $s3, $zero, $v0
19 # le o valor j
20 li $v0, 5 # configura le inteiro
21 syscall   # executa le inteiro
22 add $s4, $zero, $v0
23 # realiza a expressao usando os temporarios $t0 e $t1
24 add $t0,$s1,$s2 # t0 = (g + h)
25 add $t1,$s3,$s4 # t1 = (i + j)
26 sub $s0,$t0,$t1 # f = t0 - t1
27 # vou transferir o resultado de $s0 pra $a0 pra que seja impresso.
28 #add $a0, $v0, $zero
29 # ou, eu poderia usar a pseudo-instrucao a seguir
30 move $a0, $s0 # carrega o resultado em $a0, para impressao
31 li $v0 1      # configuro para imprimir um inteiro
32 syscall       # mando executar
33 li $v0 10     # configuro para encerrar a execucao
34 syscall       # mando executar
```


Exemplo 3

Expressões

E se o número
de registradores
não é
suficiente?

Guarda na memória principal!

```
1  .globl main
2  # Esse código calcula o resultado da operacao: (g+h) - (i+j)
3  # vou usar g -> $s1, h -> $s2, i -> $s3, j -> $s4 e vou guardar
4  # o resultado da expressao f em $s0. NMa expressao, vou precisar
5  # de dois temporarios: $t0 para (g+h), e $t1 para (i+j)
6  main:
7  # le o valor g
8  li $v0, 5 # configura le inteiro
9  syscall # executa le inteiro
10 add $s1, $zero, $v0
11 # le o valor h
12 li $v0, 5 # configura le inteiro
13 syscall # executa le inteiro
14 add $s2, $zero, $v0
15 # le o valor i
16 li $v0, 5 # configura le inteiro
17 syscall # executa le inteiro
18 add $s3, $zero, $v0
19 # le o valor j
20 li $v0, 5 # configura le inteiro
21 syscall # executa le inteiro
22 add $s4, $zero, $v0
23 # calcula a expressao usando os temporarios $t0 e $t1
24 add $t0, $s1, $s2 # t0 = (g + h)
25 add $t1, $s3, $s4 # t1 = (i + j)
26 sub $s0, $t0, $t1 # f = t0 - t1
27 # vou transferir o resultado de $s0 pra $a0 pra que seja impresso.
28 #add $a0, $v0, $zero
29 # ou, eu poderia usar a pseudo-instrucao a seguir
30 move $a0, $s0 # carrega o resultado em $a0, para impressao
31 li $v0 1 # configuro para imprimir um inteiro
32 syscall # mando executar
33 li $v0 10 # configuro para encerrar a execucao
34 syscall # mando executar
```

Antes... Entendendo a memória



Qual a menor unidade de informação no computador?
Qual a menor unidade endereçável?

Antes... Entendendo a memória

Qual a menor unidade de informação no computador?
bit

Qual a menor unidade endereçável? Byte (8 bits)

0	00000000
1	00000101
2	00000000
3	00000000
4	00000000
5	10101010
...	
N	11101010

Antes... Entendendo a memória

Qual a menor unidade de informação no computador?
bit

Qual a menor unidade endereçável? **Byte (8 bits)**

Acessar dado do tipo **Char** na
posição 2.

r

00000000

Representado por 1
Byte

0	00000000
1	00000101
2	00000000
3	00000000
4	00000000
5	10101010
...	
N	11101010

Antes... Entendendo a memória

Qual a menor unidade de informação no computador?
bit

Qual a menor unidade endereçável? **Byte (8 bits)**

Acessar dado do tipo **Int** na posição

2.

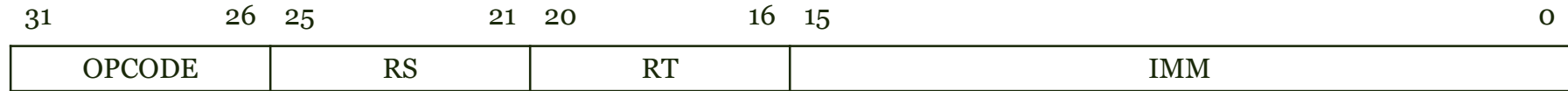
Representado por 4

Bytes

0	00000000
1	00000101
2	00000000
3	00000000
4	00000000
5	10101010
...	
N	11101010

00000000 00000000 00000000 10101010

Tipo I



Não há atualização do PC (além da regular)

RS é operando fonte

Todos os Opcode, exceto: 000000, 00001x, 0100xx

A operação da ALU é determinada pelo Opcode

O resultado da operação é escrito em RT

Tipo I

Exemplos

SW \$to, 1200(\$t1)

31 26 25 21 20 16 15 0

43	9	8	1200
----	---	---	------

31 26 25 21 20 16 15 0

101011	01001	10000	0000010010110000
--------	-------	-------	------------------

LW \$to,1200(\$t1)

31 26 25 21 20 16 15 0

35	9	8	1200
----	---	---	------

31 26 25 21 20 16 15 0

100011	01001	10000	0000010010110000
--------	-------	-------	------------------


$$A[300] = h + A[300]$$

Assuma que \$t1 guarda o endereço de A, vetor de inteiros.


Traduzindo, temos.

