**Apostila para teste de instruções MIPS no MARS**

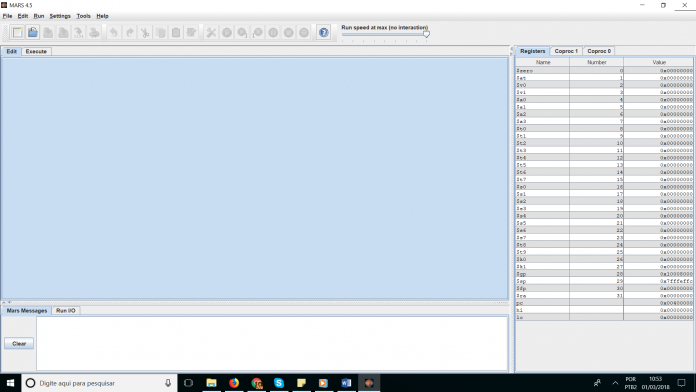
O **MARS** é um simulador para linguagem MIPS.

**Como instalar o MARS**

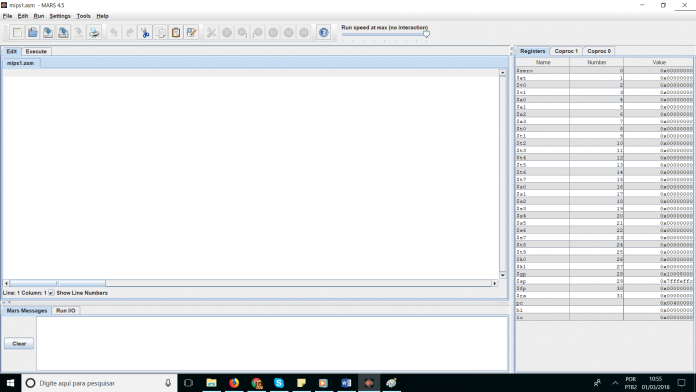
Baixe [aqui](http://courses.missouristate.edu/KenVollmar/mars/) (http://courses.missouristate.edu/KenVollmar/mars/) o simulador. Neste site você também encontra outras informações relevantes sobre o simulador.

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 1:

[](https://www.embarcados.com.br/wp-content/uploads/2018/03/mars-tela-1.png)Figura 1: Tela Inicial do MARS

Agora faça o seguinte, vá em FILE e escolha NEW, a sua tela vai ficar conforme a Figura 2:

[](https://www.embarcados.com.br/wp-content/uploads/2018/03/mars-tela-2.png)Figura 2: Criando um novo arquivo ASM no MARS

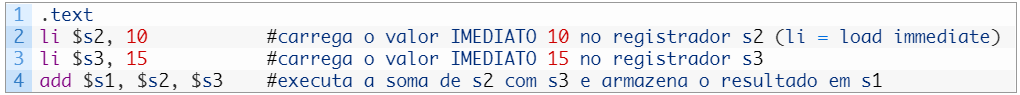
Agora vamos testar o nosso primeiro exemplo que aprendemos quando começamos a estudar as instruções.

**Exemplo 1**

Vamos supor que $s2 seja 10 e $s3 seja 15, para podermos testar aqui a operação.

**ADD      $s1, $s2, $3**

Para testar esse código você vai digitar o seguinte na aba **mips1.asm**



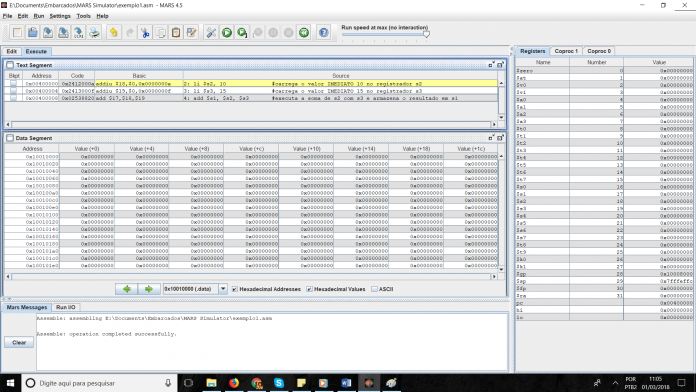
Após digitar isto, salve o arquivo.

