MIPS

Parte 1
Primeiro programa
Escrevendo um programa Assembly

As Mensagens do computador simulado aparecem na janela do console quando um programa que está sendo executado (na simulação), escreve para o monitor (simulado). Se um computador MIPS real as estivesse executando, veríamos as mesmas mensagens em um monitor real.

As Mensagens do simulador são todas as informações que o simulador precisa escrever para o usuário do simulador. Estas são mensagens de erro, avisos e relatórios.
Um arquivo de origem (em linguagem assembly ou qualquer linguagem de programação) é um arquivo de texto contendo instruções da linguagem de programação criada (geralmente) por um programador humano. Um editor como o Bloco de notas vai funcionar. Você provavelmente vai querer usar um editor melhor, podemos usar o Notepad (mas qualquer editor de texto vai ficar bem).

PERGUNTA : (Revisão) Qual o tipo de arquivo criado pelo Bloco de notas?
Resposta: Arquivos de texto - os arquivos de bytes que podem ser interpretados como caracteres ASCII.

Dois mais três

Processadores de texto mais complexos geralmente criam arquivos em "binário" e por isso não são adequados para a criação de arquivos fonte.
Crie o seguinte programa (texto) e dê o nome soma1.asm.

```
## Programa para somar 2 mais 3
.text
.globl main
main:
addi $8,$0,0x2          # coloca o comp. de dois do dois no reg. 8
addi $9,$0,0x3          # coloca o comp. de dois do três no reg. 9
add $10,$8,$9           # soma regs. 8 e 9, coloca resultado em 10

## fim
```

O caractere "#" inicia um comentário, tudo na linha de "#" para a direita é ignorado. Cada uma das três linhas a partir de main: corresponde a uma instrução de máquina.

PERGUNTA: (Revisão) O que é uma instrução de máquina?

Resposta: A instrução de máquina é um padrão de bits que solicita uma operação de máquina a ser executada.

Cada instrução de máquina é de 32 bits MIPS (quatro bytes) de comprimento. As três linhas após o main: são três instruções de máquina.
As linhas restantes consistem de informações para o montador e comentários (para humanos).

Explicação do Programa

Existem várias maneiras de executar um programa em uma máquina real para devolver o controle ao sistema operacional. Mas não temos OS, então agora vamos para as instruções passo.

```
## Programa para somar 2 mais 3
.text
.globl main
main:
ori $8,$0,0x2           # coloca o comp. de dois do dois no reg. 8
ori $9,$0,0x3           # coloca o comp. de dois do três no reg. 9
add $10,$8,$9           # soma regs. 8 e 9, coloca resultado em 10

## fim
```

A primeira linha do programa é um comentário. Ele é ignorado pelo montador e não resulta em nenhuma instrução de máquina.

.text é uma diretiva. Uma diretiva é uma declaração que diz ao montador algo sobre o que o programador quer, mas não resulta em quaisquer instruções da máquina. Esta diretiva diz ao assembler que as linhas a seguir são ".Text" do código-fonte - para o programa.

.globl main é outra diretiva. Ela diz que o identificador main será utilizado fora desse arquivo de origem (isto é, será utilizado globalmente) como o label de um determinado local na memória principal.

PERGUNTA 12: (teste de memória) Onde estava primeira instrução de máquina na memória?
Resposta: 0x00400000

As linhas em branco são ignoradas. A linha main: define o endereço simbólico (chamado label). Um endereço simbólico é um símbolo (um identificador) que é o nome que o código fonte usa para um local na memória. Nesse programa, main indica o endereço da primeira instrução de máquina (que acaba por ser

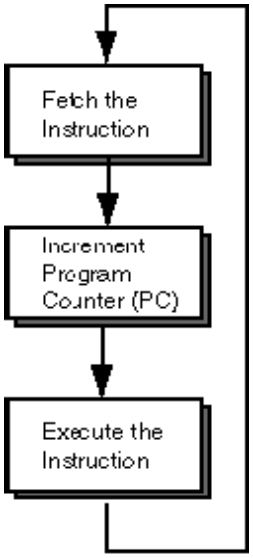
0x00400000). Usar um endereço simbólico é muito mais fácil do que usar um endereço numérico.
Com um endereço simbólico, o programador se refere a posições de memória pelo nome e permite que o montador determine o endereço numérico.

