Arquitetura e Organização de Computadores

Caros alunos, as videoaulas desta disciplina encontram-se no AVA (Ambiente Virtual de Aprendizagem).

UniFil

Unidade 4

Linguagem de Montagem.

UniFil

Introdução da Unidade

Olá, amigo(a) discente! Seja bem-vindo(a)!

Um sistema computacional precisa entender as instruções dadas para poder executá-las. Esse processo ocorre por meio de uma linguagem de montagem, que é o conteúdo que vamos estudar.

Linguagens de montagens se localizam no nível 4 dos níveis dos sistemas de computação modernos. É nesse nível que começam as atuações dos desenvolvedores de sistemas para computação.

Atualmente o conceito de programar sistemas, não engloba apenas softwares para o uso comercial, mas também desenvolvimento para sistemas de infraestrutura, tais como, os sistemas embarcados. Por esse fato, esse estudo que faremos é de suma importância.

Inicialmente na aula 1, veremos os conceitos de linguagem de máquina e linguagem de montagem. Mostraremos os elementos que compõem esse tipo de linguagem.

Na aula 2, vamos abordar a linguagem de montagem adotada para a programação na arquitetura de processadores MIPS. Veremos os seus aspectos mais importantes. Por fim, apresentaremos um importante simulador em que poderemos programá-lo com notação para o processador MIPS.

Objetivos

- Conhecer os conceitos básicos da linguagem de montagem;
- Estudar a linguagem de montagem do MIPS e aprender sobre um simulador para a mesma linguagem.

Conteúdo Programático

Aula 1: Introdução à Linguagem de montagem.

Aula 2: Linguagem de montagem em processador MIPS.

Você poderá também assistir às videoaulas em seu celular! Basta apontar a câmera para os **QRCodes** distribuídos neste conteúdo.

Pode ser necessário instalar um aplicativo de leitura QRcode no celular e efetuar login na sua conta Gmail.

Aula 1 Introdução à Linguagem de montagem

Na unidade 01 estudamos que os sistemas de computação que podem ser divididos em níveis. Você, querido(a) aluno(a) que está estudando a área de computação e vai ser um programador, com certeza vai trabalhar no nível 04 dos sistemas de computação.

Esse nível é onde encontramos o que chamamos de linguagens simbólicas, as quais, podem ser traduzidas para uma linguagem de máquina, ou seja, você programa em linguagem simbólica e seus códigos são transformados em bytes para o entendimento da máquina.

Por isso, é de essencial importância programadores saberem sobre esse nível computacional. Mas atenção! Não estamos no referindo às linguagens de alto nível, pois essas estão no nível 5.

Inicialmente para começarmos nosso estudo, é interessante distinguir dois tipos de linguagem: linguagem de montagem e linguagem de máquina (CÔRREA, 2016).

- **Linguagem de Montagem**: linguagem de montagem utiliza-se de nomes simbólicos para atribuir nomes a posições específicas da memória principal e das instruções. É uma linguagem na qual cada declaração produz exatamente uma instrução de máquina pura. Dessa forma, existe uma relação um para um entre as instruções de máquina e as declarações no programa de montagem. Conhecida popularmente como *assembly*.
- **Linguagem de máquina**: são instruções que podem ser executadas diretamente pelo processador. As instruções, chamadas de *opcodes (*código de operações), é uma referência à instrução que um determinado processador possui para conseguir realizar determinadas tarefas.

Videoaula 1

Utilize o QRcode para assistir!

Nesse momento assista ao vídeo que aborda sobre os tipos de linguagens de programação.

Linguagem de máquina

Os computadores são fabricados com um conjunto de instruções já determinadas, o conjunto dessas instruções são conhecidas como ISA (*Instruction Set Architecture*). Para se programar

em código de máquina, deve-se obter os códigos de instruções do processador utilizado, contendo *opcodes*, operandos e formatos de cada instrução.

Geralmente, essas instruções são compostas por uma sequência de bytes que equivalem às instruções do processador e são representadas por valores em hexadecimal. Como que programar em linguagem de máquina não é algo trivial, foi criado ferramentas de programação chamadas de **montadores**, pois esses contêm instruções simbólicas que expressam as mesmas instruções do processador e são fáceis de programar. Veja um exemplo de linguagem de máquina representada em hexadecimal:

B4 03 CD 10 B0 01 B3 0A B9 0B 00 BD 13 01 B4 13

CD 10 C3 4F 69 20 6D 75 6E 64 6F 21 0D 0A

Linguagem de montagem

A função de um montador (*assembler*) é converter a linguagem simbólica (mnemônicos) em linguagem de máquina pura. Os programadores programam em linguagem simbólica que na verdade são símbolos dos números binários, esses programas são coletados pelo montador e são convertidos em instruções binárias ou código de máquina equivalente.

A linguagem de montagem é mais fácil de programar do que a linguagem de máquina. A utilização de símbolos em vez de binário ou hexadecimais, facilita muito o desenvolvimento de programas. Veja um exemplo de linguagem simbólica abaixo que depois será convertida em linguagem de máquina:

mov dword [ebp + ebx*4], eax

Segundo Côrrea (2016), uma propriedade interessante da linguagem de montagem é que os programadores dessa linguagem têm acesso a todos os recursos e as instruções disponíveis da máquina-alvo. Mas os programadores de linguagem de alto-nível (JAVA, PYTHON, C++) não tem.

Indicação de Vídeo

Agora assista a esse vídeo que explica as diferenças de linguagem de alto e baixo nível.

Disponível em: <https://www.youtube.com/watch?v=F9zG_Oow-_Q>. Acesso em: 20.06.2020.

4:23 min.

Mas por que usar a linguagem de montagem?

A linguagem de montagem é mais fácil de programar em relação à linguagem de máquina, isso não significa que programar nesse nível seja uma coisa fácil. Pelo contrário, é algo complexo e

exige bastante conhecimento dos desenvolvedores. Desenvolver um programa em linguagem de montagem demora mais do que desenvolver em linguagem de alto nível.

Aí você pergunta: se é difícil a programação nesse tipo de nível, por que estou estudando isso? Calma! Segundo Tanenbaum (2013), há duas principais vantagens:

1. Desempenho;
2. Acesso a máquina.

Muitas vezes o programador especializado em linguagem de montagem pode produzir códigos menores e mais rápidos do que programadores de linguagem de alto nível.

Em algumas aplicações, rapidez e tamanho são essenciais. Veja, caro(a) aluno(a): aplicações de cartões inteligentes, aplicações de telefone móveis, rotinas de *BIOS*, programação de *Drives* e sistemas embarcados se encaixam nessa categoria.

Além disso, em alguns casos pode-se precisar de acesso completo ao hardware, é o caso dos controladores de dispositivos de sistemas embarcados de tempo real. Esse procedimento para linguagens de alto nível muitas vezes é difícil.

Para completar nosso raciocínio, em STALLINGS (2011, p.582), é listado uma série de vantagens e desvantagens do uso da linguagem de montagem, analise a tabela 01 abaixo:

Tabela 1. Vantagens e desvantagens da linguagem de montagem.

Desvantagem	**Vantagem**
Tempo de desenvolvimento: escrever um código em linguagem de montagem leva mais tempo do que escrever em linguagem de alto nível.	**Desenvolver compiladores**: entender técnicas de codificação em linguagem de montagem é necessário para criar compiladores, depuradores e outras ferramentas de desenvolvimento.
Confiabilidade e segurança: é fácil cometer erros no código em linguagens de montagem.	**Sistemas Embarcados**: sistemas embarcados pequenos possuem menos recursos do PC´s. A programação de linguagem de montagem pode ser necessária para otimizar o código em velocidade ou em tamanho em sistemas embarcados.