Vá em FILE, escolha SAVE, escolha a pastinha onde você deseja salvar seus arquivos “.asm”, que são os arquivos Assembly, e dê um nome para o seu arquivo.

Aqui neste exemplo foi escolhida a própria pasta do MARS. Nela foi criada uma pasta chamada EXEMPLOS, e demos os nomes de exemplo1.asm, exemplo2.asm e assim por diante.

**“.text”** é uma diretiva do montador, indicando que as linhas a seguir contêm instruções.

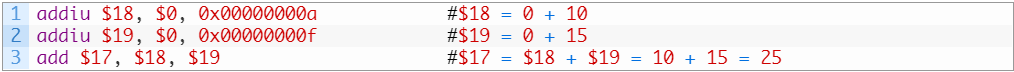
Para executar, vá até o menu RUN e escolha ASSEMBLE. Quando fizer isto, a tela mudará, conforme a Figura 3:

[](https://www.embarcados.com.br/wp-content/uploads/2018/03/mars-tela-3.png)Figura 3: Executando o código assembly MIPS no MARS

Observe as seguintes seções nesta tela:

**Execute:**Apresenta uma tabela com todas as linhas do seu código com o respectivo endereço de memória, e código de máquina, em hexadecimal, assim como o código Basic e o Source, que é o código fonte.

Note que em Basic, o código fonte que digitamos em MIPS fica da seguinte forma:



Conforme a referência MIPS, ADDIU é uma instrução de adição imediata sem overflow, que coloca a soma do registrador RS e o imediato com sinal estendido no registrador RT.

Basicamente, aqui os valores imediatos que determinamos, 10 e 15, foram convertidos para o sistema de numeração hexadecimal, A e F, respectivamente, e então somados com ZERO ($0).

Só relembrando...., o número 25 convertido em hexadecimal é **19**, em binário é **00011001** e em octal é **31**.

Observe que na coluna ADDRESS os endereços estão sempre como múltiplos de 4: 0x00400000, 0x00400004 e 0x00400008. Note também que a primeira linha do código está destacada em amarelo.

**Data Segment:** Essa é a área de segmento de dados (DS), ou mais famosamente conhecida como Memória! Vamos aprender mais sobre essa Aba em outros exemplos. Podemos escolher a forma que se quer ver os endereços de memória selecionando-os nos checks boxes logo abaixo: endereços hexadecimais, valores hexadecimais e ainda ASCII. Também podemos escolher a região de memória a partir das setinhas para a direita e esquerda e do combobox.

**Mars Messages:** Apresenta mensagens produzidas pelo menu RUN.

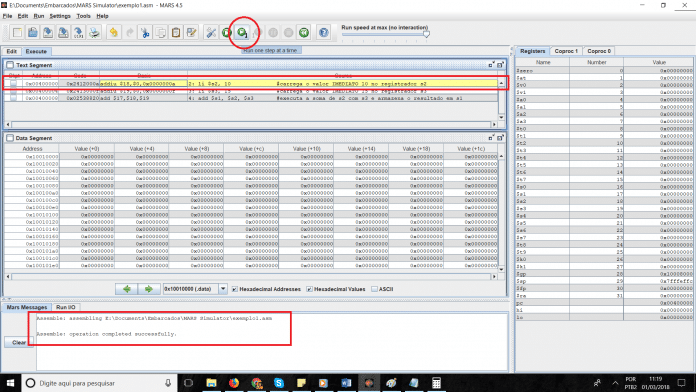
**Run I/O:** É um console de entrada e saída do simulador.