O símbolo main é global. Isso significa que vários arquivos fontes podem usar o símbolo main para se referir ao mesmo local de armazenamento. (No entanto, o SPIM não usa este recurso.) Todos os nossos programas serão incluídos em um único arquivo fonte.

PERGUNTA: (Teste a sua intuição:) É provável que várias seções de um sistema de software necessitem de se referir uns aos outros?
Resposta: Sim.

Modelo do ciclo de máquina

O ciclo sem fim de execução possui 3 passos e executa uma instrução de máquina.
Como a execução segue uma lista de instruções, um registrador especial, o contador de programa (PC), que aponta para a próxima linha a executar:



Fetch - O contador de programa contém o endereço da próxima instrução de máquina. A instrução é lida da memória.

Incrementar o PC. O endereço no contador de programa é incrementado de quatro.

Execute a Instrução. A operação de máquina especificada pela instrução é executada.

Operandos imediatos e lógicos Bitwise
Algumas instruções de máquina contêm dados como parte da instrução. Estes dados constituem o que chamamos um operando imediato. A instrução ori usada previamente contém um operando imediato. Com uma operação de lógica de bitwise dois padrões de bits são alinhados e operações como OU ou E podem ser executadas entre pares de bits.
Pergunta: Um padrão de 32-bit do MIPS possui espaço para um inteiro de 16-bit em complemento de dois?
Resp.: Sim, desde que o espaço restante seja suficiente para especificar a operação.

Operandos imediatos
Algumas instruções de máquina usam 16 dos 32 bits para armazenar um do dois operandos. Um operando que é codificado diretamente como parte de uma instrução de máquina é chamado um operando imediato. Por exemplo:

address machine code assembly code
0x00400000 0x34080002 ori $8,$0,0x2

OR Operation on Bits				
Prim. Operando	0	0	1	1
Seg. Operando	0	1	0	1
	__	__	__	__
Resultado	0	1	1	1

Os últimos 16 bits (quatro caraterers em hexa) da instrução de máquina contêm o operando 0x0002. (A instrução é simplificada por "0x2"). A instrução de máquina diz para a ALU que execute um bitwise OU entre os conteúdos do registrador $0 e o de operando imediato 0x0002. O resultado é colocado no registrador $8.
Uma operação de bitwise é onde uma operação lógica é executada nos bits de cada coluna do operandos. Aqui é o bitwise OU entre dois padrões de 8 bits.

0110 1100 operando
0101 0110 operando

0111 1110 resultato

Pergunta
Qual o padrão de bits do registrador $0?
Quantos bits existem por instrução de MIPS?
Resposta:
Todos os bits valem zero.
32 bits

Extensão do zero
Observe novamente a instrução:

address machine code assembly code

0x00400000 0x34080002 ori $8, $0, 0x2

E 16 bits de operando imediato:
0000 0000 0000 0010
...que são combinados bit a bit (bitwise) com o registrador zero de 32 bits,

0000 0000 0000 0000 0000 0000 0000 0000
Isto ordinariamente não seria possível porque os operandos possuem comprimentos diferentes. Porém, zero de MIPS estende o operando de dezesseis-bit e ambos passam a possuir o mesmo comprimento.
---- ---- ---- ----
0000 0000 0000 0000 0000 0000 0000 0010 – operando com zero estendido
0000 0000 0000 0000 0000 0000 0000 0000 – dado no registrador $0

0000 0000 0000 0000 0000 0000 0000 0010 -- resultado, no registrador $8

Uma operação OR é realizada em cada coluna. O resultado de 32-bit é armazenado no registrador $8.

A parte constante da instrução pode ser um decimal positive ou uma constante em hexadecimal. O montador traduz o número para um padrão de 16 bit. Por exemplo, as dias instruções abaixo são traduzidas para a mesma instrução:

ori $5,$4,0x10
ori $5,$4,16

Pergunta:
A instrução ori $0,$9,0x32 está correta ?
Incorreta. Ela diz para armazenar o resultado no registrador zero, o que não é possível.