Depuração e verificação: como existe a chance de acontecer mais erros em relação à linguagem de alto nível a depuração fica mais difícil.	**Drivers para hardware e códigos de sistemas**: acessar hardware, registradores de controle do sistema e BIOS. Às vezes pode ser difícil ou impossível com um código de alto nível.
Manutenção: o código de montagem é mais difícil de modificar, pois geralmente não segue uma programação estruturada e todo tipo de truques são difíceis de serem entendidos por outras pessoas**.**	**Acessar instruções que não são acessíveis a partir das linguagens de alto nível**: certas instruções em linguagem de montagem não possuem um equivalente na linguagem de alto nível.

Fonte: STALLINGS (2011, p.582).

Videoaula 2

Utilize o QRcode para assistir!

Nesse momento assista ao vídeo que aborda sobre o motivo de usar as linguagens de montagem.

Indicação de Leitura

O aprofundamento do conteúdo começa por uma boa leitura da referência bibliográfica. Para alicerçar seus conhecimentos leia o livro: https://plataforma.bvirtual.com.br/Leitor/Publicacao/151479/pdf/0?code=iD9peEkNcGPEO8po/05L8ZKTQPcIxygFSw9VLPvazQRU35SH6e6NtQQ70/klytcYTS4TpDCzgRVo+rYNT+86Cw==

STALLINGS, William; VIEIRA, Daniel; BOSNIC, Ivan; PANNAIN, Ricardo (Rev. téc.). **Arquitetura e organização de computadores.** 8. ed. São Paulo: Pearson.

Especificamente as páginas 581 até a página a 584.

Elementos que compõem a linguagem de montagem

Vimos até agora apenas explicações teóricas, mas essenciais sobre a linguagem de montagem. Agora vamos tentar entender sobre o funcionamento prático dessa linguagem.

A estrutura da sentença da linguagem de montagem é composta por 4 importantes elementos (CÔRREA, 2016):

1. Rótulo;
2. Mnemônico;
3. Operando(s);
4. Comentário.

Observe a figura 1 abaixo:

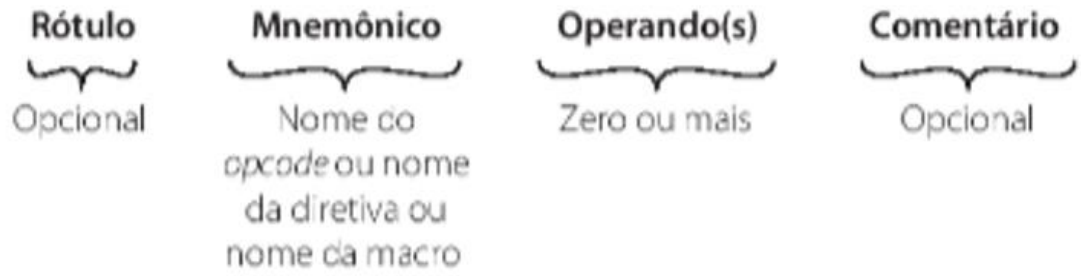

Figura 1. Estrutura da sentença da Linguagem de montagem.

Fonte: CÔRREA (2016, p.93).

Vamos estudar cada um desses quatro elementos.

Rótulo: se um rótulo está presente, o montador define o rótulo como equivalente ao endereço no qual o primeiro byte do código objeto (código de máquina) gerado para essa instrução será carregado. Observe o exemplo abaixo:

```
L2:  SUB   EAX,  EDX  ; subtrai conteúdo do registrador EDX do conteúdo
                      ; de EAX e armazena o resultado em EAX
     JG    L2         ; salta para L2 se resultado da subtração for positivo
```

O código acima continuará no laço de volta para a posição L2 até que o resultado seja zero ou negativo. JG é um Mnemônico de salto. Não se preocupe em aprender mnemônico agora, o queremos mostrar aqui é apenas como funciona a linguagem.

Mnemônico: é um nome de operação ou função da sentença da linguagem de montagem. Uma sentença pode corresponder a uma instrução de máquina ou uma diretiva do montador. No caso de uma instrução de máquina um mnemônico é o nome associado com

determinado *opcode (instrução de máquina).* Resumindo, especifica a operação a ser efetuada. Observe exemplos de mnemônicos abaixo:

Adiciona inteiros e vai acumulando no registrador acumulador eax:

add eax, 4; eax = eax + 4

mov eax, 3; grava 3 no registrador eax

Operandos: um operando pode ser de três tipos: um valor imediato, um registrador ou uma posição de memória. Geralmente a linguagem de montagem distingui esses três. Exemplo:

mov eax, 3; grava 3 no registrador acumulador eax

mov ebx, eax; grava o conteúdo de eax em ebx

Para os casos dos exemplos acima, o montador vai traduzir o nome simbólico em um identificador binário do registrador.

Comentário: todas as linguagens de montagem permitem a inserção de comentários dentro do programa. Os comentários podem ocupar o lado direito de um comando em linguagem ou ocupar uma linha inteira de texto.

Nos dois casos inicia-se o comentário com algum caractere especial que sinaliza para o montador que o restante da linha é um comentário. As linguagens de montagem para a arquitetura x86 utiliza-se de ponto e vírgula (;) como caractere especial. Veja o exemplo:

mov ah,01h ;move o valor 01h para o registrador ah

Videoaula 3

Utilize o QRcode para assistir!

Nesse momento assista ao vídeo que aborda sobre as características de uma linguagem de montagem.

Indicação de Vídeo

Agora assista a esse vídeo que exemplifica um pequeno programa em *assembly.*

Disponível em: <https://www.youtube.com/watch?v=eAQSQSOhs0s>. Acesso em: 20.06.2020.

Indicação de Leitura

O aprofundamento do conteúdo começa por uma boa leitura da referência bibliográfica. Para alicerçar seus conhecimentos leia o livro de:

STALLINGS, William; VIEIRA, Daniel; BOSNIC, Ivan; PANNAIN, Ricardo (Rev. téc.). **Arquitetura e organização de computadores.** 8. ed. São Paulo: Pearson.

Especificamente as páginas 584 a 585.

<https://plataforma.bvirtual.com.br/Acervo/Publicacao/151479>.

Montadores com passos

Nos tópicos anteriores estudamos que os montadores pegam um programa em linguagem simbólica do programador, que na verdade é uma representação simbólica dos números binários, e os convertem para instruções binárias, ou código de máquinas equivalente. O montador lê um **arquivo fonte** (programa simbólico) e produz um arquivo **objeto** (código de máquina) (NULL, 2010).

Existem duas abordagens gerais para os montadores (STALLINGS,2011): Montador de dois passos e montador de um passo. Vamos examiná-los individualmente.

Começaremos pelo montador de dois passos, parece estranho começar pelo dois, mas o de dois passos e mais fácil de se entender. O montador faz duas passagens pelo código fonte.

1. No primeiro passo, o montador se atenta apenas com as definições dos rótulos. Dessa forma, no primeiro passo é construído uma tabela de símbolos que contêm uma lista de todos os rótulos e seus valores de contador de posição.
2. No segundo passo o programa lê novamente o programa desde o começo. Assim as instruções irão se transformando em código binário de máquina apropriado. Podemos falar que tradução envolve as principais operações:
    a. Traduzir os mnemônicos em *opcode* binário.
    b. Usar o *opcode* para determinar o formato da instrução, a posição e o tamanho de vários campos da instrução.
    c. Traduzir o nome de cada operando para o registrador ou código de memória apropriado.
    d. Traduzir cada valor imediato em uma cadeia binária.
    e. Traduzir quaisquer referências a rótulos em valores do contador de posição (LC – *location counter*) apropriados usando a tabela de símbolos.

f. Definir diversos bits necessários dentro da instrução, que inclui indicadores do modo de endereçamento, bits de códigos condicionais, etc.