```
lw $t0,1200($t1) # O reg. temporário $t0 recebe A[300]
add $t0,$s2,$t0  # $t0 recebe a soma de $s2 (h) e $t0 (A[300])
sw $t0,1200($t1) # Guarda o resultado ($t0) em A[300]
```


Exemplos




```
1 #include <stdio.h>
2
3 int main(){
4     int c[15] = {3, 0, 1, 2, -6, -2, 4, 10, 3, 7, 8, -9, -15, -20, -87};
5     int h = 30;
6     c[10] = h + c[10];
7     printf("%d", c[10]);
8     return 0;
9 }
```



$$C[10] = h + C[10]$$

Exemplos



```
1 #include <stdio.h>
2
3 int main(){
4     int c[15] = {3, 0, 1, 2, -6, -2, 4, 10, 3, 7, 8, -9, -15, -20, -87};
5     int h = 30;
6     c[10] = h + c[10];
7     printf("%d", c[10]);
8     return 0;
9 }
```

$$C[10] = h + C[10]$$

lw \$t0,40(\$t1) # O reg. temporário \$t0 recebe C[10]
add \$t0,\$s2,\$t0 # \$t0 recebe a soma de \$s2 (**h**) e \$t0 (**C[10]**)
sw \$t0,40(\$t1) # Guarda h + C[10] (\$t0) de volta em C[10]

Exemplos

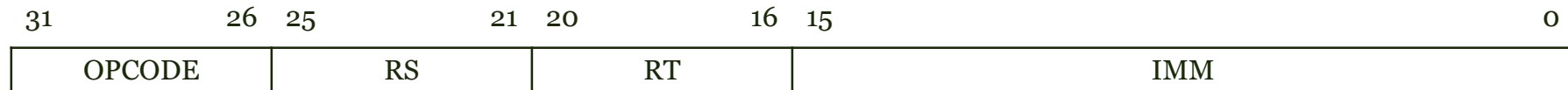
O código traduzido para
assembly MIPS



Com algumas diferenças
em relação ao slide
anterior. Consegue notar?

```
1 .globl main
2 .data
3 C: .word 3, 0, 1, 2, -6, -2, 4, 10, 3, 7, 8, -9, -15, -20, -87
4 # A diretiva ".data" indica que as instruções a seguir contêm dados.
5 # A diretiva ".word" armazena n quantidades de 32 bits em palavras de memórias sucessivas.
6 #vou guardar em $s1 o endereço de C, e vou guardar em $s2 o valor de h
7 .text
8 main:
9 # le o valor g
10 li $s2, 30      # carrega o valor 30 no registrador que representa h
11 la $s1, C       # load address: guarda em $s1 guarda o endereço de C
12 # guardo no temporario $t1 o indice que quero acessar no vetor C
13 li $t1, 10
14 # e multiplico por 4, ja que cada inteiro ocupa 4 enderecos de memoria.
15 # para isso vou usar o operador de deslocamento a esquerda, porque sim. :)
16 sll $t1, $t1, 2 # deslocar 2 a esquerda equivale a multiplicar por 4.
17 # o dado que quero buscar em C está distante de $t1 endereços a partir do endereço C ($s1)
18 # logo, preciso somar esse valor do endereço base ao valor de $t1
19 add $t1, $t1, $s1
20 #enfim, carrego o dado apontado pelo endereço $t1 no registrador $t0
21 lw $t0, 0($t1) # esse 0 indica o deslocamento de endereços a partir de $t1
22 # ou seja, tudo isso poderia ter sido substituído por lw $t0, 40, ($s1).
23 # mas resolvi mostrar direto uma tradução para percorrer um vetor com uma variavel i, por ex.
24 #agora vou fazer a conta, acumulando em $t0 o valor de h ($s2)
25 add $t0, $t0, $s2
26 # posso entao devolver para a memória.
27 sw $t0, 0($t1)
28 # vou transferir pra $a0 pra que seja impresso.
29 move $a0, $t0
30 # agora imprime.
31 li $v0, 1 # configura escreve inteiro
32 syscall  # executa escreve inteiro
33 li $v0, 10 # configura encerra programa
34 syscall  # executa encerra programa
```

Instruções de Desvio Condicional



Desvia o fluxo de execução de acordo com a condição dada.

Pode atualizar o PC

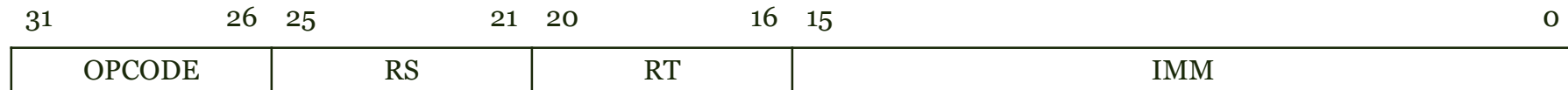
$PC \leftarrow PC + 4 + (IMM \ll 2)$. # IMM é considerado com sinal.

RS e RT são dois operandos

Os operandos são subtraídos na ALU para comparação

Não realiza escrita em registradores ou memória

Instrução BEQ



Desvia se os dois registradores RS e RT são iguais.

Opcode: 4 (000100)

Sintaxe: beq \$s, \$t, offset

Operação:

if (\$s == \$t) avança_pc (4 + offset << 2); else avança_pc (4);

Exercício 1



Escreva um programa em Assembly MIPS que leia dois números e informe o maior deles.

Exercício 2

Escreva um programa em Assembly MIPS que apresente na tela os números de 1 a N, onde N deve ser informado pelo usuário.

Chamada de Funções

