Eduardo Bischoff Grasel - 22200355
Universidade Federal de Santa Catarina - UFSC

Organização de computadores I

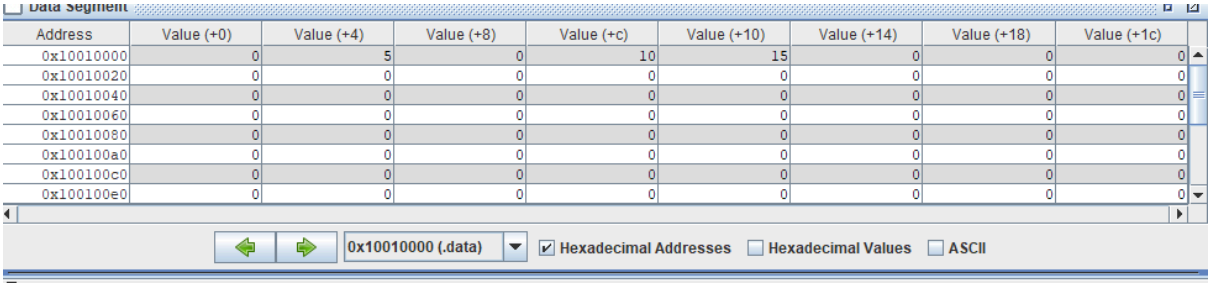
Florianópolis - SC
2024

Introdução

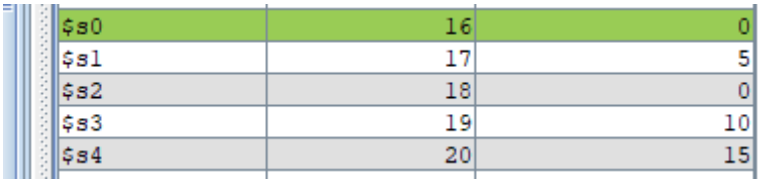
Com o objetivo de aprofundar o conhecimento em baixo nível, começamos o primeiro ‘projeto’ de organização de computadores usando a linguagem assembly através do IDE mars. Nesse projeto foi dado o objetivo de se familiarizar com a linguagem e suas peculiaridades, como o uso direto de registradores, endereços de memória e operações realizadas diretamente pelo processador. O mars nos dá certa facilidade e controle para ver o que está acontecendo nos endereços de memórias e registradores, o que facilita a visualização juntamente com a compreensão que essas informações oferecidas nos proporciona frente a dificuldade do assembly.

Projeto 1

Começamos declarando as variáveis que estarão presentes na memória no .data (a, b, c, d, e), como os valores de ‘a’ e ‘c’ serão usados para guardar valores de cálculos atribuímos a eles o valor 0, enquanto podemos decidir os valores de ‘b’, ‘d’, ‘e’ livremente, neste caso foram escolhidos os valores 5, 10, 15, respectivamente, que agora estão armazenados na memória.



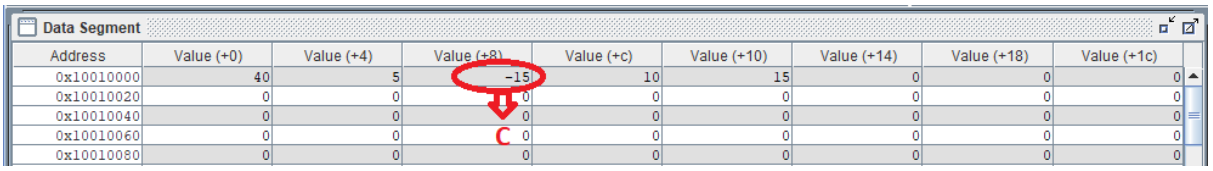
Após essa declaração iniciamos o .text no qual estará toda lógica do nosso programa. Agora iniciamos uma sequência de recuperação de valores para cada variável criada no .data, carregamos os endereços através do load address, e em seguida carregamos esses valores em registradores usando o load word, o endereço das variáveis que foram diferentes de 0 serão guardadas no registrador $t0 e logo em seguida carregadas em outro registrador para ser liberado para o próximo uso, entretanto o endereço de ‘a’ e ‘c’ se destinarão aos registradores $t1 e $t2, respectivamente, pois precisaremos destes endereços mais tarde, os load words serão salvos como: ‘a’ = $s0, ‘b’ = $s1, ‘c’ = $s2, ‘d’ = $s3, ‘e’ = $s4.