A figura 2 demonstra graficamente a execução do montador de dois Passos. Observe abaixo:

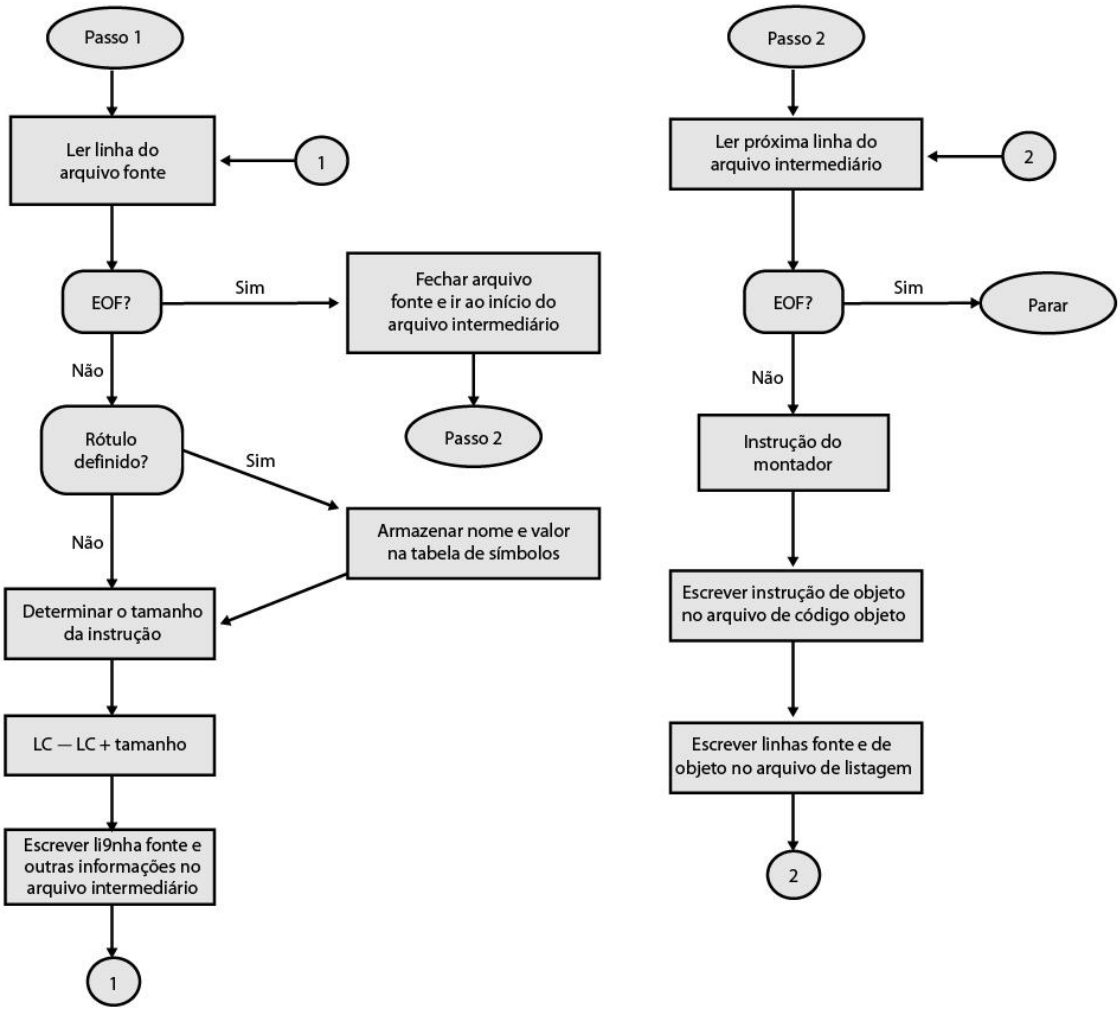

Figura 2. Fluxograma do montador de dois passos.

Fonte: STALLINGS (2011, p.590).

No final de todo esse processo teremos sequência de números binários representando a linguagem de máquina pura. A figura 3 abaixo, mostra a tradução de uma instrução em linguagem de montagem da arquitetura ARM para uma instrução binária.

Figura 3. Tradução de uma instrução em linguagem de montagem da arquitetura ARM para uma instrução binária.

Fonte: STALLINGS (2011, p.590).

Agora vamos entender o montador de 1 passo. Implementar um montador de um passo também é possível, ou seja, um montador que passa apenas uma vez pelo código. Esse fato implica em algumas restrições no programa fonte. A dificuldade envolve referências futuras a rótulos.

O problema acontece quando os operandos das instruções podem ser símbolos que ainda não foram ajustados no programa fonte. Dessa forma, o montador não sabe qual endereço relativo inserir na instrução traduzida (STALLINGS,2011).

Esse problema de referências futuras pode ser ajustado da seguinte forma pelo montador, segundo explica STALLINGS (2011):

1. Deixar o campo do operando da instrução vazio na instrução binária montada.
2. Insere o símbolo usado como um operando na tabela de símbolos. A entrada da tabela é marcada para indicar que o símbolo não está definido.
3. Adiciona ao endereço do campo de operando da instrução, que se refere ao símbolo indefinido, a uma lista de referências futuras associadas a entrada na tabela de símbolos.

Quando a definição de símbolo é encontrada de tal forma que o contador de posição (LC) possa ser associado a ele, o montador insere o valor LC na entrada adequada dentro da tabela de símbolos. Se por acaso existir uma lista de referência futura com o símbolo, então o montador insere o endereço apropriado em qualquer instrução gerada previamente que esteja na lista de referência futura.

Até aqui tivemos uma breve ideia do que é uma linguagem de montagem. Não fizemos esse estudo esperando de que você irá se tornar um programador de montadores. Em vez disso, esse estudo serve para mostrar aos estudantes uma melhor compreensão da arquitetura de máquina e como instruções e arquitetura de relacionam.

Indicação de Leitura

O aprofundamento do conteúdo começa por uma boa leitura da referência bibliográfica. Para alicerçar seus conhecimentos leia o livro de:

STALLINGS, William; VIEIRA, Daniel; BOSNIC, Ivan; PANNAIN, Ricardo (Rev. téc.). **Arquitetura e organização de computadores.** 8. ed. São Paulo: Pearson.

Especificamente as páginas 588 a 591.

<https://plataforma.bvirtual.com.br/Acervo/Publicacao/151479>. Acesso em: 20.06.2020.

Arquiteturas de computador Intel

Em computação, x86 ou 80x86 é o nome genérico dada à família (arquitetura) de processadores baseados no Intel 8086, da Intel Corporation. A arquitetura é chamada x86 porque os primeiros processadores desta família eram identificados somente por números terminados com a sequência "86": o 8086, o 80186, o 80286, o 80386 e o 80486.

Ao longo dos anos, a Intel foi adaptando a arquitetura aos novos tempos, sempre dando especial importância à compatibilidade com software escrito para versões anteriores da arquitetura *(backwards compatibility*). Talvez a principal mudança tenha sido a introdução do Modo Protegido (ver Modos de Operação), que trouxe a arquitetura para o mundo da computação moderna, possibilitando a implementação da memória virtual, diversos mecanismos de proteção e o acesso a até quatro gigabytes de memória física. A arquitetura x86 é também chamada "IA-32". Atualmente modernizada e estamos na “IA-64” (WIKIPEDIA, 2020).