**Registers:** Referente aos registradores do MIPS.

**Coproc1:** Referente aos registradores da unidade de ponto flutuante.

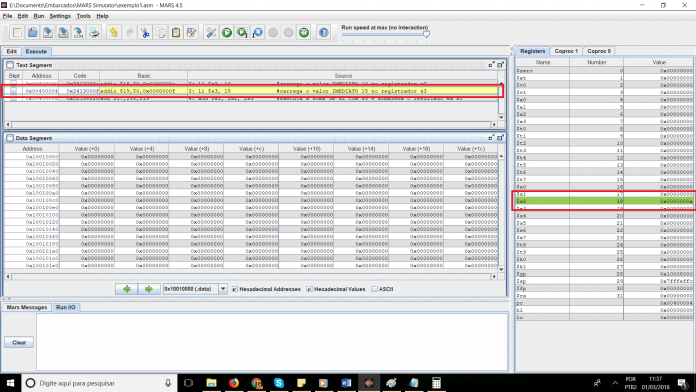
**Coproc0:** Referente aos registradores de interrupções e exceções.

Para ver a execução, você utilizará o botão de execução passo a passo, conforme apresenta a Figura 4:

[](https://www.embarcados.com.br/wp-content/uploads/2018/03/mars-tela-4-1.png)Figura 4: Botão de execução passo a passo (linha a linha)

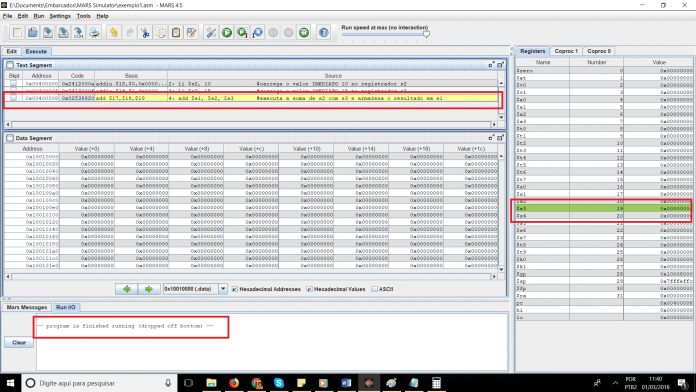
Clique apenas uma vez neste botão.

Note que ela marca o registrador **$s2** de verde, mostrando o valor dele, **0x00000000a**, e marca a segunda linha como ativa. Verifique na Figura 5:

[](https://www.embarcados.com.br/wp-content/uploads/2018/03/mars-tela-5.png)Figura 5: Valores e estados dos registradores.

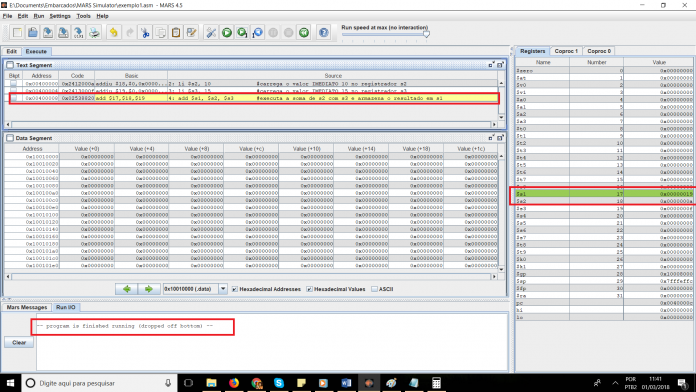
Clique mais uma vez neste botão.

Agora a terceira linha está ativa e o registrador **$s3** ficou verde, com o valor **0x00000000f**. Verifique na Figura 6:

[](https://www.embarcados.com.br/wp-content/uploads/2018/03/mars-tela-6.png)Figura 6: Valores e estado dos registradores

Clique mais uma vez neste botão.

A execução do código é finalizada e o registrador **$s1** é marcado de verde, com o valor **0x000000019**. Verifique na Figura 7:

[](https://www.embarcados.com.br/wp-content/uploads/2018/03/mars-tela-7.png)Figura 7: Finalizando a execução passo a passo.

Dessa forma, vimos como executar nossos programas de forma bem simples no MARS, e verificar seus valores.