```
1 #include <stdio.h>
2
3 int funcao_f( int g, int h, int i, int j){
4     return (g+h) - (i+j);
5 }
6
7 int main(){
8     int a, b, c, d;
9     scanf("%i%i%i%i", &a, &b, &c, &d);
10    printf("%d", funcao_f(a,b,c,d));
11    return 0;
12 }
```


Chamada de Funções

Transcreva o código a seguir para acompanhar a explicação. Guie-se pelos números das linhas.

```
1  .globl main
2  # agora esse código vai implementar f como uma função.
3  # vai receber g, h, i e j como argumentos, e retornar
4  # o resultado da operação: (g+h) - (i+j)
5  # vou usar g -> $a0, h -> $a1, i -> $a2, j -> $a3
6  # por que esses são os registradores convencionados p/
7  # serem argumentos. Só por convenção mesmo.
8  # vou colocar o resultado do cálculo em $s0.
9  # na função, vou precisar de dois temporários:
10 #     $t0 para (g+h), e
11 #     $t1 para (i+j).
12 # Por convenção, vou colocar o valor de retorno em $v0
13
```

Chamada de Funções

Transcreva o código a seguir para acompanhar a explicação. Guie-se pelos números das linhas.

```
14 main:
15
16 # le o valor g
17 li $v0, 5 # configura le inteiro
18 syscall # executa le inteiro
19 add $a0, $zero, $v0
20
21 # le o valor h
22 li $v0, 5 # configura le inteiro
23 syscall # executa le inteiro
24 add $a1, $zero, $v0
25
26 # le o valor i
27 li $v0, 5 # configura le inteiro
28 syscall # executa le inteiro
29 add $a2, $zero, $v0
30
31 # le o valor j
32 li $v0, 5 # configura le inteiro
33 syscall # executa le inteiro
34 add $a3, $zero, $v0
```

Chamada de Funções

Transcreva o código a seguir para acompanhar a explicação. Guie-se pelos números das linhas.

```
35
36 # a instrucao a seguir desvia para a funcao_f,
37 # e salva o endereco da proxima instrucao nesse
38 # bloco. Eh responsabilidade da funcao usar os
39 # registradores e devolver como estavam. Note que
40 # alguns registradores ($v0, $v1), por convencao,
41 # precisarao ser modificados com o valor de retorno
42 # logo, os valores destes nao serao mantidos.
43 # Esse cuidado e da funcao que chama (caller), nao da
44 # que e chamada (callee)
45
46 # <descomente as 3 linhas a seguir para testar>
47 #li $t0, 8
48 #li $t1, 9
49 #li $s0, 10
50 jal funcao_f
51
```

Chamada de Funções

Transcreva o código a seguir para acompanhar a explicação. Guie-se pelos números das linhas.