Especificar toda essa arquitetura é muito difícil, mas podemos colocar os principais itens que foram estudados nessa aula:

1. Para a arquitetura x86, tem 8 registradores de uso geral (General Purpose Registers – GPR:AX, BX, CX.), 6 registradores de segmento, 1 registrador de flags e um Apontador de Execução (Instruction Pointer).
2. A arquitetura x86 é uma arquitetura CISC que, resumindo, é uma arquitetura com um conjunto complexo de instruções.
3. Cada instrução de montagem x86 é representada por uma mnemônica, que geralmente com um ou mais operadores, traduz para um ou mais bytes, chamados opcode.

A arquitetura x86 deriva diversos modelos de processadores usados até o momento. Se você se interessar nessa arquitetura pode ler o manual completo que está na nossa bibliografia em (INTEL, 2020).

Indicação de Vídeo

Agora assista a esse vídeo que aborda a família x86 de processadores, explicando a sua arquitetura.

Disponível em: <https://www.youtube.com/watch?v=qTEvYs6THwQ>. Acesso em: 20.06.2020.

Arquitetura MIPS

MIPS, acrônimo para *Microprocessor without interlocked pipeline stages* (microprocessador sem estágios interligados de pipeline - não confundir com os outros significados de "MIPS"), as arquiteturas MIPS iniciais foram de 32 bits, com versões de 64 bits adicionados mais tarde.

O MIPS é o nome de uma arquitetura de processadores baseados no uso de registradores. As suas instruções têm à disposição um conjunto de 32 registradores para realizar as operações (SANDRO; WEIZENMANN, 2016).

Entretanto, alguns destes registradores não podem ser usados por programadores, pois são usados pela própria máquina para armazenar informações úteis. A arquitetura MIPS atual tem tamanho fixo de 64 bits e é chamada de MIPS 64 Bits.

Processadores MIPS desenvolvido pela *MIPS Computer Systems* são do tipo RISC (*Reduced Instruction Set* Computer) - ou seja, Computadores com Conjunto de Instruções Reduzidas. Isso significa que existe um conjunto bastante pequeno de instruções que o processador sabe fazer. Combinando este pequeno número, podemos criar todas as demais operações.

Projetos MIPS são usados em uma ampla gama de dispositivos: sistemas embarcados, roteadores, telefones e videogames. Vamos então ver as suas principais características:

Algumas características do MIPS de 64 bits são:

- 32 registradores de uso geral;
- Tipos de dados de 8, 16, 32 e 64 bits;
- Atuação sobre inteiros de 64 bits (MIPS4);
- Memória endereçável por byte com endereços de 64 bits;
- Modos de endereçamento Imediato, registrador e Deslocamento.

É muito interessante saber que existem muitas arquiteturas para fabricação de processadores, cada qual atuando em ramo do mercado. Até aqui estudamos duas principais arquiteturas de computadores usadas x86(Intel) e MIPS.

Mas exemplos de arquitetura não se restringe a apenas essas arquiteturas. Podemos citar outras como: **SPARC, ARM e POWERPC.** Interessante notar é que todas têm a sua linguagem de montagem para implementação de sistemas computacionais.

Termos usados no estudo da programação

Em nosso estudo, nos deparamos com diversos termos usados quando se faz um estudo sobre as arquiteturas dos computadores. Por muitas vezes, esses termos acabam confundindo o estudante e atrapalhando o aprendizado. Por isso, para diferenciar alguns termos usados no nosso estudo, é apresentado na tabela 2 abaixo os significados referentes dos termos.

Montador
Um programa que traduz a linguagem de montagem para o código de máquina.
Linguagem de montagem (Linguagem de montagem)
Uma representação simbólica da linguagem de máquina de um processador específico, acrescidas de tipos de instruções adicionais que facilitam a escrita do programa e que fornecem instruções para o montador.
Compilador
Um programa que converte outro programa de alguma linguagem fonte (ou linguagem de programação) para linguagem de máquina (código aberto). Alguns compiladores geram saída em linguagem de montagem que é então convertida para linguagem de máquina por um montador diferente. Um compilador se distingue de um montador pelo fato de que cada instrução de entrada, em geral, não corresponde a uma única instrução de máquina ou uma sequência fixa de instruções. Um compilador pode suportar recursos como alocação automática de variáveis, expressões aritméticas arbitrárias, estruturas de controle de laços como FOR e WHILW, escopo de variável, operações de entrada/saída, funções de alto nível e portabilidade de código fonte.
Código executável
O código de máquina gerado por um processador de linguagem de código fonte como um montador ou um compilador, isto é, software em uma forma que pode ser executada no computador.
Conjunto de instruções

O conjunto de todas as instruções possíveis para um determinado computador, isto é, conjunto de instruções de linguagem de máquina que um determinado processador entende.
Linker (ligado)
Um programa utilitário que combina um ou mais arquivos contendo código objeto de módulos de programa compilados separadamente para um arquivo único contendo código carregável ou executável.
Loader (Carregador)
Uma rotina de programa que carrega um programa executável na memória para execução.
Linguagem de máquina ou código de máquina
Representação binária de um programa de computador que é lido e interpretado de fato pelo computador. Um programa em código de máquina consiste de uma sequência de instruções de máquina (possivelmente intercaladas com dados). Instruções são cadeias binárias que podem ser todas do mesmo tamanho (por exemplo, uma palavra de 32 bits para muitos microprocessadores RISC modernos) ou de tamanhos diferentes.
Código objeto
Representação, em linguagem de máquina, do código fonte de programação. O código objeto é criado pelo compilador ou montador e é transformado em código executável pelo linker.

Tabela 2. Significados dos termos usados no estudo da arquitetura de computadores.

Fonte: STALLINGS (2011, p.582).

Aula 2 Linguagem de montagem em processadores MIPS

Olá! Seja bem-vindo(a) novamente, querido(a) aluno(a). Vamos continuar em nossa missão de aprender. Você já está chegando ao fim, mantenha a calma que está quase acabando.

Na aula anterior, o estudo sobre a linguagem de montagem foi introduzido. Obviamente não foi um estudo detalhado, pois precisaríamos de capítulos e mais capítulos de estudo para aprender ao todo.

O intuito da introdução de uma linguagem de montagem é para que você adquira compreensão sobre arquitetura de máquina e também possa aprender o que acontece nos bastidores de programas de alto nível. Quando for um desenvolvedor de sistemas, verá que isso é muito importante. Além disso, outros conhecimentos em paralelo vão se juntando também, tais como os da lógica digital: registradores, memória e endereços hexadecimais.

Nessa aula vamos partir para a prática. A experiência nos mostra que os alunos muitas vezes não absorvem os conceitos de arquitetura de computador porque ficam muito presos à teoria de seus materiais, muitas vezes levando-o a não gostar da matéria. Pensando nisso, vamos estudar uma ferramenta de simulação para que possamos colocar em prática a teoria.

Os simuladores são ideais para se estudar computadores e softwares. Adotaremos como simulador o software MARS. O MARS é um ambiente de desenvolvimento interativo leve (IDE) para programação na linguagem *assembly* MIPS, destinado ao uso em nível educacional. O software pode ser baixado em <http://courses.missouristate.edu/KenVollmar/mars/download.htm>.