**Exemplo 2**

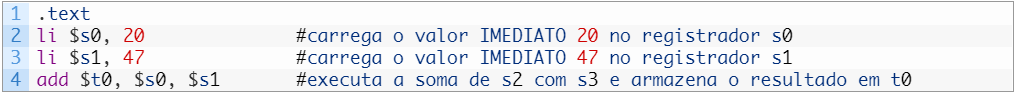
Vamos ver mais um exemplo, que já foi visto em aula:

a = b + c;

ADD $t0, $s0, $s1

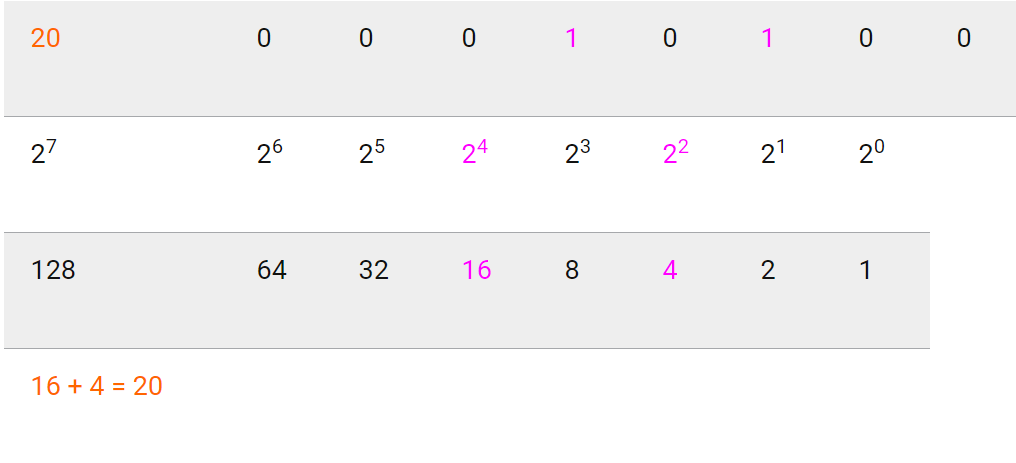
ADD $8, $16, $17

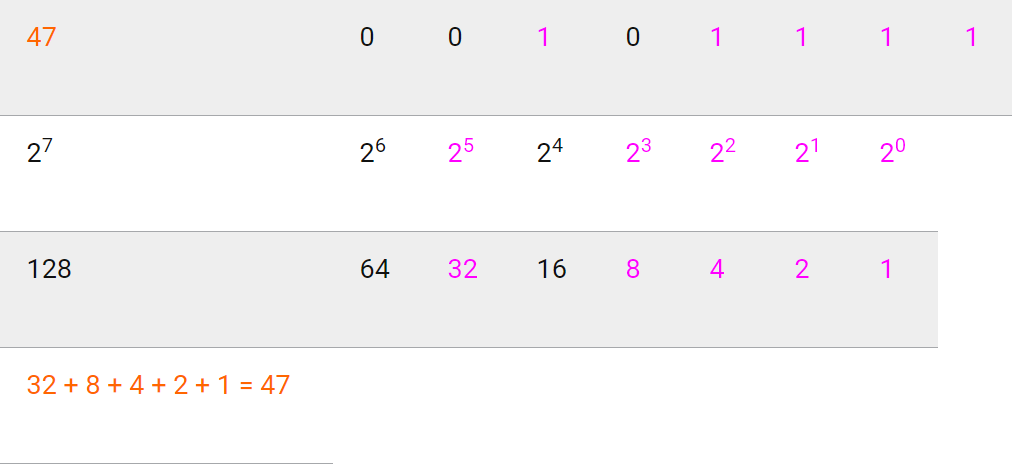
No MARS vamos criar um novo arquivo e digitar o seguinte código:



Salve-o e execute-o passo a passo.