Exemplos

```
## Programa bitwise OR de dois padrões
.text
.globl main
main:
ori $8,$0,0x0FA5            # primeiro padrão em $8
ori $10,$8,0x368F           # or ($8) com o Segundo padrão, Resultado em $10
## End of file
```

Novas instruções:

```
andi  d,  s,  const         # registrador d <-- bitwise AND
xori   d,  s,  const      # registrador d <-- bitwise XOR
```

Executar o programa a seguir

```
## Programa bitwise OR, AND, e XOR entre dois padrões de bits
```

```
.text
.globl main
main:
ori $15, $0,0x0FA5          # armazena um padrão no reg. $15
ori $8,$15,0x368F           # OR com Segundo padrão
andi $9,$15,0x368F                  # AND com Segundo padrão
xori $10,$15,0x368F                 # XOR com Segundo padrão
## End of file
```

No-Op

O registrador $0 (ou $zero) sempre contém o número zero. Assim, um sll nesse registrador de zero posições e armazenando o resultado no próprio registrador zero, não ocasionará nenhum efeito.

Essa instrução é denominada no-op. Futuramente você verá que uma instrução no-op será de grande utilidade.

Instruções addi e addiu

As instruções addi e addiu incluem um imediato de 16 bits. Quando a ULA executa esse tipo de instrução, o sinal do imediato (de 16 bits) é estendido para que a soma com um registrador de 32 bits possa ser executada.

O formato é basicamente o seguinte:

```
addi d,s,const      # $d <-- s + const.
                    # Const é um inteiro de 16-bit com o
                    # sinal estendido para 32 bits quando a
                    # adição é realizada.
                    # Causa um trap de Overflow.
addiu d,s,const     # $d <-- s + const.
                    # Const é um inteiro de 16-bit com o
                    # sinal estendido para 32 bits quando a
                    # adição é realizada.
                    # NÃO Causa um trap de Overflow.
```

TRAP de overflow:

Um trap é uma interrupção no ciclo normal da máquina. Tipicamente o processador retorna o controle para o sistema operacional. A maioria dos programadores trabalha com o overflow garantindo que os operandos não o cause.

O sistema operacional possui handlers especiais para lidar com esses traps, isto é, caso o programa gere um erro (ou trap) o sistema operacional possui ações especiais para lidar com o erro.

Teste o seguinte programa:

```
## Programa para testar addi e addiu
.text
.globl  teste
teste:
addiu $9,$0,0x7FFF
sll $9,$9,16
addiu $9,$9,0x7FFF
addiu $9,$9,0x7FFF
addiu $9,$9,0x7FFF
```

addi $8,$0,0x7FFF
sll $8,$8,16
addi $8,$8,0x7FFF
addi $8,$8,0x7FFF
addi $8,$8,0x7FFF

O que você observou, qual o erro ocasionado pelo addi e que não ocorreu com o addiu.

Relatório

Responda

1. **O que é um arquivo fonte?**
   A. um arquivo de texto que contém instruções de linguagem de programação.
   B. um subdiretório que contém os programas.
   C. um arquivo que contém dados para um programa.
   D. um documento que contém os requisitos para um projeto.

2. **O que é um registro?**
   A. parte do sistema de computador que mantém o controle dos parâmetros do sistema.
   B. uma parte do processador que possui um padrão de bits.
   C. parte do processador que contém o seu número de série único.
   D. parte do bus de sistema que contém dados.

3. **Qual o caracter que, na linguagem assembly do SPIM, inicia um comentário?**
   A. #
   B. $
   C. / /
   D. *

4. **Quantos bits há em cada instrução de máquina MIPS?**
   A. 8
   B. 16
   C. 32
   D. instruções diferentes possuem diferentes comprimentos.

5. **Quando você abre um arquivo de origem a partir do menu Arquivo do SPIM, quais as duas coisas que acontecem?**
   A. O arquivo está carregado na memória e começa a execução.
   B. SPIM é iniciado eo arquivo é aberto no editor.
   C. O arquivo é montado em instruções de máquina, e as instruções de máquina são carregados na memória do SPIM.

D. O programa é executado e os resultados são salvos em disco.

6. **O que é o contador de programa?**
   A. um registrador que mantém a conta do número de erros durante a execução de um programa.
   B. uma parte do processador que contém o endereço da primeira palavra de dados.
   C. uma variável na montadora que os números das linhas do arquivo de origem.
   D. parte do processador que contém o endereço da próxima instrução de máquina para ser obtida.