Enfim estamos prontos para iniciar os cálculos, iniciamos adição com imediato addi $s0, $s1, 35 equivalente a: a = b + 35, então prosseguimos para operação store word $s0, 0($t1), ou seja, guardamos o valor de $s0 no local da memória destinado a variável ‘a’, a seguir devemos separar a operação (c = d - a + e) guardando num registrador temporário o resultado da subtração e depois somando com o ‘e’, desta forma temos: sub $t3, $s3, $s0 (subtração guardada em $t3 de d - e) usamos esse valor para realizar a adição add $s2, $t3, $s4, assim temos guardado em $s2 o resultado final.

Em finalização, usamos um store word $s2, 0($t2), isso é referente a guardar o valor de $s2, que representa nosso ‘c’, no endereço de ‘c’ na memória.

valores usados b=5, d=10, e=15, ao executar o cálculo proposto é retornado o valor -15 para c.



Projeto 2

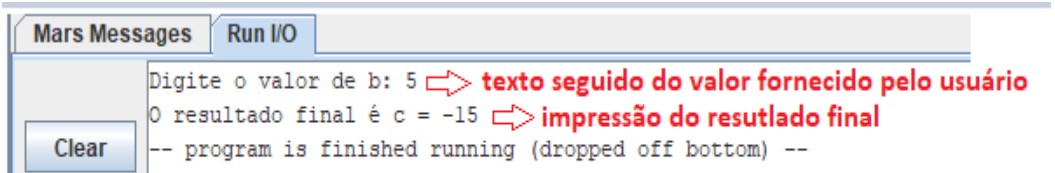
Para a realização do projeto 2 era necessário utilizar o código já escrito anteriormente e tornar ‘b’ um input do usuário, para isso será explicado a seguir apenas as partes modificadas.

Primeiramente tiramos a variável ‘b’ do .data, pois como não usaremos esse endereço para guardar quaisquer valores na memória, podemos pegar o dado diretamente no .text, todavia no local de ‘b’ no .data adicionamos uma string que será apresentada na hora de pedir o input para o usuário (b_msg), além disso adicionamos também outra string para o imprimir o resultado final (final_msg).

```
.data #  declarando as variáveis
        a: .word 0
        b_msg: .asciiz "Digite o valor de b: "
        c: .word 0
        d: .word 10
        e: .word 15
        final_msg: .asciiz "O resultado final é c = "
```

A próxima modificação foi referente ao modo de obtenção do valor ‘b’, é imprimido a mensagem “Digite o valor de b: ”, que por sua vez teu o endereço carregado através do load address no registrador $t4, seguidamente iniciamos instrução la $v0, 4, pois essa é a forma do mips imprimir uma string, mas para isso é preciso realizar um move $a0, $t4, afinal ao realizar o syscall será imprimido o valor salvo no registrador $a0, finalmente executamos o syscall e imprimimos a mensagem na tela. Usamos a instrução la $v0, 5 para ler inteiros e já podemos executar o syscall, o valor dado pelo usuário será salvo em $v0, movemos ele para $s1 utilizando um move $s1, $v0.

Agora partimos para o fim do programa, para imprimir o resultado. Em tese seguiremos o mesmo padrão dito anteriormente, a mensagem: "O resultado final é c = " será carregada em $t2, então executamos novamente a instrução move $a0, $t2 e damos o syscall na mensagem que se encontra no registrador correto. A última mudança significativa é na instrução para imprimir um inteiro, que usamos como la $v0, 1, seguidamente realizámos um move $a0, $s2 para que o dado esteja no registrador alvo da impressão, em seguida usamos o syscall e finalizamos o programa. Vale lembrar que o valor de ‘c’ contido no registrador $s2 foi salvo na memória da mesma maneira da primeira versão do programa.



Conclusão

Ao chegar ao fim do projeto foi adquirido o conhecimento básico referente às operações, sintaxe e syscalls de assembly (mips), o programa serviu seu papel didático e certamente trouxe o primeiro passo para o desafio que se tornará essa disciplina, a maior dificuldade foi se acustumar com o modo excêntrico de receber inputs do usuário e imprimir textos na tela, pois o mesmo é executado com base em valores em registradores específicos e comandos utilizados com mais de um propósito, que com valores diferentes muda sua função, este sendo o la (load address).