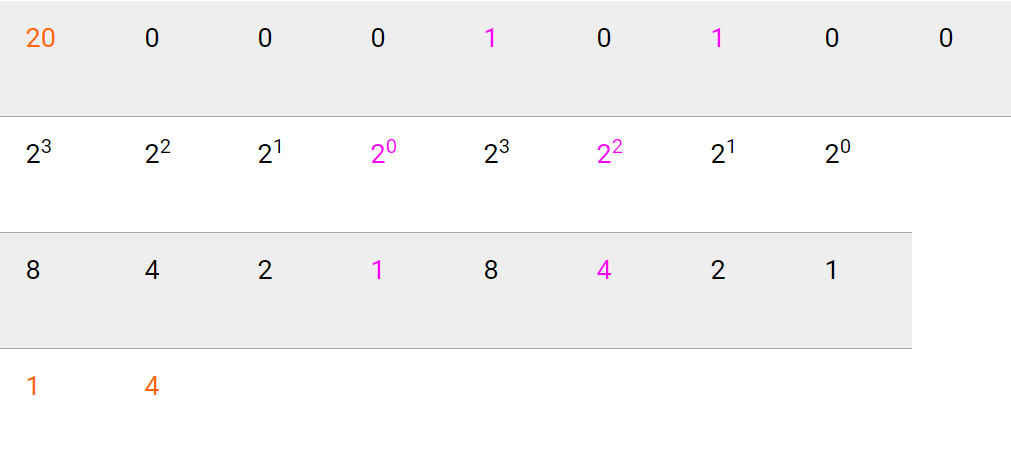
Vamos ver como ficam os números convertidos na base binária:

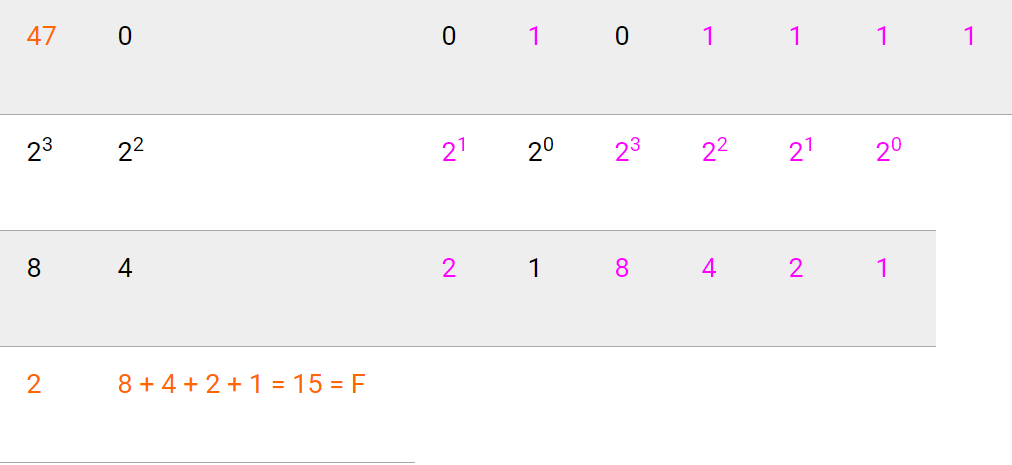




Portanto (20)10 = (00010100)2e (47)10 = (00101111)2.

Convertendo agora para hexadecimal:

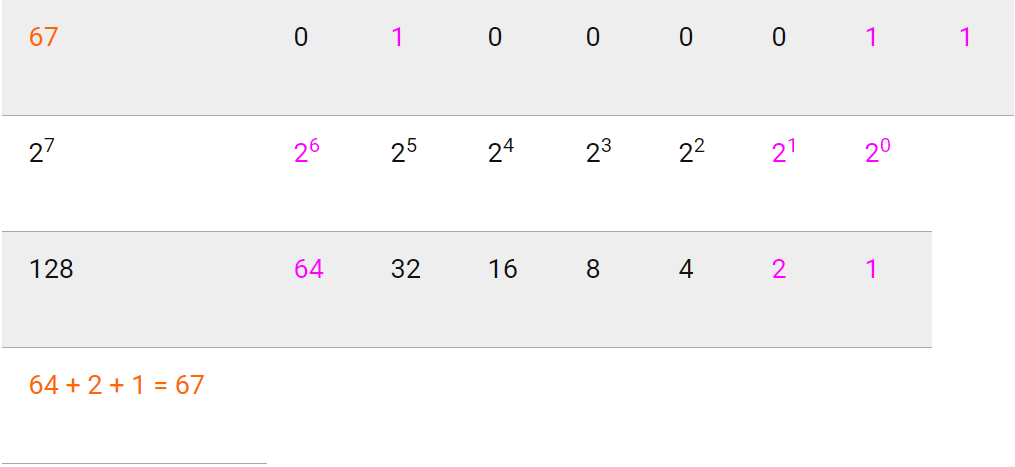


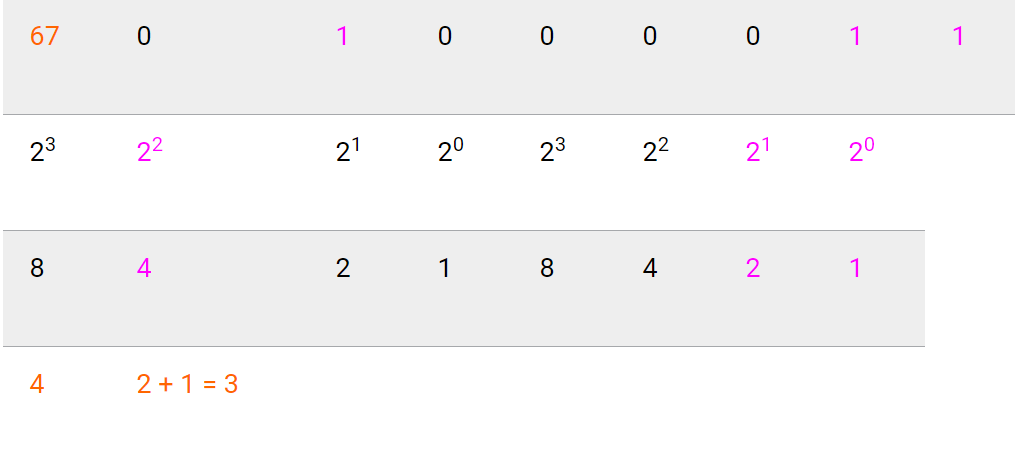


Onde A = 10, B = 11, C = 12, D = 13, E = 14 e F = 15.

Portanto (20)10 = (14)16 e (47)10 = (2F)16.

Somando 20 com 47 temos 67 que é (67)10 = (01000011)2 = (43)16.





Verifique os valores nos registradores.

Podemos perceber que estão corretos!

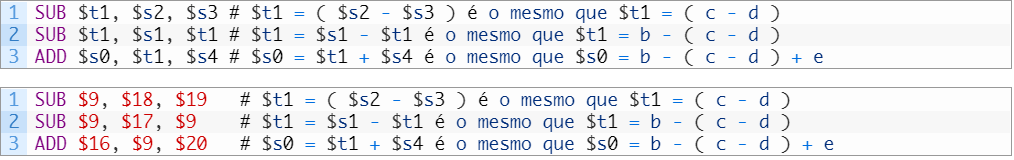
É importante ter esse base de conversão entre sistemas de numeração decimal, binário e hexadecimal bem clara.

Podemos tirar a prova usando a calculadora de programador disponível na calculadora do Windows.

**Exemplo 3**

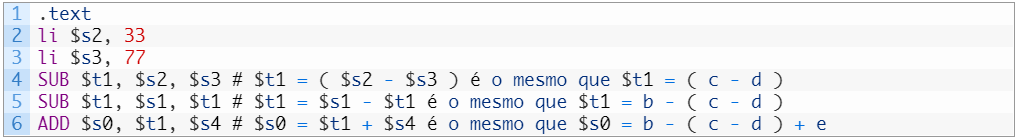
Vamos testar mais um código já conhecido nosso das aulas:

**a = b – (c  –  d ) + e**

****

Considere que **$s2 = 33** e **$s3 = 77**.