```
52  # o valor de retorno da funcao vai estar em $v0
53  # vou transferir pra $a0 pra que seja impresso.
54  add $a0, $v0, $zero
55
56  # agora imprime.
57  li $v0, 1 # configura escreve inteiro
58  syscall   # executa escreve inteiro
59
60  li $v0, 10 # configura encerra programa
61  syscall   # executa encerra programa
62
63
```

Chamada de Funções

Transcreva o código a seguir para acompanhar a explicação. Guie-se pelos números das linhas.

```
64 funcao_f:
65 # no código que implementa a funcao, destacado abaixo,
66 # eu vou precisar alterar os registradores $t0, $t1 e $s0
67 # por isso eu preciso salvar seus dados na memoria RAM
68 # antes de fazer qualquer modificacao nos seus dados.
69 # vou usar pra isso a regioao de memoria indicada por $sp
70 # note que devo salvar, independente de saber se esses
71 # registradores sao usados ou nao pelo caller (funcao que
72 # chama a execucao dessa). Nao tem como saber, entao nos
73 # sempre salvamos.
74
75 # como sao 3 registradores, vou abrir espaco pra 3.
76 addi $sp, $sp, -12
77 # agora eu salvo os registradores que vou precisar usar.
78 sw $t0, 0($sp)
79 sw $t1, 4($sp)
80 sw $s0, 8($sp)
81 # agora posso calcular.
82
```

Chamada de Funções

Transcreva o código a seguir para acompanhar a explicação. Guie-se pelos números das linhas.

```
83 #####
84 ##### 0 código abaixo é o mesmo do exemplo anterior####
85 ##### exceto pelos registradores mencionados antes.####
86 #####
87 # t0 = (g+h)
88 add $t0, $a0, $a1
89 # t1 = (i+j)
90 add $t1, $a2, $a3
91 # a0 = t0 - t1
92 sub $s0, $t0, $t1
93 #####
94
95 # $s0 guarda o resultado mas, por convenção, o retorno
96 # é feito em $v0. logo:
97 add $v0, $zero, $s0
98 # Ou, alternativamente, posso usar a pseudofunção move
99 # move $v0, $s0
```


Chamada de Funções

Transcreva o código a seguir para acompanhar a explicação. Guie-se pelos números das linhas.

```
100
101 # tudo ok, mas antes de voltar pro código que chamou a
102 # função (com a instrução jr $ra), temos que restaurar
103 # os valores salvos dos registradores que usamos aqui.
104 lw $t0, 0($sp)
105 lw $t1, 4($sp)
106 lw $s0, 8($sp)
107 # já que não precisamos mais do espaço de memória que
108 # reservamos pra salvar registradores, podemos liberar
109 addi $sp, $sp, 12
110 # agora sim, registradores restaurados, memória liberada
111 # voltamos a função que chamou.
112 jr $ra
```

Referências



PATTERSON, D. A., HENNESSY, J. L. Computer Organization and Design: The Hardware/Software Interface, 3rd Edition, 2005

Hora-Trabalho de Hoje

Leia o capítulo Apêndice A do livro PATTERSON & HENNESSY. Computer Organization and Design: The Hardware/Software Interface, para referência das instruções MIPS. Principalmente a seção A.10 do Apêndice.

Trabalho 2

Escolha um programa desenvolvido por você em C, Python ou outra linguagem (em ordem de preferência), e o traduza para o código assembly MIPS. Abuse dos comentários, inserindo-os em todas as linhas (como nos exemplos deste slide).

- 1) O que o programa faz? Qual problema ele resolve? Apresente o enunciado no início.
- 2) Após a tradução para o assembly, você enxerga pontos de otimização da descrição original em seu código? Discuta nas conclusões.

Requisitos:

A temática do programa é livre. Use as provas da disciplina de Programação Imperativa como referência de complexidade. Questões do Beecrowd também são bons guias.

Critérios:

O programa deve estar correto. As especificações contidas neste slide têm que ser cumpridas. Programas mais complexos terão maior nota. Considero a seguinte ordem de complexidade (e se misturar, melhor): *sequenciais* < *desvio condicional* < *repetição* < *c/ vetores* < *funções* < *recursivo*

Entrega (até 13/01/2025)

Slides em PDF contendo prints do código comentado e explicações, combinados com prints do X-Ray do MARS (Tools -> XRay); e a listagem do código em assembly. Tudo em um zip (PDF + .asm)

Dúvidas?

Na próxima aula...

Continuaremos a detalhar o processador MIPS;

Não falte! 😊

Obrigado pela atenção