ISA (instruction set architecture) do MIPS

Antes de começar a simular, é necessário o conhecimento básico da arquitetura de processador MIPS. Em nosso estudo, concentraremos na arquitetura de processador MIPS, pois trata-se de uma arquitetura mais simples para estudo. Um diagrama simplificado dessa construção é mostrado na figura 1 abaixo:

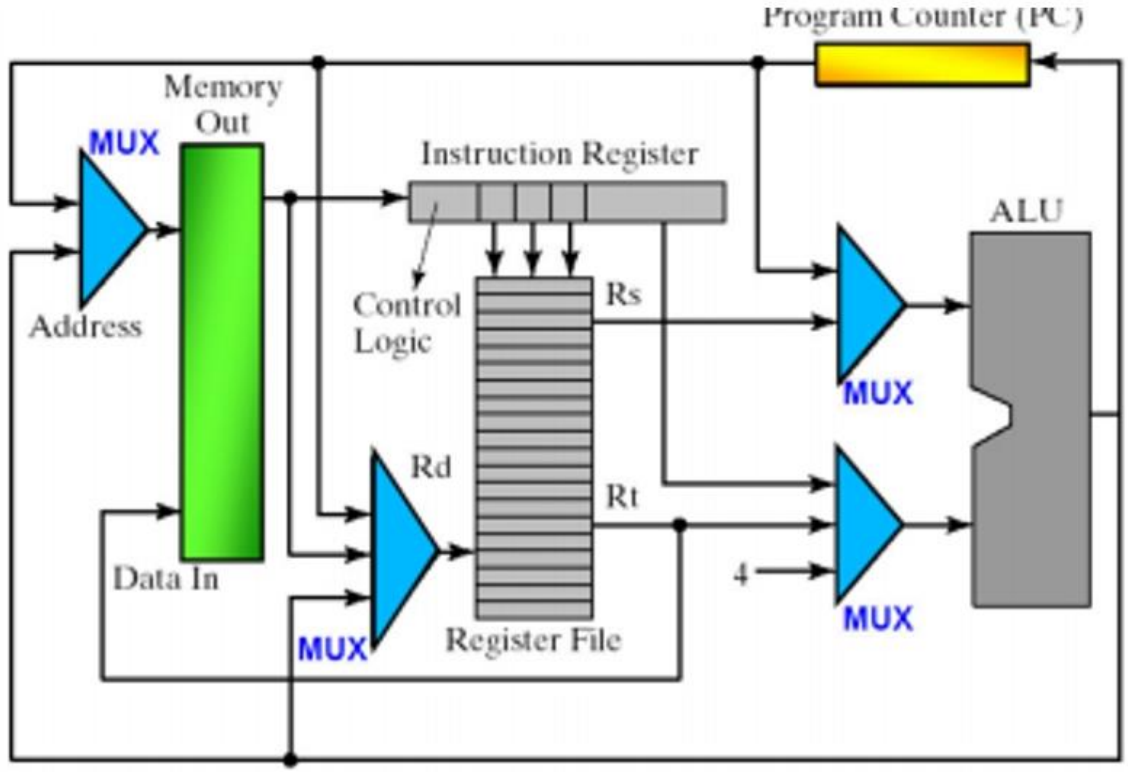

Figura 1. Diagrama simplificado do MIPS.

Fonte:<http:/files.ccfacape.webnode.com/200000135-e9634ea5d4/MIPS I.pdf >.
Acesso em: 20.06.2020.

Analisando a figura acima, notamos vários componentes lógicos básicos estudados anteriormente em nosso material: Barramento, Unidade de controle, Banco de registradores Unidade lógica e aritmética (ALU), Contador de programa (PC), Memória e Registrador de instruções (IR). Todas executando suas funções específicas dentro da arquitetura.

Todo processador tem sua **ISA (*instruction set architecture***), ou seja, um conjunto de instruções para manipulação de dados em processador. O processador MIPS é um RISC de 32 Bits, tamanho da palavra: 32 bits, tamanho dos registradores: 32 bits, tamanho das Instruções: 32 bits e tamanho dos endereços de memória: 32 bits.

Todo dado que é processado precisa passar por registradores. A CPU é composta por registradores, que é um tipo de memória mais rápida e tem um custo mais elevado no projeto de um microprocessador, pois precisa estar internamente a uma CPU, para ajudar a executar as instruções.

Como estudado anteriormente, o MIPS é composto de um banco 32 registradores de uso geral, na tabela 1 é elencado os registradores.

Os registradores são procedidos de $ nas instruções, as variáveis que armazenam valores usam os registradores de nome **t** e as variáveis que contêm os operandos usam os registradores de nome **s**.

Tabela 1. Representação dos registradores da arquitetura MIPS.

Nome do Registrador	Número	Binário	Uso
$zero	0	000 000	constante zero
$at	1	000 001	reservado para o montador
$v0	2	000 010	avaliação de expressão e resultados de uma função
$v1	3	000 011	avaliação de expressão e resultados de uma função
$a0	4	000 100	argumento 1 (passam argumentos para as rotinas)
$a1	5	000 101	argumento 2
$a2	6	000 110	argumento 3
$a3	7	000 111	argumento 4
$t0	8	001 000	temporário (valores que não precisam ser preservados entre chamadas)
$t1	9	001 001	temporário

$t2	10	001 010	temporário
$t3	11	001 011	temporário
$t4	12	001 100	temporário
$t5	13	001 101	temporário
$t6	14	001 110	temporário
$t7	15	001 111	temporário
$s0	16	010 000	temporário salvo (valores de longa duração e devem ser preservados entre chamadas)
$s1	17	010 001	temporário salvo
$s2	18	010 010	temporário salvo
$s3	19	010 011	temporário salvo
$s4	20	010 100	temporário salvo

$s5	21	010 101	temporário salvo
$s6	22	010 110	temporário salvo
$s7	23	010 111	temporário salvo
$t8	24	011 000	temporário
$t9	25	011 001	temporário
$k0	26	011 010	reservado para o Kernel do sistema operacional
$k1	27	011 011	reservado para o Kernel do sistema operacional
$gp	28	011 100	ponteiro para área global
$sp	29	011 101	stack pointer (aponta para o último local da pilha)
$fp	30	011 110	frame pointer (aponta para a primeira palavra do frame de pilha do procedimento)
$ra	31	011 111	endereço de retorno de uma chamada de procedimento

Fonte: <http://www.inf.ufpr.br/wagner/ci243/GuiaMIPS.pdf>. Acesso em: 20.06.2020.

Temos também a tabela de operações com seus respectivos valores em decimal e binário. Por ser tratar de muitas operações, será apresentado apenas algumas como exemplo.

Tabela 2. Tabelas de códigos do opcode.

OPCODE	DECIMAL	BINÁRIO
ADD	32	100 000
SUB	34	100 010
OR	36	100 100
AND	37	100 101

Fonte: elaborado pelo próprio autor.

Armazenamento na memória

A arquitetura MIPS utiliza o método ***Big endian*** para armazenar dados. Mas o que será isso? Tudo bem! Vou explicar os métodos existentes para ficar mais claro. Há dois sistemas para numeração dos bytes dentro uma palavra.

- ***Big endian*** ⇒ byte mais à esquerda marca endereço da palavra.
- ***Little endian*** ⇒ byte mais à direita marca endereço da palavra.

Palavra é uma unidade completa de informação que consiste de uma unidade de dados em binário. Quando aplicada às instruções de um computador, uma palavra pode ser mais especificamente definida com 1 byte, 2 bytes, 4 bytes.

Vamos mostrar um exemplo para esclarecer esse conceito. Observe abaixo os itens **palavra e endereço** com valores em hexadecimal.

Exemplo: palavra = $6151CE94_h$, endereço = $0F40h_h$

1) Big Endian

$0F40_h$	61
$0F41_h$	51
$0F42_h$	CE
$0F43_h$	94

Observando esse exemplo acima, notamos que o mapeamento armazena o bit mais significativo (mais à esquerda) no endereço de memória mais baixo; isso é conhecido como ***big-endian***, e equivale a você escrever da esquerda para direita.