No MARS vamos criar um novo arquivo e digite o seguinte código:



Salve, execute e verifique os valores.

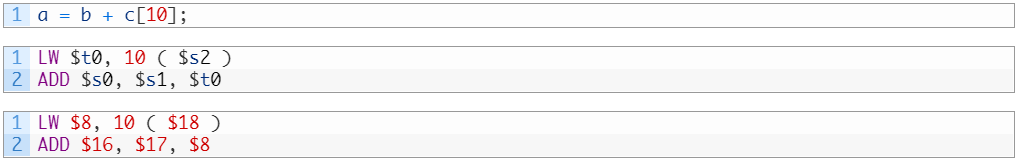
Porque tem um monte de **F** no resultado da subtração?

Explicando de forma bem simples, isso acontece pois o resultado dá negativo, visto que **33-77 = -44**, e o **F** neste caso está representando o sinal negativo.

Não vamos tratar aqui de sinais negativos. Ainda não aprendemos sobre isso. Mais a frente falaremos.

## Exemplo 4: Array

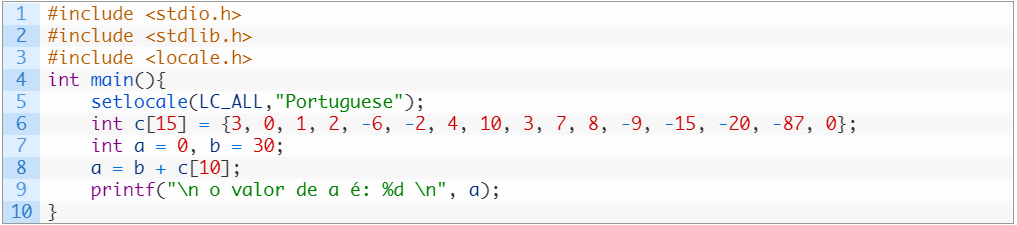
Vamos ver agora o outro exemplo dado em aula:



Esse exemplo é um pouco mais trabalhoso do que os outros pois estamos falando de um array, de obter o valor de uma posição específica desse array, e depois somar esses valores.

Não esqueça de que é preciso fazer o cálculo correto do endereço também.

Antes de mais nada, vejamos como é o código completo em C:



Para testar vamos descartar algumas coisas.

Como o FOR, por exemplo, visto que ainda não vimos instruções sobre LOOP. Porém, ter o programa de alto nível como referência, para conseguirmos entender um pouco melhor a compilação aqui no MARS, é muito útil.

Primeira coisa: notem que a variável **b** foi declarada como um tipo inteiro e a ela foi atribuído o valor **30**.

Segunda coisa: um array do tipo inteiro chamado **c**foi declarado com o tamanho de **15** posições, e já inicializado com os respectivos valores.

Terceira: também é declarada uma variável do tipo inteiro chamada **“a”**, que não é inicializada.

Quarta: a soma é executada.

Quinta coisa: impressão do resultado, a qual não mostraremos agora.

Faremos algo bem semelhante no MARS para testar o exemplo. Vamos declarar e inicializar um array no MARS, conforme demonstra a linha abaixo:



A diretiva **“.data”** indica que as instruções a seguir contêm dados.

A diretiva **“.word”** armazena **n** quantidades de **32 bits** em palavras de memórias sucessivas.

Assim, determinamos que **“c”** é um **label** (ou rótulo) que contém os valores designados da **word.**

Essas diretivas permitem usar o segmento de dados para armazenar e manipular dados.

Assim, pode-se dizer que a linha de código abaixo, em linguagem C…



…pode ser traduzida como abaixo, no MIPS:



Porém, há outras formas de trabalhar com Arrays na compilação para o MIPS.

Por hora, deixaremos assim para que consigamos testar os exercícios depois.