7. **Ao pressionar a tecla F10 para executar uma instrução, quanto será adicionado ao contador de programa?**
   A. 1
   B. 2
   C. 4
   D. 8

8. **O que é uma diretiva, tal como a diretiva .text?**
   A. uma instrução em linguagem assembly que resulta em uma instrução em linguagem de máquina.
   B. uma das opções de menu do sistema SPIM.
   C. uma instrução em linguagem de máquina que faz com que uma operação sobre os dados ocorra.
   D. uma declaração que diz o montador algo sobre o que o programador quer, mas não corresponde diretamente a uma instrução de máquina.

9. **O que é um endereço simbólico?**
   A. um local de memória que contém dados simbólicos.
   B. um byte na memória que contém o endereço de dados.
   C. símbolo dado como argumento para uma directiva.
   D. um nome usado no código-fonte em linguagem assembly para um local na memória.

10. **Em qual endereço o simulador SPIM coloca a primeira instrução de máquina quando ele está sendo executado com a opção Bare Machine ligada?**
    A. 0x00000000
    B. 0x00400000
    C. 0x10000000
    D. 0xFFFFFFFF

11. Algumas instruções de máquina possuem uma constante como um dos operandos. Como é chamado tal operando?
A. operando imediato
B. operando embutido
C. operando binário
D. operando de máquina

12. Como é chamada uma operação lógica executada entre bits de cada coluna dos operandos para produzir um bit de resultado para cada coluna?
A. operação lógica
B. operação bitwise
C. operação binária
D. operação coluna

13. Quando uma operação é de fato executada, como estão os operandos na ALU?
A. Pelo menos um operando deve ser de 32 bit.
B. Cada operando pode ser de qualquer tamanho.
C. Ambos operandos devem que vir de registros.
D. Cada um dos registradores deve possuir 32 bit.

14. Dezesseis bits de dados de uma instrução de ori são usados como um operando imediato. Durante execução, o que deve ser feito primeiro?
A. Os dados são estendidos em zero à direita por 16 bits.
B. Os dados são estendidos em zero à esquerda por 16 bits.
C. Nada precisa ser feito.
D. Apenas 16 bits são usados pelo outro operando.

15. Qual o nome para um padrão de bits copiados em um registrador?
A. load.
B. filled.
C. stuffed.
D. fixed.

16. Qual das instruções seguintes armazenam no registrador $5 um padrão de bits que representa positivo 48?
A. ori $5,$0,0x48
B. ori $5,$5,0x48
C. ori $5,$0,48
D. ori $0,$5,0x48

17. A instrução de ori pode armazenar o complemento de dois de um número em um registrador?
A. Não.
B. Sim.

18. Qual das instruções seguintes limpa todos os bits no registrador $8 com exceção do byte de baixa ordem que fica inalterado?
A. ori $8,$8,0xFF
B. ori $8,$0,0x00FF
C. xori $8,$8,0xFF
D. andi $8,$8,0xFF

19. Qual é o resultado de um ou exclusivo de padrão sobre ele mesmo?
A. Todos os bits em zero.
B. Todos os bits em um.
C. O padrão original utilizado.
D. O resultado é o contrário do original.

20. Todas as instruções de máquina têm os mesmos campos?
A. Não. Diferentes de instruções de máquina possuem campos diferentes.
B. Não. Cada instrução de máquina é completamente diferente de qualquer outra.
C. Sim. Todas as instruções de máquina têm os mesmos campos na mesma ordem.
D. Sim. Todas as instruções de máquina têm os mesmos campos, mas eles podem estar em ordens diferentes.

Implementar os seguintes programas (usando apenas as instruções indicadas)

//**programa 1** (add, addi, sub, lógicas)

```
{
        a =2;
        b =3;
        c =4;
        d =5;
        x = (a+b) - (c+d);
        y = a – b + x;
        b = x – y;
}
```

//**programa 2** (add, addi, sub, lógicas)

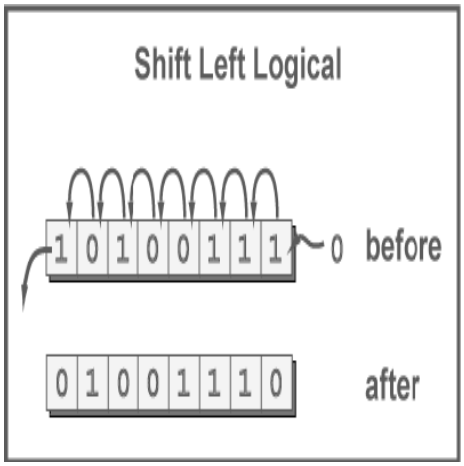
```
{
        x = 1;
        y = 5*x + 15;
}
```