2) Little Endian

$0F40_h$	94
$0F41_h$	CE
$0F42_h$	51
$0F43_h$	61

Notando esse outro exemplo acima, vemos que o mapeamento armazena o byte menos significativo (mais à direita) no endereço de byte mais baixo; isso é conhecido como ***little-endian.***

Existe toda uma discussão sobre o assunto de autores da área sobre o uso de uma ou de outra técnica. Para (NULL, 2010, p.275) existem vantagens e desvantagens em ambos os métodos. O que de fato é importante para nós nesse momento, é que o MIPS utiliza o método *BIG-ENDIAN.*

Em algumas aplicações práticas, como por exemplo, as representações gráficas em *bitmap,* utilizam-se do esquema “o bit mais significativo na esquerda”. Nesse caso, em específico, o modelo *big-endian* tem a sua vantagem.

Instruções

Todas as instruções aritméticas e lógicas do MIPS possuem três operandos. Esse fato favorece a simplicidade, pois mais do que três operandos por instrução exigiria um projeto de hardware mais complexo. Observe o exemplo abaixo.

[Label:]	***Opcode***	**[Operando],**	**[Operando],**	**[Operando],**	**#comentário**

- ***Label*:** opcional, identifica bloco do programa.
- **Código de operação (*Op-Code)***: indicado por um mnemônico.
- **Operandos**: Registradores ou memória.
- **Comentários**: opcional, tudo que vem depois do # é apenas comentário do código.

Exemplo de linguagem de montagem (*Assembly*):

- **add $s0, $s1, $s2 >> soma**
- **sub $s0,$t0,$t1 >> subtração**
- **lw $t0, 8($s3) >> movimentação de dados**

Representando Instruções no computador

As instruções de uma linguagem de montagem devem ser traduzidas em números binários para que a máquina a reconheça e as execute. Sendo assim, cada instrução e cada registrador devem ser mapeados com base em um código e dispostos segundo um dos seguintes formatos:

Formato registrador (R)

Esse tipo de formato é usado para instruções aritméticas e lógicas.

- **Opcode** - sempre zero para o formato R
- **rs** - o primeiro registrador fonte.

- **rt -** o segundo registrador fonte.
- **rd** - o registrador destino.
- **shamt** - quantidade de deslocamento.
- **functcode** - Seleciona variações das operações especificadas pelo *opcode*.

opcode	**rs**	**rt**	**rd**	**shamt**	**Funct**
6 bits	**5 bits**	**5 bits**	**5 bits**	**5 bits**	**6 bits**

Como exemplo para o formato R, vamos representar um código de montagem em seu código de máquina. Queremos fazer a seguinte operação **a=b+c**. Então o código digitado pelo programador seria:

ADD $t0, $s0, $s1.

Cada registrador tem um número, conforme apresenta a Tabela 1. A Linguagem de Máquina corresponde a trocar o nome do registrador pelo seu número. Então a instrução ficaria **ADD $8, $16, $17, pois t0=8, s0=16, s1=17.**

Opcode	Rs	Rt	Rd	Shamt	Funct
0	$16	$17	$8	0	32

Instruções de formato tipo R sempre terão o **opcode** igual a zero e **funct** será correspondente à instrução específica. O código de **Funct** = 32 está contido na tabela 2.

Então a instrução está representada no seu formato específico, todos seus valores acima devem ser transformados em binário chegando ao código final de máquina**,** conforme abaixo:

op	rs	rt	rd	shamt	funct
000000	10000	10001	01000	00000	100000

O código binário **00000010000100010100000000100000** é a instrução a = b + c em MIPS 32 bits. Os demais formatos seguem a mesma ideia.

As principais instruções no formato R são: add,sub,and,or,xor,nor,slt,sll,srl,jr.

Formato imediato (I)

Esse tipo de formato é usado para instrução de transferência de dados e operandos imediatos.

- **Opcode:** especifica qual operação a ser executada**.**
- **Rs:** registrador do operando de origem**.**
- **Rt:** registrador que recebe o resultado da operação (destino).
- **Immediate:** endereço de memória ou constante numérica.

Opcode	rs	rt	Immediate
6 bits	5	5	(16 bits)

As principais instruções no formato I são: addi, andi, ori, xori, slti,lw,sw,beq,bne.

Formato de jump (J)

Esse tipo de formato é usado para instrução que contenha saltos.

- **Opcode:** especifica qual operação a ser executada.
- **Target:** local da memória a saltar, onde estão as próximas instruções a serem executadas.

op	Target
6 bits	26 bits

As principais instruções no formato J são: j e jal.

Instruções de Movimentação de Dados

Load e Store

- **lw** : instrução de movimentação de dados da memória para registrador (load word).
- **sw**: instrução de movimentação de dados do registrador para a memória (store word).

Exemplos de instrução de movimentação de dados:

lw $t0, 32($s3)

sw $t0, 48($s3)

Indicação de Vídeo

Agora assista a esse vídeo que explica sobre a arquitetura MIPS. Este vídeo tem ótimos exemplos.

Disponível em: <https://www.youtube.com/watch?v=pDHVrkiG4eQ>. Acesso em: 20.06.2020.

16:51 min.

Indicação de Leitura

O aprofundamento do conteúdo começa por uma boa leitura da referência bibliográfica. Para alicerçar seus conhecimentos leia o livro de:

Corrêa, Ana Grasielle Dionísio**. Organização e Arquitetura de Computadores**. Pearson Educational do brasil. 1 ed. 2016.

Disponível em: <https://plataforma.bvirtual.com.br/Acervo/Publicacao/124147>. Acesso em: 20.06.2020.

Especificamente as páginas 131 a 139.

Instalando o MARS (Mips Assembler Runtime Simulator)

Depois de estudar a teoria básica, vamos nos concentrar no estudo do simulador MARS para aprender a usar instruções no MIPS.

Após realizar o download, tudo o que você precisará fazer é clicar duas vezes na aplicação, que na verdade é um JAR, ou seja, uma aplicação Java Desktop. Ao fazer isso, você verá uma tela parecida com a apresentada na Figura 2:

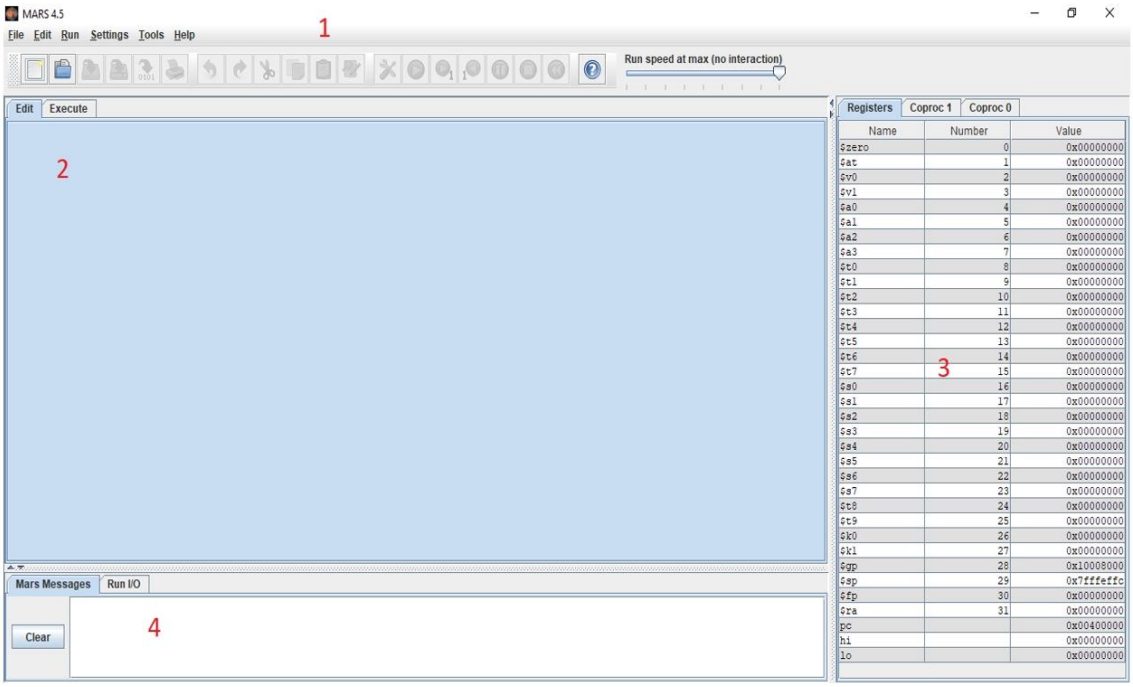

Figura 2. Interface do MARS.