Além de definir e inicializar o array **“c”**, é necessário também definir o índice e o endereço, isto é, serão atribuídos a registradores temporários o índice específico (no caso 10) e o endereço do array.

Para atribuir o endereço do array a um registrador, é preciso usar uma instrução chamada **la**,que significa **load address**, em português “carga de endereço”.

Determinado o array, agora basta determinarmos **b = 30**, que ficará da seguinte forma:



O valor imediato, **30**, é carregado para **$s1**, que é o registrador de **b,**lembrando que **li** é a instrução de load immediate.

Após, é preciso carregar o endereço do array em um registrador, como a seguir:



Suponha que o endereço do array “**c**” seja **(1001)2** ou **(4097)10**.

Neste momento, na hora da execução do código Assembly no MARS, vamos notar que o registrador **$at** será utilizado para ajustar o endereço, de forma que ele passa de **0x00001001** para **0x10010000**.

O registrador **$at** é reservado para o montador, para uso exclusivo dele.

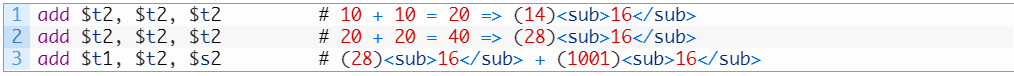
A instrução **la** carrega o endereço do array **c** para o registrador **$s2**.

Agora, é preciso especificar o índice para outro registrador, no caso do nosso exemplo, é a posição **10**:



Novamente uso**li**, para a carga do número inteiro **10**, que é um imediato, para o registrador **$t2**.

Agora que o endereço do array já está guardado em um registrador temporário, e o índice **10** também, é preciso calcular o endereço da forma correta, conforme já expliquei em artigos anteriores.



Na terceira linha o cálculo é feito como uma soma dos dois registradores:



Portanto, o endereço resultante armazenado em **$t1** é **0x10010028**.

Depois de calculado o endereço, já é possível carregar o valor contido em **c[10]** para o registrador**$t0:**



Na posição **c[10]** está armazenado o valor **8**, que em hexadecimal é **00000008**.

Reforçando também que a contagem no array começa em **zero** e não em um!

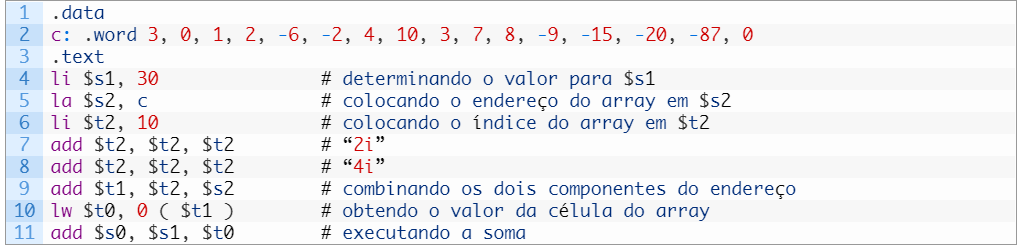
Cuidado com este pequeno grande detalhe, ok?

Bom, todo esse trabalho apenas para carregar o valor correto e então usá-lo em uma soma, que você já conhece bem:



O código completo fica conforme a seguir.

Teste o mesmo no MARS e verifique os valores, endereços e estados dos registradores.



## Concluindo

Agora que vimos quanto trabalho temos para fazer tudo isso dá para entender por que muitas pessoas preferem programar em alto ou médio nível.

Mas, apesar das dificuldades, linguagens de baixo nível em geral, são um grande aprendizado, pois nos desafiam a sair da zona de conforto e a pensar um pouco diferente do que estamos acostumados.

Sem contar que, nunca sabemos quando é que vamos precisar usar algo como isto em algum projeto, portanto, nenhum conhecimento é desperdiçado.

**EXERCÍCIO**

Escolha 3 códigos que já demos em aula (um deles tem que conter array) e utilize o MARS.

Apresente seus resultados.