// **programa 3** (add, addi, sub, lógicas)

```
{
        x = 3;
        y = 4 ;
        z = ( 15*x + 67*y)*4
}
```

Instruções sll, srl, sra

Shift Left Logical



É a instrução de deslocamento lógico. Em um deslocamento de um bit, o bit de mais baixa ordem é deslocado à esquerda e substituído por zero. O bit de mais alta ordem é descartado. O procedimento é similar tendo em vistas a quantidade de bits a deslocar.
Exemplo:
O padrão original é 1010 0111. O padrão resultante é 0100 1110.
O processador do MIPS sempre executa a operação em um registrador de 32-bit e armazena o resultado em um registrador de 32-bit.

```
sll d,s,shft     # $d <-- os bits em $s são deslocados à esquerda
                 # por shft posições,
                 # onde  0 <= shft < 32
```

A ALU que faz a operação não se preocupa com que os bits significam. Se os bits representarem um inteiro sem sinal (unsigned), então uma troca esquerda é equivalente a multiplicar o inteiro por dois.

Contudo observe o número a seguir:

Original	**Shift Left dois**
0110 1111	1011 1100
0x6F	0xBC

Se os 8-bit originais representarem um número inteiro, o deslocamento de dois bits provavelmente não representará o dobro mas um possível erro, tendo em vista a perda do bit mais significativo.
Avalie o programa a seguir:

```
## desloca2.asm
##
## Programa para deslocar à esquerda
.text
.globl main
main:
ori $8, $0, 0x6F      # armazena um padrão em  $8
sll $9, $8, 2         # shift left logical de 2
## End of file
```

Pergunta:

É possível deslocar o conteúdo do registrador $8 e armazenar o resultado no próprio registrador $8:

```
ori $8, $0, 0x6F      # armazena um padrão no $8
sll $8, $8, 2         # shift left logical de 2
```

Resp: teste e responda.

Instruções srl e sra

Existe ainda a instrução srl, que corresponde a um shift lógico para a direita, o processo e a sintaxe são idênticas ao sll.
Uma instrução semelhante ao srl é o sra, shift right aritmético (a sintaxe é idêntica) mas os efeitos não.

Teste a seguinte situação:

1) armazene um número positivo e execute um srl e um sra com um deslocamento de 1 bit.
2) Repita a operação com um número negativo.
3) O que você observou ?

Shift Right Arithmetic

A instrução shift right logical não pode ser utilizada para dividir. O problema é que ela insere zeros no bit de mais alta ordem. Pode funcionar em certos casos más ocasiona problemas para números negativos. Já a instrução **Shift Right Arithmetic,** replica o bit mais significativo.
Perg: Existe a necessidade de um *arithmetic shift left* instruction?
Resp: Não. Um logical shift left move zeros nos bits de mais baixa ordem, o que é correto para inteiros signed e unsigned (com e sem sinal).
sra d,s,shft # $d <-- s deslocado para direita

```
                    # deslocado shft bits.
                    # 0 =< shft < 31
```

Relatório
Implementar os seguintes programas (procure usar as inst. sll, srl e sra)

// programa 4

```
{
        x = 3;
        y = 4 ;
        z = ( 15*x + 67*y)*4
```

}

// programa 5

```
{
        x = 100000;
        y = 200000;
        z = x + y;
}
```

// programa 6

```
{
        x = o maior inteiro possível;
        y = 300000;
        z =  x - 4y
}
```

// programa 7

Considere a seguinte instrução iniciando um programa:
ori $8, $0, 0x01

Usando apenas instruções reg-reg lógicas e/ou instruções de deslocamento (sll, srl e sra), continuar o programa de forma que ao final, tenhamos o seguinte conteúdo no registrador $8:
$8 = 0xFFFFFFFF

// programa 8

Inicialmente escreva um programa que faça:

$8 = 0x12345678.

A partir do registrador $8 acima, usando apenas instruções lógicas (or, ori, and, andi, xor, xori) e instruções de deslocamento (sll, srl e sra), você deverá obter os seguintes valores nos respectivos registradores:
$9 = 0x12
$10 = 0x34
$11 = 0x56
$12 = 0x78