Fonte: elaborado pelo próprio autor.

Videoaula 1

Utilize o QRcode para assistir!

Nesse momento assista ao vídeo que aborda sobre as características de uma linguagem de montagem.

É importante conhecer um pouco da ferramenta de simulação MARS, trata-se de uma ferramenta conhecida pela sua simplicidade. Note que na figura 2, as partes mais importantes da ferramenta estão numeradas. Vamos explicar essas partes separadamente.

1. Nessa parte se encontra os principais menus da ferramenta onde pode-se alterar algumas configurações e os botões que executam alguma função.
2. Área onde você edita seu programa e também executa.
3. Registers: Mostra os registradores do MIPS, e seus conteúdos. É possível alterá-los.
4. É o console no qual os programas podem imprimir mensagens e receber dados pelo usuário.

Como você pode notar, realmente é uma aplicação muito simples que não precisa de muitas explicações detalhadas. Os detalhes serão explicados ao mesmo tempo em que desenvolvemos exercícios práticos.

Indicação de Vídeo

Agora assista a esse vídeo que ajudará você a entender o simulador MARS. O vídeo contém exemplos práticos de utilização.

Disponível em: < https://www.youtube.com/watch?v=j2eeyDAKzuk>. Acesso em: 20.06.2020.

Utilizando o montador MARS

Começaremos com um projeto de somar dois números e acumular esse valor em uma variável. A instrução em alto nível seria do tipo a = b+ c.

Começaremos abrindo um novo arquivo no MARS: Barra de **MENU FILE >> NEW.** A janela mostrada na figura 2.2 irá aparecer, mostrando as guias EDIT e EXECUTE e os respectivos registradores em branco.

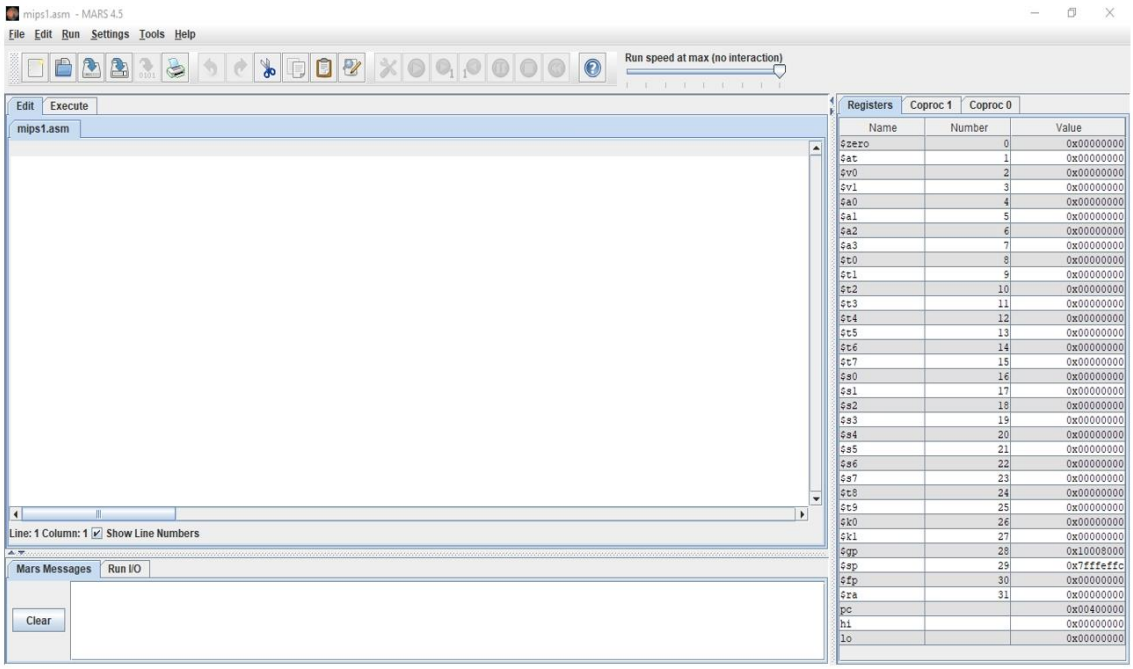

Figura 3. Interface do MARS sem dados.

Fonte: elaborado pelo próprio autor.

Iremos literalmente programar agora para o processador MIPS. Comece a digitar o seguinte código mostrado na figura 4 abaixo no MARS.

Edit Execute

PrimeiroPrograma.asm*

```
1  #a=b+c
2     .data
3
4  x: .word 10            # cria uma variável x do tipo WORD com valor 10
5  y: .byte  5            # cria uma variável y do tipo BYTE com valor 5
6
7     .text
8
9  lw $t2,x               #pega o valor de x e coloca em t2, t2 seria o b
10 lb $t3,y               #pega o valor de y e coloca em t3, t3 seia o c
11 add $t1,$t2,$t3        # armazena t2 e t3 em t1
```

Figura 4. Código em *assemby* para somar dois números.

Fonte: elaborado pelo próprio autor.

O código está totalmente comentado, ficando fácil para você, aluno(a), entender a lógica. Pois bem! Você desenvolveu um somador em linguagem de baixo nível.

Videoaula 2

Utilize o QRcode para assistir!

Nesse momento assista ao vídeo que aborda sobre as características de uma linguagem de montagem.

Depois de digitado o código vamos ver o resultado dentro dos registradores. Execute as funções F3 do teclado para montar o código. Após isso, execute a tecla F5 do teclado para executar o programa. O programa aparecerá da seguinte forma conforme mostra a figura 5.

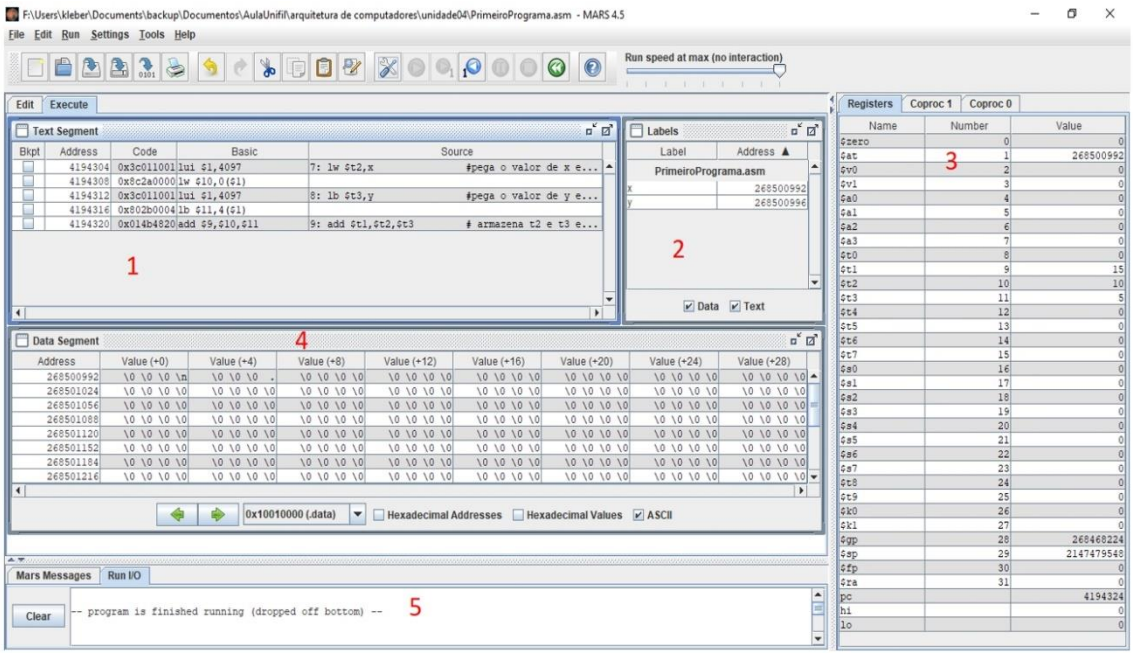

Figura 5. Resulta da programação.

Fonte: elaborado pelo próprio autor.

Foi enumerada a figura 5 para podermos explicar os campos assinalados e os resultados do projeto. Então vamos aos campos:

1. **Text Segment**: mostra a memória de programa (*address*), com os *opcodes* destacando a instrução atual. Nessa área você analisa o seu código digitado anteriormente. Repare o seu código mostrado tendo um endereçamento.
2. **Variáveis:** está janela contém as variáveis declaradas e os seus respectivos endereços.

3. **Registers:** mostra os registradores do MIPS e seus conteúdos. É possível alterá-los. Note os registradores $t1=15, $t2=10, $t3=5, que são do tipo temporário e estão preenchidos com os valores conforme a programação feita. Vale a pena notar também, o uso do registrador PC, onde o mesmo aponta para a próxima instrução. Também conseguimos visualizar o registrador SP que aponta para o último local da pilha.
4. **Data Segment**: mostra a memória de dados, é possível alterar seus valores e o formato no qual são exibidos. Repare os valores da programação incluídos no endereço de memória.
5. **Run I/O**: é o console no qual os programas podem imprimir mensagens e receber dados pelo usuário.

Videoaula 3

Utilize o QRcode para assistir!

Nesse momento assista ao vídeo que aborda sobre as características de uma linguagem de montagem.

Adorados(as) alunos(as), chegamos aqui ao final dessa aula. Existem muitos conceitos dentro desse pequeno projeto que foi executado por vocês, sendo impossível de falar de todos aqui devido ao nosso limite de páginas.

Mas com certeza, se você ler todo o texto dessa unidade, acompanhar os vídeos indicados e ler as indicações obrigatórias de leitura, terá um conhecimento bem amplo sobre uma linguagem de montagem. Um abraço e até mais!

Encerramento

Chegamos ao final de nossa jornada pelos níveis do computador. Nessa unidade você começou estudando o conceito de linguagem de montagem.

Você notou as principais diferenças entre uma linguagem de montagem e linguagem de máquina. Uma diferença importante, é que a linguagem de máquina utiliza de números binários, enquanto que a linguagem de montagem utiliza de nomes simbólicos (mnemônicos). Não podemos esquecer que abordamos de maneira básica os mecanismos de funcionamento de uma linguagem de montagem.

Na última aula abordamos um pouco a linguagem de montagem que dá suporte aos processadores MIPS. Você aprendeu a notação e a forma que se utiliza os comandos do MIPS. Algumas características importantes também a respeito da arquitetura foram abordadas.

Mostramos o uso do simulador MARS, identificamos que o uso de simuladores para aprender determinado tema é muito importante para o aprendizado. Com esse simulador, conseguimos colocar em prática os conceitos de linguagem de montagem.

Por fim, a abordagem do tema “linguagem de montagem” nos mostrou como de fato trabalha o computador em níveis mais baixos, começando pela linguagem e chegando até os registradores. Com isso, aluno(a), acredito que sua visão está ampliada para vencer os desafios no ramo da programação.

Encerramento da Disciplina

Nossa, foi tão rápido a nossa disciplina! Espero que tenha gostado dos assuntos tratados durante os seus estudos. Desenvolver conteúdo para área tecnológica sempre é um desafio, pois, todos os dias tem uma nova inovação.

Tentamos desenvolver o conteúdo sempre com base em livros que tem seu reconhecimento fundamentado. Porém, alguns assuntos precisam de um novo enfoque, nesse caso a internet foi uma grande aliada.

Abordamos assuntos com o enfoque ao curso de análise de desenvolvimento de sistemas. Começamos com as bases mais distantes do conhecimento sobre a computação, pois, apesar de serem velhas, ainda servem como referência até hoje e, é caso da máquina de *John von Neumann*.

Trabalhamos também com números e códigos binários, pois, esses representam ainda nos dias atuais a base da computação, nenhum profissional da área da computação deve ficar sem esse conhecimento. Não poderíamos deixar de estudar também o mecanismo de funcionamento dos circuitos digitais, uma vez que o mundo está totalmente eletrônico.

No final, trabalhamos no estudo de uma linguagem de montagem, visto que esse assunto é uma junção de diversos assuntos desde hardware até a programação. Seu objetivo foi colocar algo prático em meio de tanta teoria.

Por fim, acredito que a motivação de você, aluno(a), estudar como os computadores funcionam, é querer deixar os sistemas mais otimizados e alinhados para o bom funcionamento. Boa sorte nessa sua viagem!

Encerramento da Disciplina

Utilize o QRcode para assistir!

Referências

CORRÊA, Ana Grasielle Dionísio. **Organização e Arquitetura de Computadores**. Pearson Educational do brasil. 1 ed. 2016. Disponível em: < https://plataforma.bvirtual.com.br/Acervo/Publicacao/124147>. Acesso em: 20.06.2020.

TANENBAUM, Andrew S.; AUSTIN, Todd. **Organização estruturada de computadores.** 5. ed. São Paulo: Pearson, 2013. ISBN 9788581435398. Disponível em: <https://plataforma.bvirtual.com.br/Acervo/Publicacao/355>>. Acesso em: 20.06.2020.

STALLINGS, William; VIEIRA, Daniel; BOSNIC, Ivan; PANNAIN, Ricardo (Rev. téc.). **Arquitetura e organização de computadores.** 8. ed. São Paulo: Pearson, 2011. 624 p. ISBN 978-85-7605-564-8. Disponível em: <https://plataforma.bvirtual.com.br/Acervo/Publicacao/151479>>. Acesso em: 20.06.2020.

NULL, Linda; LISBÔA, Maria Lúcia B.; LISBÔA, Carlos Arthur L. (Rev. téc.). **Princípios básicos de arquitetura e organização de computadores.** 2. ed. Porto Alegre: Bookman, 2010. 821 p. ISBN 978-85-7780-737-6 – Acervo físico da Universidade.

WIKIPEDIA (Estados Unidos) (org.). **X86**. 2020. Disponível em: <https://pt.wikipedia.org/w/index.php?title=X86&oldid=54792445>. Acesso em: 20.06.2020.

INTEL (Estados Unidos) (org.). **Intel® 64 and IA-32 Architectures Software Developer Manuals**. 2020. Disponível em: <https://software.intel.com/content/www/us/en/develop/articles/intel-sdm.html>. Acesso em: 20.06.2020.

SANDRO; WEIZENMANN, Dinara. **MIPS**. 2016. Pesquisa em youtube.com. Disponível em <https://www.youtube.com/watch?v=pDHVrkiG4eQ&list=PLMp_uyIdPLbwn2x8z88vZ89UB6y16_jT0&index=2> Acesso em: 17 jul. 2020.

Esperamos que este guia o tenha ajudado compreender a organização e o funcionamento de seu curso. Outras questões importantes relacionadas ao curso serão disponibilizadas pela coordenação.

Grande abraço e sucesso!

UniFil